

A Supplementary Material

Videos and more information can be found on the project website:

<https://sites.google.com/view/wheeledlab>

B Design and Implementation Details

The typical ranges on friction, mechanical tolerances, damages, etc. for low-cost platforms are far from ideal. However, iterating and developing on individual platforms in unique environments is practically and pedagogically critical, especially for real-world robotics. We hope that this work allows practitioners to focus their efforts on precisely this development cycle rather than infrastructure. The following implementation details should be regarded as a starting point and potential reference for troubleshooting inevitable issues.

Isaac Lab environment configurations primarily include parameter settings and reward functions, and can be quickly parsed while efficiently communicating environment behavior. We highlight some notable details in this section.

B.1 Drifting

We found that drifting was an exemplary task to demonstrate the physical Sim2Real gap. Much of the development cycle was focused on closing this gap, as supported by countless other seminal works [8, 6].

B.1.1 Actuators

In many related works for agile tasks, data is typically collected to assess actuator response and estimate gains. However, this process can be unfamiliar to beginning roboticists or non-traditional enthusiasts. *Instead, all actuator parameters are extracted exactly from their technical specification sheet online, and gains are domain-randomized across environments.* Achieving the task in this manner helps to convey that, while domain expertise can certainly improve performance (domain randomized policies are known to find conservative solutions [10]), it is not strictly necessary through modern methods.

It is worth noting that a few “Sim2Real2Sim” (or Sim2Sim) cycles were spent narrowing the randomized gain ranges for the throttle actuator settings. Due to neural network policy tendencies to exhibit “bang-bang” (on-off) behavior, gain values can become extremely important to the actual forces exerted on the vehicle.

Establishing and improving pipelines for this process can be an accessible yet effective future contribution for use by potential practitioners.

B.1.2 Friction

Initially, the domain randomization over friction was used to cover a wide range (0.2 to 0.8) of potential friction values between the tape-covered tires and the carpeted ground. However, resulting policies would perform poorly and inconsistently on deployment. A long thread of Sim2Sim cycles was used to search for friction parameters but still resulting policies were not reliably transferring.

In the end, a cheap (less than 5 USD) spring scale was used to estimate the coefficient of friction, measured by dragging the vehicle with the scale along the carpet. This value (about 0.4) was then used as the midpoint of the friction randomization during training.

The project website also contains videos of deployments on different surfaces (e.g. finished concrete). Lower friction values appear to cause the vehicle to take wider turns (due to lack of turning friction) and spin out quickly during drift maneuvers. Enabling the platform to drift over multi-

544 ple surfaces is also of fundamental interest to the robotics community as it is a form of domain
545 adaptation.

546 **B.1.3 Articulations**

547 While we initially used the MuSHR [2] articulations defined by the open-source URDF file, we
548 found that its modeling inaccuracies affected the precise control expectations of the platform. As a
549 result, we improved the articulations: (1) by moving the steering linkages from the wheel axis to the
550 actual steering joint and (2) by articulating the suspension joints of the vehicle.

551 Steering inaccuracies were noticeable due to the gap between turning radii in real and sim. The
552 moment arm on the linkage can also adversely affect the tracking of the steering joint during high-
553 velocity maneuvers. This can even be observed in simulation when steering joints behave strangely
554 asymmetrically with unsuitable actuator settings.

555 Suspension helps close the gap on load shifting during turns. With a suspension travel of 4 cm,
556 these shifts have consequences in the total friction and “thrust” produced while throttling through
557 turns. Interestingly, suspension articulation also helped to stabilize the simulation. At a wheel
558 speed of 3 m/s, the rotation rate of the wheels reaches 60 rad/s, occasionally resulting in collision
559 penetrations during simulation which cause unrealistic “jumping” through drifted turns. Damped
560 suspension joints helped to stabilize these collisions without requiring smaller simulation time steps
561 which would significantly lengthen training times.

562 **B.1.4 Rewards**

563 Six rewards were designed for this task:

- 564 i.) **Cross-Track Distance.** The lateral distance of the vehicle from an oval track line. This
565 is defined without time-dependent reference trajectories as often done in model-based ap-
566 proaches or even in deep RL methods [30]. This is highly weighted otherwise, the agent
567 tends to take larger, safer turns.
- 568 ii.) **Velocity.** Reward for high speeds.
- 569 iii.) **Side-Slip.** Only stable side-slip angles were rewarded. Stable side-slip angles are manage-
570 able through the steering limits (0-30°).
- 571 iv.) **Progress.** Reward for making progress along an arbitrary direction (counter-clockwise, in
572 this work).
- 573 v.) **Turn-Energy.** Shaping term for increasing velocity rewards, specifically when inside turn
574 regions.
- 575 vi.) **Turn-Left-Go-Right.** Shaping term for encouraging exploration to discover an alternate
576 turning mode (e.g., drifting) by counter-steering. Strictly positive when the angular velocity
577 and the steering command are opposite to each other. 0 otherwise.

578 Penalties were also given for going out-of-bounds.

579 **B.2 Elevation**

580 **B.2.1 Training**

581 Training elevation in simulation proved to be a challenging task. Training would often collapse
582 into one of: (1) seeking goals while avoiding obstacles or (2) climbing ramps. Striking a balance
583 between the two often required re-designing the scene or task configuration in simulation.

584 As observed for model-based methods [14], evaluating uneven terrain traversal for vehicles can be
585 challenging due to the reasonable alternate mode of avoiding risky terrain altogether, which may
586 often be faster. However, the primary goal of this work is to better enable the broader community to

engage with state-of-the-art methods in elevation-based robotics. Thus, the reward and scene were constructed to incentivize the agent to actively seek out risky terrain while penalizing failures.

Supported by machine learning literature, noise injection becomes less effective for generalization at higher dimensions. For states such as elevation maps, perceived noise is likely more structured than the additive Gaussians injected through observation corruption. While debugging, we found that the policy would improve if the elevation features were brought closer in shape to the features seen in the simulation. To resolve this without “training on test”, the scene was augmented with slopes of various gradations and heights.

B.2.2 Suspension

As mentioned in Appendix B.1, suspension joints were added to the open-source MuSHR articulations. This is critical for elevation where maintaining momentum up a ramp helps training stability. Without suspension articulation, climbing ramps have an abrupt and destabilizing impact at higher speeds.

B.2.3 Rewards

- i.) **Goal Velocity.** Rewards velocity projected onto the goal heading vector.
- ii.) **Z-Position.** Rewards larger z-positions (gained through ascending ramps).
- iii.) **Falling Penalty.** Penalizes negative body-frame z velocities.
- iv.) **Roll on elevation.** Penalizes roll angles while on elevation features.

B.3 Visual

While visual information is essential in modern robotics to solve complex real-world problems, simulating such information requires state-of-the-art rendering techniques to close the Sim2Real gap. Despite such computationally heavy efforts, machine learning models trained with simulated data often fail at real-world generalization due to diverse types of environmental changes, e.g., illuminations, weather, etc. Instead, in this work, we “simplify” visual information from both simulation and real by compressing RGB pixels to grayscale, closing the Sim2Real gap. Still, we observed that such compressed visual information exhibits a non-trivial amount of Sim2Real gap. This section describes how we tackle this problem by providing details of our environment, training augmentations, architectures, and reward designs.

B.3.1 Visual map generation

We designed the problem by encoding traversability into color information. Black tiles are non-traversable regions while white indicates safe to travel. We divided the whole map into multiple sub-environments that are individually configurable. Then, we randomized white paths over the black area in each sub-environment. Specifically, each sub-environment comprises 9 cells, and we sample a start point in each cell and generate paths toward random endpoints in the sub-environment. We made the number of random paths from a single start point configurable, allowing the users to adjust the difficulty of the training (see Fig. 8). This can later be used for curriculum learning by assigning different difficulties for sub-environments. We tested the generated map with more than 2,000 agents in parallel, running seamlessly with a single Nvidia RTX 3080 GPU. The algorithm overview for sub-environment generation is presented in Alg. 1.

B.3.2 Model Architectures

We adopt a multi-layer perceptron (MLP) as our basic policy network. For ablations, we add a simple 3-layer convolutional neural network (CNN) that serves as an image feature extractor before the policy networks. The extracted features are concatenated to the rest of the observation terms, such as linear/angular velocities and last action, and fed into the MLP policy networks. We observe

Algorithm 1 Visual Policy Sub-environment Generation

```
1: Input:
2:  $(H_{\text{env}}, W_{\text{env}}) \leftarrow \# \text{ of rows and } \# \text{ of columns for each sub environment}$ 
3:  $(H_{\text{cell}}, W_{\text{cell}}) \leftarrow \# \text{ of rows and } \# \text{ of columns for each cell}$ 
4:  $N_{\text{walkers}} \leftarrow \# \text{ of walkers for path generation}$ 
5:
6: Pseudo-code:
  ▷ Initialize traversability map; 0 - not traversable / 1 - traversable
7:  $\mathbf{T} \in \{0, 1\}^{H_{\text{env}} \times W_{\text{env}}} \leftarrow \mathbf{0}$ 
8:
  ▷ Sample start points from each cell
9:  $\mathbf{P}^{\text{start}} \leftarrow []$  ▷ Initialize an array to save start points
  ▷ Sample random start points for each cell
10: for row  $r = 0, 1, \dots, H_{\text{env}}/H_{\text{cell}}$  do
11:   for col  $c = 0, 1, \dots, W_{\text{env}}/W_{\text{cell}}$  do
12:      $p_{\text{row}} \leftarrow \text{UniformSampling}(0, H_{\text{cell}}) + r \cdot H_{\text{cell}}$ 
13:      $p_{\text{col}} \leftarrow \text{UniformSampling}(0, W_{\text{cell}}) + c \cdot W_{\text{cell}}$ 
14:      $\mathbf{P}^{\text{start}} += [(p_{\text{row}}, p_{\text{col}})]$  ▷ Append the start point to  $\mathbf{P}^{\text{start}}$ 
15:   end for
16: end for

17: for  $(p_{\text{row}}, p_{\text{col}})$  in  $\mathbf{P}^{\text{start}}$  do
18:   for iter =  $0, 1, \dots, N_{\text{walkers}}$  do
  ▷ Sample random end points
19:   Do
20:      $q_{\text{row}} \leftarrow \text{UniformSampling}(0, H_{\text{env}})$ 
21:      $q_{\text{col}} \leftarrow \text{UniformSampling}(0, W_{\text{env}})$ 
22:   do While  $\mathbf{T}(q_{\text{row}}, q_{\text{col}}) == 1$ 
  ▷ Generate random paths and update traversability map
23:      $\text{GenerateRandomPaths}(\mathbf{T}, (p_{\text{row}}, p_{\text{col}}), (q_{\text{row}}, q_{\text{col}}))$ 
24:   end for
25: end for
26: Return:  $\mathbf{T}$ 
```

631 that inducing CNN takes longer training time to learn optimal actions and often collapses into a
632 trivial solution - agents decide to spin in a circle, only maximizing the velocity rewards.

633 B.3.3 Image Augmentations

634 To close the sim-to-real gap, we apply image augmentations for policy training. This includes
635 aggressive color jittering (*i.e.*, brightness, contrast, saturation, and hue adjustments) and Gaussian
636 blur. As demonstrated in both qualitative and quantitative evaluations, image augmentations play
637 a crucial role in closing sim-to-real gaps. Further augmentations or randomization can be easily
638 applied to our code design.

639 B.3.4 Rewards

640 Two rewards were designed for this task

- 641 i.) **Velocity Reward.** Rewards higher velocities
- 642 ii.) **Traversability Reward and Penalty.** Rewards being on white regions, and penalizes it for
643 being on black regions

644 B.3.5 Terminal Conditions

645 We experimented with different terminations. In addition to the standard time-out and out-of-bounds
646 terminal conditions, we added a terminal condition when the robot is on black. However, this often
647 resulted in a lack of exploration, and policies that produced “jerky” or undesirable actions, such as
648 idling at the initial location.