A MASTER THESIS ON

Variable Stiffness for Robust Locomotion Using Reinforcement Learning

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Diplom-Ingenieur
(Equivalent to Master of Science)

in

Automation and Robotic Systems UE 066 515

by

Dario Spoljaric BSc.

Student ID: 11806417

Supervisor(s):
Univ.Ass. Yashuai Yan M.Sc.
Univ.Prof. Dongheui Lee PhD

Vienna, Austria

October 2024

Abstract

Reinforcement-learned locomotion offers a promising approach for legged robots as it generalises well and enables traversing in challenging terrains. However, current methods often rely on a position-based control paradigm, requiring additional joint stiffness tuning, which requires joint stiffness tuning—a time-consuming process that may need retuning for different tasks. This thesis addresses this limitation by exploring the benefits of learning variable stiffness control for quadruped robots. Specifically, it incorporates joint stiffnesses into the action space alongside joint positions. To simplify training and reduce the dimensionality of the action space, we introduce stiffness grouping strategies, including per-joint stiffness (PJS), per-leg stiffness (PLS), and hybrid joint-leg stiffness (HJLS). We demonstrate the efficiency of our approach by comparing it to position-based control across the metrics of velocity tracking, push recovery and energy efficiency. Our results show that variable stiffness policies using per-leg stiffness (Per Leg Stiffness (PLS)) outperform position-based control in velocity tracking and push recovery. Moreover, when stiffnesses are predicted for individual joint and leg groups (Hybrid Joint-Leg Stiffness (HJLS)), our approach achieves superior energy efficiency compared to high-stiffness position-based policies. Sim-to-real transfer experiments validate the feasibility of our method, revealing minimal differences in performance during general walking tasks. Overall, this thesis simplifies the design process by eliminating the need for manual stiffness tuning while enhancing or maintaining performance in velocity tracking, push recovery and energy efficiency.

Kurzfassung

Die mit Reinforcement Learning erlernte Lokomotion bietet einen vielversprechenden Ansatz für beinige Roboter, da sie eine gute Generalisierung ermöglicht und das Überwinden von schwierigem Terrain erleichtert. Aktuelle Methoden basieren jedoch oft auf einem positionsbasierten Steuerungsparadigma, das eine zusätzliche Abstimmung der Gelenksteifigkeit erfordert – ein zeitaufwändiger Prozess, der für verschiedene Aufgaben möglicherweise erneut durchgeführt werden muss. Diese Arbeit untersucht, wie das Lernen variabler Steifigkeitssteuerungen für vierbeinige Roboter diese Einschränkungen überwinden kann. Konkret werden Gelenksteifigkeiten zusammen mit Gelenkpositionen in den Aktionsraum integriert. Um das Training zu vereinfachen und die Dimensionalität des Aktionsraums zu reduzieren, führen wir Strategien zur Gruppierung der Steifigkeiten ein, darunter steifigkeitsbasierte Steuerungen pro Gelenk (PJS), pro Bein (PLS) und eine hybride Steuerung für Gelenk- und Beingruppen (HJLS). Wir demonstrieren die Effizienz unseres Ansatzes, indem wir ihn anhand der Metriken Geschwindigkeitsverfolgung, Störkraftkompensation und Energieeffizienz mit positionsbasierter Steuerung vergleichen. Wenn die Steifigkeiten nach Beinen gruppiert werden, übertrifft unser Controller (PLS) die positionsbasierte Steuerung sowohl bei der Geschwindigkeitsverfolgung als auch bei der Störkraftkompensation. Werden die Steifigkeiten hingegen für einzelne Bein- und Gelenkgruppen (HJLS) vorhergesagt, übertrifft unser Ansatz positionsbasierte Steuerungen mit hoher Steifigkeit in Bezug auf die Energieeffizienz. Experimente zum Sim-to-Real-Transfer bestätigen die Umsetzbarkeit unserer Methode und zeigen nur geringe Unterschiede bei allgemeinen Gehaufgaben. Insgesamt vereinfacht unser Ansatz das Design, da keine Abstimmung der Steifigkeiten für jedes Gelenk erforderlich ist, während gleichzeitig die Leistung bei Geschwindigkeitsverfolgung, Störkraftkompensation und Energieeffizienz verbessert wird oder gleich bleibt.

Preface

Parts of this thesis were submitted to the 14th IFAC Joint Symposia on Mechatronics and Robotics as D. Spoljaric, Y. Yan, D. Lee, *"Variable Stiffness for Robust Locomotion through Reinforcement Learning".*

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Copyright Statement

I, Dario Spoljaric BSc., hereby declare that this thesis is my own original work and, to the best of my knowledge and belief, it does not:

- Breach copyright or other intellectual property rights of a third party.
- Contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
- Contain material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.
- Contain substantial portions of third party copyright material, including but not limited to charts, diagrams, graphs, photographs or maps, or in instances where it does, I have obtained permission to use such material and allow it to be made accessible worldwide via the Internet.

Signature: ____________________________

Vienna, Austria, October 2024 Dario Spoljaric BSc.

Acknowledgment

This thesis would not have been possible without the help of some wonderful and intelligent people. First, I would like to thank my supervisor, Yashuai Yan, who introduced me to the world of reinforcement learned locomotion. He continuously supported me throughout the whole thesis and pinpointed me into the right directions. I still remember the excitement when my first AI controller showed successful sim-to-real transfer. Next, I would like to thank my colleagues like Stefan Lechner, who helped me figure out the right evaluation and Benedikt Frey as well as Zoltan Varga, who helped me record during the experiments.

Dedication

I dedicate this thesis to my beloved parents, Helga Haller and Davor Spoljaric, and my girlfriend Marlene Platt, whose endless support and encouragement have been my greatest source of strength throughout my academic journey. Their belief in me, their patience, and their endless love have made this accomplishment possible. I am forever grateful for their presence in my life.

Contents

Abstract iii

Kurzfassung iv

Preface v

1 Introduction 1

1.1 Contribution of this thesis 2

1.2 Thesis overview 3

2 Background 5

2.1 Reinforcement learning 5

2.2 Reinforcement learning in locomotion 14

3 Related works 15

3.1 Legged locomotion in robotics 15

3.2 Model-based control methods 15

3.3 Reinforcement learning in locomotion 15

3.4 Variable Stiffness 19

4 Methods and Design 21

4.1 Proposed Methodology 21

4.2 Design choices 23

4.3 Hardware implementation 29

5 Analysis and Results 33

5.1 Baseline 33

5.2 Performance on walking and running 33

5.3 Push recovery 36

5.4 Power efficiency 41
5.5 Sim-to-real transfer 43
5.6 Discussion 45
5.7 Future work 47
5.8 Conclusion 47

Bibliography **49**

A Appendix **55**
A.1 Notation 55

List of Tables

4.1 Hyperparameters for the proposed approach 23

4.2 Observation (actor), Privileged observations (critic) and Normalisation 26

4.3 Domain randomisation in simulation 27

4.4 Reward Function Components, exp(·) and var(·) are exponential and variance operators, respectively. $()^{cmd}$ indicates the commanded values, respectively, whereas $()^{des}$, $()^{target}$ and $()^{default}$ indicate the desired, target and default values. x, y and z are defined on the robot's body frame, with x and z pointing forward and upward, respectively. $v_{xy}, v_{f,xy}, \omega_{yaw}, \mathbf{g}_{x,y}, \boldsymbol{\tau}, \boldsymbol{q}, \mathbf{a}, p_{f,xy/z,k}, p_{com}$ represent the body-, foot velocity in the xy plane, yaw rate, projected gravity component in the xy plane, joint torque, joint angles, the position of the kth foot and the centre of mass. 28

5.1 Mean absolute tracking error over different commanded walking speeds in eight directions. Task speeds are 0.5 and 1.0 m/s. The comparison shows that the tracking error is lowest for method PLS in all cases. 35

5.2 Measured force push success rates under domain randomisation. Comparison of the variable stiffness policies PJS, PLS, IJS and HJLS. 38

5.3 Push recovery Success Rates under Domain Randomisation. Comparison of the position-controlled policy baselines P20 and P50 and our approach PLS, for force magnitudes within thresholds. 39

5.4 Cost of Transport at Different Walking Speeds 42

List of Figures

1.1 Examples for quadruped locomotion (a.) [1] and biped locomotion (b.) [2] 1

2.1 Reinforcement learning concept [3, p. 48] 6

2.2 Actor critic concept 12

2.3 Actor-critic paradigm in the context of quadruped locomotion 14

3.1 Model predictive control on a quadruped [4] 16

3.2 Timeline of RL based locomotion in recent years: (A)Learning agile and dynamic motor skills for legged robots [5], (B) Learning quadrupedal locomotion over challenging terrain [6], (C) Learning Agile Robotic Locomotion Skills by Imitating Animals [7], (D) Multi-expert learning of adaptive legged locomotion [8], (E) Vision and Proprioception for Navigation of Legged Robots [9], (F) Learning robust perceptive locomotion for quadrupedal robots in the wild [10],(G)Imitate and Repurpose: Learning Reusable Robot Movement Skills From Human and Animal Behaviours [11], (H) Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers [12]; figure from: [13] 17

3.3 **Common control concepts:** Position-based control (left) and torque-based control (right) 18

4.1 **Control overview for proposed methodology:** Different stiffness control approaches, PJS: "Per joint stiffness", PLS: "Per leg stiffness", IJS: "Individual Joint Stiffness", HJLS: "Hybrid Joint Leg Stiffness" 22

4.2 Network architecture of actor and critic. For every step in the simulation, the actor receives its observation vector o_{t_k} and the critic receives its privileged observation s_{t_k}. Therefore, the actor predicts an action and the critic a value for the given state. 24

4.3 Go2 MJX implementation from [14] 25

4.4 (a) Unitree Go2 robot: AIR, PRO and EDU. [15] (b) Unitree Go2-W [1] 29

4.5 ROS architecture includes different nodes for controller, hardware and simulation. . . 30

4.6 Deployment pipeline: 1. Training in Mujoco-MJX, 2.With the ROS nodes either in simulation or hardware. 31

5.1 Speed tracking comparison across target directions for PJS, PLS, IJS and HJLS. Velocities are averaged over 1000 environments, each lasting 8 seconds. The different coloured bars show the tracking error for the different walking speeds. 34

5.2 Speed tracking comparison across target directions for position control baselines with low (P20) and high stiffness (P50) and best performant variable stiffness approach PLS. Velocities are averaged over 1000 environments, each direction lasting 8 seconds. The different coloured bars show the tracking error for the different models. 35

5.3 (a) Force kick visualisation for the policy under evaluation. The visualisation illustrates the push directions applied to the robot. (b) Evaluation of push recovery robustness for the position-based policy with low stiffness (P20). Successful recoveries are depicted as green dots, while failures are shown as red dots. The blue line represents the success boundary, determined using an SVM with an RBF kernel, indicating the maximum force magnitude from which the policy can recover. The directions are defined as follows: 0° corresponds to a push from the back of the robot onto the trunk, 180° from the front, 90° from the right, and 270° from the left. 36

5.4 **Comparison of maximal recoverable force push.** The robot is pushed on the trunk, visualised in (a.). The polar scatter plot (b.) shows the experiment's outcome for specific forces (red=failure, green=success). An SVM with RBF kernel (confidence 90%) shows the maximum force magnitude from which the policy can recover. SVM parameters are C=0.6, γ=0.2. 37

5.5 Comparison of success boundary(confidence 90%), determined using an SVM with an RBF kernel, indicating the maximum force magnitude from which the policy can recover. Compared policies are the baselines P20 and P50 and our method PLS. SVM parameters: C=0.9, γ=0.2 39

5.6 Policies exposed to a left push in simulation and P-gains over time. Push conditions: $t_{push} = 3s$, $t_{duration} = 0.1s$, $F = 210N$, $\theta = 225$ accounting for the point in time, duration, magnitude and direction at which the push is applied. The force is applied from the frontal left direction. The demonstrated policies are the baselines P20, P50 and our approach PLS 40

5.7 Bar chart for Cost of Transport (CoT) across different walking speeds for six models. . 42

5.8 Despite being trained solely on a flat floor in simulation, we showcase the robot's ability with our variable-stiffness RL policy to walk robustly on diverse outdoor terrains, such as grass, stones, and sand. 43
5.9 **Payload experiment:** Adding a 5kg mass during walking to the PLS policy. 44
5.10 **Unexpected inclination:** The policy is exposed to never-before-seen inclination in the form of a ramp. 44

Acronyms

CoT Cost of Transport. 41, 42, 46, 47

DQN Deep Q-Learning. 10
DRL Deep Reinforcement Learning. 15, 25

GAE Generalized Advantage Estimation. 13

HJLS Hybrid Joint-Leg Stiffness. iii, iv, xiii, xvi, 22, 34, 37, 38, 41, 42, 45, 47

IJS Individual Joint Stiffness. xiii, xvi, 21, 34, 37, 38, 41, 45

MDP Markov Decision Process. 6, 7, 14
MLP Multi Layered Perceptron. 23
MPC Model Predictive Control. 15

PJS Per Joint Stiffness. xiii, xvi, 21, 34, 37, 38, 41
PLS Per Leg Stiffness. iii, iv, xiii, xvi, 22, 34, 35, 37, 38, 39, 40, 41, 43, 45, 46, 47
PPO Proximal Policy Optimization. 13, 17, 23, 55

RL Reinforcement Learning. xv, 5, 6, 7, 15, 16, 17, 24, 30

SAC Soft Actor Critic. 16

TD Temporal Difference. 6, 9, 10, 12, 55
TRPO Trust Region Policy Optimization. 13, 16, 56

WBC Whole Body Controller. 15

Chapter 1

Introduction

Animal and human-like locomotion has a significant advantage over wheeled robotics. They can traverse unstructured and challenging terrains. Currently, most of the planet applies to this description and remains unreachable for wheeled robotics. Thus, various approaches have been developed to solve quadrupedal and bipedal locomotion. This is exemplified by the robots shown in Fig. 1.1. Traditionally, this involved model-based controllers with a complex pipeline managing state estimation, path planning trajectory optimization and actuator control. Recently, this discipline has made significant progress through model-free reinforcement learning methods. This enabled the design of controllers that can follow a high-level command (walking, running, jumping, etc.) and actuate the joint motors accordingly without these extensive controls. Usually, these controllers follow a position- or torque-based paradigm [16] [17]. In the position-based paradigm, the high-level controller (RL agent) learns a position policy, which a PD controller follows. The RL agent sets desired joint positions at low fre-

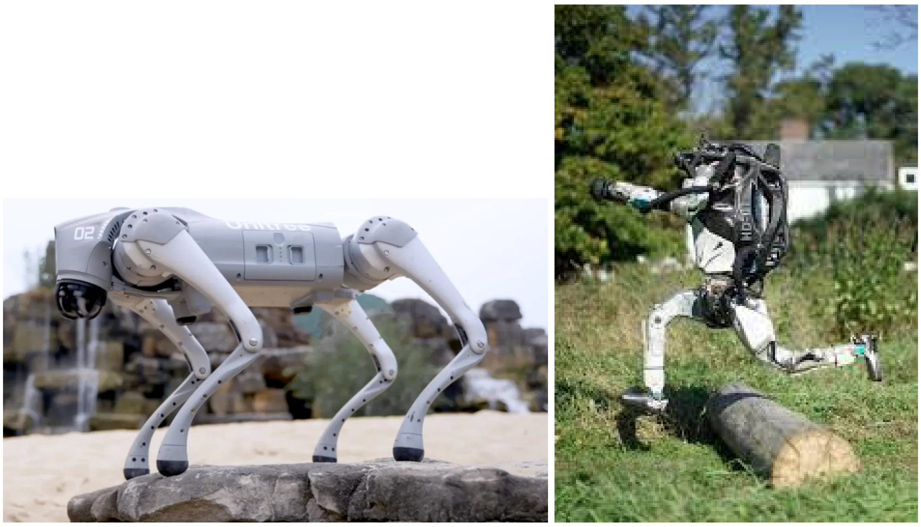

(a) (b)

Figure 1.1: Examples for quadruped locomotion (a.) [1] and biped locomotion (b.) [2]

quencies, which are then transferred into torques by the high-frequency PD controller. This control loop's P and D gains can be interpreted as each joint's respective stiffness and damping. Typically, these gains are engineered individually for each robot and task/setting and not altered afterwards. In contrast, humans and animals can adjust their stiffness and damping to handle different tasks. For example, humans stiffen their respective legs during touchdowns and relax them during the swing phases. The torque-based control circumvents this by directly learning a torque policy and shows higher compliance [18] [17]. Generally, torque-based policies are more difficult to train because they must also learn complex dynamics. Torque-based controllers must be executed at much higher speeds to achieve comparable results to position control, necessitating more powerful hardware or a smaller network.

Either additional tuning is required to specify the joint stiffness for each task, or a hard-to-train policy limits the scope of the RL for locomotion. Finding a satisfactory and robust controller that can work regardless of the task remains unsolved. Regarding nature, animals' success in movement is often attributed to their ability to vary stiffness.

Variable stiffness control is widely used in robotics, particularly in manipulator applications, to improve safety while still being able to track accurately [19, 20]. Preliminary studies from [19] have shown that applying this technique to locomotion can enhance energy efficiency. This also seems reasonable as higher stiffness is required during contact with the ground, but less is needed during the swing phase. In quadruped locomotion, research from [21] within model-based frameworks has shown that variable stiffness can reduce contact forces. However, successfully achieving locomotion with variable stiffness remains a significant challenge. This has motivated the exploration of learning joint stiffness as an integral part of the reinforcement learning process.

1.1 Contribution of this thesis

This thesis studies reinforcement-learned controllers that learn joint stiffness alongside target positions. Consequently, the robot can adjust the motor stiffness according to the task requirements. This thesis aims to investigate the impact of extending the action space to predict joint positions alongside joint stiffnesses. In the process of doing that, the following **research questions** should be answered:

- Does variable impedance control improve task performance, e.g., walking and running?
- Is it more stable for push recovery?
- Is the learned policy more energy efficient?
- Is the proposed method more robust to sim-to-real transfer?

Ideally, this approach circumvented the tedious tuning of joint stiffness.

The **contribution** of this thesis threefold:

- We introduce a software framework that includes the tools for training, evaluating and deploying reinforcement-learned agents in software and hardware.
- We propose a control paradigm that predicts joint stiffness alongside joint positions and apply this paradigm to a quadruped robot.
- We introduce stiffness grouping for adjustment to ease training and improve performance.
- We evaluate the performance of such agents and compare them to commonly used control paradigms.

1.2 Thesis overview

In chapter 2, we explain the background in reinforcement learning and explore the fundamentals and its concepts. The related works for legged locomotion are then investigated in chapter 3. Chapter 4 will then explain the new methodology and the design choices for training and deploying the policies. Finally, chapter 5 will discuss and evaluate the findings to answer the research questions.

Chapter 2

Background

2.1 Reinforcement learning

2.1.1 Introduction

When it comes to "learning", the most natural thing one can think of is an animal or human interacting with the real world. This interaction creates vast amounts of sensory data, which is used to satisfy our needs. Perhaps most people think of toddlers trying to move from one place to another. In this case, it won't be shown how to crawl; it takes advantage of this interaction between itself and the environment and starts exploring different strategies/policies to reach its goal. If some good strategies are found, they will automatically be repeated and refined. This also is a suitable description for the locomotion task, which this thesis focuses on.

In computer science, approaches to achieve learning-like behaviour are called ***machine learning***. Generally speaking, ***machine learning*** can be understood as various methods to uncover patterns in data sets and make predictions based on them or perform sequential decision-making problems [22]. This can be divided into three major methods called ***supervised-, unsupervised-*** and ***reinforcement learning***. While supervised and unsupervised methods are essential, they do not resemble the example initially. This brings us to the third method: ***Reinforcement Learning (RL)***. In this method, the learner (agent) is not provided with the best actions (labels). Instead, it must explore actions independently, dependent on receiving a reward and a state (observation), as depicted in Fig. 2.1. The agent must take sequential actions in a stochastic world. The goal for this agent is not to obtain some hidden structure, as in other learning methods, but rather a series of actions to maximise the reward signal [3, p.2-4]. Typically, these rewards differ from problem to problem and must be defined beforehand.

The dilemma that arises from this structure is the balance between exploration and exploitation, which is not present in supervised-/unsupervised learning. The maximum reward is achieved if the

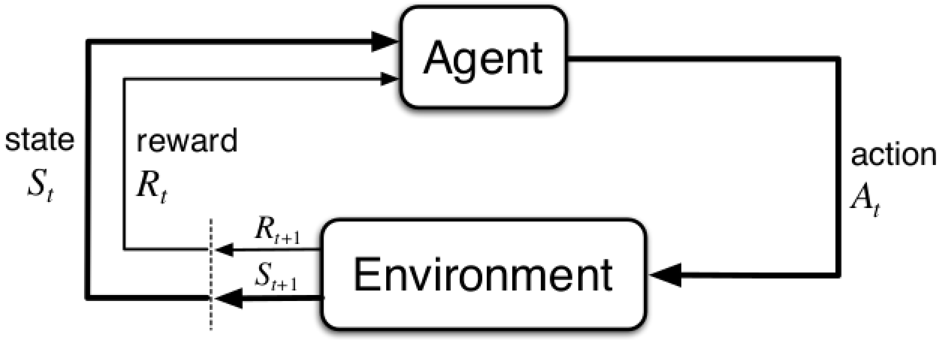


Figure 2.1: Reinforcement learning concept [3, p. 48]

agent exploits actions that previously led to high rewards, but to experience such actions, they first must be explored [3, p.2-4].

The described setting can be modelled as a Markov Decision Process (MDP), described in the following chapter 2.1.2. In simple settings, such processes can be solved via dynamic programming. Still, apart from these simple problems, RL algorithms are the only viable option to solve these problems satisfactorily [3].

Types of Reinforcement Learning

There are model-based and model-free approaches in RL. In model-based approaches, it is assumed the model dynamics (transition probabilities) of the underlying MDP are known; in model-free approaches, these dynamics must be estimated. In practical scenarios, the model dynamics are not known beforehand. This is why model-free approaches are chosen in this work and will be explained further. To solve the MDP and reach the goal of finding the best actions and maximise the overall reward, model-free approaches often rely on Monte Carlo methods and Temporal Difference (TD) methods [3, p. 23], which are grounded in the bellman equation, and will be explained in sec. 2.1.3

2.1.2 Markov Decision Process

Generally, the probability of a system being in a state is influenced by the history of the states it has encountered. MDP simplifies the analysis of a system by assuming the state is only dependent on its predecessor state. "Markov" means the future and past are independent. The learner and decision maker in the MDP is called "agent". It interacts with the environment in a series of discrete time steps ($t = 0, 1, 2, 3, ...$) and receives at each step a state $S_t \in \mathcal{S}$, which represents the environment as it tells us what discrete state the system is occupying [23, p.45]. Based on the state, S_t it selects an action $A_t \in \mathcal{A}$ and receives an immediate reward $R_{t+1} \in R \subset \mathbb{R}$ as consequence of the taken action and evolves into the next state [3, p. 69] [24, p.23]. So, a trajectory in this world could be represented like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3... \tag{2.1}$$

The dynamics of the MDP is defined by the probability p, in eq. 2.2, which defines the probability for particular states $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ occurring given the previous state s and action a [3, p.48].

$$p(s', r|s, a) := \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{2.2}$$

Given this probability 2.2 it is possible to calculate *state transition probabilities* shown in Eq. 2.3 [3, p. 48]

$$p(s'|s, a) := \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2.3}$$

as well as expected rewards for state-action pairs in Eq. 2.4 [3, p. 48]

$$r(s, a) := \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \tag{2.4}$$

The expected reward is especially interesting for RL because finding optimal states and actions becomes trivial once the expected reward for each state and action is known. In a similar setting, this is called *k-armed bandit problem.* In this analogy, one has the repeated choice to pull the lever at k slot machines, called "one-armed bandit". For each choice, you receive a reward that represents the jackpot. When done repeatedly, one will focus on the machines with the highest rewards or, in other words, on the machines that are most likely to win. In this analogy, the expected reward for a given selected action is called *value.* More on that later [3, p. 26].

In the reinforcement setting from Fig. 2.1, the agent acts at each time step and receives a simple scalar reward. The agent's objective is to maximise the current reward but to maximise the total amount of rewards, also known as the cumulative rewards or *expected return.* In the finite MDP, this can be formulated as in Eq. 2.5, where T denotes the final time step (formerly known as time horizon) and G_t represents the expected return. [3, p. 53-55]

$$G_t := R_{t+1} + R_{t+2} + R_{t+3}....R_T \tag{2.5}$$

This concept of the *expected return* makes if the interaction between agent and environment is finite or can be broken down into sub-sequences, formerly known as *episodes.* In this convention, each episode starts in an initial state and terminates with a terminal state.

However, this convention often cannot be applied, especially in the case of controlling tasks where the interaction can not be broken down into episodes. For continuing tasks, the time horizon would be $T = \infty$, and the expected return is impossible to calculate. This is why the *discounted return* is introduced in eq. 2.6, where $0 < \gamma < 1$ is the discount rate. In this way, the return cannot rise indefinitely, as each reward is discounted more the further it is received in the future.

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.6}$$

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2.7}$$

Another aspect this introduces is that to maximise this function, and short-term rewards must be prioritised while maintaining long-term goals. Recalling the *k-armed bandit problem*, the *value* represents the *expected reward*, indicating "how good" it is for an agent to be in a given state (or perform a given action for a given state). This "goodness" is determined by rewards it can expect to receive in the future, which depend on the actions taken.

A decision rule is required to determine which actions to take for a given state. This is conventionally called a policy $\pi(s)$. It is important to distinguish between *deterministic policy* $\pi(s)$ and *stochastic policy* $\pi(a|s)$. A deterministic policy is a simple decision rule that, given a state, executes a clearly defined action. On the other hand, a *stochastic policy* represents the actions for a given state as a probability distribution. [3, p. xx + 80]. Like *p*, the stochastic policy $\pi(a|s)$ is described as a probability distribution, but this time for selecting an action $a \in \mathcal{A}$ given a state $s \in \mathcal{S}$.

Since the expected return (or value) depends on both the current state and the subsequent actions that follow, and the decision rule has been defined, it can now be expressed as a function of a given policy π and state s, as shown in Eq. 2.8 [3, p. 58].

$$v_\pi(s) := \mathbb{E}_\pi \left[\mathrm{G_t} | \mathrm{S_t} = \mathrm{s} \right] = \mathbb{E}_\pi \left[\sum_{\mathrm{k}=0}^{\infty} \gamma^{\mathrm{k}} \mathrm{R}_{\mathrm{t+k+1}} | \mathrm{S_t} = \mathrm{s} \right] = \sum_{\mathrm{a}} \pi(\mathrm{a}|\mathrm{s}) \sum_{\mathrm{s}',\mathrm{r}} \mathrm{p}(\mathrm{s}', \mathrm{r}|\mathrm{s}, \mathrm{a})[\mathrm{r} + \gamma \mathrm{v}_\pi(\mathrm{s}')] \tag{2.8}$$

While this function provides the expected reward for a given state, how is the return for a state-action pair represented? This is where the *q-function* or *action-value function* comes in, defined as in

Equation 2.9:

$$q_\pi(s,a) := \mathbb{E}_\pi\left[\mathrm{G_t}|\mathrm{S_t}=\mathrm{s},\mathrm{A_t}=\mathrm{a}\right] = \sum_{\mathrm{s}',\mathrm{r}} \mathrm{p}(\mathrm{s}',\mathrm{r}|\mathrm{s},\mathrm{a})[\mathrm{r}+\gamma \mathrm{v}_\pi(\mathrm{s}')] \tag{2.9}$$

One policy always produces more or equal value than all other policies $v_{\pi^*} \geq v_\pi$. It is called *optimal policy* and cast into a mathematical description, and it can be defined as π^* in Eq. 2.10

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left[r(s) + \gamma \sum_{s'} p(s'|s,a)v(s') \right] = \underset{a}{\operatorname{argmax}} \left[Q(s,a)\right] \tag{2.10}$$

This policy then produces the optimal state-value function $v_*(s)$ and the optimal action-value function $q_*(s,a)$. Finding the optimal policy is the primary goal of reinforcement learning, and methods for finding these are described in sec. 2.1.1.

2.1.3 Reinforcement Learning Approaches: Value-Based and Policy-Based Methods

Experience in the model-free approaches is accumulated via executing a policy π and recording all properties relevant to the transitions s, r, a. The Monte Carlo estimation states that the value of a state can be estimated via repeated sampling of policies and averaging over the cumulative reward observed while visiting the state. This method is easy to understand and requires no knowledge about the environment. However, it wastes information about the state transitions and requires the episode to finish until the values are updated. TD learning overcomes this by learning from every experience/transition [25] [3].

Furthermore, model-free approaches can be grouped into two branches: *Value-based methods* and *policy-based methods.* The main difference between these approaches is that value-based ones learn the *values/q-values* of any given state and extract the optimal policy from it. Policy-based methods, on the other hand, learn the optimal policy directly [26, 109]. The actor-critic method explained later relies on both of these concepts. This is why it is explained here

Value-based methods

In these methods, first, the *value-* or *q-function* is estimated for each state, either by the use of Monte Carlo estimation or TD evaluation. The *q-function* will be used further in this chapter for easier computation of the optimal policy [25].

To estimate the *q-function*, the bellmann equation 2.11 is used, and the policy improved after taking the action, which maximises the q function according to Eq. 2.10. In this formulation, Q_k holds the current value estimation, and Q_{k+1} is then the updated value, respectively [3].

$$Q_{k+1}(s,a) \leftarrow R(s) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_k(s',a') \tag{2.11}$$

In sample-based approaches, the transition probabilities are unknown. Therefore, TD is used to obtain the value from experience (shown in Eq. 2.12) [27]. TD receives the reward r for the state s and adjusts the estimations of q-values based on the reward, the next state and the learning rate. With the learning rate, the first hyperparameter is introduced.

$$Q_{k+1}(s,a) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q_k(s,a)}_{\text{current value}} + \alpha \cdot \underbrace{\left(r + \gamma \cdot \underbrace{\max_a Q_k(s',a')}_{\text{estimate of optimal future value}} \right)}_{y_k \text{ new value (TD target)}} \tag{2.12}$$

$$\Rightarrow Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \underbrace{\left(r + \gamma \max_a Q_k(s',a') - Q_k(s,a)\right)}_{\delta_k} \tag{2.13}$$

The error metric δ_k, highlighted in Eq. 2.13, tells how far the expected return is off the actual return and is called TD error [27].

So far, it has been explained how q-values are learned for every state through random exploration and choose an optimal policy based on the learned values. This approach necessitates holding all values in a tabular storage, which is unfeasible given a large or continuous state space. Consequently, it is common to estimate the *q-function* via a linear function approximator such as $Q(s,a;\theta) \approx Q(s,a)$. In practice, this is usually done via deep neural networks, leading us to the concept of Deep Q-Learning (DQN). This network can learn to predict the q-values given a loss function like the one shown in Eq. 2.14. The term $\mathbb{E}_{\mathrm{s,a}\sim\pi}$ indicates that the expected (averaged) quadratic error is calculated for actions taken from a given policy. In this equation, y_k represents the target defined in Eq. 2.12. The parameters of the network can further be adjusted (given in Eq. 2.15) using the stochastic gradient descent rule [28].

$$L_k(\theta_k) = \mathbb{E}_{\mathrm{s,a}\sim\pi}\left[(\mathrm{y_k} - \mathrm{Q(s,a;}\theta_\mathrm{k}))^2\right] \tag{2.14}$$

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_{\theta_k} L_k \tag{2.15}$$

Once the *q-values* are learned, the *optimal policy* can be extracted using Eq. 2.10.

Policy-based methods

The policy-based methods seek to directly improve the policy without learning the *action-value functions* beforehand. Therefore, the policy is parametrised $\pi(a|s;\theta) = \pi_\theta(a|s)$ and the parameters are adjusted to maximise some performance metric, like in the REINFORCE algorithm [29].

The performance metric used is the state-value function. It can be maximised by derivation of the parameters θ and performing gradient ascend. This is shown in Eq. 2.16 to 2.17 [3, p. 321- 329].

$$L^{PG}(\theta_k) = v_\pi(s,\theta_k) = \sum_a \pi_{\theta_k}(a|s) \cdot Q^\pi(s,a) \tag{2.16}$$

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta L^{PG}(\theta_k) \tag{2.17}$$

This gradient can be computed using the policy gradient theorem, from [30] [31] [3, p. 326-329], This theorem simplifies the calculation and can be expressed as Eq. 2.18. The expectation value $\mathbb{E}_{\mathrm{A}\sim\pi(\cdot|\mathrm{s};\theta)}$ of Eq.2.19 can be estimated via Monte Carlo estimation.

$$\nabla_\theta V(s;\theta) = \nabla_\theta \sum_{a\in A} \pi(a|s;\theta) \cdot Q_\pi(s,a) = \sum_a \pi(a|s;\theta) \left[\nabla_\theta log(\pi(a|s;\theta)\ Q_\pi(s,a)\right] \tag{2.18}$$

$$\Rightarrow \nabla_\theta V(s;\theta) = \mathbb{E}_{\mathrm{A}\sim\pi(\cdot|\mathrm{s};\theta)} \left[\nabla_\theta \mathrm{log}(\pi(\mathrm{A}|\mathrm{s};\theta)\ \mathrm{Q}_\pi(\mathrm{s},\mathrm{A})\right] \tag{2.19}$$

Since policies are not randomly explored like value-based methods, it is vital to ensure exploration in this approach. A deterministic policy would consistently be executing the same actions given the states. Therefore, it would be hard to calculate the gradient. In the case of a deterministic policy, one can ensure exploration, for example, by adding noise to the actions or by choosing a stochastic policy [32]. Therefore, the policy is defined as stochastic over the output of the network $h(s,a;\theta_k)$ as a soft-max distribution 2.20 [3, p. 322].

$$\pi(a|s,\theta) := \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}, \tag{2.20}$$

2.1.4 Actor-Critic RL methods

The actor-critic combines both methods. The actor represents the learned policy $\pi(a|s;\theta)$ and the critic the learned value $q(s,a;\mathbf{w})$. Applied to an actual problem, the critic learns to rate the actor's actions via q, and the actor learns which actions to take. This concept is illustrated in Fig. 2.2.

Putting it all together, the learning process starts with observing a state S_t. Then an action is sampled given the distribution of the actor $A_t \sim \pi(\cdot|S_t;\theta_t)$ and record the new state S_{t+1} and reward

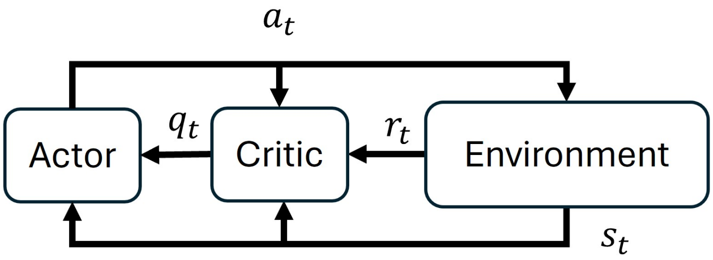


Figure 2.2: Actor critic concept

R_t. The critic is updated as described in 2.1.3. The current estimations $Q_t = q(S_t, A_t; \mathbf{w}_t)$ and $Q_{t+1} = q(S_{t+1}, A_{t+1}; \mathbf{w}_t)$ are then used together with the reward signal received to calculate the TD error and target (given in Eq. 2.21-2.22).

$$y_t = R_t + \gamma \cdot q(S_{t+1}, A_{t+1}; \mathbf{w}_t) \tag{2.21}$$

$$\delta_t = R_t + \gamma \cdot q(S_{t+1}, A_{t+1}; \mathbf{w}_t) - q(S_t, A_t; \mathbf{w}_t) \tag{2.22}$$

This TD error is then used to adjust the weightings of the critic, and the predicted *action-value* is used to calculate the policy-gradient (in Eq. 2.23, 2.24). α and β represent the respective learning rates [3].

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \delta_t \cdot \frac{\partial q(S_t, A_t; \mathbf{w})}{\partial \mathbf{w}} \tag{2.23}$$

$$\theta_{t+1} = \theta_t + \beta \cdot q(S_t, A_t; \mathbf{w}) \cdot \frac{\partial \log \pi(A_t | S_t; \theta)}{\partial \theta} \tag{2.24}$$

As the critic initially gives random scores (q), combining this "REINFORCE" method with a baseline is common. That means that Q_π is replaced with the difference $\hat{A}(s, a) = Q(s, a) - V(s)$, which is called *advantage*. As the name tells, this term states how much better or worse the reward for a particular action is than the average reward for an action in this state [33]. Since the advantage also depends on the critic and the action carries the same letter, a hat is added to the notation.

2.1.5 Generalized Advantage Estimation (GAE)

An important aspect to consider in using actor-critic methods is estimating the advantage and, subsequently, analysing the *action-value* or *state-value function*. As already mentioned, this can be done using the Monte Carlo method, which estimates the value of a state by simply averaging over the reward received since the given state s_t of this episode. TD uses the one or n-step look-ahead for estimation. On the one hand, Monte Carlo only relies on measured values in an episode. These measurements can vary greatly during episodes, leading to high variance but slight bias. On the other hand, TD calculates values based on other estimates, which reduces the variance but induces high bias.

Generalized Advantage Estimation (GAE) is an improvement in the estimation for the advantage used to compute the policy gradient. It introduces another hyperparameter λ, enabling a trade-off between bias and variance. The estimated advantage is calculated in Eq. 2.25.

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} := (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + ...\right) = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V \tag{2.25}$$

2.1.6 Proximal Policy Optimisation (PPO)

Proximal Policy Optimization (PPO) [34] is a common policy-gradient method that builds on the work of Trust Region Policy Optimization (TRPO) [35], which introduced a surrogate objective to maximise Eq. 2.26 instead of the traditional objective introduced in Eq. 2.16. This subsequently led to better convergence and improved the performance of the learning process. In this Eq., the θ_{old} refers to the old policy parameters. Furthermore, the probability ratio between old and new policy can be written as $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|s_t)}}$. The same policy results in $r(\theta_{old} = \theta) = 1$. This ratio serves as a weighting point for the advantages of the loss.

$$L^{CPI}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(A_t \mid S_t)}{\pi_{\theta_{\text{old}}}(A_t \mid S_t)}\hat{A}(S_t, A_t)\right] = \mathbb{E}_t\left[r_t(\theta)\hat{A}(S_t, A_t)\right] \tag{2.26}$$

Without constraints, maximising this objective would lead to extensive policy updates. PPO solves this issue by introducing the objective 2.27, which holds the probability ratio in a reasonable range. The objective is then the smaller of the unclipped and the clipped one.

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(S_t, A_t)\right)\right] \tag{2.27}$$

The parameter ϵ is an additional hyperparameter defining the probability ratio's clipping range. Hyperparameters are variables set before the training process begins and determine the learning behaviour of the model. They can vary from task to task, so there are usually sweet spots for these parameters, but they do not apply to all problems.

Applied to the actor-critic setting, the objectives, which should be maximised or minimised, must be combined. The total aim of the PPO algorithm is then composed of the actor loss L^{CLIP} the value loss L^{VF} and an entropy bonus $S[\pi_\theta]$. This total loss must be minimised. Therefore the objectives L_t^{CLIP} and $S[\pi_\theta](s_t)$ are maximised, and the value loss L_t^{VF} minimised.

$$L^{PPO}(\theta, w) = \mathbb{E}_t\left[-L_t^{CLIP}(\theta) + c_1 \cdot L_t^{VF}(\mathbf{w}) - c_2 \cdot S[\pi_\theta](s_t)\right] \tag{2.28}$$

2.2 Reinforcement learning in locomotion

Quadruped locomotion is formally understood as a MDP, explained in chapter 2.1.2. Applied to locomotion, the state S_t can be represented via sensory information. Usually, this data does not represent the whole information, so this partial observable state is rebranded as *observation* O_t. The agent or robot, therefore, receives a reward R_t and an observation O_t from the environment and draws an action derived from the policy π, which lets the robot evolve into another state. Data is gathered chiefly via Monte Carlo rollouts. In a Monte Carlo rollout, multiple trajectories are simulated in parallel. For each time step in a trajectory, the value of a state must be estimated, and based on that, actions are calculated [3, p. 183]. This is done for a specified amount of time steps to gather enough data and perform a value-based and/or policy-based-method, as described in section 2.1.3, to improve the policy.

This is often accomplished by using the actor-critic method, as explained in section 2.1.4. The critic serves by estimating a value for a given state, and the actor predicts an action, shown in Figure 2.3. First, the actor and critic are trained in combination with the simulation (as indicated by the dotted lines). If the training is finished, the critic is dropped, and the actor applied to the real implementation (indicated by the continuous line)

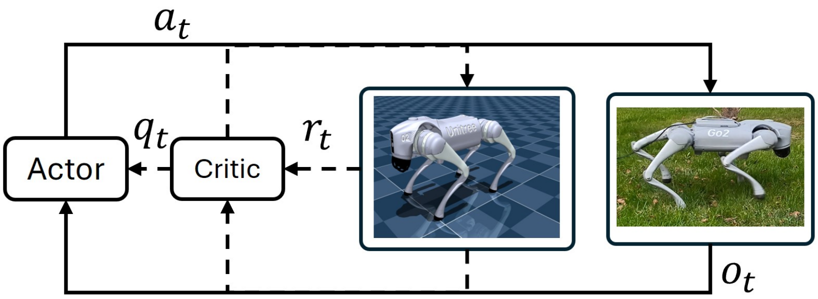


Figure 2.3: Actor-critic paradigm in the context of quadruped locomotion

During training, the data points gathered to perform an update are called *batch.* As a rule of thumb, it can be said that the greater the batch size, the more stable the learning process, but also the slower the computing [36]. However, this faces a researcher with the trade-off between speed and stable learning.

Chapter 3

Related works

3.1 Legged locomotion in robotics

Legged locomotion is a subject that has been studied extensively in robotics. Model-based control for quadruped locomotion has been the primary way to achieve it since the 1980s [37]. However, due to the emergence of machine learning and the increased computing power in recent years, RL based methods have experienced significant progress.

3.2 Model-based control methods

The primary goal in locomotion tasks is for the robot to follow high-level commands, such as base velocities or rotation speed. Model-based control methods often use hierarchical architectures with multiple control loops, as shown in Figure 3.1. The high-level loop uses a gait scheduler to calculate desired trajectories based on commands, while the Model Predictive Control (MPC) controller outputs ground reaction forces using desired and estimated states. The inner loop employs a Whole Body Controller (WBC) to compute desired joint positions from these forces, foot joint angles, and velocities. A high-speed joint-level loop then applies torques via a PD controller and sensory data is used for state estimation [4].

3.3 Reinforcement learning in locomotion

With the recent increase in computing power and the rapid development of deep reinforcement learning (Deep Reinforcement Learning (DRL)), RL-based controllers have emerged as alternatives to traditional model-based controllers. Unlike model-based methods, which are predictable, transparent, and sample-efficient but suffer from simplified models, restrictive assumptions, and compute-intensive algorithms,

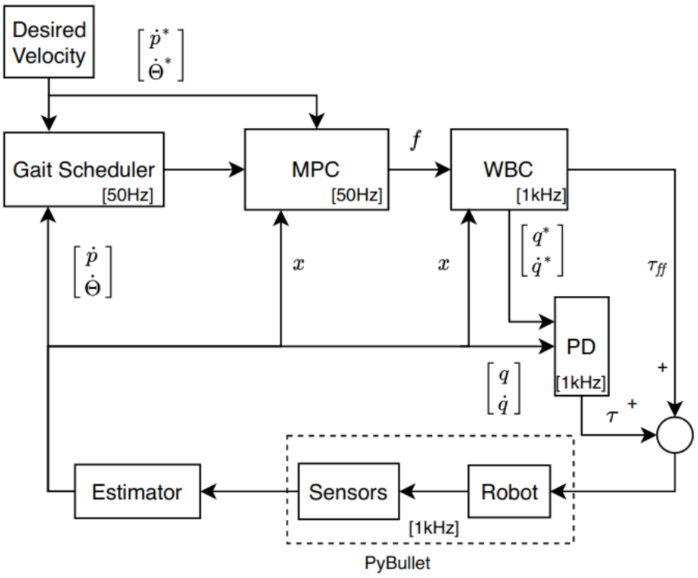

Desired Velocity
$\begin{bmatrix} \dot{p}^* \\ \dot{\Theta}^* \end{bmatrix}$
Gait Scheduler
[50Hz]
MPC
[50Hz]
f
WBC
[1kHz]
$\begin{bmatrix} \dot{p} \\ \dot{\Theta} \end{bmatrix}$
x
x
$\begin{bmatrix} q^* \\ \dot{q}^* \end{bmatrix}$
τ_{ff}
PD
[1kHz]
+
$\begin{bmatrix} q \\ \dot{q} \end{bmatrix}$
τ
+
Estimator
Sensors
Robot
[1kHz]
PyBullet

Figure 3.1: Model predictive control on a quadruped [4]

RL-based methods offer the advantage of reduced manual design and the ability to adapt to diverse scenarios. They are also efficient during deployment. However, RL-based methods are often intransparent, unsafe and computationally expensive during the training phase. Moreover, their reliance on simulation introduces challenges in sim-to-real transfer, as overfitting to the simulation environment can occur [38, 39].

Figure 3.2 provides an overview of the most influential works in RL-based locomotion on quadrupeds. Some implementations have surpassed their model-based counterparts in performance [5, 6, 8, 10].

In the first identified work [5], they trained policies for the Anymal robot with TRPO [35] and introduced an actuator network with supervised learning to mimic the actuator dynamics. With this approach, they could deploy a policy for standing up after a fall and one that could follow high-level commands like walking and turning. However, this work was not able to walk on more challenging terrain, which was then addressed by the following work of [6], which employed a student-teacher learning paradigm and introduced a learning curriculum to train a policy that can traverse more challenging terrain like mud, snow and rubble with only proprioceptive sensory information. This showed the effectiveness of student-teacher learning combined with a locomotion curriculum. Other works like [7, 11] demonstrated learning locomotion by imitation. [8] introduced a multi-expert framework that consists of multiple policies being trained for specific tasks. Each policy was trained using the Soft Actor Critic (SAC) [40] and, during deployment, blended in according to online synthesis. This approach demonstrated the autonomous deployment of trotting, steering, and fall recovery were autonomously deployed. The works that followed [9, 10, 12] concentrated on incorporating exteroceptive information to achieve obstacle avoidance and traversing more challenging terrain. Since exterocep-

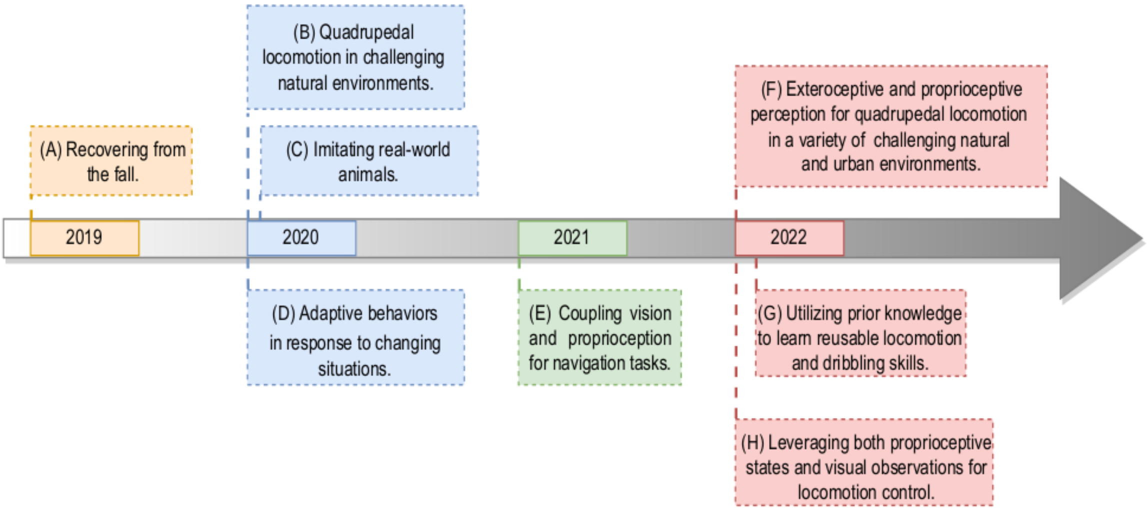

(B) Quadrupedal locomotion in challenging natural environments.
(A) Recovering from the fall.
(C) Imitating real-world animals.
(F) Exteroceptive and proprioceptive perception for quadrupedal locomotion in a variety of challenging natural and urban environments.
2019
2020
2021
2022
(D) Adaptive behaviors in response to changing situations.
(E) Coupling vision and proprioception for navigation tasks.
(G) Utilizing prior knowledge to learn reusable locomotion and dribbling skills.
(H) Leveraging both proprioceptive states and visual observations for locomotion control.

Figure 3.2: Timeline of RL based locomotion in recent years: (A)Learning agile and dynamic motor skills for legged robots [5], (B) Learning quadrupedal locomotion over challenging terrain [6], (C) Learning Agile Robotic Locomotion Skills by Imitating Animals [7], (D) Multi-expert learning of adaptive legged locomotion [8], (E) Vision and Proprioception for Navigation of Legged Robots [9], (F) Learning robust perceptive locomotion for quadrupedal robots in the wild [10],(G)Imitate and Repurpose: Learning Reusable Robot Movement Skills From Human and Animal Behaviours [11], (H) Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers [12]; figure from: [13]

tive sensors are noise-prone, other works proposed using a proprioceptive history to estimate future observations [41] or privileged information [42] and achieved robust locomotion on challenging terrains. Training in such policies is mostly very complex and takes tremendous time. The work of [36] addressed this by introducing a learning framework that enabled learning walking in minutes with a massive parallelisation of computation on GPU within Isaac gym using PPO. This work laid the foundation for further works in the works of [41, 43].

3.3.1 Common control paradigms

The two most common control choices in a reinforcement learning setting are **position-based control** and **torque-based control**. Fig. 3.3 shows illustrations of these concepts. **Position based control** is the most usual choice for the action space [16]. It is common to add or subtract joint position values $a_t = q$ from a nominal (standing) position q_{stand}, shown in eq. 3.1. The θ indicates that the action was calculated according to the parameters of the actor network.

$$q_t^{target} = q^{stand} + a_t(\theta) \tag{3.1}$$

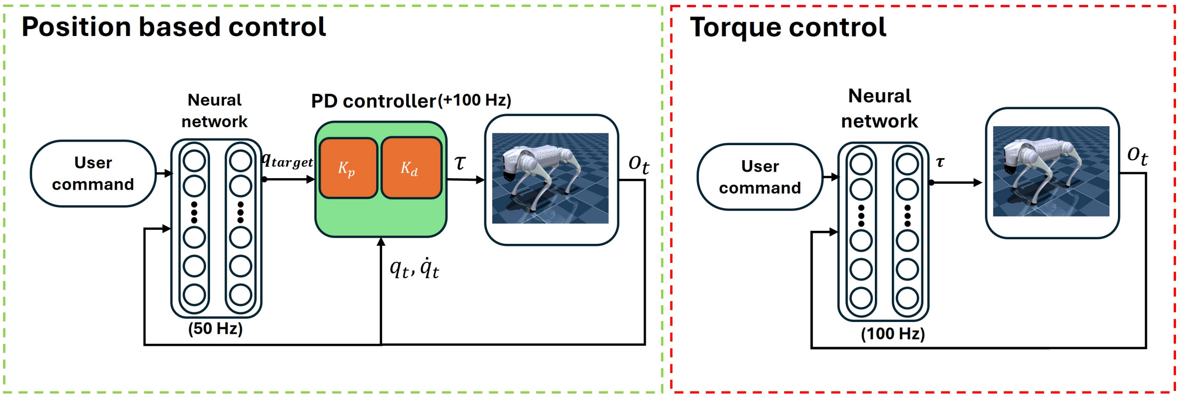


Figure 3.3: **Common control concepts:** Position-based control (left) and torque-based control (right)

This choice provides a good starting point for policy exploration, as it explores all the configurations around this position and stabilises the training in the early stages [6,17,41,43,44]. In this control concept, the actor's predicted joint positions are fed into a low-level control loop, which operates at higher frequencies (200-400 Hz) compared to the outer loop (typically 50 Hz) [5, 12, 41]. This PD controller in the low-level loop converts these positions to torques for the robot motors, calculated as shown in Eq. 3.2:

$$\tau_t = K_p(q_t^{target}(\theta) - q_t) - K_d\,\dot{q}_t \tag{3.2}$$

Besides the advantages of stable learning and good starting conditions, it has the advantage of executing the network at lower speeds, enabling the deployment of more complex networks, which opens more design freedom. However, the low-level controller's P and D gains must be manually set in this configuration, representing each joint's respective stiffness and damping. Usually, they must be altered for different tasks and robots [16, 17].

[45] investigated the impact of the p-gains, suggesting that large proportional gain leads to instabilities in training. In contrast, the low proportional gain has significant tracking errors and behaves like a torque controller. Other research studied the impact of derivative gain. [46] showed small derivative gains result in learning instabilities, and large gains prevented tracking the target velocity.

To circumvent this PD controller, [17] and [18] studied torque control as an alternative and applied it to quadruped and biped locomotion. In this control, the actions predicted are directly applied to the robot motors: $\tau = a(\theta)$.

Although this control shows higher achievable rewards in the long term, it must be executed at higher speeds (usually 200 Hz) to perform similarly to position control and is more challenging to train. The higher control speeds limit the design freedom of torque-based controllers. The researcher will be faced with the choice of using either more powerful hardware or simpler networks.

3.4 Variable Stiffness

As variable stiffness is extensively studied in manipulator tasks [19, 20] for increasing safety in collaborative tasks or improving energy efficiency and enhancing tracking capabilities, the application to locomotion seems reasonable. [21] demonstrated a model-based control that adapts stiffness according to the contact force during walking for a quadruped. This approach was able to walk successfully. Further, this work demonstrated that variable stiffness can reduce contact forces. In reinforcement learning, the work [47] studied the impact of including joint stiffness alongside joint positions. The torque was then calculated by Eq. (3.3).

$$\tau_t = K_p(\theta)(q_{t,\, target}(\theta) - q_t) - K_d(\theta)\dot{q}_t \tag{3.3}$$

This work did apply their approach to locomotion but showed the superiority of this concept against torque and position control for two scenarios: a single-legged hopper and a manipulator. The single-legged hopper's task was to jump as high as possible, and the manipulator had to follow a target trajectory while maintaining a constant contact force. The single-legged hopper achieved more considerable jumping heights, and the manipulator maintained a continuous contact force during reference motions with the adjustment of the action space. This motivates including stiffness alongside joint positions into the action space and applying it to quadruped locomotion.

Building upon the previous works, we investigate the opportunity of joint stiffnesses alongside joint positions in the action space for a reinforcement learning policy applying to quadruped locomotion.

Chapter 4

Methods and Design

This chapter presents the methodology and the design choices. Section 4.1 introduces our proposed methodology, and section 4.2 explains the design choices made for training the policies. To encourage future research and development in this field, we publish our code[1] for training any policy we demonstrate in this thesis.

4.1 Proposed Methodology

We aim to train a policy π_θ with parameters θ that can follow high-level velocity commands. The velocity commands include lateral velocity v_{xy}^{cmd} as well as angular rotation speed ω_z^{cmd}. The prediction of joint targets along joint stiffnesses should accomplish this.

4.1.1 Variable stiffness

In addition to the previously introduced methods, our approach incorporates the prediction of joint stiffnesses, requiring the determination of 12 additional control actions. An overview of the proposed methodology is illustrated in Fig. 4.1. To simplify the control problem and limit the action space, we propose four distinct groupings of joint stiffnesses, as illustrated in Figure 4.1.

- **Individual Joint Stiffness (IJS):** This control mode predicts the stiffness for each joint individually, requiring the prediction of one value per joint. This paradigm inflates the action space to 24 dimensions.
- **Per Joint Stiffness (PJS):** This method predicts individual stiffness values for each joint group. For a typical quadruped robot, the joints are categorised into three groups: hip, thigh, and knee.

---

[1] `https://github.com/gautica/TALocoMotion`

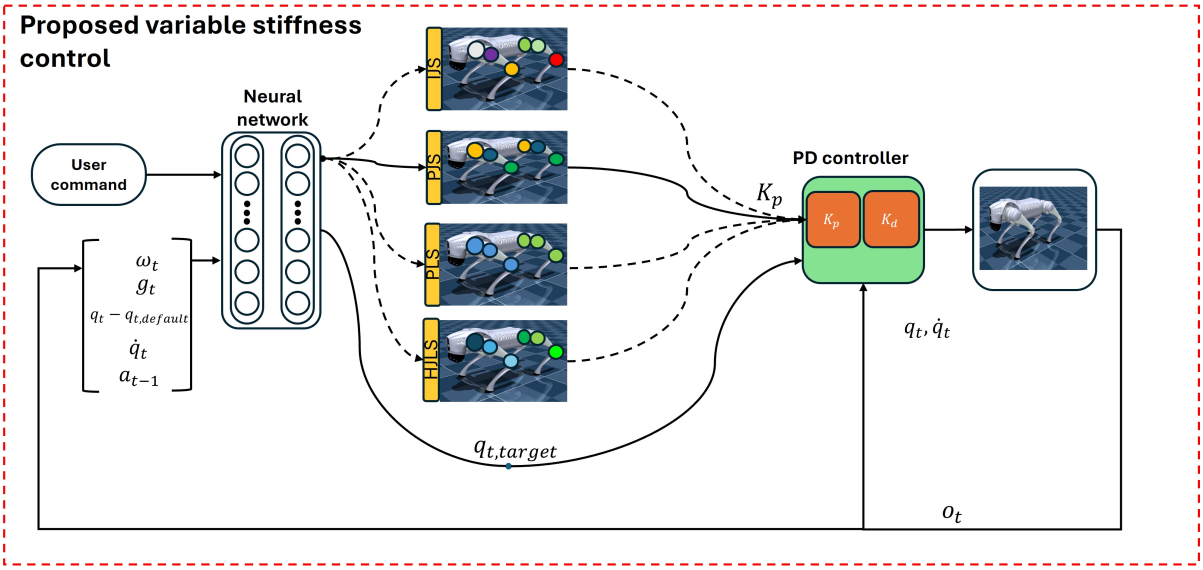


Figure 4.1: **Control overview for proposed methodology:** Different stiffness control approaches, PJS: "Per joint stiffness", PLS: "Per leg stiffness", IJS: "Individual Joint Stiffness", HJLS: "Hybrid Joint Leg Stiffness"

Consequently, three stiffness values must be predicted, extending the action space to 15 dimensions.

- **Per Leg Stiffness (PLS):** This approach predicts a single stiffness value for each leg, resulting in four additional control actions and an action space of dimension 16.
- **Hybrid Joint-Leg Stiffness (HJLS)** This hybrid approach combines PJS and PLS by spanning a stiffness matrix using the outer product. This way, three dimensions are necessary for joint groups and four for every leg, which results in an action space of dimension 19.

Regarding HJLS, the stiffness matrix S, as defined in equation 4.1, is calculated as the outer product of the leg stiffness vector $\mathbf{k}^{\mathbf{l}} \in \mathbb{R}^4$ and the group stiffness vector $\mathbf{k}^{\mathbf{j}} \in \mathbb{R}^3$. The resulting matrix is flattened into a single stiffness vector to effectively represent the predicted stiffness values.

$$\mathbf{S} = \mathbf{k}^{\mathbf{l}} \otimes \mathbf{k}^{\mathbf{j}} \tag{4.1}$$

A single output action sets both stiffness K_P and damping K_D of each joint by imposing a fixed relationship between stiffness and damping, similar to the critical damping law. We set the relationship as in eq, inspired by the ratio between PD gains in the works of [16] [36]. 4.2.

$$K_D = 0.2\sqrt{K_P} \tag{4.2}$$

Parameter	**MJX Implementation**
Number of environments	4096
Time steps per rollout	48
Number of mini-batches	4
Batch size	196608
Mini-batch size	49152
Number of learning epochs	5
Discount factor γ	0.99
GAE parameter λ	0.95
Learning rate	1.e-4
Learning Schedule	adaptive
Value loss coefficient	1
Entropy coefficient	0.01
Optimizer	Adam
Desired KL	0.01
Maximum Grad Norm	1
Episode length	500(10s)
Number of Obs	48
Number of Privileged Obs	59
Number of Actions	12
Action scale	0.5
Hip scale	0.6
Simulation Frequency [Hz]	200
Control Frequency [Hz]	50

Table 4.1: Hyperparameters for the proposed approach

4.2 Design choices

This section explains further the implementation details and design choices for the learning framework and simulation environment.

4.2.1 Reinforcement Learning framework

As explained in section 3.3, an actor-critic method is used with PPO, given the advantages explained in section 2.1.6–2.1.5. As a starting point of this work, the learning counterpart of the paper [36] is used and employs PPO. We train each policy with 4096 environments in parallel for 2000 epochs. The hyperparameters used are listed in table 4.1 and applied to each approach.

Network

For both actor and critic, we employ a simple Multi Layered Perceptron (MLP) architecture, as it is sufficient, a common choice for this problem [36] [5] [41] and allows us to focus on the effects of variable stiffness in reinforcement learned locomotion. The architecture consists of an input layer with the dimension of the observation followed by two dense blocks that inherit two hidden layers with 256 dimensions each and an output layer whose size is given by the choice of control mode. In the case of position-based control, only twelve actions are predicted. Figure 4.2 shows an overview of the described architecture.

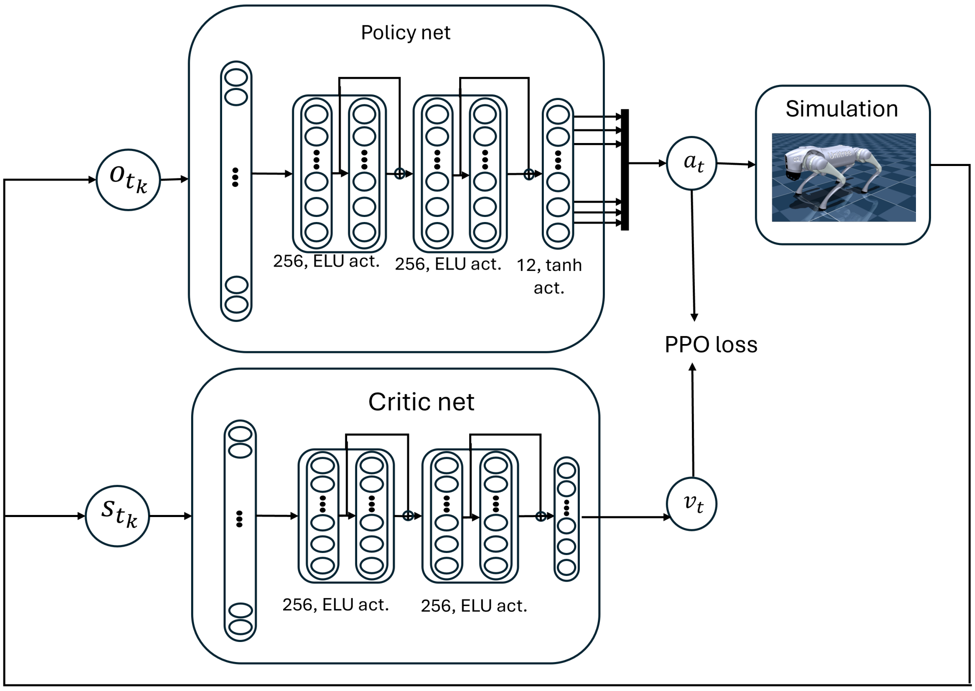


Figure 4.2: Network architecture of actor and critic. For every step in the simulation, the actor receives its observation vector o_{t_k} and the critic receives its privileged observation s_{t_k}. Therefore, the actor predicts an action and the critic a value for the given state.

Unlike in previous works [36] and [5], which incorporate soft boundary learning for action spaces, we select a more straightforward approach by attaching a tanh activation function to the actor's output layer. This ensures that the output is constrained to the range [-1, 1], a common choice for continuous action spaces [39] and suitable for our application. We also investigate the impact of including observation histories directly into the policy network, as suggested by [36]. However, our experiments indicate that this has no significant impact on learning performance and leads to the same conclusion as in the works of [36], so we exclude it from the final architecture. While the chosen network design has not been extensively experimented on within this work, it represents a straightforward and effective baseline for reinforcement learning tasks for quadruped locomotion. Discussions with collaborators inform additional insights into activation functions and architectural choices.

4.2.2 Simulation environment

The simulation platform plays a huge role in the development of RL based controllers, as the accuracy of the simulation is crucial for the overall performance of the controller. The most common choice in the research community is probably NVIDIA's ISAAC GYM [48] or its successor Isaac sim [49]. However, this environment is poorly documented, and the predecessor lacks support. Google DeepMind's mujoco [50] and its recently published vectorised version, Mujoco-MJX, offer a promising alternative. Mujoco is generally known to be physically accurate, openly developed and widely used as a DRL verification simulator. While many works rely on Isaac Gym, we choose to use Mujoco-MJX, given the advantages. The simulation is computed for every $0.002s$, and the control step is executed at $0.02s$. The Go2 robot is loaded into the environment with collision geometries, as depicted in Fig. 4.3, taken from [14]. The collisions between the collision spheres against each other have been excluded for speeding up the computation. This way, only collisions between these spheres and the floor are detected, not between each other.

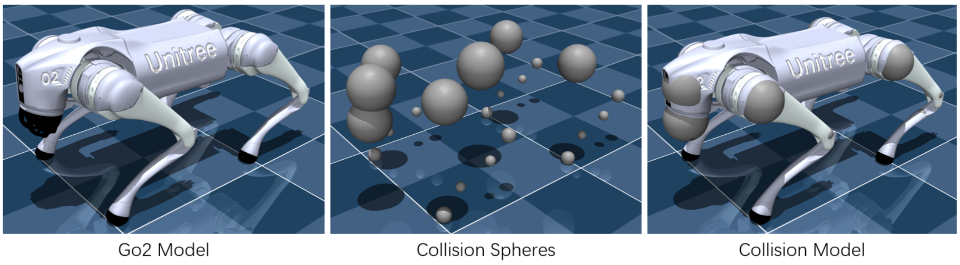


Figure 4.3: Go2 MJX implementation from [14]

The overall environmental settings are summarised and listed in the tab. 4.1.

Observations

The observations are chosen, considering the available sensor data of the robot, described in chapter 4.3, and the capabilities of the simulation environment. As in other works, we consider the use of a student/teacher architecture [6] [51] [52]. Still, simpler architecture is chosen because this approach is data inefficient, and Mujoco-MJX does not provide the height information, which is the central part of the privileged information [6, 41]. In the presented approach, observations passed to the actor contain data that can also be directly obtained from sensor feedback. The privileged information adds domain information to the observation. The privileged observation is passed to the critic and the observation to the actor. The passed information and the individual scaling are detailed in the tab. 4.2. Since no information about the surroundings (exteroceptive) is passed, the paradigm is called *blind locomotion.* Scaling the data of the observations is inspired by [36] [53] and chosen so that each entry does not exceed the area of -10, 10.

Observation	Privileged Observation	Normalisation (scaling)
Local Velocity	Local Velocity	2.0
Local Angular Velocity	Local Angular Velocity	0.25
Projected Gravity	Projected Gravity	1.0
Joint Position Difference ($q - q_{default}$)	Joint Position Difference ($q - q_{default}$)	1.0
Joint Velocity	Joint Velocity	0.1
Last Action	Last Action	1.0
Command	Command	2.0
	K_p factor (Proportional Gain)	1.0
	K_d factor (Derivative Gain)	1.0
	Motor Strength	1.0
	Geometric Friction	1.0
	Body Mass	1.0
	Kick	1.0
	Contact	1.0

Table 4.2: Observation (actor), Privileged observations (critic) and Normalisation

Early termination

Early termination plays a vital role in data efficiency for reinforcement learning. In the case of locomotion, we terminate for states and reset the environment which we do not think is beneficial to explore or some states from which it is unlikely to recover into a regular walking policy. Following the work [36], we terminate an environment if the robot's hips, trunk, or lidar sensor collides with the floor. Additionally, we terminate if the torso's z-axis forms an angle of over 90 degrees with the global z-axis, which indicates that the trunk is perpendicular to the ground and can be regarded as falling. Finally, we also terminate if the joint limits are violated.

Domain Randomization

For successful sim-to-real transfer, domain randomisation is crucial. All the measurements taken within the simulation can be regarded as clean and nearly ideal. In real scenarios, all measurements include some noise. Further production tolerances impose additional deviations from the simulation. Following the approaches [43] and [36], we adopt the randomisation of various environmental properties and add noise to the observation space. Privileged information, however, is noise-free. All approaches we train use the same randomisation parameters if not mentioned otherwise. The parameters can be found in the tab. 4.3, where $\mathcal{U}$ stands for uniformly sampled.

Disturbances

Additionally, to domain randomisation, disturbances should be imposed during training. So far, domain randomisation has already represented disturbances like noise or variations of actuator and sensor

Term	Distribution	Units	Operator
Environmental properties			
Payload Mass (trunk)	$\mathcal{U}(-1.0, 3.0)$	kg	additive
Hip Masses	$\mathcal{U}(-0.5, 0.5)$	kg	additive
Ground friction	$\mathcal{U}(0.3, 1.25)$	-	multiplicative
Gravity offset	$\mathcal{U}(-1.0, 1.0)$	m/s²	additive
Noise in the observation space			
Joint positions	$\mathcal{U}(-0.01, 0.01)$	rad	additive
Joint velocities	$\mathcal{U}(-1.5, 1.50)$	rad/s	additive
Local velocity	$\mathcal{U}(-0.1, 0.1)$	m/s	additive
Local ang. Velocity	$\mathcal{U}(-0.2, 0.2)$	rad/s	additive
Projected gravity	$\mathcal{U}(-0.05, 0.05)$	rad/s²	additive
System delay	$\mathcal{U}(0.0, 15.0)$	ms	additive

Table 4.3: Domain randomisation in simulation

properties. However, in real life, disturbances from the outside can happen. This is why we also impose a push during training. Because we greatly evaluate robustness with pushes, we implement the push differently than other approaches. Conventionally, a push is implemented in simulation by adding a randomised xy-velocity to the body trunk. [36] [43] This is not very representative of how pushes happen in the real world. Physically, a push happens by applying a force to a mass and thus accelerating it. The resulting velocity is then determined after the duration of the push. Inspired by the works [36] [43], which randomly added a velocity of up to $\pm 1 m/s$, we reversely define an impulse after the mass of the go2 robot in eq. 4.3. The mass of the robot is taken from the official website [54].

$$I = \int_{t=0}^{t=T} F dt = mv = 15kg \cdot 1.0 \frac{m}{s} = 15Ns \tag{4.3}$$

Since the velocity is randomised in training, we also randomise the impulse during training in a reasonable range (between 8 and 15 Ns), sample the force magnitude, and apply it for a duration calculated after eq. 4.4.

$$t = \frac{I}{F} \tag{4.4}$$

The range of the force is once again randomised within a reasonable range between 50N and the force, which is required to lift the mass $F_g = 150N$. This push is applied randomly in the xy-plane onto the robot trunk. The pushing strategy is then determined to have satisfactory walking and robust gait. A good balance is found in pushing the robot during training every six seconds, with a force magnitude of 50 - 150 N.

Rewards

The reward functions, similar to [53] [41] [36], consist of task rewards for tracking the command given and auxiliary rewards for stabilising the learning process. An overview is provided in table 4.4.

Reward	Equation (r_i)	Weight (w_i)
Lin. velocity tracking	$\exp\left(-4\left(\mathbf{v}_{xy}^{\text{cmd}}-\mathbf{v}_{xy}\right)^2\right)$	1.5
Ang. velocity tracking	$\exp\left(-4\left(\omega_{\text{yaw}}^{\text{cmd}}-\omega_{\text{yaw}}\right)^2\right)$	0.8
Linear velocity (z)	v_z^2	-2.0
Angular velocity (xy)	$\boldsymbol{\omega}_{xy}^2$	-0.05
Orientation	$\mathbf{g}_{xy}^2$	-5.0
Feet air time	$\sum_{f=0}^4 (t_{air,f}-0.1), \lvert\mathbf{v}_{cmd}\rvert > 0.1$	0.2
Joint accelerations	$\ddot{\boldsymbol{q}}^2$	-2.5×10^{-7}
Joint power	$\lvert\boldsymbol{\tau}\rvert\cdot\lvert\dot{\boldsymbol{q}}\rvert$	-2×10^{-5}
Power distribution	$\text{var}(\lvert\boldsymbol{\tau}\rvert\cdot\lvert\dot{\boldsymbol{q}}\rvert)$	-10^{-5}
Foot slip	$\sum_{f=0}^4 \lvert\lvert\mathbf{v}_{f,xy}\rvert\rvert^2, \text{if } z<0.01$	-0.1
Action rate	$(\mathbf{a}_t-\mathbf{a}_{t-1})^2$	-0.01
Foot clearance	$\sum_{f=1}^4 \left(\mathbf{p}_{f,z}^{des}-\mathbf{p}_{f,z}\right)^2 \lvert v_{f,xy}\rvert$	-0.1
Center of mass	$\left(\mathbf{p}_{com,xy}-\mathbf{p}_{xy}^{des}\right)^2, \ \mathbf{p}_{xy}^{des}=\frac{\sum_{f=1}^4 \mathbf{p}_{f,xy}}{4}$	-1.0
Joint tracking	$(\mathbf{q}_t^{target}-\mathbf{q}_{t+1})^2$	-0.1
Base height	$(h^{des}-h)^2$	-0.6
Hip	$\exp\left(-4*\sum_{k=1}^4 (q_{hip,k}-q_{hip,k}^{default})^2\right)$	0.05
Collisions	$n_{collisions}$	-10.0
Termination	$n_{termination}$	-10.0

Table 4.4: Reward Function Components, exp(·) and var(·) are exponential and variance operators, respectively. $()^{cmd}$ indicates the commanded values, respectively, whereas $()^{des}$, $()^{target}$ and $()^{default}$ indicate the desired, target and default values. x, y and z are defined on the robot's body frame, with x and z pointing forward and upward, respectively. $v_{xy}, v_{f,xy}, \omega_{yaw}, \mathbf{g}_{x,y}, \boldsymbol{\tau}, \boldsymbol{q}, \mathbf{a}, p_{f,xy/z,k}, p_{com}$ represent the body-, foot velocity in the xy plane, yaw rate, projected gravity component in the xy plane, joint torque, joint angles, the position of the kth foot and the centre of mass.

The total reward is then calculated for each time step and each environment after eq. 4.5. It is downscaled with the control time step to improve stability and convergence speed. [36] [55].

$$r_t = \sum r_i \cdot w_i \cdot dt \tag{4.5}$$

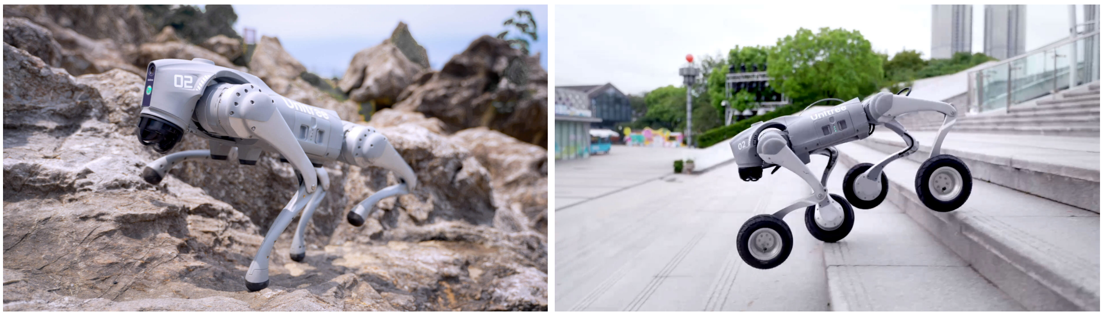
In addition to the already known rewards from literature, we introduce the "Center of mass" reward, with the incentive to train a more stable walking gait. It essentially computes the desired position for the centre of mass via the feet' positions. Typically, for a stable walk, the centre of mass should be in the middle of the support polygon spanned by the feet. This is why we calculate the desired position p_{xy}^{des}, according to the mean xy-components of the feet positions, and then penalise the squared error loss of the distance to it. The joint tracking reward is another substantial auxiliary reward introduced regarding variable stiffness. This reward term penalises the difference between the target joint position, given by the policy at time step t, and the actual position after applying the action to the actuator (t+1). It incentivises the policy to predict reasonable joint positions that are trackable. Otherwise, the policy could set the stiffness very low and predict non-interpretable joint targets [47].

4.3 Hardware implementation

Before deploying the trained agent onto the hardware, it must be tested in simulation and converted to fast executable code adapted to the used hardware. This chapter first introduces the hardware platform we use to validate our trained policies and our deployment pipeline. To encourage future research and development in this field, we publish our code[2] for deploying trained agents in simulation and on the hardware platform.

4.3.1 GO2 robot

The GO2 robot, developed and manufactured by Unitree Robotics, is a humanoid quadruped robot designed for logistics, inspection and surveillance applications. Generally, there are three different versions: AIR, PRO and EDU. With GO2-W, a wheeled version was introduced as well. The EDU version separates itself from the other versions by featuring an NVIDIA Jetson Orion, foot sensors, and longer battery life. [1]



(a) (b)

Figure 4.4: (a) Unitree Go2 robot: AIR, PRO and EDU. [15] (b) Unitree Go2-W [1]

We deploy our policies on the EDU version. The sensors include joint encodes, IMU, a gyroscope, a lidar, a camera and a foot force sensor. By default, it has controllers that enable it to run, crunch, jump, and climb small stairs. Connecting to the robot via LAN, Bluetooth or WIFI [54] is possible.

---

[2] `https://github.com/DarioRepoRuler/unitree_mujoco`

4.3.2 ROS architecture

The deployment architecture is built on top of the communication architecture provided by Unitree [56]. This architecture utilises ROS2 [57] to run the simulation or robot in one ROS node and the controlling script in another node. The high-level design of our controller architecture is shown in Fig. 4.5. The controller node listens to user commands and executes the controller. It is subscribed to the topics *lowstate* and publishes *lowcmd*. The *lowstate* includes the joint angles, joint velocities, angular velocity and the gravitational vector. The *lowcmd* carries the actuator data. This data consists of the target joint position and the p and d gains for the low-level controller in the hardware. This publish/-subscriber architecture allows us to use the node to control either hardware or simulation to control the robot in real-time. For testing purposes, another ROS node is implemented, which subscribes to the *lowcmd* and publishes *lowstate* as well as *sportsmodestate*. The *sportsmodestate* contains linear velocity, which can be easily measured in simulation. On the real robot, the *sportsmodestate* is not directly subscribed since it must be disabled to control the robot by *lowcmd*.

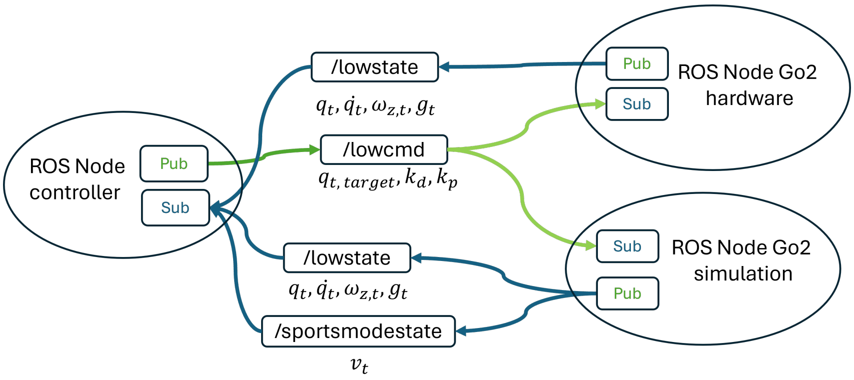


Figure 4.5: ROS architecture includes different nodes for controller, hardware and simulation.

This architecture enables the same controlling script for the simulation and the physical robot by switching the target ID. The controlling script runs two loops, representing the control loops in section 3.3. The low frequent loop, running at 50 Hz, executes the reinforcement learned controller and calculates the actions, given the subscribed commands and low/high states. The faster publisher loop at 200Hz represents the PD-controlled feedback loop, which continuously publishes low-level commands, given the actions predicted from the network and the subscribed data. Our implementation has successfully been deployed and tested with ROS2 foxy and ROS2 humble.

4.3.3 Deployment pipeline

Fig. 4.6 provides an overview of the deployment pipeline. First, the RL-agent is trained in the simulation using Mujoco-MJX. The software for training is explained in chapter 4. After training, the trained network is deployed in simulation using the keyboard's real-time user command inputs. This step involves the same communication structure, explained in Sec. 4.3.2, as in the deployment on hardware. Our experience suggests that the performance in this simulation closely matches the real-world performance. To address the time-critical execution requirements of the neural network(50 Hz on a HUAWEI Matebook), we optimised the deployment by transferring the trained weights to a torch script and implementing the publisher/subscriber architecture in C++.

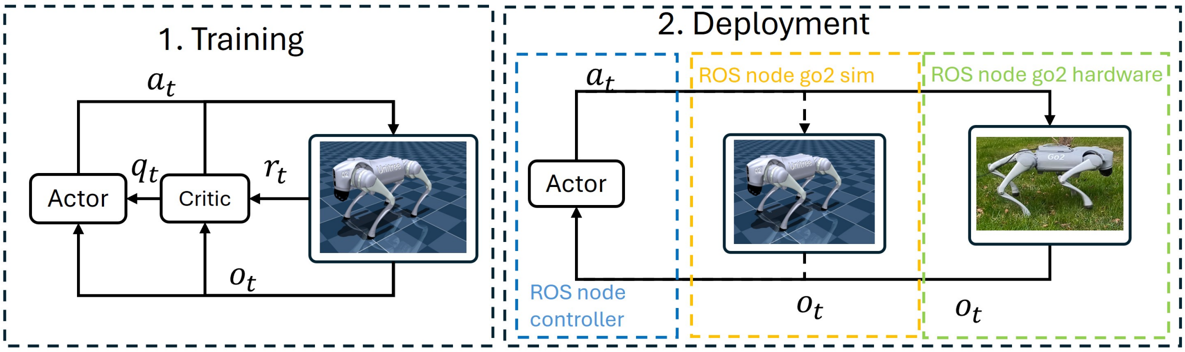

1. Training
a_t
q_t
r_t
Actor
Critic
o_t
2. Deployment
a_t
ROS node go2 sim
ROS node go2 hardware
Actor
ROS node controller
o_t
o_t

Figure 4.6: Deployment pipeline: 1. Training in Mujoco-MJX, 2.With the ROS nodes either in simulation or hardware.

Chapter 5

Analysis and Results

To demonstrate the efficiency of our method, we conduct experiments. This chapter explains the experiment settings and the outcomes. Specifically, we aim to evaluate the performance of task accuracy, the robustness of the learned walking gait, energy efficiency, and sim-to-real transfer. Most of these metrics are evaluated in simulation, except the sim-to-real experiments are conducted on the actual hardware.

5.1 Baseline

To evaluate the performance of our method, we compare it against a baseline. The baseline is formed by position control policies with low stiffness 20, referred to as P20, and high stiffness 50, referred to as P50. These baselines, trained under identical conditions, follow prior choices in [?,41] for lower stiffness and [16] for higher stiffness. While [16] used even higher gains, we observed instabilities in training and selected 50 as the upper stiffness. This selection of these two baseline stiffness values is motivated by their complementary characteristics. High stiffness is expected to perform well in specific tasks while suboptimal in others, whereas low stiffness is commonly shows the opposite trend. By contrast, the proposed approach aims to unify the benefits of both stiffness settings into a single end-to-end learnable controller capable of dynamically adapting to diverse task requirements.

5.2 Performance on walking and running

Different controller types' walking and running performance is evaluated by measuring tracking performance. This evaluation is commonly used, as in the work [6], which is also reasonable because this error is used to form the task reward. In our setting, we command each controller to follow a target velocity in eight different heading directions and measure the locomotion speed. The tracking error is

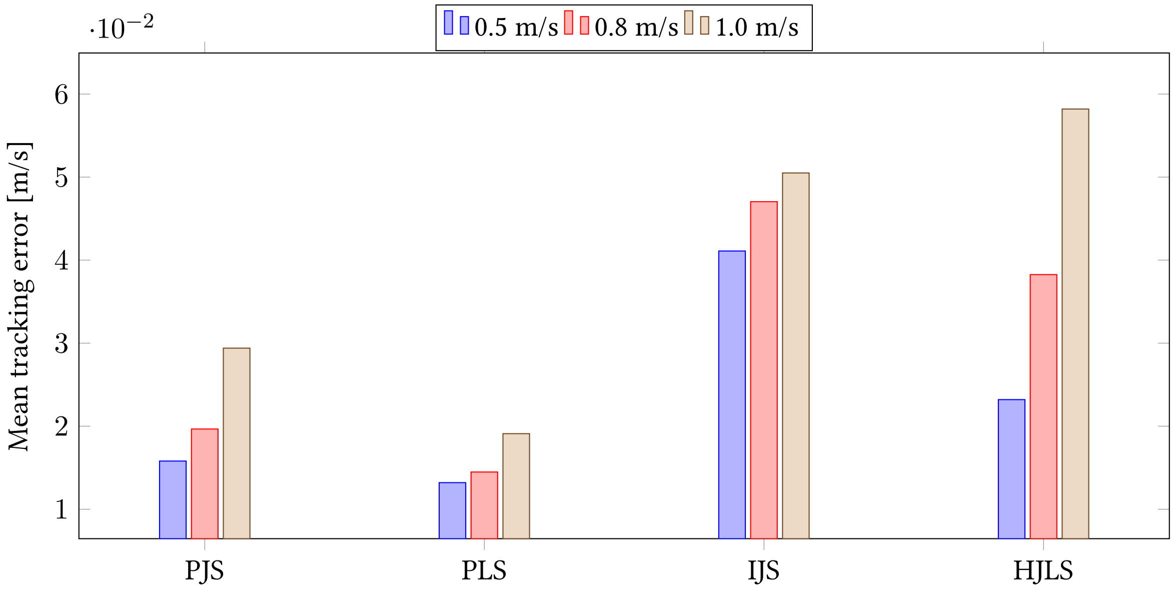


Figure 5.1: Speed tracking comparison across target directions for PJS, PLS, IJS and HJLS. Velocities are averaged over 1000 environments, each lasting 8 seconds. The different coloured bars show the tracking error for the different walking speeds.

calculated after the absolute distance between the target velocity and the measured velocity. The target speed is set statically, evaluating the different tasks of walking and running. For assessing the walking performance, we set the target velocity to $0.5\ m/s$, whereas for higher-speed walking, it is $0.8\ m/s$, and for running, it is set to $1.0\ m/s$. The target velocity is defined in eight discrete directions, with each direction forming an angle of 45° relative to the forward direction (0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°). Each of these directions is maintained for 8 seconds. We turn off domain randomisation to focus on velocity tracking in this experiment.

5.2.1 Comparison of variable stiffness methods

In section 4.1.1, we introduce different versions of stiffness control. Before comparing our method to the baseline, we first compare the performance of these methods relative to each other.

The results are shown in Fig. 5.1. As can be seen, all stiffness controls learn reasonable walking behaviour with low tracking errors. IJS shows bad performance in every speed category. HJLS, on the other hand, does perform better in low-speed tasks, but the tracking error quickly surges as the speed increases, even surpassing the tracking error of IJS at speeds of $1.0 m/s$. In contrast, PLS achieves the best performance overall tasked speeds, with tracking errors of just 1.32 cm/s, 1.448 cm/s and 1.91 cm/s for low, intermediate and high speeds. The tracking errors recorded can be observed in the tab. 5.1 for more details. Given these results, we compare PLS, which shows the lowest tracking error over all tasks, to the position-controlled baseline.

Control Paradigm	Mean tracking error [m/s]		
	0.5 m/s	0.8 m/s	1.0 m/s
PJS	0.0158	0.01966	0.0294
PLS	**0.0132**	**0.01448**	**0.0191**
IJS	0.0411	0.04704	0.0505
HJLS	0.0232	0.038254	0.0582
P20	0.049	0.0833	0.0834
P50	0.0215	0.02509	0.0253

Table 5.1: Mean absolute tracking error over different commanded walking speeds in eight directions. Task speeds are 0.5 and 1.0 m/s. The comparison shows that the tracking error is lowest for method PLS in all cases.

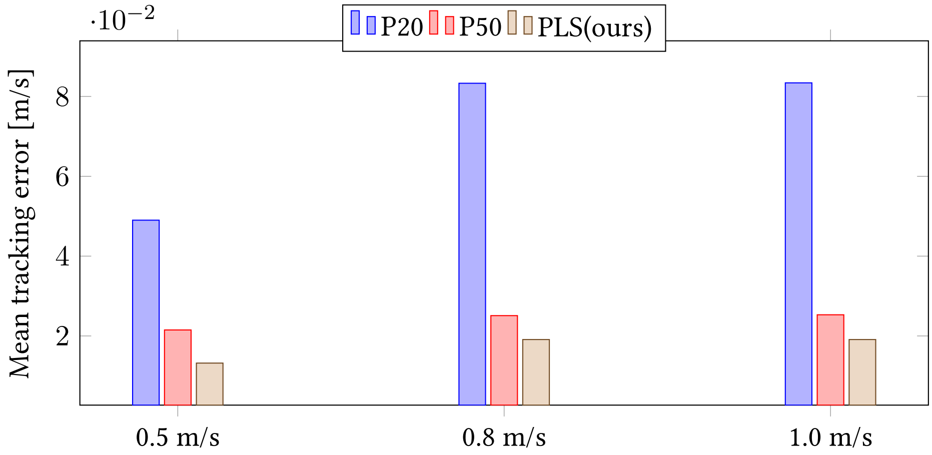


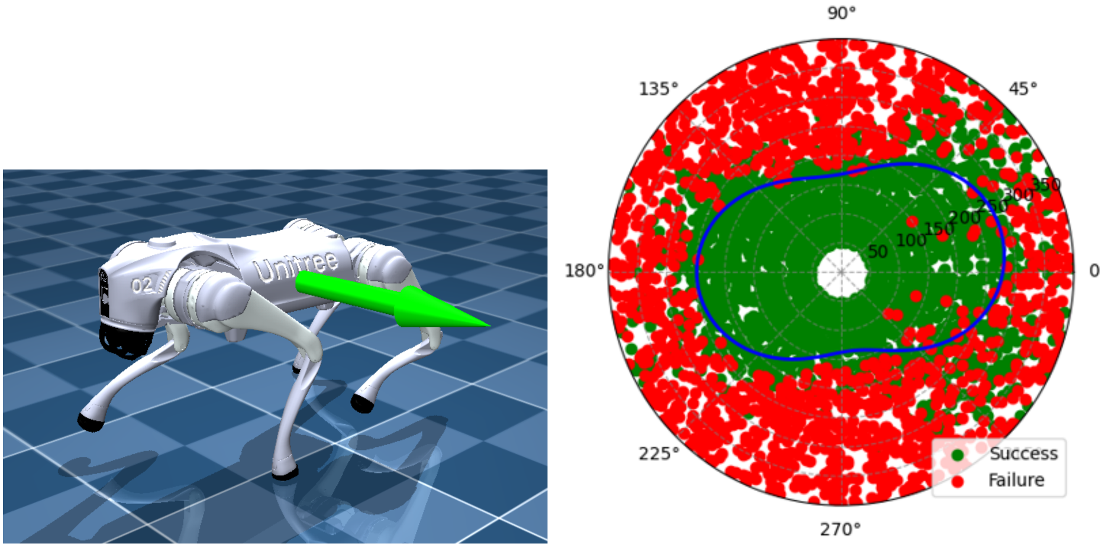
Figure 5.2: Speed tracking comparison across target directions for position control baselines with low (P20) and high stiffness (P50) and best performant variable stiffness approach PLS. Velocities are averaged over 1000 environments, each direction lasting 8 seconds. The different coloured bars show the tracking error for the different models.

5.2.2 Comparison to baseline

Given the high stiffness values, we expect the P50 policy to show minimal position error and achieve velocity tracking performance comparable to or even exceeding the variable stiffness approaches. The comparison of our baseline to the variable stiffness is depicted in figure 5.2. The bar chart shows that the P50 and the PLS policies perform better than those with low stiffness. Especially at high speeds, the gap between high and low stiffness becomes more evident as the tracking error shows a higher distance. As expected, with higher speeds, the tracking error also increases. Our approach PLS surpasses the performance of the P50 and P20 policies as the tracking error is lower than the high stiffness policy in all tasks.

Table 5.1 reveals the tracking error recorded and shows that for low speeds(0.5m/s), PLS is **38.6%** and at high speeds (1.0 m/s) **24.5%** more accurate than the best performing position-based policy we train under the same conditions.

5.3 Push recovery



(a) (b)

Figure 5.3: (a) Force kick visualisation for the policy under evaluation. The visualisation illustrates the push directions applied to the robot. (b) Evaluation of push recovery robustness for the position-based policy with low stiffness (P20). Successful recoveries are depicted as green dots, while failures are shown as red dots. The blue line represents the success boundary, determined using an SVM with an RBF kernel, indicating the maximum force magnitude from which the policy can recover. The directions are defined as follows: 0° corresponds to a push from the back of the robot onto the trunk, 180° from the front, 90° from the right, and 270° from the left.

Evaluating the robustness of a given locomotion policy involves exposing it to external disturbances and uncertainties. In this work, we expose it to pushes through applying forces onto the robot trunk in simulation while facing it with domain randomisation (explained in sec. 4.2.2). These disturbances must also be introduced during training to ensure the agent learns to adapt effectively. To ensure a fair comparison across different control paradigms, we uniformly apply the same force-pushing strategy to all policies/policies during training. The evaluation is conducted in the same simulation environment as the training.

Push recovery is measured under the following conditions:

- Walking speed: 0.3 m/s
- Force push magnitude: 50 - 300N
- Push duration: 0.1 sec

Pushes are applied to the robot's trunk from random directions in the xy-plane. A policy's ability to recover from a push is determined through a walking task. The robot is required to walk straight for 5 seconds. At a randomised point between the 2.5 and 3.5-second mark, it is exposed to a randomised push with the specified magnitude and duration. This randomisation of the time the robot gets pushed is done to prevent bias. For example, at one exact point in time, given a command, the pose of the robot might be the same and, therefore, show comparably more resilience in specific directions. A fall resulting in early termination is considered a failure. Successfully regaining balance and continuing to walk until the 5-second mark is regarded as a success. This experiment is repeated for each policy with 4,000 randomised pushes to evaluate robustness comprehensively. The results highlight the directions and force magnitudes the policy can and cannot recover. The experiment outcomes are visualised as polar plots, as depicted in Fig. 5.3b. To better visualise the capabilities of force push recovery, we utilise a Support Vector Machine(SVM) with a Radial Bias Function(RBF) kernel to classify the results into success and failure. The resulting decision boundary (depicted blue in fig. 5.3b) shows the maximum force push with specified confidence from which the policy can recover.

5.3.1 Comparison of variable stiffness methods

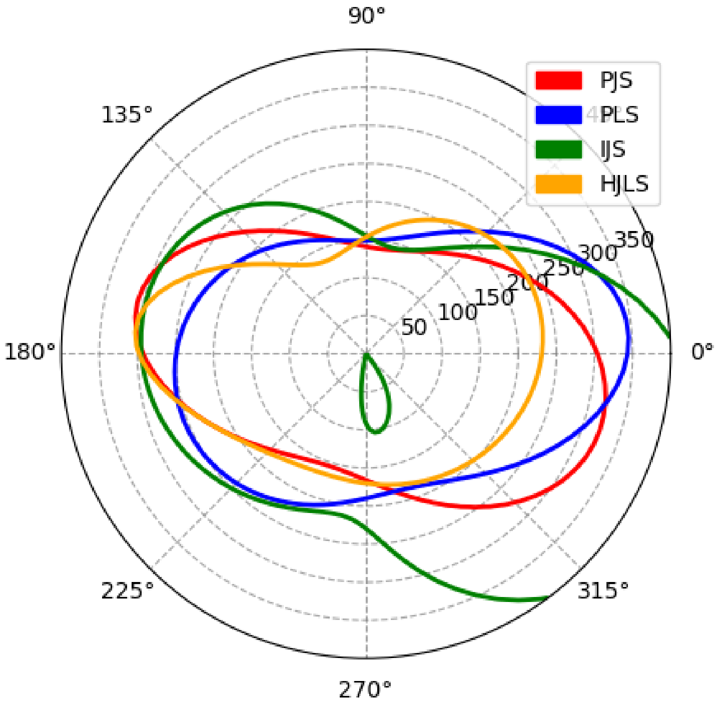


Figure 5.4: **Comparison of maximal recoverable force push.** The robot is pushed on the trunk, visualised in (a.). The polar scatter plot (b.) shows the experiment's outcome for specific forces (red=failure, green=success). An SVM with RBF kernel (confidence 90%) shows the maximum force magnitude from which the policy can recover. SVM parameters are C=0.6, γ=0.2.

The success boundary illustrated in Fig. 5.4 shows spherical and convex shapes for PJS, PLS and HJLS. IJS does not show this characteristic as failure regions appear within the success boundary. Generally, all policies show better recovery behaviour if pushed from the back or front (0° and 180°) and are more prone to pushes from the side. This also seems reasonable since we limited the hip range to achieve a more natural-looking gait.

Table 5.2 presents the success rates of various policies when subject to disturbances under defined constraints. During training, the agents are only exposed to pushes with magnitudes up to 150N. Despite this limitation, the evaluation demonstrates that the policies learn to adapt and recover even when pushed with higher forces up to 300N.

Within force pushes up to 150N, our controller PJS shows the highest recovery rate, with no falls recorded. Beyond this range, its performance quickly degrades, showing the lowest success rate overall for higher magnitudes. For pushes within a force magnitude, 250 and 300 N IJS shows the best performance. Notably, while PLS demonstrates inferior performance compared to the IJS at higher force magnitudes, it consistently delivers the second-highest success rates above 150N and closely follows the performance of the best-performing policy. This is consistent with its strong performance in previous experiments. These findings suggest that IJS exhibits superior force absorption capabilities. However, this policy performs poorly in other tests. For this reason, the baseline comparison in this experiment also considers the second-best-performing policy, PLS.

Control paradigm	Success Rate (%)				
	<100N	<150N	<200N	<250N	<300N
PJS	100.00	**100.00**	96.63	89.34	81.64
PLS	100.00	99.66	**97.66**	92.34	85.65
IJS	100.00	98.73	96.06	**93.53**	**90.44**
HJLS	99.83	99.32	97.03	91.58	83.93

Table 5.2: Measured force push success rates under domain randomisation. Comparison of the variable stiffness policies PJS, PLS, IJS and HJLS.

5.3.2 Comparison to baseline

It is anticipated that P20 will perform very well in this experiment as the low stiffness in the joints offers a passive way to absorb pushes. Generally, variable stiffness is expected to outperform the baselines as the joints can be relaxed and stiffened as needed, offering more reaction freedom. The success boundary in Fig. 5.5 reflects this expectation, as P20 performs well with PLS showing slightly better performance and P50 being surpassed. The results of the comparison between the variable stiffness policy (PLS) and the position-controlled policies are presented in Table 5.3. Remarkably, although not by a wide margin, P50 shows the highest success within the trained range, whereas P20 surpasses the performance at higher pushes. Compliant with the expectation, PLS outperforms the baselines when exposed to pushes with higher force magnitudes than 200N. The overall success rate is even 2.56% higher than the best position-controlled policy. These results demonstrate that PLS features the most robust behaviour among the tested policies, particularly under higher force magnitudes.

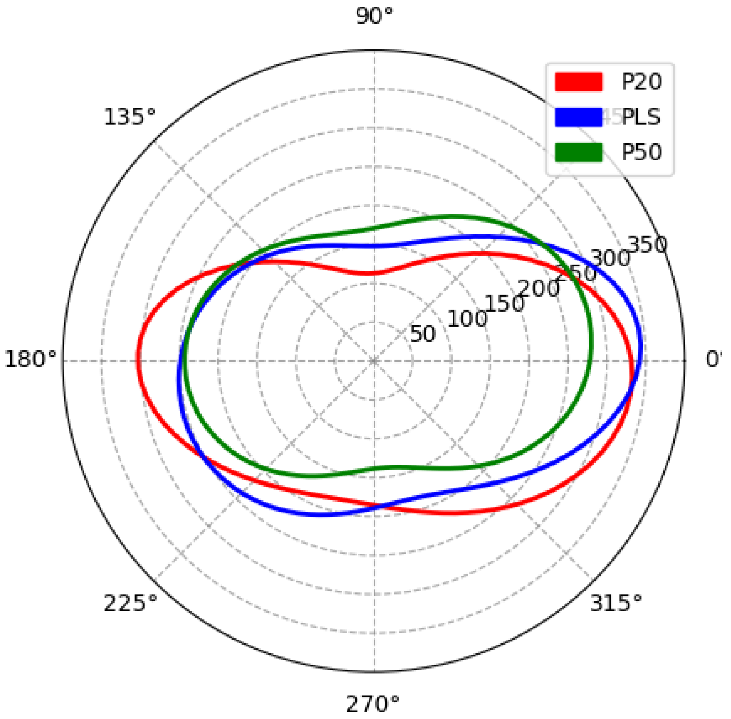


Figure 5.5: Comparison of success boundary(confidence 90%), determined using an SVM with an RBF kernel, indicating the maximum force magnitude from which the policy can recover. Compared policies are the baselines P20 and P50 and our method PLS. SVM parameters: C=0.9, γ=0.2

Control paradigm	Success Rate (%)				
	<100N	<150N	<200N	<250N	<300N
P20	99.48	99.07	94.98	89.93	83.08
P50	**100.00**	**99.75**	97.20	89.93	81.77
PLS	**100.00**	99.66	**97.66**	**92.34**	**85.65**

Table 5.3: Push recovery Success Rates under Domain Randomisation. Comparison of the position-controlled policy baselines P20 and P50 and our approach PLS, for force magnitudes within thresholds.

To further investigate the utilisation of stiffness in this scenario, Fig. 5.6 compares the predicted stiffnesses during a push to those of the baseline policies. The variable stiffness policy successfully compensates for the illustrated push, whereas both position-based policies result in falls. Our approach leverages its stiffness adaptation capabilities when subjected to a push, applied from the frontal left direction. Specifically, the legs opposite the push—namely, the front and rear right legs—are stiffened, while the stiffness of the other legs is relaxed. The stiffness values in these legs exceed those of the high-stiffness position-based policy. This adaptive strategy is consistently observed when the variable stiffness policy successfully compensates for the push.

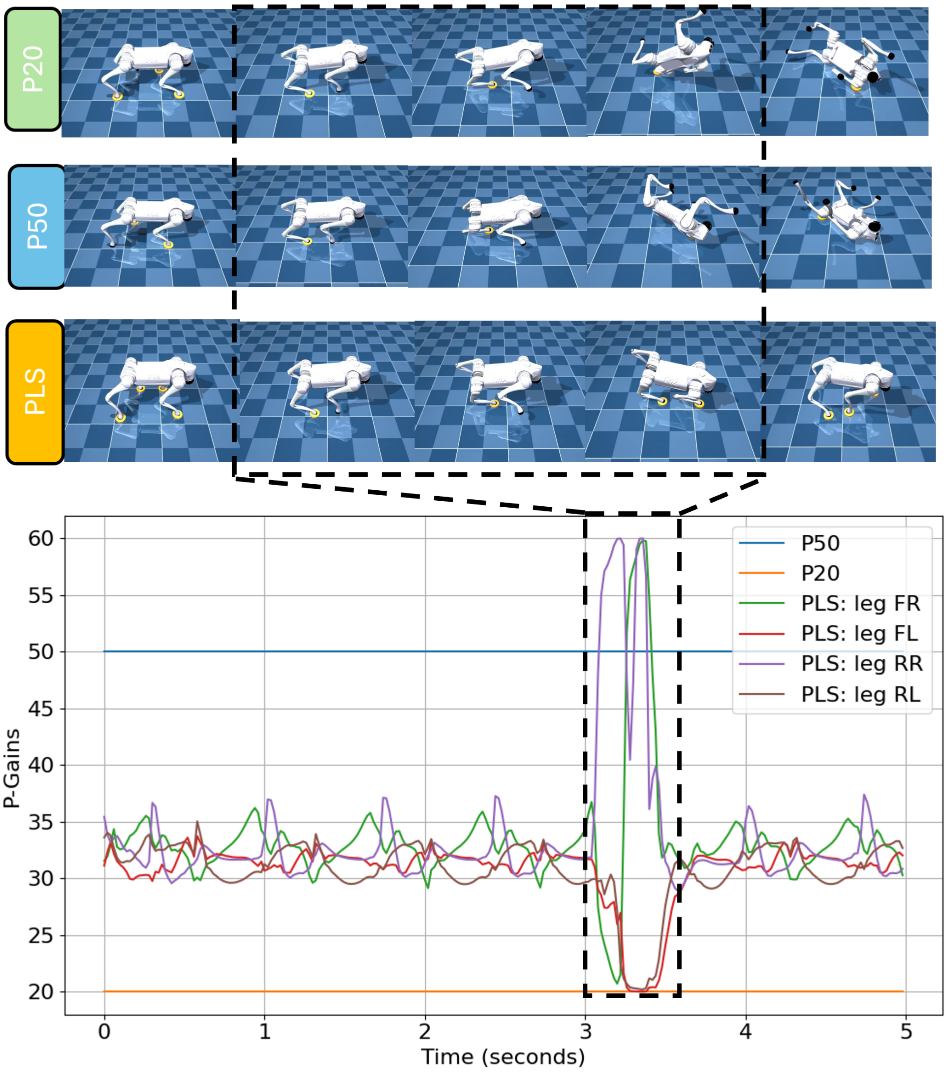


Figure 5.6: Policies exposed to a left push in simulation and P-gains over time. Push conditions: $t_{push} = 3s$, $t_{duration} = 0.1s$, $F = 210N$, $\theta = 225$ accounting for the point in time, duration, magnitude and direction at which the push is applied. The force is applied from the frontal left direction. The demonstrated policies are the baselines P20, P50 and our approach PLS

5.4 Power efficiency

Variable stiffness can enhance energy efficiency during walking by adapting stiffness to specific situations. Energy efficiency is typically assessed by measuring power consumption during a walking task. As discussed in sec. 5.2, the commanded velocity rarely matches the measured velocity precisely due to differences in tracking performance. This discrepancy suggests that power consumption alone is insufficient as an evaluation metric. Therefore, the policies are also evaluated using the Cost of Transport (CoT) metric [6] [58], which incorporates achieved velocity into the assessment, providing a more comprehensive measure of power efficiency. The CoT is calculated using eq. 5.1, where M_{total} accounts for the total mass of the robot, τ denotes the measured torques, $\dot{q}$ the joint velocities, g the gravitational acceleration and v for the measured velocity.

$$CoT = \frac{E}{M_{total} g d} = \frac{P}{M_{total} g v} = \frac{\tau \dot{q}}{M_{total} g v} \tag{5.1}$$

The power alongside the CoT is evaluated on the walking tasks used to evaluate the tracking performance in sec. 5.2, without domain randomisation.

5.4.1 Comparison of variable stiffness methods

Figure 5.7 illustrates each model's Cost of Transport (CoT) at three walking speeds: 0.5 m/s, 0.8 m/s, and 1.0 m/s. The exact values are listed in table 5.4. A lower CoT indicates greater energy efficiency for a given walking speed. The values are calculated as the mean CoT overall heading directions for the experiment explained in sec. 5.2. As expected, CoT increases with higher speeds, as running consumes more energy than walking. However, a noticeable trend emerges from the results: Model HJLS consistently demonstrates the most energy-efficient behaviour across all speeds, achieving the lowest CoT at every tested velocity. Although the CoT for HJLS increases with higher speeds, it maintains a clear advantage over the other models. In contrast, PJS shows strong energy efficiency at lower speeds but becomes less efficient at higher speeds. At higher speeds of 0.8 m/s, PJS is surpassed by PLS as the second most energy-efficient policy. The IJS policy shows less energy efficiency as its CoT values are considerably higher than other methods.

5.4.2 Comparison to baseline

The identified most energy-efficient policy, HJLS, is compared to the baselines P20 and P50. It is anticipated that the P20 policy will be the most energy efficient as it has the lowest p-gains compared to the other approaches. HJLS can adapt its stiffness between 20 and 60. Further, our approaches are

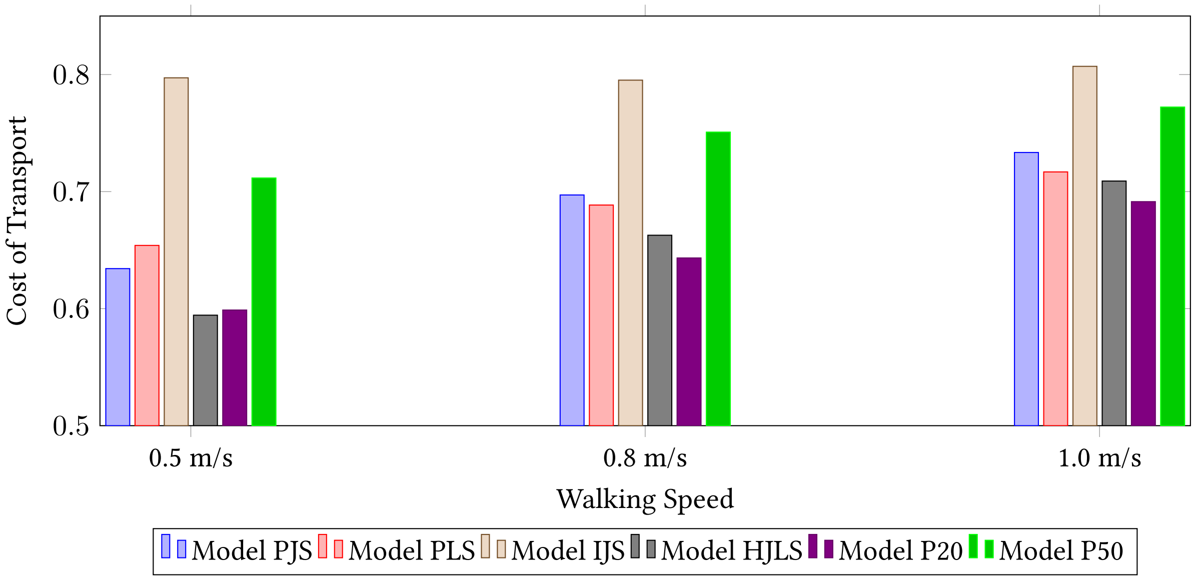


Figure 5.7: Bar chart for Cost of Transport (CoT) across different walking speeds for six models.

Control Paradigm	**Cost of Transport (CoT)**		
	0.5 m/s	**0.8 m/s**	**1.0 m/s**
PJS	0.6342	0.6971	0.7334
PLS	0.654	0.6885	0.7168
IJS	0.7972	0.7952	0.807
HJLS	0.5944	0.6627	0.709
P20	0.5988	0.6433	0.6914
P50	0.7116	0.7508	0.7722

Table 5.4: Cost of Transport at Different Walking Speeds

anticipated to be more energy efficient than P50 but less than P20. HJLS demonstrates a CoT that is 2–3% higher than P20, which is close regarding the range in which the stiffness can be adjusted. Interestingly, at lower speeds (0.5 m/s), HJLS surpasses P20, with a CoT 0.01 lower, indicating that at these speeds, the variable stiffness approach is more energy-efficient than the fixed stiffness policy. However, the high stiffness policy, on the other hand, shows significantly worse performance. For P50, the CoT values remain consistently above 0.7. At the same time, P20 stays below this threshold, highlighting the inefficiency of the high stiffness policy in comparison to both variable stiffness and low stiffness baseline approaches.

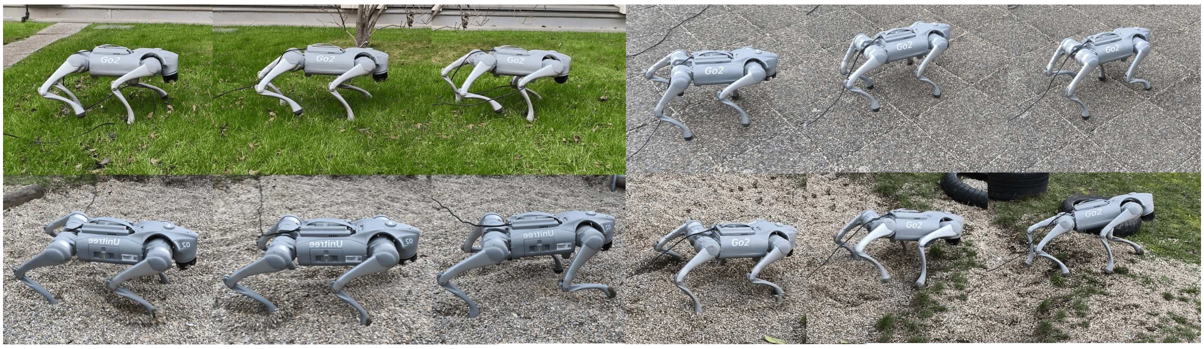


Figure 5.8: Despite being trained solely on a flat floor in simulation, we showcase the robot's ability with our variable-stiffness RL policy to walk robustly on diverse outdoor terrains, such as grass, stones, and sand.

5.5 Sim-to-real transfer

During hardware deployment, we evaluate the robustness of the learned locomotion policy by adding a payload and walking over diverse terrains. As experiments have shown in the previous sections, the most capable model is the PLS policy. The low stiffness policy P20 and P50 are baseline references in these experiments.

5.5.1 Payload experiment

To show our approach's stiffness adaption capabilities, we expose the policies with a payload of $5kg$. During training, the policies are only encountered with an additional payload of $3kg$. Figure 5.9 shows the stiffness of the front two legs for this experiment predicted by our policy PLS. It can be observed that the stiffness generally increases when the payload is added. If the payload is removed, the stiffness decreases accordingly. Compared to P20, our policy maintains a proper gait and higher foot lifting, whereas the baseline demonstrates a narrow stance and a much lower trunk. P50 shows similar performance.

5.5.2 Walking performance

We demonstrate the robustness of our policy PLS by walking on outside terrain. This is depicted in Fig. 5.8. Even though our policies are only trained on flat floor our policy demonstrated robust walking. It managed to walk on grass, mud, sand and sidewalks. Inclinations should be incredibly challenging since such are also not observed during training. Fig. 5.10 demonstrates our policies can also handle inclinations. Despite these competitive results, the differences between our policy and the baseline policies show in their posture, as shown in Fig. 5.9. While P20 and P50 show a lower base position, our policy demonstrates a higher, more stable position.

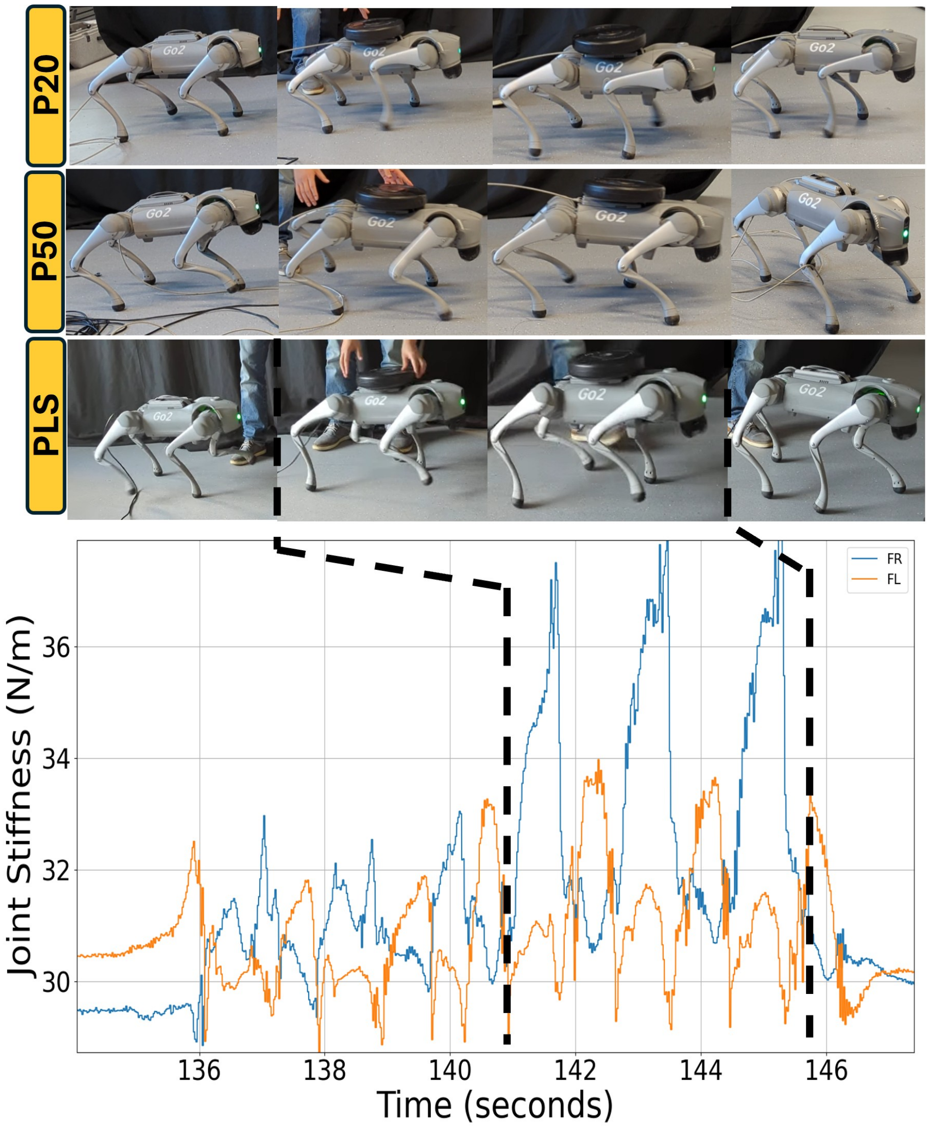


Figure 5.9: **Payload experiment:** Adding a 5kg mass during walking to the PLS policy.

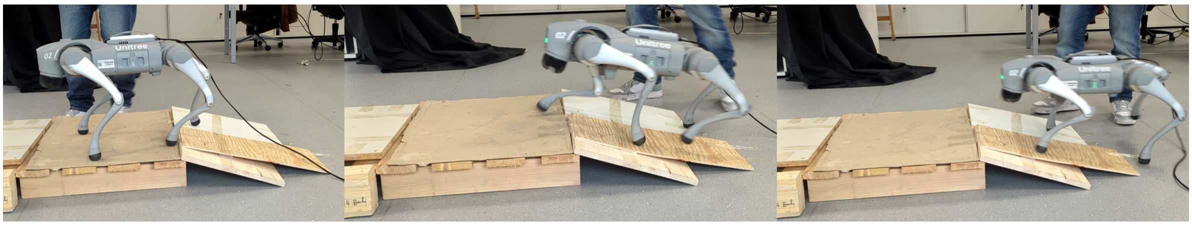


Figure 5.10: **Unexpected inclination:** The policy is exposed to never-before-seen inclination in the form of a ramp.

5.6 Discussion

In this thesis, we investigate model-free reinforcement learned quadruped locomotion. We extended the action space for the quadruped locomotion agent to predict joint stiffness along joint positions and introduced a software framework to train and deploy. We introduce stiffness groupings to minimise the additional action space and ease training. With these contributions, we aim to improve the current model of free locomotion and answer the research questions raised in chapter 1.

1.) Performance on walking/running: Velocity tracking error over different walking speeds and directions is used to evaluate whether our controller architecture improves performance over tasks such as walking and running. Velocity tracking error over different walking speeds and directions is used to answer the first research question. A comparison with the baselines demonstrates that our approach, which can adjust the stiffnesses per leg (PLS), outperforms both position-controlled with high and low stiffness in every speed category. Essentially, PLS shows a 23.75 to 38.6 % lower tracking error than the best position-based policy. Regarding the other introduced methods, we see similar performance. At the same time, the policy which can predict joint stiffnesses for each joint individually (IJS) does show worse performance than the position-based. This can be attributed to the extensive action space that has to be explored, which needs either more training or more tuning of parameters to work. The improved performance for PLS might be attributed to the stiffness being increased at the stance phase and relaxing the joints during the swing phase (shown in fig. 5.9). This way, disturbances from the swinging legs do not influence the base. The findings highlight the advantage of variable stiffness in improving velocity tracking accuracy and underline the benefit of grouping the joints.

2.) Force Push Recovery The performance on robustness is measured via successful recoveries over multiple push experiments. The results show that our approaches PLS, IJS and HJLS outperform position-controlled policies overall, with success rates of 83.93 to 90.44%, while the best performing position-based policy shows 83.08% for pushes up to 300N. Further, we could show that our policies use their stiffness capabilities to adjust to pushes. Generally, our approach IJS performs best in this experiment. However, the policy shows inferior performance in all the other experiments. This shortcoming can once again be attributed to the larger action space. In contrast to IJS, our other methods, PLS and HJLS, utilise stiffness to balance and recover and show satisfactory gait. Low stiffness position controlled policies (P20) also perform reasonably well in this experiment ($< 300N$). Generally, it is anticipated that the variable stiffness policies would outperform the position-based policies by a more considerable margin than recorded; the results show improvement but not at a pace as expected. This could be addressed in future research by providing temporal information to the policy. Temporal information could enable the policy to react to stumbling more efficiently. However, this has to be incorporated with an adaption or estimation module, as our trials revealed that naively providing this temporal information to the policy does not improve the method overall. This raises the question of how robust a variable stiffness policy with temporally encoded information could be. Overall, the results show the superior performance of our variable stiffness method over position-controlled baselines and, therefore, answer the research question of whether the introduced methods improve stability in terms of push recovery.

3.) Energy Efficiency In our locomotion setting, we use the cost of transport (CoT) to evaluate if our methods perform more energy efficiently than position-controlled policies. We record this metric for different walking speeds and directions. Generally, we expect the low stiffness baseline P20 to outperform policies with higher stiffnesses, such as the high stiffness baseline P50. Our approaches can adjust their joint stiffness between 20 and 60, so this should also be no exception. Surprisingly, the results show that our approach is even more energy efficient for low speeds than P20. The other investigated speeds meet the expectations and show that our policy PLS outperforms P50 by a large margin, whereas P20 is the most energy efficient. To make this policy more energy efficient, we could alter the range, where the stiffness can be predicted, or change the weighting of the rewards. Choosing the range too low resulted in unsatisfactory gaits for us, so we decided on the mentioned range. To conclude these findings and answer the research question of whether variable stiffness is more energy efficient than position-controlled, the results show that variable stiffness has the potential to be more energy efficient but with a lower stiffness range than comparable position-based policies.

4.) Sim-to-Real Transfer Our approach demonstrates a successful sim-to-real transfer as it can walk on diverse terrains such as grass, mud, and sidewalks, even though it is only trained on flat floors. With a payload experiment, we show the capabilities of our approach to adjust the stiffness. Overall, it maintained a superior walking gait compared to position-based policies, with low stiffness gains. All policies showed successful walking on the real hardware with minor differences. An unexpected shortcoming is the lifting of a leg during a standstill. This undesired behaviour sometimes leads the policy to do unnecessarily balance movements. Overcompensation or unmodelled noise could be the cause of this unexpected behaviour. To answer the research question, if the proposed methodology is superior in sim-to-real transfer, the results demonstrate a successful sim-to-real transfer but not necessarily an improvement in transfer. The proposed method reacts superior to low stiffness baselines if encountered with payloads by stiffening its joints and maintaining a proper gait.

Overall, with these results, our proposed controller requires less tuning of the stiffnesses and combines the advantages of position-based policies with low or high stiffness. Whereas in tracking, P50 performs well but badly in power efficiency, and P20 performs well in power efficiency and push recovery but badly in tracking, our controller always performs considerably well and even outperforms both in velocity tracking.

5.7 Future work

Our work demonstrates the benefits of variable stiffness. This method could be applied in future works to learn even more diverse tasks like crouching, hopping stair traversal and imitating motions. As this approach can adjust the stiffness, this method might also know these tasks for multiple robot types and combine them into one policy.

5.8 Conclusion

In this thesis, we studied an alternative approach to learning locomotion on a quadruped robot, which uses joint positions alongside stiffnesses as the action space in a reinforcement learning paradigm. The underlying software framework for understanding and deploying the agents in simulation and hardware has been introduced. Simulation and real-world experiments are conducted to investigate performance on walking and running, as well as push recovery, energy efficiency and sim-to-real transfer. Regarding walking and running performance, the PLS policy outperformed position-based policies by achieving up to 38.6% more accurate velocity tracking. This underscores the benefits of variable stiffness, mainly when stiffness is grouped by leg. In robustness tests involving force push recovery, our approaches demonstrated greater resilience than position-based baselines.

Energy efficiency analysis revealed that among our approaches, HJLS performed best. This indicates the advantage of adjusting joint group stiffness alongside leg stiffness over other groupings. Compared to P20 and P50, our most energy-efficient policy performs better than P50 but less than P20 regarding the CoT. At low speeds (0.5 m/s), HJLS even shows lower CoT than P20.

Finally, our PLS policy demonstrated successful sim-to-real transfer alongside the baselines in deployment on hardware. While showing adoption capabilities to payloads and higher resilience in maintaining a good walking gait, PLS demonstrated the potential of variable stiffness control to generalise effectively to real-world scenarios.

These findings show that variable stiffness control combines the advantages of low and high stiffness policies without additional stiffness tuning. This could be shown as high stiffness policies showed advantages in velocity tracking but did poorly in energy efficiency. Low stiffness policies perform badly in tracking but are good in energy efficiency and push recovery. Our policy performs well in all tasks and even surpasses the performance in velocity tracking. While challenges remain and limitations are faced, such as the shortcomings in energy efficiency, we believe our results and work show the potential of using variable stiffness in policies. This thesis contributes to the growing body of reinforcement learned locomotion in robotics research, and by addressing the challenges mentioned and building upon the foundation laid here, future research can unlock new frontiers in locomotion.

Bibliography

[1] U. Robotics, "Unitree go2," https://shop.unitree.com/products/unitree-go2, accessed: 2025-01-07.

[2] Boston Dynamics, "About boston dynamics," 2025, accessed: 2025-01-16. [Online]. Available: https://bostondynamics.com/about/

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 2018.

[4] T. Corbères, T. Flayols, P.-A. Léziart, R. Budhiraja, P. Souères, G. Saurel, and N. Mansard, "Comparison of predictive controllers for locomotion and balance recovery of quadruped robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[5] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, 2019.

[6] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, 2020.

[7] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv*, 2020.

[8] C. Yang, K. Yuan, Q. Zhu, W. Yu, and Z. Li, "Multi-expert learning of adaptive legged locomotion," *Science Robotics*, 2020.

[9] Z. Fu, A. Kumar, A. Agarwal, H. Qi, J. Malik, and D. Pathak, "Coupling vision and proprioception for navigation of legged robots," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[10] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science robotics*, 2022.

[11] S. Bohez, S. Tunyasuvunakool, P. Brakel, F. Sadeghi, L. Hasenclever, Y. Tassa, E. Parisotto, J. Humplik, T. Haarnoja, R. Hafner *et al.*, "Imitate and repurpose: Learning reusable robot movement skills from human and animal behaviors," *arXiv*, 2022.

[12] R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang, "Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers," *arXiv*, 2021.

[13] H. Zhang, L. He, and D. Wang, "Deep reinforcement learning for real-world quadrupedal locomotion: a comprehensive review," *Intelligence and Robotics*, 2022.

[14] 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

[15] Chironix, "Unitree go2 - base platforms," https://www.chironix.com/base-platforms/unitree-go2, accessed: 2025-01-07.

[16] G. Feng, H. Zhang, Z. Li, X. B. Peng, B. Basireddy, L. Yue, Z. Song, L. Yang, Y. Liu, K. Sreenath *et al.*, "Genloco: Generalized locomotion controllers for quadrupedal robots," in *Conference on Robot Learning.* PMLR, 2023.

[17] S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath, "Learning torque control for quadrupedal locomotion," in *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids).* IEEE, 2023.

[18] D. Kim, G. Berseth, M. Schwartz, and J. Park, "Torque-based deep reinforcement learning for task-and-robot agnostic learning on bipedal robots using sim-to-real transfer," *IEEE Robotics and Automation Letters*, 2023.

[19] L. Yu, H. Zhao, S. Qin, G. Jin, and Y. Chen, "A compact variable stiffness actuator for agile legged locomotion," *IEEE/ASME Transactions on Mechatronics*, 2024.

[20] J. Fu, Z. Yu, H. Lin, L. Zheng, and D. Gan, "A novel variable stiffness compliant robotic link based on discrete variable stiffness units for safe human–robot interaction," *Journal of Mechanisms and Robotics*, 2024.

[21] X. Zhao, Y. Wu, Y. You, A. Laurenzi, and N. Tsagarakis, "Variable stiffness locomotion with guaranteed stability for quadruped robots traversing uneven terrains," *Frontiers in Robotics and AI*, 2022.

[22] K. P. Murphy, *Machine learning : a probabilistic perspective.* Cambridge, Mass. [u.a.]: MIT Press, 2013.

[23] W. B. Powell, *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions.* Wiley, 2022.

[24] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994.

[25] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, 1949.

[26] B. Belousov, A. H., P. Klink, S. Parisi, and J. Peters, *Reinforcement Learning Algorithms: Analysis and Applications*, ser. Studies in Computational Intelligence, 2021.

[27] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *CoRR*, 1999.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, 2013.

[29] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 2004.

[30] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, 2017.

[31] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, 1999.

[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," *arXiv*, 2015.

[33] V. Mnih, "Asynchronous methods for deep reinforcement learning," *arXiv*, 2016.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, 2017.

[35] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, 2015.

[36] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, 2022.

[37] M. H. Raibert, *Legged robots that balance.* MIT press, 1986.

[38] D. Kang, J. Cheng, M. Zamora, F. Zargarbashi, and S. Coros, "Rl+ model-based control: Using on-demand optimal control to learn versatile legged locomotion," *IEEE Robotics and Automation Letters*, 2023.

[39] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, 2019.

[40] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[41] I. M. A. Nahrendra, B. Yu, and H. Myung, "Dreamwaq: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[42] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion," *IEEE Robotics and Automation Letters*, 2022.

[43] G. B. Margolis and P. Agrawal, "Walk these ways: Tuning robot control for generalization with multiplicity of behavior," in *Proceedings of The 6th Conference on Robot Learning*. PMLR, 2023.

[44] H. Shi, Q. Zhu, L. Han, W. Chi, T. Li, and M. Q.-H. Meng, "Terrain-aware quadrupedal locomotion via reinforcement learning," *arXiv*, 2023.

[45] Z. Xie, X. Da, M. van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation*, 2021.

[46] L. Smith, I. Kostrikov, and S. Levine, "Demonstrating a walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning," *Robotics: Science and Systems (RSS) Demo*, 2023.

[47] M. Bogdanovic, M. Khadiv, and L. Righetti, "Learning variable impedance control for contact sensitive tasks," *IEEE Robotics and Automation Letters*, 2020.

[48] V. M. et al., "Isaac gym: High performance gpu-based physics simulation for robot learning," *CoRR*.

[49] 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*, 2023.

[50] 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*, 2012.

[51] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning." in *Robotics: Science and Systems*, K. Hauser, D. A. Shell, and S. Huang, Eds., 2022.

[52] A. Kumar, Z. Fu, D. Pathak, and J. Malik, “Rma: Rapid motor adaptation for legged robots,” *arXiv*, 2021.

[53] G. Deepming, “Mujoco-mjx tutorial,” 2024, [Last accessed on 13 Sep 2024]. [Online]. Available: https://colab.research.google.com/github/google-deepmind/mujoco/blob/main/mjx/tutorial.ipynb#scrollTo=MpkYHwCqk7W-

[54] I. Unitree Robotics, “Unitree basic service interface,” 2024, [Last accessed on 12 Sep 2024]. [Online]. Available: https://support.unitree.com/home/en/developer/Basic_services

[55] J. Liu, “Reward rescaling — nanoppo 0.1 documentation,” 2023, accessed: 2024-12-05. [Online]. Available: https://nanoppo.readthedocs.io/en/stable/reward_rescaling.html

[56] I. Unitree Robotics, “Unitree mujoco simulator,” 2024, [Last accessed on 11 Sep 2024]. [Online]. Available: https://github.com/unitreerobotics/unitree_mujoco/

[57] O. robotics, “Robot operating system,” 2024, [Last accessed on 12 Sep 2024]. [Online]. Available: https://www.ros.org/

[58] H. Q. Vu, X. Yu, F. Iida, and R. Pfeifer, “Improving energy efficiency of hopping locomotion by using a variable stiffness actuator,” *IEEE/ASME transactions on mechatronics*, 2015.

Appendix A

Appendix

A.1 Notation

- $\mathcal{D}$ dataset
- s, r, a general variabels for state, reward and action
- S_t states at time step t
- R_t reward at time step t
- A_t action at time step t
- $p(s', r|s, a)$ probability for receiving reward r and evolving into s' given the state s and action a
- $p(s'|s, a)$ probability for evolving into s' given the state s and action a
- r(s,a) reward for given state s and action a
- G_t cumulative reward at time step t
- γ discount factor
- $v_\pi(s, a)$ general value function for policy π at general state s
- $q_\pi(s)$ general action value function for policy π at general state s
- π^*, v_π^*, q_π^* optimal policy, optimal value, optimal q value
- Q_k q-value at evaluation step k
- δ TD error
- α, β learning rate
- L_k loss or objective in general
- L^{PG} policy loss
- $\theta, \mathbf{w}$ network parameters
- $\hat{A}$ advantage
- L^{PPO} loss for PPO algorithm

- L^{CLIP} clipped TRPO loss
- L_t^{VF} value loss at timestep t
- $S[\pi_\theta]$
- λ variance reduction parameter