POLITECNICO DI TORINO

Master's Degree
in Computer Engineering

M.Sc. Thesis

Constrained Reinforcement Learning for Safe Quadruped Robot Locomotion

Supervisors
Prof. Raffaello Camoriano
Prof. Davide Tateo Prof. Jan Peters

Candidate
Paolo Magliano

Academic Year 2024-2025

Abstract

In recent years, robotic locomotion has made considerable advances, allowing legged robots to walk successfully across a wide range of terrains. Many of the most successful approaches utilize Reinforcement Learning (RL) as a framework that allows robots to learn useful behaviors to achieve a goal through a trial-and-error approach, specifying only the desired objective.

However, while RL has demonstrated impressive capabilities, it also presents some limitations. One of the main concerns is the lack of safety during both the learning process and the implementation of the resulting policy. In real-world scenarios, unsafe behavior can damage the robot, harm the surrounding area, or even become a risk to humans. So, ensuring safety is a fundamental requirement for deploying RL-based strategies on physical robots.

Safe Reinforcement Learning addresses this issue by introducing constraints that encourage safer and more controlled behavior, not only after training but ideally during the learning process while maintaining the advantages of traditional RL.

Furthermore, most current research focuses on training in simulation and then transferring the policy to the physical robot using Sim-to-Real techniques. However, reaching a sufficient level of safety during learning opens up the possibility of training directly on real robots, avoiding the potential mismatches caused by simulation-to-reality transfer.

The thesis explores state-of-the-art Safe RL methods for quadruped locomotion as a way to enforce safety constraints into the learning process. Specifically, it examines the ATACOM method, which transforms a constrained RL problem into an unconstrained one where the action space is mapped onto a manifold to limit possible actions according to the desired safety requirements. These constraints are derived from the robot model using the kinematic and geometric properties provided by the Lie theory.

The study investigates the effectiveness of several safety restrictions and analyzes their impact on task learning and locomotion performance. The goal is to determine which constraints support learning, identify the boundary conditions that still allow successful walking, and observe how the walking style adapts under different configurations. The considered constraints are joint position limits, minimum and maximum height of the robot base, and desired feet position and orientation.

The thesis compares the results of the constrained learning approach enabled by ATACOM with state-of-the-art unconstrained locomotion policies, evaluating trade-offs in terms of performance, safety, and robustness. It also extends the ATACOM method by improving its effectiveness in the task by redesigning the way the action space is morphed to ensure safety and by correcting the unsafe actions with an improved error correction approach.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Robotics is a discipline that arises from the intersection of multiple fields that work together to design, develop, and operate autonomous machines capable of performing actions by processing information from sensors that capture the surrounding environment.

In recent decades, technological advances have significantly enhanced the capabilities of robotic systems. Today, robots play a crucial role in a wide range of applications, from industrial automation and medical surgery to autonomous driving and the exploration of unstructured environments.

At the core of robotics lies motion planning and control, which enable robots to perform tasks with high precision and efficiency, assisting humans in a variety of activities. Modern research in robotics aims to develop robust and efficient behaviors in complex and dynamic environments, tackling increasingly challenging tasks that would otherwise require considerable human effort. This complexity also introduces uncertainty, making it difficult to rely only on model-based approaches, which often depend on simplifying assumptions to remain computationally tractable.

Reinforcement Learning. Reinforcement Learning (RL) offers a promising alternative. By allowing robots to learn from interaction, RL enables the development of behaviors in environments too complex to be modeled analytically. In particular, robotic locomotion has benefited significantly from this approach, achieving significant results in recent years.

The RL paradigm exploits an iterative procedure that enables an *agent* to learn a behavioral strategy, known as *policy*, from direct experience in an unknown *environment*, guided by a feedback scalar signal called *reward*. This reward evaluates the agent's actions with the goal of accomplishing a given task. Such a structure allows the task to be defined only through the reward function, rather than requiring a precise specification of each joint movement needed to complete it.

Despite notable improvements over previous techniques, RL presents some intrinsic limitations due to its trial-and-error nature. Standard RL methods are inherently stochastic and may exhibit behaviors that deviate from the intended ones, especially during the learning phase. This can lead to serious damage to the robot hardware or its surroundings, including potential risks to nearby humans. Safe Reinforcement Learning aims to provide safety guarantees during both training and deployment by enforcing a set of constraints that the agent must respect.

Acting on the Tangent Space of the Constraint Manifold. Several approaches have been developed to incorporate safety into the RL paradigm, typically by introducing soft or hard constraints. This thesis focuses on the Safe RL method known as *Acting on the Tangent Space of the Constraint Manifold* (ATACOM) [1], which builds a safe action space by computing the tangent space relative to the constraint manifold, in this way, every action that belongs to it is guaranteed to be safe. This particular design allows a constrained RL problem to be reformulated as an unconstrained one by introducing a safe controller between the agent and the environment. Consequently, the policy remains safe not only during deployment but also throughout training, as every action executed by the ATACOM controller belongs to the safe region.

ATACOM has been shown to perform particularly well with robotic manipulators across various tasks [1], [2], demonstrating the ability to safely learn directly in the real world without the need for prior simulation-based training. This thesis aims to investigate ATACOM's potential by evaluating its performance in a different domain, that is, quadruped robot locomotion.

Although in this thesis we consider the model-free RL setting, ATACOM relies on the robot model to estimate constraint violations and prevent unsafe movements. Therefore, the safety guarantees provided by the method hold only under the assumption that the model accurately represents the system's dynamics. A large discrepancy between the model and the actual robot may result in safety violations.

Contribution. The goal of this thesis is to understand whether the introduction of constraints in the quadruped locomotion task could improve the efficiency of the learning process by reducing the samples needed to solve the task and to guarantee safety by respecting constraints that could be designed to help learning or to impose different walking style.

The complexity of this new task revealed minor incompatibilities between ATACOM and state-of-the-art locomotion methods, which have been addressed by adapting both the method and the experimental environment. As a result, this thesis contributes by:

- Updating the state-of-the-art setup to enable ATACOM to allow the introduction of safety constraints by modifying the low-level controller;
- Extending the ATACOM method to overcome design limitations in the context of quadruped locomotion, improving its applicability to other domains;
- Designing and studying locomotion-specific constraints that leverage the robot model to restrict joint movements, foot positions, and base height, and providing an extensive empirical analysis of their performance in simulation.

Overview. The structure of the thesis is briefly outlined to provide a general overview.

- **Chapter 2**: presents state-of-the-art RL methods for robot locomotion, both unconstrained and incorporating safety requirements.
- **Chapter 3**: introduces the background concepts and algorithms supporting the main contributions of the thesis. It presents the foundations of robot modeling and RL, including the notation and mathematical formulations used throughout the work.
- **Chapter 4**: describes the key contributions of this thesis, detailing the proposed low-level controller, the safety constraints design, and two extensions introduced in ATACOM.
- **Chapter 5**: presents ablations for the proposed extensions and evaluates the performance of constrained locomotion policies compared to the unconstrained baseline, in terms of both learning efficiency and safety guarantees.
- **Chapter 6**: provides a final summary, highlighting the main findings and proposing possible directions for future research.

Chapter 2

Related works

2.1 Robot Locomotion

Quadruped robot locomotion is a challenging task that has been studied for decades and has inspired a wide range of approaches. In recent years, one approach has gained particular popularity due to its successful results and relative ease of implementation: Reinforcement Learning (RL). RL, especially *model-free* RL, has proven to be a strong competitor to traditional methods based on analytical models of kinematics and dynamics for control and planning.

Modern RL algorithms are capable of achieving quadruped locomotion over diverse terrains with obstacles such as steps, gaps, inclined surfaces, slippery patches, and even rough, rocky ground. A strong baseline was established in the work by Rudin et al. [3], which demonstrated the potential of RL applied to diverse quadruped robotic platforms on complex terrains. The study highlighted key factors such as parallel training, which significantly accelerates learning, and Sim-to-Real techniques like domain randomization, which enable the direct deployment of policies trained in simulation into real hardware.

The success of RL in locomotion extends beyond quadrupeds to bipedal robots. For instance, Rodriguez et al. [4] is one of the first to show the capabilities of a RL agent applied to real-world bipedal robot locomotion. They effectively achieve stable walking in multiple directions in simulation and real world. In a separate study, Siekmann et al. [5] show that a bipedal agent could navigate stair-like terrains using only proprioceptive sensors.

Locomotion extensions Building on these foundations, several works have further expanded the capabilities of RL for robot locomotion. Margolis et al. [6] achieved highly dynamic behaviors that resulted in very high locomotion speeds.

Other studies [7]–[9] explored a wide range of agile behaviors, including jumping, sprinting, crawling, leaping, traversing tilted ramps, and even performing handstands, demonstrating the robustness and versatility of RL-based locomotion policies.

Other researchers, such as Miki et al. [10] and Aractingi et al. [11], improve upon the previous work by Rudin et al. [3] by incorporating information from additional sensors, combining proprioceptive and exteroceptive perception. The studies [12]–[14] aim to enhance the adaptability of the agent by dynamically adjusting its walking style in real time based on the surrounding terrain.

Smith et al. [15] achieve excellent results in real-world locomotion by training directly in the physical environment, using a sample-efficient RL algorithm that collects observations on the fly. Jenelten et al. [16] improve the learning process by combining a model-based planner with a reinforcement learning algorithm, where the planner rolls out reference motions tracked by the deep policy. Feng et al. [17] develop general-purpose locomotion controllers deployable across multiple quadruped platforms without requiring separate policies for each robot.

Implemented techniques The Proximal Policy Optimization (PPO) algorithm by Schulman et al. [18] has proven to be one of the most effective in this domain, being employed in many of the most relevant works [3]–[5], [7], [8], [10]–[12]. It offers strong policy convergence through its trust region approach and performs well in parallelized training environments. In contrast, Smith et al. [15] use an extended version of the Soft Actor-Critic (SAC) algorithm by Haarnoja et al. [19], which accelerates learning by reducing the total number of samples needed, an essential feature for real-world training.

To achieve good results on challenging terrains, a curriculum-based approach is often employed [3], [5], [11], [12]. The advantage of this method is its ability to automatically increase the level of difficulty of the terrain when the agent has mastered the current level, enabling the policy to adapt gradually to increasingly complex scenarios. In contrast, training the agent directly on the most difficult scenarios has been shown to be considerably harder than using a curriculum.

All the discussed works evaluate policy performance not only in simulation but also in the real world, adopting Sim-to-Real techniques to transfer the simulated policy into physical systems. Since simulators approximate and model real-world dynamics and properties to keep computational costs feasible, a gap between simulation and reality inevitably arises. To bridge this gap, the most effective technique is *Domain Randomization* [20], which involves randomizing simulation parameters, such as masses and friction coefficients, to make the agent generalize across a range of conditions. This improves policy robustness and enables successful real-world deployment, as shown in [3], [5], [7], [10]–[13]. An alternative approach is proposed by Campanaro et al. [21], which replaces Dynamics Randomization with *Extended*

Random Force Injection (ERFI). ERFI perturbs system dynamics during training and produces policies that are more robust to variations in system dynamics.

2.2 Safe RL methods

Ensuring safety in Reinforcement Learning is not a straightforward task, and many approaches have been developed over the years. Traditional methods enforce constraints by adding a penalty term to the reward or by using Lagrange multipliers to trade off reward and constraint violation. However, the reward shaping process can be highly sensitive to small changes in reward design and often requires extensive tuning of weight coefficients [22].

A key distinction among Safe RL techniques lies in the type of constraints they impose. These can be categorized as *hard constraints* [1], [23]–[26]; or *soft constraints* [8], [24]–[31]. Soft constraints promote safe behavior without guaranteeing it, whereas hard ones enforce strict limits that must never be violated.

Due to the diversity of approaches, a wide variety of strategies have emerged to ensure safety. Some methods, such as those by Liu et al. [1], Ames et al. [23], and Yang et al. [29], exploit system models to analyze and predict safety violations. Others, including Yang et al. [29] and He et al. [31], rely on two policies, one trained with an unconstrained method for normal behavior, and a second designed to recover from states near constraint violations. The final policy switches between them when entering a critical state space region.

Other studies [24], [25], [27], [28], extend the RL algorithm itself by incorporating safety constraints directly into the optimization process, often using a Lagrangian multiplier approach.

Chane et al. [26] employ a simple yet effective technique that terminates episodes based on proximity to constraint violations and the associated risk. Smith et al. [30] propose using a secondary neural network to evaluate the range of permissible actions, which progressively becomes more permissive to allow wider exploration.

ATACOM offers the advantage of being built on top of any unconstrained RL method that best suits the task, as long as an accurate system model is available. Another benefit, when compared to the approach by Ames et al. [23], is the reduced computational cost: ATACOM only requires solving a least squares problem for a linear system, rather than a full quadratic programming problem.

Chapter 3

Background

This Chapter introduces the essential background concepts that support and contextualize the analyses presented in the later chapters. Section 3.1 provides the essential information needed to understand basic robot descriptions and motion. Next, Section 3.2 presents the core concepts of the Reinforcement Learning (RL) approach, outlining its key features across various methods, leading up to the one adopted in this thesis.

3.1 Robot modeling and control

This Section introduces the basic concepts involved when working with robotic systems. It covers the mathematical representation of the robot's structure and dynamics, as well as the control techniques used to drive its motion. These concepts are essential for formulating the constraints and designing the low-level controllers employed throughout this work. Being able to formally describe the system's evolution is also crucial for building reliable simulated environments that closely replicate real-world behavior. Furthermore, having an accurate model of the system is fundamental to respect the assumptions on which ATACOM relies.

3.1.1 Lie theory

A *Lie group* is a mathematical structure that combines the characteristics of the group and the concept of a smooth manifold [32]. This means that its elements satisfy the group axioms (closure, associativity, identity, and invertibility), and at the same time, the set of elements forms a continuous, differentiable surface with no sharp corners or edges (smooth manifold).

A manifold $\mathcal{M}$ is a space that has a curved structure such as the surface of a sphere. The smoothness of the manifold allows us to define a tangent space at each point, which is a local linear approximation of the manifold and enables the application of linear properties.

In the specific case of a Lie group, the manifold has a uniform shape that causes all tangent spaces to be similar due to the group symmetry. The tangent space at the identity element $\mathcal{E}$ of the group plays a special role and is called the *Lie algebra* $T_{\mathcal{E}}\mathcal{M}$ of the Lie group. The Lie algebra captures the local, linear structure of the group and allows us to study complex, nonlinear behavior using linear tools. The figure 3.1 shows the graphical description of elements in these spaces.

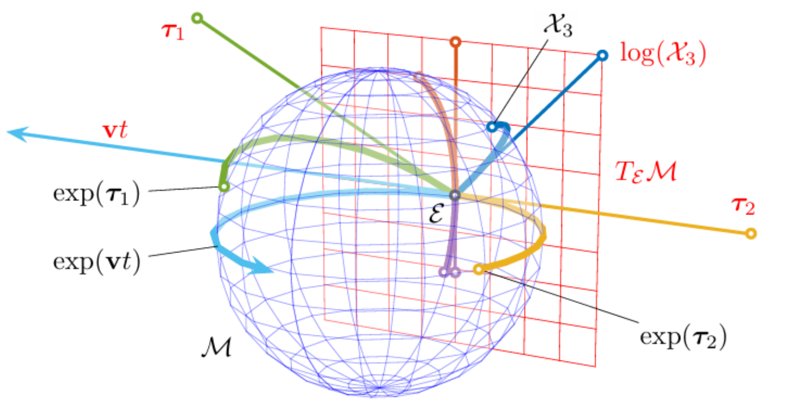


Figure 3.1: This image visualizes the connection between a Lie group and its corresponding Lie algebra. The red plane represents the Lie algebra $T_{\mathcal{E}}\mathcal{M}$, which is the tangent space at the identity $\mathcal{E}$ of the Lie group's manifold $\mathcal{M}$, illustrated as a blue sphere. A straight line $\mathbf{v}t$ in the Lie algebra maps to a curved path $\mathrm{Exp}(\mathbf{v}t)$ on the manifold through the exponential map, tracing the geodesic from the identity. Each group element can be associated with a point in the Lie algebra, enabling complex, nonlinear operations on the manifold to be performed within the simpler, linear structure of the Lie algebra. Image taken from [32].

The key operations that link a Lie group with its Lie algebra are the *exponential map* and the *logarithm map*. They make it possible to obtain an element of the Lie algebra from the corresponding one in the Lie group, and vice versa. The exponential map $\mathrm{Exp}(\tau)$ precisely transfers elements of the Lie algebra to the group. Intuitively, $\mathrm{Exp}(\tau)$ wraps the tangent element around the manifold. The inverse map is the logarithm map $\mathrm{Log}(\mathcal{X})$, i.e., the unwrapping operation.

$$\mathrm{Exp}(\tau) : \mathbb{R}^m \to \mathcal{M} \qquad \mathrm{Log}(\mathcal{X}) : \mathcal{M} \to \mathbb{R}^m.$$

The Jacobian defined on Lie groups extends the concept of derivatives to functions on smooth manifolds. They are compatible with the classical chain rule, which makes them especially powerful for computing complex derivatives through composition. For a multivariate function $f : \mathbb{R}^m \to \mathbb{R}^n$, the Jacobian matrix is defined as the $n \times m$ matrix stacking all partial derivatives.

$$\mathbf{J} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \triangleq \lim_{\mathbf{h}\to 0} \frac{f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})}{\mathbf{h}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{n\times m}.$$

Based on the standard definition, the Jacobian of a function $f : \mathcal{M} \to \mathcal{N}$ acting on manifolds is

$$\frac{{}^{\mathcal{X}}Df(\mathcal{X})}{D\mathcal{X}} \triangleq \lim_{\tau\to 0} \frac{\mathrm{Log}\left(f(\mathcal{X})^{-1} \circ f(\mathcal{X} \circ \mathrm{Exp}(\tau))\right)}{\tau}.$$

3.1.2 Position and orientation

The ability to work with robots relies on a fundamental understanding of how to represent the position and orientation of a three-dimensional object, which is essential to describe how each part of the robot affects the others. This Section introduces the notation and operations used to represent the rotation and translation of a rigid body, as required to model robot joints.

An object in three-dimensional space is fully described by its position and orientation, defined relative to the world reference frame $\mathcal{F}_W$ using the object reference frame $\mathcal{F}_o$. The position is expressed as a vector $\mathbf{t}$ of Cartesian coordinates from $\mathcal{F}_w$ to $\mathcal{F}_o$.

$${}^{w}\mathbf{t}_o = \begin{bmatrix} {}^{w}t_{o,x} \\ {}^{w}t_{o,y} \\ {}^{w}t_{o,z} \end{bmatrix}.$$

The orientation is represented by expressing the axes of the rotated frame $\mathcal{F}_o$ in the coordinates of the world frame $\mathcal{F}_w$. The rotation matrix groups the components for each axis.

$${}^{w}\mathbf{R}_o = \begin{bmatrix} {}^{w}x_{o,x} & {}^{w}y_{o,x} & {}^{w}z_{o,x} \\ {}^{w}x_{o,y} & {}^{w}y_{o,y} & {}^{w}z_{o,y} \\ {}^{w}x_{o,z} & {}^{w}y_{o,z} & {}^{w}z_{o,z} \end{bmatrix},$$

where ${}^{w}x_{o,y}$ is the y component (w.r.t $\mathcal{F}_w$) of the unit vector associated with the x axis of object reference frame $\mathcal{F}_o$.

Rotation matrices belong to the *Special Orthogonal* Lie group $SO(3)$, which means they satisfy the properties of a Lie group. The group acts on 3-vectors by rotation, and its tangent space elements can be identified with rotation vectors in $\mathbb{R}^3$. The space $\mathbb{R}^3$ is where rotation rate vectors $\mathbf{u}w$, and angle-axis representations $\mathbf{u}\theta$ are defined. The exponential and logarithm maps of a rotation matrix can be formulated as follows:

$$\mathrm{Exp}(\theta\mathbf{u}) \triangleq \mathbf{I} + \sin\theta\, [\mathbf{u}]_\times + (1-\cos\theta)\, [\mathbf{u}]_\times^2 \in \mathbb{R}^{3\times3}$$

$$\theta\mathbf{u} = \mathrm{Log}(\mathbf{R}) \triangleq \frac{\theta(\mathbf{R}-\mathbf{R}^\top)^\vee}{2\sin\theta} \in \mathbb{R}^3,$$

$$\text{with } \theta = \cos^{-1}\left(\frac{\mathrm{trace}(\mathbf{R})-1}{2}\right),$$

where $[\mathbf{u}]_\times$ denotes the skew-symmetric matrix associated with the vector $\mathbf{u}$

$$[\mathbf{u}]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}.$$

The Jacobian has a closed-form expression derived from the definition.

$$\mathbf{J}(\boldsymbol{\theta}) = \mathbf{I} - \frac{1-\cos\theta}{\theta^2}[\boldsymbol{\theta}]_\times + \frac{\theta-\sin\theta}{\theta^3}[\boldsymbol{\theta}]_\times^2.$$

3.1.3 Homogeneous transformation

By combining translation and rotation, the coordinates of a point $\mathbf{p}$ in frame $\mathcal{F}_0$ can be expressed in terms of its coordinates in frame $\mathcal{F}_1$.

$${}^0\mathbf{p} = {}^0\mathbf{t}_1 + {}^0\mathbf{R}_1\, {}^1\mathbf{p}.$$

The homogeneous transformations provide a way to handle rotations and translations in 3D space using matrix operations, which simplifies the mathematical treatment of rigid body motion.

$$\begin{bmatrix} {}^0\mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{R}_1 & {}^0\mathbf{t}_1 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} {}^1\mathbf{p} \\ 1 \end{bmatrix}$$

$${}^0\tilde{\mathbf{p}} = {}^0\mathbf{T}_1\, {}^1\tilde{\mathbf{p}},$$

where

$${}^0\mathbf{T}_1 = \begin{bmatrix} {}^0\mathbf{R}_1 & {}^0\mathbf{t}_1 \\ \mathbf{0} & 1 \end{bmatrix}.$$

This transformation belongs to the *Special Euclidean* Lie group $SE(3)$, in which each element is represented by a transformation matrix $\mathbf{T}$. The corresponding Lie algebra is expressed as a 6D vector $\boldsymbol{\tau} \in \mathbb{R}^6$, which combines translational and rotational components.

$$\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} \in \mathbb{R}^6.$$

The relative exponential and logarithm maps are described in the Appendix A.1.1.

3.1.4 Robot model

A robot is composed of rigid bodies, called *links*, connected by *joints* to form a kinematic chain. The most common types of joints are *revolute*, which allow relative rotation, and *prismatic*, which allow relative linear motion along an axis.

The *configuration* of a robot is a complete specification of the position of all its parts. The set of all feasible configurations defines the *configuration space* ($\mathcal{C}$). In most cases, the configuration can be fully described by the values of the joint variables. Specifically, for a revolute joint, the variable is a joint angle θ_i; for a prismatic joint, it is a joint offset d_i.

A robot with n joints has a configuration vector

$$\mathbf{q} = [q_1, q_2, \dots, q_n]^\top \in \mathbb{R}^n,$$

where $q_i = \theta_i$ for revolute joints and $q_i = d_i$ for prismatic joints. Since the links are assumed to be rigid and the base of the robot is fixed, knowledge of $\mathbf{q}$ determines the position and orientation of any point on the robot.

The number of *degrees of freedom* (DoF) corresponds to the dimension of the configuration space, and typically equals the number of joints. For comparison, a rigid body in three-dimensional space has six DoF: three translational and three rotational.

While the configuration describes the robot's geometry at a specific instant, it does not account for motion or dynamics. The *state* of a robot extends the configuration by incorporating dynamic quantities such as joint velocities. A common representation of the state is a vector

$$\mathbf{s} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \in \mathbb{R}^{2n},$$

which includes both position and velocity components. The set of all such states forms the *state space*, which, together with a dynamic model and control inputs, allows prediction of future behavior.

3.1.5 Kinematics

Kinematics studies the geometric and temporal aspects of a robot's motion without looking at the forces or torques that cause it. The goal is to establish the relationship between the motion of each joint and the overall motion of the robot.

Forward Kinematics solves the problem of determining the position and orientation of the end effector given the joint variables $\mathbf{q}$. To perform kinematic analysis, a coordinate frame is rigidly attached to each link. In particular, the frame $\mathcal{F}_i$ is assigned to link i. Let A_i be the homogeneous transformation matrix that expresses the pose of frame i with respect to frame $i-1$. This transformation changes with the robot's configuration and depends only on the joint variable q_i, under the assumption that joints are either revolute or prismatic.

The pose of the end effector, placed at the end of the n-th link and expressed in the inertial frame $\mathcal{F}_0$, is given by combining the individual transformations:

$$^{0}T_n = A_1(q_1)\cdots A_n(q_n),$$

where each homogeneous transformation matrix A_i has the form:

$$A_i = \begin{bmatrix} ^{i-1}R_i & ^{i-1}t_i \\ \mathbf{0} & 1 \end{bmatrix}.$$

The derivative of the forward kinematics equations with respect to time provides the **Jacobian matrix**, which describes a linear relationship between the joint velocities and the linear and angular velocity of the end effector. The Jacobian $\mathbf{J}(\mathbf{q})$ is a $6 \times n$ matrix that captures the instantaneous mapping between the joint space and the task space velocities. The joint space is an n-dimensional vector of joint velocities, while the task space velocity is a 6-dimensional vector representing linear and angular velocity of the end effector.

Inverse Kinematics solves the inverse problem, that is, computing the joint variables $\mathbf{q}$ given the description of the desired position and orientation of the end effector. This typically requires solving a system of nonlinear equations and may not admit a unique or closed-form solution.

3.1.6 Dynamics

While kinematic equations describe the motion of a robot without considering their causes, dynamic equations explicitly describe the connection between motion and the forces and torques that produce it. A proper dynamic model is essential for accurate simulation, animation, and control of robotic systems.

In most robotic applications, the dynamic equation is expressed in the following standard form:

$$\mathbf{B}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}. \tag{3.1}$$

This model includes the following terms:

- $\mathbf{B}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the *inertia matrix*, a symmetric and positive definite matrix that captures the mass and geometry of the robot in joint space.
- $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \in \mathbb{R}^n$are the vectors of generalized joint positions, velocities, and accelerations, respectively.
- $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} \in \mathbb{R}^n$ is the vector of *Coriolis and centrifugal forces.*
- $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ is the vector of gravitational forces acting on the robot.
- $\boldsymbol{\tau} \in \mathbb{R}^n$ is the vector of generalized forces or torques applied at the joints.

The explicit dynamic model of a robot can be derived using different methods, where *Lagrange* and *Newton-Euler* approaches are the most common.

The Lagrange method is based on the kinetic and potential energy of the system, and applies the Euler–Lagrange equations to get the equations of motion in closed form. In contrast, the Newton–Euler method relies on the direct application of Newton's laws to each link of the robot. It is a numerical and recursive approach that propagates forces and torques along the kinematic chain.

3.1.7 Robot control

Motion control refers to the problem of determining how a robot moves. For the purpose of the thesis, we describe only the goal of reaching a desired fixed configuration, expressed in joint space, and maintaining that configuration. Formally, the goal is to achieve $\mathbf{q} = \mathbf{q}_d$ with $\dot{\mathbf{q}} = \mathbf{0}$.

In this work, the focus is on a class of dynamical systems known as *control-affine systems.* These are systems where the control input appears linearly and the dynamics is described by a differential equation of the form:

$$\dot{\mathbf{s}} = f(\mathbf{s}) + G(\mathbf{s})\,\mathbf{u}_s,$$

where:

- $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n$ is the system state,
- $\mathbf{u}_s \in \mathcal{U} \subset \mathbb{R}^m$ is the m-dimensional control input,
- $f : \mathcal{S} \to \mathbb{R}^n$ represents the drift dynamics (i.e., the system behavior in the absence of control),

- $G : \mathcal{S} \to \mathbb{R}^{n\times m}$ is the input matrix determining how the control input affects the system.

Both f and G are assumed to be locally Lipschitz continuous functions.

To regulate the system, the *proportional–integral–derivative controller* (PID) controller automatically compares the desired target value $\mathbf{q}_d$ to the actual state value $\mathbf{q}$, reducing the distance between them through three components: proportional, integral, and derivative.

The method defines the value of the control input $\mathbf{u}(t)$ to address the error $(\mathbf{q}_d - \mathbf{q})$:

$$\mathbf{u}(t) = K_P(\mathbf{q}_d - \mathbf{q}) + K_I \int_0^t (\mathbf{q}_d - \mathbf{q}(\tau))\, d\tau - K_D\, \dot{\mathbf{q}}(t).$$

The proportional component $K_P(\mathbf{q}_d - \mathbf{q})$ outputs a corrective term that is directly proportional to the current error, providing an immediate response that quickly reduces the gap from the desired setpoint. The integral component $K_I \int_0^t (\mathbf{q}_d - \mathbf{q}(\tau))\, d\tau$ is the sum of the error over time. It eliminates persistent steady-state offsets by making a corrective action based on the integral term, minimizing long-term error. The derivative component $K_D\, \dot{\mathbf{q}}(t)$ evaluates the rate of change of the error to predict future states, with the goal of reducing overshooting and improving stability, especially in systems subject to rapid movements.

3.2 Reinforcement Learning

3.2.1 Basics

The goal of Reinforcement Learning (RL) is to learn a behavior strategy (policy) that maximizes the performance metric reward in an unknown, stochastic environment. Unlike supervised learning, RL does not provide explicit target actions; instead, the agent must infer effective behavior only from a delayed reward signal. Moreover, data arrive sequentially (not i.i.d.), and each action influences future observations and rewards.

The interaction between the agent and the environment in RL is modeled as a *Markov Decision Process* (MDP), which assumes full observability that means the current state fully describe all relevant past information. An MDP is defined by the tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \iota, \gamma \rangle,$$

where

- $\mathcal{S}$ is the state space;
- $\mathcal{A}$ is the action space;

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, producing $\mathcal{R}(s, a, s')$ when action a in state s leads to s';

- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ gives transition probabilities $\mathcal{P}(s, a, s') = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$;

- ι is the initial-state distribution;

- $\gamma \in [0, 1)$ is the discount factor.

The *Markov property* asserts that the future is independent of the past given the present state:

$$\mathbb{P}(S_{t+1} = s' \mid S_t = s, S_{t-1}, \ldots) = \mathbb{P}(S_{t+1} = s' \mid S_t = s).$$

A policy $\pi(a \mid s)$ specifies the agent's behavior: for each state s, it gives the probability of choosing action a. Under a policy π, the agent generates a trajectory $\tau = (s_0, a_0, r_1, s_1, \ldots)$, and accumulates rewards. The *discounted return* is defined from time t as

$$J_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_{t+T},$$

where T may be finite (episodic tasks) or infinite (continuing tasks). When $\gamma = 1$ in a finite-horizon problem, this sum is often called the *undiscounted return*.

Policy performance is assessed using two fundamental functions:
The *value function* gives the expected return when starting in state s and following π in the next steps.

$$V^\pi(s) \triangleq \mathbb{E}_\pi[J_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right],$$

and it also satisfies the following Bellman recursive equation:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s', r} \mathcal{P}(s', r \mid s, a) \left[r + \gamma V^\pi(s') \right].$$

The *action-value function* captures the expected return when the agent takes action a in state s and then follows π:

$$Q^\pi(s, a) \triangleq \mathbb{E}_\pi[J_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right].$$

From these definitions, the following functions are defined:

- The *optimal value function.*

$$V^*(s) \triangleq \max_\pi V^\pi(s)$$

- The *optimal action-value function.*

$$Q^*(s,a) \triangleq \max_\pi Q^\pi(s,a)$$

- The *advantage function*, which measures how much better action a is compared to the policy's average in state s.

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Solving an MDP means finding a policy π^* that maximizes the expected discounted return from every state.
Its Bellman recursive formulation is:

$$Q^\pi(s,a) = \sum_{s',r} \mathcal{P}(s',r \mid s,a) \left[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s',a') \right].$$

From these definitions, the following functions are defined:
The *optimal value function.*

$$V^*(s) \triangleq \max_\pi V^\pi(s).$$

The *optimal action-value function.*

$$Q^*(s,a) \triangleq \max_\pi Q^\pi(s,a).$$

The *advantage function*, which measures how much better action a is compared to the policy's average in state s.

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

Solving an MDP means finding a policy π^* that maximizes the expected discounted return from every state.

3.2.2 Reinforcement Learning algorithms

Several RL algorithms have been developed to solve a wide range of tasks, from trivial problems to highly complex ones. They can be classified according to five key characteristics:

Model-Based vs. Model-Free Model-based methods exploit the MDP's transition function $\mathcal{P}$ and reward function $\mathcal{R}$ to compute the value function V, while model-free methods estimate V only through exploration.

Sampling vs. Planning Sampling methods improve policies or value estimates by drawing samples (real or simulated trajectories), while planning methods use a known model to perform look-ahead computations over possible future states.

Bootstrapping vs. Episodic Bootstrapping methods update their estimates at each time step using existing estimates (Temporal Difference), and do not require an episode to terminate. Episodic (Monte Carlo) methods instead wait until the end of an episode and use the full return.

Discrete vs. Continuous Discrete methods assume finite state and action spaces (often handled via tabular representations), while continuous methods operate over infinite spaces and typically rely on function approximators (e.g., neural networks).

On-Policy vs. Off-Policy On-policy methods improve the same policy that they use to collect data from the environment, while off-policy methods learn a separate target policy from data generated by a different behavior policy.

The most suitable algorithm for the locomotion task handles continuous state and action spaces. It is model-free, as the dynamics model of the environment is not trivial to obtain and therefore relies on sampling interactions to gather experience. It employs bootstrapping updates and an off-policy learning scheme to maximize flexibility.

3.2.3 Deep Actor-critic Reinforcement Learning

An Actor-Critic algorithm is composed of two core elements: the actor, which selects actions based on the policy, and the critic, which estimates the value of those actions using an approximated value function. The actor is responsible for making decisions, while the critic provides feedback to improve the policy. Both handle high-dimensional spaces with deep neural networks, in fact, they have learnable parameters adjusted during training.

The policy is defined as a parametric function $\pi_{\boldsymbol{\theta}}(a \mid s)$, where $\boldsymbol{\theta}$ represents the learnable parameters. The objective of policy optimization is to determine the parameters $\boldsymbol{\theta}$ that maximize the expected episodic reward $J(\boldsymbol{\theta})$. In practice, the policy is often rappresented as a Gaussian distribution over the state space:

$$\pi_{\boldsymbol{\theta}}(a \mid s) = \mathcal{N}(a \mid \mu_{\boldsymbol{\theta}}(s), \Sigma_{\boldsymbol{\theta}}(s)).$$

Policy gradient methods offer a way to optimize $J(\boldsymbol{\theta})$ by performing gradient ascent on the policy gradient $\nabla J(\boldsymbol{\theta})$. The *Policy Gradient Theorem (PGT)* provides a way to compute the gradient of the policy given the Q-value function:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{(s,a)\sim d^{\pi_{\boldsymbol{\theta}}},\pi_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)\, Q^{\pi_{\boldsymbol{\theta}}}(s,a)\right].$$

Deep Actor-Critic methods trade off rigorous mathematical formulation for performance, because the PGT formulation is difficult to apply in the context of deep neural networks. This is due to the fact that the original loss function is very complex to optimize as the state distribution depends on the current policy. The most commonly used surrogate loss is:

$$L(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\substack{s\sim d^q \\ a\sim\pi}} \left[\hat{A}(s,a)\right],$$

where $\pi_{\boldsymbol{\theta}}$ is the updated policy and q is the current one being used. $\hat{A}$ is an estimate of the advantage function, $\hat{A}(s,a) \approx A^q(s,a)$. The approximation is accurate when $\pi_{\boldsymbol{\theta}} \approx q$, and d^q is the discounted state distribution.

3.2.4 Proximal Policy Optimization

The state-of-the-art RL algorithms introduce the concept of *entropy* related to a policy π in a state s. It is defined as:

$$H(\pi(\cdot \mid s)) = -\sum_{a\in\mathcal{A}} \pi(a \mid s) \log \pi(a \mid s) = -\mathbb{E}_{a\sim\pi}[\log \pi(a \mid s)].$$

This quantity measures the amount of randomness in the policy. To encourage exploration, entropy regularization can be added to the surrogate loss:

$$L_{\alpha}^{\text{reg}}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \alpha\, \mathbb{E}_{s\sim u^q}\left[H(\pi(\cdot \mid \mu(s), \Sigma(s)))\right].$$

Entropy regularization prevents the policy from becoming too deterministic, maintaining exploration.

The *Proximal Policy Optimization* (PPO) [18] algorithm exploits the concept of trust regions. The approach is formalized as a simple optimization of a modified surrogate loss:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(s,a)\sim q,\mathcal{P}} \left[\min\left(\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{q(a \mid s)}\, \hat{A}(s,a),\ \text{clip}\left(\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{q(a \mid s)}, 1-\epsilon, 1+\epsilon\right) \hat{A}(s,a)\right)\right].$$

The clipping of the importance sampling ratio $\frac{\pi_{\boldsymbol{\theta}}(a|s)}{q(a|s)}$ prevents policy updates from deviating excessively from the sampling distribution. This acts as an implicit mechanism to enforce a trust region.

3.2.5 Safe Reinforcement Learning

In *Safe Reinforcement Learning*, the environment is modeled as a *Constrained Markov Decision Process* (CMDP). A CMDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \iota, \gamma, \mu_0, \mathcal{K} \rangle$, where $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \iota, \gamma \rangle$ corresponds to a standard MDP, and $\mathcal{K}$ is the set of constraint functions:

$$\mathcal{K} := \{k_i : \mathcal{S} \to \mathbb{R} \mid i \in \{1, \dots, K\}\}.$$

To ensure safe exploration, the objective is to avoid constraint violations throughout the learning process. This is done by solving the following constrained optimization problem:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \right], \quad \text{s.t. } k_i(s_t) \leq 0, \quad \forall i \in \{1, \dots, K\},$$

where the trajectory $\tau = [s_0, a_0, \dots, s_T]$, with states $s_t \in \mathcal{S} \subset \mathbb{R}^n$, and actions $a_t \in \mathcal{A} \subset \mathbb{R}^m$, is sampled from the policy π. The inequality constraints $k_i : \mathcal{S} \to \mathbb{R}$ define the safety requirements that must be satisfied at each time step across all trajectories.

3.2.6 ATACOM: Acting on the Tangent Space of the Constraint Manifold

The *Acting on the Tangent Space of the Constraint Manifold* (ATACOM) framework provides a method that achieves safe exploration by leveraging knowledge of the system's dynamics and constraints. At its core, ATACOM introduces the notion of a *constraint manifold*, transforming the original constrained optimization problem into an unconstrained one defined over this manifold.

To maintain safety, the action space is redefined as the tangent space of the constraint manifold. This guarantees that any action selected in the safe space will always satisfy the system's constraints, ensuring safety during the learning process. A visual representation of the method is provided in Figure 3.2.

Based on the definition of the constraints, the safe region within the state space is characterized by the set $\mathcal{C}$, defined as:

$$\mathcal{C} := \{\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n \mid k(\mathbf{s}) \leq 0\}.$$

To enable a more flexible formulation of the constraints, a vector of *slack variables* $\boldsymbol{\mu} \in [0, +\infty)^K$ is introduced. This leads to the augmented constraint function:

$$\mathbf{c}(\mathbf{s}, \boldsymbol{\mu}) := \mathbf{k}(\mathbf{s}) + \boldsymbol{\mu}.$$

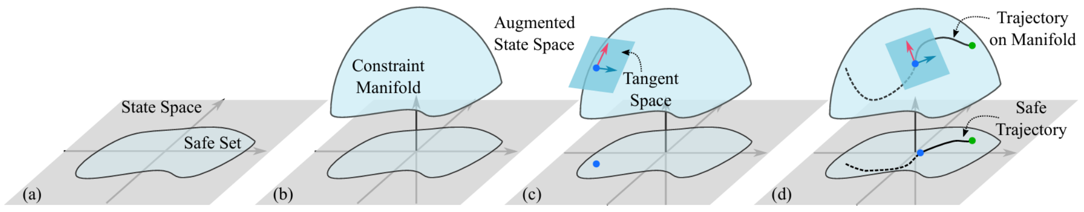


Figure 3.2: Conceptual description of ATACOM. (a) The safe set is defined by constraint functions in the original state space. (b) These constraints define the constraint manifold in an augmented state space that captures the safe set. (c) At each point on the constraint manifold, the corresponding tangent space is computed. (d) The resulting trajectory evolves on these tangent spaces, ensuring that, once projected back into the original state space, the motion remains safe. Image taken from [1].

The corresponding constraint manifold is then defined as the set of all state-slack pairs that satisfy the equality condition:

$$\mathcal{M} := \{(\mathbf{s}, \boldsymbol{\mu}) \in \mathcal{D} \mid \mathbf{c}(\mathbf{s}, \boldsymbol{\mu}) = \mathbf{0}\}.$$

Here, $\mathcal{D} := \mathcal{S} \times [0, +\infty)^K \subset \mathbb{R}^N$ denotes the *augmented state space*, with dimensionality $N = n + K$, where n represents the dimension of the original state space $\mathcal{S}$. The set $\mathcal{C}$ represents the projection of the manifold $\mathcal{M}$ onto the original state space. This implies that for every pair $(\mathbf{s}, \boldsymbol{\mu})$ belonging to $\mathcal{M}$, the associated state $\mathbf{s}$ lies within the safe set $\mathcal{C}$.

To control the slack variables $\boldsymbol{\mu}$, a virtual dynamical system is defined. For each component $i \in \{1, \dots, K\}$:

$$\dot{\mu}_i = \alpha_i(\mu_i) u_{\mu,i},$$

where α_i is a class-$\mathcal{K}$ function that is locally Lipschitz continuous, and $\mathbf{u}_\mu \in \mathbb{R}^K$ denotes the virtual control input vector.

By incorporating these dynamics, the augmented system can be written as:

$$\begin{bmatrix} \dot{\mathbf{s}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} = \begin{bmatrix} f(\mathbf{s}) \\ 0 \end{bmatrix} + \begin{bmatrix} G(\mathbf{s}) & 0 \\ 0 & A(\boldsymbol{\mu}) \end{bmatrix} \begin{bmatrix} \mathbf{u}_\mathbf{s} \\ \mathbf{u}_\mu \end{bmatrix},$$

where $A(\boldsymbol{\mu}) \in \mathbb{R}^{K \times K}$ is a diagonal matrix with elements $A_{ii} = \alpha_i(\mu_i)$, and $\mathbf{u}_\mu = [u_{\mu,1}, \dots, u_{\mu,K}]^\top$ is the corresponding control input.

The safe controller ensures that the system respects the constraint manifold. If the augmented state remains on the manifold, the original system state stays within the safe set. To achieve this, the controller is designed to adjust the augmented state along directions that remain tangent to the constraint manifold. In other words, the velocity vector $\begin{bmatrix}\dot{\mathbf{s}} & \dot{\boldsymbol{\mu}}\end{bmatrix}^\top$ must belong to the tangent space $\mathcal{T}_{(\mathbf{s},\boldsymbol{\mu})}\mathcal{M}$.

Under this condition, the safe control input is defined as:

$$\begin{bmatrix}\mathbf{u}_s \\ \mathbf{u}_\mu\end{bmatrix} = -\mathbf{J_u}^\dagger\psi - \lambda\mathbf{J_u}^\dagger\mathbf{c} + \mathbf{B_u}\mathbf{u}, \tag{3.2}$$

where $\mathbf{J_u}^\dagger$ is the pseudoinverse of $\mathbf{J_u} = \begin{bmatrix}J_k(\mathbf{s})G(\mathbf{s}) & A(\boldsymbol{\mu})\end{bmatrix}$, the constraint drift $\psi = J_k(\mathbf{s})f(\mathbf{s})$ is induced by the system drift $f(\mathbf{s})$ and $\mathbf{B_u}$ is the tangent space basis in the matrix form such that $\mathbf{J_u}\mathbf{B_u} = 0$. The function u is the task-specific feedback controller i.e., the RL agent.

The first term $-\mathbf{J_u}^\dagger\psi$ is the *drift compensation term*, which corrects the drift caused by the system dynamics. The second term $-\lambda\mathbf{J_u}^\dagger\mathbf{c}$ is the *contraction term*, responsible for pulling back the state toward the manifold. The final term $\mathbf{B_u}\mathbf{u}$ is the *tangential term*, which generates a vector field lying in the tangent space of the constraint manifold.

In this way, the RL agent learns the policy directly from the safe action space rather than the original one. This allows the use of any RL algorithm, as the ATACOM controller simply reshapes the action, ensuring that the sampled actions operate only within the tangent space that is the safe subset of the action space. The complete algorithm is described in Algorithm 1.

Algorithm 1 Safe Reinforcement Learning with ATACOM

```
1: **Initialize:** policy $\pi$, replay buffer $\mathcal{D}$, number of steps $N$
2: **for** $k \leftarrow 1$ to $N$ **do**
3:     Sample action $\mathbf{u} \sim \pi(\cdot|\mathbf{s})$
4:     Obtain safe control $\mathbf{u}_s \leftarrow \mathrm{ATACOM}(\mathbf{s}, \mathbf{u})$        ▷ Alg. 2
5:     Execute $\mathbf{u}_s$ and obtain $\mathbf{s}'$ and reward $r$
6:     Update replay buffer $\mathcal{D} \leftarrow (\mathbf{s}, \mathbf{u}, \mathbf{s}', r)$
7:     Update RL agent using the replay buffer $\mathcal{D}$
8: **end for**
```

Chapter 4

Methodology

This chapter outlines the primary contributions of this thesis, which enable the application of ATACOM to the quadruped locomotion setting. Section 4.1 details the low-level controller developed to track joint velocities instead of joint positions. Next, Section 4.2 introduces the constraints designed for the locomotion task, providing a description of each constraint function and its corresponding Jacobian. Finally, Section 4.3 motivates and presents the proposed ATACOM extensions.

4.1 Low-level Control Adaptation

The original ATACOM formulation is based on the assumption that the control dynamics is modeled by a velocity-controlled system $\dot{\mathbf{s}} = f(\mathbf{s}) + G(\mathbf{s})\,\mathbf{u}_s$. Thus, the control input $\mathbf{u}$ must be expressed as desired joint velocities $\dot{\mathbf{q}}_d$, which in this case correspond to the RL agent's actions. Consequently, the ATACOM state $\mathbf{s}$ is expressed by the joint positions $\mathbf{q}$.

This work adopts the approach of Rudin et al. [3] as a baseline on which the ATACOM method is applied, and then evaluates the resulting performance by comparing it with the unconstrained version. To ensure a more unbiased evaluation of the constraint contribution, Rudin's configuration is modified as little as possible.

4.1.1 Position-based Controller

In the work of Rudin et al., the RL policy outputs actions that represent the desired joint positions of the robot. To convert the position commands into torque commands applied to the robot's actuators, a PD controller is employed. Its role is to compute the appropriate torque to effectively track the target joint positions. The controller is formulated as follows:

$$\mathbf{u}(t) = K_P\left(\mathbf{q}_d - \mathbf{q}(t)\right) + K_D\dot{\mathbf{q}}(t) \tag{4.1}$$

where $\mathbf{q}_d$ is the desired joint position, $\mathbf{q}(t)$ is the current joint position, and $\dot{\mathbf{q}}(t)$ is the current joint velocity. The proportional term aims to quickly reduce the tracking error between the desired and actual joint positions, while the derivative term penalizes excessively fast responses that may cause overshooting and lead to oscillatory behavior around the target.

4.1.2 Velocity-based Controller

To apply ATACOM, actions must represent desired joint velocities. To meet this requirement, a new low-level controller is proposed to replace the previously discussed position-based controller. The objective is to track joint velocities directly, in order to compute the appropriate joint torques. For this purpose, a PI controller is designed as follows:

$$\mathbf{u}(t) = K_P\left(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}(t)\right) + K_I\int_0^t \left(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}(\tau)\right)\, d\tau,$$

where $\dot{\mathbf{q}}_d$ is the desired joint velocity and $\dot{\mathbf{q}}(t)$ is the current joint velocity. The proposed controller uses a proportional term as the main source of error correction and an integral term to compensate for constant disturbances such as gravity.

This approach enables the transformation of actions, i.e., joint velocities, into torques, thereby replacing the original controller introduced by Rudin. The results presented later in Section 5 are strongly influenced by this design choice, which is required by the ATACOM method.

Since actions now represent desired joint velocities instead of desired joint positions, the reward function must be adjusted accordingly, specifically when penalizing metrics related to the actions. Further details about the reward function are provided in Section 5.1.2.

4.2 Safe Locomotion Constraints

The design of constraints aims to support the learning process by reducing the full action space to a smaller safe subset $\mathcal{C}$ and avoiding undesirable behaviors that could cause damage to the robot or its surroundings, or simply waste energy exploring uninformative states.

Each constraint function $\mathbf{k}(\mathbf{q})$ depends only on the state, i.e., the joint positions $\mathbf{q}$. The result is a vector with one or more elements, which, when concatenated, form the constraint space $\mathbb{R}^K$. The constraint function must satisfy a simple requirement: the values associated with desired safe states must be negative, i.e.,

$$k(\mathbf{q}) \leq 0 \quad \forall \mathbf{q} \in \mathcal{C}.$$

To build the safe controller, the constraint Jacobian $\mathbf{J}(\mathbf{q})$ is required in order to predict how the constraints will change in response to the control input. This Jacobian maps joint velocities to constraint velocities, resulting in a matrix of size $K \times n$.

4.2.1 Joint Position Constraint

The joint position constraint is relatively simple and acts directly on the values of the joint positions, imposing a limit on the allowable range for each joint. This ensures that each joint remains within its minimum and maximum bounds.

The main advantage is to constrain the robot to more useful configurations, avoiding awkward or misleading positions that could slow down the learning process or cause crashes. The goal is to select limits that do not restrict natural movement patterns conducive to successful locomotion, while excluding less relevant ones. Since analyzing each trajectory explored by the robot would be overly complex, plausible bounds are estimated by observing the distribution of joint positions during training epochs. The values that remain prevalent in the final learned policy are assumed to be the most important.

In this case, the constraint functions and their Jacobians are straightforward to define. Two separate constraint functions are used: one to enforce the upper bounds and one to enforce the lower bounds.

$$\begin{aligned} \mathbf{k}_{\max}(\mathbf{q}) &= \mathbf{q} - \mathbf{q}_{\max} \qquad & \mathbf{J}_{\max}(\mathbf{q}) &= \mathbf{I}, \\ \mathbf{k}_{\min}(\mathbf{q}) &= \mathbf{q}_{\min} - \mathbf{q} \qquad & \mathbf{J}_{\min}(\mathbf{q}) &= -\mathbf{I}^{\cdot} \end{aligned}$$

The only parameters to be tuned are the joint limits $\mathbf{q}_{\max}$ and $\mathbf{q}_{\min}$.

4.2.2 Foot Position Constraint

The foot position constraint limits the x and y coordinates of the feet within a restricted area, described by an ellipse centered under the corresponding hip. Figure 4.1 illustrates the shape of this constraint.

The constraint is designed to improve learning by preventing foot placements that are too far from the associated hip and not aligned with natural walking patterns. In this way, the action space is reduced and the robot can explore more efficiently.

The constraint design involves two components: the ellipse function and the foot position function. These are combined to compute the foot's distance from

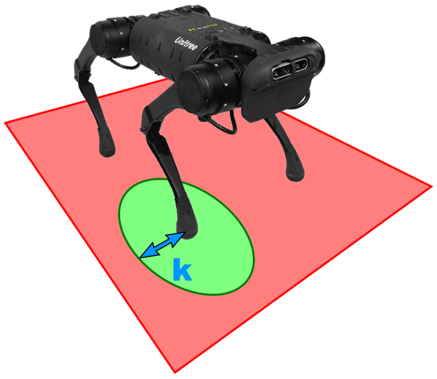


Figure 4.1: Illustration of foot position constraint structure. The ellipse defines the safe area for the foot in the xy-plane. The value $k(\mathbf{q})$ represents the distance of the foot from the ellipse boundary, shown by the blue line.

the ellipse given the joint positions. The foot position is obtained from the robot's kinematic model using the homogeneous transformation of the foot f with respect to the hip h:

$$\mathbf{f}_{\text{pose}}(\mathbf{q}) = {}^{h}\mathbf{T}_{f} = {}^{h}\mathbf{T}_{b} \cdot {}^{b}\mathbf{T}_{f},$$

where ${}^{h}\mathbf{T}_{b} = ({}^{b}\mathbf{T}_{h})^{-1}$ and b denotes the robot base frame.

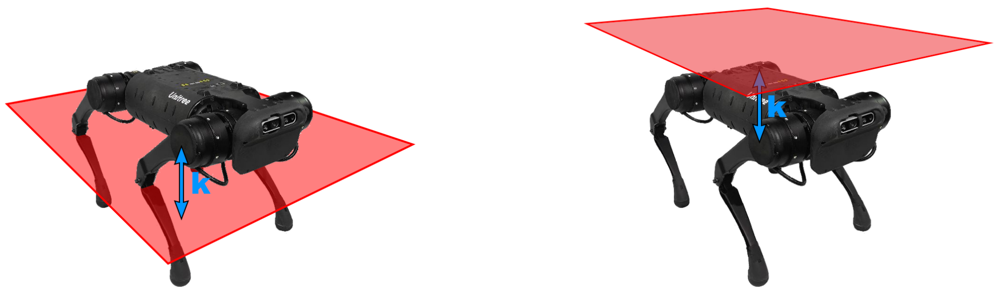
$$f_x(\mathbf{q}) = [\mathbf{f}_{\text{pose}}(\mathbf{q})]_{0,3} \quad f_y(\mathbf{q}) = [\mathbf{f}_{\text{pose}}(\mathbf{q})]_{1,3}\,.$$

The constraint value is defined as the square root of the ellipse equation to ensure linear growth rather than quadratic. The Jacobian is derived using the chain rule, combining the Jacobian of the forward kinematics $\mathbf{J}_f(\mathbf{q})$ and the Jacobian of the root ellipse function:

$$k(\mathbf{q}) = s(\mathbf{q}) - 1$$

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \frac{f_x(\mathbf{q})}{\alpha^2 s(\mathbf{q})} & \frac{f_y(\mathbf{q})}{\beta^2 s(\mathbf{q})} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{f_x}(\mathbf{q}) \\ \mathbf{J}_{f_y}(\mathbf{q}) \end{bmatrix}$$

$$s(\mathbf{q}) = \sqrt{\frac{f_x(\mathbf{q})^2}{\alpha^2} + \frac{f_y(\mathbf{q})^2}{\beta^2}}.$$



(a) Minimum base height (b) Maximum base height

Figure 4.2: Illustration of base height constraints structure. The blue line $k(\mathbf{q})$) represents the constraint value. In the case (a) the safe space is described by states where the base is above the red plane. In the case (b) the safe space is described by states where the base is below the red plane.

The adjustable parameters are α and β, which define the size and shape of the ellipse. An enhanced version of this constraint scales these parameters proportionally to the commanded velocity, adapting the ellipse to better suit the desired movement direction.

4.2.3 Base Height Constraint

This constraint can be divided into two components, both of which aim to regulate the height of the robot's base from the ground. They are visually described in Figure 4.2. The first limits the minimum distance and can be visualized by placing a virtual plane below the base that pushes back and supports the robot when it gets too close to the ground. The second acts in the opposite direction, enforcing a maximum height limit that can be imagined as a plane above the robot, keeping it below a certain threshold.

The expected benefit of the second constraint is not related to faster learning or improved walking behavior. Instead, it is introduced to evaluate whether and how the robot is able to adapt its walking style in confined environments or in the presence of overhead obstacles. In contrast, the minimum height constraint is expected to provide structural support to prevent falls, encouraging the robot to return to a standing position and helping it to quickly learn how to stabilize itself.

The constraint function approximates the height of the base from the ground by computing the vertical distance between the lowest foot and the base. This is a valid approximation under the assumption that at least one foot is in contact with the ground. Although this condition is not always guaranteed, it is typically satisfied during normal operation.

The distance between the base and the foot, along with its Jacobian, is computed using forward kinematics:

$$\begin{aligned} \mathbf{k}_{\max}(\mathbf{q}) &= \mathbf{h}(\mathbf{q}) - \mathbf{h}_{\max} & \qquad \mathbf{k}_{\min}(\mathbf{q}) &= \mathbf{h}_{\min} - \mathbf{h}(\mathbf{q}) \\ \mathbf{J}_{\max}(\mathbf{q}) &= \mathbf{J}_h(\mathbf{q}) & \qquad \mathbf{J}_{\min}(\mathbf{q}) &= -\mathbf{J}_h(\mathbf{q}) \end{aligned}$$

,

where

$$\mathbf{h}(\mathbf{q}) = \max_{\text{feet}} \left[\mathbf{f}_{\text{pose}}(\mathbf{q})\right]_{2,3}, \quad \mathbf{J}_h(\mathbf{q}) = \max_{\text{feet}} \left[\mathbf{J}_{f_{\text{pose}}}(\mathbf{q})\right]_{2,:}.$$

The parameters $\mathbf{h}_{\max}$ and $\mathbf{h}_{\min}$ define the height of the constraint planes.

4.2.4 Foot Orientation Constraint

The foot orientation constraint forces each foot to approach the terrain with a desired angle. It reduces the range of allowed orientations when the foot is closer to the ground while permitting more freedom when the foot is farther from it. The angle margin should not be too narrow, in order to support long steps that require a wide range of orientations. The benefits are similar to the previous constraints: reducing useless movements and encouraging those beneficial for walking.

The constraint function exploits the tangent space of $SO(3)$ to measure the distance between the current foot orientation and the desired one. The norm of the Lie algebra element expresses how close the foot is to the target orientation.

The margin around the desired orientation that defines the safe area is qualitatively shown in Figure 4.3. It is linearly proportional to the foot's height. Using the same assumption as before, the height can be approximated as the foot's distance from the ground (i.e., the furthest foot).

$$\alpha_{\text{threshold}} = d(\mathbf{q}) \cdot \frac{\alpha_{\max} - \alpha_{\min}}{d_{\max}} + \alpha_{\min},$$

where $d(\mathbf{q})$ is the vertical distance of the foot from the ground. The value $\alpha_{\text{threshold}}$ denotes the maximum allowable deviation angle between the desired and current orientation. The current angle is computed as follows:

$$\alpha_{\text{foot}} = \left\| \text{Log}\left(\mathbf{R}_{\text{foot}} \cdot \text{Exp}(\boldsymbol{\theta}_{\text{target}})^\top\right) \right\|,$$

where $\mathbf{R}_{\text{foot}} = \left[\mathbf{f}_{\text{pose}}(\mathbf{q})\right]_{[:3,:3]}$, and $\boldsymbol{\theta}_{\text{target}}$ is the desired foot orientation expressed in Lie algebra coordinates.

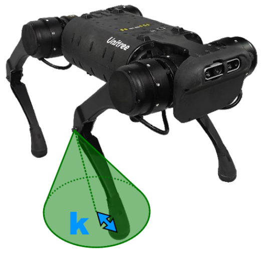


Figure 4.3: Illustration of foot orientation constraint structure. Without rigorous detail, it can be visualized as a cone centered on the desired orientation, represented by the cone axis. The base diameter of the cone increases or decreases with the height of the foot. The blue arrow represents the constraint value $k(\mathbf{q})$.

The actual constraint function and its Jacobian are then described as:

$$
\begin{aligned}
k(\mathbf{q}) &= \alpha_{\text{foot}} - \alpha_{\text{threshold}} \\
\mathbf{J}(\mathbf{q}) &= \mathbf{J}_{\alpha_{\text{foot}}}(\mathbf{q}) - \mathbf{J}_{\alpha_{\text{threshold}}}(\mathbf{q}) \\
\mathbf{J}_{\alpha_{\text{foot}}}(\mathbf{q}) &= \frac{\operatorname{Log}\left(\mathbf{R}_{\text{foot}} \cdot \operatorname{Exp}(\boldsymbol{\theta}_{\text{target}})^{\top}\right)}{\alpha_{\text{foot}}} \cdot \mathbf{J}_{\mathbf{R}_{\text{foot}}} \\
\mathbf{J}_{\alpha_{\text{threshold}}}(\mathbf{q}) &= \mathbf{J}_{d}(\mathbf{q}) \cdot \frac{\alpha_{\max} - \alpha_{\min}}{d_{\max}}.
\end{aligned}
$$

The parameters used to adjust the constraint are $\alpha_{\max}$ and $\alpha_{\min}$, which define the allowable angular margin around the desired orientation when the foot is elevated ($\alpha_{\max}$) and when it is in contact with the ground ($\alpha_{\min}$). The value $\boldsymbol{\theta}_{\text{target}}$ specifies the desired foot orientation.

4.3 ATACOM Extension

This thesis enhances the ATACOM method by improving the design of both the tangential term and the contraction term in the safe controller (3.2). These modifications allow the robot to better explore the entire safe action space. The need to update the ATACOM controller arises from the inherent instability of the locomotion task, which presents an environment where the robot begins training from an unbalanced initial condition. This is in contrast with previously tested manipulation tasks, such as Air Hockey, where the robotic arm starts from a stable default position, and the safe space is neither high-dimensional nor complex.

An other important issue is about the mismatch between the desired and the current joints velocity due to the implication about exploits a PI controller. It tracks the desired velocity by reacting on the base of this discrepancy by quickly reducing it. Despite that, there are many moments where the action, i.e., the desired joint velocity does not correspond to the actual one because the controller still has to reach it. This problem reflects on the capability of ATACOM to control the action based on the system dynamics. The safe action provided by ATACOM is the desired joint velocity, and meanwhile it is reached, the actual joint velocity is not actually controlled by ATACOM, therefore there is no guarantee that it is safe. So the agent performs an approximate safe action that will approximate not violate the constraints and that is why constraint violation could occur more often than expected.

In locomotion, constraints provide an attractive but only partial solution to the early lack of equilibrium. The robot's initial sub-goal is simply to remain upright and take its first steps in the desired direction. Constraints are particularly effective in achieving this sub-goal, especially the minimum height constraint, which provides immediate support to prevent falls and helps stabilize the robot in a balanced configuration, although not necessarily the optimal one.

Indeed, the robot gains immediate benefit from this assistance, but it complicates further exploration toward discovering a more effective policy. This occurs because the robot tends to learn a suboptimal posture that increases the cumulative return by preventing episode termination, but, in the meantime, it builds a poor foundation for learning to walk.

This phenomenon occurs to varying degrees with all of the proposed constraints and is caused by the way in which ATACOM modifies the action space near the constraint boundaries. Specifically, the method transforms the RL agent's action by rotating it into the slack variable space $\boldsymbol{\mu}$, according to the constraint values. This has the effect of scaling down the action in proportion to its distance from the constraint boundary.

While this mechanism is useful when the robot is pushing against a constraint, discouraging unsafe motion, it becomes a limitation when the quadruped attempts to move away from that boundary, thus encouraging overly conservative behavior near already discovered stable configurations.

Directional Constraints and *Exponential Moving Average Error Correction* are proposed in this thesis to address this limitation, enabling more effective exploration and ultimately convergence to an optimal policy.

4.3.1 Directional Constraints

To address the issue of symmetric morphing, the *Directional Constraints* mechanism is implemented. The goal is to eliminate the effect that scales down every action near the constraint boundary, regardless of whether the action would lead the agent to violate the constraint or move toward a safer region.

To achieve this result, we modify the safe controller (3.2), specifically the tangential term $\mathbf{B}_u\mathbf{u}$. This term is the only one that depends on the agent's action $\mathbf{u}$. While the other two terms are responsible for keeping the state on the constraint manifold, the tangential term transforms the RL action so that it lies within the tangent space. This is achieved by removing from the action all components orthogonal to the tangent space and retaining only the tangential components. This projection is performed by multiplying the agent's action by the matrix $\mathbf{B}_u$, which is a basis of the tangent space satisfying $\mathbf{J}_u\mathbf{B}_u = 0$. As a result, the projected action lies entirely in the tangent space.

This multiplication is the key mechanism for morphing the action, and the matrix $\mathbf{B}_u$ determines which components are suppressed to maintain safety. Directional Constraints provide a method to build the tangent space basis using only a subset of the full constraint set, selecting only the constraints that prevent the agent from entering unsafe regions. The decision of which constraints to retain is made by evaluating the constraint derivatives $\dot{\mathbf{c}}(\mathbf{q})$, which describe how the constraint values evolve based on the current action. If the derivative is positive, the constraint value is increasing; otherwise, it is decreasing. The constraint derivative is computed as follows:

$$\dot{\mathbf{c}}(\mathbf{s}) = \mathbf{J}_k(\mathbf{s})\, G(\mathbf{s})\, \mathbf{u}(\mathbf{s}).$$

Once the constraints derivative are computed, we can select the equivalent constraints based on the derivative sign, keeping the one with positive sign. The overall extended ATACOM procedure to compute safe actions is described in Algorithm 2, where the highlighted part (steps 4 and 5) implements the proposed Directional Constraints.

This solution allows us to morph only the actions that would increase the constraint value, by temporarily removing the constraints that are guaranteed not to be violated, because the robot moves in the opposite direction. A comparison between the original and the Directional Constraints approach is illustrated in Figure 4.4. Both (b) and (c) ensure constraint satisfaction, unlike case (a), where ATACOM is not applied. Additionally, Directional Constraints (c) selectively scale down only the actions that move towards the constraint. In contrast, the original ATACOM method (b) also modifies the other actions, causing the agent to stay closer to the boundary than in (c).

In conclusion, the Directional Constraints extension offers an improved version

Algorithm 2 Extended ATACOM with Directional Constraints

```
1: Input: $\mathbf{s}$, $\mathbf{u}$                                   ▷ At each step
2: Determine the slack variable
       $\mu \leftarrow \max(-k(\mathbf{s}), \mathrm{tol})$
3: Compute the Jacobians and the drift
       $\mathbf{J}_G \leftarrow J_k(\mathbf{s})G(\mathbf{s})$
       $\mathbf{J}_u(\mathbf{s}, \mu) \leftarrow \begin{bmatrix}\mathbf{J}_G(\mathbf{s}) & A(\mu)\end{bmatrix}$
       $\psi(\mathbf{s}) \leftarrow J_k(\mathbf{s})f(\mathbf{s})$
4: Compute constraints derivative
       $\dot{\mathbf{c}}(\mathbf{s}) \leftarrow \mathbf{J}_k(\mathbf{s})\, G(\mathbf{s})\, \mathbf{u}(\mathbf{s})$
5: Select Jacobians with positive derivative
       $\mathbf{J}_{\text{pos}} \leftarrow \mathbf{J}_u$  IF  $\dot{\mathbf{c}}(\mathbf{s}) \geq 0$
6: Compute the tangent space basis
       $\mathbf{B}_u \leftarrow \mathrm{SmoothBasis}(\mathbf{J}_{\text{pos}})$
7: Compute the constraint value
       $c(\mathbf{s}, \mu) \leftarrow k(\mathbf{s}) + \mu$
8: Compute safe control output $u_s$                               ▷ Eq. 3.2
9: Output: $u_s$
```

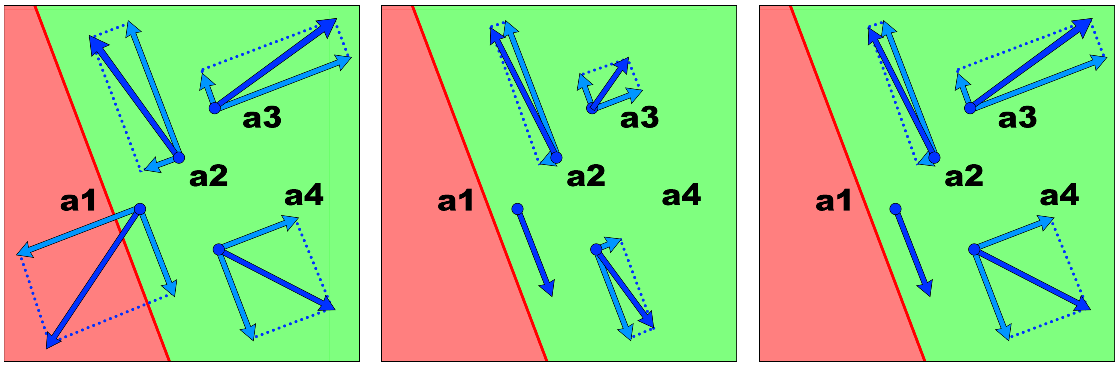
of ATACOM, enabling the agent to freely explore the safe action space without unnecessary interference.

4.3.2 Exponential Moving Average Error Correction

The *Exponential Moving Average Error Correction* (EMAEC) modifies the safe ATACOM controller (3.2) to increase the strength of error correction when one or more constraints are violated. To specifically achieve this objective, we focus on the contraction term $-\lambda \mathbf{J}_u^\dagger \mathbf{c}$, which is primarily responsible for forcing the agent back into the safe region. Whenever a state no longer belongs to the tangent space of the constraint manifold, this term actively pushes the state back onto the manifold.

The required corrective action is determined using the constraint Jacobian pseudo inverse $\mathbf{J}_u^\dagger$, which indicates whether the constraint value is increasing (positive) or decreasing (negative). The magnitude of this correction is proportional to the constraint value $\mathbf{c}$, representing the severity of the violation, and it is scaled by a constant gain λ, which is a task-dependent hyperparameter. This correction is then combined with the other two controller's terms, drift compensation and tangential terms, to compute the final safe action applied to the environment.

EMAEC enhances the contraction term by adding a new component. While



(a) Unconstrained (b) ATACOM (c) ATACOM with Directional Constraints

Figure 4.4: Four different actions near a constraint (red line), handled by three methods:

(a) Unconstrained method without ATACOM:
each action is shown as sampled from the agent without modification.

(b) With ATACOM applied:
all actions are scaled down in their orthogonal component,
while tangential components are preserved.

(c) With ATACOM extended with Directional Constraints:
only actions directed toward the constraint (a1 and a2) are scaled in their orthogonal component;
actions heading toward the safe region (a3 and a4) remain unchanged.

it still uses $\mathbf{J}_u^\dagger$, it no longer depends only on the constraint value $\mathbf{c}$. Instead, it incorporates an error signal $e(\mathbf{c})$, computed as the Exponential Moving Average (EMA) of $\mathbf{c}$, which estimates the average magnitude of recent violations, giving more weight to recent samples. The error signal at step t is defined recursively as:

$$e(\mathbf{c}_t) = \frac{1}{W}\mathbf{c}_t + \left(1 - \frac{1}{W}\right) e(\mathbf{c}_{t-1}),$$

where the smoothing factor $\frac{1}{W}$ acts as a moving window of size W over recent violations. With this new term, the safe controller can be reformulated as:

$$\begin{bmatrix} \mathbf{u}_s \\ \mathbf{u}_\mu \end{bmatrix} = -\mathbf{J_u}^\dagger \psi - \mathbf{J_u}^\dagger \lambda(\mathbf{c}) + \mathbf{B_u}\mathbf{u}, \tag{4.2}$$

where the updated contraction term $\mathbf{J_u}^\dagger\lambda(\mathbf{c})$ consists of the Jacobian pseudo-inverse and a new function $\lambda(\mathbf{c})$ that combines the original correction with the EMA-based term:

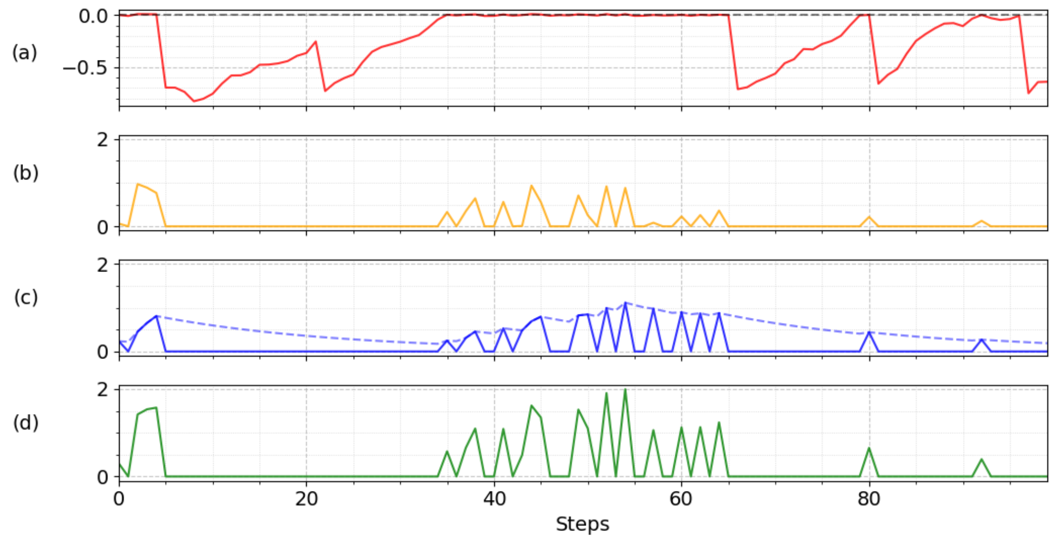


Figure 4.5: Behavior of the contraction term with and without EMAEC when violations occur:
(a) Constraint value $k(\mathbf{q})$: the contraction term is active when positive.
(b) Magnitude of the original contraction term $\lambda_c\mathbf{c}$.
(c) EMA error contribution $\lambda_e\mathbf{e}$: solid line shows active violation response; dotted line shows running average over time.
(d) EMAEC: combination of (b) and (c), that is $\lambda(\mathbf{c})$.

$$\lambda(\mathbf{c}) = \lambda_c\mathbf{c} + \lambda_e e(\mathbf{c}),$$

with λ_c and λ_e being hyperparameters that control the relative strength of each component. The addition of the EMA error $e(\mathbf{c})$ allows the controller to respond more aggressively to repeated violations: its value increases quickly upon violations and decays slowly otherwise.

Figure 4.5 displays a demonstrative example of the difference between the original approach (b) and the EMAEC approach (d). Specifically, plot (c) isolates the $e(\mathbf{c})$ contribution and illustrates how the correction strength gradually increases with successive constraint violations: the first violation at step 35 reaches approximately 0.2, which is lower than (b), around 0.4. By the last hit at step 65, after several violations, the value climbs close to 1, becoming dominant compared to (b), which remains around 0.4, the same as at the beginning.

In conclusion, EMAEC improves the behavior of the contraction term in the

ATACOM controller, particularly when the robot state lies outside the safe region, i.e., when the state no longer lies on the constraint manifold and must be corrected. The updated contraction term becomes more responsive during successive violations, resulting in stronger corrections. The same results cannot be achieved simply by increasing λ, as this could lead to overcorrection in situations where excessive force is unnecessary, potentially causing undesirable instability.

Chapter 5

Experimental Analysis

This chapter evaluates the performance of the constrained approaches in comparison to the unconstrained one. Section 5.1 introduces the robot structure and defines the quadruped locomotion task. Next, Section 5.2 outlines the experimental setup, the metrics used to evaluate performance, and the unconstrained baseline adopted as a starting point. Section 5.3 then analyzes the contribution of the two ATACOM extensions. Afterwards, Section 5.4 compares the performance of the proposed constraint-based approaches with the unconstrained baseline in terms of learning curves. Finally, Section 5.5 compares these approaches in terms of constraint violations.

5.1 Quadruped Robot Locomotion Task

This Section presents the specifications of the robot structure and the locomotion task, detailing the observation space, the action space, and the reward function. Additionally, it covers the domain randomization strategy employed to improve the robustness and generalization of the learned policies.

5.1.1 Robot Description

The quadruped robot used in this thesis is the Unitree A1, designed for research and development in autonomous systems. The robot can be modeled as a floating base with four legs, each consisting of three revolute joints, for a total of 12 degrees of freedom. Figure 5.1 shows a the

The kinematic structure of each leg begins with the hip joint, which connects the base to the hip link. The thigh joint is attached between the hip link and the thigh link, followed by the calf joint, which connects the thigh link to the calf link. At the end of the calf link is the foot, which makes contact with the ground to

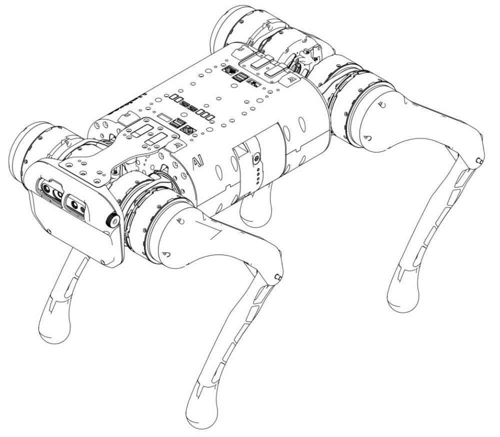


Figure 5.1: Unitree A1

support the robot.

All joints are revolute and controlled by torque, so the robot's configuration can be described by the joint angles $\boldsymbol{\theta} \in \mathbb{R}^{12}$.

The kinematic chain of a leg is defined by the sequence of homogeneous transformations between each joint and its corresponding link. The forward kinematics of the foot with respect to the base frame $\mathcal{F}_b$ is given by:

$$^{b}\mathbf{T}_f = {}^{b}\mathbf{T}_h \cdot {}^{h}\mathbf{T}_t \cdot {}^{t}\mathbf{T}_f$$

Each transformation $^{i}\mathbf{T}_j \in SE(3)$ includes both rotation and translation, and is a function of the corresponding joint angle θ_i.

5.1.2 Task Description

The objective of the task is to train a quadruped robot to walk on flat terrain while tracking a target velocity. The simulation environment is designed to support parallelized episodes, allowing the agent to learn from batches of trajectories collected simultaneously. The whole configuration replicates the work of Rudin et al. [3].

At the start of each episode, the robot is initialized in a randomized configuration close to its default pose. The velocity command is also randomized in the x and y directions and yaw component, allowing the policy to generalize over varying targets and initial conditions. The commands have a probability of 0.2% of being resampled at every environment step.

The policy receives a 48-dimensional observation vector composed of proprioceptive and command measurements. These include linear and angular velocities of the base ($\mathbb{R}^3$ each), the gravity vector ($\mathbb{R}^3$), commanded velocities ($\mathbb{R}^3$), joint

Term	**Definition**	**Weight**
Linear velocity tracking	$\phi(\mathbf{v}_b^{*,xy} - \mathbf{v}_b^{xy})$	$1dt$
Angular velocity tracking	$\phi(\boldsymbol{\omega}_b^{*,z} - \boldsymbol{\omega}_b^{z})$	$0.5dt$
Linear velocity penalty	$-\mathbf{v}_{b,z}^2$	$4dt$
Angular velocity penalty	$-\lVert\boldsymbol{\omega}_b^{xy}\rVert^2$	$0.05dt$
Joint motion	$-\lVert\ddot{\mathbf{q}}_j\rVert^2 - \lVert\dot{\mathbf{q}}_j\rVert^2$	$0.001dt$
Joint torques	$-\lVert\boldsymbol{\tau}_j\rVert^2$	$0.00002dt$
Joint position rate	$-\lVert\mathbf{q}_j^*\rVert^2$	$0.25dt$
Action rate	$-\lVert\dot{\mathbf{q}}_j^*\rVert^2$	$0.25dt$
Collisions	$-n_{\text{collision}}$	$0.001dt$
Feet air time	$\sum_{f=0}^{4}(t_{\text{air},f} - 0.5)$	$2dt$

Table 5.1: Definition of reward terms. The z-axis is aligned with gravity. $\phi(x) := \exp\left(-\frac{\lVert x\rVert^2}{0.25}\right)$. Symbols definition:

$\mathbf{q}_j$, $\dot{\mathbf{q}}_j$, $\ddot{\mathbf{q}}_j$:	joint positions, velocities, and accelerations
$\mathbf{q}_j^*$, $\dot{\mathbf{q}}_j^*$:	target joint positions and velocities
$\boldsymbol{\tau}_j$:	joint torque vector
$\mathbf{v}_b$, $\boldsymbol{\omega}_b$:	base linear and angular velocities
$\mathbf{v}_b^*$, $\boldsymbol{\omega}_b^*$:	commanded linear and angular velocities
$n_{\text{collision}}$:	number of collisions
t_{air}:	feet air time
dt:	environment time step

positions and velocities ($\mathbb{R}^{12}$ each) and the previously applied action ($\mathbb{R}^{12}$). Each observation component is normalized, and all inputs include added noise except for the most recently applied action and the commanded velocities.

Episodes terminate when the robot enters an absorbing state, defined as any state in which the trunk makes contact with the ground plane, indicating a fall or loss of stability.

The reward function is a weighted sum of ten components listed in Table 5.1. The main terms encourage tracking of the commanded velocity while minimizing motion along undesired directions. Additional terms promote smooth and natural locomotion by penalizing joint torques, joint accelerations, changes in joint targets, and undesired collisions. An extra reward term incentivizes longer steps, leading to more visually appealing gaits.

The action space consists of 12 dimensions, corresponding to target joint positions. A low-level PD controller converts the position error into motor torques to drive the actuators accordingly.

To allow sim-to-real transfer, domain randomization techniques are applied during training. These include randomizing the ground friction and applying unexpected pushes to the robot. Specifically, the robot's base is accelerated in both x and y directions, to teach them a more stable walking style. The details of the randomization parameters are provided in the supplementary material.

5.2 Experimental Configuration

This Section provides details about the hardware and software setup used to run the experiments. It also lays the groundwork for the constraint-based experiments by introducing the evaluation metrics and the baseline configuration.

5.2.1 Experimental Setup

The experimental setup relies on the ISAAC Sim simulator [33], which provides a parallel simulation environment for the Unitree A1 robot. The agent is trained using the Proximal Policy Optimization (PPO) algorithm [18] that leverages the implementation provided by the MushroomRL library [34]. Training is conducted on a GPU NVIDIA RTX 2080 with 11 GB of VRAM, which enables efficient storage and handling of data from parallel simulations.

A total of 4096 environments are simulated in parallel. The training consists of 15 epochs, each comprising 4.915.200 steps, calculated as 4096 robots running for 1200 steps per epoch. All plots and reported results are obtained by evaluating the same configuration across five different random seeds to ensure statistical consistency.

5.2.2 Metrics

The metrics used for evaluating the experiments are the undiscounted cumulative return $\mathcal{R}$, which measures the policy's ability to complete the task, and the average constraint violation. The latter represents the percentage of steps in which the constraint is violated, i.e., when $\mathbf{k}(s) \geq 0$, relative to the total number of steps. This metric reflects how frequently the constraint boundary is breached.

Each of these metrics is typically visualized in plots with respect to another quantity, usually epochs. For instance, the learning curve in Figure 5.2 shows the values of $\mathcal{R}$ across epochs. Similarly, the average violation is plotted over epochs, as shown in Figure 5.4.

To further understand the severity of constraint violations, the actual violation values $\mathbf{k}(s)$ are measured and plotted to represent their distribution. These plots (e.g., Figure 5.5) show the violation magnitude on the x-axis and the percentage

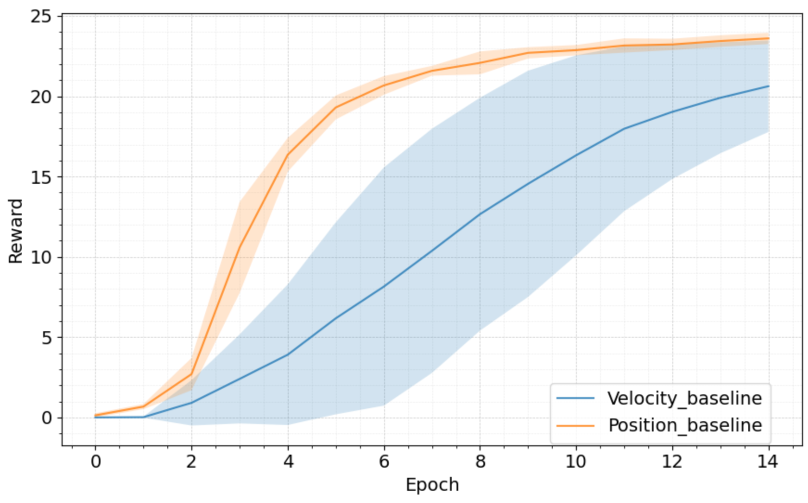


Figure 5.2: Undiscounted cumulative return $\mathcal{R}$ across 5 seeds for the two unconstrained methods based on Rudin's setup. The position baseline uses joint positions as actions, while the velocity baseline uses joint velocities.

of occurrences relative to the total number of violations on the y-axis, effectively acting as a probability density function of violation severity.

Further details about the context and specific purposes of these metrics are provided in the experimental analysis.

5.2.3 Unconstrained Baseline

The baseline provides the starting point against which all experiments are compared, in order to assess the benefits of adopting a strategy different from it. In this case, the baseline serves as a reference for the unconstrained approach.

The most unbiased baseline mirrors the ATACOM setup without applying any constraints. As previously described in Section 4.1, actions are represented as desired joint velocities to be compatible with the ATACOM method and are converted into torques using a PI controller. This configuration is referred to as the *velocity baseline*.

Despite multiple attempts, the velocity-controlled baseline converges more slowly than the state-of-the-art approach by Rudin, which uses desired joint positions as actions. Figure 5.2 compares the velocity and position baselines based on the undiscounted cumulative return $\mathcal{R}$.

While both baselines eventually complete the task with comparable final rewards, their learning dynamics differ significantly. The position baseline rapidly converges to a high-reward policy in just a few epochs and then plateaus. In contrast, the velocity baseline shows a more gradual learning curve with a wider confidence interval across runs, indicating lower consistency and higher sensitivity to stochastic conditions during training.

In the rest of the analysis, we primarily compare the results with the velocity baseline to evaluate the improvement gained by adopting constraints. Subsequently, the best results are compared to the position baseline to assess their performance against the state of the art.

5.3 ATACOM Extension Ablation

This Section describes the contribution of the two new features integrated into ATACOM that are introduced in Section 4.3.1 and in Section 4.3.2, designed to address the problem of the agent getting stuck near constraint boundaries and being unable to escape, which limits its ability to explore effectively. The experiments provide problematic situations that are solved by enabling the ATACOM extension. A comparison between ATACOM with and without extension is used to evaluate its benefits.

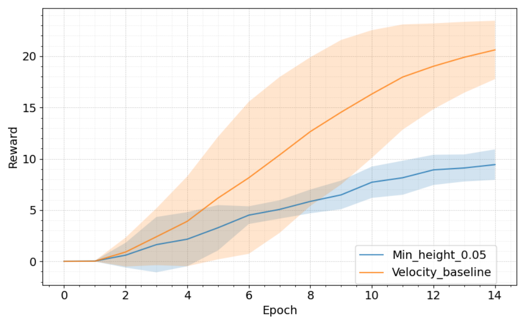
5.3.1 Directional Constraints

The chosen setup to evaluate the performance of the Directional Constraints extension includes a minimum base height constraint. The constraint parameter $\mathbf{h}_{\min}$ is set to 0.05 m, while the average base height of the baseline walking policy is approximately 0.22 m (ranging from 0 m to 0.45 m). The expected behavior of the agent is to initially exploit the constraint to avoid falling and then progressively learn to walk properly by rising above it. It should not intact the final policy, thus quite far from the baseline walking behavior.

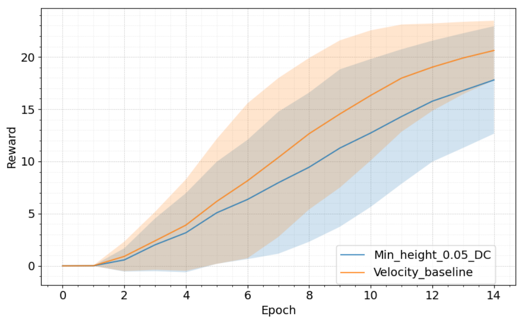
Figure 5.3 reports the undiscounted cumulative return $\mathcal{R}$, showing a significant improvement of our proposed ATACOM with Directional Constraints compared to the original ATACOM.

The original ATACOM approach does not behave as desired. Figure 5.4, which reports the average constraint violation over training epochs, highlights persistent violations throughout the entire learning process, including the final policy. This suggests that the agent remains consistently around or below the minimum height of 0.05 m, failing to reach the average height of the baseline policy.

To investigate more deeply the behavior of the policy without extensions, we analyze the distribution of constraint violations to assess their magnitude. Figure 5.5 shows that almost 70% of the violations exceed the constraint by less than 0.02 m, indicating that the agent remains constantly near or in contact with the constraint plane.

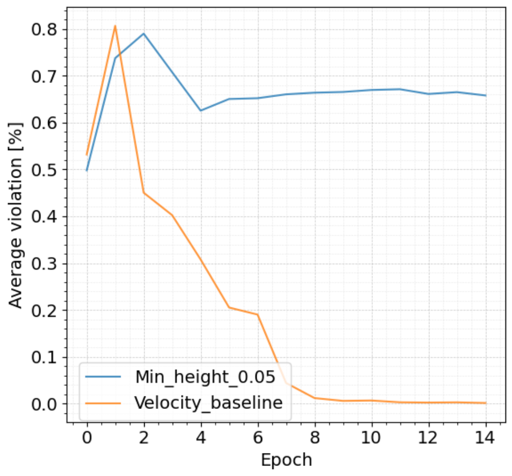


(a) Original ATACOM

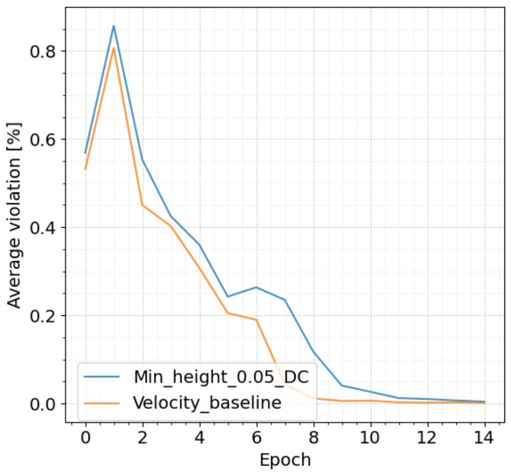


(b) ATACOM w/ Directional Constraints

Figure 5.3: Undiscounted cumulative return for experiments with a minimum base height constraint set to 0.05 m. Plot (a) shows results without any ATACOM extensions, while (b) shows results with Directional Constraints (DC) enabled.



(a) Original ATACOM



(b) ATACOM w/ Directional Constraints

Figure 5.4: Average violation [%] of the experiment with minimum base height constraint set to 0.05 m. Plot (a) shows the setup without any ATACOM extensions, while (b) shows the setup with Directional Constraints (DC) enabled.

In contrast, the policy with Directional Constraints enabled exhibits a substantially different behavior, displaying consistency with the velocity baseline. Indeed, both the cumulative return and the average violations closely follow the trajectory of the baseline policy. This behavioral difference is likely due to the insufficient incentive for the agent without extensions to move away from the constraint. Once

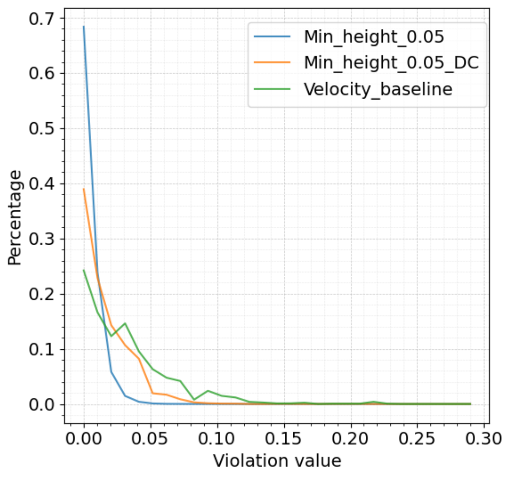


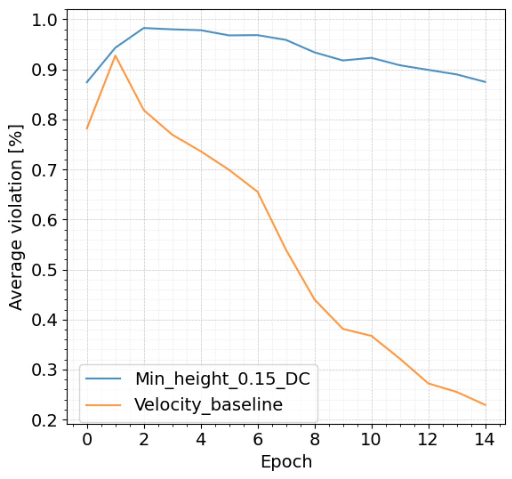
Figure 5.5: Violation distribution of the experiment with minimum base height constraint set to 0.05 m. It shows orginal ATACOM and ATACOM with Directional Constraints (DC).

the agent learns to stay close to the constraint, finding it a stable and easy solution, it becomes difficult to stand up again due to the action space being morphed in a way that suppresses the orthogonal components of the action, including those necessary for rising.

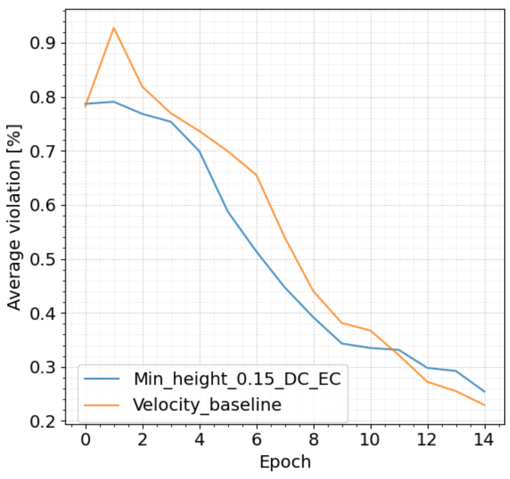
5.3.2 Exponential Moving Average Error Correction

The experiment to evaluate the benefits of introducing EMAEC is conducted under the same setup as the Directional Constraints experiments. The minimum base height constraint is set to 0.15 m to investigate whether this threshold again leads the agent to learn a sub-optimal policy by lying down. We compare the method that uses only the Directional Constraints extension with the one that combines both extensions: Directional Constraints and EMAEC, to assess whether EMAEC introduces any significant improvements.

To immediately assess the behavior of the policies with respect to the constraint, Figure 5.6 illustrates the difference in average violations between the two approaches. Despite the Directional Constraints being enabled, the policy still exhibits the same issue that the extension is designed to address. In this case, however, it is not sufficient, and therefore the EMAEC extension is introduced to handle such situations. The policy using only Directional Constraints remains below the constraint for nearly 90% of the time in each epoch, while the one with both extensions exhibits a behavior similar to the baseline, reaching around 25%



(a) ATACOM w/ Directional Constraints



(b) ATACOM w/ both extensions

Figure 5.6: Comparison of average constraint violation [%] for the minimum base height set to 0.15 m. (a) uses only the Directional Constraints; (b) combines both Directional Constraints and EMA Error Correction.

of violations in the final policy.

The cumulative return plots of both approaches are shown in Appendix B.2.2, as they do not reveal significant differences. Both methods perform similarly to each other and also compared to the baseline. The only notable distinction is the narrower confidence interval in the setup with both extensions, suggesting greater reliability across different training runs.

To better understand the reason behind the frequent violations of the approach with Directional Constraints, the violation distribution of the policy is shown in Figure 5.7. The violations are evenly distributed within 0.1 m below the constraint plane, indicating that the error correction is not sufficient to force the agent to detach from the constraint. This suggests that the policy learns to walk in the region just below the constraint by possibly exploiting the constraint contraction force to counteract gravity. In contrast, the policy with both extensions behaves almost identically to the baseline. The EMAEC extension provides additional support to enhance safety guarantees through stronger corrective action.

A drawback of EMAEC is that its aggressive corrective force, intended to discourage constraint violations, can introduce instability due to sharp reactive pushes. This is especially problematic in fragile configurations. For instance, enforcing a relatively low maximum base height may lead to instability during early training epochs, as the strong corrections might push the robot toward the ground,

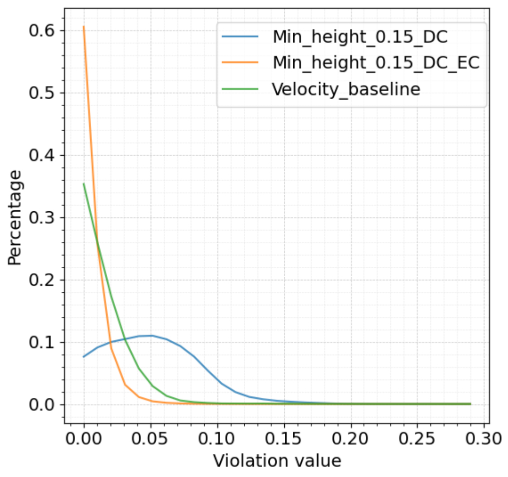


Figure 5.7: Violation distribution of the experiment with the minimum base height constraint set to 0.15 m. It shows ATACOM with Directional Constraints (DC) and ATACOM with Directional Constraints and EMA Error Correction (DC and EC).

degrading performance. Although EMAEC is designed to gradually scale the correction strength depending on repeated violations, its effectiveness still depends on careful parameter tuning. Moreover, there is no guarantee that suitable values for λ_c and λ_e can be found for every scenario.

5.4 Learning Curve Comparison with vs. without Constraints

The constraints experiments are designed to evaluate their benefits based on their specific characteristics. In particular, we investigate whether they can improve the learning process.

All the results presented are from experiments that do not include the EMAEC feature, as it can often slightly degrade performance when not properly tuned for a given constraint. To ensure an unbiased comparison among constraint types, without introducing additional variability due to EMAEC configuration, this feature has been excluded from these experiments.

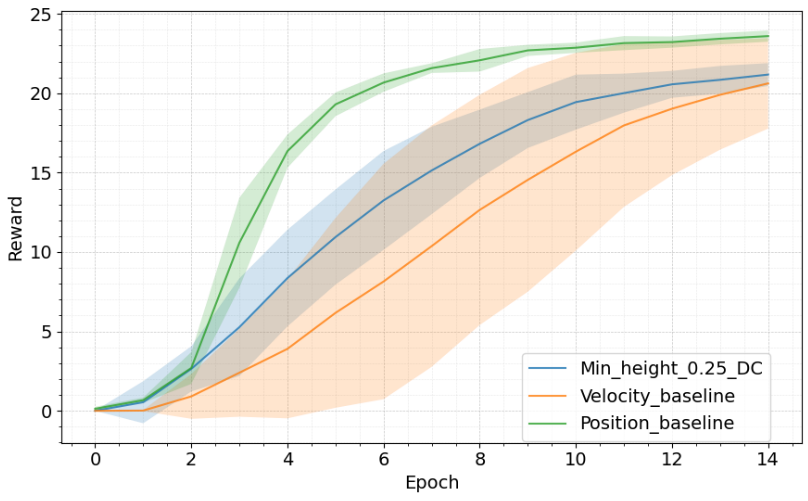


Figure 5.8: Undiscounted cumulative return for the experiment with minimum height set to 0.25 m.

5.4.1 Base Height Constraint

The experiment based on the base height constraint aims to accelerate learning by supporting the robot with a minimum height constraint set near the average height of the baseline policy. The best performance is observed when the height parameter $\mathbf{h}_{\text{min}}$ is set to 0.25 m.

As shown in Figure 5.8, the results for this configuration exceed the velocity baseline in terms of learning speed, policy performance, and consistency across experiments. As hypothesized, the constraint helps counteract gravity, accelerating the learning process during the early epochs. Additionally, it improves stability by reducing the likelihood of falling or initializing in poor configurations. Despite these results, the final policy does not reach the reward level or learning speed of the position baseline proposed by Rudin et al.

5.4.2 Joint Position Constraint

The experiment related to the joint position constraint begins with the analysis of the joint position distribution provided in Appendix B.2.3. This analysis highlights that, under the unconstrained baseline policy, the thigh joint operates within a narrower range than physically allowed, whereas the hip and calf joints exploit most of their available motion range.

Based on this observation, the experimental setup introduces a constraint that restricts the thigh joint's motion to 70% of its original range.

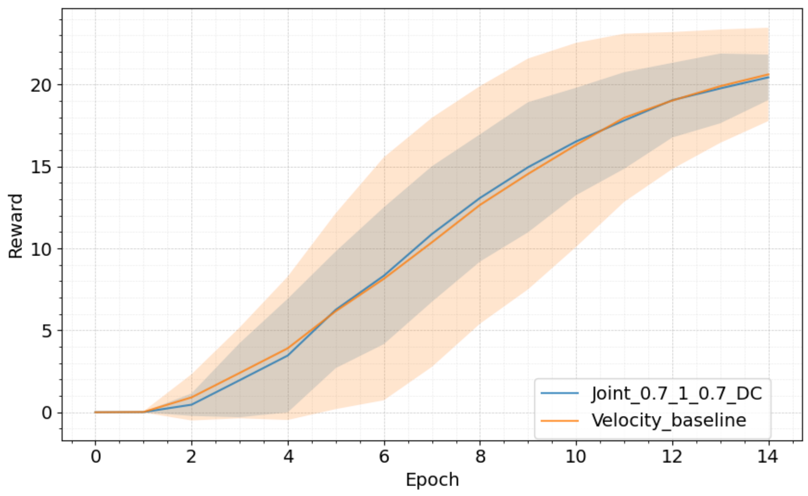


Figure 5.9: Undiscounted cumulative return of the experiment with joint position constraint applied to the thigh, limited to 70% of its original range.

Figure 5.9 presents the performance results of this configuration. While the learning curve closely follows the velocity baseline in terms of average cumulative return, a notable difference emerges in the width of the confidence intervals. Specifically, the narrower confidence band observed in the constrained setup indicates improved consistency across training runs. This suggests that the joint position constraint contributes to a more stable optimization trajectory. By reducing the size of the viable action space in a targeted and informed manner, the constraint appears to filter out unstable behaviors.

5.4.3 Foot Orientation Constraint

The setup for constraining foot orientation involves configuring several parameters. The key parameter is the desired angle $\boldsymbol{\theta}_{\text{target}}$, expressed as a vector in the Lie algebra of SO(3). Determining this value, along with the auxiliary parameters $\alpha_{\max}$ and $\alpha_{\min}$, is non-trivial for two main reasons.

First, the final policy's foot orientation is significantly different from the default pose. Consequently, setting the target orientation directly to that value may push an untrained agent into an unstable configuration early in training, resulting in frequent falls.

Second, the velocity commanded in the final policy introduces additional complexity. The reward function encourages the agent to move quickly, which leads to rapid and wide leg movements that continuously alter the feet's orientation.

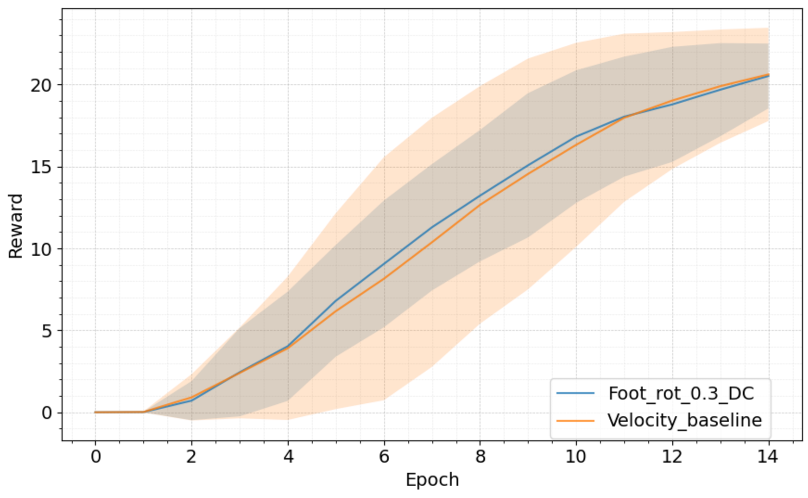


Figure 5.10: Cumulative return over training epochs for the experiment with the foot orientation constraint and Directional Constraints enabled. The target orientation is set to -0.3 in the y-component of $\boldsymbol{\theta}_{\text{target}}$, with $\alpha_{\max} = \pi$ and $\alpha_{\min} = \frac{\pi}{2}$.

Therefore, the tolerance margins around the target orientation need to be sufficiently wide to account for the natural variation in foot pose that occur during fast locomotion.

Considering these factors, the constraint is implemented in a more relaxed and general form. Instead of enforcing a rigid orientation, it encourages a preferred and commonly observed orientation, while still allowing for natural variability. The selected parameters are:

$$\boldsymbol{\theta}_{\text{target}} = \begin{bmatrix} 0 \\ -0.3 \\ 0 \end{bmatrix} \qquad \begin{aligned} \alpha_{\max} &= \pi \\ \alpha_{\min} &= \frac{\pi}{2} \end{aligned}$$

Figure 5.10 shows the results of the experiment. The most evident outcome of applying this constraint is improved stability in the learning process, which becomes more consistent and predictable. However, the constraint does not lead to a significant improvement in learning speed.

5.4.4 Foot Position Constraint

The experiment to evaluate the performance of the foot position constraint is designed based on the analysis of the foot position distribution under the baseline policy, included in Appendix B.2.3. Figure 5.11 shows the learning curve for an

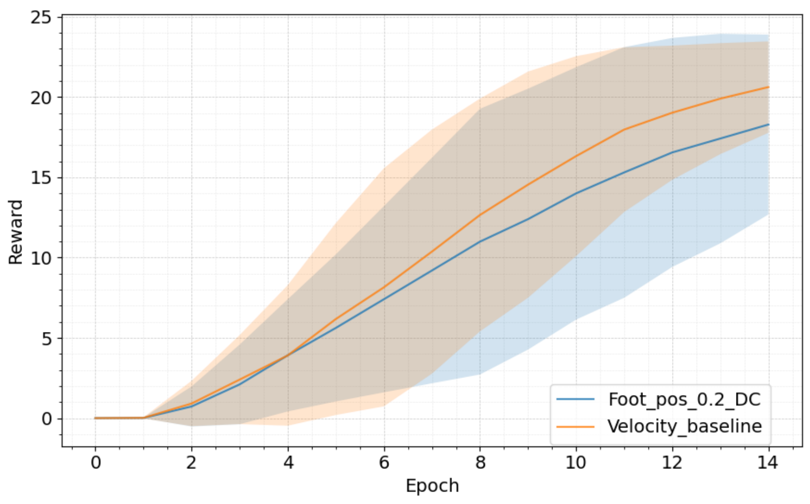


Figure 5.11: Undiscounted cumulative return of the foot position constraint with $\alpha = \beta = 0.2$m.

experiment in which the constraint is set just below the threshold that covers the majority of the foot positions produced by the baseline, namely α and β are set to 0.2 m. The results are nearly indistinguishable from the baseline and do not provide any additional benefits.

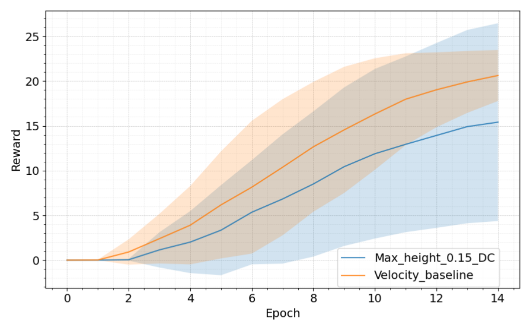
Although several values for the ellipse size are tested, none significantly improved learning performance. Larger values have no impact on the task, while smaller ones overly restrict the robot's movements without yielding any benefits. Even the version that scales α and β proportionally to the command velocity performs approximately the same as the original formulation.

In conclusion, the constraint neither supports nor enforces any notable behavior that would justify its integration into the task.

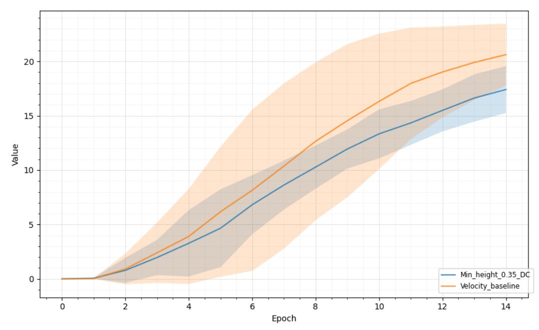
5.5 Constraints Safety Evaluation

This Section analyzes the overall behavior of the constraints in terms of safety, specifically, whether ATACOM is able to guarantee compliance with the imposed limits during both training and deployment of the final policy. This is particularly relevant since ATACOM enforces *hard* constraints, in contrast to *soft* constraints, which merely encourage safe behavior without strictly ensuring it.

To assess constraint-driven safety behaviors, we investigate scenarios in which conventional walking styles are restricted by the constraints. In such cases, the agent is expected to learn a challenging locomotion strategy that remains within



(a) Max height constraint



(b) Min height constraint

Figure 5.12: Undiscounted cumulative return for experiments with unusual walking styles. Experiment (a) uses a maximum height constraint set to 0.15 m, while experiment (b) uses a minimum height constraint set to 0.35 m.

the prescribed limits.

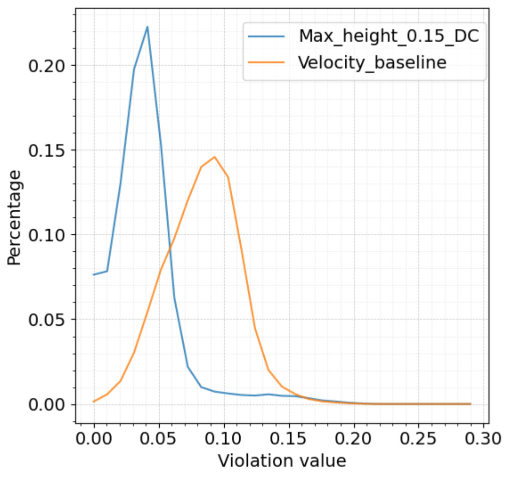
The safety evaluation is performed in the most challenging configuration to demonstrate a non-trivial case in which the agent must find a solution that both enables locomotion and satisfies the imposed constraints. The results presented below can be generally extended to many other setups with looser constraints, where the agent is expected to complete the task while remaining within the safe space without significant difficulty.

5.5.1 Constrained Walking Style

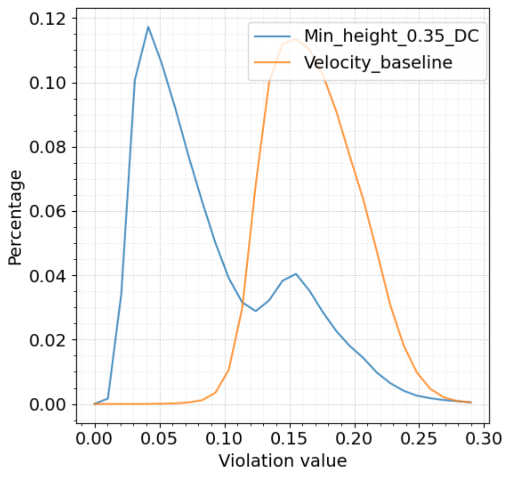
The experiments that evaluate constraint violations leverage the base height constraint to enforce either a low or high-stance walking posture. The first experiment introduces a maximum base height constraint $\mathbf{h}_{\text{max}} = 0.15$ m, which confines the robot below a plane positioned just above the ground. The second experiment applies a high minimum height constraint $\mathbf{h}_{\text{min}} = 0.35$ m, compelling the agent to maintain a high-stance gait.

Figure 5.12 shows that both policies are able to accomplish the task successfully. While the minimum height setup demonstrates great stability across experiments, the maximum height configuration presents greater difficulty in consistently achieving a high-reward policy. This is due to the challenging requirement that forces the agent to remain very close to the ground without falling.

In both cases, the constraint violation distributions shown in Figure 5.13 indicate that the majority of violations are concentrated within 0.1 m and 0.15 m beyond the constraint limits, respectively. This makes it interesting to analyze the average violations under a relaxed criterion, where small violations within a



(a) Max height constraint



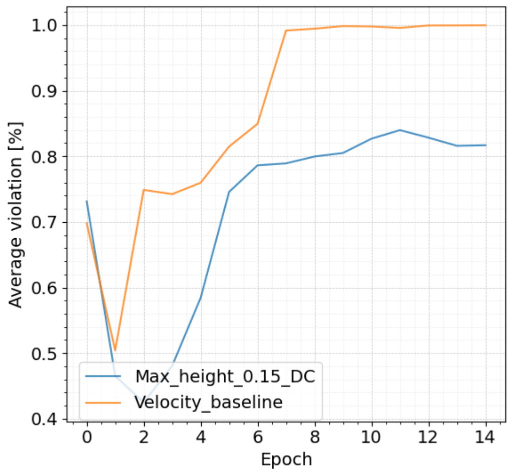
(b) Min height constraint

Figure 5.13: Constraint violation distribution for experiments with unusual walking styles. Experiment (a) uses a maximum height constraint set to 0.15 m, while experiment (b) uses a minimum height constraint set to 0.35 m.

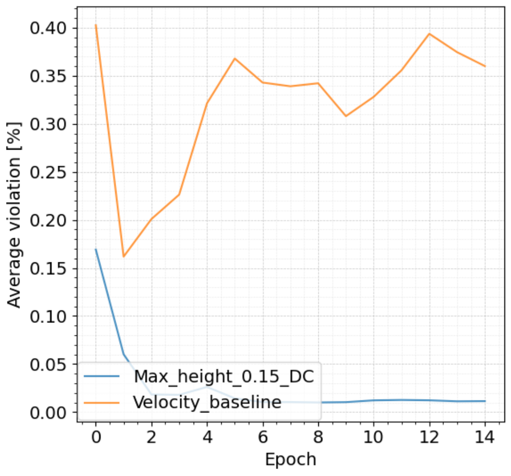
tolerance margin are not counted.

Figure 5.15 for the minimum height setup and Figure 5.14 for the maximum height setup show that the average constraint violation becomes nearly zero when an ϵ-margin is applied. In contrast, the results without a margin clearly indicate persistent violations. Using a tolerance margin allows for distinguishing between minor acceptable deviations and substantial violations, such as those observed in the velocity baseline.

Ultimately, these findings suggest that ATACOM can provide *safety within an ϵ-margin)*, accounting for model approximations and observation noise. Moreover, enabling EMAEC or tuning ATACOM's parameters, such as the action space morphing strength or error correction gain, can further reduce the necessary ϵ-margin, potentially approaching zero.

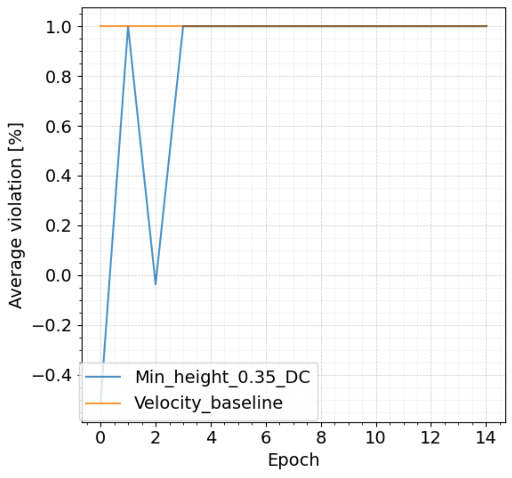


(a) No margin

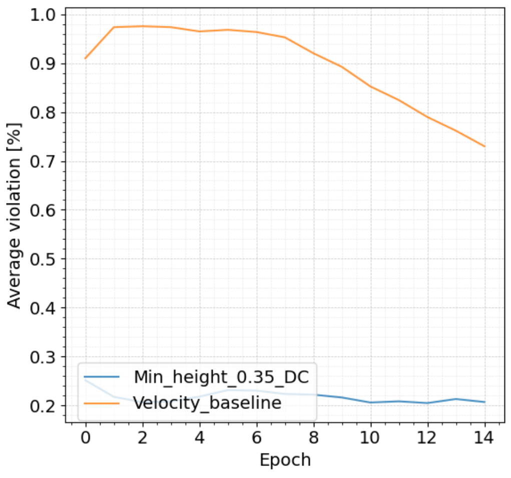


(b) ϵ-margin

Figure 5.14: Average constraint violation [%] for experiments with maximum height constraint set to 0.15 m. Configuration (a) includes all violations, while configuration (b) applies an ϵ-margin of 0.1 m to ignore minor violations within this threshold.



(a) No margin



(b) ϵ-margin

Figure 5.15: Average constraint violation [%] for experiments with minimum height constraint set to 0.35 m. Configuration (a) includes all violations, while configuration (b) applies an ϵ-margin of 0.15 m to ignore minor violations within this threshold.

Chapter 6

Conclusions

This thesis investigates the implications of adopting constraints into the locomotion task through the ATACOM method, which enforces hard constraints both during learning and in policy deployment.

The contributions of the study include two key extensions to the original ATACOM framework, addressing its limitations in handling highly unstable tasks such as locomotion. Directional Constraints and EMA Error Correction alleviate the undesired attraction to states near constraint boundaries, which may be stable but suboptimal for walking. These improvements open new possibilities for exploring safety across multiple domains and tasks.

The designed locomotion constraints were found to be competitive with the velocity baseline in terms of learning speed, generally improving reliability across different experiments and, in the best cases, accelerating the learning process. In the worst cases, their performance remained comparable to the baseline.

Among the evaluated constraints, the minimum height constraint consistently led to faster learning. However, it still falls short of the performance achieved by the state-of-the-art approach of Rudin et al. [3], which remains both faster and more robust than any of the constrained configurations tested. The gap between Rudin's results and those obtained with ATACOM is primarily due to the suboptimal velocity-based policy used as the foundation for all constrained experiments.

Although the constraints did not significantly enhance learning performance, they succeeded in delivering the desired safety guarantees. In these terms, ATACOM functions as expected, ensuring safety from the beginning of training through to the final deployable policy. As discussed in the experiments, this safety is guaranteed within an ϵ-margin that depends on various configuration aspects, particularly the mismatch between the model and the real system.

In conclusion, the thesis lays the foundations for future research in constrained locomotion based on ATACOM. Future directions may include: developing ad-hoc low-level controllers that translate target joint velocities into torque commands with performance comparable to position-based approaches such as Rudin's work; modifying the control assumption toward an acceleration-controlled system with tailored low-level controllers; designing new constraints, such as self-collision or terrain adaptation; and exploring Sim-to-Real strategies for deploying real-world policies or even enabling direct training on physical systems.

Appendix A

Background Details

A.1 Lie theory formulas

A.1.1 Special Euclidean group $SE(3)$

The exponential and logarithm maps are defined as:

$$\mathrm{Exp}(\boldsymbol{\tau}) \triangleq \begin{bmatrix} \mathrm{Exp}(\boldsymbol{\theta}) & \mathbf{V}(\boldsymbol{\theta})\,\boldsymbol{\rho} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\mathrm{Log}(\mathbf{T}) \triangleq \begin{bmatrix} \mathbf{V}^{-1}(\boldsymbol{\theta})\,\mathbf{t} \\ \mathrm{Log}(\mathbf{R}) \end{bmatrix}.$$

where $\boldsymbol{\theta} = \mathrm{Log}(\mathbf{R})$ and

$$\mathbf{V}(\boldsymbol{\theta}) = \mathbf{I} + \frac{1-\cos\theta}{\theta^2}[\boldsymbol{\theta}]_\times + \frac{\theta - \sin\theta}{\theta^3}[\boldsymbol{\theta}]_\times^2$$

Appendix B

Experimental Analysis Details

B.1 Hyperparameters

B.1.1 PPO Hyperparameters

PPO hyperparameters Table B.1.

Batch size	98304 (4096×24)
Mini-batch size	6144 (4096×1.5)
Number of epochs	5
Clip range	0.2
Entropy coefficient	0.01
Discount factor	0.99
GAE discount factor	0.95
Desired KL-divergence kl^*	0.01
Learning rate α	adaptive*
Gradient norm clipping	1

Table B.1: PPO hyper-parameters used for the training of the tested policies. The adaptive learning rate is based on the KL-divergence, the corresponding algorithm is taken from [3].

B.1.2 ATACOM Hyperparameters

ATACOM hyperparameters Table B.2.

Slack dynamics	Exponential
Slack β	10
error correction coefficient λ_c	2
EMAEC coefficient λ_e^*	0.5
EMAEC window W^*	100

Table B.2: ATACOM hyper-parameters used for the training of the constrained tested policies. * The EMAEC variables are used only when the EMAEC feature is enable.

B.2 Additional Results

B.2.1 ATACOM parallel implementation issue

The ATACOM controller is implemented on top of the PyTorch library, leveraging its functionalities to build efficient parallel operations and optimize GPU computation, thereby reducing training time. To compute the safe action $\mathbf{u}_s$, the following linear system must be solved, yielding the safe controller as previously discussed:

$$\psi(\mathbf{s}) + J_u(\mathbf{s}, \boldsymbol{\mu}) \begin{bmatrix} u_s \\ u_\mu \end{bmatrix} = \mathbf{0}$$

The `lstsq()` function from PyTorch's `torch.linalg` module is used to solve the least squares problem associated with this system. In this context, 4096 batched linear systems are expected to be solved in parallel. However, in the current implementation, this parallelism is not achieved, and the computation time grows linearly with the number of systems, suggesting they are being processed sequentially.

Several alternative implementations were tested to overcome this limitation. These included using SVD decomposition and pseudo-inverse solutions in combination with vector mapping, compilation, and scripting techniques, aiming to replace the direct use of the `lstsq` function. Figure B.1 presents a comparative analysis of the computation time for each method.

This bottleneck drastically increases the total training time, vanishing much of the benefit provided by the parallelized environment setup. Experiments using ATACOM require approximately five times more time compared to unconstrained baselines. Developing a truly parallel version of the `lstsq` function would yield significant improvements in terms of both computation time and resource efficiency.

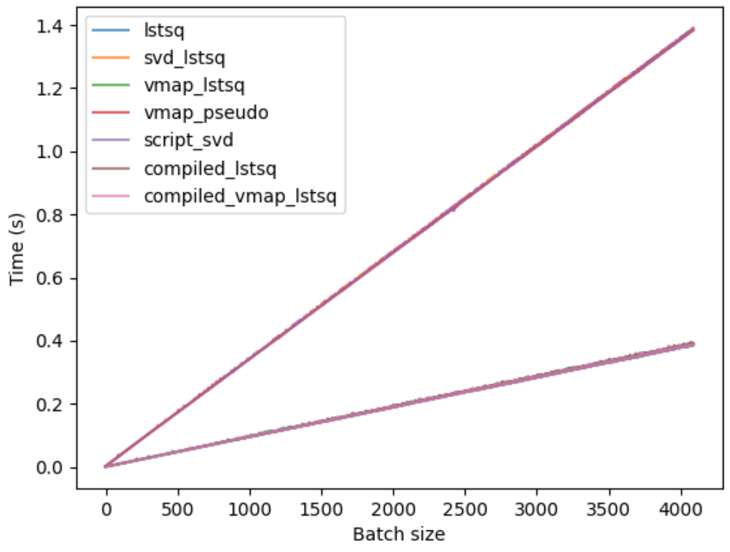


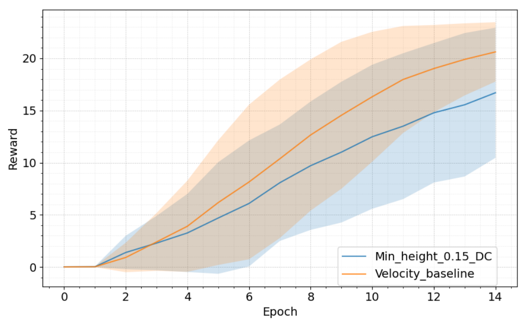
Figure B.1: The plot shows the time required to solve a linear system with increasing batch size using different techniques. The approaches can be grouped into two main categories: those that use the `lstsq` Torch function directly (faster), and those that rely on SVD or the pseudo-inverse (slower). Specifically, `lstsq`, `vmap_lstsq`, `compiled_lstsq`, and `compiled_vmap_lstsq` belong to the first group, while `svd_lstsq`, `vmap_pseudo`, and `script_svd` are in the second. Both groups scale linearly with respect to batch size, suggesting a lack of full parallel computation.

B.2.2 EMAEC Ablation

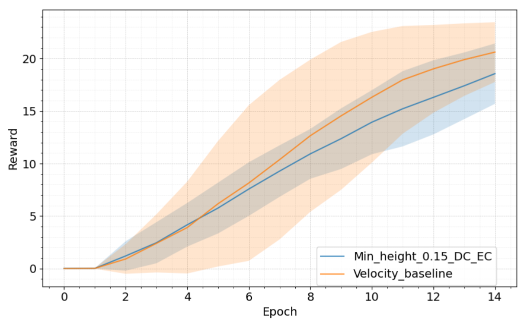
Here are reported the plots of the cumulative return of ATACOM with only Directional Constraints enables and ATACOM with both extensions.

B.2.3 Velocity baseline behavior

Analysis of the joints and feet position distribution when the agent uses the velocity baseline policy.



(a) Directional Constraints



(b) Both extensions

Figure B.2: Undiscounted cumulative return for experiments with a minimum base height constraint set to 0.15 m. Plot (a) shows results with Directional Constraints, while (b) shows results with both Directional Constraints and EMAEC enabled.

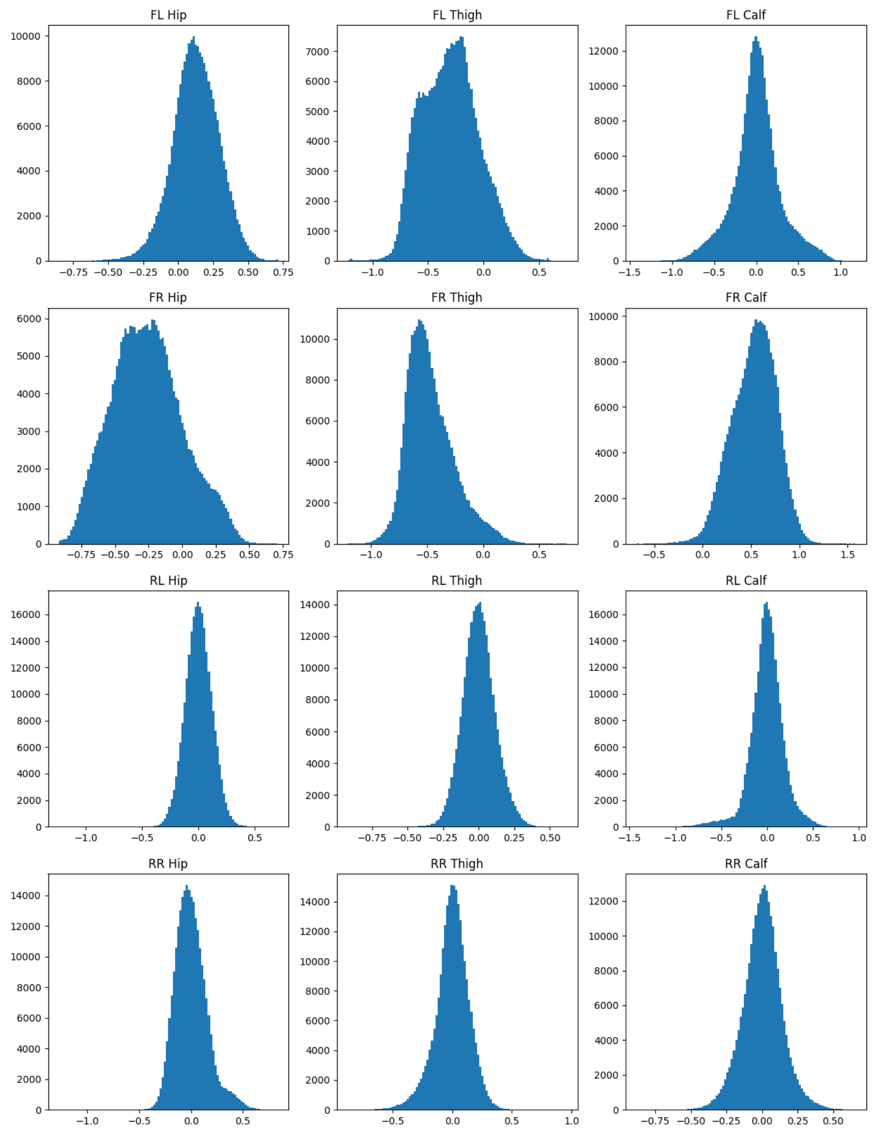


Figure B.3: Joint pos values during an episode controlled by velocity baseline.

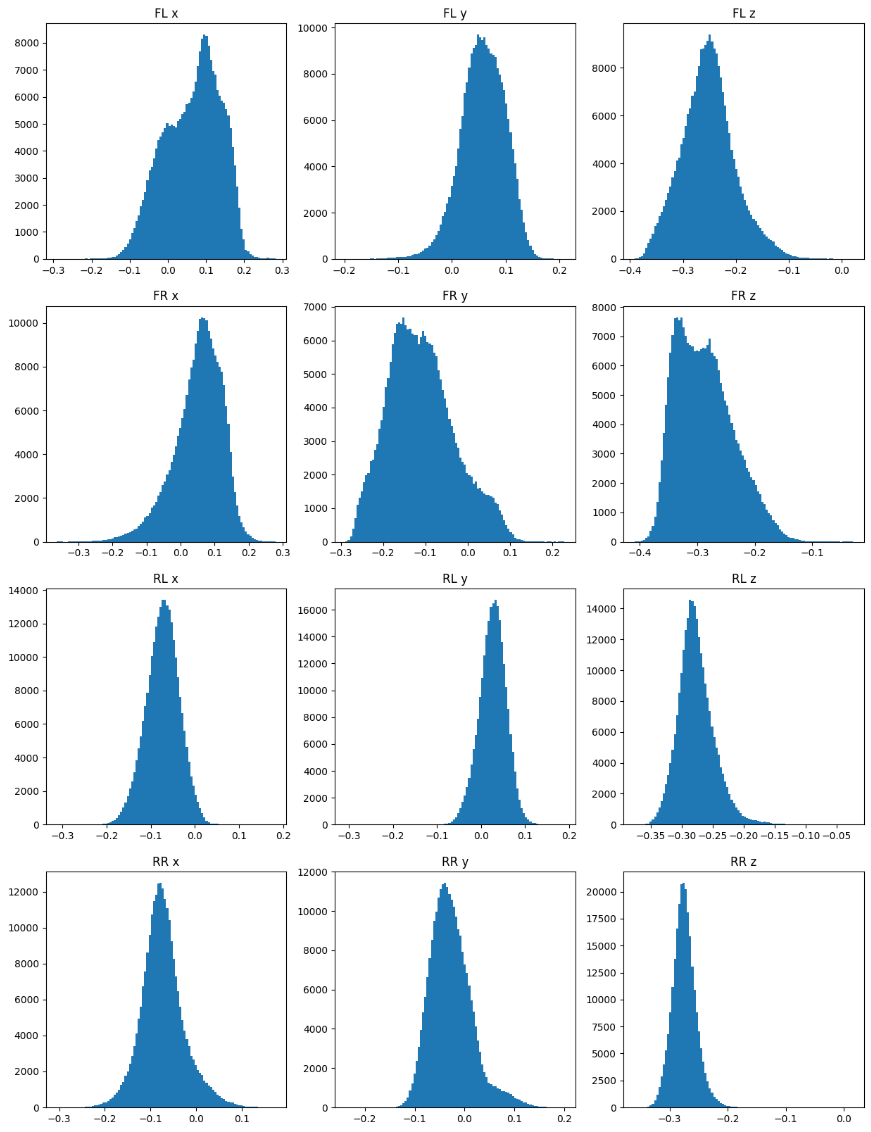


Figure B.4: Foot position values during an episode controlled by velocity baseline.

Bibliography

[1] P. Liu, H. Bou-Ammar, J. Peters, and D. Tateo, «Safe reinforcement learning on the constraint manifold: Theory and applications», *IEEE Transactions on Robotics*, 2025.

[2] P. Liu, K. Zhang, D. Tateo, *et al.*, «Safe reinforcement learning of dynamic high-dimensional robotic tasks: Navigation, manipulation, interaction», in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9449–9456.

[3] 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*, PMLR, 2022, pp. 91–100.

[4] D. Rodriguez and S. Behnke, «Deepwalk: Omnidirectional bipedal gait by deep reinforcement learning», in *2021 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2021, pp. 3033–3039.

[5] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, «Blind bipedal stair traversal via sim-to-real reinforcement learning», *arXiv preprint arXiv:2105.08328*, 2021.

[6] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, «Rapid locomotion via reinforcement learning», *The International Journal of Robotics Research*, vol. 43, no. 4, pp. 572–587, 2024.

[7] K. Caluwaerts, A. Iscen, J. C. Kew, *et al.*, «Barkour: Benchmarking animal-level agility with quadruped robots», *arXiv preprint arXiv:2305.14654*, 2023.

[8] Z. Zhuang, Z. Fu, J. Wang, *et al.*, «Robot parkour learning», *arXiv preprint arXiv:2309.05665*, 2023.

[9] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, «Extreme parkour with legged robots, 2023», *URL https://arxiv. org/abs/2309.14341.[15] S. Kareer, N. Yokoyama, D. Batra, S. Ha, and J. Truong. Vinl: Visual navigation and locomotion over obstacles*, 2023.

[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*, vol. 7, no. 62, eabk2822, 2022.

[11] M. Aractingi, P.-A. Léziart, T. Flayols, J. Perez, T. Silander, and P. Souères, «Controlling the solo12 quadruped robot with deep reinforcement learning», *scientific Reports*, vol. 13, no. 1, p. 11 945, 2023.

[12] A. Kumar, Z. Fu, D. Pathak, and J. Malik, «Rma: Rapid motor adaptation for legged robots», *arXiv preprint arXiv:2107.04034*, 2021.

[13] G. B. Margolis and P. Agrawal, «Walk these ways: Tuning robot control for generalization with multiplicity of behavior», in *Conference on Robot Learning*, PMLR, 2023, pp. 22–31.

[14] S. Choi, G. Ji, J. Park, *et al.*, «Learning quadrupedal locomotion on deformable terrain», *Science Robotics*, vol. 8, no. 74, eade2256, 2023.

[15] L. Smith, I. Kostrikov, and S. Levine, «A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning», *arXiv preprint arXiv:2208.07860*, 2022.

[16] F. Jenelten, J. He, F. Farshidian, and M. Hutter, «Dtc: Deep tracking control», *Science Robotics*, vol. 9, no. 86, eadh5401, 2024.

[17] G. Feng, H. Zhang, Z. Li, *et al.*, «Genloco: Generalized locomotion controllers for quadrupedal robots», in *Conference on Robot Learning*, PMLR, 2023, pp. 1893–1903.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, «Proximal policy optimization algorithms», *arXiv preprint arXiv:1707.06347*, 2017.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, «Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor», in *International conference on machine learning*, Pmlr, 2018, pp. 1861–1870.

[20] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, «Sim-to-real transfer of robotic control with dynamics randomization», in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 3803–3810.

[21] L. Campanaro, S. Gangapurwala, W. Merkt, and I. Havoutis, «Learning and deploying robust locomotion policies with minimal dynamics randomization», in *6th Annual Learning for Dynamics & Control Conference*, PMLR, 2024, pp. 578–590.

[22] Y. Kim, H. Oh, J. Lee, *et al.*, «Not only rewards but also constraints: Applications on legged robot locomotion», *IEEE Transactions on Robotics*, 2024.

[23] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, «Control barrier functions: Theory and applications», in *2019 18th European control conference (ECC)*, Ieee, 2019, pp. 3420–3431.

[24] S. Gangapurwala, A. Mitchell, and I. Havoutis, «Guided constrained policy optimization for dynamic quadrupedal robot locomotion», *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3642–3649, 2020.

[25] C. Xuan, F. Zhang, F. Yin, and H.-K. Lam, «Constrained proximal policy optimization», *arXiv preprint arXiv:2305.14216*, 2023.

[26] E. Chane-Sane, P.-A. Leziart, T. Flayols, O. Stasse, P. Souères, and N. Mansard, «Cat: Constraints as terminations for legged locomotion reinforcement learning», in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2024, pp. 13 303–13 310.

[27] J. Achiam, D. Held, A. Tamar, and P. Abbeel, «Constrained policy optimization», in *International conference on machine learning*, PMLR, 2017, pp. 22–31.

[28] Y. Liu, J. Ding, and X. Liu, «Ipo: Interior-point policy optimization under constraints», in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 4940–4947.

[29] T.-Y. Yang, T. Zhang, L. Luu, S. Ha, J. Tan, and W. Yu, «Safe reinforcement learning for legged locomotion», in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 2454–2461.

[30] L. Smith, Y. Cao, and S. Levine, «Grow your limits: Continuous improvement with real-world rl for robotic locomotion», in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 10 829–10 836.

[31] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi, «Agile but safe: Learning collision-free high-speed legged locomotion», *arXiv preprint arXiv:2401.17583*, 2024.

[32] J. Sola, J. Deray, and D. Atchuthan, «A micro lie theory for state estimation in robotics», *arXiv preprint arXiv:1812.01537*, 2018.

[33] V. Makoviychuk, L. Wawrzyniak, Y. Guo, *et al.*, «Isaac gym: High performance gpu-based physics simulation for robot learning», *arXiv preprint arXiv:2108.10470*, 2021.

[34] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters, «Mushroomrl: Simplifying reinforcement learning research», *Journal of Machine Learning Research*, vol. 22, no. 131, pp. 1–5, 2021. [Online]. Available: `http://jmlr.org/papers/v22/18-056.html`.