# 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 executed by a robot (henceforth referred to as the ground-truth system). This is a system identification problem, which is necessary before employing a model-based controller.

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.

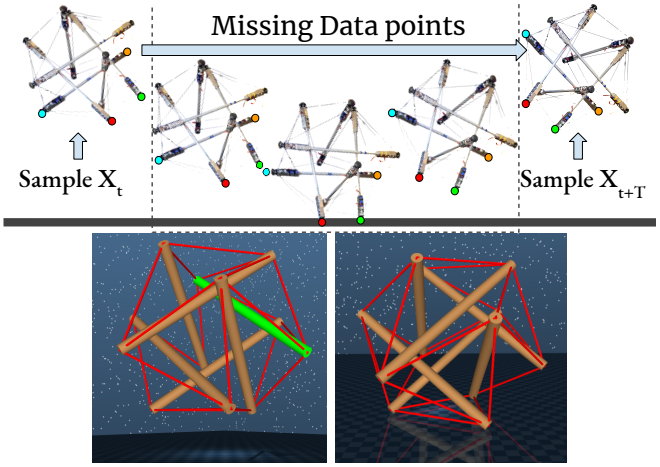


Fig. 1: (top) A sparsely sampled trajectory, given observation frequency T, may skip critical states with contacts. (bottom-left) A non-contact setup used first in a progressive training process: the green rod of the tensegrity is kept fixed, while random forces are applied to the other rods. (bottom-right) A MuJoCo terrain environment where the robot is rolling. Training data comes from MuJoCo, which is also used for visualization.

Differentiable physics engines for dynamic trajectory prediction have emerged as a promising data-driven tool for system identification as they allow for data-driven model inference using backpropagation. Some of them are built entirely on neural networks [5], [6], [7] and can model many different systems, but are data hungry since they have a large number of hidden variables. Other differentiable engines, similar to this work, are built given first principles for a specific physical system [8], [9], [10], [11], which allows them to be more data-efficient as they need to learn fewer parameters. They are also more explainable since they inform about the physical properties of the underlying system.

In previous work, the authors have developed the first differentiable engine targeted for tensegrity robots and argued its data efficiency [12]. Nevertheless, critical gaps remain for the effective deployment of such tools on real robots. In particular, the data collection process for real robots commonly utilizes sensors, such as a camera or inertial measurement unit (IMU), where the frequency of the ground-truth data is constrained by the sampling rate of the sensor. In contrast, physics engines are executed at high frequencies for accurate simulation, especially for modeling contacts. This frequency discrepancy raises the challenge that the dynamics model needs to predict the missing data points of a discretely sampled sparse trajectory. In addition, the missing data points may include events that are critical to the motion, such as

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 differentable 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 differentiated 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

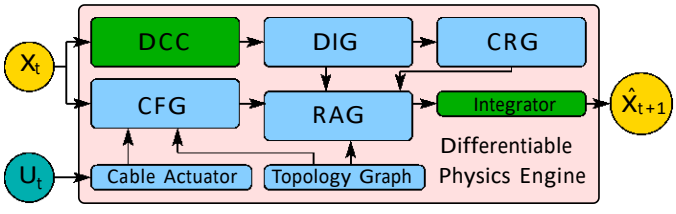


Fig. 2: The physics engine takes the current robot state $X_t$ and control $U_t$ as inputs and predicts the next state $\hat{X}_{t+1}$. Compared to previous attempts [12], this work introduces recurrent training (as shown in Fig. 3), a new numerical integrator, a progressive training pipeline and a new 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.

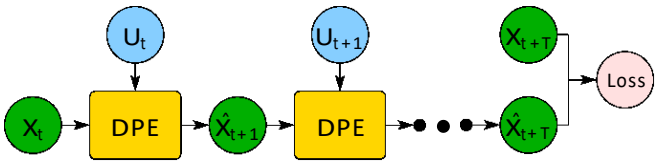


Fig. 3: The recurrent training pipeline is trained via supervision given a state-action tuple $X_t; U_t; X_{t+T}$. The engine will generate all missing states between $X_t$ and $X_{t+T}$. The mean square error (MSE) loss between the predicted state $\hat{X}_{t+T}$ and the ground truth $X_{t+T}$ generates the gradients needed to update the engine's internal parameters. DPE: Differentiable Physics Engine.

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 $U_t; U_{t+1}; :::; U_{t+T-1}$, 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):

$$\begin{bmatrix} 0 & 1 & 0 & & & & & & \\ 0 & 0 & 1 & & & & & & \\ 1 & K & kB & & & & & & \\ c & & & & & & & & \\ & & & 1 & Dt & 0 & 0 & & \\ & & & 0 & 0 & 1 & 0 & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} f_{t+1} \\ x^{m_1}_{t+1} \\ v^{m_1}_{t+1} \\ x^{R}_{t+1} \\ v^{R}_{t+1} \\ w^{R}_{t+1} \\ x^{R}_{t+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \hat{Dx}_{tx}(\hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz}) \\ \hat{Dx}_{ty}(\hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz}) \\ \hat{Dx}_{tz}(\hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz}) \end{bmatrix}; \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & [w^R_t] \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where $x^{m_1}$ and $x^R$ 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 $m_1$, $v^R$ is the rod linear velocity, $w^R$ is the rod angular velocity, $\hat{Dx}$ 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, $[w^R_t]$ and $[r_t]$ are skew-symmetric matrices of $w^R_t$ and $r_t$, $c = K(x^{m_2}_t + l^{rest}_t) + k(v^{m_2}_t \hat{Dx}_t)\hat{Dx}_t$ and $m_2$ is the other spring end-point, $\mathbf{a} = Dt(1=m; 0; I_t^{-1}[r_t]; 0)^T$, and $\mathbf{e}_1 = (1; 0; 0)^T$.

### 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 $Dt_r$ 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 = 100ms; lr = 0:1; $Dt_r$ = 1ms
Output: Identified System Parameters

```
1: Let Dt = 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 Dt > Dt_r then
7:         Dt = max(Dt=2; Dt_r)
8:     else
9:         lr = lr=2
10:    end if
11: end while
12: lr = 0:1; Dt = Dt_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
```

### 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.

## 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 $\mathbf{p}_t$, linear velocity $\mathbf{v}_t$, orientation $\mathbf{q}_t$ and angular velocity $\mathbf{w}_t$ 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 simula-tor [34]: rod mass m is 10kg, length 2r is 1.684m the cable rest length $l^{rest}$ 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 samppled 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 sensing 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

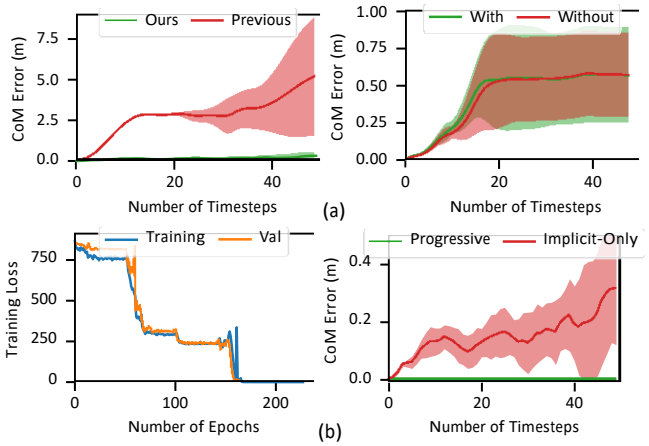


Fig. 4: (a) Left: A previously proposed feed-forward engine [12] exhibits large errors when trained over sparsely sampled trajectories (red line). The proposed recurrent process achieves significantly lower error (green line). Right: In the rolling environment, the predicted trajectory has similar performance w. or w/o the diff. collision checker. (b) Left: With implicit integration, the training loss stops decreasing after 100 epochs. After switching to semi-implicit integration at 151 epoch, the training loss drops sharply. Right: Progressive training results in a CoM error of magnitude 1e-5, which significantly outperforms implicit-only integration. The rod length is 1.682m. The average trajectory length is 2.57m. The error cloud shows the standard derivation.

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 implicit-only training. The training loss keeps going down as the simulation step Dt gets smaller. After 100 epochs even for Dt = 1ms, 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 (1267s per epoch)

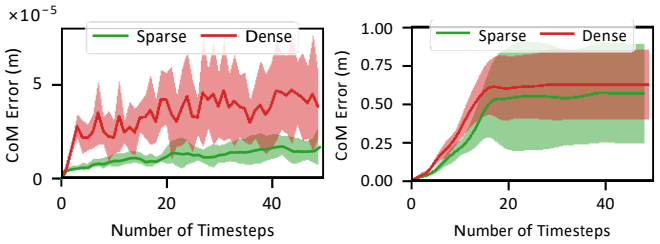


Fig. 5: (left) The proposed engine achieves smaller CoM position error in non-contact environments when trained on sparse trajectories, which contain only 1% of data points of dense trajectories, relative to previous work [12] trained on dense trajectories. (right) Even when using only 1% of data points, the proposed engine achieves comparable error regarding the CoM position in the contact environment.

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 10e-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.

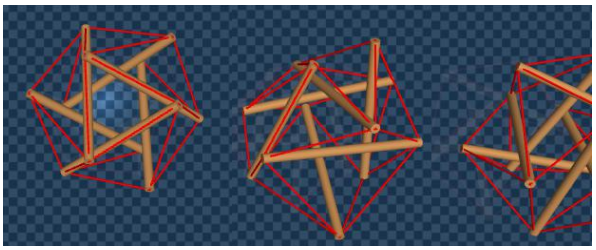

Fig. 6: The SUPERball rolling task is to control the SUPERBall bot so as to move along the X axis (to the right) with speed 1m=s.

### 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.

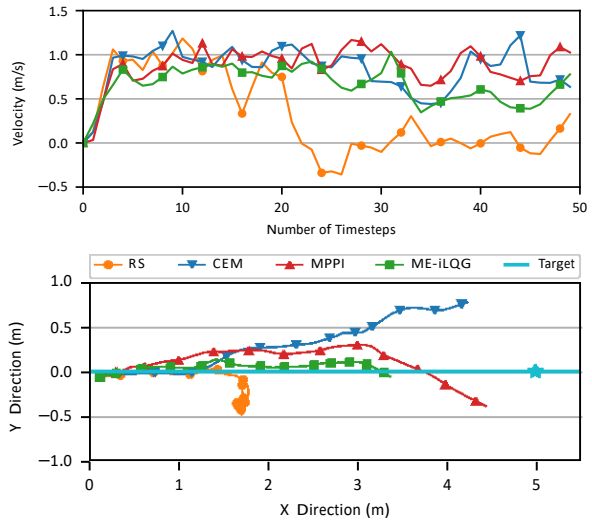


Fig. 7: MPC algorithms evaluation on SUPERball bot. The task is to generate controls for 50 time steps (5 sec.) to roll the robot along the X direction at a target velocity of 1m=s. The closer to the target velocity the better. (top) MPPI and CEM are most close to the target velocity. (bottom) The robot 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 first step is to identify a controller that is a good choice for the tesnegrity 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 (ME-iLQG) [46]. The task is to roll the robot along the x axis with a velocity of 1m=s. Each policy has 50 time steps, and each time step is 0:1s. The start is at (0; 0) and the target is (5; 0). MPPI and CEM generate the best policies among all algorithms. MPPI is slightly better since its trajectory ends closer to the target position and is the one selected. Equipped with this information, the next step is to evaluate whether the MPPI variant of MPC can be trained on the diff. physics engine and then transferred successfully on MuJoCo.

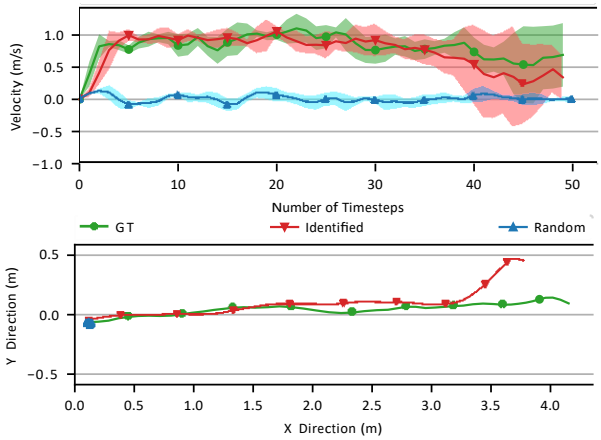


Fig. 8: Multiple trajectories are generated on MuJoCo from policies trained on the ground-truth (GT) system, the identified engine and an engine with random parameters. The policy learned on the identified engine has similar output (top:velocity,bottom:trajectory) on MuJoCo with the policy trained on MuJoCo, while the other policy does not result in motion of the CoM.

1) Random v.s Identified Parameters: MPPI is first trained and executed on MuJoCo to get a ground truth policy. The policy transfer involves training MPPI on the diff. engine and then executing it on MuJoCo. This section evaluates if the identification process is necessary for the transfer. Two diff. physics engines are compared with different parameters for stiffness, damping, mass, friction, etc.: 1) random parameters; 2) parameters identified with the recurrent training process. The engines share the same robot topology and dimensionality. MPPI is trained on these engines to get 2 sets of policies, which are executed on MuJoCo to compare with the ground truth policy. Fig 8 shows that the trajectories of the policy trained with random parameters doesn't match the policy trained on MuJoCo, since the achieved velocity is almost zero and the robot is stuck at the origin. The policy trained on the identified engine, however, is very close to the ground truth policy and achieves a velocity close to 1m=s.

2) Data Requirement: Each policy has 50 controls. For each control, MPPI runs 5 iterations, samples 40 trajectories in each iteration and each trajectory is 1sec, i.e. 1,000 time steps. The sampling interval is 100ms. If the policy is directly trained on the ground truth system, then it requires: 50 5 40 1; 000=100 = 100K data points. The diff. engine only needs 10 5 sec. trajectories to identify non-contact parameters and 10 5sec. trajectories for contact ones, i.e. (10 + 10) 5; 000=100 = 1000 data points, which is only 1 % of 100K. It takes around 8 hours to train the recurrent engine and generate the control policy, which happens offline, and replaces the collection of real world data with compute.

## 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.

## References


[1] S. Lessard, D. Castro, W. Asper, S. D. Chopra, L. B. Baltaxe-Admony, M. Teodorescu, V. SunSpiral, and A. Agogino, "A bio-inspired tensegrity manipulator with multi-dof, structurally compliant joints," in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2016, pp. 5515–5520.

[2] A. P. Sabelhaus, L. J. van Vuuren, A. Joshi, E. Zhu, H. J. Garnier, K. A. Sover, J. Navarro, A. K. Agogino, and A. M. Agogino, "Design, simulation, and testing of a flexible actuated spine for quadruped robots," arXiv preprint arXiv:1804.06527, 2018.

[3] M. Chen, J. Liu, and R. E. Skelton, "Design and control of tensegrity morphing airfoils," Mechanics Research Communications, vol. 103, p. 103480, 2020.

[4] J. Bruce, A. P. Sabelhaus, Y. Chen, D. Lu, K. Morse, S. Milam, K. Caluwaerts, A. M. Agogino, and V. SunSpiral, "Superball: Exploring tensegrities for planetary probes," 12th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS), 2014.

[5] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and k. kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in Advances in Neural Information Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/3147da8ab4a0437c15ef51a5cc7f2dc4-Paper.pdf

[6] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in International Conference on Machine Learning. PMLR, 2018, pp. 4470–4479.

[7] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. F. Fei-Fei, J. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," in Advances in Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/fd9dd764a6f1d73f4340d570804eacc4-Paper.pdf

[8] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in Advances in neural information processing systems, 2018, pp. 7178–7189.

[9] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable simulation for physical system identification," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 3413–3420, 2021.

[10] M. Lutter, J. Silberbauer, J. Watson, and J. Peters, "A differentiable newton euler algorithm for multi-body model learning," 2020.

[11] ——, "Differentiable physics models for real-world offline model-based reinforcement learning," in IEEE International Conference on Robotics and Automation (ICRA), May 2021.

[12] K. Wang, M. Aanjaneya, and K. Bekris, "Sim2sim evaluation of a novel data-efficient differentiable physics engine for tensegrity robots," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 1–8.

[13] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "Add: analytically differentiable dynamics for multi-body systems with frictional contact," ACM Transactions on Graphics (TOG), vol. 39, no. 6, pp. 1–15, 2020.

[14] J. Degrave, M. Hermans, J. Dambre, et al., "A differentiable physics engine for deep learning in robotics," Frontiers in neurorobotics, vol. 13, p. 6, 2019.

[15] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," ICLR, 2020.

[16] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Scalable differentiable physics for learning and control," arXiv preprint arXiv:2007.02168, 2020.

[17] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," Science Robotics, vol. 4, no. 26, 2019.

[18] K. Wang, M. Aanjaneya, and K. Bekris, "A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines," in Proceedings of the 2nd Conference on Learning for Dynamics and Control, ser. Proceedings of Machine Learning Research, A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, Eds., vol. 120. PMLR, 10–11 Jun 2020, pp. 651–665. [Online]. Available: https://proceedings.mlr.press/v120/wang20b.html

[19] G. Sutanto, A. Wang, Y. Lin, M. Mukadam, G. Sukhatme, A. Rai, and F. Meier, "Encoding physical constraints in differentiable newton-euler algorithm," in Learning for Dynamics and Control. PMLR, 2020, pp. 804–813.

[20] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," 2018.

[21] J. K. Murthy, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Levesque, K. Xie, K. Erleben, et al., "gradsim: Differentiable simulation for system identification and visuomotor control," in International Conference on Learning Representations, 2020.

[22] E. Heiden, D. Millard, H. Zhang, and G. S. Sukhatme, "Interactive differentiable simulation," arXiv preprint arXiv:1905.10706, 2019.

[23] E. Heiden, M. Macklin, Y. S. Narang, D. Fox, A. Garg, and F. Ramos, "DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting," in Proceedings of Robotics: Science and Systems, Virtual, July 2021.

[24] T. Du, J. Hughes, S. Wah, W. Matusik, and D. Rus, "Underwater soft robot modeling and control with differentiable simulation," IEEE Robotics and Automation Letters, vol. 6, no. 3, pp. 4994–5001, 2021.

[25] M. Bächer, E. Knoop, and C. Schumacher, "Design and control of soft robots using differentiable simulation," Current Robotics Reports, pp. 1–11, 2021.

[26] W. Wang, S. Axelrod, and R. Gómez-Bombarelli, "Differentiable molecular simulations for control and learning," in ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020. [Online]. Available: https://openreview.net/forum?id=YDNzrQRsu

[27] H. Wu, Z. Yao, J. Wang, and M. Long, "Motionrnn: A flexible model for video prediction with spacetime-varying motions," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 15 435–15 444.

[28] K. Suzuki, H. Mori, and T. Ogata, "Compensation for undefined behaviors during robot task execution by switching controllers depending on embedded dynamics in rnn," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 3475–3482, 2021.

[29] P. Kothari, B. Sifringer, and A. Alahi, "Interpretable social anchors for human trajectory forecasting in crowds," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2021, pp. 15 556–15 566.

[30] D. Guo, W. Zhou, H. Li, and M. Wang, "Hierarchical lstm for sign language translation," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[31] M. Zhang, X. Geng, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel, and S. Levine, "Deep reinforcement learning for tensegrity robot locomotion," in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 634–641.

[32] J. Luo, R. Edmunds, F. Rice, and A. M. Agogino, "Tensegrity robot locomotion under limited sensory inputs via deep reinforcement learning," in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 6260–6267.

[33] D. Surovik, K. Wang, M. Vespignani, J. Bruce, and K. E. Bekris, "Adaptive Tensegrity Locomotion: Controlling a Compliant Icosahedron with Symmetry-Reduced Reinforcement Learning," International Journal of Robotics Research (IJRR), 2019.

[34] NASA, "NASA Tensegrity Robotics Toolkit," Accessed 2020, https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim.

[35] B. T. Mirletz, I.-W. Park, R. D. Quinn, and V. SunSpiral, "Towards bridging the reality gap between tensegrity simulation and robotic hardware," in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2015, pp. 5357–5363.

[36] K. Caluwaerts, J. Despraz, A. Işçen, A. P. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral, "Design and control of compliant tensegrity robots through simulation and hardware validation," Journal of the royal society interface, vol. 11, no. 98, p. 20140520, 2014.

[37] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 8973–8979.

[38] J. Truong, S. Chernova, and D. Batra, "Bi-directional domain adap-

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.
[47] H. Goldstein, C. P. Jr., and J. Safko, Classical Mechanics. USA: Pearson; 3rd edition, 2001.

# APPENDIX

## VI. IMPLICIT INTEGRATION DERIVATION


Fig. 9: Simple Spring Rod Model for Implicit Integration

Every spring-rod system can be considered a connection of basis elements: the spring and the rod. Instead of deriving the implicit integration for the whole system, we do for the single element, which is simpler and more straightforward. Considering a spring connecting two rods as shown in Fig. 9, the dynamics with implicit integration is

$$
\begin{aligned}
x^{m_1}_{t+1} &= x^R_{t+1} + r_{t+1} \\
v^{m_1}_{t+1} &= v^R_{t+1} + w^R_{t+1}\, r_{t+1} \\
Dx_{t+1} &= x^{m_1}_{t+1} \quad x^{m_2}_{t+1} \\
\hat{Dx}_{t+1} &= Dx_{t+1} / jjDx_{t+1}jj \\
Dv_{t+1} &= v^{m_1}_{t+1} \quad v^{m_2}_{t+1} \\
l^{rest}_{t+1} &= (l^{rest} + w_{t+1}\, c)\, \hat{Dx}_{t+1} \\
v^{proj}_{t+1} &= (Dv_{t+1}\, \hat{Dx}_{t+1})\, \hat{Dx}_{t+1} \\
f_{t+1} &= K(Dx_{t+1} \quad l^{rest}_{t+1}) + kv^{proj}_{t+1} \\
t_{t+1} &= r_{t+1} \quad f_{t+1} \\
v^R_{t+1} &= v^R_t + \frac{f_{t+1}}{m} Dt \\
x^R_{t+1} &= x^R_t + v^R_{t+1} Dt \\
w^R_{t+1} &= w^R_t + I_t^{-1} t_{t+1} Dt \\
r_{t+1} &= r_t + w^R_{t+1} Dt
\end{aligned}
$$

where $x^{m_1}$ is the position of $m_1$, $x^R$ is the position of the rod on the left, r is the torque arm from $m_1$ to rod's center of mass, $v^{m_1}$ is the linear velocity of $m_1$, $v^R$ is the rod linear velocity, $w^R$ is the rod angular velocity, Dx is the spring vector, jjDxjj is 2-norm of spring vector, i.e. spring length, $\hat{Dx}$ is the spring direction, Dv is the relative velocity of two ends of the spring, $l^{rest}$ is the spring rest length, w is the motor position, c is the motor position scaler, $v^{proj}$ is the projection of relative velocity v onto the spring direction $\hat{Dx}$, f is the spring force under Hooke's law, K is spring stiffness, k is spring damping, t is the torque of spring force onto the rod, m is rod mass, and $I^{-1}$ is the inverse of rod inertia matrix. $t; t+1$ mean the current or next time step.

Solving the above 13 equations directly is impossible, since we only have 13 equations but all terms with subscript $t+1$ are unknown, which are way more than 13. Besides, the quadratic terms like $w^R_{t+1} \quad r_{t+1}; r_{t+1} \quad f_{t+1}$ make the whole system nonlinear.

So we simplify the above equations by replacing part of unknown $t+1$ terms by t terms, which conserve the stable feature of the implicit-integration but much easier to solve.

The simplified implicit integration is

$$
\begin{aligned}
x^{m_1}_{t+1} &= x^R_{t+1} + r_{t+1} \\
v^{m_1}_{t+1} &= v^R_{t+1} + w_t{}^R r_{t+1} \\
Dx_{t+1} &= x^{m_1}_{t+1} \quad x^{m_2}_t \\
\hat{Dx_t} &= Dx_t = jjDx_t jj \\
Dv_{t+1} &= v^{m_1}_{t+1} \quad v^{m_2}_t \\
I^{rest}_t &= (I^{rest} + w_t\, c)\, \hat{Dx_t} \\
v^{proj}_{t+1} &= (Dv_{t+1}\, \hat{Dx_t})\, \hat{Dx_t} \\
f_{t+1} &= K(Dx_{t+1} \quad I^{rest}_t) + kv^{proj}_{t+1} \\
t_{t+1} &= r_t \quad f_{t+1} \quad v_{t+1} \\
&= {}^R v_t + \frac{f_{t+1}\, Dt}{m} \\
x^R_{t+1} &= x^R_t + v^R_{t+1} Dt \\
w^R_{t+1} &= w^R_t + I_t{}^{-1} t_{t+1} Dt \\
r_{t+1} &= r_t + w^R_{t+1} Dt
\end{aligned}
$$

in which all quadratic terms are gone and only have less unknown variables.

Now let's convert above equations to format like $Ax = b$,

$$
\begin{aligned}
f_{t+1} &= K(Dx_{t+1} \quad I^{rest}_t) + kv^{proj}_{t+1} \\
&= K(x^{m_1}_{t+1} \quad x^{m_2}_t \quad I^{rest}_t) + k\, Dv_{t+1}\, \hat{Dx_t}\, Dx_t \\
&= K x^{m_1}_{t+1} \quad K(x^{m_2}_t + I^{rest}_t) + k(v^{m_1}_{t+1} \quad v^{m_2}_t)\, \hat{Dx_t}\, Dx_t = \\
& K x^{m_1}_{t+1} \quad K(x^{m_2}_t + I^{rest}_t) \\
& + k \begin{bmatrix} \hat{Dx}_{tx} & 0 & 0 \\ 0 & \hat{Dx}_{ty} & 0 \\ 0 & 0 & \hat{Dx}_{tz} \end{bmatrix} \begin{bmatrix} \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \\ \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \\ \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \end{bmatrix} v^{m_1}_{t+1} \\
& k(v^{m_2}_t\, \hat{Dx_t})\, Dx_t
\end{aligned}
$$

i.e.

$$
\begin{aligned}
& K x^{m_1}_{t+1} \quad f_{t+1} \\
& + k \begin{bmatrix} \hat{Dx}_{tx} & 0 & 0 \\ 0 & \hat{Dx}_{ty} & 0 \\ 0 & 0 & \hat{Dx}_{tz} \end{bmatrix} \begin{bmatrix} \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \\ \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \\ \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \end{bmatrix} v^{m_1}_{t+1} \\
& = K(x^{m_2}_t + I^{rest}_t) + k(v^{m_2}_t\, \hat{Dx_t})\, Dx_t
\end{aligned}
$$

Transform all equations to form $Ax = b$

$$
\begin{aligned}
& x^{m_1}_{t+1} \quad x^R_{t+1} \quad r_{t+1} = 0 \\
& v^{m_1}_{t+1} \quad v^R_{t+1} \quad [w^R_t] r_{t+1} = 0 \\
& K x^{m_1}_{t+1} + k \begin{bmatrix} \hat{Dx}_{tx} & 0 & 0 \\ 0 & \hat{Dx}_{ty} & 0 \\ 0 & 0 & \hat{Dx}_{tz} \end{bmatrix} \begin{bmatrix} \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \\ \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \\ \hat{Dx}_{tx} & \hat{Dx}_{ty} & \hat{Dx}_{tz} \end{bmatrix} v^{m_1}_{t+1} \\
& = f_{t+1} + K(x^{m_2}_t + I^{rest}_t) + k(v^{m_2}_t\, \hat{Dx_t})\, Dx_t \\
& v^R_{t+1} \quad \frac{Dt}{m} f_{t+1} = v^R_t \\
& x^R_{t+1} \quad Dt\, v^R_{t+1} = x^R_t \\
& w^R_{t+1} \quad Dt I_t{}^{-1} [r_t]\, f_{t+1} = w_t{}^R \\
& r_{t+1} \quad w^R_{t+1} Dt = r_t
\end{aligned}
$$

## VII. Gradients at Discontinuities

Research has shown that time discretization can result in the computation of wrong gradients at collision point [15]. Continuous collision detection (CCD) is a possible solution to circumvent this problem [15], [16]. Nevertheless, CCD is expensive and has many limitations. This work proposed a simple alternative solution that circumvents CCD.

Consider a rigid ball example [15], where a rigid ball elastically collides with a friction-less ground, as shown in Figure 10. Lowering the initial ball height will increase the final ball height, since there is less distance to travel before the ball hits the ground and more after (see the loss curves in Fig. 10, right), i.e., $\P x_T = \P x_0 = 1$, where:

$$
\frac{\P x_T}{\P x_0} = \prod_{t=0}^{T-1} \frac{\P x_{t+1}}{\P x_t}
$$

For other time steps before and after collision:

$$
\begin{aligned}
v_{t+1} &= v_t; \\
x_{t+1} &= x_t + v_{t+1} Dt; \\
\frac{\P x_{t+1}}{\P x_t} &= 1:
\end{aligned}
$$

Thus, assume the collision happens at t, then

$$
\begin{aligned}
v_{t+1} &= v_t \\
\frac{\P x_T}{\P x_0} &= \prod_{t=0}^{T-1} \frac{\P x_{t+1}}{\P x_t} = \frac{\P x_{t+1}}{\P x_t}:
\end{aligned}
$$

thus in the following sections, we only focus on $\frac{\P x_{t+1}}{\P x_t}$.

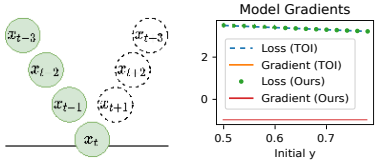


Fig. 10: left: The rigid ball drops onto the ground at time step t and bound up. right: Our adaptive naive integration (ANI) can get correct gradient and loss as previous time of impact (TOI) integration.

### A. Gradient Analysis for Naive Integration and Time of Impact (TOI) Integration

The authors [15] mentioned the naive integration would return wrong gradient at collision point and proposed Time of Impact (TOI) integration to circumvent this problem. But no theoretical analysis proposed about why. Here proposed the analysis here.

With naive time integration,

$$
\begin{aligned}
v_{t+1} &= v_t \\
x_{t+1} &= x_t + v_{t+1} Dt \\
\frac{\P x_{t+1}}{\P x_t} &= 1
\end{aligned}
$$

Thus the gradient has the opposite direction. However, considering CCD with time of impact (TOI), we have

$$
\begin{aligned}
v_{t+1} &= -v_t \\
TOI &= -x_t/v_t \\
x_{t+1} &= x_t + v_t\,TOI + v_{t+1}(Dt - TOI) \\
\frac{\partial x_{t+1}}{\partial x_t} &= 1 + v_t(-1/v_t) - v_t(-1)(-1/v_t) = -1
\end{aligned}
$$

Now the gradient is correct, since TOI introduces additional gradients to $x_t$.

B. Adaptive Naive Integration (ANI) with Correct Gradient

Instead of CCD and TOI, we proposed an adaptive naive integration (ANI) that can also return correct gradients:

$$
\begin{aligned}
v_{t+1} &= -v_t - 2x_t/Dt + 2x_e/Dt; \\
x_{t+1} &= x_t + v_{t+1}Dt; \\
\frac{\partial x_{t+1}}{\partial x_t} &= 1 + \frac{\partial v_{t+1}}{\partial x_t}Dt = 1 - 2 = -1;
\end{aligned}
$$

where $x_e$ has the same value to $x_t$ but does not back-propagate gradient. Fig. 10 right shows our ANI can get same correct gradient and loss to TOI.

The proposed contact model can be viewed as a special impulse-based contact model [47]:

$$
\begin{aligned}
v_{t+1} &= v_t + (-Kx_t - kv_t)/mDt \\
v_{t+1} &= -v_t \\
\frac{\partial x_{t+1}}{\partial x_t} &= -1
\end{aligned}
$$

Solving these equations, the result is:

$$
\begin{aligned}
Kx_tDt/m &= 2x_t/Dt \\
kv_tDt/m &= -2x_e/Dt + 2v_t
\end{aligned}
$$

Consider now a more general case of the impulse-based contact model, $v_{t+1} = -ev_t$, where $e \in [0;1]$.

Theorem 1. For impulse-based contact model, the gradient has a correct direction if $K/m > -1/Dt^2$ and $k/m = (1+e)/Dt + Kx_e/(mv_t)$, where $e$ is restitution, $K < 0; k < 0$ are constants or functions about $x_t; v_t$ but don't propagate gradient.

Proof.

$$
\begin{aligned}
v_{t+1} &= v_t + (-Kx_t - kv_t)/mDt; \\
x_{t+1} &= x_t + v_{t+1}Dt; \\
\frac{\partial x_{t+1}}{\partial x_t} &= 1 - Dt^2K/m
\end{aligned}
$$

In order to ensure the gradient has correct direction, the following condition should be satisfied:

$$
\frac{\partial x_{t+1}}{\partial x_t} = 1 - Dt^2K/m < 0
$$

Thus, $K/m > -1/Dt^2$. Since $v_{t+1} = -ev_t$:

$$
\begin{aligned}
-ev_t &= v_t - Kx_e/mDt - kv_t/mDt \\
k/m &= (1+e)/Dt + Kx_e/(mv_t)
\end{aligned}
$$

□

In the above rigid ball example, we can set $K = -2m/Dt^2$ to ensure that the gradient is -1.

C. Gradient Analysis with/without Differentiable Collision Checker

The above analysis has shown that the impulse-based contact model could return correct gradients. Here we discuss whether collision checker is necessary for getting correct gradient direction. The collision checker returns the contact point and intersection distance. Assuming the ground level is 0 in the rigid ball example, the intersection distance is $-x_t$. The dynamics at contact point is

$$
\begin{aligned}
v_{t+1} &= v_t + (-Kx_t - kv_t)/mDt \\
x_{t+1} &= x_t + v_{t+1}Dt
\end{aligned}
$$

For the system identification task to identify ground stiffness $K$, $\frac{\partial x_{t+1}}{\partial K}$ should be positive since the ball rebounds faster on a stiffer ground.

$$
\frac{\partial x_{t+1}}{\partial K} = -x_tDt^2/m > 0
$$

which is always true since $x_t < 0$ at the contact point. Since $x_t$ is not a function of $K$, whether the collision checker is differentiable will not affect the gradient correctness.

If there are multiple collisions in a time interval $T$, which probably happen during the recurrent training, will the gradients still have correct direction? Considering the rigid ball example, we assume there are two continues collisions at $t-1; t$.

$$
\begin{aligned}
v_t &= v_{t-1} - Kx_{t-1}Dt/m - kv_{t-1}Dt/m \\
x_t &= x_{t-1} + v_tDt \\
v_{t+1} &= v_t - Kx_tDt/m - kv_tDt/m \\
x_{t+1} &= x_t + v_{t+1}Dt
\end{aligned}
$$

If the collision checker is not differentiable, the gradient to $K$ is

$$
\frac{\partial x_{t+1}}{\partial K} = -x_eDt^2/m > 0
$$

where $x_e$ has same value to $x_t$ but doesn't back propagate gradients.

If the collision checker is differentiable, the gradient to $K$ is

$$
\begin{aligned}
\frac{\partial v_t}{\partial K} &= -\frac{x_{t-1}Dt}{m} > 0 \\
\frac{\partial x_t}{\partial K} &= -x_{t-1}Dt^2/m > 0 \\
\frac{\partial v_{t+1}}{\partial K} &= -\frac{x_tDt}{m}\frac{\partial x_t}{\partial K} + -\frac{kDt}{m}\frac{\partial v_t}{\partial K} \\
\frac{\partial x_{t+1}}{\partial K} &= \frac{\partial x_t}{\partial K} + \frac{\partial v_{t+1}}{\partial K}Dt \\
&= \left(-\frac{x_tDt^2}{m} + 1 - \frac{kDt}{m}\right)\frac{\partial x_t}{\partial K}
\end{aligned}
$$

where $x_{t+1}; x_t < 0$. Since $Dt = 0{:}001s$, term $\frac{x_t Dt^2}{m} + 1 \frac{kDt}{m}$ could rarely be negative which means wrong gradient direction. To guarantee correct gradient direction, we apply the detaching trick to stop gradient propagation through $v_t$ and $v_{t-1}$, (by detaching from computation graph in PyTorch), which leads to:

$$\begin{aligned} v_t &= v_{t-1} - Kx_{t-1}Dt/m - kv_{t-1}Dt/m \\ x_t &= x_{t-1} + v_t Dt \\ v_{t+1} &= v_t - Kx_t Dt/m - kv_t Dt/m \\ x_{t+1} &= x_t + v_{t+1} Dt \end{aligned}$$

where $v_{t-1}; v_t$ have same value to $v_{t-1}; v_t$, but doesn't back-propagate gradient.

Thus we can ignore the damping term k to simplify the following gradient computation,

$$\begin{aligned} v_t &= v_{t-1} - Kx_{t-1}Dt/m \\ x_t &= x_{t-1} + v_t Dt \\ v_{t+1} &= v_t - Kx_t Dt/m \\ x_{t+1} &= x_t + v_{t+1} Dt \end{aligned}$$

If the collision checker is differentiable, the gradient to K is

$$\begin{aligned} \frac{\partial x_t}{\partial K} &= -x_{t-1}Dt^2/m > 0 \\ \frac{\partial x_{t+1}}{\partial K} &= \frac{\partial x_t}{\partial K} - \frac{x_t Dt^2}{m}\frac{\partial x_t}{\partial K} \\ &= (-\frac{x_t Dt^2}{m} + 1)\frac{\partial x_t}{\partial K} > 0 \end{aligned}$$

since $x_t < 0; x_{t-1} < 0$. With mathematical induction, we can easily get if there are multiple contacts in the time interval $[i; j]$,

$$\frac{\partial x_{j+1}}{\partial K} = (-\frac{x_j Dt^2}{m} + 1)(-\frac{x_{j-1}Dt^2}{m} + 1):::(-\frac{x_{i+1}Dt^2}{m} + 1)\frac{x_i Dt^2}{m} > 0$$

since $x_i; x_{i+1}; :::; x_{j-1}; x_j < 0$. Thus no matter the collision checker is differentiable or not, we can always get correct gradient direction. Gradient clipping is necessary to avoid exploding gradients.