

# Gallant: Voxel Grid-based Humanoid Locomotion and Local-navigation across 3D Constrained Terrains

## Supplementary Material

### 1. Real-world deployment Details

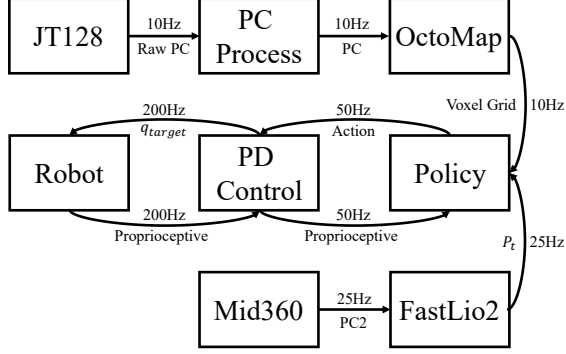


Figure 8. Diagram of information communication.

**Target Position Command** We use a Unitree G1 robot equipped with a Livox Mid360 LiDAR mounted on its head to run FastLIO2. The Mid360 is installed in a downward-facing orientation and provides a field of view of  $360^\circ$  horizontally and from  $-7^\circ$  to  $52^\circ$  vertically. The system provides the robot’s position in the world coordinate frame at a frequency of 25 Hz. To match this in simulation, the observation frequency of  $P_t$  during training is also set to 25 Hz. To align with our training setup, we initialize the robot’s starting position at  $(0, 0)$  and set the goal position for each run to  $(4, 0)$ . At each time step, FastLIO2 outputs the current position of the robot  $(x, y)$ , and the observation relative to the goal is defined as:  $P_t = (4, 0) - (x, y) = (4 - x, -y)$ .

**Voxel Grid Processing** We use two Hesai JT128 LiDARs mounted at the front and rear of the robot to collect raw point cloud data, which are merged and used for voxel grid construction. The JT128 supports 10 Hz and 20 Hz output modes; empirical testing showed that 20 Hz leads to lower point cloud quality and reduced policy success rates. Therefore, we adopt the more reliable 10 Hz mode and align the simulation accordingly. Each JT128 provides a vertical field of view of approximately  $95^\circ$ , a full  $360^\circ$  horizontal view, and 128 channels. The dual-sensor setup ensures near-complete coverage around the robot. To improve voxel grid quality, the raw point clouds are processed with the Octomap [16] before being passed to the policy. In practice, using Octomap consistently leads to better performance.

**Information Communication** Our system is fully deployed on a Unitree G1 robot using only an NVIDIA Orin

NX, which has limited communication performance. When using TCP to transmit LiDAR data and LCM for internal robot state sharing, we observed a delay of approximately 200 ms in proprioceptive data transmission, which is unacceptable. Thus, we made the following adjustments:

- LiDAR output is clipped to include only points within the voxel grid used for perception, reducing data size.
- The voxel grid from Octomap and that used for observation share memory to avoid redundant transmission.
- Robot state reading and action command delivery are also implemented via shared memory, bypassing LCM.

These optimizations eliminate nearly all communication induced latency, except for inherent sensor delays. Overall information communication process is shown in Fig. 8.

### 2. Training Details

**Hyperparameter** Our training framework is derived from [40], and below is a summary of key PPO hyperparameters used in the training process:

Hyperparameter	Value
Environment number	$1024 \times 8$
Steps per iteration PPO epochs	4
Minibatches	8
Clip range	0.2
Entropy coefficient	0.003
GAE factor $\lambda$	0.95
Discount factor $\gamma$	0.99
Learning rate	$5e^{-4}$

Table 4. Hyperparameters and their values.

**Policy Network Structure** The Actor and Critic in our policy share the same network structure but maintain separate parameters. The shared architecture is illustrated in the block diagram below. To be specific, a two-layer MLP with hidden dimensions of 256 is used to encode non-voxel information (e.g., proprioceptive input). Note that the Critic additionally receives privileged observations, resulting in a slightly higher input dimension. This produces an intermediate feature  $h_{\text{mlp}}$ . In parallel, a three-layer 2D CNN processes the voxel grid input, producing a feature vector  $h_{\text{cnn}}$ . The two features are concatenated and passed through another MLP to produce a 256-dimensional latent representation. This latent vector is then fed into a final MLP:

- The Actor outputs an action vector of dimension 29.
- The Critic outputs a scalar value estimate.

We use the Mish activation function throughout all layers.

MLP:	$h_{\text{mlp}}^{(1)} = \text{Mish}(\text{LN}(W_{\text{mlp},1}x_{\text{mlp}} + b_{\text{mlp},1}))$
	$h_{\text{mlp}} = W_{\text{mlp},2}h_{\text{mlp}}^{(1)} + b_{\text{mlp},2}, \quad \dim(h_{\text{mlp}}) = 256$
CNN:	$z_1 = \text{Mish}(\text{Conv}(x_{\text{cnn}}; C=8, k=3, s=2, p=1))$
	$z_2 = \text{Mish}(\text{Conv}(z_1; C=8, k=3, s=2, p=1))$
	$z_3 = \text{Mish}(\text{Conv}(z_2; C=8, k=3, s=2, p=1))$
	$h_{\text{cnn}}^{\text{flat}} = \text{Flatten}(z_3)$
	$h_{\text{cnn}}^{(1)} = \text{Mish}(\text{LN}(W_{\text{cnn},1}h_{\text{cnn}}^{\text{flat}} + b_{\text{cnn},1}))$
	$h_{\text{cnn}} = W_{\text{cnn},2}h_{\text{cnn}}^{(1)} + b_{\text{cnn},2}, \quad \dim(h_{\text{cnn}}) = 64$
Fusion:	$f = [h_{\text{mlp}}, h_{\text{cnn}}]$
	$h_{\text{out}}^{(1)} = \text{Mish}(f)$
	$h_{\text{out}} = \text{Mish}(W_{\text{out}}h_{\text{out}}^{(1)} + b_{\text{out}}), \quad \dim(h_{\text{out}}) = 256$

**Observation** The composition of the observation is detailed in Sec. 3, and the dimensionality of each observation component at a single time step  $t$  is summarized in Tab. 5. The dimension of  $Height\_Map_t$  shown in Tab. 5 corresponds to its flattened form. Before flattening, it is represented as a  $33 \times 33$  tensor. Specifically, this map captures the local terrain height around the robot, centered at its base, over a rectangular area with  $x \in [-0.8, 0.8]$  m and  $y \in [-0.8, 0.8]$  m. A resolution of  $0.05$  m is used along both axes, resulting in one height ( $z$ ) sample per  $(x, y)$  grid point. This resolution is consistent with that used in the voxel grid. Instead of applying fixed scaling, the observations are processed using a trainable `vecnorm` module before being fed into the policy. Vecnorm is applied in both training and deployment.

Observation Term	Dimension
$P_t$	4
$T_{\text{elapsed},t}$	1
$T_{\text{left},t}$	1
$a_t$	29
$\omega_t$	3
$g_t$	3
$q_t$	29
$\dot{q}_t$	29
$Voxel\_Grid_t$	$[32 \times 32 \times 40]$
$v_t$	3
$Height\_Map_t$	1089

Table 5. Observation terms and their dimensions.

**Reward** Most reward components used in Gallant follow Ben et al. [3], with necessary modifications to support our

target-based formulation. In addition to the sparse target-reaching reward  $r_{\text{reach}}$  introduced in Sec. 3, we incorporate auxiliary shaping terms to improve sample efficiency during early training, as suggested by Rudin et al. [31].

We design the following three general-purpose rewards to encourage effective behavior across a variety of terrain conditions:

- **Directional velocity reward:**

$$r_{\text{velocity\_direction}} = \frac{\mathbf{a}(\mathbf{p}, \mathbf{g}) \cdot \mathbf{v}_t}{\|\mathbf{a}(\mathbf{p}, \mathbf{g}) \cdot \mathbf{v}_t\|_2},$$

where  $\mathbf{v}_t$  is the robot’s instantaneous velocity and  $\mathbf{a}(\mathbf{p}, \mathbf{g})$  is a direction vector incorporating both goal alignment and obstacle avoidance. It is computed as:

$$\mathbf{a}(\mathbf{p}, \mathbf{g}) = \sum_{j \in \mathcal{N}(\mathbf{p}, r)} w_j \mathbf{u}_{r,j} + \kappa \sum_{j \in \mathcal{N}(\mathbf{p}, r)} w_j \gamma_j \mathbf{t}_j,$$

where  $\mathcal{N}(\mathbf{p}, r)$  denotes obstacle points within radius  $r = 1$  m from the robot position  $\mathbf{p}$ ;  $\mathbf{u}_{r,j}$  is the repulsion unit vector from obstacle  $j$  to the robot;  $\mathbf{t}_j$  is a tangential unit vector (left/right) around obstacle  $j$ ;

$$w_j = \frac{\left[ \max \left( 1 - \frac{\max(d_j - 0.2, 0.02)}{0.8}, 0 \right) \right]^2}{\max(d_j - 0.2, 0.02)}$$

is a distance-based weighting factor, and  $\gamma_j = \max(\mathbf{g}^\top \mathbf{d}_j, 0)$  filters obstacles behind the goal direction. We set  $\kappa = 0.8$  to weight the tangential term. This direction computation is only applied to relevant structures such as cylinders in *Forest* and walls in *Door*, and is efficiently parallelized via `warp`.

- **Head height reward:**

$$r_{\text{head\_height}} = \exp(-4(H_{\text{head\_est}} - H_{\text{head}})^2),$$

where  $H_{\text{head\_est}}$  is computed by shifting the robot 0.45 m forward along the direction to the goal, averaging the terrain height within a  $0.5 \times 0.5$  m square, and subtracting a 0.1 m offset. This reward encourages the robot to proactively lower its head to pass under overhead obstacles like ceilings.

- **Foot clearance reward:**

$$r_{\text{foot\_clearance}} = \exp(-4(H_{\text{foot\_est}} - H_{\text{foot}})^2),$$

where  $H_{\text{foot\_est}}$  is calculated similarly by querying terrain 0.5 m ahead of each foot and averaging the height in a square region. Unlike Ben et al. [3], who use terrain height directly under the foot, our design promotes proactive leg lifting over steps or platforms.

All three rewards are geometry-aware and general-purpose. They are computed consistently across all terrains without task-specific tuning and significantly improve the robot’s ability to traverse diverse obstacle configurations.

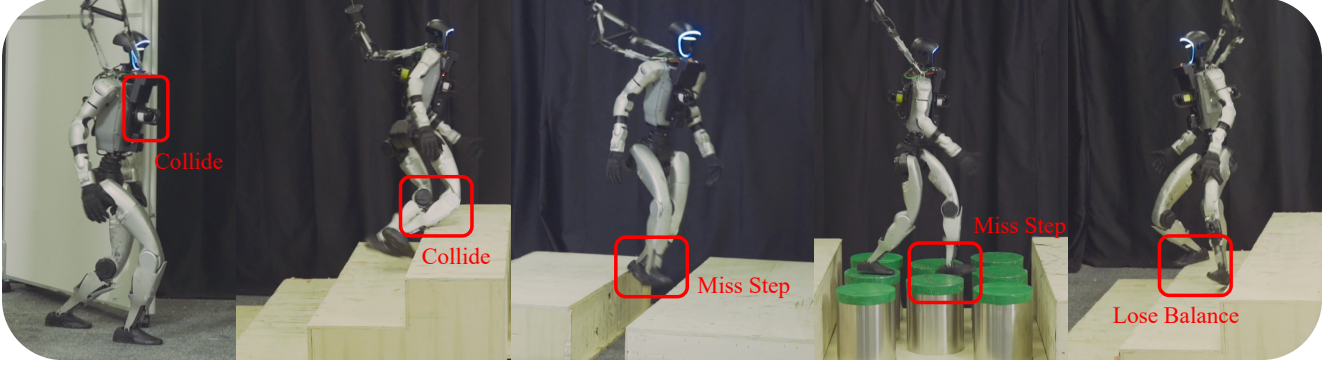


Figure 9. Failure Mode of policy trained by Gallant without LiDAR-related domain randomization.

**Domain Randomization** In addition to the LiDAR-specific domain randomization described in Sec. 3.2, we apply several general randomization strategies during training to improve policy robustness:

- **Mass randomization:** The masses of the pelvis and torso links are randomized as  $m_{\text{rand}} = m \times \mathcal{U}(0.8, 1.2)$ , where  $\mathcal{U}$  denotes a uniform distribution.
- **Foot-ground contact randomization:** While the ground friction coefficient is fixed at 1.0, the foot joint friction is sampled from  $\mathcal{U}(0.5, 2.0)$ , and the restitution coefficient from  $\mathcal{U}(0.05, 0.4)$ .
- **Control parameter randomization:** The joint stiffness and damping parameters are randomized as  $K_{p,\text{rand}} = K_p \times \mathcal{U}(0.8, 1.2)$ ,  $K_{d,\text{rand}} = K_d \times \mathcal{U}(0.8, 1.2)$ , where  $K_p$  and  $K_d$  follow the settings in Liao et al. [19].
- **Torso center-of-mass offset:** The center of mass position of the torso is perturbed by an offset sampled from  $\mathcal{U}(-0.05, 0.05)$  along each axis.
- **Init Joint Position offset:** A random offset sampled from  $\mathcal{U}(-0.1, 0.1)$  is also added to the robot’s default joint positions and default joint velocities (0 rad/s). This perturbation is applied during environment reset to randomize the robot’s initial state.

**Termination** We apply several termination conditions during training to encourage effective and safe behavior:

- **Force contact:** If any external force acting on the torso, hip, or knee joints exceeds 100 N at any timestep, the episode is terminated.
- **Pillar fall:** For pillar-based terrains, if a foot penetrates more than 10 cm below the ground level, the episode is terminated to prevent the robot from bypassing the obstacle by jumping off.
- **No movement:** To prevent the agent from exploiting reward shaping by staying on intermediate platforms, the episode is terminated if the robot fails to move at least 1 m away from its initial position within 4 seconds.
- **Fall over:** The episode terminates when the robot loses balance and falls.

- **Feet too close:** Since self-collision is disabled during training to speed up simulation, this condition prevents the robot’s feet from crossing or overlapping unnaturally.

**Symmetry** Following Ben et al. [3], we apply symmetry-based data augmentation to accelerate training. In addition to flipping the proprioceptive observations as in their method, we also apply a flip along the  $y$ -axis to the perception representation. Specifically, the  $(32, 32, 40)$  grid map is mirrored along the  $y$  dimension to align with the flipped proprioceptive input, forming a consistent flipped observation. The reward remains unchanged under the transformation. Both original and flipped samples are stored together in the rollout buffer and jointly used during training.

**Why Not SLAM-Style Mapping?** While explicit SLAM-style mapping is feasible in isolation, it is impractical for large-scale parallel RL training. On-policy methods such as PPO require thousands of parallel environments to achieve stable training; maintaining per-environment SLAM maps would incur prohibitive GPU memory overhead, severely limiting the number of environments that can run concurrently and ultimately degrading training performance.

### 3. More experiments

**Zero-Shot Deployment on Slopes and Grass.** We evaluate Gallant on terrains not seen during training, including slopes and undulating grass (Fig. 10). The policy generalizes zero-shot to these environments with a 100% success rate. We attribute this to the fact that such terrains do not involve unexpected harsh collisions (i.e., contacts on non-foot links exceeding 100 N); instead, proprioceptive observations alone suffice for stable locomotion despite the coarse 5 cm voxel resolution. **Single-LiDAR Ablation.** We ablate the sensor configuration by comparing Gallant (two panoramic LiDARs + one localization LiDAR) against a single front-facing LiDAR variant (Tab. 6). The single-LiDAR setting lacks rear visibility, leading to degraded per-

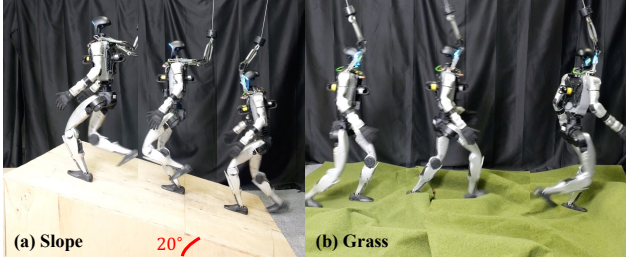


Figure 10. Zero-shot deployment on slope and grass (undulating) terrains not seen during training.

formance in ceiling and door traversal: the robot may prematurely assume it has cleared a ceiling overhang and stand up too early, or fail to determine whether a door has been fully passed, resulting in collisions with obstacles behind it.

Table 6. Comparison between **Gallant** (full sensor suite) and **Single** (front LiDAR only). Each reports successful trials out of 15.

Method	Plane	Ceiling	Door	Platform	Pile	Upstair	Downstair
<b>Gallant</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>14</b>	<b>12</b>	<b>15</b>	<b>14</b>
<b>Single</b>	15	8	10	13	10	15	13

## 4. Failure Mode

In Fig. 9, we illustrate typical failure cases of the NoDR variant (Gallant without domain randomization), as discussed in Sec. 4.3.2. These failures fall into three main categories as listed below:

- **Latency-induced collision:** Due to sensor latency, the robot perceives a voxel grid that reflects the environment state from 100–200 ms earlier. Since the policy is trained in simulation with instantaneous observations, it fails to react proactively and collides with obstacles it believes to be farther away.
- **Missed gap detection:** The robot occasionally fails to detect gaps in time, resulting in missed steps. This issue is particularly pronounced in scenarios like the *Platform* task, where long-range gap perception is essential, leading to a lower success rate for NoDR.
- **Poor state estimation:** The robot exhibits imprecise estimation of its own body state. While it may avoid collisions or missed steps on stairs, it still enters unstable configurations and loses balance.

These observations highlight the importance of domain randomization, especially in simulating LiDAR latency and noise. Without such randomization, the policy fails to generalize effectively to real-world deployments.

## 5. Extensibility.

Although evaluated on locomotion, our framework is task-agnostic and can be directly extended to related problems.

For instance, by replacing the velocity command interface with reference motion targets, the same voxel-based perception pipeline can support whole-body motion tracking in cluttered 3D environments. Beyond LiDAR, the parallel self-scanning technique developed in this work is also applicable to depth camera simulation: by casting rays according to a camera’s intrinsic parameters and mounting pose, the same infrastructure can generate realistic depth images with dynamic robot-body returns at minimal additional cost, enabling vision-based policies to be trained with faithful self-occlusion modeling. Finally, our method is not tied to a specific hardware configuration. By using the voxelized 3D occupancy grid and robot-centric pose as the interface between perception and control, the framework generalizes to any platform capable of providing full-space voxel maps and localization, regardless of the underlying sensor modality (e.g., LiDAR, depth cameras, or fused multi-sensor systems).