

A State and Action Representation

A.1 Walking

Following the RL training setup in Shi et al. [9], both the base policy $\pi(\mathbf{s}_t)$ and the dynamics-aware policy $\pi(\mathbf{s}_t, \mathbf{z})$ output \mathbf{a}_t as joint position setpoints for proportional-derivative (PD) controllers based on the observable state \mathbf{s}_t :

$$\mathbf{s}_t = (\phi_t, \mathbf{c}_t, \Delta \mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{a}_{t-1}, \boldsymbol{\omega}_t, \boldsymbol{\theta}_t), \quad (5)$$

where ϕ_t is a phase signal, \mathbf{c}_t represents velocity commands, $\Delta \mathbf{q}_t$ denotes the position offset relative to the neutral pose \mathbf{q}_0 , \mathbf{a}_{t-1} is the action from the previous time step, $\boldsymbol{\omega}_t$ represents the torso’s angular velocity, and $\boldsymbol{\theta}_t$ is the torso orientation in euler angles.

In simulation, the critic observation space includes more privileged information to provide better accuracy for the value function, we have:

$$\mathbf{s}_t^+ = (\phi_t, \mathbf{c}_t, \Delta \mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{a}_{t-1}, \mathbf{e}_{\mathbf{q}_t}, \mathbf{v}_t, \boldsymbol{\omega}_t, \boldsymbol{\theta}_t, \mathbf{m}_t, \mathbf{ref}_t, \tilde{\mathbf{v}}_t, \tilde{\boldsymbol{\theta}}_t), \quad (6)$$

where $\mathbf{e}_{\mathbf{q}_t}$ is the error of the current motor position to a reference motor position, \mathbf{v}_t is the linear velocity of the robot in the world frame, \mathbf{ref}_t is the reference motor pose, while $\tilde{\mathbf{v}}_t$ and $\tilde{\boldsymbol{\theta}}_t$ are the linear and angular velocity of the random perturbation applied to the robot.

In the real world, as some of the privileged observations in the simulation are hard to acquire, we leverage the RTR system’s force information, and augment the real-world privileged observation as:

$$\mathbf{s}_t^{r+} = (\phi_t, \mathbf{c}_t, \Delta \mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{a}_{t-1}, \mathbf{q}_t, \mathbf{v}_t, \boldsymbol{\omega}_t, \boldsymbol{\theta}_t, \mathbf{F}_t, \boldsymbol{\tau}_t), \quad (7)$$

where \mathbf{q}_t is the joint position of the humanoid, \mathbf{v}_t is the linear velocity of the torso measured by a motion tracking system, and \mathbf{F}_t and $\boldsymbol{\tau}_t$ are the force and torque measured by the force-torque sensor mounted on the robot arm teacher.

A.2 Swing-up

For the swing-up task, the observation space is defined the same as Equation 5. We also include more information to form the task’s privileged observation space:

$$\mathbf{s}_t^+ = (\phi_t, \mathbf{c}_t, \Delta \mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{a}_{t-1}, \boldsymbol{\omega}_t, \boldsymbol{\theta}_t, \mathbf{F}_t, \boldsymbol{\tau}_t, \mathbf{x}_t) \quad (8)$$

where $\mathbf{F}_t, \boldsymbol{\tau}_t, \mathbf{x}_t$ are the force reading, torque reading, and arm end effector position, respectively.

B FiLM Ablation

In this section, we discuss details regarding the FiLM layers in our sim-to-real adaptation pipeline. As shown in Equation (1), the FiLM layer introduces a scaling and shifting effect to each layer of the hidden network, altering the behavior of the policy network inherently.

One key consideration for the FiLM layer during training is its learning rate, as it decides how fast the FiLM layer will change compared to the policy network. In our implementation, we train the FiLM layers along with the policy network from scratch, as we find that initializing the policy network from a pretrained model hinders the effect of the FiLM layer over the policy. Recall that:

$$\gamma_j^{(i)}, \beta_j^{(i)} = \text{FiLM}_j(z^{(i)}), \quad h_j^{(i)} \leftarrow \gamma_j^{(i)} \odot h_j^{(i)} + \beta_j^{(i)}$$

We initialize all the FiLM layers with all-zero weights and all-zero or all-one bias, such that at the start of the training, we have: $\gamma_j^{(i)} = 1.0, \beta_j^{(i)} = 0.0$, which means the FiLM layers do not affect the network output. As the FiLM layers are trained simultaneously with the policy, they are in a

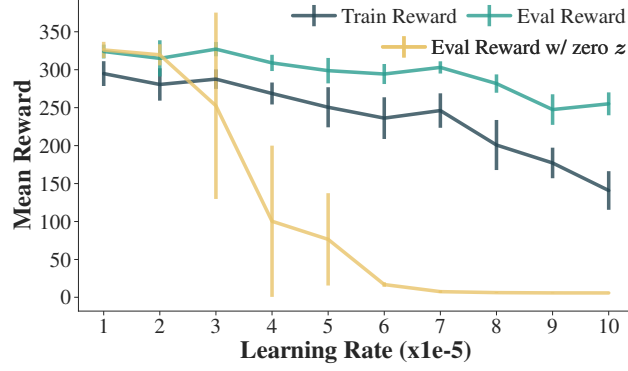


Figure 6: **FiLM learning rate ablation.** This experiment aims to evaluate the effect of FiLM layer learning rates. All experiments are run under seven seeds. Vertical bars indicate standard deviation.

Table 2: We randomize the following environment parameters for the walking task.

Parameter Names	Friction	Damping	Armature	Friction Loss	Body Mass	EE Mass
Randomize Range	[0.5, 2.0]	[0.5, 2.0]	[0.5, 2.0]	[0.5, 2.0]	[−0.3, 0.3]	[0.0, 0.1]

competing condition: the policy network is learning to gain better overall performance given the current FiLM distribution, while the FiLM layers are trained to make use of the environment’s latent and maximize the performance in each of the different environments.

We find that a small learning rate for the FiLM layers causes the policy to behave similarly to training without the modulation, limiting its ability to generalize across latent conditions and adapt to different environments. On the other hand, if the learning rate is too large, it will lead to unstable training and lower performance.

We propose a metric to measure the effectiveness for the dynamics latent conditioned $\pi(s, z)$ to utilize the environment information embedded in the latent z . Specifically, we compare the performance of the policy given the true latent z computed from the dynamics encoder $z^{(i)} = f_{\phi}(\mu^{(i)})$ and give an all-zero latent. We find that the performance of an all-zero latent could be even comparable to that given the real z when the FiLM learning rate is small, but it will gradually decrease to near zero when the learning rate increases and the policy starts to learn to use the latent information. On the other hand, the overall performance of the policy will decrease nearly monotonously when the FiLM learning rate increases. We plot the performance curves at different learning rates in Figure 6. For each learning rate, we run seven trials and plot the mean and standard deviation of the performance. We find that the learning rate of $5e - 5$ is a good trade-off between performance and the ability to utilize latent information, while $1e - 5$ is too small, leading to nearly identical zero latent and true latent performance, and $1e - 4$ is too large and leads to poor performance.

C Experiment Details

C.1 Domain Randomization

Following the domain randomization settings in [9], we slightly increase the domain randomization range to encourage the policy network to better use the dynamics information from the latent. We list the randomized parameters and their range in Table 2. We use the encoder-decoder architecture from [52] to encode the physics parameters to a 1024-dimensional latent before sending it to the FiLM layers, though we only use the encoder in our implementation.

C.2 Arm Control Policy

We implement our appliance arm controller based on the open-sourced code of Hou et al. [51].

For the walking task, we enable appliance control along the XY axis while setting the Z-height directly. We use a stiffness of $[100, 50]$ along the two axis, while setting the damping to $[0.5, 0.5]$ and inertia to $[0.03, 0.03]$, making the arm slightly more compliant on the Y-axis.

For the swing task, we disable the appliance control for the arm since it makes the phase tracking lag behind. We use position control to let the arm follow the helping or perturbing movement described in Section 3.3. During both helping and perturbing modes, each real-world data collection period is evenly divided into 5 bins. In helping mode, 3 bins are randomly selected from the last 4 to apply assistance. In perturbing mode, 1 bin is randomly selected from the first 4 to apply the perturbation.

C.3 Treadmill Control Policy

During the walking task, the robot is walking on the treadmill while the treadmill adjusts its speed dynamically to keep the robot in a good position. We use a PD controller for the treadmill speed v :

$$v = v_{base} + k_p^1 F_x + k_p^2 \psi \quad (9)$$

where $v_{base} = 0.1$ m/s is the default treadmill speed, F_x is the force reading along the x -axis and ψ is the robot’s torso pitch. $k_p^1 = 0.2$ and $k_p^2 = -5$ are the proportional gain. We set a treadmill speed limit of 0.24 m/s to assure the safety of the humanoid platform.

C.4 Online Learning Hyperparameters

During real-world adaptation for walking, we collect a batch of 1024 steps of data before updating the policy on them over 20 epochs using the PPO algorithm with a clipping ratio of 0.2. Both the actor and critic networks are optimized with a learning rate of 1×10^{-4} . To encourage exploration, we apply an entropy coefficient of 0.005.

For training the swing-up task from scratch, we also collect 1024 samples before starting to update the policy. The actor is optimized with a learning rate of 2×10^{-3} and the critic is optimized with a learning rate of 2×10^{-5} . To encourage exploration, we apply an larger entropy coefficient of 0.04. Each PPO update is performed over 20 epochs with a clipping ratio of 0.2.

D Comparison with RMA

Rapid Motor Adaptation (RMA) [29] is a sim-to-real adaptation method that is similar to our approach. The training of RMA consists of two stages: (1) A pretraining stage, where the policy is trained in a simulated environment with domain randomization. In this stage, the latent information is encoded from the physics parameters by a projection layer, and then concatenated with the observation before being sent to the policy. (2) An adaptation stage, which is to address the problem that the physics parameter of the real world is unknown. During this stage, an adaptation module is trained via supervised learning to reconstruct the latent z from the past observations and actions.

Compared to RMA, our approach bears differences in each stage: for the first stage, our method uses a FiLM layer to modulate the policy network, which is a more flexible and powerful way to utilize the latent information. In contrast, RMA simply concatenates the latent information with the observation. In the second stage, our method does not require the adaptation module to reconstruct the latent z , but instead, we optimize a universal latent z^* that is shared across all the environments. This could serve as a good initialization for further training of the optimal real-world latent z_{real}^* . On the other hand, RMA tries to reconstruct the latent z from the past observations and actions, which is prone to overfitting, especially when facing the largely out-of-distribution real-world dynamics.

In the following section, we set up simulation and real-world experiments to compare our method with RMA. We try to answer the following questions: (1) How does the FiLM layer modulation affect the performance compared to RMA’s simple concatenation? (2) How good is the adaptation module compared to our approach to get a universal latent? For each experiment, we conduct ablation studies and keep all other parameters the same. We then evaluate the fully trained RMA

model in the real world to prove that our method is indeed a better choice from each perspective for real-world adaptation.

D.1 RMA Implementation Details

We describe the details we use to implement the RMA algorithm on our humanoid platform. During phase one training, we randomize the simulation environment using the same parameter range as described in Appendix C.1, and the randomized parameters are then concatenated to form a physics-information vector. We drop the unchanged digits during the encoding process.

The RMA algorithm is originally designed for a quadruped robot that has relatively low DoFs. Compared with the original implementation, we use a similar process to train a stage one model that takes in the observation and the dynamics latent and outputs the action. During stage two, we make the following changes to adapt the algorithm to our hardware: (1) We decrease the window length of past observations and actions that used to predict the current dynamics latent from 50 to 15, as the observation of the humanoid platform has higher dimensions, and a too long horizon will cause difficulty in training. This adjustment also aligns with the stack frame length we use during training. (2) We change the 1D convolution layers used to process the past states and actions accordingly, since the context window size is reduced. The new convolution layers now have kernel sizes of (5, 3, 3) and stride steps of (1, 1, 1). Besides these changes, the RMA training reuses the existing real-world learning pipeline, while only optimizing the adaptation module in stage two instead of the dynamics latent as in the implementation of RTR.

D.2 FiLM Latent Modulation

We first compare the effect of FiLM layer modulation used by RTR with the concatenation approach used by the first stage in RMA. To this end, we run three seeds for the RMA to train a dynamics latent conditioned policy in 1024 parallel simulation environments. While the FiLM layer-based policy easily reaches an average evaluation return of higher than 300, the concatenated policy can only reach a return of just over 200. We suspect that this is due to the high dimensionality introduced by concatenation, which will hinder the policy performance. The co-existence of the observation and the dynamics latent in the MLP policy input is also likely to cause the network to ignore the dynamics. The detailed results can be found in the first column of Table 3, where the “Ground Truth” denotes that both methods have access to the dynamics latent directly predicted from the encoder.

D.3 Adaptation Module Performance

During the second stage of RMA, an adaptation module is trained to predict the dynamics latent from past observations. While in RTR, a universal latent is trained across randomized environments for initialization of a real-world learning stage. We want to investigate which method could lead to a better estimation of the real dynamics latent. In addition to the design choice of concatenation or FiLM layer modulation studied in the previous subsection, we conduct ablation studies to fully compare the performance of both methods. The results can be found in Table 3.

In the first row of Table 3, we use the stage one of RMA to train a dynamics conditioned policy via directly concatenating the dynamics latent to the observation, which will lead to inferior performance than the FiLM layers. We then use both universal latent optimization and the adaptation module to predict the dynamics latent. The result shows that both methods did pretty well to recover the latent, resulting in a nearly identical performance to the true latent used in phase one.

In the second row of Table 3, we add the dynamics conditioning to the policy via FiLM layers as in RTR. The performance of the conditioned policy is higher than that of the concatenated latent. We then compare the two methods’ ability to estimate the dynamics latent for the FiLM layers. This time, the RMA-style adaptation module failed to recover a feasible dynamics latent for the FiLM layers, causing the performance to stay at a low position, almost close to that of a random policy. While the prediction error of the adaptation module is decreasing, we suspect this is due to the FiLM

Table 3: We compare the performance for each stage of RTR (ours) and RMA [29] in this table

	Ground Truth	Adaptation Module (RMA)	Universal Latent (RTR)
Concatenation (RMA)	210.32 \pm 20.75	205.74 \pm 15.63	207.32 \pm 5.89
FiLM layer (RTR)	316.10 \pm 11.30	10.32 \pm 2.01	305.28 \pm 5.47

Table 4: We compare RTR and RMA [29] in this table, extending the results from Table 1.

Method	Torso Roll \downarrow	Torso Pitch \downarrow	EE Force X \downarrow	EE Force Y \downarrow	EE Force Z \downarrow
RMA	0.167 \pm 0.010	0.212 \pm 0.006	1.172 \pm 0.007	0.799 \pm 0.028	2.548 \pm 0.216
RTR (ours)	0.093 \pm 0.020	0.053 \pm 0.044	0.943 \pm 0.202	0.754 \pm 0.122	0.954 \pm 0.445

layers being more sensitive to the small changes of the dynamics layer, and the adaptation module can’t fully close this gap from the past observations and actions.

D.4 Real-World Performance

Finally, we run the RMA model in the real world and test its performance using the same metrics as the main paper. While our implementation of the RMA method successfully adapts to the real world to produce a walking behavior somewhat similar to the simulation, the walking gait of the RMA policy is not that stable, and the humanoid often leans forward and is frequently recovered by the RTR hardware. The metrics in Table 4 show that our method clearly outperforms RMA for real-world adaptation.

In summary, during the first stage of training, the approach of using the FiLM layer in RTR outperforms the concatenation method used by RMA. During the second stage, our approach of universal latent optimization could provide similar initial latent compared to RMA, and is more stable when combined with the FiLM layers. What’s more, our approach of universal latent optimization leads to a third real-world tuning stage, and has the ability to further boost the real-world performance with the RTR hardware. The results in both simulation and real-world experiments lead to the conclusion that the dynamics latent optimization pipeline of RTR is better suited for sim-to-real adaptation of humanoids than the approach used by RMA.