# Hiking in the Wild: A Scalable Perceptive Parkour Framework for Humanoids

Author Names Omitted for Anonymous Review. Paper-ID [146]

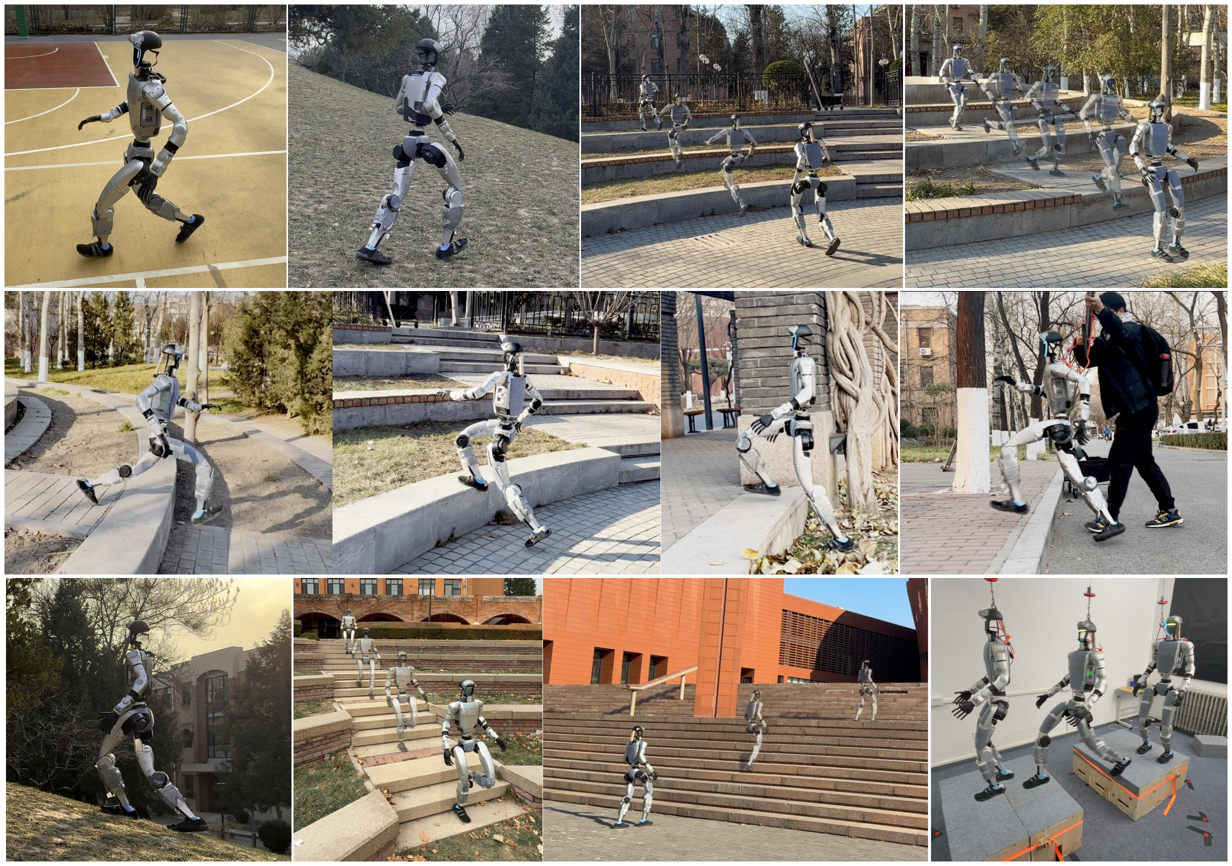

Fig. 1: **Hiking in the Wild.** Our framework enables a humanoid robot to traverse diverse terrains in the wild. The robot can run at a maximum speed of 2.5 m/s over complex terrain. It can also negotiate stairs, gaps, high platforms, and ramps. Relying on depth images for perception, our scalable end-to-end framework achieves robust performance across these scenarios.

***Abstract*—Achieving robust humanoid hiking in complex, unstructured environments requires transitioning from reactive proprioception to proactive perception. However, integrating exteroception remains a significant challenge: existing approaches often fail to achieve high-speed, stable locomotion across diverse, unstructured terrains. In this work, we present *Hiking in the Wild*, a scalable, end-to-end perceptive parkour framework designed for robust humanoid hiking. Our approach utilizes a single-stage reinforcement learning scheme, mapping raw depth inputs and proprioception directly to joint actions, without relying on external state estimation. To ensure safety and training stability, we introduce two key mechanisms: a foothold safety mechanism combining scalable *Terrain Edge Detection* with *Foot Volume Points* to prevent catastrophic slippage on edges, and a *Flat Patch Sampling* strategy that mitigates reward hacking by generating feasible navigation targets. Extensive experiments on a full-size humanoid robot demonstrate that our policy enables robust traversal of complex terrains at speeds up to 2.5 m/s. The training and deployment code will be open-sourced to facilitate further research and deployment on real robots with minimal hardware modifications.**

## I. Introduction

Humanoid robots hold immense promise for traversing complex real-world environments and executing difficult tasks. Unlike their wheeled counterparts, humanoids can step over obstacles and cross discontinuous terrain. Recently, significant strides have been made in humanoid control, enabling robots to perform dynamic maneuvers such as dancing, backflips, and mimicking human motions [23, 25]. However, there is a fundamental distinction between tracking a predefined motion and hiking in the wild. While tracking resembles memorizing a routine, hiking requires the robot to actively perceive the terrain, adapt to irregularities, and handle the unknown.

Over the past few years, blind locomotion has established a robust baseline [13, 34, 33, 38]. Relying solely on pro-

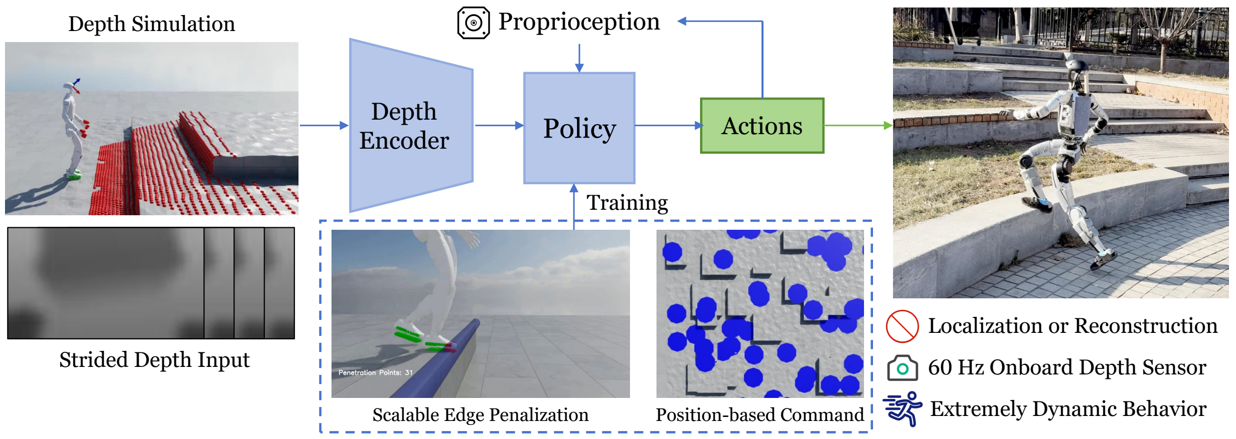
Depth Simulation
Strided Depth Input
Depth Encoder
Proprioception
Policy
Actions
Training
Scalable Edge Penalization
Position-based Command
Localization or Reconstruction
60 Hz Onboard Depth Sensor
Extremely Dynamic Behavior

Fig. 2: System overview. Our framework trains an end-to-end policy using simulated depth and proprioception. To ensure safety and agility on complex terrains, we incorporate Scalable Edge Penalization to avoid risky footholds and Position-based Command generation for precise tracking. The trained policy is directly deployed to the real robot (Zero-shot) using only a 60 Hz onboard depth camera as exteroception, achieving extremely high-dynamic locomotion without explicit localization or map reconstruction.

prioception, these methods are remarkably robust, capable of handling grass or gravel by reacting to contacts. However, the reactive nature of blind locomotion imposes intrinsic limits. Since the robot only responds after a collision, it is vulnerable to significant hazards. Failure to perceive a deep gap or a high step can lead to catastrophic falls. To hike safely, a robot must transition from reactive stability to proactive planning by “looking ahead”.

Integrating exteroceptive perception into locomotion remains a challenge. Existing approaches largely fall into two categories, each with critical bottlenecks. The first category relies on LiDAR to construct elevation maps (2.5D maps) [24, 15] or voxel grids [3], which depend heavily on precise state estimation. In the wild, however, position sensors are prone to drift. Additionally, LiDAR typically has low frequency and suffers from motion distortion, limiting performance in highly dynamic tasks and environments. Other works utilize depth images to reconstruct heightmaps [7, 40, 39], which are not scalable on unseen wild environments. They are limited to low-speed movements on simple structures like planes and stairs. Lastly, due to highly customized configurations (e.g., camera position), the code is often not open-sourced, making these methods difficult to scale and further explore by the community.

To address these challenges, we present *Hiking in the Wild*, a scalable, end-to-end perceptive parkour framework designed for robust humanoid locomotion in unstructured environments. Unlike complex modular pipelines, our approach leverages a single-stage reinforcement learning scheme that maps depth input and proprioception directly to joint actions. To handle the high dimensionality of visual data and the complexity of diverse terrain skills, we incorporate a Mixture-of-Experts (MoE) architecture [17, 16]. Our design allows the policy to capture depth inputs at high frequencies (up to 60 Hz). This high-bandwidth perception loop enables the robot to perform dynamic obstacle traversal and high-speed running (up to 2.5 m/s). Crucially, our system incorporates a realistic depth synthesis module that models sensor noise and artifacts during training, enabling zero-shot Sim-to-Real transfer without external mapping and localization of the environment.

A critical yet often overlooked issue in humanoid RL is the precision of foothold placement. Unlike quadrupeds, humanoids are less stable, and stepping partially on an edge (e.g., the lip of a stair) sometimes leads to catastrophic slippage. Previous model-based planners can generate precise footholds but are fragile to map errors [19]. We propose a robust soft-constraint mechanism: a geometric **Terrain Edge Detector** coupled with **Volume Points** attached to the robot’s feet. Previous approaches using virtual obstacles [51, 4] are limited in scalability, as they rely on manually designed constraints for specific terrain types. Conversely, our approach automatically detects edge features from arbitrary input meshes. By penalizing the penetration of these volume points with terrain edges during training, the policy implicitly learns to center its feet on safe, flat surfaces, significantly enhancing safety on stairs and gaps without requiring explicit trajectory planning.

Furthermore, training robust policies for the wild is complicated by the “reward hacking” phenomenon, where agents tasked with random velocity commands often tend to spin in place rather than traverse difficult terrain [4]. Some methods using a goal-based command alleviate this problem, but they lack control over the speed of the robot [48, 49]. To enforce meaningful exploration, we introduce a **Flat Patch Sampling** strategy. Instead of arbitrary commands, we identify reachable flat regions in the terrain mesh to serve as feasible navigation targets. Velocity commands are then generated based on the relative position of these patches, with randomized speed limits. This curriculum ensures that the agent is always challenged with physically solvable tasks, accelerating convergence and improving directional compliance.

We validate our framework on a humanoid robot through extensive field experiments. The robot successfully completed hiking tasks in the wild, robustly traversing stairs, slopes, uneven grassy ground, and discrete gaps. Our system not only demonstrates the ability to navigate previously unseen environments but also achieves high-speed locomotion agility. The contributions of this work are summarized as follows:

1) A scalable, single-stage end-to-end **perceptive training and deployment framework** based on depth input, enabling zero-shot Sim-to-Real transfer without external state estimation.
2) A novel **safety mechanism** using Terrain Edge Detection and Foot Volume Points, alleviating the precise foothold problem in learning-based control.
3) A **geometry-aware command generation** method that derives velocity commands by pointing towards reachable flat terrain patches, eliminating reward hacking and ensuring the robot actively traverses complex environments.
4) Demonstration of robust wild hiking capabilities in a humanoid. The training and deployment code is going to be **open-sourced to the community**, enabling deployment on real robots with minimal hardware modifications.

## II. Related Works

### A. Learning-based Legged Locomotion

Legged locomotion control has shifted from model-based approaches, such as Model Predictive Control (MPC) [10, 37, 20], to data-driven Reinforcement Learning (RL) to better handle unmodeled dynamics. Early blind RL policies achieved impressive robustness on irregular terrains using only proprioception [34, 38, 13, 47]. However, these methods lack the exteroception required to navigate large obstacles fast and safely.

To enable more agile ability, recent works have integrated exteroceptive perception into the control loop [4, 35]. One category of methods relies on LiDAR to build elevation maps (2.5D) [15, 24, 42, 41] or voxel grids [3]. Nevertheless, these approaches depend heavily on precise localization, which limits their update frequency and overall robustness. Furthermore, LiDAR often suffers from motion distortion during fast movement, restricting these methods to low-speed scenarios. Alternatively, depth-based methods have been proposed. Some of these operate at low frequency [51], while others employ intermediate modules to predict heightmaps [7, 40, 39], potentially compromising performance in high-speed or unseen wild environments. In contrast, our method directly utilizes high-frequency depth images as input to train an end-to-end policy, achieving exceptionally high-dynamic behaviors across complex, unstructured terrains in the wild.

### B. Perceptive Foothold Control

Explicit foothold control typically decouples perception and planning, utilizing terrain representations like elevation maps to solve for optimal placements. Approaches range from heuristic search [45, 8, 18, 21, 6, 28, 1] and learned feasibility costs [27, 46, 29, 44] to rigorous nonlinear optimization [12, 30]. While hybrid frameworks that utilize probabilistic map uncertainty [18, 9] or combine pre-planned references with online tracking [19, 11, 22] have achieved impressive agility, they remain brittle to state-estimation drift and reconstruction artifacts. Recently, end-to-end policies [4] uses edge penalization to learn safe footholds implicitly. However, it is not scalable to arbitrary meshes. Our method takes a trimesh as input and automatically detects the edges, retaining the robustness of the implicit paradigm while improving safety.

## III. Method

### A. Problem Formulation

The problem of perceptive humanoid locomotion is formulated as a Partially Observable Markov Decision Process (POMDP). We leverage Reinforcement Learning (RL) to optimize the locomotion policy. Specifically, the Proximal Policy Optimization (PPO) [36] algorithm is employed for policy training. The key components of our MDP formulation, including the observation space, action space, termination criteria, and reward functions, are defined as follows:

*1) Observation Space:* The observation space is formulated to provide the policy with sufficient state information for stable perceptive locomotion. Specifically, the actor's observation comprises both proprioceptive and perceptive signals, including base angular velocity $\boldsymbol{\omega}_t \in \mathbb{R}^3$, projected gravity vector $\mathbf{g}_t \in \mathbb{R}^3$, velocity commands $\mathbf{c}_t \in \mathbb{R}^3$, joint positions $\mathbf{q}_t \in \mathbb{R}^{29}$, joint velocities $\dot{\mathbf{q}}_t \in \mathbb{R}^{29}$, last action $\mathbf{a}_{t-1} \in \mathbb{R}^{29}$, and depth images $\mathbf{I}_t \in \mathbb{R}^{W\times H}$. To capture temporal dependencies, we employ a sliding window of history consisting of $h$ steps. During training, the actor's input is subject to stochastic noise to improve robustness and bridge the sim-to-real gap. The comprehensive actor observation $\mathbf{o}_t^a$ is defined as:

$$\mathbf{o}_t^a = \{(\boldsymbol{\omega}_i, \mathbf{g}_i, \mathbf{c}_i, \mathbf{q}_i, \dot{\mathbf{q}}_i, \mathbf{a}_{i-1})\}_{i=t-h+1}^{t} + \mathcal{H}_t, \qquad (1)$$

where $\mathcal{H}_t$ is the historical input sequence of depth images. We adopt an asymmetric actor-critic architecture to facilitate simulation training. The critic's observation $\mathbf{o}_t^c$ includes all noise-free actor observations, combined with base linear velocity $\mathbf{v}_t \in \mathbb{R}^3$.

*2) Action Space:* The policy outputs the target joint positions $\mathbf{a}_t \in \mathbb{R}^{29}$. Then, the joint torques $\boldsymbol{\tau}_t$ are computed via PD control:

$$\boldsymbol{\tau}_t = k_p(\mathbf{a}_t - \mathbf{q}_t) - k_d\dot{\mathbf{q}}_t. \qquad (2)$$

The specific PD gain values are adopted from *BeyondMimic* [23]. These computed torques are then applied to the robot's actuators to execute the desired motion.

*3) Termination Criteria:* An episode is terminated upon (i) time-out, (ii) exceeding terrain boundaries, (iii) illegal torso contact, (iv) unstable orientation, or (v) insufficient root height. These constraints prevent the exploration of physically unfeasible states and accelerate training convergence.

*4) Reward Functions:* The total reward $R$ is defined as the sum of four primary components: task, regularization, safety, and AMP-style rewards. Formally, $R = r_{\text{task}} + r_{\text{reg}} + r_{\text{safe}} + r_{\text{amp}}$. These terms respectively facilitate command tracking, energy efficiency, constraint satisfaction, and natural locomotion styles.

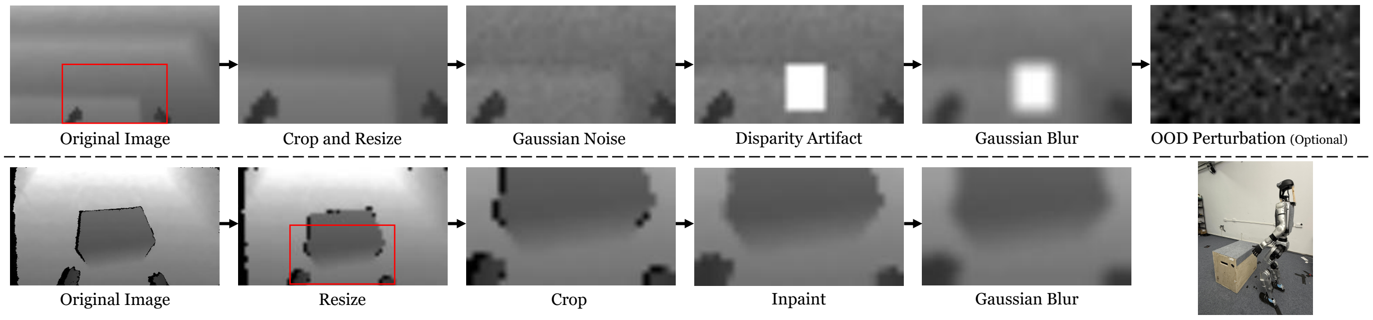

Fig. 3: Depth processing: Top row shows $\mathcal{F}_{sim}$ on synthetic data, and bottom row shows $\mathcal{F}_{real}$ on real-world data.

## *B. Ego-centric Depth Perception*

*1) Efficient Depth Simulation via Parallelized Ray-casting:* To synthesize high-fidelity depth observations, we leverage the NVIDIA Warp framework [26] to implement a GPU-accelerated ray-caster. Given the camera's extrinsic parameters, which comprise the optical center position $\mathbf{p}_c \in \mathbb{R}^3$ and orientation matrix $\mathbf{R}_c \in SO(3)$, we can cast a set of rays corresponding to the camera's intrinsic manifold.

For each pixel $(i, j)$ in the image plane, a ray is emitted in the direction $\mathbf{v}_{i,j}$. The ray-caster computes the radial distance $d_{i,j}$ by identifying the first intersection point between the ray and the scene geometry $\mathcal{G}$, which includes both the terrain environment and the robot's visual meshes:

$$d_{i,j} = \min\{\tau \mid \mathbf{p}_c + \tau\mathbf{v}_{i,j} \cap \mathcal{G} \neq \emptyset\}, \tag{3}$$

To accurately emulate the output of physical RGB-D sensors, we transform the radial distance $d_{i,j}$ into the orthogonal depth $z_{i,j}$. This is achieved by projecting the distance onto the camera's principal axis $\mathbf{n}_c$:

$$z_{i,j} = d_{i,j} \cdot (\mathbf{v}_{i,j} \cdot \mathbf{n}_c). \tag{4}$$

where $\mathbf{v}_{i,j}$ is the unit ray direction and $\mathbf{n}_c$ is the camera's forward-facing unit vector. This parallelized approach ensures real-time synthesis of dense depth maps within the simulation loop.

*2) Bidirectional Depth Alignment and Noise Modeling:* To minimize the sim-to-real gap, we define two transformation pipelines, $\mathcal{F}_{sim}$ and $\mathcal{F}_{real}$, which map raw depth observations from their respective domains into a unified perception space $\mathcal{O}$. Our objective is to ensure that the processed distributions are harmonized, such that $P(\mathcal{F}_{sim}(d_{sim})) \approx P(\mathcal{F}_{real}(d_{real}))$, where $d \in \mathbb{R}^{H\times W}$ denotes the raw depth map.

*a) Simulation Pipeline $\mathcal{F}_{sim}$:* The simulation pipeline degrades the ideal depth to emulate physical sensor limitations through the following sequential stochastic operations:

1) **Crop and Resize**: The raw depth map is cropped and rescaled to the target resolution to focus on the key features at the bottom center of the image.
2) **Range-dependent Gaussian Noise**: To account for precision decay, additive Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is injected into pixels within a valid sensing range $[d_{min}, d_{max}]$. The perturbed depth $z'_{i,j}$ is defined as:

$$z'_{i,j} = \begin{cases} z_{i,j} + \epsilon, & \text{if } z_{i,j} \in [d_{min}, d_{max}] \\ z_{i,j}, & \text{otherwise} \end{cases}. \tag{5}$$

3) **Disparity Artifact Synthesis**: To simulate binocular matching failures (e.g., in over-exposed or textureless regions), we mask contiguous pixels as invalid "white regions" using structural masks.
4) **Gaussian Blur**: A Gaussian blur kernel $\mathbf{K}$ is convolved with the image to simulate optical motion blur.
5) **Clip and Normalization** Values are clipped and normalized to $[0, 1]$.
6) **Out-of-Distribution (OOD) Perturbation**: To enhance robustness against brief obstructions or temporary sensor glitches, we introduce a Bernoulli-distributed dropout. With a probability $P_{ood}$, the entire observation is replaced by random Gaussian noise, forcing the policy to handle transient perceptual failures.

*b) Real-world Pipeline $\mathcal{F}_{real}$:* During deployment, the physical depth stream is refined to match the characteristics of the trained policy's input space:

1) **Crop and Resize**: The raw stream is cropped and resized to ensure the input manifold is consistent with the simulation geometry.
2) **Depth Inpainting**: Physical sensors often exhibit "black region" (zero-valued pixels) due to occlusion or disparity shadows. We apply a spatial inpainting operator to recover these missing depth values.
3) **Gaussian Blur**: A Gaussian blur is applied to suppress high-frequency sensor jitter and electronic noise.

*3) Temporal Depth Aggregation via Strided Sampling:* To enhance policy robustness, particularly during high-speed locomotion where rapid terrain changes necessitate a broader temporal context, we incorporate a long-term history of depth observations. Unlike proprioceptive states that typically utilize a dense history of the last $n$ consecutive steps, the depth input employs a strided temporal sampling strategy to balance the look-back window with computational efficiency.

Specifically, we define a history buffer consisting of $m$ frames sampled with a temporal stride $\ell$. Furthermore, a single-frame delay is introduced during training to emulate physical sensor latency. Let $I_t$ denote the depth image at current time step $t$. The historical input sequence $\mathcal{H}_t$ is formulated as:

$$\mathcal{H}_t = \{I_{t-k\cdot\ell-1} \mid k = 0, 1, \ldots, m-1\}. \tag{6}$$

where $k$ is the frame index. This configuration allows the policy to perceive a total temporal horizon of $(m-1)\cdot\ell$ steps

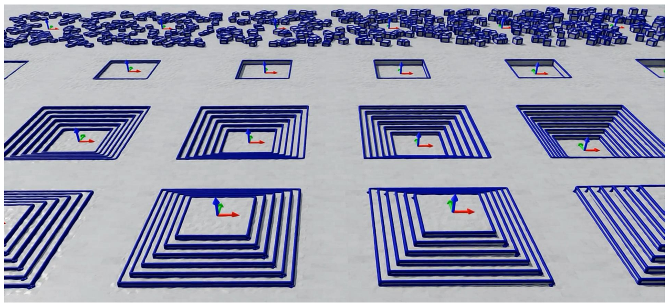
Fig. 4: Automatically detected edges across diverse terrains.

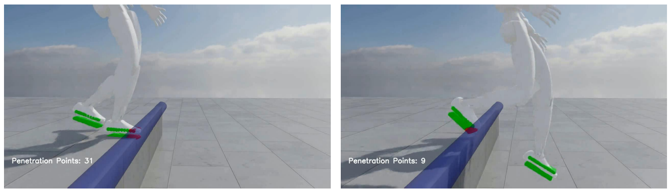

Fig. 5: Volume points distributed within the foot manifold.

while only processing $m$ discrete frames.

By utilizing this sparse yet extended temporal representation, the agent can effectively capture the trend of the terrain profile and the robot's relative velocity without the redundant information overlap inherent in consecutive high-frequency frames. This strided history proves critical for anticipating obstacles and adjusting gait during high-speed maneuvers.

### *C. Terrain Edge Contact Penalization*

Driven by energy efficiency, RL agents naturally tend to minimize effort by taking shorter steps, often placing feet on the nearest terrain edges [4]. While this strategy can yield high success rates in simulation due to simplified contact physics, it significantly increase the Sim-to-Real gap. In the real world, contact dynamics at structural edges are far more unpredictable than in simulation, and stepping partially on them frequently leads to catastrophic slippage and deployment failures. To mitigate this, we propose a soft-constraint mechanism that penalizes foot contacts near sharp geometric boundaries. First, we use a detector to find the sharp edges of the terrain mesh. Second, we attach a set of "volume points" to the robot's feet. By penalizing these points when they penetrate the edge, the policy learns to select more stable, centered foot positions rather than risky ones.

*1) Terrain Edge Detector:* First, we identify sharp terrain edges by comparing the dihedral angle between adjacent faces to a predefined threshold $\tau$. To enhance computational efficiency, the resulting raw edges are further processed to filter noise and concatenate short segments. The edge processing step employs a greedy concatenation strategy to merge fragmented segments and reduce the total number of primitives. More details can be found in Appendix.

*2) Volumetric Point Penetration Penalization:* To accurately monitor the contact state relative to the terrain edges, we distribute a set of volume points $\mathcal{P}$ within each foot's collision manifold, as illustrated in Figure 5. We leverage NVIDIA Warp [26] to perform massively parallel distance queries, computing the penetration depth of each point relative to the previously constructed spatial collision grid $\mathcal{S}$.

To discourage unstable foot-ground interactions near terrain edges, we define a penalty term that considers both the geometric penetration and the velocity of the foot following [50]. For each point $i \in \mathcal{P}$, let $\mathbf{d}_i$ be its penetration offset and $\mathbf{v}_i$ be its linear velocity in the world frame. The penalization reward $r_{vol}$ is formulated as:

$$r_{vol} = \sum_{i=1}^{|\mathcal{P}|} \|\mathbf{d}_i\| \cdot (\|\mathbf{v}_i\| + \epsilon). \tag{7}$$

where $\epsilon = 10^{-3}$ is a small constant for numerical stability. This formulation ensures that the policy is penalized more heavily for high-velocity impacts or scraping motions near terrain edges, thereby encouraging the robot to seek stable footholds.

### *D. Position-based Velocity Command*

During training, conventional velocity commands that are sampled uniformly often lead to "reward hacking", where the robot turns in circles to collect rewards instead of actually crossing obstacles [4]. Previous research has attempted to solve this by modifying velocity tracking rewards [4, 5] or using goal-based commands [48, 49]. However, relying only on reward tuning with randomly sampled commands makes it difficult to reach maximum performance. Furthermore, pure goal commands often lack control over the robot's speed, and randomly sampled goals may not provide suitable targets. To address these issues, we introduce a new method that generates targets using "flat patches" and creates specific velocity commands to improve training performance.

*1) Target Generation via Flat Patch:* Following the approach in IsaacLab [31], we identify "flat patches" on the terrain mesh to serve as reachable navigation targets. A location is considered a valid target if the terrain within a radius $r$ is sufficiently level. Specifically, we sample potential locations and use ray-casting to check the height difference of the surrounding terrain. A patch is accepted only if the maximum height difference is below a threshold $\delta$. This ensures that targets are placed on stable ground rather than steep slopes or unreachable areas. An example can be found in Figure 6.

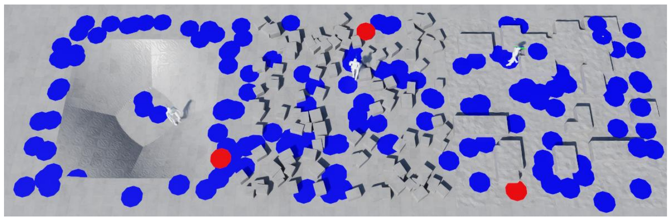
Fig. 6: Flat patches on different terrains.

*2) Position-based Velocity Command Generation:* After generating the flat patches, we periodically select one as the navigation goal. The robot's velocity commands are then generated based on the relative position of this goal. Let $\mathbf{p}^B_{goal} = [x_g, y_g]^T$ represent the goal position in the robot's

**Algorithm 1:** Flat Patch Sampling

```
Input  : Mesh $\mathcal{M}$, Patch radius $r$, Max height
         difference $\delta$
Output: Set of valid targets $\mathcal{P}$
1 while $|\mathcal{P}| < N_{targets}$ do
2     1. Sample a random 2D position $(x, y)$ in the
       environment;
3     2. Ray-cast to get a set of heights $H$ within radius
       $r$ around $(x, y)$;
4     if $\max(H) - \min(H) < \delta$ then
5         Add $(x, y, \text{avg}(H))$ to $\mathcal{P}$;
6 return $\mathcal{P}$;
```

base frame. The desired linear velocity $v_x$ and angular velocity $\omega_z$ are calculated as follows:

$$v_x = \text{clip}(k_v \cdot x_g, 0, v_{max}), \tag{8}$$

$$\omega_z = \text{clip}(k_\omega \cdot \text{atan2}(y_g, x_g), -\omega_{max}, \omega_{max}). \tag{9}$$

where $k_v$ and $k_\omega$ are the linear and angular stiffness coefficients. Because we use the forward camera on the robot's head link, we only focus on forward locomotion and heading alignment; therefore, the lateral velocity command $v_y$ is set to zero. The velocity limits $(v_{max}, \omega_{max})$ are adaptively adjusted based on the terrain category.

However, relying solely on these position-based commands can limit the robot's ability to learn in-place turning, as the goal is usually far away. To address this, we assign a small subset of agents on flat terrain to receive pure turning commands. For these agents, we set $v_x = 0$ and provide a random $\omega_z$, forcing the policy to learn effective in-place rotation maneuvers. This combination improves the overall maneuverability of the robot across different environments.

### *E. Policy Training with Adversarial Motion Priors*

Following previous works [43, 40], we use the Adversarial Motion Priors (AMP) framework [32] to improve the robot's gait style and overall locomotion ability. Our reference dataset $\mathcal{D}$ is collected at a frequency of $f = 50\,\text{Hz}$ from three primary sources:

1) **Synthetic Data**: Walking motions generated by a Model Predictive Control (MPC) controller [10] to provide stable movement.
2) **Human Motion**: High-quality human data captured via the NOKOV motion capture system.
3) **Running Motion**: High-speed running motions selected from the LAFAN dataset [14].

The human motions specifically include challenging tasks such as climbing up/down high platforms and ascending/descending stairs. We use GMR [2] to retarget these human trajectories into robot motions.

To avoid the "mode collapse" problem, we train the walking and running policies separately using different datasets. The walking dataset $\mathcal{D}_{\text{walk}}$ combines sources 1 and 2, totaling $T = 379.62\,\text{s}$ of motion data. The running dataset $\mathcal{D}_{\text{run}}$ consists of source 3, with a total duration of $T = 1.54\,\text{s}$. This multi-source approach allows the policy to learn both the stability of MPC and the natural agility of human-like movement.

Unlike previous works that only use a single state pair $(\mathbf{s}_t, \mathbf{s}_{t+1})$ as the transition for the discriminator, we use a short sequence of past states to better capture the motion's temporal features. The transition is defined as $(\mathbf{S}_t, \mathbf{S}_{t+1})$, where $\mathbf{S}_t = [\mathbf{s}_{t-n}, \ldots, \mathbf{s}_t]$ represents a history of $n$ frames.

$$\mathbf{s}_t = \{(\mathbf{v}_t, \boldsymbol{\omega}_t, \mathbf{g}_t, \mathbf{q}_t, \dot{\mathbf{q}}_t)\}\,, \tag{10}$$

To increase the diversity of the motion data, we also apply symmetric augmentation to the dataset. Following the AMP framework, the discriminator $D(\mathbf{S})$ is trained using a least-squares (MSE) loss:

$$L_D = \mathbb{E}_{\mathcal{M}}[(D(\mathbf{S}) - 1)^2] + \mathbb{E}_{\mathcal{P}}[(D(\mathbf{S}) + 1)^2], \tag{11}$$

where $\mathcal{M}$ represents the reference motion dataset and $\mathcal{P}$ represents the motions produced by the current policy. To ensure training stability, we incorporate gradient penalty and weight decay. The style-reward $r_t$ provided by the discriminator to the policy is calculated as:

$$r_t = \max\left[0, 1 - 0.25(D(\mathbf{S}_t) - 1)^2\right]. \tag{12}$$

Compared to the binary cross-entropy loss and log-based rewards, the combination of MSE loss and quadratic rewards provides smoother, non-saturating gradients that prevent the vanishing gradient problem and ensure more stable convergence toward the reference motion manifold.

## IV. Experiments

In this section, we evaluate the robustness and performance of our framework through extensive experiments in both simulation and real-world environments. We test the policy across various challenging terrains, including stairs, high platforms, grassy ramps, and discrete gaps, using both walking and running gaits. Our evaluation aims to answer the following three questions:

- **Q1**: Does our framework enable efficient training and reliable zero-shot deployment on physical hardware?
- **Q2**: Does the edge-aware penalization mechanism improve foothold safety and stability when traversing discrete terrain features? Is it scalable to new terrains without any extra design?
- **Q3**: What are the individual contributions of the key design components (e.g., perception, AMP, and command generation) to the robot's performance?

### *A. Experiment Configurations*

**Training Environment:** We train our policies using NVIDIA Isaac Sim and Isaac Lab [31]. All training sessions are conducted on an NVIDIA RTX 4090 GPU, parallelized with 4096 humanoid robot agents. We utilize the 29-DoF Unitree G1 humanoid robot for both simulation training and physical deployment. The history length for proprioception and depth image is 8 frames.

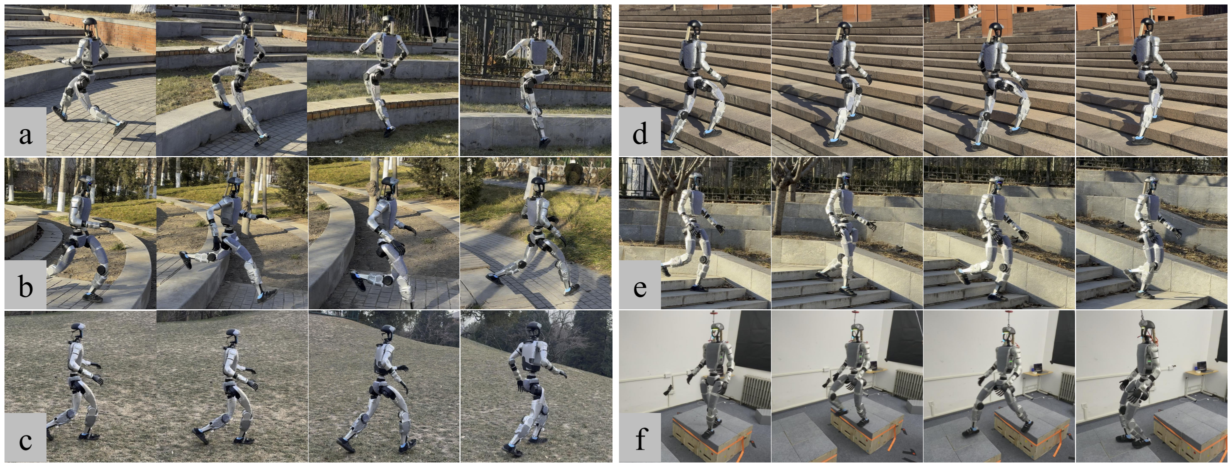

Fig. 7: Snapshots of the humanoid robot traversing diverse indoor and outdoor environments via zero-shot transfer: (a) running ascent onto a high platform; (b) running descent from a high platform; (c) running on a grassy slope; (d) stair ascent; (e) stair descent; (f) traversing a discrete gap.

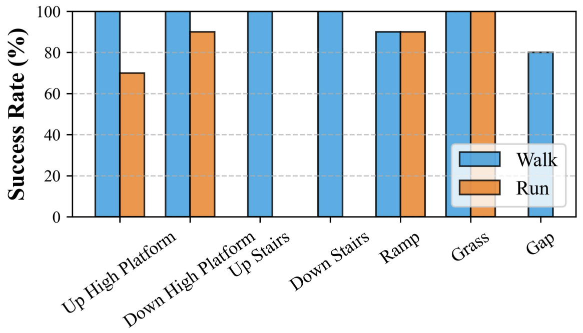

Fig. 8: Success rates across different terrains and modes.

**Hardware and Perception:** For real-world deployment, we utilize the factory-integrated Intel RealSense D435i depth camera that comes standard with the Unitree G1 without any hardware modification. Depth images are captured at 60 Hz with a raw resolution of $480 \times 270$, which are then downsampled to $64 \times 36$ and cropped to a final input size of $32 \times 18$ for the policy.

**Onboard Deployment:** The policy operates at a frequency of 50 Hz on the robot's onboard NVIDIA Jetson Orin NX. We utilize `onnxruntime` for efficient policy inference. To ensure low-latency performance, depth acquisition and image processing are handled in a dedicated asynchronous process at 60 Hz. The base policy is post-trained for high-speed running and some specific terrains.

## B. Real-world Deployment

**We highly recommend watching videos in the supplementary materials for better demonstration.** We evaluate our framework by deploying the trained policies directly onto the humanoid robot via zero-shot sim-to-real transfer. As illustrated in Figure 7, the robot successfully performs various locomotion primitives across challenging terrains. Notably, the robot achieves a maximum running speed of 2.5 m/s and successfully traverses difficult obstacles, including high platforms up to 32 cm and discrete gaps with a width of 50 cm.

Crucially, the high-frequency depth perception (60 Hz) is the key component for these high-dynamic tasks, such as running onto a high platform. The low-latency perception feedback allows the policy to make rapid adjustments to the robot's posture and terrain changes in real-time.

To quantify the system's reliability, we conduct 10 trials for each combination of terrain and gait. The resulting success rates are summarized in Figure 8. Our policy maintains a high success rate across nearly all tested scenarios, demonstrating remarkable robustness to real-world sensory noise and physical discrepancies.

Furthermore, we conduct a long-duration test to evaluate the stability of the system over a prolonged period. The robot is tasked with continuous locomotion across multiple staircases and flat surfaces. The robot successfully maintains its balance and walks for 4 minutes without any falls or human intervention. This sustained performance highlights the effectiveness of our control framework and its potential for deployment in complex, long-range real-world tasks.

## C. Edge-aware Penalization Mechanism

As introduced in subsection III-C, we utilize the *Terrain Edge Detector* and *Volumetric Point Penetration Penalization* to improve foothold safety. As illustrated in Figure 9, with edge-aware penalization enabled, the robot tends to seek higher safety margins, effectively placing its feet away from terrain edges to maintain stable contact.

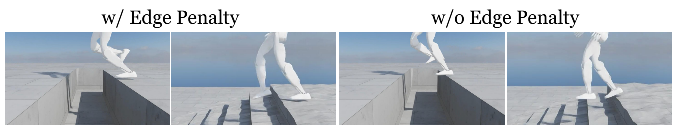

Fig. 9: Visualization of foothold placement.

To quantitatively evaluate this effect, we measure the **Success Rate** and **Mean Feet Landing Area Percentage** in simulation across various challenging terrains. Landing Area quantifies the portion of the foot link that is on the terrain surface. We test each policy for 10,000 time-steps with 1,000 robots.

TABLE I: Ablation results across all terrain types and configurations.

| | Success Rate (%) ↑ | | | | | | | | | Mean Reaching Time (s) ↓ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Large Box** | **Ramp** | **Small Box** | **Rough** | **Stair down** | **Platform down** | **Stair up** | **Platform up** | **Gap** | **Large Box** | **Ramp** | **Small Box** | **Rough** | **Stair down** | **Platform down** | **Stair up** | **Platform up** | **Gap** |
| w/o depth history | 100.0 | 100.0 | 1.66 | 100.0 | 99.49 | 99.70 | 99.93 | 100.0 | 100.0 | 5.90 | 6.72 | 11.68 | 5.78 | 5.54 | 5.82 | 6.98 | 5.91 | 5.79 |
| w/o pose-based | 100.0 | 100.0 | 90.68 | 100.0 | 100.0 | 100.0 | 99.74 | 100.0 | 100.0 | 5.90 | **5.74** | 11.32 | 5.72 | 6.02 | 5.82 | 6.39 | 6.34 | 5.70 |
| w/o MoE | 100.0 | 100.0 | 84.51 | 100.0 | 99.68 | 100.0 | 100.0 | 100.0 | 100.0 | 6.06 | 6.22 | 10.98 | 5.49 | 5.23 | 5.65 | 6.12 | 5.51 | 5.59 |
| w/o Amp | 99.56 | 99.94 | 0.00 | 100.0 | 99.94 | 100.0 | 100.0 | 99.94 | 99.94 | 7.27 | 5.68 | – | 6.50 | 5.80 | 6.31 | 6.17 | 6.25 | 6.07 |
| 10Hz heightmap | 100.00 | 100.00 | 15.28 | 100.0 | 99.10 | 100.0 | 100.0 | 100.0 | 100.0 | 6.80 | 6.01 | 14.73 | 6.34 | 5.48 | 6.36 | 6.21 | 7.35 | 6.40 |
| w/o depth | 100.00 | 100.00 | 3.33 | 100.0 | 98.58 | 100.0 | 99.92 | 11.60 | 67.95 | 7.59 | 8.48 | 10.59 | 7.75 | 5.62 | 7.05 | 7.90 | 9.24 | 7.58 |
| Ours | **100.0** | **100.0** | **99.09** | **100.0** | **99.95** | **100.0** | **100.0** | **100.0** | **100.0** | **5.50** | 5.84 | **8.03** | **5.40** | **4.45** | **5.48** | **5.35** | **5.45** | **5.43** |

TABLE II: Quantitative comparison Results.

| | Success Rate (SR) ↑ | | Landing Area % ↑ | |
|---|---|---|---|---|
| **Terrain Type** | **Ours** | **No Edge** | **Ours** | **No Edge** |
| Stair Ascent | 100.00% | 100.00% | **0.99** | 0.98 |
| Stair Descent | **99.95%** | 99.82% | **0.94** | 0.87 |
| Deep Gap | **100.0%** | 99.94% | **0.96** | 0.94 |
| Small Box | **99.09%** | 93.17% | **0.96** | 0.95 |

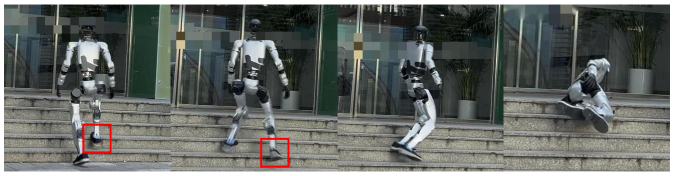
Fig. 10: The robot slips and falls after stepping on a stair edge.

As shown in Table II, our method outperforms *No Edge* in both success rate and landing area. Notably, the mean landing area values are generally high because the robot spends a considerable portion of each episode on flat ground; thus, the observed gap indicates a substantial improvement, specifically in edge-dense areas.

In the real world, stepping on edges sometimes causes slippage or unpredictable contact dynamics, driving the robot into Out-of-Distribution (OOD) states that lead to immediate falls, as shown in Figure 10. By incentivizing the policy to maximize the landing area and avoid edges, our method significantly reduces these real-world risks, ensuring greater robustness during physical deployment. As shown in Figure 7, our policy exhibits redundant safety margins and maintains stable footholds throughout the traverse.

To further demonstrate the scalability of our framework, we evaluate the edge detector on two new terrain types: *Stones* and *Stakes*. As illustrated in Figure 11, the detector precisely identifies terrain edges without any manual feature engineering or parameter tuning. This robust zero-shot performance confirms that our edge-aware mechanism can scale to arbitrary terrain topologies.

### D. Ablation Study on Training Recipe

To investigate the individual contributions of our proposed components, we conduct a series of ablation studies by comparing our full framework against the following versions:

- **Single-frame Depth (w/o depth history)**: The policy receives only the current depth image instead of the strided temporal history.
- **Uniform Command (w/o pose-based)**: The policy is trained using standard uniform velocity sampling with heading commands.

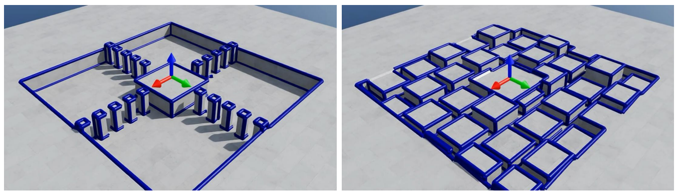
Fig. 11: Generalization of edge detector to new terrains.

- **Vanilla MLP (w/o MoE)**: The policy network is replaced by a standard Multi-Layer Perceptron (MLP) with an equivalent number of parameters.
- **No Motion Prior (w/o AMP)**: The Adversarial Motion Prior is removed, training the policy solely through task-related rewards.
- **Blind Policy (w/o depth)**: The policy relies exclusively on proprioceptive feedback.
- **Low-freq Elevation Map (10hz heightmap)**: A 10 Hz elevation map is used for perception, which is a common configuration in LiDAR-based methods.

We evaluate the policies separately with 1,000 robots in simulation, record their success rates, and the mean reaching time. As shown in Table I, our method achieves the highest success rate, confirming that all proposed components are essential for complex terrains. The poor performance of the 10Hz height map baseline highlights the importance of high-frequency depth perception.

## V. Conclusion

In this paper, we present a scalable end-to-end perceptive locomotion framework for humanoid robots. By integrating a novel volumetric edge-aware penalization mechanism with position-based velocity command generation, our approach achieves high-dynamic behaviors on complex terrains. We conduct extensive experiments in both simulation and real-world deployment. It demonstrates that our framework enables humanoid robots to perform agile running and stable walking across a variety of challenging environments.

Despite these advancements, our current system has some limitations. First, the perception system relies solely on a single onboard forward-facing depth camera, which leads to a lack of backward or lateral movement capability. Future work may attach multiple cameras to the robot to enable omnidirectional perception and agility. Second, we observe that training a variety of terrains and gait modes simultaneously can lead to mode collapse and performance degradation compared to specialized policies. Advanced multi-task reinforcement learning techniques can be investigated to enhance the capacity of a single unified policy.

## References

[1] Ayush Agrawal, Shuxiao Chen, Akshara Rai, and Koushil Sreenath. Vision-aided dynamic quadrupedal locomotion on discrete terrain using motion libraries. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4708–4714. IEEE, 2022.

[2] Joao Pedro Araujo, Yanjie Ze, Pei Xu, Jiajun Wu, and C Karen Liu. Retargeting matters: General motion retargeting for humanoid motion tracking. *arXiv preprint arXiv:2510.02252*, 2025.

[3] Qingwei Ben, Botian Xu, Kailin Li, Feiyu Jia, Wentao Zhang, Jingping Wang, Jingbo Wang, Dahua Lin, and Jiangmiao Pang. Gallant: Voxel grid-based humanoid locomotion and local-navigation across 3d constrained terrains. *arXiv preprint arXiv:2511.14625*, 2025.

[4] 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.

[5] Yi Cheng, Hang Liu, Guoping Pan, Houde Liu, and Linqi Ye. Quadruped robot traversing 3d complex environments with limited perception. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9074–9081. IEEE, 2024.

[6] Annett Chilian and Heiko Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4571–4576. IEEE, 2009.

[7] Helei Duan, Bikram Pandit, Mohitvishnu S Gadde, Bart Van Marum, Jeremy Dao, Chanho Kim, and Alan Fern. Learning vision-based bipedal locomotion for challenging terrain. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 56–62. IEEE, 2024.

[8] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. Robust rough-terrain locomotion with a quadrupedal robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5761–5768. IEEE, 2018.

[9] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters*, 3(4):3019–3026, 2018.

[10] Manuel Yves Galliker. Whole-body humanoid mpc: Realtime physics-based procedural loco-manipulation planning and control. https://github.com/1x-technologies/wb_humanoid_mpc, 2024.

[11] Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5973–5979. IEEE, 2021.

[12] Ruben Grandia, Fabian Jenelten, Shaohui Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model-predictive control. *IEEE Transactions on Robotics*, 39(5):3402–3421, 2023.

[13] Xinyang Gu, Yen-Jen Wang, and Jianyu Chen. Humanoid-gym: Reinforcement learning for humanoid robot with zero-shot sim2real transfer. *arXiv preprint arXiv:2404.05695*, 2024.

[14] Félix G Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening. *ACM Transactions on Graphics (TOG)*, 39(4):60–1, 2020.

[15] Junzhe He, Chong Zhang, Fabian Jenelten, Ruben Grandia, Moritz Bächer, and Marco Hutter. Attention-based map encoding for learning generalized legged locomotion. *Science Robotics*, 10(105):eadv3604, 2025.

[16] Runhan Huang, Shaoting Zhu, Yilun Du, and Hang Zhao. Moe-loco: Mixture of experts for multitask locomotion. *arXiv preprint arXiv:2503.08564*, 2025.

[17] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[18] Fabian Jenelten, Takahiro Miki, Aravind E Vijayan, Marko Bjelonic, and Marco Hutter. Perceptive locomotion in rough terrain–online foothold optimization. *IEEE Robotics and Automation Letters*, 5(4):5370–5376, 2020.

[19] Fabian Jenelten, Junzhe He, Farbod Farshidian, and Marco Hutter. Dtc: Deep tracking control. *Science Robotics*, 9(86):eadh5401, 2024.

[20] Sotaro Katayama, Masaki Murooka, and Yuichi Tazaki. Model predictive control of legged and humanoid robots: models and algorithms. *Advanced Robotics*, 37(5):298–315, 2023.

[21] Donghyun Kim, Daniel Carballo, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, Bryan Lim, and Sangbae Kim. Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2464–2470. IEEE, 2020.

[22] Hyeongjun Kim, Hyunsik Oh, Jeongsoo Park, Yunho Kim, Donghoon Youm, Moonkyu Jung, Minho Lee, and Jemin Hwangbo. High-speed control and navigation for quadrupedal robots on complex and discrete terrain. *Science Robotics*, 10(102):eads6192, 2025.

[23] Qiayuan Liao, Takara E Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C Karen Liu. Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion. *arXiv preprint arXiv:2508.08241*, 2025.

[24] Junfeng Long, Junli Ren, Moji Shi, Zirui Wang, Tao Huang, Ping Luo, and Jiangmiao Pang. Learning humanoid locomotion with perceptive internal model. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9997–10003. IEEE, 2025.

[25] Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Sirui Chen, Fernando Castañeda, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, et al. Sonic: Supersizing motion tracking for natural humanoid whole-body control. *arXiv preprint arXiv:2511.07820*, 2025.

[26] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[27] Octavio Antonio Villarreal Magana, Victor Barasuol, Marco Camurri, Luca Franceschi, Michele Focchi, Massimiliano Pontil, Darwin G Caldwell, and Claudio Semini. Fast and continuous foothold adaptation for dynamic locomotion through cnns. *IEEE Robotics and Automation Letters*, 4(2):2140–2147, 2019.

[28] Carlos Mastalli, Ioannis Havoutis, Alexander W Winkler, Darwin G Caldwell, and Claudio Semini. On-line and on-board planning and perception for quadrupedal locomotion. In *2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–7. IEEE, 2015.

[29] Carlos Mastalli, Michele Focchi, Ioannis Havoutis, Andreea Radulescu, Sylvain Calinon, Jonas Buchli, Darwin G Caldwell, and Claudio Semini. Trajectory and foothold optimization using low-dimensional models for rough terrain locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1096–1103. IEEE, 2017.

[30] Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. Motion planning for quadrupedal locomotion: Coupled planning, terrain mapping, and whole-body control. *IEEE Transactions on Robotics*, 36(6):1635–1648, 2020.

[31] Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

[32] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

[33] Ilija Radosavovic, Sarthak Kamat, Trevor Darrell, and Jitendra Malik. Learning humanoid locomotion over challenging terrain. *arXiv preprint arXiv:2410.03654*, 2024.

[34] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*, 9(89):eadi9579, 2024.

[35] Nikita Rudin, Junzhe He, Joshua Aurand, and Marco Hutter. Parkour in the wild: Learning a general and extensible agile locomotion policy using multi-expert distillation and rl fine-tuning. *arXiv preprint arXiv:2505.11164*, 2025.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[37] Nicola Scianca, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo. Mpc for humanoid gait generation: Stability and feasibility. *IEEE Transactions on Robotics*, 36(4):1171–1188, 2020.

[38] Jonah Siekmann, Kevin Green, John Warila, Alan Fern, and Jonathan Hurst. Blind bipedal stair traversal via sim-to-real reinforcement learning. *arXiv preprint arXiv:2105.08328*, 2021.

[39] Haolin Song, Hongbo Zhu, Tao Yu, Yan Liu, Mingqi Yuan, Wengang Zhou, Hua Chen, and Houqiang Li. Gait-adaptive perceptive humanoid locomotion with real-time under-base terrain reconstruction. *arXiv preprint arXiv:2512.07464*, 2025.

[40] Jingkai Sun, Gang Han, Pihai Sun, Wen Zhao, Jiahang Cao, Jiaxu Wang, Yijie Guo, and Qiang Zhang. Dpl: Depth-only perceptive humanoid locomotion via realistic depth synthesis and cross-attention terrain reconstruction. *arXiv preprint arXiv:2510.07152*, 2025.

[41] Wandong Sun, Baoshi Cao, Long Chen, Yongbo Su, Yang Liu, Zongwu Xie, and Hong Liu. Learning perceptive humanoid locomotion over challenging terrain. *arXiv preprint arXiv:2503.00692*, 2025.

[42] Huayi Wang, Zirui Wang, Junli Ren, Qingwei Ben, Tao Huang, Weinan Zhang, and Jiangmiao Pang. Beamdojo: Learning agile humanoid locomotion on sparse footholds. *arXiv preprint arXiv:2502.10363*, 2025.

[43] Yushi Wang, Changsheng Luo, Penghui Chen, Jianran Liu, Weijian Sun, Tong Guo, Kechang Yang, Biao Hu, Yangang Zhang, and Mingguo Zhao. Learning vision-driven reactive soccer skills for humanoid robots. *arXiv preprint arXiv:2511.03996*, 2025.

[44] Lorenz Wellhausen and Marco Hutter. Rough terrain navigation for legged robots using reachability planning and template learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6914–6921. IEEE, 2021.

[45] Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. Navigation planning for legged robots in challenging terrain. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189. IEEE, 2016.

[46] Bowen Yang, Lorenz Wellhausen, Takahiro Miki, Ming Liu, and Marco Hutter. Real-time optimal navigation planning using learned motion costs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9283–9289. IEEE, 2021.

[47] Mingqi Yuan, Tao Yu, Haolin Song, Bo Li, Xin Jin, Hua Chen, and Wenjun Zeng. Pvp: Data-efficient humanoid robot learning with proprioceptive-privileged contrastive representations. *arXiv preprint arXiv:2512.13093*, 2025.

[48] Chong Zhang, Nikita Rudin, David Hoeller, and Marco Hutter. Learning agile locomotion on risky terrains. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11864–11871. IEEE, 2024.

[49] Shaoting Zhu, Runhan Huang, Linzhan Mou, and Hang Zhao. Robust robot walker: Learning agile locomotion

over tiny traps. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15987–15993. IEEE, 2025.

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

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

## Appendix

### A. Details in Policy Training

*1) Policy Observation with Noise:* In this section, we provide the detailed specifications of the observation space used in our Reinforcement Learning framework. The policy input includes both proprioceptive states and exteroceptive depth information. To bridge the gap between simulation and reality, we apply specific noise models and delays during training.

All proprioceptive observations include a history of $H = 8$ time steps. The detailed configuration, including scaling factors and noise distributions, is summarized in Table III. The detailed exteroceptive depth observation is illustrated in subsection III-B.

TABLE III: Detailed policy proprioceptive observation space, scaling factors, and noise models.

| Observation Term | Scale | Noise Type | Noise Parameters |
|---|---|---|---|
| Base Angular Velocity ($\omega_t$) | 0.25 | Uniform | $\mathcal{U}(-0.2, 0.2)$ rad/s |
| Projected Gravity ($\mathbf{g}_t$) | 1.0 | Uniform | $\mathcal{U}(-0.05, 0.05)$ |
| Velocity Commands ($\mathbf{c}_t$) | 1.0 | None | N/A |
| Joint Positions ($q_t$) | 1.0 | Uniform | $\mathcal{U}(-0.01, 0.01)$ rad |
| Joint Velocities ($\dot{q}_t$) | 0.05 | Uniform | $\mathcal{U}(-0.5, 0.5)$ rad/s |
| Last Action ($a_{t-1}$) | 1.0 | None | N/A |

*2) Termination Criteria:* To ensure feasible and fast learning, the training episode terminates immediately if any of the following conditions are met. These criteria prevent the policy from exploring physically invalid states or dangerous configurations. The specific thresholds and parameters used in our MDP are detailed below:

- **Time Out:** The episode terminates naturally if the maximum episode duration is reached.
- **Terrain Out-of-Bounds:** The episode ends if the robot moves beyond the valid terrain boundaries. A distance buffer of 2.0 m is applied to allow for minor deviations at the boundaries.
- **Illegal Contact:** To penalize falls, the episode terminates if the robot's torso (`torso_link`) detects a contact force exceeding a threshold of 1.0.
- **Bad Orientation:** To maintain stability, the episode ends if the tilt angle of the robot's base exceeds 1.0 rad relative to the gravity vector.
- **Insufficient Root Height:** The episode terminates if the robot's root height drops below 0.5 m relative to the environment origin, indicating a fall or a failure to stand up.

*3) Reward Function:* The reward function is designed to encourage velocity tracking while enforcing physical safety. It consists of four components: task reward, regularization reward, safety reward, and AMP style reward. The detailed definition of each term, along with its corresponding weight and key parameters, is provided in Table IV.

*4) Terrain Configuration:* To ensure the robot learns robust locomotion skills across diverse environments, we employ a curriculum-based terrain generation pipeline. The training environment consists of a grid of $10 \times 20$ sub-terrains, each measuring $8.0 \times 8.0$ meters. The height field resolution is set to $0.05$ m horizontally and $0.005$ m vertically.

The detailed specifications for each terrain type, including their geometric parameters, curriculum proportions, and the associated velocity command ranges used for training, are summarized in Table VI. We randomly generate walls around the terrains to simulate indoor environment with probability $\mathcal{P} = 0.3$. Note that we use Rough (Stand) to train the robot in-place turning.

*5) Policy Network:* The learning framework consists of three primary components: a CNN-based visual encoder, a Mixture-of-Experts (MoE) policy, and an Adversarial Motion Priors (AMP) discriminator. The visual encoder processes the strided depth history through a convolutional network, followed by a projection MLP to extract a visual embedding. This embedding is concatenated with the proprioception and fed into the MoE-based actor-critic network, where a gating mechanism dynamically weights the contributions of multiple expert networks to generate robust actions. Additionally, the AMP discriminator distinguishes between the actor's generated motions and the reference dataset. The detailed hyperparameters for these components are listed in Table V.

TABLE V: Detailed hyperparameters for the network architecture.

| Parameter | Value |
|---|---|
| CNN channels | [4] |
| CNN kernel sizes | [3] |
| CNN strides | [1] |
| CNN paddings | [1] |
| CNN pooling | MaxPool |
| Encoder projection units | $[256, 256]$ |
| Encoder embedding dim | 128 |
| Encoder activation | ReLU |
| Policy Type | MoE |
| Number of Experts | 4 |
| Gating hidden units | [256] |
| Expert hidden units | $[256, 128, 64]$ |
| Policy activation | ELU |
| Discriminator hidden units | $[1024, 512]$ |
| Discriminator activation | ReLU |

### B. Terrain Edge Detector Algorithm

We first identify sharp terrain edges by comparing the dihedral angle between adjacent faces to a predefined threshold $\tau$, as shown in algorithm 2

Then, we employ a greedy edge concatenation process to refine raw edges, as shown in algorithm 3. By merging adjacent collinear segments into single extended edges, this approach significantly reduces storage requirements and accelerates computational speed.

TABLE IV: Detailed specification of reward terms, weights, and mathematical formulations.

| **Reward Term** | **Weight** | **Mathematical Formulation** |
|---|---|---|
| ***Task Reward*** | | |
| Linear Vel Tracking ($v_{xy}$) | 2.0 | $\exp(-\|\mathbf{v}_{xy}^* - \mathbf{v}_{xy}\|^2/0.5^2)$ |
| Angular Vel Tracking ($\omega_z$) | 2.0 | $\exp(-(\omega_z^* - \omega_z)^2/0.5^2)$ |
| Heading Error | $-1.0$ | $\|\omega_z^*\|$ |
| Don't Wait | $-0.5$ | $\mathbb{I}(v_x^* > 0.3) \cdot (\mathbb{I}(v_x < 0.15) + \mathbb{I}(v_x < 0) + \mathbb{I}(v_x < -0.15))$ |
| Is Alive | 3.0 | $+1$ |
| Stand Still | $-0.3$ | $(\|\mathbf{q} - \mathbf{q}_{default}\|_1 - 4.0) \cdot \mathbb{I}(\|\mathbf{v}^*\| < 0.15) \cdot \mathbb{I}(\|\omega_z^*\| < 0.15)$ |
| ***Regularization Reward*** | | |
| Edge Penetration | $-4.0$ | $\sum_{i=1}^{\|\mathcal{P}\|} \|\mathbf{d}_i\| \cdot (\|\mathbf{v}_i\| + \epsilon)$ |
| Feet Air Time | 0.5 | $\min_f(t_{phase,f}) \cdot \mathbb{I}(\sum c_f = 1) \cdot \mathbb{I}(\|\mathbf{v}^*\| > 0.15)$ |
| Feet Slide | $-0.4$ | $\sum_f \|\mathbf{v}_{xy,f}\| \cdot \mathbb{I}(c_f)$ |
| Joint Deviation (Hip) | $-0.5$ | $\sum_{j \in \text{hips}} (q_j - q_{j,default})^2$ |
| Base Ang Vel (XY) ($L_2$) | $-0.05$ | $\|\omega_{xy}\|^2$ |
| Joint Torques ($L_2$) | $-1.5$e-7 | $\|\boldsymbol{\tau}_{legs}\|^2$ |
| Joint Acc ($L_2$) | $-1.25$e-7 | $\|\ddot{\mathbf{q}}\|^2$ |
| Joint Vel ($L_2$) | $-1.0$e-4 | $\|\dot{\mathbf{q}}\|^2$ |
| Action Rate ($L_2$) | $-0.005$ | $\|\mathbf{a}_t - \mathbf{a}_{t-1}\|^2$ |
| Flat Orientation | $-3.0$ | $\|\mathbf{g}_{xy}^{proj}\|^2$ |
| Pelvis Orientation | $-3.0$ | $\|\mathbf{g}_{xy}^{proj,pelvis}\|^2$ |
| Feet Orientation | $-0.4$ | $\sum_f \|\mathbf{g}_{xy,f}^{proj}\| \cdot \mathbb{I}(c_f)$ |
| Feet Height Error | $-0.1$ | $\sum_f \sum_p \text{clip}(h_f - h_{terr,p} - 0.035, 0, 0.3) \cdot \mathbb{I}(c_f)$ |
| Feet Distance | 0.4 | $\exp(-\max(0, 0.12 - \|y_L^R - y_R^R\|)/0.05) - 1$ |
| Energy Consumption | $-5.0$e-5 | $\sum_j (\tau_j \dot{q}_j / k_j)^2$ |
| Freeze Upper Body | $-0.004$ | $\|\mathbf{q}_{upper} - \mathbf{q}_{upper}^{default}\|_1$ |
| ***Safety Reward*** | | |
| Joint Pos Limits | $-1.0$ | $\sum_j (\max(0, q_j - q_{j,max}) + \max(0, q_{j,min} - q_j))$ |
| Joint Vel Limits | $-1.0$ | $\sum_j \max(0, \|\dot{q}_j\| - 0.9\dot{q}_{j,max})$ |
| Torque Limits | $-0.01$ | $\sum_j \max(0, \|\tau_j\| - 0.8\tau_{j,max})^2$ |
| Undesired Contacts | $-1.0$ | $\mathbb{I}(\text{count}(\text{collision}_{body \setminus feet}) > 0)$ |
| ***AMP Reward*** | | |
| AMP Style | 0.25 | $\max\left[0, 1 - 0.25(D(\mathbf{S}_t) - 1)^2\right]$ |

TABLE VI: Detailed specifications of terrain types and their corresponding velocity command ranges for walk and run modes. Note that the lateral velocity command $v_y$ is set to 0 for all terrain types.

| **Terrain Settings** | | **Command Range (walk)** | | | **Command Range (run)** | | |
|---|---|---|---|---|---|---|---|
| **Terrain Type** | **Geometric Parameters** | **Prop.** | $\mathbf{v_x}$ **(m/s)** | $\boldsymbol{\omega_z}$ **(rad/s)** | **Prop.** | $\mathbf{v_x}$ **(m/s)** | $\boldsymbol{\omega_z}$ **(rad/s)** |
| Rough | Perlin Noise Scale $\in [0.0, 0.1]$ m | 5% | $[0.45, 1.0]$ | $[-1.0, 1.0]$ | 20% | $[1.95, 3.05]$ | $[-1.0, 1.0]$ |
| Rough (Stand) | Perlin Noise Scale $\in [0.0, 0.1]$ m | 5% | 0.0 | $[-1.0, 1.0]$ | 5% | 0.0 | $[-1.0, 1.0]$ |
| Square Gaps | Gap Dist $\in [0.1, 0.7]$ m, Depth $\in [0.4, 0.6]$ m | 10% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | – | – | – |
| Stairs Down | Step Height $\in [0.05, 0.23]$ m, Width 0.3 m | 15% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | – | – | – |
| Platform Down | Platform Height $\in [0.05, 0.45]$ m | 10% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | 25% | $[1.95, 3.05]$ | $[-1.0, 1.0]$ |
| Stairs Up | Step Height $\in [0.05, 0.23]$ m, Width 0.3 m | 15% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | – | – | – |
| Platform Up | Platform Height $\in [0.05, 0.45]$ m | 10% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | 25% | $[1.95, 3.05]$ | $[-1.0, 1.0]$ |
| Discrete Boxes | Height $\in [0.05, 0.45]$ m, Width $\in [0.8, 1.5]$ m | 10% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | – | – | – |
| Mesh Boxes | Height Mean $\in [0.1, 0.4]$ m | 10% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | – | – | – |
| Slopes | Slope $\in [0.0, 0.7]$ rad | 10% | $[0.45, 0.8]$ | $[-1.0, 1.0]$ | 25% | $[1.95, 3.05]$ | $[-1.0, 1.0]$ |

**Algorithm 2:** Terrain Edge Detection Algorithm

```
Input  : Triangular Mesh $\mathcal{M} = (\mathcal{V}, \mathcal{F})$, Sharpness
         Threshold $\tau$, Cylinder Radius $r$, Grid
         Resolution $N_{\text{grid}}$
Output: Spatial Collision Grid $\mathcal{S}$
/* 1. Sharp Edge Detection */
1  Let $\mathcal{A}$ be the set of face adjacencies in $\mathcal{M}$;
2  Initialize raw edge set $E_{\text{raw}} \leftarrow \emptyset$;
3  foreach adjacency $a \in \mathcal{A}$ do
4      Let $\alpha_a$ be the dihedral angle of $a$;
5      if $\alpha_a > \tau$ then
6          Let $(v_i, v_j)$ be the vertex indices shared by $a$;
7          Add segment coordinate $(\mathcal{V}[v_i], \mathcal{V}[v_j])$ to $E_{\text{raw}}$;
/* 2. Edge Processing */
8  if $E_{raw}$ is empty then
9      return None;
10 else
11     $E_{\text{final}} \leftarrow$ ProcessEdges($E_{\text{raw}}$);
/* 3. Spatial Structure Construction */
12 Initialize cylinder set $\mathcal{C} \leftarrow \emptyset$;
13 foreach segment $e \in E_{final}$ do
14     Construct cylinder $c$ from segment $e$ with radius $r$;
15     Add $c$ to $\mathcal{C}$;
16 $\mathcal{S} \leftarrow$ CylinderSpatialGrid($\mathcal{C}, N_{\text{grid}}$);
17 return $\mathcal{S}$;
```

**Algorithm 3:** Greedy Concat Edge Process

```
Input  : Edge set $E_{raw}$, Angle thresh $\theta$, Min points
         $k_{\min}$, Tol $\epsilon$
Output: Processed Edge $E_{final}$
1  $G \leftarrow$ BuildAdjacency($E_{raw}$);
2  $S_{avail} \leftarrow$ Vertices($G$);
3  $\tau \leftarrow \cos(\theta)$;
4  $L_{final} \leftarrow \emptyset$;
5  while $S_{avail} \neq \emptyset$ do
6      $v \leftarrow$ PopArbitrary($S_{avail}$);
7      $P \leftarrow [v]$;
8      if $G[v] \neq \emptyset$ then
9          Append neighbor of $v$ to $P$ and update $G$;
10     repeat
11         foreach $end \in \{head, tail\}$ do
12             Let $\mathbf{d}$ be the direction vector at $end$;
13             Find neighbor $n \in G[P.\text{get}(end)]$
                 maximizing alignment with $\mathbf{d}$;
14             if alignment $> \tau$ then
15                 $P$.extend($n$) at $end$;
16                 Remove edge to $n$ from $G$ and $S_{avail}$;
17     until cannot extend;
18     while $|P| \geq k_{\min}$ do
19         Find smallest index $i$ such that
             Dist($P[i \ldots \text{end}]$, Line($P[i], P[\text{end}]$)) $< \epsilon$;
20         if segment found then
21             $E_{final}$.append($\{P[i], P[\text{end}]\}$);
22             $P \leftarrow P[0 \ldots i]$;
23         else
24             break;
25 return $E_{final}$;
```