UNIVERZITA
KARLOVA

École Nationale Supérieure d'Ingénieurs
de Caen

Internship report

Deep Reinforcement Learning for Humanoid Robot Programming

Intern : Oussama AQEBLI
Internship supervisor : Stefan EDELKAMP
Academic tutor : Karim-eric ZIAD-FOREST
Academic year : 2024–2025

August 19, 2025

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Mr.**Karim-eric Ziad-forest**, my academic supervisor at **ENSICAEN**, for his guidance, constructive feedback, and availability throughout the course of this internship. His support has been invaluable in shaping both the experience and this report.

I also would like to warmly thank Mr.**Stefan Edelkamp**, my internship supervisor at **Univerzita Karlova**, for welcoming me to the university and ensuring the smooth progress of my mission with professionalism and dedication.

A special thank you also goes to Mr.**Xuzhe Dang**, my day-to-day mentor, for his continuous support, patience, and the many skills he helped me develop over the course of the internship.

I also want to express my appreciation to Mr. **David Obdržálek** and Ms. **Jarmila Vágnerová** for their help to get to the university and showing me around the Robotics Lab and the *Impakt Building*.

I am so grateful to the entire **Reinforcement Learning and Robotics Team** for their warm welcome, collaborative spirit, and insightful discussions we shared.

Finally, I would like to thank all those, both at the school and at the university, who contributed directly or indirectly to the success of this internship.

Contents

Chapter 1

Introduction

Recently, we are getting more and more exposure to AI-based products in our daily life. Especially now, we are watching boxing robots, transformers in real life, robot dogs with upgraded and more refined precision, mini-robots like *NAO6* that are being used for educational goals towards autistic kids with behavioral issues.

However, this exposure is still too restricted because the world of humanoid robots is always undergoing huge changes and faces a lot of challenges, from the unavailability of the prototypes in question to the size of the workforce researching the field of **Reinforcement Learning (RL)**.

Thus, I enthusiastically chose to do this internship in one of the largest laboratories in the Czech Republic, to discover the bread and butter of **RL** and how it is used to train models from scratch, starting by learning the very basic concepts of RL, then discovering the numerous training algorithms, then jumping into simulation in **IsaacSim**, and eventually having fun with the newest versions of humanoid robots.

Figure 1.1: Social event in Prague with Unitree robots

Chapter 2

Presentation of Charles University

Founded in 1348, *Charles University (Univerzita Karlova)* is one of the oldest and most prestigious universities in Europe. Within its Faculty of Mathematics and Physics (MFF UK), the university hosts advanced research in artificial intelligence, machine learning, and robotics. Although hands-on robotics research is more prominent at the *Czech Technical University (ČVUT)*, Charles University plays a critical role in collaborative projects, such as the MAPLE project, which focuses on multi-agent path planning and execution for mobile robots in dynamic environments.

Reinforcement learning is an active area of both teaching and research at the university. The course *NPFL139 – Deep Reinforcement Learning* offers students practical and theoretical exposure to modern RL algorithms, including policy gradient methods, actor-critic architectures, and deep Q-networks. Recent student theses and research explore applications of RL to algorithmic trading, symbolic control, and navigation tasks, often in collaboration with industrial or academic partners.

Figure 2.1: Charles University

Charles University also contributes to robotics via interdisciplinary efforts like *Charles Automata*, a spinoff developing AI-powered laboratory automation systems. These initiatives highlight the university's strong theoretical foundations and its growing engagement in applied robotics and intelligent systems research.

Chapter 3

Context & Objectives of the Internship

The main purpose of my internship was to grasp all the basics of Reinforcement Learning, and apply it to develop - or mostly reproduce - the policy optimization algorithms that are used to dictate the model's behavior, then hopping into more complex algorithms for more stable training and more desirable outcome. And then finally, use the acquired skills to work on actual real-life models such as the *Unitree humanoid robot G1* or the *Unitree robot dog Go2*.

Starting from the foundation, my fellow tutor Mr. **Xuzhe Dang** urged me to study the key concepts of Reinforcement Learning by consulting the OpenAI documentation *Spinning Up* which contains all the tools necessary for a beginner in the field of RL. The main notions I had to remember and absorb are:

- **State :** a complete description of the state of the world where our model is spawned.
- **Observation :** a partial description of a state, in other words, what the agent *observes*.
- **Action :** the behavioral response to the environment, this action is generated by a policy.
- **Policy :** a rule used by the agent to decide what actions to take.
- **Reward :** a positive/negative value given to reward/penalize the agent based on how desirable or unwanted its action was in the context.

With these critical keywords, we can define a loop in which the agent observes, acts, evaluates, and ultimately **learns** from the previous episodes what to do in the next state, and it could be represented as such:

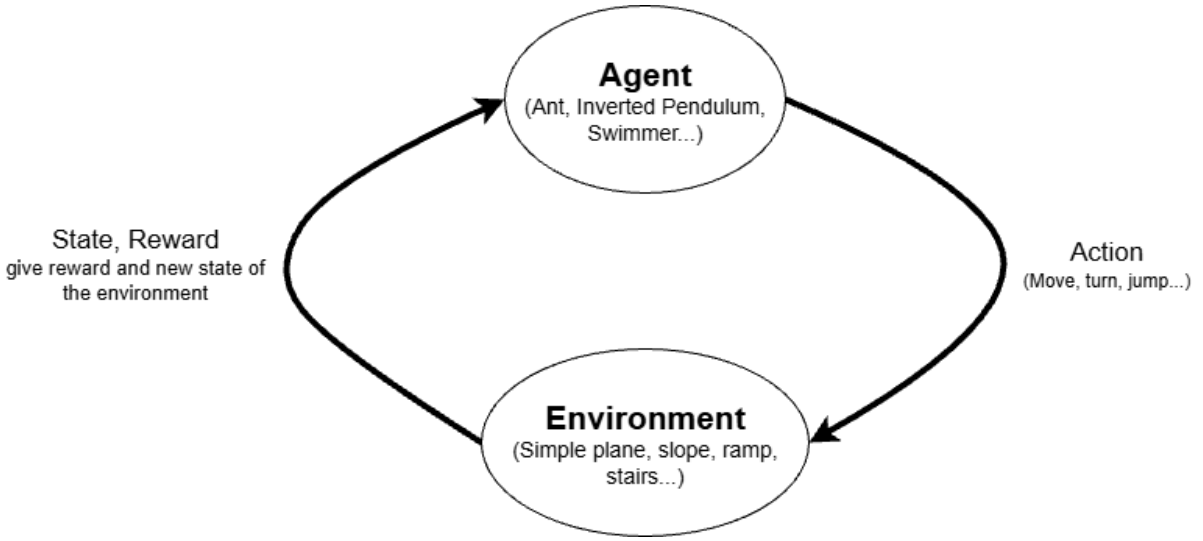

Figure 3.1: Training loop of an agent

The next step was to study the basic RL algorithms, one of the most important and central parts of Reinforcement Learning. Some algorithms served as stepping stones for further studies and more optimized and fruitful algorithms. In the course of my studies, I focused especially on the policy optimization side, especially the parts marked in the following figure, but I still got a glimpse of the other RL algorithms like **AlphaZero**, **DQN** and **TRPO**.

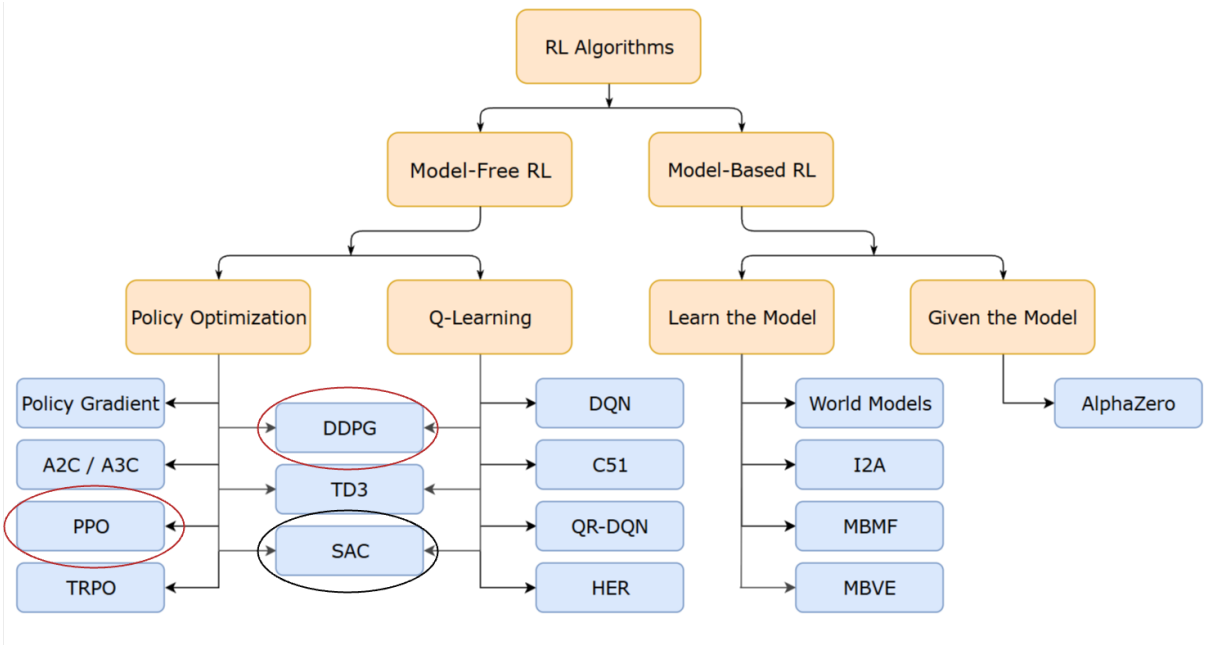

Figure 3.2: Reinforcement Learning Algortihms Tree

After a few weeks of trial and error trying to optimize the training process, the new objective was to **design the reward function** for certain tasks. This was by far the hardest thing I had to do; not only does the task take a lot of architecting and brain-

storming, it also requires a lot of testing which is very time consuming. The main goals of this heavy task are to define the rewards and penalties to be applied in order to complete the mission at hand, and, most importantly, to accord the appropriate weights to each component of the reward function; this way we prevent the agent from ***cheating*** the desired task. "**Cheating**" in this context means getting very high rewards but still not accomplishing the desired results, and this occurred very often when a certain component is given too little or too much weight. Balance is everything! This is an explanatory example of a reward function:

Hammer Task Reward Function

The task at hand is for the robotic arm to use its pincer-shaped end-effector to grab the hammer and hit the nail in the wooden block as shown in the following figure:

Figure 3.3: The Hammer task visualization

The total reward is restructured as a weighted sum of interpretable sub-rewards:

$$R = \sum_{i=1}^{7} \gamma_i \cdot r_i \tag{3.1}$$

Where:

- γ_i are task-specific weights,
- r_i are individual reward (or penalty) components.

ID	Name	Reward Name	Description / Formula Idea
1	Align with hammer	r_{align}	Lower the end-effector to the hammer's height, e.g., $\mathcal{L}_1(z_{\text{ee}} - z_{\text{hammer}})$
2	Grasp hammer	r_{grasp}	Successful attachment of hammer to gripper, e.g., $\mathbb{I}_{\text{hammer attached}}$
3	Lift hammer to nail	r_{lift}	Align vertical position with nail, e.g., $\mathcal{L}_1(z_{\text{hammer}} - z_{\text{nail}})$
4	Horizontal movement	r_{move}	Move end-effector towards nail position, e.g., $\mathcal{L}_1(x_{\text{ee}} - x_{\text{nail}})$
5	Overshoot penalty	$r_{\text{overshoot}}$	Penalize going past the nail, e.g., $\mathbb{I}_{x_{\text{ee}} > x_{\text{nail}} + \epsilon}$
6	Insufficient speed	r_{slow}	Penalize too-slow motion before impact, e.g., $\mathbb{I}_{\|\dot{x}_{\text{ee}}\| < v_{\min}}$
7	Redundant hits	$r_{\text{redundant}}$	Penalize hitting already-inserted nail, e.g., $\mathbb{I}_{\text{nail depth} > d_{\max}} \cdot \mathbb{I}_{\text{impact}}$

Table 3.1: Reward components for the hammer environment

The full reward can be written as:

$$\begin{aligned} R = {} & \gamma_{\text{align}} r_{\text{align}} + \gamma_{\text{grasp}} r_{\text{grasp}} + \gamma_{\text{lift}} r_{\text{lift}} + \gamma_{\text{move}} r_{\text{move}} \\ & - \gamma_{\text{overshoot}} r_{\text{overshoot}} - \gamma_{\text{slow}} r_{\text{slow}} - \gamma_{\text{redundant}} r_{\text{redundant}} \end{aligned}$$

Example of Good Weights (Effective Task Completion)

$$\begin{aligned} \gamma_{\text{align}} &= 1.0 \\ \gamma_{\text{grasp}} &= 2.0 \\ \gamma_{\text{lift}} &= 1.5 \\ \gamma_{\text{move}} &= 2.5 \\ \gamma_{\text{overshoot}} &= 3.0 \\ \gamma_{\text{slow}} &= 2.0 \\ \gamma_{\text{redundant}} &= 5.0 \end{aligned}$$

These weights strongly reward meaningful progress while preventing inefficient or incorrect actions.

Example of Bad Weights (Cheating Behavior)

$$\begin{aligned}
\gamma_{\text{align}} &= 0.1 \\
\gamma_{\text{grasp}} &= 0.1 \\
\gamma_{\text{lift}} &= 0.1 \\
\gamma_{\text{move}} &= 10.0 \\
\gamma_{\text{overshoot}} &= 0.0 \\
\gamma_{\text{slow}} &= 0.0 \\
\gamma_{\text{redundant}} &= 0.0
\end{aligned}$$

For example, this configuration allows the agent to maximize the reward by quickly moving horizontally without ever interacting properly with the hammer or nail.

With all these tools in the bag, it was time for some real-life application, I was lucky to be able to maneuver and steer the trending Unitree robots **G1** and **Go2**. I got the opportunity to discover with my own eyes the actual progress in the **Algorithmic Intelligence** field, and how much effort is put into making such high-performance models. Soon after attending some events, I started studying more complicated algorithms that sum up most of the previous work I had done, algorithms such **GAIL** that uses a neural network called "Discriminator" dedicated to test the robustness of the agent by filtering out the data from a large pre-made dataset and newly generated data provided by a Policy Optimization algorithm such as PPO. Later, I used this algorithm to study an even more complicated algorithm that targets the natural aspect of training: the **AMP**.

Ultimately, the learning part ended here and it was time to see the fruits of my work. I started working in **IsaacSim** to simulate the walking gait of the robot dog **Go2** using instructions from my tutor; I had to design the observation and reward getters. Basically, I gather the bodily information of the virtual model and then filter them based on relevance to the reward function architecture; this information contains, for example, the positions of the torques and joints, the linear and angular velocities, the height and position of the root of the model. After that, I had to check Mr.**Federico Sarrocco**'s blog as a reference for the reward function, and then implemented the trainer and evaluation codes using PPO to train and visualize the trained policy. All of the work was performed on the lab computer due to the GPU and CPU requirements since I needed to work on a large number of environments for faster and more accurate behavior: 4096 for the training and 12 for the evaluation.

Figure 3.4: Unitree Go2 Walking gait visualization

The rest of my internship was dedicated to tuning the trainer hyperparameters, calibrating the weights in the reward function, even possibly adding other rewards for improved results, and a lot of testing and readjusting. The architecture of the work can be described as the following:

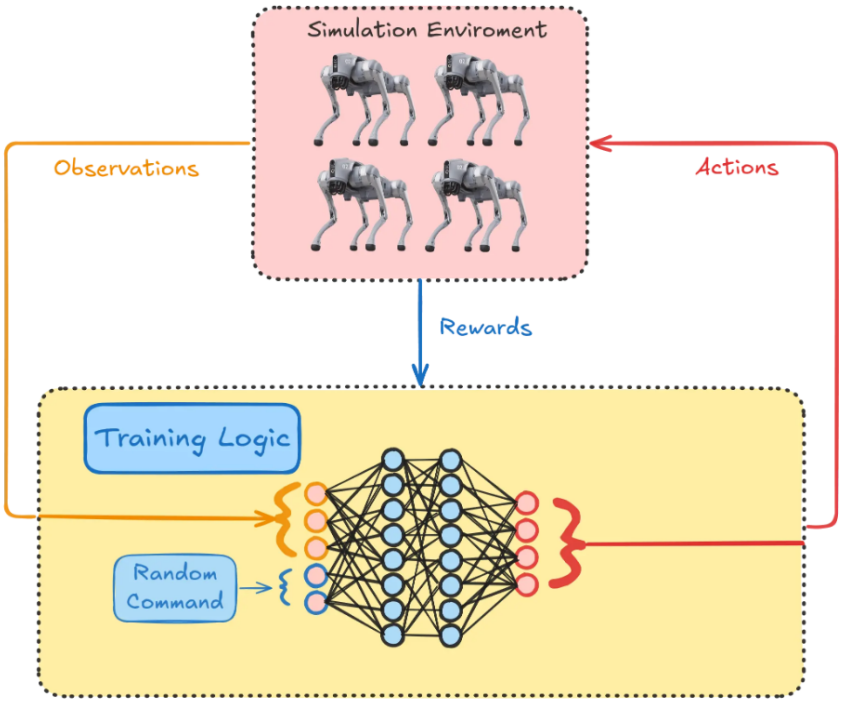

Figure 3.5: Reinforcement Learning Framework for Training Quadruped Robots - Taken from Federico Sarrocco's blog

In the simulation, the model seemingly follows a certain pattern, but it was still ambiguous since the user commands are randomized uniformly between -1.0 and 1.0 for the x and y axis and the yaw rotation around the z-axis. The issue was that it was impossible to add extra code to visually simulate an directional arrow with the time I had left. Instead, I tried to tweak the command range, for example, disabling the y-axis translation and the yaw rotation to see clearly if the model can walk properly on the x-axis, and the same for the other user commands, and it actually gave very good results with little to no early termination of the episode.

Chapter 4

Tasks accomplished

The way the internship has taken place was through weekly tasks and a meeting on every Friday. The large tasks were usually separated into two weeks; the first week for learning about the tools I was supposed to use or reading a documentation about the algorithm I would work on, and the second week for implementing the algorithm and testing on a few RL models.

The most useful tool I used to test the performance of my code was **Mujoco**, this library offered many realistic models to train with a variety of difficulty based on the complexity of the model; the action and observation spaces of the model, the reward function architecture, the early termination conditions, etc. I mostly worked on "HalfCheetah-v5" which is the simplest Mujoco model simulating bipedal animal-like behavior.

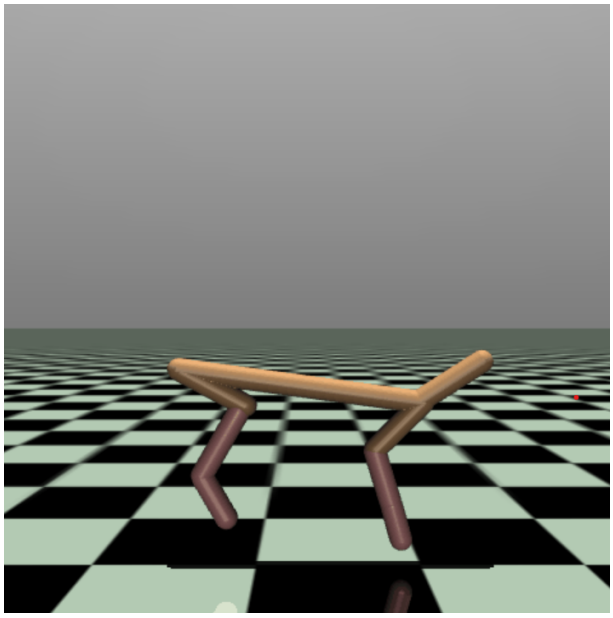

Figure 4.1: Mujoco HalfCheetah-v5 model

Using the **Farama gymnasium documentation**, it was easier to expect the output when visualizing the reward of each episode.

Algorithm	Episodes to Good Results	Time (Hours)	Reward Range
DDPG (HalfCheetah-v5)	800–1500	3–5	4000–6000
PPO (HalfCheetah-v5)	600–1200	2–4	6000–8500

Table 4.1: Training performance comparison on HalfCheetah-v5

Using for example HalfCheetah-v5 as a reference, it allowed me to compare the performance of each RL algorithm and build on it for future tasks.

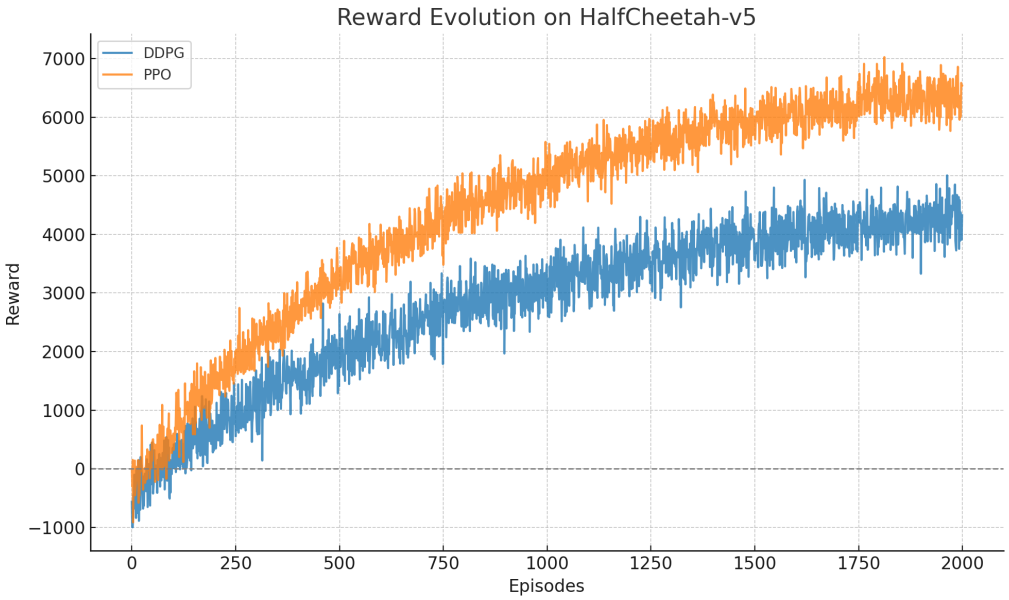

Figure 4.2: Reward evolution over episodes for DDPG and PPO on HalfCheetah-v5

I was able to get a well-trained agent capable of generating the correct actions to get the model to go forward, in a straight line and without falling over and causing the early termination of the episode with all agents. As for more complex models, such as "*Ant-v5*", "*Walker2D-V5*", "*HumanoidStandup-v5*", it did not go as smoothly either because of the incompatibility of the algorithm or the immense CPU power needed for the computation.

I was also able to implement the **GAIL** algorithm which was an interesting way to reintroduce PPO and experiment with different **Farama Minari** datasets. The main difficulty of this task specifically was the training instability and the wide range of different behaviors that I had as a result. Thus, I had to search for new ways to stabilize it by looking up the original papers of GAIL researchers, and consequently decided to use the LSGAN method, which reshapes the reward function of the discriminator and clips the reward to a more reasonable scale. This allowed the results to fall within a smaller scope, which meant therefore that it was only a matter of fine-tuning the hyperparameters.

And finally, with the acquired knowledge, I was able to adapt to a completely different

environment. I had to use the isaaclab environment in order to train the Unitree Go2 model, and I needed to implement two of the most important parts of any training; the observation function and the reward function. I started from scratch, studied the documentation of the model's articulation, filtered out the unnecessary details and extra data in the observation space, implemented the sub-reward functions which will encourage the robot to walk in a randomized direction and prevent it from doing undesirable actions, and finally started testing with different weights for each sub-reward, which was very time-consuming. The task took me nearly 4 weeks of trial and error; one week to implement the functions, one week to implement an adapted and effective trainer and evaluator, and two weeks of trying different setups.

Chapter 5

Critical analysis & Perspectives

Working on this internship was a cognitive trip accompanied by occasional setbacks, it was no easy task to work in an entirely different field with new tools and a lot of self-dependence. From time to time, I would encounter new puzzles and have to work my brain gears to find a solution.

The first difficulty was to set my computer up to be compatible with all the new versions of the required libraries and the eventual updates. The first thing I learned is that Windows is never a good idea for coding; the amount of deprecated versions I stumbled upon was overwhelming, so rather than working on the tasks that matter, I spent an excessive amount of time just trying to meet the standards of every version; some versions were not even available for Windows OS.

Also, the documentation was not easy to read and I faced this separation between the algorithm and the implementation over and over again. Most of the time, especially in the beginning, I would forget what some elements of the code stand for, and I had to base my code off a basic model on the Internet for a kickstart, that way I can go over every single line and decipher the meaning of every single part in the code. It was a slow start and a harsh blow to my coding ego, but it was also humbling and an opportunity to start over and polish my skills. Eventually, I was very comfortable with the code; I could freely manipulate it, troubleshoot it, detect its flaws, and find appropriate solutions to patch it.

The algorithms and new concepts were not difficult to absorb and the coding part was not problematic. The main hindrance was the training time; Not only does training take a lot of time, but it does not give clear signs if the code is working properly. A typical training takes 20 to 30 minutes to be eligible for judgement, which means 20-30 minutes to be faced with disappointment. It really tested my patience, my level of understanding, and especially my ability to generate solutions. I decided to set checkpoints where I would check the behavior of the model I am working on and print the reward, the entropy, and the loss at the end of each episode. In this way, I could analyze the logs and predict the

outcome of the training without waiting until the end. Thus, the waiting time went from 20 minutes or more down to 7-10 minutes, allowing me to focus more on optimizing the agent and wait less for the results.

And due to the lack of computers in the lab at the time, we were forced to use our own machines. But at a certain point, it was no longer possible to do so, since the tasks got increasingly demanding, which meant that we had to take turns to use the lab computer. It was not optimal, especially for me, because my last task was impossible to accomplish without the NVIDIA card. Eventually, Mr. **Edelkamp** took matters into his own hands and provided a new computer which was a great relief.

And finally, right before my last task, I was faced with a difficult choice. Since I started on Unitree models only a month before the end of my internship, it was clear that I was not going to be able to deploy any of my work on the humanoid robot. So my tutor gave me new options: to continue to work on the humanoid model knowing that I might not be able to have a working agent, to work on the robot arm, or to work on the Unitree Go2 only in simulation. After analyzing the options, the safest choice was the latter, as it is feasible within the time limit, and I will have a colleague to discuss and exchange the results with.

Chapter 6

Conclusion

All in all, my internship was a very constructive experience full of happy little surprises and a lot of self-reflection. The time I spent in the lab was priceless, as I was able to physically be present to witness the greatness of the Reinforcement Learning field.

Going from absolutely zilch to knowing as much as I did in the end was a clear sign of self-growth and determination, and it would not have been possible without the kindness of the people I met in the internship, my colleagues, and my go-to tutors when I struggled.

I am so grateful for the precious time spent in Prague in the Troja lab, which would not have been possible without the help of my supervisor Mr. **Stefan Edelkamp**.

Beyond the technical skills acquired, this experience fostered problem-solving autonomy, resilience in the face of long and uncertain training cycles, and adaptability to computational constraints. The collaborative environment of the Troja lab, along with the guidance of my supervisors and colleagues, played a decisive role in the successful completion of the internship's objectives. Ultimately, this project has not only deepened my expertise in reinforcement learning for robotics but also strengthened my readiness to tackle research and engineering challenges in artificial intelligence and autonomous systems.

Bibliography

Federico Sarrocco Leonardo Bertelli, *Making quadrupeds Learning to walk: Step-by-Step Guide*, MIT Press, 2018.

"Reinforcement learning," *Wikipedia*, retrieved 2025, defining RL and exploration–exploitation trade-off :contentReference[oaicite:0]index=0.

"Machine Learning Glossary: Reinforcement Learning," Google Developers, April 2025 :contentReference[oaicite:1]index=1.

"Glossary of Terminology in Reinforcement Learning," UMass Amherst :contentReference[oaicite:2]index=2.

"Reinforcement Learning Definitions," Pathmind RL glossary :contentReference[oaicite:3]index=3.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, "Proximal Policy Optimization Algorithms," arXiv 2017 :contentReference[oaicite:4]index=4.

"Deep Deterministic Policy Gradient (DDPG)," Spinning Up in Deep RL (OpenAI) :contentReference[oaicite:5]index=5.

Gabriel Barth-Maron et al., "Distributed Distributional Deterministic Policy Gradients (D4PG)," arXiv 2018 :contentReference[oaicite:6]index=6.

(Placeholder: Add references to GAIL; typically—Jonathan Ho Stefano Ermon, "Generative Adversarial Imitation Learning," NIPS 2016.)

Chapter 7

Annex

7.1 Appendix A. Glossary of Key Terms

Agent The decision-making entity interacting with the environment :contentReference[oaicite:7]index=

Environment The external system in which the agent operates and from which it receives states and rewards :contentReference[oaicite:8]index=8.

State / Observation A representation of the environment's current situation; "state" is complete, while "observation" may be partial :contentReference[oaicite:9]index=9.

Action A move chosen by the agent that leads to a state transition :contentReference[oaicite:10]index=10.

Reward A scalar feedback signal indicating the quality of an action in a given state :contentReference[oaicite:11]index=11.

Policy A (possibly stochastic) mapping from states to actions :contentReference[oaicite:12]index=12.

Q-Function The expected return when taking an action in a state and then following a policy :contentReference[oaicite:13]index=13.

Return Sum of discounted future rewards: $R = r_0 + \gamma r_1 + \ldots$:contentReference[oaicite:14]index=14.

Discount Factor (γ) Weighting of future rewards between 0 and 1 :contentReference[oaicite:15]index=

Exploration–Exploitation Dilemma Trade-off between exploring new actions and exploiting known good ones :contentReference[oaicite:16]index=16.

Markov Decision Process (MDP) Framework describing RL as state transitions with the Markov property :contentReference[oaicite:17]index=17.

Actor-Critic Architecture where the "actor" proposes actions and the "critic" evaluates them :contentReference[oaicite:18]index=18.

Experience Replay Buffer of past transitions used for off-policy learning (e.g., in DQN) :contentReference[oaicite:19]index=19.

Trajectory Sequence of (state, action, reward) tuples in an episode :contentReference[oaicite:20]index=

7.2 Appendix B. Summary of Selected Algorithms

7.2.1 Proximal Policy Optimization (PPO)

PPO is a model-free, on-policy actor-critic method that improves stability by clipping policy updates—a simpler alternative to Trust Region Policy Optimization (TRPO). Since its introduction in 2017, it has become the default RL algorithm at OpenAI due to its balance of performance, sample efficiency, and ease of implementation :contentReference[oaicite:21]index=21.

7.2.2 Deep Deterministic Policy Gradient (DDPG)

DDPG is an off-policy, actor-critic algorithm designed for continuous action spaces. It concurrently learns a Q-function and a deterministic policy, leveraging the Bellman equation and experience replay for stable learning :contentReference[oaicite:22]index=22.

7.2.3 Distributed Distributional DDPG (D4PG)

D4PG combines distributional RL with DDPG in a distributed framework, incorporating techniques like n-step returns and prioritized experience replay. It achieves state-of-the-art performance on challenging continuous control tasks :contentReference[oaicite:23]index=23.

7.2.4 Generative Adversarial Imitation Learning (GAIL)

GAIL is an imitation learning method that trains a policy to mimic expert behavior through an adversarial framework, akin to GANs. A discriminator distinguishes expert from learner behavior while the generator (policy) learns to produce indistinguishable trajectories. (Reference: Ho Ermon, NIPS 2016.)