Real2Sim2Real Transfer for Control of Cable-driven Robots via a Differentiable Physics Engine

Kun Wang[1], William R. Johnson III[2], Shiyang Lu[1], Xiaonan Huang[2],
Joran Booth[2], Rebecca Kramer-Bottiglio[2], Mridul Aanjaneya[1], and Kostas Bekris[1]

***Abstract*— Tensegrity robots, composed of rigid rods and flexible cables, exhibit high strength-to-weight ratios and significant deformations, which enable them to navigate unstructured terrains and survive harsh impacts. They are hard to control, however, due to high dimensionality, complex dynamics, and a coupled architecture. Physics-based simulation is a promising avenue for developing locomotion policies that can be transferred to real robots. Nevertheless, modeling tensegrity robots is a complex task due to a substantial sim2real gap. To address this issue, this paper describes a Real2Sim2Real (`R2S2R`) strategy for tensegrity robots. This strategy is based on a differentiable physics engine that can be trained given limited data from a real robot. These data include offline measurements of physical properties, such as mass and geometry for various robot components, and the observation of a trajectory using a random control policy. With the data from the real robot, the engine can be iteratively refined and used to discover locomotion policies that are directly transferable to the real robot. Beyond the `R2S2R` pipeline, key contributions of this work include computing non-zero gradients at contact points, a loss function for matching tensegrity locomotion gaits, and a trajectory segmentation technique that avoids conflicts in gradient evaluation during training. Multiple iterations of the `R2S2R` process are demonstrated and evaluated on a real 3-bar tensegrity robot.**

I. Introduction

Tensegrity robots are actuated systems composed of rigid struts (rods) and flexible elements (cables) connected to form lightweight, deformable structures. Their natural compliance makes them adaptable and safe robots that are well-suited for many applications, such as manipulation [1], locomotion [2], morphing airfoils [3], and spacecraft landing [4].

At the same time, tensegrity robots are difficult to accurately model and control due to their many degrees of freedom (DoF) and complex dynamics [5]. The difficulty in modeling has led some researchers to propose model-free solutions for learning control [6], but these strategies still encounter challenges since they require a large amount of training data, and collecting trajectories from tensegrities is time-consuming, cumbersome, and expensive. The authors have previously introduced a differentiable, but modular and explainable, physics engine [7]–[9] as a data-efficient tool to identify physical parameters and develop locomotion policies for tensegrity robots. Yet, the transfer of simulated policies to real hardware is typically impeded by the so-called simulation-to-reality (sim2real) gap [10], [11]. Sim2real transfer can be improved by tuning the simulation to minimize differences between predicted and real robot trajectories for the same controls.

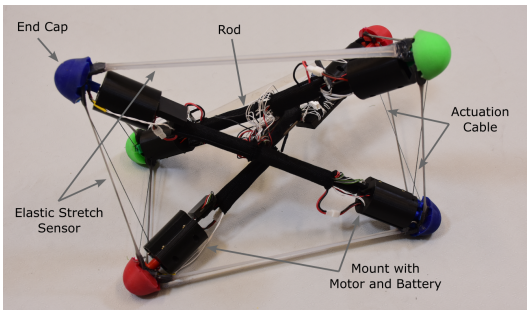


Fig. 1: The target platform: a 3-bar prismatic tensegrity robot with 9 stretch sensors. The 6 short cables are contracted and extended by the motors in the 3D-printed housings on each rod.

To overcome the sim2real gap, many methods are applied [11]–[15], however, the robot is assumed to be rigid [16] as a prior. Systems that include compliance, such as tensegrity robots, are less frequently targeted, as compliance makes closing the sim2real gap more challenging. Only recently have Real2Sim2Real (`R2S2R`) frameworks been developed that focus on deformable elements (e.g., a system that manipulates deformable cables [17]).

This work applies the principle of `R2S2R` to the control of deformable robots composed of both rigid and soft elements. More specifically, this work introduces an `R2S2R` pipeline for tensegrity robots, through which a policy learned on a differentiable engine is transferred to a 3-bar tensegrity robot (Fig. 1) by first training the engine with data from the real robot. This paper contributes:

- A complete pipeline for identifying the parameters of a differentiable simulator from real tensegrity robot trajectories, generating locomotion policies in the simulator, and transferring the policies back to the real robot.
- A method to compute non-zero gradients at contact points to enable efficient learning of contact parameters in the optimization step of the engine's identification process.
- A loss function and trajectory segmentation strategy to avoid conflicts in gradient direction during training under noisy observations. Non-convex trajectories lead to gradients with opposite directions at different time steps. Thus, the trajectory is segmented into convex segments to ensure the gradient directions computed are aligned.

This work has been supported by NSF Robust Intelligence award 1956027. The opinions expressed here are those of the authors and do not reflect the positions of the sponsor. [1]Computer Science, Rutgers University, NJ 08901, USA. Email: {kun.wang2012, shiyang.lu, mridul.aanjaneya, kostas.bekris}@rutgers.edu. [2]Mech. Eng. & Material Sc., Yale University, New Haven, CT, USA. Email: {will.johnson, xiaonan.huang, joran.booth, rebecca.kramer}@yale.edu.

II. Related Work

Sim2real transfer has been applied in autonomous underwater vehicles [18], drones [19], [20], muscles [21], quadruped robots [22], soft robots [23]–[25], and grasping manipulators [26]. To the best of the authors' knowledge, this is the first work that mitigates the sim2real gap for a tensegrity robot.

Differentiable physics has been actively applied to system identification. Compared to artificial evolution approaches (e.g., genetic algorithms, particle swarm optimizations, and covariance matrix adaptation evolution strategies) and domain-randomization methods, gradient-based methods like differential physics are data-efficient and can lead to faster convergence for complex robot systems [27], [28]. We previously developed a differentiable engine for tensegrity robots [7]–[9], although prior work has been limited to only sim2sim transfer and never demonstrated on a real tensegrity robot.

Prior work on tensegrity locomotion [29]–[31] has achieved complex behaviors, sometimes on uneven terrain, using the NASA Tensegrity Robotics Toolkit (NTRT) simulator [32], which was manually tuned to match a real platform [33], [34]. Many prior approaches use reinforcement learning (RL) to learn policies given sparse inputs, which can be provided by onboard sensors [30] and aim to address the large data requirements of RL [31], including by training in simulation. Simulated locomotion, however, is hard to replicate on a real platform, even after hand-tuning, which emphasizes the importance of training a simulator that can produce policies that can be transferred to a real system. A website accompanying this paper with videos and additional evaluation is available[1].

III. Robot Design

This work demonstrates and evaluates the R2S2R pipeline on the untethered 3-bar prismatic tensegrity robot shown in Fig. 1. The tensegrity robot has a rod length of 36 cm, and it is driven by motors that extend and contract its cables to shift its center of mass. The six short cables (three on each side) are actuated by the motors while the three longer tendons in the middle are passive elastic elements. These passive tendons double as stretch sensors, and there are also six sensor tendons in parallel with the six actuated cables. The design and characterization of the stretch sensors are detailed in previous work [35]. Each sensor is calibrated individually by fitting a linear model to map capacitance measurements to corresponding lengths [36]. The stretch sensors are used both for feedback control and for pose estimation, as described in section IV-B.

IV. Real2Sim2Real Pipeline

The following discussion describes the various components of the proposed R2S2R pipeline as outlined in Fig. 2.

[1]An appendix with addtional material and accompanying multimedia can be found at: https://sites.google.com/view/sim2real

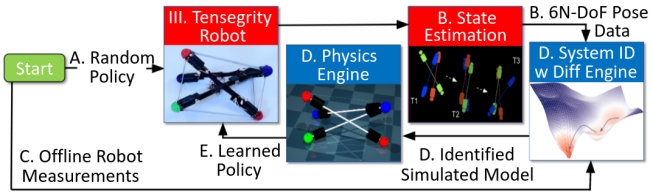


Fig. 2: R2S2R pipeline. A randomly generated policy is executed first on the robot. An overhead RGB-D camera captures the trajectory to estimate the robot's state at each frame. Internal robot parameters are then identified with a differentiable physics engine given robot states and offline measurements. Given the identified model, new policies are generated and executed on the robot. The process can be repeated to further close the sim2real gap. Red blocks correspond to real-robot tasks; blue blocks correspond to simulation tasks. Labels correspond to sections in the text.

A. Random Policy and Real World Execution

The first step in the R2S2R pipeline is to generate a random policy and execute it on the real robot. The policy is defined as a list of robot cable lengths (0 means fully contracted and 1 means fully extended) that represent a series of target robot shapes. To transition between these high-level shapes, a PID controller extends or contracts the actuated cables by sending low-level control signals based on the feedback from the robot cable length sensors until the robot reaches the next target shape. All policies start from the robot rest state where the robot sits on the ground with the six actuated cables fully extended, as shown in Fig. 1.

B. State Estimation

The next step is to estimate the robot states, i.e., the position and orientation of each rod, at each time frame. We adopt a state estimation method [37] designed for tensegrity robots. This method tracks the 6-DoF pose of each rod given a multi-modal input sequence of RGB images, depth images, and measured cable lengths from the onboard stretch sensors. The tracking method incorporates a variety of physical constraints, and it performs state estimation with high accuracy and robustness to self-occlusions. Further, RANSAC [38] is used for ground detection so that contacts between end caps and the ground could be found(Section IV-D).

end cap	mass	10.5
	radius	17.5
rod	mass	3.8
	radius	1.6
	length	325
mount with motor and battery	mass	35.3
	radius	17.5
	length	45
	low end to rod center	95

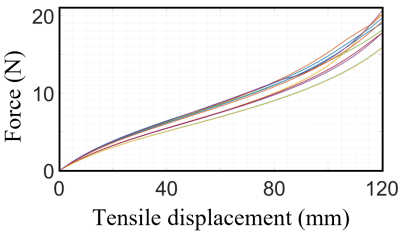


Fig. 3: (Left) Measured physical parameters. Mass and length are in grams and mm, respectively. (Right) Force-displacement curves for long tendons (N = 11). A third-order polynomial was fit to each curve, and the average was used in the simulator.

C. Offline Robot Measurements

Physical parameters inside a simulator (e.g., mass, radius of the end caps, lengths of the rods, inertia tensor, stiffness coefficient, coefficient of friction, etc.) are key to modeling a real robot with high fidelity. However, identifying all of these parameters in one optimization is challenging. Instead,

we measure some of these parameters offline and use them as inputs to the simulator. The measured mass and physical dimensions of the robot's components are listed in Fig. 3. We also collected stiffness data for the long, passive tendons. A batch of 11 sensor tendons was measured on a materials testing system (Instron 3345) and fit with a third-order polynomial; we averaged the curves to input a single model into the simulator. The short, actuated cables are inelastic and modeled with high stiffness in the simulation.

D. System Identification

Physical parameters like the coefficient of friction and the speed of the robot's motors are difficult to measure offline. We have previously introduced a differentiable physics engine that uses gradient descent to identify system parameters from ground truth data [7]–[9]. However, this engine has been demonstrated only in simulation. This section describes the key differences that are necessary for system identification when a real robot is the target platform.

Some parameters, such as the cable attachment points, are not measured on the real robot but are crucial for the success of the simulation. The benchmark tensegrity simulator, NTRT [32], attaches cables at the ends of each rod, which simplifies the modeling process. However, this attachment strategy can lead the simulated robot to collapse and get stuck, unable to recover because all the actuated cables are coplanar, as shown in Figure 4 (Left). Although, this never happens to the real robot. Instead, we put the cable attachment points on the surface of the end caps. This setup prevents the collapsing and results in a more stable simulation, as shown in Figure 4 (Right). Note that these attachment points are not available in the observations; we implement a heuristic to infer their positions.

Each rod has six attachment points that are distributed symmetrically and evenly on two end caps. Attachment points are placed evenly at robot construction every $2\pi/3$ along the disk that is perpendicular to the rod. So it is possible to infer all attachment points from only one of them. At the robot rest state, one rod is in the center and the other two rods are on the side. Referring to Fig. 5, we first compute point A directly, which is on top of the end cap. Then B, C, can be inferred from it. Since the segments AD, BE, CF are parallel to the rod, points D, E, F on the other end are computed based on points A, B, C. Points G, H are closest to points B, C.

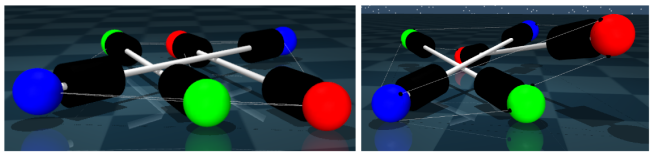


Fig. 4: (Left) After certain controls, the robot collapses onto the ground if all cables are attached to the end of the rods. (Right) With the same control sequences, the robot does not collapse if the cables are attached on the surface of the end caps.

Before each trajectory, the real robot resets itself to the rest state. We also reset the simulator to align with the real robot's initial state using the data collected at time $t = 0$, including both cable lengths and end cap positions. The initial pose

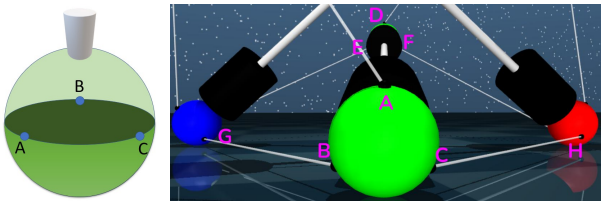


Fig. 5: Cable attachment points on the end caps in the rest state.

generated by the state estimation algorithm (Section IV-B) may not have the robot perfectly resting on the ground due to the limitations of the sensors. Thus, we added gravity to the simulation to force the lowest end caps to come in contact with the ground.

After initializing the robot in simulation, we use the differentiable physics engine to identify the remaining physical parameters by performing gradient descent over a trajectory. To accomplish this task, we introduce a detachment method to compute nonzero gradients at contact points in order to efficiently learn contact parameters. Furthermore, the long-horizon trajectories and noisy observations from the real robot can lead to conflicts in the gradient direction. To combat these conflicts, we introduce the Key Frame Loss (KFL) function to segment the trajectories and compute losses at gait transitions. The detachment method and KFL are described in Sections IV-F and IV-G, respectively.

E. Developing Policies with Symmetry Reduction

Policy search on tensegrity robots is challenging due to combinatorial explosion since a sequence of control signals for each actuation cable with unknown time horizon are need to be discovered. Symmetry reduction has been developed for 6-bar tensegrity robots [31], [39] to reduce the policy search space. Here, we apply symmetry reduction to our 3-bar tensegrity robot and combine it with hybrid control, where a high-level planner outputs the next target state and a low-level PID controller commands the cable lengths.

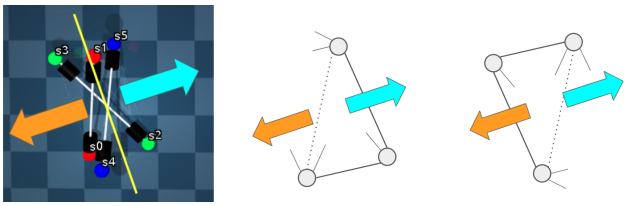


Fig. 6: (Left) The robot principle axis (yellow) is the average direction of the three rods. The forward direction is indicated by the cyan arrow; the backward direction is indicated by the orange arrow. (Right) Two robot poses, with different types of bottom triangles, are mapped after rotating π radians around the ground normal.

To reduce the state space via symmetry reduction, we use a relabeling scheme defined with respect to the support triangle. We define the robot's state in Fig. 6 (Left) as the reference state, from which we can rotate the robot about its principle axis (the yellow line in Fig. 6) by intervals of $\pi/6$ to get 2 types of support triangles with opposite forward and backward directions. To map from one type of support triangle to the other, we rotate the robot by π around the ground normal. After all rotations, we identify the mapping by computing the closest end caps to the reference state.

To generate gaits, i.e., time-ordered sequences of target poses, we conduct a graph search starting from the rest pose. These short gaits are mapped to long control policies via symmetry expansion. The nodes of the graph are robot poses represented by binary cable lengths (0 is retracted and 1 is extended), and the edges of the graph are cable controls (retract, extend, or hold). To limit the search space, we search to a depth of 4. The root node and leaf nodes are in the same rest pose where all cables are fully extended. We use the center of mass displacement as the reward function to generate one forward and one backward rolling gait. We use principal axis rotation as the reward function to generate clockwise and counterclockwise turning gaits.

F. Detachment Approach for Contact Gradients

Impulse-based dynamics can't be directly applied to differentiable simulations due to a zero gradient issue [40]. We propose a "detachment" method to obtain nonzero gradients for contact parameters e, μ. Passive forces, such as restitution and friction generate equal gradients in opposite directions, and gradients of these forces should not be backpropagated. The detach() function [41] returns a new tensor detached from the computational graph, truncating the gradient propagation path.

Our detachment method can be applied to learn the restitution e and the force $F = mg - N$, which is shown in Algorithm 1. The difference of speed (Δv) and position (Δx) at time $t+1$ and time t are detached to stop the passive force gradient from backpropagating. Otherwise, the gradient would be $dx_{t+1}/de = 0$.

Algorithm 1 Detachment Method for Restitution

$v'_{t+1} = v_t - g\Delta t + F/m\Delta t$
$\Delta v = -(1+e)v'_{t+1}.detach()$
$v_{t+1} = v'_{t+1} + \Delta v$
if $x_t < ground$ **then**
  $\Delta x = ground - x_t$
  $x_{t+1} = x_t + v_{t+1}\Delta t + \Delta x.detach()$
else
  $x_{t+1} = x_t + v_{t+1}\Delta t$
end if

The detachment approach is also used for learning friction parameters, as shown in Algorithm 2. We detach Δv because the friction force is a passive force. We add the term $-\Delta p_f + \Delta p_f.detach()$ to pass the gradient to μ in the case where there is only static friction. This term guarantees that μ can always be updated by the gradient, no matter if the initial μ is higher or lower than the actual one. The evaluation of the contact gradients and integration with Time of Impact (ToI) computations can be found in the accompanying appendix available online[1].

G. Key Frame Loss

System identification with long trajectories is difficult due to sensing noise and the non-monotonicity of trajectories.

As shown in Fig. 7, the later observations are preferable to reduce the fitting error. Consider a simple example of a 1D trajectory from $x = vt$ with length T. We are given two

Algorithm 2 Detachment Method in Friction Computation

$v'_{t+1} = v_t + F/m\Delta t$
$\Delta v = -v'_{t+1}.detach()$
$\Delta p_f = \mu N \Delta t/m$
if $|\Delta v| > \Delta p_f$ **then** $\Delta v = \Delta v * \Delta p_f/|\Delta v|$
else $\Delta v = \Delta v - \Delta p_f + \Delta p_f.detach()$
end if
$v_{t+1} = v_t + F/m\Delta t + \Delta v$
$x_{t+1} = x_t + v_{t+1}\Delta t$

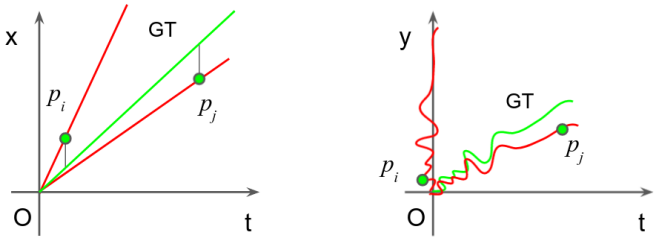


Fig. 7: Fitting the ground truth line (green) with two noisy observations, p_i and p_j. Although p_i and p_j have the same sensing error, the fitted line (red) by p_j is better than that by p_i. $x = vt$ is a linear function (Left). $y = f(t)$ is a non-convex function (Right).

data points, $p_i = (t_i, x_i)$ and $p_j = (t_j, x_j)$. t_i is close to 0 and t_j is close to T. x_i and x_j have the same observation error relative to the ground truth. v is the parameter to estimate. Fig. 7 (Left) shows that the fitted line by p_j is better than that by p_i. Moreover, if $x = f(t)$ is a non-monotonic function, which is closer to reality for tensegrity robot trajectories, the fitted line by p_i is worse as shown in Fig. 7 (Right). The trajectory of an end cap is noisier around time $t = 0$ because of the sudden motor activation. Note that the loss landscape may be non-convex due to contradictions in gradient directions, which can interfere with the optimization step. As shown in Fig. 8, three losses yield opposite gradient directions.

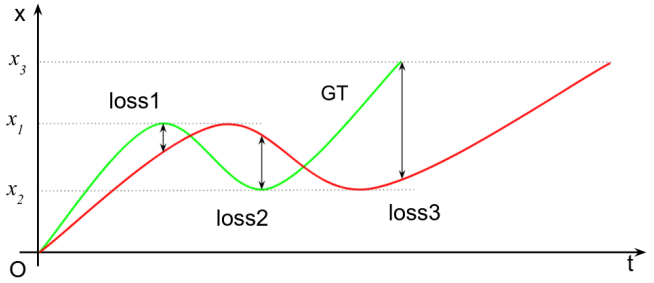


Fig. 8: The ground truth trajectory (green) includes three targets, x_1, x_2 and x_3. The unidentified engine (red) has a lower speed and takes more time to reach the targets. Three losses are computed at these targets. The gradients at loss1 and loss3 pull up the red line; however, the gradient at loss2 drags it down.

The multiple-shooting (MS) method has been applied for system identification with long trajectories [42]. MS, however, needs full robot observations (i.e., positions and velocities of each end cap) to initialize the simulator. The defects may lead to contradicting gradient directions.

We introduce the Key Frame Loss (KFL) function, which splits the robot trajectory into a list of monotonic segments using unidirectional control intervals. We only consider the loss from the last frame in each segment. There are four

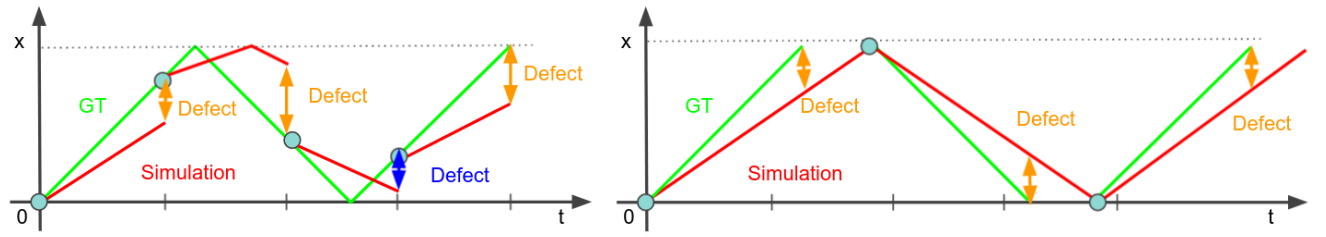


Fig. 9: (Left) The multiple-shooting (MS) method needs both positions and velocities at each time step so the simulator can align with the ground truth at intermediate time steps. It uses the entire length of a time window to compute defects between the ground truth and the simulator. The window size is empirical, and the defects may provide contradicting gradient directions. (Right) The KFL method only needs positions at each time step. The trajectory is split into segments based on unidirectional control intervals. The time step at the end of the control signal is the key frame where the defects are computed. The window size is the same size as the control interval, and the defects provide consistent gradient directions.

departures of KFL from MS: 1) KFL only needs partial observation, i.e., it only requires end cap positions, not their velocities; 2) In the intermediate time steps, we do not aim to recover the simulator from the partially observed state since we do not know velocities; 3) The observation frequency does not have to be constant; 4) The shooting time window size is not fixed. Fig. 9 provides a comparison between MS and KFL.

The benefits of KFL are that it 1) eliminates the factor of time, 2) makes system identification robust to observation noise, and 3) ensures the correct gradient direction. This strategy can be generalized to other optimization problems using trajectories as ground truth data: trajectories can be split into monotonic segments, and the KFL can be applied at the final frame of each segment like in this work.

Fig. 10: (Above) The ground truth (GT) and engine-predicted trajectory are split into three gaits based on the control intervals. (Below) The gaits are aligned at the starting points and their key frames (KF0, KF1, KF2) are the last time frames of each intersection. The predicted trajectory takes less time in gait 0 and 2, but more time in 1, which leads to contradicting gradient directions.

To get the key frames, we split the trajectory into gaits for monotonic segmentation, as in Fig. 10. In each gait, each cable follows only one of the possible controls: retracting, holding, or extending. These controls form a convex trajectory segment, avoiding the zig-zag trajectory that leads to contradicting gradients shown in Fig. 8. The key frame (KF) is the last time step of each gait. Consider an example trajectory with three gaits, as shown in Fig. 10. Both the ground truth (red) and the engine-predicted trajectory are split into three gaits. The corresponding gaits are remapped, and the KFs are the last time frames of their intersections. The KFL is the system difference, i.e., the sum of the differences in the positions of the end caps at the KF. We select the last time step because 1) the observation noise around the start of each gait may lead to worse fitting as shown in Fig. 7 and 2) the gait transition point is where one gait is finished; however, whether the gait execution is complete is determined by measurements from the cable length sensors, so it is possible the gait transition occurs earlier or later than predicted, as shown by the gaps in Fig. 10.

Losses from different gaits may have contradicting gradients. In Fig. 10, the predicted trajectory takes less time for gait 0 and 2, but it takes more time for gait 1. These contradicting gradients can confuse the optimizer, as it will not know whether to speed up or slow down. Some possible reasons for these time differences are 1) The ground truth timestamp is noisy due to the limitations of the sensors; 2) The noisy cable length sensor readings lead to early or later gait transitions; 3) The starting states are not perfectly aligned; 4) There are gaps between the simplified physics engine and the real robot. To solve the problem, we adopt a mask filter to handle the contradicting gradients. Our heuristic states that the identified engine should execute the same gaits on the robot with similar time and behavior. Then, we take the execution time of the whole trajectory as an indicator to filter out the opposing gradients. For example, in Fig. 10, since the predicted trajectory takes a shorter length of time than the ground truth, we only consider key frame losses of gait 0 and gait 2 to slow down the engine, and we ignore the loss from gait 1. Additional evaluation about the KFL can be found in the link to the appendix[1].

V. Experimental Results

Our experiments evaluate the `R2S2R` pipeline both in simulation and on the real 3-bar tensegrity robot. Section V-A evaluates key components of the proposed process, i.e., the detachment strategy and the Key Frame Loss (KFL), using synthetic data. Section V-B evaluates the full `R2S2R` pipeline on the 3-bar tensegrity robot shown in Fig. 1 with the dimensions in Fig. 3. Section V-C shows the additional improvement when a second iteration of the `R2S2R` pipeline is performed.

A. System Identification with Synthetic Trajectories

We evaluate KFL by comparing three different loss functions: 1) The average defects from all time steps (*All Step*) in each window; 2) The average defects of the last time step (*Last Step*) in each window; 3) Our method (Ours) with KFL,

which averages defects of the last time step in each window and detaches the observations between each window. The difference between these three methods is shown in Fig. 11.

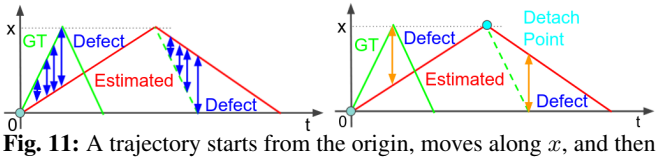


Fig. 11: A trajectory starts from the origin, moves along x, and then returns to the origin. This trajectory can be split into 2 windows. The ground truth trajectory (green) takes less time than the estimated one (red). The second window of the ground truth is shifted to the right to compute defects. (Left) The All Step loss function averages the defects of all time steps in each window. (Right) The Last Step loss function averages defects from the last time step in each window. KFL also detaches the system state at the transition point (cyan) of 2 windows [41].

To simplify the problem, only motor speed is identified in this task (in addition to motor speed, the friction coefficient is also identified in V-B). A synthetic trajectory where the motor speed is set to 0.5 is used as ground truth. The defects are the differences in end cap positions. An Adam optimizer with learning rate 0.1 is applied. Fig. 12 shows that our method converges fast, smooth and stable. The curve of the Last Step method fluctuates more due to the gradient propagation across windows, even if the motor speed is close to the target in later iterations. This shows the neeed for detaching. The multiple windows approach considering all time steps performs worse.

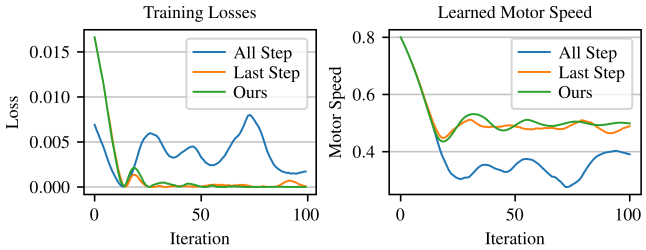


Fig. 12: To identify the motor speed parameter, three loss functions are considered: All Step (blue), Last Step (orange), and KFL loss with state detachment (green). KFL loss converges better.

B. Complete Evaluation of the `R2S2R` *Pipeline*

The `R2S2R` pipeline has been applied twice to show continuous improvement each iteration. In the first iteration, a single robot trajectory given random controls is used to identify the physics engine parameters, including motor speed and friction coefficient. From the identified engine, we generate three policies corresponding to "forward rolling," "backward rolling," and "counterclockwise turning" behaviors. These policies are mapped to two long, open-loop gaits with symmetry reduction: a straight gait (Straight) composed of forward and backward rolling policies and a counterclockwise turning gait (CCW Turning) composed of "counterclockwise turning" polices. These two gaits are executed in simulation and on the real robot, and these executions are shown as blue and orange lines, respectively, in Figure 13. We also execute these gaits starting from different positions on the ground to test the repeatability on the real platform (Table I). The Center of Mass (CoM) and orientation are measured at the end of each trajectory. We observe greater variability in the CCW Turning gait. In the simulator, this gait capitalizes on uniform friction to rotate the robot's principal axis; however, in the real world, the friction between the ground and the end caps is nonuniform. This discrepancy explains the higher variability.

TABLE I: Real Robot Gait Execution Repeatability Test

Metric	End CoM Position (m)		End Orientation (radian)	
Trajectory	Straight	CCW Turning	Straight	CCW Turning
Mean ± std	[-1.14± 0.02, 0.29±0.05]	[0.01±0.02, 0.29±0.02]	0.136±0.07	2.7± 0.23

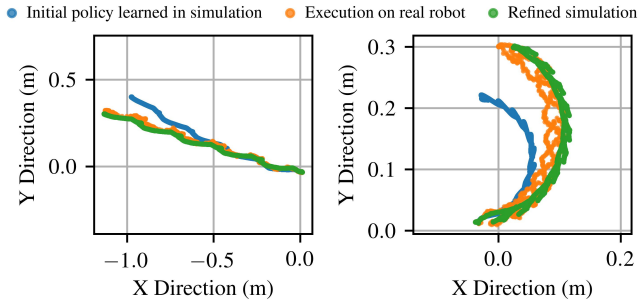


Fig. 13: Trajectories in simulation and on the real robot for the same policy. The open-loop trajectories start at (0, 0). The robot trajectory is observed in order to generate new data to further refine the simulator. (Left) The output for a straight-line trajectory of the center of mass. The policy executes forward and backward motion in the symmetry-reduction frame. (Right) The output from a counterclockwise turning policy.

C. Continuous Improvement with `R2S2R`

The second iteration re-identifies the engine using the two real robot trajectories from the first iteration, and then we execute these policies again in simulation to show how the `R2S2R` pipeline reduces the sim2real gap. The trajectories sampled from the refined simulation are plotted as green lines in Fig. 13. The robot position and orientation at the end of the trajectories are compared in Table II. Generally, the CoM and orientation errors go down in the refined simulation; however, the slightly increased CCW Turning orientation error shows the limitations of simulating nonuniform environment friction with a uniform friction coefficient.

TABLE II: Iterative Refined Simulation with Real Data

Metric	End CoM Position (m)		End Orientation (radian)	
Trajectory	Straight	CCW Turning	Straight	CCW Turning
Initial Simulation	[-0.97, 0.40]	[-0.02, 0.22]	0.31	2.36
Real Robot Execution	[-1.11, 0.32]	[0.02, 0.29]	0.02	1.78
Refined Simulation	[-1.11, 0.29]	[0.05, 0.30]	0.12	2.45
Initial Error	0.16	0.26	0.09	0.04
Refined Error	0.03	0.06	0.03	0.13

Qualitatively, the trajectory executed on the robot has the same behavior as the initial simulation. After the second iteration, the simulation trajectory is even more similar to the real robot trajectory, as shown in Fig 15 and 16.

After the second iteration of system identification, two new clockwise turning polices are generated, and these policies are mapped to two long, open-loop gaits (Figure 14). The deviations are larger for the arching gait (Figure 14 Left) because of the nonuniform friction between the end caps and the ground. This gait covers a larger area, and the coefficient of friction is not uniform everywhere on the floor.

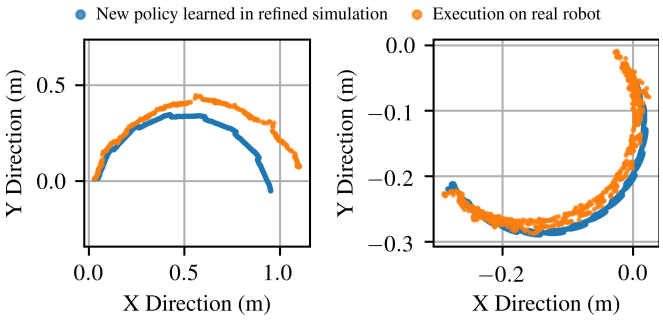


Fig. 14: Two new clockwise turning trajectories are executed in simulation and on the real robot. The open-loop trajectories start at (0, 0). (Left) The arch turning trajectory. (Right) The in-place turning trajectory.

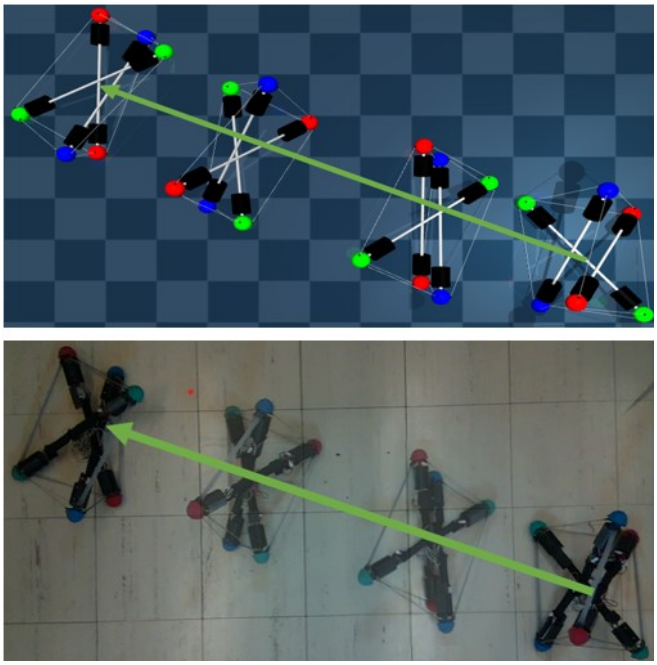


Fig. 15: The frame-by-frame comparison of the straight trajectory execution in simulation and on the real robot.

We observe smaller deviations for the in-place turning gait (Figure 14 Right) where the robot is confined to a much smaller region. Qualitative visualization is also avaliable in Fig 17 and Fig 18.

VI. Conclusion

This paper demonstrates an R2S2R pipeline that mitigates the sim2real gap for tensegrity robots and develops locomotion policies that can be seamlessly transferred to real robots. The sim2real gap is reduced by using a differentiable physics engine that learns system parameters from real robot data via gradient descent. The simulation can then generate new locomotion policies and extend them for long trajectories via symmetry reduction. After transferring these policies to the real robot, the simulation can be continuously improved by re-identifying the system parameters from the recorded trajectories. To enable efficient system identification, we introduce and experimentally validate the detachment approach for computing contact gradients and the Key Frame Loss with a trajectory segmentation strategy. In future work,

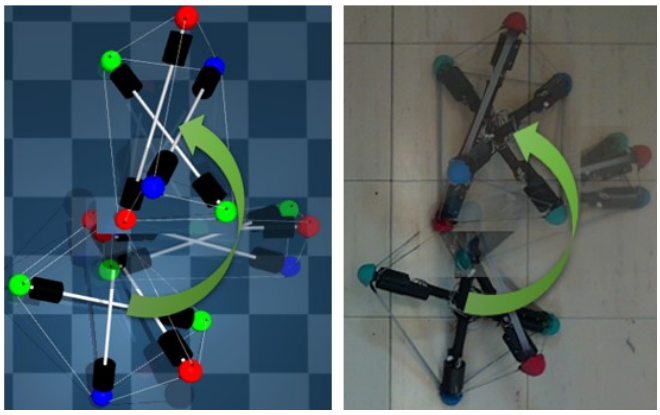


Fig. 16: The frame-by-frame comparison of the counterclockwise turning trajectory execution in simulation and on the real robot.

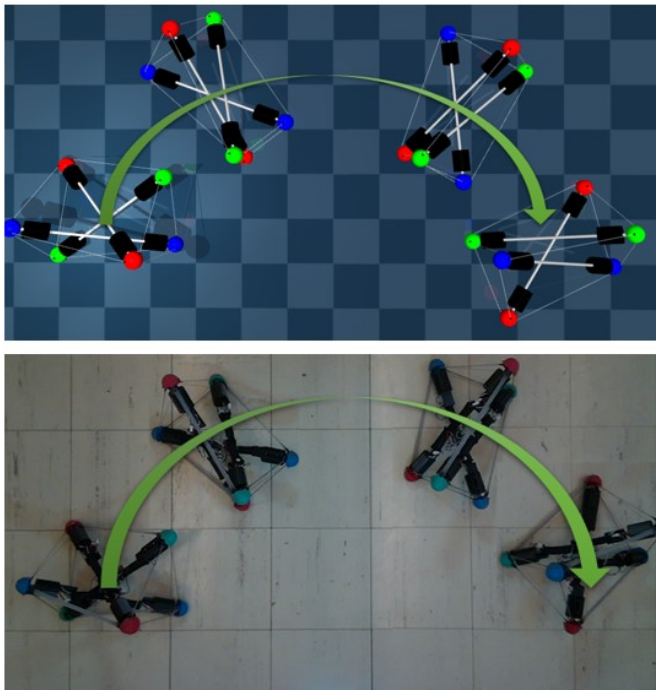


Fig. 17: The frame by frame comparison of the clockwise arching trajectory execution in simulation and on the real robot.

the accuracy of the physics engine could be improved by allowing the system to identify the physical parameters of each component (e.g., the stiffness of each sensor tendon) and adapt online as these parameters change over time.

References

[1] S. Lessard, D. Castro, W. Asper, S. D. Chopra, L. B. Baltaxe-Admony, M. Teodorescu, V. SunSpiral, and A. Agogino, "A bio-inspired tensegrity manipulator with multi-dof, structurally compliant joints," in *IROS*. IEEE, 2016, pp. 5515–5520.

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

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

[4] J. Bruce, A. P. Sabelhaus, Y. Chen, D. Lu, K. Morse, S. Milam, K. Caluwaerts, A. M. Agogino, and V. SunSpiral, "Superball: Exploring tensegrities for planetary probes," *12th International Symposium*

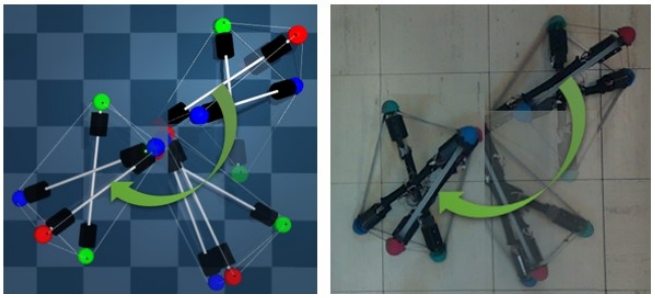


Fig. 18: The frame by frame comparison of the in-place clockwise turning trajectory execution in simulation and on the real robot.

on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS), 2014.

[5] D. S. Shah, J. W. Booth, R. L. Baines, K. Wang, M. Vespignani, K. Bekris, and R. Kramer-Bottiglio, "Tensegrity robotics," *Soft Robotics*, vol. 9, no. 4, pp. 639–656, 2022.

[6] D. A. Surovik, K. Wang, M. Vespignani, J. Bruce, and K. E. Bekris, "Adaptive tensegrity locomotion: Controlling a compliant icosahedron with symmetry-reduced reinforcement learning," *The International Journal of Robotics Research*, vol. 40, pp. 375 – 396, 2021.

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

[8] ——, "A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines," in *L4DC*, vol. 120. PMLR, 10–11 Jun 2020, pp. 651–665.

[9] ——, "A recurrent differentiable engine for modeling tensegrity robots trainable with low-frequency data," in *ICRA*, 2022.

[10] K. Bousmalis and S. Levine, "Closing the simulation-to-reality gap for deep robotic learning (2019)," *Google AI Blog http://ai. googleblog. com/2017/10/closing-simulation-to-reality-gap-for. html*, 2019.

[11] P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez, "Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation," *Virtual Reality*, vol. 24, no. 2, pp. 271–288, 2020.

[12] J. K. Murthy, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben, *et al.*, ""gradsim: Differentiable simulation for system identification and visuomotor control"," in *ICLR*, 2020.

[13] J. Collins, D. Howard, and J. Leitner, "Quantifying the reality gap in robotic manipulation tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6706–6712.

[14] F. Ramos, R. Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," in *Robotics: Science and Systems (RSS)*, 2019. [Online]. Available: https://arxiv.org/abs/1906.01728

[15] F. Muratore, T. Gruner, F. Wiese, B. Belousov, M. Gienger, and J. Peters, "Neural posterior domain randomization," in *Conference on Robot Learning*. PMLR, 2022, pp. 1532–1542.

[16] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[17] V. Lim, H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg, ""real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting"," in *ICRA*. IEEE Press, 2022, p. 8282–8289.

[18] A. Sethuraman and K. A. Skinner, "Towards sim2real for shipwreck detection in side scan sonar imagery," *3rd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics*, 2022.

[19] M. Navardi, P. Dixit, T. Manjunath, N. R. Waytowich, T. Mohsenin, and T. Oates, "Toward real-world implementation of deep reinforcement learning for vision-based autonomous drone navigation with mission," *UMBC Student Collection*, 2022.

[20] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE TRO*, vol. 36, no. 1, pp. 1–14, 2019.

[21] A. Sena, H. Kavianirad, S. Endo, E. Burdet, and S. Hirche, "The gap in functional electrical stimulation simulation," *3rd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics*, 2022.

[22] J. Jabbour, "Closing the sim-to-real gap for ultra-low-cost, resource-constrained, quadruped robot platforms," *3rd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics*, 2022.

[23] W. Huang, X. Huang, C. Majidi, and M. K. Jawed, "Dynamic simulation of articulated soft robots," *Nature Communications*, vol. 11, no. 1, pp. 1–9, 2020.

[24] X. Huang, W. Huang, Z. Patterson, Z. Ren, M. K. Jawed, and C. Majidi, "Numerical simulation of an untethered omni-directional star-shaped swimming robot," in *ICRA*. IEEE, 2021, pp. 11 884–11 890.

[25] N. N. Goldberg, X. Huang, C. Majidi, A. Novelia, O. M. O'Reilly, D. A. Paley, and W. L. Scott, "On planar discrete elastic rod models for the locomotion of soft robots," *Soft Robotics*, vol. 6, no. 5, pp. 595–610, 2019.

[26] B. Wen, W. Lian, K. Bekris, and S. Schaal, "Catgrasp: Learning category-level task-relevant grasping in clutter from simulation," in *ICRA*. IEEE, 2022, pp. 6401–6408.

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

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

[29] M. Zhang, X. Geng, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel, and S. Levine, "Deep reinforcement learning for tensegrity robot locomotion," in *ICRA*. IEEE, 2017, pp. 634–641.

[30] J. Luo, R. Edmunds, F. Rice, and A. M. Agogino, "Tensegrity robot locomotion under limited sensory inputs via deep reinforcement learning," in *ICRA*. IEEE, 2018, pp. 6260–6267.

[31] D. Surovik, K. Wang, M. Vespignani, J. Bruce, and K. E. Bekris, "Adaptive Tensegrity Locomotion: Controlling a Compliant Icosahedron with Symmetry-Reduced Reinforcement Learning," *IJRR*, 2019.

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

[33] B. T. Mirletz, I.-W. Park, R. D. Quinn, and V. SunSpiral, "Towards bridging the reality gap between tensegrity simulation and robotic hardware," in *IROS*. IEEE, 2015, pp. 5357–5363.

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

[35] W. R. Johnson, A. Agrawala, X. Huang, J. Booth, and R. Kramer-Bottiglio, "Sensor tendons for soft robot shape estimation," in *IEEE Sensors*. IEEE, 2022, pp. 1–4.

[36] W. R. Johnson, J. Booth, and R. Kramer-Bottiglio, "Integrated sensing in robotic skin modules," in *2021 IEEE Sensors*. IEEE, 2021, pp. 1–4.

[37] S. Lu, W. R. Johnson III, K. Wang, X. Huang, J. Booth, R. Kramer-Bottiglio, and K. Bekris, "6n-dof pose tracking for tensegrity robots," *arXiv preprint arXiv:2205.14764*, 2022.

[38] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[39] D. Surovik, J. Bruce, K. Wang, M. Vespignani, and K. E. Bekris, "Any-axis tensegrity rolling via bootstrapped learning and symmetry reduction," in *ISER*, Buenos Aires, Argentina, 11/2018 2018.

[40] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints," in *RSS*, 2021.

[41] Pytorch, "Pytorch Detach Method," Accessed 2022, https://pytorch.org/docs/stable/generated/torch.Tensor.detach.html.

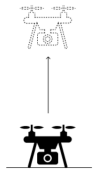
[42] E. Heiden, C. E. Denniston, D. Millard, F. Ramos, and G. S. Sukhatme, "Probabilistic inference of simulation parameters via parallel differentiable simulation," *ICRA*, 2022.

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

APPENDIX

A. Contact Gradients with/without "Detach"

To highlight our "Detach" method in the contact model, we design three toy problems to show how our method could generate correct gradient for parameter optimization.



The first test evaluates Algorithm 1 and corresponds to a trajectory optimization challenge from the literature [40], where a drone is taking off from the ground and reaching a fixed height at $t = 500$. The loss is the Mean Square Error (MSE) of the estimated drone's height $\hat{x}$ against the ground truth height that is $x = 10m$.

The clamping contact between the drone and ground causes zero gradients and halts progress in the optimization as the passive contact generates an opposite equal gradient to F. After detaching the contact velocity impulse Δv and position impulse Δx from the computation graph of Algorithm 1, the correct gradient keeps increasing F even if there is no change in the loss for the first 100 iterations (Figure 19).

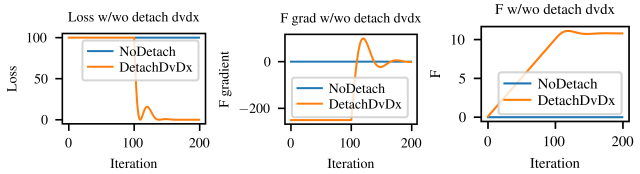


Fig. 19: We train a drone to lift off the ground to evaluate our "Detaching" method. Loss is the MSE of the distance from drone $\hat{x}$ to target x at $t = 500$. The drone is initialized resting on the ground. The gradient $\dfrac{\partial \hat{x}}{\partial F}$ is zero with clamping contact. However, after detaching the contact response Δv(dv) from the computation graph, the correct gradient on F can guide the optimizer to increase F and reach the target.

We bring the box problem, pulling a box to the target position in 200 time steps, to evaluate Algorithm 2. The pulling force F is known and the ground friction coefficient μ is to be estimated. We take the MSE of box position and target position as a loss function.

The initial μ is 1 and box can't be moved with such large friction. Our detach method can generate correct negative gradient to reduce μ and the loss. However, without our method, the gradient is always zero and the loss curve never goes down.

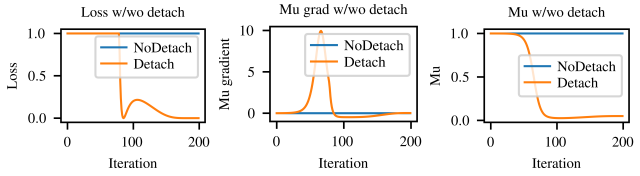


Fig. 20: The box pulling problem shows the detaching method can get correct gradient to reduce the friction coefficient μ. Otherwise, zero gradients would happen on optimization of μ.

In general, we suggest to apply the "detach" method to clamping contacts of all actuated objects. However, this method should not be applied to objects whose movement only relies on these passive contact forces, e.g. the billiard. The philosophy is that we keep the "trending" gradient from the small actuation force even if the object can't move. This avoids the gradient discontinuities between clamping contact and separating contact.

Finally, we take the billiards problem, applying F on ball A and aiming the ball B to reach a target position in 500 timesteps, to evaluate the detaching philosophy - only detach actuated object. The loss is the mean MSE of B's position and target position.

We compare the differences between detaching both A, B and detaching A only. As we suggested, we should only detach the actuated object A's contact response. We shouldn't detach B's contact response, Because B is only affected by the passive contact forces. The result in Figure 21 shows that detaching both A and B could get zero gradient on F and no change in F and loss. However, if we only detach the actuated object A, we can still get the correct gradient direction and escape the saddle point.

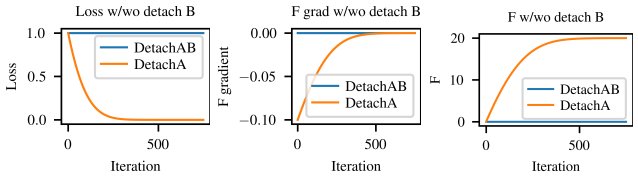


Fig. 21: The billiards problem shows that only detaching the actuated object A can get the correct gradient on actuation force F to reduce the loss. Detaching the passive moved object B will isolate B from gradient passing, which result to zero gradients on F.

B. Gaits Transition Smoothing

After generating locomotion gaits for all bottom triangles, we have a controller to generate longer trajectories. However, these trajectories have redundant actuation between gait transitions. We adopt a smoothing algorithm to smooth out the transition.

Let's consider a simple example of gaits transition in Figure 22. Although the following gait includes 3 steps, we only keep the third step after removing the reset step and finding the shortcut with minimum actuated cables. But these shortcuts should be reevaluated because some of them may not work due to changes in center of gravity.

C. Comparison to Alternatives on a Small-scale Problem

This section evaluates Key Frame Loss (KFL) on a small-scale problem where the MS alternative can be applied. A learning rate of 0.1 and an Adam optimizer are used.

The problem involves identifying the velocity of a vehicular model so as to match a demonstration trajectory from a ground truth vehicle that drives between points A and B. The ground truth trajectory uses a velocity $v = 1$ m/s to reach point B at $t = 100$, return to

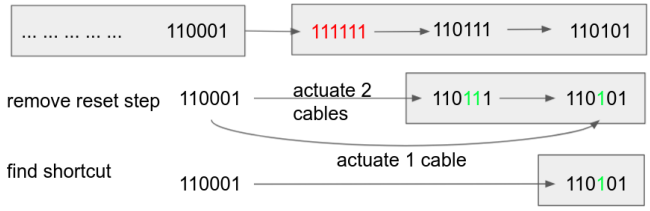


Fig. 22: The current gait ends at state 110001, i.e., cables 1,2,6 are at maximum length and cables 3,4,5 are at minimum length. The next gait includes three steps. Step one, which is to reset the robot to a neural state, is removed. Then we compute the distance from 110001 to the remaining two steps. We transit to step 3 directly since fewer cables are actuated.

A at $t = 200$, and reach B again at $t = 300$. We compare 3 methods: a) A naive approach that computes loss at each step as the difference between ground truth and estimated trajectory; b) A multiple shooting (MS) method [42] that splits the trajectory to multiple windows and computes the loss by defects at the end of each window (different numbers of windows are tested for variants MS2, MS3, MS4); and c) the proposed KFL method. Fig. 23 provides the comparison.

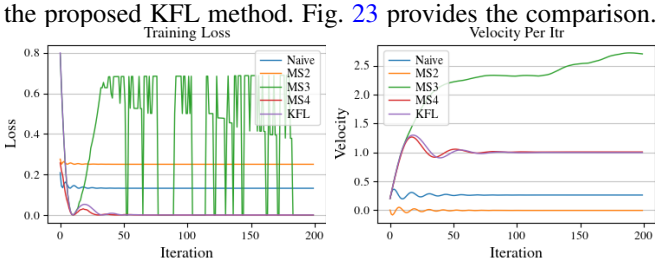


Fig. 23: Loss and velocity curve comparison during the training process of Naive, multiple shooting with 2, 3, and 4 windows (MS), and the Key Frame Loss method (KFL).

Only KFL and MS4 converge to the correct velocity (1 m/s). The other methods fail due to conflicting gradients. We also tested MS for a higher number of windows (5 to 9). The resulting loss curves are non-smooth, but the velocity eventually converges. The MS method samples discrete gradients on a continuous trajectory, so the sampling frequency must be high enough to compute correct gradients, which need to be discovered empirically. At the same time, the frequency should not be too high for noisy ground truth data, as the noise may dominate the optimization. The KFL approach splits the trajectory by "gaits" (i.e., the trajectory segment where the robot moves to a specific point), and this allows it to compute correct gradients.

D. Detachment Method for Restitution with Time of Impact (TOI)

For more accurate contact point computation, the Time of Impact (TOI) is introduced [43] to compute the exact time that the contact happens. The detachment method can also integrate with TOI for restitution inference, which is shown in Algorithm 3.

E. Tensegrity Robot Locomotion and Applications

Tensegrity robots' compliance gives them the ability to adapt to unstructured terrain and survive harsh impacts, motivating them as future planetary rovers [34]. Many tensegrity robots achieve locomotion by changing their tendon lengths to shift their center of mass outside of their polygon of stability and therefore roll [5]. Electric motors that drive winches are commonly used to extend and contract tensegrity robots' cables [4]. The 3-bar tensegrity robot used in this work has six such motors that drive winches to extend and contract six cables (the remaining three tendons are passive elastic elements) in order to shift the robot's center of mass and achieve locomotion. The robot can demonstrate the different locomotion policies enumerated in this paper (forward rolling, backward rolling, counterclockwise turning, and clockwise turning) by controlling its six cable lengths to match a sequence of target shapes. Sensor tendons [35] that run parallel to each actuator sense the length of the cables and provide feedback to a PID controller so that the robot can execute the locomotion policies with high fidelity. In future work, we aim to demonstrate this lightweight, adaptable tensegrity robot navigating unstructured terrain and maintaining robust control even when subjected to harsh impacts.

Algorithm 3 Detachment Method for Restitution with TOI

$v'_{t+1} = v_t - g\Delta t + F/m\Delta t$
$x'_{t+1} = x_t + v'_{t+1}\Delta t$
$\Delta v = -(1+e)v'_{t+1}.detach()$
$v_{t+1} = v'_{t+1} + \Delta v$
if $x_t > ground$ **then**
  $toi = (x_t - ground)/max(-v_t, 10^{-4})$
  $toi = toi.detach()$
else
  $toi = 0$
end if
if $x'_{t+1} < ground$ **then**
  $\Delta x = ground - x'_{t+1}$
  $x_{t+1} = x_t + v_{t+1}\Delta t + \Delta x.detach() - toi\Delta v$
else
  $x_{t+1} = x_t + v_{t+1}\Delta t$
end if