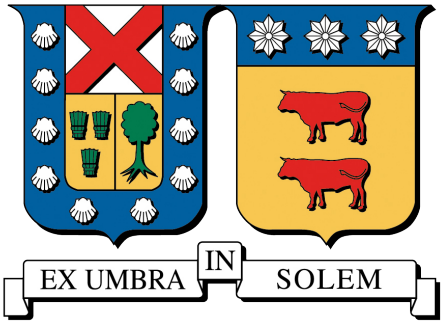
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



ADAPTING BIPEDAL NEURO-MOTOR POLICIES ON PLANNED FOOTSTEPS

MIGUEL YERÓN ROJAS SÁNCHEZ

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA ELECTRÓNICA

PROFESOR GUÍA : Dr. Werner Creixell

Abril 2024

TÍTULO DE LA TESIS:

ADAPTING BIPEDAL NEURO-MOTOR POLICIES ON PLANNED FOOTSTEPS

AUTOR:

Miguel Yerón Rojas Sánchez

TRABAJO DE TESIS, presentado en cumplimiento parcial de los requisitos para el grado de Magíster en Ciencias de la Ingeniería Electrónica de la Universidad Técnica Federico Santa María.

Dr. Werner Creixell _____

Dr. Mauricio Araya _____

Dr. Juan Pablo Vásconez _____

Valparaíso, Abril de 2024.

Dedicado a mi familia
y amigos

ABSTRACT

This study investigates a hierarchical reinforcement learning approach to achieve human-like walking in bipedal robots while following marked footsteps. Traditionally, state machines and model-based methods were used for this task, ensuring stability and safety but lacking natural human-like motion. Our approach utilizes a two-level architecture: a high-level policy trained specifically for following footsteps and a low-level policy distilled from motion capture data to generate natural gaits. Experiments demonstrate that this hierarchical approach significantly outperforms training a single network, particularly for complex tasks on human-sized robots.

The low-level network plays a crucial role, substantially reducing joint torques and speeds while achieving stable walking. However, a current limitation is the inability to follow footsteps on stairs. We observed that both general and locomotion motion capture datasets achieved similar results in following footsteps, but the locomotion dataset generated more visually natural human-like walking, especially for forward walking.

Future work will aim to improve the robot's walking robustness for navigating uneven terrains like stairs and slopes. Our findings suggest that low-level networks pre-trained on motion capture data are a viable approach for achieving human-like walking gaits in real-world, human-sized robots. This research paves the way for developing bipedal robots with efficient and natural walking capabilities.

Accompanying videos[1] and code[2] are available online.

Keywords. Bipedal locomotion, deep reinforcement learning, hierarchical networks, human-like motion, motion capture.

[1] https://www.youtube.com/watch?v=QlcFHNlCwUA
[2] https://github.com/mrojasanchez/paper_rojas_werner_2024_ieeeral_walkingmocapact

Contents

List of Figures

1 | Introduction

Legged locomotion has long been one of the central topics in robotics, as it incorporates the appeal of replicating biological-like movements and, on the other hand, is a formidable technical challenge. The progress in the development of hardware and control techniques has allowed the creation of numerous robotic platforms, both quadrupedal and bipedal (1, 2, 3, 4).

Initially, conventional motion planning and linear control techniques were utilized. Nevertheless, these methods are insufficient for scenarios where the environment constantly changes or the system is quite complex. As a result, there has been a shift towards more sophisticated and adaptable approaches. Advanced model-based control techniques tackle mainly the problem of stable and balanced locomotion, optimization of energy consumption, adaptation to different terrains, and coordination of complex movements in legged robots. Currently, the use of model predictive control (MPC) techniques is a subject of extensive study since it allows the generation of trajectories in a non-linear domain, considering kinematic ranges, torque limits, and to some extent, is robust to perturbations by frequently doing motion replanning. However, this versatility is possible at a high computational cost since it is necessary to solve a discontinuous and highly dimensional optimization problem (5).

Researchers are diligently addressing these challenges, leveraging multidisciplinary approaches to augment the capabilities of legged robots. The ultimate goal is to make these robots functional and capable of moving in a manner that mirrors the agility of their biological counterparts. As the demand for legged robots in various real-world applications continues to grow, from search and rescue missions to industrial applications (6, 7, 8), the need for more versatile and adaptable control techniques becomes increasingly urgent (9).

In the field of learning-based methods, Deep Reinforcement Learning (DRL) is a new paradigm that is revolutionizing robotics. This method allows robots to obtain a controller, in this context called policy, which changes iteratively through rewards and observations that the environment gives it every time it executes actions on it. Ultimately, this allows them to learn complex behaviors autonomously.

Nowadays, DRL research has captured the attention of fields such as computer graphics and simulated control applications. There are notable examples of successful implementation of complex behavior, like the emergence of motor skills through simple rewards (10, 11) and the coordination of multiple agents in cooperative tasks (12, 13).

Regarding learning-based legged locomotion, quadruped robots have made significant strides in walking with minimal data, whether in a simulated or real-world environment. They have successfully transferred the knowledge acquired from simulations to reality. This could be attributed to their configuration, which provides them greater stability and resilience to external perturbations (14, 15, 16).

In contrast, bipedal locomotion with human-sized robots has made some progress. However, it still needs to improve the level of agility seen in quadruped robots, seeing works where the control is limited to only the leg's joints, which reduces the system's complexity when training a policy. Thus, the generated motions are far from human-like, as reported in (17, 18). Success cases with human-like robots on real hardware appear on the same page, but these cases are limited to small-size robots with few degrees of freedom (19, 20, 21).

On the other hand, many advances in motion generation use motion capture, but the uses focus on computer graphics applications (22, 23, 24). In addition, it could be infeasible to apply these learned controllers in real robots since the movements generated go beyond their mechanical capabilities (25, 26).

One task a real robot must perform is the ability to follow the user's marked footsteps, which indicates where the robot has to place each foot along a path. In this scenario, the controller must be able to adapt to different walking modes and environments.

The classic method to accomplish this task is to use model-based controllers and a finite state machine, which indicates the step plan to be followed by the controller, whether moving forward, backward, going in curves, or standing in place. This architecture, linear controller plus state machine, produces a safe and predictable behavior applicable to real robots. However, the resulting movement is far from what one could consider normal human walking.

Can a humanoid robot be trained to follow marked footsteps while maintaining a human-like motion?

In this study, we investigate the impact of employing hierarchical networks, which incorporate a pre-trained low-level network with motion captures, for the task of following marked footsteps. Our findings demonstrate that:

1. Integrating a pre-trained low-level network with motion captures accomplishes the task of following marked footsteps.
2. It produces human-like movement by not restricting to only joint legs.
3. The quality of the generated movement is dependent on the size and actions comprising the motion capture dataset.

The proposed architecture exhibits versatility in executing various walking modes, including straightforward, backward, sideways, curved walking, and maintaining a stationary position. The results consistently highlight the effectiveness of the hierarchical network in achieving a broad range of walking behaviors.

2 | Research Method

2.1 Hypothesis

"It is possible to generate a human-like motion on a human-sized robot by integrating a low-level policy based on motion capture to follow marked footsteps."

2.2 Objectives

The main goal of the *"Adapting bipedal neuro-motor policies on planned footsteps"* thesis is:

1) Develop a hierarchical reinforcement learning approach for bipedal walking robots to follow marked footsteps.
2) Leverage motion capture data to create a low-level policy for natural walking gaits.
3) Compare the effectiveness of a single-network vs. a two-level hierarchical network for complex walking tasks.

2.3 Methodology

The methodology to complete each objective consists of:

1. **High-level policy training**:
   - Employ a Multi-layer Perceptron (MLP) architecture for actor and critic networks.
   - Use of curriculum learning for gradually increasing walking difficulty.
   - Train the high-level policy using the PPO (Proximal Policy Optimization) reinforcement learning algorithm.
2. **Low-level policy based on motion capture**:
   - Leverage the use of the MoCapAct dataset which provides motion capture data and pre-trained low-level networks.
   - Integration of the low-level network's decoder as part of the environment for the high-level policy.
3. **Evaluation**:
   - Evaluate different walking modes including forward, backward, sideways, and curved walking on flat terrain.
   - Comparison of training performance (mean episode duration and reward) between single and hierarchical networks.
   - Analysis of foot velocities, contact forces, joint torques, and maximum speeds during walking.

- Visual comparison of walking gaits generated using different motion capture datasets (general vs locomotion).

3 | Related work

3.1 RL on two-legged robots

In (27), a single controller is trained to walk on marked footsteps on the Cassie robot. During the training phase, rewards are designed to promote cyclic walking by encouraging it to follow a two-phase clock signal in conjunction with randomized footsteps. The controller is an LSTM network that delivers the setpoints of each joint to a PD controller inside the Cassie robot. The observation comprises proprioceptive data, the two-phase clock signal, and randomized footstep commands indicating where to place the foot. Building on their previous work, in (28), they demonstrate controlled walking using a hierarchical 1-step control policy and a reachability prediction model. This hierarchical policy comprises a low-level feed-forward network that aims to receive the step commands, the two-phase signals, and a latent space from an LSTM network that receives the robot's proprioceptive states. Note that the control of this type of robot is less problematic than a humanoid robot since they have many more degrees of freedom, and their mechanical characteristics do not allow them to be agile.

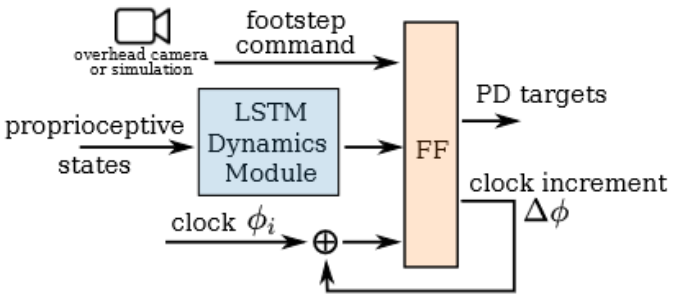


Figure 3.1: Hierarchical policy for the Cassie robot for the task of following marked footsteps (28).

3.2 Locomotion policies for humanoids

Studies, where the policy is composed of a single network for a given task, can be seen in works like in (17), where they study the impact of using different action spaces, i.e., position-based or torque-based, to control the human-sized TOCABI robot. The former uses a hierarchical architecture, where the high-level controller is a neural network that generates the desired position, which is fed to a low-level PD controller in charge of generating the torque commands. On the other hand, their proposed architecture uses only a network in charge of generating the torque commands. However, it needs a pre-training step based on imitating a whole-body controller to allow it to remain standing.

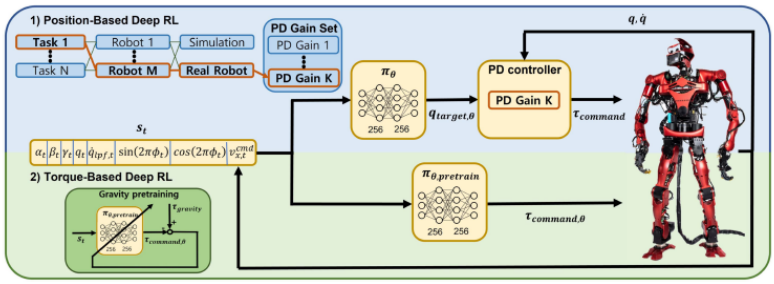


Figure 3.2: Control framework for the TOCABI robot using a single policy (17).

Similarly, in (18), a hierarchical model to follow marked footsteps is applied to the HRP-5P and JVRC-1 humanoid robots. The high-level policy receives information about the current state of the robot, two clock phase signals, and the next two steps to be reached from an external planning stage. The output of the high-level network is added to the robot's joint positions corresponding to the "half-sitting" position. These positions are converted to torque commands by the low-level PD controller, achieving the walk-on marked footsteps. However, the trained network is limited to controlling the legs' joints, leaving the trunk, head, and arm joints fixed, making the movement look unnatural.

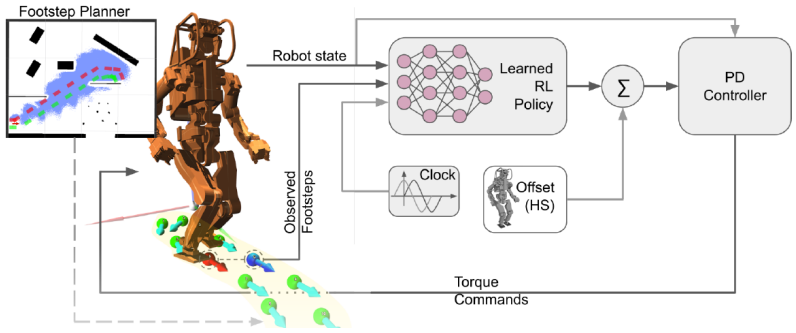


Figure 3.3: HRP-5P control framework. A single policy es trained to follow marked footsteps (18).

Lastly, in (19), using the small-sized OP3 robot directly in real scenarios, they study the emergence of walking behaviors using only proprioceptive information, including RGB vision. Then, in (20), they train on individual skills, compose them, and apply them in a 1v1 soccer game end-to-end. Although their results are encouraging, the policies learned in these studies have yet to apply to human-sized robots since their movements are unstable and unnatural.

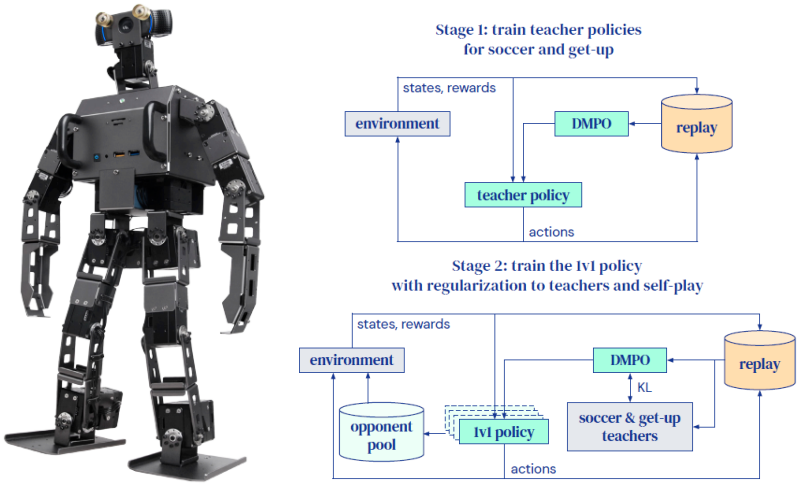


Figure 3.4: OP3 robot framework. Using off-policy training, the robot can explore its environment on the real hardware (19).

Our work is closely related to (21), where they employed imitation learning to train a hierarchical network using motion captures from dogs and humans for the ANYmal and OP3 robots. The low-level network was extracted, and a new high-level network was trained for a specific task involving performing actions smoothly and tracking a ball within an area. However, unlike our work, they did not specify which subset of motion captures they used; instead, they only mentioned that the low-level network was trained from walking and turning clip behaviors. Our work, on the other hand, aims to investigate how different subsets of the motion capture dataset impact the ability of a human-sized robot with 56 degrees of freedom to achieve human-like motion while following marked footsteps provided by a planning stage.

4 | Background

4.1 Reinforcement Learning Framework

Reinforcement learning is centered around solving a task through the agent's actions over the environment. The robot's behavior is influenced by the rewards (positive or negative) it receives over time. Thus, the agent's goal is to maximize its cumulative reward over many interactions with the environment.

Formally, the problem is stated as a discrete Markov Decision Process (MDP), composed by the tuple $\langle S, A, R, P, \rho_0 \rangle$ where $S \in \mathbb{R}^n$ is the set of valid states, $A \in \mathbb{R}^m$ the set of valid actions, $R : S \times A \times S \to \mathbb{R}$ the reward function, $P : S \times A \to \mathcal{P}(S)$ an unknown *a priori* state-transition probability function and ρ_0 the initial state distribution.

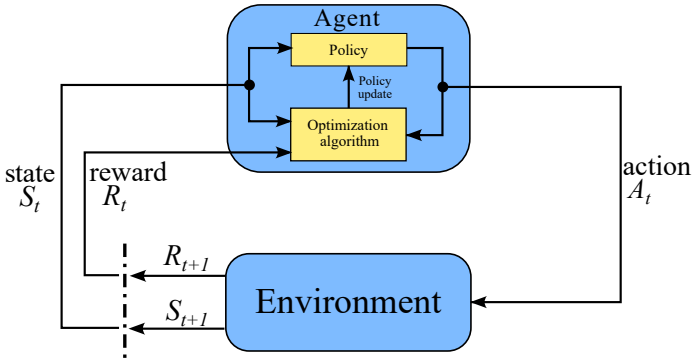


Figure 4.1: Agent-environment interaction loop (29). Depending on the optimization algorithm used, this diagram could get more elaborate.

When the agent interacts with the world, for each time step t, it receives a representation of the state of the environment $s_t \in S$, executes an action according to a policy $a_t \in A$, receiving in return a reward $r_t \in R$, and next state $s_{t+1} \in S$ according to the dynamics of the environment $P(s_{t+1}|s_t, a_t) \in \mathcal{P}$.

A policy can be deterministic $a_t = \mu_\theta(s_t)$ or stochastic $a_t \sim \pi_\theta(-|s_t)$, where θ is a set of parameters. In DRL, this parameterized policy usually comprises neural networks that can be adjusted to change their behavior through optimization algorithms.

A trajectory is a series of state-action pairs $\tau = (s_0, a_0, s_1, a_1, \dots)$ an agent makes according to a policy, where the initial state follows a distribution $s_0 \sim \rho_0(-)$ and the following transitions can be deterministic $s_{t+1} = f(s_t, a_t)$ or stochastic $s_{t+1} \sim P(-|s_t, a_t)$.

The reward is a function that depends on the current state, the action taken, and the next state, $r_t = R(s_t, a_t, s_{t+1})$, although it is usually simplified to depend only on the state, $r_t = R(s_t)$, or on the state-action pair, $r_t = R(s_t, a_t)$.

In the case of an infinite horizon, the agent's objective is to learn a policy that maximizes the

cumulative discounted return over a trajectory, i.e:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t. \tag{4.1}$$

Where $\gamma \in (0, 1)$. The γ parameter in equation 4.1 encourages the agent to seek immediate rewards instead of future rewards, and mathematically, allows the infinite sum (if $T \to \infty$) to converge.

By the Markov property, and assuming that the transition function of the environment and the policy are stochastic, the probability of realizing a T-step trajectory is defined as:

$$P(\tau|\pi_\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t). \tag{4.2}$$

Then, using (4.1) and (4.2), the expected return is as follows:

$$J(\pi_\theta) = \int_\tau P(\tau|\pi_\theta)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{4.3}$$

The goal is to optimize this cost function by finding the optimal policy:

$$\pi^* = \arg\max_\pi J(\pi_\theta) \tag{4.4}$$

The optimization is done iteratively by doing stochastic gradient ascent over the cost function (4.3) with respect to the current parameters θ_k.

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|_{\theta_k}. \tag{4.5}$$

4.2 Value functions

Value functions play a critical role in guiding an agent's decision-making process. They act as estimates of the long-term expected reward an agent can accumulate by taking a specific action in a given state, considering future rewards received along the way. These estimates are crucial for the agent to navigate the environment and learn optimal behavior.

There are four main types of value functions:

1. **State-value function**, $V^\pi(s)$: This function represents the expected discounted future reward an agent can receive starting from state 's' and following the current policy 'π'. In simpler terms, it estimates how good it is for the agent to be in a particular state, considering the potential rewards it can gather in the long run if it adheres to its current course of action (policy).

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

2. **Optimal State-value function**, $V^*(s)$: Which gives the expected return if you start in state s and act according to the optimal policy in the environment, i.e:

$$V^*(s) = \max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

3. **State-action value function**, $Q^\pi(s, a)$: This function goes a step further by estimating the expected discounted future reward the agent can expect by taking action 'a' in state 's' and following the current policy 'π'. It essentially tells the agent how good it is to take a specific action in a specific state, considering the immediate reward for that action and the potential future rewards it can lead to.

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

4. **Optimal State-action value function**, $Q^*(s, a)$: Which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment, i.e:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

By estimating these values, the agent can learn to choose actions that maximize its long-term reward within the environment. It essentially explores different state-action combinations, appraising each based on the value functions, and refines its policy over time to prioritize actions leading to higher expected rewards in the long run.

4.3 Bellman equations

The Bellman equations are fundamental concepts in reinforcement learning (RL) that describe the relationship between the value of a state or a state-action pair and the expected future rewards obtained from that state or action. They essentially provide a recursive formula for calculating these values.

There are four main Bellman equations, each focusing on a different aspect of value:

- **Recursive state-value function**, $V^\pi(s)$:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi,s'\sim P}[r(s, a) + \gamma V^\pi(s')]$$

- **Recursive optimal state-value function**, $V^*(s)$:

$$V^*(s) = \max_{a} \mathbb{E}_{s'\sim P}[r(s, a) + \gamma V^*(s')]$$

- **Recursive state-action value function**, $Q^\pi(s, a)$:

$$Q^\pi(s, a) = \mathbb{E}_{s'\sim P}[r(s, a) + \gamma \mathbb{E}_{a'\sim\pi}[Q^\pi(s', a')]]$$

- **Recursive optimal state-action value function**, $Q^*(s, a)$:

$$Q^*(s, a) = \mathbb{E}_{s'\sim P}[r(s, a) + \gamma \max_{a'} Q^*(s', a')]$$

Where $s' \sim P$ is a shorthand for $s' \sim P(\cdot|s, a)$, indicating that the next state s' is sampled from the environment's transition rules; $a \sim \pi$ is a shorthand for $a \sim \pi(\cdot|s)$; and $a' \sim \pi$ is a shorthand for$a' \sim \pi(\cdot|s')$.

The Bellman equations provide a theoretical foundation for various reinforcement learning algorithms. These algorithms leverage iterative techniques to solve the Bellman equations and estimate the value functions ($V(s)$ or $Q(s, a)$) for each state or state-action pair. By understanding the value of each state or state-action pair, the agent can learn which actions lead to the highest long-term rewards within the environment.

Some remarks about the bellman equations:

- The difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the max over actions. Its inclusion reflects the fact that whenever the agent gets to choose its action, in order to act optimally, it has to pick whichever action leads to the highest value.
- The discount factor (γ) plays a crucial role in balancing the importance of immediate rewards vs. future rewards.
- Solving the Bellman equations exactly can be computationally expensive for complex environments. However, RL algorithms utilize various techniques to approximate the value functions effectively.

The Bellman equations offer a powerful framework for understanding and implementing reinforcement learning algorithms. By leveraging these equations, agents can learn to navigate complex environments and make decisions that maximize their long-term rewards.

4.4 Policy optimization

Policy optimization is a powerful approach which refers to a family of algorithms that make use of the gradient of a parameterized policy (which maps states to probabilities of taking different actions) in order to improve its performance. These types of algorithms works directly on the agent's policy and are often referred to as on-policy methods.

Unlike value-based methods that focus on estimating state or action values, policy optimization directly update a policy function (through gradient descent or similar techniques) in a way that increases the expected future reward the agent receives.

Recalling equation (4.5), we want to obtain an expression for the policy gradient which we can numerically compute

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\pi_\theta)R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\pi_\theta)R(\tau)\end{aligned}$$

Using the fact that the derivative of $\log(x)$ is $1/x$, then, when rearranged and combined with the chain rule, we get:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_\tau P(\tau|\pi_\theta)\nabla_\theta \log(P(\tau|\pi_\theta))R(\tau) \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log(P(\tau|\pi_\theta))R(\tau)]\end{aligned}$$

By expanding the transition function, we can eliminate terms that has no dependance on θ, thus:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log\left(\rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)\right)R(\tau)\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\cancel{\nabla_\theta \log \rho_0(s_0)R(\tau)} + \sum_{t=0}^{T}\left(\cancel{\nabla_\theta \log P(s_{t+1}|s_t,a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t)\right)R(\tau)\right] \\ \therefore\ \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right].\end{aligned}$$

This last expression allows us to compute the gradient assuming that we have a policy π represented by a set of parameters θ which are able to run on the environment and allow us to calculate $\nabla_\theta \log \pi_\theta(a|s)$. Then, the expectation can be estimated as a sample mean if we collect a set of trajectories and store them as a dataset $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$, i.e.:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau).$$

With $|\mathcal{D}| = N$, the number of trajectories made with the current policy.

4.4.1 Reward-to-go and baselines

It is possible to show that the policy gradient can also be expanded as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}).\right] \tag{4.6}$$

Where $\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$ is the **reward-to-go** from timestep t onward. In this form, actions are only reinforced based on rewards obtained after they are taken.

Lemma 4.4.1 (Expected Grad-Log-Prob)**.** Suppose that P_θ is a parameterized probability distribution over a random variable, x. Then:

$$\mathbb{E}_{x\sim P_\theta}[\nabla_\theta \log P_\theta(x)] = 0$$

Proof. Recall that all probability distributions are normalized:

$$\int_x P_\theta(x) = 1$$

Take the gradient of both sides of the normalization condition:

$$\nabla_\theta \int_x P_\theta(x) = \nabla_\theta 1 = 0$$

Using the log derivative trick we get:

$$\begin{aligned} 0 &= \nabla_\theta \int_x P_\theta(x) \\ &= \int_x \nabla_\theta P_\theta(x) \\ &= \int_x P_\theta(x)\nabla_\theta \log P_\theta(x) \\ \therefore\ 0 &= \mathbb{E}_{x\sim P_\theta}[\nabla_\theta \log P_\theta(x)]. \end{aligned}$$

□

Lemma 4.4.1 show us that for any function b which only depends on the state, then:

$$\mathbb{E}_{a_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)b(s_t)] = 0.$$

This property allows us to add or subtract any term in equation (4.6), without changing the expected value, then:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\Big(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)\Big)\right]. \tag{4.7}$$

Where $b(s)$ is called the **baseline**. Empirically, a common choice is to use the state-value function $V^\pi(s_t)$, because it has the desirable effect of reducing the variance of a sample estimate for the policy gradient, which in turn results in a faster and more stable policy learning. This also has the conceptual implication for an agent, which is that *the agent should "feel" neutral if it gets what it is expected*.

In practice, $V^\pi(s_t)$ is approximated by a neural network, $V_\phi(s_t)$, which is updated by the current policy using a mean-squared-error objective:

$$\phi_k = \arg\min_\phi \mathbb{E}_{s_t,\hat{R}_t\sim\pi_k}\left[\left(V_\phi(s_t) - \hat{R}_t\right)^2\right]$$

4.4.2 Generalized policy gradient

Equation (4.7) can be expressed in a general form:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right].$$

Where Φ_t may be one of the following:

- $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
- $\sum_{t=t'}^{\infty} r_{t'}$: reward following action a_t.
- $\sum_{t=t'}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
- $Q^\pi(s_t, a_t)$: state-action value function.
- $A^\pi(s_t, a_t)$: advantage function.
- $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: Temporal Difference (TD) residual.

  Where:

$$V^\pi(s_t) \coloneqq \mathbb{E}_{s_{t+1:\infty},\, a_{t:\infty}}[\sum_{l=0}^{\infty} r_{t+l}] \quad \text{and} \quad Q^\pi(s_t, a_t) \coloneqq \mathbb{E}_{s_{t+1:\infty},\, a_{t+1:\infty}}[\sum_{l=0}^{\infty} r_{t+l}]$$

Each variant of Φ_t has advantages and disadvantages, but within all the options, the advantage function offers the lowest variance of all. This can be further discussed in more depth in (30).

4.4.3 Advantage function

The advantage function act as a refinement tool for the agent's decision-making process. While value functions estimate the long-term expected reward from a state or state-action pair, the advantage function offer a relative measure of how much better or worse a specific action is compared to the average action within a particular state. Mathematically is formulated as:

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

By focusing on the relative advantage rather than absolute values, advantage functions help the agent distinguish between good and bad actions within the same state, providing more nuanced information for efficient learning.

Some of the key points on using the advantage function are:

1. **Faster convergence:** By focusing on the **differential reward** between actions, the learning process can converge on the optimal policy more efficiently compared to solely relying on absolute value functions.
2. **Reduced variance:** The advantage function can help address the issue of high variance in Q-value estimates, leading to more stable and reliable learning.
3. **Improved sample efficiency:** It can allow the agent to learn effectively even with limited data compared to methods solely relying on absolute value functions.

In conclusion, the generalized formulation of policy gradients with advantage functions is very common in algorithms like "Vanilla Policy Gradient" (VPO), "Trust Region Policy Optimization" (TRPO), and "Proximal Policy Optimization" (PPO).

4.5 PPO algorithm

Proximal Policy Optimization (PPO) is a popular reinforcement learning algorithm used for training agents to perform tasks in environments where the agent interacts with the environment and receives feedback through rewards. PPO belongs to the family of policy gradient methods, which directly optimize the policy function to maximize cumulative rewards.

Overall, PPO balances exploration and exploitation by updating the policy stably and efficiently. It has been widely used in various applications, including robotics, game-playing, and autonomous systems.

Algorithm (1) gives a basic implementation used in this work.

Algorithm 1 Pseudo-code for the PPO-clip algorithm

Require: Initialize policy network parameters θ_0 and value network parameters ϕ_0.
Require: Initialize environment and set hyperparameters.

for each iteration $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ using the current policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, A_t (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Normalize advantage estimates.
  for each epoch $j = 0, 1, 2, \ldots$ **do**
    Shuffle and batch trajectories.
    for each batch $l = 0, 1, 2, \ldots$ **do**
      Compute ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

      Compute clipped surrogate objective

$$L_{\text{clip}}(\theta) = \mathbb{E}_t[\min(r_t(\theta) \cdot A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t)]$$

      Compute value function loss

$$L_{\text{VF}}(\phi) = \mathbb{E}_t[(V_\phi(s_t) - \hat{R}_t)^2]$$

      Compute total loss

$$L_{total} = L_{\text{clip}}(\theta) - c_1 \cdot L_{\text{VF}}(\phi) + c_2 \cdot H$$

      Update policy network using gradient descent

$$\theta_{k+1} = \theta_k - \alpha \nabla_{\theta_k} L_{total}$$

      Update value network using gradient descent

$$\phi_{k+1} = \phi_k - \alpha \nabla_{\phi_k} L_{total}$$

    end for
  end for
end for

In the pseudo-code above:

- $\hat{R}_t$ is the empirical return (also called "Reward-to-go"), which is the sum of rewards from time step t onwards discounted by a factor γ, i.e., $\hat{R}_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$ where r_i is the reward at time step i, T is the total number of time steps, and γ is the discount factor.
- A_t is the advantage function, in RL represents the "advantage" of taking a particular action in a given state compared to the average action value in that state, specifically $A_t(s_t, a_t) = Q(s_t, a_t) - V_\phi(s_t)$.

  Alternatively, the advantage function can also be calculated using the returns $\hat{R}_t$ (or the discounted rewards), i.e., $A_t(s_t, a_t) = \hat{R}_t - V_\phi(s_t)$.

- $r_t(\theta)$ represents the ratio between the probabilities of actions taken under the new and old policies.
- clip$(\cdot)$ is a function that clips the ratio $r_t(\theta)$ to lie within the range $[1 - \epsilon, 1 + \epsilon]$.
- ϵ is the clipping parameter
- c_1 and c_2 are coefficients controlling the trade-off between the clipped surrogate objective and the value function loss.
- H is an "entropy bonus," which is an optional term added to encourage exploration.

4.6 Imitation Learning by motion capture

Training controllers for humanoid robots that can not only perform tasks but also exhibit natural, non-idiosyncratic behaviors remains a significant challenge. Motion capture (mocap) data offers a valuable tool in this pursuit, providing a rich source of kinematic information to guide the robot's movement towards desired behaviors. This chapter delves into the application of motion capture data within the framework of imitation learning for humanoid robots.

Developing controllers for complex tasks like walking, running, or manipulating objects presents a unique challenge for humanoid robots. While achieving the desired task outcome is crucial, the robot's movement should also appear natural and human-like. This avoids creating jerky, robotic motions that can be inefficient or even unsafe.

Traditional reinforcement learning approaches often struggle to achieve this balance. While a robot may learn to complete a task through trial-and-error, the resulting movements might be highly idiosyncratic and unnatural. This is because the reward function typically focuses solely on task success, neglecting aspects like movement style or energy efficiency.

Motion capture data offers a powerful solution to address this challenge. By capturing the detailed kinematics of human movements, mocap data serves as a valuable reference point for the robot's controller. The captured information includes joint positions, orientations, and velocities throughout the movement, providing a comprehensive picture of how a human achieves a specific task.

Integrating this information into the robot's learning process allows the controller to not only focus on task completion but also strive to replicate the natural human-like movement observed in the mocap data. This can be achieved through various imitation learning techniques.

Several imitation learning techniques have been successfully employed to leverage motion capture data for training humanoid robots. Here are some prominent approaches:

- **Reinforcement Learning with Imitation Loss:** This approach combines the strengths of reinforcement learning and imitation learning. The robot learns a policy that maximizes task reward while also minimizing the difference between its movements and the movements observed in the motion capture data (15, 22).
- **Adversarial Imitation Learning:** This method utilizes a generative adversarial network (GAN) framework. One network (generator) aims to produce robot movements that mimic the motion capture data, while another network (discriminator) tries to distinguish between the generated movements and real human movements. This adversarial training encourages the robot to generate highly natural movements (25, 26, 31).

These techniques demonstrate the effectiveness of using motion capture data to bridge the gap between task completion and natural movement in humanoid robots

5 | Proposal

5.1 Control structure

To test our proposal, that is, to follow marked footsteps with human-like motion, we reuse "CMU Humanoid" in the dm-control environment. The "CMU Humanoid" is a 56-degree of freedom (DoF) designed to resemble an average human body; it can deliver a wide range of internal and custom observations s_t, including an egocentric camera (not used in this task).

Our control framework consists of a hierarchical policy running at a control rate of 20 Hz. The high-level policy learns to incorporate desired footsteps and root directions from a previous planning stage in addition to the signals coming from the robot's sensors.Also, to further guide the high-level policy for the task, recent approaches [18, 32, 33, 34] define a phase signal $\phi \in [0, 1]$ that instantiates the cyclic locomotion transitions from Double Support (DS) to Single Support (SS) and vice-versa. This phase signal ϕ also helps synchronize the rewards given to promote or demote the foot forces and velocities on different time steps to generate the cyclic motion.

The low-level policy aims to receive a latent vector z_t and is responsible for predicting the desired joint angles of the humanoid a, which are internally converted to joint torques by a predefined PD controller. This low-level network is not trainable, as it comes from a previous stage of imitation learning following the work done in (35).

The overall structure can be visualized in Figure 5.1. The environment consists of the simulator (robot in the scene) plus the low-level network (π_{dec}), since the low-level network has fixed parameters. Therefore, the high-level network (π_{task}) is what is trained using some optimization algorithm.

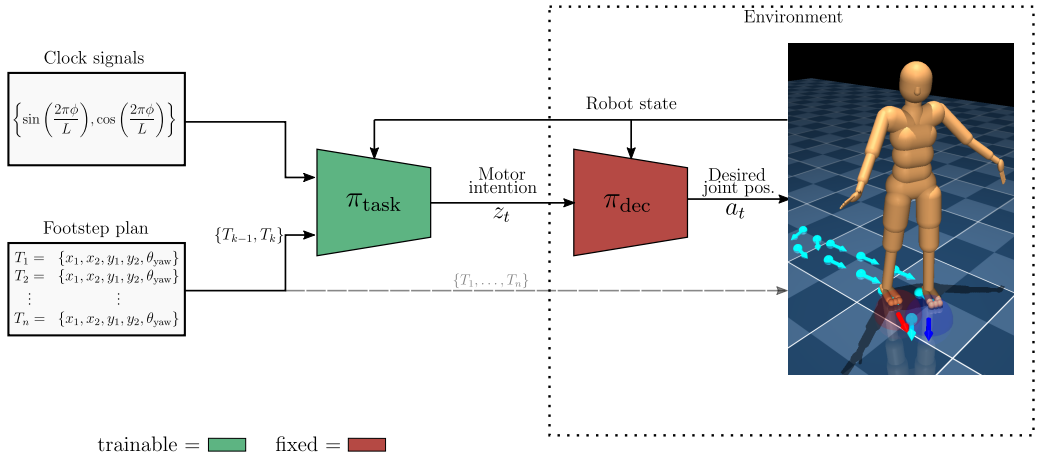


Figure 5.1: Proposed architecture. The hierarchical network is composed of a low-level policy (π_{dec}) previously trained from MoCap clips, and the high-level policy (π_{task}) is trained to follow marked footsteps.

5.2 Policy parameterization

5.2.1 Observation space

The state s_t as defined in Table 5.1 comprises the humanoid's internal and external signals. The internal signals are composed of proprioceptive sensors such as angle and angular velocity of the joints, θ and $\dot{\theta}$ respectively, desired joint angles a which are converted to joint torques by a predefined PD controller internally, height of the robot h_{root} measured from the waist to the floor, a vector ${}^{z}p_{root}$ which represents the projection of the z-axis on the robot w.r.t the world frame, a root orientation vector q_{root}, an end-effector position vector ${}^{x,y,z}p_{hands,feet}$, and the same vector as above but including the head position ${}^{x,y,z}p_{hands,feet,head}$. It should be noted that these last two vectors are not helpful for the current task, but they are necessary since the low-level network was trained with the complete proprioceptive observations vector. However, these vectors may be beneficial in future work in which the network needs to be transferred to a different humanoid robot since they are model-agnostic observations and serve as signals for some imitation training algorithms using adversarial networks as described in [31].

The kinematics sensors comprise a 3-axis gyro $\omega_{roll,pitch,yaw}$, a 3-axis velocimeter $\mathbf{v}$, and a 3-axis accelerometer $\mathbf{a}$, measured at the root of the humanoid. On the other hand, the Dynamic Sensors are composed of joint torques τ and binary contacts $\mathbf{b}$.

The external signals are designed for the task, which is to follow marked footsteps. We leverage the work done in [18], where they use two-phase clock signals that represent the timing of the walk. The phase of the sinusoidal signals is controlled by a cyclic parameter ϕ, which changes at each timestep and is normalized by a total number of phase steps L.

Additionally, the closest footstep $\mathbf{T}_1$ and the next closest footstep $\mathbf{T}_2$ are given. These signals comprise their 3D positions (x, y, z) and root yaw orientation θ.

5.2.2 Action space

The high-level policy outputs the mean and diagonal covariance of a Gaussian distribution over a 60-dimensional vector that compresses the state s_t to a "motor intention" z_t. Then, the fixed low-level policy converts the motor intention z_t and current state s_t to a 56-dimensional vector where each action is the mean

Table 5.1: Agent observations

	Observation	Dim.
Propioceptive sensors	Joint angles θ	56
	Joint velocities $\dot{\theta}$	56
	Actuator activations a	56
	Body height h_{root}	1
	End-effector positions ${}^{x,y,z}p_{hands,feet}$	12
	Appendages positions ${}^{x,y,z}p_{hands,feet,head}$	15
	Inclination w.r.t. world z-axis ${}^{z}p_{root}$	3
	Orientation q_{root}	9
Kinematic sensors	Gyro $\omega_{roll,pitch,yaw}$	3
	Velocimeter $\mathbf{v}$	3
	Accelerometer $\mathbf{a}$	3
Dynamic sensors	Torque τ	6
	Binary contacts $\mathbf{b}$	10
External	Clock signals $\begin{cases}\sin\left(\frac{2\pi\phi}{L}\right)\\ \cos\left(\frac{2\pi\phi}{L}\right)\end{cases}$	2
	Footsteps $\mathbf{T}_i = [x_i, y_i, z_i, \theta_i]$ with $i \in \{1, 2\}$	8

of a Gaussian distribution over actions with a standard deviation of 0.1. Note that we are only training the high-level network and the low-level network is formulated as part of the environment.

5.3 Reward definitions

Equation 5.1 defines the total reward at each time step. This reward seeks to direct the robot to follow planned footsteps as defined in [18]. The corresponding equations are shown in Table 5.2:

$$r = w_1 r_{grf} + w_2 r_{spd} + w_3 r_{step} + w_4 r_{orient} + w_5 r_{height} + w_6 r_{upper} + w_7 r_{action} + w_8 r_{torque}. \tag{5.1}$$

Table 5.2: Reward functions definitions

Reward	Weight (w_i)	Function
r_{grf}	0.15	$I^{grf}_{left}(\phi)\|\|F_{left}\| + I^{grf}_{right}(\phi)\|\|F_{right}\|$
r_{spd}	0.15	$I^{spd}_{left}(\phi)\|\|S_{left}\| + I^{spd}_{right}(\phi)\|\|S_{right}\|$
r_{step}	0.45	$k_{hit} \cdot \exp(\frac{d_{foot}}{0.25}) + (1 - k_{hit}) \cdot \exp(\frac{d_{foot}}{0.25})$
r_{orient}	0.05	$\exp(-10 \cdot (1 - \langle q_{root}, \hat{q}_{root} \rangle^2))$
r_{height}	0.05	$\exp(-40 \cdot (h_{root} - \hat{h}_{root})^2)$
r_{upper}	0.05	$\exp(-10 \cdot \|\|^{x,y} p_{head} - {}^{x,y} p_{root}\|\|^2)$
r_{action}	0.05	$\exp(-5 \cdot \sum \|a - a_{prev}\|/56)$
r_{torque}	0.05	$\exp(-0.25 \cdot \sum \|\tau - \tau_{prev}\|/6)$

Where the reward r_{grf} regulates the normalized ground reaction forces $F_{right/left}$ applied at the feet, with phase indicator functions $I^{grf}_{right/left}(\phi) \in [-1, 1]$, r_{spd} is the reward that regulates the normalized body speeds $S_{left/right}$ at the feet, with phase indicator functions $I^{spd}_{right/left}(\phi) \in [-1, 1]$, r_{step} is the reward which promotes the robot to place any of its feet and root orientation according to the desired step plan indicated by the variable $\mathbf{T}_1$, and r_{orient} is a reward to encourage the root body to align with the desired orientation according to the step plan, r_{height}, r_{upper}, r_{action} and r_{torque} are the rewards that regulate a desired root height above ground, a desired upright posture, and penalties for applied action and torque respectively.

To guide the robot to perform a symmetrical walk, that is, to transition from a double support (DS) phase, where both feet are in contact with the ground, to a single support (SS) phase, where only one foot is in contact while the other is swinging, it is necessary to provide information that has a cyclic pattern. For this, we follow the approach done in (18, 32), which adapts the reward so that a walking cycle consists of two DS phases and two SS phases of fixed time.

Figure 5.2 shows the waveform corresponding to the functions (from top to bottom); foot speed indicator $I^{spd}_{right/left}(\phi)$, ground reaction force indicator $I^{grf}_{right/left}(\phi)$, and phase indicator ϕ, the latter being the one that regulates the reaction and velocity indicators, for the left and right foot.

5.4 Initialization and ending conditions

Training an agent in a single, fixed environment can lead to policies that are brittle and perform poorly when deployed in slightly different settings. Domain randomization addresses this issue by introducing variations within the training environment. By exposing the agent to a wider range of conditions during training, domain randomization encourages it to learn more robust policies that can adapt to unseen variations when transferring the policy to the real world (36).

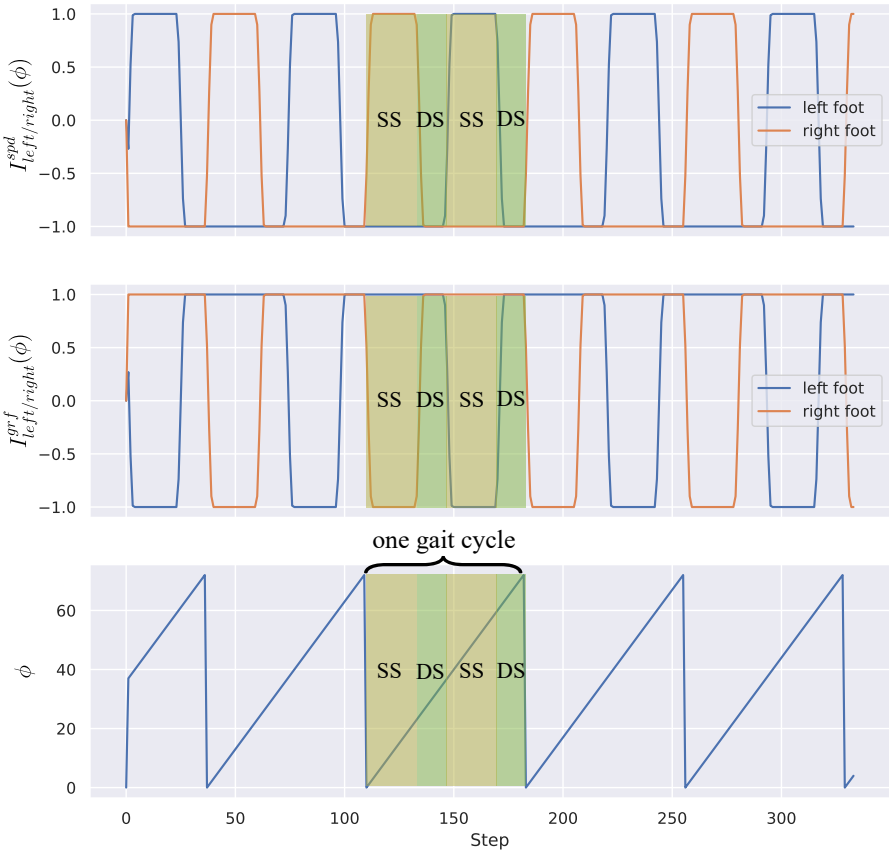


Figure 5.2: (From top to bottom) Feet velocity, ground reaction forces, and phase indicators vs environment steps; these three functions regulate the rewards r_{grf}, r_{spd} at each time step. Additionally, ϕ is implicitly added as an observation to the agent by the clock signals.

Using this knowledge, we randomize the robot's spawn position, root yaw orientation, and joint angles according to Table 5.3. Unlike the "half-sitting" posture used extensively in humanoid robots, the CMU-humanoid robot starts at a specific mocap frame which is close to a standing pose but far from being stable. Thus, a random fraction ($\pm q_0 \times 0.02$) is added to the initial q_0. On the other hand, since the global yaw angle of the robot's root is observed by the policy, it is important to randomize the initial heading of the robot. Finally, the initial value of the phase variable ϕ can be 0 or 0.5.

Table 5.3: Domain randomization variables

Parameter[(a)]	Unit	Range
Spawn position (x, y)	mts	± 2.5
Spawn yaw angle	rad	$\pm\pi$
Joint angles	rad	$\pm q_0 \times 0.02$[(b)]
Initial phase ϕ	rad	0 or 0.5

[(a)] All parameters are sampled from a uniform distribution
[(b)] q_0 is a 56D vector of default joint angles

Episodes define the beginning and end of an interaction sequence between the agent and the environment. Properly defining termination conditions ensures the agent learns from complete interactions and avoids getting stuck in unproductive loops. Thus, the episode ends when the root height from the lowest point of foot-floor contact is less than a threshold ($60[cm]$), and a self-collision condition. Finally, the episode ends

after a fixed number of environment steps (800 steps) or when any of the previously mentioned conditions are met.

5.5 Footstep generation

Footstep generation in robotics refers to the process of planning and determining the placement of a robot's feet during locomotion. This is a crucial aspect of robot motion, particularly for legged robots like bipedal walkers, quadrupeds, or even robots with more complex leg configurations.

In this work, each footstep corresponds to a 4-dimensional vector, where the first three indices correspond to the desired 3D position that a foot should reach, and the last index indicates the desired root yaw orientation of the robot's base. By adding orientation to the footstep plan, it is possible to enable the robot to perform walking modes such as turning in place, walking sideways, or backward (37).

5.6 Curriculum learning

Curriculum training is a widely used methodology today, as it avoids the problem of getting stuck in a local minimum when starting training due to its high difficulty, focusing on maintaining balance so that one does not fall at first (38).

Initially, the training of the high-level policy is exposed to each sequence of steps on flat terrain. After a certain number of environment steps, the height of each step is increased linearly according to the following formula:

$$p_z = \begin{cases} 0, & \text{step} < 30 \cdot 10^6. \\ k_c \cdot 0.1 \cdot (step - 30 \cdot 10^6/80 \cdot 10^6), & 30 \cdot 10^6 \leq \text{step} < 80 \cdot 10^6. \\ k_c \cdot 0.1 & \text{step} \geq 80 \cdot 10^6. \end{cases}$$

Where $k_c = -1, 1$ is a random variable that determines if the steps are ascending or descending. Thus, increasing or decreasing the height of the boxes (which serve as staircases) as the training progresses. This aims to encourage the agent to initially focus on standing up and then follow the step planner's T_1 and T_2 variables.

6 | Baseline Comparison Methods

To test the hypothesis that using a low-level network based on motion capture generates more human-like motion, we conducted a study where we trained three different architectures. All three architectures were designed to receive the robot's internal state vector, two-phase signals, and the two next-step positions to be followed as described by Table 5.1. Its output is a 56-dimensional vector corresponding to the desired joint positions, which are then converted to torque by an internal PD controller in the robot.

6.1 Simple Policy

In the first case, our baseline test architecture consists of using a simple network with no prior training, e.g., using some imitation learning algorithm or an on-policy RL algorithm. Figure 6.1 shows the baseline architecture.

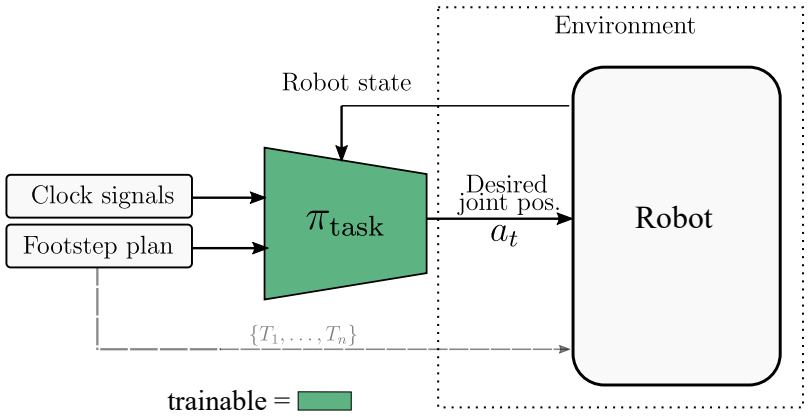


Figure 6.1: Baseline architecture. π_{task} network is trained using the PPO algorithm.

Additional details like network hyperparameters and training of the policy π_{task} are described in Chapter 7.

6.2 High-Level Policy + Low-Level Locomotion Policy

The second architecture consists of a trainable high-level network. The high-level network receives the complete state vector and generates a latent vector at its output. This latent vector is then used as input to the low-level policy, which generates a 56-dimensional vector of desired joint angles, similar to the simple policy.

This architecture has the advantage of learning a hierarchical representation of the movement, where the high-level policy captures the overall structure of the movement, and the low-level policy generates the detailed motion. Note that the low-level policy was trained in a previous stage using a subset of the motion

capture dataset, specifically the locomotion subset, corresponding to about 40 minutes of running, walking, and standing actions.

6.3 High-Level Policy + General Low-Level Policy

Lastly, the third architecture follows the same approach as the second architecture. However, the low-level network was trained with the complete MoCap dataset, corresponding to 4.2 hours of various activities, including the locomotion subset. This architecture has the advantage of learning a more general representation of the movement, as it was trained using a more extensive and diverse dataset.

Figure 6.2 represents the second and third architectures. In both cases, π_{task} (high-level policy) is trained using the PPO algorithm, while π_{dec} (low-level policy, fixed) comes from previous training, using an imitation learning algorithm further explained in (35).

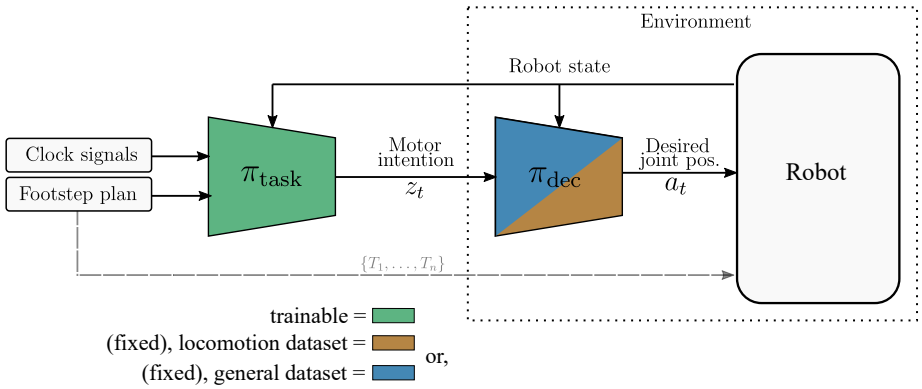


Figure 6.2: Proposed hierarchical policy. This work studies the effect on motion generation when the low-level policy is distilled using different subsets of the MoCap dataset, namely, the General and Locomotion subsets introduced in (39).

7 | Experimental approach

The following section describes the parameters used during the high-level policy training. These parameters were carefully chosen to ensure the model was optimized for the task at hand. Then, the use of the MoCapAct dataset is explained. Their work provides a detailed description of the motion capture dataset used to train the low-level policy. This dataset (and trained models) were selected for its ability to accurately capture the movements required for the task, ensuring that the high-level policy could learn from the low-level policy.

7.1 Network architecture

The high-level policy (π_{task}) is based on the MLP architecture with three hidden layers for actor and critic networks. Each layer has 1024 units with Hyperbolic Tangent activation function. During training, each episode lasts 800 steps until a buffer of 57600 steps is reached for policy optimization with a learning rate of 0.0001. This buffer is divided into mini-batches of 9600 steps with four epochs per minibatch. The training lasts 100 million steps with 32 workers running in two Intel Xeon E5-2630 v4, 252GB of RAM, and a Tesla PC100 12GB graphics card, although most training is done on CPU. Additional hyperparameters are listed in Table 7.1 following the work done in [39].

Table 7.1: Hyperparameters for High-level policy training

Total environment steps	100 million
Curriculum learning start step	30 million
Curriculum learning end step	80 million
Number of workers	32
Episode steps	800
Env. steps per policy optimization	57600
PPO epochs	4
PPO minibatch size	9600
PPO clipping range (ϵ)	0.2
Target KL divergence (early stopping)	0.3
l_2 gradient norm clipping value	1.0
GAE λ	0.95
Discount factor γ	0.99
Entropy coefficient	0.0
Reward & observation normalization	True
Learning rate	1e-4
Initial standard deviation	2.5
Max. per-element action magnitude	3.0

7.2 MoCapAct Dataset

The MoCapAct Dataset [39] is a compound of expert agents that can track over three hours of motion capture (MoCap) data, performing different actions (e.g., walking, running, jumping, among others) in the physics-based environment called the dm_control [40]. This dataset also includes observations, actions, and rewards generated from rolling out these experts. They show the usefulness of this dataset by training a single hierarchical policy, as shown in Fig. 7.1 that can track the entire MoCap dataset within the dm_control environment. They also found that the learned low-level policy can be reused to learn downstream high-level tasks by constraining the humanoid's behaviors, speeding up the learning process. Finally, they use MoCapAct to train an autoregressive GPT model that can control a simulated humanoid to perform natural motion completion given a motion prompt.

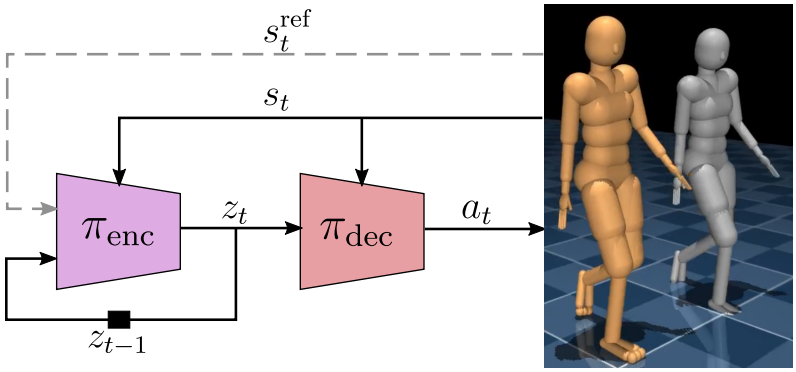


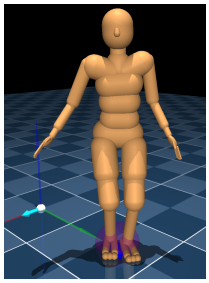
Figure 7.1: Multi-clip tracking policy architecture used in (39). We leverage the low-level network (π_{dec}) for a downstream high-level task.

In this work, we leverage the hierarchical-level policy using only the low-level network. This network follows the architecture realized in (35), which is composed of an encoder network $\pi_{enc}(z_t|s_t, s_t^{ref}, z_{t-1})$ whose objective is to follow the references s_t^{ref} coming from the MoCap, generating at the output an embedding z_t representing the "motor intention." The decoder network $\pi_{dec}(a_t|s_t, z_t)$, in turn, receives the embedding z_t together with the current state s_t, generating an action a_t.

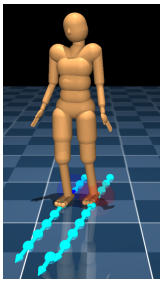
For downstream tasks, by reusing the π_{dec} decoder network as part of the environment, the motor intention z is the new action the environment receives. Then, a high-level policy network $\pi_{task}(z|s)$ is trained using a reinforcement learning algorithm to guide the low-level policy to maximize the reward for the specified task, as shown in Fig. 5.1.

8 | Results

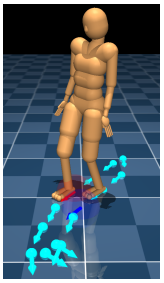
Figure 8.1 illustrates the various walking modes employed to train the high-level network, including forward, backward, sideways (left or right), and curved. Moreover, curriculum training was implemented for the forward walking mode, gradually increasing each step's height until a randomly generated ascending or descending staircase was formed. This staircase's height between steps could go up to ±0.1[m].



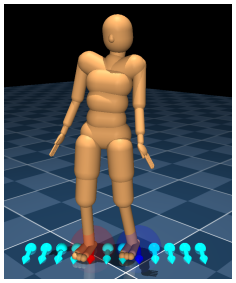
(a) Standing.



(b) Forward/backward walking.



(c) Curved walking.



(d) Lateral walking.

Figure 8.1: Trained walking modes (standing, forward, backward, lateral, and curved).

8.1 Training details

It takes approximately nine days to collect 100 million steps for learning all modes by training the high-level policy using the parameters described in chapter 7.1. The simulations and optimizations are done entirely on the CPU. Our experiments demonstrate that a periodic reward composition with a lower-level policy generates a bipedal gait suitable for realistic gait cycle durations.

To improve the robot's initial learning process, we modified the episode termination condition. It will only trigger when the distance between the root's base and the feet falls below a specific threshold, which means the robot will not explore states that do not promote an upright posture, which will speed up the learning process.

It was also important to include the robot's base posture $\omega_{roll,pitch,yaw}$, which is measured in terms of roll, pitch, and yaw, in the state vector. Relying solely on the direct gyro sensor information was not enough

for the robot to accurately determine its orientation with respect to the planned step trajectory.

In conjunction with the low-level policy, the high-level policy demonstrates the ability to follow marked footstep plans in forward, backward, lateral, and curved directions. However, it is essential to note that these successes are limited to flat terrains, and so far, the ability to follow marked footsteps on stairs has yet to be achieved. This limitation may be due to the need for low-level network training with movement clips that include these actions.

On the same page, the training results are shown in Figure 8.2, which demonstrates the mean episode duration and mean reward during the training process (500 samples per point). The findings indicate that a hierarchical architecture that utilizes a low-level policy based on motion captures leads to better results than training a single network to follow the marked footsteps in a human-sized robot with 56 degrees of freedom.

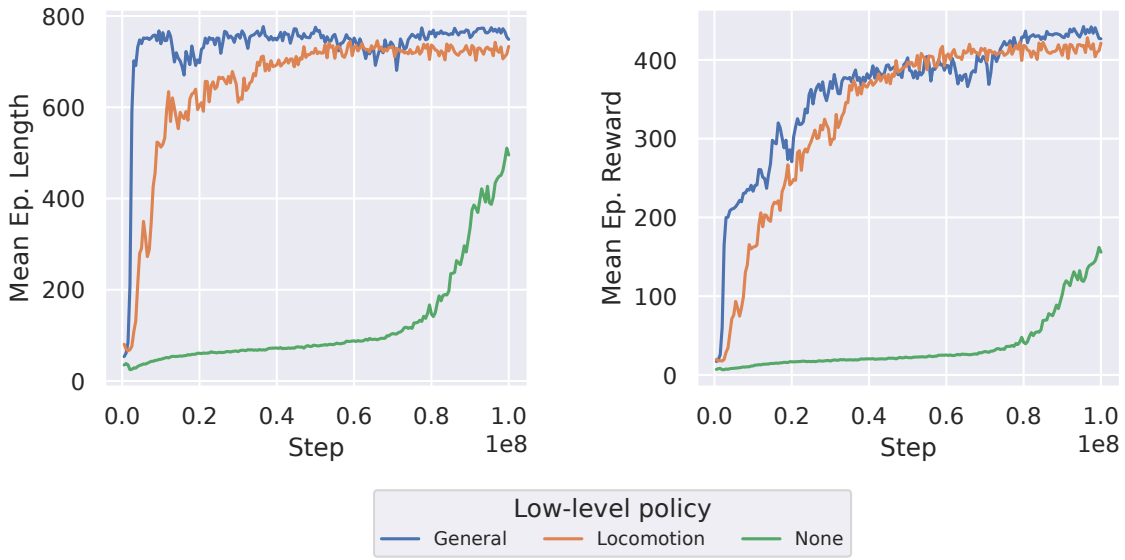
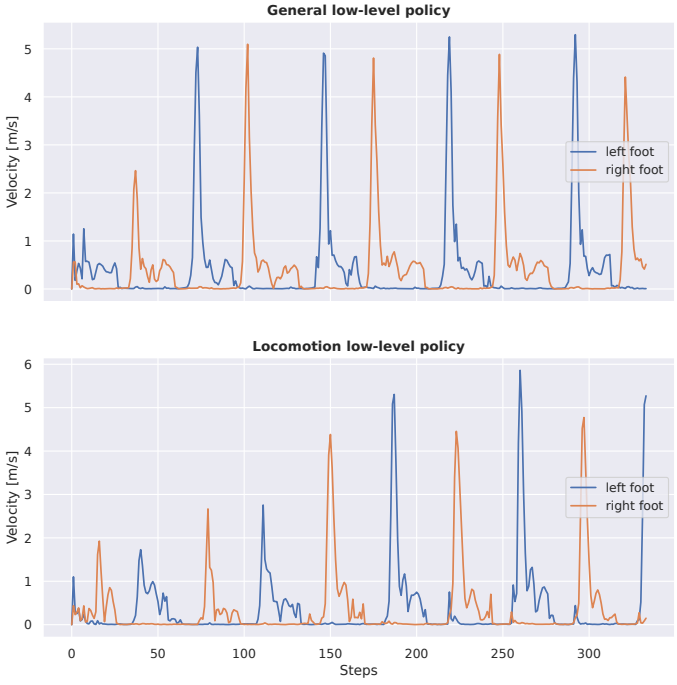


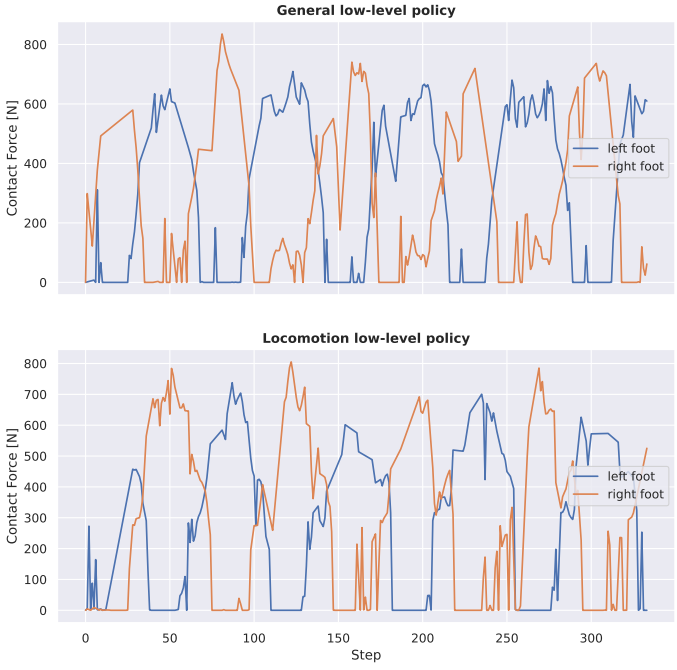
Figure 8.2: Mean episode length (left) and mean reward (right) vs environment steps.

Both the general and locomotion datasets resulted in similar cumulative reward values after training, indicating that using a more complete motion dataset is not required for the task. Additionally, the motions generated with the locomotion dataset were found to be visually more similar to human walking, especially for forward walking, when compared to those generated by the general dataset. You can watch the accompanying video[1] to see the comparison.

It has been observed that the foot velocities and forces exhibit similar characteristics in terms of amplitude and shape for both low-level networks trained with the general dataset and those trained with the locomotion dataset, as shown in Figures 8.3a and 8.3b. This observation is significant as it indicates that the trained network follows the conditions set by the rewards r_{grf}, r_{spd}, and r_{step}, which encourage the production of stable walking within the marked steps.



(a) Foot velocities vs Environment steps during forward walking.



(b) Contact forces at the feet vs Environment steps during forward walking.

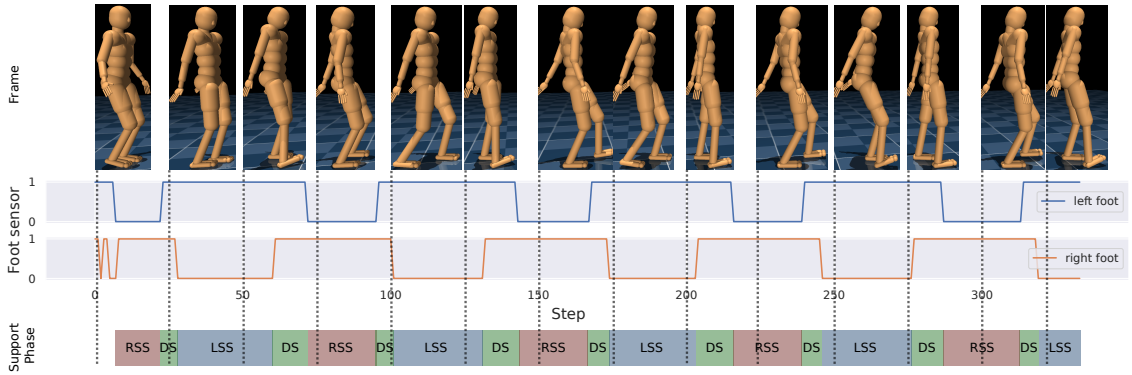
Figure 8.3: (8.3a) Velocities and (8.3b) Contact forces at both feet for general and locomotion datasets.

The torques and maximum speeds generated in the leg joints during forward movement are outlined in Table 8.1. It is evident that by employing a low-level network based on motion captures, these values can be significantly reduced while still adhering to the constraints set by the rewards r_{torque} and r_{action}. In contrast, in the absence of a low-level network, joint speeds reach ten times higher values, and joint torques double.

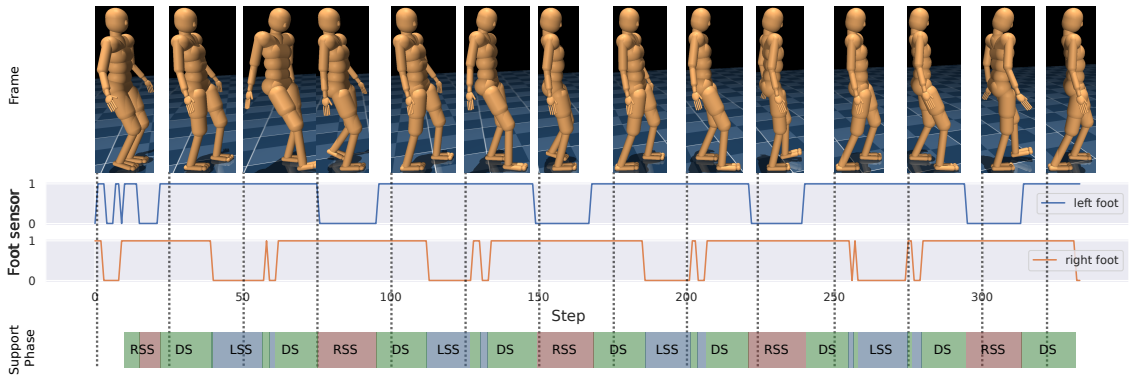
Table 8.1: Joints torques and velocities on legs during forward walking

Max Joint Torques [Nm]								
Low-level policy	Leg	femur rx	femur ry	femur rz	tibia rx	foot rx	foot rz	toes rx
General	Left	170.91	17.86	82.61	68.18	69.89	11.30	9.11
	Right	101.17	37.96	77.26	36.10	83.12	7.43	3.90
Locomotion	Left	140.13	30.24	118.57	48.75	87.12	12.69	9.02
	Right	152.90	44.60	71.69	29.25	94.24	18.58	3.74
None	Left	300,00	200,00	200,00	87,25	120,00	47,78	20,00
	Right	300,00	200,00	200,00	103,82	120,00	50,00	20,00
Max Joint Velocities [rad/s]								
General	Left	4.28	1.58	1.07	9.07	2.93	1.62	4.27
	Right	2.75	2.01	0.83	4.85	2.69	3.76	5.52
Locomotion	Left	3.65	2.62	1.05	7.84	6.26	2.89	5.56
	Right	4.40	2.76	1.43	6.71	2.80	2.90	3.30
None	Left	11,48	20,37	9,82	11,21	14,43	14,44	18,84
	Right	11,40	19,98	11,00	7,66	12,94	16,81	18,74

Lastly, Figure 8.4 presents the sequence of a forward walking gait using the general and locomotion low-level networks. It is observed that the signals generate the cyclic movement pattern, that is, it goes through the phases of double support (DS), left single support (LSS), double support (DS), right single support (RSS), and double support (DS). This is consistent with the constraints imposed by the rewards r_{grf}, r_{spd}, and r_{step} mentioned previously, which are implicitly modulated by the phase indicator ϕ. Furthermore, the foot sensor waveforms for both walks have similar waveforms, except for small jumps in the signal generated by the low-level network based on the locomotion dataset. Additionally, the motion of the left leg presents movements that are not entirely natural to the naked eye. This could be solved by modifying the default loss function of the PPO algorithm to include a term that promotes mirrored states and actions, as described in [41].



(a) General low-level policy.



(b) Locomotion low-level policy.

Figure 8.4: Foot phases during forward walking. (8.4a) General low-level policy, (8.4b) Locomotion low-level policy.

9 | Conclusions

In this study, we investigated a hierarchical reinforcement learning approach for bipedal walking robots. This approach employs a high-level policy trained to follow marked footsteps in conjunction with a low-level policy based on motion capture data. Our experiments demonstrate that this two-level architecture outperforms training a single network, particularly for complex tasks like following footsteps on a human-sized robot.

We examined the influence of the low-level network on the robot's walking behavior, focusing on joint-level velocities, torques, and the impact of using different motion capture data subsets. The results highlight the significant reduction in joint torques and maximum speeds achieved by the low-level network compared to a single-network approach despite maintaining the desired walking behavior.

While our approach successfully achieves stable and natural walking gaits on flat terrain, a current limitation is the inability to follow marked footsteps on stairs. We observed that a general motion capture dataset can achieve similar results to a locomotion-specific dataset. However, the locomotion dataset generates visually more natural human-like walking, especially for forward walking.

In future work we aim to improve the robot's walking robustness for navigating uneven terrains like stairs and slopes. Our findings strongly suggest that low-level networks pre-trained on motion capture data are a viable approach for generating human-like walking gaits applicable to real-world, human-sized robots. This research paves the way for more efficient and natural walking capabilities in bipedal robots.

Bibliography

[1] Matthew Chignoli, Donghyun Kim, Elijah Stanger-Jones, and Sangbae Kim. The mit humanoid robot: Design, motion planning, and control for acrobatic behaviors. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2021. 1

[2] Zhitao Song, Linzhu Yue, Guangli Sun, Yihu Ling, Hongshuo Wei, Linhai Gui, and Yun-Hui Liu. An optimal motion planning framework for quadruped jumping. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11366–11373. IEEE, 2022. 1

[3] Stefano Dafarra, Giulio Romualdi, and Daniele Pucci. Dynamic complementarity conditions and whole-body trajectory optimization for humanoid robot locomotion. *IEEE Transactions on Robotics*, 38(6):3414–3433, 2022. 1

[4] Eric Vollenweider, Marko Bjelonic, Victor Klemm, Nikita Rudin, Joonho Lee, and Marco Hutter. Advanced skills through multiple adversarial motion priors in reinforcement learning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5120–5126. IEEE, 2023. 1

[5] Masaki Murooka Sotaro Katayama and Yuichi Tazaki. Model predictive control of legged and humanoid robots: models and algorithms. *Advanced Robotics*, 37(5):298–315, 2023. 1

[6] Bjorn Lindqvist, Samuel Karlsson, Anton Koval, Ilias Tevetzidis, Jakub Haluska, Christoforos Kanellakis, Ali-akbar Agha-mohammadi, and George Nikolakopoulos. Multimodality robotic systems: Integrated combined legged-aerial mobility for subterranean search-and-rescue. *ROBOTICS AND AUTONOMOUS SYSTEMS*, 154, AUG 2022. 1

[7] Christian Gehring, Péter Fankhauser, Linus Isler, Remo Diethelm, Samuel Bachmann, Marcel Potz, Lars Gerstenberg, and Marco Hutter. Anymal in the field: Solving industrial inspection of an offshore hvdc platform with a quadrupedal robot. In *Field and Service Robotics: Results of the 12th International Conference*, pages 247–260. Springer, 2021. 1

[8] C. Dario Bellicoso, Marko Bjelonic, Lorenz Wellhausen, Kai Holtmann, Fabian Günther, Marco Tranzatto, Péter Fankhauser, and Marco Hutter. Advances in real-world applications for legged robots. *Journal of Field Robotics*, 35(8):1311–1326, 2018. 1

[9] Chen Li and Kevin Lewis. The need for and feasibility of alternative ground robots to traverse sandy and rocky extraterrestrial terrain. *Advanced Intelligent Systems*, 5(3):2100195, 2023. 1

[10] Nicolas Heess, Dhruva Tb, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017. 1

[11] Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022. 1

[12] Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, SM Ali Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. From motor control to team play in simulated humanoid football. *Science Robotics*, 7(69):eabo0235, 2022. 1

[13] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2020. 1

[14] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 403–415. PMLR, 14–18 Dec 2023. 1

[15] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020. 1, 4.6

[16] Jinze Wu, Guiyang Xin, Chenkun Qi, and Yufei Xue. Learning robust and agile legged locomotion using adversarial motion priors. *IEEE Robotics and Automation Letters*, 8(8):4975–4982, 2023. 1

[17] Donghyeon Kim, Glen Berseth, Mathew Schwartz, and Jaeheung Park. Torque-based deep reinforcement learning for task-and-robot agnostic learning on bipedal robots using sim-to-real transfer. *IEEE Robotics and Automation Letters*, 8(10):6251–6258, 2023. 1, 3.2, 3.2

[18] Rohan P Singh, Mehdi Benallegue, Mitsuharu Morisawa, Rafael Cisneros, and Fumio Kanehiro. Learning bipedal walking on planned footsteps for humanoid robots. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 686–693. IEEE, 2022. 1, 3.2, 3.3, 5.1, 5.2.1, 5.3, 5.3

[19] Michael Bloesch, Jan Humplik, Viorica Patraucean, Roland Hafner, Tuomas Haarnoja, Arunkumar Byravan, Noah Yamamoto Siegel, Saran Tunyasuvunakool, Federico Casarini, Nathan Batchelor, Francesco Romano, Stefano Saliceti, Martin Riedmiller, S. M. Ali Eslami, and Nicolas Heess. Towards real robot learning in the wild: A case study in bipedal locomotion. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1502–1511. PMLR, 08–11 Nov 2022. 1, 3.2, 3.4

[20] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H Huang, Dhruva Tirumala, Markus Wulfmeier, Jan Humplik, Saran Tunyasuvunakool, Noah Y Siegel, Roland Hafner, et al. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *arXiv preprint arXiv:2304.13653*, 2023. 1, 3.2

[21] Steven Bohez, Saran Tunyasuvunakool, Philemon Brakel, Fereshteh Sadeghi, Leonard Hasenclever, Yuval Tassa, Emilio Parisotto, Jan Humplik, Tuomas Haarnoja, Roland Hafner, et al. Imitate and repurpose: Learning reusable robot movement skills from human and animal behaviors. *arXiv preprint arXiv:2203.17138*, 2022. 1, 3.2

[22] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4), jul 2018. 1, 4.6

[23] Ariel Kwiatkowski, Eduardo Alvarado, Vicky Kalogeiton, C. Karen Liu, Julien Pettré, Michiel van de Panne, and Marie-Paule Cani. A survey on reinforcement learning methods in character animation. *Computer Graphics Forum*, 41(2):613–639, 2022. 1

[24] Yinhuai Wang, Jing Lin, Ailing Zeng, Zhengyi Luo, Jian Zhang, and Lei Zhang. Physhoi: Physics-based imitation of dynamic human-object interaction. *arXiv preprint arXiv:2312.04393*, 2023. 1

[25] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, 40(4), jul 2021. 1, 4.6

[26] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.*, 41(4), July 2022. 1, 4.6

[27] Guillermo A. Castillo, Bowen Weng, Wei Zhang, and Ayonga Hereid. Robust feedback motion policy design using reinforcement learning on a 3d digit bipedal robot. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5136–5143, 2021. 3.1

[28] Helei Duan, Ashish Malik, Mohitvishnu S. Gadde, Jeremy Dao, Alan Fern, and Jonathan Hurst. Learning dynamic bipedal walking across stepping stones. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6746–6752, 2022. 3.1, 3.1

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 4.1

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. 4.4.2

[31] Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017. 4.6, 5.2.1

[32] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315. IEEE, 2021. 5.1, 5.3

[33] Helei Duan, Ashish Malik, Jeremy Dao, Aseem Saxena, Kevin Green, Jonah Siekmann, Alan Fern, and Jonathan Hurst. Sim-to-real learning of footstep-constrained bipedal dynamic walking. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10428–10434. IEEE, 2022. 5.1

[34] Rohan P. singh, Zhaoming Xie, Pierre Gergondet, and Fumio Kanehiro. Learning bipedal walking for humanoids with current feedback. *IEEE Access*, 11:82013–82023, 2023. 5.1

[35] J Merel, L Hasenclever, A Galashov, A Ahuja, V Pham, G Wayne, Y Teh, and N Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2019. 5.1, 6.3, 7.2

[36] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. 5.4

[37] Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. Allsteps: Curriculum-driven learning of stepping stone skills. *Computer Graphics Forum*, 39(8):213–224, 2020. 5.5

[38] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *JOURNAL OF MACHINE LEARNING RESEARCH*, 21, 2020. 5.6

[39] Nolan Wagener, Andrey Kolobov, Felipe Vieira Frujeri, Ricky Loynd, Ching-An Cheng, and Matthew Hausknecht. Mocapact: A multi-task dataset for simulated humanoid control. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 35418–35431. Curran Associates, Inc., 2022. 6.2, 7.1, 7.2, 7.1

[40] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. 7.2

[41] Farzad Abdolhosseini, Hung Yu Ling, Zhaoming Xie, Xue Bin Peng, and Michiel van de Panne. On learning symmetric locomotion. In *Proceedings of the 12th ACM SIGGRAPH Conference on Motion, Interaction and Games*, MIG '19, New York, NY, USA, 2019. Association for Computing Machinery. 8.1