Symmetry-Aware RL Kick

Julia A. de Vries

Layout: typeset by the author using LaTeX.
Cover illustration: J.A. de Vries

Symmetry-Aware RL Kick

Incorporating Symmetry into Reinforcement Learning to Train a Kicking Policy for the Booster K1

Julia A. de Vries
13322532

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester 1, 2025-2026

Abstract

This thesis investigates whether incorporating symmetry into reinforcement learning can improve accuracy, training efficiency, and generalisation for humanoid robot motor control. A ball-kicking task is used to evaluate two symmetry-aware approaches (data augmentation and equivariant neural networks) against a standard Proximal Policy Optimisation baseline. Both symmetry-aware methods outperform the baseline in accuracy, training stability, and generalisation. The equivariant approach performs best overall. In a generalisation test where training was restricted to one side of the workspace, the baseline fails completely on the mirrored configuration while both symmetry-aware approaches maintain consistent performance. The baseline also develops asymmetric leg preferences in all training runs, suggesting convergence to path-dependent local optima that the symmetry constraints prevent. The results provide evidence that incorporating structural symmetry priors improves performance in robotic motor control.

Contents

1 Introduction **3**
1.1 Related Works . . . 4

2 Theoretical Background **6**
2.1 Reinforcement Learning . . . 6
2.1.1 Markov Decision Process . . . 6
2.1.2 Actor-Critic Methods . . . 7
2.1.3 Proximal Policy Optimisation . . . 7
2.2 Equivariant Neural Networks: Representations and Equivariant Layers . 8
2.2.1 Regular Representation . . . 9
2.2.2 Sign Representation . . . 10
2.2.3 G-Equivariant Mappings . . . 10
2.2.4 Constructing Equivariant Networks . . . 11
2.2.5 Multi-field Representations and Block-Structured Layers . . . 14

3 Methods **16**
3.1 Training Environment . . . 16
3.2 Reinforcement Learning Formulation . . . 18
3.2.1 Observation and Action space . . . 19
3.2.2 Terminal States . . . 20
3.2.3 Reward . . . 21
3.2.4 Symmetry-Based Data Augmentation . . . 23
3.3 Equivariant Neural Networks . . . 24
3.3.1 Representations for the Input, Hidden, and Output spaces . . . 24
3.3.2 Equivariant Actor and Critic Models . . . 26

4 Experiments and Results **29**
4.1 Experimental Setup . . . 29
4.1.1 Task Definition . . . 29
4.1.2 Terminal States . . . 29

4.1.3 Reward 30
4.2 Experiments 32
4.2.1 General 33
4.2.2 Home Field Advantage 34
4.3 Results 36
4.3.1 General 36
4.3.2 Home Field Advantage 39

5 Conclusion 40
5.1 Future Work 41

Appendices 45

A Remaining Reward Descriptions 45

B Training Hyperparameters 48

C Representation for Angular Velocity 50

D Extra Plots 51

Chapter 1

Introduction

Humanoid robot football offers a challenging platform for research and development in Artificial Intelligence (AI) and robotics (Kitano et al. 1997). Football, while also being appealing to the public, provides a highly dynamic environment in which a wide range of technologies can be evaluated. Demonstrating proficiency in the sport requires capabilities such as real-time rapid decision-making, teamwork and precise as well as efficient body control. Consequently, robot football necessitates seamless integration of a broad range of technologies, including computer vision, multi-agent collaboration, strategy, planning, intelligent locomotion and sensor fusion.

One way to approach robot football is by separately handling high-level decision-making, such as strategic planning, and low-level motor skill. An important part of football is the ability to kick the ball to a specific target, for example a teammate or between the goalposts. It requires a robot to be able to take into account both external factors, such as the relative position of the ball and target, as well as internal factors, such as the joint positions of the robot. To overcome the tedious manual programming required to realise dynamic movements, reinforcement learning has shown great promise (Atanassov et al. 2024; Spitznagel, D. Weiler, and Dorer 2021; Y. Wang, P. Chen, et al. 2025).

However, constructing a reward function for kicking behaviour is difficult as various skills need to be combined to achieve the task. The robot needs to be able to stay standing up, move towards the ball, orient itself so it is able to kick the ball towards the target and move the ball with its foot in an efficient and precise manner. In fact, the robot has to learn all of this twice. It has to learn to deal with the ball being to its left and to its right. It then also runs the risk of learning it in an unprincipled, asymmetric way. A player who can only strike with their dominant foot would likely play suboptimally in many situations. They must often make an extra effort to reposition the ball, which generates a delay that allows defenders to close the gap and negate the opportunity. In a high-speed, dynamic game, the ability to treat both sides of the body as functionally equivalent is a tactical advantage. It turns out that this issue of learning symmetric motor skills has a simple and well-understood geometric structure. When a humanoid robot kicks a ball to its left versus to its right, the two scenarios are mirror images of each other. The robot's bilateral morphology is symmetric, the task objectives are symmetric, and the sensorimotor loop connecting observations to actions should respect this symmetry. There are even mature frameworks for how to incorporate these symmetries in neural networks and learning (Bronstein et al. 2021; Mittal et al. 2024; M. Weiler et al. 2023). This raises the question of whether these symmetry-aware techniques can improve and simplify the reinforcement learning of such a kick. This thesis addresses two sets of intertwined research questions:

- Can reinforcement learning successfully train a humanoid robot to kick a ball accurately toward a target? In particular by simply using proximal policy optimization (PPO) (Schulman et al. 2017). This requires investigating which reward functions are effective, and whether the resulting policy can kick the ball into a small target area.
- Does incorporating symmetry knowledge into this learning process improve performance? Specifically, does it simplify the learning problem, accelerate convergence, produce better final policies, and/or improve generalisation to novel initial conditions? Two approaches for incorporating symmetry will be compared: symmetry-based data augmentation and equivariant neural networks.

1.1 Related Works

A popular approach for robotic locomotion has been model-based control, which uses an explicit mathematical model of the environment, and has shown great performance but often limits the search space, resulting in conservative performance (Kalakrishnan et al. 2010). Model-free reinforcement learning has been shown to be able to overcome these limitations (Atanassov et al. 2024; Lee et al. 2020). Recent work in end-to-end reinforcement learning has been able to realise complex humanoid skills, such as walking (Y. Wang, P. Chen, et al. 2025), table tennis (Hu et al. 2025), and ball dribbling. However, designing the reward function for complex tasks is often challenging and requires long training cycles due to the large search space.

The policy (and critic) are generally implemented as a trainable generic neural network like a MLP, CNN, RNN, etc. Such networks can learn different behaviour for situations that are mirrored with respect to the chirality of the robot. In other words, they can learn a different kick for both legs even if the ball is in an identical position relative to that side of the robot. This expressivity is however not necessarily wanted, as it can converge to learning to prefer one leg over the other and fail to generalise to the left-right symmetry in the task. This has been noted by others and it has been proposed to make use of this symmetry through data augmentation and mirror loss function (Mittal et al.

2024). There is however a subfield of machine learning that studies how to construct neural networks that ensure such symmetric behaviour in their architecture (Bronstein et al. 2021; M. Weiler et al. 2023). Such equivariant neural networks have found success in tasks such as CartPole, Pong, robotic arm tasks (Nguyen et al. 2023; Van der Pol et al. 2020; D. Wang, Walters, and Platt 2022). They have furthermore shown strong generalisation when applied to real-world scenarios (Su et al. 2024). However, this approach has not been widely explored for the humanoid robotic task of ball-kicking, but has shown promising results for humanoid robotic tasks such as walking (Abdolhosseini et al. 2019; Nie et al. 2025).

Chapter 2

Theoretical Background

2.1 Reinforcement Learning

Reinforcement learning (RL) refers to an agent learning from interaction with an environment. A learner is not told what to do but must discover by exploring different actions which actions yield the most rewards (Sutton, Barto, et al. 1998). An agent's objective in RL is to learn a policy π that maps states (or observations) to a distribution over actions, maximising the expected cumulative reward over time.

2.1.1 Markov Decision Process

Reinforcement learning problems are often modeled as a Markov decision process (MDP) (Sutton, Barto, et al. 1998). An MDP is formally defined as a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where

- $\mathcal{S}$ denotes the set of all possible states,
- $\mathcal{A}$ the set of possible actions,
- r the immediate reward function as a function of the current state s and action a,
- T the state transition function $p(s'|s, a)$,
- $\gamma \in [0, 1]$ the discount factor.

The goal is to find a policy $\pi(a|s)$ that maximises the expected cumulative discounted return:

$$G_t = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t\Big],$$

where the expectation is taken over trajectories generated by following policy π.

In practice, an agent in the real world (such as a robot) does not have direct access to the complete state. Since robot control problems often must work with incomplete or noisy observations, they are frequently modeled as partially observable Markov decision processes (POMDPs) (Kaelbling, Littman, and Cassandra 1998). In a POMDP, the policy maps observations to actions $\pi(a|o)$ rather than complete states to actions. Having made that caveat, state and observation will be used interchangeably in this thesis.

2.1.2 Actor-Critic Methods

Policy gradient methods are well-suited for continuous action spaces and directly optimise a parameterised stochastic policy $\pi_\theta(a|s)$, where θ denotes the network parameters. Since this thesis addresses a robot motor control problem, where a continuous action space is most natural, policy gradient methods are most suitable. The gradient of the objective function takes the form:

$$\nabla_\theta J(\theta) = \mathbb{E}_t\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t\Big],$$

where $\hat{A}_t$ is an estimate of the advantage at timestep t. The advantage measures how much better an action is compared to the average action in that state. This update rule reinforces behaviour based on the advantage: when $\hat{A}_t > 0$, the probability of the taken action is increased; when $\hat{A}_t < 0$, the probability is decreased.

To estimate the advantage, actor-critic methods employ two neural networks:

- The actor $\pi_\theta(a|s)$ is the policy network that selects actions.
- The critic $V_\varphi(s)$ is a value function that estimates the expected return from a given state.

The value function $V^\pi(s)$ represents the expected return when starting from state s and following policy π. The critic network $V_\varphi(s)$ is a parameterised approximation of this value function, trained to minimise the prediction error between its estimates and the observed returns. The advantage can then be estimated as the difference between the observed return and the critic's prediction $V_\varphi(s)$. By training the critic alongside the policy, the actor-critic method reduces the variance of the policy gradient estimator, leading to more stable and efficient learning.

2.1.3 Proximal Policy Optimisation

Standard policy gradient updates are sensitive to step size. Because the policy gradient is estimated from a finite batch of sampled trajectories, there is uncertainty in the gradient estimate. Updating the policy too aggressively based on this noisy estimate can cause the policy to move into regions of parameter space where it performs poorly, leading to catastrophic performance collapse from which recovery is difficult. Proximal Policy Optimisation (PPO) addresses this issue by constraining updates to a ‘trust region’. This is a neighborhood around the current policy where the gradient estimate is considered reliable (Schulman et al. 2017).

PPO introduces a probability ratio $r_t(\theta)$ between the new and old policies:

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

To prevent the policy from diverging too far, PPO uses a clipped surrogate objective:

$$L^{CLIP}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\Big)\Big]$$

where ε is a hyperparameter (typically $\varepsilon = 0.2$) determining the size of the trust region. When $r_t(\theta)$ moves outside the interval $[1-\varepsilon, 1+\varepsilon]$, the advantage term is clipped, removing the incentive for further updates in that direction. This ensures that the policy changes gradually, preventing destructive updates while still allowing meaningful learning progress. PPO combines this clipped policy objective with a value function loss for training the critic.

2.2 Equivariant Neural Networks: Representations and Equivariant Layers

A group G is a set of transformations equipped with a binary operation that satisfies closure, associativity, and must contain an identity element and an inverse element for each element. Some examples are:

- The symmetric group (of order n) S_n is the group of all possible permutations (rearrangements) of n distinct objects. For example, $S_2 = \{e, s\}$ represents the two ways to arrange two items: keeping them as they are (e) or swapping them (s).
- The cyclic group (of order n) C_n is the group of n circular shifts or discrete rotations.
- The special orthogonal group $SO(n)$ is the group of continuous rotations in $\mathbb{R}^n$ or $\mathbb{Z}^n$ like $SO(2)$ for 2D images or $SO(3)$ for 3D shapes.
- The translation groups $\mathbb{R}^n$ or $\mathbb{Z}^n$ are the groups of (continuous and discrete) shifts or translations in n-dimensional Euclidian space.

In this thesis, only the group with the identity (e) and a flip or transposition (s) will be relevant. The elements are defined by $s^{-1} = s$ (i.e. $s \cdot s = e$) and e follows the rules of the identity element. As this group has such a simple structure, it is isomorphic to members of multiple families of groups (note that $\{e, s\} \cong S_2 \cong C_2$), but here it will be referred to as S_2 or the symmetric group of order 2.

The elements of such groups can be interpreted to act on objects. For example, $s \in S_2$ could be thought of as flipping a point around an axis. Another example would be the elements from the special orthogonal group $SO(3)$ rotating a molecule. The goal of equivariant neural networks (ENNs) is to build models that behave predictably under such group actions (Bronstein et al. 2021; M. Weiler et al. 2023). Representation theory provides a way to "represent" abstract groups as concrete linear transformations on vector spaces. Formally, a linear representation (ρ, V) of a group G on a vector space V on a field $\mathbb{F}$ (such as the real or complex numbers) is a homomorphism $\rho : G \to GL(V)$, where $GL(V)$ is the group of invertible linear transformations of V. This means that for any $g_1, g_2 \in G$, it holds that $\rho(g_1 g_2) = \rho(g_1)\rho(g_2)$, and $\rho(e) = I_V$, where e is the identity element of G and I_V is the identity transformation on V. The respective V is called the representation space. It turns out that all groups can be represented linearly (Simon 1996). In other words, one can consider any symmetry in terms of its representation as matrices in the way just described. In the context of ENN, the representation spaces V are the feature spaces of the data points and the intermediate and final states in the network. The representation ρ describes how these features should transform when the input data undergoes a symmetry operation from the group G. Furthermore, although complex numbers naturally arise as the underlying field in representation theory, equivariant neural networks are typically constructed purely in terms of real vector spaces and real-valued matrices ($\mathbb{F} = \mathbb{R}$) (Cesa, Lang, and M. Weiler 2022; Lang and M. Weiler 2021).

2.2.1 Regular Representation

The regular representation represents group actions with permutation matrices (Simon 1996). Since group elements are defined by the way they compose with the other group elements, they are in a sense defined by the way they permute the group.[1] The matrices of the regular representation are constructed by first letting each group element correspond to a basis vector in the representation space. The matrices representing the group elements then are the linear transformations that permute this basis exactly like the composing with the respective element would permute the group.

More formally, the (left) regular representation (ρ, V) is vector space V over field $\mathbb{F}$ with a basis $\{e_g \mid g \in G\}$. So this space is $|G|$-dimensional. The representation map $\rho_{\text{reg}} : G \to GL(V)$ is a group homomorphism such that for each $h \in G$, the linear transformation $\rho_{\text{reg}}(h) : V \to V$ is determined by its action on the basis vectors:

$$\rho_{\text{reg}}(h)(e_g) = e_{hg} \quad \forall g \in G$$

[1]This observation is known as Cayley's theorem and the regular representation can be seen as a linear (representation) version of it (Simon 1996).

As an illustration, consider again the (real) regular representation of the symmetric group of degree 2, $S_2 = \{e, s\}$. The representation space is $\mathbb{R}^2$ with basis $\{e_e, e_s\}$. The representation maps $\rho(g)$ for $g \in S_2$ are given by $\rho(g)(e_h) = e_{gh}$. The corresponding representation matrices with respect to the basis $\{e_e, e_s\}$ are:

$$\rho_{\text{reg}}(e) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \rho_{\text{reg}}(s) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

2.2.2 Sign Representation

Another representation is the sign representation. It reduces the actions of the group elements to the parity of their respective permutations.

More formally, let S_n be the symmetric group on n elements, and let $\mathbb{F}$ be a field. The sign representation (ρ_{sign}, V) is a one-dimensional vector space V over $\mathbb{F}$ ($V = \mathbb{F}$). The representation map $\rho_{\text{sign}} : S_n \to GL(V)$ is defined by:

$$\rho_{\text{sign}}(g) = \begin{cases} 1 & \text{if } g \text{ is an even permutation} \\ -1 & \text{if } g \text{ is an odd permutation} \end{cases}$$

When viewed as a linear transformation on a one-dimensional space, $\text{sign}(g)$ maps a vector $v \in V$ to $-1 \cdot v$.

Consider again how this would work for the symmetric group $S_2 = \{e, s\}$. Here e is the identity (an even permutation) and s is a single transposition (an odd permutation). With $V = \mathbb{R}$, the (real) sign representation maps these elements as follows:

$$\rho_{\text{sign}}(e) = 1, \quad \rho_{\text{sign}}(s) = -1.$$

2.2.3 G-Equivariant Mappings

An important object of study in representation theory is the set of G-equivariant mappings between representation spaces. These are functions that preserve the geometric structure by commuting with the group actions defined by their respective representations. When such mappings are linear, they are often called intertwiners. This property implies that the transformation of an input by a group action directly corresponds to the transformation of its output in the output space.

More formally, a function $f : X \to Y$ is said to be equivariant with respect to the action of a group G on X and Y (via representations ρ_X on X and ρ_Y on Y respectively) if for all $g \in G$ and $x \in X$, the following holds:

$$f(\rho_X(g)(x)) = \rho_Y(g)(f(x))$$

In the context of neural networks, f represents a layer of the network, X the input feature space, and Y the output feature space. Equivariance means that if the input data is transformed according to some group action ($\rho_X(g)(x)$), the output of the network layer will be transformed in a corresponding way ($\rho_Y(g)(f(x))$). The representations ρ_X and ρ_Y dictate how the input and output features should transform under the relevant group actions. This property can also be represented graphically. It is the same as asserting that the diagram in fig. 2.1 commutes for all elements in X and all $g \in G$.

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\rho_X(g)"'] & Y \arrow[d, "\rho_Y(g)"'] \\
X \arrow[r, "f"] & Y
\end{tikzcd}$$

Figure 2.1

In other words, for any input $x \in X$ and any group element $g \in G$, applying the function (f) and then transforming the result with the output representation ($\rho_Y(g)$) yields the same result as first transforming the input with the input representation ($\rho_X(g)$) and then applying the function (f).

Invariance is a special case of equivariance, where the representation on the output space is the trivial representation. A function $f : X \to Y$ is said to be invariant with respect to the action of a group G on X (via representation ρ_X) if for all $g \in G$ and $x \in X$, the following holds:

$$f(\rho_X(g)(x)) = \rho_{\text{triv}}(g)(f(x)) = f(x)$$

In the context of neural networks, an invariant layer or an entire invariant network produces an output that remains unchanged when the input undergoes a symmetry transformation. This is often desired for tasks like classification, where the class label should be independent of the object's orientation or position. An example from reinforement learning would be a value function or critic whose evaluation of a state may also be invariant under certain transformations. fig. 2.2 and fig. 2.3 are two equivalent graphical ways to represent this property.

2.2.4 Constructing Equivariant Networks

To construct a fully equivariant neural network, the entire composition of layers must preserve the group structure. Since equivariance is transitive, deep equivariant models are built by stacking equivariant layers. Since a neural network layer in essence consists of compositions of linear transformation followed by a non-linear activation function,

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\rho_X(g)"'] & Y \arrow[d, "\rho_{\mathrm{triv}}(g)"'] \\
X \arrow[r, "f"] & Y
\end{tikzcd}$$

Figure 2.2

$$\begin{tikzcd}
X \arrow[rd, "f"] \arrow[dd, "\rho_X(g)"'] & \\
& Y \\
X \arrow[ru, "f"'] &
\end{tikzcd}$$

Figure 2.3

it suffices to construct version of these layers that satisfies the relevant equivariance constraint.[2] These constructions depend on the group and the representation under consideration. There is a rather sophisticated general framework for solving these constraints (Lang and M. Weiler 2021; M. Weiler et al. 2023). However, for the simple case of S_2 under a handful of representations, it is easier to consider their particular solutions. In particular, the layers that are actually used in the experiments for this thesis will be explained here.

For a linear map $L : X \to Y$, the general equivariance constraint $f(\rho_X(g)x) = \rho_Y(g)f(x)$ simplifies significantly. By treating $f(x)$ as the matrix-vector product Lx, the x can be factored out, resulting in the linear constraint:

$$L\rho_X(g) = \rho_Y(g)L.$$

This condition implies that the matrix L itself must commute with the group action. Depending on the input representation ρ_X and output representation ρ_Y, this constraint restricts the possible values of the weights in the matrix L.

Linear Layer Solutions for S_2

For $S_2 = \{e, s\}$, the representations used in the policy are the trivial $\rho_{\mathrm{triv}}(s) = 1$, the sign $\rho_{\mathrm{sign}}(s) = -1$, the standard regular $\rho_{\mathrm{reg}}(s) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, and a fourth representation for paired features that are swapped and negated, which here will be called the Regular + Sign Flip representation, $\rho_{\mathrm{reg+sign}}(s) = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$. This representation is the tensor product $\rho_{\mathrm{reg}} \otimes \rho_{\mathrm{sign}}$, combining the swapping behaviour of the regular representation with the sign flip of the sign representation. The resulting equivariant linear layers, which form the building blocks of the network, are structured as follows:

- **Sign to Regular:** For a linear map $L : \mathbb{R} \to \mathbb{R}^2$, the constraint $L\rho_{\mathrm{sign}}(s) = \rho_{\mathrm{reg}}(s)L$ requires that swapping the output rows is equivalent to negating the input. This

[2]The same principles hold for other layers types, such as various forms of pooling or bias summation.

forces the two weights to be anti-symmetric: $L = \begin{pmatrix} w \\ -w \end{pmatrix}$.

- **Regular to Regular:** For a linear map $L : \mathbb{R}^2 \to \mathbb{R}^2$, the constraint $L\rho_{\text{reg}}(s) = \rho_{\text{reg}}(s)L$ requires the matrix to commute with the permutation matrix. This results in a circulant (weight-sharing) structure: $L = \begin{pmatrix} w_1 & w_2 \\ w_2 & w_1 \end{pmatrix}$.
- **Regular to Regular + Sign Flip:** For a map $L : \mathbb{R}^2 \to \mathbb{R}^2$ from a regular input to a "Regular + Sign Flip" output, the constraint $L\rho_{\text{reg}}(s) = \rho_{\text{reg+sign}}(s)L$ forces the weight matrix to have an anti-circulant structure: $L = \begin{pmatrix} w_1 & w_2 \\ -w_2 & -w_1 \end{pmatrix}$.
- **Regular + Sign Flip to Regular:** For the reverse map $L : \mathbb{R}^2 \to \mathbb{R}^2$, the constraint $L\rho_{\text{reg+sign}}(s) = \rho_{\text{reg}}(s)L$ results in the same anti-circulant matrix structure: $L = \begin{pmatrix} w_1 & w_2 \\ -w_2 & -w_1 \end{pmatrix}$.
- **Regular to Sign:** For a linear map $L : \mathbb{R}^2 \to \mathbb{R}$, the constraint $\rho_{\text{sign}}(s)L = L\rho_{\text{reg}}(s)$ requires that swapping the input elements negates the output. This results in a difference-based mapping: $L = \begin{pmatrix} w & -w \end{pmatrix}$.
- **Regular to Trivial:** For a linear map $L : \mathbb{R}^2 \to \mathbb{R}$, the constraint $\rho_{\text{triv}}(s)L = L\rho_{\text{reg}}(s)$ requires that swapping the input elements has no effect on the output. This results in a summation-based mapping where both weights are identical: $L = \begin{pmatrix} w & w \end{pmatrix}$.

Non-linear Equivariance

For a non-linear activation function $\sigma : X \to Y$ to be equivariant, it must satisfy $\sigma(\rho(g)v) = \rho(g)\sigma(v)$. In the case of the regular representation ρ_{reg} of S_2, the group action simply permutes the components of the vector $v = (v_1, v_2)^T$. Any pointwise activation function (such as ReLU or ELU) naturally satisfies this, as applying the function to each component independently commutes with the permutation:

$$\sigma\left(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}\right) = \sigma\left(\begin{pmatrix} v_2 \\ v_1 \end{pmatrix}\right) = \begin{pmatrix} \sigma(v_2) \\ \sigma(v_1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\sigma\left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}\right).$$

Consequently, standard point-wise deep learning activations can be used within regular representation layers without modification.

2.2.5 Multi-field Representations and Block-Structured Layers

All these solutions assume the input and output transform with some respective representation type (and group) while in practice one may deal with a more diverse set of input features and outputs which may transform under different transformations. It turns out that in such a case, these different constraints can be solved one at a time and then stacked on top of each other by a direct sum. When multiple fields are concatenated into a single vector, the group action is represented by a block-diagonal matrix:[3]

$$\rho_{\text{total}}(g) = \rho_1(g) \oplus \rho_2(g) \oplus \cdots \oplus \rho_k(g) = \begin{pmatrix} \rho_1(g) & 0 & \cdots \\ 0 & \rho_2(g) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

The equivariance constraint for a linear layer L acting between an input space with representations $\{\rho_{X,j}\}$ and an output space with representations $\{\rho_{Y,i}\}$ decomposes into independent constraints for each block of the weight matrix. Specifically, if L is partitioned into blocks L_{ij} corresponding to the j-th input field and the i-th output field, each block must independently satisfy their respective equivariance constraint.

This formulation allows for the construction of equivariant multilayer perceptrons (MLPs) by stacking these block-structured layers. For example, consider a simple architecture with the following specifications:

- Input: Two fields transforming under the regular representation and one field transforming under the sign representation (total dimension 5).
- Hidden Layer: A hidden state transforming under the regular representation (dimension 2).
- Output: A single element transforming under the sign representation (dimension 1).

The first linear layer $L^{(1)}$ is a 2×5 matrix composed of three equivariant blocks: two 2×2 regular-to-regular blocks and one 2×1 sign-to-regular block. The resulting weight matrix takes the form:

$$L^{(1)} = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \\ w_2 & w_1 & w_4 & w_3 & -w_5 \end{pmatrix}.$$

Since the hidden layer transforms under the regular representation of S_2, RELU (a point-wise activation function) can be used while preserving equivariance. The final layer $L^{(out)}$ maps from the 2D hidden regular representation to the 1D sign representation. This 1×2 matrix must satisfy the regular-to-sign constraint:

[3]Here $\oplus$ is the direct sum.

$$L^{(out)} = \begin{pmatrix} w_6 & -w_6 \end{pmatrix}.$$

This block-wise parameterisation ensures that the entire MLP remains equivariant, effectively enforcing weight sharing and anti-symmetry across the different feature types. Furthermore, notice that the input and output representations are generally determined by the problem set up, while the representation under which the hidden layer(s) should transform was an architectural design choice.

Chapter 3

Methods

3.1 Training Environment

The simulation environment is constructed using the mjlab environment, as it has proven RL abstractions, direct physics control, GPU acceleration, and parallel simulation and training (Zakka et al. 2025). It allows an environment to be initialised in some initial state and then actions of the agent(s) can be sampled with some set step frequency and the underlying physics engine handles the rest. Two entities have been introduced to the simulation environment on top of the default entities like the floor: a football and a Booster K1 humanoid robot, as show in fig. 3.1. The characteristics of these entities and their interactions are designed to create a realistic and challenging scenario for the agent. The robot's default starting pose is a neutral standing position with an approximate total height of 0.84 meters. The basic physical properties are summarised in table 3.1.

The environment uses a full-body contact model, allowing for realistic physical interactions between the robot and the ball. Specific contact sensors are configured to detect collisions both between the robot's feet and the ball, and between any other part of the robot's body and the ball. These detected contacts are used to calculate rewards and determine terminations, enabling the agent to learn kicking behaviours. The ball has fixed friction coefficients defined for three types of interaction. These values are rough estimates and could be chosen more carefully if there is a better sense of the balls and terrains to expect.

- Sliding friction (1.0): Resists translational motion when the ball is in contact with a surface.
- Torsional friction (0.5): Resists spinning motion around the contact normal (for example spinning in place).
- Rolling friction (0.015): Resists rolling motion.

Entity	**Property**	**Value**
Robot (K1)	Approx. Standing Height	0.84 m
	Trunk Mass	6.39 kg
	Foot Friction Coefficient	Randomised: [0.3, 1.2]
	Joint Friction Loss	0.1 (per joint)
Ball	Mass	0.1 kg
	Radius	0.08 m
	Friction Coefficients	(1.0, 0.5, 0.015)
Physics	Interaction Model	Full-body contact physics
	Simulation Step	2 ms (500 Hz)
	Control Step (Agent)	20 ms (50 Hz)

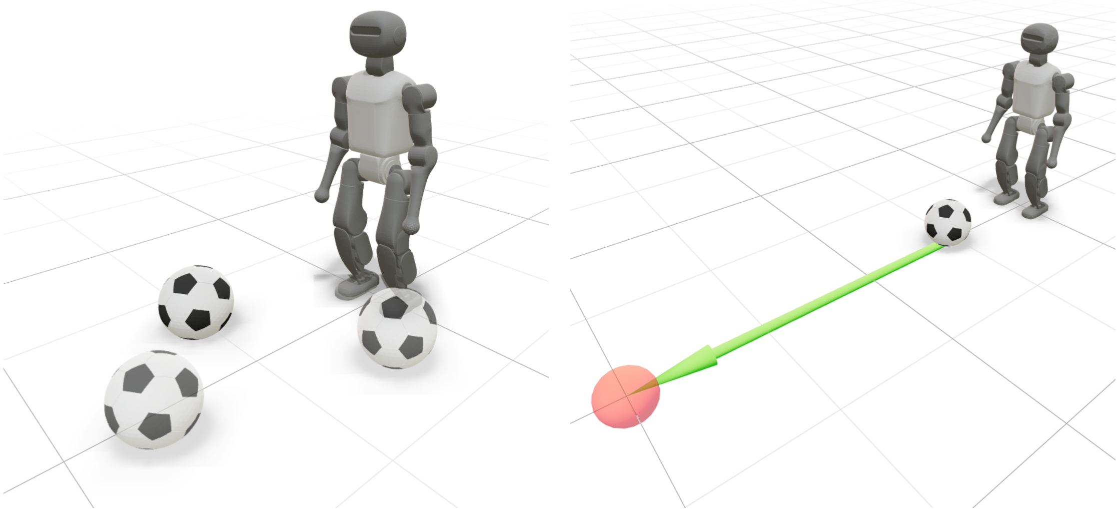
Table 3.1: Basic Physical Properties of Entities in the Kick Environment.

To enhance policy robustness, the environment systematically introduces several forms of noise and randomisation. Such domain randomisation ensures the agent does not overfit to the exact setup and noiseless physics of the simulation.

- Robot's foot friction: At the start of each episode, the sliding friction coefficient for each of the robot's feet is set to a new, random value sampled uniformly from the range $[0.3, 1.2]$. This forces the agent to learn policies that are not overly dependent on a single, specific friction value.
- Sensor Noise: Small, uniformly sampled noise is added to most proprioceptive and exteroceptive observations at each timestep. This includes the robot's base linear and angular velocities, joint positions and velocities, and the relative position of the ball. This simulates the imperfections of real-world sensors.
- Action Delay: The commands sent to the robot's motors are not applied instantly. Instead, they are passed through a delay buffer that introduces a random lag of between 2 and 8 simulation steps (10-40 ms), with a 30% chance of holding the previous command for an additional step. This models real-world communication latencies and actuator response times.
- Physical Perturbations: At random intervals during each episode (every 1 to 3 seconds), a random planar force is applied to the robot's torso. This "push" simulates external disturbances and forces the agent to learn a more stable, reactive policy that can recover from unexpected impacts.
- Initial State Randomisation: At the start of each episode, the ball is placed at a

random position in a wide area in front of the robot. This ensures the agent learns a general kicking behaviour rather than memorising a single approach trajectory.

- Target Randomisation: The target is sampled at a randomised distance and angle with respect to the agent.



(a) (b)

Figure 3.1: The simulation environment where (a) demonstrates a configuration with 3 possible initial ball positions and (b) showcases debugging visualisation for an initial ball and target position.

3.2 Reinforcement Learning Formulation

The robot control problem is formulated as a POMDP (Kaelbling, Littman, and Cassandra 1998), where the agent must determine its action distribution based on sensory and proprioceptive data available at deployment, along with its previous action. The learning algorithm used is Proximal Policy Optimisation (PPO) (Schulman et al. 2017), an on-policy actor-critic algorithm particularly well suited for this task.

PPO is chosen for several reasons. First, its clipped policy objective ensures stability during training. This is important for the complex, high-dimensional task of humanoid locomotion and ball manipulation, where policy collapse can be difficult to recover from (Schulman et al. 2017). Second, as an on-policy method, PPO naturally leverages massively parallel simulation environments, enabling efficient data collection across thousands of simultaneous rollouts. This is particularly advantageous in GPU-accelerated simulators like mjlab, where the marginal cost of additional parallel environments is minimal Zakka et al. 2025. Third, PPO has demonstrated strong empirical performance on similar humanoid locomotion and manipulation tasks (Y. Wang, P. Chen, et al. 2025; Y. Wang, Luo, et al. 2025; Z. Wang, Zhou, and Q. Wu 2025), establishing it as a reliable baseline for legged robot control.

Observation	**Actor**	**Critic**
Base Linear Velocity	✓	✓
Base Angular Velocity	✓	✓
Projected Gravity	✓	✓
Ball Relative Position	✓	✓
Target Position	✓	✓
Joint Positions	✓	✓
Joint Velocities	✓	✓
Previous Action	✓	✓
Foot Height		✓
Foot Air time		✓
Foot Contact		✓
Foot Contact Forces		✓
Ball Velocity		✓
Ball-Foot Contact		✓

Table 3.2: Observation space for the Actor and Critic networks.

3.2.1 Observation and Action space

An asymmetric actor-critic architecture is used, where both networks share most observations but the critic receives additional privileged information. While both networks observe the information listed as available to the actor in table 3.2, the critic additionally has access to ground-truth state information such as actual ball velocity, precise foot contact forces, and exact foot heights. This asymmetry allows the critic to provide more accurate value estimates during training, reducing variance in the advantage estimates without compromising the deployability of the actor policy. During deployment, only the actor policy is used, relying solely on the observations marked for the actor in table 3.2.

The action space consists of 22-dimensional continuous joint position targets, one for each actuated joint of the K1 humanoid robot. These targets are specified as offsets from a default standing pose, allowing the agent to modulate its posture and movement while maintaining a stable reference configuration. An underlying proportional-derivative (PD) controller in the simulator translates these joint position targets into low-level actuator torque commands.

3.2.2 Terminal States

Each episode begins with a state sampled from a chosen distribution and resets upon reaching a terminal state. The episode terminates if any of the conditions listed in table 3.3 evaluate to true at timestep t. Here, c_t denotes the contact counter, which tracks the number of timesteps since the first contact between the agent and the ball occurred. Note that an episode is terminated after completion of the step in which the termination condition is satisfied. Therefore, the reward earned during the final step is included in the total episodic return.

Termination	**Description**	**Equation**
Time out	Episode exceeded length	$t > T_{\text{max}}$
Fell over	Orientation exceeds angle limit	$\theta_t > \theta_{\text{limit}}$
Fell down	Trunk height below minimum	$z_t^{\text{trunk}} < z_{\text{falldown}}$
Target hit	Ball stopped within target radius	$(\lVert \mathbf{d}_t^{\text{target,ball}} \rVert < r) \wedge (\lVert \mathbf{v}_t^{\text{ball}} \rVert < \varepsilon)$
Target missed	Ball stopped outside target radius	$(\lVert \mathbf{d}_t^{\text{target,ball}} \rVert > r) \wedge (\lVert \mathbf{v}_t^{\text{ball}} \rVert < \varepsilon) \wedge (c_t > T_{\text{window}})$
Double touch	Illegal contact after window	$(c_t > T_{\text{window}}) \wedge (contact_t^{\text{agent,ball}})$

Table 3.3: Terminal States

To accelerate initial training, four early terminations are defined. A time out occurs when the episode surpasses the maximum episode length $t > T_{\text{max}}$. Falling over is defined as the agent's tilt angle θ_t exceeding a threshold θ_{limit}. The tilt θ_t is defined as the deviation of the robot's base z-axis from the global vertical axis. Falling down is defined as the agent's trunk height z_t^{trunk} dropping below a height threshold z_{falldown}. Additionally, double touch termination, is included, which ends the episode if the agent is in contact with the ball $contact_t^{\text{agent,ball}}$ after a specified contact window has passed $c_t > T_{\text{window}}$. Here $contact_t^{\text{agent,ball}}$ is a boolean whose truth value is determined by whether the agent and ball are physically in contact.

To track task performance, two outcome terminations are defined. Target hit is true when the ball has come to rest ($\lVert \mathbf{v}_t^{\text{ball}} \rVert < \varepsilon$) within the target radius ($\lVert \mathbf{d}_t^{\text{target,ball}} \rVert < r$). Conversely, target missed is true when the ball has come to rest outside the target radius. Note that the target missed termination requires that the contact window has passed $c_t > T_{\text{window}}$. This prevents premature termination at the start of an episode when the ball is stationary but the agent has not yet attempted to move it. This check is redundant for the target hit termination, as the ball cannot have entered the target radius without having moved.

3.2.3 Reward

The reward function consists of multiple components. However, only the the main components specifically developed for the task in this thesis will be discussed here. For description of the remaining reward components see appendix A.

Target-Ball Distance

Target-ball distance reward is a sparse reward at the end of an episode when the ball has been kicked and is nearly stagnant. This is done by taking the global positions of the target and the ball. The positional difference vector on the xy-plane is then defined as:

$$\mathbf{d}_t^{\text{target,ball}} = \begin{pmatrix} x^{\text{target}} - x^{\text{ball}} \\ y^{\text{target}} - y^{\text{ball}} \end{pmatrix}.$$

A reward is only given if a kick has been attempted. This is done by checking whether the contact window at the current timestep has reached a specified threshold $c_t > T_{\text{window}}$. To avoid the agent from overshooting, the ball's velocity $||\mathbf{v}_t^{\text{ball}}||$ is required to be below a specified threshold ε. When these conditions are satisfied, the reward is calculated as an exponential decay function of the Euclidean distance between the ball and the target on the xy-plane $||\mathbf{d}_t^{\text{target,ball}}||$.

$$r_t^{\text{target-reached}} = \begin{cases} \exp(-\frac{||\mathbf{d}_t^{\text{target,ball}}||}{\sigma^2}) & \text{if} \quad c_t > T_{\text{window}} \quad ||\mathbf{v}_t^{\text{ball}}|| < \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

Ball Approach Target

The ball approach target reward is a dense reward that encourages the agent to align the ball velocity with the horizontal difference vector from the ball to the target.

$$r_t^{\text{b-approach-t}} = \begin{cases} \max(0, \frac{\mathbf{v}_t^{\text{ball}} \cdot \mathbf{d}_t^{\text{target,ball}}}{||\mathbf{v}_t^{\text{ball}}|| \, ||\mathbf{d}_t^{\text{target,ball}}||}) & \text{if} ||\mathbf{v}^{\text{ball}}|| > \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

Agent Approach Ball

The agent approach ball reward is a dense reward that encourages the agent to align its velocity vector with the difference vector from the agent to the ball, while dynamically scaling the reward based on the ball's progress toward the target. This is done by taking the horizontal difference vector from the agent to the ball $\mathbf{d}_t^{\text{agent,ball}}$. To make the reward independent of the magnitude of the agent's speed, the cosine similarity between the two vectors is calculated, which is clipped to 0 or larger to ensure only approaching behaviour is rewarded:

$$\text{CosineSim}^{\text{CLIP}}(\mathbf{d}_t^{\text{agent,ball}}, \mathbf{v}_t^{\text{agent}}) = \max(0, \frac{\mathbf{v}_t^{\text{agent}} \cdot \mathbf{d}_t^{\text{agent,ball}}}{||\mathbf{v}_t^{\text{agent}}|| \, ||\mathbf{d}_t^{\text{agent,ball}}||}).$$

The reward is then calculated by inversely scaling this cosine similarity based on how well the ball is moving toward the target. Specifically, the scaling factor uses the dot product $\mathbf{v}_t^{\text{ball}} \cdot \mathbf{d}_t^{\text{target,ball}}$, clipped to be 0 or larger, ensuring the agent is only rewarded for approaching the ball when the ball is moving (or positioned to move) toward the target. Furthermore, to ensure that the agent only receives a reward signal when it is actively moving closer, the reward is gated by a velocity threshold ε. The reward is then defined as:

$$r_t^{\text{a-approach-b}} = \begin{cases} \frac{\text{CosineSim}^{\text{CLIP}}(\mathbf{d}_t^{\text{agent,ball}}, \mathbf{v}_t^{\text{agent}})}{1+\max(0, \mathbf{v}_t^{\text{ball}} \cdot \mathbf{d}_t^{\text{target,ball}})} & \text{if} \quad ||\mathbf{v}_t^{\text{agent}}|| > \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

Arm Posture

The arm posture penalty is a dense reward term, that penalises the shoulder roll joint for deviating from the robot's default pose. This term calculates the squared error between the current joint position and the robot's default pose, normalised by a sensitivity coefficient σ. The penalty is defined as:

$$r_t^{\text{arm-posture}} = \exp\left(- \frac{(q_{\text{roll}} - q_{\text{roll, def}})^2}{\sigma_{\text{roll}}^2} \right) - 1,$$

where q_{roll} is the current shoulder roll joint position, $q_{\text{roll, def}}$ is the joint position in the default standing pose, and σ_{roll} is the sensitivity coefficient.

Arm Swing

The arm swing penalty is a dense reward term, that penalises symmetric shoulder movement to encourage more natural stabilisation. The penalty is calculated based on the positions of the shoulder pitch joints and utilises the product of the joint angles to detect if the arms are on the same side of the torso's vertical axis. The penalty is only active if both arms are simultaneously pitched to either the front or back ($q_{\text{pitch},L} \cdot q_{\text{pitch},R} > 0$), and a magnitude threshold is exceeded ($|q_{\text{pitch},L} + q_{\text{pitch},R}| > 2.0$). The magnitude threshold creates a safe zone to ensure that small neutral movements do not trigger the penalty. The penalty is then defined as:

$$r_t^{\text{arm-swing}} = \begin{cases} \exp\left(-\frac{1}{(q_{\text{pitch},L}+q_{\text{pitch},R})^2}\right) & \text{if } (q_{\text{pitch},L} \cdot q_{\text{pitch},R} > 0) \wedge |q_{\text{pitch},L} + q_{\text{pitch},R}| > 2.0 \\ 0 & \text{otherwise} \end{cases}$$

3.2.4 Symmetry-Based Data Augmentation

One way to exploit the left–right symmetry in the robot and task is to use it for data augmentation (Mittal et al. 2024). A straightforward approach to symmetry-based data augmentation would be to augment each collected trajectory with its symmetrically transformed copy. For every transition (s_t, a_t, r_t, s_{t+1}) generated by the policy π_θ, an augmented transition $(\rho_{\mathcal{S}}(g)[s_t], \rho_{\mathcal{A}}(g)[a_t], r_t, \rho_{\mathcal{S}}(g)[s_{t+1}])$ would be added to the batch, where $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{A}}$ denote the representations on the state and action spaces respectively.[1] Both original and augmented samples would then be used to update the policy.

Mittal et al. 2024 point out that this approach is fundamentally flawed. The augmented action $\rho_{\mathcal{A}}(g)[a_t]$ was not actually sampled from the policy when observing the augmented state $\rho_{\mathcal{S}}(g)[s_t]$. For instance, if the original sample shows the robot taking a large step with its left foot, the augmented sample shows the robot taking a large step with its right foot. But the current policy may consider this right-foot action highly unlikely given the mirrored state, since perhaps the policy has learned something like an asymmetric gait pattern.

Recall that PPO uses the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ to measure how much the new policy differs from the old policy that collected the data. This ratio enables PPO to reuse samples collected by $\pi_{\theta_{\text{old}}}$ when updating to the new policy π_θ. It can do this using importance sampling and with which it can perform multiple update steps from the same batch of experience. For augmented samples, the naive approach would compute

$$r_t(\theta) = \frac{\pi_\theta(\rho_{\mathcal{A}}(g)[a_t] | \rho_{\mathcal{S}}(g)[s_t])}{\pi_{\theta_{\text{old}}}(\rho_{\mathcal{A}}(g)[a_t] | \rho_{\mathcal{S}}(g)[s_t])}.$$

The denominator $\pi_{\theta_{\text{old}}}(\rho_{\mathcal{A}}(g)[a_t] | \rho_{\mathcal{S}}(g)[s_t])$ can be arbitrarily small for imperfectly symmetric policies, leading to extremely large probability ratios. This causes high variance in the policy gradient and training instability (Mittal et al. 2024).

To resolve this issue, Mittal et al. 2024 proposed a method which corrects for this effect. It is already incorporated in the RSL-RL library (Schwarke et al. 2025). The basic idea is to keep the action probability from the original samples in the denominator of the probability ratio. Rather than evaluating how likely the augmented action is under the old

[1]For notational consistency with the theoretical background, this section uses $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{A}}$ to denote the symmetry transformations on states and actions. The original paper (Mittal et al. 2024) uses the notation L_g and K_g for these operators.

policy given the augmented state, the method uses how likely the original action was under the old policy given the original state. This works because for a symmetric (PO)MDP, the advantage of taking action $\rho_{\mathcal{A}}(g)[a]$ in state $\rho_{\mathcal{S}}(g)[s]$ equals the advantage of taking action a in state s: $A^{\pi}(\rho_{\mathcal{S}}(g)[s], \rho_{\mathcal{A}}(g)[a]) = A^{\pi}(s, a)$. The augmented samples can thus be treated as if they provide information about equivalent state-action pairs without needing to evaluate the problematic action probabilities.

In practice, for each batch of experience collected during training, symmetric copies are created by applying the left–right transformation to all samples. Both original and augmented samples are used to compute the policy update, effectively doubling the training data while encouraging the policy to exhibit symmetric behaviour for equivalent goals. The authors claim that this approach enables faster convergence and more consistent performance across symmetric task configurations without the instability of naive data augmentation (Mittal et al. 2024).

3.3 Equivariant Neural Networks

Equivariant neural networks can encode the symmetries inherent to a task in the weights of the neural network (Bronstein et al. 2021; M. Weiler et al. 2023). To find the exact parameterisation for such networks, the representations with respect to which the layers should be equivariant have to be made explicit. The actor and critic model are similar with respect to their symmetric structure, so both are explained at the same time. Contrary to the symmetry-based data augmentation, equivariant neural networks are not yet incorporated in the RSL-RL library (Schwarke et al. 2025). An extension has been written for the purpose of being able to run the experiments for this thesis.

3.3.1 Representations for the Input, Hidden, and Output spaces

Let the input space be written as a direct sum of N_{in} fields,

$$X = \bigoplus_{j=1}^{N_{\text{in}}} X_j,$$

where each field X_j transforms under a representation

$$\rho_j^{\text{in}} \in \{\rho_{\text{triv}}, \rho_{\text{sign}}, \rho_{\text{reg}}, \rho_{\text{reg+sign}}\},$$

for both the critic and the actor. However, the input of the critic is a superset of the actor and thus has some extra fields and representations to consider. The exact ordering of fields, their dimensionalities, representations, and their semantic interpretation can be found in table 3.4.

For spatial vectors defined in the robot's local coordinate system (where the forward, lateral, and vertical axes correspond to the robot's body frame), the S_2 left–right symmetry is a reflection that swaps the left and right sides of the robot. The feature fields representing positions or velocities in this frame therefore transform as the direct sum $\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$. This reflects that while the forward and vertical components remain unchanged under a left–right flip, the lateral component changes sign. Angular velocities, however, transform under $\rho_{\text{sign}} \oplus \rho_{\text{triv}} \oplus \rho_{\text{sign}}$ since follow different conventions. This point is explained in appendix C.

The joint-related observations (and actions) all share a consistent 22-dimensional structure as detailed in the breakdown of table 3.4. These features consist of scalar values for joints located on the robot's central axis and pairs of values for the bilateral limbs. For the head pitch, the symmetry operation is trivial as the joint axis remains unaffected by a reflection. However, for the head yaw, the representation is ρ_{sign}. This reflects that a rotation to the left in the original frame must be negated to represent the corresponding rotation in the reflected frame, as the positive direction of the yaw axis is inverted under the symmetry transformation.

For the limbs, the representation depends on whether the joint axes themselves are mirrored in the robot's hardware or coordinate convention. In the case of pitch joints, such as the left and right knees, the symmetry transformation follows the regular representation ρ_{reg}. For these pairs, a reflection simply swaps the value of the left joint with that of the right joint. For example, if the left knee is at 0.5 radians and the right is at 0.2 radians, the symmetric configuration simply places the left at 0.2 and the right at 0.5.

On the other hand, for joints like the hip roll or shoulder yaw, the sign of the values also need to be changed. In other words, here the regular plus sign-flip representation $\rho_{\text{reg+sign}}$ applies. In this case, to obtain the symmetric state, one must swap the left and right values and additionally invert their signs. Taking the shoulder yaw as an example, if a positive value indicates an outward rotation, a reflection requires that the left and right values are exchanged and negated to ensure the physical meaning of the rotation is preserved in the mirrored state. This logic applies uniformly across joint positions, joint velocities, and the previous action, ensuring the network perceives and commands bilateral movements consistently.

As is also described in that table, the outputs for both models are different. For the actor it is again a direct sum of N_{out} fields,

$$Y = \bigoplus_{i=1}^{N_{\text{out}}} Y_i,$$

where each Y_i transforms under a representation

$$\rho_i^{\text{out}} \in \{\rho_{\text{triv}}, \rho_{\text{sign}}, \rho_{\text{reg}}, \rho_{\text{reg+sign}}\}.$$

For the critic, the output space is one dimensional and transforms under the trivial representation,

$$\rho^{\text{out}} = \rho_{\text{triv}}.$$

In other words, the score (output) of the critic is invariant under the representations of S_2 on its input, whereas the actor is strictly equivariant.

All hidden layers for both the actor and the critic transform under the regular representation of S_2. A hidden layer with K regular fields has representation

$$H = \bigoplus_{k=1}^{K} \rho_{\text{reg}},$$

and total dimension $2K$ (since each regular field of S_2 is 2-dimensional, contributing 2 scalar values). This architectural choice lifts all information into a representation where left–right symmetric and antisymmetric components can be mixed equivariantly across layers.

3.3.2 Equivariant Actor and Critic Models

These representations types for the input, hidden and output states directly determine which equivariances the layers connecting them should satisfy. For any linear layer L, the requirement $f(\rho_X(g)x) = \rho_Y(g)f(x)$ implies that the weight matrix must commute with the group actions of the input and output spaces. Because these spaces are constructed as direct sums of different feature fields, the weight matrix L is partitioned into a block structure. Each sub-block L_{ij} corresponds to the mapping from the j-th input field to the i-th output field and must independently satisfy the constraint $L_{ij}\rho_j^{\text{in}}(g) = \rho_i^{\text{out}}(g)L_{ij}$.

The equivariant models are thus constructed as a composition of these block-structured mappings and pointwise activations:

$$L^{(\text{out})} \circ \text{ReLU} \circ L^{(L)} \circ \cdots \circ \text{ReLU} \circ L^{(1)}.$$

The first layer $L^{(1)}$ maps from the heterogeneous input representations to the first hidden representation. All hidden layers for both the actor and the critic transform under the regular representation ρ_{reg}. Consequently, $L^{(1)}$ is composed of equivariant blocks that map from ρ_{triv}, ρ_{sign}, ρ_{reg}, and $\rho_{\text{reg+sign}}$ to ρ_{reg}. For example, the block corresponding to the head yaw (ρ_{sign}) uses the anti-symmetric solution $\begin{pmatrix} w \\ -w \end{pmatrix}$, while a pitch joint ($\rho_{\text{reg}}$) uses a 2×2 circulant block.

The internal hidden layers $L^{(L)}$ all map from and to spaces transforming under ρ_{reg}. These layers consist entirely of 2×2 circulant blocks, which ensures that information can be mixed between symmetric features while remaining equivariant. Following each linear layer, a pointwise ReLU activation is applied. As noted, this activation commutes with the permutation of the regular representation, meaning the non-linearities do not break the symmetry of the hidden states.

Finally, $L^{(\mathrm{out})}$ maps from the regular hidden representation to the output representation. For the actor, this layer is built from the specific blocks required to produce the 22 joint targets in their respective representations. For the critic, however, $L^{(\mathrm{out})}$ consists of blocks mapping from ρ_{reg} to ρ_{triv}. These blocks take the form $\begin{pmatrix} w & w \end{pmatrix}$, which sums the symmetric components of the hidden state. This mapping ensures that the final output of the critic is invariant, while the actor remains strictly equivariant. By transitivity, the composition of these layers forms a model that preserves the S_2 symmetry across the entire architecture.

Table 3.4: Consolidated Input and Output Representations for the Equivariant Actor and Equivariant Critic. A checkmark (✓) indicates which network uses a given field.

Field Name	**Component/Pair**	**Rep. Type**	**Dim.**	**Actor**	**Critic**
Network Inputs (Observations)					
Base Linear Velocity	(Forward, Lateral, Vertical)	$\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$	3	✓	✓
Base Angular Velocity	(Roll, Pitch, Yaw)	$\rho_{\text{sign}} \oplus \rho_{\text{triv}} \oplus \rho_{\text{sign}}$	3	✓	✓
Projected Gravity	(Forward, Lateral, Vertical)	$\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$	3	✓	✓
Ball Relative Position	(Forward, Lateral, Vertical)	$\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$	3	✓	✓
Target Position	(Forward, Lateral, Vertical)	$\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$	3	✓	✓
The following three 22-dimensional fields all share the representation structure detailed below:					
Joint Positions	(22 joints)	Mixed	22	✓	✓
Joint Velocities	(22 joints)	Mixed	22	✓	✓
Previous Action	(22 joints)	Mixed	22	✓	✓
— Breakdown for Joint-space Representation —					
	Head Yaw	Sign (ρ_{sign})	1	✓	✓
	Head Pitch	Trivial (ρ_{triv})	1	✓	✓
	Left/Right Shoulder Pitch	Regular (ρ_{reg})	2	✓	✓
	Left/Right Elbow Pitch	Regular (ρ_{reg})	2	✓	✓
	Left/Right Hip Pitch	Regular (ρ_{reg})	2	✓	✓
	Left/Right Knee Pitch	Regular (ρ_{reg})	2	✓	✓
	Left/Right Ankle Pitch	Regular (ρ_{reg})	2	✓	✓
	Left/Right Shoulder Roll	Regular + Sign Flip	2	✓	✓
	Left/Right Elbow Yaw	Regular + Sign Flip	2	✓	✓
	Left/Right Hip Roll	Regular + Sign Flip	2	✓	✓
	Left/Right Hip Yaw	Regular + Sign Flip	2	✓	✓
	Left/Right Ankle Roll	Regular + Sign Flip	2	✓	✓
— End of Breakdown —					
Critic-Specific Observation Fields					
Foot Height	(left, right)	Regular (ρ_{reg})	2		✓
Foot Air Time	(left, right)	Regular (ρ_{reg})	2		✓
Foot Contact State	(left, right)	Regular (ρ_{reg})	2		✓
Foot Contact Forces	(all 6 components)	Trivial (ρ_{triv})	6		✓
Ball Velocity	(Forward, Lateral, Vertical)	$\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$	3		✓
Ball-Foot Contact	(left, right)	Regular (ρ_{reg})	2		✓
Network Outputs					
Actor: Joint Position Targets	(all 22 joints)	As above	22	✓	
Critic: State Value	Single scalar value	Trivial (ρ_{triv})	1		✓

Chapter 4

Experiments and Results

4.1 Experimental Setup

The development of a task environment is itself a process of trial and error. Therefore, the task environment is itself a result of this thesis. As noted by Sutton, Barto, et al. 1998, representational design is often more art than science. Given the heuristic and iterative nature of development, it has too many stages to present comprehensively here. Instead, comments and observations on this will be part of the text motivating the final environment setup.

4.1.1 Task Definition

This thesis investigates a robotic task requiring the precise kicking of a ball towards a designated target. The primary objective is to maximise targeting accuracy, defined by the ball's final resting position relative to the target center at the end of an episode. To ensure the behaviour is both physically viable and consistent with the intended task, the movement must remain dynamically stable and have the visual characteristics of a kick motion. Following the kick, the robot is required to stabilise its posture and return to a stable, standing state.

4.1.2 Terminal States

To ensure efficient training, four early terminal states were included in the final setup: time out, double touch, falling over, and falling down. These early terminal states serve to prevent indefinite episodes caused by inactivity (e.g. the agent is standing still) and to minimise time spent in unrecoverable states (e.g. the robot is lying on the ground). Additionally, to enforce a kicking motion rather than a dribbling strategy, a double touch termination was included. Empirical observations indicated a tendency for the agent to converge on a suboptimal strategy of shuffling or dribbling the ball to the target. Terminating the episode after the ball is touched a second time (with some margin) by the robot prevents this behaviour, thereby accelerating training by removing this undesirable stage from the training process.

4.1.3 Reward

A minimalist and principled design was prioritised for the reward function. Increasing the number of reward components complicates the balancing and tuning of the weights and introduces the risk of conflicting signals, thereby increasing the risk of the policy converging to undesirable local optima.

Sparse vs. Dense Rewards

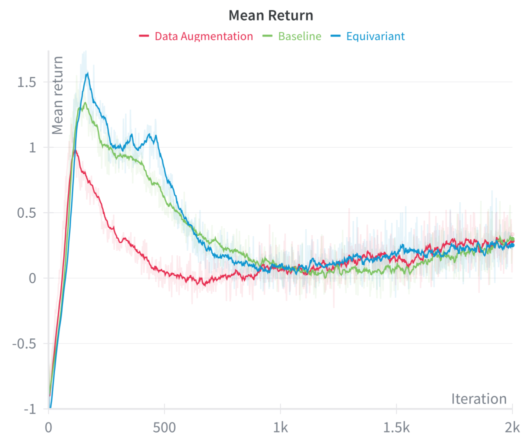
While targeting accuracy is the primary objective, relying solely on a sparse reward makes it very difficult for the agent to learn the task objective. To address this problem, in addition to a high sparse reward (Target-Ball Distance) at the end of an episode, two dense rewards were introduced: Agent Approach Ball and Ball Approach Target. However, initial experiments showed a tendency for the agent to get stuck at a local optimum where the robot would rock back and forth near the ball without moving it. To address this problem, a low penalty was included to incentivise the agent to move the ball (Ball Not Moving Penalty).

Reward Misalignment

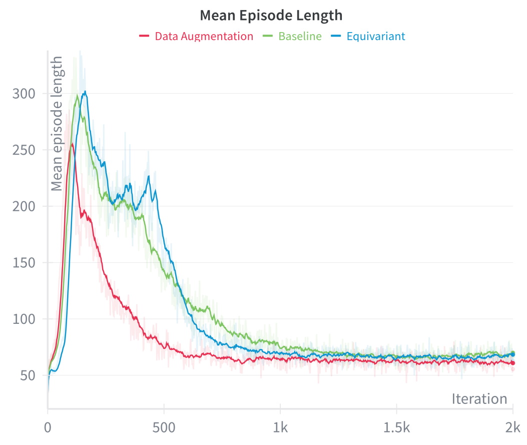
Initial experiments indicated that excessive positive dense rewards, such as Upright and Survival Reward rewards could led the agent to converge to a passive local optimum, where the robot would stand still to maximise these rewards as well as avoid most penalties. Since dense rewards are provided at every timestep, they scale with the duration of the episode. This can create an incentive for the agent to exploit rewards by maximising the episode length.

In certain configurations where reward weights were imbalanced, the theoretically optimal strategy was to stand still for the entire episode duration. As shown in **??**, the mean cumulative reward closely tracks the mean episode length. Simulation results verified that standing still until the time-out limit was reached, accumulated high episodic rewards. However, because the discount factor was set to 0.99, the agent's effective horizon was limited. Consequently, the agent did not know that standing still for the entire episode would eventually yield the highest total reward. Instead, the policy diverged toward a strategy that provided a higher short-term return: kicking the ball to the target.

While the limited time horizon of the agent led to the desired behaviour, this setup is fragile. For a reward function to be robust, it should ensure that the global optimum



(a)



(b)

Figure 4.1: Mean episodic reward (a) and episode length in timesteps (b) during training with a misaligned reward function.

aligns with the primary task objective, rather than rely on training hyperparameter tuning to prevent undesirable convergence.

The Agent Approach Ball reward requires the ball to not have been moved yet. In early variants where this condition was not included, the robot would walk after the ball (not dribble) after it had been kicked to maximise this reward. Therefore, the reward was zeroed out after the ball contact condition was satisfied.

Balancing Regularisation and Exploration

Regularisation penalties are necessary for smooth and stable motion. However, early exploration often leads to erratic motions that trigger these penalties during the initial stages of training. When these penalties are not sufficiently offset by dense positive rewards, the agent frequently receives a negative cumulative return. This leads to a problem during training where the agent can easily discover a local optimum where the agent terminates the episode as quickly as possible (e.g. by falling over or falling down immediately). By terminating early the accumulation of negative rewards can be avoided.

To counter this behaviour, the reward for early termination must be lower than the potential return from attempting the task. Consequently, two heavy penalties were introduced: Fall Down and Fell Over. By ensuring that early termination results in a significantly lower reward than continuing the episode, it is not a local optimum any more.

Aesthetic Considerations

During initial training, the agent frequently maintained balance by extending its arms laterally or raising them excessively (see fig. 4.2a). While effective for stabilisation, this resulted in unnatural posture during the kick and subsequent standing phase. To ensure the motion remains visually consistent with a human-like kick, two stylistic reward terms were introduced: Arm Posture penalises the robot for spreading its arms wide, while Arm Swing penalises symmetric shoulder movement while allowing for reciprocal swinging. The latter penalty resulted in reciprocal arm swinging, without explicitly rewarding this behaviour, as can be seen in fig. 4.2b. It seems plausible that this is also better for reducing collisions or getting entangled with other robots.

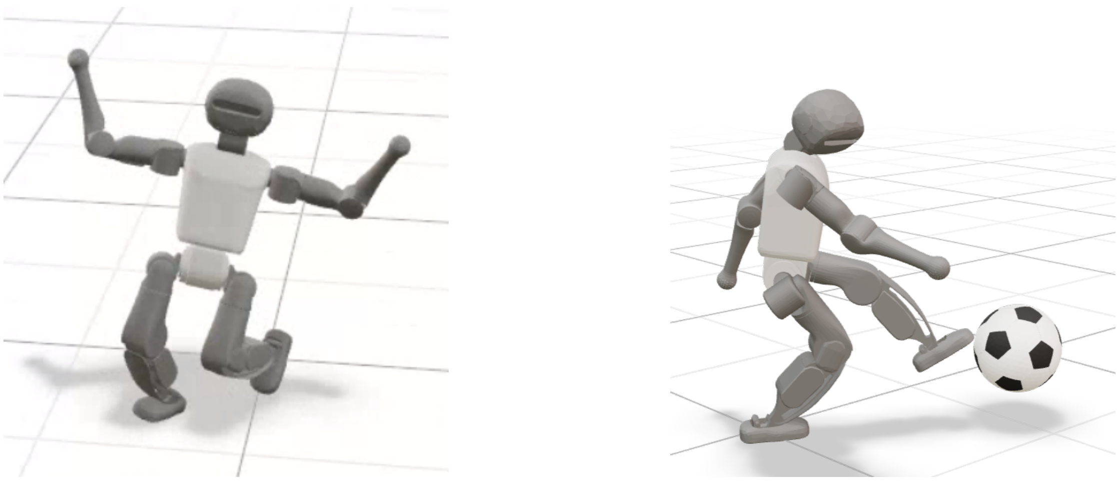

(a) (b)

Figure 4.2: (a) demonstrates posture before introducing the arm posture and arm swing penalties while (b) demonstrates posture after introducing these penalties.

4.2 Experiments

The following experiments were designed to evaluate the agent's ability to learn the complex task of approaching and kicking a ball, while simultaneously evaluating how the incorporation of knowledge about symmetry impacts learning speed and generalisation. Across both experiments two different approaches of incorporating symmetry, one utilising data augmentation during training and one that makes use of an equivariant neural network, were evaluated against a vanilla PPO baseline. Both experiments used the same reward configuration, which can be found in table 4.1. They also use the same training hyperparameters (see table B.1), as well as the same environment and reward hyperparameters (see table B.2).

The first experiment is a general performance experiment to assess task learnability and performance across the three different approaches. This was followed by a Home Field Advantage experiment, which investigated whether the robot could successfully generalise its kicking ability to the right side when its training was restricted exclusively to the left side.

Reward Term	**Weight**
Task Objectives	
Target distance	250.
Ball approach target	0.3
Agent approach ball	0.1
Ball not moving	-0.01
Survival and Stability	
Survival	0.01
Upright	0.05
Fall over	-50.
Fall down	-50.
Regularization and Style	
Joint positions exceed limit	-1.
High-frequency actions	-0.01
Feet sliding	-0.01
Arm swing	-0.05
Arm posture	0.3

Table 4.1: Experiment reward weights grouped by objective type

4.2.1 General

Two variants of the PPO algorithm, one utilising data augmentation and another employing an equivariant neural network, were evaluated against a vanilla PPO baseline. Each approach was trained using three independent seeds and for 15,000 training iterations. During the training phase, target positions were sampled from a radial distance in the range $[2.5, 8.5]$ and an angle in the range $[-1, 1]$ radians ($\approx \pm 57°$), as depicted in fig. 4.3.

To evaluate the performance of the trained policies, the final checkpoints were used to run simulations/episodes whose initial conditions were sampled from the same distribution as was used during training. The terminal distance between the ball and the target was tracked as a measure of accuracy and task performance. Furthermore, a success threshold of 1.0m was established to provide a different level of granularity in

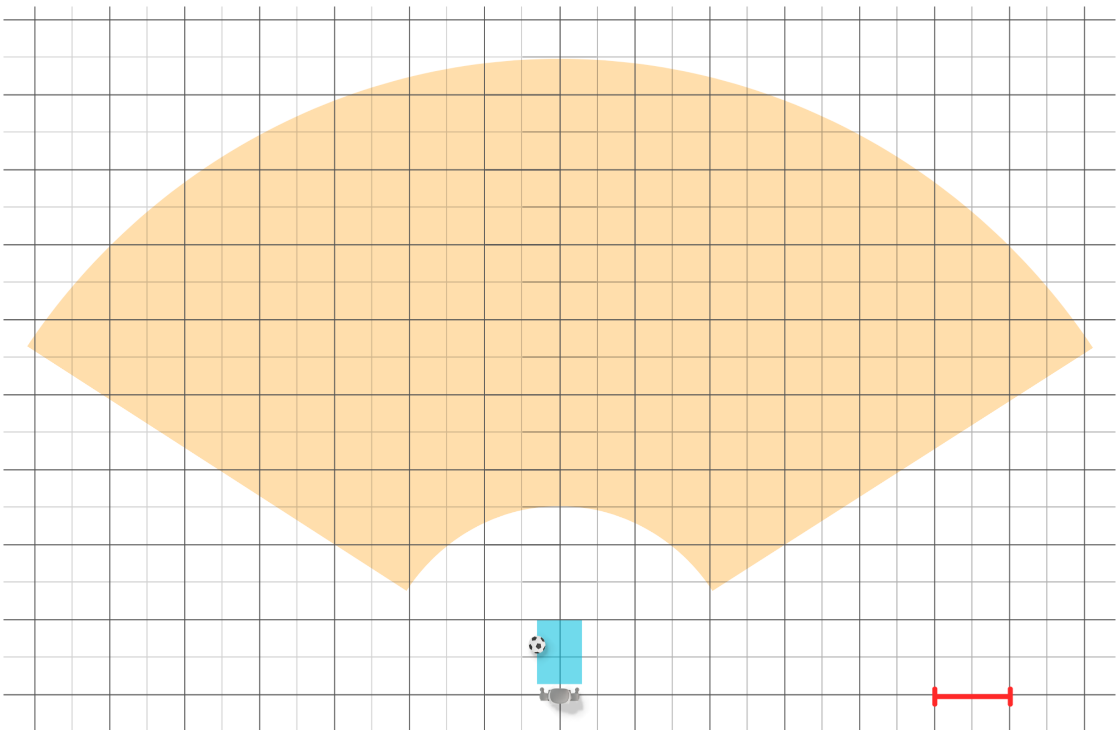


Figure 4.3: Target (yellow) and ball (blue) sampling zones for the general performance experiment. The red line on the bottom right indicates 1 meter.

performance assessment. This threshold is somewhat arbitrary and based on a rough estimate for the margin that another robot can be expected to cover at this scale to correct for an imperfect pass. Evaluation was done for 10,000 episode simulations per model per seed.

4.2.2 Home Field Advantage

To investigate the impact of incorporating symmetry knowledge on generalisation, an experiment was conducted where the training distribution was strongly biased towards a specific part of the state space. Specifically, the initial positions of the ball and target were restricted to one side of the robot's sagittal plane. Target positions were sampled from a radial distance in the range $[2.5, 8.5]$ and an angle in range $[0, \pi]$ radians, while the initial ball position was sampled from $x \in [0.15, 1.0]$ and $y \in [0., 0.5]$, as depicted by the light green and dark green areas in fig. 4.4. This was trained with one seed for each approach (Baseline, Data Augmentation and Equivariant) for 12,000 training iterations.

The models were subsequently evaluated on scenarios both within the original training distribution and on its reflection across the robot's sagittal plane. The reflected distribution consisted of samples from a radial distance in range $[2.5, 8.5]$ and an angle in range

$[-\pi, 0]$ radians, with an initial ball position sampled from $x \in [0.15, 1.0], y \in [-0.5, 0.]$, as depicted by the light red and dark red area in fig. 4.4. Again, both the final distance of the ball to the target and whether it met the 1.0m threshold were tracked.

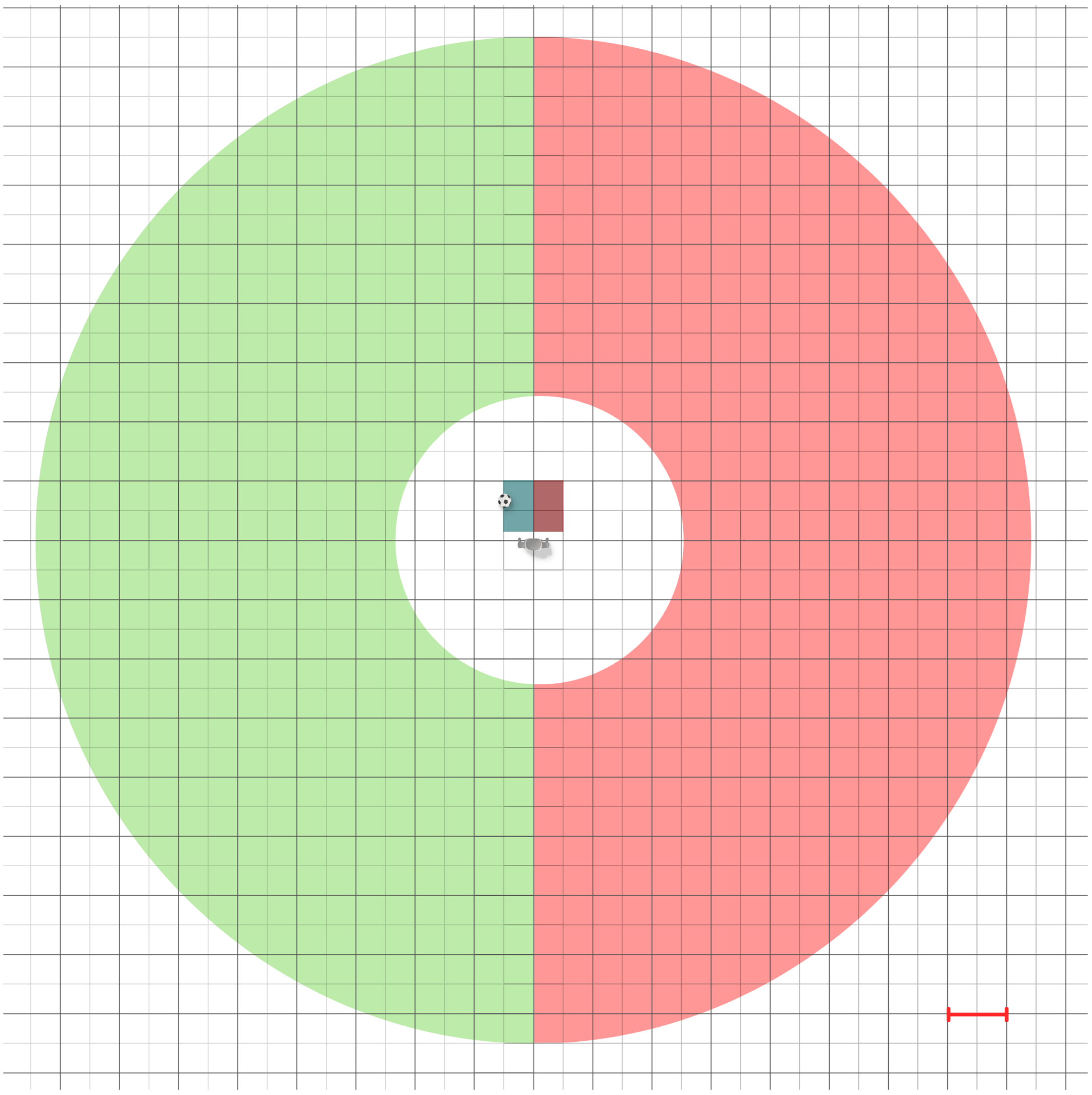
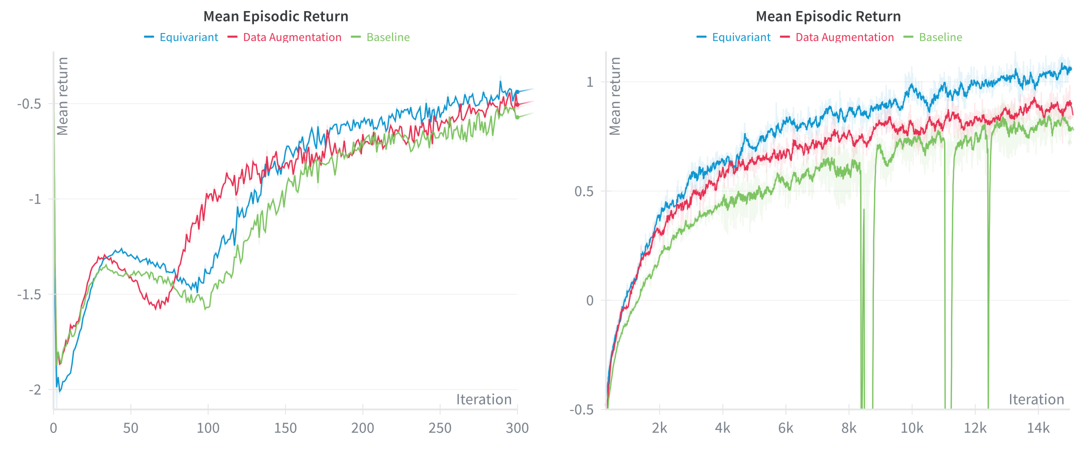


Figure 4.4: Target (light) and ball (dark) sampling zones for the Home Field Advantage experiment. Green regions indicate the training distribution, while red regions indicate the reflected evaluation distribution. The red line on the bottom right indicates 1m.

Evaluation was done for 60,000 simulations per model: 30,000 for in distribution scenarios and 30,000 for out-of-distribution scenarios. As the primary goal was to observe the structural properties of the networks rather than peak performance, training until full convergence was not necessary to evaluate the generalisation gap.

4.3 Results

4.3.1 General



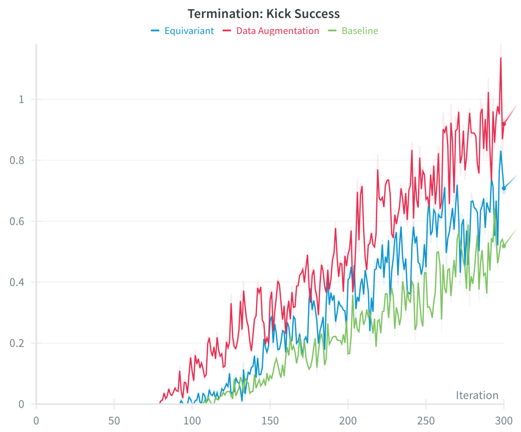
(a)

(b)

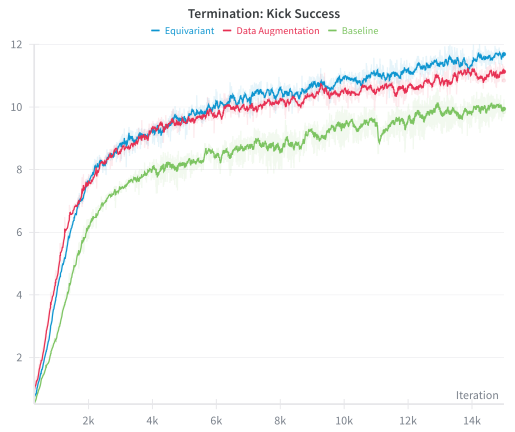
Figure 4.5: Mean episodic return across three independent training runs. Note that these two subplots are of the same runs. Since these plots are used to highlight patterns at different scales, they are split at training iteration 300 so the initial learning phases are legible. The y-axis is truncated for readability.

The training curves in fig. 4.5a indicate that all three approaches initially yield low mean episodic rewards. This is primarily a result of heavy penalties assigned to falling over and falling down events. This observation is supported by the termination data, as these terminal states peak at the beginning of training and decline rapidly as the agents achieves stability (see fig. 4.7). As these early terminations decline, a corresponding increase in task objective terminations is observed (see fig. 4.6c, fig. 4.6a and fig. 4.6e), signifying the agent transitioning from maintaining balance to successfully directing the ball toward the target. All three approaches go through these same training phases.

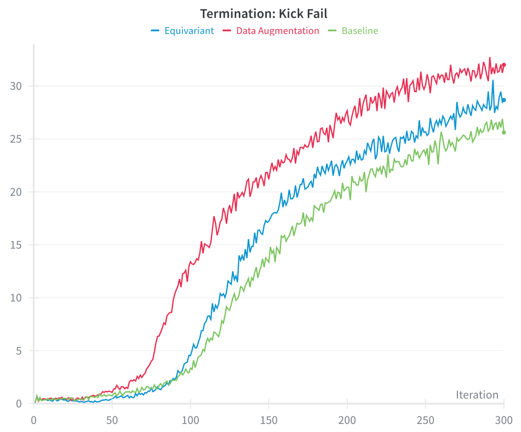
Significant instability was observed in the baseline, where troughs in mean episodic reward can be seen in fig. 4.5b between iteration 8,000 and 13,000. These performance drops occurred in two out of three seeds (see appendix D for individual plots per seed). In contrast, the two symmetry variants maintained higher and more stable mean episodic returns throughout training. Specifically, the equivariant approach slightly outperformed data augmentation. Furthermore, qualitative analysis shows that the baseline breaks symmetry as training progresses, while both the equivariant and the data augmentation approach successfully maintain symmetric properties throughout the entire training duration. Notably, the baseline learns to have a strongly preferred leg in all three seeds.



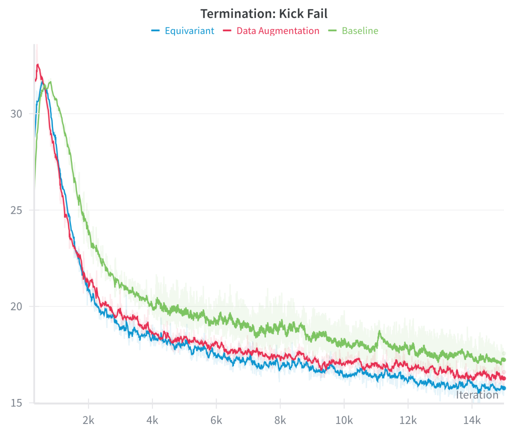
(a)



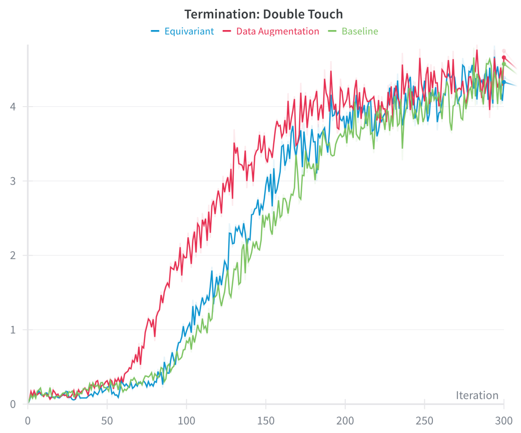
(b)



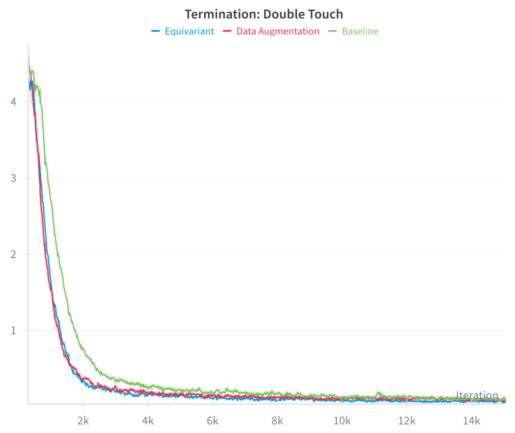
(c)



(d)

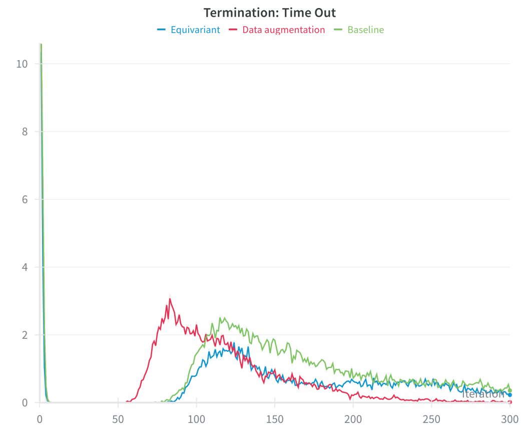


(e)

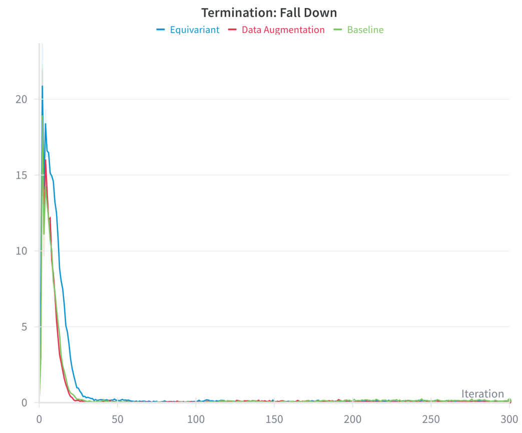


(f)

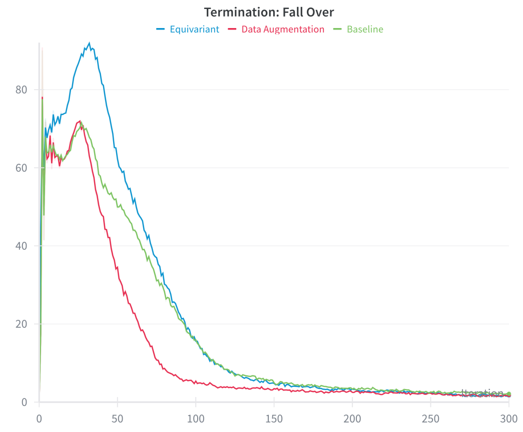
Figure 4.6: Kick success (a,b), kick fail (c,d) and double touch (d,e) terminations throughout training. Note that these left–right pairs of subplots are of the same runs. Since these plots are used to highlight patterns at different scales, these plots are split at training iteration 300 so the initial learning phases are legible.



(a)



(b)



(c)

Figure 4.7: Time out (a), fall down (b) and fall over (c) terminations during the first 300 training iterations. They stay close to zero after that point.

For all these final checkpoints for the baseline approach, the robot will go out of its way to kick the ball with its preferred leg even if intuitively this seems suboptimal.

Model	**Mean Distance to Target (m)**	**Success Rate (%)**
Baseline	0.899 ± 0.092	66.36 ± 5.82
Data Augmentation	0.885 ± 0.112	65.11 ± 11.60
Equivariant	**0.806** ± 0.097	**71.95** ± 7.91

Table 4.2: Performance comparison across different models. Metrics represent the mean ± standard deviation calculated across three independent seeds (10,000 episodes per seed).

Looking at the evaluation metrics, the first thing to note is that the baseline scores very similar to the data augmentation approach. On the other hand, the equivariant approach scores best on all metrics. In regards to the training, the equivariant also performs the best, with the exception of data augmentation starting to learn to kick the ball slightly earlier. The equivariant model however quickly catches up and stays ahead. A singular extra long run corroborates that this pattern continues (see fig. D.1). Furthermore, this holds for all three seeds (see fig. D.2).

4.3.2 Home Field Advantage

In the Home Field Advantage experiment, the baseline is unable to generalise to the scenarios outside the training distribution. The equivariant model and the model trained with data augmentation both show close to no difference in performance between scenarios in and out-of-distribution. While the models were not trained until convergence, it can be observed that at this point in training the equivariant model overall outperforms the data augmentation variant. All approaches perform approximately equally well on the in distribution validation test with the baseline being the best by a thin margin.

	In Distribution		**Out-of-Distribution**	
Model	**Mean Dist. (m)**	**Success (%)**	**Mean Dist. (m)**	**Success (%)**
Baseline	1.011	55.58	3.416	15.28
Data Augmentation	1.1365	49.76	1.1534	49.72
Equivariant	1.0118	55.11	1.0165	54.06

Table 4.3: Evaluation results for the Home Field Advantage experiment, comparing performance on initial states within the training distribution and against initial states outside of the training distribution.

Chapter 5

Conclusion

This thesis investigated whether incorporating symmetry knowledge into the learning process can improve training efficiency and generalisation in robotic motor control tasks. The results provide preliminary but substantial evidence that incorporating symmetry does indeed improve performance and generalisation. The least surprising result came from the test biased towards their strengths (the Home Field Advantage experiment), in which the symmetry-based approaches predictably generalised to the out-of-distribution setting, while the baseline failed to do so. That both symmetry-based approaches outperformed the baseline in the general setting was a less obvious result. The equivariant approach performed particularly well, achieving the best results across all metrics and across all seeds. Furthermore, this pattern held across all three seeds, in the extended training run, and showed up during the trial-and-error runs for finding the right reward functions and their weighting.

Both symmetry-aware approaches outperformed the baseline, which can plausibly be explained by their incorporation of a valid inductive bias. The baseline treats kicking with the left foot and kicking with the right foot as independent problems to solve. The symmetry-aware approaches learn one solution and obtain the other through the symmetry constraint. This reduces the effective size of the hypothesis space, which should make the optimisation problem easier and improve generalisation. The baseline's development of a preferred leg in all three seeds could provide some evidence for this interpretation. During early exploration, one leg likely succeeds before the other by chance, and the policy then reinforces this pattern. Over time, this leads to specialisation around a contingent asymmetry that has nothing intrinsically to do with the actual task. The symmetry-aware approaches structurally prevent this failure mode. It is a very interesting question whether avoiding such path-dependent local optima could fully explain the performance gap with the baseline. However, the evidence collected in this thesis only points towards the discrepancy, but does not provide clear evidence for an explanation.

Similarly, understanding why exactly the equivariant approach outperforms data augmentation remains an open question. That it does outperform is itself an interesting empirical finding, since one might have expected both methods to converge to similar solutions given sufficient training or at least to have done so faster than could be observed in the experiments. One speculative explanation points to the difference in how each method enforces symmetry: equivariant networks restrict the parameter space directly through their architecture, while data augmentation steers the optimisation towards symmetric solutions through exposure to mirrored samples. In other words, the former solves analytically what the latter tries to solve numerically. This means that the equivariant approach integrates the prior knowledge of the symmetry before training, while data augmentation does so during training. While the experiments confirm that the equivariant approach achieves better performance in this setting, they do not provide clear, substantial evidence whether it does so for such theoretically satisfying reasons.

5.1 Future Work

Longer training runs would provide stronger evidence about asymptotic performance and convergence behaviour. More extensive hyperparameter sweeps and ablation studies could also clarify which aspects of the symmetry-aware approaches most influence for their success and how universal these patterns are across different tasks, domains, hyperparameter settings, and reward functions. It is furthermore not immediately clear how equivariance would work in combination with other successful techniques in robotics RL like motion priors such as Adversarial Motion Priors (AMP) (Peng et al. 2021). Whether those already capture a part of the gain of equivariance or whether such combination would form a synergy could also help understand the mechanism for the improvement better.

The simulation environment used for the experiments in this thesis was rather sterile and static. Learning and generalising in such a setting is even with some noisy domain randomisation relatively easy and perhaps should discount the conclusions somewhat. Testing whether the generalisation benefits hold in more challenging settings that still involve relevant symmetries would strengthen the evidence. Such settings could include environments with obstacles, uneven terrain, dynamic elements, or scenarios requiring the agent to process raw visual input rather than pre-computed observations. Extending to multi-agent scenarios with passing and receiving would test whether symmetry-aware methods maintain their advantages when agent-agent interactions introduce additional complexity.

If these performance and generalisation results hold for such a broader set of settings, one could even hypothesise that this may translate into a smoother and more reliable sim-to-real. However, one special concern here is that real-world deployment also introduces asymmetries that do not naturally come up in simulation. Manufacturing tolerances, mechanical wear, sensor biases, and overheating joints can all break perfect bilateral symmetry. This does not need to sour the promise of equivariance. One potential approach to explore would be to add learnable asymmetric corrections on top of the symmetric base policy. The equivariant base provides strong generalisation through its perfectly symmetric structure, while the asymmetric correction layer adapts to the idiosyncratic features of physical hardware. This would allow the equivariant model to descend gracefully from the ideal world of perfect symmetries into the imperfect reality of actual robots: a symbiosis of symmetry and asymmetry.

Bibliography

Abdolhosseini, Farzad et al. (2019). "On learning symmetric locomotion". In: *Proceedings of the 12th ACM SIGGRAPH Conference on Motion, Interaction and Games*, pp. 1–10.

Atanassov, Vassil et al. (2024). "Curriculum-based reinforcement learning for quadrupedal jumping: A reference-free design". In: *IEEE Robotics & Automation Magazine*.

Bronstein, Michael M et al. (2021). "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges". In: *arXiv preprint arXiv:2104.13478*.

Cesa, Gabriele, Leon Lang, and Maurice Weiler (2022). "A program to build E (N)-equivariant steerable CNNs". In: *International conference on learning representations*.

Hu, Muqun et al. (2025). "Towards Versatile Humanoid Table Tennis: Unified Reinforcement Learning with Prediction Augmentation". In: *arXiv preprint arXiv:2509.21690*.

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2, pp. 99–134.

Kalakrishnan, Mrinal et al. (2010). "Fast, robust quadruped locomotion over challenging terrain". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE, pp. 2665–2670.

Kitano, Hiroaki et al. (1997). "RoboCup: A challenge problem for AI". In: *AI magazine* 18.1, pp. 73–73.

Lang, Leon and Maurice Weiler (2021). "A Wigner-Eckart Theorem for Group Equivariant Convolution Kernels". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: `https://openreview.net/forum?id=ajOrOhQOsYx`.

Lee, Joonho et al. (2020). "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 5.47, eabc5986.

Mittal, Mayank et al. (2024). "Symmetry considerations for learning task symmetric robot policies". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7433–7439.

Nguyen, Hai Huu et al. (2023). "Equivariant reinforcement learning under partial observability". In: *Conference on Robot Learning*. PMLR, pp. 3309–3320.

Nie, Buqing et al. (2025). "Coordinated Humanoid Robot Locomotion with Symmetry Equivariant Reinforcement Learning Policy". In: *arXiv preprint arXiv:2508.01247*.

Peng, Xue Bin et al. (2021). “Amp: Adversarial motion priors for stylized physics-based character control”. In: *ACM Transactions on Graphics (ToG)* 40.4, pp. 1–20.

Schulman, John et al. (2017). “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347*.

Schwarke, Clemens et al. (2025). “RSL-RL: A Learning Library for Robotics Research”. In: *arXiv preprint arXiv:2509.10771*.

Simon, Barry (1996). *Representations of finite and compact groups*. 10. American Mathematical Soc.

Spitznagel, Martin, David Weiler, and Klaus Dorer (2021). “Deep reinforcement multi-directional kick-learning of a simulated robot with toes”. In: *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, pp. 104–110.

Su, Zhi et al. (2024). “Leveraging symmetry in rl-based legged locomotion control”. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6899–6906.

Sutton, Richard S, Andrew G Barto, et al. (1998). *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge.

Van der Pol, Elise et al. (2020). “Mdp homomorphic networks: Group symmetries in reinforcement learning”. In: *Advances in Neural Information Processing Systems* 33, pp. 4199–4210.

Wang, Dian, Robin Walters, and Robert Platt (2022). “SO(2)-Equivariant Reinforcement Learning”. In: *arXiv preprint arXiv:2203.04439*.

Wang, Yushi, Penghui Chen, et al. (2025). “Booster Gym: An End-to-End Reinforcement Learning Framework for Humanoid Robot Locomotion”. In: *arXiv preprint arXiv:2506.15132*.

Wang, Yushi, Changsheng Luo, et al. (2025). “Learning Vision-Driven Reactive Soccer Skills for Humanoid Robots”. In: *arXiv preprint arXiv:2511.03996*.

Wang, Zhuoheng, Jinyin Zhou, and Qi Wu (2025). “Dribble Master: Learning Agile Humanoid Dribbling Through Legged Locomotion”. In: *arXiv preprint arXiv:2505.12679*.

Weiler, Maurice et al. (2023). *Equivariant and coordinate independent convolutional networks: A gauge field theory of neural networks*. `https://maurice-weiler.gitlab.io/cnn_book/EquivariantAndCoordinateIndependentCNNs.pdf`.

Zakka, Kevin et al. (Sept. 2025). *MJLab: Isaac Lab API, powered by MuJoCo-Warp, for RL and robotics research.* Version 0.1.0. URL: `https://github.com/mujocolab/mjlab`.

Appendix A

Remaining Reward Descriptions

Upright

The upright reward is a dense reward that encourages the robot to maintain an upright posture. It measures the deviation of the robot's projected gravity vector from the vertical axis and rewards values close to zero (perfectly upright).

$$r_t^{\text{upright}} = \exp\left(-\frac{\theta_t^2}{\sigma_{\text{upright}}^2} \right),$$

where θ_t is the deviation of robot's base z-axis from the global z-axis, and σ is a standard deviation parameter.

Survival Reward

The survival reward is a dense reward, given at every environment step as long as the episode continues.

Ball Not Moving Penalty

The ball not moving penalty, is a dense penalty that discourages the agent from letting the ball lay still.

$$r_t^{\text{ball-stagnant}} = \begin{cases} 1 & \text{if} \quad \|\mathbf{v}_t^{\text{ball}}\| < \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Fall Down

The fall down penalty is a penalty, that discourages the robot's trunk height falling below a specified minimum threshold.

$$r_t^{\text{fall-down}} = \begin{cases} 1 & \text{if} \quad z_t^{\text{trunk}} < z_{\text{falldown}} \\ 0 & \text{otherwise} \end{cases}$$

Fell Over

The fell over penalty is a penalty, that discourages robot's body orientation deviating beyond a certain angle from the global vertical axis.

$$r_t^{\text{fell-over}} = \begin{cases} 1 & \text{if} \quad \theta_t > \theta_{\text{limit}} \\ 0 & \text{otherwise} \end{cases}$$

Illegal Contact Penalty

The illegal contact penalty is a dense penalty that discourages contact between the ground and any part of the robot's body other than its feet.

$$r_t^{\text{illegal-contact}} = \begin{cases} 1 & \text{if illegal contact detected} \\ 0 & \text{otherwise} \end{cases}$$

Joint Position Limit

The joint Position limit penalty is a dense penalty that discourages the robot's joints exceeding their configured position limits.

$$r_t^{\text{joint-limit}} = \begin{cases} 1 & \text{if any joint limit is exceeded} \\ 0 & \text{otherwise} \end{cases}$$

Action Rate

The action rate penalty, is a dense penalty that discourages overly rapid changes in the commanded joint positions.

$$r_t^{\text{action-rate}} = -||\mathbf{a}_t - \mathbf{a}_{t-1}||_2^2$$

Foot Slip

This penalty is applied when a robot's foot is in contact with the ground, while also exhibiting horizontal velocity, indicating slipping.

$$r_t^{\text{foot-slip}} = -\sum_{\text{foot}\in\{\text{left, right}\}} (\|\mathbf{v}_t^{\text{foot}}\|^2 \cdot contact_t^{\text{foot, ground}}),$$

where $\mathbf{v}_{t,xy}^{\text{foot},i}$ is the horizontal velocity of foot i, and $contact_t^{\text{foot,ground}}$ is a boolean whose value is determined by whether a foot touches the ground.

Appendix B

Training Hyperparameters

Parameter	**Value**
Initial noise std	1.0
Noise std type	scalar
Actor observation normalization	True
Critic observation normalization	True
Actor hidden dimensions	(512, 256, 128)
Critic hidden dimensions	(512, 256, 128)
Activation	ELU
Number of learning epochs	5
Number of mini-batches	4
Learning rate	0.0001
Learning rate schedule	adaptive
Discount factor	0.99
GAE lambda	0.95
Desired KL	0.01
Max gradient norm	1.0
Value loss coefficient	1.0
Clipping	0.2
Steps per environment update	24
Parallel environments	4096

Table B.1: Training hyperparameters

Parameter	**Value**
Maximum episode length (T_{max})	400 steps
Contact window (T_{window})	50 steps
Fell over angle limit (θ_{limit})	1.2217 rad ($\approx$ 70°)
Fell down height (z_{falldown})	0.2 m
Target radius (r)	1.0 m
Velocity threshold (ε)	0.5 m/s
Target distance sensitivity (σ)	$\sqrt{0.9}$
Arm posture sensitivity (σ_{roll})	0.5
Upright sensitivity (σ_{upright})	$\sqrt{0.1}$

Table B.2: Environment and reward hyperparameters

Appendix C

Representation for Angular Velocity

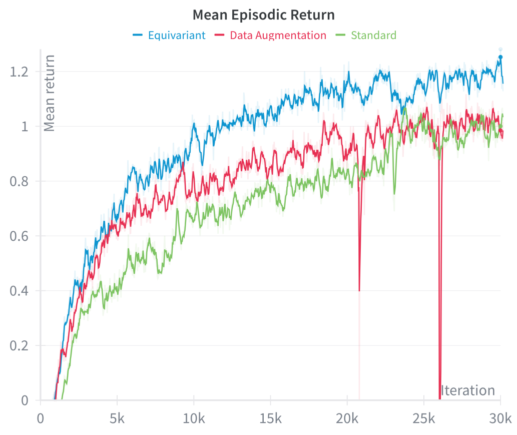
Here the particular representations associated with the angular velocities are explained. On the one hand it is not completely obvious why they are the way they are but on the other hand these details are not so consequential for the content of the thesis. Angular velocities are pseudovectors (or axial vectors) rather than standard vectors. Their direction is defined by the right-hand rule and represents rotation around an axis. The base angular velocity consists of roll (rotation around the forward axis), pitch (rotation around the lateral axis), and yaw (rotation around the vertical axis):

- Roll: Under reflection, the forward axis direction is preserved, but the sense of rotation reverses. A clockwise roll becomes counterclockwise and vice versa. Thus roll $\mapsto$ −roll, giving ρ_{sign}.
- Pitch: Under reflection, both the lateral axis direction reverses (left $\leftrightarrow$ right) and the sense of rotation reverses, resulting in a double negation. A pitch-up motion relative to the body remains pitch-up after reflection. Thus pitch $\mapsto$ pitch, giving ρ_{triv}.
- Yaw: Under reflection, the vertical axis direction is preserved, but the sense of rotation reverses. A leftward yaw becomes a rightward yaw. Thus yaw $\mapsto$ −yaw, giving ρ_{sign}.

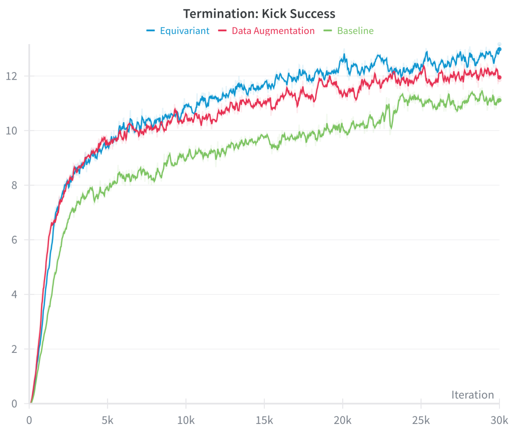
This explains why base angular velocity transforms as $\rho_{\text{sign}} \oplus \rho_{\text{triv}} \oplus \rho_{\text{sign}}$, in contrast to the $\rho_{\text{triv}} \oplus \rho_{\text{sign}} \oplus \rho_{\text{triv}}$ transformation of linear velocities.

Appendix D

Extra Plots

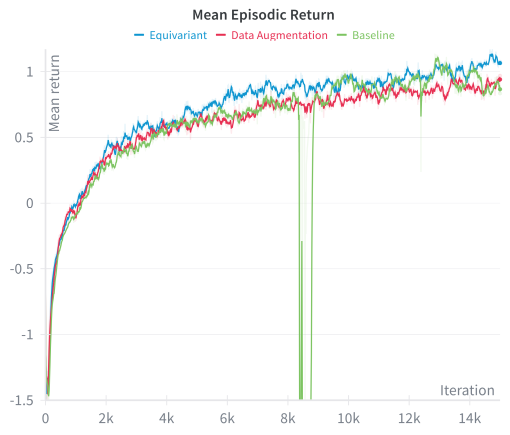


(a)

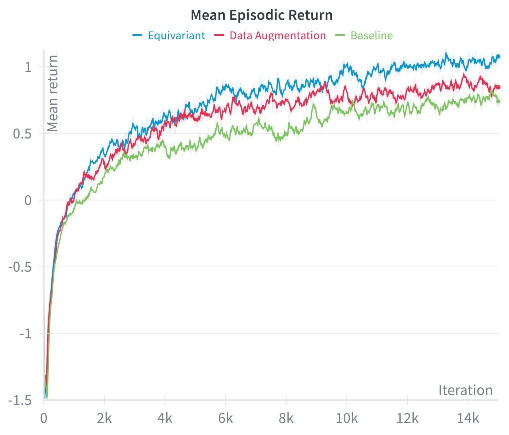


(b)

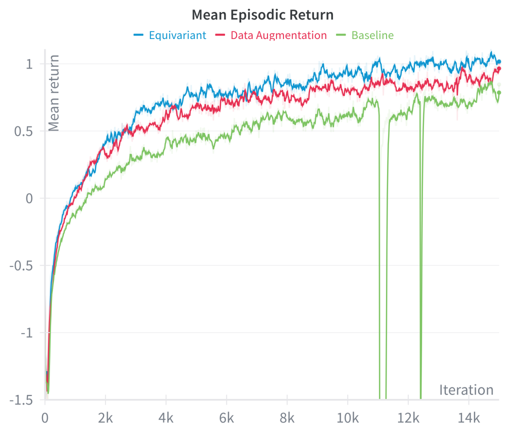
Figure D.1: Mean episodic reward and success terminations for the extra long training run. Truncated for readability.



(a)



(b)



(c)

Figure D.2: Disaggregated runs for the general experiment, separated by seed.