



TECHNICAL UNIVERSITY OF CRETE
School of Electrical and Computer Engineering

DIPLOMA THESIS

Building Configurable Reinforcement Learning Robotic Environments

Andreas Kallinteris

Thesis Committee

Professor Georgios Chalkiadakis (supervisor)

Professor Thrasyvoulos Spyropoulos

Professor Michail G. Lagoudakis

Chania, March 2025



Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματικής Εργασίας

Δημιουργία Παραμετροποιήσιμων
Ρομποτικών Περιβαλλόντων Ενισχυτικής
Μάθησης

Ανδρέας Καλλιντέρης

Εξεταστική Επιτροπή

Καθηγητής Γεώργιος Χαλκιαδάκης (επιβλέπων)

Καθηγητής Θρασύβουλος Σπυρόπουλος

Καθηγητής Μιχαήλ Λαγουδάκης

Χανιά, Μάρτιος 2025

Declaration of Authorship

I, Andreas Kallinteris, declare that this thesis titled, “Building Configurable Reinforcement Learning Robotic Environments” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a diploma degree at the Technical University of Crete.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Abstract

The creation of standardized environment implementations and an Application Programming Interface (API) for *OpenAI/Gym* has had a transformative impact on reinforcement learning (RL) research. However, the current set of standardized environments has to be extended, so as to contribute to the further advancement of reinforcement learning algorithms. In this diploma thesis, we have developed, and we provide a plethora of novel environments and frameworks for robotic reinforcement learning, including *Gymnasium/Mujoco-v5*, *Gymnasium-Robotics/Maze-v5*, and *Gymnasium-Robotics/MaMuJoCo*, along with offline RL datasets for *Gymnasium/MuJoCo* environments with the *Minari* API. These advancements can potentially enable researchers to develop and test new algorithms in more realistic and challenging environments, which will ultimately lead to more robust and generalizable reinforcement learning algorithms.

Περίληψη

Η δημιουργία τυποποιημένων υλοποιήσεων περιβάλλοντος και ενός **Application Programming Interface (API)** για το *OpenAI/Gym*, έπαιξε καθοριστικό ρόλο στην πρόσφατη πρόοδο της έρευνας για **reinforcement learning (RL)**. Ωστόσο, το περιορισμένο σε αριθμό και δυνατότητες τρέχον σύνολο τυποποιημένων περιβαλλόντων αποτελεί εν δυνάμει εμπόδιο για την περαιτέρω πρόοδο των αλγορίθμων ενισχυτικής μάθησης. Στην παρούσα διπλωματική εργασία, έχουμε αναπτύξει και παρέχουμε μια πληθώρα νέων περιβαλλόντων και πλαισίων για ρομποτική ενισχυτική μάθηση, συμπεριλαμβανομένων των *Gymnasium/Mujoco-v5*, *Gymnasium-Robotics/Maze-v5*, και *Gymnasium-Robotics/MaMuJoCo*, μαζί με σύνολα δεδομένων για χρήση από **offline RL** μεθόδους σε περιβάλλοντα *Gymnasium/MuJoCo* με το *Minari API*. Οι συνεισφορές μας αυτές μπορούν δυνητικά να επιτρέψουν στους ερευνητές να αναπτύξουν και να δοκιμάσουν νέους αλγορίθμους σε πιο ρεαλιστικά και δύσκολα περιβάλλοντα, το οποίο τελικά θα οδηγήσει σε πιο ισχυρούς και γενικεύσιμους αλγορίθμους ενισχυτικής μάθησης.

Acknowledgements

This thesis was written exclusively by **Kallinteris Andreas**, with proofreading by **Georgios Chalkiadakis**.

Each of the following projects was made by the people listed below:

0.1 Gymnasium/MuJoCo-v5

Lead Developer: **Andreas Kallinteris**

Specifications/Requirements and Code Review: Mark Towers (Project manager of *Gymnasium*)

Debugging assistance: Rodrigo de Lazcano (Former *Gymnasium-Robotics* Project manager)

0.2 Minari Gymnasium/MuJoCo Datasets

Lead Developer: **Andreas Kallinteris**

Specifications/Requirements and Code Review: Omar Younis (Project Manager of *Minari*)

0.3 Gymnasium-Robotics/Maze-v5

Solo Developer: **Andreas Kallinteris**

0.4 Gymnasium-Robotics/MaMuJoCo

0.4.1 Gymnasium-Robotics/MaMuJoCo-v0

Lead Developer: **Andreas Kallinteris**

Specifications/Requirements and Code Review: Mark Towers (Project manager of *Gymnasium*) and Rodrigo de Lazcano (Former *Gymnasium-Robotics* Project manager)

0.4.2 Gymnasium-Robotics/MaMuJoCo-v1

Lead Developer: **Andreas Kallinteris**

Code Review: Rodrigo de Lazcano (Former *Gymnasium-Robotics* Project manager)

0.5 Centralized Critic Experiment

Researcher: **Andreas Kallinteris**

Supervisor: Georgios Chalkiadakis

Thanks to TUC / ECE / Intelligence Systems Lab for providing the computational resources for this experiment.

0.6 Impact of Contact Forces Observations on Quadruped Experiment

Researcher: **Andreas Kallinteris**

Contents

Declaration of Authorship	ii
Abstract	iii
Περίληψη	iv
Acknowledgements	v
0.1 Gymnasium/MuJoCo-v5	v
0.2 Minari Gymnasium/MuJoCo Datasets	v
0.3 Gymnasium-Robotics/Maze-v5	v
0.4 Gymnasium-Robotics/MaMuJoCo	v
0.4.1 Gymnasium-Robotics/MaMuJoCo-v0	v
0.4.2 Gymnasium-Robotics/MaMuJoCo-v1	vi
0.5 Centralized Critic Experiment	vi
0.6 Impact of Contact Forces Observations on Quadruped Experiment	vi
List of Figures	ix
List of Tables	xi
List of Abbreviations	xii
1 Introduction	1
1.1 Contributions	3
1.2 Thesis Outline	4
2 Background and Related Work	5
2.1 Markov Decision Process	5
2.2 Partially Observable Markov Decision Process	5
2.3 Reinforcement Learning	6
2.4 Deep Reinforcement Learning	6
2.5 Multi-agent Reinforcement Learning	7
2.6 Offline RL	8
2.7 Gymnasium API	8
2.8 Gymnasium/MuJoCo Environments	9
2.8.1 Cartpole based environments	9
2.8.2 Arm manipulation based environments	11
2.8.3 2D Locomotion Environments	11

2.8.4	Swimmer Environment	12
2.8.5	Quadruped Environment	13
2.8.6	Humanoids Environments	13
2.9	Minari	14
2.10	Related Work	14
3	New Environments and Frameworks	16
3.1	Gymnasium/MuJoCo-v5 framework	16
3.1.1	Using Third-Party Robot Models	17
3.1.2	Performance Improvements	17
3.1.3	Increased Configurability	17
3.1.4	<i>info</i> fields	18
3.1.5	Bug Fixes	18
3.1.6	Documentation	19
3.2	Offline RL Minari Datasets	19
3.3	Other Robotics Environments	21
3.3.1	Gymnasium-Robotics/Maze-v5	21
3.4	Gymnasium-Robotics/MaMuJoCo-v1 framework	21
4	Conclusions and Future Work	26
4.1	Future Work	27
A	Gymnasium MuJoCo-v5 full changelog	28
B	Gymnasium-Robotics/MaMuJoCo Full Changelog	35
B.1	Gymnasium-Robotics/MaMuJoCo-v1	35
B.2	Gymnasium-Robotics/MaMuJoCo-v0	36
C	Centralized Critic Experiment	39
C.1	Centralized Critic Re-use Policy Reconstruction Across Differ- ent Factorizations	39
D	Impact of Contact Forces Observations on Quadruped Experiment	42
	Bibliography	45

List of Figures

1.1	Timeline of the most significant RL developments.	1
1.2	The collection of <i>Gymnasium/MuJoCo</i> environments, from left to right, (1, 2) <i>InvertedPendulum</i> and <i>InvertedDoublePendulum</i> , CartPole robots, (3,4) <i>Reacher</i> and <i>Pusher</i> , arm manipulator, (5,6,7) <i>HalfCheetah</i> , <i>Hopper</i> and <i>Walker2D</i> , 2d walking robots, (8, 9, 10) <i>Swimmer</i> , <i>Ant</i> and <i>Humanoid</i> 3d locomoting robots, (11) <i>HumanoidStandup</i> , a standup task.	2
3.1	Unitree Go1 Robot Model in a <i>Gymnasium</i> environment, with the objective to locomote as fast as possible while minimizing total control. Code example at Listing 4	19
3.2	Example of the Unitree Go2 robot inside a trivial maze; the robot's objective is to reach the red goal: env = gymnasium.make('AntMaze_UMaze-v5', xml_file="./mujoco_menagerie/unitree_go2/scene.xml", maze_size_scaling=2, maze_height=0.4, max_episode_steps=100)	22
3.3	Agent factorization spaces, examples	22
3.4	Example Factorization of Boston Dynamics's spot with arm robot. The robot is factorized into two agents: the quadruped agent and the arm agent. Code example can be found in the Documentation: https://robotics.farama.org/envs/MaMuJoCo/#example-boston-dynamics-spot-arm-with-custom-quadruped-arm-factorization	23
3.5	Sample factorizations of the <i>Gymnasium/Ant</i> environment in <i>MaMuJoCo</i> . Showing a vizualization the factorized action spaces. (Top Left) Single agent factorization, (Top Right) Two agent factorization, (Bottom Left) Two agent in a diagonal factorization, (Bottom Right) four agent factorization.	24
C.1	Experiments on <i>MuJoCo/Ant</i> and <i>MaMuJoCo/Ant</i> with TD3, MATD3, and MATD3-cc. The X-axis shows the training time steps, and the Y-axis shows the max agent episodic return. (Blue Line, 1st in legend) Single Agent TD3. (2nd, 3rd, 4th in legend) MATD3 of the various factorized cases. (5th, 6th, 7th in legend) MATD3-cc of the various factorized cases.	41

D.1	Experiment on <i>MuJoCo/Ant-v5</i> , The X-axis shows the training time steps, and the Y-axis shows the maximum agent episodic return. Two Algorithms show here are TD3 and SAC, with subcases for with and without external contact forces being observed ($o_{cfr_{c_ext}}$).	43
D.2	Experiment on Unitree Go1, The X-axis shows the training time steps, and the Y-axis shows the maximum agent episodic return. Two Algorithm show here is SAC, with subcases for with and without external contact forces being observed ($o_{cfr_{c_ext}}$).	44
D.3	Experiment on Unitree Go1, The X-axis shows the training time steps, and the Y-axis shows the average episode length. Two Algorithm show here is SAC, with subcases for with and without external contact forces being observed ($o_{cfr_{c_ext}}$). . . .	44

List of Tables

3.1	Offline Minari Datasets for <i>Gymnasium/MuJoCo-v5</i> , "Dataset Steps" column shows the size of the generated datasets, "Learning Algorithm" shows the algorithm used and which options were used. On <i>Swimmer</i> , the learning rate was set to $\gamma = 1$ because [58]. The Ant "expert" dataset was generated without $o_{cfr_{ext}}$, because of appendix D.	25
D.1	Comparison of learning algorithms and the impact of $o_{cfr_{ext}}$ in the Ant environment. Note: even though we are comparing different revisions of <i>Gymnasium/MuJoCo/Ant</i> , the differences are within $\pm 2\%$	43

List of Abbreviations

AI	Artificial Intelligence
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MARL	Multi Agent Reinforcement Learning
API	Application Programming Interface
CTDE	Centralized Training Decentralized Execution
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
MuJoCo	Multi-Joint dynamics with Contact
MENACE	Matchbox Educable Noughts And Crosses Engine
ASE/ACE	Associative Search Element with Adaptive Critic Element
SARSA	State Action Reward State Action
SRV	Stochastic Real-Valued
A2C	Advantage Actor Critic
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Network
PPO	Proximal Policy Optimization
SAC	Soft Actor Critic
TD3	Twin Delayed Deep Deterministic Policy Gradient
TQC	Truncated Quantile Critics

Chapter 1

Introduction

Reinforcement Learning (RL) is one of the major fields of Artificial Intelligence (AI) that enables the learning of intelligent agents capable of interacting with an environment. The field of reinforcement learning is employed in many fields, such as Autonomous Driving [1], Medical System Control [2], Financial Automated Trading [3], Energy Systems [4], Mathematics [5], Telecommunication Systems [6] and Robotics [7] [8] [9] [10] [11] [12] [13].

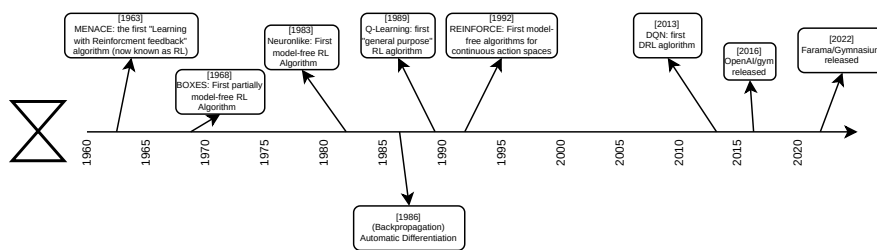


FIGURE 1.1: Timeline of the most significant RL developments.

A concise historical overview of RL is presented here and the timeline is also illustrated in Figure 1.1. The focus is on both the development of RL algorithms and the corresponding RL environments they solved. In 1963, the MENACE algorithm [14] was introduced as the first instance of an RL algorithm, and it was used to solve the "TIC-TAC-TOE against a random opponent" environment. In 1968, [15] developed BOXES, the first partially model-free ¹ RL algorithm, and used it to solve a simplified version of the CartPole environment, which consisted of a pole attached to a cart and controlled by an actuator. The objective was to maintain the pole's balance by learning probabilistic decision boxes. In 1983, ACE/ACE [16], was the first paper to employ Model-Free RL, which used a simple neural network (prior to backpropagation [17]) to solve a CartPole environment. This environment implementation constituted the first-ever standardized implementation of an

¹By "partially model-free" we define as an approach in which certain attributes of the environment, were abstracted away. such as the environment dynamics, while others, such as observation correlations, were not.

RL environment. It was used in other research to cross-evaluate learning algorithms [18], [19], [20], [21], [22], [23], [24], [25], [26] (these are just from 1984-1989).

In 1989, Q-Learning [27] marked the development of the first "general purpose" RL algorithm, which was capable of solving a diverse range of (then) state-of-the-art control problems. This was followed by other similar algorithms such as, SARSA [28] in 1993. Then, employing backpropagation [17] and deep neural networks, DQN [29] was created in 2013, solving a wide variety of ATARI video games and many real-world problems, effectively opening the road for modern deep reinforcement learning (DRL) algorithms (i.e., RL algorithms employing deep neural networks for function approximation).

It has to be noted that in 1990, SRV [30] was the first algorithm to learn on continuous action spaces. This was followed by REINFORCE in 1992 [31], which also learned on continuous action spaces and was model-free. REINFORCE was the basis for subsequent DRL algorithms, like DDPG [32] and TD3 [33].

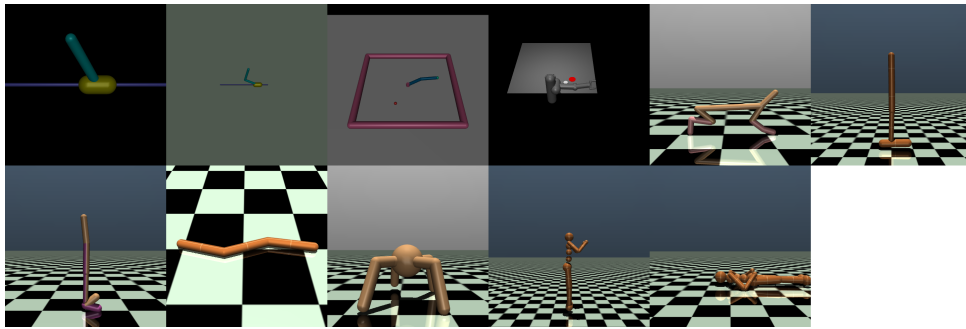


FIGURE 1.2: The collection of *Gymnasium/MuJoCo* environments, from left to right, (1, 2) *InvertedPendulum* and *InvertedDoublePendulum*, *CartPole* robots, (3,4) *Reacher* and *Pusher*, arm manipulator, (5,6,7) *HalfCheetah*, *Hopper* and *Walker2D*, 2d walking robots, (8, 9, 10) *Swimmer*, *Ant* and *Humanoid* 3d locomoting robots, (11) *HumanoidStandup*, a standup task.

Recently, an important milestone for the use of (deep) RL in many fields including robotics, was the creation of *OpenAI/Gym* [34], a Python library, in 2016. It provided two significant advancements to the field of RL: (1) The *gym.ENV* API, which enabled for the first time environments and learning algorithms to be interoperable, and (2) a collection of high-quality, well-documented RL environments were made available, including *Gym/MuJoCo*, a collection of 11 robot control environments based on the *MuJoCo* simulator [35] (Figure 1.2). The most challenging of the *OpenAI/Gym* environments, which had become a cornerstone of RL development, have been used to evaluate many of the state-of-the-art training algorithms, including PPO [36], TD3 [33] and SAC [37].

The *Gym/MuJoCo* environments have undergone several revisions, commencing with the release of version 0 in 2016 and followed shortly thereafter by version 1 in 2016, which addressed numerous bugs. The release of version 2 in 2018 saw the implementation of significant enhancements to the back-end physics simulation improvements by using the new (at the time) `mujoco-py==1.5`. The release of version 3 in 2018 introduced the ability to customize a limited number of environmental elements. The release of version 4 in 2022 also introduced significant enhancements to the back-end physics simulation improvements by moving to `mujoco==2.2` [35], but the customization options introduced in version 3 were removed.

The most significant limitation of the *Gym/MuJoCo* environments, across all versions, is that they use the same robot models. These models were created around 2012, preceding the "modern" robotics revolution. As a result, they are not representative of real robots (see example [38]).

In October 2022, the Farama Foundation was established. Its objective was to further develop the *OpenAI/Gym*, now known as *Farama/Gymnasium* [39] and many other open-source libraries. This development brings improvements to the API. These include a new API for vectorized environments, sibling projects that include new domain-specific environments like those in *Farama/Gymnasium-Robotics* [40], and the subject of this thesis, the new version of the *Gymnasium/MuJoCo-v5* framework/environments.

Real Applications of autonomous robotics include toy examples like, RoboCup, a soccer competition for robots founded in 1996, [41], [42], [43]. But RL robotics have not seen mass adoption in real-world industrial problems. Only some applications, such as pick-and-place tasks, object rotation, and contact-based object representation, have shown success in diverse lab environments for arm and hand manipulation tasks [44] [45] [46]. Most RL robotics research is conducted in simulations before being tested on real-world robots [47]. It is anticipated that tools like the one we provide in this thesis will assist researchers in developing new algorithms and testing them across a wide range of environments to address real-world problems.

1.1 Contributions

Our most significant contributions, that constitute the main part of this thesis, are:

- Development of the *Gymnasium/MuJoCo* version 5 environments and framework, which was integrated into the *Gymnasium* 1.0 release in October 2024. The initial alpha release, 1.0a1, was made available in February 2024.
- Creation of a set of baseline offline RL datasets for the *Gymnasium/MuJoCo-v5* environments using the *Minari* API [48]. The datasets were made

available in January 2025.

- Development of the *Gymnasium-Robotics/Maze-v5* environments and framework, based on *Gymnasium/MuJoCo*, which was included in the *Gymnasium-Robotics* 1.3 release in October 2024.
- Development of the *Gymnasium-Robotics/MaMuJoCo* version 0 and version 1 frameworks, multi agent factorizations of the *Gymnasium/MuJoCo* environments, based on [49]. This is the first standardized framework for multi-agent factorized robotics environments. Version 0 was included in release 1.2.1 in May 2024, while version 1 was integrated into release 1.3 in October 2024.

Additionally, as side-projects to this thesis, we conducted experiments to investigate:

- The performance impact of centralized critics versus decentralized critics in multi-agent factorized robotics environments and centralized critic re-use across different factorizations.
- The effect of observing contact forces on the Ant environment and other quadruped locomotion environments.

We present these results in Appendices C and D.

1.2 Thesis Outline

In Chapter 2, we present the background of RL, DRL, and MARL. In Chapter 3, we present our contributions to the new frameworks and environments. In Chapter 4, we present our conclusions and future work. In Appendix A, we present the full changelog for the *Gymnasium/MuJoCo-v5* environments. In Appendix B, we present the full changelog for the *Gymnasium-Robotics/MaMuJoCo* environments. In Appendix C, we present our evaluation experiments of centralized critics in multi-agent factorized robotics environments, and the transfer RL method of critic re-use. In Appendix D, we present our evaluation experiments of the impact of contact forces on the Ant environment and other quadruped locomotion environments.

Chapter 2

Background and Related Work

This chapter provides an overview of fundamental concepts and techniques in reinforcement learning. We cover deep reinforcement learning, multi-agent reinforcement learning, and offline RL. Additionally, we introduce the *Gymnasium* API and the *Gymnasium/MuJoCo* environments. These environments are widely used to evaluate the performance of reinforcement learning algorithms across a variety of challenging scenarios.

2.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making problems in a stochastic environment. MDPs are an extension of Markov chains that introduce the concept of actions and rewards. MDPs operate in discrete time steps.

An MDP consists of the following components:

- A set of possible states S , called the state space.
- A set of possible actions A , called the action space.
- An initial state distribution $I_s \mapsto S$.
- A state transition function $T(s, a) \mapsto S$.
- A reward function $R(s, a, s') \mapsto \mathbb{R}$.
- A termination function $T_{termination}(s) \mapsto \mathbb{B}$.

2.2 Partially Observable Markov Decision Process

Partially observable Markov Decision Process (POMDP) is a generalization of a Markov decision process (MDP). A POMDP models an agent's decision process in which it is assumed that the system dynamics are determined

by an MDP, but the agent cannot directly observe the underlying state. Instead, it receives observations that are probabilistically related to the underlying state. This partial observability introduces uncertainty into the agent's decision-making process, as it must infer the true state of the environment based on its observations.

Therefore, a POMDP additionally consists of the following components:

- A set of possible observations O , called the observation space.
- An observation stochastic function $O(s) \mapsto o$.

2.3 Reinforcement Learning

Reinforcement learning is a process wherein an intelligent agent interacts with a an MDP or more generally a POMDP environment, with the aim of determining a sequentially optimal course of actions (a “policy”), learning via trial and error. Specially, the agent learns a policy $\pi(o) \mapsto a$ that maps observations to actions, with the objective of maximizing the cumulative rewards over time.

2.4 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) integrates the principles of reinforcement learning with deep neural networks to tackle complex decision-making problems. This approach is particularly well-suited for environments with large state spaces, where traditional reinforcement learning methods may struggle due to their reliance on hand-crafted features and limited scalability.

In DRL, deep neural networks are used to approximate the value function ($V(s)$), the action-value function ($Q(s, a)$), or to directly learn a policy ($\pi(a|s)$). These networks serve as function approximators that can generalize from observed experiences, allowing agents to make informed decisions in previously unseen situations.

Key DRL algorithms include:

- **Critic-only Methods:** like Deep Q-Network (DQN) [29], the deep learning version of Q-Learning. It utilizes a deep neural network to approximate the action-value function ($Q(s, a)$). Experience Replay and Target Networks are employed to stabilize learning and improve generalization.

- **Actor-only Methods:** like Proximal Policy Optimization (PPO) [36] directly optimizes the policy using a clipped¹ surrogate objective function. This method ensures that updates to the policy remain close to the previous policy, facilitating stable and efficient training.
- **Actor-Critic Methods:** like Twin Delayed Deep Deterministic Policy Gradient (TD3) [33] and Soft Actor Critic (SAC) [37]. These methods combine an actor network, which learns the policy ($\pi(a|s)$), with a critic network Q , which estimates the value function ($V(s)$) or action-value function ($Q(s, a)$). This dual approach facilitates more effective learning and faster convergence.

Despite its successes, DRL faces challenges such as sample inefficiency and high computational demands. Ongoing research aims to address these issues through improved algorithm design and transfer learning to enhance the scalability and robustness of deep reinforcement learning systems.

2.5 Multi-agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) extends the traditional reinforcement learning framework to scenarios involving multiple interacting agents. These agents operate within a shared environment and must take actions to achieve their objectives while considering the impact of other agents' behaviors. MARL introduces additional complexity due to the non-stationary environment dynamics caused by the interactions among agents.

In MARL, multi-agent POMDPs, also known as partially observable Markovian games, model the environment. Each agent executes its own action (a_i) using its policy (π_i) and receives its own observation (o_i) and reward (r_i). The environment dynamics are influenced by the interactions among all agents, making the decision-making process more complex compared to single-agent scenarios. The primary goal is for each agent to learn an optimal policy that maximizes its cumulative rewards over time while taking into account the actions of other agents.

MARL can be categorized into three main types:

- **Cooperative MARL:** Agents work together towards a common objective. The reward functions are designed such that the collective performance is improved when agents coordinate their actions.
- **Competitive MARL:** Agents compete against each other to achieve their individual objectives. The reward structure often incentivizes adversarial behaviors, requiring strategies like deception or defense.
- **Mixed MARL:** Where agents may cooperate or compete with each other.

¹Clipping is the mathematical function that limits the value of a variable to a specified range, preventing it from exceeding a maximum or falling below a minimum threshold.

MARL training methods can be categorized into two main types:

- **Centralized Training Decentralized Execution (CTDE):** Agents are trained using global observation elements that are not available to individual agents during execution.
- **Decentralized Training Decentralized Execution (DTDE):** Agents are trained using only local observations and rewards, and must learn to cooperate or compete based on these limited inputs.

The applications of MARL are widespread across diverse fields, including autonomous driving, robotics, network security, and game playing, where multiple entities need to interact intelligently in dynamic environments.

2.6 Offline RL

Offline Reinforcement Learning (also known as batch RL) is a subfield of reinforcement learning that focuses on learning policies from previously collected data, without interacting with the environment. This approach is particularly useful when interaction with the environment is expensive, risky, or impossible. In traditional online reinforcement learning, an agent learns by trial and error, exploring the environment and receiving feedback in the form of rewards. However, in many real-world applications, such as health-care, finance, or robotics, it is not feasible or safe to have an agent explore the environment through trial and error.

Offline RL addresses this challenge by leveraging existing datasets of experiences, which can be collected from various sources, including human demonstrations, simulations, or previous experiments. The goal of offline RL is to learn a policy that maximizes the cumulative reward over a sequence of actions, based solely on the available offline data.

2.7 Gymnasium API

Farama/Gymnasium and its predecessor *OpenAI/Gym*, provide a simple Pythonic API, capable of representing general RL problems. The API consists of 3 core functions: `gymnasium.make()`, `env.reset()`, and `env.step()`.

- `gymnasium.make()` creates an environment instance by specifying the environment name and any additional parameters.
- `env.reset()` resets the environment to an initial state and returns:
 - *observation*: the initial observation of the environment.
 - *info*: a dictionary containing additional information such as the environment's state and metadata.

- `env.step()` executes an action in the environment and returns:
 - *observation*: the next observation from the environment.
 - *reward*: the reward received after taking the action.
 - *termination*: a status indicating whether the episode has ended due to reaching a terminal state.
 - *truncation*: a status indicating whether the episode has ended due to a time limit or other constraints.
 - *info*: a dictionary containing diagnostic information such as the state of the environment, debugging data, or additional metrics.

Additionally, the API provides functions for visualizing the environment, providing compatibility with the `gym.Env` API, debugging the environment, and more. A minimal example of the Gymnasium core API, demonstrating how to initialize, reset, and step through an environment, is shown in Listing 1.

2.8 Gymnasium/MuJoCo Environments

The Gymnasium/MuJoCo environments provide a diverse set of robot control tasks designed to evaluate reinforcement learning algorithms. The *Gymnasium/MuJoCo* environments are a collection of 11 robot control environments based on the *MuJoCo* simulator [35]. These environments are designed to test and evaluate reinforcement learning algorithms in a variety of challenging scenarios, including locomotion, manipulation, and control tasks. The environments are characterized by continuous action and observation spaces, making them suitable for deep reinforcement learning algorithms. The action spaces represent the control signals sent to the robots' actuators/-motors. The observation spaces consist of sensor readings from the robots. The reward function and termination conditions are specific to each environment and are designed to encourage the agent to learn the desired behavior. The initial state distribution is random, and the state transition function is deterministic.

2.8.1 Cartpole based environments

The *InvertedPendulum* environment is a Cartpole environment (first row, first column of Figure 1.2), based on the work of [16]. It is a simple 2D environment where the agent must balance a pole on a cart by applying forces to the cart. The environment is considered solved when the pole remains upright for a specified duration. The agent observes the cart's position and velocity, and the pole's angle and angular velocity, and applies a force to the cart to control the system, and receives a reward of +1 for each step the pole

```

import gymnasium

# Initialize the environment
env = gymnasium.make("LunarLander-v3",
                     render_mode="human")

# Reset the environment
# to generate the first observation
observation, info = env.reset(seed=42)
for _ in range(1000):
    # this is where you would insert your policy
    action = env.action_space.sample()

    # step (transition) through the environment
    # with the action
    # receiving the next observation, reward
    # and if the episode has terminated or truncated
    observation, reward, terminated, truncated, info
        = env.step(action)

    # If the episode has ended
    # then we reset to start a new episode
    if terminated or truncated:
        observation, info = env.reset()

env.close()

```

LISTING 1: This minimal example demonstrates how to initialize an environment, reset it to obtain the initial observation, and step through it using the Gymnasium core API. It highlights the basic interaction loop and the handling of episode termination and truncation. More examples can be found at <https://gymnasium.farama.org/>

remains upright. Unlike the original Cartpole environment, the *InvertedPendulum* environment has continuous observation and action spaces, making it more challenging for reinforcement learning algorithms.

The *InvertedDoublePendulum* environment is an extension of the *InvertedPendulum* environment, where a second pendulum is attached to the first pendulum (first row, second column of Figure 1.2). The addition of the second pendulum increases the complexity of the environment, making it more challenging for the agent to balance both pendulums simultaneously. The reward function is given by:

$$r = r_{\text{healthy}} - c_{\text{distance}} - c_{\text{velocity}}$$

where:

- $r_{healthy} = 10$ if the pole is upright; otherwise $r_{healthy} = 0$.
- $c_{distance} = 0.01 \cdot x_{pole2-tip}^2 + (y_{pole2-tip} - 2)^2$ is the cost associated with the distance from the upright position.
- $c_{velocity} = 10^{-3} \cdot \omega_{pole1-angle} + 5 \times 10^{-3} \cdot \omega_{pole2-angle}$ is the cost for moving too fast.

2.8.2 Arm manipulation based environments

The *Reacher* environment is a 2D robotic arm environment (first row, third column of Figure 1.2). The environment consists of a robotic arm with two joints that must reach a random target location. The significance of the 2D robotic arm lies in its simplicity and effectiveness for testing basic control algorithms. The 2 degrees of freedom of the robotic arm correspond to 2 continuous actions, making it more complex than the Cartpole environments. The reward function is given by:

$$r = r_{distance} - c_{ctrl}$$

where:

- $r_{distance} = -\|(P_{fingertip} - P_{target})\|_2$ is the reward for reaching the target.
- $c_{ctrl} = 0.1\|action\|_2^2$ is the cost of control.

The *Pusher* environment is a 3D robotic arm environment (first row, fourth column of Figure 1.2). The environment consists of a robotic arm with 3 joints that must push a cylindrical object to a random target location. The reward function is given by:

$$r = r_{distance} + r_{near} - c_{ctrl}$$

where:

- $r_{distance} = -\|(P_{object} - P_{target})\|_2$ is the reward for moving the object to the target.
- $r_{near} = -0.5\|(P_{fingertip} - P_{target})\|_2$ is the reward for moving the arm to the object.
- $c_{ctrl} = 0.1\|action\|_2^2$ is the cost of control.

2.8.3 2D Locomotion Environments

The *HalfCheetah* environment is a 2D cat-like robot environment based on the work of [50] (first row, fifth column of Figure 1.2). The robot's objective is to run as fast as possible while minimizing the control cost. The reward

function is given by:

$$r = r_{forward} - c_{ctrl}$$

where:

- $r_{forward} = \frac{dx}{dt}$ is the reward for moving forward.
- $c_{ctrl} = 0.1 \|action\|_2^2$ is the cost of control.

The *Hopper* environment is a 2D monoped robot environment based on the work of [51] (first row, sixth column of Figure 1.2). The robot consists of four body parts: the torso, the thigh, the leg, and the foot. The objective is to hop as far as possible while minimizing the control cost. The reward function is given by:

$$r = r_{healthy} + r_{forward} - c_{ctrl}$$

where:

- $r_{healthy} = 1$ if the *Hopper* is standing; otherwise $r_{healthy} = 0$.
- $r_{forward} = \frac{dx}{dt}$ is the reward for moving forward.
- $c_{ctrl} = 10^{-3} \|action\|_2^2$ is the cost of control.

The environment terminates when the *Hopper* falls over.

The *Walker2D* environment is a 2D bipedal robot environment, based on the *Hopper* (second row, first column of Figure 1.2). The robot consists of five body parts: the torso, the thigh, the leg, the foot, and the toes. The goal, reward function, and termination condition are similar to the *Hopper* environment.

2.8.4 Swimmer Environment

The *Swimmer* environment is a 2D swimming robot environment based on the work of [52] (second row, second column of Figure 1.2). The robot consists of three body parts: the torso, the tail, and the head, and 2 joints. The robot is submerged in water and must swim as fast as possible while minimizing the control cost. The reward function is given by:

$$r = r_{forward} - c_{ctrl}$$

where:

- $r_{forward} = \frac{dx}{dt}$ is the reward for moving forward.
- $c_{ctrl} = 10^{-4} \|action\|_2^2$ is the cost of control.

2.8.5 Quadruped Environment

The *Ant* environment is a 3D quadruped robot environment based on the work of [53] (second row, third column of Figure 1.2). The robot consists of four legs, each with two joints and 1 torso. Despite the name, the robot is not ant-sized but rather a 75cm tall and 911g heavy robot. The reward function is given by:

$$r = r_{healthy} + r_{forward} - c_{ctrl} - c_{contact}$$

where:

- $r_{healthy} = 1$ if the *Ant* is standing; otherwise $r_{healthy} = 0$.
- $r_{forward} = \frac{dx}{dt}$ is the reward for moving forward.
- $c_{ctrl} = 0.5 \|action\|_2^2$ is the cost of control.
- $c_{contact} = 5 \times 10^{-4} \|F_{contact}\|_2^2$ is the cost associated with contact forces.

The environment terminates when the *Ant* falls over.

2.8.6 Humanoids Environments

The *Humanoid* environment is a 3D bipedal robot environment based on the work of [54] (second row, fourth column of Figure 1.2). The robot consists of 11 body parts, 17 joints, and 2 tendons. The reward function is given by:

$$r = r_{healthy} + r_{forward} - c_{ctrl} - c_{contact}$$

where:

- $r_{healthy} = 5$ if the *Humanoid* is standing; otherwise $r_{healthy} = 0$.
- $r_{forward} = 1.25 \frac{dx}{dt}$ is the reward for moving forward.
- $c_{ctrl} = 0.1 \|action\|_2^2$ is the cost of control.
- $c_{contact} = 5 \times 10^{-7} \|F_{contact}\|_2^2$ is the cost associated with contact forces.

The environment terminates when the *Humanoid* falls over. The *Humanoid* environment is the most complex environment in the *Gymnasium/MuJoCo* environments.

The *HumanoidStandup* environment uses the same robot as the *Humanoid* environment, but the robot starts lying on the ground and must stand up (second row, fifth column of Figure 1.2). The reward function is $r = r_{up} + 1 - c_{ctrl} - c_{contact}$, where:

- $r_{up} = \frac{z_{after_action} - 0}{dt}$ is the reward for standing up, where z_{after_action} is the height of the humanoid after taking the action, and dt is the time step

duration.

- $c_{ctrl} = 0.1 \|action\|_2^2$ is the cost of control.
- $c_{contact} = 5 \times 10^{-7} \|F_{contact}\|_2^2$ is the cost associated with contact forces.

2.9 Minari

Minari[48] provides an API for offline RL and hosts a diverse collection of reference datasets. It is designed to facilitate the creation (example in Listing 3), sharing and usage of offline datasets (example in Listing 2), from environments that follow the family of Farama’s APIs including the Gymnasium API.

```
import minari

dataset = minari.load_dataset("D4RL/door/human-v2")
dataset.set_seed(seed=123)

for i in range(5):
    # sample 5 episodes from the dataset
    episodes = dataset.sample_episodes(n_episodes=5)
    # get id's from the sampled episodes
    ids = list(map(lambda ep: ep.id, episodes))
    print(f"EPISODE ID'S SAMPLE {i}: {ids}")
```

LISTING 2: Minimal example of loading a *Minari* dataset and sampling from it.

2.10 Related Work

In this section, we review existing open-source libraries and frameworks offering environments for robotic reinforcement learning. These libraries have been instrumental in advancing the field by offering standardized benchmarks and tools for evaluating RL algorithms.

Open source general-purpose robotic reinforcement learning environment libraries, such as *RoboSchool* [36] and *PyBullet Gymporium* [55], have been developed using Python. However, these libraries are not actively maintained and are limited to providing only the same type of simplified, not realistic robot models as the *Gymnasium/MuJoCo* environments, which do not accurately represent real-world robotic systems. The closest library to *Gymnasium-Robotics* is *Nvidia/IsaacLab* [9], which provides reference environments with realistic robot models but does not offer a general-purpose API for creating new environments with any robot model. Moreover, it is not open source and requires *Nvidia*-specific hardware.

```
import minari
import gymnasium as gym
from minari import DataCollector

env = gym.make('CartPole-v1')
env = DataCollector(env, record_infos=True)

total_episodes = 100

for _ in range(total_episodes):
    env.reset(seed=123)
    while True:
        # random action policy
        action = env.action_space.sample()
        obs, rew, terminated, truncated, info = env.step(action)

        if terminated or truncated:
            break

dataset = env.create_dataset(
    dataset_id="cartpole/test-v0",
    algorithm_name="Random-Policy",
    code_permalink=...
    author=...
    author_email=...
)
```

LISTING 3: Example of creating a *Minari* dataset with a random policy.

Chapter 3

New Environments and Frameworks

This chapter introduces the new environments and frameworks developed as part of the *Gymnasium* and *Gymnasium-Robotics* projects. Additionally, we present the new *Minari* offline RL datasets, which are used for training reinforcement learning agents offline. We clarify that all environments and frameworks presented in the subsequent sections in this chapter are all our implementations, developed as part of this thesis, unless explicitly stated otherwise.

3.1 Gymnasium/MuJoCo-v5 framework

The latest version, version 5, of the *Gymnasium/MuJoCo* environments was released with the *Gymnasium* 1.0 update. This new version offers a plethora of enhancements and improvements over version 4:

1. Third-party robot models, including realistic ones, can now be loaded.
2. Performance: Training performance improved by 5-7% for *Ant*, *Humanoid*, and *HumanoidStandup*.
3. Customizability: New arguments have been added for all environments, and arguments that were removed during the version 3 to version 4 transition for customizing the environments have been restored.
4. Quality of Life: New fields have been added to `info`, which now includes all reward components and non-observable state elements. Additionally, the `reset()` function now returns an `info` dictionary containing non-observable state elements.
5. A total of 24 bugs have been fixed.
6. The documentation has been updated to provide more detailed explanations of the observation, action, and reward functions.

More specifically,

3.1.1 Using Third-Party Robot Models

Version 5 not only enhances the existing environments but also facilitates the loading of third-party robot models via the `xml_file` argument. Users can provide their own model in either MJCF or URDF format. An illustrative example of this process can be seen in Listing 4, where a third-party robot model is being loaded (the Unitree Go1 robot model). The resulting environment is shown in Figure 3.1. Once the argument has been provided, the class will proceed to create the environment. It will load the model into the simulator and set the observation function, reward function, and initialization function of the environment. The class creates the environment using the provided argument, loads the model into the simulator, and sets the observation function, reward function, and initialization function of the environment. The supported robot types include quadrupeds (e.g., dog-like), bipedal (e.g., humanoids), and crawlers, with limited support for Hands, Arms, and Aerial Robots. This modification is noteworthy for two reasons. Most notably, it removes barriers to training real robots using RL in a simulator. Secondly, it provides a means of evaluating the efficacy of novel RL training algorithms.

3.1.2 Performance Improvements

In the *Ant*, *Humanoid*, and *HumanoidStandup* environments, the observation function included elements that always had a constant value of 0, regardless of the current state. In the *Ant* environment, 6 out of 111 observation elements had a constant value of 0, while in the *Humanoid* and *HumanoidStandup* environments, 28 out of 376 observation elements had a constant value of 0. These observations were of no benefit to the learning algorithms and resulted in a slight decrease in runtime performance. The correction of this programming error resulted in a 5-7% increase in learning speed. This was determined by measuring the elapsed time for training SB3/TD3 [56].

3.1.3 Increased Configurability

In version 3, a few arguments existed that allowed the configuration of certain environment behaviors. However, these were removed in version 4. Version 5 reintroduces the previously removed arguments and systematically adds new arguments for the environments/frameworks to customize every component of the reward function, termination conditions, and episode initial state distribution, and to include/exclude observation elements. For example, the *Ant* environment now has the *forward_reward_weight*, *ctrl_cost_weight*, and *contact_cost_weight* arguments, which allow for easy customization of the reward function. Furthermore, this is also advantageous when used in conjunction with the loading of third-party models.

3.1.4 *info* fields

The functions `env.reset()` and `env.step()` in the *gymnasium* API return an `info` dictionary. This dictionary includes additional information intended for the user rather than for the training algorithm. In version 5 of *Gymnasium/MuJoCo*, the `env.step()` function provides more `info` and now fully includes all reward components and non-observable state elements. The `env.reset()` function now returns non-observable state elements in the `info` dictionary, whereas previously it returned an empty dictionary.

3.1.5 Bug Fixes

- The coefficient of friction of *Walker2D*'s feet was different for each foot. The coefficient of friction of the right foot was $\mu = 0.9$, while the left foot had $\mu = 1.9$, resulting in an unnatural walking motion. This issue has been rectified in version 5, wherein both feet now have a friction coefficient of $\mu = 1.9$, resulting in a more natural walking motion.
- In the *Pusher* environment, the cylinder movable object's density was lighter than air, resulting in both unrealistic pushing behavior and sometimes unexpected behavior. The density was increased to a more realistic value.
- *healthy_reward* is a reward component that is given to the *Ant*, *Hopper*, *Humanoid*, *InvertedDoublePendulum*, *InvertedPendulum*, and *Walker2d* environments when they are "healthy". By "healthy", we mean that they have not fallen over. This incentivizes the training agent not to fall over. In version 4 and earlier, the value of *healthy_reward* was constantly 1, regardless of if the robot had actually fallen over. Now, in version 5, it is given properly, resulting in faster early learning (the agent learns not to fall over more quickly). Ensuring the proper functioning of this reward component is imperative for the development of realistic robots, as falling over can result in significant damage to the robot.
- The *contact_cost* is a reward component that penalizes the *Ant* and *Humanoid* for taking actions that have high contact forces, but in version 4 and earlier, the value of *contact_cost* was constantly 0. In version 5, it is now $c_{contact} = w_{contact} ||F_{contact}||^2$, which did not have a significant impact on the learning performance because of the low default value of *contact_cost_weight* $w_{contact} = 0.5 \cdot 10^{-3}$. Ensuring the proper functioning of this reward component is imperative for realistic robots, as high contact forces could potentially compromise their integrity.
- *Reacher*'s and *Pusher*'s reward function was calculated based on the previous state before it was updated for the new step, not the current state after the step was executed. In practice, it did not have a significant impact on learning performance with state-of-the-art RL algorithms

but could theoretically cause training instability on other training algorithms.

- Fixed several `info` fields.
 - Ant: Fixed `info["reward_ctrl"]` sometimes containing `contact_cost` instead of `ctrl_cost`.
 - Ant: Fixed `info["x_position"]`, `info["y_position"]` and `info["distance_from_origin"]` giving `xpos` instead of `qpos` state (`xpos` state are behind `1 mj_step()`).
 - *Pusher* and *Reacher*: Fixed `info["reward_ctrl"]` not being multiplied by the reward weight.

3.1.6 Documentation

A substantial portion of the documentation has been revised, and numerous technical inaccuracies have been rectified.

A full changelog of *Gymnasium/MuJoCo-v5* can be found in appendix A.

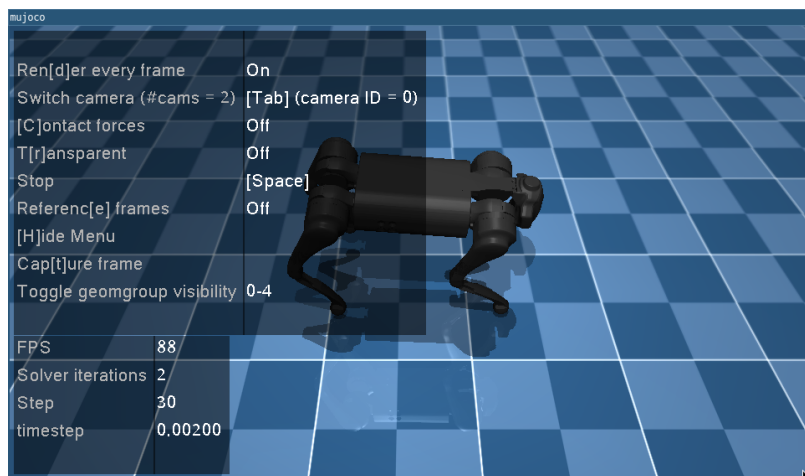


FIGURE 3.1: Unitree Go1 Robot Model in a *Gymnasium* environment, with the objective to locomote as fast as possible while minimizing total control. Code example at Listing 4

3.2 Offline RL Minari Datasets

Farama/Minari [48] is an API library for offline RL. A Minari dataset is a collection of episode trajectories that can be used to train a reinforcement learning agent offline. For each timestep, the observation, action, reward, and next observation are stored, in addition to termination, truncation signals, and info dictionaries. We are providing RL datasets for all *Gymnasium/MuJoCo* environments. A comprehensive list of these datasets can be found in Table 3.1. The datasets were created using a policy that was trained for a variable

```

import gymnasium

env = gymnasium.make(
    'Ant-v5', # Quadruped Framework
    xml_file='./mujoco_menagerie/unitree_go1/scene.xml',
    forward_reward_weight=1,
    ctrl_cost_weight=0.05,
    contact_cost_weight=5e-4,
    healthy_reward=1,
    main_body=1,
    healthy_z_range=(0.195, 0.75),
    include_cfrc_ext_in_observation=True,
    exclude_current_positions_from_observation=False,
    reset_noise_scale=0.1,
    frame_skip=25,
    max_episode_steps=1000,
)
... # run your RL algorithm

```

LISTING 4: Example of loading the Unitree Go1 Robot Model in a *Gymnasium* environment and defining the reward function parameters, observation space, starting state distribution, simulation speed, and episode truncation. More instructions can be found at the Tutorial: https://gymnasium.farama.org/main/tutorials/gymnasium_basics/load_quadruped_model/

length of time using stable baselines 3 algorithms [56]. Most environments were trained using SAC [37], which produced optimal agent performance in testing, but TQC [57] and PPO [36] were used for some environments.

Expert and medium datasets were created for the CartPole-based environments (*InvertedPendulum* and *InvertedDoublePendulum*). The expert dataset is generated from an agent that never falls over, while the medium dataset is from an agent that falls after 100 steps. Expert and medium datasets were created for the Arm manipulation environments (*Reacher* and *Pusher*). The expert dataset is from an agent that always achieves the objective quickly, while the medium dataset is from an agent that achieves it most of the time but slowly. Expert, medium, and simple datasets were created for the 2D locomotion environments (*HalfCheetah*, *Hopper*, *Walker2D*). In the *HalfCheetah* environment, the expert dataset is from an agent that runs as fast as possible with a natural motion. The medium dataset is from an agent that runs fast but slower than the expert, and the simple dataset is from an agent that runs slowly and occasionally falls over. In the *Hopper* environment, the expert dataset is from an agent that hops as fast as possible while remaining upright with a natural motion. The medium dataset is from an agent that hops fast while remaining mostly upright, and the simple dataset is from an agent that hops slowly while remaining mostly upright. In the *Walker2D*

environment, the expert dataset is from an agent that runs fast with a wide gait and a natural motion. The medium dataset is from an agent that walks fast, and the simple dataset is from an agent that walks slowly. In the *Swimmer* environment, the expert dataset is from an agent that swims fast with a serpentine motion, while the medium dataset is from an agent that swims slowly with a serpentine motion but also makes unnecessary movements. In the *Ant* environment, the expert dataset is from an agent that gallops quickly but occasionally falls over. The medium dataset is from an agent that walks fast without falling over, and the simple dataset is from an agent that walks slowly without falling over. In the *Humanoid* environment, the expert dataset is from an agent that runs. The medium dataset is from an agent that walks quickly, and the simple dataset is from an agent that walks slowly by sliding its feet. Non-expert datasets are important for evaluating the robustness of learning algorithms, as they provide more challenging scenarios. Expert datasets may be too easy for the algorithms to learn from. In the real world, learning often involves imperfect agents, such as humans.

The Datasets can be found at <https://minari.farama.org/datasets/mujoco/>.

3.3 Other Robotics Environments

Gymnasium/MuJoCo offers not only a set of environments but also a base class that is utilized across all environments in *Gymnasium-Robotics*, *Gymnasium/MuJoCo*, *MO-Gymnasium/MuJoCo* [59], and a few third-party environments. Consequently, a significant number of improvements to *Gymnasium/MuJoCo* are inherited by those projects.

3.3.1 Gymnasium-Robotics/Maze-v5

The *Gymnasium-Robotics/Maze* [60] contains environments in which a robot navigates a maze to reach a designated goal. In version 4, the robot can be either an abstract point or the *Gymnasium/Ant* robot. In version 5, the maze environments are capable of utilizing any robot model that is supported by *Gymnasium/MuJoCo* version 5 (example in Figure 3.2).

3.4 Gymnasium-Robotics/MaMuJoCo-v1 framework

MaMuJoCo [49] is a set of multiagent factorizations [61] of *Gym/MuJoCo* (Figure 3.5). Agent factorization, as described by [61], is the process of decomposing the domain's state-action space into subsets to be controlled by each agent and defining the communications between the agents (Figure 3.3).

In the context of factorization, the action space A must be partitioned into disjoint sub-action spaces ($\bigcup_{i \in I} a_i = A$, $\forall i \neq j \ a_i \cap a_j = \emptyset$) where I is the

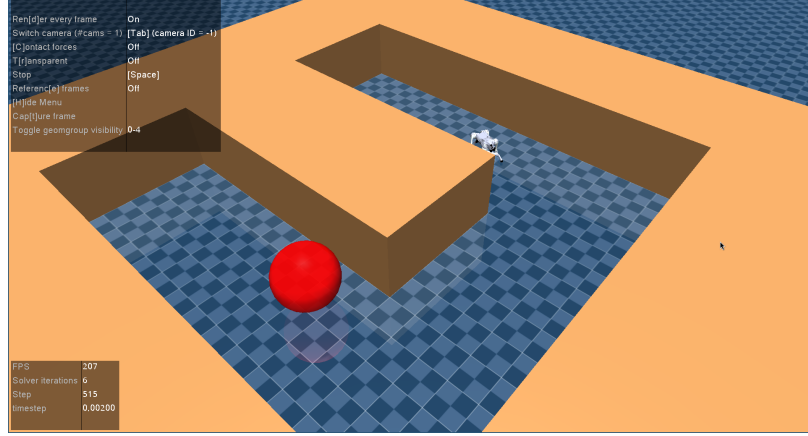


FIGURE 3.2: Example of the Unitree Go2 robot inside a trivial maze; the robot's objective is to reach the red goal.:

```
env = gymnasium.make('AntMaze_UMaze-v5',
xml_file="./mujoco_menagerie/unitree_go2/scene.xml",
maze_size_scaling=2, maze_height=0.4,
max_episode_steps=100)
```

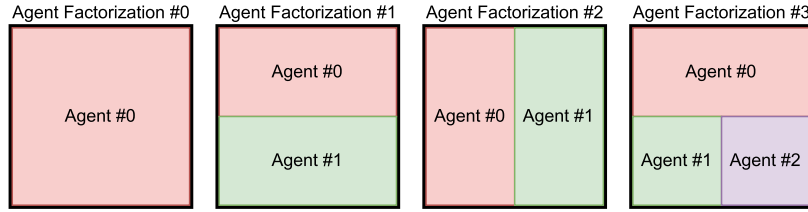


FIGURE 3.3: Agent factorization spaces, examples

index set of the action spaces. Furthermore, the observation space O can be partitioned in accordance with the specifications set forth by the designer, ranging from a scenario in which all agents have a comprehensive view of the system¹ to one in which they only have access to the observations directly associated with their actions.

The original implementation of *MaMuJoCo* constituted the basis for the subsequent developments of *Gymnasium-Robotics/MaMuJoCo-v0* and *Gymnasium-Robotics/MaMuJoCo-v1*. Here, we present the cumulative improvements resulting from both versions over the original version [49].

1. Based on *Gymnasium/MuJoCo-v5* instead of *Gym/MuJoCo-v2*, and therefore inherits all their improvements, including the ability to utilize third-party models with the `xml_file` argument.
2. Added support for custom agent factorizations and the `gym_env` argument, enabling the usage of environment wrappers. It can also be used to load third-party `Gymnasium.MujocoEnv` environments, such as a

¹It is important to acknowledge that this configuration is not logically coherent. In essence, this approach trains a single policy to generate a joint action for all agents within the environment.

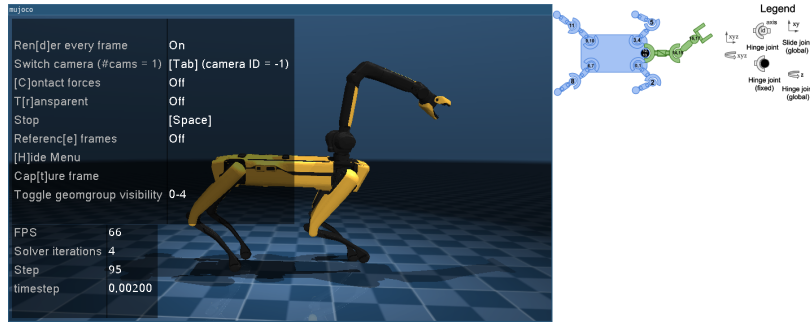


FIGURE 3.4: Example Factorization of Boston Dynamics's spot with arm robot. The robot is factorized into two agents: the quadruped agent and the arm agent. Code example can be found in the Documentation: <https://robotics.farama.org/envs/MaMuJoCo/#example-boston-dynamics-spot-arm-with-custom-quadruped-arm-factorization>

"*quadruped | arm*" factorization of spot with arm robot (Figure 3.4).

3. Uses *PettingZoo* APIs in lieu of an original API, allowing interoperability with the existing infrastructure of multi-agent reinforcement learning algorithms implementations.
4. Added new functions:


```
MultiAgentMujocoEnv.map_global_action_to_local_actions(),
MultiAgentMujocoEnv.map_local_actions_to_global_action(),
MultiAgentMujocoEnv.map_global_state_to_local_observations(),
MultiAgentMujocoEnv.map_local_observations_to_global_state(),
```

 which are useful for interoperation between factorized and single agent environments.
5. General code cleanup, factorization, type hinting, adding documentation, and comments.

This constitutes a significant development, as it represents the inaugural standardized framework for multi-agent factorizations, thereby enabling the comparison of different algorithms on multi-agent factorized environments.

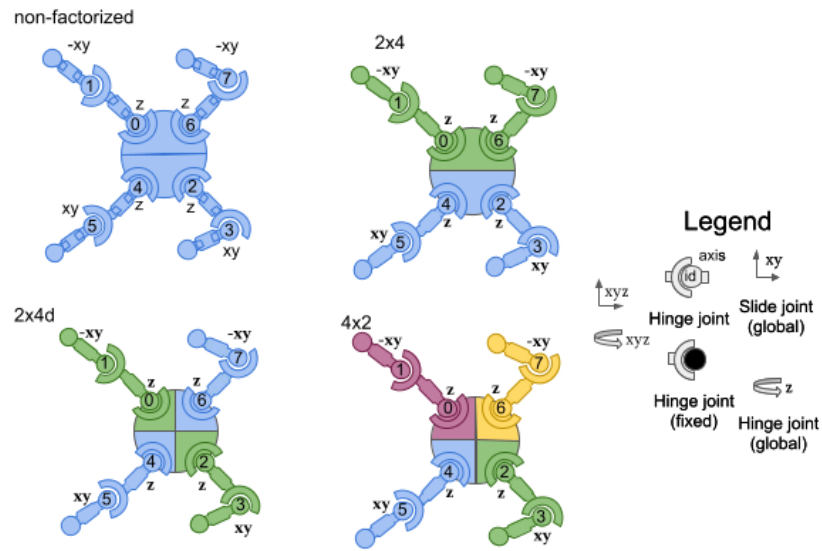


FIGURE 3.5: Sample factorizations of the *Gymnasium/Ant* environment in *MaMuJoCo*. Showing a vizualization the factorized action spaces. (Top Left) Single agent factorization, (Top Right) Two agent factorization, (Bottom Left) Two agent in a diagonal factorization, (Bottom Right) four agent factorization.

Environment - Dataset	Dataset Steps	Learning Algorithm	Training steps	Agent Return
<i>InvertedPendulum</i> "Medium"	$100 \cdot 10^3$	SB3/SAC	$10 \cdot 10^3$	131.00 ± 18.26
<i>InvertedPendulum</i> "Expert"	$100 \cdot 10^3$	SB3/SAC	10^6	1000.00
<i>InvertedDoublePendulum</i> "Medium"	$100 \cdot 10^3$	SB3/SAC	$50 \cdot 10^3$	5556.20 ± 3822.09
<i>InvertedDoublePendulum</i> "Expert"	$100 \cdot 10^3$	SB3/SAC	$10 \cdot 10^6$	9356.02 ± 0.06
<i>Reacher</i> "Medium"	$500 \cdot 10^3$	SB3/SAC	$100 \cdot 10^3$	-4.11 ± 1.65
<i>Reacher</i> "Expert"	$500 \cdot 10^3$	SB3/SAC	$10 \cdot 10^6$	-3.28 ± 1.38
<i>Pusher</i> "Medium"	10^6	SB3/SAC	10^6	-29.87 ± 4.91
<i>Pusher</i> "Expert"	10^6	SB3/SAC	$10 \cdot 10^6$	-22.05 ± 5.96
<i>HalfCheetah</i> "Simple"	10^6	SB3/TQC	10^6	7249.97 ± 103.59
<i>HalfCheetah</i> "Medium"	10^6	SB3/TQC	$6 \cdot 10^6$	15516.83 ± 134.50
<i>HalfCheetah</i> "Expert"	10^6	SB3/TQC	$25 \cdot 10^6$	17641.80 ± 61.88
<i>Hopper</i> "Simple"	10^6	SB3/SAC	10^6	3177.53 ± 17.49
<i>Hopper</i> "Medium"	10^6	SB3/SAC	$4 \cdot 10^6$	3483.58 ± 405.47
<i>Hopper</i> "Expert"	10^6	SB3/SAC	$25 \cdot 10^6$	4098.17 ± 247.70
<i>Walker2d</i> "Simple"	10^6	SB3/SAC	$1.5 \cdot 10^6$	4150.61 ± 66.36
<i>Walker2d</i> "Medium"	10^6	SB3/SAC	$6 \cdot 10^6$	6173.86 ± 197.86
<i>Walker2d</i> "Expert"	10^6	SB3/SAC	$25 \cdot 10^6$	6956.61 ± 15.95
<i>Swimmer</i> "Medium"	10^6	SB3/PPO ($\gamma = 1$)	$300 \cdot 10^3$	219.39 ± 4.99
<i>Swimmer</i> "Expert"	10^6	SB3/PPO ($\gamma = 1$)	$100 \cdot 10^6$	363.70 ± 1.81
<i>Ant</i> "Simple"	10^6	SB3/SAC	10^6	4195.36 ± 1097.35
<i>Ant</i> "Medium"	10^6	SB3/SAC	$10 \cdot 10^6$	6022.47 ± 1093.46
<i>Ant</i> "Expert"	10^6	SB3/SAC (<i>cfrs_obs=False</i>)	$10 \cdot 10^6$	6683.96 ± 732.38
<i>Humanoid</i> "Simple"	10^6	SB3/TQC	$2 \cdot 10^6$	5543.17 ± 825.31
<i>Humanoid</i> "Medium"	10^6	SB3/TQC	$5 \cdot 10^6$	8021.95 ± 912.19
<i>Humanoid</i> "Expert"	10^6	SB3/TQC	$20 \cdot 10^6$	10370.61 ± 1542.02

TABLE 3.1: Offline Minari Datasets for *Gymnasium/MuJoCo-v5*, "Dataset Steps" column shows the size of the generated datasets, "Learning Algorithm" shows the algorithm used and which options were used.

On *Swimmer*, the learning rate was set to $\gamma = 1$ because [58].

The Ant "expert" dataset was generated without *o_cfrs_ext*, because of appendix D.

Chapter 4

Conclusions and Future Work

This thesis has presented an in-depth exploration and advancement in the field of RL, with a particular focus on its application within robotic systems. The contributions aim to enhance both the capabilities and usability of RL environments, specifically through the development and enhancement of several key frameworks.

First, we introduced *Gymnasium/MuJoCo* version 5, a significant advancement that provides support for third-party robot models, performance improvements, increased configurability, enhanced information availability, and numerous bug fixes. This framework not only improves upon its predecessor but also lays the groundwork for future research by enabling researchers to train physical robots using RL algorithms.

Second, we developed offline RL datasets for the *Gymnasium/MuJoCo-v5* environments using the *Minari* API. These datasets offer a valuable resource for testing and benchmarking RL algorithms in controlled environments, thus facilitating more effective research and development.

Furthermore, we explored additional robotic environments through the development of *Gymnasium-Robotics/Maze-v5*, which extends maze-based navigation tasks to any robot model supported by *Gymnasium/MuJoCo*. This work enhances the versatility and applicability of RL in real-world scenarios.

Lastly, we presented advancements in multi-agent reinforcement learning through the development of the *Gymnasium-Robotics/MaMuJoCo-v1* framework. By leveraging agent factorizations and utilizing existing *PettingZoo* APIs, this framework facilitates interoperability with a wide range of multi-agent RL algorithms, providing a standardized platform for comparison and evaluation.

In summary, the contributions made in this thesis have significantly advanced the state of the art in RL robotic environments. These developments not only improve the functionality and usability of current frameworks but

also pave the way for future research that will further enhance the capabilities of intelligent agents in complex, dynamic environments. As reinforcement learning continues to evolve, the frameworks and datasets developed here are expected to play a crucial role in shaping the next generation of AI-driven systems in robotics.

4.1 Future Work

Gymnasium-Robotics should be adaptable to the needs of ongoing RL research, and any bugs that are found should be fixed. Parallelizing robotics environments on GPUs/NPUs using *MuJoCo-JAX* is an ongoing development topic.

Additional Minari datasets could be created for the *Gymnasium/MuJoCo* framework, incorporating third-party robot models in addition to the included ones.

Future versions of *Gymnasium/MuJoCo* could be extended to support additional robot types, such as aerial, aquatic, arms, and hands. Furthermore, expanding the range of robot tasks within *Gymnasium-Robotics* would be beneficial.

These projects are open to community contributions, and we encourage researchers to submit their own environments and datasets.

Appendix A

Gymnasium MuJoCo-v5 full changelog

- General
 - Minimum `mujoco` version is now 2.3.3.
 - Added support for fully custom/third party *MuJoCo* models using the `xml_file` argument (previously only a few changes could be made to the existing models).
 - Added `default_camera_config` argument, a dictionary for setting the `mj_camera` properties, mainly useful for custom environments.
 - Added `env.observation_structure`, a dictionary for specifying the observation space compose (e.g., `qpos`, `qvel`), useful for building tooling and wrappers for the MuJoCo environments.
 - Return a non-empty `info` with `reset()`, previously an empty dictionary was returned, the new keys are the same state information as `step()`.
 - Added `frame_skip`, used to configure the `dt` (duration of `step()`), default varies by environment check environment documentation pages.
- Ant:
 - Fixed bug: `healthy_reward` was given on every step (even if the *Ant* is unhealthy), now it is only given when the *Ant* is healthy. The `info["reward_survive"]` is updated with this change (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/526>).
 - The reward function now always includes `contact_cost`, before it was only included if `use_contact_forces=True` (can be set

to 0 with `contact_cost_weight=0`).

- Excluded the `cfrc_ext` of `worldbody` from the observation space, as it was always 0 and thus provided no useful information to the agent, resulting in slightly faster training (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/204>).
 - Added the `main_body` argument, which specifies the body used to compute the forward reward (mainly useful for custom *MuJoCo* models).
 - Added the `forward_reward_weight` argument, which defaults to 1 (effectively the same behavior as in *v4*).
 - Added the `include_cfrc_ext_in_observation` argument, previously in *v4* the inclusion of `cfrc_ext` observations was controlled by `use_contact_forces` which defaulted to `False`, while `include_cfrc_ext_in_observation` defaults to `True`.
 - Removed the `use_contact_forces` argument (note: its functionality has been replaced by `include_cfrc_ext_in_observation` and `contact_cost_weight`) (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/214>).
 - Fixed `info["reward_ctrl"]` sometimes containing `contact_cost` instead of `ctrl_cost`.
 - Fixed `info["x_position"]` & `info["y_position"]` & `info["distance_from_origin"]` giving `xpos` instead of `qpos` observations (`xpos` observations are behind 1 `mj_step()` more at <https://github.com/deepmind/mujoco/issues/889#issuecomment-1568896388>) (related GitHub issues <https://github.com/Farama-Foundation/Gymnasium/issues/521> & <https://github.com/Farama-Foundation/Gymnasium/issues/539>).
 - Removed `info["forward_reward"]` as it is equivalent to `info["reward_forward"]`.
- HalfCheetah:
 - Restored the `xml_file` argument (was removed in *v4*).
 - Renamed `info["reward_run"]` to `info["reward_forward"]` to be consistent with the other environments.

- Hopper:
 - Fixed bug: `healthy_reward` was given on every step (even if the *Hopper* was unhealthy), now it is only given when the *Hopper* is healthy. The `info["reward_survive"]` is updated with this change (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/526>).
 - Restored the `xml_file` argument (was removed in *v4*).
 - Added individual reward terms in `info` (`info["reward_forward"]`, `info["reward_ctrl"]`, `info["reward_survive"]`).
 - Added `info["z_distance_from_origin"]` which is equal to the vertical distance of the *"torso"* body from its initial position.
- Humanoid:
 - Fixed bug: `healthy_reward` was given on every step (even if the *Humanoid* was unhealthy), now it is only given when the *Humanoid* is healthy. The `info["reward_survive"]` is updated with this change (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/526>).
 - Restored `contact_cost` and the corresponding `contact_cost_weight` and `contact_cost_range` arguments, with the same defaults as in *Humanoid-v3* (was removed in *v4*) (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/504>).
 - Excluded the `cinert` & `cvel` & `cfrc_ext` of `worldbody` and `root/freejoint` `qfrc_actuator` from the observation space, as it was always 0 and thus provided no useful information to the agent, resulting in slightly faster training (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/204>).
 - Restored the `xml_file` argument (was removed in *v4*).
 - Added `include_cinert_in_observation`, `include_cvel_in_observation`, `include_qfrc_actuator_in_observation`, `include_cfrc_ext_in_observation` arguments to allow for the exclusion of observation elements from the observation space.
 - Fixed `info["x_position"]` & `info["y_position"]` &

`info["distance_from_origin"]` returning `xpos` instead of `qpos` based observations (`xpos` observations are behind 1 `mj_step()` more at (github.com/deepmind/mujoco/issues/889#issuecomment-1568896388) (related GitHub issues <https://github.com/Farama-Foundation/Gymnasium/issues/521> & <https://github.com/Farama-Foundation/Gymnasium/issues/539>).

- Added `info["tendon_length"]` and `info["tendon_velocity"]` containing observations of the *Humanoid*'s 2 tendons connecting the hips to the knees.
- Renamed `info["reward_alive"]` to `info["reward_survive"]` to be consistent with the other environments.
- Renamed `info["reward_linvel"]` to `info["reward_forward"]` to be consistent with the other environments.
- Renamed `info["reward_quadctrl"]` to `info["reward_ctrl"]` to be consistent with the other environments.
- Removed `info["forward_reward"]` as it is equivalent to `info["reward_forward"]`.
- **HumanoidStandup:**
 - Excluded the `cinert` & `cvel` & `cfrc_ext` of `worldbody` and `root/freejoint qfrc_actuator` from the observation space, as it was always 0, and thus provided no useful information to the agent, resulting in slightly faster training (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/204>).
 - Restored the `xml_file` argument (was removed in *v4*).
 - Added `uph_cost_weight`, `ctrl_cost_weight`, `impact_cost_weight`, `impact_cost_range` arguments to configure the reward function (defaults are effectively the same as in *v4*).
 - Added `reset_noise_scale` argument to set the range of initial states.
 - Added `include_cinert_in_observation`, `include_cvel_in_observation`, `include_qfrc_actuator_in_observation`,

`include_cfrc_ext_in_observation` arguments to allow for the exclusion of observation elements from the observation space.

- Added `info["tendon_length"]` and `info["tendon_velocity"]` containing observations of the *Humanoid*'s 2 tendons connecting the hips to the knees.
 - Added `info["x_position"]` & `info["y_position"]` which contain the observations excluded when `exclude_current_positions_from_observation == True`.
 - Added `info["z_distance_from_origin"]` which is the vertical distance of the *"torso"* body from its initial position.
- InvertedDoublePendulum:
 - Fixed bug: `healthy_reward` was given on every step (even if the Pendulum is unhealthy), now it is only given if the DoublePendulum is healthy (not terminated)(related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/500>).
 - Excluded the `qfrc_constraint` ("constraint force") of the hinges from the observation space (as it was always 0, thus providing no useful information to the agent, resulting in slightly faster training) (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/228>).
 - Added `xml_file` argument.
 - Added `reset_noise_scale` argument to set the range of initial states.
 - Added `healthy_reward` argument to configure the reward function (defaults are effectively the same as in *v4*).
 - Added individual reward terms in `info` (`info["reward_survive"]`, `info["distance_penalty"]`, `info["velocity_penalty"]`).
 - InvertedPendulum:
 - Fixed bug: `healthy_reward` was given on every step (even if the Pendulum is unhealthy), now it is only given if the Pendulum is healthy (not terminated) (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/>

[issues/500](#)).

- Added `xml_file` argument.
- Added `reset_noise_scale` argument to set the range of initial states.
- Added `info["reward_survive"]` which contains the reward.
- Pusher:
 - Fixed bug: increased the density of the object to be higher than air (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/950>).
 - Added `frame_skip` argument, used to configure the dt (duration of `step()`), default varies by environment check environment documentation pages.
 - Added `xml_file` argument.
 - Fixed bug: `reward_distance` & `reward_near` was based on the state before the physics step, now it is based on the state after the physics step (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/821>).
 - Added `reward_near_weight`, `reward_dist_weight`, `reward_control_weight` arguments to configure the reward function (defaults are effectively the same as in *v4*).
 - Fixed `info["reward_ctrl"]` not being multiplied by the reward weight.
 - Added `info["reward_near"]` which is equal to the reward term `reward_near`.
- Reacher:
 - Fixed bug: `reward_distance` was based on the state before the physics step, now it is based on the state after the physics step (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/821>).
 - Removed "*z - position_fingertip*" from the observation space since it is always 0 and therefore provides no useful information to the agent, this should result is slightly faster training (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/204>).
 - Added `xml_file` argument.

- Added `reward_dist_weight`, `reward_control_weight` arguments to configure the reward function (defaults are effectively the same as in *v4*).
- Fixed `info["reward_ctrl"]` not being multiplied by the reward weight.
- Swimmer:
 - Restored the `xml_file` argument (was removed in *v4*).
 - Added `forward_reward_weight`, `ctrl_cost_weight`, to configure the reward function (defaults are effectively the same as in *v4*).
 - Added `reset_noise_scale` argument to set the range of initial states.
 - Added `exclude_current_positions_from_observation` argument.
- Walker2d:
 - In *v2*, *v3* and *v4* the models have different friction values for the two feet (`left foot friction == 1.9` and `right foot friction == 0.9`). The *Walker-v5* model is updated to have the same friction for both feet (set to 1.9). This causes the *Walker2d*'s the right foot to slide less on the surface and therefore require more force to move (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/477>).
 - Fixed bug: `healthy_reward` was given on every step (even if the *Walker2D* is unhealthy), now it is only given if the *Walker2d* is healthy. The `info "reward_survive"` is updated with this change (related GitHub issue <https://github.com/Farama-Foundation/Gymnasium/issues/526>).
 - Restored the `xml_file` argument (was removed in *v4*).
 - Added individual reward terms in `info` (`info["reward_forward"]`, `info["reward_ctrl"]`, `info["reward_survive"]`).
 - Added `info["z_distance_from_origin"]` which is equal to the vertical distance of the *"torso"* body from its initial position.

Appendix B

Gymnasium-Robotics/MaMuJoCo Full Changelog

B.1 Gymnasium-Robotics/MaMuJoCo-v1

- General
 - Based on *Gymnasium/MuJoCo-v5* instead of *Gymnasium/MuJoCo-v4* (<https://github.com/Farama-Foundation/Gymnasium/pull/572>).
 - When `factorization=None`, the `env.gent_action_partitions.dummy_node` now contains `action_id` (it used to be `None`).
 - Added `map_local_observations_to_global_state` & optimized runtime performance of `map_global_state_to_local_observations`.
 - Added `gym_env` argument for using environment wrappers, also can be used to load third-party `Gymnasium.MujocoEnv` environments.
- Ant
 - Now observes `local_categories` of `cfrc_ext` by default (same as *Gymnasium/MuJoCo-v5/Ant*).
 - Renamed global node `torso` → `root`.
- Humanoid
 - No longer observes `qfrc_actuator` of `root` & `cinert`, `cvel`, `qfrc_actuator`, `cfrc_ext` of `worldbody` (same as *Gymnasium/MuJoCo-v5/Humanoid*).
- Humanoid Standup

- No longer observes `qfrc_actuator` of `root` & `cinert`, `cvel`, `qfrc_actuator`, `cfrc_ext` of `worldbody` (same as *Gymnasium/MuJoCo-v5/HumanoidStandup*).
- ManySegmentSwimmer
 - Now uses the same `option.timestep` as *Gymnasium/Swimmer* (0.01).
 - Updated model to work with `mujoco` $\geq 3.0.0$.
- Walker2d
 - Fixed bug: global nodes are now `[root_x, root_z, root_y]` (used to be `[root_x, root_x, root_z]`).

B.2 Gymnasium-Robotics/MaMuJoCo-v0

- General
 - Based on *Gymnasium/MuJoCo-v4* instead of *Gym/MuJoCo-v2*.
 - Uses *PettingZoo* APIs instead of an original API.
 - Added support for custom agent factorizations.
 - Added new functions
`MultiAgentMujocoEnv.map_global_action_to_local_actions`,
`MultiAgentMujocoEnv.map_local_actions_to_global_action`,
`MultiAgentMujocoEnv.map_global_state_to_local_observations`.
- Ant
 - Fixed diagonal factorization (" $2 \times 4d$ ") not being diagonal.
 - Fixed Global observations (The Ant's Torso: `rootx`, `rooty`, `rootz`) not being observed.
 - Changed action ordering to be same as *Gymnasium/MuJoCo/Ant*.
- Coupled Half Cheetah
 - Fixed action mapping of the second cheetah (It would previously not work)
 - Fixed tendon Jacobean observations
 - Added/Fixed Global observations (The Cheetahes's front tips: `rootxs`, `rootys`, `rootzs`) not being observed.

- Improved node naming
 - Changed action ordering to be same as *Gymnasium/MuJoCo/HalfCheetah*.
- Half Cheetah
 - Added/Fixed Global observations (The Cheetah’s front tip: `rootx`, `rooty`, `rootz`) not being observed.
 - Changed action ordering to be same as *Gymnasium/MuJoCo/HalfCheetah*.
- Hopper
 - Fixed Global observations (The *Hopper*’s top: `rootx`, `rooty`, `rootz`) not being observed.
- Humanoid
 - Added/Fixed Global observations (The *Humanoids*’s torso: `rootx`, `rooty`, `rootz`) not being observed.
 - Fixed abdomen observations ordering.
 - Added support for body observations (`cvel`, `cinert`, `cfrc_ext`)
 - Changed action ordering to be same as *Gymnasium/MuJoCo/Humanoid*.
- Humanoid Standup
 - Added/Fixed Global observations (The *Humanoids*’s torso: `rootx`, `rooty`, `rootz`) not being observed.
 - Fixed abdomen observations ordering.
 - Added support for body observations (`cvel`, `cinert`, `cfrc_ext`)
 - Changed action ordering to be same as *Gymnasium/MuJoCo/Humanoid*.
- Reacher
 - Added/Fixed Global observations (The *Targets*’s coordinates: `targetx`, `targety`) not being observed.
- Swimmer
 - Added/Fixed Global observations (The *Swimmer*’s front tip: `free_body_rot`) not being observed.

- Walker2d
 - Added/Fixed Global observations (The *Walker*'s top: `rootx`, `rooty`, `rootz`) not being observed.

Appendix C

Centralized Critic Experiment

We tested the performance of TD3 [33], MATD3 [62] and MATD3 with a centralized critic (MATD3-cc)¹ on *MuJoCo/Ant*. As illustrated in Figure C.1, the results demonstrate first that the baseline of single Agent TD3 is the best performing case. Secondly, a comparison of MATD3 and MATD3-cc reveals that MATD3-cc exhibits superior performance across all factorizations. Thirdly, an examination of the performance of different factorizations reveals that the *Ant/2x4* factorization performs similarly to the *Ant/2x4d* factorization, while the most decentralized factorization, *Ant/4x2* performs the worst.

Algorithm 1 presents the MATD3-cc algorithm, which is a modification of MATD3 that utilizes a centralized critic.

Further research is required to investigate the advantages and disadvantages of centralized critic approaches in cooperative environments. It is possible to apply the centralized critic in other actor-critic multi-agent algorithms. Prior to this work, the centralized critic has previously been utilized in COMA [63], yet a comparison with a decentralized critic has not been conducted. To date, there hasn't been sufficient comparative analysis of centralized and decentralized/independent critics, the only existing work in this area is that of [64].

C.1 Centralized Critic Re-use Policy Reconstruction Across Different Factorizations

A noteworthy similarity of the TD3 and MATD3-cc algorithms is the sharing of the critic architecture across all factorizations. Consequently, a trained critic can be utilized in any of these factorizations by simply reusing it.

As shown in Figure C.1, the best policy of single agent TD3 achieved a return of 6556.94. Upon reusing the corresponding critic, *Ant/2x4* gave a return of 6154.69, *Ant/2x4d* gave a return of 6196.59, and *Ant/4x2* gave a return of

¹An alternative full name would be "Multi Agent Centralized Twin Delayed Critic Deterministic Policy Gradient"

Algorithm 1: MATD3 with Centralized critic.**Data:** Hyperparameters:

- Discount factor γ
- Network architectures
- Experience Replay Buffer Size \mathcal{RB}_{size}
- Mini batch sizes \mathcal{MB}_{size-Q} , $\mathcal{MB}_{size-\pi}$
- Target update rates τ_Q , τ_π

Initialize the Experience Replay Buffer \mathcal{RB} , networks Q , π_i with parameters θ_Q , θ_{π_i} , target networks Q' , π'_i

for $timestep \in \{1, \dots, T_{Max}\}$ **do**

select $(a_1, \dots, a_N) \sim \pi_i(o_i) + \epsilon$, $\epsilon \sim \mathcal{N}_{0, I_N \cdot \sigma}$

step with actions (a_1, \dots, a_N) , get local observation (o'_1, \dots, o'_N) , global reward r , terminated signal T

store transition $(o_1, \dots, o_N, a_1, \dots, a_N, r, T, o'_1, \dots, o'_N)$ in \mathcal{RB}

$o_1, \dots, o_N = o'_1, \dots, o'_N$

Update Centralized Twin Critic

Sample a random mini-batch of \mathcal{MB}_{size-Q} samples

$(o_1^b, \dots, o_N^b, a_1^b, \dots, a_N^b, r^b, o_1'^b, \dots, o_N'^b, T^b)$ from \mathcal{RB}

$a_1', \dots, a_N' = \pi'_i(o_1^b, \dots, o_N^b) + \epsilon$, $\epsilon \sim clip(\mathcal{N}_{0, \sigma'}, \pm c')$

$y = r^b + \gamma \cdot \neg T^b \cdot \min(Q_{\theta'}(o_1^b, \dots, o_N^b, a_1', \dots, a_N'))$

update twin critic $\theta_Q \leftarrow \frac{1}{\mathcal{MB}_{size-Q}} \sum_b (y - Q_{\theta}(o_1^b, \dots, o_N^b, a_1^b, \dots, a_N^b))^2$

if $t \bmod d$ **then**

Update Actors

for agent $i \in 1, \dots, N$ **do**

Sample a random mini-batch of $\mathcal{MB}_{size-\pi}$ samples

$(o^b, a^b, _, _, _)$ from \mathcal{RB}

Update policy π with deterministic policy gradient:

$\nabla_{\theta_{\pi_i}} J(\theta_{\pi_i}) \approx$

$\frac{1}{\mathcal{MB}_{size-\pi}} \sum_b \nabla_{\theta} \pi_{\theta_{\pi_i}}(o_i^b) \nabla_{a_i} Q_1(o^b, a_1^b, \dots, \pi_{\theta_{\pi_i}}(o_i), \dots, a_N^b)$

Update target networks

$\theta'_Q \leftarrow \tau_Q \theta_Q + (1 - \tau_Q) \theta'_Q$

$\theta'_\pi \leftarrow \tau_\pi \theta_\pi + (1 - \tau_\pi) \theta'_\pi$

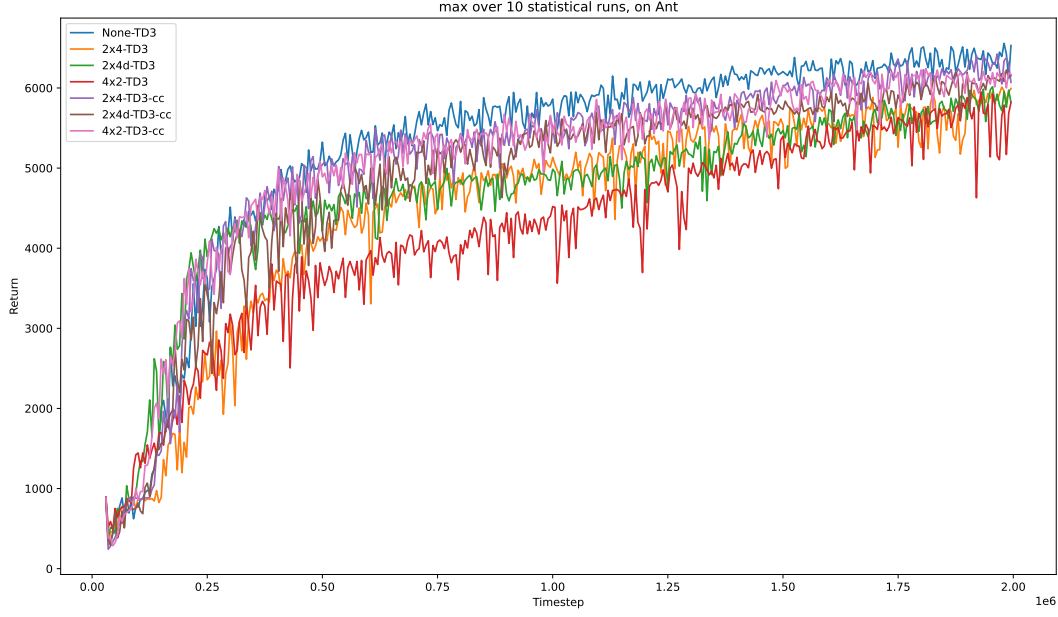


FIGURE C.1: Experiments on *MuJoCo/Ant* and *MaMuJoCo/Ant* with TD3, MATD3, and MATD3-cc. The X-axis shows the training time steps, and the Y-axis shows the max agent episodic return.

(Blue Line, 1st in legend) Single Agent TD3.
 (2nd, 3rd, 4th in legend) MATD3 of the various factorized cases.
 (5th, 6th, 7th in legend) MATD3-cc of the various factorized cases.

6270.75. The discrepancy in performance can be attributed, at least in part, to the inability to identify the optimal critic through the training process.

This training paradigm may be considered as an alternative to Centralized Training Decentralized Execution (CTDE), in which the multi-agent system is trained using global observation elements not available to individual agents during execution [65]. The training paradigm we used is that of Single Agent Training Transfer to Decentralized Execution (SATDE), in which a single agent is trained, and then the policy is transferred to a multi-agent system through the reuse of the centralized critic. However, additional testing is necessary to substantiate the efficacy of this approach.

Appendix D

Impact of Contact Forces Observations on Quadruped Experiment

The *Gymnasium/MuJoCo/Ant*'s observation space, consists of three components: the position of the body parts (o_{qpos}), the velocity of those parts (o_{qvel}), and external contact forces of those parts from the ground (o_{cfrc_ext}).

The *MuJoCo/Ant* was tested with and without the external contact forces observation elements (o_{cfrc_ext}). The results of SAC and TD3 are shown in Figure D.1, we also tested PPO and A2C, but they did not produce useful results. Two points are evident from the results: First, SAC outperforms TD3 (same as [66]). Secondly, when external contact forces are excluded, superior performance is observed, although the episode variance is markedly elevated. This can be attributed to the higher effective exploration rate, which can be attributed to the observation-action space representing a smaller percentage of the total space being explored.

Indeed, the performance was markedly superior, reaching a record-breaking level in comparison to all previous bests. The return was approximately +900 points higher than the previous best (Table D.1). This policy was used for the *Minari's Ant-expert* dataset¹

An identical test was conducted on the Unitree Go1 robot, (Figure D.2). As with the *MuJoCo/Ant-v5* robot, a clear performance improvement was observed when external contact forces were not present, accompanied by a slight increase in training variance. One potential explanation for this discrepancy is that the agent is more adept at maintaining its balance while running (Figure D.3).

¹It should be noted that there was a discrepancy between the return performance difference $6702.535 \neq 6683.96$. This can be attributed to the use of different versions of the MuJoCo simulator. The aforementioned experiments were conducted using version 2.3.3, whereas the *Minari* datasets were created using version 3.2.3.

Algorithm	Environment	Best episodic return
RLzoo's TD3 (Previous best)	Ant-v3	5813.274
Mine TD3 without CTN	Ant-v4	6169.537
Mine TD3 with CTN	Ant-v4	5284.448
Mine TD3 without CTN	Ant-v5	5692.833
Mine TD3 with CTN	Ant-v5	5975.278
Mine SAC without CTN	Ant-v5	6702.535
Mine SAC with CTN	Ant-v5	6582.3189

TABLE D.1: Comparison of learning algorithms and the impact of $o_{cfr_{ext}}$ in the Ant environment.

Note: even though we are comparing different revisions of Gymnasium/MuJoCo/Ant, the differences are within $\pm 2\%$.

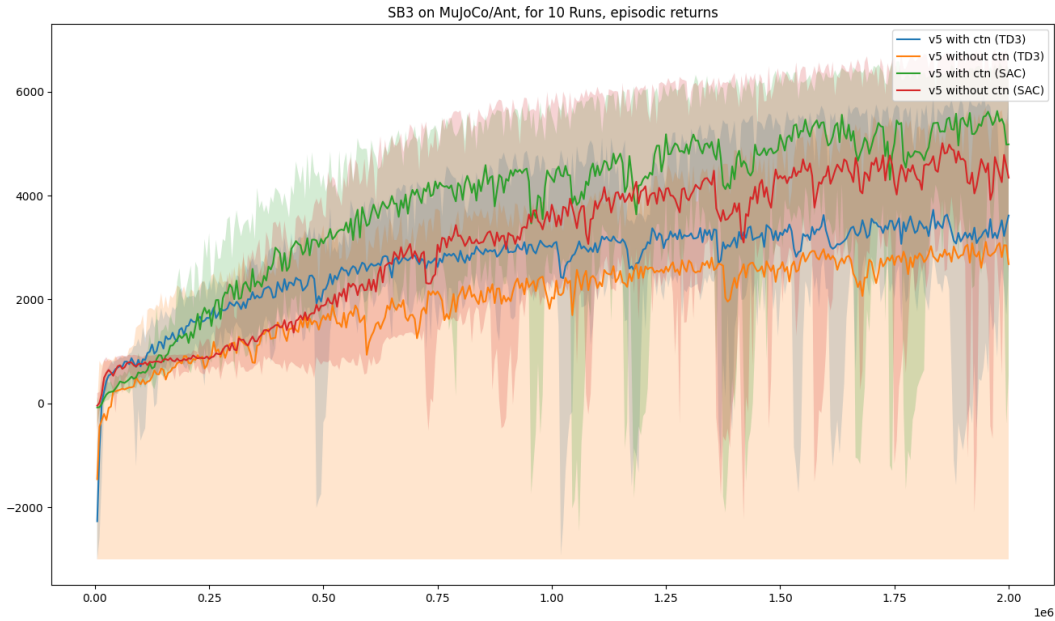


FIGURE D.1: Experiment on *MuJoCo/Ant-v5*, The X-axis shows the training time steps, and the Y-axis shows the maximum agent episodic return.

Two Algorithms show here are TD3 and SAC, with subcases for with and without external contact forces being observed ($o_{cfr_{ext}}$).

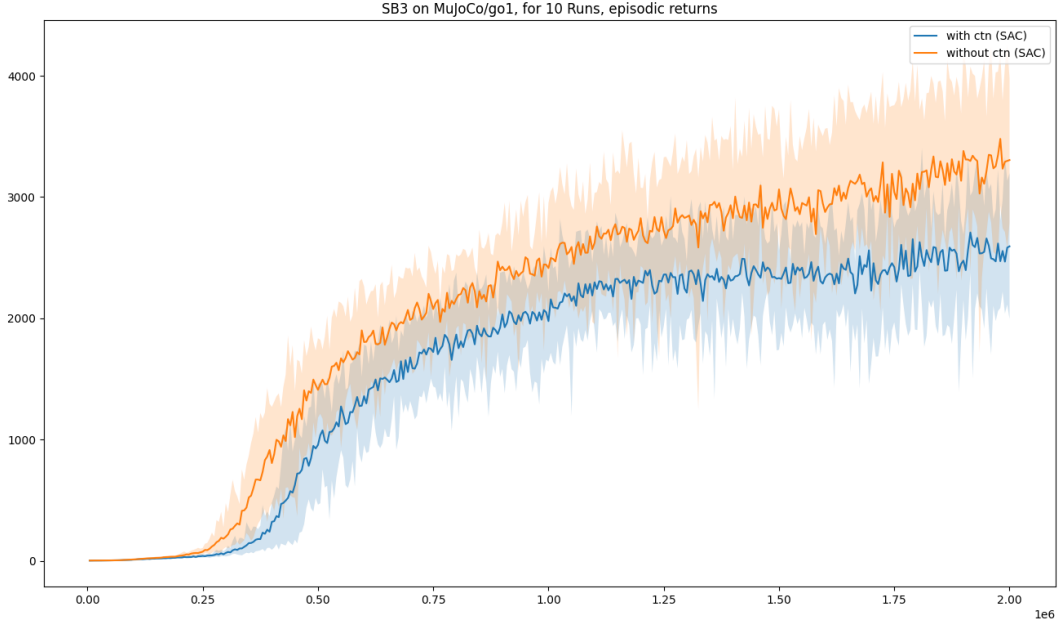


FIGURE D.2: Experiment on Unitree Go1, The X-axis shows the training time steps, and the Y-axis shows the maximum agent episodic return.

Two Algorithm show here is SAC, with subcases for with and without external contact forces being observed ($o_{cfr_{ext}}$).

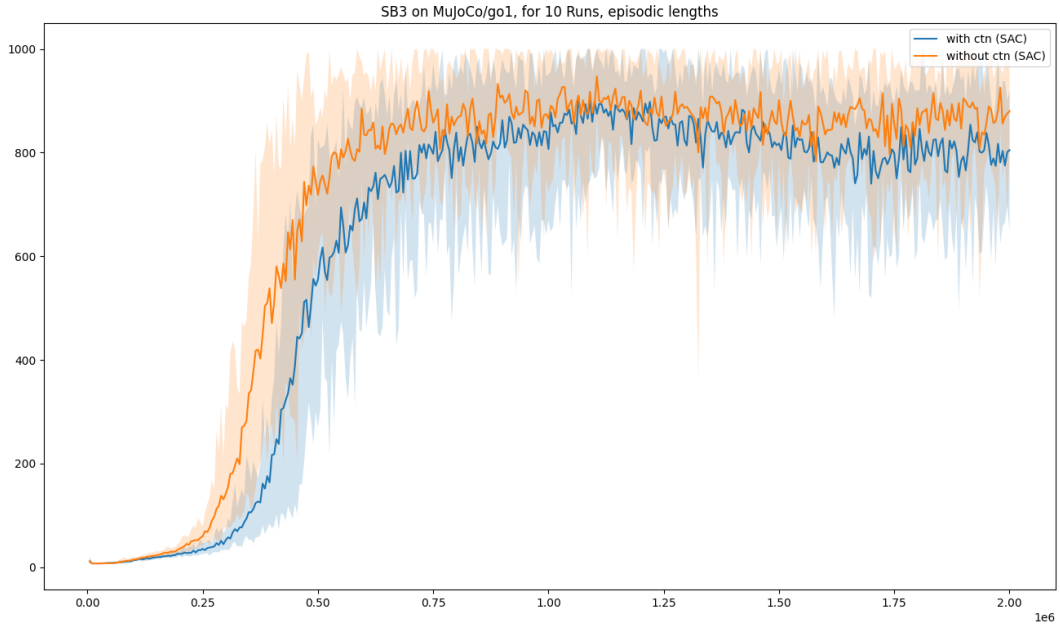


FIGURE D.3: Experiment on Unitree Go1, The X-axis shows the training time steps, and the Y-axis shows the average episode length.

Two Algorithm show here is SAC, with subcases for with and without external contact forces being observed ($o_{cfr_{ext}}$).

Bibliography

- [1] L. N. Alegre, *SUMO-RL*, <https://github.com/LucasAlegre/sumo-rl>, 2019.
- [2] K. Choudhary, D. Gupta, and P. S. Thomas, “ICU-Sepsis: A benchmark MDP built from real medical data”, in *Reinforcement Learning Conference*, 2024. [Online]. Available: <https://arxiv.org/abs/2406.05646>.
- [3] M. A. Haghpanah, *gym-mtsim*, version 1.1.0, Sep. 2021.
- [4] S. Orfanoudakis, C. Diaz-Londono, Y. E. Yilmaz, P. Palensky, and P. P. Vergara, *Ev2gym: A flexible v2g simulator for ev smart charging research and benchmarking*, 2024. arXiv: [2404.01849](https://arxiv.org/abs/2404.01849).
- [5] B. Shminke, “Gym-saturation: Gymnasium environments for saturation provers (system description)”, in *Automated Reasoning with Analytic Tableaux and Related Methods*, R. Ramanayake and J. Urban, Eds., Cham: Springer Nature Switzerland, 2023, pp. 187–199, ISBN: 978-3-031-43513-3.
- [6] S. Schneider, S. Werner, R. Khalili, A. Hecker, and H. Karl, “Mobile-env: An open platform for reinforcement learning in wireless mobile networks”, in *Network Operations and Management Symposium (NOMS)*, IEEE/IFIP, 2022.
- [7] J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang, “Safety gymnasium: A unified safe reinforcement learning benchmark”, in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: <https://openreview.net/forum?id=WZmlxIuIGR>.
- [8] Z. Yuan, A. W. Hall, S. Zhou, L. Brunke, M. Greeff, J. Panerati, and A. P. Schoellig, *Safe-control-gym: A unified benchmark suite for safe learning-based control and reinforcement learning*, 2021. arXiv: [2109.06325](https://arxiv.org/abs/2109.06325) [cs.R0].
- [9] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, “Orbit: A unified simulation framework for interactive robot learning environments”, *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023. DOI: [10.1109/LRA.2023.3270034](https://doi.org/10.1109/LRA.2023.3270034).
- [10] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, “Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control”, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*

- (IROS), 2021, pp. 7512–7519. DOI: [10 . 1109 / IROS51168 . 2021 . 9635857](https://doi.org/10.1109/IROS51168.2021.9635857).
- [11] R. T. Lange, *gymnax: A JAX-based reinforcement learning environment library*, version 0.0.4, 2022. [Online]. Available: <http://github.com/RobertTLange/gymnax>.
 - [12] X. Gong, H. Wang, D. Feng, and W. Wang, “Flycraft: An efficient goal-conditioned reinforcement learning environment for fixed-wing uav velocity vector control”, 2024.
 - [13] M. A. Stephenson and H. Schaub, “BSK-RL: Modular, High-Fidelity Reinforcement Learning Environments for Spacecraft Tasking”, in *75th International Astronautical Congress*, Milan, Italy: IAF, Oct. 2024.
 - [14] D. Michie, “Experiments on the mechanization of game-learning part i. characterization of the model and its parameters”, *Comput. J.*, vol. 6, pp. 232–236, 1963. [Online]. Available: <https://api.semanticscholar.org/CorpusID:62514450>.
 - [15] D. Michie and R. A. Chambers, “Boxes: An experiment in adaptive control”, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:18229198>.
 - [16] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983. DOI: [10.1109/TSMC.1983.6313077](https://doi.org/10.1109/TSMC.1983.6313077).
 - [17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: <https://api.semanticscholar.org/CorpusID:205001834>.
 - [18] O. G. Selfridge, R. S. Sutton, and A. G. Barto, “Training and tracking in robotics”, in *International Joint Conference on Artificial Intelligence*, 1985. [Online]. Available: <https://api.semanticscholar.org/CorpusID:5776535>.
 - [19] C. W. Anderson, “Learning to control an inverted pendulum with connectionist networks”, in *1988 American Control Conference*, 1988, pp. 2294–2298. DOI: [10.23919/ACC.1988.4790107](https://doi.org/10.23919/ACC.1988.4790107).
 - [20] Guez and Selinsky, “A neuromorphic controller with a human teacher”, in *IEEE 1988 International Conference on Neural Networks*, 1988, 595–602 vol.2. DOI: [10.1109/ICNN.1988.23976](https://doi.org/10.1109/ICNN.1988.23976).
 - [21] Tolat and Widrow, “An adaptive ‘broom balancer’ with visual inputs”, in *IEEE 1988 International Conference on Neural Networks*, 1988, 641–647 vol.2. DOI: [10.1109/ICNN.1988.23982](https://doi.org/10.1109/ICNN.1988.23982).
 - [22] C. Anderson, “Learning to control an inverted pendulum using neural networks”, *IEEE Control Systems Magazine*, vol. 9, no. 3, pp. 31–37, 1989. DOI: [10.1109/37.24809](https://doi.org/10.1109/37.24809).
 - [23] V. Chen and Y.-H. Pao, “Learning control with neural networks”, in *Proceedings, 1989 International Conference on Robotics and Automation*, 1989, 1448–1453 vol.3. DOI: [10.1109/ROBOT.1989.100183](https://doi.org/10.1109/ROBOT.1989.100183).
 - [24] A. Guez and J. Selinsky, “Neurocontroller design via supervised and unsupervised learning”, *Journal of Intelligent and Robotic Systems*,

- vol. 2, pp. 307–335, 1989. [Online]. Available: <https://api.semanticscholar.org/CorpusID:45905863>.
- [25] E. Grant and B. Zhang, “A neural-net approach to supervised learning of pole balancing”, in *Proceedings. IEEE International Symposium on Intelligent Control 1989*, 1989, pp. 123–129. DOI: [10.1109/ISIC.1989.238707](https://doi.org/10.1109/ISIC.1989.238707).
- [26] T. Troudet and W. Merrill, “Neuromorphic learning of continuous-valued mappings in the presence of noise: Application to real-time adaptive control”, in *Proceedings. IEEE International Symposium on Intelligent Control 1989*, 1989, pp. 312–319. DOI: [10.1109/ISIC.1989.238676](https://doi.org/10.1109/ISIC.1989.238676).
- [27] C. J. C. H. Watkins, “Learning from delayed rewards”, 1989.
- [28] G. Rummery and M. Niranjan, “On-line q-learning using connectionist systems”, *Technical Report CUED/F-INFENG/TR 166*, Nov. 1994.
- [29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, 2013. arXiv: [1312.5602](https://arxiv.org/abs/1312.5602) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1312.5602>.
- [30] V. Gullapalli, “A stochastic reinforcement learning algorithm for learning real-valued functions”, *Neural Networks*, vol. 3, pp. 671–692, 1990. [Online]. Available: <https://api.semanticscholar.org/CorpusID:21486916>.
- [31] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning”, *Machine Learning*, vol. 8, pp. 229–256, 1992. [Online]. Available: <https://api.semanticscholar.org/CorpusID:19115634>.
- [32] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, 2019. arXiv: [1509.02971](https://arxiv.org/abs/1509.02971) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1509.02971>.
- [33] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018. arXiv: [1802.09477](https://arxiv.org/abs/1802.09477) [cs.AI]. [Online]. Available: <https://arxiv.org/abs/1802.09477>.
- [34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. eprint: [arXiv:1606.01540](https://arxiv.org/abs/1606.01540).
- [35] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control”, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033. DOI: [10.1109/IR05.2012.6386109](https://doi.org/10.1109/IR05.2012.6386109).
- [36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: [1707.06347](https://arxiv.org/abs/1707.06347) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1707.06347>.
- [37] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: [1801.01290](https://arxiv.org/abs/1801.01290) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1801.01290>.

- [38] K. Zakka, Y. Tassa, and MuJoCo Menagerie Contributors, *MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo*, 2022. [Online]. Available: <http://github.com/google-deepmind/mujoco-menagerie>.
- [39] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, *et al.*, “Gymnasium: A standard interface for reinforcement learning environments”, *arXiv preprint arXiv:2407.17032*, 2024.
- [40] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry, *Gymnasium robotics*, version 1.3.1, 2024. [Online]. Available: <http://github.com/Farama-Foundation/Gymnasium-Robotics>.
- [41] L. Leottau, C. Celemin, and J. Ruiz-del-Solar, “Ball dribbling for humanoid biped robots: A reinforcement learning and fuzzy control approach”, in *RoboCup 2014: Robot World Cup XVIII*, R. A. C. Bianchi, H. L. Akin, S. Ramamoorthy, and K. Sugiura, Eds., Cham: Springer International Publishing, 2015, pp. 549–561, ISBN: 978-3-319-18615-3.
- [42] E. Elibol, J. Calderon, M. Llofriu, C. Quintero, W. Moreno, and A. Weitzenfeld, “Power usage reduction of humanoid standing process using q-learning”, in *RoboCup 2015: Robot World Cup XIX*, L. Almeida, J. Ji, G. Steinbauer, and S. Luke, Eds., Cham: Springer International Publishing, 2015, pp. 251–263, ISBN: 978-3-319-29339-4.
- [43] D. L. Leottau, J. Ruiz-del-Solar, P. MacAlpine, and P. Stone, “A study of layered learning strategies applied to individual behaviors in robot soccer”, in *RoboCup 2015: Robot World Cup XIX*, L. Almeida, J. Ji, G. Steinbauer, and S. Luke, Eds., Cham: Springer International Publishing, 2015, pp. 290–302, ISBN: 978-3-319-29339-4.
- [44] Y. Chebotar, Q. Vuong, A. Irpan, K. Hausman, F. Xia, Y. Lu, A. Kumar, T. Yu, A. Herzog, K. Pertsch, K. Gopalakrishnan, J. Ibarz, O. Nachum, S. Sontakke, G. Salazar, H. T. Tran, J. Peralta, C. Tan, D. Manjunath, J. Singht, B. Zitkovich, T. Jackson, K. Rao, C. Finn, and S. Levine, *Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions*, 2023. arXiv: 2309 . 10150 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2309.10150>.
- [45] H. Qi, B. Yi, S. Suresh, M. Lambeta, Y. Ma, R. Calandra, and J. Malik, *General in-hand object rotation with vision and touch*, 2023. arXiv: 2309 . 09979 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2309.09979>.
- [46] Y. Cho, J. Han, Y. Cho, and B. Kim, *Corn: Contact-based object representation for nonprehensile manipulation of general unseen objects*, 2024. arXiv: 2403 . 10760 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2403.10760>.
- [47] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, *Deep reinforcement learning for robotics: A survey of real-world successes*, 2024. arXiv: 2408 . 03539 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2408.03539>.
- [48] O. G. Younis, R. Perez-Vicente, J. U. Balis, W. Dudley, A. Davey, and J. K. Terry, *Minari*, version 0.5.0, Sep. 2024. DOI: 10 . 5281/zenodo .

13767625. [Online]. Available: <https://doi.org/10.5281/zenodo.13767625>.
- [49] B. Peng, T. Rashid, C. A. S. de Witt, P.-A. Kamienny, P. H. S. Torr, W. Böhmer, and S. Whiteson, *Facmac: Factored multi-agent centralised policy gradients*, 2021. arXiv: 2003.06709 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2003.06709>.
- [50] P. Wawrzyński, “A cat-like robot real-time learning to run”, in *Adaptive and Natural Computing Algorithms*, M. Kolehmainen, P. Toivanen, and B. Beliczynski, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 380–390, ISBN: 978-3-642-04921-7.
- [51] T. Erez, Y. Tassa, and E. Todorov, “Infinite-horizon model predictive control for periodic tasks with contacts”, in *Robotics: Science and Systems*, 2011. [Online]. Available: <https://api.semanticscholar.org/CorpusID:12424685>.
- [52] R. Coulom, “Reinforcement learning using neural networks, with applications to motor control”, 2002. [Online]. Available: <https://api.semanticscholar.org/CorpusID:263889103>.
- [53] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation*, 2018. arXiv: 1506.02438 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1506.02438>.
- [54] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization”, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4906–4913. DOI: 10.1109/IRoS.2012.6386025.
- [55] B. Ellenberger, *Pybullet gymperium*, <https://github.com/benelot/pybullet-gym>, 2018–2019.
- [56] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dornmann, “Stable-baselines3: Reliable reinforcement learning implementations”, *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>.
- [57] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, *Controlling overestimation bias with truncated mixture of continuous distributional quantile critics*, 2020. arXiv: 2005.04269 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2005.04269>.
- [58] M. Franceschetti, C. Lacoux, R. Ohouens, A. Raffin, and O. Sigaud, *Making reinforcement learning work on swimmer*, 2022. arXiv: 2208.07587 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2208.07587>.
- [59] F. Felten, L. N. Alegre, A. Nowé, A. L. C. Bazzan, E. G. Talbi, G. Danoy, and B. C. d. Silva, “A toolkit for reliable benchmarking and research in multi-objective reinforcement learning”, in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.
- [60] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, *D4rl: Datasets for deep data-driven reinforcement learning*, 2020. arXiv: 2004.07219 [cs.LG].

- [61] D. H. V. Dyke, "Chapter 4 : Applications of distributed artificial intelligence in industry", 1994. [Online]. Available: <https://api.semanticscholar.org/CorpusID:3894015>.
- [62] J. Ackermann, V. Gabler, T. Osa, and M. Sugiyama, *Reducing overestimation bias in multi-agent domains using double centralized critics*, 2019. arXiv: [1910.01465](https://arxiv.org/abs/1910.01465) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1910.01465>.
- [63] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, *Counterfactual multi-agent policy gradients*, 2017. arXiv: [1705.08926](https://arxiv.org/abs/1705.08926) [cs.AI]. [Online]. Available: <https://arxiv.org/abs/1705.08926>.
- [64] X. Lyu, Y. Xiao, B. Daley, and C. Amato, *Contrasting centralized and decentralized critics in multi-agent reinforcement learning*, 2021. arXiv: [2102.04402](https://arxiv.org/abs/2102.04402) [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2102.04402>.
- [65] J. N. Foerster, Y. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning", *ArXiv*, vol. [abs/1605.06676](https://api.semanticscholar.org/CorpusID:53391180), 2016. [Online]. Available: <https://api.semanticscholar.org/CorpusID:53391180>.
- [66] A. Raffin, *Rl baselines3 zoo*, <https://github.com/DLR-RM/rl-baselines3-zoo>, 2020.