VR-Robo: A Real-to-Sim-to-Real System for Robot Navigation and Locomotion

Author Names Omitted for Anonymous Review. Paper-ID [45]

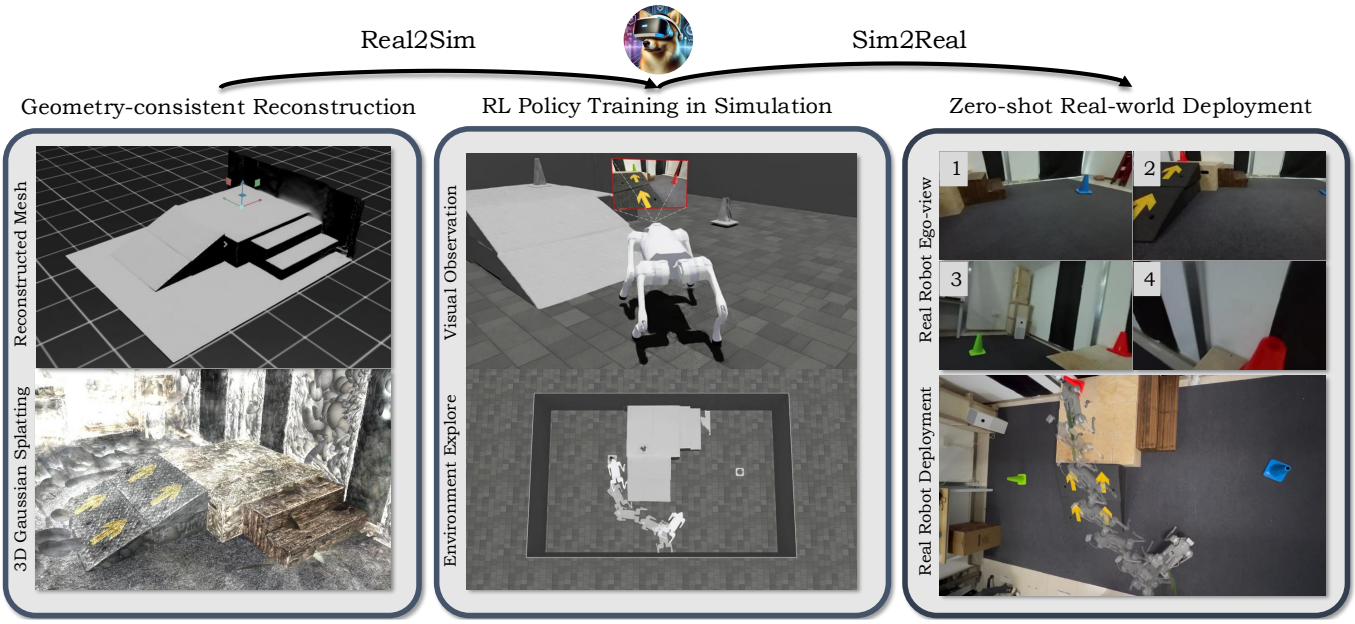


Fig. 1: **Our VR-Robo introduce a *Real-to-Sim-to-Real* system.** We generate photorealistic and physically interactive "digital twin" simulation environments for ego-centric visual navigation and locomotion. VR-Robo facilitates the RL policy training of legged agents with exploration capability in complex scenarios. Notably, our system achieves RGB-only zero-shot sim-to-real policy transfer to the real world. **Please refer to the supplementary video for simulation and real-world experiment results**.

***Abstract*—Recent success in legged robot locomotion is attributed to the integration of reinforcement learning and physical simulators. However, these policies often encounter challenges when deployed in real-world environments due to sim-to-real gaps, as simulators typically fail to replicate visual realism and complex real-world geometry. Moreover, the lack of realistic visual rendering limits the ability of these policies to support high-level tasks requiring RGB-based perception like ego-centric navigation. This paper presents a *Real-to-Sim-to-Real* system that generates photorealistic and physically interactive "digital twin" simulation environments for visual navigation and locomotion learning. Our approach leverages 3DGS-based scene reconstruction from multi-view images and integrates these environments into simulations that support ego-centric visual perception and mesh-based physical interactions. To demonstrate its effectiveness, we train a reinforcement learning policy within the simulator to perform a visual goal-tracking task. Extensive experiments show that our system achieves RGB-only sim-to-real policy transfer. Additionally, it facilitates rapid adaptation of robots to new environments, highlighting its potential for applications in households and factories.**

I. Introduction

Exploring, perceiving, and interacting with the physical real world is crucial for legged robotics, which supports many applications such as household service and industrial automation. However, conducting real-world experiments poses considerable challenges due to safety risks and efficiency constraints. Consequently, training in simulation [41, 28, 37, 30, 22, 49, 13] has emerged as an effective alternative, allowing robots to experience diverse environmental conditions, safely explore failure cases, and learn directly from their actions. Despite these advantages, transferring policies learned in simulated environments to real-world remains challenging, owing to the significant *Sim-to-Real* reality gap [23, 39, 41].

Substantial efforts have been made to narrow the *Sim-to-Real* gap in legged locomotion. Previous studies have employed cross-domain depth images to train agents in simulation, achieving impressive zero-shot policy transfer for both quadruped [58, 8, 15, 47] and humanoid [59] robots. To further integrate RGB perception into the *Sim-to-Real* pipeline, LucidSim [53] leverages generative models within simulation [41] for visual parkour. However, these depth-based and generation-based visual policies are predominantly constrained to low-level locomotion tasks, primarily because standard simulators struggle to replicate the visual fidelity and complex geometry of real-world environments.

Recently, advanced neural reconstruction techniques such as NeRF [29] and 3DGS [20] have emerged as promising solutions for creating *Real-to-Sim* "digital twins" from real-

world data. Nonetheless, most existing approaches [2, 32, 26, 25, 33, 48, 51] focus on enhancing photorealism, offering limited support for physical interaction with complex terrains. Moreover, the simulations in these works often lack mechanisms for robust environment exploration, thus constraining their deployment in complex real-world scenarios that demand dynamic interaction.

To address these challenges, we introduce VR-Robo, a novel *Real-to-Sim-to-Real* system that enables realistic, interactive *Real-to-Sim* simulation and reinforcement learning (RL) policy training for legged robot navigation and locomotion, minimizing the *Sim-to-Real* gap. Given the multi-view images, we employ planar-based [5, 17, 6] 3DGS [20] and foundation-model priors [52] to reconstruct geometry-consistent scenes and objects. We then propose a GS-mesh hybrid representation with coordinate alignment to create a "digital twin" simulation environment in Isaac Sim, which supports ego-centric visual perception and mesh-based physical interactions. To enable robust policy training, we introduce an agent-object randomization and occlusion-aware scene composition strategy, as illustrated in Figure 6.

With the reconstructed simulation, we adopt a two-level asynchronous control strategy for training the visual RL policy. In the first stage, we train a low-level locomotion policy that enables the robot to perceive and utilize terrain like slopes and stairs to reach the high platforms. However, this policy alone does not address high-level ego-centric navigation tasks with RGB inputs. In the second stage, we train a high-level policy to explore the environment, identify target goals, and plan feasible paths. The agent utilizes ego-centric visual observations rendered via GS, interacts with the mesh in Isaac Sim, and updates its policy using rewards, as depicted in Figure 5. Through our extensive experiments, the trained policy demonstrates sufficient robustness to be transferred zero-shot to real-world scenarios using only RGB observations.

We summarize our contributions as follows:

1) **A *Real-to-Sim-to-Real* system for robot navigation and locomotion.** We propose to reduce the *Sim-to-Real* gap by training RL policies within a realistic and interactive simulation environment reconstructed directly from real-world scenarios.
2) **Photorealistic and physically interactive *Real-to-Sim* environment reconstruction.** We introduce a novel pipeline for transferring real-world environments into simulation using a GS-mesh hybrid representation with coordinate alignment. We further incorporate object randomization and occlusion-aware scene composition for robust and efficient RL policy training.
3) **Zero-shot *Sim-to-Real* RL policy transfer.** Through extensive experimental evaluations, we demonstrate that VR-Robo produces effective navigation and locomotion policies in complex, real-world scenarios using RGB-only observations.

II. Related Works

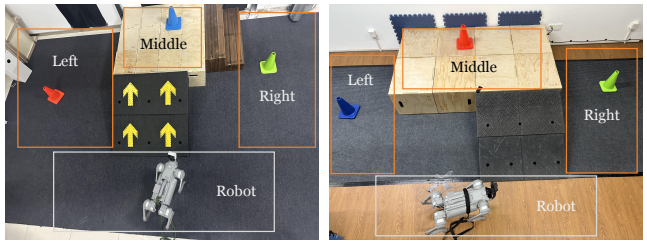
A. Sim-to-Real Policy Transfer

Transferring reinforcement learning (RL) policies trained in simulation to the real world remains a major challenge due to substantial domain gaps. Traditional simulator-based methods, such as domain randomization [45, 31, 40, 50, 35, 43] and system identification [18, 38, 1, 4], aim to reduce the *Sim-to-Real* gap by aligning simulations with physical setups. Recent work [24, 46, 7, 54] proposes leveraging conditional generative models to augment visual observations for more robust agent training. LucidSim [53], for example, incorporates RGB color perception into the *Sim-to-Real* pipeline to learn low-level visual parkour from generated images. However, these approaches remain constrained by conventional simulators, which fail to capture the full breadth of real-world physics and visual realism necessary for high-level policy training and real-world deployment.

B. Real-to-Sim Scene Transfer

Recently, advances in scene representation and reconstruction such as Neural Radiance Fields (NeRF)[29] and 3D Gaussian Splatting (3DGS)[20] have facilitated the creation of high-fidelity digital twins that closely replicate real-world environments for *Real-to-Sim* scene transfer. For instance, NeRF2Real [2] integrates NeRF into simulation for vision-based bipedal locomotion policy training, yet lacks physical interaction with reconstructed geometries. Meanwhile, RialTo [42] augments digital twins with articulated USD representations to enable manipulative interactions. More recent works employ 3DGS to generate realistic simulations[19] for both robot manipulation [26, 25, 33, 48] and navigation [32, 51]. In contrast, VR-Robo is designed to produce photorealistic and physically interactive "digital twin" environments specifically tailored for ego-centric visual locomotion learning.

III. Task Definition



Scene 1 Scene 2

Fig. 2: **Task Definition.** The robot needs to reach the target cone of a specified color, relying solely on RGB observations and proprioception.

As shown in Figure 2, the task is defined as reaching the target cone of a specified color within a limited time horizon. We consider two distinct scenes, each featuring a terrain in the center composed of a high table and a slope connecting the ground to the platform. At the start of each trial, the robot's initial position is sampled from $\mathbf{p}_{\text{robot}} \sim \mathcal{U}(\mathbb{P}_{\text{robot}})$, and its

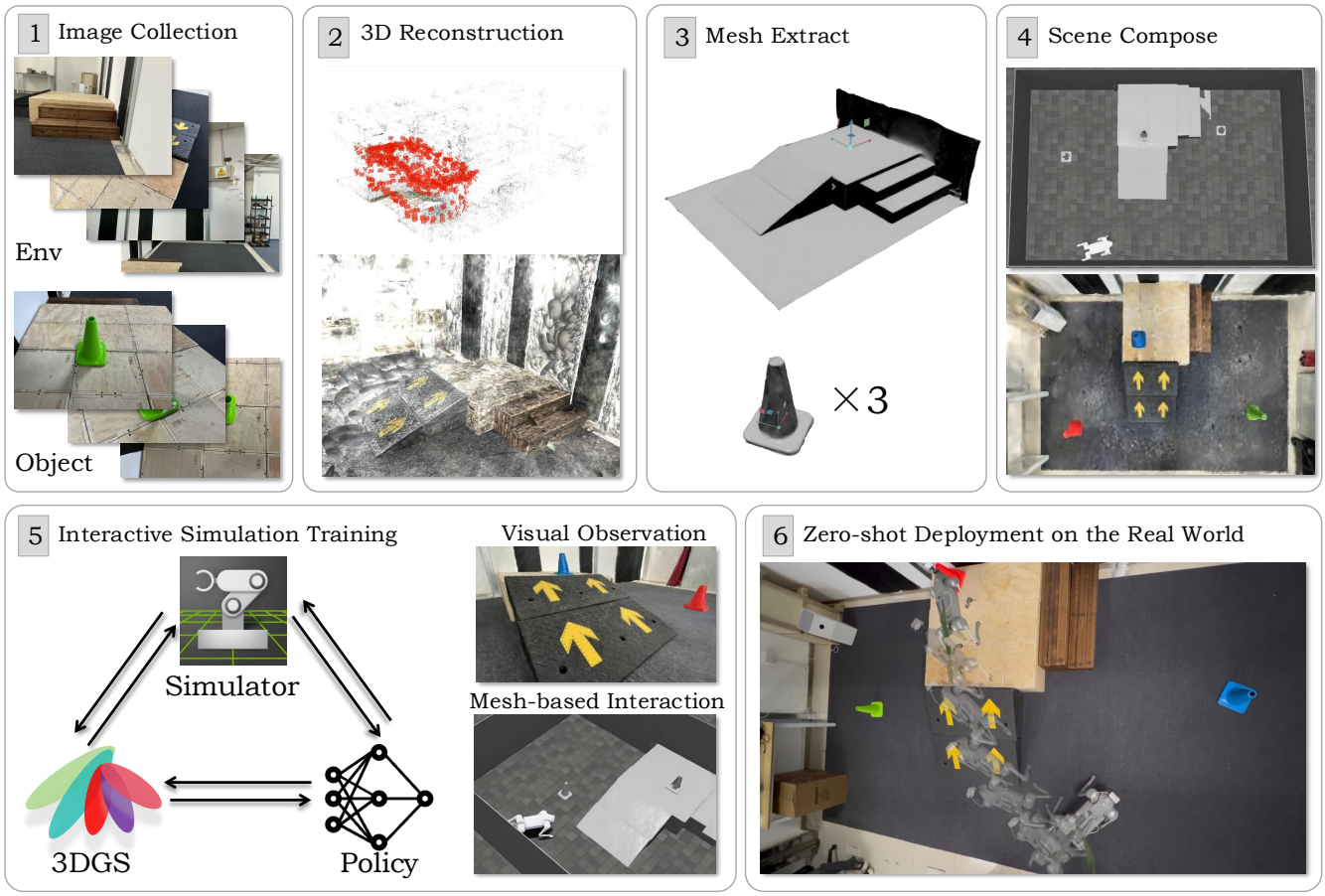


Fig. 3: **Framework overview of our Real-to-Sim-to-Real system.** We first reconstruct the 3D Gaussians and meshes of the environment and objects from the captured images. These reconstructions are then randomly composed. After that, we train the policy network in simulation. Finally, the policy is deployed to the real robot without requiring additional fine-tuning.

orientation is similarly randomized as $\theta_{\text{robot}} \sim \mathcal{U}(\theta_{\text{robot}})$. Each scene contains three cones with identical shapes and textures but different colors $\mathcal{C} = \{$ red, green, blue $\}$. These cones are placed in three designated regions: left, middle, and right with one cone per region at a random position $\mathbf{p}_{\text{cone}} \sim \mathcal{U}(\mathbb{P}_{\text{cone}}^{\text{region}})$. Since the robot may not initially face the target cone, it must explore the environment to locate and identify the correct cone. Additionally, if the target cone is placed atop the table, the robot must navigate up the slope to access the platform.

IV. A Real-to-Sim-to-Real System for Ego-Centric Robot Locomotion

A. Geometry-Consistent Reconstruction

Given multi-view RGB images of a scene with corresponding camera poses from COLMAP [36], we aim to generate a photorealistic, physically interactive simulation environment for agent policy training, thereby minimizing the sim-to-real gap. To this end, we first reconstruct a high-quality, geometry-aware 3D environment using planar-based 3D Gaussian Splatting (GS) [20] with geometric regularization.

Gaussian-based 3D Scene Representation. 3D Gaussian Splatting (3DGS) represents the scene as a set of Gaussian primitives. In particular, each Gaussian splat $\mathcal{G}_i(\mathbf{x})$ is parameterized by its mean $\mu_i \in \mathbb{R}^3$, a 3D covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$, an opacity term o_i, and a color term c_i, as follows:

$$\mathcal{G}_i(\mathbf{x}) = \exp\left(-\frac{1}{2}\left(\mathbf{x}-\mu_i\right)^T \Sigma_i^{-1}\left(\mathbf{x}-\mu_i\right)\right), \quad (1)$$

During optimization, Σ_i is reparametrized using a scaling matrix $S_i \in \mathbb{R}^3$ and a rotation matrix $R_i \in \mathbb{R}^{3\times3}$ as $\Sigma_i = R_i S_i S_i^T R_i^T$. The color value of a pixel $\mathbf{c}(x)$ can be rendered through a volumetric alpha-blending [20] process:

$$\mathbf{c}(x) = \sum_{i\in N} T_i c_i \alpha_i(\mathbf{x}), \quad T_i = \prod_{i=1}^{i-1}\left(1-\alpha_i(\mathbf{x})\right). \quad (2)$$

where $\alpha_i(\mathbf{x}) = o_i \mathcal{G}_i(\mathbf{x})$ denotes the alpha of the Gaussian $\mathcal{G}_i$ at $\mathbf{x} \in \mathbb{R}^3$. Meanwhile, the view-dependent color c_i of $\mathcal{G}_i$ is represented by its spherical harmonics (SH) [21] coefficients.

Flattening 3D Gaussians for Geometric Modeling. Inspired by [17, 6], we flatten the 3D Gaussian ellipsoid with covariance $\Sigma_i = R_i S_i S_i^T R_i^T$ into 2D flat planes to enhance the scene geometry modeling by minimizing its shortest axis scale $S_i = \text{diag}\left(s_1, s_2, s_3\right)$:

$$\mathcal{L}_{\texttt{scale}} = \frac{1}{N}\sum_{i\in N} \left\|\min\left(s_1, s_2, s_3\right)\right\|, \quad (3)$$

where N denotes the total number of planar-based Gaussian splats. Following PGSR [5], we utilize the plane representation to render both the plane-to-camera distance map $\mathcal{D}$ and the normal map $\mathcal{N}$, then convert them into unbiased depth maps by intersecting rays with the corresponding planes as:

$$\boldsymbol{D}(p)_{\texttt{render}} = \frac{\mathcal{D}}{\mathcal{N}(p)K^{-1}\tilde{p}}. \tag{4}$$

where p is the 2D position on the image plane. $\tilde{p}$ denotes the homogeneous coordinate of p, and K is the camera intrinsic.

Depth Prior Regularization. In regions with poor texture (e.g., ground and walls), geometric constraints tend to be insufficient. To improve surface reconstruction in these areas, we employ an off-the-shelf monocular depth network, Depth Anything V2 [52], as a dense per-pixel depth prior. We address the inherent scale ambiguity between the estimated depths and the actual scene geometry by comparing them to sparse Structure-from-Motion (SfM) points, following [44, 9, 55]. Specifically, we align the scale of the monocular depth map $\boldsymbol{D}_{\texttt{mono}}$ with the SfM points projected depth map $\boldsymbol{D}_{\texttt{sfm}}$ by solving the per-image scale $\boldsymbol{s}$ and shift $\boldsymbol{t}$ parameters using linear regression:

$$\hat{\boldsymbol{s}}, \hat{\boldsymbol{t}} = \arg\min_{\boldsymbol{s},\boldsymbol{t}} \sum_{p \in \boldsymbol{D}_{\texttt{sfm}}} \|(\boldsymbol{s} * \boldsymbol{D}(p)_{\texttt{mono}} + \boldsymbol{t}) - \boldsymbol{D}(p)_{\texttt{sfm}}\|_2^2 . \tag{5}$$

Finally, we use the re-scaled monocular depth map $\hat{\boldsymbol{s}} * \boldsymbol{D}(p)_{\texttt{mono}} + \hat{\boldsymbol{t}}$ to regularize the rendered depth map $\boldsymbol{D}(p)_{\texttt{render}}$.

Multi-view Consistency Constraints. Inspired by [12], a patch-based normalized cross-correlation (NCC) loss is applied between two neighboring frames $\{\mathbf{I}_{\texttt{render}}, \hat{\mathbf{I}}_{\texttt{render}}\}$ to force the multi-view photometric consistency:

$$\mathcal{L}_{\texttt{mv}} = 1 - \frac{1}{\|\mathcal{P}\|} \sum_{\mathbf{p} \in \mathcal{P}} \sum_{p \in \mathbf{p}} NCC(\hat{\mathbf{I}}(\mathcal{H}p)_{\texttt{render}}, \mathbf{I}(p)_{\texttt{render}}). \tag{6}$$

where $\mathcal{P}_{\texttt{render}}$ is the set of all patches extracted from $\mathbf{I}$ and $\mathcal{H}$ is the homography matrix bwteen the two frames.

B. Building Realistic and Interactive Simulation

To enable the agent-environment interaction, we integrate a GS-mesh hybrid scene representation into the Isaac Sim with coordinate alignment. We further leverage agent-object randomization and occlusion-aware scene composition to advance robust visual policy training.

GS-mesh Hybrid Representation. To enhance the realism and interactivity of our simulation environment, we introduce a hybrid scene representation that combines our Gaussian-based model with its corresponding scene mesh primitives.

1) The ***Gaussian*** representation generates photorealistic visual observations from the robot's ego-centric viewpoints. Because the robot camera differs from the camera used for 3D scene reconstruction, we first align the intrinsic parameters between *Sim* and *Real* by calibrating the robot camera's focal length and distortion parameters. We then obtain the camera extrinsic within the simulation coordinate system by retrieving the ego-view position and quaternion-based orientation from Issac.

2) The ***Mesh*** representation facilitates physical interaction and precise collision detection. We render the depth for each input view and leverage a Truncated Signed Distance Function (TSDF) [10] fusion algorithm to build the corresponding TSDF field. We then extract the mesh from this TSDF field to enable physically accurate interactions within the simulation.

Coordinate Alignment. To align the reconstructed COLMAP coordinate system with the Isaac Sim environment, we first compute the homogeneous transformation matrix $T_{\texttt{homo}} \in \mathbb{R}^{4\times4}$ using manual correspondence based point matching.

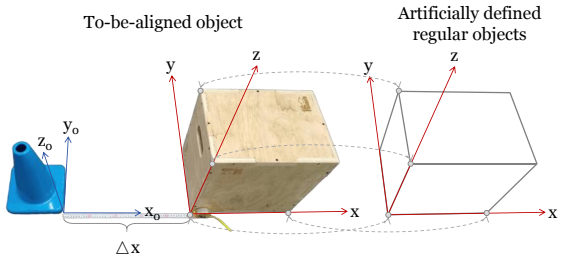


Fig. 4: **Coordinate Alignment.** We first compute the transformation for a regular object and then apply this transformation to the irregular object.

Specifically, as shown in Figure 4, we use objects with regular shapes to transform coordinate systems. For instance, to obtain the transformation matrix for the blue cone, we first manually define a rectangular block $\mathcal{B}$ with the correct coordinate system and size. Then, using the four-point registration method, we align the block's coordinates with the reconstructed COLMAP system. The transformation matrix T_{block} is determined as:

$$T_{\text{block}} = \arg\min_{T} \sum_{i=1}^{4} \|T \cdot \mathbf{p}_i^{\mathcal{B}} - \mathbf{p}_i^{\text{COLMAP}}\|^2, \tag{7}$$

Finally, the transformation matrix for the cone, T_{cone}, is determined by translating the cone's position relative to the aligned rectangular block:

$$T_{\text{cone}} = T_{\text{block}} \cdot \Delta\mathbf{p}_{\text{cone}}. \tag{8}$$

Gaussian Attributes Adjustment. To further adjust the Gaussian attributes after applying the transformation, we first decompose $T_{\texttt{homo}}$ into its rotation component $R_{\texttt{homo}}$, translation vector $t_{\texttt{homo}}$, and scale factor $s_{\texttt{homo}}$. In addition, the 3D covariance matrix Σ of the Gaussian points is parameterized by a scaling matrix $S \in \mathbb{R}^3$ and a rotation matrix $R \in \mathbb{R}^{3\times3}$, such that $\Sigma = RSS^TR^T$. The mean μ, scaling S and rotation R are adjusted as follows:

$$\mu' = R_{\texttt{homo}}\mu + t_{\texttt{homo}}, \tag{9}$$

$$S' = S + log(s_{\texttt{homo}}), \tag{10}$$

$$R_{\texttt{norm}} = \frac{R_{\texttt{homo}}}{s_{\texttt{homo}}} \quad \Sigma' = R_{\texttt{norm}}\Sigma R_{\texttt{norm}}^T, \tag{11}$$

Since the spherical harmonics (SHs) for the 3D Gaussians are stored in world space (i.e., the COLMAP coordinate system), view-dependent colors change when the Gaussians

rotate. To accommodate different rotations, we first extract the Euler angles α, β, γ from the rotation matrix $R_{\texttt{homo}}$ and construct the corresponding Wigner D-matrix D. We then apply D to rotate the SH coefficients. Formally, each band ℓ of the SH coefficients (of length $2\ell + 1$) transforms as:

$$\mathbf{C}^{(\ell)'} = D_\ell(\alpha, -\beta, \gamma)\mathbf{C}^{(\ell)}. \tag{12}$$

where $\mathbf{C}^{(\ell)}$ is the vector of the spherical-harmonic coefficients for degree ℓ and D_ℓ is the $(2\ell+1)\times(2\ell+1)$ Wigner D-matrix.

Occlusion-Aware Randomization and Composition. To generate 3D object assets for environment randomization, we first capture high-quality multi-view images of real-world objects and reconstruct them using the aforementioned pipeline. With the resulting mesh and 3D Gaussians, we employ an interactive mesh editor to obtain both the 3D bounding box and its corresponding transformation matrix $T_{\texttt{bbox}} \in \mathbb{R}^{4\times4}$. Since the object mesh and the 3D Gaussians share the same COLMAP coordinate system, we use the mesh bounding box to crop the object Gaussians and merge them into the environment coordinate space according to the transformation below:

$$T_{\texttt{obj}}^{\texttt{COLMAP}-\texttt{env}} = T_{\texttt{sim}}^{\texttt{COLMAP}-\texttt{env}} \cdot T_{\texttt{COLMAP}-\texttt{obj}}^{\texttt{sim}} \cdot T_{\texttt{bbox}}. \tag{13}$$

The merged object Gaussians is synchronized with the object mesh in Issac and can be rendered jointly with the environment Gaussians via a volumetric alpha-blending [20] process. By incorporating z-buffering [3] on the opacity attributes of the Gaussian points, our method achieves occlusion-aware scene composition with accurate visibility. This randomization strategy substantially increases the diversity of the environment, thereby enabling more robust and scalable agent training.

C. RL in Reconstructed Simulation

Given the simulation environment generated in Section IV-A, the next step in VR-Robo involves training a robust locomotion policy using reinforcement learning in simulation, as shown in Figure 5. This policy is designed to solve desired tasks across a wide range of configurations and environmental conditions. Due to the limited computational resources for real robots, we adopt a two-level asynchronous control strategy: the high-level policy operates at 5 Hz, while the low-level policy runs at 50 Hz. The high-level policy communicates with the low-level policy via velocity commands, represented as $V_{\text{cmd}} = (V_x, V_y, V_{yaw})$. We take our experiments mainly on Unitree Go2 quadruped robot.

Low-Level Policy. The first step is to train the low-level policy, which is similar to previous works [47, 56]. The low-level policy takes $V_{\text{cmd}} = (V_x, V_y, V_{\text{yaw}})$ and robot proprioception as input and outputs the desired joint positions. The actor network is a simple LSTM combined with MLP layers. Details regarding state and action space, reward design, network architecture, and terrain settings can be found in Appendix A.

The low-level policy used in this work enables the quadruped robot to climb slopes and stairs (up to 15 cm in height). However, unlike previous parkour works [58, 8, 15], the robot cannot directly climb onto a terrain taller than 30 cm. While it is feasible to train the robot to climb higher terrains using these frameworks, our approach focuses on teaching the robot to recognize and utilize slopes or stairs to reach high platforms. This intuition is particularly meaningful in scenarios where the terrain is very tall, making direct climbing or jumping infeasible for any policy.

High-Level Policy. We freeze the previously trained low-level policy and train the high-level policy independently. The high-level policy is trained using reinforcement learning (PPO) within our GS-mesh hybrid simulation environment.

State Space: The actor input of the high-level policy consists of four components: `1) RGB Image Feature` (I_a): We use a frozen, pre-trained Vision Transformer (ViT) [11] to extract features from the RGB image. ViT's attention mechanism is suitable for the task of finding positions and specific colors. This design can greatly compress the input size and improve the training speed. `2) RGB Command` (C_a): A vector indicating the desired color of the target cone, normalized to $[0, 1]$. For example, $[1, 0, 0]$ represents the red cone. `3) Last Action`: The previous output of the policy. `4) Proprioception` (P_a): Robot's base angular velocity, projected gravity, joint positions, and joint velocities. We use an asymmetric actor-critic structure. The critic's observation includes the actor's observation (with noise removed) as well as the robot's world coordinate position and orientation, and the goal's world coordinate position. This design improves the estimation of value function and enhances the training process.

Action Space: The actor outputs the raw velocity command, $V_{\text{raw\_cmd}} = (V_x, V_y, V_{\text{yaw}})$. This raw command will pass through a $\tanh$ layer and scaled by the velocity range $V_{\text{max\_cmd}} = (V_{\text{max\_x}}, V_{\text{max\_y}}, V_{\text{max\_yaw}})$ to ensure safety. Note that the $\tanh$ layer and velocity range scaling are applied outside the actor network. The resulting velocity command $V_{\text{cmd}} = (V_x, V_y, V_{\text{yaw}})$ is sent to the low-level policy, enabling the robot to move.

Reward Design: The total reward consists of two main categories: **Task Rewards** r_T and **Regularization Rewards** r_R. r_T include `Reach_goal`, `Goal_dis`, `Goal_dis_z`, and `Goal_heading`. To be specific, the robot receives a reward if it comes close enough to the goal.

$$r_{\text{reach\_goal}} = \begin{cases} R_{\max}, & \text{if } \|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{goal}}\|_2 \le \epsilon, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

`Goal_dis` is based on the change in the Euclidean distance to the goal between consecutive time steps:

$$r_{\text{goal\_dis}} = d_{\text{prev}} - d_{\text{current}}, \tag{15}$$

$$d = \|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{goal}}\|_2. \tag{16}$$

`Goal_dis_z` is based on the change in the vertical (z-axis) distance to the goal:

$$r_{\text{goal\_dis\_z}} = d_{z,\text{prev}} - d_{z,\text{current}}, \tag{17}$$

$$d_z = |z_{\text{robot}} - z_{\text{goal}}|. \tag{18}$$

`Goal_heading`: This reward encourages the robot to face the goal by minimizing the yaw angle error, defined as a linear

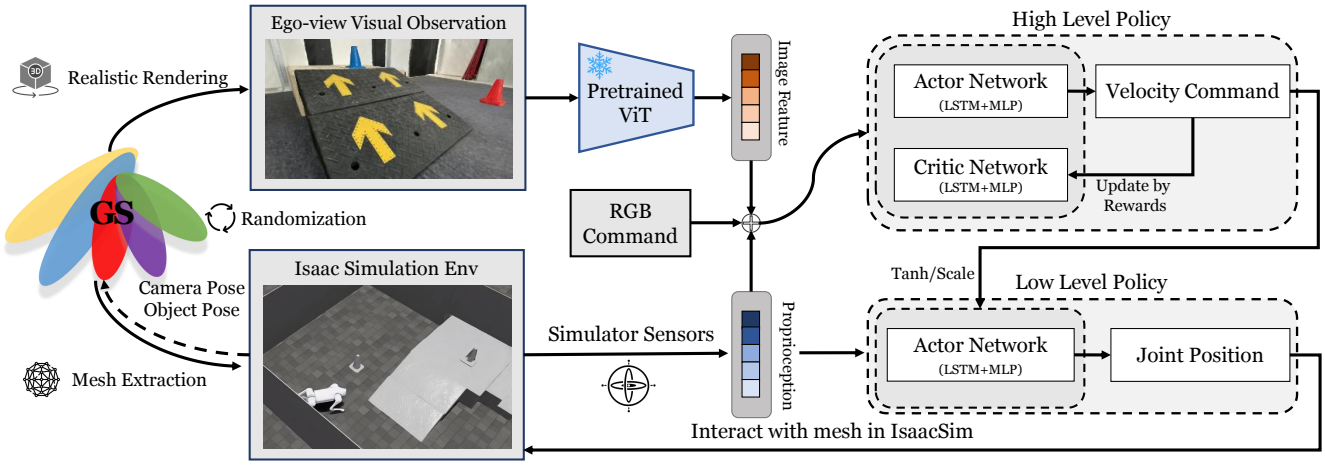


Fig. 5: **RL policy training in the reconstructed simulation environment.** The agent leverages the ego-view GS photorealistic rendering as visual observations and interacts with the mesh extracted from GS in the Issac Sim environment. The agent receives the RGB image feature from ViT [11] encoder, proprioception from simulator sensors, and a task-specific RGB command as input, using an asymmetric actor-critic LSTM structure to output the velocity command for low-level policy control.

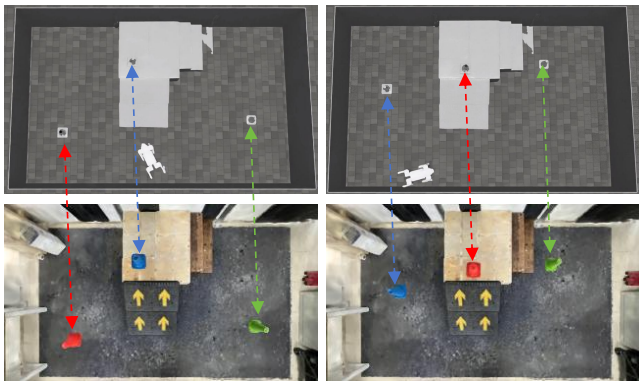


Fig. 6: **Agent-object Randomization and Scene Composition.** At the beginning of each episode, we randomly sample the mesh positions for the robot and three cones in the Isaac Sim environment (upper row). We synchronously merge the agent and object Gaussians into the environment and compose for joint rendering (lower row). Both mesh and Gaussian rendering results shown here are in the Bird's-Eye View (BEV) with the same camera pose.

function of the yaw difference:

$$r_{\text{goal\_heading}} = -|\psi_{\text{robot}} - \psi_{\text{goal}}|, \tag{19}$$

where ψ_{robot} is the robot's current yaw angle, ψ_{goal} is the desired yaw angle toward the goal.

In addition, regularization rewards are incorporated to further refine the robot's performance and ensure stable and efficient behavior. It contains `Stop_at_goal`, `Track_lin_vel`, `Track_ang_vel`, and `Action_l2`. The detailed mathematical expressions and weights of all rewards can be found in Appendix B.

Training Process: We randomly sample the positions and orientations of the robot and cones at the beginning of each episode. The robot's camera poses and cone poses are obtained from IsaacSim and sent to the aligned and editable 3D Gaussians mentioned above to render a photorealistic image. The policy then outputs an action based on this image, which is applied in Isaac Sim to interact with the mesh.

Domain Randomization: We apply domain randomization to reduce the sim-to-real gap. This includes three key components: `1) Camera Pose Noise`: Before requesting the 3D Gaussians to render an image, we add uniform noise to the camera pose extracted from Isaac Sim. Specifically, we apply uniform noise of scale $[1\text{ cm}, 1\text{ cm}, 1\text{ cm}]$ in $[x, y, z]$ and $[1°, 1°, 2°]$ in $[\text{roll}, \text{pitch}, \text{yaw}]$. `2) Image Augmentation`: We randomly apply brightness, contrast, saturation, and hue adjustments to the image. This is followed by Gaussian blur to simulate camera blur caused by robot movement. Additionally, we randomly add Gaussian noise to the image with a small probability ($p = 0.05$). `3) Image Delay`: We randomly apply a 0- or 1-step delay to the rendered images.

V. Experiments

To evaluate the effectiveness of VR-Robo, we curate two real2sim environments by reconstructing 3D scenes from phone-captured image collections. These two environments capture an indoor scene with complex terrains. Our experiments are designed to evaluate two main aspects of VR-Robo: 1) its ability to reconstruct high-quality 3D environments with mesh-GS hybrid representation and support the training of locomotion policy with visual observation and mesh interaction; 2) its capability to minimize the sim2real gap when deployed to the real world. We conducted extensive experiments in both simulated and real-world environments.

A. Experiment Setup

Real-to-Sim Reconstruction. We reconstruct two distinct indoor room environments, each characterized by specific terrains. We also randomize the environments by incorporating three cones colored red, green, and blue. We use iPads or

TABLE I: Comparison and ablation experimental results in the real-world setting.

Method	Exteroception	Success Rate ↑			Average Reaching Time (s) ↓		
		Easy	Medium	Hard	Easy	Medium	Hard
Ours	RGB	**100.00%**	**93.33%**	**100.00%**	**4.96**	**6.28**	**9.09**
Imitation Learning (IL)	RGB	0.00%	0.00%	0.00%	15.00	15.00	15.00
Depth Policy	Depth	6.67%	0.00%	0.00%	14.33	15.00	15.00
SARO [57]	RGB	66.67%	26.67%	0.00%	46.49	57.24	60.00
Texture Mesh	RGB	20.00%	6.67%	0.00%	12.90	14.90	15.00
CNN Encoder	RGB	73.33%	66.67%	6.67%	9.10	11.41	14.90
w/o Domain Randomization	RGB	53.33%	6.67%	0.00%	10.04	14.76	15.00

iPhones to take photos, which are easily available. Detailed procedures for image acquisition and coordinate alignment are provided in the Appendix C.

Simulated Locomotion Training. We use IsaacSim on a single 48GB NVIDIA RTX 4090D GPU for policy training. For the low-level policy, we deploy 4,096 quadruped robot agents for parallel training. The training process consists of 80,000 iterations from scratch. The low-level policy operates at a frequency of 50 Hz. For the high-level policy, we deploy 64 quadruped robot agents and train for 8,000 iterations from scratch. The high-level policy operates at a frequency of 5 Hz. The entire training process takes approximately 3 days to complete. We use the "vit_tiny_patch16_224" model as our ViT vision encoder, removing the final classification head. The IsaacSim simulator and the 3DGS renderer are connected via TCP network.

Sim-to-Real Deployment. We deploy our policy on the Unitree Go2 quadruped robot, which is equipped with an NVIDIA Jetson Orin Nano (40 TOPS) as the onboard computer. We use ROS2 [27] for communication between the high-level policy, low-level policy, and the robot. Both policies run onboard, with the high-level policy operating at 5 Hz and the low-level policy running at 50 Hz. The robot receives desired joint positions from the low-level policy for PD control ($K_p = 40.0$, $K_d = 1.0$). We use an Insta360 Ace camera to capture RGB images. The camera captures images at a resolution of 320×180. After calibration, it has a horizontal field of view (FOVX) of 1.5701 radians and a vertical field of view (FOVY) of 1.0260 radians.

B. Simulation Experiments

We conduct comparison and ablation experiments with various baselines. For comparison experiments,

- **Imitation Learning:** We teleoperate the robot to collect 60 different trajectories in the real world and train the policy using regression optimization.
- **Depth Policy:** We replace the RGB image with a depth image as the policy input and change the vision encoder to a simple CNN, following previous parkour work[58].

For ablation experiments,

- **Texture Mesh:** We integrate SuGaR [14]-reconstructed texture mesh into IssacSim and directly render the RGB observation.
- **CNN Encoder:** We replace the ViT vision encoder with a CNN image encoder, specifically "MobilenetV3" [16], and remove the last classification layer.

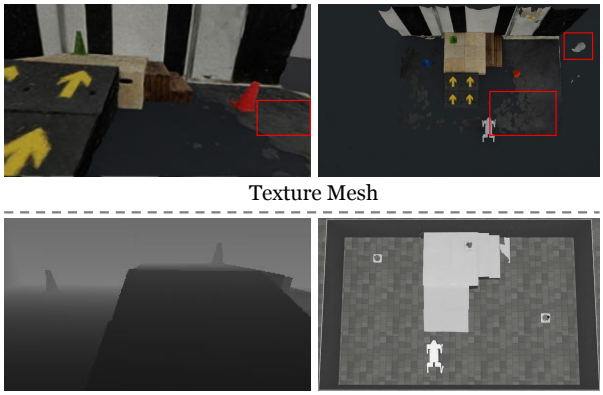


Fig. 7: **Illustration of the Texture Mesh and Depth Input Images.** (Left: policy ego-centric image input, right: simulation bird-eye-view). The mesh is cropped due to the original size being too large for the simulator. As shown in the area marked by the red box, the mesh exhibits bulges in texture-less regions, such as the ground and walls, which hinder the robot's movement. Additionally, controlling the lighting in the simulator poses a significant challenge. The depth map is rendered directly from the simulator but cannot differentiate the color of the cones.

We conduct all the experiments on the task of reaching the red cone in Scene 1. The red cone can be sampled in three areas: left (on the ground), middle (on the terrain), and right (on the ground). We randomly generate 150 sets of different cone positions. Note that we determine these tasks before conducting any experiments and maintain the same setup for all methods to ensure consistency.

For evaluation metrics, we report the success rate (SR) and average reaching time (ART). An episode is considered successful if the robot reaches within 0.25 m of the target cone at least once within the maximum time of 15 seconds. If the robot successfully reaches the target cone, we record the reaching time. If not, the reaching time for that episode is recorded as the maximum time of 15 seconds. Finally, we compute the average of the reaching time.

TABLE II: Comparison results in the simulation setting.

Method	SR ↑	ART (s) ↓
Ours	**100.00%**	**4.94**
Imitation Learning (IL)	8.67%	14.01
Depth Policy	6.00%	14.13

TABLE III: Ablation results in the simulation setting.

Method	SR ↑	ART (s) ↓
Ours	**100.00%**	**4.94**
Texture Mesh	22.00%	12.73
CNN Encoder	54.67%	10.04

The results are shown in Table II and Table III. It demonstrates that our method achieves consistently better performance across all evaluation metrics. Imitation learning suffers from a lack of data samples and cannot learn a policy through exploration. The depth policy fails to distinguish between different cones and cannot be effectively trained, resulting in behavior akin to random actions. Similarly, the texture mesh is unable to generate realistic images, and the CNN encoder struggles to extract precise features from the images. These findings collectively confirm the effectiveness and robustness of our proposed approach in diverse experimental setups.

C. Real-World Experiments

We further conduct two baseline methods for real-world experiments in addition to the method mentioned above (these can only be compared in the real world):

- **SARO [57]:** SARO enables the robot to navigate across 3D terrains leveraging the vision-language model (VLM).
- **w/o Domain Randomization:** We exclude domain randomization as described in Section IV-C.

We also conduct experiments to find the red cone in Scene 1. We categorize the task into three levels of difficulty based on the robot's starting position: easy, medium, and hard. For easy tasks, the robot starts with the target cone directly visible. For medium tasks, the robot needs to turn to a certain degree to see the target cone. For hard tasks, the robot starts very far from the cone and faces a nearly opposite direction to the target cone (an image example is needed). We randomly select 3 sets of cone positions, with each set containing one easy, one medium, and one hard task, as shown in Figure 8. For each task, we repeat the experiment 5 times. We then calculate the success rate and average reaching time. For SARO, the maximum time is extended to 60 seconds due to the long inference time of the Vision-Language Model (VLM). In the real robot experiments, we consider the task successful if the robot makes contact with the target cone.

The quantitative results are shown in Table I. Our method achieves the highest success rates across all difficulty levels. Furthermore, it consistently records the shortest average reaching times, demonstrating its efficiency. In addition to the methods mentioned above, SARO achieves moderate success on easy tasks (66.67%) but performs poorly on medium and hard tasks. This is primarily due to the lack of historical context and the long inference time in the VLM framework. If the robot cannot see the goal initially, it is likely to fail. Additionally, the absence of domain randomization significantly reduces the model's effectiveness, primarily due to the sim-to-real gap in RGB observations.

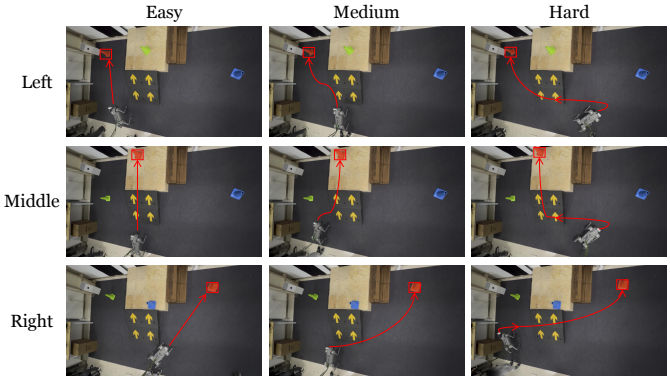


Fig. 8: Tasks of different difficulties (easy, medium, and hard) and different positions (left, middle, and right).

We further conduct qualitative experiments under varying conditions, including various target cone colors, changes in lighting conditions, different scenes, random interference, background variations, and training on the Unitree G1 humanoid robot, as shown in Figure 9. The detailed demo can be found in the supplementary materials. These experiments demonstrate the robustness of our method, showcasing its ability to adapt effectively to a wide range of environments.

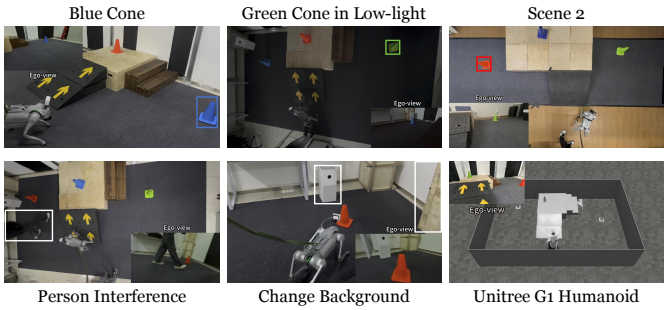


Fig. 9: qualitative experiments under varying conditions.

VI. Limitations and Conclusion

A. Limitations

1) Simple Scene and Task: while VR-Robo facilitates easier real-to-sim transfer through its GS-mesh hybrid representation and coordinate alignment workflow, our current simulation environment is restricted to indoor scenes with static terrains. Future work could explore the simulation of large-scale and complex scenarios, such as factory settings or outdoor street environments, incorporating both static and dynamic obstacles to support a wider range of downstream tasks. **2) Workflow Efficiency**: despite the accelerated visual locomotion policy training afforded by the real-time rendering capabilities of 3DGS, training a large number of agents in parallel remains relatively slow due to the serial rendering pipeline across agents and the significant communication costs

between the simulator and the GS renderer. This also limits training on finding cones of different colors at the same time. Future efforts may focus on designing CUDA-based parallel randomization and rendering pipelines, as well as more efficient interaction methods between the GS and the simulator. **3) Model Pretraining**: as discussed in subsection V-A, VR-Robo currently requires approximately three days to train a reinforcement learning (RL) locomotion policy from scratch for each task. We anticipate that more effective pretraining methods could mitigate this time bottleneck.

B. Conclusion

This work introduces VR-Robo, a real-to-sim-to-real system designed to train visual locomotion policies within a photorealistic and physically interactive simulation environment, thereby minimizing the sim-to-real gap. Extensive experimental results demonstrate that by integrating real-world environments into the simulator with randomization, agents successfully learn robust and effective policies for challenging high-level tasks and can be zero-shot deployed to real-world scenarios. In future work, we aim to extend our framework to more large-scale and complex environments, incorporating diverse agents such as humanoid robots.

References

[1] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39 (1):3–20, 2020.

[2] Arunkumar Byravan, Jan Humplik, Leonard Hasenclever, Arthur Brussee, Francesco Nori, Tuomas Haarnoja, Ben Moran, Steven Bohez, Fereshteh Sadeghi, Bojan Vujatovic, et al. Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9362–9369. IEEE, 2023.

[3] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces*. The University of Utah, 1974.

[4] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[5] Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *arXiv preprint arXiv:2406.06521*, 2024.

[6] Hanlin Chen, Chen Li, and Gim Hee Lee. Neusg: Neural implicit surface reconstruction with 3d gaussian splatting guidance. *arXiv preprint arXiv:2312.00846*, 2023.

[7] Zoey Chen, Zhao Mandi, Homanga Bharadhwaj, Mohit Sharma, Shuran Song, Abhishek Gupta, and Vikash Kumar. Semantically controllable augmentations for generalizable robot learning. *The International Journal of Robotics Research*, page 02783649241273686, 2024.

[8] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11443–11450. IEEE, 2024.

[9] Jaeyoung Chung, Jeongtaek Oh, and Kyoung Mu Lee. Depth-regularized optimization for 3d gaussian splatting in few-shot images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 811–820, 2024.

[10] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[11] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] Qiancheng Fu, Qingshan Xu, Yew Soon Ong, and Wenbing Tao. Geo-neus: Geometry-consistent neural implicit surfaces learning for multi-view reconstruction. *Advances in Neural Information Processing Systems*, 35: 3403–3416, 2022.

[13] Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

[14] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024.

[15] David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024.

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[17] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. *arXiv preprint arXiv:2403.17888*, 2024.

[18] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[19] Yufei Jia, Guangyu Wang, Yuhang Dong, Junzhe Wu, Yupei Zeng, Haizhou Ge, Kairui Ding, Zike Yan, Weibin Gu, Chuxuan Li, Ziming Wang, Yunjie Cheng, Wei Sui, Ruqi Huang, and Guyue Zhou. Discoverse: Efficient robot simulation in complex high-fidelity environments, 2024. URL https://air-discoverse.github.io/.

[20] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

[21] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1440–1449, 2021.

[22] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021.

[23] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In

Conference on Robot Learning, pages 80–93. PMLR, 2023.

[24] Xiaofan Li, Yifu Zhang, and Xiaoqing Ye Drivingdiffusion. Layout-guided multi-view driving scene video generation with latent diffusion model. *arXiv preprint arXiv:2310.07771*, 2(3), 2023.

[25] Xinhai Li, Jialin Li, Ziheng Zhang, Rui Zhang, Fan Jia, Tiancai Wang, Haoqiang Fan, Kuo-Kun Tseng, and Ruiping Wang. Robogsim: A real2sim2real robotic gaussian splatting simulator. *arXiv preprint arXiv:2411.11839*, 2024.

[26] Haozhe Lou, Yurong Liu, Yike Pan, Yiran Geng, Jianteng Chen, Wenlong Ma, Chenglong Li, Lin Wang, Hengzhen Feng, Lu Shi, et al. Robo-gs: A physics consistent spatial-temporal model for robotic arm with hybrid representation. *arXiv preprint arXiv:2408.14873*, 2024.

[27] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022.

[28] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[29] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[30] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6): 3740–3747, 2023.

[31] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[32] Alex Quach, Makram Chahine, Alexander Amini, Ramin Hasani, and Daniela Rus. Gaussian splatting to real world flight navigation transfer with liquid networks. *arXiv preprint arXiv:2406.15149*, 2024.

[33] Mohammad Nomaan Qureshi, Sparsh Garg, Francisco Yandun, David Held, George Kantor, and Abhisesh Silwal. Splatsim: Zero-shot sim2real transfer of rgb manipulation policies using gaussian splatting. *arXiv preprint arXiv:2409.10161*, 2024.

[34] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[35] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[36] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

[37] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in neural information processing systems*, 34:251–266, 2021.

[38] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[39] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[40] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

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

[42] Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *arXiv preprint arXiv:2403.03949*, 2024.

[43] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018.

[44] Matias Turkulainen, Xuqian Ren, Iaroslav Melekhov, Otto Seiskari, Esa Rahtu, and Juho Kannala. Dn-splatter: Depth and normal priors for gaussian splatting and meshing. *arXiv preprint arXiv:2403.17822*, 2024.

[45] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023.

[46] Xiaofeng Wang, Zheng Zhu, Guan Huang, Xinze Chen, Jiagang Zhu, and Jiwen Lu. Drivedreamer: Towards

real-world-driven world models for autonomous driving. *arXiv preprint arXiv:2309.09777*, 2023.

[47] Jinze Wu, Guiyang Xin, Chenkun Qi, and Yufei Xue. Learning robust and agile legged locomotion using adversarial motion priors. *IEEE Robotics and Automation Letters*, 2023.

[48] Yuxuan Wu, Lei Pan, Wenhua Wu, Guangming Wang, Yanzi Miao, and Hesheng Wang. Rl-gsbridge: 3d gaussian splatting based real2sim2real method for robotic manipulation learning. *arXiv preprint arXiv:2409.20291*, 2024.

[49] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020.

[50] Zhaoming Xie, Xingye Da, Michiel Van de Panne, Buck Babich, and Animesh Garg. Dynamics randomization revisited: A case study for quadrupedal locomotion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4955–4961. IEEE, 2021.

[51] Ziyang Xie, Zhizheng Liu, Zhenghao Peng, Wayne Wu, and Bolei Zhou. Vid2sim: Realistic and interactive simulation from video for urban navigation. *arXiv preprint arXiv:2501.06693*, 2025.

[52] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv preprint arXiv:2406.09414*, 2024.

[53] Alan Yu, Ge Yang, Ran Choi, Yajvan Ravan, John Leonard, and Phillip Isola. Learning visual parkour from generated images. In *8th Annual Conference on Robot Learning*, 2024.

[54] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Jodilyn Peralta, Brian Ichter, et al. Scaling robot learning with semantically imagined experience. *arXiv preprint arXiv:2302.11550*, 2023.

[55] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Advances in neural information processing systems*, 35:25018–25032, 2022.

[56] Shaoting Zhu, Runhan Huang, Linzhan Mou, and Hang Zhao. Robust robot walker: Learning agile locomotion over tiny traps. *arXiv preprint arXiv:2409.07409*, 2024.

[57] Shaoting Zhu, Derun Li, Linzhan Mou, Yong Liu, Ningyi Xu, and Hang Zhao. Saro: Space-aware robot system for terrain crossing via vision-language model. *arXiv preprint arXiv:2407.16412*, 2024.

[58] Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023.

[59] Ziwen Zhuang, Shenzhe Yao, and Hang Zhao. Humanoid parkour learning. *arXiv preprint arXiv:2406.10759*, 2024.

APPENDIX

A. Details of Low-Level Policy

State Space. The observation of low-level policy using only proprioception $\boldsymbol{p}_t$ and velocity command $\boldsymbol{v}_t$. Proprioception $\boldsymbol{p}_t$ contains gravity vector and base angular velocity from IMU, joint positions, joint velocities, and last action. Velocity command is the desired $\{V_{\text{x}}, V_{\text{y}}, V_{\text{yaw}}\}$ of the robot.

Action Space. The action space $\boldsymbol{a}_t \in \mathbb{R}^{12}$ is the desired joint positions of 12 joints.

Reward Design. The reward function consists of four components: task reward r_t^T, regularization reward r_t^R, style reward r_t^S, and alive reward r_t^A. The total reward is the sum of these components, given by $r_t = r_t^T + r_t^R + r_t^S + r_t^A$. The detailed formulas are provided in Table IV.

Network Architecture. Both the actor and critic use a simple LSTM network followed by an MLP.

Terrain Settings. We adopt the "terrain curriculum" approach as proposed in [34]. Specifically, we design a total of 200 unique terrains, arranged in a 10×20 grid layout. These terrains are categorized into 6 distinct types, each containing 20 variations that progressively increase in difficulty. Each terrain spans an area of 8×8 meters.

B. Details of High-Level Policy

Reward Design. The total reward consists of two main categories: **Task Rewards** r_T and **Regularization Rewards** r_R. In addition to rask rewards r_T in Section IV-C, Regularization rewards r_R include:

`Stop_at_goal`: Penalize the velocity command (action) when the robot is near the goal:

$$r_{\text{stop\_at\_goal}} = -\|\mathbf{a}\|^2 \cdot (\|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{goal}}\|_2 \leq \epsilon). \tag{20}$$

`Track_lin_vel`: The exponential error between the robot base's linear velocity and the velocity command. This encourages the robot to avoid getting stuck:

$$r_{\text{track\_lin\_vel}} = \exp\left(-\|\mathbf{v}_{\text{robot}} - \mathbf{v}_{\text{command}}\|^2 / 0.25\right), \tag{21}$$

where $\mathbf{v}_{\text{robot}}$ is the robot's linear velocity and $\mathbf{v}_{\text{command}}$ is the commanded linear velocity.

`Track_ang_vel`: The exponential error between the robot base's angular velocity and the commanded angular velocity. This helps the robot to avoid getting stuck:

$$r_{\text{track\_ang\_vel}} = \exp\left(-\|\boldsymbol{\omega}_{\text{robot}} - \boldsymbol{\omega}_{\text{command}}\|^2 / 0.25\right), \tag{22}$$

where $\boldsymbol{\omega}_{\text{robot}}$ is the robot's angular velocity and $\boldsymbol{\omega}_{\text{command}}$ is the commanded angular velocity.

`Action_l2`: Penalizes the L2 norm of the action to prevent excessively large actions and improve training stability:

$$r_{\text{action\_l2}} = -\|\mathbf{a}\|^2, \tag{23}$$

where $\mathbf{a}$ represents the action vector, and λ_a is a scaling coefficient.

All of the rewards and their weights are in Table VII.

C. Real-to-Sim Reconstruction Details

We have designed a set of efficient shooting methods tailored for indoor scenes with simple terrain, as shown in Figure 10. These methods allow a moderately trained operator to complete the entire process of shooting, reconstruction, and calibration within 3 hours. The shooting process consists of four predefined routes: one inner circle and three outer circles. The camera orientations for each route are illustrated in the figure. Each route includes 20 evenly spaced nodes, and at each node, photos are captured from 5 distinct camera poses. Consequently, the entire process involves capturing approximately 400 photos, ensuring sufficient coverage for accurate reconstruction.

TABLE IV: Reward Function

Type	Item	Formula	Weight
Task	Lin vel tracking	$\exp\left(-\lVert \mathbf{v}_{t,xy}^{\text{des}} - \mathbf{v}_{t,xy} \rVert_2 / 0.25\right)$	1.5
	Ang vel tracking	$\exp\left(-\lVert \omega_{t,z}^{\text{des}} - \omega_{t,z} \rVert_2 / 0.25\right)$	0.75
Regularizations	Lin vel (z)	$\lVert \mathbf{v}_{t,z} \rVert_2^2$	-2.0
	Ang vel (x, y)	$\lVert \omega_{t,x} \rVert_2^2 + \lVert \omega_{t,y} \rVert_2^2$	-0.05
	Joint vel	$\lVert \dot{\mathbf{q}} \rVert_2^2$	-3×10^{-4}
	Joint acc	$\lVert \ddot{\mathbf{q}} \rVert_2^2$	-2.5×10^{-7}
	Joint torque	$\lVert \tau \rVert_2^2$	-2×10^{-4}
	Joint pos limits	$\sum \max(0, \lvert \mathbf{q} \rvert - q_{\text{lim}})$	-0.75
	Joint vel limits	$\sum \max(0, \lvert \dot{\mathbf{q}} \rvert - \dot{q}_{\text{lim}})$	-0.02
Style	Feet in air	$\sum_{i=0}^{3} (\mathbf{t}_{air,i} - 0.3) + 10 \cdot \min(0.5 - \mathbf{t}_{air,i}, 0)$	0.05
	Stand still	$(\sum_{i=1}^{12} \lvert q - q_{default} \rvert) \cdot (\lVert V \rVert_2 < 0.1)$	-0.5
	Balance	$\lVert F_{feet,0} + F_{feet,2} - F_{feet,1} - F_{feet,3} \rVert_2$	-2×10^{-5}
Alive	Alive	1	3.0

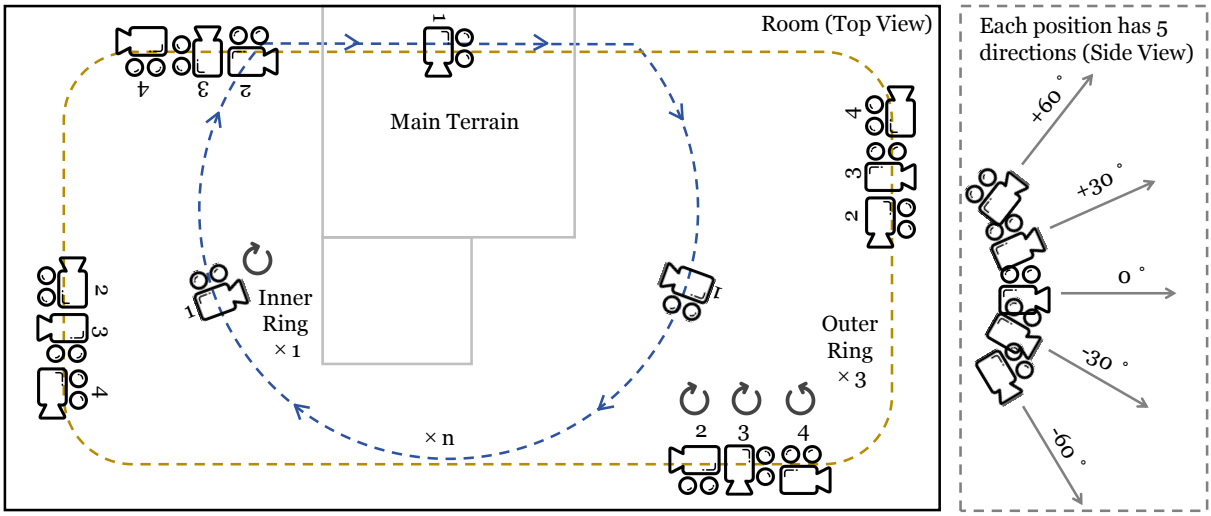


Fig. 10: **Illumination of data collection pipeline for indoor scene reconstruction.**

TABLE V: Network architecture details

Network	Type	Input	Hidden layers	Output
Actor RNN	LSTM+MLP	$\boldsymbol{p}_t$, $\boldsymbol{v}_t$	[512, 256]	$\boldsymbol{a}_t$
Critic RNN	LSTM+MLP	$\boldsymbol{p}_t$, $\hat{\boldsymbol{v}}_t$	[512, 256]	$\boldsymbol{V}_t$

TABLE VI: Terrain Categories and Specifications

Category	Key Parameters	Columns
Ascending stairs	Step height: 0.05 $\sim$ 0.2 m	4
Descending stairs	Step height: 0.05 $\sim$ 0.2 m	4
Ascending ramps	Inclination angle: 0° $\sim$ 45°	2
Descending ramps	Inclination angle: 0° $\sim$ 45°	2
Random boxes	Box height: 0.025 $\sim$ 0.1 m	4
Rough terrain	Noise amplitude: 0.01 $\sim$ 0.06 m	4

TABLE VII: Reward Function

Type	Item	Formula	Weight
Task	Reach goal	$r_{\text{reach\_goal}} = \begin{cases} 1, & \text{if } \|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{goal}}\|_2 \leq \epsilon, \\ 0, & \text{otherwise.} \end{cases}$	0.5
	Goal distance	$r_{\text{goal\_dis}} = d_{\text{prev}} - d_{\text{current}},\ d = \|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{goal}}\|_2$	5.0
	Goal distance (z-axis)	$r_{\text{goal\_dis\_z}} = d_{z,\text{prev}} - d_{z,\text{current}},\ d_z = \|z_{\text{robot}} - z_{\text{goal}}\|$	30.0
	Goal heading	$r_{\text{goal\_heading}} = -\|\psi_{\text{robot}} - \psi_{\text{goal}}\|$	0.3
Regularizations	Stop at goal	$r_{\text{stop\_at\_goal}} = -\|\mathbf{a}\|^2 \cdot (\|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{goal}}\|_2 \leq \epsilon)$	1.0
	Linear velocity tracking	$r_{\text{track\_lin\_vel}} = \exp\left(-\|\mathbf{v}_{\text{robot}} - \mathbf{v}_{\text{command}}\|^2/0.25\right)$	0.2
	Angular velocity tracking	$r_{\text{track\_ang\_vel}} = \exp\left(-\|\boldsymbol{\omega}_{\text{robot}} - \boldsymbol{\omega}_{\text{command}}\|^2/0.25\right)$	0.2
	Action L2 norm	$r_{\text{action\_l2}} = -\|\mathbf{a}\|^2$	0.002