Supplementary: Rethinking Optimization with Differentiable Simulation from a Global Perspective

Rika Antonova[1*] **Jingyun Yang**[1*] **Krishna Murthy Jatavallabhula**[2] **Jeannette Bohg**[1]
Stanford University[1] Massachusetts Institute of Technology[2]

A Additional Environment Descriptions and Details

In this section, we list details of environments we covered in the paper. We describe the simulation framework, physics model, contact type, parameter information, loss configuration, as well as landscape and gradient characteristics for each environment.

A.1 Rigid Body Environments

A.1.1 3-link Cartpole

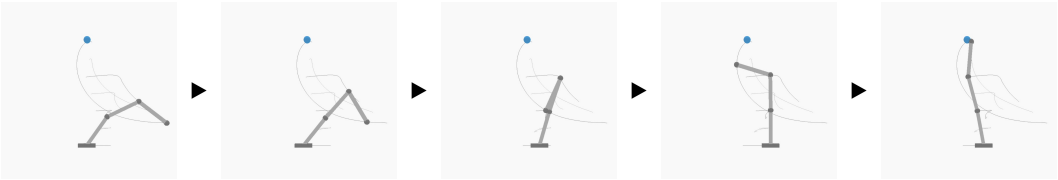

Figure 1: Illustration of the *3-link Cartpole* environment.

A cart carries a triple inverted pendulum where each link has length 1m. The weight of the links that are farther away from the cart are made lighter so that controlling the mechanism is easier. The goal is to move the cart and actuate the joints so that the tip of the pendulum is as close as possible to a preset goal location.

- **Simulation Framework or Physics Model:** Nimble [1]
- **Types of Contacts:** none (there is no self-contact between different parts of the cartpole)
- **Parameter Dimensionality:** 400
- **Parameter Description:** at each of the 200 timesteps, the parameters specify cart velocity with 1 dimension and torques of 3 joints.
- **Loss:** L2 distance from final tip position to target position.
- **Landscape and Gradient Characteristics:** landscape is very smooth. Gradient quality is good.

A.1.2 Pinball

Figure 2: Illustration of the *Pinball* environment with 16 colliders.

On a vertical platform of 8m wide and 10m tall, a ball is dropped on to a grid of $n_h \times n_w$ spinning colliders that have one revolute joint each attached to the platform. The goal is to guide the ball to a goal position at the end of the episode by adjusting the orientation of each collider.

- **Simulation Framework or Physics Model:** Nimble [1]

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

- **Types of Contacts:** rigid (pinball collides with colliders and walls)
- **Parameter Dimensionality:** $n_h \times n_w$ (in this work, we consider two setups with $n_h = 1, n_w = 2$ and $n_h = 4, n_w = 4$)
- **Parameter Description:** rotation angle of each spinning collider in the $n_h \times n_w$ grid.
- **Loss:** L2 distance from final pinball position to target pinball position near the bottom-right of the platform.
- **Landscape and Gradient Characteristics:** landscape has large flat regions as well as discontinuities. Gradients are zero in flat regions and not useful at locations of discontinuities.

A.2 Deformable Object Environments

A.2.1 Fluid

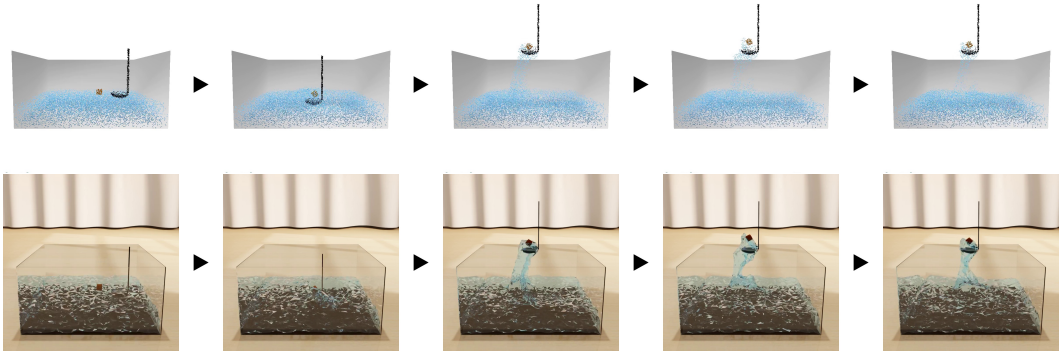

Figure 3: Illustration of the *Fluid* environment. The upper and lower rows are renderings of the same episode in the same environment. The upper row uses the built-in realtime rendering engine in DiffTaichi; while the lower row uses Blender, which is slower but higher quality.

A ladle with 2 DOF is manipulated to scoop a sugar cube from a tank of transparent syrup with width 0.4m, depth 0.4m, and height 0.2m. The goal is to scoop the cube to as high of a position as possible while being close to the ladle and the center vertical axis of the tank. This environment is made with DiffTaichi. The dynamics of syrup and the sugar cube in this environment are modeled with MLS-MPM [2], a state-of-the-art particle-based fluid simulation method.

- **Simulation Framework or Physics Model:** DiffTaichi [3] with MLS-MPM [2] as physics model for fluid and the sugar cube
- **Types of Contacts:** collision between rigid objects and fluid (such as fluid particles bouncing back off container walls)
- **Parameter Dimensionality:** 10
- **Parameter Description:** an episode splits into 5 equal-length segments. In each segment, 2 parameter values control the horizontal (forward-backward) and vertical (up-down) speed of the ladle. Note that although the ladle might not directly make contact with the cube, the ladle can push the liquid particles, which can then push the cube away from the ladle.
- **Loss:** the loss is computed as

  $$\max(0, y - y_w) + 3 \cdot \sqrt{(x - x_s)^2 + (z - z_s)^2} + \sqrt{(x - x_c)^2 + (z - z_c)^2},$$

  where (x, y, z) denotes the final sugar cube position, (x_s, y_s, z_s) denotes final ladle body center position, and $y_w = 0.2$ denotes height of the container. In this and all following DiffTaichi environments, the x-axis points rightward, the y-axis points upward, and the z-axis points to the front.
- **Landscape and Gradient Characteristics:** landscape is rugged and has many local minima. Gradient quality is good.

A.2.2 PlasticineLab Environments

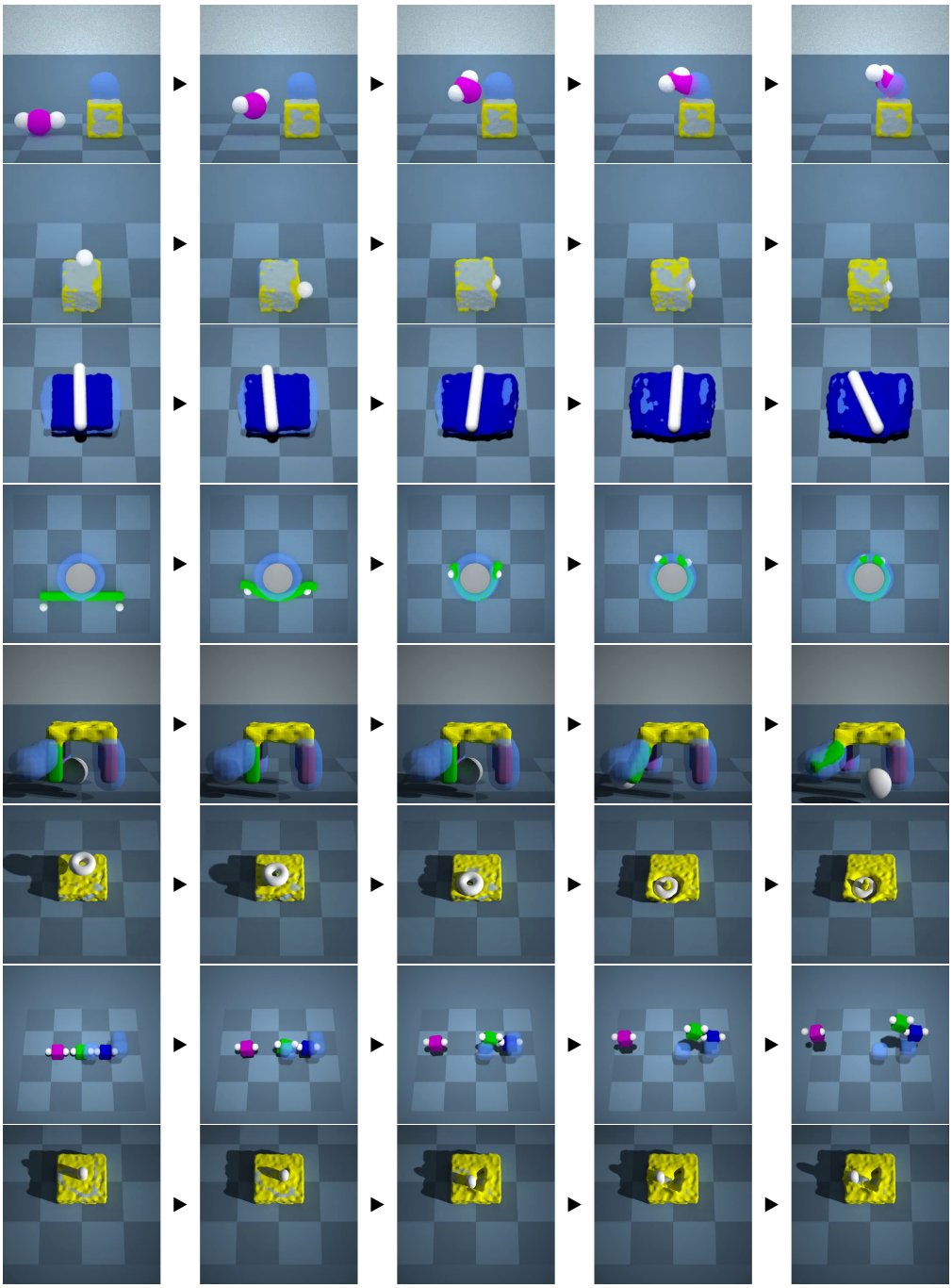

Figure 4: Illustration of environments derived from *PlasticineLab*. From top to bottom, we show *Assembly*, *Pinch*, *RollingPin*, *Rope*, *Table*, *Torus*, *TripleMove*, and *Writer*.

We consider 8 different environments derived from PlasticineLab [4] — Assembly, Pinch, RollingPin, Rope, Table, Torus, TripleMove, and Writer. These environments involve 1-3 anchors or a pin manipulating one or several pieces of deformable objects. The goal of all environments are to make the final deformable object configuration close to a target shape. When we adapt the environments for our purpose, we only modify the format of optimizable parameters and leave other aspects of the environments such as dynamics, episode length, and loss formulation unchanged. As we already described the high-level objectives of several PlasticineLab environments we analyzed in detail in the main paper, we direct readers to the original paper [4] for additional details of each environment.

- **Simulation Framework or Physics Model:** PlasticineLab is based on DiffTaichi [3]; its environments use MLS-MPM [2] to model interactions between rigid and deformable objects; rigid bodies are modeled using signed distance fields (SDF)
- **Types of Contacts:** collision between rigid and deformable objects
- **Parameter Dimensionality:** 90 (TripleMove), 30 (Assembly, Rope), 15 (Pinch, RollingPin, Table, Writer)
- **Parameter Description:** an episode splits into 5 equal-length segments. In each segment, the 3D velocities of each anchor or pin are controlled by optimizable parameters. In TripleMove, there are six anchors to be controlled, so the optimizable parameter has a total of [5 segments $\times$ 6 anchors $\times$ 3 velocity values = 90] dimensions. In Assembly and Rope, there are two anchors to be controlled, so the parameter has [5 segments $\times$ 2 anchors $\times$ 3 velocity values = 30] dimensions. In Pinch, Table, and Writer, there is one anchor, so the parameter has [5 segments $\times$ 1 anchor $\times$ 3 velocity values = 15] dimensions. In RollingPin, instead of controlling 3D velocity of the pin in each segment, the environment uses 3 values to control the left-right, up-down, and top-down tilting angle of the pin, leading to [5 segments $\times$ 1 anchor $\times$ 3 control parameters = 15] dimensions.
- **Loss:** PlasticineLab uses a 3-part loss that encourages the anchors or pins to be closer to the deformable objects while penalizing the distance between the final deformable object shape and the target shape. The only modification we made to the original loss function is increasing the weight of the loss component that encourages the manipulators to get close to the target shape. For more details of the original loss function, please refer to Section 3.1 of the original paper [4]).
- **Landscape and Gradient Characteristics:** the landscape is smooth with local minima, while the high dimensionality makes the problem challenging; gradient quality is good.

A.2.3 Swing

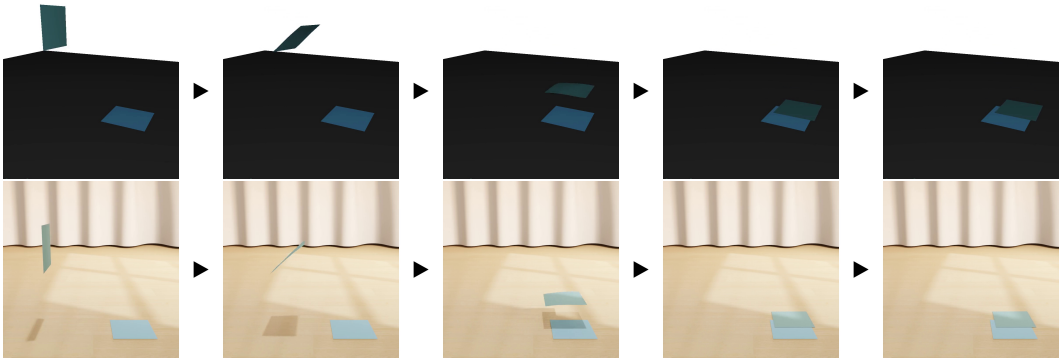

Figure 5: Illustration of the *Swing* environment. The upper and lower rows are renderings of the same episode in the same environment. The upper row uses the built-in realtime rendering engine in DiffTaichi; while the lower row uses Blender, which is slower but has higher quality.

Two anchors grasp the two corners of a 20 $\times$ 20cm piece of cloth and swing it onto the floor. The goal is to make the final cloth configuration as close as possible to a goal configuration.

- **Simulation Framework or Physics Model:** DiffTaichi with mass-spring model as cloth simulation technique; to handle contact, we update the velocities of cloth vertices when contacts occur so that the updated speed is perpendicular to the normal of the contact surface
- **Types of Contacts:** collision between rigid (floor) and deformable objects (cloth)
- **Parameter Dimensionality:** 16 if stiffness is optimized; 3 if initial speed is optimized
- **Parameter Description:** we have two different parameter setups in this task. In the first setup, we split the cloth into $4 \times 4 = 16$ cloth patches and fix the swinging motion. The stiffness values of the 16 cloth patches are optimized. In the second setup, we fix the stiffness of the cloth and optimize the initial 3D velocity of the cloth.
- **Loss:** we have three different loss formulations for this task — loss formulation 'single' optimizes the distance between the center of mass of the cloth at the final frame to a goal position.

Loss formulation 'corner' optimizes the average distance between the four corners of the cloth to their corresponding goal positions. Loss formulation 'mesh' optimizes the average distance between final vertex positions of the cloth to a target cloth mesh.

- **Landscape and Gradient Characteristics:** landscape is rugged. Gradients are noisy in large areas of the parameter space.

A.2.4 Flip

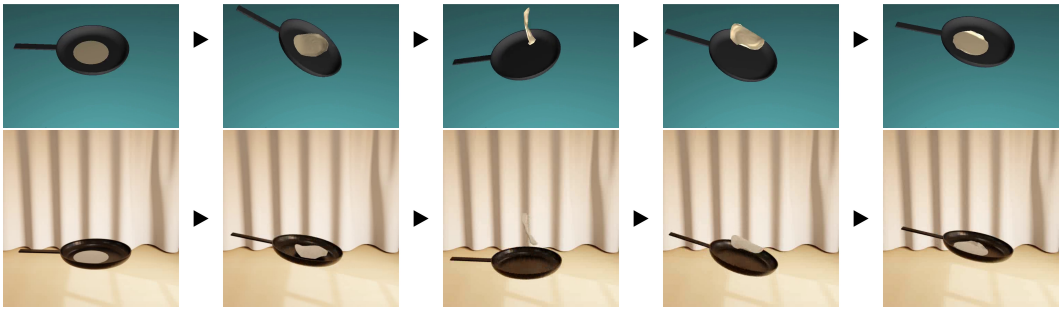

Figure 6: Illustration of the *Flip* environment. The upper and lower rows are renderings of the same episode in the same environment. The upper row uses the built-in realtime rendering engine in DiffTaichi; while the lower row uses Blender, which is slower but has higher quality.

A pancake is placed in a pan with 20cm radius and smooth edges that move and tilt with 3 DOF. The goal is to manipulate the pan so that the pancake is flipped at the end of the episode.

- **Simulation Framework or Physics Model:** DiffTaichi with mass-spring model for simulating the pancake; compared to the Swing task, the stiffness value in this task is smaller to make sure collision forces are not too large during the dynamic movement of the pancake; to handle contact, we update the velocities of pancake vertices when contacts occur so that the updated speed is perpendicular to the normal of the contact surface
- **Types of Contacts:** collision between rigid (floor) and deformable objects (pancake)
- **Parameter Dimensionality:** 15
- **Parameter Description:** an episode is split into 5 equal-length segments. The parameters set the x (left-right) and y (up-down) positions as well as the tilt of the pan at the end of each segment. This leads to [5 segments $\times$ 3 control parameters = 15] dimensions; position and angular velocity of the pan is linearly interpolated within each segment.
- **Loss:** average L2 distance between final position of the four pancake corners (the four pancake vertices with highest and lowest x and y values at the start of the episode) and four corresponding target positions.
- **Landscape and Gradient Characteristics:** landscape is extremely rugged. Gradients are noisy in large areas of the parameter space.

B Additional Analysis

B.1 Analysis of Challenges with Gradients for Rigid Contacts

In this section, we analyze the quality of gradients, observing the nature of discontinuities induced by contact. We define the *Bounce* task (Figure 7), where the goal is to steer a bouncing (red) ball with known friction and elasticity parameters to a target position (green). We seek a policy that imparts an initial 3D velocity $\mathbf{v}_{\text{init}}$ to a ball such that, at the end of the simulation time t_{max}, the center of mass of the ball achieves a pre-specified target position. Importantly, the policy must shoot the ball onto the ground plane, and upon a bounce, reach the target location (this is achieved by restricting the cone of velocities to contain a vertically downward component). This enforces at least one discontinuity in the forward simulation.

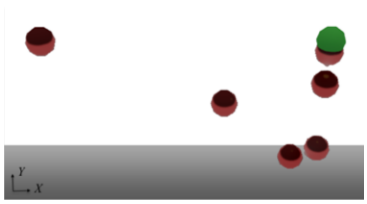

Figure 7: A simple rigid body *Bounce* task implemented using Warp [5]. The ball moves in 3D: X left-right, Y down-up, Z in-out of the image plane.

We use the relaxed contact model from Warp [5], and compute the gradients for a wide range of initial velocities (-10 to $10\ m/s$ along both X and Y directions). Notice that, in the X-direction (horizontal speeds), the gradients are smoother, as changes to X components of the velocity only push the ball further (closer) to the goal, and have little impact on the discontinuities (bounces). However, the Y-axis components of velocities (vertical speeds) tend to have a significant impact on the location and nature of the discontinuities, and therefore induce a larger number of local optima. While prior work (e.g., DiffTaichi [3]) has extensively analyzed gradients through similar contact scenarios, they leverage the conceptually simple (but numerically unstable) Euler integrator and perfectly elastic collisions. Warp [5], on the other hand, uses both a symplectic time integrator and a contact model that includes both friction and elasticity parameters. Our *Bounce* experiments confirm the fact that significant challenges with computing gradients through rigid contacts remain, even in these more recent and advanced differentiable simulation frameworks.

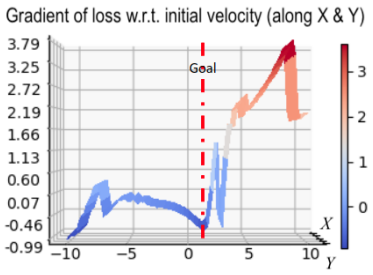

Figure 8: We focus on the Y direction and see that discontinuities caused by rigid contacts remain even in Warp (a recent engine with semi-implicit Euler integration and advanced relaxed-contact & stiffness models).

B.2 Landscape Gallery

In this section, we present more landscape and gradient plots to provide more insight into the differentiable simulation environments we presented in the paper. Apart from this section, we also present animated landscape plots in the supplementary video.

Pinball Below, we show two landscape and gradient plots, one plotting the landscape of the *Pinball* 2D environment with two colliders, and the other plotting a 2D slice for the *Pinball* 16D environment with a grid of 4-by-4 colliders. In the right plot, the x and y axes correspond to the rotation of the center two colliders at the bottom of the collider grid. From the plots, we see that the rugged landscapes occur in different variations of the *Pinball* task.

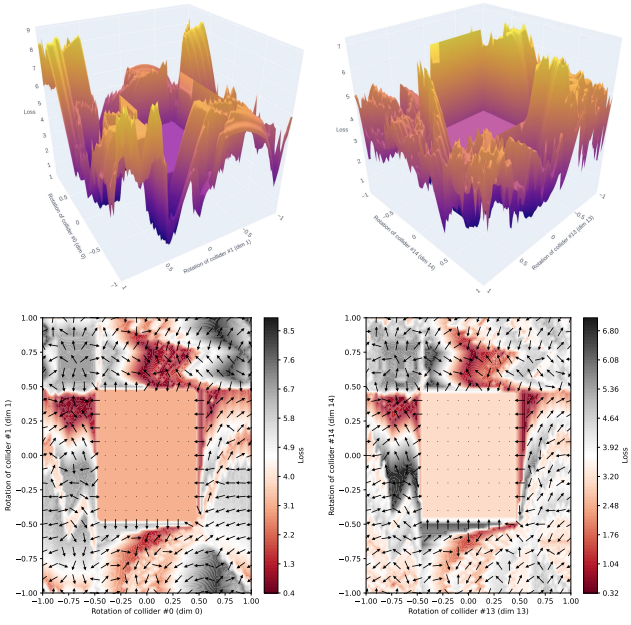

Figure 9: Landscape (top) and gradient (bottom) plots for Pinball environment. *Left column* – Pinball 2D dimensions 0 and 1. *Right column* – Pinball 16D dimensions 13 and 14.

Fluid In the main paper, we presented *Fluid* as an environment where the landscape is rugged and showed one 2D slice of the loss landscape of the 10D environment. Here, we show two more

slices of the landscape (the middle and right plots below). In the plots, we see that the optimization landscape is similarly rugged in these dimensions.

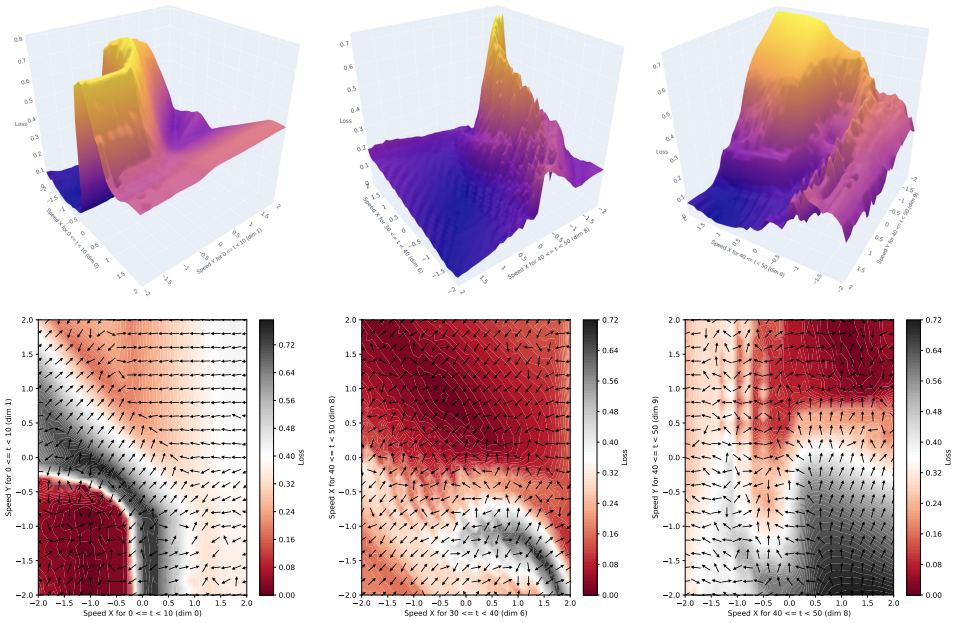

Figure 10: Landscape (top) and gradient (bottom) plots for Fluid environment with 10D parameters. *Left column* – dimensions 0 and 1. *Middle column* – dimensions 6 and 8. *Right column* – dimensions 8 and 9.

Assembly *Assembly* is an environment in *PlasticineLab* where two anchors need to pick up a soft purple ball on the left side of the scene and place it on a yellow stand on the right side. The landscape and gradients plotted below show that these environments have smooth landscapes with local minima and good quality gradients.

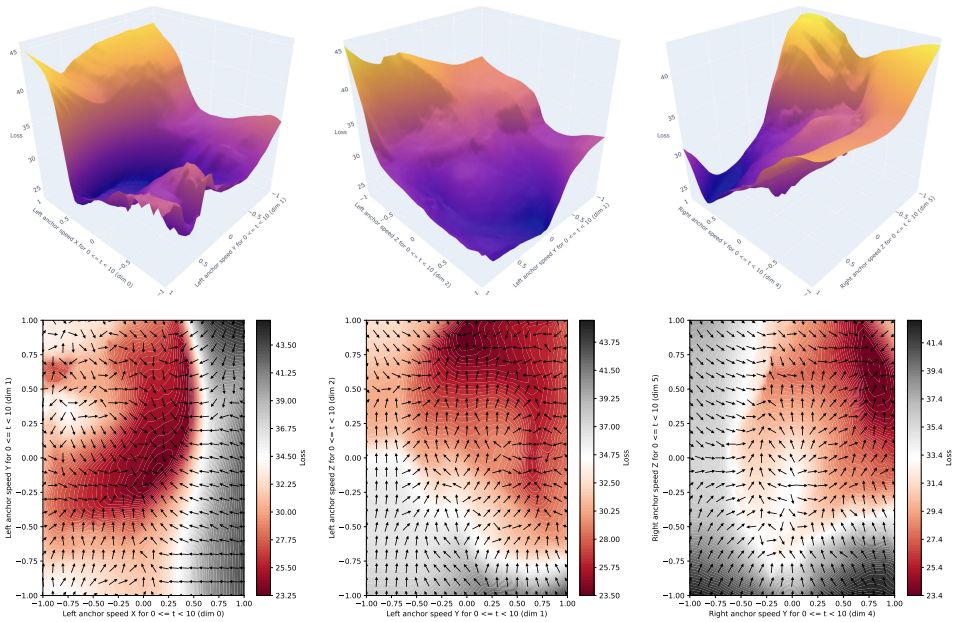

Figure 11: Landscape (top) and gradient (bottom) plots for Assembly environment with 30D parameters. *Left column* – dimensions 0 and 1. *Middle column* – dimensions 1 and 2. *Right column* – dimensions 4 and 5.

Table In the *Table* environment, an anchor pushes one leg of a table so it points outward. The visualizations below reveal that small changes in the action can result in very different loss values.

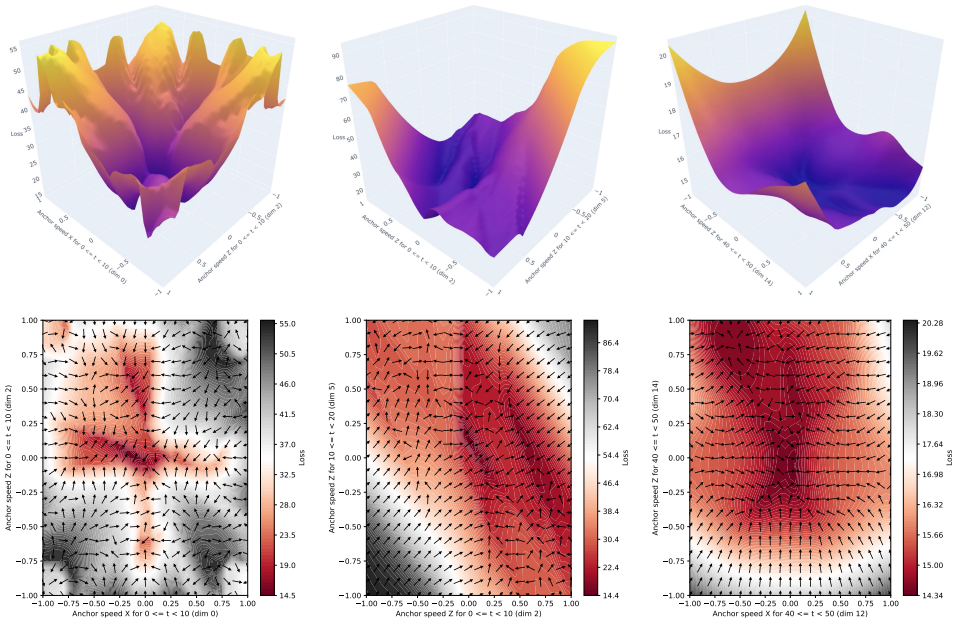

Figure 12: Landscape (top) and gradient (bottom) plots for Table environment with 30D parameters. *Left column* – dimensions 0 and 2. *Middle column* – dimensions 2 and 5. *Right column* – dimensions 12 and 13.

TripleMove In the *TripleMove* environment, six anchors manipulate three blocks to move them to three corresponding goal positions. It can be seen in the plots below that the landscape generated by this environment is pretty rugged.

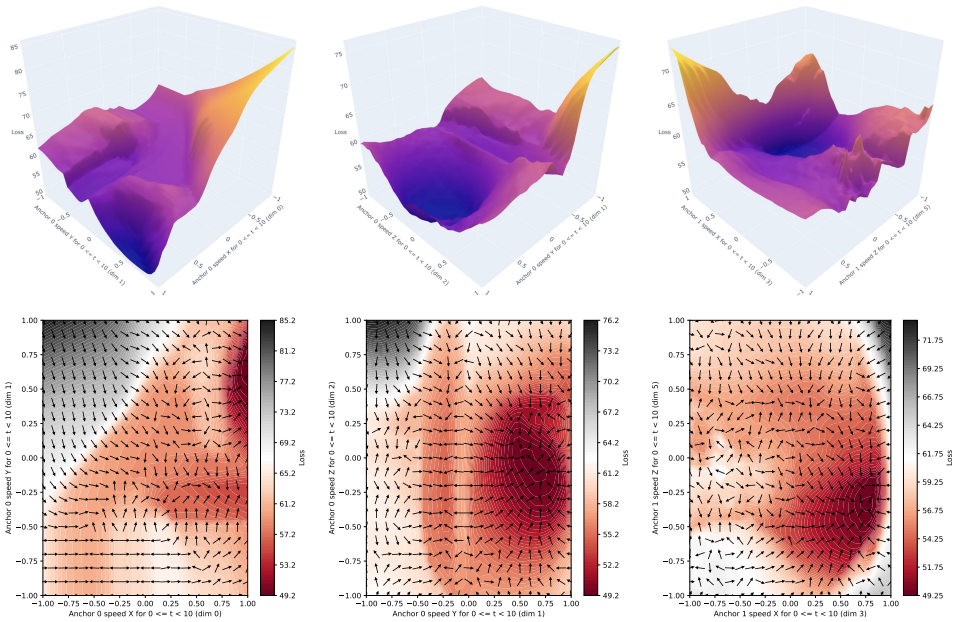

Figure 13: Landscape (top) and gradient (bottom) plots for TripleMove environment with 90D parameters. *Left column* – dimensions 0 and 1. *Middle column* – dimensions 1 and 2. *Right column* – dimensions 3 and 5.

Writer Below we show landscape and gradient plots of the *Writer* environment. We observe that in this environment, the landscape is smoother than that of the previous environments, but there are still a number of local minima at different areas of the landscape (see the left plot and the right plot).

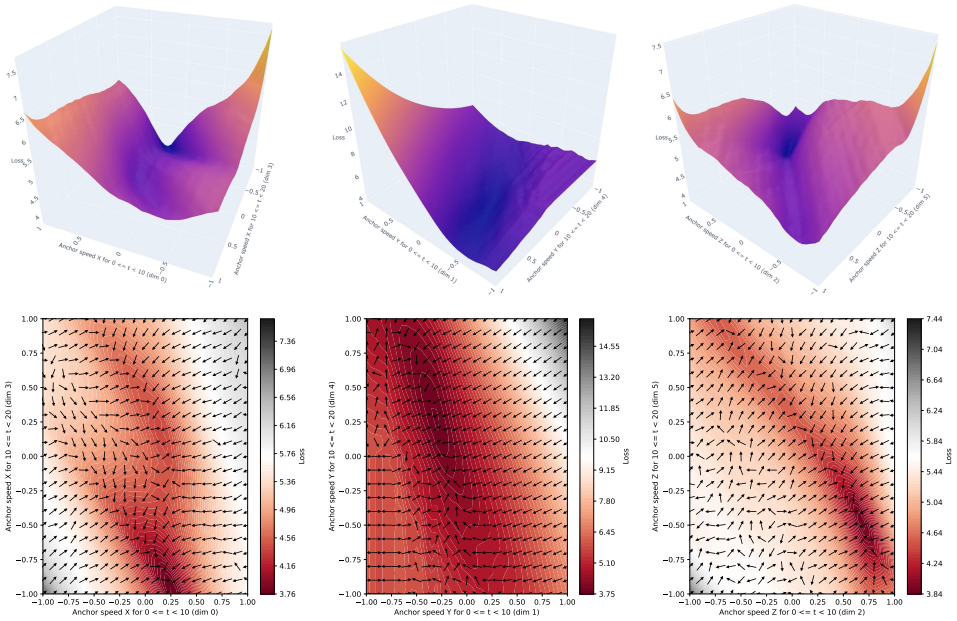

Figure 14: Landscape (top) and gradient (bottom) plots for Writer environment with 15D parameters. *Left column* – dimensions 0 and 3. *Middle column* – dimensions 1 and 4. *Right column* – dimensions 2 and 5.

Flip In the paper, we showed that the *Flip* environment has rugged landscapes with suboptimal gradient quality. Below we show additional landscape and gradient plots of the *Flip* environment to confirm this. As seen in the plots, the landscape is very rugged, and gradients are pointing to rather random directions.

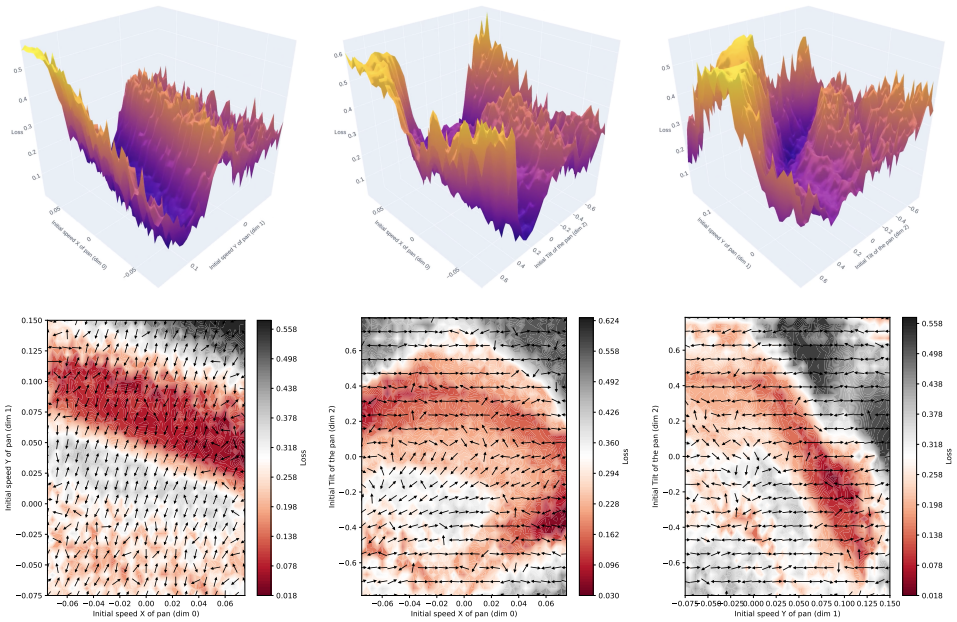

Figure 15: Landscape (top) and gradient (bottom) plots for Flip environment. *Left column* – dimensions 0 and 1. *Middle column* – dimensions 0 and 2. *Right column* – dimensions 1 and 2.

C Details for Method Implementation and Compute Resources

The main paper contains a short outline of the proposed BO-Leap method, and below we include a full version with pseudocode and comments.

Algorithm 1: BO-Leap : Bayesian Optimization with Semi-local Leaps

Initialize : $\boldsymbol{S}_0 \leftarrow \{\}$ ▷ Set of points seen so far (initially empty)
$\mathcal{L} \leftarrow \mathcal{GP}\big(m(\boldsymbol{x})=\boldsymbol{0},\, k(\boldsymbol{x},\boldsymbol{x'})\,|\boldsymbol{S}_0\big)$ ▷ Gaussian process as a global loss model
BO acquisition function $\leftarrow$ LCB ▷ Lower Confidence Bound
for $n = 1..max\_steps$ **do**
  $\mathcal{GP}_{\boldsymbol{S}_n}=\mathcal{GP}\big(m(\cdot), k(\cdot,\cdot)|\boldsymbol{S}_n\big)$ ▷ Compute GP posterior using Eq. 2.25-26 from [6]
  $\boldsymbol{x}_n \leftarrow \arg\min_{\boldsymbol{x}} LCB\big(\boldsymbol{x}|\, \mathcal{GP}_{\boldsymbol{S}_n}\big)$ ▷ Get next simulation (or control) parameter vector
  $\boldsymbol{\mu}_1 \leftarrow \boldsymbol{x}_n;\ \sigma_1 \leftarrow 1.0;\ \boldsymbol{C}_1 \leftarrow \boldsymbol{I}$ (identity); K=10 ▷ Initialize local population distribution
  for $i = 1..local\_steps$ **do**
    $\boldsymbol{x}_k \sim \mathcal{N}\big(\boldsymbol{\mu}_i, \sigma_i^2\boldsymbol{C}_i\big), k = 1..K$ ▷ Sample K population candidates
    $l_k \leftarrow Sim(\boldsymbol{x}_k), k = 1..K$ ▷ Compute losses l_k with $\boldsymbol{x}_k$ as sim/control parameters
    $K_{best} \leftarrow sort(l_k)$ ▷ Get candidates with the lowest loss
    $\boldsymbol{s}_1 = \frac{1}{|K_{best}|}\sum_{k\in K_{best}} \boldsymbol{x}_k$ ▷ Compute descent start point (CMA-ES population mean)
    for $j = 1..J$ **do**
      $l_k, \nabla_{Sim}|_{\boldsymbol{s}_j} \leftarrow Sim(\boldsymbol{s}_j)$ ▷ Compute sim. loss l_j and gradients $\nabla_{Sim}|_{\boldsymbol{s}_j}$
      $\boldsymbol{s}_j \leftarrow \boldsymbol{s}_{j-1} - \alpha\nabla_{Sim}|_{\boldsymbol{s}_j}$ ▷ Take a gradient step
      Break if l_j stagnates for more than 3 steps
    $\boldsymbol{\mu}_{i+1} \leftarrow \boldsymbol{s}_J;\ \sigma_{i+1}, \boldsymbol{C}_{i+1} \leftarrow$ Eq.14-17 from [7] ▷ Update local population distribution
    $\boldsymbol{S}_{n+1} = \boldsymbol{S}_n \cup \{(\boldsymbol{s}_j, l_j)\}_{j=1}^{J} \cup \{(\boldsymbol{x}_k, l_k)\}_{k=1}^{K}$ ▷ Update data for GP posterior

For implementing Bayesian optimization (BO), we used the BOTorch library [8], which supports various versions of Gaussian processes (exact & approximate) and various BO acquisition functions.

We experimented with various versions of Gaussian process (GP) models, including exact GP and sparse variational GP versions that are provided in BOTorch. BOTorch uses the lower-level GPyTorch library [9] for GP implementations. We found that exact GPs performed best, hence we used it for all BO experiments reported in the main paper. In future work, it would be interesting to experiment with other implementations of approximate GPs, to support posteriors with a much larger number of points.

For all experiments described in the main paper, we used the Lower Confidence Bound (LCB) acquisition function, with default parameters (i.e. exploration coefficient $\alpha = 1.0$). In our previous experience, LCB had a similar performance as other commonly used functions (such as the Expected Improvement acquisition function), and LCB has the advantage of being very easy to implement and interpret. See Section IV in [10] for more information. BOTorch also implements automatic hyperparameter optimization based on maximizing the marginal likelihood (see [10], Section V-A). We used this for all our BO-based experiments.

For implementing CMA-ES, we used a fast and lightweight library provided by [11].

For implementing PPO and SAC, we used the Stable Baselines3 library provided by [12]. We parameterize our policy as 3-layer MLPs with hidden layer sizes of 64. We used default state representations in each environment as the observation space used for RL. In particular, in Cartpole, we used the joint position and velocities of the cartpole; in Fluid, we used the object and ladle positions; in PlasticineLab environments, we used positions and velocies of deformable object particles and manipulating anchors as observations. We use default action spaces in each environment for our RL baseline. To make sure the RL baselines take a similar number of actions to other baselines and methods, we use action repeats in all our environments. In cartpole, as other methods take actions once per timestep, action repeat is set to 1; in other environments, action repeat is set to 5. We set a fixed horizon without early termination in all our environments. As for reward function, we use sparse reward in all environments, providing the negative episode loss to the agent at the final step of the episode as reward. RL baselines are trained for 1,000 interaction episodes, which is the same in amount of interaction as all other methods and baselines we reported in the paper.

Our experiments were performed using NVIDIA Tesla T4 GPUs and 32 cores of an Intel Xeon 2.3GHz CPU. The computational requirements of each simulation environment differ widely. To give a general sense of the resources required: our environments based on Warp, Nimble and mesh-based DiffTaichi were fastest, requiring only a few minutes for 1,000 episodes (including gradient computations). Particle-based DiffTaichi environments (*Fluid* and the PlasticineLab environments) required significantly more time. For example, *Fluid* took ≈ 2 hours for 1,000 episodes (including gradient computations).

References

[1] 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 *Robotics: Science and Systems*, 2021.

[2] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics*, 37(4):150, 2018.

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

[4] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan. PlasticineLab: A soft-body manipulation benchmark with differentiable physics. 2021.

[5] M. Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[6] C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[7] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[8] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020. URL http://arxiv.org/abs/1910.06403.

[9] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. GPyTorch: Blackbox Matrix-matrix Gaussian Process Inference with GPU Acceleration. *Advances in neural information processing systems*, 31, 2018.

[10] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[11] M. N. Masashi Shibata, Hideaki Imamura. A Lightweight Covariance Matrix Adaptation Evolution Strategy (CMA-ES) Implementation. URL https://github.com/CyberAgentAILab/cmaes.

[12] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.