CableRobotGraphSim: A Graph Neural Network for Modeling Partially Observable Cable-Driven Robot Dynamics

Nelson Chen[1] NELSON.CHEN@RUTGERS.EDU
William R. Johnson III[2] WILL.JOHNSON@YALE.EDU
Rebecca Kramer-Bottiglio[2] REBECCA.KRAMER@YALE.EDU
Kostas Bekris[1] KOSTAS.BEKRIS@CS.RUTGERS.EDU
Mridul Aanjaneya[1] MRIDUL.AANJANEYA@RUTGERS.EDU
[1]*Rutgers University,* [2]*Yale University*

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

General-purpose simulators have accelerated the development of robots. Traditional simulators based on first principles, however, typically require full-state observability or depend on parameter search for system identification. This work presents `CableRobotGraphSim`[1], a novel Graph Neural Network (GNN) model for cable-driven robots that aims to address shortcomings of prior simulation solutions. By representing cable-driven robots as graphs, with the rigid-bodies as nodes and the cables and contacts as edges, this model can quickly and accurately match the properties of other simulation models and real robots, while ingesting only partially observable inputs. Furthermore, trajectory rollout accuracy and inference speed are enhanced with prediction chunks, simultaneous multistep forward prediction. Accompanying the GNN model is a sim-and-real co-training procedure that promotes generalization and robustness to noisy real data. This model is further integrated with a Model Predictive Path Integral (MPPI) controller for closed-loop navigation, which showcases the model's speed and accuracy.

Keywords: Graph Neural Networks, Simulation, Cable-driven Robots, Model-based Control

1. Introduction

General-purpose simulators such as MuJoCo [Todorov et al. (2012)], IsaacSim [NVIDIA], and Drake [Tedrake and Team (2019)], together with GPU-based parallelization, have made simulation central to evaluating methods and training learned controllers. Common use cases include reinforcement learning (RL) prior to real-world deployment and generating expert demonstrations for imitation learning (IL). Because robot data are scarce and expensive to collect, simulation is increasingly used to bridge this gap, especially for unconventional platforms.

A large category of robot platforms corresponds to cable-driven robots, which include tensegrity robots that consist of rigid rods interconnected by flexible, actuated cables. A 3-bar tensegrity used in this work and shown in Fig. 1, is an instance of such structures. The properties of cable-driven robots enable applications in manipulation [Lessard et al. (2016)], locomotion [Sabelhaus et al. (2018)], morphing airfoils [Chen et al. (2020)], and spacecraft landing [Bruce et al. (2014)]. Their compliant and contact-rich dynamics, however, make modeling and control challenging, underscoring the need for improved simulation tools.

Traditional robot simulators have several shortcomings in this context. They face a sim-to-real gap due to oversimplified physics or incorrect system parameters, and identifying these parameters is often a manual, time-consuming process. This has motivated differentiable and learnable

1. https://nchen9191.github.io/cablerobotgraphsim/

simulators. Differentiable analytical simulators are data-efficient but prone to local minima and restrictive assumptions, while fully learned simulators are flexible but data-hungry. Graph neural network (GNN) simulators offer a favorable middle ground, encoding structural priors via graph representations.

Real-world data, however, is typically noisy, sparse, and only partially observable. In particular, partial observability can directly impede the use of traditional simulators that operate over a robot's full state. Soft robots are a prime example where full-state information is often not available due to their high number of degrees-of-freedom. For instance, prior work in soft robotics [Gao et al. (2024)] indicates that, online, there may be only access to several markers on the soft robot's surface to approximate its configuration. This is also an issue with tensegrity robots, such as the one in Fig. 1, where perception methods [Lu et al. (2023)] only capture the positions of end caps and are not able to estimate the rod's twist orientation (rotation about its center-axis) or instantaneous linear and angular velocities, providing only five out of 12 dimensions of a rod's state.

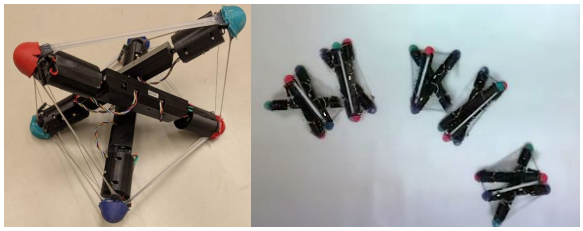

Figure 1: Left: Static, open-source 3-bar tensegrity platform. Right: The platform rolling clockwise.

`CableRobotGraphSim`, proposed in this work, is a fully learnable graph neural network (GNN) dynamics model for cable-driven robots that is effective under partial observability, while being data-efficient. The architecture features: (i) a fully learnable GNN for simulation stability, (ii) configuration-based node features with a recurrent block to support partial observability, (iii) a cable-edge decoder for direct actuation modeling, and (iv) multi-step forward, prediction chunks to accelerate rollouts. A sim-and-real co-training procedure further enhances generalization and robustness. The method is evaluated on tensegrity robots first in simulation and then in real-world settings. Comparisons are made against existing state-of-the-art differentiable tensegrity simulators [Wang et al. (2023); Chen et al. (2024)] in a sim2sim (MuJoCo as ground truth) and real2sim setup using data from a real 3-bar tensegrity. Additionally, the model is integrated into a Model Predictive Path Integral (MPPI) controller for a simulated 6-bar tensegrity. Finally, three ablation studies assess the contributions of key architectural components, the effect of the amount of co-training data, and the impact of the size of the prediction chunk on accuracy. In summary, the contributions of this work are the following:

- A novel GNN architecture for learning cable-driven dynamics given partial observations.
- Introduction of prediction chunks for enhancing both model accuracy and inference speed for short and long horizon trajectory rollouts.
- A sim-and-real co-training procedure that promotes model generalization and robustness to noise.
- Evaluations, in simulation and reality, of the GNN applied to modeling tensegrity robots.
- Integration of GNN-based modeling in MPPI-based control and demonstrations of tensegrity robots navigating obstacles using this controller.

2. Related Work

Data-driven, learnable simulation methods are increasingly used in robotics to reduce the manual effort of system identification. Approaches range from differentiable physics-based simulators [de Avila Belbute-Peres et al. (2018); Degrave et al. (2019); Werling et al. (2021)] that enable gradient computation, to purely neural network-based dynamics models [Raissi et al. (2019); Xu

et al. (2025); Baldan et al. (2025)]. Hybrid formulations [Heiden et al. (2020); Bianchini et al. (2023)] combine first-principles with learned components for better generalization. A particularly effective middle ground is **graph neural network (GNN)-based dynamics modeling** [Zhang et al. (2024); Rubanova et al. (2024)], which encodes structural priors through graph representations for data-efficient learning while maintaining flexibility.

For **tensegrity robots**, simulators have evolved from non-differentiable analytical ones [Friesen; Tadiparthi et al. (2019); Goyal et al. (2019); Paul et al. (2006); Caluwaerts et al. (2014)], which suffer from large sim-to-real gaps, to differentiable analytical simulators [Wang et al. (2021, 2022, 2023)] capable of gradient-based optimization but prone to local minima and restrictive assumptions. More recent learned simulators [Chen et al. (2024)] employ hybrid GNN approaches to model contact dynamics but require full-state observability. This work introduces a learned GNN-based simulator that addresses partial observability, while further reducing the sim-to-real gap.

In **model-based planning and control**, partial observability remains a key challenge for methods, such as model predictive control (MPC) [Katayama et al. (2023)], sampling-based variations [Williams et al. (2016)], and iterative re-planning [Tsianos and Kavraki (2008); Bekris and Kavraki (2007)]. Prior closed-loop controllers for tensegrities [Sabelhaus et al. (2021)] rely on simplified models or offline-generated motion primitives [Johnson et al. (2025)]. In contrast, this work demonstrates the first use of a fully learned dynamics model within an online sampling-based model predictive path integral (MPPI) controller.

Finally, insights from **robot imitation learning** [Chi et al. (2023); Black et al. (2025)], notably sim-and-real co-training [Maddukuri et al. (2025)] and action chunking [Zhao et al. (2023)], inspired part of this approach. Sim-and-real co-training aims to simultaneously use both simulation and real data to train an overall better model. Action chunking predicts a short, open-loop, temporal sequence of actions, instead of just a single action, per model run. These strategies are adapted here for dynamics learning to improve model accuracy, generalization, and simulation efficiency.

3. Approach

In cable-driven tensegrity robots, there are K rigid-bodies that are connected by a set of M cables, forming the robot's system topology. The state of the system is composed of the set of inner rigid-body states, where the k-th rigid-body at time t has state $\mathbf{X}_t^k = (\mathbf{P}_t^k, \mathbf{R}_t^k, \mathbf{V}_t^k, \mathbf{\Omega}_t^k)$. Here, $\mathbf{P}_t^k$ is the position, $\mathbf{R}_t^k$ is the orientation, $\mathbf{V}_t^k$ is the linear velocity, and $\mathbf{\Omega}_t^k$ is the angular velocity. Additionally, there is a hidden state, the rest lengths of the actuated cables, ℓ_t^{rest}, which changes as the attached motors are actuated. In the proposed simulator, shown in Fig. 2, the first step is mapping state $\mathbf{X}_t$ and controls $\mathbf{U}_t$ to the system's graph $\mathcal{G}_t$, made up of nodes $\mathcal{V}_t$ and edges $\mathcal{E}_t$.

$$\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t) \leftarrow F(\mathbf{X}_t, \mathbf{U}_t) \tag{1}$$

Where $F(\cdot)$ is the graph and feature generating function. Each node's state $\mathbf{x}_t$ at time t consists of their position $\mathbf{p}_t$ and linear velocity $\mathbf{v}_t$. The edges corresponding to the cables have a rest length state, ℓ_t^{rest}. In the second step, the GNN, parameterized by weights θ, is used to predict the changes in node velocity $\Delta\mathbf{v}_{t+1:t+n}$ and cable rest lengths $\Delta\ell_{t+1:t+n}^{rest}$, for n steps forward.

$$(\Delta\mathbf{v}_{t+1:t+n},\ \Delta\ell_{t+1:t+n}^{rest}) = \mathbf{GNN}_\theta(\mathcal{G}_t) \tag{2}$$

The third and last step is to numerically integrate the changes in velocity and rest lengths to compute the final state. In this work, the semi-explicit Euler integration scheme is chosen.

$$(\mathbf{v}_{t+n},\ \ell_{t+n}^{rest}) = (\mathbf{v}_t,\ \ell_t^{rest}) + \textstyle\sum_{i=t+1}^{t+n}(\Delta\mathbf{v}_i,\ \Delta\ell_i^{rest}) \tag{3}$$

$$\mathbf{p}_{t+n} = \mathbf{p}_t + \Delta t \textstyle\sum_{i=t+1}^{t+n} \mathbf{v}_i \tag{4}$$

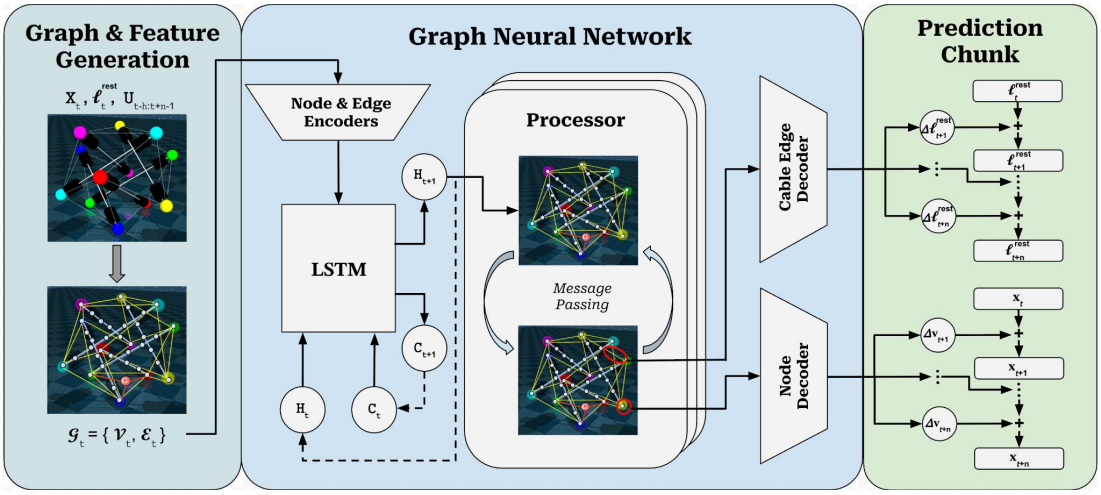

Figure 2: The three stages of a simulation step. **Left**: Graph and feature generation, where state $\mathbf{X}_t$, rest length ℓ_t^{rest}, control history $U_{t-h:t}$, and future controls $U_{t+1:t+n}$ form the graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. **Middle:** $\mathcal{G}_t$ is passed through MLP encoders and an LSTM block (with hidden state H_t and memory C_t) to produce H_{t+1}, C_{t+1}. H_{t+1} then undergoes L rounds of message passing, after which the latent node and cable-edge vectors are decoded into n velocity and rest-length changes. **Right:** The predicted chunk is integrated iteratively to advance ℓ_t^{rest} to ℓ_{t+n}^{rest} and $\mathbf{X}_t$ to $\mathbf{X}_{t+n}$.

3.1. GNN Model

Graph & feature generation: The graph is made up of body nodes $\mathcal{V}$, body edges $\mathcal{E}^{body}$, cable edges $\mathcal{E}^{cable}$, and contact edges $\mathcal{E}^{con}$. Each rigid body is decomposed into smaller primitive bodies such as spheres, cylinders, etc., and each is represented as a body node in the graph. Neighboring nodes that are part of the same rigid body are connected by a body edge. Here, the ground is represented by a single body node. Together, this forms $\mathcal{V} = \{V_i\}$ and $\mathcal{E}^{body} = \{E_{ij}^{body}\}$. Next, each cable is represented by edges, forming $\mathcal{E}^{cable} = \{E_{ij}^{cable}\}$. Contact edges are dynamically assigned to connect body nodes and the ground node if the distance between the two are below a user-defined threshold, forming $\mathcal{E}^{con} = \{E_{ij}^{con}\}$.

Two feature sets distinguish this model from prior GNN-based simulators. First, configuration-based features include (i) relative distance to a designated node in the robot graph, (ii) relative distance to the rigid body's CoM, and (iii) the body frame's z-axis. While many learned simulators [Allen et al. (2023)] only use velocity inputs to preserve translational equivariance, configuration-based features aid inference under partial observability by providing implicit state context. Second, for cable edges, a sequence of past h and future n control signals ($U_{t-h:t+n}$) are included to support both actuation learning and multi-step prediction.

GNN architecture: In prior work [Chen et al. (2024)], the cable and motor dynamics are computed analytically and combined with a GNN to learn contact dynamics, forming a hybrid approach. This work instead opts to be fully learnable. The authors have observed, in their experiments, that the hybrid approach becomes unstable when the inputs are only partial observations. This is likely due to the cables being a stiff linear system, where the cables create a negative feedback loop with the GNN. The stiff cables would greatly compound the error of the GNN, and in turn, would rapidly take the state out of the GNN's training distribution.

Following other established GNN architectures [Allen et al. (2023); Rubanova et al. (2024)], this work also employs the encode-process-decode structure. In the **GNN encoder**, there is similarly multi-layered perceptrons (MLP), per node and edge type, to encode the raw features to a larger

latent space. In this work, a long-short term memory (LSTM) block is added to the encoder to retain temporal hidden states H_i^0 and memory information C_i. The usage of hidden states and memory encodes history information and is a common method to deal with partial observability.

$$(H_i^0, C_i) = \mathbf{LSTM}(\ \mathbf{MLP}_{node}^{enc}(V_i), H_i, C_i) \tag{5}$$

$$\mathrm{E}_{ij}^{\epsilon,0} = \mathbf{MLP}_{\epsilon}^{enc}(E_{ij}^{\epsilon}) \quad \epsilon \in \{body, cable, contact\} \tag{6}$$

In the **GNN processor**, L message passes and node updates are executed. For the l^{th} message pass, a set of MLPs corresponding to each edge type, and an MLP for node updates, are used. First, messages are computed per edge based on the nodes it connects and its current latent vector. Second, these messages are aggregated via a sum operation across same edge types, and concatenated with other aggregated edge messages, per node. Last, the node's latent vector and the aggregated edges are concatenated, and passed to an MLP to update the node's updated latent vector:

$$\mathrm{E}_{ij}^{\epsilon,l} = \mathbf{MLP}_{l}^{MP}\left(H_i^{l-1}, H_j^{l-1}, \mathrm{E}_{ij}^{\epsilon,l-1}\right) \quad \epsilon \in \{body, cable, contact\} \tag{7}$$

$$H_i^l = \mathbf{MLP}_{l}^{update}\left(H_i^{l-1}, \sum \mathrm{E}_{ij}^{body,l}, \sum \mathrm{E}_{ij}^{cable,l}, \sum \mathrm{E}_{ij}^{con,l}\right) \tag{8}$$

After the processor, two **GNN decoders**, each an MLP, are used to take the latest node and cable edge latent vectors and map them to n-step velocity changes $\Delta\mathbf{v}_{t+1:t+n}$ and cable rest-length changes $\Delta\ell_{t+1:t+n}^{rest}$. The cable-edge decoder acts as a learnable cable actuation model by taking the latest cable-edge latent vector as input. This latent vector contains information about the properties of the cable, relevant control signals, and node effects on the cable motor.

$$\Delta\mathbf{v}_{t+1:t+n} = \mathbf{MLP}_{node}^{dec}\left(H_i^L\right) \qquad \Delta\ell_{t+1:t+n}^{rest} = \mathbf{MLP}_{cable}^{dec}\left(\mathrm{E}_{ij}^{cable,L}\right) \tag{9}$$

Prediction chunk: Traditional simulators use a timestep Δt to advance the state from t to $t+1$, requiring $T/\Delta t$ steps to simulate a rollout of duration T. Increasing Δt reduces the number of steps but degrades numerical accuracy. In contrast, the proposed method predicts n future steps from state $t+1$ to $t+n$ simultaneously, henceforth referred to as a prediction chunk. This approach (i) preserves fine timestep size discretization while achieving an n-fold speedup, (ii) provides dense supervision through intermediate ground truths, and (iii) enables efficient long-range propagation via a more compact computational graph. The decoders output $\Delta\mathbf{v}_{t+1:t+n}$ and $\Delta\ell_{t+1:t+n}^{rest}$, from which the final states are iteratively integrated using the chosen scheme.

Loss function: To train the GNN, as well as the internal cable actuation decoder, a single standard mean-squared-error (MSE) loss $\mathcal{L}$, over the predicted and ground-truth (GT) node velocity changes and rest length changes, averaged over the prediction chunk size n, is used:

$$\mathcal{L} = \frac{1}{n}\sum_{\tau=1}^{n}\left[\frac{\rho_n}{B_n}\sum_{i\in\mathcal{V}}\left(\Delta\mathbf{v}_{i,\tau}^{pred} - \Delta\mathbf{v}_{i,\tau}^{GT}\right)^2 + \frac{\rho_c}{B_c}\sum_{j\in\mathcal{E}^{cable}}\left(\Delta\ell_{j,\tau}^{rest,pred} - \Delta\ell_{j,\tau}^{rest,GT}\right)^2\right] \tag{10}$$

where B_n and B_c are the number of nodes and actuated cables in a training batch, respectively, and ρ_n and ρ_c are user-defined weights balancing the two.

3.2. Sim and real data co-training

For real robots, their data are often noisy and sparse. This is especially true for unconventional robots that do not have large communities or resources to collect abundant amounts of data. However, neural networks are very data hungry, causing learned models to be prone to overfitting and to be sensitive to noise. This is why differentiable analytical simulators are the go-to choice due to the physics principles embedded in the model. A line of research that recently emerged in robot

learning is the idea of co-training [Maddukuri et al. (2025)] control policies with a mix of simulation and real data. For this work, sim-and-real co-training has been adapted for learning dynamics. The inclusion of simulation data provides the GNN with a foundational source of physics knowledge, making it robust to noise in the real data.

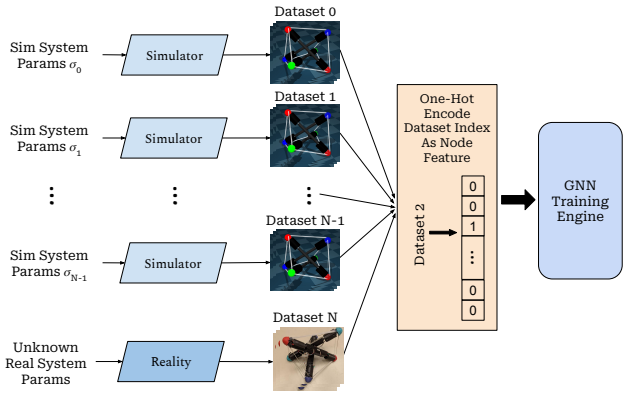

Figure 3: Sim-and-real co-training. Multiple simulation datasets are generated by varying system parameters, and are pooled with a real dataset of unknown system parameters. Data points are distinguished by a one-hot encoding node feature of the dataset enumerated index.

To generate the simulation co-training data, select a simulator or simulation platform that can roughly model the system of interest, but cannot easily perform system identification. Next, identify a set of relevant system parameters to vary and generate multiple parameter sets either by grid or random sampling. Then, take the controls and initial states of the real robot trajectories, and run rollouts in simulation for each set of system parameters σ_i to generate dataset $\mathcal{D}_i$.

Because the real data's system parameters are not actually known, the simulation datasets' system parameters cannot be used by the GNN. Consequently, a naive mixing of all data sources would result in the GNN becoming confused, i.e., the same input state results in different outputs. To combat this, the simulation datasets are enumerated from 0 to $N-1$, and the real dataset is enumerated as the N^{th} dataset. Finally, this information is passed to the GNN as a one-hot encoded node feature. This process can be seen in Fig. 3.

3.3. MPPI integration

To demonstrate the effectiveness of the proposed GNN model, this paper integrates the full GNN model into an MPPI controller. Furthermore, the MPPI controller also acts as a diverse data generator that is used to further train the GNN model, which subsequently, improves the controller. This can be repeated for multiple iterations to finetune both the model and controller to a particular task.

Background MPPI is a sampling-based MPC method for stochastic optimal control. Given a state-transition model f_{sim}, MPPI samples K noisy input control sequences $\{U^k\}$. Next, K trajectories $\{\mathcal{T}^k\}$ are generated by simulating $\{U^k\}$ and initial state X_0 with f_{sim} over a short time horizon T.

$$\mathcal{T}^k = \left[\mathbf{X}_0, f_{sim}\left(\mathbf{X}_0, u_0^k\right), ..., f_{sim}\left(\mathbf{X}_{T-1}, u_{T-1}^k\right)\right] \tag{11}$$

With a cost function $\mathcal{C}$ to be minimized, importance sampling weights w^k can be computed with an inverse exponential:

$$w^k = \frac{1}{\eta}\exp\left(-\frac{1}{\beta}(\mathcal{C}\left(\mathcal{T}^k\right) - \mu)\right), \quad \sum w^k = 1 \tag{12}$$

where η is a normalization factor, β is the inverse temperature parameter, and $\mu = \min_k C^k$. Finally, the optimal control sequence U^* is approximated with the weighted average

$$U^* = \sum w^k V^k \tag{13}$$

The first control of U^* is then executed and the MPPI process is repeated again to compute the next optimal control sequence. The GNN model and MPPI controller naturally work well together, since the GNN can natively simulate all rollouts in parallel on GPUs. In this work, f_{sim} is substituted with the proposed trained GNN model. The cost function used is

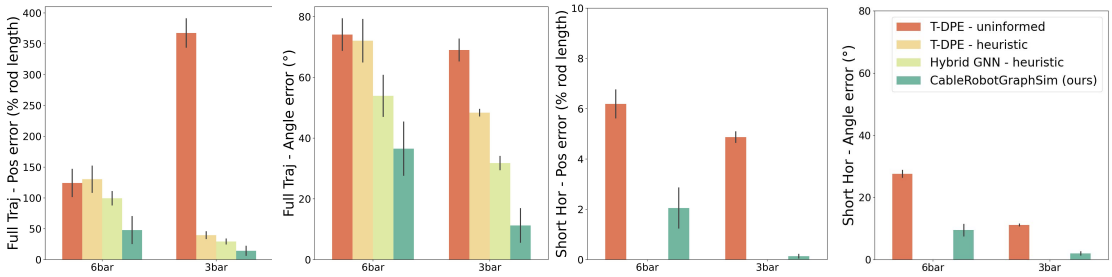

Figure 4: Sim2sim evaluation for full trajectories and short horizons, as well as a 3-bar and 6-bar tensegrity.

$$\mathcal{C}(\mathbf{X}_t) = \alpha_1 d_{L_1}(\mathbf{X}_t) + \alpha_2 / d_{collision}(\mathbf{X}_t) \tag{14}$$

where $d_{L_1}(\cdot)$ is the *obstacle-aware* 2D manhattan distance of the current center of mass (CoM) to the goal, $d_{collision}(\cdot)$ is the minimum distance between the robot end caps and the obstacles, and α_1 and α_2 are the respective cost weights.

The MPPI controller serves as an effective data generator due to its stochastic sampling and ability to produce meaningful locomotion. This is advantageous for learned models, which require broad coverage of the state–control space to avoid overfitting and bias. Previously, tensegrity control relied on hand-crafted gaits [III et al. (2025)] or random actions that failed to induce motion. In this work, the MPPI controller operates in a loop: (i) train a GNN model, (ii) execute the controller with the updated model over a task, and (iii) record and add resulting trajectories to the dataset. This process is repeated across iterations while tracking model accuracy and task-level metrics such as success rate and completion time.

4. Experimental Results

For the sim2sim and real2sim experiments, the proposed approach is compared against Chen et al. (2024), a hybrid analytical and GNN tensegrity simulator, and Wang et al. (2023), a tensegrity differentiable physics engine, henceforth referred as Hybrid GNN and T-DPE, respectively. Each baseline adapts two cable attachment variants: a heuristic scheme and an uninformed scheme. The heuristic attachment, developed in prior work Wang et al. (2023), specifies how to attach cables to the surface of the rod end caps when the platform is at rest. The uninformed attachment places cables at the center of the end caps, simplifying the model by removing the need for the missing rod twist orientations. Because the heuristic attachment is only valid at rest, its corresponding baselines cannot be used in an online, closed-loop setting, which requires the simulator to run from any valid non-rest state. All methods are trained and tested on a single machine with an Nvidia RTX 4090 GPU and an AMD Ryzen 7950 16-core 4.5 GHz CPU, and all simulators were built and executed using PyTorch and PyGeometric.

Evaluation metrics: For the following evaluation results, there are several common metrics used. These metrics are CoM MSE $\mathbf{e}_{pos}$ (normalized by rod length L_{rod}), and angular error $\mathbf{e}_{rot}$ over the rods' center-axis $\hat{\mathbf{r}}$, both averaged over the trajectory length T. Formally, the error metrics are

$$\mathbf{e}_{pos} = \frac{1}{L_{rod}T} \sum \left(\mathbf{P}_i^{GT} - \mathbf{P}_i^{pred} \right)^2, \quad \mathbf{e}_{rot} = \frac{1}{T} \sum \cos^{-1} \left(\hat{\mathbf{r}}_i^{GT} \cdot \hat{\mathbf{r}}_i^{pred} \right) \tag{15}$$

These metrics are applied in two different settings, full trajectory $(\mathbf{e}_{pos}^{full}, \mathbf{e}_{rot}^{full})$ and short horizon $(\mathbf{e}_{pos}^{SH}, \mathbf{e}_{rot}^{SH})$. A full trajectory evaluation compares a full simulation rollout against the ground

truth. The short horizon evaluation samples N_{SH} random initial states from a trajectory, rolls the samples out for a short horizon T_{SH}, and compares the simulation rollouts against the ground-truth rollouts. The short horizon metrics give insight into how the model would perform in receding horizon controllers like MPPI.

Simulation-to-Simulation: In the sim2sim experiments, there are six different simulation datasets generated with MuJoCo as the ground-truth environment. Each is generated with a different set of system parameters, specifically the coefficient of friction and the contact solver's stiffness parameter. Each dataset has nine trajectories ranging from 30 to 60 seconds each, with a train-test split of six to three trajectories, respectively. This was done both for a 3-bar tensegrity and a 6-bar tensegrity. The 3-bar used human-engineered gait patterns, while the 6-bar used a mix of passive throws, random controls, and MPPI-based trajectories. The proposed method and the baselines are trained and tested on each dataset, and the mean and standard deviation of the errors are shown in Fig. 4. As a reminder, the heuristic variants can only be used from the initial rest states of each trajectory, so they cannot be applied in the short horizon setting. Moreover, the uninformed Hybrid GNN baseline resulted in unstable behavior and diverged for all metrics, so numerical evaluation is omitted. Fig. 4 shows that the proposed method yielded strong performance gains over the baselines.

Model	Full Traj Pos Error (%)	Full Traj Rot Error (°)	Short Horizon Pos Error (%)	Short Horizon Rot Error (°)
Hybrid GNN - uninformed	Unstable	Unstable	Unstable	Unstable
Hybrid GNN - heuristic	91.12	35.29	N/A	N/A
T-DPE - uninformed	465.59	60.32	9.59	28.76
T-DPE - heuristic	102.76	38.48	N/A	N/A
CableRobotGraphSim (ours)	**48.95**	**26.53**	**3.95**	**7.031**

Table 1: Real2sim evaluation results on a real 3-bar tensegrity platform.

Real-to-Simulation: In the real2sim experiments, the goal is to best match the simulator to reality. A dataset of nine trajectories is generated by running the human-engineered gait patterns with a real 3-bar tensegrity. The data are recorded using a tensegrity perception algorithm [Lu et al. (2023)], which only captures the rods' end cap positions, missing the rods' twist orientation and instantaneous velocities. As shown in Table 1, the proposed method shows strong relative performance compared to the alternatives. The real2sim errors are overall larger than the sim2sim errors. This is to be expected, since simulation dynamics are a simplified model of reality. Another source of error is that the ground truth used here is state estimation from the perception algorithm that may be noisy, which would also be reflected in the real2sim results.

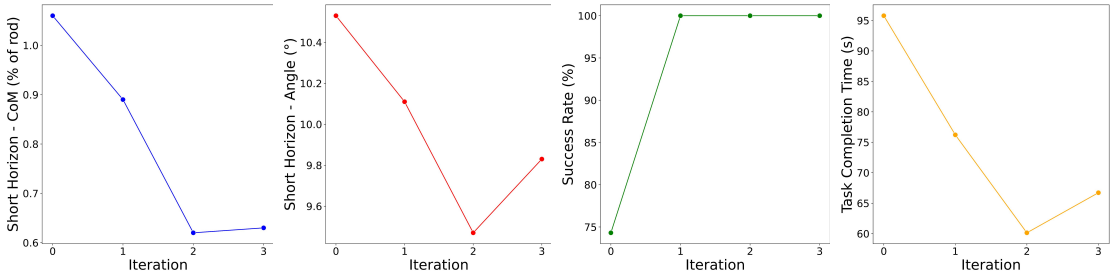

Figure 5: Running MPPI controller in a loop of model training, task execution, and new data generation for model retraining over three iterations. Positional and rotational errors are evaluated on a pooled test set from all iterations. Success rate and task completion time are measured per iteration.

Model	Full Traj Pos Error (%)	Full Traj Rot Error (°)	Short Horizon Pos Error (%)	Short Horizon Rot Error (°)
(baseline) Hybrid GNN	Unstable	Unstable	Unstable	Unstable
(i) Fully Learned GNN (F-GNN)	20.6 ± 26.14	47.1 ± 27.03	0.11 ± 0.04	4.91 ± 0.99
(ii) F-GNN + Config Feats (FC-GNN)	1.15 ± 1.13	15.43 ± 15.54	0.1 ± 0.04	4.6 ± 1.21
(iii) FC-GNN + Recurrent (FCR-GNN)	$\mathbf{1.04 \pm 1.3}$	$\mathbf{7.61 \pm 4.72}$	$\mathbf{0.06 \pm 0.02}$	$\mathbf{3.24 \pm 0.88}$
FCR-GNN + analytical motor	4.49 ± 2.21	31.18 ± 7.22	0.1 ± 0.04	5.14 ± 0.66
(iv) FCR-GNN + learned motor (ours)	$\mathbf{1.1 \pm 0.9}$	$\mathbf{14.02 \pm 7.84}$	$\mathbf{0.06 \pm 0.02}$	$\mathbf{3.11 \pm 0.48}$

Table 2: GNN architecture component ablation study. The first four rows compare GNN components using ground-truth motor outputs. The last two rows compare an analytical vs. a learned motor model.

MPPI and controller-in-the-loop data collection: The GNN is integrated with an MPPI controller as an internal state-transition model. The controller is also used for active data collection. Experiments are conducted on a simulated 6-bar tensegrity navigating an obstacle course (Fig. 6). An initial model (iteration 0) is trained using trajectories from random controls and passive drops, followed by three data collection and retraining iterations. For each iteration, success rate (goal reached within 120 s) and task completion time are measured. After the final iteration, all test data are aggregated into a single evaluation set, and each iteration's model is assessed for short-horizon, positional and rotational error (Fig. 5). Both the model and controller improve over iterations, with slight regression at iteration 3, likely due to model capacity. The final model, when deployed with the MPPI in a previously unseen maze (Fig. 7) can complete the navigation task successfully.

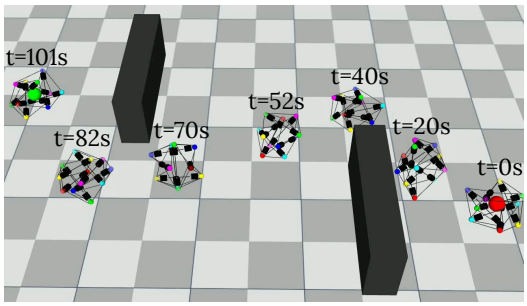

Figure 6: S-shaped obstacle course for navigating from the red to the green sphere.

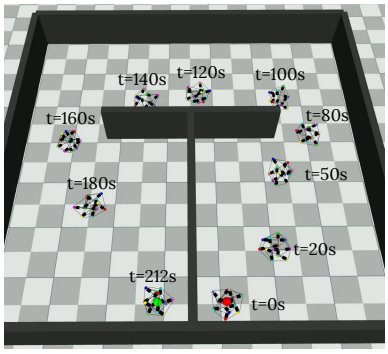

Figure 7: Maze map for navigating from the red to the green sphere.

Ablations: Three ablations assess the contributions of individual components. The first (Table 2) incrementally transitions the Hybrid GNN baseline to the proposed architecture by: (i) converting it to a fully learned GNN, (ii) adding configuration-based features, (iii) incorporating a recurrent block, and (iv) introducing an actuation model (analytical or learned). Experiments (i)-(iii) use ground-truth actuation outputs, while (iv) compares analytical versus learned actuation given control inputs. On sim2sim data, the full proposed setup achieves the best performance without requiring ground-truth actuation.

The second ablation (Fig. 8) varies the number of simulation datasets used for sim-and-real co-training, evaluating each resulting model on real test data. Models trained without simulation data were unstable, likely due to the sparsity and noise of real data, and are omitted. Positional errors improve with up to four simulation datasets and then degrade, suggesting that simulation data acts as a regularizer that becomes too strong in excess. Rotational errors plateau or slightly increase, likely because the position-based

loss weighs positional accuracy more heavily, as the angular component is bounded by the robot's geometry.

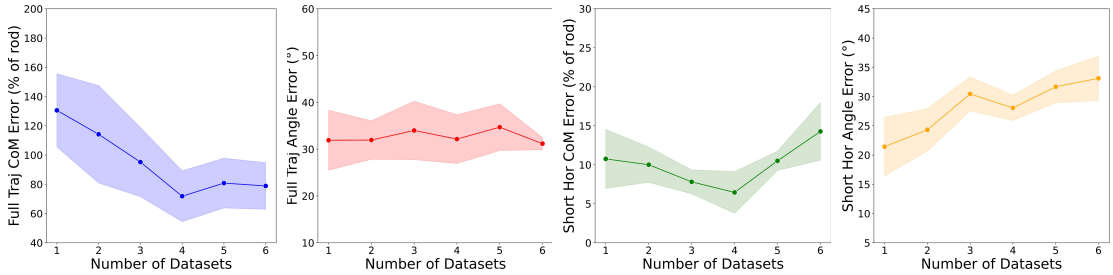

Figure 8: Sim-and-real co-training ablation study for varying number of simulation datasets.

The third ablation examines the effect of the number of forward prediction steps, varying from one to 16 in increments of two. As shown in Fig. 9, performance peaks at six steps. This is unexpected, since shorter horizons are typically easier to predict. A likely explanation is that learned models suffer from compounding errors as each auto-regressive step moves the state further out of distribution, so increasing steps per prediction reduces total inferences per rollout. Beyond six steps, accuracy degrades again. While model size is not explored here, evidence from visual-language-action (VLA) models suggests larger models could support longer horizons, hinting at an optimal pairing of model size and prediction length for rollout efficiency.

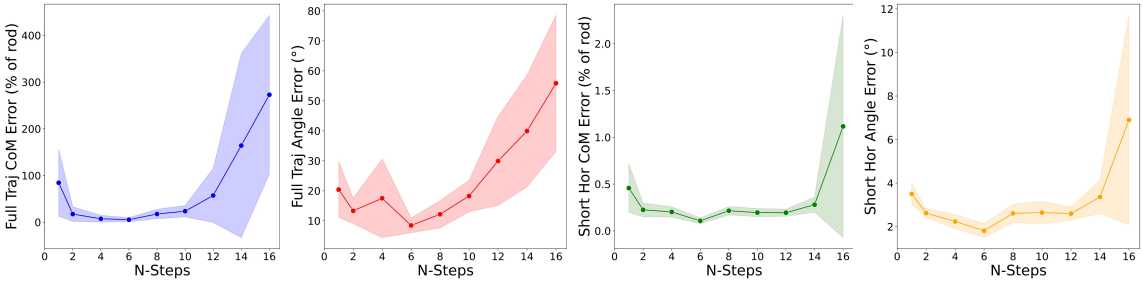

Figure 9: Prediction chunk size ablation over the number of forward time steps per simulation call.

5. Discussion

`CableRobotGraphSim` demonstrates the flexibility of learned simulators, while addressing challenges related to noise and data sparsity. By combining a graphical representation, a novel GNN architecture for cable-driven robots, and a sim-and-real co-training scheme, the model effectively handles partial observability as well as limited and lower quality data. Experiments on cable-driven tensegrity platforms show strong performance against baselines in both sim2sim and real2sim settings. The model is further integrated with an MPPI controller, which allows the successful completion of navigation tasks while generating additional training data.

At the same time, the current model assumes flat terrain. Future work will extend it to uneven surfaces for more complex navigation. A natural next step is to apply similar GNN-based frameworks to other robotic systems that can benefit from a learned graphical representation. This will also allow a deeper understanding of the benefits of sim-and-real co-training and multi-step prediction chunking for robot modeling purposes.

Acknowledgments

Nelson Chen and Mridul Aanjaneya were supported in part by the National Science Foundation (NSF) under awards CCF-2110861, IIS-2132972, IIS-2238955, and CCF-2312220 as well as a research gift from Red Hat, Inc. William R. Johnson III and Rebecca Kramer-Bottiglio were supported by the NSF under award IIS-195522. Kostas Bekris was supported in part by NSF under award IIS-1956027.

References

Kelsey R Allen, Tatiana Lopez Guevara, Yulia Rubanova, Kim Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1157–1167. PMLR, 14–18 Dec 2023.

Giacomo Baldan, Qiang Liu, Alberto Guardone, and Nils Thuerey. Flow matching meets pdes: A unified framework for physics-constrained generation, 2025. URL https://arxiv.org/abs/2506.08604.

Kostas E. Bekris and Lydia E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 704–710, 2007. doi: 10.1109/ROBOT.2007.363069.

Bibit Bianchini, Mathew Halm, and Michael Posa. Simultaneous learning of contact and continuous dynamics. In *Conference on Robot Learning (CoRL)*, November 2023.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Jonathan Bruce, Andrew P. Sabelhaus, Yang zhen Chen, Dizhou Lu, Kyle J. Morse, Sophie Milam, Ken Caluwaerts, Alice M. Agogino, and Vytas SunSpiral. Superball : Exploring tensegrities for planetary probes. 2014.

Ken Caluwaerts, Jérémie Despraz, Atil Iscen, Andrew Sabelhaus, Jonathan Bruce, Benjamin Schrauwen, and Vytas Sunspiral. Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of the Royal Society, Interface / the Royal Society*, 11, 09 2014.

Muhao Chen, Jiacheng Liu, and Robert Skelton. Design and control of tensegrity morphing airfoils. *Mechanics Research Communications*, 103:103480, 01 2020.

Nelson Chen, Kun Wang, William R. Johnson III, Rebecca Kramer-Bottiglio, Kostas Bekris, and Mridul Aanjaneya. Learning differentiable tensegrity dynamics using graph neural networks, 2024. URL https://arxiv.org/abs/2410.12216.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

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 S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 2019.

Jeffrey Michael Friesen. Tensegrity matlab objects. URL https://github.com/Jfriesen222/Tensegrity_MATLAB_Objects.

Junpeng Gao, Mike Y. Michelis, Andrew Spielberg, and Robert K. Katzschmann. Sim-to-real of soft robots with learned residual physics. *IEEE Robotics and Automation Letters*, 9(10):8523–8530, October 2024. ISSN 2377-3774. doi: 10.1109/lra.2024.3446287. URL http://dx.doi.org/10.1109/LRA.2024.3446287.

Raman Goyal, Muhao Chen, Manoranjan Majji, and Robert E. Skelton. Motes: Modeling of tensegrity structures. *Journal of Open Source Software*, 4(42):1613, 2019.

Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S. Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9474–9481, 2020.

William R. Johnson III, Xiaonan Huang, Shiyang Lu, Kun Wang, Joran W. Booth, Kostas Bekris, and Rebecca Kramer-Bottiglio. Impact-resistant, autonomous robots inspired by tensegrity architecture, 2025. URL https://arxiv.org/abs/2501.15078.

William R. Johnson, Patrick Meng, Nelson Chen, Luca Cimatti, Augustin Vercoutere, Mridul Aanjaneya, Rebecca Kramer-Bottiglio, and Kostas E. Bekris. An open-source, reproducible tensegrity robot that can navigate among obstacles, 2025. URL https://arxiv.org/abs/2511.05798.

Sotaro Katayama, Masaki Murooka, and Yuichi Tazaki. Model predictive control of legged and humanoid robots: models and algorithms. *Advanced Robotics*, 37(5):298–315, 2023. doi: 10.1080/01691864.2023.2168134. URL https://doi.org/10.1080/01691864.2023.2168134.

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, 2016.

Shiyang Lu, William R. Johnson, Kun Wang, Xiaonan Huang, Joran Booth, Rebecca Kramer-Bottiglio, and Kostas Bekris. 6n-dof pose tracking for tensegrity robots. In *Robotics Research*, pages 136–152, Cham, 2023. Springer Nature Switzerland.

Abhiram Maddukuri, Zhenyu Jiang, Lawrence Yunliang Chen, Soroush Nasiriany, Yuqi Xie, Yu Fang, Wenqi Huang, Zu Wang, Zhenjia Xu, Nikita Chernyadev, Scott Reed, Ken Goldberg, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Sim-and-real co-training: A simple recipe for vision-based robotic manipulation, 2025. URL https://arxiv.org/abs/2503.24361.

NVIDIA. Isaac Sim. URL https://github.com/isaac-sim/IsaacSim.

Chandana Paul, Francisco J. Valero Cuevas, and Hod Lipson. Design and control of tensegrity robots for locomotion. *IEEE Transactions on Robotics*, 22:944–957, 2006.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991.

Yulia Rubanova, Tatiana Lopez-Guevara, Kelsey R. Allen, William F. Whitney, Kimberly Stachenfeld, and Tobias Pfaff. Learning rigid-body simulators over implicit shapes for large-scale scenes and vision. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 125809–125838. Curran Associates, Inc., 2024. doi: 10.52202/079017-3997. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e3abc125ecacb71786cefb9f67b08c5d-Paper-Conference.pdf.

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

Andrew P. Sabelhaus, Huajing Zhao, Edward L. Zhu, Adrian K. Agogino, and Alice M. Agogino. Model-predictive control with inverse statics optimization for tensegrity spine robots. *IEEE Transactions on Control Systems Technology*, 29(1):263–277, 2021. doi: 10.1109/TCST.2020.2975138.

Vaishnav Tadiparthi, Shao-Chen Hsu, and Raktim Bhattacharya. Stedy: Software for tensegrity dynamics. *Journal of Open Source Software*, 4(33):1042, 2019.

Russ Tedrake and Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

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, 2012.

Konstantinos I. Tsianos and Lydia E. Kavraki. Replanning: A powerful planning strategy for hard kinodynamic problems. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1667–1672, 2008. doi: 10.1109/IROS.2008.4650965.

Kun Wang, Mridul Aanjaneya, and Kostas Bekris. Sim2sim evaluation of a novel data-efficient differentiable physics engine for tensegrity robots. pages 1694–1701, 09 2021.

Kun Wang, Mridul Aanjaneya, and Kostas Bekris. A recurrent differentiable engine for modeling tensegrity robots trainable with low-frequency data. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3230–3237, 2022.

Kun Wang, William R. Johnson, Shiyang Lu, Xiaonan Huang, Joran Booth, Rebecca Kramer-Bottiglio, Mridul Aanjaneya, and Kostas Bekris. Real2sim2real transfer for control of cable-driven robots via a differentiable physics engine. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2534–2541, 2023.

Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and Karen Liu. Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints. 07 2021.

Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016. doi: 10.1109/ICRA.2016.7487277.

Jie Xu, Eric Heiden, Iretiayo Akinola, Dieter Fox, Miles Macklin, and Yashraj Narang. Neural robot dynamics, 2025. URL https://arxiv.org/abs/2508.15755.

Kaifeng Zhang, Baoyu Li, Kris Hauser, and Yunzhu Li. Adaptigraph: Material-adaptive graph-based neural dynamics for robotic manipulation, 2024. URL https://arxiv.org/abs/2407.07889.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL https://arxiv.org/abs/2304.13705.