A First Principles Approach for Data-Efficient System Identification of Spring-Rod Systems via Differentiable Physics Engines

Kun Wang KUN.WANG2012@RUTGERS.EDU
Robotics Lab, 1 Spring Street, New Brunswick, NJ 08901

Mridul Aanjaneya MRIDUL.AANJANEYA@RUTGERS.EDU
CBIM, 617 Bowser Rd, Piscataway, NJ 08854

Kostas Bekris KOSTAS.BEKRIS@CS.RUTGERS.EDU
Robotics Lab, 1 Spring Street, New Brunswick, NJ 08901

Editors: A. Bayen, A. Jadbabaie, G. J. Pappas, P. Parrilo, B. Recht, C. Tomlin, M.Zeilinger

Abstract

We propose a novel differentiable physics engine for system identification of complex spring-rod assemblies. Unlike black-box data-driven methods for learning the evolution of a dynamical system *and* its parameters, we modularize the design of our engine using a discrete form of the governing equations of motion, similar to a traditional physics engine. We further reduce the dimension from 3D to 1D for each module, which allows efficient learning of system parameters using linear regression. As a side benefit, the regression parameters correspond to physical quantities, such as spring stiffness or the mass of the rod, making the pipeline explainable. The approach significantly reduces the amount of training data required, and also avoids iterative identification of data sampling and model training. We compare the performance of the proposed engine with previous solutions, and demonstrate its efficacy on tensegrity systems, such as NASA's icosahedron.

Keywords: system identification, differentiable physics engine, spring-rod systems, tensegrity

1. Introduction

Performing experiments on real robots can be time-consuming, expensive, or dangerous. As such, it is often preferable to explore policies in simulation first, and then transfer them to the real robot. To minimize the reality gap, accurate system identification is important. The traditional approach Swevers et al. (1997); Hansen and Ostermeier (2001) for system identification is to collect ground truth data and subsequently optimize the dynamics model over certain parameters of a physics simulation engine, so as to minimize the difference between the predicted trajectory and the ground truth. This process is repeated iteratively until the difference becomes small enough. Apart from being slow, this iterative process can potentially damage a vulnerable robot. As an alternative, recent approaches have been exploring the use of a *differentiable* physics engine, such as a neural network, that would allow for parameter inference using backpropagation. Nevertheless, training a differentiable physics engine requires massive amounts of data. While there has been some work on methods for online system identification Yu et al. (2017); Allevato et al. (2019), which can learn parameters with much less real data, such methods are limited to a small number of parameters.

The problem of system identification becomes exacerbated for soft robots, which have infinite degrees of freedom. Physics-based methods for simulation require accurate models that capture non-linear material behavior, which are difficult to construct. In contrast, data-driven methods can simulate any system from observed data, with sufficient training data. But the large number of variables and non-linear material properties necessitate copious amounts of training data.

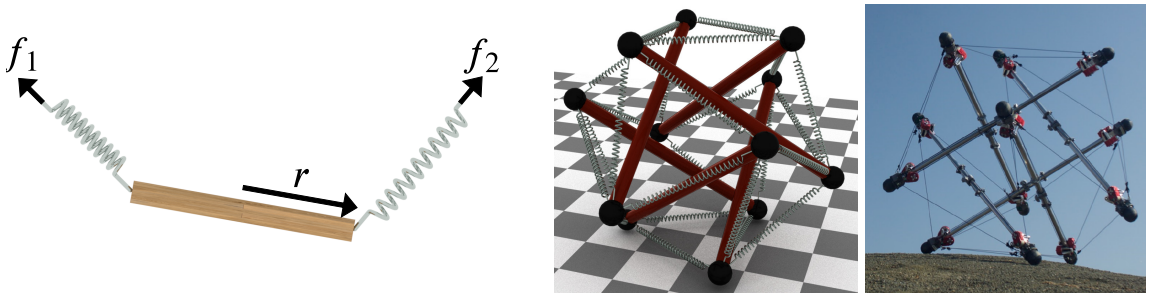


Figure 1: A basic element with one rod connected by two springs (left). A complex assembly of rods and springs forming a tensegrity robot in simulation (middle), and the real world (right).

Cable-driven robots are gaining increasing attention due to their adaptiveness and safety. Tensegrity structures have many applications: from manipulation Lessard et al. (2016), locomotion Sabelhaus et al. (2018), morphing airfoil Chen et al. (2020) to spacecraft lander Bruce et al. (2014). While useful and versatile, they are difficult to accurately model and control. Identifying system parameters is necessary, either to learn controllers in simulation (as real-world experiments are time-consuming, expensive and dangerous), or for traditional model-based control. In all these cases, the spring-rod representation considered in this work is the basic modeling element.

Motivated by these issues, we propose a data-driven differentiable physics engine that combines the benefits of data-driven and physics-based models, while alleviating most of their drawbacks, and is designed from first principles. Previous data-driven models have required large amounts of data, because they learn the parameters *and* the physics of the system. Furthermore, the hidden variables and black box nature of these models are not explainable, and difficult to transfer to new environments. Our approach is based on the observation that the equations that govern the motion of such systems are well-understood, and can be directly baked into the data-driven model. Such a design can reduce demands on training data and can also generalize to new environments, as the governing principles remain the same. We further simplify the differentiable engine by using a modular design, which compartmentalizes the problem of learning the dynamics of the whole system to smaller well-contained problems. For each module, we also reduce the dimension from 3D to 1D, by taking advantage the properties of spring-rod systems, which allows for efficient parameter inference using linear regression. As a side benefit, the regression parameters correspond to physical quantities, such as the spring stiffness or the mass of the rod, making the framework explainable. A video accompanying this work is available here[1].

2. Related Work

Traditional methods for system identification build a dynamics model by minimizing the prediction error Swevers et al. (1997) Hansen and Ostermeier (2001). These methods require parameter refinement and data sampling in an iterative fashion, to decrease the prediction error. This iterative process can be avoided using data-driven techniques that directly fit a physics model to data Rosenblatt (1958); Rumelhart et al. (1986); Asenov et al. (2019). However, these techniques treat the dynamics as a black box, are data hungry, and require retraining in a new environment.

Instead of treating the environment as a black box, Battaglia et al. (2016) took the first step to modularize objects and their interactions in an *interaction network*. Later, Mrowca et al. (2018) introduced a hierarchical relation network for graph-based object representation of rigid and soft bodies by decomposing them into particles. Recently, Li et al. (2019b) proposed the multi-step propagation network and Greydanus et al. (2019) applied a Hamiltonian network to conserve an

1. https://rutgers.box.com/shared/static/i9vvxpc8152i5e0zg897fj47sanen7nj.mp4

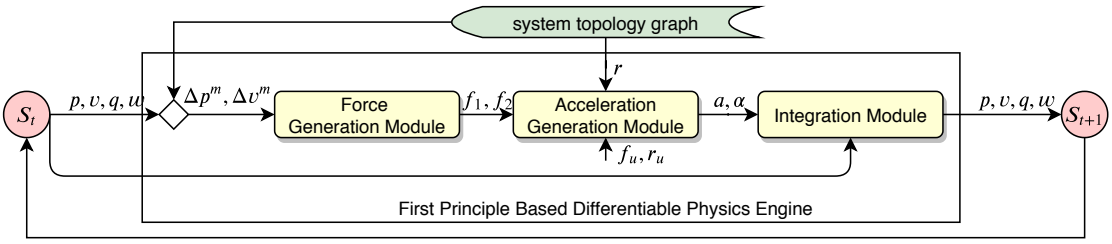


Figure 2: Flow chart showing the data flow when simulating one time step with our physics engine.

energy-like quantity without damping. While these methods are an improvement over previous approaches, they still treat the interactions between different objects as black boxes and try to learn them from data, even though the governing equations of motion are well-understood.

Koopman operator theory from dynamical systems provides an alternative approach to learning the dynamics. The Koopman operator is a mechanism to "lift" lower dimensional non-linear features into higher dimensional linear features, which can subsequently be used to compute a linear dynamics model. Bruder et al. (2019) and Li et al. (2019a) have applied this technique to soft robot dynamics identification. However, the design of the Koopman operator is non-trival, and the high dimensional features are not explainable, making it challenging to estimate all physical parameters.

Quite a few authors have recently introduced differentiable physics engines that focus on many aspects not central to our work. For example, Heiden et al. (2019) predict forward dynamics of articulated rigid bodies, Hu et al. (2019b) solve inverse problems using the Material Point Method (MPM), de Avila Belbute-Peres et al. (2018) address multi-body contact with linear complementarity problems (LCP), and Landry et al. (2019) treat nonlinear optimization with the augmented Lagrangian method. To provide a general interface, Hu et al. (2019a) introduced a compiler for differentiable programming. Ajay et al. (2019) use differentiable engines with traditional physics simulators for control. Researchers have also proposed differentiable engines specific to certain kinds of objects, such as molecules Schoenholz and Cubuk (2019), fluids Schenck and Fox (2018), and cloth Liang et al. (2019). Our work on spring-rod systems is motivated by our recent work on tensegrity robots Surovik et al. (2019); Littlefield et al. (2019); Surovik et al. (2018).

3. Methods

Our system views a spring-rod system as a composition of basic *elements* (see Fig. 1(left)), where springs generate forces that influence rod dynamics. We subdivide each time step of the simulation into three modules: force generation, acceleration computation, and state update/integration (see Fig. 2). The physics engine takes as input the current rod state $S_t = \{p, v, q, \omega\}$, where p is position, v is linear velocity, q is orientation (expressed as a quaternion), and ω is the angular velocity. Based on S_t, the position and linear velocity p^m, v^m of the two rod endpoints is computed, and is used to compute the relative compression (or expansion) and velocity $\Delta p^m, \Delta v^m$ of the two attached springs. Then, the first module predicts the spring forces f, the second module computes the linear and angular accelerations a, α, and the third module computes the new state S_{t+1}.

3.1. System Topology Graph

We use a topology graph to represent interconnections between different components of the spring-rod system. Each rod and spring has a corresponding vertex, and directed edges represent relations between them. Figure 3 shows an example topology graph for the basic spring-rod element shown in Figure 1(left). Unlike Battaglia

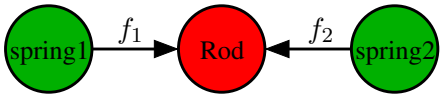


Figure 3: Element topology graph.

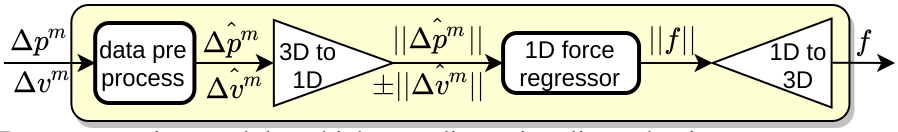


Figure 4: Force generation module, which uses dimensionality reduction to compute spring forces.

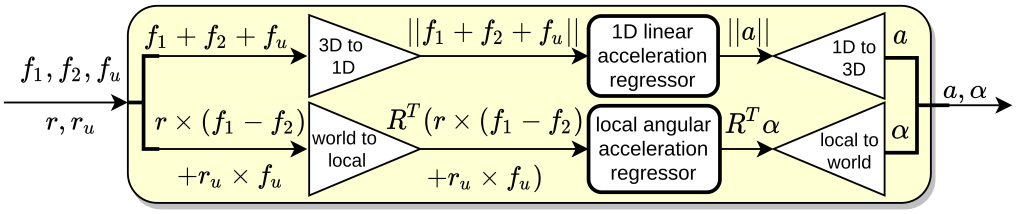


Figure 5: Acceleration generation module, which uses dimensionality reduction similar to the force generation module to compute rod accelerations, given spring forces at the two endpoints as input.

et al. (2016), who assumed all forces are applied at the center of mass of a rod, we apply forces at the endpoints p^m of each rod, which is critical for accurate angular momentum computation.

3.2. Force Generation Module

The relative compression (or expansion) Δp^m and velocity Δv^m of each spring is given as input to the force generation module, which computes the spring forces f as per the equations below:

$$\Delta \hat{p}^m = \Delta p^m - \Delta p^{\text{rest}}, \qquad \Delta \hat{v}^m = (\Delta v^m \cdot \Delta p^m) \cdot (\Delta p^m / ||\Delta p^m||) \tag{1}$$

$$||f|| = -K||\Delta \hat{p}^m|| \mp c||\Delta \hat{v}^m||, \qquad f = ||f|| \cdot (\Delta p^m / ||\Delta p^m||) \tag{2}$$

where K is the spring stiffness, c is the damping parameter, and Δp^{rest} is the spring rest length. To exploit the one-dimensional nature of each spring, we reduce Δp^m and Δv^m from 3D to 1D along the spring direction. This has the benefit of leaving only two unknown parameters, K and c, which can be easily learned using linear regression in a data-efficient fashion. After computing the 1D spring force f, we project it back to 3D by multiplying with the normalized spring direction vector.

3.3. Acceleration Generation Module

The spring forces f and control force f_u are given as input to the acceleration generation module, which computes the linear and angular accelerations a, α of each rod as per the equations below:

$$||f_1 + f_2 + f_u|| = M||a||, \qquad R^T(r \times (f_1 - f_2) + r_u \times f_u) = I(R^T\alpha) \tag{3}$$

$$a = ||a|| \cdot \frac{f_1 + f_2 + f_u}{||f_1 + f_2 + f_u||}, \qquad \alpha = RI^{-1}R^T(r \times (f_1 - f_2) + r_u \times f_u) \tag{4}$$

where f_1 and f_2 are spring forces on the two rod ends, f_u is control force, r is the half-length rod vector, r_u is control force arm, R is the rod local/world frame rotation matrix, M is the rod mass and I is the local frame moment of inertia of the rod. M and I are unknown parameters to identify. In a complex system where a rod is connected with multiple springs on one (or both) of its endpoints, f_1 and f_2 denote the *aggregate* spring force. Analogous to the spring case, this work exploits the 1D nature of each rod by computing the norm of the cumulative force $f_1 + f_2 + f_u$, reducing the dimension from 3D to 1D. After computing the 1D linear acceleration a, the next step is to map them back to 3D using equation (4). We compute angular acceleration $R^T\alpha$ in local frame, where I is a diagonal 3x3 matrix because of the symmetry of a rod. This allows to represent it as a 3x1 vector instead, requiring less data for regression.

3.4. Integration Module and Method Implementation

The integration module computes forward dynamics of each rod using the current accelerations a, α. We apply the semi-implicit Euler method Stewart and Trinkle (2000) to compute the updated state $S_{t+1} = \{p_{t+1}, v_{t+1}, q_{t+1}, \omega_{t+1}\}$ at the end of the current time step.

The learning module receives the current state S_t and returns a prediction $\hat{S_{t+1}}$. The loss function is the MSE between the predicted $\hat{S_{t+1}}$ and ground truth state S_{t+1}. The proposed decomposition, first-principles approach and the cables linear nature allow the application of linear regression, which helps with data efficiency. This linear regression step has been implemented as a single layer neural network without activation function on pyTorch Paszke et al. (2019). We trained 30 epochs with an Adam optimizer by a learning rate starting from 0.1 and reducing 50% every 3 epochs.

4. Experiments

We use two setups: 1) a simple spring-rod system (Fig. 1(a)) and 2) a complex tensegrity system (Fig. 1(b)). For the tensegrity, we first assume uniform parameters for all rods and springs and then non-uniform ones, which is more realistic. The comparison methods represent two categories: 1) techniques that treat the system as a black box, such as Least Square (LS) optimization (L-BFGS-B: Swevers et al. (1997)), Covariance Matrix Adaptation Evolution Strategy (CMA-ES: Hansen and Ostermeier (2001)), 2) methods that reason about physics and system topology, such as the Interaction Network Battaglia et al. (2016) and Koopman operator theory Koopman (1931), which have been used for dynamics model identification Bruder et al. (2019) Li et al. (2019a) Abraham et al. (2017). All these approaches get access to the system state along executed trajectories, i.e., the 3D position, 3D linear velocity, quaternion and 3D angular velocity of each rigid element. This work does *not* use any additional information, such as spring forces, accelerations, etc. The task is to estimate system parameters including spring stiffness K, damping c and rod mass M, inertia I.

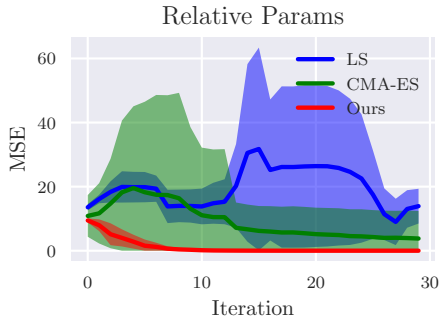
These comparison algorithms were optimized to improve performance on the target experiments. For LS and CMA-ES: the search was constrained to a reasonable range, which increases success ratio. The CMA tolerance was adjusted to 1, which allows to tune the trade-off between accuracy and speed. For the Interaction Network: instead of using the raw object state as in the original paper, the performed experiments used the relative position and velocity corresponding to spring-based models, which is easier for the neural network to learn. For the Koopman operator: instead of basis functions like sinusoids/exponentials, a physics and topology-aware operator was used. All these adaptations increased performance. The methods were tested on an Intel Core i7 6700K CPU, 32G DRAM, 1T SSD and a GeForce GTX TITAN X Pascal GPU.

4.1. Simple Spring-Rod System Identification

This system contains 2 springs and 1 rod as in Fig. 1(a). Each spring is attached to one end of a rod and a fixed nail. The system is modeled in MuJoCo Todorov et al. (2012). Each rod is 2 meters long. The rest length of each spring is 1 m and 1.414 m. Gravity is ignored in this case.

BLACK BOX SYSTEM IDENTIFICATION Comparisons to black box identification methods are included since one of this paper's objectives is to emphasize the limitations of such increasingly popular black-box models. Note that the black-box models have full access to the dynamics, as they resample a new trajectory from the simulator after each optimization iteration. The proposed method doesn't. It instead integrates knowledge from first principles into the optimization process of a differentiable physics engine. We sample 100 trajectories with different initial conditions as ground truth data in order to train the methods. Each trajectory is 2000 time steps long.

The task is to estimate stiffness K, damping c and mass M, which are set to 100, 10 and 10 for ground truth. To assist the optimizer of LS and CMA-ES, we assume inertia I is inferred from the rod geometry and constrain the range of solution to a positive interval $[0.1, 1000]$ for $K > 0, c > 0$, whilst the proposed approach doesn't benefit. From equations (2) (3), we can infer that: $a = -(K/M)(\Delta x_1 + \Delta x_2) - (c/M)(\Delta v_1 + \Delta v_2)$. Thus, any parameter combination satisfying $K/M = 100/10 = 10$ and $c/M = 10/10 = 1$ is a solution. Instead of evaluating the absolute parameters K, c, we test the relative ratios K/M and c/M. MSE in Fig.6 means difference in relative parameters $K/M, c/M$. Success is achieved when the parameter error within 5% of the ground truth. The proposed approach works best whilst both LS and CMA-ES failed in most of the cases as this problem is non-convex and has infinite number of possible answers. The additional blackbox experiments can be found in appendix A.



method	K/M	c/M	sec/itr	success%
GT	10	1		
LS	19.19±7.91	1.51±1.16	1.97±1.43	0.01
CMA-ES	10.29±3.73	3.61±8.22	2.25±0.30	0.18
Ours	**10.00±0.01**	**1.00±0.00**	**0.11±0.01**	**1.00**

Figure 6: (left) mean square error after N iterations for the simple rod-spring systeml; (right) parameters estimated after 30 iterations. The proposed approach is the most accurate and the fastest.

PHYSICS AND SYSTEM TOPOLOGY AWARE IDENTIFICATION We consider alternatives, which do reason about physical properties. **Interaction** is an improved version of the Interaction Network Battaglia et al. (2016) as shown in Fig. 7. It has two Multilayer Perceptrons (MLPs), one to generate spring forces f and the other to generate rod state S_{t+1}. Unlike Battaglia et al. (2016), which takes raw state S_t as input, we apply equation (1) to generate $\Delta\hat{p}_t^m, \Delta\hat{v}_t^m$ as input. **Interaction+Int** appends the integration module to the Interaction Network, and replaces input S_t by r.

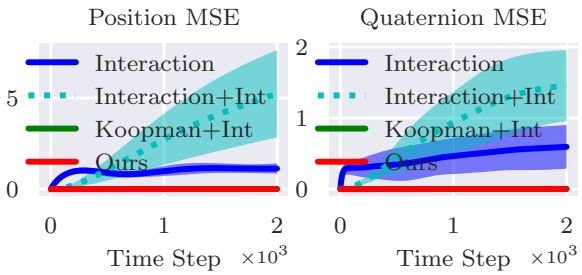


Figure 9: Physics and Topology-aware Identification

Recently, researchers focus on the Koopman operator to fit dynamical models. It applies the kernel trick, similar to SVMs, to map nonlinear features to many dimensions, where we can find linear relations. Instead of kernels, we construct Koopman operators based on the physical properties. We apply the following polynomial basis functions as vector-valued functions to generate the approximate Koopman operator: $\Phi(X) = [1, r, \Delta\hat{p}_1^m, \Delta\hat{p}_2^m, \Delta\hat{v}_1^m, \Delta\hat{v}_2^m, \phi_i]$, where $\phi_i = a^{\alpha_i} b^{\beta_i}$, $a, b \in [r_j, (\Delta\hat{p}_k^m)_j, (\Delta\hat{v}_k^m)_j]$ and $j \in \{x, y, z\}$, $k \in \{1, 2\}$.The terms α_i and β_i are non-negative integers, index i tabulates all the combinations such that $\alpha_i + \beta_i \leq Q$ and $Q > 1$ defines the largest allowed polynomial degree. We define $Q = 2$. Detailed derivation can be found in Appendix B. We only use the Koopman operator to predict accelerations and apply the Integration Module to map them to S_{t+1}, which helps to reduce the complexity of the basis functions. We denote the approach as **Koopman+Int** (Fig. 8). We sampled 1000 trajectories, lasting 2,000

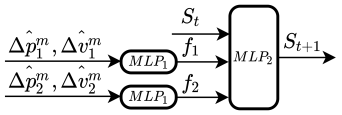


Figure 7: Interaction Network

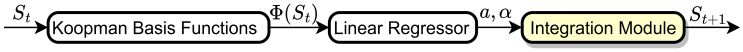


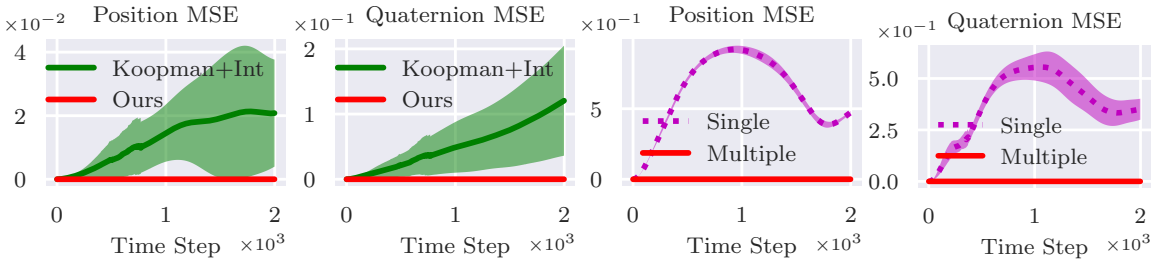
Figure 8: Koopman with Integration Module

time steps, with different initial conditions for training and 200 for validation, and 100 for testing. Trajectories are converted to (S_t, S_{t+1}) pairs for training. Position/Quaternion MSE in Fig.9 means accumulated trajectory difference. **Interaction** only predicts a S_{t+1} in training data that is close to S_t. **Interaction+Int** experiences increasing error from accumulated prediction errors. The Koopman operator **Koopman+Int** designed from first principles gives accurate predictions similar to **Ours** in this simple system.

4.2. Complex Tensegrity Model Identification

We consider an icosahedron tensegrity system as shown in Fig. 1 (c). It is composed of 6 rods and 24 springs. Each rod is connected to 8 springs and has a length of 1.04m. Each spring's rest length is 0.637m. We set the gravity constant to $g = -9.81$ in Mujoco. We collect 1000 trajectories with different initial conditions for training, 200 for validation and 100 for testing. Since only Koopman operator performs well in the simple system, it is the comparison point in this complex setup.

TENSEGRITY WITH UNIFORM PARAMETERS Initially, we set uniform parameters for all rods and springs, i.e. set the mass of rod to 10, and set stiffness and damping to 100 and 10. Thus we keep the number of parameters to estimate, but increase the complexity of the environment. The structure of the Koopman operator is the same as before. The result is shown in Figure 10. **Our** approach outperforms **Koopman+Int** because designing basis functions for The Koopman operator has an increased data requirement relative to our approach.



a) Uniform Parameters b) Different Parameters

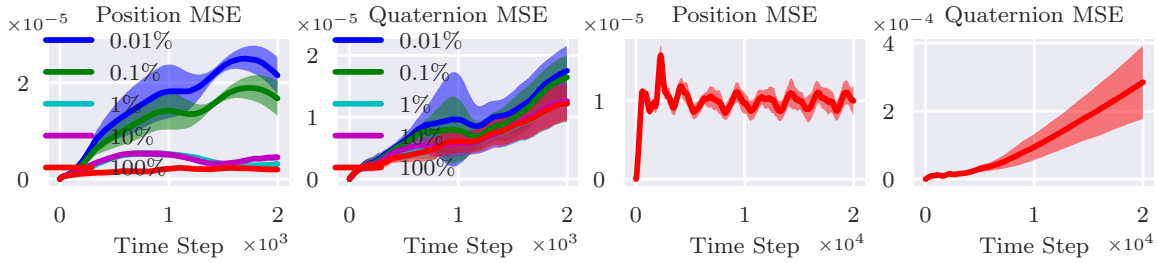
Figure 10: Comparison with the Koopman approach on the complex tensegrity system.

TENSEGRITY WITH NON-UNIFORM PARAMETERS On a real platform the parameters of each element are not the same. We added Gaussian noise to each parameter with $\mu = 0, \sigma = 0.2 * parameter$ resulting in 54 parameters for rod masses, spring stiffness and damping. To deal with this, we use individual regressors for each of rod and spring, which can adapt parameter divergence and achieve higher accuracy. Fig. 10 b) compares two versions of our approach. The **Multiple** uses individual regressors to achieve lower error compared relative to the **Single** regressor. The errors reduce at later time steps because of the periodical property of spring oscillation.

4.3. Data Efficiency Experiment

The proposed method has relatively small data requirements as shown in Fig. 11 a). Instead of training on 1000 trajectories, which have 736,167 time steps in total, we train our model with less data and evaluate performance. We randomly select 10%, 1%, 0.1%, 0.01% of the 736,167 time steps for training. The model achieves good performance even with 73 time steps for training. All

trajectories are from the complex tensegrity setup with different parameters. We performed similar experiments on the Koopman operator, which can be found in Appendix C.



a) Efficiency on Various Training Set b) Generalization on New Testing Set

Figure 11: Data Efficiency and Model Generalization Experiment.

The proposed solution achieves very low error at a magnitude of 10^{-5}, since it: 1) introduces a first-principles approach in learning physical parameters (compared against the popular Interaction Network); 2) removes redundant data from regression (compared to the Koopman operator); 3) operates -for now- over relatively clean data from simulation before moving to real-world data.

4.4. Model Generalization Experiment

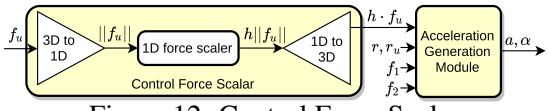


Figure 12: Control Force Scalar

This section generalizes the physics engine trained with a dataset without external forces to a dataset with such forces. We are interested in evaluating: 1) how the physics engine performs for longer time horizons (e.g., after 2000 time steps); 2) if it can adapt to new scenarios. We generate a new test set for the complex tensegrity setup with different parameters: 100 trajectories lasting 20,000 time steps. We also add a random directed perturbation force f_u to a random selected rod every 100 time steps. The external force f_u does not have the same scale as the internal spring forces, so we add a new module that aims to account for the external force, i.e. the control force scalar module, containing only one parameter h, as in Fig. 12. We also apply dimensionality reduction to improve data efficiency. The tuning process is to freeze all other modules' weights and train with object states from the new dataset. The module converges to a stable value. The error graphs are shown in Fig. 11 b). The results on a third data set (with 4,000 time steps) is available in Appendix D.

5. Conclusion and Future Work

This paper proposes a differentiable physics engine for system identification of spring-rod systems based on first principles. The engine has three modules: force generation, acceleration generation and integration, which express the corresponding physical processes of spring-rod systems. This results in reduced data requirements and improved parameter accuracy. It also provides an explainable, accurate and fast physics engine. In the future, we plan to address contacts and friction. This will involve replacing the linear regressor with nonlinear models in the existing modules. To overcome noise in real data, we plan the addition of a residual network along with the nonlinear model. These changes may also help with temporal scalability.

Acknowledgment: Chengguizi Han for rendering the different systems in the figures and video, and Craig Schroeder for sharing his ideas about photorealistic rendering of springs. This work was supported by NASA ECF grant NNX15AU47G, NSF award 1723869, Rutgers University start-up grant, and the Ralph E. Powe Junior Faculty Enhancement Award. Any opinions and conclusions expressed in this work are made by the authors and do not necessarily reflect the views of the sponsor.

References

Ian Abraham, Gerardo De La Torre, and Todd David Murphey. Model-based control using koopman operators. In *2017 Robotics: Science and Systems, RSS 2017*. MIT Press Journals, 2017.

Anurag Ajay, Maria Bauza, Jiajun Wu, Nima Fazeli, Joshua B Tenenbaum, Alberto Rodriguez, and Leslie P Kaelbling. Combining Physical Simulators and Object-Based Networks for Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Adam Allevato, Elaine Schaertl Short, Mitch Pryor, and Andrea L Thomaz. Tunenet: One-shot residual tuning for system identification and sim-to-real robot task transfer. *arXiv preprint arXiv:1907.11200*, 2019.

Martin Asenov, Michael Burke, Daniel Angelov, Todor Davchev, Kartic Subr, and Subramanian Ramamoorthy. Vid2param: Online system identification from video for robotics applications. *arXiv preprint arXiv:1907.06422*, 2019.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

Jonathan Bruce, Andrew P Sabelhaus, Yangxin Chen, Dizhou Lu, Kyle Morse, Sophie Milam, Ken Caluwaerts, Alice M Agogino, and Vytas SunSpiral. Superball: Exploring tensegrities for planetary probes. *12th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, 2014.

Daniel Bruder, C David Remy, and Ram Vasudevan. Nonlinear system identification of soft robot dynamics using koopman operator theory. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6244–6250. IEEE, 2019.

Muhao Chen, Jiacheng Liu, and Robert E Skelton. Design and control of tensegrity morphing airfoils. *Mechanics Research Communications*, 103:103480, 2020.

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems 32*, pages 15353–15363. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9672-hamiltonian-neural-networks.pdf.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019a.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE, 2019b.

Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5):315, 1931.

Benoit Landry, Zachary Manchester, and Marco Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. *arXiv preprint arXiv:1902.03319*, 2019.

Steven Lessard, Dennis Castro, William Asper, Shaurya Deep Chopra, Leya Breanna Baltaxe-Admony, Mircea Teodorescu, Vytas SunSpiral, and Adrian Agogino. A bio-inspired tensegrity manipulator with multi-dof, structurally compliant joints. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5515–5520. IEEE, 2016.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. *arXiv preprint arXiv:1910.08264*, 2019a.

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211. IEEE, 2019b.

Junbang Liang, Ming C. Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Z. Littlefield, D. Surovik, M. Vespignani, J. Bruce, W. Wang, and K. E. Bekris. Kinodynamic planning for spherical tensegrity locomotion with effective gait primitives. *International Journal of Robotics Research (IJRR)*, accepted 2019. URL https://www.cs.rutgers.edu/~kb572/pubs/kinodynamic_tensegrity.pdf.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, pages 8799–8810, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Andrew P Sabelhaus, Lara Janse van Vuuren, Ankita Joshi, Edward Zhu, Hunter J Garnier, Kimberly A Sover, Jesus Navarro, Adrian K Agogino, and Alice M Agogino. Design, simulation, and testing of a flexible actuated spine for quadruped robots. *arXiv preprint arXiv:1804.06527*, 2018.

C. Schenck and D. Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Proceedings of the Second Conference on Robot Learning (CoRL)*, Zurich, Switzerland, 2018.

Samuel S. Schoenholz and Ekin D. Cubuk. Jax m.d.: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python. https://github.com/google/jax-md, https://arxiv.org/abs/1912.04232, 2019.

David Stewart and Jeffrey C Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 162–169. IEEE, 2000.

D. Surovik, J. Bruce, K Wang, M. Vespignani, and K. E. Bekris. Any-axis tensegrity rolling via bootstrapped learning and symmetry reduction. In *International Symposium on Experimental Robotics (ISER)*, Buenos Aires, Argentina, 11/2018 2018. URL https://www.cs.rutgers.edu/~kb572/pubs/any_axis_tensegrity_rolling.pdf.

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.

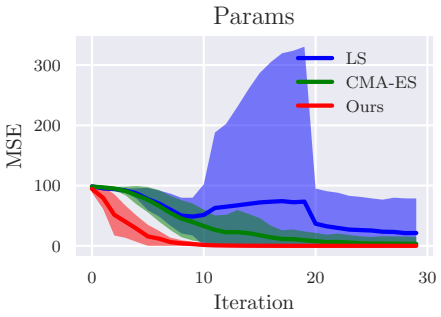
Jan Swevers, Chris Ganseman, D Bilgin Tukel, Joris De Schutter, and Hendrik Van Brussel. Optimal robot excitation and identification. *IEEE transactions on robotics and automation*, 13(5):730–740, 1997.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017. doi: 10.15607/RSS.2017.XIII.048.

Appendix A. Simple Spring-Rod System Black Box Identification

Comparison with Numerical Optimization Methods We assume the rod mass is known, i.e. $M = 10kg$, and only try to identify spring stiffness K and damping c. The ground truth stiffness and damping are set to $K = 100$ and $c = 10$ respectively. For each method, a ground truth trajectory is used as reference and then the mean and standard deviation are computed for robustness evaluation. To assist the optimizer of LS and CMA-ES, we constrain the range of solution to a positive interval $[0.1, 1000]$ for $K > 0, c > 0$. Success is achieved when the parameter error within 5% of the ground truth. MSE in Fig.13 means difference in parameters K, c.



method	stiffness K	damping c	sec/iter	success%
GT	100	10		
LS	98.02±50.71	51.49±247.58	1.32±3.14	0.42
CMA-ES	102.35±16.73	11.01±5.88	1.94±0.22	0.92
Ours	**100.00±0.07**	**10.00±0.01**	**0.11±0.01**	**1.00**

Figure 13: (left) mean square error after N iterations for the simple rod-spring system; (right) parameters estimated after 30 iterations. The proposed approach is the most accurate and the fastest.

The initial guess for LS and CMA-ES is 1. The initial parameters in our physics engine are also set to 1. The CMA tolerance is set to 1. Our approach can converge to a precise model within 10 optimization iterations and takes 1.1s. LS has a large variance as it often doesn't converge. CMA-ES is more stable but still worse than our solution.

Comparison against a Neural Network We consider two baselines as in Fig. 14. A Multi Layer Perceptron, **MLP**, takes S_t as input and predicts S_{t+1}. MLP has 6 fully connected layers. Each layer has 30 hidden variables. The second baseline **MLP+Int** appends the integration module to MLP. It takes S_t as input to predict accelerations a_t, α_t, and then integrates for S_{t+1}. We sampled 1000 trajectories with different initial conditions from Mujoco. Each trajectory has 2,000 time steps. We convert the trajectories to pairs (S_t, S_{t+1}) for training and sample 100 more trajectories for testing. Position/Quaternion MSE in Fig.15 means accumulated trajectory difference. **Our** method outperforms the alternatives, which fail as they do not reason about the underlying physics.

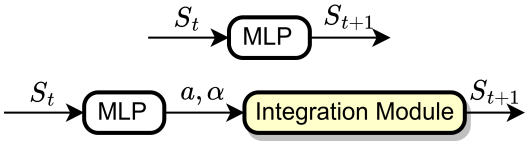


Figure 14: MLP architectures

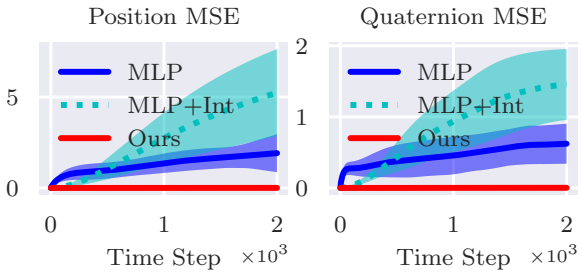


Figure 15: Comparison against Neural Networks

Appendix B. Koopman Operator Derivation from First Principles

In scenarios without external forces f_u, we design a physics based Koopman operator from first principles.

From equations (2), we could infer that

$$\begin{aligned}\boldsymbol{a} &= (f_1 + f_2)/M \\ &= ((-K\Delta\hat{p}_1^m - c\Delta\hat{v}_1^m) + (-K\Delta\hat{p}_2^m - c\Delta\hat{v}_2^m))/M \\ &= (-K(\Delta\hat{p}_1^m + \Delta\hat{p}_2^m) - c(\Delta\hat{v}_1^m + \Delta\hat{v}_2^m))/M \\ &= -\frac{K}{M}(\Delta\hat{p}_1^m + \Delta\hat{p}_2^m) - \frac{c}{M}(\Delta\hat{v}_1^m + \Delta\hat{v}_2^m))\end{aligned}$$

Thus $\boldsymbol{a}$ can be got with a linear function with $\Delta\hat{p}_1^m, \Delta\hat{p}_2^m, \Delta\hat{v}_1^m, \Delta\hat{v}_2^m$.

From equations (3), we could infer that

$$\begin{aligned}\boldsymbol{\alpha} &= RI^{-1}R^T(r \times (f_1 - f_2)) \\ &= RI^{-1}R^T \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \begin{bmatrix} (f_1 - f_2)_x \\ (f_1 - f_2)_y \\ (f_1 - f_2)_z \end{bmatrix} \\ &= RI^{-1}R^T \begin{bmatrix} -r_z(f_1 - f_2)_y + r_y(f_1 - f_2)_z \\ r_z(f_1 - f_2)_x - r_x(f_1 - f_2)_z \\ -r_y(f_1 - f_2)_x + r_x(f_1 - f_2)_y \end{bmatrix} \\ &= RI^{-1}R^T \begin{bmatrix} -r_z(-K(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m) - c(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m))_y + r_y(-K(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_y - c(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m))_z \\ r_z(-K(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m) - c(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m))_x - r_x(-K(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_x - c(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m))_z \\ -r_y(-K(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m) - c(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m))_x + r_x(-K(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_x - c(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m))_y \end{bmatrix} \\ &= -K(RI^{-1}R^T) \begin{bmatrix} -r_z(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_y + r_y(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_z \\ r_z(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_x - r_x(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_z \\ -r_y(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_x + r_x(\Delta\hat{p}_1^m - \Delta\hat{p}_2^m)_y \end{bmatrix} \\ &\quad - c(RI^{-1}R^T) \begin{bmatrix} -r_z(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m)_y + r_y(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m)_z \\ r_z(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m)_x - r_x(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m)_z \\ -r_y(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m)_x + r_x(\Delta\hat{v}_1^m - \Delta\hat{v}_2^m)_y \end{bmatrix} \\ &= -K(RI^{-1}R^T) \begin{bmatrix} -r_z(\Delta\hat{p}_1^m)_y + r_z(\Delta\hat{p}_2^m)_y + r_y(\Delta\hat{p}_1^m)_z - r_y(\Delta\hat{p}_2^m)_z \\ r_z(\Delta\hat{p}_1^m)_x - r_z(\Delta\hat{p}_2^m)_x - r_x(\Delta\hat{p}_1^m)_z + r_x(\Delta\hat{p}_2^m)_z \\ -r_y(\Delta\hat{p}_1^m)_x + r_y(\Delta\hat{p}_2^m)_x + r_x(\Delta\hat{p}_1^m)_y - r_x(\Delta\hat{p}_2^m)_y \end{bmatrix} \\ &\quad - c(RI^{-1}R^T) \begin{bmatrix} -r_z(\Delta\hat{v}_1^m)_y + r_z(\Delta\hat{v}_2^m)_y + r_y(\Delta\hat{v}_1^m)_z - r_y\Delta\hat{v}_2^m)_z \\ r_z(\Delta\hat{v}_1^m)_x - r_z(\Delta\hat{v}_2^m)_x - r_x(\Delta\hat{v}_1^m)_z + r_x(\Delta\hat{v}_2^m)_z \\ -r_y(\Delta\hat{v}_1^m)_x + r_y(\Delta\hat{v}_2^m)_x + r_x(\Delta\hat{v}_1^m)_y - r_x(\Delta\hat{v}_2^m)_y \end{bmatrix}\end{aligned}$$

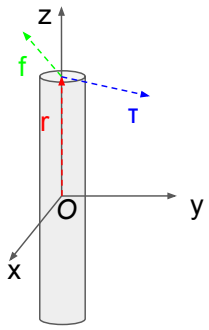


Figure 16: Torque in Rod Local Frame

Consider a uniform solid cylinder of mass M, radius R, height $2r$ as shown in Fig. 16. The moment of inertia of rod in local frame is

$$I = M \begin{bmatrix} \frac{1}{12}r^2 + \frac{1}{4}R^2 & 0 & 0 \\ 0 & \frac{1}{12}r^2 + \frac{1}{4}R^2 & 0 \\ 0 & 0 & \frac{1}{2}R^2 \end{bmatrix} = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{11} & 0 \\ 0 & 0 & I_{33} \end{bmatrix}$$

Since r is parallel to z axis in local frame, the local frame torque $\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{f}$ is perpendicular to z axis, i.e. $\boldsymbol{\tau}[2] = 0$.

$$\begin{aligned} \boldsymbol{\tau}_{world} &= I_{world}\boldsymbol{\alpha} \\ \boldsymbol{\tau}_{world} &= RIR^T\boldsymbol{\alpha} \\ R^{-1}\boldsymbol{\tau}_{world} &= I(R^T\boldsymbol{\alpha}) \\ \boldsymbol{\tau} &= I(R^T\boldsymbol{\alpha}) \\ I^{-1}\boldsymbol{\tau} &= R^T\boldsymbol{\alpha} \end{aligned}$$

Since $\boldsymbol{\tau}[2] = 0$, we can get that $I^{-1}\boldsymbol{\tau}[2] = 0$ as well no matter what I_{33} is. Thus we can set $I_{33} = I_{11}$. Then we have

$$RI^{-1}R^T = R(I_{11}^{-1}diag(1,1,1))R^T = I_{11}^{-1}RR^T = I_{11}^{-1}E$$

where E is a 3x3 identity matrix. Thus

$$\begin{aligned} \boldsymbol{\alpha} = -KI_{11}^{-1} &\begin{bmatrix} -r_z(\Delta\hat{p}_1^m)_y + r_z(\Delta\hat{p}_2^m)_y + r_y(\Delta\hat{p}_1^m)_z - r_y(\Delta\hat{p}_2^m)_z \\ r_z(\Delta\hat{p}_1^m)_x - r_z(\Delta\hat{p}_2^m)_x - r_x(\Delta\hat{p}_1^m)_z + r_x(\Delta\hat{p}_2^m)_z \\ -r_y(\Delta\hat{p}_1^m)_x + r_y(\Delta\hat{p}_2^m)_x + r_x(\Delta\hat{p}_1^m)_y - r_x(\Delta\hat{p}_2^m)_y \end{bmatrix} \\ - cI_{11}^{-1} &\begin{bmatrix} -r_z(\Delta\hat{v}_1^m)_y + r_z(\Delta\hat{v}_2^m)_y + r_y(\Delta\hat{v}_1^m)_z - r_y\Delta\hat{v}_2^m)_z \\ r_z(\Delta\hat{v}_1^m)_x - r_z(\Delta\hat{v}_2^m)_x - r_x(\Delta\hat{v}_1^m)_z + r_x(\Delta\hat{v}_2^m)_z \\ -r_y(\Delta\hat{v}_1^m)_x + r_y(\Delta\hat{v}_2^m)_x + r_x(\Delta\hat{v}_1^m)_y - r_x(\Delta\hat{v}_2^m)_y \end{bmatrix} \end{aligned}$$

Thus $\boldsymbol{\alpha}$ is a linear combination of $r_i(\Delta\hat{v}_a^m)_j$ and $r_i(\Delta\hat{p}_b^m)_j$ where $i, j \in \{x, y, z\}$ and $a, b \in \{1, 2\}$.

Appendix C. Data Efficiency Experiment on Koopman operator

Since all parameters are different, recent work compositional Koopman operator Li et al. (2019a) is no longer applicable, but our physics and topology-ware Koopman operator is still easy to scale up by adding more linear regressors. Here we apply six linear regressors for six rods. Koopman has larger error because of the non-triviality of Koopman basis functions design as we discussed in 4.2. Especially Koopman would also fail for training on a very small dataset, 0.01%, because the redundant basis functions may form a wrong linear function.

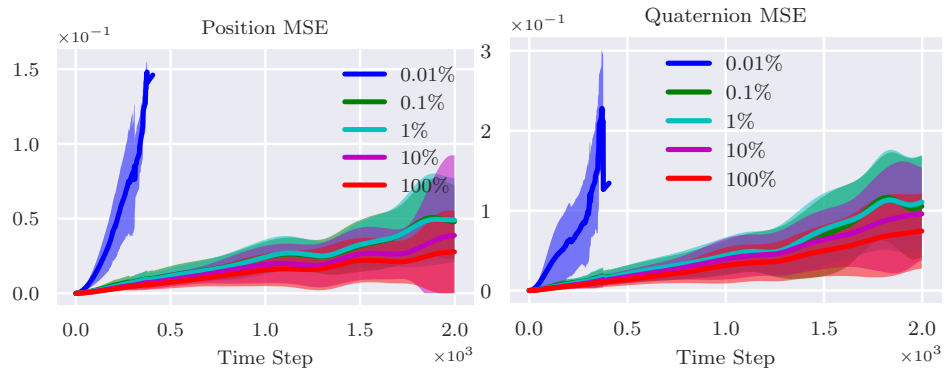


Figure 17: Position and quaternion error comparison of **Koopman operator** with different amount of training data in tensegrity model identification with *different* parameters for rods and springs

Appendix D. Model Generalization Experiment on 4,000 Time Steps Test Set

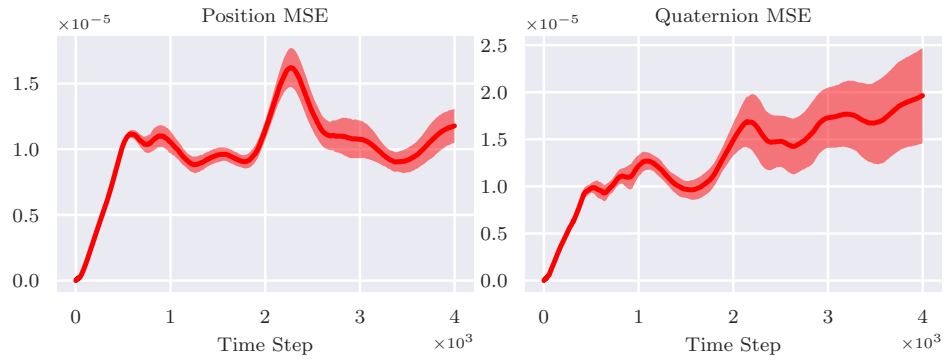


Figure 18: Position and quaternion error for 4,000 time steps test set of **Ours** tensegrity model identification with *different* parameters for rods and springs, adding an arbitrary force every 100 time steps.