A Recurrent Differentiable Engine for Modeling Tensegrity Robots
Trainable with Low-Frequency Data

Kun Wang, Mridul Aanjaneya and Kostas Bekris

Abstract—Tensegrity robots, composed of rigid rods and flexible cables, are difficult to accurately model and control given the presence of complex dynamics and high number of DoFs. Differentiable physics engines have been recently proposed as a data-driven approach for model identification of such complex robotic systems. These engines are often executed at a high-frequency to achieve accurate simulation. Ground truth trajectories for training differentiable engines, however, are not typically available at such high frequencies due to limitations of real-world sensors. The present work focuses on this frequency mismatch, which impacts the modeling accuracy. We proposed a recurrent structure for a differentiable physics engine of tensegrity robots, which can be trained effectively even with low-frequency trajectories. To train this new recurrent engine in a robust way, this work introduces relative to prior work: (i) a new implicit integration scheme, (ii) a progressive training pipeline, and (iii) a differentiable collision checker. A model of NASA’s icosahedron SUPERballBot on MuJoCo is used as the ground truth system to collect training data. Simulated experiments show that once the recurrent differentiable engine has been trained given the low-frequency trajectories from MuJoCo, it is able to match the behavior of MuJoCo’s system. The criterion for success is whether a locomotion strategy learned using the differentiable engine can be transferred back to the ground-truth system and result in a similar motion. Notably, the amount of ground truth data needed to train the differentiable engine, such that the policy is transferable to the ground truth system, is 1% of the data needed to train the policy directly on the ground-truth system.

I. INTRODUCTION

Tensegrity robots are compliant systems composed of rigid (rods) and flexible elements (cables) connected to form a lightweight deformable structure. Their adaptive and safe features motivate applications, such as manipulation [1], locomotion [2], morphing airfoil [3] and spacecraft lander design [4]. At the same time, they are difficult to accurately model and control due to the high number of DoFs and complex dynamics. This work is motivated by these exciting robotic platforms and aims to provide scalable solutions for modeling them in a data-driven, but explainable, manner.

Learning effective control policies for tensegrity robots is challenging with model-free solutions since they require a large amount of training data and collecting trajectories from tensegrities is time-consuming, cumbersome and expensive. Thus, a better alternative is to tune a dynamical model, or a simulator, of the system to minimize the difference between trajectories predicted by the model relative to those

The authors are with the Department of Computer Science, Rutgers University, NJ 08901, USA. Email: kun.wang2012, mridul.aanjaneya, kostas.bekris@rutgers.edu.

This work has been partially supported by NSF award IIS 1956027, IIS-2132972, CCF-2110861 and the Rutgers University startup grant.
collisions (see Fig. 1(top)). The previous differential engine for tensegrities [12] used a feed-forward architecture and can only be trained from trajectories sampled at high frequencies, similar to that of the engine’s frequency, i.e., in the order of 1000Hz, which beyonds what most sensors can achieve.

Motivated by this frequency gap, this work proposes a recurrent architecture for a differentiable physics engine targeted for tensegrity robots, which remains explainable and can be trained effectively even with low-frequency ground-truth data. Beyond the recurrent nature of the engine, this work introduces a new implicit integration and an adaptive semi-implicit integration process as part of the differentiable engine to increase its robustness. Furthermore, a progressive training algorithm is proposed, which avoids gradient explosion and leads to a fast and stable training process. To further accelerate the training process, a fast differentiable collision checker has been implemented as part of the proposed engine. The authors will release the integrated solution as an open-source software package upon acceptance.

To demonstrate the benefits of the proposed engine, sim2sim experiments are performed as a step towards the application on real robots. In particular, a model of NASA’s icosahedron SUPERballBot on MuJoCo is used as the ground truth system to collect training data. The experiments show that the recurrent differentiable engine can match the output of the ground truth system even if it is provided low-frequency data about the MuJoCo trajectories. This paper demonstrates that a control strategy trained on the proposed recurrent engine after identification can be transferred back to the ground truth system and results in a similar motion. Notably, the amount of ground truth data needed to identify the recurrent engine is 100× less than the data needed to train the policy directly on the ground-truth system.

II. RELATED WORK

Differentiable physics is an active area of research. A hybrid engine with 2-way coupling for both rigid and soft bodies with smoothed frictional contact was proposed in [13]. A sphere and plane-based differentiable engine has been used for training a controller recurrently but without system identification [14]. A differentiable programming language was proposed in [15], but requires users to implement the physics engine themselves. To reduce data requirements, analytical differentiable physics engines based on linear complementarity (LCP) [8] or quadratic programming (QP) [9] have been proposed. However, they require densely sampled simulation trajectories. Interactions that can be efficiently simulated and differentiation have been studied [16], but interactions represented by neural networks [6], [17], [7] have been shown to perform poorly for tensegrity robots [18]. The Constrained Recursive Newton-Euler Algorithm (RNEA) [10], [11] uses differentiable physics engines for rigid-body chain manipulators [19], [6]. Differentiable dynamical constraints can also be applied to path optimization [20], training by image supervision [21], [22], soft object manipulation [23], soft robot control [24], [25] and quantum molecular control [26]. A differentiable engine specifically designed for tensegrity robots [12] provides analytically explainable models for both the robot and the ground, but still requires densely sampled ground-truth trajectories for system identification.

Recurrent structures use previous outputs as inputs while maintaining hidden states. This feature allows them to be trained on sequences to reflect the underlying temporal dynamics. Thus, recurrent structures have become popular solutions for video prediction [27], embedded dynamics [28], trajectory forecasting [29] and translation [30], etc.

Prior work on tensegrity locomotion [31], [32], [33] has achieved complex behaviors, on uneven terrain, using the NTRT simulator [34], which was manually tuned to match a real platform [35], [36]. Many of these approaches use reinforcement learning to learn policies given sparse inputs, which can be provided by onboard sensors [32] and aim to address the data requirements of RL approach [33], including by training in simulation. But simulated locomotion is hard to replicate on a real platform, even after hand-tuning, which emphasizes the importance of learning a transferable policy that is the focus of this work. Domain randomization [37], [38], [39] is a possible way to close the sim2real gap. Previous domain randomization efforts [37] often assume the system parameters are within a Gaussian distribution and use a non-differentiable physics engine. Nevertheless, the parameter distribution here is unknown and the range varies a lot. The current work is a step in mitigating the reality gap by showing sim2sim transfer with low-frequency data. The different, closed-source and unknown physical model of the MuJoCo simulator [40] is used as the ground-truth. MuJoCo does not follow the same first principles for modeling contacts as that of the proposed differentiable physics engine. This feature makes MuJoCo a good candidate for a ground-truth system.

III. PROPOSED ENGINE

At the core of the proposed approach lies a differentiable physics engine, shown in Fig. 2, which builds on top of previous work [12]. The engine brings together a series of analytical models based on first principles, which are linear and differentiable. The input to the engine is the current robot state $X_t$ and the instantaneous control $U_t$. The engine internally stores a representation of the robot in a static topology graph indicating the connectivity of rods and cables. The control $U_t$ is passed to a Cable Actuator module, which maps the control to desired cable rest-lengths. Together with

$$\hat{X}_{t+1}$$

Fig. 2: The physics engine takes the current robot state $X_t$ and control $U_t$ as inputs and predicts the next state $\hat{X}_{t+1}$. Compared to previous attempts [12], this work introduces recurrent training (as shown in Fig. 3), a new numerical integrator, a progressive training pipeline and a new Differentiable Collision Checker (DCC) to account for the frequency mismatch. DIG: Dynamic Interaction Graph, CRG: Collision Response Generator, CFG: Cable Force Generator, RAG: Rod Acceleration Generator.
the topological graph, the cable actuator informs the Cable Force Generator (CFG), which is responsible to predict the forces applied on the rods due to the cables given the latest robot state. In parallel and given the robot state, a Differentiable Collision Checker (DCC) detects collisions and informs a Dynamic Interaction Graph (DIG), which stores the colliding bodies (either rod-to-rod or rod-to-ground) and the corresponding contact points. This information is passed to a Collision Response Generator (CRG), which is responsible to compute the reaction forces applied to the rods. The forces and torques from the cables (as computed by CFG) and those from contacts (as computed by CRG) are forwarded to the Rod Acceleration Generator (RAG), which computes the linear and angular acceleration experienced by the rods. These accelerations and torques are integrated by an Integrator module so as to update the robot state.

Overall, these models introduce two sets of parameters: robot parameters and contact parameters, which need to be identified. The new contributions of this work include: (i) a recurrent architecture for training this engine with low frequency data; (ii) an implicit integration scheme, (iii) a progressive training pipeline with a new adaptive semi-implicit integration module, and (iv) a differentiable collision checker. These components enable stable and robust recurrent training with low-frequency training data on a high-frequency physics engine.

A. Recurrent Differentiable Physics Engine

At a high level, the proposed training pipeline calls the differentiable physics engine in a recurrent fashion through a sequence of temporal connections. With help from Back-Propagation Through Time (BPTT), the engine parameters can be identified given low-frequency sampled trajectories. Figure 3 presents the recurrent nature of the proposed architecture. The trajectory is split to data point tuples, \([X_t, U_t, X_{t+T}]\), where \(X_t\) is the system state at time \(t\) and \(U_t\) is the sequence of actions to be executed in the interval \([t, t+T]\). The control sequence \(U_t\) can be split into control signals \(U_t, U_{t+1}, \ldots, U_{t+T-1}\), which have the same high-frequency as the engine. In recurrent mode, the engine receives as inputs \((X_t, U_t)\) and generates output \(\hat{X}_{t+T}\). The loss is the mean square error (MSE) between the predicted state \(\hat{X}_{t+T}\) and the ground truth state \(X_{t+T}\). The gradients of the loss function are then back-propagated through the physics engine and used for updating the engine’s parameters.

To adapt this engine for recurrent training and achieve robustness, this work introduces an implicit integration scheme and a differentiable collision checker.

B. Implicit Integration

While implicit integration schemes are standard for simulating stiff multi-body systems, their dynamics are typically handled by explicit methods, such as semi-explicit integration [13]. Semi-explicit integration is not stable for stiff systems, such as tensegrities. Deriving an implicit system, however, for the whole tensegrity is very complex. For instance, more complex than what has been achieved using a particle-based mesh system [16]. To address this complexity, this work follows a modular approach by focusing on the basic elements of tensegrity robots, rods and springs, and derives the following \(Ax = b\) form linear system for each time step, where \(A\) is a 21x21 matrix and \(x, b\) are 21x1 vectors (see appendix\(^1\) for details):

\[
\begin{align*}
0 & 1 & 0 & -C \\
-1 & K & iB & 0 \\
-ae^T & 1 & -\Delta t & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -\Delta t & 1 \\
\end{align*}
\]

\[
B = \begin{bmatrix}
\Delta x_{t+1} & \Delta y_{t+1} & \Delta z_{t+1} \\
\Delta x_{t+2} & \Delta y_{t+2} & \Delta z_{t+2} \\
\Delta x_{t+3} & \Delta y_{t+3} & \Delta z_{t+3} \\
\end{bmatrix}, \quad C = \begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & [\omega^R] \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

where \(x^m_1\) and \(x^R_1\) are the positions of the left end-point of the spring (denoted as \(m_1\)) and the rod, \(r\) is the torque arm from \(m_1\) to rod’s center of mass, \(v^m_1\) is the linear velocity of \(m_1\), \(v^R_1\) is the rod linear velocity, \(\omega^R_1\) is the rod angular velocity, \(\Delta x\) is the spring direction, \(l^{rest}\) is the spring rest-length, \(f\) is the spring force under Hooke’s law, \(K\) is the spring stiffness, \(k\) is the spring damping coefficient, \(m\) is the rod mass, \(I^{-1}\) is the inverse of the rod inertia matrix, \([\omega^R \times]\) and \([r \times]\) are skew-symmetric matrices of \(\omega^R\) and \(r\), \(c_1 = K(x^m_{1+1} + l^{rest}) + k(v^m_{1+1} \cdot \Delta x^1) + \Delta x_1\) and \(m_2\) is the other spring end-point, \(a = \Delta t(1/m, 0, I^{-1}[r \times_1, 0], 0)^T\), and \(e_1 = (1, 0, 0)^T\).  

C. Progressive Training via Implicit/Semi-implicit Integr

The objective of progressive training is to mitigate the frequency discrepancy between the training data and the physics engine. Instead of 2-way coupling [13], [41], which applies different integration methods for soft and rigid bodies, this work proposes a progressive training method for fast and stable convergence. The idea is to train the engine using implicit integration first in a time-stepping way to find a coarse grained model first, and then fine tune with semi-implicit integration for a more accurate model. The implicit integration has a large region of absolute stability in terms of parameters and time step but it is computationally expensive and reduces system energy steadily over time. Fine tuning with semi-implicit integration is fast and more accurate while it maintains system energy. The proposed progressive training approach takes advantage of both schemes’ benefits. In order to achieve a correct gradient, a variation of semi-implicit integration is also proposed, named as adaptive naive integration (ANI), which is described in the appendix.

---

\(^1\)https://sites.google.com/view/recurrentengine
The progressive training algorithm is shown in Alg. 1, where $T$ is the sampling interval, $lr$ is the learning rate, and $\Delta t_r$ is the engine’s simulation step. The engine is first trained recurrently with implicit integration (line 3-10), starting with a large time step and a large learning rate. Once the validation loss increases, the time step reduces gradually for a more precise simulation until the time step agrees with the physics engine’s frequency (line 6-7). Once the training time step agrees with the frequency, the learning rate can reduce to stabilize the parameters and achieve a coarse grained model (line 8-9). Then, this model is fine-tuned with semi-implicit integration (line 12-20) and the learning rate keeps reducing (line 17-18), until a stable accurate model is achieved.

**Algorithm 1: Progressive Training Algorithm**

**Input:** Sparse Sampled Trajectory

**Parameter:** $T = 100\text{ms}, lr = 0.1, \Delta t_r = 1\text{ms}$

**Output:** Identified System Parameters

1. Let $\Delta t = T$.
2. while $lr > 10^{-4}$ do
3.   Train engine recurrently with implicit integration.
4.   if validation loss is reducing then
5.     continue
6.   else if $\Delta t > \Delta t_r$ then
7.     $\Delta t = \max(\Delta t/2, \Delta t_r)$
8.   else
9.     $lr = lr/2$
10. end if
11. end while
12. $lr = 0.1, \Delta t = \Delta t_r$
13. while $lr > 10^{-4}$ do
14.   Train recurrently with semi-implicit integration.
15. if validation loss is reducing then
16.     continue
17. else
18.     $lr = lr/2$
19. end if
20. end while

**IV. EXPERIMENTS**

The experiments use a model of the SUPERballBot tensegrity robot platform [42] simulated in the MuJoCo engine as the ground truth system. The state $X_t$ is 72-dimensional since there are 6 rigid rods for which the 3D position $p_r$, linear velocity $v_r$, orientation $q_r$, and angular velocity $\omega_r$ are tracked over time. The space of control signals $U_t$ is 24-dimensional, since there are 24 cables and it is possible to control the target length for each of them individually.

The task is to estimate the robot’s parameters, i.e., spring stiffness, damping, rod mass, and contact parameters, i.e., reaction and friction coefficients. The evaluation compares the predicted trajectories of the robot’s center of the mass (CoM) by the differential engine against the ground truth CoM trajectory for the same controls. To generate control, this work evaluate multiple sample-based model predictive control (MPC) algorithms. The objective of the controller is to achieve a desired direction for the platform’s CoM.

The ground truth system on MuJoCo has been setup to mimic empirical motions of the real system at NASA Ames [42] and reflect the NTRT tensegrity robot simulator [34]: rod mass $m$ is 10kg, length 2$r$ is 1.684m the cable rest length $r_{\text{rest}}$ is 0.95m, the stiffness $K$ is 10,000 and damping $k$ is 1,000; the activation dynamics type is filter, the control range is $[-100, 100]$, and the gain parameter is 1,000. All sliding, torsional and rolling frictions are enabled with coefficient equal to 1. A non-contact and a contact environment were setup as shown in Fig. 1. For each environment, 10 trajectories were sampled from MuJoCo for training, 2 for validation and 10 for testing. All trajectories are 5 seconds long. The sampling frequency is 10Hz, however, the engine’s frequency is 1000Hz.

**A. Ablation Study**

Ablation experiments evaluate the proposed components and show that: 1) recurrent training with implicit integration outperforms feed forward training with semi-explicit integration [12]; 2) progressive training outperforms implicit integration only training; 3) differentiable collision detection is faster than non-differentiable one.

1) **Necessity of Recurrent Training:** A naive idea to mitigate the frequency gap between simulation and sensory updates is to reduce the simulation frequency. In the considered setup, the simulation time step would increase from 1ms to 100ms, same to the sampling interval of ground truth trajectories, so as to avoid recurrent training. Fig. 1 (top) shows that this may miss important collision states. Even without collisions, this is still problematic. For instance, consider a non-contact environment where the robot is held in the air and executes random controls, as in Fig. 1 (bottom-left), we compare: a) training the differential engine in a feed-forward fashion with a fixed 100ms time steps and b) using the time-stepping recurrent training (lines 1-11 of Alg. 1). Because of the large time step, both of them apply implicit integration for stable simulation. Fig. 4a (left) shows that the proposed recurrent training outperforms feed-forward training [12], since its mean square error (MSE) over the CoM prediction is much lower.

2) **Improvement of Progressive Training:** Previous efforts [16], [13] claimed good performance with implicit integration only. This section shows that the proposed combination of implicit and semi-implicit integration significantly...

**D. Differentiable Collision Checker**

The motion of the robot also depends upon the reaction forces and the friction between the robot and the ground. Thus, rich contacts should be modeled properly to simulate robot motions. In principle, the physics engine could use an off-the-shelf collision checker. The requirement for training the engine in a recurrent fashion and backpropagating the loss through the engine means that the collision checker should also be differentiable. An accompanying appendix provides an analysis of the gradients at contact points to show that regardless if the collision checker is differentiable or not, its output does not affect the computation of a correct gradient direction. A differentiable collision checker, however, can still accelerate the training process.
improves performance. The same task as in the above section above, where the process first trains with implicit-integration only and then fine-tunes with semi-implicit integration.

The comparison of implicit-only and progressive training is shown in Figure 4b. The first 150 epochs are from implicitly-only training. The training loss keeps going down as the simulation step $\Delta t$ gets smaller. After 100 epochs even for $\Delta t = 1\, \text{ms}$, the loss is still very high. After switching to semi-implicit integration at epoch 151, the loss curve drops sharply since the semi-implicit Euler method conserves energy while the implicit method reduces it, which adds damping. Even though the implicit-only training cannot converge to the best parameters, it converges to a coarse solution good enough for stable training with semi-implicit integration.

3) Speed up due to the Diff. Collision Checker: Since a sparsely sampled trajectory may skip important contact states, the recurrent physics engine needs a collision checker that compensates for such information. But whether this collision checker should also be differentiable needs verification. This experiment trains the engine with a differentiable collision checker and, for comparison, with MuJoCo’s non-differentiable collision checker. The trajectories involve the robot rolling on the ground as in Fig. 1 (right) along random directions and with random speed. The rolling trajectory includes multiple collisions between the rod and the ground.

Fig. 4a (right) shows that for the same training setting, the engine trained with a differentiable collision checker and the engine trained with a traditional one have the same performance. This also means that the proposed engine can benefit from any off-the-shelf collision checker. This can save work from developing collision primitives for new objects. Nevertheless, the drawback of the traditional collision checker relates to training speed. Training with the differentiable collision checker on GPUs (126 ± 7s per epoch) is about 10x faster than using MuJoCo’s collision checker on CPUs (1296 ± 14s per epoch). Although the experiment used a pool with 10 MuJoCo instances in parallel, training with a MuJoCo collision checker was significantly slower. The collision checking models rods as capsules, which is a computationally efficient primitive.

B. System Identification with Sparse Trajectories

This section compares the proposed training using sparsely sampled trajectories against the alternative [12] is trained with dense trajectories. The proposed solution achieves better performance at 100 times smaller sampling frequency, which amounts to only 1% of data. 10 trajectories were sampled from MuJoCo for training, 2 for validation and 10 for testing. All trajectories are 5 seconds long. For the sparse trajectory dataset, data points $X_t$ are sampled every 100ms, then each trajectory has 50 data points. For the dense trajectory dataset, data points are sampled every 1ms, which is the same as the simulation step, then each trajectory has 5,000 data points, i.e., the sparse trajectory contains only 1% of data points of the dense trajectory. The evaluation is performed separately for non-contact trajectories and contact trajectories.

Fig. 5 shows that the proposed engine with sparse data can achieve similar and even better CoM error. The recurrent training process can average out the noise over smaller simulation steps, resulting to a more robust and accurate model. The prediction for non-contact trajectories has very low error with magnitude $10^{-5}$, and on trajectories with contacts has error approx. 0.58m after 5,000 time steps. For the contact environment, the CoM error is divided by trajectory length to get CoM relative error. Considering the average trajectory length is 2.35m, the relative error is in the order of 24.7% of its length. The robot starts with a random initial speed and stops in a range between 100 to 2,000 time steps. After approx. 2,000 time steps, the CoM error curve becomes flat because the ball stops rolling. The CoM error arises from the simplification of the contact model. This simplification reduces the need for training data, but also impacts identification accuracy, which is a trade-off between data requirements and model complexity. The error with contacts appears high but the objective is to achieve a data-efficient process for finding an explainable model for the target system that is accurate enough to train control policies that can be transferred back to the ground truth system. The following sections demonstrate this success criterion.
C. Sim2Sim MPC Policy Transfer

The goal is to train effective control policies in the engine that can be transferred to the original system painlessly. This work demonstrates sim2sim transfer given the MuJoCo simulator as a ground-truth system. Trajectories are sampled from it, and the proposed physics engine is used to identify critical parameters using these input trajectories. Then a policy is trained on the identified engine. The same policy generation is applied on MuJoCo and the comparison evaluates if the policies agree when executed on MuJoCo and how much training data are needed in either case.

The first step is to identify a controller that is a good choice for the tensegrity robot. The SUPERball rolling task shown in Fig. 6 is used to evaluate sampling-based MPC algorithms in MuJoCo that use random shooting (RS) [43], cross-entropy method (CEM) [44], model predictive path integral (MPPI) control [45] or max entropy iLQG (MElQG) [46]. The task is to roll the robot along the x axis at a target velocity of 1m/s. The closer to the target velocity the better. (top) MPPI and CEM are closest to the target velocity. (bottom) The robot starts at (0,0) and the target position is at (5,0). The trajectories of MPPI and CEM are closer along the X axis and end close to the target.

The next step is to train the engine with sensing data for a frequency training data. It can be used to train a controller and transfer it to the ground truth system without adaptation. The engines share the same robot topology and partial observability must be also addressed.

The goal is to train effective control policies in the engine that can be transferred to the original system painlessly. This work demonstrates sim2sim transfer given the MuJoCo simulator as a ground-truth system. Trajectories are sampled from it, and the proposed physics engine is used to identify critical parameters using these input trajectories. Then a policy is trained on the identified engine. The same policy generation is applied on MuJoCo and the comparison evaluates if the policies agree when executed on MuJoCo and how much training data are needed in either case.

The first step is to identify a controller that is a good choice for the tensegrity robot. The SUPERball rolling task shown in Fig. 6 is used to evaluate sampling-based MPC algorithms in MuJoCo that use random shooting (RS) [43], cross-entropy method (CEM) [44], model predictive path integral (MPPI) control [45] or max entropy iLQG (MElQG) [46]. The task is to roll the robot along the x axis at a target velocity of 1m/s. The closer to the target velocity the better. (top) MPPI and CEM are closest to the target velocity. (bottom) The robot starts at (0,0) and the target position is at (5,0). The trajectories of MPPI and CEM are closer along the X axis and end close to the target.

Fig. 7: MPC algorithms evaluation on SUPERball bot. The task is to generate controls for 50 time steps (5 sec.,) to roll the robot along the X direction at a target velocity of 1m/s. The closer to the target velocity the better. (top) MPPI and CEM are closest to the target velocity. (bottom) The robot starts at (0,0) and the target position is at (5,0). The trajectories of MPPI and CEM are closer along the X axis and end close to the target.

This paper proposes a recurrent differentiable physics engine to identify parameters of tensegrity robots. This engine is data efficient, explainable and robust even with low-frequency training data. It can be used to train a controller and transfer it to the ground truth system without adaptation. The next step is to train the engine with sensing data for a real robot and use it to learn controllers, where uncertainty and partial observability must be also addressed.

V. Conclusion

This paper proposes a recurrent differentiable physics engine to identify parameters of tensegrity robots. This engine is data efficient, explainable and robust even with low-frequency training data. It can be used to train a controller and transfer it to the ground truth system without adaptation. The next step is to train the engine with sensing data for a real robot and use it to learn controllers, where uncertainty and partial observability must be also addressed.
tation for sim2real transfer of embodied navigation agents,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2634–2641, 2021.

[39] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, “Active domain randomization,” in *Conference on Robot Learning*. PMLR, 2020, pp. 1162–1176.

[40] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[41] T. Shinar, C. Schroeder, and R. Fedkiw, “Two-way coupling of rigid and deformable bodies,” in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Citeseer, 2008, pp. 95–103.

[42] M. Vespignani, J. M. Friesen, V. SunSpiral, and J. Bruce, “Design of superball v2, a compliant tensegrity robot for absorbing large impacts,” *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2865–2871, 2018.

[43] S. H. Brooks, “A discussion of random methods for seeking maxima,” *Operations research*, vol. 6, no. 2, pp. 244–251, 1958.

[44] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L’Ecuyer, “The cross-entropy method for optimization,” in *Handbook of statistics*. Elsevier, 2013, vol. 31, pp. 35–59.

[45] G. Williams, A. Aldrich, and E. A. Theodorou, “Model predictive path integral control: From theory to parallel computation,” *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[46] S. Levine and P. Abbeel, “Learning neural network policies with guided policy search under unknown dynamics,” in *NIPS*, vol. 27. Citeseer, 2014, pp. 1071–1079.