Autonomous Trail Racing with a Bicycle Robot

Fabian Jenelten[1,†], Deepana Ishtaweera[1,2,†], Pablo Garcia[1], Liana Margolese[1], Igor Bogoslavskyi[1], Luca Buonocore[1], Jan Migliazza[1], Aleksandrs Ecins[1], Annika Wollschlaeger[1], Dario Bellicoso[1], Gabriel Nelson[3], Abel Gawel[1] and Marco Hutter[1,2]

Abstract

Navigating off-road trails on a mountain bike requires diverse locomotion skills, including aerial maneuvers, cornering, and slip recovery, paired with sophisticated perception for scene understanding. Translating these capabilities to autonomous, two-wheeled robots remains a formidable challenge. We show that autonomous, high-speed trail navigation can be achieved through a single idea: encoding the trail as a semantic layer of the terrain. In simulation, this layer embeds the navigation task into the training environment; during deployment, it is reconstructed onboard from camera and LiDAR data. First, we introduce a unified learning framework that treats local navigation and locomotion as a joint optimization problem, bypassing the twist or pose command interface of classical control hierarchies. With fixed reward weights, the same framework produces both trail racing and agile parkour, shaped only through terrain semantics and episode length. Second, our perception pipeline combines a frozen vision foundation model with a lightweight dual-head decoder that predicts per-pixel terrain semantics and confidence scores, which are fused with LiDAR range measurements into an egocentric semantic elevation map at $30\,\mathrm{Hz}$ onboard an NVIDIA AGX Orin. We validate our framework on the Ultra Mobility Vehicle (UMV), a bicycle robot that autonomously raced across all sections of a kilometer-long, unstructured mountain bike trail at speeds exceeding $5\,\mathrm{m/s}$ and averaging $3.26\,\mathrm{m/s}$. In simulation, our policy fails up to an order of magnitude less often than hierarchical baselines while driving faster. We release our simulation and perception tools as open source.

1 Introduction



Figure 1. The Ultra Mobility Vehicle (UMV) autonomously navigates a mountain-bike trail in Zurich. Left: The robot corners a curve. Right: The robot traverses pump-track waves.

Off-road mountain biking demands a robust fusion of high-level world understanding, spatial planning, and split-second physical coordination. Deploying statically unstable systems, like a bicycle robot, on a mountain bike trail is inherently difficult because uneven, unpredictable terrain in outdoor forests makes it hard to plan and execute maneuvers. Friction fluctuates with changing ground conditions, seasonal shifts alter vegetation, and changing sun positions and cloud coverage introduce unpredictable shadows and harsh lighting variations. Therefore, *autonomous trail racing* emerges as a profoundly challenging, tightly coupled perception-and-control problem. It combines research challenges from navigation (Ravichandran et al. 2022; Rana et al. 2023; Lee et al. 2024; Werby et al. 2024; Rajvanshi et al. 2024), autonomous hiking (Miki 2025; Mattamala et al. 2025; Reed et al. 2025; Giusti et al. 2016), legged parkour (Hoeller et al. 2024; Cheng et al. 2024; Kim et al. 2025), high-speed autonomous ground vehicle racing (Kabzan et al. 2020; Cataffo et al. 2022; Betz et al. 2023), and drone racing (Loquercio et al. 2021; Kaufmann et al. 2023; Bahnam et al. 2026), while adding the complexity related to the unstable bike dynamics. Autonomous trail racing sits at the intersection of these domains, here summarized as "*time-optimal traversal of mountain bike trails relying strictly on onboard compute and sensing using wheeled, statically unstable platforms*". Each of the aforementioned fields addresses one or more relevant aspects, but no method provides a complete solution to trail racing. We tackle this challenge by (i) unifying local navigation and control and (ii) casting pathfinding as an explicit perception problem, without the need for a global map or destination-based planning.

The prior work by Bokser et al. (2026) presented the UMV, a two-wheeled bicycle robot capable of high-speed driving up to $8\,\mathrm{m/s}$ on flat ground. Two actuated Degrees

[1]Robotics and AI Institute (RAI), Zurich, Switzerland.
[2]Robotic Systems Lab, ETH Zurich, Zurich, Switzerland.
[3]Robotics and AI Institute (RAI), Cambridge, MA, USA.
[†] These authors contributed equally to this work.

Corresponding author:
Fabian Jenelten and Deepana Ishtaweera, Robotics and AI Institute (RAI), Zurich, Switzerland; Robotic Systems Lab, ETH Zurich, Zurich, Switzerland.
Email: fjenelten@rai-inst.com, dishtaweera@rai-inst.com

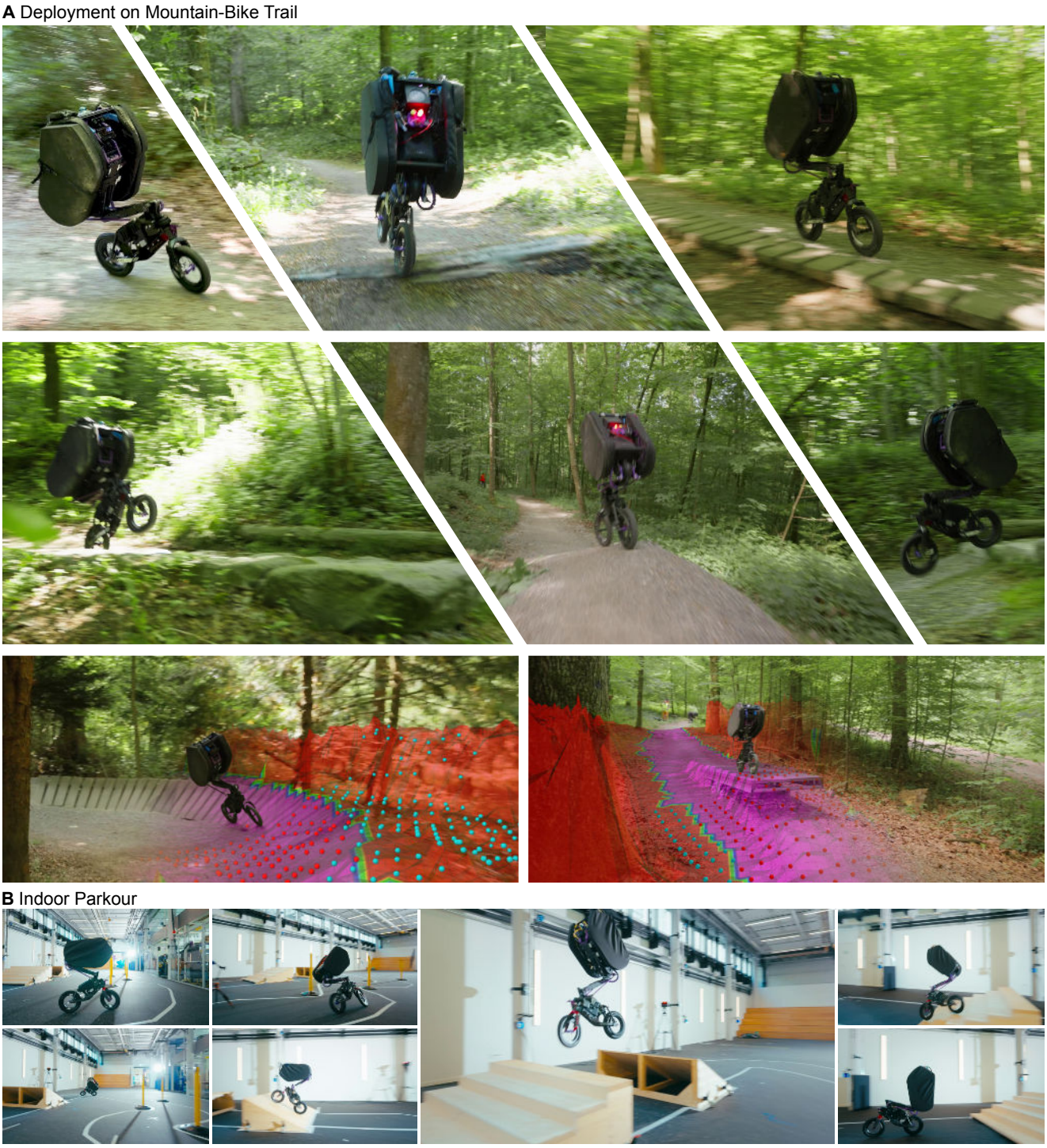


Figure 2. Autonomous trail racing and dynamic parkour. (A) The UMV autonomously navigates a mountain bike trail in Zurich at speeds exceeding $5\,\mathrm{m/s}$, relying exclusively on onboard perception and computation. The robot masters diverse obstacles, including roots, rocks, waves, and tabletop drops (first and second row). The policy jointly combines control and local navigation, while the two-layered grid map encodes a traversable path (third row). **(B)** With only the episode length and mesh as tuning degrees, the identical learning pipeline generalizes to agile parkour: A single policy navigates a slalom segment, accelerates up a ramp at $5\,\mathrm{m/s}$, clears a $1.2\,\mathrm{m}$ long gap, lands on a five-step staircase, and executes a track-stand, all within an $8\,\mathrm{s}$ window.

of Freedom (DOFs) in the upper body allow the robot to trigger vertical jumps reaching $1\,\mathrm{m}$ in height. While it demonstrated the physical feasibility of driving down a trail within a human-operated experiment, we consider the UMV the ideal platform to push toward full autonomy and high-speed racing. Expanding upon this hardware foundation, we integrate onboard perception, develop real-time visual world understanding, and learn vision-locomotive skills to present a comprehensive framework for autonomous trail racing (Fig. 3 and multimedia extensions E1 and E2).

We break this challenge down into two core scientific contributions. First, we introduce a platform-agnostic Markov decision process (MDP), herein referred to as Terrain Embedded Navigation (TENa) (subsection 3.2). By encoding the traversable path as a semantic layer directly into the terrain mesh and treating it as a soft

boundary during training, locomotion and local navigation collapse into a unified optimization problem. To support our specific application, we also present the Trail-Library (TL), an approach for the procedural generation of diverse, unidirectional synthetic trails that enables zero-shot deployment of racing policies (subsection 3.1). Second, to render the semantic path observable to the control policy during real-world deployment, we introduce Semantic Maps (SemMap) (subsection 3.3). This module constructs a two-layered, robot-centric elevation map containing both geometric and semantic path information. Leveraging a frozen Vision Foundation Model (VFM) paired with a lightweight decoder, incoming RGB images are processed at up to 90 Hz in isolation and fused with LiDAR range measurements at 30 Hz, producing a 2.5D elevation map that holds the path as a segmented trail.

We demonstrate autonomous deployment on a kilometer-scale, real-world mountain bike trail, sustaining average speeds of $3.26\,\mathrm{m/s}$ and peaking above $5\,\mathrm{m/s}$ (Fig. 2A). Compared to prior work Bokser et al. (2026), our work (i) introduces a single training environment applicable to both trail racing (subsections 4.4, 4.5) and parkour (subsection 4.2) with fixed reward weights; (ii) removes the human operator by following a trail observed by SemMap (subsections 4.3, 4.4); (iii) outperforms the teleoperated policy by a factor of 2.3 in speed on a real-world mountain-bike trail where baseline data is available (subsection 4.5); (iv) in simulation, fails up to an order of magnitude less often than hierarchical baselines built on the policy of Bokser et al. (2026), including one with a learned navigation policy, while driving faster (subsection 4.6); and (v) integrates the necessary state-estimation tools and onboard perception hardware (subsection 3.4). Compared to related works in autonomous hiking, we (i) remove the command interface by learning locomotion and local navigation as a joint optimization problem to achieve high-speed traversal; and (ii) demonstrate these capabilities on a statically unstable bicycle robot. By open-sourcing our core modules, TL and SemMap, this work establishes a comprehensive baseline for autonomous trail racing.[1]

2 Related Work

In robotics, the problem of *navigation* – defined as moving from an initial state to a target location – is typically solved by separating planning from control. This hierarchy relies on high-level representations such as trajectories, waypoints, checkpoints, or language. Most often, a *global navigation* planner acts on a pre-generated (Ravichandran et al. 2022; Rana et al. 2023; Lee et al. 2024; Werby et al. 2024; Miki 2025) or online-built graph (Shah et al. 2023; Rajvanshi et al. 2024). A *local navigation* planner then follows this global path while accommodating additional constraints, such as obstacle avoidance, before a downstream locomotion controller executes the high-level commands. To achieve higher overall performance and agility, locomotion and local navigation can be jointly solved by tracking pose targets (Loquercio et al. 2021; Rudin et al. 2022; Hoeller et al. 2024). Our work follows this trend by fully absorbing local navigation into the locomotion task, which, in simulation, reduces failures by up to an order of magnitude compared to hierarchical baselines (subsection 4.6).

The problem of *racing* – defined as minimizing lap time while adhering to a predefined path – has attracted considerable research, leading to a variety of control methods deployed across diverse robotic platforms. In autonomous *ground-vehicle racing*, the environment typically simplifies to a 2D track. Here, most approaches employ Model Predictive Control (MPC) (Goldfain et al. 2019) or follow a model-based planning-and-tracking paradigm (Srinivasan et al. 2021). For example, Liniger et al. (2015) introduced a mathematical optimization framework using a nonlinear Model Predictive Contouring Control (MPCC) controller for RC cars, a method later adapted for small-scale Formula-style racing cars (Kabzan et al. 2020). Leveraging offline optimization combined with online MPC (Betz et al. 2023) enabled a full-scale autonomous vehicle to win the Indy Autonomous Challenge. While end-to-end reinforcement learning (RL) (Fuchs et al. 2021; Song et al. 2021; Zhang et al. 2021; Stachowicz et al. 2023) offers potential deployment advantages, classical optimization-based methods continue to dominate ground-vehicle racing. This dominance is likely driven by their ability to provide strict mathematical guarantees for safety-critical spatial constraints.

In *aerial drone racing*, the 2D track-following problem expands into a 3D gate-traversal task. Early approaches tackled the mathematical challenge of generating time-optimal trajectories through multiple waypoints offline (Foehn et al. 2021). Later methods introduced online re-planning and gate-collision constraint satisfaction using a variant of MPCC (Krinner et al. 2024). Laying the groundwork for physical deployment, Foehn et al. (2022) detailed a comprehensive, modular autonomous racing system, establishing the architectural blueprint for modern drone racing pipelines. Building upon this vision backbone and its gate detector, recent work demonstrated that a learned control policy can outperform human expert pilots by directly observing the next gate location (Kaufmann et al. 2023). As noted by Song et al. (2023), the growing dominance of RL in this field is driven by its ability to directly optimize for the holistic task objective, coupled with increased robustness achieved through domain randomization. Given a predefined and observable track, our work draws direct inspiration from this research domain.

A closely related class of problems is *parkour*, traditionally defined as the physical discipline of traversing complex routes in the most efficient manner possible. In robotics, the term has been adopted to broadly describe highly dynamic locomotion through environments composed of obstacles and gaps on the scale of the robot itself. To navigate these terrains, most recent approaches train RL policies conditioned on explicit position commands (Rudin et al. 2025; Hoeller et al. 2024; Vogel et al. 2025; Kim et al. 2025). While our work uses a related goal-reaching reward structure, we do not condition the policy on goal positions. Instead, we encode the path itself in the terrain, so that the policy requires neither a goal location nor a command at deployment.

While most navigation and parkour tasks use geometric information readily obtained from range-based elevation

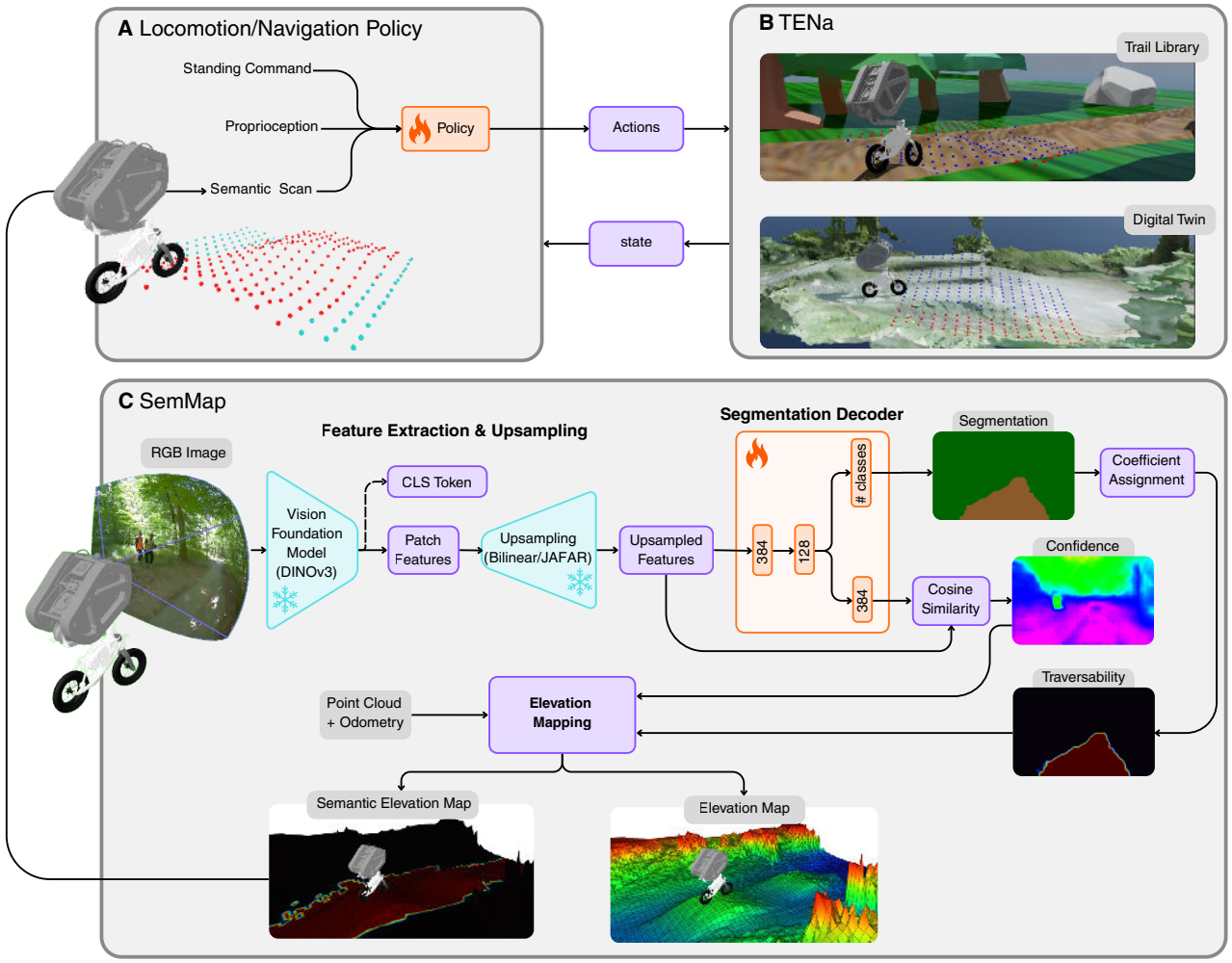


Figure 3. Approach overview. (A) *Deployment:* The policy, parametrized as Multi-Layer Perceptron (MLP) with three hidden layers, observes proprioceptive states, an operator-triggered stand command, and a semantic height scan. Joint action targets are computed at $50\,\mathrm{Hz}$ to control the rear wheel velocity, the fork angle, and the two DOFs of the torso. **(B)** *Training:* Our learning framework utilizes synthetic trails or high-fidelity digital twins and applies to both trail racing and parkour. **(C)** *Semantic pipeline*: An incoming RGB image passes through a frozen DINOv3 model and a frozen upsampler to produce high-resolution per-patch features. A lightweight trainable decoder maps these features to a per-pixel segmentation mask. A parallel reconstruction head, scored by cosine similarity to the original features, yields a per-pixel confidence score. A coefficient assignment operation converts the segmentation mask into a traversability image. Finally, an elevation mapping pipeline fuses traversability and confidence scores with odometry and point clouds at $30\,\mathrm{Hz}$.

mapping (Miki et al. 2022), estimating high-level traversability is fundamentally harder because it requires expert demonstrations or semantic information. In autonomous driving, one research field focuses on learning such top-down representations directly, lifting multi-view camera features into a common bird's-eye view (BEV) grid to predict dense semantic maps (Philion and Fidler 2020; Li et al. 2024; Liao et al. 2025). This paradigm was recently extended to off-road traversability and elevation estimation (Frey et al. 2024). In contrast, parameter-free geometric lifting has been shown to perform on par with learned strategies, with metric range sensing providing the dominant gains (Harley et al. 2023). A second line of research estimates traversability in a 2D space by leveraging human or robot demonstrations (Jung et al. 2024; Kim et al. 2024; Ægidius et al. 2025). To further reduce the labeling burden, some approaches are capable of online adaptation (Mattamala et al. 2025; Schreiber and Driggs-Campbell 2025). However, their typical update rates of $1.6\,\mathrm{Hz}$ to $8\,\mathrm{Hz}$ render these methods unsuitable for high-speed racing tasks. Similar to Łucki et al. (2025), demonstrating high-rate segmentation, we decode the general-purpose features of a modern VFM backbone into a semantically annotated image. We improve upon this baseline by pushing the rate of the complete semantic pipeline from $50\,\mathrm{Hz}$ to about $90\,\mathrm{Hz}$, by adding a per-pixel confidence score, and by enabling adaptation to a new environment from 20 labeled images within minutes on the robot's own computer (subsection 4.3).

A traversability estimator trained on a closed label set must also indicate when its predictions cannot be trusted. The maximum softmax probability (Hendrycks and Gimpel 2017) often remains high on anomalous inputs (Pinggera et al. 2016; Blum et al. 2021), while Monte Carlo dropout requires multiple forward passes (Kendall and Gal 2017). Single-pass alternatives combine evidential learning with latent density estimates (Ancha et al. 2024; Cai et al. 2024) or refine anomaly scores with a promptable segmentation

model (Zhao et al. 2024). Feature reconstruction offers a lighter signal, serving as segment-level confidence in Mattamala et al. (2025) and as per-pixel uncertainty in Schreiber and Driggs-Campbell (2025), the work closest to ours. We adopt this reconstruction branch but use cosine similarity to obtain a bounded score that directly weights map fusion, and exclude the background class from the reconstruction loss. Unlike softmax confidence, the resulting score flags unseen obstacles at a usable threshold (subsection 4.3).

Navigating unstructured outdoor trails is frequently explored under the umbrella of "autonomous hiking." Related methods typically do not focus on traversal time minimization and often rely on a hierarchical structure that bridges navigation with locomotion through a low-dimensional, human-engineered command interface (Miki 2025; Mattamala et al. 2025; Reed et al. 2025; Giusti et al. 2016). From an application perspective, this domain is well-represented by Miki (2025), which introduces an architecture that uses a vision-language model (VLM) to convert RGB images every ten seconds into pairs of labels and risk values. An open-vocabulary segmentation network converts these pairs into a semantic map, which is then projected onto a 2.5D elevation map and subsequently consumed by two hierarchical networks. Compared to this and similar methods, we focus on traversal speed, achieving average velocities of $3.0\,\mathrm{m/s}$ to $3.8\,\mathrm{m/s}$ and peak velocities above $5\,\mathrm{m/s}$. These traversal speeds are enabled by a unified MDP formulation, procedurally generated trail terrains, and the high update rate of our mapping pipeline. Our comparison with hierarchical baselines (subsection 4.6) indicates that a command interface, as used in these systems, limits reliability at high speed.

3 Method

We reinterpret both parkour and racing as instances of time-optimal traversal through structured or unstructured environments. Our approach does so by encoding the navigation task as a semantic layer within the terrain mesh (subsection 3.1), providing this privileged information directly to the policy, and treating it as a soft boundary during learning (subsection 3.2). During deployment, we utilize a dedicated perception system to reconstruct this semantic layer in real-time (subsection 3.3).

The overall approach is presented in Fig. 3 and detailed in the following subsections.

3.1 Terrain Generation

In prior work introducing UMV (Bokser et al. 2026), symmetrical Isaac-lab terrains were utilized to learn a generic driving policy, where the robot starts in the center of a terrain patch and is tasked to follow a velocity command. In attempting to add an autonomy layer (e.g., by naively learning a high-level target), we encountered several problems with this formulation. First, randomly sampled twist and standing commands are difficult to align with the local terrain properties. Second, the twist commands influence the terrain curriculum and therefore the skill progression, leading to an entangled tuning problem between terrain topology and task rewards. Third, the agents learn more skills than required to drive down a trail, whereas the relevant skills are underrepresented for reaching peak racing performance. The method presented here resolves these pain points by introducing unidirectional terrains and embedding the directional and standing commands within the terrain. We train policies on two complementary classes of semantic terrain meshes: procedurally generated mountain bike trails designed to learn generic downhill skills, and hand-composed Digital-Twins (DTs), i.e., high-fidelity representations of real-world environments, for learning specialized parkour maneuvers.

Trail-Library The TL is a modular framework for generating parameterizable mountain bike trails. Each terrain patch comprises four components: two platforms (the starting and ending zone), a connecting trail segment, and distinct skill elements. The resulting terrain is represented as a single mesh using the `trimesh` library (Dawson-Haggerty et al. 2019), which models geometry as triangles and stores visual information within the RGB channels. Further algorithmic details are illustrated in Fig. 4.

We maintain two parallel mesh representations: a semantically annotated mesh used for collision modeling and, optionally, a visually rich mesh for realistic rendering. Semantic information is explicitly stored in the vertex colors. The exact RGB values can be chosen arbitrarily, provided the mapping from RGB to semantic label remains bijective. In this work, we designate red $(1, 0, 0)$ for the primary trail, yellow $(1, 1, 0)$ for the starting zone, magenta $(1, 0, 1)$ for the final zone, and blue $(0, 0, 1)$ for illegal zones.

For the training environment, we generate 1000 randomized terrain patches, with each featuring two $10\,\mathrm{m}$ long platforms connected by a $36\,\mathrm{m}$ long trail, effectively yielding over $46\,\mathrm{km}$ of traversable tracks. The number of vertices per patch varies depending on the specific trail type, ranging from 2800 to 10 000, summing to a total of approximately 4.5 million vertices for the combined training environment.

Digital-Twin: While the TL offers an environment for learning generic mountain bike skills, it may not be well suited for learning specific parkour skills. We therefore introduce DTs, high-fidelity meshes replicating real-world parkour tracks. Visual meshes are constructed from a library of pre-meshed assets. Using a handheld capture rig, containing a stereo camera paired with a Livox Mid-360 LiDAR for LiDAR-inertial odometry (LIO) state estimation, we record dense imagery of physical obstacles in our laboratory. We then reconstruct the geometry and texture of ramps, stairs, slopes, and poles into high-resolution meshes.

Next, we derive a semantic version of each DT by manually embedding semantic classes directly into the RGB channels ("painting" the desired path). We note that the specific geometry and the manual annotations substantially influence the robot's emergent behavior. To reduce simulation overhead, we minimize its polygon count using a quadratic edge collapse decimation algorithm (Garland and Heckbert 1997). The process for designing the parkour mesh used for our control evaluation is detailed in Fig. 4F.

To further capture the realistic topography of natural outdoor environments, we also generate a large-scale DT of our target mountain bike trail. We reuse the same capture rig to reconstruct the entire outdoor trail as a single, continuous

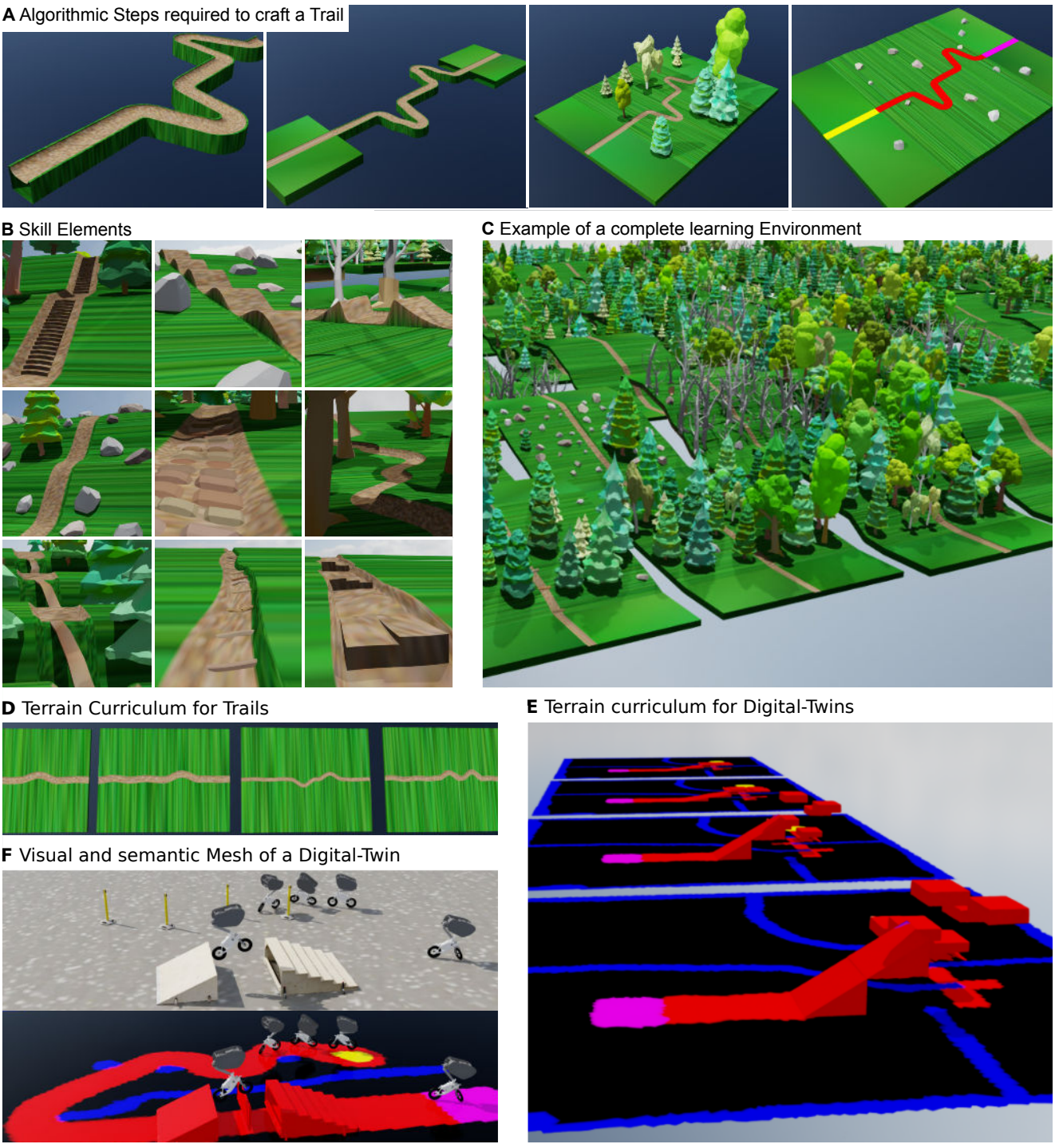


Figure 4. Mesh generation. (A) The procedural generation of mountain bike meshes is divided into four stages. First, a cross-section polygon is swept along a 4D sweeping path (x, y, z, and yaw). Second, two geometric platforms are welded to the start and end of the trail. Third, the lateral vertices are extended to align with the terrain patch size. Forest objects can optionally be placed. Finally, semantic information is embedded into the vertex colors. **(B)** Each trail segment may contain one or more skill element(s) of a unique type, such as stairs, pump-track waves, jumps, drops, rocks, curves, skinnies, roots, and ramps. **(C)** Multiple terrain patches are instantiated across a uniform grid to construct the learning environment. **(D)** Example of the terrain curriculum for a curved segment. **(E)** Example of the height-based curriculum for parkour. **(F)** A high-resolution ($17\,\mathrm{MB}$) *visual* mesh is constructed by composing pre-meshed objects. The corresponding *semantic* mesh is generated by down-sampling the geometry to $5\,\mathrm{MB}$ and manually encoding high-level navigation signals into the vertex colors.

mesh. Here, the semantic annotation is fully automated: we process the recorded camera streams through SemMap and use the corresponding LIO poses to project the predicted pixel classes directly onto the mesh vertices.

Terrain Curriculum Both the TL and the DTs employ a terrain curriculum. An agent is promoted to a higher difficulty level if it successfully reaches the final zone within the episode, and demoted to the previous level otherwise. For the TL, the parameterization of the skill elements scales linearly with the terrain level. Conversely, because DTs lack

this parametric flexibility, we implement a height-scaling-based curriculum: At the lowest difficulty, all mesh vertices are projected onto a flat, two-dimensional plane. As the level progresses, the vertical dimensions incrementally scale until they reflect the true geometry. Two examples of the terrain curriculum are illustrated in Fig. 4D and 4E.

Terrain Randomization For the TL, the terrain mesh is systematically randomized by overlaying the vertices with a sequence of geometric disturbances

$$\vec{v}_i \leftarrow \vec{v}_i + f_{\text{dist},j}(\vec{v}_i, \text{difficulty}), \quad \forall j = 0, 1, \dots \tag{1}$$

where $f_{\text{dist},j}$ can apply uniform noise, a constant slope, sinusoidal distortions, or other geometric perturbations. Additionally, decorative elements such as trees, roots, or stones can be placed adjacent to the trail. These perturbations break predictable structural patterns, preventing the policy from specializing to specific procedural trail outlines.

For the DTs, we apply both affine geometric transformations and semantic color randomizations. First, we construct an affine transformation matrix $A \in \mathbb{R}^{4\times4}$, initially set to the identity matrix. We then apply a random skew perturbation, $A[i, 0{:}3] \mathrel{+}= \mathcal{U}(-0.1, 0.1)$. Second, we embed the previously mentioned height curriculum directly into this transformation matrix, $A[2, 2] = \text{difficulty}$. Third, to simulate boundary bleed effects, we randomly perturb the semantic labels of the mesh: If a vertex is located on a semantic boundary, its color is randomly flipped with a probability of $p_{\text{switch}} = 0.08$.

3.2 TENa – The Learning Environment

Our racing and parkour policies are trained with on-policy RL where all interaction data is gathered in simulation. The MDP components remain identical for all tasks, unless noted differently. The core formulation is described in this subsection, while additional information can be found in the Appendix of sections C (optimizer settings), D (simulation settings), E (network architecture), F (action space), G (style rewards), and H (domain randomization).

Observations The policy network observes a vector of direct proprioceptive (i.e., state estimation independent) measurements. In addition, the policy receives a two-layer semantic height scan, as visualized in Fig. 3B. The elevation layer informs the agent about the topological properties of the local surroundings. The semantic layer acts as a boolean indicator of whether a given scan point belongs to the traversable trail. Unlike goal-conditioned legged parkour frameworks, our policy does not observe the target location directly. Instead, it is only informed whether the agent has reached the final zone. This design choice is motivated by the fact that the time-optimal racing trajectory does not depend on the remaining distance to the goal, assuming long-horizon energy management is negligible. A list of observation space variables is provided in the Appendix E.

Commands Rather than relying on twist or pose commands, the agent autonomously discovers its motion while remaining softly constrained to the trail boundaries. This avoids sampling command targets that might conflict with the local terrain. The only command in use is a boolean standing command, which is activated when a single, gravity-aligned ray cast from the base position intersects a mesh-face labeled as a standing zone.

Rewards We distinguish between *task* rewards, which define the core objective, and *style* rewards, which are required to facilitate smooth sim-to-real transfer. The full formulation of the style rewards is detailed in the Appendix G. The primary racing task consists of three objectives: (i) forward exploration, (ii) remaining within the trail boundaries, and (iii) coming to a stable track-stand at the final zone. The total task reward is defined as the weighted sum:

$$r = w_1 r_1 + w_2 r_2 + w_3 r_3. \tag{2}$$

High-speed racing emerges implicitly as a time-optimal behavior driven by the fixed episode length and the terminal standing reward. For all experiments and behaviors, we utilize the weights $w_1 = 1$, $w_2 = 0.2$, and $w_3 = 5$.

- **Exploration** (r_1): To mitigate the sparsity of the terminal standing reward, we encourage forward progression during the active racing phase:

$$r_1 = \begin{cases} 0, & c = 1, \\ \frac{v_{B,x}}{v_{\text{min}}}, & v_{B,x} \le v_{\text{min}}, \\ 1, & v_{B,x} > v_{\text{min}} \end{cases} \tag{3}$$

  where $v_{B,x}$ is the base heading velocity expressed in the base frame, v_{min} is the threshold velocity required to saturate the exploration reward, and the boolean indicator $c = 1$ denotes that the agent has reached the final zone. We set $v_{\text{min}} = 2$ for all parkour and racing environments. The only exception is the high-speed Allmend trail simulation, where we used $v_{\text{min}} = 10$.[2]

- **Survival** (r_2): To encourage the agent to remain within the valid track boundaries, we apply a dense survival reward:

$$r_2 = \begin{cases} 1, & \text{alive} \wedge \text{on trail}, \\ 0, & \text{otherwise}. \end{cases} \tag{4}$$

  We found that framing boundary adherence as a soft dense reward rather than a hard termination criterion leads to a slightly higher success rate (see Appendix R.1). To detect whether the agent is driving on the trail, we use a single, gravity-aligned ray to probe the terrain beneath it.

- **Track-stand** (r_3): The agent is incentivized to come to a complete rest upon reaching the target zone using:

$$r_3 = e^{-\|\xi_B\|_1} \cdot c, \tag{5}$$

  where ξ_B is the base twist expressed in the base frame.

Because the maximum standing reward per step ($w_3 + w_2$) exceeds the maximum reward collected while driving ($w_1 + w_2$), arriving earlier strictly increases the return within the fixed episode length, which induces time-optimal behavior without an explicit time objective or a path parameterization, as required by progress-based racing rewards. As discounting ($\gamma = 0.99$) attenuates this incentive far from the goal, the dense exploration reward r_1 provides the learning signal over long distances.

Terminations During training, episodes are terminated early under three specific conditions. First, an episode terminates if any joint violates its predefined position or velocity limits. Compared to Bokser et al. (2026), joint torque limits do not trigger termination; instead, they are enforced by clipping the output actuator signals before they are applied to the robot. Second, the agent is terminated if it navigates into designated illegal zones. These are user-defined semantic regions within the terrain mesh that act as hard boundaries, preventing the policy from exploiting shortcuts across the track. Third, an episode terminates if the robot's upper body makes physical contact with the semantic mesh environment. Actuator and joint position limits can be found in the preliminary work (Bokser et al. 2026).

Tuning We tuned all MDP components to achieve speeds between $3\,\mathrm{m/s}$ to $5\,\mathrm{m/s}$, an interval that we found yields a good trade-off between real-world success rate and high-speed racing. We can achieve higher speeds by increasing the weight on the track-stand reward in Eq. 5 or decreasing the episode length. Once fixed, we do not change these components[3] anymore, even when introducing a new task. Instead, we use the terrain mesh as a tool to shape the overall behavior of the robot. Two examples of shaping artifacts are shown in Fig. 4F. The beam placed within the gap forces the policy to learn an active jump rather than rolling through the gap, which would otherwise be the dominant discovered mode. The extended straight path connecting the turn with the entrance of the slope was designed as short as possible, but long enough to allow the robot to enter the slope in a fully upright position.

Training We trained and tuned all our policies on a single NVIDIA L4 GPU equipped with $24\,\mathrm{GB}$ of VRAM, utilizing 4096 parallelized environments. Notably, instantiating the entire curriculum of 1000 procedurally generated trail terrains within the Isaac ecosystem consumes approximately $4\,\mathrm{GB}$ of VRAM.

3.3 SemMap – Real-World Perception

SemMap bridges the sim-to-real gap by producing the fixed-size, egocentric 2.5D semantic elevation map that the policy requires for real-world deployment. It comprises two pipelines (Fig. 3C): First, a *semantic pipeline* maps each RGB image to three per-pixel-aligned outputs: a segmentation image, a traversability image, and a confidence image. Second, a *semantic elevation mapping* pipeline fuses the traversability and confidence images with the geometry built from point clouds and the robot pose. We describe the semantic pipeline and its label-efficient training, then the elevation mapping, and finally the system design that allows the whole pipeline to run in real-time and on-board.

Semantic Segmentation We cast the problem of semantic traversability estimation as supervised semantic segmentation followed by per-class scoring. As shown in Fig. 3C, an RGB image passes through a frozen VFM backbone, a feature upsampler, and a lightweight dual-head decoder that predicts per-pixel class probabilities together with a per-pixel confidence.

We extract features with a frozen DINOv3-s (Siméoni et al. 2025) backbone. Such VFM backbones provide dense, general-purpose features that transfer to downstream tasks through a lightweight decoder, here used for semantic segmentation. We resize the input RGB image to 320×512 pixels (height $\times$ width), which the backbone tokenizes into a 20×32 patch grid. This resolution is large enough to retain the spatial details needed by the decoder, yet small enough to keep inference fast. The resolution also matches the camera's $10{:}16$ aspect ratio, avoiding the information loss of square cropping.

Patch tokenization reduces the features to a coarse 20×32 grid, blurring object boundaries. We recover fine details with a learned feature upsampler (Couairon et al. 2025) that conditions on the image's low-level cues to produce content-aware, sharper features. The inference cost of such upsamplers depends heavily on the guidance image size and the upsampling factor. We bound this cost in two ways: we use a downscaled guidance image rather than the full-resolution image, and we upsample only partially (by a factor of 4 rather than the full $16\times$). The Appendix section O provides an ablation of how the upsampler affects inference speed and segmentation quality.

The decoder is kept deliberately lightweight because it is the only component trained per environment. Following Schreiber and Driggs-Campbell (2025), it first projects the upsampled features from 384 to 128 channels and then splits into two heads. A segmentation head maps the 128 channels to N_c per-pixel class logits, where N_c is the number of classes. A reconstruction head maps $128 \to 384$ channels back to the original backbone features and supplies the signal for confidence estimation.

Because the decoder is trained on a closed set of classes and a relatively small number of training examples, it will encounter unfamiliar terrain and objects at deployment. As a surrogate for epistemic uncertainty estimation, we obtain a per-pixel confidence from the cosine similarity between the original backbone features and those reconstructed by the reconstruction head. Where reconstruction succeeds, the content resembles the training distribution and confidence is high; where it fails, the pixel is likely out-of-distribution (OOD) and confidence is low. This yields an OOD signal that captures epistemic uncertainty without any negative or OOD examples during training, and we use it downstream to filter unreliable observations. We deliberately target epistemic rather than total uncertainty. Because the decoder is trained from a handful of labels, the dominant risk at deployment is unfamiliar content rather than label ambiguity. Aleatoric ambiguity, which concentrates at class boundaries (Kendall and Gal 2017), is instead absorbed downstream by design: the temperature-weighted softmax in Eq. (6) blends the traversability of ambiguous pixels rather than committing to a single class, and the confidence-weighted fusion averages out transient per-pixel noise as observations from successive frames accumulate in each map cell.

Traversability Estimation Segmentation classifies the surface type; traversability states whether the robot should traverse it. We keep these two concerns separate so that a single segmentation model can serve different robots and tasks. We assign each class c a weight $w_c \in [0, 1]$ and compute a per-pixel traversability $T_{i,j}$ as a temperature-weighted softmax

over the class logits $P_{c,i,j}$,

$$T_{i,j} = \sum_{c=1}^{N_c} w_c \frac{\exp(P_{c,i,j}/\tau)}{\sum_{c'=1}^{N_c} \exp(P_{c',i,j}/\tau)}, \quad (6)$$

where τ controls the sharpness of the class distribution. Because the weights w_c act only on the decoder output, the same network can be reused across robots and scenarios by reassigning w_c, with no retraining.

Label-Efficient Training Only the decoder and the class weights are learned per environment; for adaptation to be practical, it must succeed from a handful of labels. We reduce the labeling burden on two fronts: selecting the most informative images to annotate, and generating their labels with minimal manual effort.

To choose which images to label from a large unlabeled pool, we extract the DINOv3-s class token of every image and build a FAISS (Douze et al. 2025) index over these embeddings. We cluster the embeddings into as many clusters as the label budget and label only the representative of each cluster, the image closest to its centroid, so the labeled set spans the semantic diversity of the data while staying bounded in size.

Public datasets are used with their existing annotations. For data collected in a new environment, we use a promptable open-vocabulary model, SAM3 (Carion et al. 2025). However, we do this only offline to generate labels, since running such a model online would be computationally costly for on-board inference. We prompt it with points or a text description of each target class to generate masks, which are then converted into per-pixel class labels for training. SAM3 can produce spurious masks when a prompted concept is absent, so a lightweight human filtering step removes incorrect masks before training.

We train the decoder with a cross-entropy loss on the segmentation head and a mean-squared-error reconstruction loss on the reconstruction head, keeping the backbone and upsampler frozen. A dedicated background class absorbs all unlabeled and irrelevant pixels, such as sky or space, which sharpens the decision boundaries of the task classes. We mask this background class in the reconstruction loss so that confidence is calibrated only on the semantics relevant to the task.

Map Fusion The policy is trained on a 2.5D semantic elevation map; hence, at deployment the perspective-view traversability must be lifted into the same egocentric world parametrization. We build on top of an existing GPU elevation mapping system (Miki et al. 2022) and integrate the multi-modal elevation mapping (MEM) framework (Erni et al. 2023) as a library, which lets the segmentation and mapping stages share GPU memory and avoids CPU/GPU round-trips. Geometry is obtained from point clouds together with the robot pose from onboard state estimation; following the MEM update scheme, each grid-cell center is then projected into the traversability image to retrieve its traversability and per-pixel confidence, which are fused into the cell as described below. The egocentric map spans $15 \times 15\ \mathrm{m}^2$ at a cell size of $0.1\ \mathrm{m}$.

Each cell is updated by confidence-weighted fusion: An observation's update weight grows with its predicted confidence and with its proximity to the camera. Each cell accumulates a capped evidence weight, so that well-supported cells resist being overwritten by sparse, low-confidence observations. Observations below a small confidence floor are skipped. This soft weighting, in place of a hard threshold, keeps the map stable during fast turns, when the camera briefly points away from the trail, or at grazing angles where segmentation degrades. The full update equations and parameters are stated in Appendix section M. To keep the map egocentric without computationally costly shifting, we recenter it only once the robot moves past a distance threshold from the current center. Unlike a pre-mapped elevation map, online onboard mapping suffers from occlusions and a limited horizon. For the trail deployment, we therefore implemented an OpenCV-based inpainting filter, which inpaints both elevation and semantics based on grid cells that are observable in both depth and RGB. The joint inpainting allows us to flood-fill neighboring unobserved cells from adjacent observed cells by propagating geometric and semantic values (multimedia extension E9).

3.4 Deployment Architecture and Real-Time Execution

The full autonomy stack runs on ROS2 Humble (Macenski et al. 2022) with Zenoh (Corsaro et al. 2023) as the middleware, which in our deployments transported large messages such as images and point clouds more reliably than the CycloneDDS and FastDDS alternatives.

Control Our deployment stack leverages the ONNX runtime for efficient neural network inference, with policies exported via the *exploy* framework (Bellicoso and Wollschläger 2026). The trigger mechanism for the track-stand differs between deployment environments. During parkour, the standing command is inferred from the underlying "virtual" semantic map. For outdoor racing, however, the desired stopping locations would be arbitrary. To accommodate the standing feature, we bypass the semantic map and directly map a joystick button to the corresponding policy observation channel. Because this manual override logic is natively embedded into the exported ONNX graph via exploy, it requires no downstream modifications to the deployment code.

State Estimation Similar to Bokser et al. (2026), for the indoor parkour experiments we use a Motion Capture System (MoCap) system to track the robot's 6D base pose within the world frame and sample the elevation map. In our case, this offline map is generated directly from the semantic mesh by projecting the geometry and semantics into an RGB and a depth image. These two images are then converted into a colored MuJoCo height-field, which is subsequently sampled by a ray-caster to produce a two-layered elevation map.

For all other deployments, indoor and outdoor racing alike, state estimation runs onboard. We employ a custom LIO pipeline, inspired by KISS-ICP (Vizzo et al. 2023). Our adapted method extends the baseline with voxel normals for robust scan alignment and provides a stable, robust state estimate at frame rates far exceeding those of input LiDAR data, which is important when traversing complex

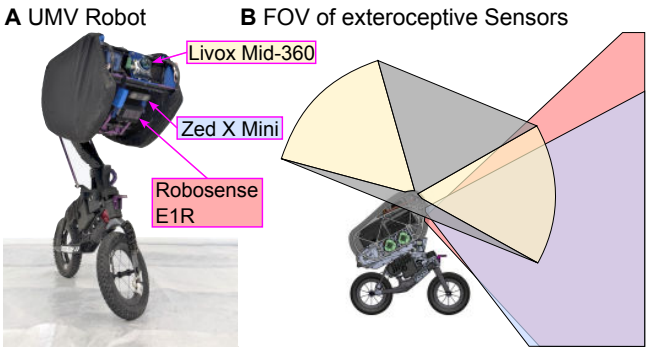


Figure 5. Hardware overview. (A) The modified UMV robot in a track-stand position showing the three different visual sensors. The robot has four actuated DOFs: the rear wheel, the fork, and the lower and upper torso joints, whereas the front wheel is unactuated. **(B)** The field-of-view of the cameras when the robot is in a resting position.

real-world environments at high speeds. The pipeline aligns incoming scans from a Livox Mid-360 LiDAR against an internal local geometric world representation to estimate a consistent 10 Hz 6D pose. We fuse these poses with the raw IMU readings in a sliding-window factor graph to produce a high-frequency 6D pose estimate, which we use to motion-compensate both the primary Livox and the secondary Robosense E1R point clouds. Finally, the motion-compensated point clouds and pose estimates serve as input to SemMap, while also getting merged into an internal LIO local geometric world representation used for subsequent scan alignment.

Perception SemMap receives three input streams over ROS2: RGB (30 Hz to 60 Hz), gravity-aligned pose (30 Hz to 1000 Hz), and point clouds from stereo or LiDAR (10 Hz to 30 Hz). Perspective-view traversability alone requires only the RGB stream. To consume these high-rate streams in real time, we adopt the multi-process, shared-GPU-buffer architecture of the visual perception engine (Łucki et al. 2025): a C++ front-end handles subscriptions and time synchronization, while the semantic and elevation-mapping pipelines run as separate GPU workers, keeping all data in GPU memory (Appendix O.1). Finally, we export the networks from PyTorch to TensorRT via ONNX, which gives a 3- to 4-times speedup on the NVIDIA AGX Orin and raises the maximum throughput of the semantic pipeline, covering segmentation, traversability, and confidence, to about 90 Hz. In deployment, we operate the camera at 30 Hz and match the pipeline accordingly. The GPU-based (CuPy) elevation-mapping fusion node publishes the complete semantic elevation map at the same rate: the control policy reads the most recently published map at 50 Hz, decoupling control from perception throughput. The 30 Hz map-update rate is sufficient. At the peak speed of 5 m/s, successive map updates are separated by less than 0.17 m, under two grid cells, and geometry updates are bounded by the 10 Hz LiDAR, whose range far exceeds the map extent.

Hardware We build on the UMV (Bokser et al. 2026), a previously developed bicycle robot, and equip it with the exteroceptive sensors and compute required for onboard perception. Low-level control runs on the robot's existing onboard UP Xtreme i12, while perception and state estimation run on a separate NVIDIA AGX Orin 64GB added for this work. The sensor suite comprises a ZEDX Mini stereo camera that supplies RGB to SemMap, a Livox Mid-360 LiDAR used primarily for LIO, and a Robosense E1R LiDAR that provides the dense point cloud for elevation mapping. The full assembled robot is visualized in Fig. 5.

All sensors are mounted to a custom Kevlar roll cage, which has been ruggedized with waterjet-cut foam to withstand high-impact crashes during testing. The braided Kevlar core provides lightweight robustness, while the foam damping reduces shock loads on the sensors. The sensors themselves are mounted on a single 3D-printed CF-PLA support, enabling rapid positional iteration while maintaining alignment between sensors during testing. On the bike side, the native nylon covers shield the bike electronics from dirt and debris, while custom TPU bumpers on the fork and frame protect the carbon fiber from gouges. This robust design protects both the sensors and internal components from severe mechanical shock, minimizing downtime and enabling sustained, full-day testing during policy deployment.

4 Experiments and Results

The following results are structured to reflect our methodological progression, advancing from isolated simulation validation toward fully autonomous, high-speed real-world deployment. While all developed modules are conceptually platform-agnostic, we show the main results on the UMV robot.

4.1 Simulation

We first verified that TENa generalizes across a diverse set of DTs by showing different emerging behaviors for parkour and racing tasks. To this end, we trained specialized policies in each environment and evaluated their performance, measured by success rate. During training, we enabled all domain randomizations required for real-world deployment, whereas for evaluation, these randomizations were disabled to establish nominal performance. In the appendix I, we further demonstrated cross-embodiment generalization in simulation by applying our identical learning framework to the ANYmal-D quadruped for both trail and parkour.

Parkour We designed 17 parkour tracks featuring stairs, wall-rides, gaps, drop-ins, ramps, and boxes. These tracks serve two purposes: First, they let us design and tune a general MDP formulation so different tasks can be solved through a shared interface. Second, once implemented in the real world (see subsection 4.2), these tasks allow us to identify the control sim-to-real gap in terms of measurable success rate.

We found that our MDP formulation successfully discovers completing behaviors for all environments, i.e., behaviors that solve the last difficulty level with non-zero success rate. We used only episode length as a tuning parameter (see Appendix C). Across 1000 simulated runs with randomized initial conditions, we report success rates between 95 % and 100 %, depending on the parkour type. A run is deemed successful only if the robot navigates the



Figure 6. Parkour and trail racing learned using DT meshes. All tracks share a similar core concept: Semantic information is embedded into the mesh, creating a unique trail connecting two virtual platforms, the starting and the final zone. This trail is made observable to the policy and used to shape the gradient landscape. **(A)** The time-lapse images show individual parkour tracks together with the emerging locomotion behavior. **(B)** High-speed racing on a reconstructed real-world mountain bike trail, reaching average speeds of $9.81\,\mathrm{m/s}$. The colored dots illustrate the annotated height scan, with blue labeling the trail and red labeling non-trail.

entire course and smoothly transitions into a stable track-stand on the final zone. A time-lapse of 11 selected parkour environments is presented in Fig. 6A, whereas in multimedia extension E3 we show the complete set of 17 parkour tracks.

Compared to previous work (Bokser et al. 2026), we extend the skill set of UMV from track-stand, active jumps, hops, drive, and flips to further include cornering, wall-rides, gaps, drop-ins, hip-jumps (jump and yaw), stair landing, and slaloms. We further increase the active jump height from $1\,\mathrm{m}$ to $1.75\,\mathrm{m}$; however, we note that this result may not transfer zero-shot to the real robot due to operation close to both joint and torque limits.

Trail Our target deployment mountain bike trail, the *Allmend trail*, measures $969\,\mathrm{m}$ in length with a total elevation drop of $59\,\mathrm{m}$ and a maximum steepness of $-18°$. At the time of writing, the all-time Strava leaderboard for this combined segment reported a record of $105\,\mathrm{s}$ (averaging $9.5\,\mathrm{m/s}$), with an all-time median of $5.3\,\mathrm{m/s}$ across 2861 human riders.[4]. Due to a pedestrian hiking path, the mountain bike route is artificially divided into two parts. We reconstructed a high-fidelity DT of the upper segment, which spans a distance of $419.1\,\mathrm{m}$ with an elevation drop of $21\,\mathrm{m}$ and a maximum steepness of $-5.4°$. The DT captures all skill elements, including a wall-ride, two drop zones, a seesaw, a rock bed, two skinnies, a slalom section, and several root networks (Fig. 9).

We trained a specialized policy on this DT and evaluated this policy in simulation across 1000 trials with randomized initial conditions. Due to the nature of the training setup, we will refer to the resulting policy as "specialized racing policy". Performance was quantified by the time required to reach the end of the trail and the overall success

Table 1. Performance comparison between human riders (real-world data) and TENa (simulated data) on the Allmend trail. Human velocity metrics were retrieved from Strava. To further demonstrate generalization across diverse systems, we also include the traversal metrics of ANYmal. We note that the simulation runs are averaged over the upper part of the Allmend trail, whereas Strava provides measurements averaged over the entire trail.

Agent	Time	Velocity	Success Rate
Human (Med.)		$5.3\,\mathrm{m/s}$	unknown
Human (Rec.)		$9.5\,\mathrm{m/s}$	unknown
UMV	$42.7\,\mathrm{s}$	$\mathbf{9.81}\,\mathrm{m/s}$	$89\,\%$
ANYmal	$89.5\,\mathrm{s}$	$4.68\,\mathrm{m/s}$	$81\,\%$

rate. Only those agents contribute to the total racing time that survive the episode. The agents completed the trail with an average time of $42.7\,\mathrm{s}$ and an $89\,\%$ success rate. Assuming the traveled distance matches the official segment length provided by Trailforks[5], the agent maintained an average racing speed of approximately $9.8\,\mathrm{m/s}$. While our results suggest that TENa can, in simulation, reach speeds comparable to the fastest human riders, this comparison is indicative only: the Strava times cover the entire trail, whereas our DT covers the flatter upper section. Any behavior robustification typically counteracts peak performance, and thus, we do not expect this policy to transfer to the real world. Some snapshots of the simulated run are provided in Fig. 6B, a summary of the results is shown in Table 1, and multimedia extension E4 visualizes the entire run in a top-down view.

If episode durations are relatively large, as in the case of the Allmend trail (Tab. S1), the standing reward becomes more sparse in time and hence, more difficult to discover. We found that increasing the minimum velocity instead provided a better handle for tuning the base velocity. However, this only works for tracks that can be entirely traversed at high speeds and do not require large velocity leaps, as, e.g., the parkour tracks. Consequently, to achieve the large velocities in this particular experiment, we increased the minimum velocity in the reward function 3 from $2\,\mathrm{m/s}$ to $10\,\mathrm{m/s}$.

Without this change, the traversal time increases to $75.2\,\mathrm{s}$ (corresponding to $5.57\,\mathrm{m/s}$ average speed) and the success rate reaches $100\,\%$.

4.2 *MoCap Assisted Parkour*

In the second phase of validation, we isolated and bridged the control sim-to-real gap for the UMV by deploying it in a perception-independent environment. This isolated setup allows us to quantify the gap by comparing the success rates achieved in simulation against those on the physical track. To this end, we trained a policy on the parkour track displayed in Fig. 4F, one of the 17 parkour meshes. It features a four-pole slalom, a slope for lift-off, and a staircase for touch-down. The maximum height of both the slope and the stairs is $0.555\,\mathrm{m}$. The slope has an inclination of $24.8°$, while the stairs have a step width of $0.27\,\mathrm{m}$ and a step height of $0.09\,\mathrm{m}$. A gap of $1.2\,\mathrm{m}$ separates the slope and the stairs. These elements are specifically designed to abstract the core challenges of our target mountain bike trail (e.g., tight curves, jumps, and uneven ground) into a controlled indoor environment.

To break the dependency on perception, we used a MoCap system to localize the robot within a pre-mapped DT. For each test, the robot is initialized at the starting location in an upright position. Once the controller was engaged, the robot autonomously raced through the track and navigated to the dedicated stopping location. A representative successful run is visualized in Fig. 7A, and four successful runs are shown in multimedia extension E5. The robot entered the slope at a peak base velocity of $5\,\mathrm{m/s}$, lifted off at $3.3\,\mathrm{m/s}$, remained airborne for $0.4\,\mathrm{s}$, re-accelerated to $5\,\mathrm{m/s}$ on the stairs, and decelerated within $0.5\,\mathrm{s}$ to a stable track-stand, completing the sequence in $8\,\mathrm{s}$.

Out of 10 consecutive trials on our final test day, the robot completed the run with a stable track-stand 8 times. We recorded two falls, both occurring immediately after the landing phase on the stairs. We identified the landing as the most fragile phase of the track, as success mostly depends on beneficial initial conditions set at lift-off. Due to a lack of a roll joint in the upper body, the robot cannot recover from large roll angles after landing. For comparison, in the equivalent simulated parkour track, we achieved a $99\,\%$ success rate, where near-optimal lift-off conditions are usually achieved. The difference in success rate is what we consider to be the sim-to-real gap.

Throughout the development, we have observed three other failure modes not present in simulation: (i) If the robot accelerates too fast into the turn before lift-off, it tends to slip off the ramp and fall into the gap. (ii) If the robot drives too fast over the slope, it may land on the flat ground, completely clearing the staircase. The resulting impact forces are typically too large for the upper body to absorb, leading to joint position or torque violations. (iii) If the robot drives too slowly over the ramp, the rear wheel will hit the front edge of the stairs before landing. While this strategy works in simulation, it leads to violent hardware destruction in the real world (e.g., exploded tires, broken rims, or even broken carbon frames). Those issues were mitigated through careful domain randomization (Appendix section H) and track tuning over the course of several months, aimed at rendering the policy sufficiently conservative to not excite these failure modes, but aggressive enough to complete the track.

"Table jumps" were previously demonstrated in Bokser et al. (2026), showing UMV jumping on top of a table and then driving down from it. Both setups use MoCap and human-designed tracks; ours replaces waypoint following with the unified formulation and adds a slalom, a gap and a staircase landing. We note that the rewards and tuning parameters proposed for the "table jumps" led to slightly conservative behaviors failing our parkour track, but still provided a meaningful starting point for our tuning process (Appendix H).

4.3 *Evaluation of Perception*

We evaluate the SemMap pipeline in isolation, quantifying its robustness, confidence signal, label efficiency, inference speed, and generalization across environments and tasks.

Qualitative Evaluation We assessed SemMap qualitatively across diverse visual conditions and environments; Figure 8 shows this characterization, and multimedia extension E9

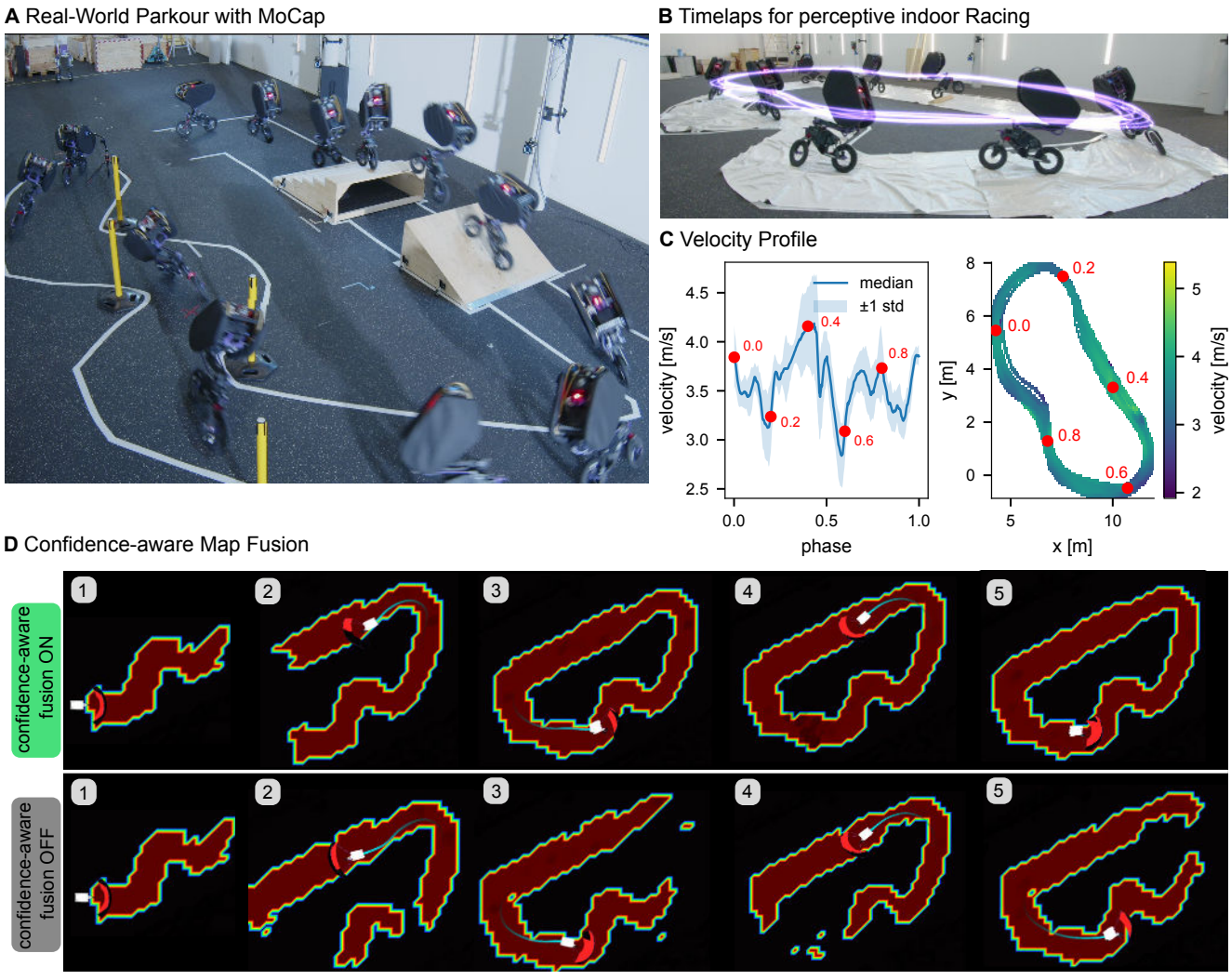


Figure 7. Indoor parkour and indoor racing. (A) *MoCap-assisted indoor parkour.* The parkour track features a slalom segment, a ramp for lift-off, and a staircase for landing. **(B)** *Autonomous indoor racing using onboard perception.* The UMV navigates a high-contrast circuit utilizing only full onboard compute and perception, completing 130 consecutive laps over a duration of 891 s. **(C)** *Telemetry of the indoor racing experiment.* The left plot illustrates the median base velocity with ± 1 standard deviation. The right plot displays a 2D positional trace of all laps overlaid with a velocity heatmap. Red markers denote five phase positions (0: starting location, and 1: full lap). **(D)** *Confidence-weighted map fusion during indoor racing.* Top: With confidence-weighted fusion. Bottom: Low-confidence observations from high-speed turns progressively corrupt the map.

presents the pipeline, the cross-domain and zero-shot evaluations, the inpainting and fusion mechanisms, and the new-environment adaptation workflow in video form. On natural trails, the input imagery is subject to harsh sunlight, glare, cast shadows, and motion blur, yet the trail remains segmented in each of these instances (Fig. 8A). The resulting per-pixel traversability projects into a coherent semantic elevation map over the trail's skill elements (Fig. 8B). We further evaluated across six datasets spanning natural trails, an indoor racing track, an industrial floor, and off-road and urban scenes (Fig. 8C). Across all six domains, only the lightweight decoder is trained per dataset while the DINOv3-s backbone and upsampler stay frozen and shared, showing that one pipeline recipe transfers across natural, indoor, industrial, off-road, and urban scenes. In each case, a Principal Component Analysis (PCA) of the JAFAR-upsampled (Couairon et al. 2025) features already separates the scene into semantically meaningful regions with sharp object boundaries, and the decoder maps these features to task classes, from which a weighted softmax layer produces the per-pixel traversability.

Confidence and OOD Detection SemMap's per-pixel confidence enables confidence-weighted map fusion, down-weighting unreliable observations before they enter the map, which is essential at racing speed. We demonstrated this on the robot during indoor racing (Appendix section N, Fig. 7D): with fusion enabled, the semantic elevation map stays consistent in each lap. With it disabled, the low-confidence predictions produced when the camera swings away from the trail in fast turns progressively corrupt the accumulated map. We do not target state-of-the-art OOD detection, but a signal that is nearly free at inference, needs no OOD examples, and is retrained with the decoder; we therefore compare against the softmax confidence of standard segmentation networks. Because no standardized OOD benchmark exists for off-road trails, we further quantified the underlying confidence signal on the widely used Lost and Found benchmark (Pinggera et al.

2016), which contains real lost-cargo obstacles on urban roads. Trained on Cityscapes (Cordts et al. 2016) and evaluated at three classes on the 1,111 test images that contain obstacles, the SemMap decoder predicts 35 % of obstacle pixels as traversable before any confidence filtering, compared to 60.2 % for DeepLabV3+ (Chen et al. 2018) and 69.4 % for SegFormer-B0 (Xie et al. 2021), i.e., 1.7× and 2.0× fewer (per-image paired Wilcoxon $P < 0.001$, rank-biserial $r = 0.76$–0.85). Its confidence further reaches a 95 % true-positive rate at a threshold of 0.72, whereas both softmax baselines require 1.0, leaving no tunable range. This confidence-weighted fusion reduces the perception errors the policy must tolerate at deployment.

Label Efficiency Adapting SemMap to a new environment is computationally cheap, as the decoder is the only component trained per environment. To quantify this label efficiency, we evaluated on the off-road RELLIS-3D benchmark (Jiang et al. 2021). At a budget of 20 labeled images, our frozen-backbone decoder (∼100 000 trainable parameters) reaches $33.9 \pm 1.4\%$ mean Intersection over Union (mIoU) (mean ± SD, $n = 3$ seeds), whereas the fully fine-tuned baselines reach $23.9 \pm 0.9\%$ (DeepLabV3+) and $24.1 \pm 3.5\%$ (SegFormer-B0); with 100 labeled images, only 3 % of the training set, the decoder already matches its own full-dataset performance. These absolute values are low because the benchmark's 20 classes are imbalanced; pixel accuracy exceeds 88 % even at the lowest budgets. On our own outdoor trail dataset, 20 labeled images reach 82.6% mIoU on the held-out test split, while the full training set of 28 763 images reaches 90.0 %, indicating that the additional data buys robustness margin rather than base functionality. Because the decoder converges in a few epochs on frozen features, adaptation can run on the robot's own computational resources, an NVIDIA AGX Orin: starting from a 5-minute recording of an unseen environment, we produced a deployed model in under 15 minutes of combined human and compute time, comprising 2 minutes of automated data curation, under 5 minutes of SAM3-assisted labeling (Carion et al. 2025), and 4 minutes of on-device training and TensorRT export. The full study is shown in the Appendix (section L).

Three factors explain why the frozen-backbone decoder outperforms the fine-tuned baselines when few labeled images are available. First, the decoder trains approximately 100 000 parameters on top of the frozen backbone and upsampler, whereas the fine-tuned baselines adapt 3.7M and 42.0M parameters from the same 20 images; the gap narrows at larger budgets (Fig. S3), indicating a difference in sample efficiency rather than capacity. Second, the gain depends on the pretrained representation rather than on freezing alone, as the frozen ResNet-50 is the weakest configuration. A PCA of the upsampled DINOv3 features already separates the scene into class-consistent regions (Fig. 8C), such that the decoder acts as a near-linear readout, and the same structure is what renders the reconstruction-based confidence informative. Third, additional labels improve robustness rather than base performance: on the outdoor trail dataset, increasing the training set from 20 to 28 763 images raises the mIoU from 82.6 % to 90.0 % with diminishing returns (Table S8), suggesting that semantic coverage through clustered image selection matters more than label volume.

Inference Speed The pipeline must keep pace with a robot moving above 4 $\mathrm{m\,s^{-1}}$. Our recommended JAFAR×4 configuration runs model inference, including the confidence head, at 99 fps (mean over $n = 100$ timed iterations, latency SD $\leq$ 0.33 ms), while improving mIoU on the ORFD test set (Min et al. 2022) by 4.1 percentage points over bilinear upsampling. Higher upsampling factors add under 0.2 percentage points while reducing inference speed to roughly a third. Including input preprocessing, the traversability computation, and upsampling of all outputs to input resolution, the complete semantic pipeline runs at about 90 Hz (11.1 ms per frame), or at about 163 Hz with bilinear upsampling (Appendix section O).

Generalization To probe generalization, we apply the decoder, trained only on our own outdoor trail data, zero-shot to the TRIM-1 sequence of the Grand Tour dataset (Frey et al. 2026), a forest gravel trail recorded under cloudy late-autumn conditions by an ANYbotics ANYmal-D quadruped, differing from our recordings in sensor, platform, viewpoint, season, and location. Since no labels exist for this location, the assessment remains qualitative: the decoder still produces coherent traversability, indicating that a single decoder can serve a broad class of trails rather than a single site. We further adapted the decoder to three unseen Grand Tour sequences, an alpine snow field, an indoor building, and an ice cave, each from 20 clustered, SAM3-labeled images. Trained on the AGX Orin in 1.5 min to 3 min, the decoders reach 77 % to 91 % mIoU on held-out frames (Appendix Section L), confirming that the workflow yields a functional decoder on new sites.

SemMap is robot-agnostic by design, observing the world only through the camera; its training data was collected primarily with a handheld capture device and, in some cases, a smartphone, independent of the deployment robot. Adapting only the decoder, we additionally deploy it RGB-only on a Unitree G1 humanoid for taped no-go-zone avoidance in a factory, where traversability is defined by the tape boundary rather than the surface and is thus invisible to geometry (Appendix Section P).

Limitations These deployments also delineate the limits of our design choice. The zero-shot Grand Tour transfer shows that a decoder generalizes across trails of a similar appearance; however, trained on a small, closed dataset, it does not generalize universally, and sustained appearance shifts, even within the same environment, are handled by adding samples from the shifted distribution, which the label-efficient training makes cheap. In practice, we extended the training set once: as the vegetation changed with the season, we recorded additional footage and retrained the decoder through the same rapid adaptation workflow (Appendix section Q). We also observed transient segmentation errors, such as the trail briefly appearing to fork into two paths for one or two frames; none of these required intervention.

4.4 Indoor Racing

Our next objective was to close the control-and-perception loop. We approached the trail racing challenge by training a

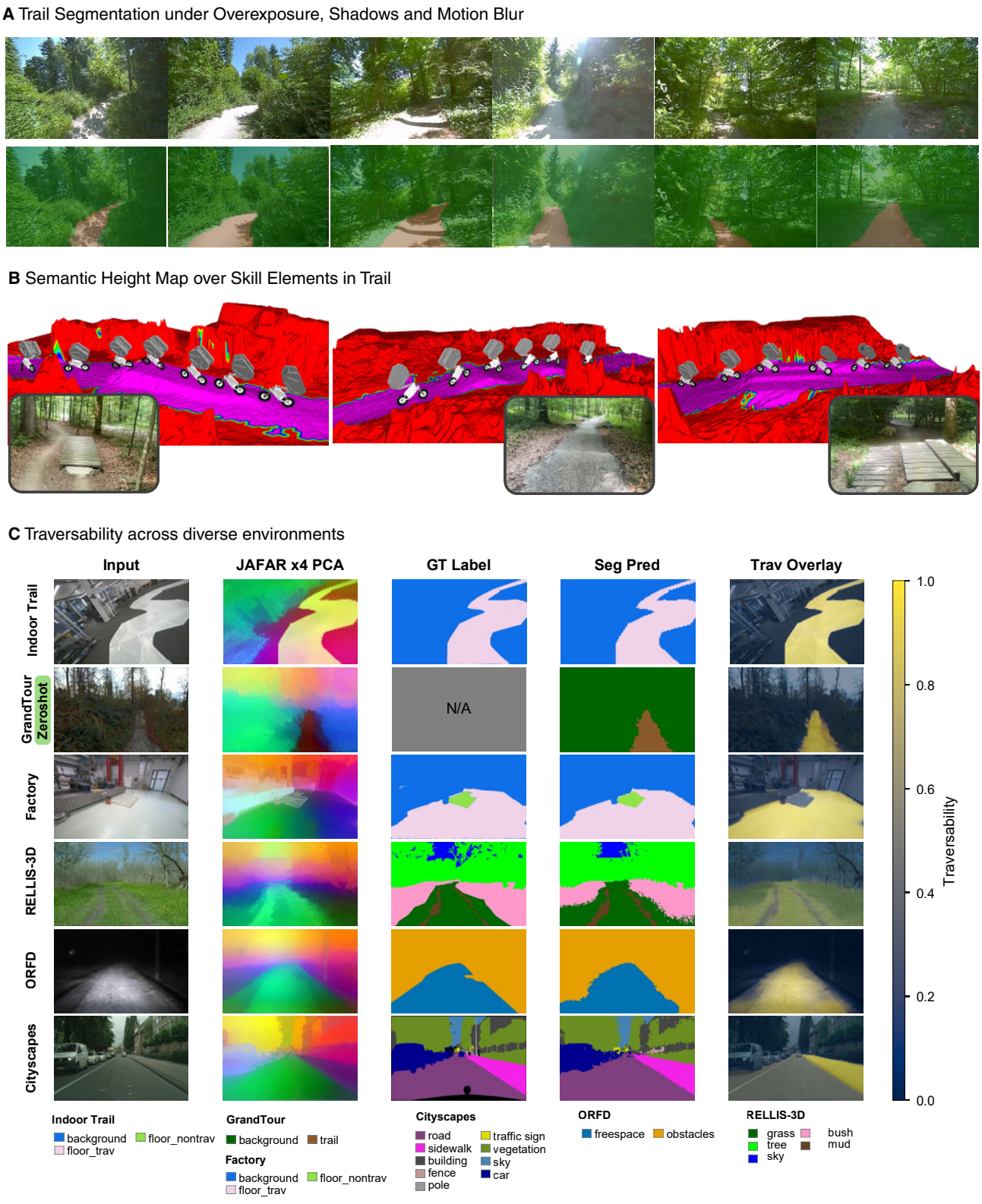


Figure 8. Real-world semantic perception with SemMap. (A) Input frames (top) and segmentation overlays (bottom) under glare, cast shadows, and motion blur. **(B)** Semantic elevation map over a seesaw, pump-track waves, and a wooden tabletop. Insets: reference photos, not onboard. **(C)** Results over six datasets. The frozen DINOv3-s backbone and upsampler are shared; only the lightweight decoder is trained per dataset. GrandTour is evaluated zero-shot, hence no ground-truth labels (N/A). Rows, columns, and legends are labeled in-figure.

racing policy across 46 km of synthetic trails generated by the TL. As this policy was not "specialized" to a particular track as done in the previous experiments, we refer to it as "generic racing policy".

For physical deployment, we constructed a narrow racing track using high-contrast white fleece, featuring two sharp turns and a slalom segment, essentially outlining the previous parkour track without the slope and the stairs. Additionally,

we introduced a $0.1\,\mathrm{m}$ tall wooden incline just before the first turn (Fig. 7B). The MoCap-state estimator and ground-truth map are replaced with our LIO solution and SemMap. The robot was initialized inside the track boundaries, and the racing controller was engaged from a track-stand position. In total, the UMV completed 130 continuous laps, maintaining an average speed of $3.54\,\mathrm{m/s}$ and reaching a peak velocity of up to $5\,\mathrm{m/s}$ at the end of the straightaway. The average lap time measured was $6.85\,\mathrm{s}$. Further details are provided in Fig. 7C, whereas multimedia extension E6 documents all laps.

This experiment allowed us to further tune additional MDP components related to the perception interface (e.g., the noise model in Appendix E or the terrain distribution), and to quantify the sim-to-real gap. In simulation, with all domain randomization terms disabled, we achieved a lap time of $6.1\,\mathrm{s}$, as opposed to $6.85\,\mathrm{s}$ in the real world. We attribute the difference in speed to unmodeled or inaccurate actuator dynamics and rolling resistance.

Due to the controlled indoor environment, the missing occlusions, and the flat ground simplification, we expect the results to constitute an upper bound of performance for real-world mountain bike trail deployment. We note that this generic racing policy was not capable of traversing the simulated Allmend trail due to the appearance of narrow slalom poles, which do not show up in the height scan, but lead to termination upon impact. As during the time between mapping and deployment, the "slalom" segment was restructured to a "wave" segment (consisting of seven pump-track waves), we did not attempt to solve this issue, and instead, directly deployed the policy on the Allmend trail.

4.5 Trail Racing

We deployed our fully integrated pipeline on the Allmend trail in Zurich using the generic racing policy synthesized previously. As outlined in Fig. 9, the upper section features wide areas populated with human-made skill elements and natural obstacles. In contrast, the lower section transitions into a flow-trail, characterized by several waves, steep descents, and narrow segments that are heavily overgrown with dense vegetation. In the following, we summarize the results from three independent successful deployment days. The reported mean velocities are computed from the norm of the state estimator's base velocity, and traveled distances are estimated as the integral of this value.

Our proposed racing framework traversed the Allmend trail across six continuous runs. These runs are successful runs that cover the entire trail, selected from a set of 79 runs, with a median traveled distance of about $74\,\mathrm{m}$ (excluding the second deployment day; see Appendix section Q for a complete analysis). The longest continuous segment of this list measured $220\,\mathrm{m}$ and was traversed with an average velocity of $3.8\,\mathrm{m/s}$, reaching a peak velocity of $5.2\,\mathrm{m/s}$. A complete breakdown of the individual runs can be found in Table 2. Accordingly, the UMV completed $946\,\mathrm{m}$ of real-world mountain bike trail across six runs, sustaining autonomous racing speeds above $3\,\mathrm{m/s}$ consistently. The longest autonomous run measured $76\,\mathrm{s}$ (Appendix section Q), which is about 12 times shorter than the autonomous run achieved indoors. The difference reflects

Table 2. Performance on the real-world Allmend trail. The table shows three runs on the upper and three runs on the lower part. The checkpoints refer to the skill elements shown in Fig. 9. Velocities are computed as the mean over the run. The total average velocities are computed as the total distance traveled divided by total time traveled.

part	checkpoints	velocity	distance	time
upper	1	$3.52\,\mathrm{m/s}$	$126\,\mathrm{m}$	$35.8\,\mathrm{s}$
	2, 3	$3.01\,\mathrm{m/s}$	$138\,\mathrm{m}$	$45.8\,\mathrm{s}$
	4. . .9	$3.04\,\mathrm{m/s}$	$189\,\mathrm{m}$	$62.2\,\mathrm{s}$
lower	10	$3.80\,\mathrm{m/s}$	$220\,\mathrm{m}$	$57.9\,\mathrm{s}$
	11	$3.09\,\mathrm{m/s}$	$134\,\mathrm{m}$	$43.4\,\mathrm{s}$
	12, 13, 14	$3.10\,\mathrm{m/s}$	$139\,\mathrm{m}$	$44.8\,\mathrm{s}$
total	entire trail	$3.26\,\mathrm{m/s}$	$946\,\mathrm{m}$	
	upper part	$3.15\,\mathrm{m/s}$		
	lower part	$3.37\,\mathrm{m/s}$		

the complexity of an outdoor deployment in wild forests. We provide a summary of all failure cases in the following subsection 4.7.

Based on video footage in Bokser et al. (2026), we estimate that the human-operated UMV averaged $1.36\,\mathrm{m/s}$ on the upper section of the trail.[6] Averaged over this same segment, our autonomous policy was 2.3 times faster.[7] We attribute the performance difference to two factors: the removal of the intermediate command interface and a training distribution tailored to the task. A navigation policy learned on the same trail terrains reduced, but did not close, this gap (subsection 4.6), indicating that the command interface itself accounts for a substantial part of the difference.

Bokser et al. (2026) also demonstrated velocity tracking of up to $8\,\mathrm{m/s}$ on flat indoor ground under teleoperation, which marks the hardware limit of the UMV. On the trail, our policy operated well below this limit, at average speeds between $3.0\,\mathrm{m/s}$ and $3.8\,\mathrm{m/s}$ per run. Here, speed is bounded not by the hardware but by the terrain, the perception horizon, and the trade-off between speed and robustness discussed in subsection 5.2.

As demonstrated in multimedia extensions E1 and E2, the entire trail was annotated as traversable, allowing local path selection to emerge naturally from the policy rather than being strictly dictated, a skill which we refer to as "local navigation". For instance, the robot chose to drive over the seesaw rather than taking an available bypass. This behavior likely stems from the training distribution, where simulated slopes typically span the full width of the track, biasing the agent to traverse rather than circumvent such features. Conversely, the robot tended to drive alongside skinnies instead of balancing on top of them. This aligns with our observation made in simulation, where the policy only reacts to skinnies upon front-wheel impact – a limitation caused by the sparsity of the height scan, which renders narrow geometric features largely unobservable prior to contact. At tabletop drops, we imposed no target descent points; instead, the robot dynamically committed to a descent trajectory based on its approach angle. Finally, the policy was not forced onto the wall-ride, as the adjacent gravel path was also segmented as traversable. Because the policy learned to corner on both flat ground and banked surfaces during training, it fluidly transitioned between these two modes



Figure 9. Deployment on a mountain bike trail. The Allmend trail, a kilometer-long technical mountain bike route located in Zurich, is outlined in red; square/circle markers denote the start/end location. The dotted blue line marks the pedestrian path that cuts the trail into two segments. The UMV navigates diverse skill elements at the numbered checkpoints: (1) rock bed, (2) sequence of seven waves, (3) two skinnies, (4) rocky tabletop drop, (5) wooden tabletop drop, (6) wall-ride, (7) tight turn, (8) natural transversal log, (9) locked seesaw forming a ramp, (10) natural hill, (11) extended and overgrown curve, (12) S-shaped descending curve, (13) steep descent, and (14) cliff-like trail segment with a sharp drop-off on one side.

during deployment, dictated solely by its entry velocity at the obstacle.

4.6 Comparison with a Hierarchical Baseline

To compare against a hierarchical alternative, we retrained the velocity-tracking policy of Bokser et al. (2026) with our height-scan observation, keeping the remaining training setup, including the terrain generator, identical to Bokser et al. (2026). Commanded linear velocities ranged from $0\,\mathrm{m/s}$ to $3\,\mathrm{m/s}$ and angular velocities up to $1.5\,\mathrm{rad/s}$ in magnitude. We trained the policy for 16,000 learning iterations, and refer to it as "baseline". For completeness, we also include the original blind locomotion policy of Bokser et al. (2026), which we call "baseline† ". Each locomotion policy was paired with one of two trail-following controllers: one that steers toward the centroid of the trail-labeled scan points, and a pure-pursuit controller with a look-ahead distance of $2\,\mathrm{m}$. In both cases, the trail follower controller implemented a fixed heading velocity of $1.5\,\mathrm{m/s}$ or $3\,\mathrm{m/s}$.

All policies, including ours, were evaluated on 850 terrain patches from the TL, comprising 10 instances of each terrain type across five difficulty levels. We removed all trees to avoid OOD observations for the perceptive baselines, which, if anything, favors the baseline. For each difficulty level, we simulated 4,000 robots with all domain randomizations disabled and recorded the survival rate, the time to reach the final platform, the time spent off the trail, and the average speed. The episode length was $30\,\mathrm{s}$ for the $1.5\,\mathrm{m/s}$ target and $16\,\mathrm{s}$ for the $3\,\mathrm{m/s}$ target. The results are summarized in Tab. 3.

The centroid-based controller performed slightly better than pure pursuit, and the blind locomotion policy outperformed its perceptive counterpart. We attribute the latter to the difference in training terrains: the perceptive baseline receives height scans on the TL terrains that lie outside its training distribution, whereas the blind policy does not depend on them. This underlines the importance of matching the training terrain to the deployment environment, which the TL provides.

The decline in success rate at higher target velocities suggests that a competitive command interface would also need to modulate the linear velocity based on the local terrain. We therefore trained a separate navigation policy on top of the blind baseline, using the same trail terrains as for our generic racing policy. The navigation policy observes the same signals as the generic racing policy and outputs twist commands at $50\,\mathrm{Hz}$, the update rate of the locomotion policy, with actions clipped at $1.5\,\mathrm{rad/s}$ and $3\,\mathrm{m/s}$. We sample heading velocity targets for the downstream locomotion policy uniformly and keep them constant within a window of $2\,\mathrm{s}$ to $7\,\mathrm{s}$. The navigation policy is encouraged to output the same values as the heading target, but can deviate from it to maximize survival rate. The locomotion policy remains fixed during training and is treated as part of the environment, i.e., no gradients are propagated through it. As reward, we use the exploration reward (Eq. 3) with $v_{\min} = 1\,\mathrm{m/s}$, the survival reward (Eq. 4), and a reward for twist tracking of the same shape as the track-stand reward (Eq. 5) with a weight of 2. Episodes terminate when the locomotion policy triggers a termination (Bokser et al. 2026) or when the agent reaches the final platform; as we are only interested in traversal, there is no standing phase.

With a learned navigation policy, the hierarchy became the most competitive baseline (Table 3). The best such setup, baseline-$3^{\dagger\top}$, survived in $85\,\%$ to $46\,\%$ of the runs at a realized average velocity of $2.2\,\mathrm{m/s}$, whereas our generic policy survived in $99\,\%$ to $91\,\%$ of the runs at $3.3\,\mathrm{m/s}$. Despite driving about $50\,\%$ faster, our policy thus failed roughly an order of magnitude less often (6 to 15 times, depending on the difficulty level), with the gap widening on the harder terrain levels. The baseline also spent substantially more time off the trail. Unlike the heuristic controllers, the learned navigation policy maintained its survival rate at the higher speed limit, presumably because it learned to modulate its speed.

4.7 Failure Case Analysis for Trail Deployment

We conducted 79 autonomous runs on four deployment days between early April and late May. We exclude the second day (April 24), on which testing was stopped after 10 runs, from the quantitative analysis; its failures are summarized in Appendix section Q. The following analysis covers the remaining 67 runs, excluding the two runs that ended in a track-stand. We then categorized the primary failure modes, which are exemplified in Fig. 10.

The most frequent cause of critical balance loss (23/67) occurred when the robot deviated from the physical track (10A). This trajectory deviation was generally caused either by the control policy failing to safely reject external disturbances (10Ai), or by the semantic perception pipeline generating permissive traversability boundaries (10Aii). In some scenarios (20/67), the robot suffered unrecoverable wheel slip on dry gravel (10C), which happened during tight cornering (10Ci, ii) or off-camber descents (10Ciii, iv). Despite dedicated efforts in parameterizing ground friction and choosing conservative friction intervals (Appendix section H), the problem of wheel slip could not be completely resolved. Another prevalent failure mode (14/67) involved front-wheel or upper body collisions with narrow obstacles (10B), such as roots (10Bi), skinnies (10Bii, iv), or thin tree trunks (10Biii). These obstacles are too small to appear in the height scan and are mostly invisible to the policy.

Finally, perception artifacts occasionally (10/67) prevented the elevation mapping pipeline from reconstructing the geometry of the upcoming path (Fig. 10D). We identified two primary mechanisms driving these mapping failures, both rooted in the 2.5D elevation map recording the maximum height within each grid cell. First, trees and dense woody vegetation are registered as tall, solid geometries, constructing untraversable walls across the path (Fig. 10Di). Second, tall grass yields out-of-distribution height maps: although compliant and traversable, it is captured at its full height, which can narrow or even block the perceived corridor (Fig. 10Dii). Both effects are aggravated at high speed, when the policy pitches the torso backward for dynamic stability and angles the onboard range sensor upward, reducing the density of ground returns, forcing the elevation map to extrapolate from sparse data.

Table 3. Evaluation of the generic racing policy and comparison with hierarchical baselines. The "ours" policy is the one deployed on the last Allmend deployment day; its heading base velocity, averaged over all terrain levels, is $3.3\,\mathrm{m/s}$. The baseline locomotion policies are conditioned on a twist command, generated either by a controller that steers toward the centroid of the observed trail (no symbol), by a pure-pursuit controller (*), or by a learned navigation policy ($^\top$), with a heading target velocity of $1.5\,\mathrm{m/s}$ (baseline-1.5) or $3\,\mathrm{m/s}$ (baseline-3). The dagger (†) denotes the blind locomotion policy.

	Level	Survival Rate	Time to Goal (s)	Time Not on Trail (s)	Avg Speed (m/s)
ours	0	**0.99**	**8.82**	**0.013**	**3.29**
	1	**0.98**	**8.88**	**0.015**	**3.30**
	2	**0.98**	**9.00**	**0.011**	**3.30**
	3	**0.95**	**9.11**	**0.013**	**3.33**
	4	**0.91**	**9.61**	**0.038**	**3.30**
baseline-1.5	0	0.85	19.7	0.767	1.45
	1	0.74	19.9	1.380	1.46
	2	0.60	20.3	1.860	1.45
	3	0.41	20.3	2.780	1.48
	4	0.22	22.1	6.230	1.41
baseline-1.5*	0	0.86	19.8	0.820	1.46
	1	0.72	20.1	1.47	1.45
	2	0.57	20.4	2.19	1.44
	3	0.37	20.7	3.58	1.45
	4	0.20	22.6	7.43	1.38
baseline-1.5†	0	0.86	19.4	0.270	1.46
	1	0.78	19.6	0.389	1.49
	2	0.72	19.9	0.390	1.45
	3	0.60	20.4	0.795	1.46
	4	0.44	21.2	1.420	1.44
baseline-1.5$^{\dagger\top}$	0	0.83	22.3	0.162	1.28
	1	0.78	22.4	0.334	1.29
	2	0.71	22.2	0.341	1.32
	3	0.64	22.3	0.350	1.34
	4	0.45	22.6	0.620	1.38
baseline-3	0	0.83	9.8	0.881	2.95
	1	0.62	10.1	1.160	2.90
	2	0.42	10.2	1.720	2.88
	3	0.26	10.4	2.320	2.88
	4	0.12	10.7	3.830	2.91
baseline-3*	0	0.74	9.90	0.962	2.94
	1	0.53	10.2	1.66	2.85
	2	0.33	10.2	2.02	2.88
	3	0.20	10.5	2.56	2.86
	4	0.09	11.1	4.94	2.80
baseline-3†	0	0.85	9.77	0.162	2.96
	1	0.67	9.93	0.378	2.94
	2	0.49	10.1	0.793	2.90
	3	0.33	10.3	1.540	2.91
	4	0.19	10.5	2.130	2.96
baseline-3$^{\dagger\top}$	0	0.85	13.1	0.133	2.18
	1	0.79	12.3	0.210	2.35
	2	0.72	12.9	0.212	2.28
	3	0.58	14.2	0.255	2.12
	4	0.46	14.7	0.488	2.11

5 Discussion

5.1 Performance Gain Through Specialization

Our results are in line with recent legged parkour frameworks, which indicate that specializing control policies

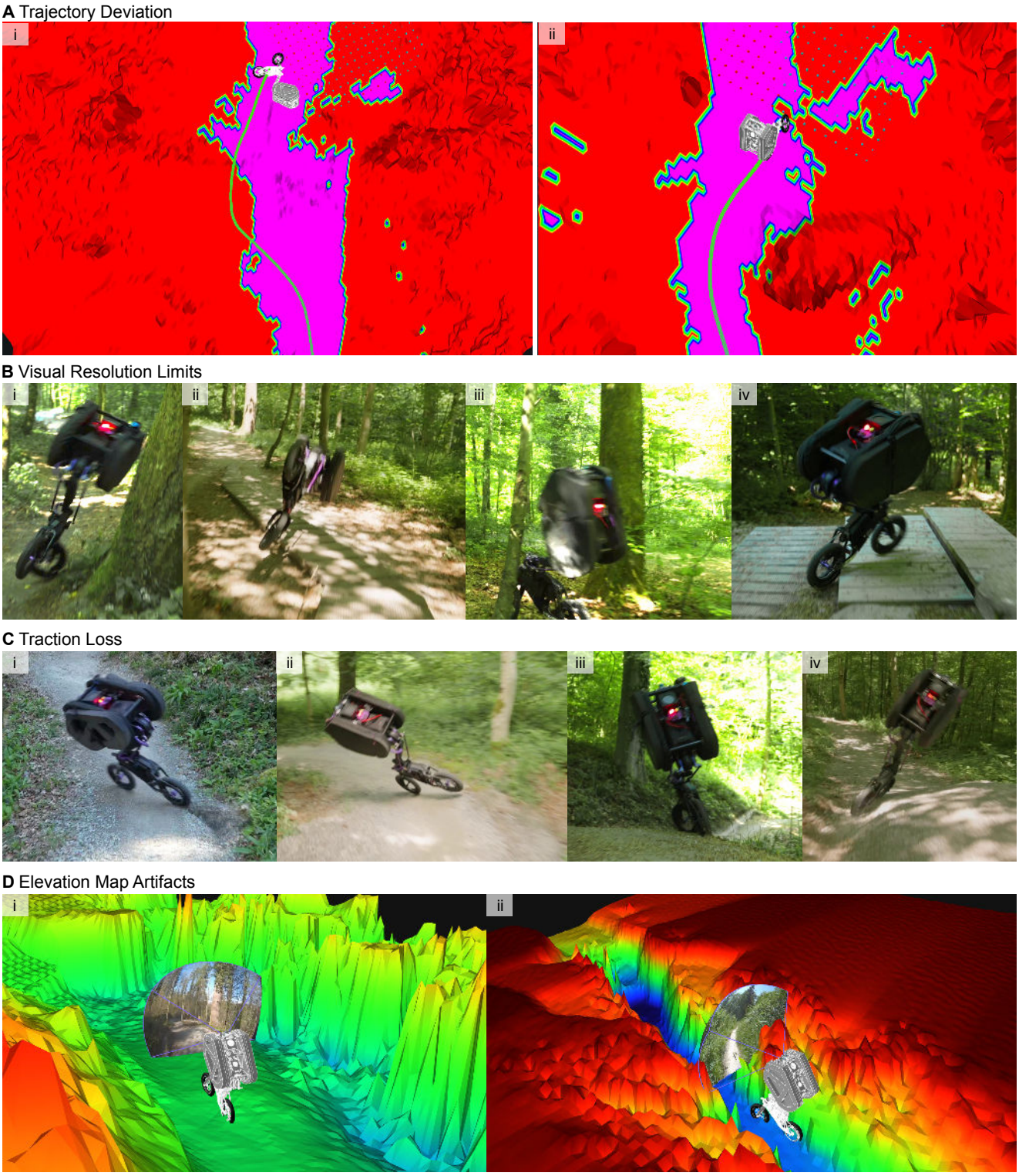


Figure 10. Primary failure modes during real-world deployment. Each snapshot captures the state immediately preceding a critical loss of balance. **(A)** The robot exited the trail and failed to route back into the safe zone. The green path visualizes the base position. (i) The robot deviates from track boundaries and aggressively steers inwards, leading to slip. (ii) A wrongly perceived trail causes the robot to drive off the track. **(B)** The robot collided with an obstacle whose height or width falls below the spatial resolution of the elevation map. **(C)** The rear wheel entered a slip phase. **(D)** Trees are registered as untraversable walls in the elevation map (i), while tall grass narrows the traversable path (ii). Insets display the corresponding onboard camera perspective.

to known geometric topologies significantly enhances peak performance (Rudin et al. 2022). When training the specialized racing policy, the average simulated speed reached $5.57\,\mathrm{m/s}$ on the Allmend trail (subsection 4.1). Comparing this to the generic real-world deployment on Allmend ($3.26\,\mathrm{m/s}$) (Tab. 2), the performance gain of geometric specialization corresponds to a roughly 1.7-fold increase in average velocity (simulated vs. real-world).

While the generic racing policy collides with the slalom poles, the specialized policy is capable of completing the same trail with a $89\,\%$ (for $v_{\min} = 10$) to $100\,\%$ (for $v_{\min} = 2$) success rate (subsection 4.1). The specialized policies seem

to remember the critical turns, likely based on surrounding landmarks. While this presents a potential solution to the problem of "blind spots", it also introduces a new weak link for deployment, as those geometric or semantic landmarks are not clearly defined nor guaranteed to persist over time.

5.2 Sim-To-Real Gap

In simulation, the specialized racing policy with dedicated tuning reached an average speed of $9.8\,\mathrm{m/s}$ on the upper Allmend section (subsection 4.1), whereas real-world deployment velocities ranged from $3.0\,\mathrm{m/s}$ to $3.8\,\mathrm{m/s}$ (subsection 4.5). This discrepancy does not measure the sim-to-real gap, as the two results stem from different policies: the specialized policy was trained on a single DT with the minimum exploration velocity raised to $10\,\mathrm{m/s}$, whereas the generic racing policy deployed in the real world was trained across $46\,\mathrm{km}$ of randomized synthetic trails. The real-world speeds reflect a deliberate operating point rather than a hardware or simulation limit: we tuned the MDP to trade off peak speed for the robustness required to survive unstructured terrain, sensor noise, and perception errors.

We therefore quantify the sim-to-real gap only where the same policy runs on the same track in simulation and on hardware. For control alone, the MoCap parkour policy succeeded in $99\,\%$ of simulated runs and in 8 out of 10 real runs (subsection 4.2). For the full perception-and-control loop, the generic racing policy completed the indoor circuit in $6.1\,\mathrm{s}$ per lap in simulation and $6.85\,\mathrm{s}$ on hardware, corresponding to an $11\,\%$ lower average speed (subsection 4.4). For the outdoor trail, no like-for-like comparison exists (subsection 4.4).

Deploying the specialized racing policy to the physical trail is outside of the scope of this work, as it would introduce several compounding challenges, revealing an unsolved sim-to-real gap at higher velocities. First, previous work identified the UMV's physical speed limit on flat indoor ground as $8\,\mathrm{m/s}$ (Bokser et al. 2026), establishing a hardware ceiling below the simulated peak velocity. Second, the height scan provides a relatively small lookahead window of roughly 3 meters, and the information within is relatively sparse. Third, the specialized policy is inherently overfitted to the geometry of a single DT (subsection 5.1). The simulated mesh was generated from point clouds stripped of vegetation, and vertex simplification smoothed out fine-grained geometric disturbances like small rocks and roots. Because the real-world semantic elevation map contains these rich details, a specialized policy relying on memorized geometry would likely fail.

5.3 Intersections and Forks

When encountering trail intersections or forks, SemMap segments multiple valid paths simultaneously. Because the control policy was not exposed to branching trajectories during training, it lacks an implicit mechanism for global routing. Empirically, we observe that the policy resolves this ambiguity by defaulting to the path that requires the least steering effort (e.g., multimedia extension E2 at 1:17). While this emergent behavior allows the system to smoothly clear simple intersections at random, it creates a vulnerability when the perception pipeline generates spurious semantic artifacts, which can inadvertently divert the robot into untraversable terrain (subsection 4.7).

5.4 Comparison to Autonomous Hiking

Our work intersects closely with the field of autonomous hiking (sections 1 and 2). While our perception capabilities can be evaluated against existing methods, a direct comparison of the control policies is difficult. Specifically, prior work (Miki 2025; Mattamala et al. 2025; Reed et al. 2025; Giusti et al. 2016) prioritized traversal success over time-optimal performance. Consequently, to the best of our knowledge, these studies do not report the explicit velocity metrics required for a quantitative comparison.

5.5 Limitations and Future Work

Our extensive real-world deployments also expose clear limitations of the current system. We group these limitations into four core categories.

- Field deployment revealed perception vulnerabilities caused by tall grass and overhanging vegetation being erroneously fused into the elevation layer. These geometric artifacts resulted in paths that are artificially narrowed or even blocked entirely by the appearance of "phantom" vegetation walls. Two parallel directions could resolve this: Filter vegetation from the point cloud before fusing, or remove the explicit height representation altogether by learning locomotion from RGB streams.

- Exposed roots, loose stones, thin tree trunks, and slippery gravel frequently caused unrecoverable force impacts or traction loss. Relevant visual details are captured in neither the sparse depth nor the semantic layer, but are rather considered as external disturbances to the controller. Substituting the sparse height scan with rich, high-dimensional visual latent features represents a vital path forward to enable human-competitive trail racing. The hardware could further benefit from a roll joint, potentially improving reactive behaviors of the upper body.

- While cross-terrain generalization ensures broad policy robustness, it also restricts the maximum achievable racing velocities. We hypothesize that future strategies will bridge this gap by deploying specialized policies trained for the precise geometric nuances of isolated trail segments.

- As our work focuses explicitly on "local" navigation, a foundational bottleneck of our proposed framework is its implicit reliance on a unique and continuous path. Elevating this architecture to true cross-country navigation capabilities requires integrating sparse directional guidance, potentially sourced from topological maps.

5.6 Lessons Learned

Beyond the quantitative results, the development of the system yielded several lessons that we expect to transfer to other agile platforms and tasks.

- **Shape behavior through the environment, not the reward.** With fixed reward weights, the terrain mesh and its semantic annotation became the main design tool: a beam placed in a gap, for instance, turned a rolling solution into an active jump (subsection 3.2).
- **Generalization costs speed.** Policies trained on diverse procedural trails transferred robustly but raced well below the speeds of policies specialized to a single DT, which in turn rely on landmarks that are not guaranteed to persist (subsection 5.1).
- **At speed, perception reliability matters as much as accuracy.** Without confidence-weighted fusion, low-confidence observations during fast turns progressively corrupted the map (Fig. 7D), and visual appearance shifted noticeably within a single season, so cheap in-field adaptation proved essential (subsection 4.3).
- **Observability, not dynamics, limited the real-world performance.** Most failures stemmed from features the policy could not perceive, such as thin obstacles below the resolution of the height scan and vegetation registered as solid geometry (subsection 4.7).
- **Close one gap at a time.** Validating control, perception, and their integration separately, from MoCap parkour to indoor racing, allowed us to attribute failures to a single component before deploying the full system outdoors.

6 Conclusion

In this work, we successfully tackled the challenge of autonomous trail racing based on the two core contributions, TENa and SemMap. Rather than serving as a domain-specific solution, TENa is a unified MDP formulation that scales across racing and parkour. Because the task objective is encoded in the terrain, the task reward and its weights are shared across all tasks, and behaviors can be shaped solely through terrain semantics and episode length. Furthermore, by collapsing the planning-and-tracking hierarchy into a single learning framework, TENa reduces deployment complexity.

Because the task objective is embedded in the terrain, navigation simplifies to following the perceived trail. Leveraging a frozen backbone and a lightweight dual-head decoder, SemMap produces such a semantic 2.5D map at $30\,\mathrm{Hz}$ (subsection 3.4). Because the map aggregates observations over time and a learned confidence signal down-weights unreliable predictions, transient perception errors are absorbed rather than propagated (section M). As the decoder is lightweight and environment-specific, adaptation to a new deployment requires minutes of training on data that can be collected without a robot.

While our framework is platform-agnostic (Fig. S2 and S9), we validated it on a statically unstable bicycle on a kilometer-scale mountain bike trail. The technical complexity of this endeavor was twofold. First, unlike stable quadrupeds or ground rovers, the UMV must constantly balance. Second, the outdoor trail is subject to seasonal changes, leading to fluctuations in the visual appearance and terrain friction. To systematically mitigate these sim-to-real risks, we employed a progressive validation scheme:

1. Using two different robotic platforms, the UMV and ANYmal, and 19 different tasks (17 parkour tracks, a $419\,\mathrm{m}$ long trail, and over $46\,\mathrm{km}$ of generic trails from the TL), we demonstrated that agile skills emerge naturally as a function of the environment's geometry and semantics, without the need for task-specific reward tuning.
2. We isolated the control sim-to-real gap by deploying a locomotion policy in a MoCap-instrumented parkour, achieving an $80\,\%$ success rate on a track comprising a slalom, a ramp jump, and a staircase landing.
3. We evaluated SemMap in isolation, demonstrating robust onboard reconstruction of terrain semantics, label-efficient adaptation competitive with fully fine-tuned baselines, and confidence-weighted fusion for reliable sim-to-real deployment.
4. We closed the perception-and-control loop by deploying a generic racing policy to autonomously navigate a narrow, twisted racing track, where it completed 130 continuous laps at an average speed of $3.5\,\mathrm{m/s}$. We further quantified the sim-to-real gap by comparing against simulated runs.
5. We demonstrated full onboard autonomy by traversing the complete kilometer-long Allmend trail within six runs, reaching average segment velocities of up to $3.8\,\mathrm{m/s}$ and peak velocities larger than $5\,\mathrm{m/s}$. We compared the results to a teleoperated locomotion policy and showed an increase in average velocity of a factor of 2.3 on the upper Allmend trail. In simulation, our generic policy failed up to an order of magnitude less often than hierarchical baselines built on the policy of Bokser et al. (2026), including one with a navigation policy learned on the same trail terrains, while driving faster.

Notes

1. TL (https://github.com/isaac-sim/IsaacLab/pull/6539) and SemMap (https://github.com/rai-opensource/sem_map).
2. We note that for the high-speed simulation experiment, the achieved average speed is similar to the minimum velocity of the exploration reward, essentially rendering the exploration reward a task reward.
3. The only exception is the simulated Allmend trail racing experiment, where we have increased $v_{\min}$.
4. Strava: https://www.strava.com/segments/29736346, accessed 03 June 2026
5. https://www.trailforks.com/trails/allmendtrail/
6. We divided the length of the trail $419\,\mathrm{m}$ by the measured time that passed between the point of entrance and the point of exit. We started the counter as soon as the robot passed the starting location used in our experiment (time $18\,\mathrm{s}$) and stopped the timer when it reached the point where we stopped our experiment (time $115\,\mathrm{s}$). We also compensated for the speedup factor of four applied between time $23\,\mathrm{s}$ to $93\,\mathrm{s}$, leading to a total estimate of $(23-18)+4\cdot(93-23)+(115-93)=$

$5 + 280 + 22 = 307\,\mathrm{s}$. The average velocity comes out to $1.36\,\mathrm{m/s}$.

7. For comparison, we only consider the average velocity achieved on the upper trail.

Acknowledgements

The authors would like to thank Michael Riner and Per Frivik for their assistance during field deployment; Marco Zenti for hardware support; Diego Ferigo for his contributions to the deployment code; Petr Listov and Adrian Brandemuehl for their help with the MoCap system; Andrea Sgobbi for assistance with sensor integration; James Tigue for assistance with the simulation and training code; and Tristan Laidlow for feedback on perception. Lastly, the authors extend their gratitude to the entire UMV and FIELD teams for continuous support across hardware and software.

Funding

This research is supported by the Robotics and AI (RAI) Institute, which received funding from the Hyundai Motor Group. The opinions expressed herein are those of the authors and do not necessarily reflect the views of the funding organizations.

Declaration of conflicting interests

The authors declare that they have no competing financial or personal interests that could influence the work reported in this paper.

Supplemental material

The supplementary datasets, ROS bags from the Allmend trail deployment, and plotting scripts that support the findings of this study are openly available in Zenodo (https://doi.org/10.5281/zenodo.23063816). Specifically, the repository contains:

data S1: (dataS1.zip) Tabulated datasets used to generate Figures S3, S4, S5, and S7.

data S2: (rosbag_trail1.zip,... rosbag_trail6.zip) ROS bags recorded during the Allmend trail deployment, containing the raw telemetry and perception data used to create the values in Tab. 2.

data S3: (foxglove_layout.json) Foxglove configuration layout used to visualize the provided ROS bags.

data S4: (Figure_7C.zip) Raw data and the corresponding Python script required to reproduce the plots in Fig. 7C.

data S5: (plot_friction_Fig_S1.py) Python script used to generate the friction plot shown in Fig. S1.

References

Ægidius S, Hadjivelichkov D, Jiao J, Embley-Riches J and Kanoulas D (2025) Watch your STEPP: Semantic traversability estimation using pose projected features. In: *ICRA*. IEEE, pp. 2376–2382.

Ancha S, Osteen PR and Roy N (2024) Deep evidential uncertainty estimation for semantic segmentation under out-of-distribution obstacles. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6943–6951.

Bahnam SA, Ferede R, Blaha TM, Lang AE, Lucassen E, Missinne Q, Verraest AEC, Wagter CD and de Croon GCHE (2026) Monorace: Winning champion-level drone racing with robust monocular ai.

Bellicoso D and Wollschläger A (2026) Exploy: Export and deploy reinforcement learning policies. URL https://github.com/rai-inst/exploy.

Betz J, Betz T, Fent F, Geisslinger M, Heilmeier A, Hermansdorfer L, Herrmann T, Huch S, Karle P, Lienkamp M, Lohmann B, Nobis F, Ögretmen L, Rowold M, Sauerbeck F, Stahl T, Trauth R, Werner F and Wischnewski A (2023) Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge. *Journal of Field Robotics* 40(4): 783–809.

Blum H, Sarlin PE, Nieto J, Siegwart R and Cadena C (2021) The Fishyscapes benchmark: Measuring blind spots in semantic segmentation. *International Journal of Computer Vision* 129(11): 3119–3135.

Bokser B, Gonzalez D, Preston A, Bahner A, Wollschläger A, Ilvonen A, Eckert-Erdheim A, Khadke A, Hammoud B, Molinaro D, Jenelten F, Mayne H, Choset H, Bogoslavskyi I, Tinman I, Tigue J, Preisig J, Zheng K, Sharma K, Ang K, Lee L, Margolese L, Lin N, Frias O, Drews P, Boggavarapu R, Burnham R, Zapolsky S, Kim S, Biddlestone S, Mayorga S, Fahmi S, Singh S, McCollum T, Dimitrov V, Moyne W, Chen YM, Farshidian F, Hutter M, Perry D, Rizzi A and Nelson G (2026) System design of the ultra mobility vehicle: A driving, balancing, and jumping bicycle robot. *arXiv preprint arXiv:2602.22118* .

Cai X, Ancha S, Sharma L, Osteen PR, Bucher B, Phillips S, Wang J, Everett M, Roy N and How JP (2024) EVORA: Deep evidential traversability learning for risk-aware off-road autonomy. *IEEE Transactions on Robotics* 40: 3756–3777.

Carion N, Gustafson L, Hu YT, Debnath S, Hu R, Suris D, Ryali C, Alwala KV, Khedr H, Huang A, Lei J, Ma T, Guo B, Kalla A, Marks M, Greer J, Wang M, Sun P, Rädle R, Afouras T, Mavroudi E, Xu K, Wu TH, Zhou Y, Momeni L, Hazra R, Ding S, Vaze S, Porcher F, Li F, Li S, Kamath A, Cheng HK, Dollár P, Ravi N, Saenko K, Zhang P and Feichtenhofer C (2025) Sam 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719* .

Cataffo V, Silano G, Iannelli L, Puig V and Glielmo L (2022) A nonlinear model predictive control strategy for autonomous racing of scale vehicles. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1–8.

Chen LC, Zhu Y, Papandreou G, Schroff F and Adam H (2018) Encoder-decoder with atrous separable convolution for semantic image segmentation. In: *ECCV*. Springer, pp. 801–818.

Cheng X, Shi K, Agarwal A and Pathak D (2024) Extreme parkour with legged robots. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 11443–11450. DOI: 10.1109/ICRA57147.2024.10610200.

Cordts M, Omran M, Ramos S, Rehfeld T, Enzweiler M, Benenson R, Franke U, Roth S and Schiele B (2016) The cityscapes dataset for semantic urban scene understanding. In: *CVPR*. IEEE, pp. 3213–3223.

Corsaro A, Cominardi L, Hecart O, Baldoni G, Avital JEP, Loudet J, Guimares C, Ilyin M and Bannov D (2023) Zenoh: Unifying

communication, storage and computation from the cloud to the microcontroller. In: *2023 26th Euromicro Conference on Digital System Design (DSD)*. IEEE, pp. 422–428.

Couairon P, Chambon L, Serrano L, Haugeard JE, Cord M and Thome N (2025) JAFAR: Jack up any feature at any resolution. In: *NeurIPS*.

Dawson-Haggerty M, Bacher J, Brown-Bivins A, Forest C and O'Kane J (2019) trimesh: Load and analyze triangular meshes (version 3.2.0). URL https://trimesh.org/.

Douze M, Guzhva A, Deng C, Johnson J, Szilvasy G, Mazaré PE, Lomeli M, Hosseini L and Jégou H (2025) The Faiss library. *IEEE Transactions on Big Data* .

Erni G, Frey J, Miki T, Mattamala M and Hutter M (2023) MEM: Multi-modal elevation mapping for robotics and learning. In: *IROS*. IEEE, pp. 11011–11018.

Foehn P, Brescianini D, Kaufmann E, Cieslewski T, Gehrig M, Muglikar M and Scaramuzza D (2022) Alphapilot: Autonomous drone racing. *Autonomous Robots* 46: 307–320.

Foehn P, Romero A and Scaramuzza D (2021) Time-optimal planning for quadrotor waypoint flight. *Science Robotics* 6(56).

Frey J, Patel M, Atha D, Nubert J, Fan D, Agha A, Padgett C, Spieler P, Hutter M and Khattak S (2024) Roadrunner—learning traversability estimation for autonomous off-road driving. *IEEE Transactions on Field Robotics* 1: 192–212.

Frey J, Tuna T, Fu F, Patterson K, Xu T, Fallon M, Cadena C and Hutter M (2026) GrandTour: A legged robotics dataset in the wild for multi-modal perception and state estimation. *arXiv preprint arXiv:2602.18164* .

Fuchs F, Song Y, Kaufmann E, Scaramuzza D and Dürr P (2021) Super-human performance in gran Turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters* 6(3): 4257–4264. DOI:10.1109/LRA.2021.3064284.

Garland M and Heckbert PS (1997) Surface simplification using quadric error metrics. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, pp. 209–216. DOI:10.1145/258734.258849.

Giusti A, Guzzi J, Cireşan DC, He FL, Rodríguez JP, Fontana F, Faessler M, Forster C, Schmidhuber J, Caro GD, Scaramuzza D and Gambardella LM (2016) A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters* 1(2): 661–667. DOI:10.1109/LRA.2015.2509024.

Goldfain B, Drews P, You C, Barulic M, Velev O, Tsiotras P and Rehg JM (2019) Autorally: An open platform for aggressive autonomous driving. *IEEE Control Systems Magazine* 39(1): 26–55. DOI:10.1109/MCS.2018.2876958.

Harley AW, Fang Z, Li J, Ambrus R and Fragkiadaki K (2023) Simple-BEV: What really matters for multi-sensor BEV perception? In: *ICRA*. IEEE, pp. 2759–2765.

He K, Zhang X, Ren S and Sun J (2016) Deep residual learning for image recognition. In: *CVPR*. IEEE, pp. 770–778.

Hendrycks D and Gimpel K (2017) A baseline for detecting misclassified and out-of-distribution examples in neural networks. In: *International Conference on Learning Representations*.

Hoeller D, Rudin N, Sako D and Hutter M (2024) Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics* 9(88): eadi7566. DOI:10.1126/scirobotics.adi7566. URL https://www.science.org/doi/abs/10.1126/scirobotics.adi7566.

Jiang P, Osteen P, Wigness M and Saripalli S (2021) RELLIS-3d dataset: Data, benchmarks and analysis. In: *ICRA*. IEEE, pp. 1110–1116.

Jung S, Lee J, Meng X, Boots B and Lambert A (2024) V-STRONG: Visual self-supervised traversability learning for off-road navigation. In: *ICRA*. IEEE, pp. 1766–1773.

Kabzan J, de la Iglesia Valls M, Reijgwart V, Hendrikx HFC, Ehmke C, Prajapat M, Bühler A, Gosala NB, Gupta M, Sivanesan R, Dhall A, Chisari E, Karnchanachari N, Brits S, Dangel M, Sa I, Dubé R, Gawel A, Pfeiffer M, Liniger A, Lygeros J and Siegwart R (2020) Amz driverless: The full autonomous racing system. *Journal of Field Robotics* 37(7): 1267–1294.

Kaufmann E, Bauersfeld L, Loquercio A, Müller M, Koltun V and Scaramuzza D (2023) Champion-level drone racing using deep reinforcement learning. *Nature* 620(7976): 982–987. DOI:10.1038/s41586-023-06419-4.

Kendall A and Gal Y (2017) What uncertainties do we need in Bayesian deep learning for computer vision? In: *Advances in Neural Information Processing Systems*, volume 30. pp. 5574–5584.

Kim H, Oh H, Park J, Kim Y, Youm D, Jung M, Lee M and Hwangbo J (2025) High-speed control and navigation for quadrupedal robots on complex and discrete terrain. *Science Robotics* 10(102): eads6192. DOI:10.1126/scirobotics.ads6192. URL https://www.science.org/doi/abs/10.1126/scirobotics.ads6192.

Kim Y, Lee JH, Lee C, Mun J, Youn D, Park J and Hwangbo J (2024) Learning semantic traversability with egocentric video and automated annotation strategy. *IEEE Robotics and Automation Letters* 9(11): 10423–10430.

Krinner M, Romero A, Bauersfeld L, Zeilinger M, Carron A and Scaramuzza D (2024) MPCC++: Model Predictive Contouring Control for Time-Optimal Flight with Safety Constraints. In: *Proceedings of Robotics: Science and Systems*.

Lee J, Bjelonic M, Reske A, Wellhausen L, Miki T and Hutter M (2024) Learning robust autonomous navigation and locomotion for wheeled-legged robots. *Science Robotics* 9(89): eadi9641. DOI:10.1126/scirobotics.adi9641. URL https://www.science.org/doi/abs/10.1126/scirobotics.adi9641.

Li Y, Huang B, Chen Z, Cui Y, Liang F, Shen M, Liu F, Xie E, Sheng L, Ouyang W and Shao J (2024) Fast-bev: A fast and strong bird's-eye view perception baseline. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46(12): 8665–8679.

Liao B, Chen S, Zhang Y, Jiang B, Zhang Q, Liu W, Huang C and Wang X (2025) MapTRv2: An end-to-end framework for online vectorized HD map construction. *International Journal of Computer Vision* 133(3): 1352–1374.

Liniger A, Domahidi A and Morari M (2015) Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods* 36(5): 628–647.

Loquercio A, Kaufmann E, Ranftl R, Müller M, Koltun V and Scaramuzza D (2021) Learning high-speed flight in the wild. *Science Robotics* 6(59): eabg5810. DOI:10.1126/scirobotics.abg5810. URL https://www.science.org/doi/abs/10.1126/scirobotics.abg5810.

Łucki J, Becktor J, Georgakis G, Royce R and Khattak S (2025) Visual perception engine: Fast and flexible multi-head inference for robotic vision tasks. *arXiv preprint arXiv:2508.11584* .

Macenski S, Foote T, Gerkey B, Lalancette C and Woodall W (2022) Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics* 7(66): eabm6074.

Mattamala M, Frey J, Libera P, Chebrolu N, Martius G, Cadena C, Hutter M and Fallon M (2025) Wild visual navigation: Fast traversability learning via pre-trained models and online self-supervision. *Autonomous Robots* 49(3): 19.

Miki T (2025) *Bridging Perception and Control for Legged Locomotion and Navigation in the Wild*. PhD Thesis, ETH Zurich. DOI:10.3929/ethz-c-000787122. URL https://doi.org/10.3929/ethz-c-000787122.

Miki T, Wellhausen L, Grandia R, Jenelten F, Homberger T and Hutter M (2022) Elevation mapping for locomotion and navigation using GPU. In: *IROS*. IEEE, pp. 2273–2280.

Min C, Jiang W, Zhao D, Xu J, Xiao L, Nie Y and Dai B (2022) ORFD: A dataset and benchmark for off-road freespace detection. In: *ICRA*. IEEE, pp. 2532–2538.

Philion J and Fidler S (2020) Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3D. In: *ECCV*. Springer, pp. 194–210.

Pinggera P, Ramos S, Gehrig S, Franke U, Rother C and Mester R (2016) Lost and found: Detecting small road hazards for self-driving vehicles. In: *IROS*. IEEE, pp. 1099–1106.

Rajvanshi A, Sikka K, Lin X, Lee B, Chiu HP and Velasquez A (2024) Saynav: Grounding large language models for dynamic planning to navigation in new environments. *Proceedings of the International Conference on Automated Planning and Scheduling* 34(1): 464–474. DOI:10.1609/icaps.v34i1.31506. URL https://ojs.aaai.org/index.php/ICAPS/article/view/31506.

Rana K, Haviland J, Garg S, Abou-Chakra J, Reid I and Suenderhauf N (2023) Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. In: *7th Annual Conference on Robot Learning*. URL https://openreview.net/forum?id=wMpOMO0Ss7a.

Ravichandran Z, Peng L, Hughes N, Griffith JD and Carlone L (2022) Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks. In: *2022 International Conference on Robotics and Automation (ICRA)*. pp. 9272–9279. DOI:10.1109/ICRA46639.2022.9812179.

Reed C, Tatsch C, Gross JN and Gu Y (2025) Autonomous hiking trail navigation via semantic segmentation and geometric analysis. In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 52–58. DOI:10.1109/IROS60139.2025.11245853.

Rudin N, He J, Aurand J and Hutter M (2025) Parkour in the wild: Learning a general and extensible agile locomotion policy using multi-expert distillation and rl fine-tuning. *CoRR* abs/2505.11164. URL https://arxiv.org/abs/2505.11164.

Rudin N, Hoeller D, Bjelonic M and Hutter M (2022) Advanced skills by learning locomotion and local navigation end-to-end. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 2497–2503. DOI:10.1109/IROS47612.2022.9981198.

Rudin N, Hoeller D, Reist P and Hutter M (2021) Learning to walk in minutes using massively parallel deep reinforcement learning. In: *5th Annual Conference on Robot Learning*. URL https://openreview.net/forum?id=wK2fDDJ5VcF.

Schreiber A and Driggs-Campbell K (2025) Do you know the way? human-in-the-loop understanding for fast traversability estimation in mobile robotics. *IEEE Robotics and Automation Letters* 10(6): 5863–5870.

Schulman J, Wolski F, Dhariwal P, Radford A and Klimov O (2017) Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Shah D, Sridhar A, Dashora N, Stachowicz K, Black K, Hirose N and Levine S (2023) ViNT: A foundation model for visual navigation. In: *7th Annual Conference on Robot Learning*. URL https://arxiv.org/abs/2306.14846.

Siméoni O, Vo HV, Seitzer M, Baldassarre F, Oquab M, Jose C, Khalidov V, Szafraniec M, Yi S, Ramamonjisoa M, Massa F, Haziza D, Wehrstedt L, Wang J, Darcet T, Moutakanni T, Sentana L, Roberts C, Vedaldi A, Tolan J, Brandt J, Couprie C, Mairal J, Jégou H, Labatut P and Bojanowski P (2025) DINOv3. *arXiv preprint arXiv:2508.10104* .

Song Y, Lin H, Kaufmann E, Duerr P and Scaramuzza D (2021) Autonomous overtaking in gran Turismo sport using curriculum reinforcement learning. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 1205–1212. DOI:10.1109/ICRA48506.2021.9561280.

Song Y, Romero A, Müller M, Koltun V and Scaramuzza D (2023) Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics* 8(82): eadg1462.

Srinivasan S, Nicolas Giles S and Liniger A (2021) A holistic motion planning and control solution to challenge a professional racecar driver. *IEEE Robotics and Automation Letters* 6(4): 7854–7860. DOI:10.1109/LRA.2021.3101244.

Stachowicz K, Shah D, Bhorkar A, Kostrikov I and Levine S (2023) Fastrlap: A system for learning high-speed driving via deep rl and autonomous practicing. In: Tan J, Toussaint M and Darvish K (eds.) *Proceedings of The 7th Conference on Robot Learning*, *Proceedings of Machine Learning Research*, volume 229. PMLR, pp. 3100–3111. URL https://proceedings.mlr.press/v229/stachowicz23a.html.

Vizzo I, Guadagnino T, Mersch B, Wiesmann L, Behley J and Stachniss C (2023) KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters (RA-L)* 8(2): 1029–1036. DOI:10.1109/LRA.2023.3236571.

Vogel D, Baines R, Church J, Lotzer J, Werner K and Hutter M (2025) Robust ladder climbing with a quadrupedal robot. In: *38th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7239–7244.

Werby A, Huang C, Büchner M, Valada A and Burgard W (2024) Hierarchical Open-Vocabulary 3D Scene Graphs for Language-Grounded Robot Navigation. In: *Proceedings of Robotics: Science and Systems*. Delft, Netherlands. DOI:10.15607/RSS.2024.XX.077.

Xie E, Wang W, Yu Z, Anandkumar A, Alvarez JM and Luo P (2021) Segformer: Simple and efficient design for semantic segmentation with transformers. In: *NeurIPS*.

Zhang Z, Liniger A, Dai D, Yu F and Van Gool L (2021) End-to-end urban driving by imitating a reinforcement learning coach. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. pp. 15202–15212. DOI:10.1109/ICCV48922.2021.01494.

Zhao W, Li J, Dong X, Xiang Y and Guo Y (2024) Segment every out-of-distribution object. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 3910–3920.

A Index to Multimedia Extensions

The following videos can be found alongside the publication

Multimedia Ext. E1: narrated overview of the method and results

Multimedia Ext. E2: autonomous racing on Allmend trail

Multimedia Ext. E3: simulated parkour with UMV

Multimedia Ext. E4: simulated Allmend trail with UMV

Multimedia Ext. E5: MoCap assisted indoor parkour

Multimedia Ext. E6: perceptive indoor trail racing

Multimedia Ext. E7: simulated parkour with ANYmal

Multimedia Ext. E8: simulated Allmend trail with ANYmal

Multimedia Ext. E9: SemMap perception system overview

B Nomenclature

RL	reinforcement learning
PPO	Proximal Policy Optimization
UMV	Ultra Mobility Vehicle
MPCC	Model Predictive Contouring Control
MPC	Model Predictive Control
MLP	Multi-Layer Perceptron
MDP	Markov decision process
TL	Trail-Library
SemMap	Semantic Maps
MoCap	Motion Capture System
VFM	Vision Foundation Model
OOD	out-of-distribution
mIoU	mean Intersection over Union
PCA	Principal Component Analysis
MPS	Multi-Process Service
GIL	Global Interpreter Lock
FPS	frames per second
VLM	vision-language model
BEV	bird's-eye view
LIO	LiDAR-inertial odometry
MEM	multi-modal elevation mapping
TENa	Terrain Embedded Navigation
DT	Digital-Twin
DOF	Degree of Freedom

C Optimizer Settings

We optimize the unified locomotion and navigation policies using PPO (Schulman et al. 2017) and the hyperparameters specified in Table S1. For the extended outdoor trails, we increase the rollout duration to properly capture the longer task horizons. All remaining, unspecified hyperparameters are kept identical to Rudin et al. (2021).

D Simulator Settings

We employ the compliant contact model provided by Isaac Sim, utilizing the specific rigid-body and contact parameters previously identified in Bokser et al. (2026). Furthermore, we approximate the tire geometry using a mathematical torus primitive rather than a standard meshed cylinder. Empirically, this representation yielded more stable behaviors during tight turning scenarios. We note, however, that computing collisions between torus primitives and dense terrain meshes decreases simulation rollout speeds by approximately $20\,\%$ compared to a simplified cylinder mesh. Similar to Bokser et al. (2026), the control policy is queried at $50\,\mathrm{Hz}$, but we update the physics simulation at a frequency of $500\,\mathrm{Hz}$ as opposed to $200\,\mathrm{Hz}$.

We observed that enforcing explicit linear or angular velocity limits on articulations within Isaac Sim causes the physics engine to inject artificial external forces to satisfy those constraints. To prevent the policy from exploiting these artificial forces, we explicitly disable these constraints.

E Network Architecture and Observations

We employ an asymmetric actor-critic framework in which the policy (actor) and the value function (critic) are parameterized by separate MLPs, similar to Bokser et al. (2026) but using larger network sizes. The actor network models a stochastic policy $\pi(\vec{a} \mid \vec{o}) \sim \mathcal{N}(\vec{\mu}_\theta(\vec{o}), \vec{\sigma}_\theta)$, where the action mean $\vec{\mu}_\theta(\vec{o})$ is mapped by an MLP parameterized by θ, and the log-standard deviation $\vec{\sigma}_\theta$, with the latter being a learned state-independent layer. Conversely, the critic network is represented by a deterministic value function $V_\phi(\vec{o}, \vec{\tilde{o}})$ parameterized by ϕ. To accelerate policy convergence during simulation, the critic is granted access to both the standard observation vector $\vec{o}$ and privileged state vector $\vec{\tilde{o}}$. Both the actor and critic networks feature three hidden layers comprising 512 neurons each. To stabilize the learning process, all incoming observation streams are dynamically normalized online utilizing running means and standard deviations. A comprehensive breakdown of all observation terms, alongside the specific noise levels applied to them during training, is detailed in Table S2. The noise distributions are mostly similar to Bokser et al. (2026), however, we have slightly tuned the specific shape and parameters to balance our reward design.

During real-world deployment, SemMap may produce incorrect semantic annotations. To expose the policy to these artifacts during training, we introduce three distinct noise models. First, the label of any individual scan point is randomly flipped with a probability of $4\,\%$. Second, there is a $20\,\%$ probability that a random block of consecutive rows in the scan pattern is entirely misclassified as non-trail. Third, we flip the labels of valid trail points with a probability that

Table S1. Optimizer settings. Proximal Policy Optimization (PPO) hyperparameter configurations. Here, "indoor" denotes the short parkour circuits, whereas "outdoor" refers to outdoor trails spanning several hundred meters in length. Numbers provided in curly brackets indicate task dependency.

Setting	TL	DT indoor	DT outdoor
Number of environments	4096	4096	1024
Number of steps	24	24	192
Episode length in seconds	16	$\{8, 10, 12, 14\}$	120
Number of epochs	16000	16000	16000
Entropy coefficient	0.005	0.008	0.008

Table S2. Observation space. The critic network extends the actor's inputs with privileged, noise-free observations. The scan pattern consists of a 16×16 grid with a spatial resolution of $0.2\,\mathrm{m}$, centered $2\,\mathrm{m}$ ahead of the robot's base frame. The notation $d(0.05)$ refers to positional drift, which is applied independently across the x, y, and z axes. The semantic observation layer incorporates a custom label-corruption scheme. Finally, the IMU observation vector comprises six dimensions, sourced from two distinct IMU sensors: one mounted on the bike's base and the other on the upper body.

Type	Observations	Dim	Noise
Actor	Fork angle	1	$\mathcal{N}(\mu = 0, \sigma = 0.001)$
	Joint body angles	2	$\mathcal{U}(\pm 0.01)$
	Fork velocity	1	$\mathcal{N}(\mu = 0, \sigma = 0.1)$
	Rear-wheel velocity	1	$\mathcal{N}(\mu = 0, \sigma = 0.05)$
	Joint body velocities	2	$\mathcal{U}(\pm 0.15)$
	IMU angular velocity	6	$\mathcal{U}(\pm 0.1)$
	IMU projected gravity	6	$\mathcal{U}(\pm 0.015)$
	Standing command	1	0
	Scan, height layer	256	$\mathcal{U}(\pm 0.08)$ and $d(0.05)$
	Scan, semantic layer	256	custom
	Last action	4	0
Critic	base linear velocity	3	
	Base pose	6	
	Time left in the episode	1	

increases linearly with the scan row index. Specifically, the corruption probability is set to $0\,\%$ for the scan row closest to the robot and scales up to $20\,\%$ for the furthest row, with the probabilities for intermediate rows interpolated linearly.

The three noise models were designed to mimic error patterns we observed in SemMap outputs during development: isolated label flips at class boundaries, contiguous unsegmented regions caused by occlusions, and decreasing segmentation reliability with distance. Their parameters were set heuristically and calibrated during the indoor racing experiments (Subsection 4.4). We did not perform a systematic ablation of the individual noise terms; policies trained with the full model transferred to both the indoor and outdoor deployments without further modification.

F Action Space and Low-Level Control

The action space is identical to Bokser et al. (2026) and is repeated here for completeness. The neural network predicts positional targets for the steering fork, lower-body joints, and upper-body joints, alongside a target angular velocity for the rear wheel. The front wheel remains unactuated. The high-level policy actions are updated at $50\,\mathrm{Hz}$, while a low-level PD controller converts these targets into joint torques at $8\,\mathrm{kHz}$. For the position-targeted joints, the applied torque τ_i is computed as $\tau_i = k_{p,i}(q_{i,\mathrm{des}} - q_i) - k_{d,i}\dot{q}_i$, whereas for the velocity-targeted joint, the torque is computed using a proportional velocity error $\tau_i = k_{p,i}(\dot{q}_{i,\mathrm{des}} - \dot{q}_i)$.

Table S3. Action space and low-level control parameters. The raw, normalized outputs of the actor network are multiplied by their respective action scales to generate the desired position ($q_{i,\mathrm{des}}$) or velocity ($\dot{q}_{i,\mathrm{des}}$) targets. The low-level PD controller tracks these targets with gains k_p and k_d. The front wheel is unactuated and operates passively.

Joint	Dim	Type	Action Scale	k_p	k_d
Fork	1	pos	1.0	1.4	0.6
Body	2	pos	1.5	140	10
Rear wheel	1	vel	15.0	0.6	0
Front wheel	1	–	–	–	–

A comprehensive summary of all action scales and joint-specific PD control gains is provided in Table S3.

G Style Rewards

We compute each style reward as $r_i = w_i \cdot f_i(e_i)$, where w_i denotes the respective scalar weight, and f_i represents the mathematical shaping function that acts upon an internal state or control error e_i. These style rewards do not define the task itself, but are necessary for regularizing the policy's outputs to ensure safe and smooth sim-to-real transfer. The complete formulation of all style rewards is detailed in Table S4. We note that we use substantially fewer style rewards compared to Bokser et al. (2026) to not counteract the emergence of high-speed and agile behaviors. In particular, we do not constrain the upper body joints to track a default joint position, but let the body rather discover optimal configurations on its own. It needs to be

Table S4. Style rewards. Summary of the error signals e_i, shaping functions $f(e_i)$, and respective weights w_i. Here, s_i represents the action scale of the corresponding joint; a_k and a_{k-1} denote the raw network actions at the current and previous time steps; $\dot{q}_a$ are the measured angular velocities; τ_{des} represents the desired joint torques after clipping.

reward	e	$f(e)$	w_i
Fork action	$s_f \cdot (a_k - a_{k-1})$	$\sum \lvert e \rvert + e^2$	$-1e^{-4}$
Wheel action	$s_w \cdot (a_k - a_{k-1})$	$\sum \lvert e \rvert + e^2$	$-5e^{-6}$
Body action	$s_b \cdot (a_k - a_{k-1})$	$\sum \lvert e \rvert + e^2$	$-1e^{-3}$
Joint vel	$\dot{q}_a$	$\sum \lvert e \rvert + e^2$	$-1e^{-3}$
Joint power	$\dot{q}_a \cdot \tau_{\text{des}}$	$\sum \lvert e \rvert + e^2$	$-1e^{-7}$

said, however, that the choice of default joint positions nevertheless influences the overall behavior, even if they are not explicitly tracked.

H Domain Randomizations

We employ standard domain randomization techniques provided by Isaac Lab to bridge the sim-to-real gap. The specific randomization events, their associated parameter ranges, and their application modes are summarized in Table S5. We note that while the number of randomized parameters is substantially smaller than in Bokser et al. (2026), we employ wider interval ranges for those selected parameters. We found that this approach yields a more tractable tuning problem, facilitating faster iteration and refinement. We also observed that the base twist randomization at startup is a particularly useful event for our task; it allows the agents to discover a stable forward motion almost instantaneously. Heuristically, we found that in our case randomizing actuator gains had no clear benefits and did not positively reflect in real-world success rates. We also found that the policies evaluated in simulation result in roughly similar performance metrics (slightly smaller average velocities without domain randomization), regardless of whether they were trained with or without domain randomizations (Appendix R.2).

By default, the standard friction randomization event independently samples both static and dynamic friction coefficients from their specified ranges, and subsequently clips the dynamic friction to satisfy the physical constraint $\mu_{\text{static}} \geq \mu_{\text{dyn}}$. However, this clipping causes a disproportionate number of samples to accumulate directly on the boundary where static and dynamic friction equalize. To mitigate this boundary accumulation, we implement a modified sampling strategy: we first uniformly sample the dynamic friction coefficient, dynamically adjust the lower bound of the static friction range to match this sampled value, and finally sample the static friction from this constrained range. As illustrated in Fig. S1, this approach yields a more uniform coverage of the feasible parameter space. While an alternative solution would be to sample uniformly across the entire valid triangular region, this would inadvertently skew the expected value of the dynamic friction toward the lower end of the spectrum.

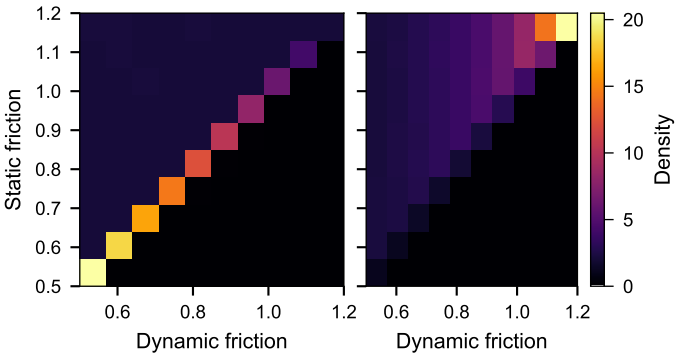


Figure S1. Comparison of friction coefficient sampling methods. Both static and dynamic friction bounds are initialized to $\{0.5, 1.2\}$. We plot 1000 sampled friction pairs for two distinct strategies. Left: The default Isaac Lab approach, causing boundary accumulation. Right: Our adapted method, resulting in a more uniform distribution. The density values indicate the number of overlapping samples.

I Cross embodied Generalization of TENa in Simulation

While the UMV serves as our primary evaluation platform, TENa is platform-agnostic. To demonstrate robust learning generalization across different kinematic morphologies, we applied our MDP formulation to an ANYmal-D quadruped robot. Out of the 12 parkour environments deployed in Fig. 6A and Fig. 7A combined, ANYmal successfully navigated the maximum difficulty level of 8 terrains with a success rate exceeding $95\,\%$, averaged over 1000 runs. For the remaining four terrains, the policy converged before mastering the final difficulty level, mostly due to foothold locations being too sparse. Furthermore, ANYmal traversed the upper part of the Allmend trail with a success rate of $81\,\%$ and within $89.5\,\text{s}$, corresponding to an average speed of $4.68\,\text{m/s}$. Fig. S2 highlights visual snapshots of all successful traversals, multimedia extension E7 shows the corresponding full parkour runs, and multimedia extension E8 demonstrates the ANYmal traversing the Allmend trail.

To interface the ANYmal-D quadruped within our simulation framework, we adapted the standard joint-level proprioceptive observations, action scales, default joint angles, and style rewards provided by the Isaac Lab quadruped environment. We slightly modified the observation space by replacing the velocity commands with our track-stand command and the height scan with the semantically annotated height scan. All core components of the MDP formulation were transferred identically from the UMV learning environment, with the episode length being the sole tuning parameter adjusted to account for the different morphological architecture.

We note that we have not put any effort into style reward tuning and do not expect these policies to transfer zero-shot to the real hardware. Ill-tuned behaviors that likely will not transfer can be seen in both multimedia extensions. In E7 ANYmal uses its knees aggressively to climb over objects, and E8 reveals an asymmetric running gait with the right-front leg being locked in.

Table S5. Domain randomization parameters. Summary of the randomization events, parameter bounds, and application modes utilized during training to facilitate sim-to-real transfer. For evaluation purposes, all events are disabled, and rigid body material friction is set to the average value of the boundaries.

Event	Parameter	Range	Mode
Randomize rigid body material	static friction	$[0.5, 1.2]$	Startup
	dynamic friction	$[0.5, 1.2]$	Startup
Reset root state uniform (in base frame)	pose.z	$[0.0, 0.2]$	Reset
	pose.roll	$[-0.04, 0.04]$	Startup
	twist.x	$[0.0, 2.0]$	Startup
Randomize rigid body inertia	relative mass	$[0.9, 1.1]$	Startup
	com.x, com.y, com.z	$[-0.03, 0.03]$	Startup
Push by adding velocity (in base frame)	twist.x, twist.y	$[-0.6, 0.6]$	Interval $[2, 12]$
Apply external force/torque	absolute force	$[-2.0, 2.0]$	Reset
Randomize joint parameters	absolute friction	$[0.0, 0.03]$	Reset

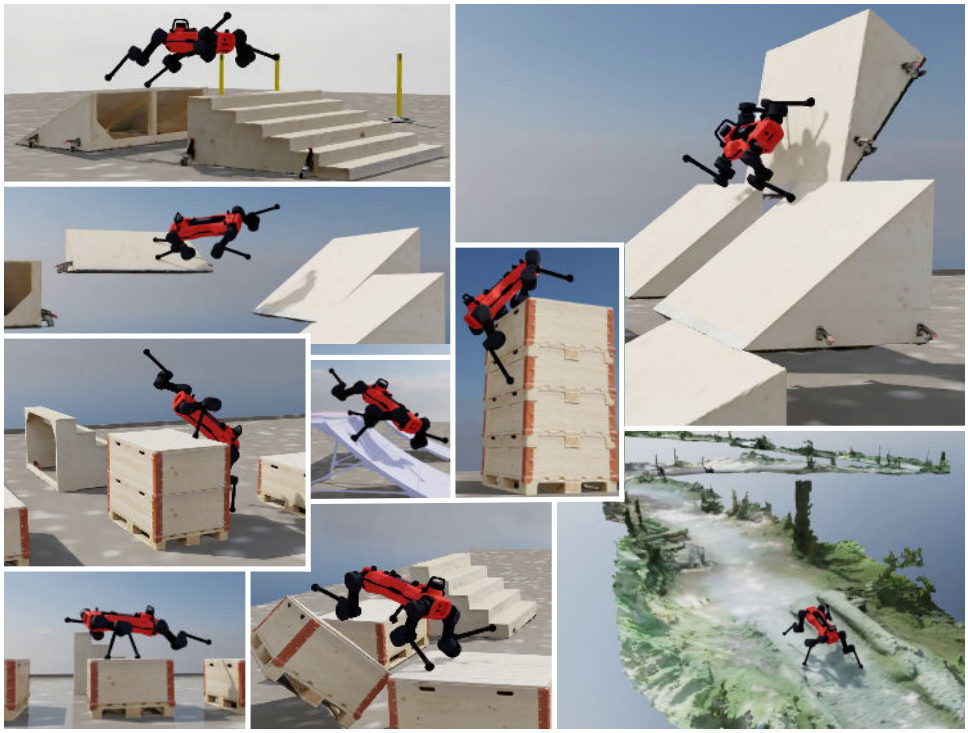


Figure S2. Generalization to quadrupedal parkour and trail running. Snapshots of the ANYmal-D robot navigating DT environments, executing jumping, climbing, and wall-run skills. All semantic meshes and the core MDP components remained for all tasks strictly identical to those used for the wheeled robot. The episode length was the sole tuning parameter adjusted to account for the quadruped's different progression velocity.

J Perception Datasets

Table S6 lists the datasets used in the perception evaluation. The Grand Tour evaluation uses the left frames of the ZED2i stereo camera from the TRIM-1 sequence.

K Statistical Analysis of Perception

All perception comparisons follow a common protocol. The per-image studies (the ORFD upsampler study and the Lost-and-Found OOD study) use paired Wilcoxon signed-rank tests on per-image metrics, with effect sizes given as the matched-pairs rank-biserial correlation r and P-values Holm-corrected within each family of comparisons; because n is large, we report effect sizes and magnitudes alongside P. Seed-based studies (label efficiency and training curves) report the mean $\pm$ standard deviation over $n = 3$ independent training seeds that vary the random initialization and augmentation order while keeping the k-means label subsets fixed. The rapid-adaptation runs on the Grand Tour sites (Table S7) are single training runs per site and are reported as such, without variability estimates. Inference latencies report the mean $\pm$ standard deviation over $n = 100$ timed iterations of a single pre-loaded image after 50 warm-up iterations on the AGX Orin, excluding transport overheads. Unless noted otherwise, the stated variability measure is the standard deviation. In all plots, error bars and shaded bands denote ± 1 SD; where they are not visible, the deviation is smaller than the marker or line width. Success rates of the real-world and simulated control experiments are reported as counts over the stated number of trials.

L Label Efficiency

We evaluate label efficiency on RELLIS-3D (Jiang et al. 2021), an off-road dataset with 20 semantic classes widely used in traversability work. We compare SemMap (frozen DINOv3-s, JAFAR upsampling, $\sim$100K trainable parameters) against DeepLabV3+ (Chen et al. 2018) with a COCO/VOC-pretrained ResNet-50 (He et al. 2016) (16.1M / 42.0M trainable parameters for decoder-only / end-to-end)

Table S6. Datasets used in the perception evaluation. The Lost and Found evaluation uses the 1,111 of 1,203 test images that contain obstacles.

Dataset	Classes	Train / Eval	Purpose
RELLIS-3D Jiang et al. (2021)	20→3	3,302 / 983	Label efficiency
Cityscapes Cordts et al. (2016)	19→3	2,975 / –	OOD confidence
Lost and Found Pinggera et al. (2016)	–	– / 1,203	OOD evaluation
ORFD Min et al. (2022)	3	8,392 / 2,193	Inference quality
Indoor trail (ours)	3	57 / 15	Deployment (bicycle)
Outdoor trail (ours)	5	28,763 / 3,196	Deployment (bicycle)
Factory floor (ours)	3	80 / 20	Deployment (G1)
Grand Tour Frey et al. (2026)	–	– / eval	Generalization

and SegFormer-B0 (Xie et al. 2021) with an ADE20K-pretrained MiT-B0 (400K / 3.7M), each with both a frozen backbone and full fine-tuning. Training images are selected with the FAISS-based pipeline of the main text, and we report the best validation mIoU (983 images) with early stopping (patience 5). We additionally collapse the 20 classes into a 3-class traversability grouping (traversable, non-traversable, background).

At 20 classes, SemMap outperforms all baselines at low label budgets: $33.9 \pm 1.4\%$ mIoU at 20 labeled images (mean $\pm$ SD over $n = 3$ seeds) versus $23.9 \pm 0.9\%$ for fine-tuned DeepLabV3+ and $24.1 \pm 3.5\%$ for SegFormer-B0 (Fig. S3). The low absolute mIoU reflects severe class imbalance in RELLIS-3D, with pixel accuracy above 88% even at the lowest budgets. Fully fine-tuned baselines need 2–5$\times$ more labels to approach SemMap, and the frozen ResNet-50 is weakest, confirming that ImageNet-supervised features are insufficient when the backbone cannot adapt. At 3 classes, the gap narrows and SemMap plateaus from 20 samples, as every label covers all classes. Fig. S4 shows that with $k \geq 5$ images, accuracy reaches $\sim$80% within 3 epochs, and that $k = 100$ (3% of the data) already matches the full training set on mIoU. This rapid, low-data convergence enables on-device adaptation: from a 5-minute recording of the SNOW-2 Grand Tour sequence (Frey et al. 2026) we produced a deployed model in under 15 minutes, with data curation reducing 3,900 frames to 20 images in 2 minutes, SAM3-assisted 3-class text-prompt labeling in under 5 minutes, and on-device training plus TensorRT export in 4 minutes.

We applied the same on-device workflow to two further Grand Tour sequences, ETH-2 (an indoor building) and ICE-1 (an ice cave), yielding three sites with 20 clustered, SAM3-labeled images each, split into 16 training and 4 validation images (Table S7). Each site uses a coarse three-class layer comprising a traversable surface class, a non-traversable class, and a background class that absorbs all unprompted pixels. SAM3 is prompted with a single noun phrase per non-background class; SNOW-2 uses the prompts "snowy floor" and "fence", ETH-2 uses "indoor floor" and "people", and ICE-1 uses "ice floor" and "glass barrier", reconstructed after the fact and matching the original labels at 0.94 mIoU. The resulting masks are reviewed by hand before training. All decoders are trained on the AGX Orin with early stopping on validation mIoU, and the validation frames are the held-out cluster representatives of the same recording. ICE-1 yields the lowest score of the three, illustrating a site at which further samples would be added through the same workflow.

Table S7. Rapid adaptation on three Grand Tour sites. Best validation mIoU and pixel accuracy (3 classes) of the decoder trained on the AGX Orin from 16 clustered, SAM3-labeled images and evaluated on the 4 held-out cluster representatives of the same recording, with early stopping on validation mIoU. Training time is the wall-clock time of the training step alone on the AGX Orin, excluding data curation, SAM3 labeling, and TensorRT export. Single run per site.

Sequence	Scene	mIoU [%]	Acc [%]	Training time [s]
SNOW-2	alpine snow field	90.7	97.4	177
ETH-2	indoor building	89.0	97.1	90
ICE-1	ice cave	77.3	88.8	123

Table S8. Label efficiency on the outdoor trail dataset. Best validation mIoU on the 3,196-image test split versus the number of labeled training images k (mean $\pm$ SD over $n = 3$ seeds).

k	mIoU [%]
20	82.61 ± 0.09
50	83.10 ± 0.06
100	83.80 ± 0.04
200	84.23 ± 0.05
500	84.83 ± 0.07
1,000	85.84 ± 0.30
2,000	87.22 ± 0.21
4,000	88.18 ± 0.23
28,763 (full)	90.01 ± 0.08

On our outdoor trail dataset, we additionally quantify the impact of the label budget, training the decoder on subsets of k images selected with the same clustering pipeline (mean $\pm$ SD over $n = 3$ seeds). With 20 labeled images, the decoder reaches $82.6 \pm 0.1\%$ mIoU on the 3,196-image test split; the full 28,763-image training set raises this to 90.0% (97.0% pixel accuracy), with returns diminishing smoothly in between (Table S8). Convergence mirrors the RELLIS-3D study: for $k \geq 100$, validation mIoU is within 1% of its final value by the second epoch. Baseline comparisons are reported on the public RELLIS-3D benchmark above; here we characterize the label-budget trade-off on our deployment domain.

M Confidence-Weighted Map Fusion

Each grid cell is updated by a confidence-weighted fusion of the projected traversability observations. For a cell with a valid image correspondence and per-pixel confidence $c_p \geq c_{\text{floor}}$, we form a distance weight

$$w_d = \frac{1}{1 + (d/r)^4}, \tag{7}$$

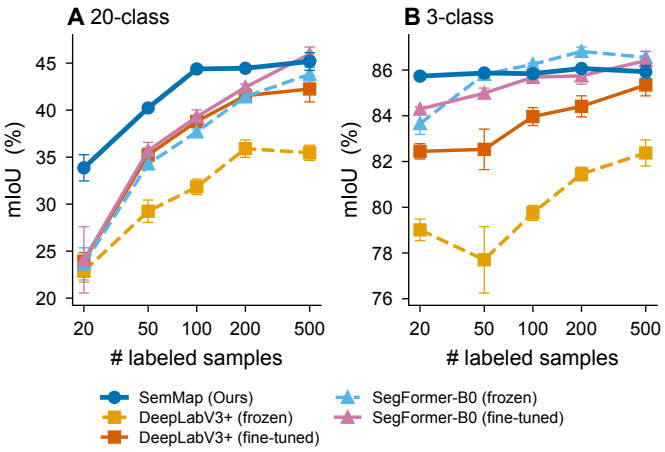


Figure S3. Label efficiency on RELLIS-3D at 20 and 3 class granularities. Best validation mIoU versus the number of labeled images (mean ± SD over $n = 3$ seeds; bands smaller than the markers are not visible). SemMap outperforms frozen and fine-tuned DeepLabV3+ and SegFormer-B0 at low label budgets, most clearly at 20 classes.

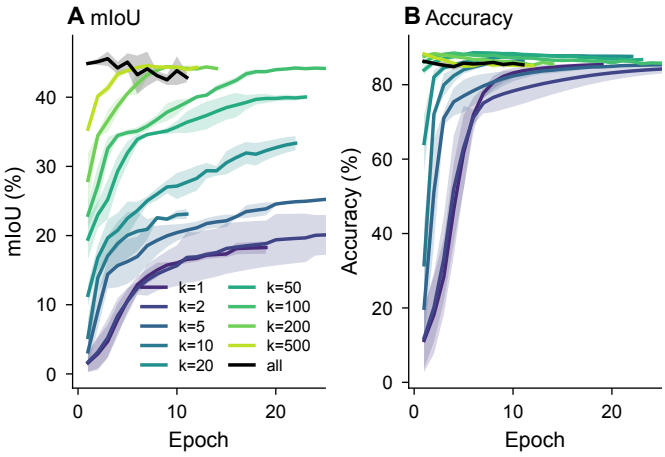


Figure S4. Validation mIoU and accuracy vs. epoch on RELLIS-3D (20 classes) at label budget k; bands show mean ± SD over $n = 3$ seeds. With $k \geq 100$ (3% of the training set), mIoU matches the full dataset; accuracy converges within 3 epochs for $k \geq 5$.

where d is the ground-plane distance from the camera to the cell and r is the fusion range (3 m in the indoor deployment; by default a quarter of the map extent). The observation confidence is $c = c_p\, w_d$, and the cell value v is blended toward the observation o as

$$v \leftarrow v + \alpha\,(o - v), \qquad \alpha = \frac{c}{c + W}, \tag{8}$$

where W is the cell's accumulated evidence, initialized to 0 so that the first valid observation is written directly ($\alpha = 1$). The evidence is then updated as $W \leftarrow \min(W + c, W_{\max})$; the cap $W_{\max}$ bounds how confident a cell becomes, preserving plasticity to later change. Cells without a valid correspondence, or with $c_p < c_{\text{floor}}$, retain their value and evidence, and the evidence layer is shifted and cleared together with the map during recentering. We use $c_{\text{floor}} = 0.5$ and $W_{\max} = 4.0$.

N Confidence and Out-of-Distribution Detection

To test whether the confidence signal detects unseen obstacles, we train all architectures on Cityscapes (Cordts et al. 2016) using 200 of 2,975 images and evaluate on the Lost and Found (Pinggera et al. 2016) test set, which contains real road obstacles absent from training. We report the object-centric false-positive rate (obj-FPR), the fraction of obstacle pixels misclassified as traversable. SemMap uses cosine-similarity confidence; baselines use maximum softmax probability.

Without filtering, SemMap predicts 1.7–2.0× fewer obstacle pixels as traversable than the baselines at 3 classes: 35.0% of obstacle pixels versus 60.2% for DeepLabV3+ and 69.4% for SegFormer-B0, pooled across the 1,111-image test set (Fig. S5). The per-image difference is significant (paired Wilcoxon signed-rank, $P < 0.001$ Holm-corrected; rank-biserial $r = 0.76$ and 0.85 against DeepLabV3+ and SegFormer-B0, respectively). At 19 classes all methods degrade, but SemMap (56%) still beats the baselines (63–77%). Reducing from 19 to 3 classes helps SemMap most (56% → 35%) while SegFormer-B0 worsens (63% → 69%), indicating that a coarser, traversability-focused label space suits cosine-similarity confidence. At a 95% true-positive rate, both baselines require threshold $t = 1.0$ at 3 classes, leaving no tunable range, whereas SemMap reaches it at $t = 0.72$ with a wide usable range. This is not a substitute for explicit synthetic-negative training, which could improve OOD detection further; the advantage is meaningful OOD awareness at negligible additional training and inference cost. Fig. S6 shows the qualitative behavior: softmax confidence highlights only class boundaries, while cosine similarity is clearly low at misclassified obstacles, enabling recovery through downstream filtering. The same confidence signal drives the on-robot elevation-map fusion during indoor racing (Fig. 7D): with confidence-weighted fusion enabled (Appendix section M) the accumulated semantic map stays consistent across laps, whereas without it the low-confidence predictions from high-speed turns progressively corrupt the map.

O Inference Performance

We profile SemMap on an NVIDIA AGX Orin 64GB (MAXN, max fan, CUDA Multi-Process Service (MPS), FP16, 320 × 512 input, 50 warmup and 100 timed iterations) and measure segmentation quality on the ORFD (Min et al. 2022) test set (2,193 images) with the decoder trained on 500 curated images. Each timed iteration passes the same pre-loaded image through the network, so the measurements capture pure inference time and exclude message transport and synchronization overheads; the pipeline is also profiled in isolation, without the rest of the autonomy stack running. The reported throughputs are therefore upper bounds, which is why we state them as "up to" throughout.

The configurations in Table S9 time model inference including confidence, i.e., backbone, upsampler, segmentation head, and cosine-similarity confidence. To obtain the complete per-frame cost, we additionally time the call that the semantic worker executes for every frame, with the clocks pinned (`jetson_clocks`) and TensorRT 10.7.

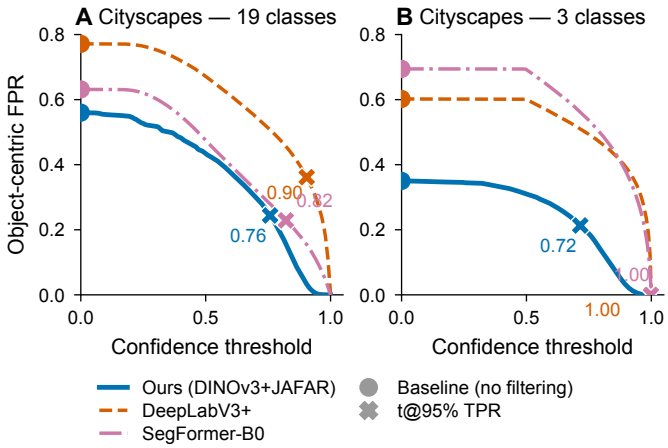


Figure S5. OOD obstacle detection on Lost and Found (trained on Cityscapes with 200 images; evaluated on the 1,111 test images containing obstacles). Pooled object-centric FPR vs. confidence threshold at 20 and 3 classes. Markers: no filtering (circle), 95% TPR threshold (triangle).

Table S9. Inference latency on AGX Orin 64GB (320×512) and segmentation quality on ORFD. Latencies are the mean over $n = 100$ timed iterations (50 warm-up); per-configuration SD ≤ 0.33 ms. TensorRT FP16 unless noted. All rows above the last block time model inference including confidence; the last block times the complete semantic pipeline per frame, adding preprocessing, traversability, and output upsampling, averaged over repeated runs (see text). Arrows denote the preferred direction (↓: lower is better, ↑: higher is better). The bilinear configurations equal the no-upsampler baseline, as interpolation adds no information. *Recommended configuration.

Configuration	ms ↓	FPS ↑	mIoU ↑	Acc ↑
Backbone only (PyTorch)	17.74	56	–	–
Backbone only (TRT)	4.38	228	–	–
Bilinear + seg 1/4	5.34	187	83.81	93.73
Bilinear + seg 1/2	8.82	113	83.85	93.75
Bilinear + seg full	21.42	47	83.86	93.75
JAFAR×4 + seg 1/4*	10.11	99	87.90	95.19
JAFAR×8 + seg 1/2	25.13	40	88.05	95.25
JAFAR×16 + seg full	84.78	12	87.91	95.17
Full pipeline, bilinear×4	6.12	163	–	–
Full pipeline, JAFAR×4	11.12	90	–	–

This call adds input preprocessing, the temperature-weighted traversability of Eq. (6), and the upsampling of segmentation, traversability, and confidence to input resolution. It excludes the host-to-device image upload, since the worker receives frames already resident in GPU memory. Averaged over three runs with two different segmentation layers, the complete semantic pipeline takes 11.12 ms (90 Hz; run means between 10.93 ms and 11.30 ms) with JAFAR×4 and 6.12 ms (163 Hz) with bilinear×4 upsampling, of which the upsampler accounts for 4.90 ms and 0.26 ms, respectively. The number of classes in the segmentation layer has no measurable effect on these timings.

Converting the DINOv3-s backbone from PyTorch to TensorRT gives an approximately 4× speedup (56 → 228 FPS) with no quality change (Table S9, Fig. S7). Bilinear upsampling yields identical mIoU (83.8%) at any resolution, confirming it adds no new information; since interpolation only resizes the predictions, this value also serves as the no-upsampler baseline, the quality of the raw patch-level predictions. JAFAR×4 at quarter resolution improves mIoU by 4.1 percentage points (87.9%) at 99 fps by attending to image-level cues, our recommended setting; ×8 and ×16 add under 0.2 percentage points while dropping to 40 and 12 fps. This per-image gain is statistically robust: on the ORFD test set ($n = 2{,}193$ images), JAFAR improves per-image mIoU by +3.5 to +3.8 points over bilinear at matched resolution (paired Wilcoxon signed-rank, $P < 0.001$ Holm-corrected; rank-biserial $r = 0.92$–0.97; JAFAR wins on 89–94% of images). Our recommended configuration is also faster than the fine-tuned segmentation baselines profiled identically on the AGX Orin: 99 fps versus 43 fps for DeepLabV3+ and 62 fps for SegFormer-B0 (1.6–2.3×). Unlike self-supervised methods (Mattamala et al. 2025; Jung et al. 2024; Ægidius et al. 2025) that incur online-learning overhead at deployment, SemMap needs only a single forward pass through a frozen backbone.

O.1 System Dataflow

Fig. S8 details the hybrid C++/Python dataflow of SemMap described in the main text, which lets the high-rate input streams be consumed without Global Interpreter Lock (GIL) contention. The reader threads hold the RGB and point-cloud streams in double buffers and poses in a ring buffer of 1,000 entries; buffer sizes follow the topic rates, with the pose buffer sized by a rule of thumb of one second of the 1000 Hz pose stream. Because sensor messages are held until a matching pose is available for interpolation, a lagging pose source, as we observed when testing with ZED visual-inertial odometry, requires correspondingly larger RGB and point-cloud buffers. CUDA MPS lets the two workers execute kernels concurrently, improving GPU utilization.

P Cross-Embodiment Deployment on a Humanoid

To show that SemMap is not specific to the bicycle, we deploy it on a Unitree G1 humanoid for factory navigation. The pipeline runs in semantic-only mode from a single ZEDX RGB camera, producing a 3-class segmentation that is converted to a 2D traversability image and consumed by an RL navigation policy. The task is no-go-zone avoidance: restricted areas demarcated by tape, where the enclosed region must be classified as non-traversable from its boundary rather than its surface (Fig. S9). The same frozen backbone and upsampler used on the bicycle are reused unchanged, with only the decoder and class-to-traversability mapping adapted to the factory classes, demonstrating cross-embodiment transfer between a high-speed wheeled robot consuming semantic elevation maps and a legged humanoid consuming 2D traversability images. As with our indoor and outdoor trail datasets, the factory training images were captured primarily with a handheld capture device and, in some cases, a smartphone, decoupling data collection from deployment.

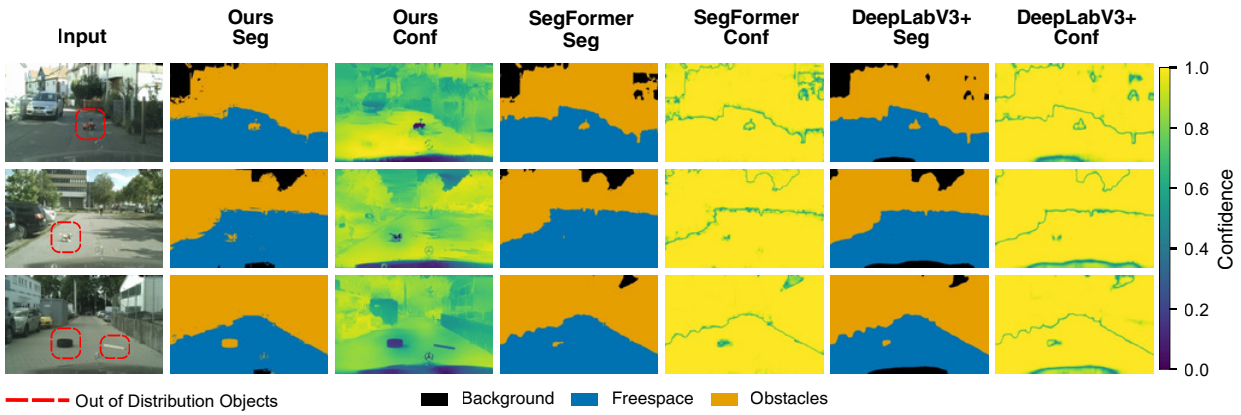


Figure S6. Qualitative OOD detection on Lost and Found. For each method: segmentation overlay and confidence map. Row 1: all methods detect the obstacle. Row 2: baselines fail while their softmax confidence stays high. Row 3: SemMap misclassifies the right obstacle, but cosine-similarity confidence is clearly low there, enabling recovery.

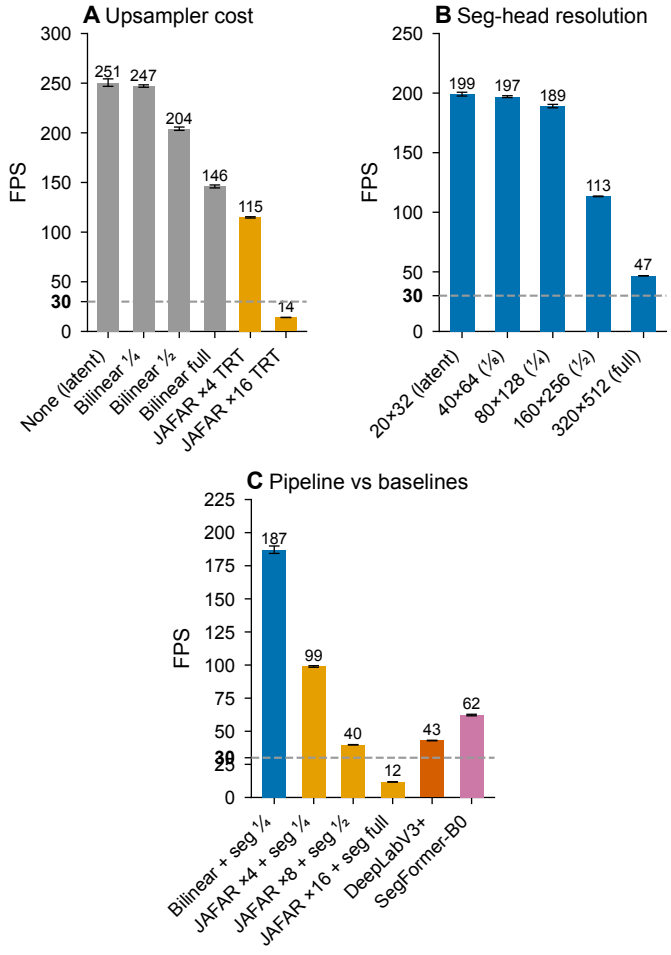


Figure S7. Inference profiling on AGX Orin 64GB. (A) Upsampler cost: bilinear is nearly free while JAFAR scales with the upsampling factor. **(B)** Segmentation-head resolution with bilinear upsampling. **(C)** End-to-end pipeline configurations (all TensorRT).

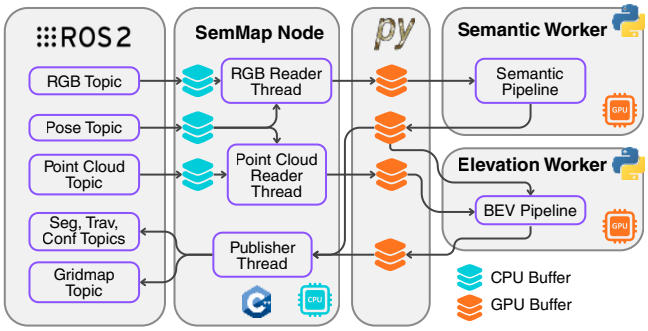


Figure S8. Dataflow architecture. A C++ ROS2 node subscribes to RGB, pose, and point cloud topics via dedicated reader threads, managing CPU buffers for synchronization. Data is passed to the Python layer via pybind11 into shared GPU buffers, which feed the semantic and elevation-mapping worker processes running as separate Python processes. Outputs (traversability, segmentation, confidence, grid map) are published back through a publisher thread. All inter-process communication uses GPU buffers to avoid memory transfers.

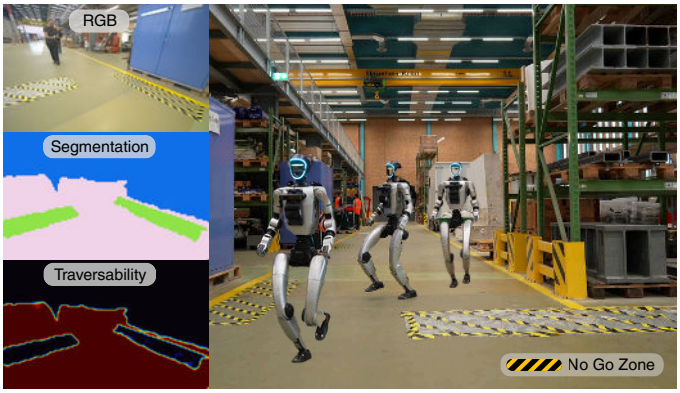


Figure S9. Cross-embodiment deployment on a Unitree G1 humanoid. SemMap runs in RGB-only mode and segments taped no-go zones into a 2D traversability image for the navigation policy. Only the decoder is adapted; the frozen backbone and upsampler are shared with the bicycle deployment.

Q Field-Deployment Summary

Our field deployment on the Allmend trail progressed through four testing days, summarized below:

- **April 8:** Initial autonomous tests comprising 24 runs. The system traversed long segments of the lower

Allmend trail, reaching distances of up to 242 m (76 s), and leading to a median traveled distance of 77 m. However, the policy lacked sufficient robustness to handle the human-made obstacles on the upper part of the trail, losing balance shortly after encountering them, exemplified by a median survival distance of 25 m.

- **April 24:** Testing was stopped after 10 runs (8 on the lower and 2 on the upper trail), with median traveled distances of 25 m and 6 m, respectively. Between the two April deployments, the trees along the trail had grown new leaves, and trees and grasses were flowering, changing the trail's appearance with fresh foliage, patchy shade, flowers and pollen, and less of the trail was segmented as traversable: compared to April 8th, a larger share of the driven path was marked untraversable (12 % vs. 3 %). In addition, without semantic inpainting, trail regions that the camera had not labelled remained untraversable. Before the next deployment, we recorded new footage on the trail, retrained the decoder, and introduced joint semantic and geometric inpainting.
- **May 8:** Autonomous navigation across 24 runs on both the upper and lower trail segments, with a longest continuous traversal of 233 m (74 s). The median on both the upper and lower parts was quite similar (90 m and 74 m). This deployment was the first to combine semantic-based inpainting with a version of SemMap trained on an expanded dataset, and it determined the tuning parameters used on the final day.
- **May 28:** Autonomous deployment comprising 21 runs, with continuous traversals of up to 210 m (51 s). For this day, we slightly widened the domain randomization ranges; all randomization parameters reported in this work refer to this final policy.

On each testing day, field operations began at approximately 10:00 and ended between 14:00 and 17:00, depending on system performance and data collection requirements.

Figure S10 shows the upper and lower segments of the Allmend trail with the checkpoints of Fig. 9, overlaid with the base trajectories of the six runs reported in Table 2, as estimated by our onboard LIO state estimator.

To evaluate system robustness, we analyzed the distance driven before a fall across all 79 runs using the Kaplan–Meier method (Fig. S11), with 36 runs on the upper and 43 runs on the lower part of the trail, and two runs ending in a stable track-stand. On the three evaluated days, the distance before a fall did not differ significantly between days on either trail section; only the April 24 deployment differs significantly. Runs on the lower trail typically covered more distance than runs on the upper trail, reflecting the difficulty of the human-made obstacles on the upper part.

R Ablation and Evaluation of Racing Policies

The evaluation environment comprised 10 instances of each terrain type across five difficulty levels, yielding a total of 850 distinct terrain patches. We simulated 4,000 robots per difficulty level with all domain randomizations disabled, and recorded the survival rate, time to reach the final platform, time spent off-trail, and average velocity.

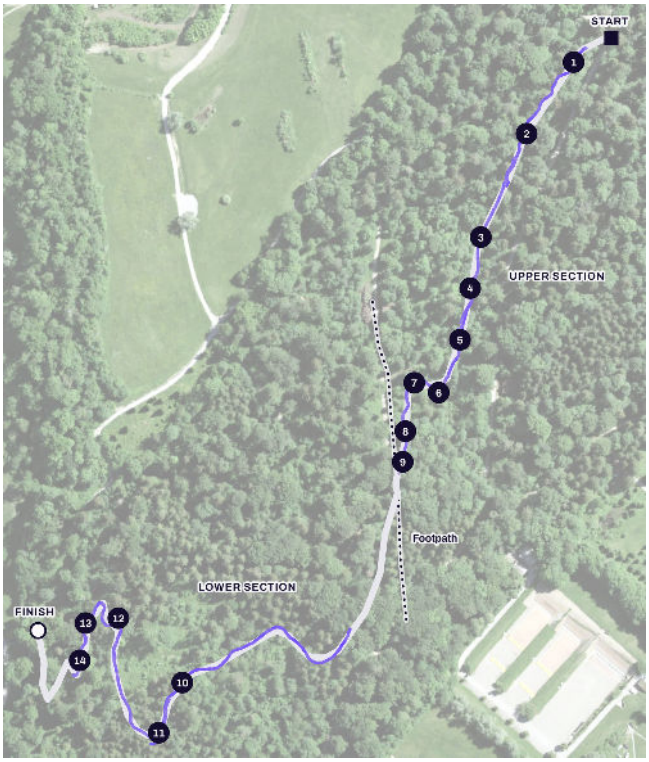


Figure S10. Trail coverage. Map of the Allmend trail (light grey) overlaid with the estimated base trajectories of the six runs in Table 2 (purple).

R.1 Hard Termination vs. Soft Reward

We compared two racing policies trained using the TL for 3,000 learning iterations, and $v_{\min} = 1$ (for faster convergence) under different boundary enforcement strategies: a hard termination upon trail violation (policy-1) versus a soft continuous reward for trail following (policy-2).

We evaluated the final trained policies in simulation to benchmark their comparative performance (Tab. S10, policy-1 and policy-2). In both cases, we did not terminate upon trail boundary violation. The data reveals that both policies adhere to the trail boundary constraints equally well, and policy-2 (average speed 2.726 m/s) is slightly faster and more robust than policy-1 (average speed 2.576 m/s).

R.2 With and without domain Randomization

To isolate the impact of domain randomization, we trained an ablation policy with all randomization events and noise terms disabled during both training and evaluation. We use the same training parameters as before (3,000 learning, $v_{\min} = 1$) for consistent comparison. As detailed in Table S10 as policy-3, the absence of domain randomization leads to slightly lower average speed (2.6 m/s) at comparable survival rates. We note that policies trained without these randomizations fail to transfer to the physical robot (strong vibrations lead to instantaneous falls).

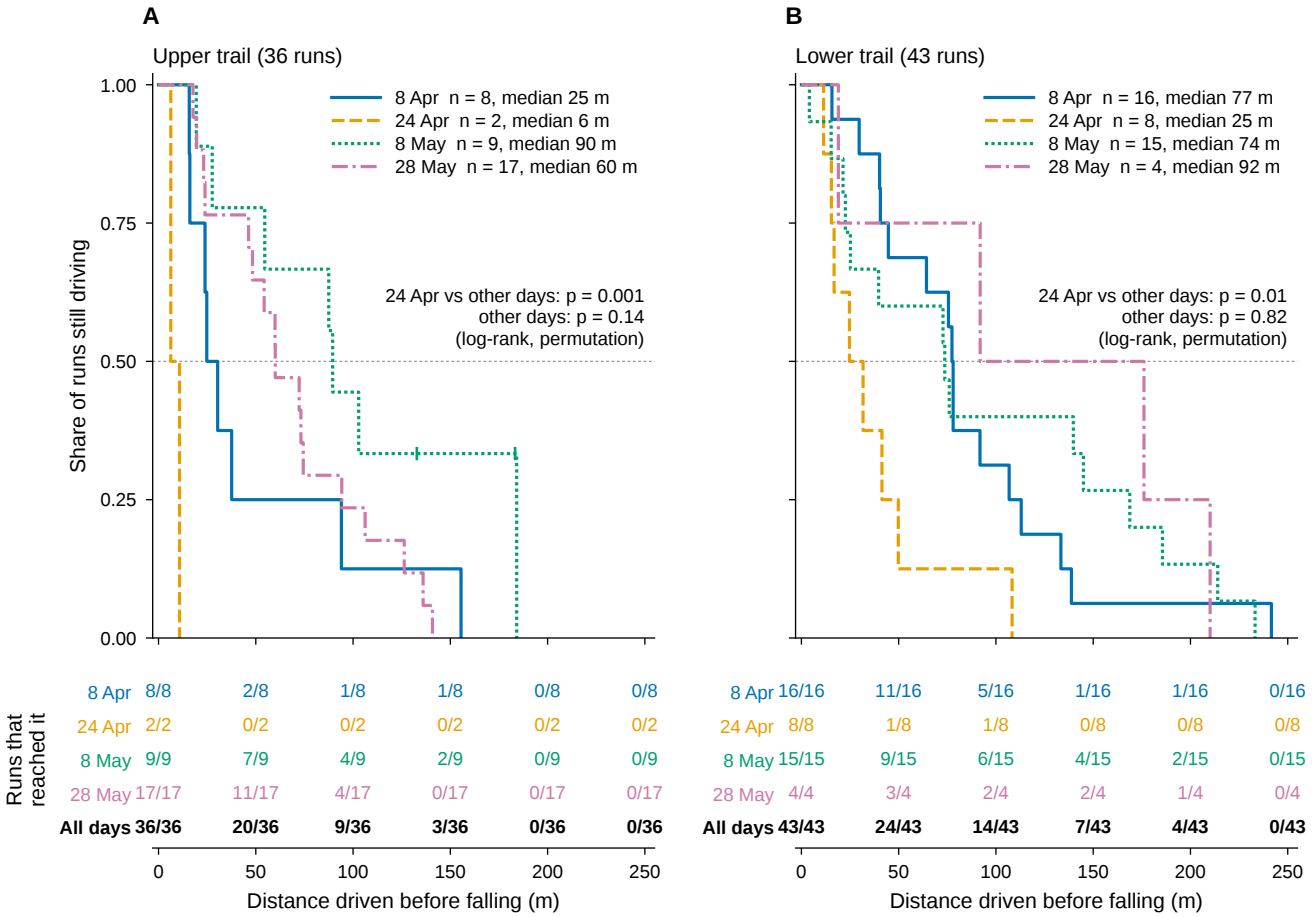


Figure S11. Deployment statistics. Kaplan–Meier estimates of the distance driven before a fall for all four deployment days (79 runs), shown separately for the upper (left, 36 runs) and lower (right, 43 runs) parts of the Allmend trail. Two of the April 24 runs were recorded on the upper part. The dotted line at 0.5 marks the median: the distance by which half of a day's runs had fallen (values in the legend). Plus marks denote the two runs that ended in a stable track-stand.

Table S10. Comparison of policies in simulation. Policy-1: trained with termination upon leaving the trail. Policy-2: trained without termination upon leaving the trail. Policy-3: trained without domain randomization. Each experiment (row) is averaged over 4,000 agents, with a single fixed seed. Training involved 3,000 learning iterations and we set $v_{\text{min}} = 1$.

Policy	Level	Survival Rate	Time to Goal (s)	Time Not on Trail (s)	Avg Speed (m/s)
policy-1	0	0.91	11.3	0.023	2.58
	1	0.90	11.3	0.022	2.59
	2	0.88	11.5	0.020	2.58
	3	0.82	11.8	0.021	2.58
	4	0.70	12.3	0.057	2.55
policy-2	0	0.94	10.6	0.017	2.74
	1	0.93	10.8	0.030	2.70
	2	0.92	10.9	0.018	2.72
	3	0.86	11.2	0.015	2.71
	4	0.78	11.8	0.051	2.67
policy-3	0	0.95	12.0	0.014	2.41
	1	0.97	11.3	0.019	2.58
	2	0.97	11.1	0.017	2.65
	3	0.94	11.1	0.017	2.70
	4	0.88	11.6	0.044	2.68