UNIVERSITÀ DI TRENTO

PhD programme in Industrial Innovation

38th Cycle

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
DEPARTMENT OF INDUSTRIAL ENGINEERING
FONDAZIONE BRUNO KESSLER

Reinforcement Learning-Based Control Strategies for a Ballbot System

Giulia Buzzetti

Tutor: Prof. Giovanni Iacca

Co-tutor: Dr. Davide Zappetti

Coordinator: Siracusa Domenico

ACADEMIC YEAR 2025-2026

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
UNIVERSITÀ DI TRENTO

Abstract

Dynamically unstable robots operating in human-centered environments must achieve not only high performance but also robustness to failures such as loss of balance and falls. While much research has focused on preventing such failures, comparatively less attention has been devoted to controlling their consequences, particularly in terms of minimizing damage to the robot during a fall. This challenge is especially pronounced for ballbots, which balance on a single spherical wheel and exhibit inherent dynamic instability combined with limited mechanical means to absorb impacts.

This dissertation investigates reinforcement learning–based control strategies for improving the resilience of ballbot robots, with a focus on damage minimization during falls and stable balancing behavior. First, the influence of reward function design and the ratio between controller and simulator frequencies is systematically analyzed, revealing their critical role in achieving stable learning and reducing impact forces across varying fall scenarios. Second, a multi-policy learning approach based on clustering is introduced to address high variability in initial configurations, demonstrating improved robustness and damage reduction compared to single-policy and curriculum learning baselines. Finally, the impact of modeling choices on reinforcement-learning-based ballbot balancing is examined through a comparative study of different simulation models, highlighting how model fidelity affects learning stability and control performance.

Overall, this work provides empirical insights into the interplay between learning configuration, policy structure, and modeling assumptions in reinforcement learning for dynamically unstable robots. The findings contribute to the design of safer and more reliable learning-based control strategies for ballbots and offer guidance applicable to a broader class of unstable robotic platforms.

Keywords

Ballbot robotics, Fall damage minimization, Multi-policy learning, Dynamic balancing

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of this dissertation is to explore the potential of reinforcement learning when applied to ballbot robots and to investigate why this combination is of interest both from a practical and an academic perspective.

Ballbots are particularly well-suited for operation in social environments. They are highly compliant, have a small footprint, and can move efficiently in narrow and cluttered spaces. Additionally, ballbots can be easily moved or pushed in any direction without immediately falling or tipping over, thanks to their dynamically stable equilibrium. Considering the increasing demand for robots in social and human-centered environments—especially in light of global population growth—ballbots represent a promising robotic platform.

From an academic perspective, the ballbot is an especially interesting case study because it operates around a dynamic equilibrium. Most of the existing literature on ballbots focuses on classical control methods, such as Linear Quadratic Regulator (LQR), Proportional–Integral–Derivative (PID) control, or hybrid approaches combining the two. However, due to the difficulty of accurately modeling the complex interactions between the spherical wheel, the ground, and the actuating wheels, it is natural to consider a model-free solution, namely reinforcement learning. Accurately describing these interactions is challenging because of strong nonlinearities, slipping effects, and frictional phenomena.

Using reinforcement learning for ballbot control, this dissertation focuses on two fundamental conditions that are intrinsic to all dynamically unstable robots: what happens when the robot falls, and what happens when it does not.

Specifically, this work is guided by the following research questions. First, how do reward function design choices and the ratio between controller and simulator frequencies influence training aimed at minimizing the damage of a falling ballbot? Second, to what extent can a multi-policy reinforcement learning approach improve robustness and performance in highly variable fall scenarios compared to single-policy solutions? Finally, how do different modeling choices affect the stability and performance of reinforcement-learning-based ballbot balancing controllers?

The first two research questions address ballbot resilience in fall scenarios. In this part of the work, we investigate how to increase robot resilience in the event of a

fall. Interestingly, we found that strategies that have been successful for other types of robots in the literature are not sufficient for ballbots. As a result, we propose two different frameworks aimed at improving fall resilience and demonstrate that the proposed methods are capable of reducing damage to the robot during falls.

Finally, the third research question focuses on reinforcement learning–based solutions for ballbot balancing. The findings in this case are particularly significant: the linear model widely used in the literature to simulate ballbots [42, 23, 13, 61, 46] fails to produce stable behavior when combined with reinforcement learning methods. This result highlights the need for a new paradigm in modeling choices when applying model-free learning approaches to ballbot control.

The modeling and experimental work presented in this dissertation is based on Mirokaï, the ballbot robots developed by Enchanted Tools. Mirokaï robots are innovative humanoid ballbots. Their anime-inspired, user-friendly design, combined with the intrinsic properties of ballbot locomotion, enables silent operation, high maneuverability, and a compact footprint, allowing effective navigation in cluttered environments. Their humanoid morphology further enables object grasping and manipulation. In addition, interaction with users is supported through an LLM-based communication interface. These combined characteristics make the platform particularly well-suited for operation in social environments.

The remainder of this dissertation is organized as follows. Chapter 2 introduces the background concepts in ballbot and humanoid robotics, reinforcement learning, and relevant control methodologies. Chapter 3 investigates various reinforcement learning configurations for minimizing damage during ballbot falls. Chapter 4 extends this analysis by introducing a multi-policy learning approach to address high variability in fall scenarios. Chapter 5 examines the impact of modeling choices on reinforcement-learning-based ballbot balancing. Finally, Chapter 6 concludes the dissertation and outlines directions for future research.

Chapter 2

Background

In this chapter, we introduce the background concepts necessary to contextualize the contributions of this dissertation. The chapter first introduces ballbot and humanoid robotic platforms, reviewing their technical foundations, historical development, and significance in scenarios involving interaction with humans.

The discussion includes a focused presentation of Mirokai, the humanoid robot developed by Enchanted Tools[1] that was the platform of choice for this work, and addresses perceptual considerations such as the uncanny valley.

The chapter then reviews reinforcement learning as a decision-making framework, highlights the transition to deep reinforcement learning, and presents Proximal Policy Optimization (PPO), the reinforcement algorithm used in the experimental work, in detail.

2.1 Ballbot and humanoid robots

In the following section, we present the main characteristics of ballbot systems and humanoid robots, discussing their respective advantages and limitations, and explaining their relevance to human–robot interaction.

2.1.1 Development of ballbot systems

Robotic systems have attracted increasing interest in recent years. Advances in this sector have accelerated the deployment of robots across a wide range of applications, both in industrial settings and in service-oriented domains such as the hospitality sector [22].

Many industrial robots are characterized by a fixed base and are not designed to directly interact with people. Conversely, mobile robots are often intended to operate in close proximity to humans and to interact with them [52]. This is one of the reasons for their growing popularity. A first distinction among mobile robots can be made between legged robots and wheeled robots. Wheeled robots offer several advantages:

[1] https://enchanted.tools/

they are less prone to oscillation while moving, they are faster on flat terrain, and they are generally quieter [1, 18, 4]. Among wheeled robots, ballbots exhibit specific characteristics that make them particularly suitable for social environments.

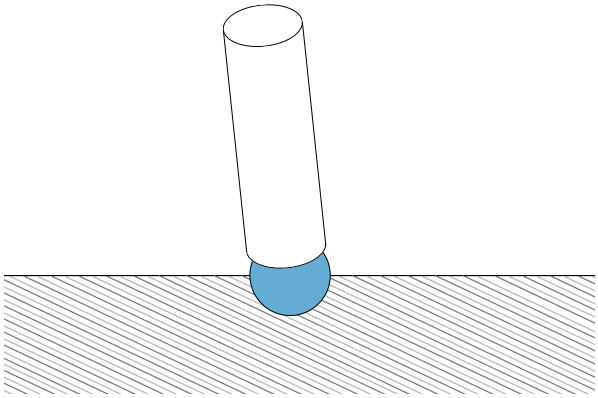


Figure 2.1: Schematic representation of a ballbot robot, where a rigid body is dynamically balanced on a single sphere in contact with the ground.

In general terms, a ballbot is a robot that balances on a ball. A generic figure of a ballbot is shown in Figure 2.1. The main characteristics of a ballbot are the body, usually modelled as a cylinder, and the sphere, or rolling globe, on which the body is placed.

Over the past years, different ballbot models have been proposed. The first prototype was proposed by Lauwers et al. [29]. Their work was motivated by the fact that statically stable wheeled robots can become dynamically unstable. Their aim was to develop a robot that was tall enough to interact with people, move in narrow and cluttered environments in an agile way, and be compliant when pushed and moved around. These kinds of characteristics can be achieved with dynamically stable robots, and, specifically, ballbots.

Their prototype is based on an inverse mouse ball mechanism combined with an inverse pendulum. This means that the rolling globe is driven by the inverse mouse ball drive mechanism. The ball is moved by four rollers positioned at the equator, with two motors actuating diametrically opposite rollers. To balance and move around, the dynamics are those of an inverted pendulum. The robot is characterized by a dynamic equilibrium and has to move continuously in order to maintain balance. The first prototype of this system was later refined in [40].

A more refined model was developed by Kumagai and Ochiai [27]. In this prototype, the ball is actuated via three omniwheels with a single line of rollers, positioned at intervals of 120 [deg] between each other and at an angle of 45 [deg] with respect to the upright z-axis of the ball. This prototype has been proposed both with a tall cylindrical body of about 1.3 [m] and a shorter cylindrical body of about 0.5 [m], which is more suitable for transporting weight on top of it. This kind of ballbot model is able to maintain a fixed position, as well as to move on flat terrain and to spin around its z-axis.

Another ballbot model was proposed by Fankhauser and Gwerder [13]. Similarly to [27], in this case the main body balances on a sphere thanks to three omniwheels with a single line of rollers, positioned at a distance of 120 [deg] from each other and at an angle of 45 [deg] with respect to the upright z-axis. An important novelty introduced in this work is the presence of ball arresters. The ball arresters are counter-holders positioned diametrically opposite to each of the omniwheels, at an angle of 115 [deg] with respect to the upright z-axis. These counter-holders have the function of keeping the ballbot in contact with the ball and augmenting the contact force between the omniwheels and the ball. They are designed in a way that makes it possible to incorporate small sensors to detect the actual movement of the globe.

A later ballbot model, with design and characteristics similar to those presented in [13] and [27], was proposed by Jespersen [23].

Beyond ballbots, several robotic systems inspired by spherical locomotion have also been proposed, taking advantage of the peculiarity of balancing on a single globe. Endo and Nakamura [12] proposed a prototype, similar to a wheelchair, that balances on a single spherical wheel. More recently, Pham et al. [46] proposed an innovative vehicle, called a ball Segway, to transport people. This vehicle has the peculiarity of balancing on a single ball while allowing people to climb onto it and move around in a manner similar to a Segway. The motivation behind these two kinds of devices is that, thanks to the use of a single ball, it is possible to move omnidirectionally and change direction immediately without reorienting the body, unlike what usually happens with conventional wheeled robots. Moreover, the use of a single wheel also provides a smaller footprint.

2.1.2 Humanoid robots

Humanoid robots are defined as robotic systems that exhibit characteristics resembling those of humans [56]. These characteristics range from facial features to overall bodily structure, such as possessing two arms and two legs.

Over the past decades, humanoid robotics has undergone significant development. From the early humanoid robot prototypes proposed in the 1980s by Kato et al. [24] to modern systems such as Atlas developed by Boston Dynamics[2], the technological gap is substantial in terms of both features and capabilities. Moreover, contemporary humanoid robots are increasingly equipped with large language models (LLMs), enabling advanced social interaction and communication abilities.

A key advantage of humanoid robots lies in their ability to be seamlessly integrated into existing human-centered infrastructures without requiring significant environmental modifications [56]. This property makes them particularly well-suited for a wide range of application domains, including hospitality, healthcare, warehousing, and agriculture [37]. In addition, humanoid robots are well-suited for performing high-risk tasks, such as search and rescue operations. Considering the overall growth of the aging population, it becomes increasingly important to develop approaches for the seamless

[2]Atlas by Boston Dynamics, https://bostondynamics.com/atlas

introduction of robots across broad work environments and to ensure that human–robot interaction remains intuitive and accessible [14].

A fundamental paradigm in human-robot interactions, which refers to how humans perceive robots, is the so-called *uncanny valley phenomenon* [38]. According to this principle, humans tend to experience feelings of discomfort or unease when encountering entities that closely resemble human beings but are not fully human. In particular, as an artificial agent's appearance becomes increasingly human-like without achieving complete realism, negative emotional responses tend to intensify. This effect helps explain why animated characters are often designed as stylized caricatures rather than highly realistic human replicas. Such design choices are typically intentional and not driven by technical limitations, but rather by the desire to avoid eliciting uncomfortable reactions from the audience.

Despite significant technological advances, several challenges remain to be addressed in order to achieve seamless and natural interaction between humans and humanoid robots. These challenges include fundamental issues such as balance control, energy efficiency, and overall robustness, which are critical for reliable operation in real-world environments.

The French robotics company Enchanted Tools has recently developed a humanoid robot that combines key characteristics of humanoid robots and ballbots. By adopting a ballbot locomotion system, the robot benefits from dynamic equilibrium and a small footprint, making it particularly suitable for social environments. These properties enable safe and efficient navigation in narrow spaces, as well as physical compliance when interacting with humans or being repositioned.

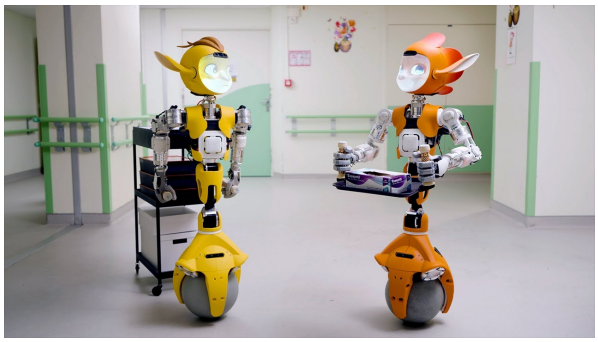


Figure 2.2: The first two prototypes of humanoid ballbot robots developed by Enchanted Tools: Miroki and Miroka.

In addition to its locomotion capabilities, this robot is equipped with two articulated arms that allow it to grasp and transport objects. It also integrates an LLM to support natural interaction with people. The robot's friendly and non-threatening design is intentionally crafted to elicit positive emotional responses, and user studies have shown favorable feedback from both elderly users and children. These characteristics make the platform especially well-suited for applications in hospitality and caregiving contexts.

Figure 2.2 shows the first two prototype robots developed by the company, dubbed Miroki and Miroka.

2.2 Reinforcement learning

Reinforcement learning (RL) [54] is a class of machine learning algorithms in which an agent learns by interacting directly with an environment. These algorithms have proven highly effective in solving complex sequential decision-making problems across a wide range of domains, including games and simulations such as chess and Atari games [3], robotics and control [25], and applications in finance and economics [7]. Their strength lies in the agent's ability to learn optimal behavior through trial-and-error interactions with the environment, using feedback from observed outcomes to guide future decisions.

This section presents the fundamental concepts of RL and describes the technical details of the algorithm employed in this work.

2.2.1 General concepts

In RL, the agent learns the best strategy to perform a task in a similar way to how humans and animals do: by interacting with the environment, it selects the best action to perform in a particular situation.

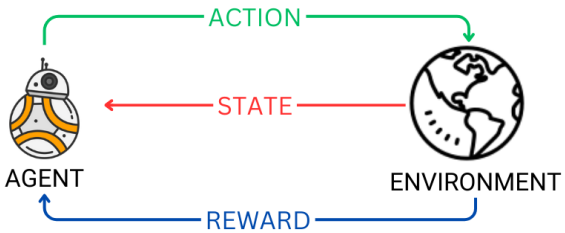


Figure 2.3: The Reinforcement Learning framework: the actor interacts with the environment.

The core idea behind RL is illustrated in Figure 2.3. As can be seen, the RL framework consists of two main components: the agent and the environment. The agent is the decision-making entity that selects the action to perform, while the environment is the external system with which the agent interacts and that governs the evolution of the system state.

The RL interaction is organized as a sequential loop. At each timestep $r(t)$, the agent performs an action. Given this action, the environment updates the system and returns to the agent the current state of the system and a reward related to the task the agent aims to accomplish. The action $a(t) \in \mathcal{A}$ is defined as one of the possible actions the agent can perform. The action space $\mathcal{A}$ may be either continuous or discrete, depending on the nature of the task. The state of the system $s(t)$ is defined as the information about the environment that is accessible to the agent at time t. The state may provide a complete description of the environment or contain only partial information. This reflects the fact that in many real-life applications, not all relevant information is available to the decision-making entity, such as in financial markets or robotic control processes, where sensor readings may be noisy, incomplete, or unavailable due to sensing limitations.

The reward function $r(t)$ is defined as a scalar function that guides the agent toward its goal. It provides a mathematical formulation of the task to be accomplished and evaluates the quality of the agent's actions. By maximizing the reward function, the agent is encouraged to learn a behavior that achieves the desired objective. The reward function must therefore be designed such that its maximization leads to accomplishing the task in an efficient manner. At each timestep t, the agent receives the state of the system $s(t)$ and the reward $r(t)$, and selects an action $a(t)$. The environment then evolves the system and returns the next state $s(t+1)$ together with the associated reward $r(t+1)$. This interaction loop continues until the system reaches a terminal state $s(T)$. The sequence of states, actions, and rewards from the initial state to the terminal state is referred to as an *episode*.

We define the policy $\pi_\theta(a(t)|s(t))$ as the function that gives the behavior of the agent. It represents the probability of choosing an action $a(t)$ given the state $s(t)$. The policy $\pi_\theta(a(t)|s(t))$ defines the behavior of the agent by specifying the probability of selecting action $a(t)$ given the current state $s(t)$, where θ denotes the parameters of the policy.

It is important to define the total reward accumulated over an episode. The total reward R is defined as the sum of the rewards $r(t)$ received at each timestep t, from the initial state to the terminal state:

$$R = r(1) + r(2) + \cdots + r(T). \tag{2.1}$$

At each timestep t, the total future reward $R(t)$ is defined as

$$R(t) = r(t) + r(t+1) + \cdots + r(T) = r(t) + R(t+1). \tag{2.2}$$

Since the reward function guides the agent toward accomplishing the task, the objective of the agent is to select, at each timestep, the action that maximizes the total future reward $R(t)$, thereby completing the task in the most efficient manner. To achieve this, the agent must estimate the expected value of $R(t)$ given the current state and action.

It is also possible to define the total discounted reward as a modification of the total reward that introduces a discount factor to account for the relative importance of future rewards. The discount factor reflects the fact that rewards received in the distant future are often less influential than immediate rewards when evaluating the quality of an action. Let $\gamma \in (0, 1)$ denote the discount factor, and define the total discounted future return $R_\gamma(t)$ as:

$$R_\gamma(t) = r(t) + \gamma R_\gamma(t+1). \tag{2.3}$$

From this definition, it follows that when $\gamma = 0$, the total discounted future return reduces to the immediate reward $r(t)$, whereas when $\gamma = 1$, it coincides with the total future return. The discount factor γ therefore controls the trade-off between prioritizing immediate rewards and accounting for long-term outcomes.

The action-value function $\mathcal{Q}(s(t), a(t))$, also referred to as the $\mathcal{Q}$-value, is defined as the expected total future reward obtained by taking action $a(t)$ in state $s(t)$ and following a given policy thereafter. Due to this definition, the optimal action at time t

is the one that maximizes the action-value function:

$$a^* = \arg\max_{a \in \mathcal{A}} \mathcal{Q}(s(t), a). \tag{2.4}$$

At the beginning of the learning process, the $\mathcal{Q}$-values are unknown and must be estimated through interaction with the environment. As a result, action selection typically has a stochastic component and depends on the current state of the system. Under these conditions, the system evolution can be modeled as a Markov process [44]. When control through action selection is considered, the interaction is more precisely described as a Markov Decision Process (MDP) with a finite horizon, since the process terminates upon reaching a terminal state at timestep T.

Classical RL algorithms can be broadly divided into two main categories: on-policy and off-policy methods. In on-policy algorithms, the agent learns and updates the parameters of the policy that is used to generate actions. In contrast, off-policy algorithms learn a policy indirectly by estimating a value function, while the behavior used to explore the environment may differ from the learned policy.

A classical off-policy algorithm is Q-learning [58]. In this approach, the action-value function is represented by a matrix Q, which is typically initialized to zero. During each episode, the agent updates the entries of the Q matrix based on the observed rewards and state transitions experienced while interacting with the environment. Action selection is governed by a trade-off between exploration and exploitation, commonly implemented through an ϵ-greedy strategy, where exploration decreases as the Q values are progressively refined.

Q-learning has been shown to be effective in a variety of problems with discrete and low-dimensional state spaces, such as the Grid World navigation task [2]. However, its reliance on a tabular representation limits its applicability to problems with large or continuous state spaces, where storing and updating the $\mathcal{Q}$ matrix becomes infeasible. For this reason, Deep Reinforcement Learning methods, which approximate value functions using neural networks, have proven successful in addressing such high-dimensional tasks. We briefly discuss this class of RL methods in the next section.

2.2.2 Deep reinforcement learning

As discussed previously, classical RL algorithms based on tabular representations become infeasible as the size of the state space increases. To address this limitation, deep reinforcement learning (deep RL) methods approximate value functions or policies using deep neural networks.

Similar to classical RL, deep RL algorithms can be broadly categorized into off-policy and on-policy methods. In off-policy algorithms, the tabular action-value function is replaced by a deep neural network that takes the state as input and outputs an estimate of the $\mathcal{Q}$-values for all possible actions. This approach enables learning in environments with large or continuous state spaces.

Conversely, on-policy algorithms directly parameterize and optimize the policy. The

objective is to maximize the expected value of the total future reward, defined as:

$$J(\theta) = \mathbf{E}[R(t)], \tag{2.5}$$

where θ denotes the parameters of the policy.

A classic example of an on-policy algorithm is the REINFORCE method [60]. At the beginning of training, the policy network parameters are randomly initialized. A number N of episodes is then generated by following the current policy, and the sequences of states, actions, and rewards are recorded. For each episode, the objective function $J(\theta)$ is computed using the total discounted reward, and the policy parameters are updated via gradient ascent. This process is repeated until convergence or until a predefined performance criterion is met.

Finally, there exists a class of algorithms that bridges the gap between on-policy and off-policy methods, known as actor–critic models [26]. In these architectures, two distinct neural networks are employed. The actor network represents the policy and is updated using on-policy learning, while the critic network estimates a value function, typically related to the $\mathcal{Q}$-value function, and can be trained in an off-policy manner. The critic provides an estimate of the expected return that is used to guide the actor's updates. This structure reduces the variance of the policy gradient estimates and generally leads to improved training stability and performance.

2.2.3 Proximal policy optimization

The objective in policy-based RL is to maximize the expected return

$$J(\theta) = \mathbf{E}[R(t)], \tag{2.6}$$

where θ denotes the parameters of the policy. According to the policy gradient theorem, maximizing this objective is equivalent to maximizing a surrogate objective with the same gradient, which enables gradient-based optimization of the policy parameters. In particular, the gradient of $J(\theta)$ can be expressed as

$$J(\theta) = \mathbf{E}\left[R(t), \log \pi_\theta(a(t) \mid s(t))\right], \tag{2.7}$$

and the policy parameters are updated using gradient ascent.

In practice, the return $R(t)$ is often replaced by an advantage function $A(t)$, which measures how favorable a given action $a(t)$ is compared to the average behavior of the policy in state $s(t)$. This substitution reduces variance while preserving an unbiased gradient estimate.

A key challenge in policy gradient methods is determining how much the policy should be updated at each iteration. Large updates may lead to instability or performance collapse, while overly small updates result in slow learning. Proximal Policy Optimization (PPO) [51] addresses this issue by constraining policy updates in a principled manner through a modified objective function.

Instead of directly optimizing $log\ \pi_\theta(a(t)|s(t))$, PPO introduces the probability ratio

$r_t(\theta)$ defined as

$$r_t(\theta) = \frac{\pi_\theta(a(t)|s(t))}{\pi_{\theta_{old}}(a(t)|s(t))}, \tag{2.8}$$

where $\pi_{\theta_{old}}$ denotes the policy before the update. This ratio measures how the probability of selecting action $a(t)$ under the new policy compares to that under the old policy. By construction, $r_t(\theta_{old}) = 1$; however, without constraints, $r_t(\theta)$ may take arbitrarily large or small values, leading to unstable updates.

To ensure that policy updates remain within a safe region, PPO restricts the probability ratio to the interval $[1-\epsilon, 1+\epsilon]$, where $\epsilon > 0$ is a small hyperparameter, typically chosen in the range $0.1 - 0.2$. The final PPO objective function is then defined as

$$J(\theta) = \mathbf{E}\left[\min\left(r_t(\theta)A(t), \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)A(t)\right)\right], \tag{2.9}$$

which discourages excessively large policy updates while still allowing meaningful improvement. This clipped objective provides a balance between stability and learning efficiency and has been shown to yield strong empirical performance.

Chapter 3

Reinforcement Learning for Falling Ballbot Damage Reduction

In this chapter, the main research question addressed consists of analyzing how the reward function and the ratio between the controller and simulator frequencies impact the training of a policy aiming at the minimization of damage to a falling ballbot. Specifically, an exploration of various reinforcement learning-based control solutions is undertaken with the objective of enhancing robot resilience to falls. This topic is of relevance in the context of the extant literature on falling resilience in robotics.

The goal of reinforcement learning is to identify a strategy that will minimize the damage caused by the ballbot robot falling. In order to achieve this objective, five different configurations were considered, and a variety of approaches were attempted. As it is evident that alterations in configuration are attributable to variations in the positioning of the arms, different arm positions are considered in the experiments. Furthermore, different aspects are investigated, namely:

- comparison of sparse versus non-sparse reward functions;
- evaluation of different components in the reward function;
- analysis of the frequency of the controller and simulator.

The experimental evaluation demonstrates that, whilst the approaches in question were found to be effective in the uncomplicated case of arm position, the majority of them are ineffective when considering a range of different arm positions.

By analyzing these different approaches, a stable configuration is then identified in which a reduction in damage can be achieved in the various initial settings.

The content of this chapter is based on the following publication:
G. Buzzetti, D. Zappetti, G. Iacca, A Reinforcement Learning method to minimize the damage on a falling Ballbot, European Robotics Forum (ERF), Springer Proceedings in Advanced Robotics, Rimini, January 2025
DOI: https://doi.org/10.1007/978-3-031-76424-0_15.

3.1 Introduction

Robotic systems play a pivotal role in contemporary society, with applications spanning various sectors such as industry, healthcare, logistics, and catering. Their ubiquity is attributed to the reliability of robotic structures, surpassing human labor in diverse scenarios. The evolution of robotic systems has yielded robust and precise robots that increasingly operate in close proximity to people, including customers and operators.

Despite the sophisticated hardware and software constituting these robots, occasional failures, such as *falls*, can still happen. When falls occur, mitigating any possible damage (to the robot and the environment) is a critical challenge. This challenge is particularly pronounced for robots capable of locomotion tasks, such as humanoid, animal-like, or wheeled robots, which lack a fixed base [31].

Several strategies have been proposed to minimize damage during falls, especially for humanoid and animal robots. Fujiwara et al. [15] introduced a movement sequence, known as the UKEMI strategy, to reduce impact. RL strategies for hand protection during falls are explored in [28], while [48] presents a fall sequence minimizing joint and robot part damage. Recent work by Ma et al. [35] focuses on a legged robot with a robotic arm, implementing an RL strategy for damage reduction and recovery after a fall. To the best of our knowledge, no previous work has addressed the problem of minimizing damage to ballbots, which is the main focus of our work.

A ballbot, distinguished by a single large wheel or ball for locomotion, exhibits dynamic stability, making it suitable for agile movement in cluttered and narrow environments [41, 6]. However, maintaining balance during movement requires constant adjustments to avoid falls. Given their intended interaction in social environments, finding an *aware* (of the fall itself and the environment) control strategy for reducing damages derived from falls in ballbots is paramount.

In this chapter, we explore different RL strategies aimed at minimizing damage when a ballbot falls. We developed a deep RL framework to train the robot based on its experiences. Our methodology was implemented and tested on Miroki, a commercial robot developed by Enchanted Tools[1], featuring a rolling globe for agile movement, comanipulating arms, AI-based speech recognition, and expressive facial features (see Fig. 3.1). With the policy developed, we assess a 52% reduction of the peak contact force on all body parts.

The remainder of the chapter is organized as follows: in Section 3.2 we outline the proposed framework; in Section 3.3 we present the results of our approach; finally, the conclusions are presented in Section 3.4.

3.2 Methodology

This section describes the implementation of the deep RL framework built to minimize the damage to the Miroki robot. The robot is 115cm tall and weighs 30kg. It stands on a ball that is controlled through 3 wheels. This enables swift movement, and it is easy

[1] https://enchanted.tools

both to move and to transport. Due to its configuration, it has an unstable equilibrium. The body is attached to the ball and has two hands with 7 Degrees of Freedom (DOF) each. Three additional DOF are present in the neck, plus one additional DOF in each ear. In the experiments we present here, we decided to control only the arms of the robot, keeping all the other DOF as fixed joints. As we will see in Section 3.3, the specific configuration of Miroki indeed allows for reducing the damage on the body using only the arms.

Simulation environment

The robot underwent training through simulating falls in 8192 parallel (vectorized) environments[2] using Isaac Gym [36]. Due to the complexity of employing triangular meshes and force sensors, the entire pipeline could not be trained on the GPU. Consequently, the simulation ran on the CPU, while the RL part was trained on the GPU. The falling task involved controlling Miroki's arms in position, with the remaining Degrees of Freedom (DOF) serving as fixed joints. The seven actuated DOF of the arm, as well as the robot's structure and design, are depicted in Figure 3.1.

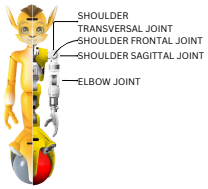


Figure 3.1: Front view of the ballbot developed by Enchanted Tools.

The RL algorithm employed was the Proximal Policy Optimization (PPO) method [51]. The neural network incorporates as input the actuated joint position and velocity, the body orientation, the linear and angular velocities, and the previous actions.The number of environments was chosen to balance the use of as many parallel environments as possible with maintaining efficient simulation speed. This value was determined empirically and set to 8192 parallel vectorized environments. Table 3.1 outlines the hyperparameters governing the training process.

In order to assess the capability of the policy to minimize damage to the robot, we considered four different starting positions, representing extreme configurations the robot may assume: both arms along the body, left arm bent, right arm bent, and both arms bent. In addition, we considered a mixed setup in which the starting configuration was randomly selected among these four at the beginning of each episode.

For each of the four fixed starting configurations, as well as for the mixed setup, the entire training procedure was repeated independently for each methodology under consideration. Specifically, a separate policy was trained for each combination of starting configuration and methodology. Within this experimental framework, we compared five different methodologies by evaluating alternative reward formulations and different

[2]See: https://github.com/NVIDIA-Omniverse/IsaacGymEnvs.

Table 3.1: Parameters used in the experimentation.

Parameter	**Value**
Max. episode length	5s
Controller frequency	50Hz
Mini-batch size	131072
KL threshold	1.6×10^{-2}
Horizon length	32
# environments	8192
# total training timesteps	786×10^6
# epochs	4
# hidden units	$256, 128, 64$

ratios between the simulation and controller frequencies. The details of the compared methods are presented in the following sections.

3.2.1 Baseline reward

The simulation and the controller run at the same frequency of $50Hz$. As stated in the previous section, a different policy was trained for each of the different arms configurations and for the mixed configuration.

For the reward function, two different rewards were considered: one given *during* the episode and one given just *at the end* of the episode. We refer to these rewards as *end-episode* (r_e) and *mid-episode* (r_m) rewards, respectively.

Before the episode ends, the *mid-episode* reward is computed at each timestep of the simulation. The goal of this reward is to achieve the final goal in the most efficient way possible, and it is related to the *minimization of the energy* spent while doing the movement to ensure that the movement is smooth and unnecessary movements are not performed. This reward has the following form:

$$r_m = \omega_v ||\dot{q}|| \tag{3.1}$$

where $||\dot{q}||$ denotes the Euclidean norm of the actuated joint angular velocities, and ω_v is a negative weight needed to minimize the actuated joints' angular velocity and therefore the energy. This reward resets at each timestep, and it is not cumulative through the steps of each episode, thereby avoiding an undesirable dependence on episode duration. It is important to notice that the mid-episode reward should be negligible when compared to the end-episode reward, since the maximization of the end-episode reward allows the accomplishment of the actual minimization of the damage to the robot. This is possible thanks to the weight parameter ω_v.

At the end of the episode, the *end-episode* reward r_e measures the *damage that the robot suffers from the fall*. Following previous works [28, 48], this reward takes into account both the damage to each body part and to each controlled joint. To do so, the

Table 3.2: Reward component weights used in the experimental evaluation of the baseline reward.

Reward component	**Weight**
End-episode: Body contact force (ω_b)	-0.005
End-episode: Joint yank (ω_j)	-0.05
Mid-episode: Velocity (ω_v)	-0.005

end-episode reward is given by the sum of the following quantities r_{eb} and r_{ej}:

$$r_{eb} = \omega_b \max_{\mathcal{T}} \sum_{b \in \mathcal{B}} ||cf_b(t)||^2 \tag{3.2}$$

$$r_{ej} = \omega_j \max_{\mathcal{T}} \sum_{i=1}^{N} ||df_i(t) - df_i(t-1)||^2 \tag{3.3}$$

where ω_b and ω_j are negative weights; $\mathcal{T}$ represents the set comprising all the timestamps that constitute an episode; $\mathcal{B}$ is the set of all the body parts; cf_b is the contact force acting on the b-th body part; N is the number of actuated DOF; and df_i represents the internal joint torque on the i-th DOF.

The component r_{eb} is proportional to the maximum contact force experienced by any body part during the episode, while r_{ej} is proportional to the joint yank experienced by all active joints during the fall. As shown in [28, 48], both terms are effective metrics for estimating damage to the robot. The first term captures the contact forces applied to the body during impact, enabling assessment of structural stress, whereas the second term reflects the forces transmitted to the joint actuators, which may fail under large variations between consecutive force values. The negative weights ωb and ωj ensure that maximizing the reward function corresponds to minimizing these quantities, thereby reducing overall damage to the robot.

We prioritize the minimization of the contact forces acting on the body with respect to the joint yank (i.e., given their relative scales, $|\omega_b| < |\omega_j|$) as, in general, it is more onerous to replace body parts rather than dealing with the damage occurred on joints. Table 3.2 displays the value of the weights utilized for each reward component.

3.2.2 Momentum-based reward

To improve the stability of the damage measurement, we investigated whether introducing an additional, independent quantity to quantify the damage sustained by the robot would lead to improved performance.

Starting from the previous reward configuration, we augmented the end-episode reward with an additional component aimed at more accurately capturing the damage experienced by the robot. This component is based on the momentum change; specifically, we consider the product of the mass and acceleration of each body part. At the instant of impact with the ground, the contact force acting on a body and its momen-

tum change are equivalent. Nevertheless, measuring these quantities independently can help reduce measurement errors. Moreover, large momentum changes over the course of an episode are undesirable, as they are indicative of unstable behavior.

The momentum change term is defined as follows:

$$r_{em} = \omega_m \max_{\mathcal{T}} \sum_{b \in \mathcal{B}} ||m_b a_b(t)||^2 \tag{3.4}$$

where m_b denotes the mass of the b-th body, $a_b(t)$ its acceleration at timestep t, and the remaining quantities are defined in equation 3.3. To compute this term, a simulated force sensor is attached to each body part of the robot.

The end-episode reward is then defined as the weighted sum:

$$r_e = r_{eb} + r_{ej} + r_{em} \tag{3.5}$$

where r_{eb} and r_{ej} are defined in equation 3.3. The weight values used for the reward components in this configuration are reported in Table 3.3.

Table 3.3: Reward component weights used in the experimental evaluation of the Momentum-based reward.

Reward component	**Weight**
End-episode: Body contact force (ω_b)	-0.005
End-episode: Joint yank (ω_j)	-0.05
End-episode: Momentum change (ω_m)	-0.00005
Mid-episode: Velocity (ω_v)	-0.005

3.2.3 Mean-based reward

To obtain a more accurate measurement of the damage experienced by the robot, we consider the mean of the maximum values of the contact forces acting on the robot and of the joint yank. Specifically, we compute the mean over five values, and we modify the computation of the reward component in equation 3.3 accordingly.

For each timestep $t \in \mathcal{T}$, we define the overall contact force $S_c(t)$ and the body yank $S_y(t)$ as follows:

$$S_c(t) = \sum_{b \in \mathcal{B}} \|cf_b(t)\|^2, \tag{3.6}$$

$$S_y(t) = \sum_{i=1}^{N} \left(df_i(t) - df_i(t-1)\right)^2. \tag{3.7}$$

Let the number of values used to compute the mean be $k = 5$. The choice of $k = 5$ was guided by an empirical evaluation of the peak contact force profiles recorded during

Table 3.4: Reward component weights used in the experimental evaluation of the Mean-based reward.

Reward component	**Weight**
End-episode: Body contact force (ω_b)	-0.005
End-episode: Joint yank (ω_j)	-0.05
Mid-episode: Velocity (ω_v)	-0.005

fall events. We observed that the largest force magnitudes consistently occurred within the first few impact instances. Averaging the top five maxima captured the dominant collision dynamics while reducing sensitivity to outliers. The reward components then take the following form:

$$r_{eb} = \frac{1}{k} \sum_{i=1}^{k} S_b^{(i)}, \tag{3.8}$$

$$r_{ej} = \frac{1}{k} \sum_{i=1}^{k} S_j^{(i)}. \tag{3.9}$$

The end reward r_e is given by the sum of these two components. As in the previous configuration, this reward is provided only at the end of the episode, while at each timestep the reward is given by the mid-episode reward r_m, defined in equation 3.1. Table 3.4 reports the weights used for the reward components. Note that these values are identical to those of the baseline configuration; this choice was made to ensure a fair comparison between experiments.

3.2.4 Dense reward

We aim to reduce the damage experienced by the robot during a fall. However, the magnitude of the damage can only be fully assessed once the fall has terminated. As a result, this task naturally involves a sparse or delayed reward, since the reward would be provided only at the end of the episode. Reinforcement learning with sparse rewards is more challenging compared to situations where a dense reward is available [10]. For this reason, we modify the reward formulation by introducing an incremental reward related to the accumulated damage at each timestep. Specifically, at each timestep t, the reward is defined as the maximum contact force and the maximum joint yank recorded up to that moment. Accordingly, for all $t \in \mathcal{T}$, the reward components $r_{eb}(t)$ and $r_{ej}(t)$ are computed as follows:

$$r_{eb}(t) = \omega_b \max_{\tau \leq t} \sum_{b \in \mathcal{B}} \|cf_b(\tau)\|^2, \; r_{ej}(t) \quad = \omega_j \max_{\tau \leq t} \sum_{i=1}^{N} \|df_i(\tau) - df_i(\tau - 1)\|^2 . \tag{3.10}$$

In this setting, the reward is provided at every timestep, and the mid-episode reward

used in the other configurations is not employed. The total reward is given by the sum of these two components. The component weights are kept identical to those used in the other experiments in order to ensure a fair comparison.

3.2.5 Controller frequency configuration

It is important to note that, due to the configuration of the ballbot, different arm positions strongly influence the location of the center of mass and the falling dynamics. Moreover, it has been shown in [17] that policies operating at lower control frequencies are more stable in systems exhibiting highly dynamic variations. We therefore investigate the effect of lower controller frequencies in solving the task.

We run the simulation at $200Hz$, while the controller operates at $25Hz$ and $12.5Hz$ for each of the four arm configurations mentioned above, as well as the mixed configuration. For each experiment, the controller frequency is fixed and remains constant across all episodes. The controller is velocity-based. At each control loop, the network outputs a velocity command for each joint. This velocity is converted into a target joint position, which is then decomposed into position increments according to the ratio between the controller frequency and the simulation frequency. The position increments are applied at each simulation step until the next controller update. This scheme is illustrated in Figure 3.2.

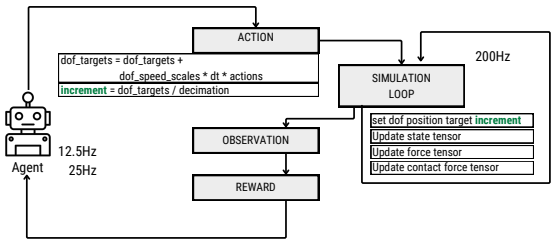


Figure 3.2: Controller–simulation decimation scheme. The simulation runs at 200 Hz, while the controller operates at a lower fixed frequency (either $12.5Hz$ or $25Hz$, depending on the experiment). At each controller update, the policy outputs joint velocity commands, which are integrated into target joint positions and converted into position increments. These increments are applied at each simulation step until the next controller update. The controller frequency remains constant for the entire duration of each experiment.

The reward structure and weights are the same as in the baseline configuration to enable a fair comparison of results. Specifically, the mid-episode reward r_m, defined in

3.1, is provided at each timestep, while the end-episode reward is given by the sum of r_{eb} and r_{ej}, as defined in 3.3. The weights are reported in Table 3.2.

3.3 Results

To evaluate each proposed approach, we compare its performance across the different arm configurations considered during training. Following [28], damage is quantified using the peak contact force along the z-direction over all body parts during the fall. To ensure a fair comparison, we perform 10 test runs for each setting and compute the following metrics:

- the maximum norm of the contact force over all body parts and all test environments;
- the maximum contact force along the z-axis in the environment exhibiting the highest contact force;
- the maximum contact force along the z-axis in the environment exhibiting the lowest contact force;
- the mean of the maximum contact force along the z-axis across all test environments;
- the 95-th percentile of the maximum contact force along the z-axis across all test environments.

All metrics are also computed for the Free Fall scenario, i.e., when the ballbot falls without any control policy. To assess damage reduction, we compute the percentage reduction P for each metric as

$$P = \frac{cf_{free} - cf_{policy}}{cf_{free}} \cdot 100, \tag{3.11}$$

where cf_{free} and cf_{policy} denote the corresponding metric values for free fall (i.e., no-policy) and policy-controlled fall, respectively.

The different arm configurations considered during training, as well as the mixed configuration, are analyzed separately in the following subsections. The contact force profiles for each body segment during the fall are reported in the Appendix A.

3.3.1 Arms along the body

We begin by assessing the performance of the different approaches in the scenario where the ballbot falls with both arms aligned along the body. The results are presented in Table 3.5.

The first notable observation concerns the Free Fall scenario. We observe a high variability in the maximum contact force between the environments exhibiting the highest and lowest contact forces. This variability arises from stochastic differences across

Table 3.5: Comparison of contact-force-related quantities for the scenario with both arms along the body across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$8.04\,10^3 N$	$8.04\,10^3 N$	$5.95\,10^3 N$	$7.26\,10^3 N$	$7.64\,10^3 N$
Baseline	$5.62\,10^3 N$	$5.62\,10^3 N$	$1.84\,10^3 N$	$3.40\,10^3 N$	$3.13\,10^3 N$
Reduction	30.06%	30.06%	69.01%	53.19%	58.99%
Momentum-Based	$4.42\,10^3 N$	$4.42\,10^3 N$	$2.42\,10^3 N$	$3.48\,10^3 N$	$3.59\,10^3 N$
Reduction	45.06%	45.06%	59.34%	52.01%	52.98%
Mean-Based	$1.02\,10^4 N$	$1.02\,10^4 N$	$2.09\,10^3 N$	$5.33\,10^3 N$	$5.63\,10^3 N$
Reduction	−27.25%	−27.25%	64.93%	26.64%	26.38%
Dense	$1.43\,10^4 N$	$1.43\,10^4 N$	$1.63\,10^3 N$	$4.89\,10^3 N$	$7.87\,10^3 N$
Reduction	−78.11%	−78.11%	72.55%	32.61%	−3.05%
Policy 25 Hz	$3.19\,10^3 N$	$3.19\,10^3 N$	$4.66\,10^2 N$	$2.10\,10^3 N$	$3.09\,10^3 N$
Reduction	60.36%	60.36%	92.16%	71.09%	59.59%
Policy 12.5 Hz	$2.53\,10^3 N$	$2.53\,10^3 N$	$1.05\,10^3 N$	$1.87\,10^3 N$	$1.67\,10^3 N$
Reduction	68.58%	68.58%	82.36%	74.25%	78.19%

simulation runs, as the simulation is not deterministic. On the other hand, the 95-th percentile and the mean of the maximum contact force are quite close to each other and are not far from the maximum contact force observed in the worst-case environment. Finally, it is worth noting that the mean and the maximum contact force in the environment with the highest contact force coincide. This is expected, since the contact force is predominantly aligned with the z-axis.

We observe that most of the proposed approaches successfully solve the task, achieving a reduction in the contact force F_z of approximately 60 % in the worst-case scenario and about 50 % on average across all environments, with the exception of the Mean-based and Dense approaches. While the Mean-based approach still achieves a reduction in contact forces—both in terms of the mean and the 95-th percentile—the Dense approach leads to worse behavior than the Free Fall scenario, resulting in higher contact forces. Overall, the approach that yields the greatest damage reduction is the one in which the controller operates at 12.5 Hz, achieving reductions of over 70% for both the mean and the 95-th percentile.

3.3.2 Left arm bent

In the left arm bent configuration, most of the proposed approaches degrade performance, with the introduction of the policy generally producing higher contact forces than the Free Fall case, as reported in Table 3.6. Specifically, for nearly all approaches, both the mean and the 95-th percentile of the contact force F_z increase when the policy is applied. The only exception is the controller operating at 12.5 Hz, which achieves a reduction in contact forces with respect to the no-policy case, yielding a decrease of approximately 55% in the mean and 43% in the 95-th percentile. These results indicate that, under this configuration, the effectiveness of the proposed methods is strongly limited, and meaningful damage reduction is obtained only when the policy is executed at the lowest control frequency.

Table 3.6: Comparison of contact-force-related quantities for the scenario with the left arm bent across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$5.29\,10^3 N$	$5.29\,10^3 N$	$4.05\,10^3 N$	$4.92\,10^3 N$	$5.25\,10^3 N$
Baseline	$8.82\,10^3 N$	$8.82\,10^3 N$	$2.62\,10^3 N$	$5.52\,10^3 N$	$6.63\,10^3 N$
Reduction	-66.79%	-66.79%	35.32%	-12.2%	-26.16%
Momentum-Based	$7.77\,10^3 N$	$7.77\,10^3 N$	$2.66\,10^3 N$	$5.73\,10^3 N$	$7.55\,10^3 N$
Reduction	-46.93%	-46.93%	34.25%	-16.52%	-43.8%
Mean-Based	$1.10\,10^4 N$	$1.10\,10^4 N$	$4.95\,10^2 N$	$5.36\,10^3 N$	$1.04\,10^4 N$
Reduction	-107.89%	-107.89%	87.77%	-8.89%	-98.82%
Dense	$5.36\,10^3 N$	$5.36\,10^3 N$	$1.69\,10^3 N$	$2.97\,10^3 N$	$3.68\,10^3 N$
Reduction	-1.31%	-1.31%	58.37%	39.55%	29.94%
Policy $25\,Hz$	$5.37\,10^3 N$	$5.37\,10^3 N$	$4.03\,10^3 N$	$4.71\,10^3 N$	$5.36\,10^3 N$
Reduction	-1.62%	-1.62%	0.59%	4.25%	-2.1%
Policy $12.5\,Hz$	$3.93\,10^3 N$	$3.93\,10^3 N$	$1.77\,10^3 N$	$2.21\,10^3 N$	$2.95\,10^3 N$
Reduction	25.75%	25.75%	56.31%	55.07%	43.85%

3.3.3 Right arm bent

Let us consider now the case of the right arm bent. Overall, the approaches exhibit better performance in this configuration than in the left arm bent case; however, most policies still fail to identify a strategy that yields a consistent reduction in the contact force along the z-axis. Interestingly, the Dense approach achieves a reduction of over 50 % in both the mean and the 95-th percentile of the contact force. The Baseline policy results in a reduction of approximately 15 % in the 95-th percentile, whereas the Mean-based approach leads to worse performance compared to the no-policy case. The most stable configurations are obtained at lower controller frequencies, which achieve reductions of over 30 % and 60 % in the mean and 95-th percentile, respectively. The results are summarized in Table 3.7.

Table 3.7: Comparison of contact-force-related quantities for the scenario with the right arm bent across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$4.55\,10^3 N$	$4.55\,10^3 N$	$2.37\,10^3 N$	$3.79\,10^3 N$	$4.49\,10^3 N$
Baseline	$4.50\,10^3 N$	$4.50\,10^3 N$	$7.19\,10^2 N$	$2.64\,10^3 N$	$3.81\,10^3 N$
Reduction	1.17%	1.17%	69.68%	30.22%	15.06%
Momentum-Based	$5.90\,10^3 N$	$5.90\,10^3 N$	$1.82\,10^3 N$	$3.49\,10^3 N$	$5.59\,10^3 N$
Reduction	-29.69%	-29.69%	23.13%	7.91%	-24.42%
Mean-Based	$7.59\,10^3 N$	$7.59\,10^3 N$	$1.30\,10^3 N$	$3.17\,10^3 N$	$4.64\,10^3 N$
Reduction	-66.93%	-66.93%	45.34%	16.33%	-3.45%
Dense	$2.47\,10^3 N$	$2.47\,10^3 N$	$8.40\,10^2 N$	$1.54\,10^3 N$	$1.94\,10^3 N$
Reduction	45.81%	45.81%	64.55%	59.36%	56.76%
Policy $25\,Hz$	$4.00\,10^3 N$	$4.00\,10^3 N$	$1.29\,10^3 N$	$2.40\,10^3 N$	$2.89\,10^3 N$
Reduction	12.11%	12.11%	45.58%	36.6%	35.63%
Policy $12.5\,Hz$	$1.70\,10^3 N$	$1.70\,10^3 N$	$1.01\,10^3 N$	$1.26\,10^3 N$	$1.21\,10^3 N$
Reduction	62.52%	62.52%	57.54%	66.83%	72.97%

Table 3.8: Comparison of contact-force-related quantities for the scenario with both arms bent across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$7.98\,10^3 N$	$7.98\,10^3 N$	$4.66\,10^3 N$	$5.22\,10^3 N$	$6.93\,10^3 N$
Baseline	$6.76\,10^3 N$	$6.76\,10^3 N$	$4.55\,10^3 N$	$5.04\,10^3 N$	$6.61\,10^3 N$
Reduction	15.3%	15.3%	2.39%	3.57%	4.63%
Momentum-Based	$1.35\,10^4 N$	$1.35\,10^4 N$	$7.46\,10^2 N$	$8.72\,10^3 N$	$1.32\,10^4 N$
Reduction	−69.46%	−69.46%	83.98%	−66.99%	−90.1%
Mean-Based	$1.86\,10^4 N$	$1.86\,10^4 N$	$5.25\,10^3 N$	$1.45\,10^4 N$	$1.83\,10^4 N$
Reduction	−132.74%	−132.74%	−12.68%	−177.86%	−164.13%
Dense	$1.27\,10^4 N$	$1.27\,10^4 N$	$1.73\,10^3 N$	$4.79\,10^3 N$	$1.06\,10^4 N$
Reduction	−59.11%	−59.11%	62.87%	8.3%	−53.72%
Policy $25\,Hz$	$2.11\,10^3 N$	$2.11\,10^3 N$	$7.38\,10^2 N$	$1.79\,10^3 N$	$2.10\,10^3 N$
Reduction	73.6%	73.6%	84.16%	65.81%	69.63%
Policy $12.5\,Hz$	$6.10\,10^3 N$	$6.10\,10^3 N$	$2.31\,10^3 N$	$4.42\,10^3 N$	$4.59\,10^3 N$
Reduction	23.6%	23.6%	50.41%	15.36%	33.71%

3.3.4 Both arm bent

In this case, the situation is largely unchanged. In the baseline configuration, the results are comparable to those obtained without the policy, with a marginal reduction of approximately 4 %. For the Momentum-based, Mean-based, and Dense approaches, the introduction of the policy leads to significantly worse performance than the Free Fall scenario; in particular, for the Mean-based approach, the contact forces are nearly doubled compared to free fall. Conversely, the policies operating at $25\,Hz$ and $12.5\,Hz$ achieve stable damage reduction. The results are summarized in Table 3.8.

3.3.5 Mixed configuration of the arms

We finally consider the scenario in which, during training, the robot is randomly initialized in one of the four configurations considered. The results confirm the overall trends observed in the individual configurations. In the case where both arms are aligned along the body, reported in Table 3.9, the policy generally exhibits improved performance, although it remains inferior to the case in which the policy is trained specifically on this configuration. Moreover, for the baseline and Momentum-based approaches, the policy does not yield a reduction in contact forces. Nevertheless, in this configuration, the resulting performance is still better than in the other scenarios, where contact forces can nearly double compared to the Free Fall case. This trend is consistently observed across all approaches. Overall, the performance is worse than that obtained when training the network on a single configuration. Finally, policies operating at lower control frequencies consistently achieve reductions in both the mean and the 95-th percentile of the contact force, with lower frequencies corresponding to more stable performance.

Table 3.9: Comparison of contact-force-related quantities for the mixed scenario tested on both arms along the body, across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$8.04\,10^3 N$	$8.04\,10^3 N$	$5.95\,10^3 N$	$7.26\,10^3 N$	$7.64\,10^3 N$
Baseline	$1.46\,10^4 N$	$1.46\,10^4 N$	$2.08\,10^3 N$	$6.63\,10^3 N$	$9.49\,10^3 N$
Reduction	-81.2%	-81.2%	64.98%	8.67%	-24.2%
Momentum-Based	$1.36\,10^4 N$	$1.36\,10^4 N$	$3.22\,10^3 N$	$6.48\,10^3 N$	$1.11\,10^4 N$
Reduction	-69.25%	-69.25%	45.89%	10.7%	-45.63%
Mean-Based	$1.23\,10^4 N$	$1.23\,10^4 N$	$1.56\,10^3 N$	$4.89\,10^3 N$	$6.79\,10^3 N$
Reduction	-53.47%	-53.47%	73.76%	32.66%	11.2%
Dense	$7.33\,10^3 N$	$7.33\,10^3 N$	$2.32\,10^3 N$	$4.55\,10^3 N$	$4.09\,10^3 N$
Reduction	8.8%	8.8%	61.09%	37.34%	46.46%
Policy $25\,Hz$	$3.93\,10^3 N$	$3.93\,10^3 N$	$2.14\,10^3 N$	$2.86\,10^3 N$	$2.60\,10^3 N$
Reduction	51.1%	51.1%	63.96%	60.54%	65.92%
Policy $12.5\,Hz$	$3.15\,10^3 N$	$3.15\,10^3 N$	$9.64\,10^2 N$	$1.88\,10^3 N$	$2.21\,10^3 N$
Reduction	60.77%	60.77%	83.8%	74.15%	71.14%

Table 3.10: Comparison of contact-force-related quantities for the mixed scenario tested with the left arm bent, across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$5.29\,10^3 N$	$5.29\,10^3 N$	$4.05\,10^3 N$	$4.92\,10^3 N$	$5.25\,10^3 N$
Baseline	$1.19\,10^4 N$	$1.19\,10^4 N$	$2.42\,10^3 N$	$5.52\,10^3 N$	$9.96\,10^3 N$
Reduction	-124.69%	-124.69%	40.22%	-12.27%	-89.65%
Momentum-Based	$5.48\,10^3 N$	$5.48\,10^3 N$	$2.67\,10^3 N$	$4.12\,10^3 N$	$5.33\,10^3 N$
Reduction	-3.65%	-3.65%	34.06%	16.35%	-1.49%
Mean-Based	$1.91\,10^4 N$	$1.91\,10^4 N$	$6.00\,10^3 N$	$1.35\,10^4 N$	$1.91\,10^4 N$
Reduction	-261.52%	-261.52%	-48.11%	-174.23%	-263.63%
Dense	$5.85\,10^3 N$	$5.85\,10^3 N$	$2.59\,10^3 N$	$4.22\,10^3 N$	$4.67\,10^3 N$
Reduction	-10.62%	-10.62%	36.1%	14.22%	11.03%
Policy $25\,Hz$	$5.08\,10^3 N$	$5.08\,10^3 N$	$1.90\,10^3 N$	$3.43\,10^3 N$	$3.55\,10^3 N$
Reduction	3.88%	3.88%	53.16%	30.22%	32.37%
Policy $12.5\,Hz$	$5.92\,10^3 N$	$5.92\,10^3 N$	$1.67\,10^3 N$	$3.38\,10^3 N$	$3.39\,10^3 N$
Reduction	-11.99%	-11.99%	58.87%	31.34%	35.53%

Table 3.11: Comparison of contact-force-related quantities for the mixed scenario tested with the right arm bent across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$4.55\,10^3 N$	$4.55\,10^3 N$	$2.37\,10^3 N$	$3.79\,10^3 N$	$4.49\,10^3 N$
Baseline	$9.31\,10^3 N$	$9.31\,10^3 N$	$2.44\,10^3 N$	$5.24\,10^3 N$	$8.68\,10^3 N$
Reduction	-104.63%	-104.63%	-3.07%	-38.24%	-93.45%
Momentum-Based	$6.88\,10^3 N$	$6.88\,10^3 N$	$2.06\,10^3 N$	$4.27\,10^3 N$	$6.31\,10^3 N$
Reduction	-51.32%	-51.32%	13.08%	-12.61%	-40.48%
Mean-Based	$7.34\,10^3 N$	$7.34\,10^3 N$	$1.58\,10^3 N$	$4.86\,10^3 N$	$6.73\,10^3 N$
Reduction	-61.32%	-61.32%	33.32%	-28.43%	-49.84%
Dense	$6.94\,10^3 N$	$6.94\,10^3 N$	$1.92\,10^3 N$	$4.59\,10^3 N$	$6.59\,10^3 N$
Reduction	-52.63%	-52.63%	18.94%	-21.22%	-46.77%
Policy $25\,Hz$	$4.58\,10^3 N$	$4.58\,10^3 N$	$2.08\,10^3 N$	$2.91\,10^3 N$	$3.18\,10^3 N$
Reduction	-0.71%	-0.71%	12.21%	23.12%	29.08%
Policy $12.5\,Hz$	$3.46\,10^3 N$	$3.46\,10^3 N$	$1.44\,10^3 N$	$1.92\,10^3 N$	$2.49\,10^3 N$
Reduction	24.0%	24.0%	39.16%	49.22%	44.54%

Table 3.12: Comparison of contact-force-related quantities for the mixed scenario tested with both arms bent across different approaches.

	Max $\|F\|$ (global)	Max F_z over max env	Max F_z over min env	Mean F_z over all env	95-th percentage F_z over all env
Free Fall	$7.98\,10^3 N$	$7.98\,10^3 N$	$4.66\,10^3 N$	$5.22\,10^3 N$	$6.93\,10^3 N$
Baseline	$1.51\,10^4 N$	$1.51\,10^4 N$	$1.18\,10^3 N$	$1.01\,10^4 N$	$1.43\,10^4 N$
Reduction	-88.75%	-88.75%	74.67%	-92.59%	-106.65%
Momentum-Based	$1.23\,10^4 N$	$1.23\,10^4 N$	$9.69\,10^2 N$	$7.67\,10^3 N$	$1.20\,10^4 N$
Reduction	-53.71%	-53.71%	79.19%	-46.92%	-72.93%
Mean-Based	$1.49\,10^4 N$	$1.49\,10^4 N$	$6.95\,10^2 N$	$4.64\,10^3 N$	$1.36\,10^4 N$
Reduction	-86.21%	-86.21%	85.08%	11.16%	-96.54%
Dense	$6.85\,10^3 N$	$6.85\,10^3 N$	$2.44\,10^3 N$	$4.19\,10^3 N$	$6.52\,10^3 N$
Reduction	14.24%	14.24%	47.54%	19.83%	5.89%
Policy $25\,Hz$	$6.14\,10^3 N$	$6.14\,10^3 N$	$3.95\,10^3 N$	$5.03\,10^3 N$	$5.61\,10^3 N$
Reduction	23.07%	23.07%	15.27%	3.63%	19.0%
Policy $12.5\,Hz$	$5.41\,10^3 N$	$5.41\,10^3 N$	$3.29\,10^3 N$	$4.29\,10^3 N$	$5.07\,10^3 N$
Reduction	32.28%	32.28%	29.33%	17.95%	26.84%

3.4 Conclusions

In this chapter, we investigated a deep reinforcement learning framework for reducing fall-induced damage in a ballbot by actively controlling its two arms. Several approaches were compared to evaluate the stability of the reward formulation and to identify control strategies that effectively minimize damage during falls. The results indicate that reducing the controller frequency is the most effective strategy, while modifications of the reward function do not consistently lead to improved stability or performance. In the most favorable scenarios, the learned policy achieves reductions from 40% to 70% in contact forces, compared to a free fall case, exhibiting behaviors reminiscent of human-inspired fall mitigation strategies.

Across the different configurations, policies operating at lower control frequencies consistently provided the most stable and reliable damage reduction, achieving reductions in both the mean and the 95-th percentile of the contact force. Conversely, several reinforcement learning approaches, such as Mean-based, Dense, and Momentum-based strategies, led to degraded performance or even increased contact forces in certain configurations, highlighting the importance of configuration awareness and control design. Training a single policy across multiple configurations improved generalization but resulted in reduced performance compared to policies trained on a specific configuration.

The study was conducted in a simulation environment due to the nature of the task. Testing the policy on a real robot would entail a significant risk of damaging multiple units, which is not sustainable from an industrial perspective given the limited number of available prototypes. At a later stage, it would be both feasible and valuable to evaluate the policy on a physical robot.

Future work will focus on improving the robustness and generalization capabilities of the proposed framework. In particular, this will involve hyperparameter tuning of the best-performing configuration, as well as the incorporation of domain randomization techniques to enhance sim-to-sim and sim-to-real transfer and reduce the reality gap. Additionally, alternative damage metrics will be investigated, accounting not only for the robot itself but also for its interaction with the surrounding environment. These directions aim to further enhance the applicability and reliability of learning-based strategies for fall damage mitigation in ballbots.

Chapter 4

A Multi-policy Approach Based on Clustering for Minimizing the Damage on a Falling Ballbot

In this chapter, our research question focuses on understanding the impact of a multi-policy approach on solving a high-variability task, where small variations in the initial configuration can lead to significantly different dynamic behaviors. Specifically, the present chapter is concerned with the enhancement of the falling resilience of a humanoid ballbot. Indeed, a potential limitation of the results presented in the previous chapter is that not all possible initial configurations have been considered.

In this chapter, an analysis and comparison of two different approaches is presented: one based on curriculum learning and one based on a multi-policy approach. The initial approach analyzed was that of a multi-policy one. The initial position is considered in its various potential forms, and these are then divided into groups by means of a clustering process. The training of a distinct network is undertaken for each cluster, subsequently resulting in the amalgamation of these networks into a multi-policy approach.

The second approach analyzed is the utilization of curriculum learning to address the identified problem. The curriculum was designed by progressively increasing task complexity through the inclusion of additional joints.

To establish a baseline for comparison with the proposed multi-policy and clustering approaches, two single-policy models were used: one incorporating clustering information and one without it. Comparing against these baselines allowed us to assess the behavior of the proposed methods. A comparison of the various approaches reveals that the multi-policy approach is associated with a greater reduction in damage to the robot.

The content of this chapter is based on the following publication:
G. Buzzetti, M. Aractingi, D. Zappetti, G. Iacca, A Multi-policy Approach Based on Clustering for Minimizing the Damage on a Falling Ballbot, International Conference on Control, Decision and Information Technologies (CoDIT), IEEE, Split, July 2025
DOI: https://doi.org/10.1109/CoDIT66093.2025.11321766.

4.1 Introduction

Legged robots have garnered significant attention and extensive exploration in recent years due to their ability to perform diverse tasks and their remarkable versatility. However, the capability for unrestricted and agile movement is accompanied by the inherent risk of falls, which pose both safety hazards and substantial costs. Addressing these challenges, particularly by mitigating damage during falls and enhancing the resilience of such robots, remains a critical area of focus. The existing body of literature reflects a variety of studies dedicated to addressing this challenge by enhancing control.

Early studies on fall mitigation in humanoids introduced a "safe landing" approach, such as Fujiwara et al. [15] "Ukemi" algorithm, designed to reduce impact via learned or optimized maneuvers. Similarly, Ha and Liu [20] proposed an optimal contact sequence to minimize impact force, demonstrating meaningful reductions in simulation and hardware tests. Goswami et al. [19] focused on controlling a robot's fall direction to avoid harming bystanders, while Samy et al. [50] devised a dual strategy that firstly computes impact points while avoiding obstacles and then uses Quadratic Programming to minimize the impact on those selected points. Reinforcement Learning (RL) methods have also been explored. Kumar et al. [28] developed a control method using RL to identify the optimal contact points to minimize impact force and to control joint actuation for safer falls. Ma et al. [35] proposed a damage-reduction and recovery strategy from falls, while Wang et al. [57] used RL to reduce damage and facilitate recovery post-fall.

Despite these advances, most existing approaches focus on legged robots, leaving ballbots (i.e., ball-balancing robots that perform locomotion via a spherical wheel) as a relatively underexplored area of research. Ballbots rely on a single ground-contact point and maintain dynamic equilibrium via continuous rolling. Due to their configuration, they exhibit exceptional compliance and agile movement, making them particularly suitable for social environments [6]. However, their dynamic equilibrium also poses unique fall-related challenges. Additionally, ballbots lack the ability to bend their bodies, making strategies such as squatting motions inapplicable.

In this chapter, we propose a novel RL-based approach designed explicitly for ballbots to mitigate fall-related damage. Specifically, our main contributions are:

- A novel, multi-policy RL-based fall mitigation strategy targeted at ballbots experiencing different types of falls (i.e., from different initial poses of the arms).
- A clustering methodology for the initial pose of the arms, enabling specialized learned policies for the diverse fall scenarios experienced by the ballbot.
- A comparative study of the proposed multi-policy approach against single-policy and curriculum RL baselines, demonstrating the benefits of clustering.

The remainder of the chapter is organized as follows. Section 4.2 introduces the background concepts, including RL, clustering, and multi-policy approaches. Section 4.3 details the experimental setup and the methods, including the tested RL approaches. Section 4.4 presents the results, and finally Section 4.5 concludes the chapter and discusses directions for future work.

4.2 Preliminaries

4.2.1 Reinforcement learning

In RL with continuous state and action spaces (which is how we model the problem at hand), an environment is defined by a Markov decision process (MDP) [54], characterized by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, P_0)$. $\mathcal{S}$ and $\mathcal{A}$ denote the set of continuous states and continuous actions, respectively. Performing an action $a \in \mathcal{A}$ in a state $s \in \mathcal{S}$ results in a reward defined by the function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to R$. The dynamics of the environment is captured by the conditional transition probability distribution $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$, where $\mathcal{T}(s, a, s') = p(s_{t+1} = s' \mid s_t = s, a_t = a)$ specifies the probability of transitioning to the state s' at timestep $t + 1$ given the current state s and the action a at timestep t. The initial state distribution is denoted by P_0.

Only the state space $\mathcal{S}$ and action space $\mathcal{A}$ of the MDP are assumed to be known to the learning agent. Through interactions with the environment, the agent collects samples from the reward function and state transition distributions.

We define a policy $\pi_\theta(s, a) = p_\theta(a_t = a \mid s_t = s)$, parameterized by θ, which gives the probability of selecting action a given the state. The objective is to optimize the parameters θ of the policy to maximize the expected discounted sum of rewards, defined as:

$$J(\theta) := E_{\pi_\theta, P_0, \mathcal{T}}\left[\sum_{t=0}^{H} \gamma^t \mathcal{R}(s_t, a_t)\right] \tag{4.1}$$

where H is the horizon of the episode (i.e., the number of timesteps) and $\gamma \in [0, 1]$ is the discount factor.

4.2.2 Clustering

In our scenario, the initial pose of the robot's arms plays a crucial role, as different poses can lead to markedly different fall dynamics. To develop a robust and reliable control strategy, we apply a clustering approach to the initial poses, grouping those with similar characteristics.

Clustering is an unsupervised learning method that partitions data into distinct groups based on shared features. Because it is unsupervised, the algorithm does not require prior knowledge or human labeling. Here, we employ the K-means algorithm, an iterative clustering method [34]. K-means represents each cluster by its centroid, and data points are assigned to their nearest centroid. The centroid positions are then recalculated based on the assigned points. This process is repeated until convergence. The overall objective is to minimize the within-cluster sum of squares, defined as:

$$WCSS = \sum_{k=1}^{N} W(C_k),$$

where $W(C_k)$ is the within-cluster variation for the cluster C_k, defined as:

$$W(C_k) = \sum_{x_i \in C_k} ||x_i - \mu_k||^2$$

where x_i is the i-th point of the cluster and μ_k is the centroid of the cluster C_k. K-means was chosen as the clustering algorithm because it has been shown to yield good results in identifying patterns in robot trajectories [32].

The quality of the cluster can be evaluated with the silhouette score metric. The silhouette score is a metric assigned to each dataset point that measures how similar the point is to the centroid. It is calculated as follows:

$$\frac{b_i - a_i}{\max\{a_i, b_i\}} \tag{4.2}$$

where a_i is the average distance to other points in the same cluster, and b_i is the average distance to points outside the cluster. The mean silhouette score across all points in the dataset provides an overall assessment of clustering quality. The score ranges from -1 to 1. A value close to -1 indicates that points may be assigned to the wrong cluster; whereas, values close to 1 signify well-separated clusters, and values near 0 suggest overlapping or ambiguous cluster boundaries.

4.2.3 Multi-policy approaches

Multi-policy approaches are often employed to develop control strategies, and they have demonstrated notable reliability, in particular in robotic tasks [21]. In various scenarios, they have also been combined with deep RL [43].

Here, we refer to multi-policy control when the policy π is composed of different policies π_i from a finite set of policies $\Pi = \{\pi_1, \ldots, \pi_n\}$ [16]. Each policy depends on the state of the system and on a policy parameterization that defines its domain of applicability. In our case, each policy $\pi_i \in \Pi$ is trained separately on initial states sampled from its corresponding cluster C_i, where each cluster refers to a specific initial pose of the robot arms.

Of note, a similar approach to ours was proposed in [32], where authors segmented the trajectory of a robotic manipulator and trained a different policy for each segment. As in our approach, they used the K-means algorithm to divide the data. In contrast to their method, we have different policies for different types of falls. Once we identify the appropriate cluster using the initial pose of the arms, we employ the corresponding policy for the entire duration of the task.

4.3 Methodology

4.3.1 Experimental setup

For our experiments, we used again the humanoid ballbot developed by the company Enchanted Tools[1] The robot is 115 cm tall, weighs approximately 30 kg, and is equipped with a head for human interaction and two arms with hands to grasp and carry items. It balances on a ball thanks to three omni-wheels that enable agile movements. Each arm has 7 Degrees of Freedom (DOF); in all experiments, we controlled only the 14 DOFs corresponding to the arms, while the other joints were treated as passive.

The control is velocity-based; the control's frequency is 12.5 Hz, while the simulation runs at 200 Hz. At each control step, the action network outputs the target speed for each of the 14 actuated joints. The speed is converted to the final position, and the positions are then divided into increments; at each simulation step, the position increment is applied to each joint.

We built our RL pipeline using Isaac Gym, which allows training on massively parallel vectorized environments [36]. The RL algorithm used to train the policies is PPO [51], which is a type of actor-critic algorithm [26]. We trained the policy on 4096 parallel environments to maximize the reward function designed in Section 4.3.2. A symmetric actor-critic model is employed, sharing the same network and observations for both the policy and value functions. The network is a multi-layer perceptron with three ELU [9] hidden layers of sizes [512, 256, 128]. Table 4.1 details the hyperparameters of PPO used in the experimentation.

Table 4.1: Hyperparameters used for PPO.

Hyperparameters	**Value**
Max. episode length	5 s
Controller frequency	12.5 Hz
No. of environments	4096
Mini-batch size	16384
KL threshold	8e−3
Steps per update	24

The observation vector consists of the position and velocity of each actuated joint, the quaternion representing the root orientation, the linear and angular velocities of the root, and the previous action. The root orientation and velocities are aligned with those of the robot's base. Additionally, for the experiments that involve clustering, the observation vector also included the cluster id.

[1] https://enchanted.tools

4.3.2 Reward function design

Our goal is to develop a control strategy that minimizes damage to the robot during a fall. We seek a strategy that is effective regardless of the initial pose of the arms. An early attempt at applying RL for minimizing the damage of a falling ballbot has been presented in the previous chapter, but that study considered only a single, fixed initial pose. In contrast, the goal of the present work is to develop a policy that reduces impact damage across a range of initial poses of the arms.

We chose to solve the task using RL to leverage the strengths of model-free solutions, given the difficulty of accurately modeling ballbot falls. The main part of the reward function is given by two different reward components: one is given at each step of the control loop, and the other is given only at the end of the episode. We refer to these two components as the *mid-episode reward* r_m and the *end-episode reward* r_e, respectively. Each episode ends when the robot is on the ground, or when it reaches a maximum duration of 5 s.

The *mid-episode reward* r_m is given by the Euclidean norm $||\cdot||$ of the actuated joint velocities $\dot{q}$:

$$r_m = ||\dot{q}||. \tag{4.3}$$

The objective of this component is to minimize unnecessary motion, hence reducing energy consumption.

The *end-episode reward* is provided only at the end of the episode and measures the total damage incurred by the robot. We evaluate the damage to the robot by considering both the contact forces on each body (due to ground impact) and the joint yank. The joint yank is the time derivative of the force applied to the joint; minimizing the yank thus helps preserve the joint's motor [48]. The *end-episode reward* r_e is in turn the weighted sum of two components r_{eb} and r_{ej}, the first related to the contact forces and the second related to the joint yank:

$$r_{eb} = \max_{t=0,\dots,H} \sum_{b\in\mathcal{B}} \left\| cf_b(t) \right\|^2 \tag{4.4}$$

$$r_{ej} = \max_{t=0,\dots,H} \sum_{i=1}^{N} \left\| df_i(t) - df_i(t-1) \right\|^2 \tag{4.5}$$

where again t is the timestep, H is the horizon of episode, and $\mathcal{B}$ is the set of robot bodies. The term $cf_b(t)$ represents the contact force acting on body b at time t, while $df_i(t)$ is the total force acting on the i-th actuated joint. Similarly to [35], the weights ω_{eb} and ω_{eb} of these components are selected so that the contact forces have a stronger influence than joint yank. The weight for ω_m is chosen so that the mid-episode reward (on the order of 10^{-2}) is about 10^4 times smaller than the end-episode reward (on the order of 10^2). This choice reflects that only the end-episode reward refers directly to damage minimization.

The reward components' weights are given in 4.2; note that all these weights are negative. Since these quantities measure the damage to the robot (which we want to minimize), having negative weights ensures that maximizing the total reward corre-

sponds to minimizing damage.

Table 4.2: Weight coefficients for each component of the reward function. These coefficients, as well as the reward function itself, are shared by both the policy and the baseline methods considered in the experimentation.

Reward component	**Weight**
r_{eb}: contact force (end-episode)	$\omega_{eb} = -0.005$
r_{ej}: joint yank (end-episode)	$\omega_{ej} = -0.05$
r_m: velocity (mid-episode)	$\omega_m = -0.005$

4.3.3 Clustering on the initial pose of the arms

We analyzed how the robot falls when starting from different arm positions. To do so, at the start of each episode, we reset the robot in simulation to an upright position with different initial arm positions, then allow it to fall without applying any control. For these different initial poses, we varied only the three shoulder joints and the elbow joint, as the position of the wrist is less relevant to the fall dynamics. Moreover, we considered only symmetric arm positions, reflecting typical real-life scenarios in which both arms are usually positioned symmetrically when moving or standing. Since the various initial poses produced a wide range of fall behaviors, we clustered them into smaller groups with similar characteristics.

We began by discretizing the joint position space. For each of the shoulder and elbow joints, we divided the range of motion into 5 equally spaced intervals, resulting in $5^4 = 625$ unique initial poses of the arms.

Next, for each of these 625 poses, we simulated 20 free-fall episodes (i.e., no policy was applied). For each pose, we then computed the following metrics across the 20 episodes:

- **Time of impact** (mean and median over 20 free-fall episodes): This is the duration from the start of the simulation until the moment of maximum contact force.
- **Body of impact** (mean and median over 20 free-fall episodes): This is the specific robot's body on which the maximum contact force occurs. Since the robot is composed of 64 bodies, this number can assume integer values from 0 to 63.
- **Maximum contact force** (mean and 95-th percentile over 20 free-fall episodes): We recorded the maximum contact force in each of the 20 trials, then computed the mean and the 95-th percentile of these values.

With these metrics in hand, we proceeded to cluster the data based on three features, namely 1) the median time of impact, 2) the median id of the body of impact,

and 3) the 95-th percentile of the maximum contact force. We varied the number of clusters from 2 to 68 (in increments of 2). The procedure involved normalizing all three features, applying the K-means algorithm, and evaluating different clustering configurations via the silhouette score. The silhouette score remained around 0.3 for all the clustering setups analyzed, suggesting an overall good separation with minimal overlap. Additionally, scores tended to be slightly lower when the number of clusters exceeded 56, indicating worse clustering quality.

To select the optimal cluster number, we examined three clustering configurations with high silhouette scores—-namely 52, 12, and 4 clusters—-representing small, medium, and large cluster sizes, respectively. For each clustering size, we selected the clusters associated with the maximum contact forces. Separate policies were trained for 300 episodes, considering only the positions within each cluster group.

For evaluation, each policy was tested over 20 episodes across all initial configurations within the corresponding group. In each episode, we compute the absolute value of the maximum contact force by taking the maximum force along the z-axis over all timesteps and all robot bodies as

$$Mcf = \max_{t\in\{0,\dots,H\},\mathcal{B}} |cf_{bz}(t)| \tag{4.6}$$

where $cf_{bz}(t)$ is the z-component of the contact force acting on the body b at time t. For each cluster group, we record the mean $\overline{M}cf$ of these maximum contact forces. We compute the same metrics for a *free-fall* scenario, where no policy is applied.

Furthermore, we define the *mean delta percentage* $\Delta\overline{M}$ of the contact forces relative to the free-fall case as:

$$\Delta\overline{M} = \frac{\overline{M}cf_{free} - \overline{M}cf_{policy}}{\overline{M}cf_{free}} \tag{4.7}$$

$$T\Delta\overline{M} = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \frac{\overline{M}cf_{free}p - \overline{M}cf_{policy}p}{\overline{M}cf_{free}p} \tag{4.8}$$

$$s\Delta\overline{M} = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \frac{\overline{M}cf_{free}p - \overline{M}cf_{policy}p}{\overline{M}cf_{free}p}\bigg|_{\text{success}} \tag{4.9}$$

$$f\Delta\overline{M} = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \frac{\overline{M}cf_{free}p - \overline{M}cf_{policy}p}{\overline{M}cf_{free}p}\bigg|_{\text{failure}} \tag{4.10}$$

$$TS = \frac{\#success}{|\mathcal{P}|}. \tag{4.11}$$

We then took the average of $\Delta\overline{M}$ across all the initial poses to obtain, for each selected cluster group, the Total Mean Delta $T\Delta\overline{M}$, considered as an indicator of the overall performance of each RL approach.

We also defined a metric of success for the task. An initial pose is considered a *success* if the contact force achieved by the policy tested on that specific initial pose is lower than the contact force observed without the policy, and a *failure* otherwise. With

this definition in hand, the Total Success rate TS is calculated by dividing the number of *successes* by the total number of initial poses within the cluster group. If we consider $\Delta\overline{M}$ only for the *successes*, we can calculate the mean $s\Delta\overline{M}$ and the standard deviation of the mean delta percentage for these *successes*. Similarly, we can calculate the mean $f\Delta\overline{M}$ and the standard deviation of the mean delta percentage for the *failures*. These metrics help us better understand the effectiveness of the *successes* and the severity of the *failures*.

Table 4.3: Metrics used to evaluate the policies for each cluster group. Each metric is computed over 20 test episodes for every initial pose within the corresponding cluster.

Cluster size	$T\Delta\overline{M}$ (%)	TS (%)	**Success** ($s\Delta\overline{M}$ ± Std)	**Failure** ($f\Delta\overline{M}$ ± Std)
Large	-2.69	0.54	26.02 ± 0.1675	-35.87 ± 0.2778
Medium	33.7	0.92	38.82 ± 0.1569	-26.0 ± 0.2245
Small	32.6	0.94	35.58 ± 0.1523	-15.02 -

As can be observed in Table 4.3, the policies trained on the small and medium-sized cluster groups yield similar performance, whereas the policy trained on the large cluster group performs poorly. Consequently, we focused our analysis on the 12-group clustering configuration, which provides a balanced trade-off between cluster size and damage reduction.

4.3.4 Reinforcement learning approaches

We now introduce the multi-policy approach alongside the set of baseline methods used for comparison. To assess the effectiveness of our method, we designed a set of baselines aimed at isolating the impact of both clustering and the use of multiple policies. Specifically, we include a single-policy baseline with and without clustering to evaluate their individual contributions.

Curriculum learning was also considered. Not only has it proven effective in handling complex robotic tasks, but it is specifically suitable for the problem we are addressing. In fact, the fall of a humanoid ballot exhibits high variability, but it is not composed of subtasks that require different skills. This makes curriculum learning more suitable than hierarchical or multi-task reinforcement learning.

We excluded model-based reinforcement learning, as model-free methods are particularly suitable for handling uncertainties and situations that are difficult to model, like the one we are addressing.

Multi-policy approach

Considering the 12-group clustering configuration explained in Section 4.3.3, we develop a multi-policy approach by training a separate policy for each of the 12 clusters, each one for 300 episodes, for a total of 3600 episodes. Since clustering is performed on a

set of discrete joint positions, we define intervals around each joint's discrete value by calculating the midpoints between consecutive discrete values.

At the start of each episode, we randomly sample an initial pose of the arms from the set of poses used for clustering. We then add random noise to each joint while ensuring that the new positions remain within the intervals corresponding to the initially selected values. In total, the training process of this approach lasts approximately 3.67 days of wall-clock time.

Single-policy approach with clustering information

In this case, at the start of each episode, the initial pose of the arms is chosen randomly. We then identify the corresponding cluster for that initial pose as follows: for each joint, we consider the interval in which its value lies and the value in the interval that was used for the clustering. These discrete values form a vector used to query the appropriate cluster. This cluster identifier is subsequently incorporated into the observation vector. In total, 5200 episodes were run, lasting approximately 5.48 days of wall-clock time.

Single-policy approach without clustering information

In this case, at the start of each episode, in every environment, the initial pose is chosen by randomly assigning a position to the three joints of the shoulder (shoulder sagittal, frontal, and transversal) and to the elbow joint. In contrast to the previous approach, we did not determine the cluster to which the initial pose belongs. Instead, we trained a policy without this information to assess its relevance. In total, 5200 episodes were run, lasting about 5.4 days of wall-clock time.

Table 4.4: Metrics used to evaluate the different RL approaches. Each metric is computed across 20 test episodes for each of the 625 initial poses used for clustering.

Method	$T\Delta\overline{M}$ (%)	TS (%)	**Success** ($s\Delta\overline{M}$ ± Std)	**Failure** ($f\Delta\overline{M}$ ± Std)
Multi-policy with clustering	+13.79	72.00	0.34 ± 0.1804	-0.37 ± 0.4124
Single-policy with cluster information	+11.26	65.44	0.39 ± 0.2193	-0.41 ± 0.3609
Single-policy w/o clustering information	+5.11	61.12	0.38 ± 0.2133	-0.47 ± 0.4972
Curriculum learning	-23.63	39.36	0.25 ± 0.1713	-0.55 ± 0.5375

Curriculum learning approach

Motivated by the success of curriculum learning methods in robotic tasks [53], we also trained a policy using a curriculum-based strategy to establish an additional baseline. To do that, we incrementally increased the difficulty of the fall task by altering the initial pose of the arms as follows:

- **First scenario**: Only the position of the first joint (shoulder sagittal) is randomized.
- **Second scenario**: The positions of the first two joints (shoulder sagittal and frontal) are randomized.

- **Third scenario**: The positions of the first three joints (shoulder sagittal, frontal, and transversal) are randomized.
- **Fourth scenario**: The positions of the first four joints (shoulder sagittal, frontal, transversal, and elbow) are randomized.

Randomizing a single joint tends to produce similar fall behaviors across episodes, whereas increasing the number of joints introduces greater variability and results in a higher-dimensional state-action space, thus making it a more difficult task. To empirically support this, we trained a policy for 500 episodes in two settings: one where only a single joint was randomized, and another where all joints were randomized. The single-joint setup achieved a Total Mean Delta $T\Delta\overline{M} = 45\%$, whereas the full-joint setup showed minimal improvement, with $T\Delta\overline{M} = 21\%$ (note that these results are not reported in Table 4.4), highlighting the increased difficulty introduced by higher-dimensional control.

In all scenarios, the pose of the left arm mirrors the pose of the right arm. We transition from one scenario to the next, a more challenging one, either when the reward r_{eb} drops below a specified threshold ($r_eb \leq 2200$), empirically chosen to indicate effective damage reduction, or when the maximum number of episodes for the current scenario is reached. We set this maximum to 1200 episodes per scenario.

Overall, this curriculum learning approach was trained for 5200 episodes, lasting a total of 5.6 days of wall-clock time.

4.4 Results

To evaluate the results of the four RL approaches described in the previous section, we tested each policy on 20 test episodes for each of the 625 initial poses used in the clustering process.

We report the results in terms of $T\Delta\overline{M}$, TS, $s\Delta\overline{M}$, and $f\Delta\overline{M}$ for the different RL approaches in Table 4.4, where the metrics are defined in Section 4.3.3.

From the results, we can state that the curriculum learning approach seems not to be particularly suitable for this task, yielding the worst performance with a Total Success rate of only 39%. Furthermore, the single-policy approaches exhibit comparable performance; however, the policy incorporating clustering information outperforms the one without clustering, with $T = 65\%$ (vs. 61%) and a $T\Delta\overline{M} = 11\%$ (vs. 5%).

The best policy is provided by the multi-policy approach, which not only achieves $T = 72\%$ but also attains a $T\Delta\overline{M}$ exceeding 13%. Moreover, it maintains a lower failure mean delta percentage $f\Delta\overline{M}$, and although its success mean delta percentage $s\Delta\overline{M}$ is slightly lower than that achieved by the single-policy approaches, it is still comparable. Overall, this method stands out as the superior choice, particularly due to its faster training time (we purposely chose a much lower number of episodes than for the other methods, 3600 vs. 5200, as we observed much faster convergence with this method).

Demonstrative videos highlighting the improved performance of our approach compared to the baselines can be accessed at:

https://github.com/giu950/Multi-policy-approach-for-damage-reduction, while Appendix B presents the image sequences illustrating the falling behavior observed from the starting position with both arms stretched out in front of the body across the evaluated approaches.

4.5 Conclusions

In this chapter, we investigated the fall resilience of a humanoid ballbot. Our approach exclusively leverages the arms to control the robot during a fall, since the wheels may lose contact with the ball, making direct ball control infeasible in such scenarios. By applying K-means, we clustered the possible initial poses of the arms and trained a separate policy for each cluster using RL. We validated our multi-policy approach against three baselines: (1) a single RL-based policy, (2) a single RL-based policy incorporating clustering information on the initial pose of the arms, and (3) an RL-based policy enhanced by curriculum learning. Overall, our proposed multi-policy approach achieved the best performance, achieving lower contact forces compared to the free fall, given an initial pose of the arms in $\sim$70% of the cases.

Future work will focus on extending the proposed method to non-symmetric poses of the arms as well as considering different body orientations.

Chapter 5

Reinforcement Learning for Ballbot Balancing

In this chapter, we focus our research question on assessing the impact that the RL model has on ballbot balancing. Specifically, we address the challenge of achieving stable balancing control for a ballbot, an inherently unstable robot that presents a particularly demanding control problem. While traditional control approaches rely heavily on accurate mathematical modeling and analytical tuning, reinforcement learning (RL) offers a model-free alternative that can adapt to nonlinearities and uncertainties in the system.

We compare the effects of stability training in simulation and assess the importance of the model. We explored several models, beginning with a linear model in which the model is decoupled into three independent planes and the torque is directly applied to each plane. We compare it with an omniwheel-based model, which revealed the sensitivity of wheel–ball interactions. Building upon recent state-of-the-art work that models omniwheels as capsules with anisotropic friction, we reproduced and validated their results. This enabled us to achieve stable navigation in simulation and confirm the feasibility of RL-based control under improved modeling assumptions.

This chapter brings together the insights gained from using different modeling approaches, shows the practical limitations that were encountered, and creates a foundation for future work on using RL to control ballbots in the real world.

The content of this chapter is based on the following publication:
G. Buzzetti, D. Zappetti, G. Iacca, Reinforcement Learning for Ballbot Balancing, International Conference on Control, Decision and Information Technologies (CoDIT), IEEE, Bari, July 2026
Proceedings to appear.

5.1 Introduction

Ballbot robots are robots that balance on a ball using wheels. Over the past years, different ballbot models have been implemented, with configurations ranging from the number and placement of omniwheels used to balance on the ball [30], to different sizes

[46] and application scopes [33].

Balancing on a single sphere allows ballbots to occupy little space and to perform swift, agile motions, while also being easy to move by users. Due to this configuration, however, ballbots operate around an unstable equilibrium. This means that they must continuously move in order to maintain balance, i.e., to avoid tipping over or falling. For this reason, reliable and stable balance control is essential.

Most of the literature focuses on classical control approaches, such as PID [5], LQR [47], or combinations of the two [46].

Only a limited number of reinforcement-learning-based approaches for balancing have been proposed, and they all rely on the planar model described in Section 5.2.1. In [61], a deep neural network is trained to recover stability after perturbations, while a PID controller is used to maintain balance. In contrast, [42] proposes a general framework for training an RL agent to balance a ballbot. However, although the framework is described, no agent is actually trained, and no results on RL-based balancing are reported.

Here, we propose a critical evaluation of ballbot modeling choices for reinforcement-learning-based balancing. Our contributions are:

- a comparative study of different ballbot modeling choices for RL-based balancing;
- an empirical analysis of how commonly used modeling abstractions affect learning stability and performance;
- the identification of a simple and simulation-stable modeling setup that enables smooth, drift-free control.

The rest of the chapter is structured as follows: Section 5.2.1 describes the different models that are compared, and how the RL framework is applied to each model; Section 5.3 reports the results; and Section 5.4 concludes the chapter.

5.2 Methodology

5.2.1 Modeling paradigms for ballbot control

To address the problem of balancing in ballbot robots, we conduct a comparative analysis of three different modeling paradigms and evaluate their applicability within a reinforcement learning framework.

Given the inherent complexity of the task, we adopt a simplified model that preserves the key characteristics of a ballbot while neglecting features specific to the target platform. In particular, although the system is inspired by the Enchanted Tools robot—a humanoid ballbot equipped with a head, arms, and additional components—these elements are not essential for capturing the core balancing dynamics.

The robot body is therefore approximated as a cylinder, omitting arms, battery, facial features, and other secondary structures. Assuming negligible limb motion during balancing, this approximation remains representative of the system while significantly

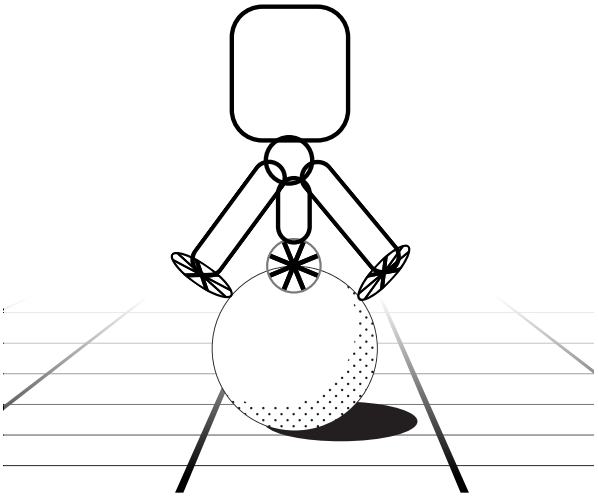


Figure 5.1: Simplified structure of the ballbot. The body is modelled as a cylinder on top of a sphere.

reducing computational complexity and improving simulation efficiency. The spherical base is retained as the primary dynamic element, while the inclusion of internal actuation components (e.g., wheels) depends on the specific modeling paradigm, as discussed in the following sections. As illustrated in Figure 5.1, this representation captures the overall structure of the system.

These simplifications are consistently applied across all models, ensuring a fair comparison. Although they introduce discrepancies with respect to the real system—particularly for sim-to-real transfer—their impact can be mitigated by retraining the model using the true inertial parameters of the physical robot.

Modeling and simulating the interaction between the ballbot body and the sphere is particularly challenging due to the complexity of accurately representing the omniwheels. To address this challenge, three models with different levels of abstraction are investigated to determine which most effectively captures the real behavior of the ballbot.

The following subsections provide a comprehensive overview of these models.

Modeling the ballbot with a linear model

The full three-dimensional robot is first approximated as three independent planar subsystems, consistent with the literature [13, 47, 39, 40, 61, 42]. Two planes, denoted xz (sagittal) and yz (coronal), are used to describe balancing while moving forward/backward and left/right, respectively, and are assumed to be identical. The third plane, denoted xy, captures yaw motion (rotation about the vertical axis) and is generally treated separately or given less emphasis for balance control.

Each planar subsystem is described using two degrees of freedom: one DoF for the ball motion along the ground (equivalently, the ball's rolling angle in that plane), and one DoF for the body rotation (tilt) in that plane.

The real actuation consists of three omniwheels; however, the planar model replaces

them with a single "virtual" actuating wheel per plane, producing an equivalent driving torque on the ball. This is a modeling convenience and does not represent the physical wheel geometry directly. Consequently, when implementing control on hardware, the virtual-plane torques must be mapped back to the three real motor torques via a suitable conversion.

The following assumptions are made in order to capture the dominant balancing dynamics:

- The decoupled planes are assumed to be independent, i.e., no coupling between planes is considered.
- No slippage occurs between the ball and the ballbot body.
- The distance between the ball and the body is fixed.
- No friction is present at the ball–body contact.
- The contact between the ball and the body is non-deformable, and all components are treated as rigid bodies.
- Motor torque is treated as a direct input rather than being modelled in detail.
- The floor is assumed to be flat.

Under these assumptions, the ballbot in a plane can be modelled as a rigid body (often idealized as a cylinder) mounted on a rolling sphere, resembling an inverted pendulum on a rolling base.

The limitations of this model stem from the idealized nature of these assumptions. In particular, the model does not account for wheel–ball slippage, motor friction, or possible loss of contact between the wheels and the sphere.

Modeling the omniwheels with spheres

The omniwheels employed in this robot correspond to the single-contact-line design originally proposed by Asama et al. [8]. This configuration, characterized by a single row of passive rollers, has been widely adopted due to its mechanical simplicity and its ability to accommodate surface irregularities. Alternative omnidirectional wheel designs incorporate rollers with cylindrical geometry mounted at a constant inclination with respect to the wheel axis, most commonly at an angle of 45°, as in conventional Mecanum-type wheels [11].

Owing to the passive rotation of the rollers, omniwheels are able to transmit traction forces along the wheel's rolling direction while allowing unconstrained motion in the perpendicular direction [27]. Consequently, the wheel–ground interaction can be decomposed into a tangential driving force generated by wheel rotation and a lateral component associated with the free rolling of the rollers, analogous to the sliding behavior observed in spherical contacts.

Several approaches have been proposed to model this behavior in simulation. A high-fidelity method consists of explicitly modeling each roller and its contact with the

ground, potentially using rollers of varying effective radii to ensure smooth transitions between successive contacts [59]. While such approaches improve numerical continuity, they significantly increase computational cost and can severely degrade real-time simulation performance. Similar drawbacks have been reported in dynamic simulations of omnidirectional and Mecanum wheels, where discretization of rollers leads to contact-induced vibrations and reduced simulation efficiency.

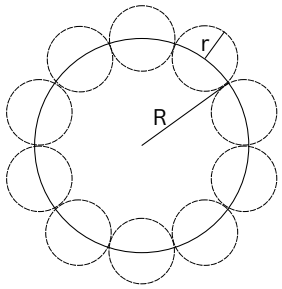


Figure 5.2: The omniwheel model used a cylindrical hub surrounded by ten small, free-rotating rollers that allow passive lateral motion.

Here, following the simplified strategy adopted in [11], the omniwheel is modelled as a cylindrical hub equipped with ten evenly spaced, fixed rollers distributed along its circumference, as illustrated in Fig. 5.2. Although this representation captures the essential kinematic properties of the omniwheel, it introduces non-ideal contact transitions between adjacent rollers, which manifest as small oscillations during motion. The simulation itself is unstable: because of the gaps between adjacent spheres, the ballbot intermittently loses continuous contact with the ground, causing it to bounce on the globe.

To mitigate this effect, ball arresters were introduced. These are fixed spheres placed on the opposite side of each omniwheel, as shown in Figure 5.3. However, this modification alone is insufficient to ensure stability. When the arresters are positioned too close to the globe, they constrain the omniwheels and prevent proper rotation. Conversely, when they are placed too far from the globe (i.e., too close to the ball), the ballbot continues to exhibit the bouncing behavior.

Modeling the omniwheels as capsules with anisotropic friction

The model developed in this chapter is inspired by the fundamental kinematic behavior of omniwheels equipped with a single row of passive rollers. Figure 5.4 illustrates the velocity components that arise at the contact between an omniwheel and the surface on which it rolls. In the case of a ballbot, this surface is the spherical globe on which the robot balances.

When the omniwheel rotates around its main axis, it generates a tangential velocity at the contact point with the globe. This velocity lies in the plane of the wheel and is perpendicular to the wheel's axis of rotation. In Fig. 5.4, this component is denoted

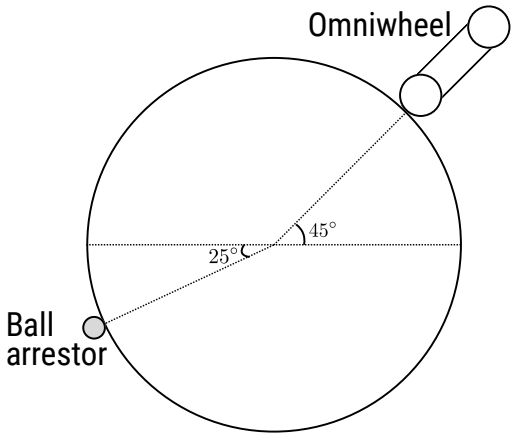


Figure 5.3: Cross-section of the rolling globe of the ballbot showing one of the three omniwheels and the ball arrester. The omniwheel is mounted at an upward angle of 45°, while the ball arrester is positioned diametrically opposite at an angle of 25°.

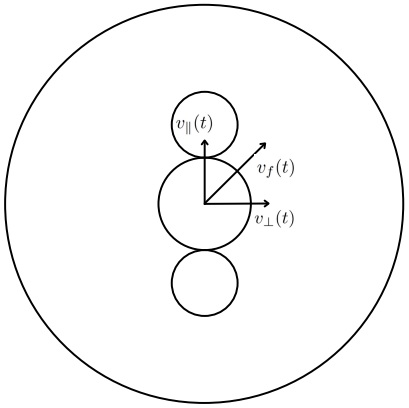


Figure 5.4: Velocity components at the contact point between an omniwheel and a rolling globe, showing the driven tangential component $v_\parallel(t)$ and the passive transverse direction $v_\perp(t)$ enabled by the rollers.

by $v_\parallel(t)$. It is the only velocity component that the wheel actively imposes on the contacting surface.

Due to the presence of the passive rollers, the omniwheel exhibits a fundamentally different behavior in the transverse direction. The rollers are free to rotate about their own axes, which are orthogonal to the main wheel axis. As a result, the contact point can move in the direction perpendicular to $v_\parallel$ with negligible resistance. If the globe has a velocity component in this transverse direction, it is not opposed by significant

friction at the contact, because this motion is accommodated by the rotation of the rollers.

Therefore, the velocity of the contact point P between the omniwheel and the globe can be decomposed into two orthogonal components: one parallel to the wheel's driven direction and one perpendicular to it. While such a decomposition is always possible, the defining characteristic of an omniwheel is that only the parallel component is constrained by friction and actuation, whereas the perpendicular component is effectively unconstrained.

This property is essential for ballbot operation. A typical ballbot employs three omniwheels arranged at 120 deg intervals around the globe. Each wheel drives the globe along its own local parallel direction, which differs from that of the other wheels. If transverse motion were not permitted at each contact, the wheels would mechanically interfere with one another, generating conflicting constraints that would prevent smooth rolling. The passive rollers eliminate these conflicts by allowing relative motion in the transverse directions, enabling the combined wheel actions to produce arbitrary planar motion of the globe.

One solution to replicate this mechanism in simulation was proposed in [45]. The authors simplify the simulation of Mecanum wheels by replacing each roller's rotating joint with a fixed connection, which greatly reduces the number of constraints and speeds up computation. To preserve the physical behavior of the rollers, they assign direction-dependent (anisotropic) friction to the roller links. High friction is applied along the roller's free-rolling axis to transmit driving force, while low friction is used across it, allowing sideways slip similar to that of real rollers.

A simple and effective solution was proposed in [49]. In their work, the authors simulate the omniwheels as capsules with anisotropic friction. The friction in the direction of wheel motion is set high to ensure force transmission, while in the perpendicular direction, it is low to simulate the effect of the passive rollers. This solution was made possible by a custom feature implemented in the MuJoCo simulator [55]. To the best of our knowledge, MuJoCo is the only simulator that allows anisotropic friction to be specified on the same body.

5.2.2 Reinforcement-learning-based balancing

In this section, we describe how the Reinforcement Learning (RL) framework for balancing was implemented for the different models under study. We first outline the elements that are common across all experiments and then discuss the model-specific design choices.

Common experimental setup

To address the balancing problem using Reinforcement Learning, we employ a deep neural network and adopt a symmetric actor–critic architecture trained with the Proximal Policy Optimization (PPO) algorithm [51]. In all experiments, the network consists of three fully connected layers with sizes 512, 256, and 128. The actor and critic networks share the same architecture and use the same observation vector, consistent with the

symmetric actor–critic formulation. The control is velocity-based. The physics simulation runs at 200, Hz, while the policy outputs actions at 50, Hz. The different models use different action spaces because they have different actuators; the specific action definitions for each model are detailed in the following subsections. The observation vector is identical across all experiments. It includes the projected gravity expressed in the trunk frame, the velocity of the actuated degrees of freedom, the linear velocities of the trunk, the angular velocity of the trunk, the desired velocities, and the previous action. Since the task is to balance at a fixed point, the velocity commands are set to zero. The projected gravity in the trunk frame represents the direction of gravity in the local coordinate system of the trunk and therefore encodes its orientation with respect to gravity. The horizontal components (g_0, g_1) correspond to the trunk tilt in the sagittal and lateral directions, while the vertical component g_z measures the alignment of the trunk's vertical axis with gravity. The reward function is composed of five different terms. One term provides positive feedback when the ballbot's base is close to horizontal:

$$r_o = -\omega_o \sum_{j=0}^{1} g_j^2 + c, \tag{5.1}$$

where $\omega_o > 0$ is a weighting factor, c is a constant chosen to keep the reward positive, and g_0 and g_1 are the x and y components of the projected gravity of the ballbot's base. A second term penalizes large control actions:

$$r_a(t) = \omega_a \sum_{j=0}^{2} a_j^2(t), \tag{5.2}$$

where $\omega_a < 0$ is a weighting factor and $a(t) = [a_0(t), a_1(t), a_2(t)]$ denotes the action vector at time t. A third term penalizes large variations in the actions:

$$r_l(t) = \omega_l \sum_{j=0}^{2} \big(a_j(t-1) - a_j(t)\big)^2, \tag{5.3}$$

where $\omega_l < 0$ is a weighting factor. Another term tracks the desired linear velocity along the x and y axes:

$$r_{vl}(t) = e^{\left(\frac{\sum \left(c_l(t) - v_l(t)\right)^2}{\sigma_l}\right)}, \tag{5.4}$$

where $c_l(t)$ is the desired trunk velocity in the x–y plane, $v_l(t)$ is the measured trunk velocity, and σ_l is a scaling constant. Finally, the angular velocity around the z axis is tracked by

$$r_{va}(t) = e^{\left(\frac{\sum \left(c_a(t) - v_a(t)\right)^2}{\sigma_a}\right)}, \tag{5.5}$$

where $c_a(t)$ is the desired angular velocity around the z axis, $v_a(t)$ is the measured angular velocity, and σ_a is a scaling constant.

Each episode terminates when the robot reaches a lean angle of 20°, which is set as a threshold beyond which the ballbot can no longer regain stability. In addition, an episode is terminated if the network outputs an action at the maximum allowed magnitude. This choice was made to avoid bang-bang control behavior that would compromise the stability of the robot. If none of these conditions are met, the maximum episode length is set to $40, \mathrm{s}$, based on the assumption that if the robot is able to balance for more than $40, \mathrm{s}$, it has reached asymptotic stability.

Reinforcement Learning for the ballbot linear model

The RL framework implemented using the linear model was trained in the Isaac Gym environment [36]. Isaac Gym provides a GPU-based platform in which both the simulation and the training of the algorithm are executed, enabling fast and highly parallelized learning. Leveraging this architecture, the agent was trained using 2048 parallel vectorized environments for 1000 episodes.

The velocity-based controller runs at $50, \mathrm{Hz}$ and outputs the desired velocities for each of the actuated linear planes of the ballbot, in which the dynamics are decomposed, namely the roll, pitch, and yaw of the ball.

The algorithm, observation vector, and reward function follow the common experimental setup highlighted in the previous subsection.

Reinforcement Learning for the omniwheels modelled with spheres

This model is computationally heavier than the planar one. Therefore, it was not possible to simulate the same number of parallel vectorized instances. Consequently, the agent was trained in Isaac Gym using 1024 parallel instances for 2000 episodes.

The network outputs the desired velocities of the three omniwheels, which are simulated as cylinders with rollers on their surfaces. As in the planar case, the algorithm, observation vector, and reward function follow the common experimental setup.

Reinforcement Learning for the omniwheels modelled with capsules with anisotropic friction

This framework could not be implemented in Isaac Gym because anisotropic friction on the same body is not supported. The RL environment was therefore built on top of MuJoCo, and both the simulation and the training pipeline were executed on the CPU. As a result, large-scale parallelization was not possible, and the network was trained using 10 parallel environments for 4,000,000 episodes.

The network outputs the desired velocities of the wheels, which are simulated as capsules with anisotropic friction. As in the previous cases, the algorithm, observation vector, and reward function follow the common experimental setup.

5.3 Results

We first present the results of the linear model, followed by those obtained with the omniwheels modelled as spheres, and finally those obtained with the model in which the omniwheels are represented as capsules with anisotropic friction.

5.3.1 Results obtained with the linear model

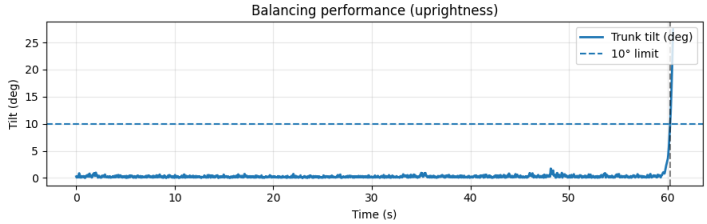


Figure 5.5: Time evolution of the ballbot trunk's deviation from upright, measured from the projection of the gravity vector onto the trunk z-axis. The dashed line denotes the 10° allowable tilt.

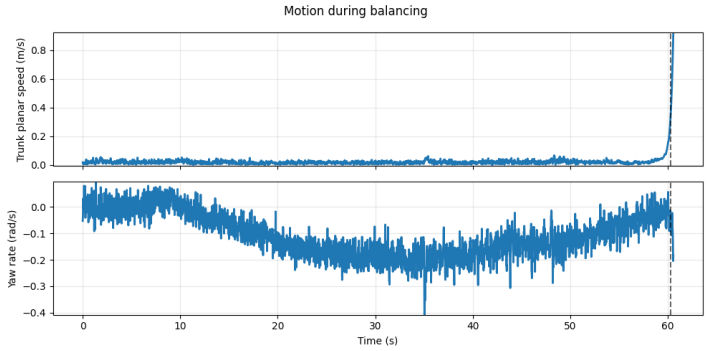


Figure 5.6: Motion of the trunk during balancing. The top panel shows the magnitude of the linear velocity in the horizontal (xy) plane, while the bottom panel shows the angular velocity about the vertical (z) axis.

Figure 5.5 and 5.6 show the orientation and the velocity of the trunk during a test episode in simulation. While the system can maintain balance for about a minute, it does not reach a stable equilibrium. We can notice a visible oscillation in the trunk orientation and an even more marked oscillation in the yaw velocity.

An inspection of the control output, presented in Figure 5.7, reveals that the actions are not smooth, but they present numerous spikes, which contribute to this instability. However, looking at Figure 5.8, we can observe that the model reasonably tracks the reference velocities until the onset of instability. Once the ballbot begins to fall, recovery is impossible, and trajectory tracking deteriorates rapidly.

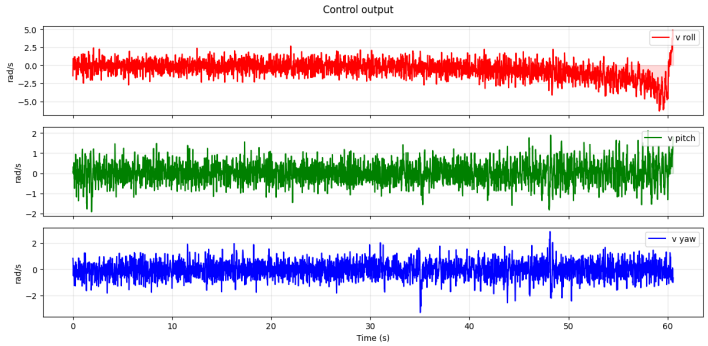


Figure 5.7: Control outputs of the neural network. The plots show the desired angular velocities of the globe about the roll (top), pitch (middle), and yaw (bottom) axes.

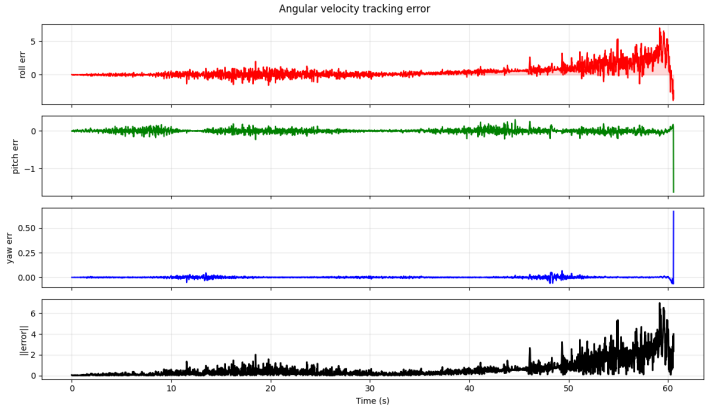


Figure 5.8: Tracking error between desired and measured globe angular velocities. The first three panels show roll, pitch, and yaw errors, and the bottom panel shows the Euclidean norm of the error vector.

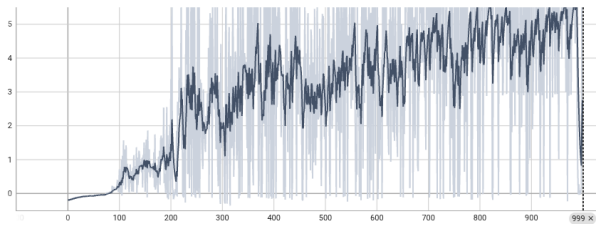


Figure 5.9: Learning progress of the planar model, measured by episode reward over training iterations. The smoothed curve highlights the underlying performance trend.

The total training time was approximately 20 minutes. The reward curve shown in Figure 5.9 indicates convergence to a steady value, suggesting that the learning process has stabilized and further training is unlikely to improve performance significantly.

5.3.2 Results obtained with the omniwheels modelled with spheres

In the model in which the omniwheels are represented as cylinders with spherical rollers, the agent is unable to maintain balance for more than half a second. Figure 5.10 presents the evolution of the trunk tilt over time. It can be observed that the robot reaches the limit angle of 10° after only 0.6, s; once this limit is reached, the system is unable to recover stability. Figure 5.11 shows the norm of the trunk's linear velocity over time and the angular velocity of the trunk about the z-axis. In comparison with Figure 5.6, it can be noted that both the linear and angular velocity magnitudes are larger than those observed in the linear model.

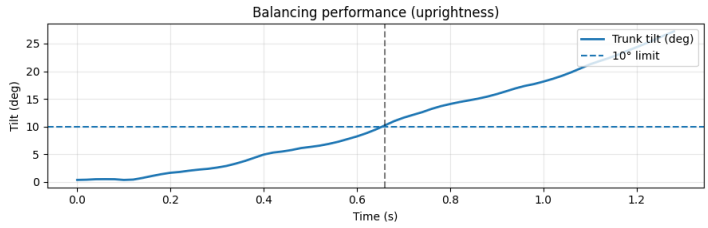


Figure 5.10: Trunk tilt over time, measured via the projection of the gravity vector onto the trunk z-axis. The dashed line marks the 10° allowable tilt.

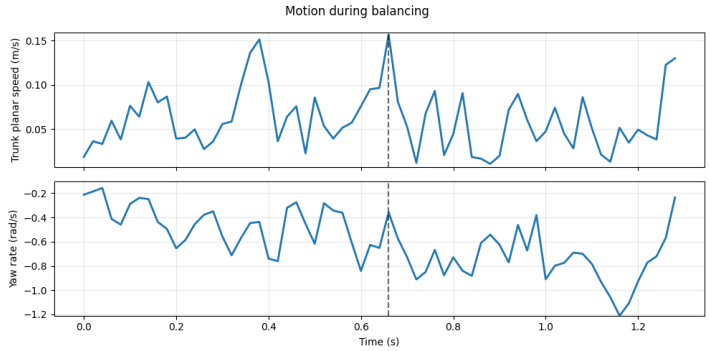


Figure 5.11: Trunk motion during balancing. The upper panel depicts the magnitude of the linear velocity in the horizontal (xy) plane, and the lower panel depicts the angular velocity about the vertical (z) axis.

If we consider the tracking error between the desired output velocities of the motors and the actual velocities, as presented in Figure 5.12, we can observe that, in this model, it is more difficult to accurately track the velocities with respect to the planar model. However, the real reason for the strong instability can be found by analyzing the control output of the network, presented in Figure 5.13. It can be seen that the network outputs the maximum allowed velocity on one motor, thereby terminating the experiment immediately.

By examining the evolution of the reward function depicted in Figure 5.14, it can be observed that this strategy of outputting the maximum velocity on one control command yields higher rewards than those obtained by actually balancing the robot.

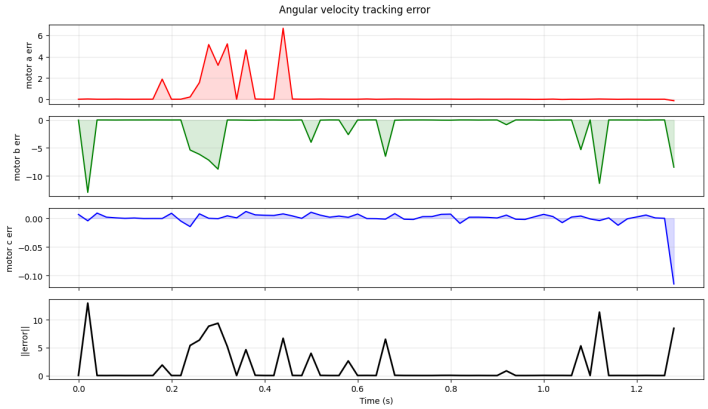


Figure 5.12: Tracking error between desired and measured angular velocities of the wheels. The first three panels show the error for each motor, and the bottom panel shows the Euclidean norm of the error vector.

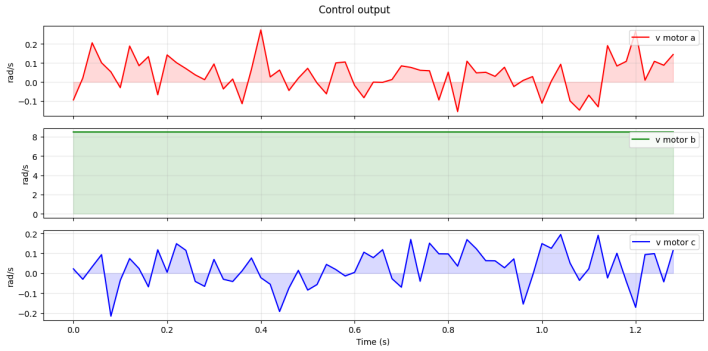


Figure 5.13: Control outputs of the neural network. The plots show the desired angular velocities of the three omniwheels.

Lastly, it should be noted that the overall training process was more demanding in this case, requiring a total training time of approximately 3 hours.

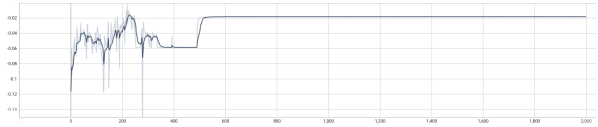


Figure 5.14: Learning progress of the roller-based omniwheel ballbot model, measured by episode reward over training iterations. The smoothed curve illustrates the evolution of learning performance.

5.3.3 Results obtained with the omniwheels modelled with capsules with anisotropic friction

Lastly, we analyze the model in which the omniwheels are represented as capsules with anisotropic friction.

As we can see from Figure 5.15, the balance is asymptotically stable: the initial oscillation of the tilt angle of the trunk reduces over time, and the tilt angle never reaches the threshold of 10°. Additionally, as depicted in Figure 5.16, the trunk linear and angular velocities also follow the same pattern. Moreover, differently from the previous models, the curves are smoother, indicating a smaller change in velocities, which indeed improves the overall stability.

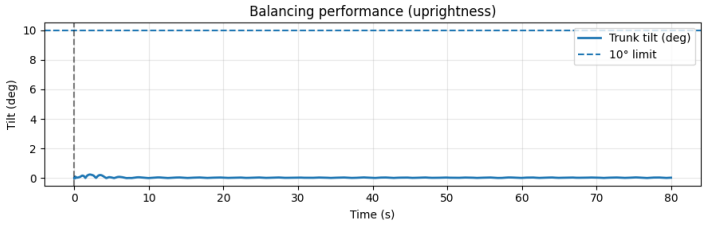


Figure 5.15: Trunk tilt angle over time, obtained from the projection of the gravity vector onto the trunk z-axis. The dashed line represents the 10° admissible tilt.

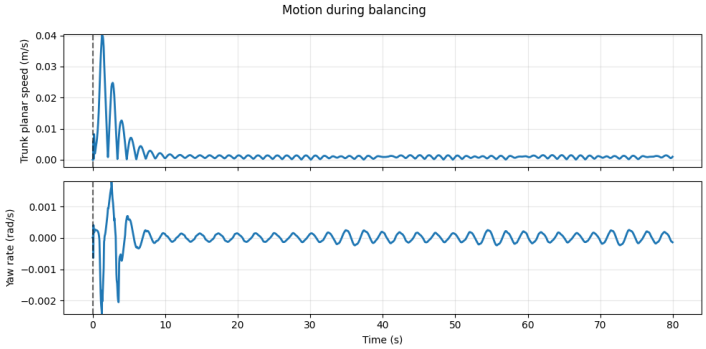


Figure 5.16: Balancing dynamics of the trunk: linear velocity magnitude in the horizontal (xy) plane (top) and angular velocity about the vertical (z) axis (bottom).

An analysis of the network outputs depicted in Figure 5.17 further shows that, in this framework, the control actions are smoother. After an initial transient peak, the control actions gradually converge to lower magnitudes. This behavior is consistent with that of an asymptotically stable system: once balance is achieved, only small corrective actions are required to maintain the upright configuration. As depicted in Figure 5.18, the capsules track the reference velocities closely, with the residual error arising primarily from the one-step delay introduced by the discrete simulation timestep.

As shown in Figure 5.19, the reward function reaches a plateau toward the end of training, indicating convergence of the learning process. In this case, the total training

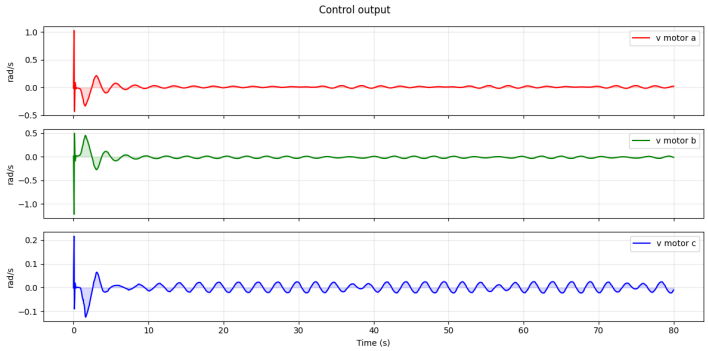


Figure 5.17: Control signals generated by the neural network, showing the desired angular velocities of the three omniwheels.

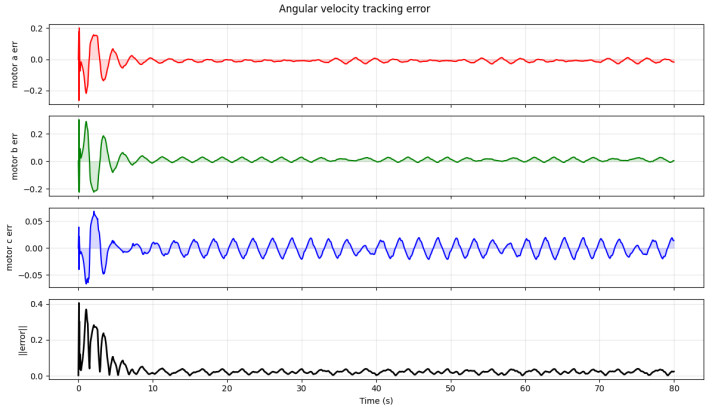


Figure 5.18: Angular velocity tracking performance of the wheels. The upper three plots depict the error for each motor, and the lower plot depicts the Euclidean norm of the error vector.

time was 94 minutes, which lies between those of the other two configurations considered.

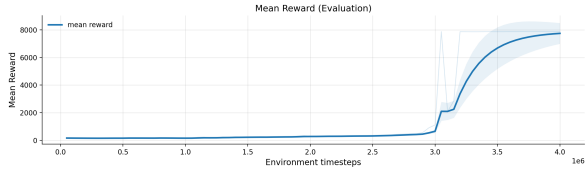


Figure 5.19: Learning progress of the roller-based omniwheel ballbot model, measured by episode reward over training iterations. The smoothed curve illustrates the evolution of learning performance.

5.4 Conclusions

In this chapter, we presented a comprehensive analysis of how different ballbot modeling choices affect the training of a deep reinforcement learning agent.

Contrary to initial expectations, we showed that the model most widely used in the ballbot literature, i.e., the planar model, exhibits intrinsic difficulties when applied to reinforcement learning. This is likely due to the fact that the roll, pitch, and yaw reference frames change with the rotation of the ball, introducing additional nonlinearities and increasing the complexity of the control problem faced by the agent.

By contrast, a simplified representation of the omniwheels as cylinders with passive rollers does not provide sufficient mechanical stability for learning or for deploying a reliable control policy.

Finally, by modeling the omniwheels as capsules with anisotropic friction, we obtained a representation that enables the successful training of an agent capable of maintaining balance in an asymptotically stable manner.

This study was conducted exclusively in simulation. A natural direction for future work is to bridge the sim-to-real gap by deploying the best-performing model on real hardware, using domain randomization to improve robustness to modeling uncertainties.

Chapter 6

Conclusions

This dissertation investigated reinforcement-learning-based strategies to improve the robustness, stability, and fall resilience of ballbot systems, with a particular focus on deep reinforcement learning, modeling fidelity, and damage mitigation through arm coordination. Across multiple studies, the work highlights how control design, policy structure, and physical modeling choices critically shape the feasibility and performance of reinforcement learning for dynamically unstable robotic platforms.

First, this thesis demonstrated that reinforcement learning can be effectively leveraged to reduce fall-induced damage in ballbots by actively controlling the arms during loss-of-balance events. Through a systematic comparison of reward formulations and control frequencies, the results showed that lowering the controller frequency consistently yields the most stable and reliable reduction in contact forces, with reductions reaching up to 70% in favorable configurations. These learned behaviors exhibit similarities to human-inspired fall mitigation strategies, underscoring the potential of learning-based approaches to discover intuitive yet nontrivial control policies. At the same time, the analysis revealed that certain reward design choices can degrade performance, emphasizing the importance of careful control and training configuration when deploying reinforcement learning for safety-critical tasks.

Building on this foundation, we then explored policy specialization through clustering of initial arm configurations. By decomposing the problem into multiple policies tailored to distinct arm pose clusters, the proposed multi-policy framework significantly outperformed single-policy baselines in most scenarios, achieving lower contact forces in approximately 70% of the tested cases. This result highlights the benefits of exploiting structure in the state space to improve fall resilience, particularly in highly nonlinear and underactuated systems. While real-world validation was not yet possible due to hardware constraints, the approach provides a scalable pathway toward more adaptive and context-aware fall mitigation strategies.

Finally, we addressed a fundamental but often overlooked aspect of learning-based ballbot control: the impact of modeling choices on reinforcement learning performance. In this regard, we showed that commonly used ballbot models introduce intrinsic difficulties for learning due to rotating reference frames and complex nonlinearities. In contrast, overly simplified models lack the mechanical stability required for successful

training. By modeling the omniwheels as capsules with anisotropic friction, this thesis identified a balanced representation that enables asymptotically stable learning and reliable balance control in simulation. These findings demonstrate that appropriate physical modeling is not merely a simulation detail but a prerequisite for successful reinforcement learning.

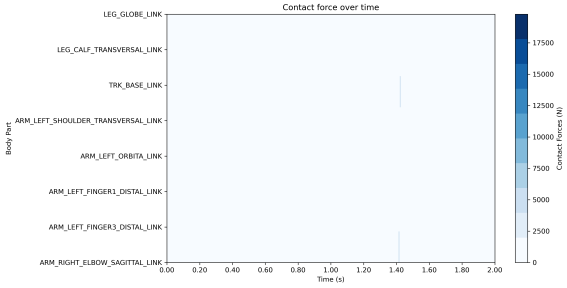
Taken together, the contributions of this dissertation advance the understanding of how reinforcement learning, control design, and physical modeling interact in the context of ballbot stability and fall resilience. They provide practical guidelines for designing learning-based controllers that are both effective and robust, while also exposing limitations that must be addressed before deployment on real hardware.

Several promising directions for future work emerge from this research. A primary objective is to bridge the sim-to-real gap through systematic sim-to-sim and sim-to-real transfer, leveraging techniques such as domain randomization and improved damage metrics that account for both the robot and its environment. Extending the proposed frameworks to non-symmetric arm configurations, varying body orientations, and more complex environmental interactions will further enhance generalization. At present, real-world validation is not feasible, as the robot remains a prototype and fall and balancing experiments pose unacceptable risks. However, future validation these approaches on real-world ballbot platforms will be a crucial step toward deploying learning-based fall mitigation strategies in practical robotic systems.

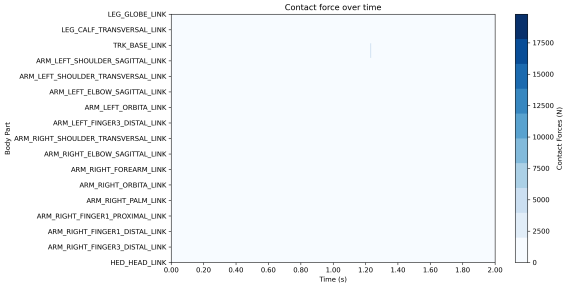
Appendix A

Distribution of Contact Forces Following a Ballbot Fall as discussed in Chapter 3

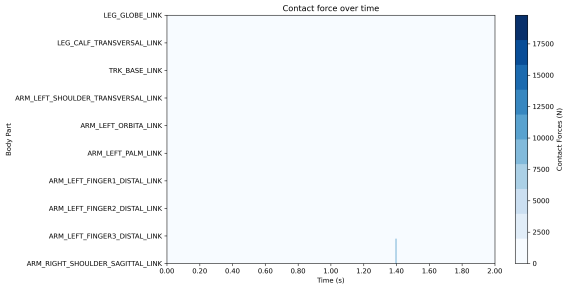
Here, we present the distribution of contact forces across the different robot bodies during the fall, for the various approaches and configurations considered. Body segments that did not experience a contact force of at least $100\,N$ are omitted. The reported contact forces correspond to the z-axis component. For each approach and configuration, the environment exhibiting the maximum contact force across ten test environments is considered.



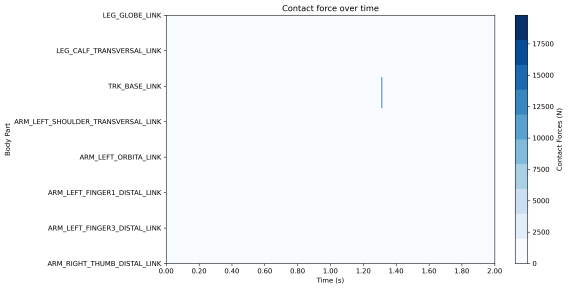
(a) Baseline Reward Configuration



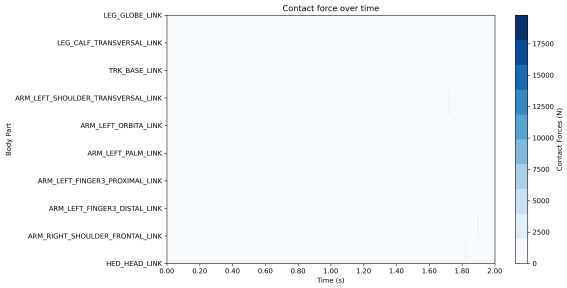
(b) Momentum-Based Reward Augmentation



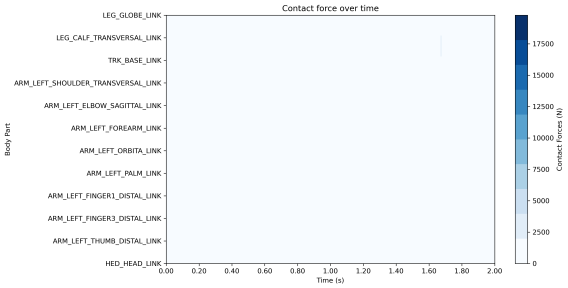
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

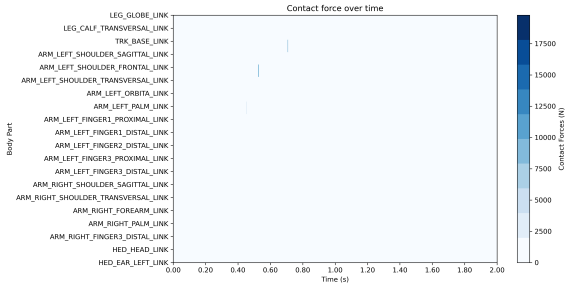


(e) Controller Frequency Configuration: 25 Hz

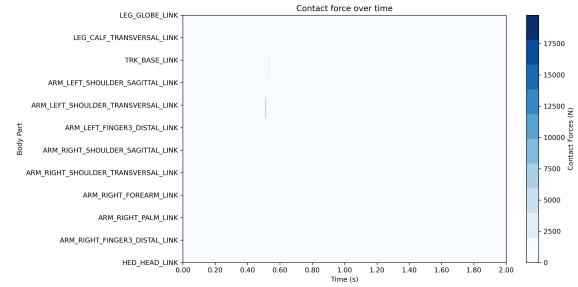


(f) Controller Frequency Configuration: 12.5 Hz

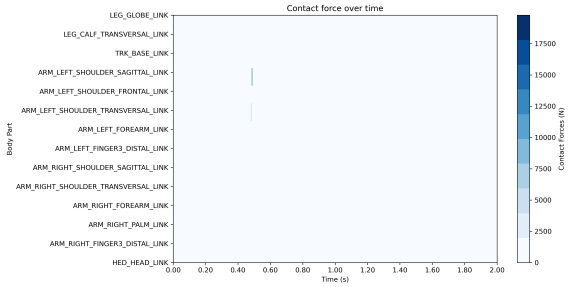
Figure A.1: Contact force distributions for each body part across the different approaches, in the case where the policy was trained with both arms along the body.



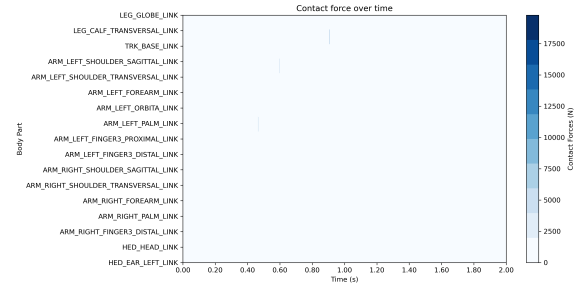
(a) Baseline Reward Configuration



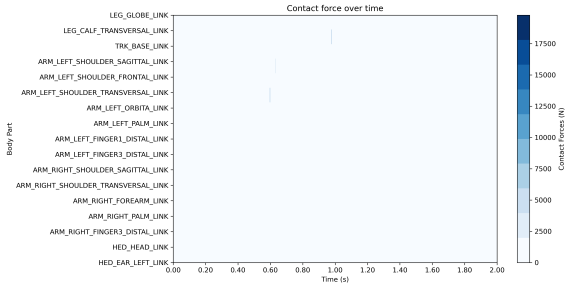
(b) Momentum-Based Reward Augmentation



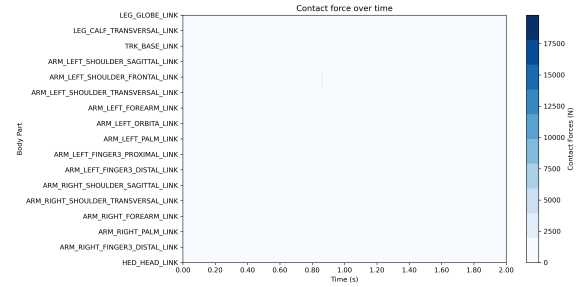
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

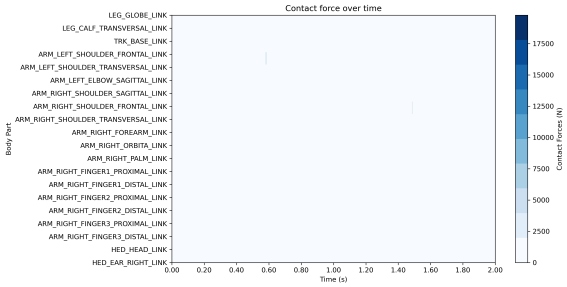


(e) Controller Frequency Configuration: 25 Hz

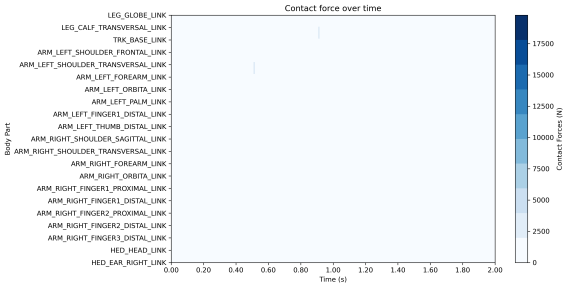


(f) Controller Frequency Configuration: 12.5 Hz

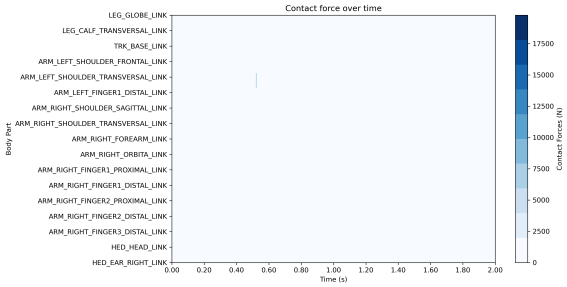
Figure A.2: Contact force distributions for each body part across the different approaches, in the case where the policy was trained with the left arm bent.



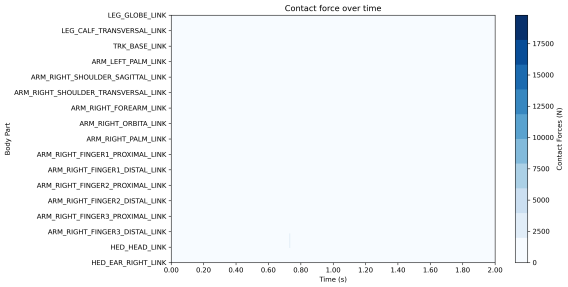
(a) Baseline Reward Configuration



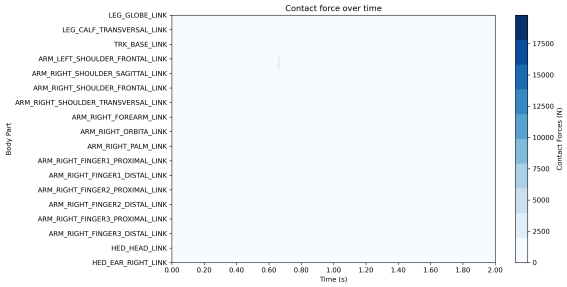
(b) Momentum-Based Reward Augmentation



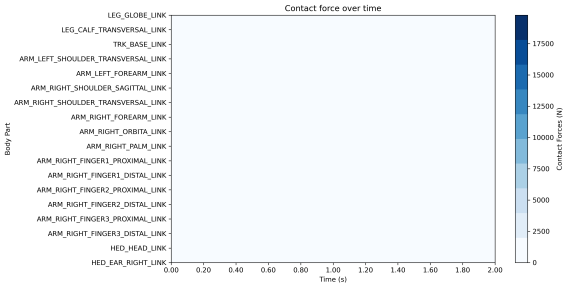
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

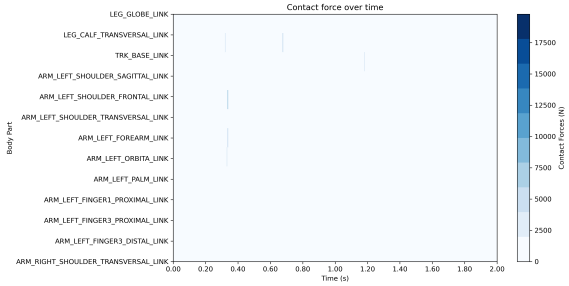


(e) Controller Frequency Configuration: 25 Hz

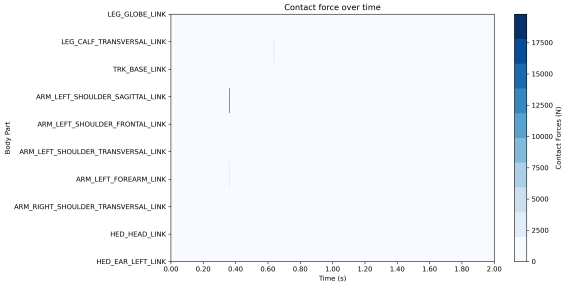


(f) Controller Frequency Configuration: 12.5 Hz

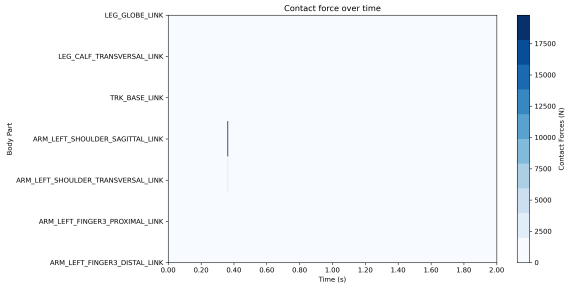
Figure A.3: Contact force distributions for each body part across the different approaches, in the case where the policy was trained with right arm bent.



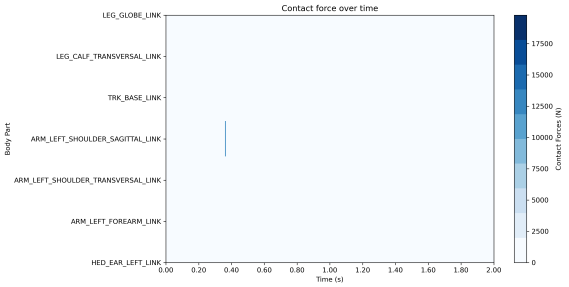
(a) Baseline Reward Configuration



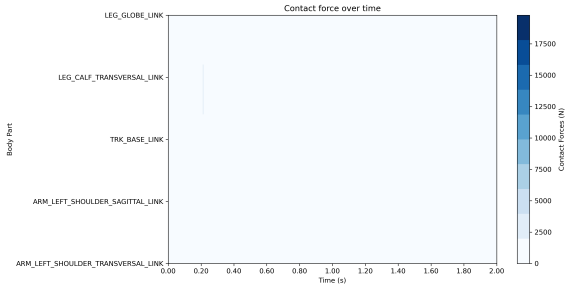
(b) Momentum-Based Reward Augmentation



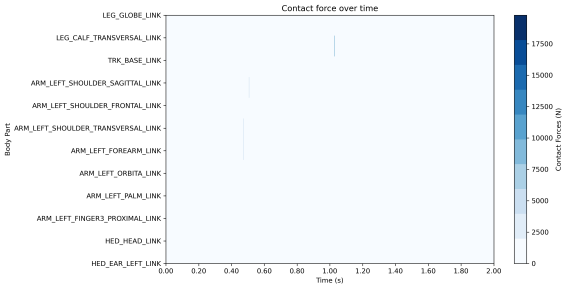
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

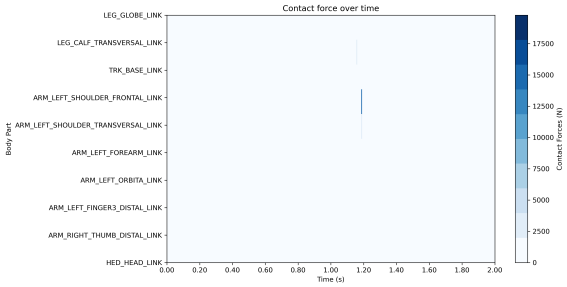


(e) Controller Frequency Configuration: 25 Hz

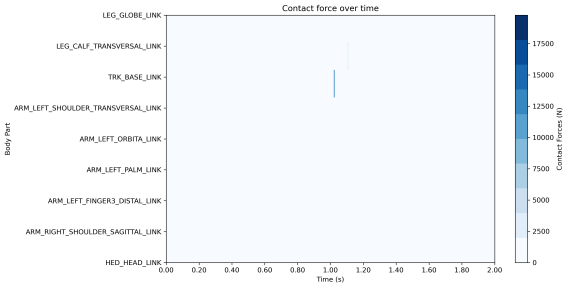


(f) Controller Frequency Configuration: 12.5 Hz

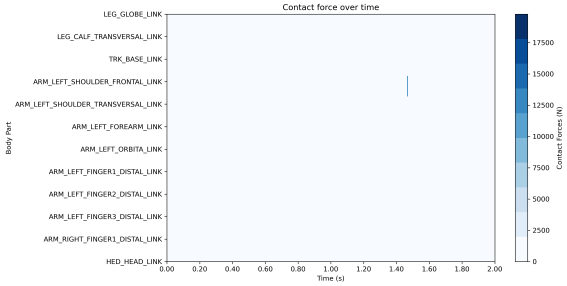
Figure A.4: Contact force distributions for each body part across the different approaches, in the case where the policy was trained with both arms bent.



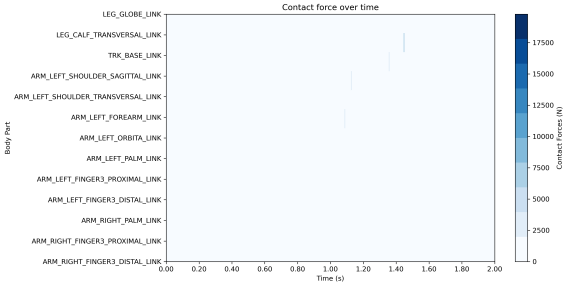
(a) Baseline Reward Configuration



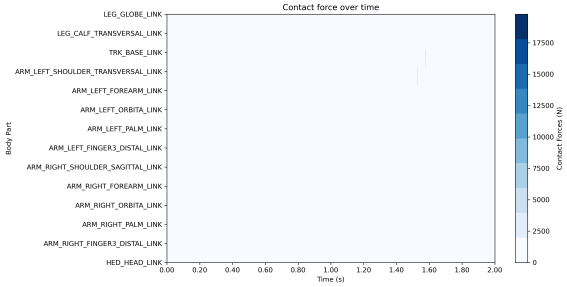
(b) Momentum-Based Reward Augmentation



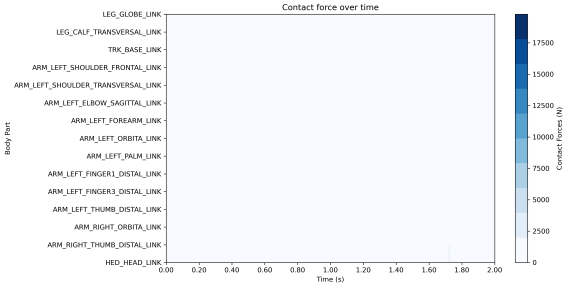
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

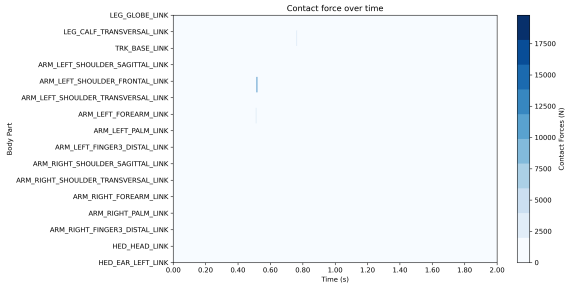


(e) Controller Frequency Configuration: 25 Hz

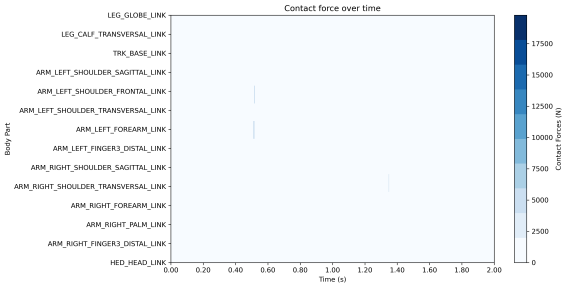


(f) Controller Frequency Configuration: 12.5 Hz

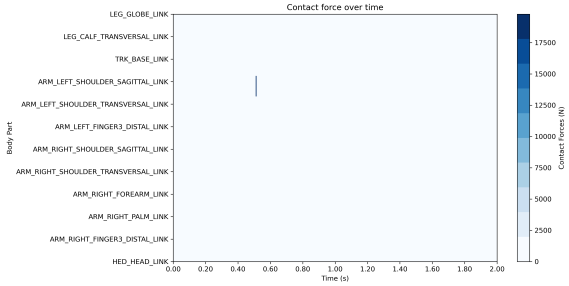
Figure A.5: Contact force distributions for each body part across the different approaches, in the case where the policy was trained for the mixed scenario tested on both arms along the body.



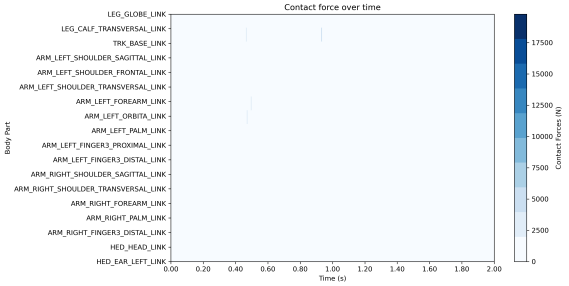
(a) Baseline Reward Configuration



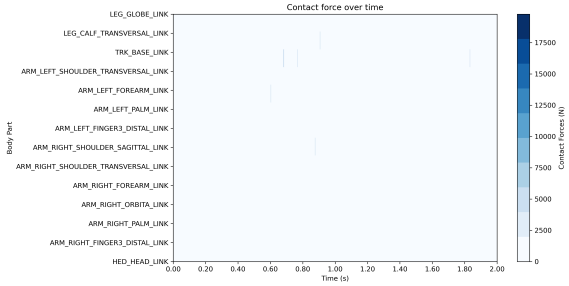
(b) Momentum-Based Reward Augmentation



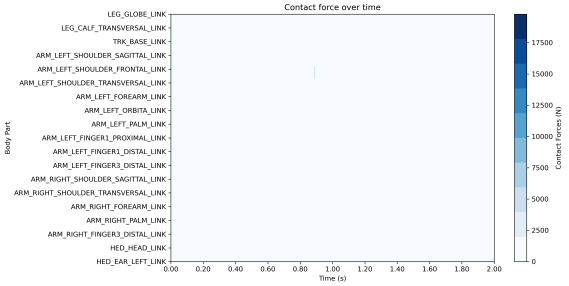
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

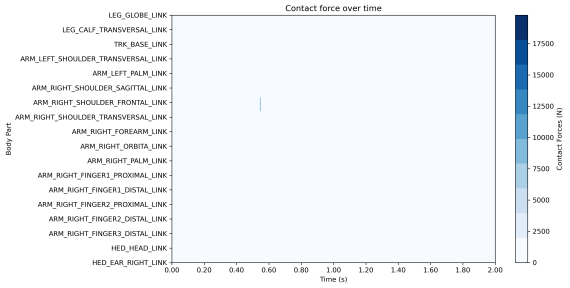


(e) Controller Frequency Configuration: 25 Hz

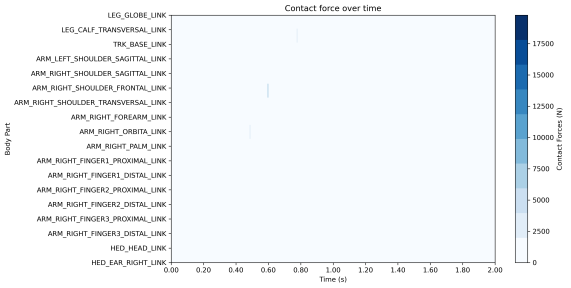


(f) Controller Frequency Configuration: 12.5 Hz

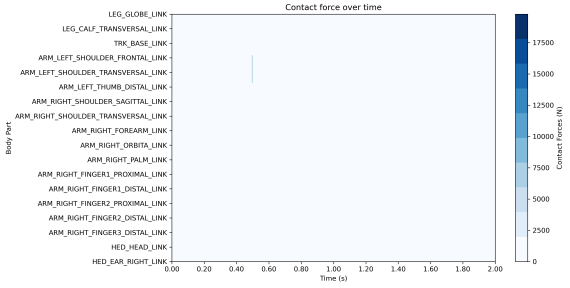
Figure A.6: Contact force distributions for each body part across the different approaches, in the case where the policy was trained for the mixed scenario tested on the left arm bent.



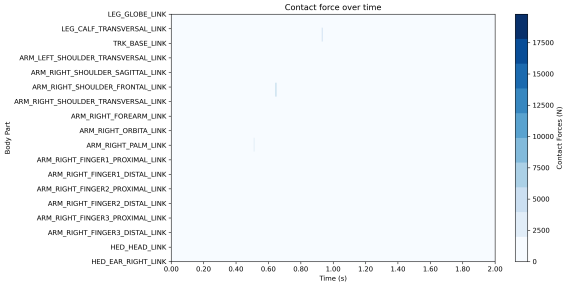
(a) Baseline Reward Configuration



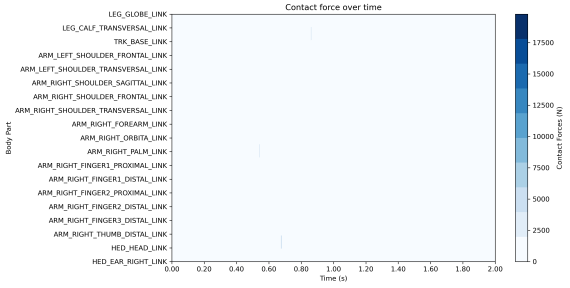
(b) Momentum-Based Reward Augmentation



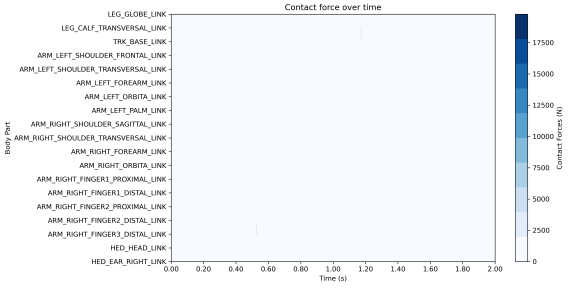
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation

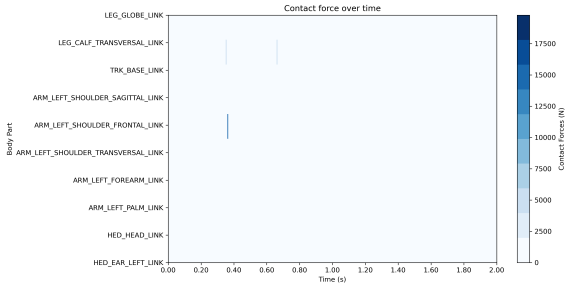


(e) Controller Frequency Configuration: 25 Hz

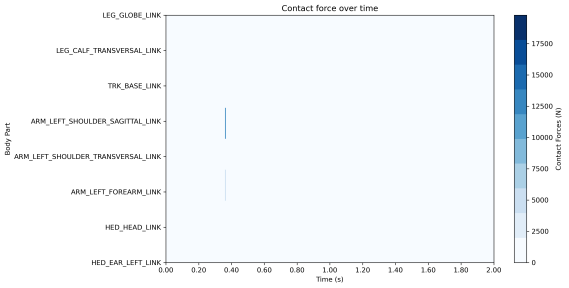


(f) Controller Frequency Configuration: 12.5 Hz

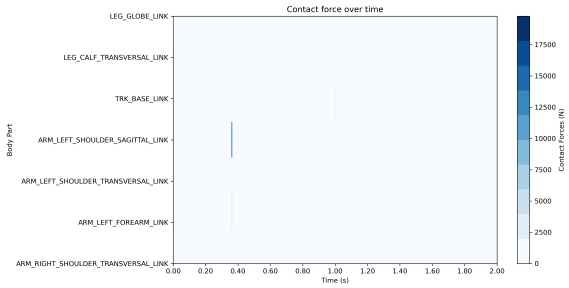
Figure A.7: Contact force distributions for each body part across the different approaches, in the case where the policy was trained for the mixed scenario tested on the right arm bent.



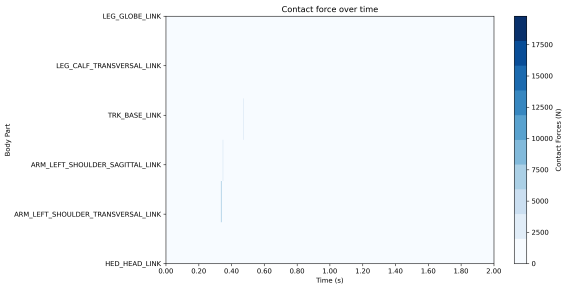
(a) Baseline Reward Configuration



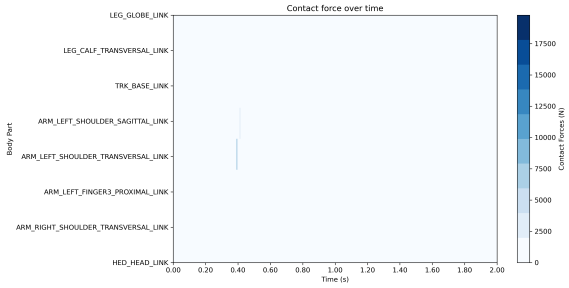
(b) Momentum-Based Reward Augmentation



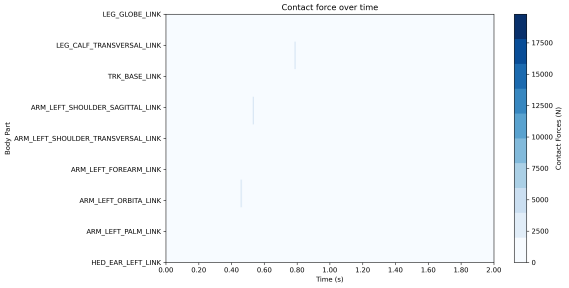
(c) Mean-Based Reward Reformulation



(d) Dense Reward Formulation



(e) Controller Frequency Configuration: 25 Hz



(f) Controller Frequency Configuration: 12.5 Hz

Figure A.8: Contact force distributions for each body part across the different approaches, in the case where the policy was trained for the mixed scenario tested on both arms bent.

Appendix B

Falling sequence for the different approaches tested in Chapter 4

In this appendix, we present the different falling behaviors observed across the various approaches tested in Chapter 4, considering a single starting position. The starting position examined is with both arms stretched out in front of the body.

Figure B.1 illustrates the falling behavior when no policy is applied. The robot first impacts both hands and subsequently rolls over.

Conversely, Figure B.2 shows that, with the multi-policy approach, after impacting the hands, the robot is able to stabilize itself, thereby mitigating the fall and reducing potential damage.

For all other approaches, the policy is not able to effectively reduce the damage. Figures B.3 B.4 B.5 show that in these cases the robot rolls over after the fall.

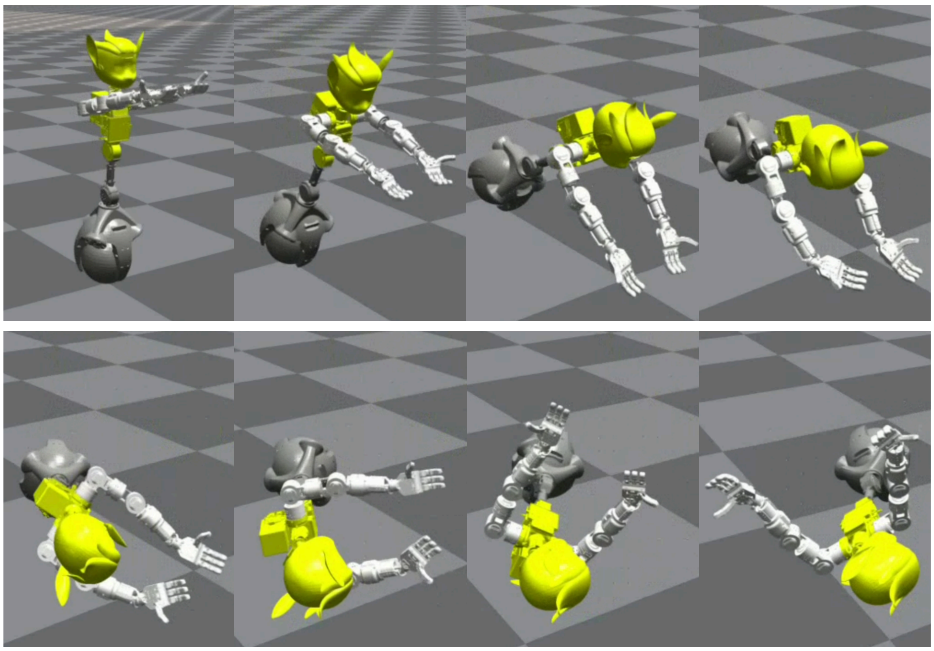


Figure B.1: Temporal sequence of the ballbot falling from the starting position with both arms outstretched, without any control policy.

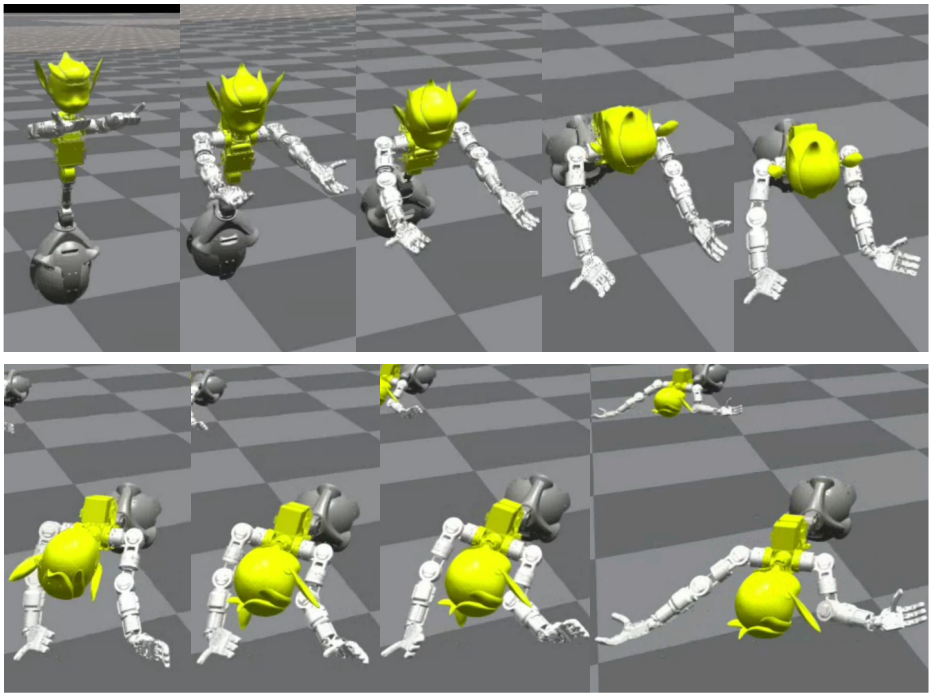


Figure B.2: Temporal sequence of the ballbot falling from the starting position with both arms outstretched, with the multi-policy approach.

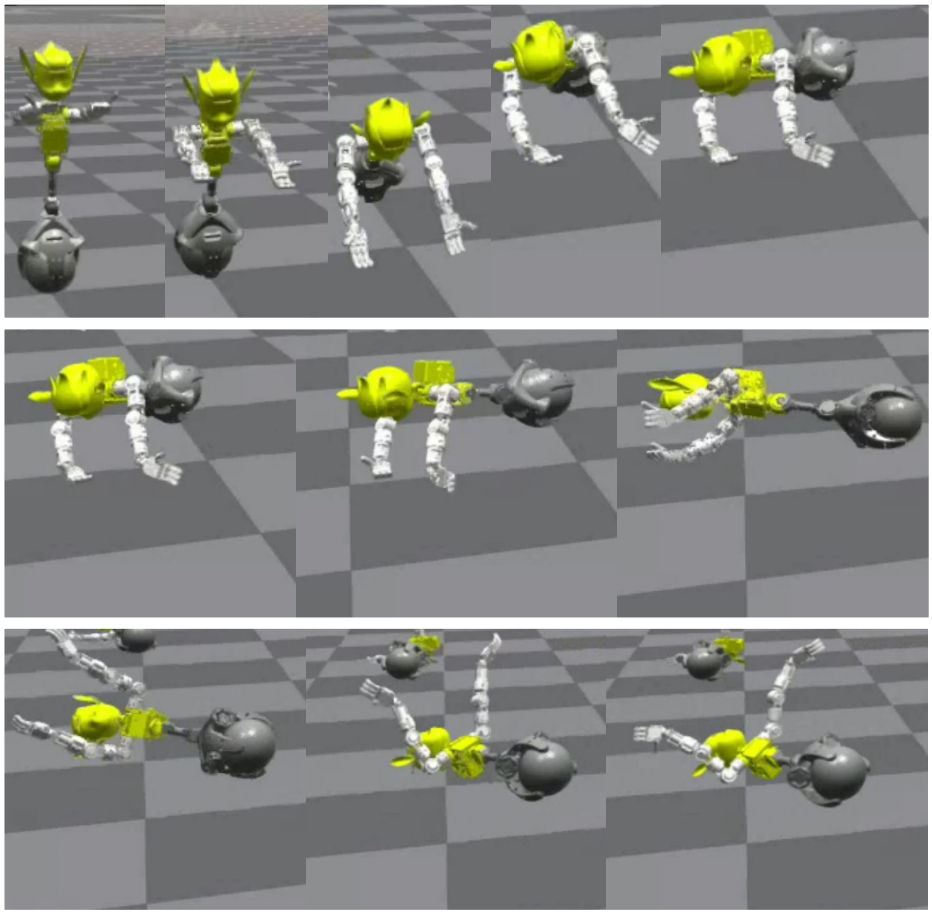


Figure B.3: Temporal sequence of the ballbot falling from the starting position with both arms outstretched, with the single-policy approach with clustering information.

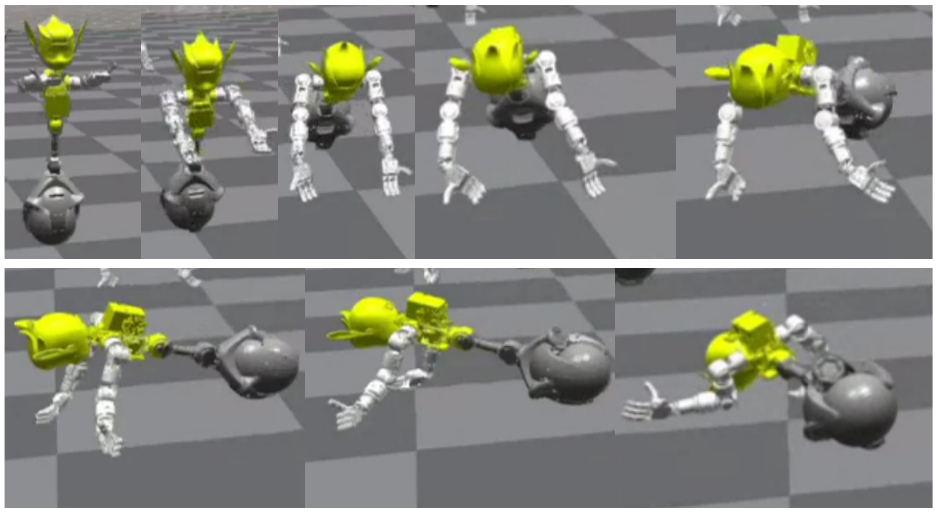


Figure B.4: Temporal sequence of the ballbot falling from the starting position with both arms outstretched, with the single-policy approach without clustering information.

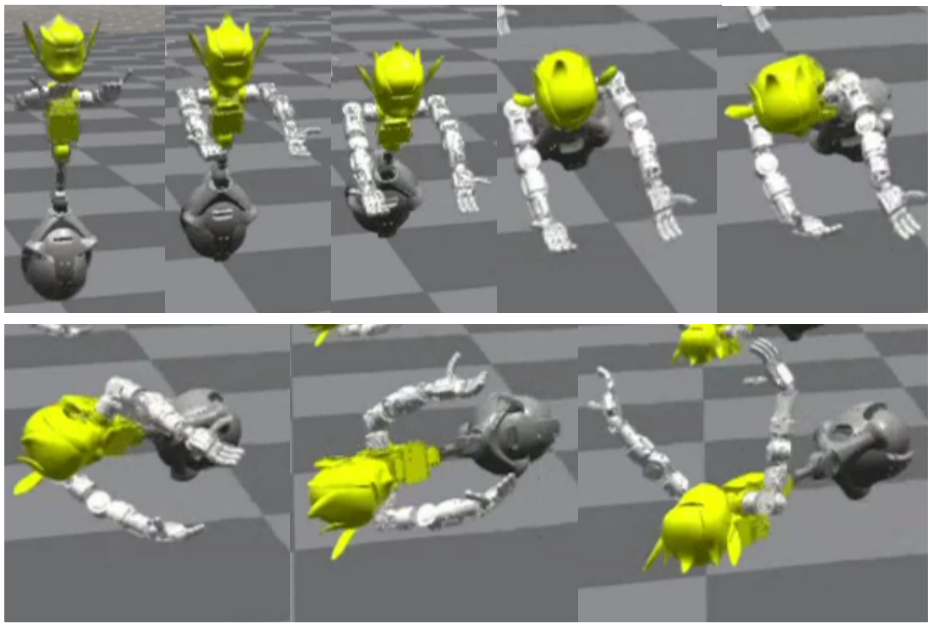


Figure B.5: Temporal sequence of the ballbot falling from the starting position with both arms outstretched, with the curriculum learning approach.

Bibliography

[1] Subham Agrawal, Marlene Wessels, Jorge de Heuvel, Johannes Kraus, and Maren Bennewitz. Sound Matters: Auditory Detectability of Mobile Robots. In *2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN)*, pages 2233–2239. IEEE, 2024.

[2] Snobin Antony, Raghi Roy, and Yaxin Bi. Q-learning: Solutions for grid world problem with forward and backward reward propagations. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 266–271. Springer, 2023.

[3] Gabriel Caldas Barros e Sá and Charles Andrye Galvão Madeira. Deep reinforcement learning in real-time strategy games: a systematic literature review. *Applied Intelligence*, 55(4):243, 2025.

[4] Marko Bjelonic, C Dario Bellicoso, Yvain de Viragh, Dhionis Sako, F Dante Tresoldi, Fabian Jenelten, and Marco Hutter. Keep rollin'—whole-body motion control and planning for wheeled quadrupedal robots. *IEEE Robotics and Automation Letters*, 4(2):2116–2123, 2019.

[5] JW Blonk. Modeling and control of a ball-balancing robot. Master's thesis, University of Twente, 2014.

[6] Chengtao Cai, Jiaxin Lu, and Zuoyong Li. Kinematic analysis and control algorithm for the ballbot. *IEEE Access*, 7:38314–38321, 2019.

[7] Arthur Charpentier, Romuald Elie, and Carl Remlinger. Reinforcement learning in economics and finance. *Computational Economics*, 62(1):425–462, 2023.

[8] D. Chugo, K. Kawabata, H. Kaetsu, H. Asama, and T. Mishima. Development of omnidirectional vehicle with step-climbing ability. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 3, page 3849–3854 vol.3, sept 2003.

[9] Djork-Arné Clevert. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

[10] Leander Diaz-Bone, Marco Bagatella, Jonas Hübotter, and Andreas Krause. DISCOVER: Automated Curricula for Sparse-Reward Reinforcement Learning. *arXiv preprint arXiv:2505.19850*, 2025.

[11] Cătălin Dosoftei, Vasile Horga, Ioan Doroftei, Tudor Popovici, and Ştefan Custura. Simplified Mecanum Wheel Modelling using a Reduced Omni Wheel Model for Dynamic Simulation of an Omnidirectional Mobile Robot. In *2020 International Conference and Exposition on Electrical and Power Engineering (EPE)*, pages 721–726, 2020.

[12] Tatsuro Endo and Yoshihiko Nakamura. An omnidirectional vehicle on a basketball. In *ICAR'05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pages 573–578. IEEE, 2005.

[13] Peter Fankhauser and Corsin Gwerder. Modeling and control of a ballbot. B.S. thesis, Eidgenössische Technische Hochschule Zürich, 2010.

[14] Mohammad Farajtabar and Marie Charbonneau. The path towards contact-based physical human–robot interaction. *Robotics and Autonomous Systems*, 182:104829, 2024. ISSN 0921-8890.

[15] Kiyoshi Fujiwara, Fumio Kanehiro, Shuuji Kajita, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. UKEMI: Falling motion control to minimize damage to biped humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2521–2526. IEEE, 2002.

[16] Enric Galceran, Alexander G Cunningham, Ryan M Eustice, and Edwin Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. *Autonomous Robots*, 41:1367–1382, 2017.

[17] Siddhant Gangapurwala, Luigi Campanaro, and Ioannis Havoutis. Learning low-frequency motion control for robust and dynamic robot locomotion. *arXiv preprint arXiv:2209.14887*, 2022.

[18] Yifeng Gong, Ge Sun, Aditya Nair, Aditya Bidwai, Raghuram CS, John Grezmak, Guillaume Sartoretti, and Kathryn A Daltorio. Legged robots for object manipulation: A review. *Frontiers in Mechanical Engineering*, 9:1142421, 2023.

[19] Ambarish Goswami, Seung-kook Yun, Umashankar Nagarajan, Sung-Hee Lee, KangKang Yin, and Shivaram Kalyanakrishnan. Direction-changing fall control of humanoid robots: theory and experiments. *Autonomous Robots*, 36:199–223, 2014.

[20] Sehoon Ha and C Karen Liu. Multiple contact planning for minimizing damage of humanoid falls. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2761–2767. IEEE, 2015.

[21] Mohammed Hossny and Julie Iskander. Biomechanic Posture Stabilisation via Iterative Training of Multi-policy Deep Reinforcement Learning Agents. *arXiv preprint arXiv:2008.12210*, 2020.

[22] Stanislav Hristov Ivanov, Craig Webster, and Katerina Berezina. Adoption of robots and service automation by tourism and hospitality companies. *Revista Turismo & Desenvolvimento*, 27(28):1501–1517, 2017.

[23] Thomas Kølbæk Jespersen. Kugle-modelling and control of a ball-balancing robot. *Master Thesis*, 2019.

[24] Ichiro Kato, Sadamu Ohteru, Katsuhiko Shirai, Toshiaki Matsushima, Seinosuke Narita, Shigeki Sugano, Tetsunori Kobayashi, and Eizo Fujisawa. The robot musician 'wabot-2'(waseda robot-2). *Robotics*, 3(2):143–155, 1987.

[25] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[26] Vijay R. Konda and John N. Tsitsiklis. Actor-Critic Algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014, 2000.

[27] Masaaki Kumagai and Takaya Ochiai. Development of a robot balancing on a ball. In *2008 International Conference on Control, Automation and Systems*, pages 433–438, 2008.

[28] Visak C. V. Kumar, Sehoon Ha, and C. Karen Liu. Learning a unified control policy for safe falling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3940–3947, 2017.

[29] Tom Lauwers, George Kantor, and Ralph Hollis. One is enough! In *Robotics Research: Results of the 12th International Symposium ISRR*, pages 327–336. Springer, 2007.

[30] Tom B Lauwers, George A Kantor, and Ralph L Hollis. A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2884–2889. IEEE, 2006.

[31] Qingqing Li, Xuechao Chen, Yuhang Zhou, Zhangguo Yu, Weimin Zhang, and Qiang Huang. A minimized falling damage method for humanoid robots. *International Journal of Advanced Robotic Systems*, 14(5):1729881417728016, 2017.

[32] Yixuan Li, Yuhao Wang, Gan Wang, and Tingna Shi. A Multi-Policy Framework for Manipulator Trajectory Tracking Based on Feature Extraction and RL Compensation. In *International Conference on Automation, Robotics and Applications*, pages 191–195. IEEE, 2024.

[33] Zhongyu Li and Ralph Hollis. Toward a ballbot for physically leading people: A human-centered approach. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4827–4833. IEEE, 2019.

[34] Stuart Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[35] Yuntao Ma, Farbod Farshidian, and Marco Hutter. Learning arm-assisted fall damage reduction and recovery for legged mobile manipulators. In *International Conference on Robotics and Automation*, pages 12149–12155. IEEE, 2023.

[36] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning, 2021.

[37] Glenn McCartney and Andrew McCartney. Rise of the machines: towards a conceptual service-robot research framework for the hospitality and tourism industry. *International Journal of Contemporary Hospitality Management*, 32(12): 3835–3851, 2020.

[38] Masahiro Mori, Karl F MacDorman, and Norri Kageki. The uncanny valley [from the field]. *IEEE Robotics & automation magazine*, 19(2):98–100, 2012.

[39] Umashankar Nagarajan, George Kantor, and Ralph L. Hollis. Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot. In *2009 IEEE International Conference on Robotics and Automation*, page 3743–3748, May 2009.

[40] Umashankar Nagarajan, Anish Mampetta, George A. Kantor, and Ralph L. Hollis. State transition, balancing, station keeping, and yaw control for a dynamically stable single spherical wheel mobile robot. In *2009 IEEE International Conference on Robotics and Automation*, page 998–1003, May 2009.

[41] Umashankar Nagarajan, George Kantor, and Ralph Hollis. The ballbot: An omnidirectional balancing mobile robot. *International Journal of Robotics Research*, 33(6):917–930, 2014.

[42] Abdelrahman Nashat, Abdelrahman Morsi, Mohamed M. M. Hassan, and Mustafa Abdelrahman. Ballbot Simulation System: Modeling, Verification, and Gym Environment Development. In *2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 198–204, 2023.

[43] Ngoc Duy Nguyen, Thanh Nguyen, and Saeid Nahavandi. Multi-agent behavioral control system using deep reinforcement learning. *Neurocomputing*, 359:58–68, 2019.

[44] James R Norris. *Markov chains*. Number 2. Cambridge University Press, 1998.

[45] Yoshito Okada, Kazuya Oguma, Kenta Gunji, Yoshiki Yokota, Hanif Aryadi, Shotaro Kojima, Ranulfo Bezerra, Masashi Konyo, Kazunori Ohno, and Satoshi Tadokoro. Fast and accurate simulation of mecanum wheels with passive rollers emulated by fixed joints and anisotropic friction. In *2023 21st International Conference on Advanced Robotics (ICAR)*, pages 592–598. IEEE, 2023.

[46] Dinh Ba Pham, Hyung Kim, Jaejun Kim, and Soon-Geul Lee. Balancing and transferring control of a ball segway using a double-loop approach [applications of control]. *IEEE Control Systems Magazine*, 38(2):15–37, 2018.

[47] Joel Roos and Marco Sütterlin. Rezero with arm: A three degrees of freedom manipulator balancing on a ball. Master's thesis, ETH Zurich, 2017.

[48] Javier Ruiz-del Solar, Rodrigo Palma-Amestoy, Román Marchant, Isao Parra-Tsunekawa, and Pablo Zegers. Learning to fall: Designing low damage fall sequences for humanoid soccer robots. *Robotics and Autonomous Systems*, 57(8): 796–807, 2009.

[49] Achkan Salehi. Reinforcement Learning for Ballbot Navigation in Uneven Terrain. *arXiv preprint arXiv:2505.18417*, 2025.

[50] Vincent Samy, Karim Bouyarmane, and Abderrahmane Kheddar. QP-based adaptive-gains compliance control in humanoid falls. In *IEEE International Conference on Robotics and Automation*, pages 4762–4767. IEEE, 2017.

[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[52] Emrah Akin Sisbot, Luis F. Marin-Urias, Rachid Alami, and Thierry Simeon. A Human Aware Mobile Robot Motion Planner. *IEEE Transactions on Robotics*, 23 (5):874–883, 2007.

[53] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565, 2022.

[54] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[55] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[56] Yuchuang Tong, Haotian Liu, and Zhengtao Zhang. Advancements in humanoid robots: A comprehensive review and future prospects. *IEEE/CAA Journal of Automatica Sinica*, 11(2):301–328, 2024.

[57] Yikai Wang, Mengdi Xu, Guanya Shi, and Ding Zhao. Guardians as You Fall: Active Mode Transition for Safe Falling. *arXiv preprint arXiv:2310.04828*, 2023.

[58] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3): 279–292, 1992.

[59] Marvin Wiedemann. Simulation of Highly Dynamic Omnidirectional Robots in Isaac Sim. Slides presented at ROSCon 2023, 2023. Accessed 2025-12-14.

[60] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[61] Yifan Zhou, Jianghao Lin, Shuai Wang, and Chong Zhang. Learning Ball-Balancing Robot through Deep Reinforcement Learning. In *2021 International Conference on Computer, Control and Robotics (ICCCR)*, page 1–8, January 2021.

La borsa di dottorato è stata cofinanziata con risorse dell'Unione europea-*NextGeneration EU*
Piano Nazionale di Ripresa e Resilienza (PNRR)