Mesh Based Analysis of Low Fractal Dimension Reinforcement Learning Policies

Sean Gillen[1] and Katie Byl[2]

Abstract— In previous work, using a process we call meshing, the reachable state spaces for various continuous and hybrid systems were approximated as a discrete set of states which can then be synthesized into a Markov chain. One of the applications for this approach has been to analyze locomotion policies obtained by reinforcement learning, in a step towards making empirical guarantees about the stability properties of the resulting system. In a separate line of research, we introduced a modified reward function for on-policy reinforcement learning algorithms that utilizes a "fractal dimension" of rollout trajectories. This reward was shown to encourage policies that induce individual trajectories which can be more compactly represented as a discrete mesh. In this work we combine these two threads of research by building meshes of the reachable state space of a system subject to disturbances and controlled by policies obtained with the modified reward. Our analysis shows that the modified policies do produce much smaller reachable meshes. This shows that agents trained with the fractal dimension reward transfer their desirable quality of having a more compact state space to a setting with external disturbances. The results also suggest that the previous work using mesh based tools to analyze RL policies may be extended to higher dimensional systems or to higher resolution meshes than would have otherwise been possible.

I. INTRODUCTION

Legged robots have clear potential to play an important role in our society in the near future. Examples include contact-free delivery during a pandemic, emergency work after an environmental disaster, or as a logistical tool for the military. Legged robots simply expand the reach of robotics when compared to wheeled systems. However, compared to wheeled systems, designing control policies for legged systems is a much more complex task, especially in the presence of disturbances, noise, and unstructured environments.

The increasing availability of massive quantities of computation has led to a resurgence of reinforcement learning (RL) in recent years. RL provides a promising approach for complex, under-actuated, hybrid control problems, such as those involved in designing control for legged locomotion. Recent examples in the context of robotics include controlling a 47 DOF humanoid to navigate a variety of obstacles [6], dexterously manipulating objects with a 24 DOF robotic hand [11], and allowing a physical quadruped robot to run [7], and recover from falls [8].

Despite the obvious promise of RL approaches, several problems need to be resolved before these systems are ready for real world applications. One of the biggest problems is that the resulting policy is typically a complete black box,

Sean Gillen and Katie Byl are with the Electrical and Computer Engineering Department at the University of California, Santa Barbara CA 93106 `sgillen@ucsb.edu, katiebyl@ucsb.edu`.

there are no good ways to make theoretical, or even empirical guarantees about the resulting policies. Prior work has used so called mesh based techniques to this end [13]. Broadly these techniques take a continuous system and approximate it with a discrete set of states. This allows us to model the system as a Markov chain, these systems are arguably easier to reason about, and it opens up a new box of tools we can bring to bear on the problem. For example we can use value iteration to switch between several controllers to improve the robustness [12] or agility [3] of the system. We can also perform eigen-analysis on the Markov chain's transition matrix, which provides us insights on the stability of the system [4]. These techniques could both be used for policy refinement, and/or for verification and analysis of existing polices.

However, these methods suffer from the "curse of dimensionality", because the number of possible states in our mesh grows exponentially with the degrees of freedom in our system. That is, if we make a change to the volume of continuous space represented by each discrete state, the number of states in our new mesh will grow exponentially with respect to the change in discrete state size. However for virtually all plausible walking controllers, the reachable state space is a small fraction of the total state space. Furthermore, although the scaling for meshes of only the reachable state space also scale exponentially as we increase the mesh resolution, the rate of scaling is also typically much smaller. The scaling factor for the reachable mesh can be seen as a **fractal dimension**, which is elaborated upon in section II-A.

In previous work [5], we introduced a modified reward function for on policy reinforcement learning algorithms. The reward explicitly encourages policicxies which induce trajectories which have a smaller fractal dimension. It's worth noting that although each individual trajectory was encouraged to have a smaller fractal dimension, that this does not obviously extend to properties of the entire reachable state space for the system, which is what was used for the previous mesh based analysis of RL policies.

In this work we take the next step and construct reachable state space meshes of agents trained with and without our modified reward. We show that these modified policies result both in much smaller reachable meshes for a given mesh size, and in smaller fractal dimensions for the reachable state space. We also show that the increased robustness seen in [5] also seem to carry over into this setting.

(a) Scaling in different dimensions

(b) A non uniform mesh
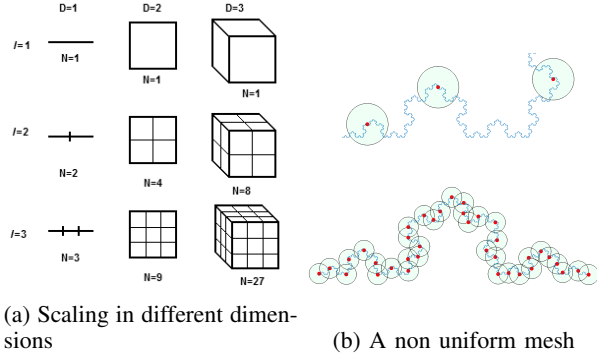
Fig. 1: Image credit: [1], [12]

II. BACKGROUND

A. Meshing and Fractal Dimensions

Let's say we have a continuous set S that we want to approximate by selecting a discrete set M composed of regions in S. We will call this set M a mesh of our space. Figure 1(a) shows some examples of this: a line is broken into segments, a square into grid spaces, and so on. The question is: as we increase the resolution of these regions, how many more regions N do we need? Again, figure 1(a) shows us some very simple examples. For a D dimensional system, if we go from regions of size d to d/k, then we would expect the number of mesh points to scale as $N \propto k^D$. But not all systems will scale like this, as 1(b) illustrates. Figure 1(b) is an example of a curve embedded in a two dimensional space. The question of how many mesh points are required must be answered empirically. Going backwards, we can use this relationship to assign a notion of "dimension" to the curve.

$$D_f = -\lim_{k \to 0} \frac{\log N(k)}{\log k} \tag{1}$$

This quantity is known as the Minkowski–Bouligand dimension, also called the box counting dimension. This dimension need not be an integer, hence the name "fractional" or "fractal" dimension. This is one of many measures of fractional dimensionality that that emerged from the study of fractal geometry. Although these measures were invented to study fractals, they can still be usefully applied to non-fractal sets. For non fractal sets, we use the slope of the log-log relation of mesh sizes to d to calculate this, rather than taking a limit.

B. Box Meshing

In this work, we identify any state s with a key obtained by:

$$s_k = \frac{s - \mu_s}{\sigma_s}$$
$$\text{key} = \text{round}(\frac{s_k}{d_{thr}})d_{thr} \tag{2}$$

Where μ_s and σ_s are the mean and standard deviation of all the states seen by the policy of interest during training.

The round function here performs an element-wise rounding to the nearest integer. We can then use these keys to store mesh points in a hash table. Using this data structure, we can store the mesh compactly, only keeping the points we come across, and lookups are done in constant time. The parameter d_{thr} is also called the **box size**. Geometrically we can think of this operation as dividing the state space into a uniform grid of hypercubes, each with a side length of d_{thr}

C. Reinforcement Learning

The goal of reinforcement learning is to train an agent acting in an environment to maximize some reward function. At every timestep $t \in \mathbb{Z}$, the agent receives the current state $s_t \in R^n$, uses that to compute an action $a_t \in \mathbb{R}^b$, and receives the next state s_{t+1}, which is used to calculate a reward $r : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$. The objective is to find a policy $\pi_\theta : \mathbb{R}^n \to \mathbb{R}^m$ that satisfies:

$$\arg\max_\theta \mathbb{E}_\eta \left[\sum_{t=0}^{T} r(s_t, a_t, s_{t+1}) \right] \tag{3}$$

Where $\theta \in \mathbb{R}^d$ is a set that parameterizes the policy, and η is a parameter representing the randomness in the environment. This includes the random initial conditions for episodes.

In [5], we introduced a modified reward function:

$$\arg\max_\theta \mathbb{E}_\eta \left[\frac{1}{D_m(s)} \sum_{t=0}^{T} r(s_t, a_t, s_{t+1}) \right] \tag{4}$$

where D_m is the "lower mesh dimension" explained in detail in [5], which is an estimate of equation 1 dimension for our non fractal set.

D. Environment

Our model system is openAI gym's hopper-v2 environment introduced in [2]. This environment is part of a popular and standardized set of benchmarking tasks for reinforcement learning algorithms. The system is a 4 link, 6 DOF hopper constrained to travel in the XZ plane, seen in figure 2. The observation space for the agent has 11 states, the position in the direction of motion is held out, since we seek a policy that is invariant to forward progress. At every timestep the agent selects 3 torques to output to each joint. The reward function for this environment is simply forward velocity minus a small penalty to actions. Successful controllers in this environment must execute a dynamic hopping motion to move robot along the x axis as quickly as possible. This is clearly a toy problem, but it captures many of the challenges of legged locomotion. The system is highly non-linear, under-actuated, and must interact with friction and ground contacts to maximize it's reward.

III. MESHING

In [5] we used meshes of individual trajectories to calculate fractional dimensions. However the previous work that has used meshing for analysis of RL policies [13] instead examines meshes of the reachable state space of a system. In
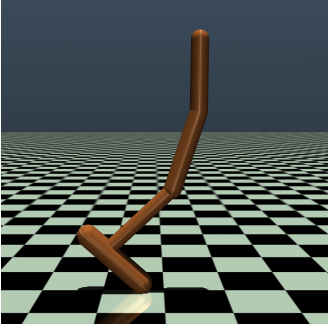
Fig. 2: A render of the hopper system studied in this work

Fig. 3: First 300 time steps of a typical hopper trajectory. Vertical red lines indicate the location of Poincare snapshots

particular that work examines the reachable state space with a fixed control policy subject to a given set of disturbances. We will now outline the process for this style of meshing.

We are interested in the set of states that our system can transition to with a fixed policy and a given set of push disturbances. We first introduce a failure state to the mesh. The failure state is assumed to be absorbing, once the robot has falls it is assumed to stay that way. For our hopper, any state where the COM falls below .7m is considered to have failed, which seems to work well in practice. This is also the failure condition for the default environment during training, and therefore we have no reason to expect the robot could recover from this state.

In addition to the reachable set of states, we want to construct a state to state transition map. That is, for a given initial state, we wish to know which state we transition to for every disturbance in our disturbance set. It's worth emphasizing that this map is completely deterministic.

To make this concrete, recall that we manifest our mesh as a hash table. The key for any given state is obtained by 2. When we insert a new key into our hash table, the value we place is a pair with a unique state ID (which is simply the number of keys in the table at the time of insertion), and an initially empty list of all mesh states which are reachable after one step from the key state. This data structure will provide both the reachable set, and the transition mapping.

For the hopper in particular, the system transitions from its initial standing position to a stable long term hopping gait. After letting the system enter its gait, we start detecting the Poincare states by selecting the state corresponding to the peak of the base link's height in every ballistic phase. Figure 3 shows an example of this. These states are then collected as the initial states to seed the mesh with. Throughout this paper, we seed the mesh with trajectories from 10 initial conditions.

For each snapshot, we initialize the system in the snapshotted state. For each disturbance in our fixed disturbance set, we simulate the system forward subject to that disturbance. If the system does not fail, then the next Poincare snapshot is captured, this state is then checked for membership in our mesh. If the new state is already in our mesh, then we simply append the new state to the list of states that the initial state can transition to. If the new state is not already
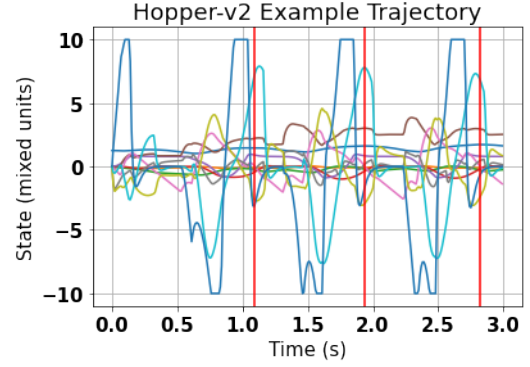
in our mesh, then we expand our mesh to include the new state, and append this new state to the transition list of the initial state. If the system does fail, then we simply append the failure state to the transition list of the initial state, and no new state is added to the mesh.

For every new state added to the mesh, we repeat this process until every state has been explored. Algorithm 1 details this process in psuedo code.

Algorithm 1: createMesh

Input: Initial states S_i, Disturbance set D
Output: Mesh M.
$Q \leftarrow S_i$ (excluding the failure state)
while *Q not empty* **do**
    pop q from Q
    for $d \in D$ **do**
        Initialize system in state q
        Run system for one step subject to
         disturbance d
        Obtain final state x
        if $x \notin M$ **then**
            M[x] = List()
            Push x onto Q
        end
        Append x to M[q]
    end
end
Return: M

A. Stochastic Transition Matrix

The stochastic transition matrix $\mathbf{T}$ is defined as follows:

$$\mathbf{T}_{ij} = \Pr(id[n+1] = j \mid id[n] = i) \qquad (5)$$

where $id[n]$ is the index in our mesh data structure of the state at step n. After constructing a mesh using algorithm 1, it is straightforward to create the stochastic transition matrix by iterating through every transition list in our mesh.

B. Mean First Passage Time

We wish to use our mesh based methods to quantify the stability of our system. To do do this we estimate the average number of steps the agent will take before falling, subject to a given distribution of disturbances. To do this we will use the so called Mean First Passage Time (MFPT) which in this case will describe expected number of footsteps, rather than the number of timesteps to failure. First recall that our assumption is that our failure state is an absorbing state in our Markov chain approximation, and this implies that the largest eigenvalue of T will always be $\lambda_1 = 1$. In [4] Byl showed that when the second largest eigen value λ_2 is close to unity, the MFPT is approximately equal to:

$$MFPT \approx \frac{1}{(1 - \lambda_2)} \qquad (6)$$

IV. TRAINING

In [9] Mania et al introduce Augmented Random Search (ARS) which proved to be efficient and effective on the locomotion tasks. Rather than a neural network, ARS used static linear policies, and compared to most modern reinforcement learning, the algorithm is very straightforward. The algorithm is known to have high variance; not all seeds obtain high rewards, but to our knowledge their work in many ways represents the state of the art on the Mujoco benchmarks. Mania et al introduce several small modifications of the algorithm in their paper, our implementation corresponds to the version they call ARS-V2t, hyper parameters are provided in the appendix.

The training process is done in episodes, each episode corresponds to 1000 policy evaluations played out in the simulator. At the start of each episode, the system is initialized in a nominal initial condition offset by a small amount of noise added to each state. During each episode we fix a static policy to let the the system evolve under, we collect the observed state, the resulting action, and the resulting reward at each timestep. This information is then used to update the policy for the next episode.

We compare four different sets of agents trained in different conditions, for each training condition we use training runs across 10 different random seeds. As mentioned ARS is a very high variance algorithm, so a common practice is to run many seeds in parallel and choose the highest performing one. The standard environment has two sources of randomness which are set by the random seed. The first is a small amount of noise added to a the nominal initial condition at the beginning of each episode. The second is noise added to the policy parameters as part of the normal ARS training procedure. Using ARS in the unmodified Hopper-v2 environment will be called the **standard** training procedure. In addition to this, we have a second set of agents which are initialized with the standard training, and then trained for another 250 epochs with the fractal reward function used in equation 4, these are called the **fractal agents**. Using the standard training agents as the initial policies for the fractal

reward was also used in [5], please see that manuscript for more details.

In addition to standard training, we repeat this standard / fractal setup but with the addition of a small amount of zero mean Gaussian noise added to both the states and actions at training time. For brevity we will call these the **Standard noise** and **Fractal noise** scenarios. Hyper parameters for ARS and noise values are reported in the appendix.

V. RESULTS

A. Mesh Sizes Across All Seeds

First we wish to compare the reachable state space mesh sizes obtained for these four different training regiments. For this we assume a disturbance profile consisting of 25 pushes equally spaced between -15 and 15 Newtons, applied for .01 seconds along the x axis at the apex of each jump. The goal for this particular exercise is to get an idea of the relative mesh sizes among the different agents across box sizes. Table I shows these results. We can see that across all box sizes, that adding noise at run time decreases the mesh sizes slightly, and that adding the fractal reward training decreases the mesh size even further. The combination of adding noise and the fractal reward seems to perform best at reducing the mesh size.

Training	$d_{thr} = .4$	$d_{thr} = .3$	$d_{thr} = .2$	$d_{thr} = .1$
Standard	64.9	129.0	289.2	2975.2
Standard Noise	40.7	73.3	231.6	2133.3
Fractal	26.0	41.8	67.7	684.4
Fractal Noise	15.1	24.6	45.1	297.2

TABLE I: Mesh sizes across all seeds for a disturbance profile of 25 pushes. All values are the average mesh size across 10 agents trained with different seeds

B. Larger Meshes

With the general trend established, we now take the best performing seed from the noisy training for further study. We chose the seed that had the smallest mesh size from both the standard noise and fractal noise agents.

For this next experiment, we consider a richer distribution of 100 randomly generated push disturbances. These disturbances have a magnitude drawn from a uniform distribution between 5-15 newtons. This force is applied in the xz plane with an angle drawn from a uniform distribution between 0 and 2π. The number of forces was chosen by increasing the number of forces sampled until the mesh sizes between two random sets did not change. The magnitude of the pushes was chosen arbitrarily, in principle one can use these methods for any distribution of disturbance they expect their robot to see during operation.

We then construct meshes for different box sizes. For each agent we construct 10 meshes. We vary the box size between .1 and .01 for the fractal noise agent. For the standard noise agent we instead vary the box size between .1 and .02

because the mesh sizes for the standard agent were proving to be too large at the smaller box sizes. Figure 4 shows the comparison, We can see clearly that at the very least, the exponential blowup in mesh size starts at much more accurate mesh resolutions for the fractal agent.
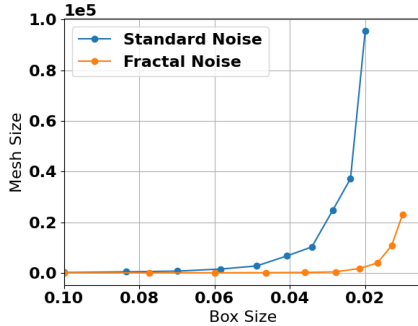

Fig. 4: Mesh sizes for the top performing standard noise and fractal noise agents

We are also interested in the exponential scaling factor in the mesh size as the box gets smaller, which is captured by the fractal dimension discused in section II-A. As mentioned before, in previous work our modified reward signal resulted in agents with a smaller fractal dimension with respect to individual trajectories. We now ask if this carries over to meshes of the reachable state space obtained by the procedure from algorithm 1. Table II shows the results, we can see that indeed, the fractal training does seem to reduce the mesh dimensionality for the reachable state space meshes.

Training	Trajectory Mesh Dim.	Reachable Mesh Dim.
Standard Noise	1.38	3.83
Fractal Noise	1.16	3.16

TABLE II: Mesh dimensions for the best performing seed from the standard with noise training, and the fractal with noise training, given the same disturbance profile of 100 pushes

C. Validating the Mean First Passage Time

We emphasize that the reward function for the hopper environment is simply to move forward with the highest velocity possible, no attempts were made to make the system robust to disturbances. Perhaps because of this, the mean first passage time for these systems are relatively small, on the order of 100 foot steps. For this small number of steps, we can validate the mean first passage time with Monte Carlo trials. It's worth noting that the eigen estimate of the mean first passage time is really much more valuable for more robust systems. This is because this estimate grows more accurate as the system grows more stable, and because the cost of calculating the MFPT with Monte Carlo trials grows much more expensive for more stable systems. In previous

works [10] it was used to quantify robustness for systems with a MFPT as high as 10^{15}.

To do this, we compare the mean first passage time as estimated by equation 6 to the value computed by looking at many monte carlo rollouts. For the rollouts we apply a random action drawn from the same distribution described above. Instead of sampling 100 pushes though we sample a new push every time we need a new disturbance. During the rollouts we still apply the push at the apex height of the ballistic phase.

Figure 5 shows the convergence of the MFPT as we expand the size of the mesh, and compares it to the mean steps to failure obtained with monte carlo trials. We can see that it does look like the MFPT is converging to the monte carlo result. Although at the largest mesh we tried, the eigen analysis gives an estimate of 110.2 steps to failure, while the monte carlo trials tell us that an average of 85 steps are taken on average before failure. It's worth noting that the distribution of failure times has a large variance with a standard deviation of 80 steps.
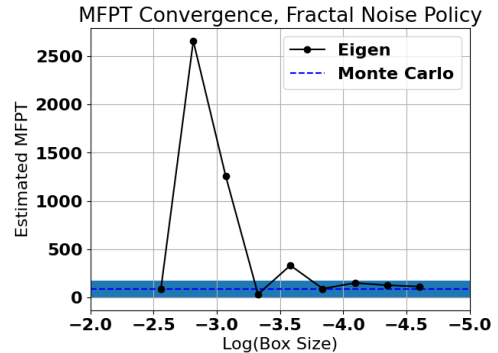

Fig. 5: Estimated mean first passage time computed from 6 compared to a monte carlo estimate. The blue dashed line and shaded region are the mean and standard deviation of the steps to failure for 2500 monte carlo rollouts

D. High Resolution Mesh

We now use the fractal agent and construct an even more accurate mesh. Figure 6 show the sparsity pattern for the state transition matrix for the fractal noise agent with a box size of .005. Recall that in the process for creating the mesh, we start with a small number initial seed states. After that every new state that we add is added in order we find them to the mesh. So if we are expanding state # 2, and there are current 100 states in the mesh, if we transition to an unseen state, that state will be labeled # 101. So although it may seem like it is not possible for states in the top right quadrant to visit states later in the mesh, this is really an artifact of how we construct our mesh and label our points.

We note that there are a smaller set of states that make up most of the transitions. In fact we can see from figure 7 that 20% of the states in our mesh account for about 90% of all transitions seen during the mesh construction. This supports our hypothesis that learning algorithms must be finding low dimensional manifolds to learn on.
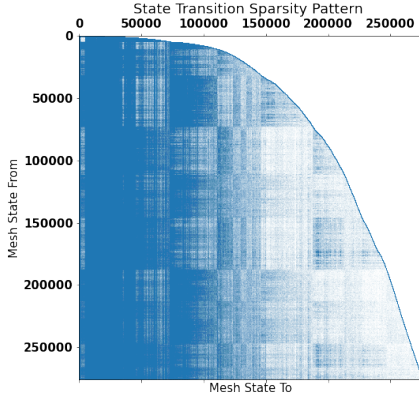
Fig. 6: Sparsity pattern for the top performing fractal noise agent. All non zero values are shown with equal size and coloration. Recall that each entry in T_{ij} tell us the probability of transitioning to row j after one step if we start in state i. Put another way, every row must sum to one, but the columns may not.
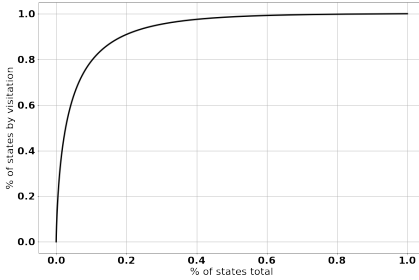

Fig. 7: Cumulative sum of probability mass excluding the failure state. We take the sum of each column of T, and sort it in descending order, then report the cumulative sum of probability. Each point on the curve tells us that x% of states make up y% of all state transitions.

One of the advantages of having a discrete set of states is that it opens up new tools and visualizations for our data. Figure 8 shows a projection of our mesh states on the top 3 principle components. We note that these three states account for $> 97\%$ of the variance, we also note that our analysis reveals that states in red account are where 99% of all failures occur. The visualization reveals that at least in PCA space, all the trouble states are clustered in one spot. It is not hard to imagine future work were we include a policy refinement step that attempts to avoid these states, or if we were designing a real robot this may give us insights into design changes that could be made.

VI. CONCLUSIONS

In this work, we apply previously developed tools that create discrete meshes for the reachable state space of a system. These tools were applied to policies obtained by a modified reinforcement learning reward function which was previously shown to encourage small mesh dimensions for individual trajectories not subject to any disturbances. We showed that these modified policies have a smaller average
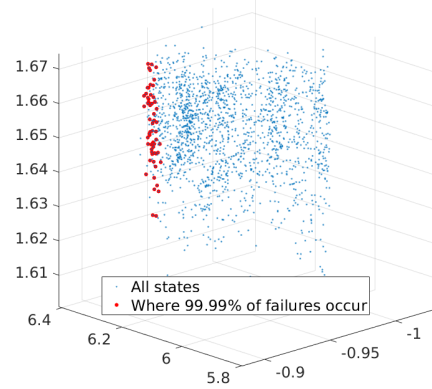

Fig. 8: View of the first 3 principle components of the mesh for a fractal noise policy

reachable mesh size across all random seeds for coarse meshes and a small number of disturbances. We then showed a clear difference in mesh sizes and mesh dimensions for the top performing seeds on a richer set of disturbances and finer mesh sizes. We also validated our use of the MFPT as a tool by comparing it to monte carlo trials. Finally, we constructed a high fidelity mesh at a resolution that would not have been feasible with standard ARS policies. In addition, we created visualizations with this mesh that revealed insights about the contracting nature of the policy, and which point to future applications of this approach. Taken together, these results show two things. First, it further validates the utility of the fractal dimension reward, which we have shown transfers it's desirable quality of having a more compact state space to a setting with external disturbances. These results are also a credit to the mesh based tools, because it shows that the fractal training can be used to extend the reach of these tools to higher dimensional systems or to higher resolution meshes than would have otherwise been possible.

APPENDIX

Hyper Parameters

ARS: (from [9]) $\alpha = .02$, $\sigma = .025$, $N = 50$, $b = 20$.
MeshDim: (from [5]) f = 1.5, d_0 = 1e-2

Noise During Training

Zero mean Gaussian noise with std = .01 added to policy actions before being passed to the environment, for reference all actions from the policy are between -1 and 1. Zero mean Gaussian noise with std =.001 added to observations before being passed to the policy.

References

[1] Brendan Ryan / Public domain. *Fractal Dimension Example*. 2020.

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[3] Katie Byl, Tom Strizic, and Jason Pusey. "Mesh-based switching control for robust and agile dynamic gaits". In: *Proceedings of the American Control Conference* (2017), pp. 5449–5455. ISSN: 07431619. DOI: `10.23919/ACC.2017.7963802`.

[4] Katie Byl and Russ Tedrake. "Metastable walking machines". In: *International Journal of Robotics Research* 28.8 (2009), pp. 1040–1064. ISSN: 02783649. DOI: `10.1177/0278364909340446`.

[5] Sean Gillen and Katie Byl. "Using Reinforcement Learning to Shrink Fractional Dimensions For Fun and Profit". In: *Conference Of Robotic Learning* (2020).

[6] Nicolas Heess et al. "Emergence of Locomotion Behaviours in Rich Environments". en. In: *arXiv:1707.02286 [cs]* (July 2017). arXiv: 1707.02286. URL: `http://arxiv.org/abs/1707.02286` (visited on 03/01/2019).

[7] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". en. In: *Science Robotics* 4.26 (Jan. 2019), eaau5872. ISSN: 2470-9476. DOI: `10.1126/scirobotics.aau5872`. URL: `http://robotics.sciencemag.org/lookup/doi/10.1126/scirobotics.aau5872` (visited on 03/01/2019).

[8] Joonho Lee, Jemin Hwangbo, and Marco Hutter. "Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning". en. In: *arXiv:1901.07517 [cs]* (Jan. 2019). arXiv: 1901.07517. URL: `http://arxiv.org/abs/1901.07517` (visited on 08/09/2019).

[9] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple random search of static linear policies is competitive for reinforcement learning". In: *Advances in Neural Information Processing Systems* 2018-December.NeurIPS (2018), pp. 1800–1809. ISSN: 10495258.

[10] Cenk Oguz Saglam and Katie Byl. "Robust Policies via Meshing for Metastable Rough Terrain Walking". In: *Robotics Science and Systems*. 2015. DOI: `10.15607/rss.2014.x.049`.

[11] OpenAI et al. "Learning Dexterous In-Hand Manipulation". en. In: *arXiv:1808.00177 [cs, stat]* (Aug. 2018f). arXiv: 1808.00177. URL: `http://arxiv.org/abs/1808.00177` (visited on 03/01/2019).

[12] Nihar Talele and Katie Byl. "Mesh-based methods for quantifying and improving robustness of a planar biped model to random push disturbances". In: *Proceedings of the American Control Conference* 2019-July (2019), pp. 1860–1866. ISSN: 07431619. DOI: `10.23919/acc.2019.8815226`.

[13] Nihar Talele and Katie Byl. "Mesh-based Tools to Analyze Deep Reinforcement Learning Policies for Underactuated Biped Locomotion". In: (2019). arXiv: `1903.12311`. URL: `http://arxiv.org/abs/1903.12311`.