Generating a Terrain-Robustness Benchmark for Legged Locomotion: A Prototype via Terrain Authoring and Active Learning

Chong Zhang[1†]

***Abstract*— Terrain-aware locomotion has become an emerging topic in legged robotics. However, it is hard to generate challenging and realistic terrains in simulation, which limits the way researchers evaluate their locomotion policies. In this paper, we prototype the generation of a terrain dataset via terrain authoring and active learning, and the learned samplers can stably generate diverse high-quality terrains. Hopefully, the generated dataset can make a terrain-robustness benchmark for legged locomotion. The dataset and the code implementation are released at https://bit.ly/3bn4j7f.**

I. Introduction

Terrain-aware locomotion has been drawing attention in the domain of legged robotics [1] [2] [3] [4]. To overcome all kinds of unstructured terrains in the wild, legged robots need to achieve great robustness in locomotion. However, such robustness is difficult to quantify, especially for the unstructured terrains that can hardly be foreseen. As a result, existing works tend to manually build specific environments and report the success rate for several toy experiments [1] [4] [5] [6].

In light of this, our paper seeks a way to statistically quantify the robustness in terms of terrains for different locomotion policies. In other words, we are trying to build a terrain-robustness benchmark for legged locomotion. The terrain samples should be diverse and resemble those in the wild, so that the robustness can be measured and improved in simulation. Also, these terrains should be challenging to traverse in order to discriminate between different policies. Besides, different from [6] [7], terrains generated in this paper are more unstructured rather than parameterized. Fig. 1 exhibits some of our terrain samples in the benchmark dataset. To our knowledge, no existing work has built such a benchmark, and this paper makes a prototype for it.

To this end, three challenges must be solved: 1) to achieve reliable quantification of robustness, the terrain samples should resemble real terrains in the wild; 2) to achieve the easy generation of high-quality terrains (i.e., be challenging to a user-specified extent), the generation process should be somewhat controllable; 3) the sampler must maintain the terrain quality and the diversity simultaneously. Among them, the first two challenges can be viewed as mapping a controlled input to a realistic terrain, and the last challenge is to obtain a sampling policy that can generate diverse high-quality inputs. Hence the problem is naturally divided into two parts: one for terrain generation, and one for sampling.

† Corresponding author. Email: chozhang@ethz.ch

[1]Chong Zhang is with the Department of Mechanical and Process Engineering, ETH Zurich, Switzerland. Email: chozhang@ethz.ch

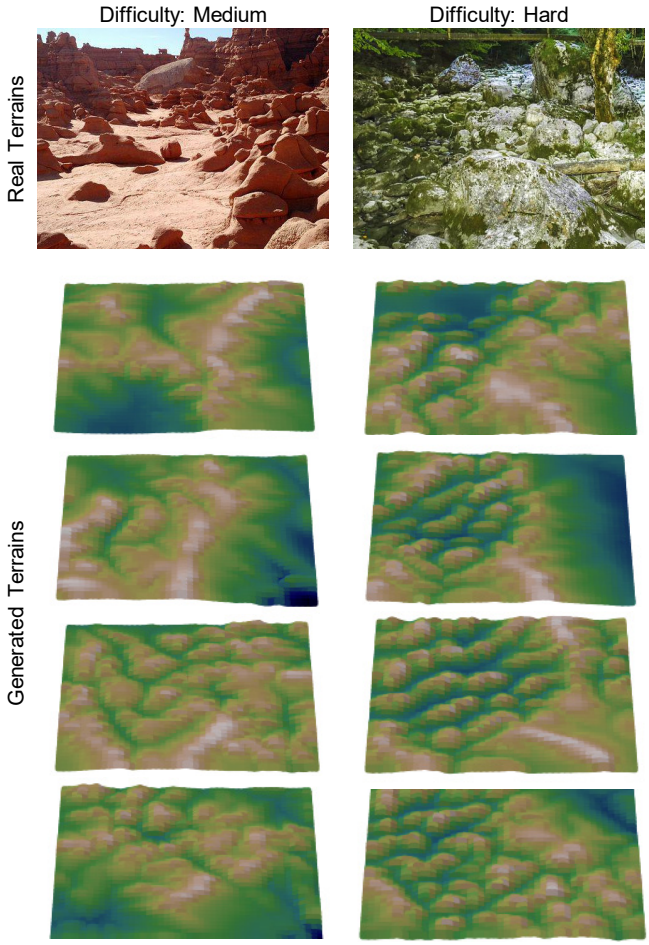

Fig. 1. Cherry-picked photos of real terrains and visualization of generated terrains in the benchmark. The first two rows are very pairs that, the generated terrain sample resembles the real terrain. The real terrains are irrelevant to dataset generation, but reflect how the generated terrains are realistic. The other three rows compare the terrains generated for two difficulty levels, "medium" (left) and "hard" (right), where the pairs in the same row share somewhat similar terrains but the "hard" one is more complex and contains more ravines.

With the idea of fractal [8] [9], we believe that a local terrain elevation map can be obtained via resizing a realistic terrain of any appropriate scale, including the scales of the terrain data obtained by remote sensing techniques. Based on this and referring to [10], we use the conditional generative adversarial networks (GANs) to generate terrains from specified control points as the inputs, and the training dataset that the outputs should resemble comes from the US map consisting of digital elevation models [11]. In this way, together with some tricks, we can generate realistic

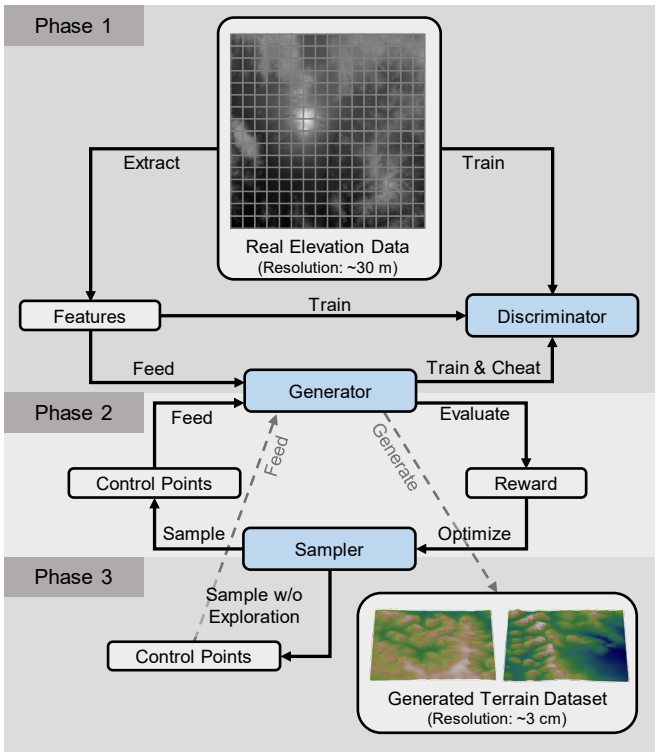

Fig. 2. The pipeline of the prototyped dataset generation method. In phase 1, a generator and a discriminator are trained to ensure that the generated terrains are realistic. In phase 2, the sampler learns through exploration and exploitation to sample high-quality control points for terrain generation. In phase 3, the trained sampler does no more exploration and is used to sample high-quality terrains for the dataset.

and unstructured terrains from small-size control point sets, which tackles the first two challenges.

To tackle the third challenge, we model the generation of control points as a Markov decision process (MDP) with a directed acyclic graph (DAG) structure, and refer to the arising active learning method generative flow networks (GFlowNets) [12] to get the control point samplers. It has been shown that, such an active learning method not only outperforms sampling-based methods in data efficiency, but also maintains a high diversity of candidates compared with reinforcement learning methods [13]. In this paper, the learned samplers can generate diverse hard-to-obtain samples or sets of control points, with high quality regarding legged locomotion.

Based on how the three challenges are coped with, this paper proposes a pipeline to learn the desired sampler, as is shown in Fig. 2. Three models are trained: 1) a terrain generator, 2) a terrain discriminator, and 3) a control-point sampler. The terrain discriminator ensures that the generated terrains are realistic. Using the sampler and the terrain generator, we can sample diverse high-quality terrain samples after only a few training iterations.

Our contributions in this paper are summarized as follows:

1) An adaptation of realistic terrain authoring to take small-size sets of control points as the inputs;
2) Learned control point samplers to generate high-quality terrain samples for legged locomotion;
3) The generation of the first benchmark dataset to our knowledge to statistically evaluate the robustness of legged locomotion on unstructured terrains.

II. Related Works

A. Terrain-aware legged locomotion

This paper tries to generate a terrain-robustness benchmark for terrain-aware legged locomotion, which is an emerging topic in the robotics community. Generally speaking, there are two kinds of solutions to terrain-aware locomotion: optimization-based methods and learning-based methods.

Optimization-based methods typically obtain reasonable footholds and trajectories through searching and optimization, with heuristic scores or losses [5] [6]. Often, the robustness tests are done on stairs or stacked bricks and tiles, without many trials or high terrain diversity.

Learning-based methods learn trajectories [14], footholds [2] [4] [15] or joint-level outputs [3] [16] [17] in a data-driven way. Although various terrains are generated in simulation to feed the data-hungry learning process, they either consist of simple stairs and steps, or are non-realistic and generated from uncontrollable noises. Despite some successful deployment in the wild [3], robustness tests are done either on limited real terrains, or statistically on non-realistic low-quality ones in simulation, on top of the heuristically defined structured terrains.

B. Terrain generation

Terrains, in the domain of terrain-aware legged locomotion, are typically represented by heightmaps, i.e., two-dimensional matrices of real numbers indicating the height at different points. A traditional method for terrain generation is to use the Perlin noise [18], as is adopted by existing works [3] [7]. Although policies can be trained in simulation with such terrains, verifications must be done on real robots after sim2real, because using Perlin noise does not lead to realistic heightmaps [9].

Alternative methods are to generate fractal terrains, e.g., to use the diamond square algorithm [19] and the fractal brownian motion algorithm [20]. It is hard to regard them as realistic, though.

An emerging way to generate realistic terrains is to use GANs [21], where a discriminator tries to classify whether a sample comes from the dataset, and a generator tries to cheat the discriminator by generating samples from noises. Examples of GAN-based terrain generation are [22] and [23]. Yet, to achieve partially controllable generation and actively generate a dataset, we need interactive terrain authoring based on conditional GANs [10]. To be specific, the discriminator classifies whether the samples together with certain features are from the training dataset, and the generator generates fake samples from not only noises but also the features. Finally, the generator can generate realistic terrains from given input features, and the noises only affect small-scale details.

C. Active learning

Active learning is a technique that selects a subset of all possible candidates and gets them labeled to improve the model performance [24]. In this paper, we adopt this technique because, it can be extremely hard and data-inefficient to directly get high-quality terrains by random sampling. Also, it can be time-consuming to evaluate a terrain, e.g., reporting the performance of multiple gaits on the same terrain with varying dynamical parameters, although we would only use a heuristic terrain score for a prototype in this paper.

Unlike the common way active learning is applied to get more labeled data, in this paper, we aim to get an effective sampler that can provide a diverse set of high-quality terrains. The recently proposed GFlowNets [12] has provided a promising solution where candidates are sampled in proportion to their given rewards. Different from reinforcement learning methods that only provide a low-variance policy to maximize the expected return [25], GFlowNets can maintain the diversity of selected candidates, which has been empirically verified in domains such as molecule synthesis [13]. Another choice for diverse sampling is the iterative Markov chain Monte Carlo (MCMC) methods [26] [27] which are data-inefficient and vulnerable to local exploration.

III. Generating Terrains as Trajectories

In this section, we present how we generate realistic terrains from features or control points, and how the terrain generation can be viewed as an MDP.

A. Conditional GANs for terrain authoring

As is mentioned, we use the interactive terrain authoring method in [10] to generate terrains from controlled inputs, which is based on conditional GANs. The training of the model goes as illustrated in Fig. 3. For each sample x in the dataset, we extract features u from it, and (x,u) makes a positive sample for the discriminator D. With the generator G and the random noise w, $G(u,w)$ denotes the generated fake sample, and $(G(u,w),u)$ makes a negative sample for D. With the positive sample and the negative sample, D predicts the pixel-level classification for x and is trained with the binary crossentropy loss:

$$\begin{aligned} L_D(x,u,G,w) = &-\mathbb{E}_{\text{pixel}}[\log(D(x,u)) \\ &+\log(1-D(G(u,w),u))]. \end{aligned} \tag{1}$$

Here we take the pixel-level prediction to ensure detailed terrain outputs according to [28] and a third-party implementation [29] of [10].

The generator G learns to fool the discriminator D. Following [28] and [29], the loss for G is definend as:

$$\begin{aligned} L_G(x,u,D,w) = \mathbb{E}_{\text{pixel}}[&-\lambda_1^G \log(D(G(u,w),u)) \\ &+\lambda_2^G |x-G(u,w)|], \end{aligned} \tag{2}$$

where λ_1^G and λ_2^G are manually defined coefficients, and $|\cdot|$ denotes the absolute error.

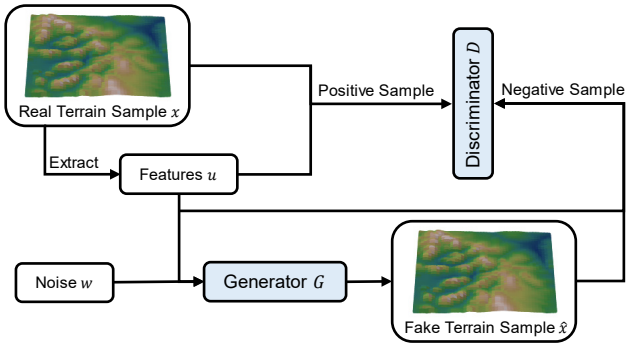

Fig. 3. The conditional GAN system for terrain authoring. The discriminator D tries to distinguish the positive samples and the negative samples, while the generator G learns to fool the discriminator D.

B. Implementation details

1) Dataset: Regarding real heightmaps, we cherry-picked some images with rich elevation information from the USGS 3D elevation dataset [11], which can be reproduced in our codes. The images are of 1 arc-second resolution (≈ 30 m), and are of size $(3600, 3600)$. We split each image into 256 patch samples of size $(225, 225)$.

2) Target resolution: Our initial purpose is to generate 2.5 cm-resolution heightmaps, leading to a total size of $\sim 5.6\text{ m} \times 5.6\text{ m}$, and the maximum height is 10×2.5 cm if no additional slope added. This is specified for the size and mobility of some popular legged robots, e.g., ANYmal [30] and A1 [31]. Despite the different resolution from the dataset, with the idea of fractal, we believe the used elevation data can be rescaled to the size we want while remaining realistic, as is verified in Fig. 1. The generated terrains can also be rescaled to other resolutions, e.g., 3 cm or 5 cm.

3) Selected features and masking: In [10], different kinds of features are extracted: rivers, ridges, peak points, and basin points. However, it is intractable to automatically draw rivers and ridges as humans intuitively do. Also, we found basin points sometimes hard to detect in our dataset. Hence, we detect pits instead via the pysheds library [32], and detect peaks by inverting the elevation for pit detection. The features we used are peak points and pit points, and they are represented as 0-1 matrices of size $(225, 225)$ to feed the models.

Yet, in this way, there can be too many features on one terrain. Sometimes, up to 200 feature points can be detected, which makes it hard to generate high-quality sets of control points. Also, we found that feature points are often too intensive at certain areas. Thus, borrowing insights from masked autoencoders [33], we believe most of the points can be redundant and we randomly masked the points with a probability of 75% per visit. In brief, extracted features in our paper are peak points and pit points that are masked with a probability of 75%.

4) Models: We refer to the model structures in [29], where the generator is a U-Net [34] as in [28], and the discriminator consists of several convolutional layers, finally activated by the sigmoid function for pixel-level prediction. Despite the

noise inputs, the output terrains only vary little in small-scale details for fixed features after training, which can be omitted.

5) Low-pass filtering: After training, the generator generates realistic terrains as expected. However, due to the instability of neural network outputs, there can be spikes. Thus, we applied a low-pass filter to the generated terrains as post-processing. To be specific, after empirical evaluation, we applied the Gaussian blurring with kernel size $(5,5)$ and $\sigma_x = \sigma_y = 1$ for unscaled height outputs in $[-1,1]$. The heights are then rescaled to $[0, \text{maximum height}]$.

C. A DAG for terrain generation

The terrain generation can be viewed as MDP trajectory generation, where a set of control points as the input features basically determines the generated terrain. Specifically, starting from an empty control point set, the MDP goes as follows:

1) Three kinds of actions can be taken: to terminate, to add a peak point, or to add a pit point;
2) If the action is to terminate, a terrain is generated with the existing control points, otherwise the new point is added to the control point set.

To make the problem tractable, we assume:

1) Two control points that are too close to each other do not make sense, and the small change in the point positions does not lead to significant changes in the terrain.
2) There cannot be too many control points, which will only lead to redundancy and complexity.

Thus we discretize the rows and the columns to 75×75 grids for actions, which means that one grid represents the center of 3×3 pixels. Each point-adding action occupies a grid, limiting the number of feasible actions to $2 \times 75^2 + 1$ at most. We also force the MDP to terminate after there have been 60 points, which limits the complexity of the problem.

The states, as assumed above, can be defined as a set of points:

$$s = \{p_1, \ldots, p_l\}, 0 \le l \le 60, \tag{3}$$

where s is empty if $l = 0$. Each point is the combination of its position and its attribute (peak or pit). Mathematically, we represent each point as a 3-d vector, where the first dimension is the x coordinate normalized to $[-1,1]$, the second dimension is the y coordinate normalized to $[-1,1]$, and the third dimension is 1 for peaks and -1 for pits. Thus the features $u = u(s)$ are easily determined by the state s.

The action a is also represented as a 3-d vector except for the termination. Exactly, it is the 3-d vector to represent the added point in the state.

Such an MDP makes a DAG because, one state can have multiple (literally the number of points in the state) parent states, and a fewer-point state can not have a more-point parent state. Denote the transition of states via actions by $s' = T(s,a)$, we have:

$$\text{card}(T(s,a)) = \text{card}(s) + 1, \tag{4}$$

where $\text{card}(\cdot)$ denotes the cardinality of a state. A trajectory τ can be represented as $\tau = (s_0, \ldots, s_n)$ in that the transition is deterministic, and s_0 is exactly $\emptyset$. The trajectory length is defined as the cardinality n of the final state s_n.

IV. GFlowNets for Terrain Generation

In this section, we detail our method to generate high-quality control point sets via GFlowNets.

A. Formulation and optimization

Equivalent to but slightly different from the GFlowNets in [12] [13], in our configuration for the state space $\mathbb{S}$ and the action space $\mathbb{A}$, we define an ideal flow function $F : \mathbb{S} \times \mathbb{A} \to \mathbb{R}^+$ s.t.

$$\sum_{T(s,a)=s'} F(s,a) = \sum_{a \in \mathbb{A}^*(s')} F(s',a), \forall s' \in \mathbb{S} \backslash \{\emptyset\}, \tag{5}$$

where $\mathbb{A}^*(s')$ denotes the feasible action set of s', i.e., to terminate the MDP or to add a peak or pit point at an unoccupied grid. For the termination action a^T, the boundary conditions are satisfied:

$$F(s,a^T) = R_G(s) = \mathbb{E}_w R(G(u(s),w)), \forall s \in \mathbb{S}, \tag{6}$$

where $R(\cdot)$ is the reward for a generated terrain. Because the reward for the same s varies little w.r.t. the noise w, we can sample only once to estimate the termination flow. Finally, when the sampling probability goes as

$$P(a|s) = \frac{F(s,a)}{\sum_{a' \in \mathbb{A}^*(s)} F(s,a')} \propto F(s,a), \forall a \in \mathbb{A}^*(s), \tag{7}$$

the probability of sampling a final state s^T at termination satisfies

$$P(s^T) = \frac{R_G(s^T)}{\sum_{s \in \mathbb{S}} R_G(s)} \propto R_G(s^T), \forall s^T \in \mathbb{S}. \tag{8}$$

We refer readers to [12] and [13] for proof.

Yet, since s is of size 60 at maximum and $A^*(s)$ can contain > 10000 feasible actions, we have to approximate the flows by a neural network. In this paper, we approximate the logarithm $\hat{F}_{\log}$ of the flows, and the flow matching loss w.r.t. each trajectory τ is defined as follows for optimization:

$$L_F(\tau) = \sum_{s' \in \tau \backslash \{s_0\}} \left(\log \left[\varepsilon + \sum_{T(s,a)=s'} \exp \hat{F}_{\log}(s,a) \right] - \log \left[\varepsilon + R_G(s') + \sum_{a \in \mathbb{A}^*(s') \backslash \{a^T\}} \exp \hat{F}_{\log}(s',a) \right] \right)^2, \tag{9}$$

where ε is a small positive value and we take 10^{-6}. During the training, new trajectories are sampled with approximated flows according to (8), and the approximation Model is optimized according to (9) on a batch of trajectories in an "off-policy" way.

B. Scoring and rewarding

The generated terrains are to be evaluated to get the challenging ones for robots to traverse. This can be done from many aspects, e.g., to test different policies on the terrain. In this paper, we use the heuristic foothold score as in [5] to prototype the learning-based dataset generation. Such a score was used to represent the safety of a foot location, and we use the mean score to indicate the difficulty over the whole terrain. To be specific, the score β for a terrain heightmap h is defined as

$$\beta(h) = \lambda_1^\beta \sigma(k(h)) + \lambda_2^\beta \overline{k(h)}^2 + \lambda_3^\beta \frac{\overline{|h-\bar{h}|}}{h_0}, \qquad (10)$$

where $\lambda_{1,2,3}^\beta$ are the weights for edges, slopes, and roughness, $\sigma(\cdot)$ is the standard deviation, $k(h)$ is the slope angles divided by $\frac{\pi}{2}$, and h_0 is the designed maximum height. In our implementation, $\lambda_1^\beta = 0.3$, $\lambda_2^\beta = 0.5$, $\lambda_3^\beta = 0.2$.

Typically, for our generator trained in Sec. III, the scores are within $[0.02, 0.05]$, and the effect of the noise inputs on the score is empirically less than 0.005 even for extreme cases. Yet, with our learning technique, the sampled terrains can even reach a high score of 0.08, or 10^{-3}-level deviation from a specified score value, while maintaining the diversity.

To demonstrate the efficacy of our method, we designed two tasks for generation: one called "hard" for the very difficult terrains with a high score, and the other called "medium" for terrains with a score close to 0.055 which is also hard to achieve via random sampling. The performance will be displayed in Sec. V-B.

The reward function $R(\cdot)$ for the "hard" task is defined as

$$R(h) = \varepsilon + 10^{150\beta(h)-6}, \qquad (11)$$

which makes the probability of sampling a terrain ~ 30 times higher if the score is 0.01 larger according to (8).

The reward function $R(\cdot)$ for the "medium" task is defined as

$$R(h) = \varepsilon + \exp\left(\frac{0.1}{|\beta(h) - 0.055| + 0.005} - 10\right), \qquad (12)$$

which forms a steep spike around 0.055.

C. Neural networks for flow approximation

The neural network structure we used for $\hat{F}_{\log}$ approximation with non-termination is illustrated in Fig. 4. To tackle the variable cardinality of states, we took an encoder-decoder structure, where the encoder encodes the state-related information, and the decoder couples the state-related information and the action-related information.

1) Encoder: The 3-d point vectors $p_1, \ldots, p_l$ in a state s are first concatenated with a 1-d cardinality-related value to inform the model of the "progress". With the maximum cardinality 60, we define this normalized cardinality value as $\frac{\mathrm{card}(s)}{0.5\times 60} - 1$. These vectors are then embedded into 128-d vectors. To deal with the case of empty set s, we introduce an additional learned 128-d vector which is concatenated to the embedded vectors, making in total $\mathrm{card}(s)+1$ vectors of length 128 to represent the states.

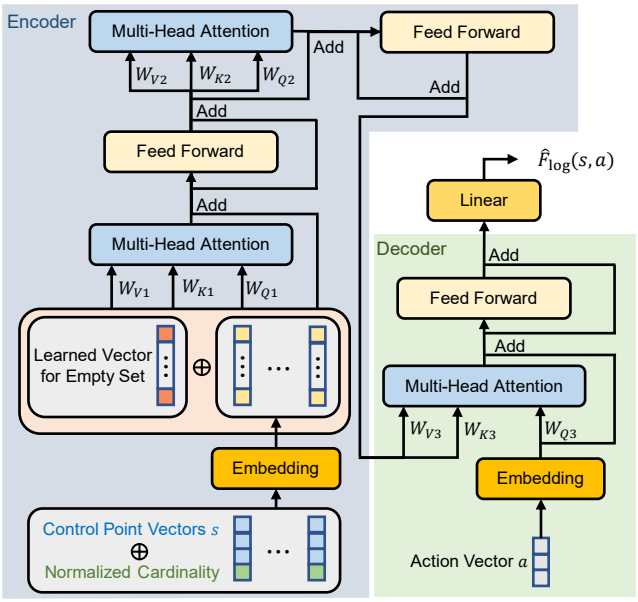

Fig. 4. The neural network structure for $\hat{F}_{\log}$ approximation.

These vectors are then fed into two repetitive combinations of the multi-head attention module [35] and the feed-forward layer. The multi-head attention module is with embedding dimension 128 and the number of heads 4, and the outputs are added with the inputs as a residual skip connection. The feed-forward layer is a 128-d linear layer activated by the GELU function [36], and also takes a residual skip connection.

2) Decoder: The 3-d action vector a is first embedded to a 128-d vector. Then a multi-head attention module (with the same structure as in the encoder) takes the outputs of the encoder for the key source and the value source, and the query source is the embedded action vector. Then, with a skip connection from the embedded action vector, the outputs of the multi-head attention module are fed into a feed-forward layer with a skip connection.

Finally, the outputs of the decoder go through a linear layer to make the final output, i.e, to approximate the $\hat{F}_{\log}(s,a)$ value.

D. Other details for implementation

1) Sampling-based outflow estimation: For each state, there are more than 10000 feasible actions except for those with 60 points. For each state-action pair, the neural network is used to compute the flow approximation, which makes the training time-consuming. Instead, we estimate the outflow term $\sum_{a\in\mathbb{A}^*(s')\setminus\{a^T\}} \exp \hat{F}_{\log}(s',a)$ in (9), by randomly sampling 2000 actions and getting the average flow that is then multiplied by the cardinality of $\mathbb{A}^*(s')\setminus\{a^T\}$. This reduces the training time by 80%.

2) Biasing the $\hat{F}_{\log}(s,a)$ model output: The neural network outputs are around 0 initially, which makes an initial guess of the ideal flows. Yet, scores of the initially sampled terrains are within $[0.02, 0.05]$, which specifies the termination flow. Therefore, if the neural network outputs too large values compared with the termination flow, flow mismatching

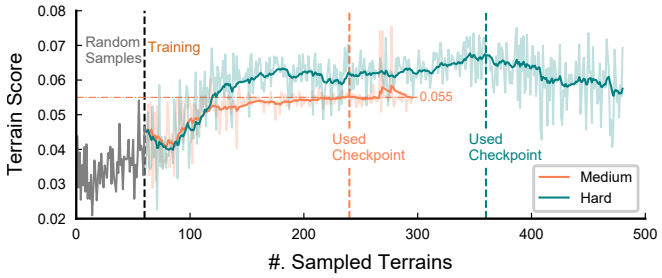

Fig. 5. Latest sampled terrain scores versus the number of samples during training of the samplers in two tasks. The opaque lines are smoothed values for the original ones in shadow.

between non-terminated state transitions will take the lead in the loss, which hinders the learning for better terrains. If the initial sampled rewards get too large, the sampler will tend to terminate early and get stuck in local exploration. Hence, we add a bias to the $\hat{F}_{\log}(s,a)$ output, 0 for "hard" and -6 for "medium", to seek a balance.

3) Clamping the flows: Flows near the root state are exponentially large, which can lead to the floating point overflow when calculating the loss after episodes of training. To prevent the overflow, we clamp the logarithms of the flows, and bias the neural network outputs instead of the rewards. Still, this can affect the learning process after where can be deemed convergent, which is further discussed in Sec. V-A.

4) Encouraging exploration: To encourage exploration, the sampler has a probability of 0.05 to sample a random action per step during training. We also sampled 60 random trajectories with lengths varying from 1 to 60 before training, so that the model can have a basic exploration of the trajectory length.

V. Discussion

A. Sample efficiency

Fig. 5 shows how scores of the sampled terrains changed during training of the flow approximation models. The both models were well trained within hundreds of samples, exhibiting high data efficiency.

Yet, their performance both went bad after certain iterations, which according to our analyses were due to the clamped flows. The flows from the root state are extremely large and can hardly avoid overflow if we do not use very-high-precision floating-point numbers. After the flows from the root are clamped, further flows get mismatched, which can mislead the optimization. That said, if we do not clamp the flows, the loss will just go to "nan".

B. Distributions of high-quality trajectories

Fig. 6 and Fig. 7 show the length and score (deviation) distributions of different samplers for the "medium" and "hard" tasks respectively. Different trajectory lengths for the random samplers were tested because it is hard to determine an optimal one.

Besides the outstanding performance, our learned samplers generate trajectories of varying lengths, which also reflects the sample diversity that can be further seen in the dataset.

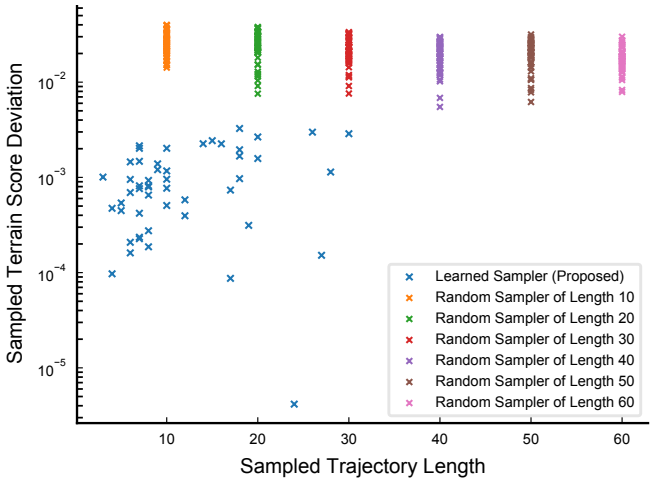

Fig. 6. Comparison of the score deviation from 0.055 for different samplers in the "medium" task, 50 points per sampler. Our method can reach 10^{-3}-level deviation, which is comparable to the effect of noise.

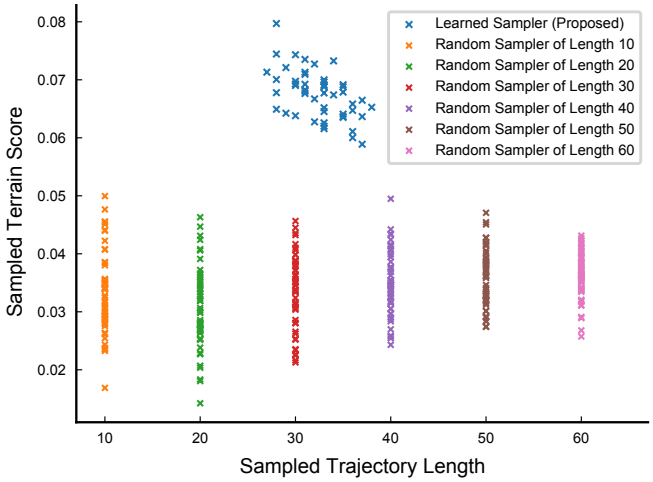

Fig. 7. Comparison of the score distributions for different samplers in the "hard" task, 50 points per sampler. Our method can reach much higher scores than the random samplers.

VI. Conclusion and Future Work

In this paper, we prototyped the generation of a terrain dataset that can be used as a terrain-robustness benchmark for legged locomotion. With techniques of terrain authoring and GFlowNets, we managed to generate diverse high-quality terrains for the "medium" and "hard" tasks. Extensive evaluation for different robot platforms and policies will be done and posted on the codebase website.

Our future works will focus on three aspects:

1) How our dataset can be generated in a goal-conditioned way, i.e., to train samplers for different difficulty levels simultaneously;
2) How our dataset can be used in a data-driven way to train terrain-robust locomotion policies, e.g., fall recovery from all kinds of terrains;
3) How our method can be generalized to the generation of non-rigid terrains.

In brief, we hope the extensions of our proposed method can change the way researchers train and evaluate policies.

REFERENCES

[1] P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5761–5768.

[2] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, 2020.

[3] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[4] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," *IEEE Transactions on Robotics*, 2022.

[5] F. Jenelten, T. Miki, A. E. Vijayan, M. Bjelonic, and M. Hutter, "Perceptive locomotion in rough terrain–online foothold optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5370–5376, 2020.

[6] F. Jenelten, R. Grandia, F. Farshidian, and M. Hutter, "Tamols: Terrain-aware motion optimization for legged systems," *IEEE Transactions on Robotics*, pp. 1–19, 2022.

[7] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[8] L. Polidori, J. Chorowicz, R. Guillande, *et al.*, "Description of terrain as a fractal surface, and application to digital elevation model quality assessment," *Photogrammetric Engineering and Remote Sensing*, vol. 57, no. 10, pp. 1329–1332, 1991.

[9] N. Shaker, J. Togelius, and M. J. Nelson, "Fractals, noise and agents with applications to landscapes," in *Procedural Content Generation in Games*. Springer, 2016, pp. 57–72.

[10] É. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez, "Interactive example-based terrain authoring with conditional generative adversarial networks," *Acm Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–13, 2017.

[11] D. Gesch, M. Oimoen, S. Greenlee, C. Nelson, M. Steuck, and D. Tyler, "The national elevation dataset," *Photogrammetric engineering and remote sensing*, vol. 68, no. 1, pp. 5–32, 2002.

[12] Y. Bengio, T. Deleu, E. J. Hu, S. Lahlou, M. Tiwari, and E. Bengio, "Gflownet foundations," *CoRR*, vol. abs/2111.09266, 2021. [Online]. Available: https://arxiv.org/abs/2111.09266

[13] E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio, "Flow network based generative models for non-iterative diverse candidate generation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 381–27 394, 2021.

[14] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 5973–5979.

[15] O. A. V. Magaña, V. Barasuol, M. Camurri, L. Franceschi, M. Focchi, M. Pontil, D. G. Caldwell, and C. Semini, "Fast and continuous foothold adaptation for dynamic locomotion through cnns," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2140–2147, 2019.

[16] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

[17] F. Acero, K. Yuan, and Z. Li, "Learning perceptual locomotion on uneven terrains using sparse visual observations," *IEEE Robotics and Automation Letters*, 2022.

[18] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions," in *Computer Graphics Forum*, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2579–2600.

[19] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Communications of the ACM*, vol. 25, no. 6, pp. 371–384, 1982.

[20] B. B. Mandelbrot and J. W. Van Ness, "Fractional brownian motions, fractional noises and applications," *SIAM review*, vol. 10, no. 4, pp. 422–437, 1968.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[22] A. Wulff-Jensen, N. N. Rant, T. N. Møller, and J. A. Billeskov, "Deep convolutional generative adversarial network for procedural 3d landscape generation based on dem," in *Interactivity, Game Creation, Design, Learning, and Innovation*. Springer, 2017, pp. 85–94.

[23] R. J. Spick, P. Cowling, and J. A. Walker, "Procedural generation using spatial gans for region-specific learning of elevation data," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[24] B. Settles, "Active learning literature survey," 2009.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[26] D. Van Ravenzwaaij, P. Cassey, and S. D. Brown, "A simple introduction to markov chain monte–carlo sampling," *Psychonomic bulletin & review*, vol. 25, no. 1, pp. 143–154, 2018.

[27] W. Grathwohl, K. Swersky, M. Hashemi, D. Duvenaud, and C. Maddison, "Oops i took a gradient: Scalable sampling for discrete distributions," in *International Conference on Machine Learning*. PMLR, 2021, pp. 3831–3841.

[28] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[29] nanoxas, "Terrain generation using pix2pix," https://github.com/nanoxas/sketch-to-terrain/releases/tag/terrain-generation, 2020.

[30] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 38–44.

[31] Unitree Robotics, "Unitree robotics a1." [Online]. Available: https://www.unitree.com/products/a1/

[32] M. Bartos, "pysheds: simple and fast watershed delineation in python," 2020. [Online]. Available: https://github.com/mdbartos/pysheds

[33] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 000–16 009.

[34] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[36] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.