

A Network design and training details

The training and inference details of our learning framework are illustrated in Figure 7. Similarly to DreamWaq, the **Prediction Encoder** predicts the current velocity \hat{v}_t and outputs the latent variable z_2 . Finally, the **Policy Network** generates action a_t based on current observation O_t , estimated velocity \hat{v}_t , and latent variables z_2, z_3 , and z_4 .

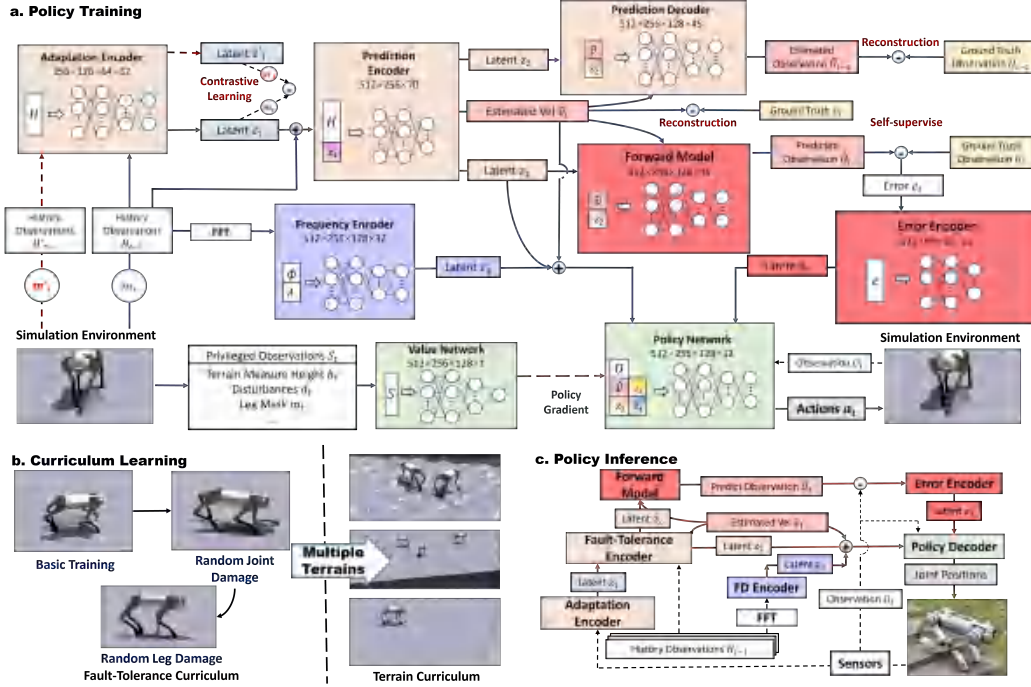


Figure 7: Training and inference details. **a**, The specific network design and parameters used in the training process. **b**, Curriculum learning during the training process. **c**, Network deployed to the real robot in the inference process.

A.1 Reward function design

In the reward function, no prior information about leg damage models is incorporated; instead, the model’s representation capability is relied upon to learn fault-tolerant control. The specific reward functions are as follows:

- **Reward of tracking linear velocity:**

$$R_{\text{tracking_lin}} = \exp\left(-\frac{1}{\sigma_{\text{tracking}}^2} \|\mathbf{c} - \mathbf{v}\|^2\right), \quad (5)$$

where \mathbf{c} represents the linear velocity command vector, \mathbf{v} represents the actual linear velocity vector, and σ_{tracking} is the scaling factor. The reward function penalizes the difference between the commanded and actual linear velocities. A smaller error results in a higher reward, which encourages the model to closely follow the commanded velocities. In this work, $\sigma_{\text{tracking}} = 0.25$.

- **Reward of tracking angular velocity:**

$$R_{\text{tracking_ang}} = \exp\left(-\frac{(\omega_{\text{cmd}} - \omega)^2}{\sigma_{\text{tracking}}}\right), \quad (6)$$

where ω_{cmd} is the angular velocity command, and ω is the actual angular velocity. This reward function penalizes the error between the commanded and actual angular velocities.

Similar to linear velocity tracking, the closer the actual angular velocity is to the commanded value, the higher the reward.

- **Reward for penalizing the z direction velocity:**

$$R_{\text{lin_vel_z}} = (v_z)^2 \quad (7)$$

where v_z is the velocity in the z direction. This reward penalizes any linear velocity along the z axis, which is typically undesirable as it may indicate instability or unintended movement in that direction.

- **Reward for penalizing the xy direction angular velocity:**

$$R_{\text{ang_vel_xy}} = \omega_x^2 + \omega_y^2 \quad (8)$$

where ω_x and ω_y are the angular velocities in the x and y directions, respectively. This function penalizes any angular velocity in the x and y directions, encouraging the model to maintain stability and avoid unnecessary rotation.

- **Reward for penalizing the torque:**

$$R_{\text{torque}} = \tau^2 \quad (9)$$

where τ represents the torque applied to the joints. This reward penalizes high torques, which can lead to excessive energy consumption and wear on the robot's components. The goal is to encourage smoother and more energy-efficient movements.

- **Reward for penalizing joint accelerations:**

$$R_{\text{dof_acc}} = \sum \left(\frac{\dot{q}_t - \dot{q}_{t-1}}{\Delta t} \right)^2 \quad (10)$$

where \dot{q}_{t-1} is the joint velocity at the previous timestep, and Δt is the timestep interval. This function penalizes large accelerations of the joints, promoting smoother transitions in joint velocities, and contributing to the robot's motion's overall stability and smoothness.

- **Reward for penalizing changes in actions:**

$$R_{\text{action_rate}} = \sum (\mathbf{a}^{\text{last}} - \mathbf{a})^2 \quad (11)$$

where \mathbf{a} and \mathbf{a}^{last} represent the current and previous actions, respectively. This reward penalizes rapid changes in actions between consecutive time steps, encouraging smoother and more consistent control signals.

- **Reward for penalizing hip joint position:**

$$R_{\text{hip_pos}} = \sum_j (\mathbf{q}_{\text{hip},j} - \mathbf{q}_{\text{default},j})^2 \quad (12)$$

where $\mathbf{q}_{\text{hip},j}$ and $\mathbf{q}_{\text{default},j}$ represent the current and default positions of the hip joint j , respectively. This reward penalizes deviations from the default or desired hip joint positions, helping to maintain a stable and balanced posture.

- **Reward of action smoothness:**

$$R_{\text{smooth}} = \sum_i \left[(\mathbf{q}_{t,i}^{\text{target}} - 2\mathbf{q}_{t-1,i}^{\text{target}} + \mathbf{q}_{t-2,i}^{\text{target}})^2 \cdot (\mathbf{a}_{t-1,i} \neq 0) \cdot (\mathbf{a}_{t-2,i} \neq 0) \right] \quad (13)$$

where $\mathbf{q}_{t,i}^{\text{target}}$ is the target position at time t for joint i , and $\mathbf{a}_{t-1,i}$ and $\mathbf{a}_{t-2,i}$ represent the actions at previous time steps. This reward penalizes abrupt changes in the joint target positions over time, which promotes smoother and more natural joint movements.

- **Reward of gait:**

$$R_{\text{gait}} = \|\mathbf{a}_{1:3} - \mathbf{a}_{7:9}\|^2 + \|\mathbf{a}_{1:3} - \mathbf{a}_{13:15}\|^2 + \|\mathbf{a}_{4:6} - \mathbf{a}_{10:12}\|^2 \quad (14)$$

$$+ \|\mathbf{a}_{4:6} - \mathbf{a}_{16:18}\|^2 + \|\mathbf{a}_{10:12} - \mathbf{a}_{16:18}\|^2 + \|\mathbf{a}_{7:9} - \mathbf{a}_{13:15}\|^2 \quad (15)$$

This reward function, R_{gait} , is utilized to encourage the 3-3 gait pattern, where the robot's legs move in coordinated pairs. The reward penalizes deviations from synchronized movement patterns across the legs, promoting a stable and efficient gait.

The reward function weights for training the quadruped and hexapod robots are shown in Table A.1.

Reward Function	Quadruped Robot	Hexapod Robot
$R_{\text{tracking_lin}}$	1.0	1.0
$R_{\text{tracking_ang}}$	0.5	0.5
$R_{\text{lin_vel_z}}$	-2.0	-2.0
$R_{\text{ang_vel_xy}}$	-0.05	-0.05
R_{ori}	-1.0	-1.0
R_{torque}	$-1e-5$	$-1e-4$
$R_{\text{dof_acc}}$	$-2.5e-7$	$-2.5e-7$
$R_{\text{action_rate}}$	-0.05	-0.05
$R_{\text{hip_pos}}$	-0.1	-0.1
R_{smooth}	-0.01	-0.05
R_{gait}	0	-0.02

Table 2: Reward Function Weights

A.2 Hyperparameters of training

These hyperparameters are used to configure the training process of the reinforcement learning algorithm and the adaptation module. The `num_learning_epochs` and `num_mini_batches` define the number of training epochs and mini-batches per policy update. The `learning_rate` and `adaptation_module_learning_rate` control the step sizes for updating model and adaptation module parameters. γ is used to discount future rewards and smoothing advantage estimation, respectively. `desired_kl` helps in maintaining the difference between old and new policies within a target range. `max_grad_norm` prevents gradient explosion by clipping gradients. The specific hyperparameter values are shown in Table 3.

Hyperparameter	Value
Number of learning epochs	5
Number of mini batches	4
Learning rate	1×10^{-3}
Adaptation module learning rate	1×10^{-3}
γ	0.99
λ_s	1.0
λ_v	1.0
λ_e	0.01
λ_{vae}	1.0
λ_{cl}	1.0
λ_{ss}	1.0
Desired KL divergence	0.02
Maximum gradient norm	1.0

Table 3: Hyperparameters used in the training process

A.3 Domain randomization

In our domain randomization setup for legged robots, several parameters are adjusted to enhance the robustness and generalization of the policy:

- `rand_interval_s`: The interval in seconds for randomizing parameters.
- `friction_range`: Randomizes the surface friction, making the terrain more or less slippery.
- `restitution_range`: Changes the bounciness of contact surfaces, affecting how the robot interacts with the ground.
- `added_mass_range`: The range for added mass variability.

- `com_displacement_range`: The range for center of mass displacement.
- `randomize_motor_strength`: Varies the strength of the motors to simulate different motor performance characteristics.
- `lag_timesteps`: The number of timesteps to introduce as delay.
- `randomize_Kp_factor`: Randomizes the proportional gain (K_p) of the PD controller.
- `Kp_factor_range`: The range for K_p variability.
- `randomize_Kd_factor`: Randomizes the derivative gain (K_d) of the PD controller.
- `Kd_factor_range`: The range for K_d variability.

Parameter	Value
Randomization interval (s)	15
Friction range	[0.3, 3.0]
Restitution range	[0.0, 0.4]
Added mass range	[-1.0, 2.0]
COM displacement range	[-0.15, 0.15]
Motor strength range	[0.9, 1.1]
Lag timesteps	6
K_p factor range	[0.9, 1.1]
K_d factor range	[0.9, 1.1]

Table 4: Domain randomization parameters

A.4 Terrain training

In our terrain randomization setup for legged robots, various terrain types are defined with specific parameters to enhance the robustness and adaptability of the policy:

- `plane_terrain`: `weight = 1.0`, `height = 0.0`. This represents a flat terrain with a specified height.
- `random_uniform_terrain`: `weight = 3.0`, `min_height = -0.12`, `max_height = 0.12`, `step = 0.01`, `downsampled_scale = 0.15`. This represents a terrain with random uniform height variations within the specified range.
- `sloped_terrain`: `weight = 1.0`, `slope = 0.5`. This represents a sloped terrain with a specified slope angle.
- `pyramid_sloped_terrain`: `weight = 3.0`, `slope = -0.5`, `platform_size = 1.5`. This represents a pyramid-shaped sloped terrain with a specified slope and platform size.
- `wave_terrain`: `weight = 2.0`, `num_waves = 4`, `amplitude = 0.05`. This represents a wavy terrain with a specified number of waves and amplitude.
- `stairs_terrain`: `weight = 1.0`, `step_height = 0.05`, `step_width = 0.5`. This represents a terrain with stairs having specified step height and width.
- `pyramid_stairs_terrain`: `weight = 3.0`, `step_width = 0.4`, `step_height = -0.05`, `platform_size = 3.0`. This represents a pyramid-shaped stairs terrain with specified step width, step height, and platform size.
- `stepping_stones_terrain`: `weight = 1.0`, `stone_size = 1.5`, `stone_distance = 0.1`, `max_height = 0.0`, `platform_size = 4.0`, `depth = -10`. This represents a terrain with stepping stones of specified size, distance, height, platform size, and depth.
- `discrete_obstacles_terrain`: `weight = 1.0`, `max_height = 0.2`, `min_size = 1.0`, `max_size = 2.0`, `num_rects = 20`, `platform_size = 1.0`. This represents a terrain with discrete obstacles of specified size, height, and platform size.



Figure 8: Custom-built hexapod robot in Isaac Gym, MuJoCo, and the real world.

B Hexapod experiment

In this section, we provide a detailed introduction to the hexapod robot used in our experiments, including the hardware specifications and some parameter tuning details. The appearance of the robot is shown in Figure 8. Although the real robot is equipped with a LiDAR, it was not used in the experiments.

Body Part	Position (pos)	Mass (kg)	Inertia ($\text{kg}\cdot\text{m}^2$)
Base	(0, 0, 0.4)	18.0	(0.05507, 0.30204, 0.34116)
RB Hip	(-0.33, -0.053, 0)	0.601	(0.00031, 0.000574, 0.00031)
RB Leg	(0, 0, 0)	0.798	(0.00212, 0.002277, 0.000497)
RB Foot	(0, -0.08025, -0.249)	0.390	(0.00033, 0.00033, 0.00000715)
LB Hip	(-0.33, 0.053, 0)	0.601	(0.00031, 0.000574, 0.00031)
LB Leg	(0, 0, 0)	0.798	(0.00212, 0.002277, 0.000497)
LB Foot	(0, 0.08025, -0.249)	0.390	(0.00033, 0.00033, 0.00000715)
LM Hip	(0, 0.19025, 0)	0.601	(0.00031, 0.000574, 0.00031)
LM Leg	(0, 0, 0)	0.798	(0.00212, 0.002277, 0.000497)
LM Foot	(0, 0.08025, -0.249)	0.390	(0.00033, 0.00033, 0.00000715)
LF Hip	(0.33, 0.053, 0)	0.601	(0.00031, 0.000574, 0.00031)
LF Leg	(0, 0, 0)	0.798	(0.00212, 0.002277, 0.000497)
LF Foot	(0, 0.08025, -0.249)	0.390	(0.00033, 0.00033, 0.00000715)
RF Hip	(0.33, -0.053, 0)	0.601	(0.00031, 0.000574, 0.00031)
RF Leg	(0, 0, 0)	0.798	(0.00212, 0.002277, 0.000497)
RF Foot	(0, -0.08025, -0.249)	0.390	(0.00033, 0.00033, 0.00000715)
RM Hip	(0, -0.19025, 0)	0.601	(0.00031, 0.000574, 0.00031)
RM Leg	(0, 0, 0)	0.798	(0.00212, 0.002277, 0.000497)
RM Foot	(0, -0.08025, -0.249)	0.390	(0.00033, 0.00033, 0.00000715)

Table 5: Hexapod Robot Hardware Specifications

B.1 Hexapod hardware

Table 5 provides a comprehensive overview of the hardware specifications for the hexapod robot used in our experiments. Each row represents a different component of the robot, detailing its position, mass, inertia, and joint range. The base of the robot is positioned at (0, 0, 0.4) meters and has a mass of 18 kg. Each leg consists of a hip, leg, and foot, with their respective positions, masses, and inertias specified. All joints have a range of motion from -3.14 to 3.14 radians. This detailed breakdown helps in understanding the physical configuration and mechanical properties of the robot, which are crucial for both simulation and real-world applications.

B.2 Sim-to-real transfer for hexapod robot

In the process of sim-to-real transfer for a hexapod robot using reinforcement learning strategies, tuning the motor parameters K_p (proportional gain) and K_d (derivative gain) significantly affects

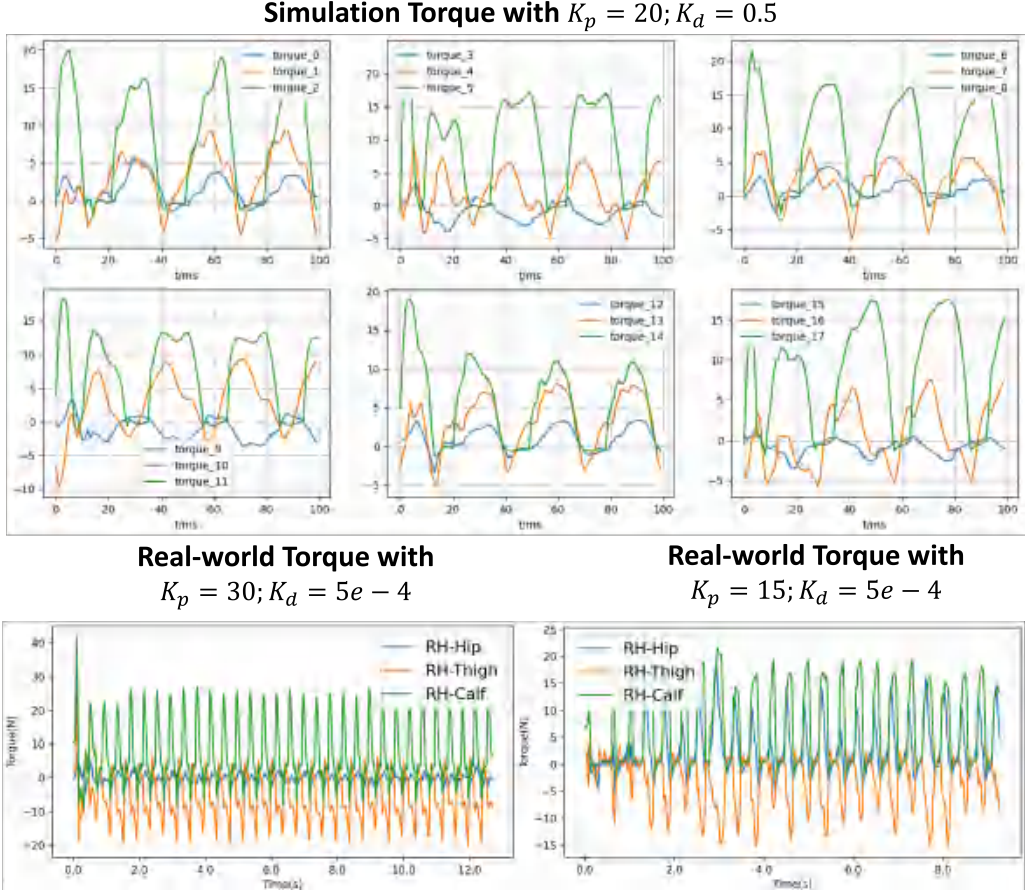


Figure 9: Sim-to-real result of joint torque with different K_p and K_d parameters.

the robot’s performance. This tuning is crucial because it influences how the robot’s joints respond to control signals, impacting stability, responsiveness, and overall movement quality. In the training environment, the motor parameters are initially set with $K_p = 20$ and $K_d = 0.5$. These settings result in joint output torques stabilizing around 20 N, providing a balance between responsiveness and stability that is well-suited for the simulated conditions. The proportional gain K_p controls how forcefully the motors react to positional errors, while the derivative gain K_d helps dampen oscillations by responding to the rate of change of the error. However, when these settings are transferred directly to the real robot, discrepancies often arise due to differences between the simulated model and the actual hardware. As shown in Figure 9, if K_p is increased to 30 in the real robot, the output torque correspondingly increases to approximately 30 N. This higher torque can lead to more aggressive and less controlled movements, as the robot reacts more forcefully to positional errors. This discrepancy results in a mismatch in performance between the simulation and the real world, where the robot may exhibit instability or overshoot desired positions. In contrast, when the real robot’s K_p is lowered to 15, the output torque decreases, and the robot’s movements become less aggressive and more controlled. In this scenario, the real robot exhibits a movement behavior that closely mirrors the simulated environment. The reduced proportional gain means the motors respond less forcefully to errors, resulting in smoother and more stable movements, which aligns better with the conditions the reinforcement learning model was trained under.

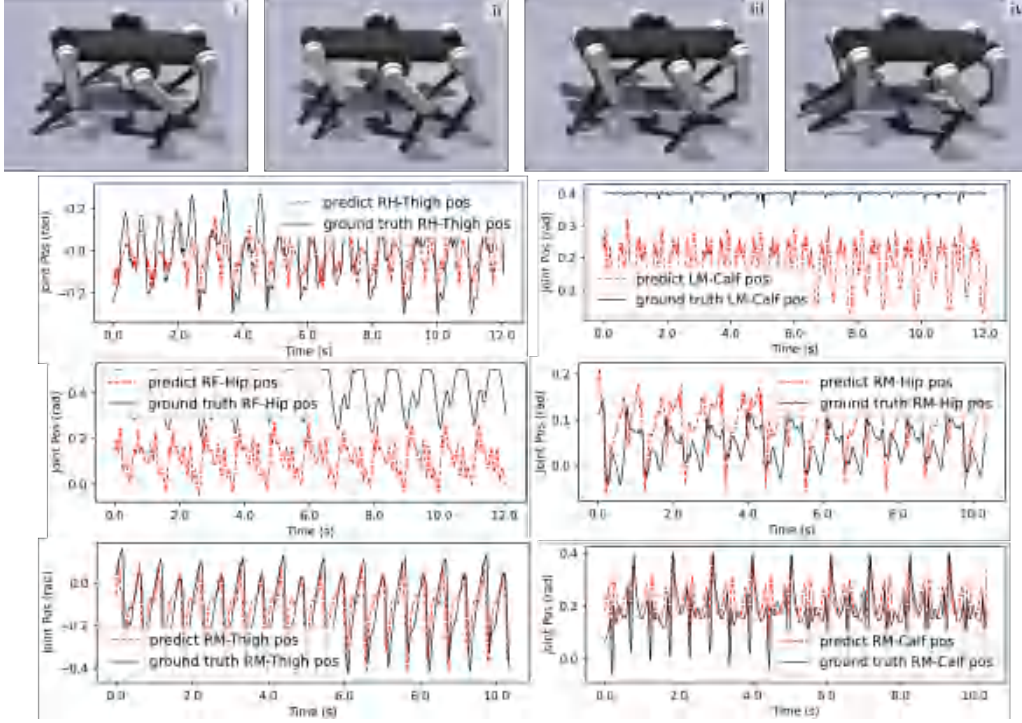


Figure 10: Custom-built hexapod robot in Isaac Gym, MuJoCo, and the real world.

B.3 Forward prediction of hexapod robot

Figure 10 demonstrates that the **Forward Model** is also effective in the hexapod robot. We conducted a robustness evaluation by damaging three joints of the hexapod robot, specifically the RH-Thigh, LM-Calf, and RF-HIP. After training within our framework, the robot was able to adapt and continue walking stably despite these significant impairments. This showcases the robustness and adaptability of the learning framework, as the robot could compensate for the damaged joints and maintain functional locomotion. As illustrated in Figure 10, the **Forward Model** performs best in forecasting the state of the damaged thigh joint, demonstrating the highest accuracy. Conversely, the model exhibits larger prediction errors for the damaged calf and hip joints. As with the quadruped robot, the **Forward Model** accurately forecasts the position changes of healthy joints.