Efficient Tactile Simulation with Differentiability for Robotic Manipulation

Jie Xu[1], **Sangwoon Kim**[1], **Tao Chen**[1], **Alberto Rodriguez**[1], **Pulkit Agrawal**[1], **Wojciech Matusik**[1], **and Shinjiro Sueda**[2]

[1]Massachusetts Institute of Technology [2]Texas A&M University

Abstract: Efficient simulation of tactile sensors can unlock new opportunities for learning tactile-based manipulation policies in simulation and then transferring the learned policy to real systems, but fast and reliable simulators for dense tactile normal and shear force fields are still under-explored. We present a novel approach for efficiently simulating both the normal and shear tactile force field covering the entire contact surface with an arbitrary tactile sensor spatial layout. Our simulator also provides analytical gradients of the tactile forces to accelerate policy learning. We conduct extensive simulation experiments to showcase our approach and demonstrate successful zero-shot sim-to-real transfer for a high-precision peg-insertion task with high-resolution vision-based GelSlim tactile sensors. The videos and code are available at: http://tactilesim.csail.mit.edu.

Keywords: Tactile Simulation, Tactile Manipulation, Differentiable Simulation, Sim-to-Real

1 Introduction

Just as humans heavily rely on rich and precise tactile cues for dexterous grasping and in-hand manipulation tasks, robots can also utilize tactile cues as an important source of sensing for interacting with the surrounding environments, especially when the visual information is unavailable or occluded. With the recent development of various tactile sensors capable of generating dense normal or shear load information [1, 2, 3, 4], researchers have been exploring how to leverage this important mode of information for robotic manipulation tasks. With the dense tactile *normal* load field, the static spatial relation between the object and the robot manipulators can easily be inferred, which is useful for tasks such as edge following [5], pose estimation [6], object reconstruction and recognition [7, 8]. On the other hand, the dense tactile *shear* force feedback more readily gives rich information about the dynamic tangential motions between the object and the manipulators, and thus can be utilized in tasks such as stable grasp [9], precise insertion [10, 11], and slip detection [12, 13, 14, 15]. However, most of the tactile manipulation work still requires a significant amount of human effort on real hardware systems for collecting data, cleverly building automatic resetting mechanisms, and carefully designing the learning strategy [10]. Such manual work can be time-consuming, cost expensive, and more importantly unsafe during policy exploration.

Due to its capability to replicate the real world with high fidelity and low cost, physics-based simulation has become a powerful recipe for learning robotic control policies [16, 17, 18, 19]. Previous work has demonstrated that the policy can be efficiently learned in simulation and successfully transferred to real robots via proper sim-to-real techniques [20, 21, 22, 23]. Despite the prevalence of simulation and the importance of tactile sensory in robotics, physics-based simulation is still under-explored to efficiently simulate dense tactile normal and shear force fields for robotic applications. Most popular simulators [18, 19] only support force-torque sensors which are attached to each robot link, only producing the contact force values at a few points on each body. Although one can acquire a dense tactile force field by attaching many small cuboids to the robot body and querying the force sensor on each small cuboid from simulation [24, 25], the obtained tactile force values are usually sparse and are unable to match the uniform force distribution on a real elastic tactile sensor such as GelSlim [1]. While researchers have also tried simulating realistic tactile feedback via pure ge-

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

ometric methods [5, 26], such methods typically only compute the normal tactile force and cannot simulate the tactile effects in shear directions. On the other hand, the tactile shear forces have been successfully simulated via the finite element method (FEM) [27, 28, 29] or data-driven approach [30], but these simulators suffer from expensive computation costs, and cannot be easily used for data-hungry policy-learning approaches such as reinforcement learning (RL).

We present a novel tactile simulator that can efficiently and reliably simulate both normal and shear tactile force fields covering the entire contact surface. We build upon rigid body dynamics formulation and develop a fast penalty-based tactile model which can run at 1000 frames/s for a ball-rolling experiment with a high-resolution tactile field on a single core of Intel i7-9700 CPU. Our tactile model can reasonably approximate the soft contact nature of soft tactile sensor material such as the elastomer used in GelSlim [31], generate dense tactile force fields (*e.g.,* the dense marker array on GelSlim), and is compatible with arbitrary tactile sensor spatial layout (*i.e.,* flat plane, hemisphere, etc.). Furthermore, our compact tactile formulation is differentiable, which allows the simulator to provide fast analytical gradients for the entire dynamics chain. We conduct extensive experiments in simulation to demonstrate the capabilities of our tactile simulator, including policy learning with reinforcement learning algorithms and gradient-based algorithms. We also conduct a zero-shot sim-to-real experiment for a high-precision tactile-based peg-insertion task, demonstrating that our simulator provides realistic tactile simulation.

2 Related Work

While there have been many physics-based simulators developed to simulate various types of robots, efficiently and reliably simulating dense tactile sensing fields is less explored. As mentioned above, most robotics simulators such as MuJoCo [18] and PyBullet [19] only support force-torque sensors that are attached to each robot link. While it is possible to augment these simulators with high-resolution tactile forces, they become computationally cumbersome. In order to acquire more realistic and dense tactile forces, Narang et al. [27, 28] and Ding et al. [29] use soft materials to model the tactile sensors and apply FEM to simulate the deformation and force fields of the tactile sensors. Despite the high fidelity of the simulated tactile feedback, these simulators suffer from expensive computation cost and are primarily used to collect supervised tactile dataset instead of learning policies which are typically data-hungry and requires fast simulations. Vision-based tactile sensors produce high-resolution tactile feedback. To simulate vision-based sensors, Wang et al. [26] and Church et al. [5] use PyBullet [19] and render the intersecting part between the object and the tactile manipulator as depth images, from which tactile information is generated. However, such purely geometry-based approaches cannot simulate the tactile effects in the shear directions such as the marker displacements of GelSlim. Si and Yuan [30] compute the marker displacement field by presenting a superposition method to approximate the FEM dynamics. While they are able to simulate the tactile shear effects, the speed of the simulation is still slow, and no control tasks are demonstrated. Bi et al. [32] build an efficient simulation specialized for a tactile-based pole swing-up task with a customized vision-based tactile sensor, but the proposed technique is not readily extensible to simulate other types of tasks and tactile sensor types. Similarly to our work, Habib et al. [33] use a spring-mass-damper model to simulate tactile normal forces, and Moisio et al. [34] use a soft bristle deflection model for simulating the tactile forces. However, there are no control tasks demonstrated to be learned with the presented simulators and no gradient information is available. In contrast to these previous works, we present a generic simulator with analytical gradients for tactile forces by leveraging the penalty-based rigid body dynamics, and we demonstrate that our simulation is efficient enough for policy learning, and simulated tactile force field can be successfully used for a sim-to-real task on the high-resolution vision-based GelSlim sensor.

3 Method

We now present our approach to simulate tactile forces for real-world tactile sensors. In §3.1, we introduce our flexible representation for tactile sensors. In §3.2-3.3, we present our penalty-based tactile model for simulation and derive the analytical gradients of the dynamics. In §3.4, we describe our intermediate tactile signal representation for the sim-to-real transfer of the policies.

3.1 Tactile Sensor Representation

Each tactile point i on a sensor pad is represented by a tuple $\langle \mathbf{B}_i, \mathbf{E}_i, \xi_i \rangle$ as shown in Fig. 1. $\mathbf{B}_i$ is the rigid body the tactile point is attached to, and $\mathbf{E}_i \in \mathrm{SE}(3)$ is the position/orientation of the point in the local coordinate frame of the body, with the $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ axes in the shear-direction plane and the $\boldsymbol{z}_i$ axis along the normal tactile direction. (These axes are the same for all points for a *planar* sensor pad.) Finally, ξ_i are the simulation parameters of the penalty-based tactile model, which will be introduced later in §3.2. Our representation of tactile points is flexible, allowing us to specify any number of points in arbitrary geometry layouts on a robot, and each tactile sensor can have its individual configuration parameters.

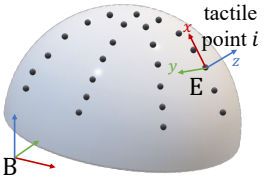


Figure 1: Tactile Sensor Representation.

3.2 Penalty-based Tactile Model

We use a penalty-based tactile model to characterize the force on each tactile point. For each point $\langle \mathbf{B}_i, \mathbf{E}_i, \xi_i \rangle$, we use the following contact model [35] to obtain the contact force at the tactile point's location represented in the local coordinate frame $\mathbf{B}_i$. (For brevity, we drop the subscript i.)

$$\boldsymbol{f}_n = (-k_n + k_d \dot{d}) d \boldsymbol{n}, \qquad \boldsymbol{f}_t = -\frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|} \min(k_t \|\boldsymbol{v}_t\|, \mu \|\boldsymbol{f}_n\|), \tag{1}$$

where $\boldsymbol{f}_n$ is the contact force at the tactile point along the contact normal direction $\boldsymbol{n}$, and $\boldsymbol{f}_t$ is the contact friction force in the plane tangential to the contact normal direction. The scalar d (nonpositive) is the penetration depth between the point and the collision object, and $\dot{d}$ is its time derivative. The vector $\boldsymbol{v}_t$ is the relative velocity at the contact point along the contact tangential direction. Scalars k_n, k_d, k_t, μ are contact stiffness, contact damping coefficient, friction stiffness, and coefficient of friction respectively, and they together form the simulation parameters ξ of the tactile point: *i.e.,* for the i^{th} tactile point, $\xi_i = \{k_n^i, k_d^i, k_t^i, \mu^i\}$. After the frictional contact force is computed for each point as $\boldsymbol{f} = \boldsymbol{f}_n + \boldsymbol{f}_t$, we transform this force into the local coordinate frame of the tactile point to acquire the desired *shear* and *normal* tactile force magnitudes:

$$T_{sx} = \boldsymbol{f}^\top \boldsymbol{x}, \qquad T_{sy} = \boldsymbol{f}^\top \boldsymbol{y}, \qquad T_n = \boldsymbol{f}^\top \boldsymbol{z}, \tag{2}$$

where $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ are the axes of frame $\mathbf{E}$.

Our penalty-based tactile model can be integrated into any simulator as long as the required values, such as the world-frame location of the tactile points, the contact normal, the collision penetration depth and its time derivatives, can be acquired from the simulator. We implement our tactile model in C++ and integrate it into differentiable RedMax (DiffRedMax) [35, 36] since DiffRedMax is open-source and readily provides all the required information for our computation, and more importantly, its differentiability allows us to make our tactile simulation differentiable with a moderate amount of modifications to its backward gradients computation.

3.3 Differentiable Tactile Simulation

Since we use an implicit time integration scheme for forward dynamics, the core step of gradient computation is to differentiate through the nonlinear equations of motion. We start by formulating a finite-horizon tactile-based policy optimization problem:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \;\; \mathcal{L} = \sum_{t=1}^{H} \mathcal{L}_t\big(\boldsymbol{u}_t, \boldsymbol{q}_t, \mathbf{v}_t(\boldsymbol{q}_t)\big) \tag{3a}$$

$$\text{s.t.} \;\; g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0 \qquad \text{(Equations of Motion)} \tag{3b}$$

$$\boldsymbol{u}_t = \pi_\theta\big(\tilde{\boldsymbol{q}}_{t-1}, \tilde{\mathbf{v}}_{t-1}(\boldsymbol{q}_{t-1}), T_{t-1}(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1})\big). \qquad \text{(Policy Execution)} \tag{3c}$$

Here, H is the task horizon, $\mathcal{L}_t$ is a step-wise task-dependent reward function, $\boldsymbol{u}$ is the action (*e.g.,* joint torque), $\boldsymbol{q}$ is the simulation state (*i.e.,* joint angles), and $\mathbf{v}$ is the derived auxiliary simulation variables (*e.g.,* fingertip positions) which themselves are a function of $\boldsymbol{q}$. Eq. 17b describes the nonlinear equations of motion (§A.1). Eq. 17c represents the inference of the control policy π_θ to obtain the desired action given the partial observation of the simulation state $\tilde{\boldsymbol{q}}$, partial observation of the simulation computed variables $\tilde{\mathbf{v}}$, and the tactile force values T from Eq. 8.

We compute the gradients $d\mathcal{L}/d\theta = (\partial \mathcal{L}/\partial \boldsymbol{u}_t)(\partial \boldsymbol{u}_t/\partial \theta)$ for policy optimization. We embed our simulator as a differentiable layer into the PyTorch computation graph and use reverse mode differentiation to backward differentiate through dynamics time integration. The first gradient, $\partial \mathcal{L}/\partial \boldsymbol{u}_t$,

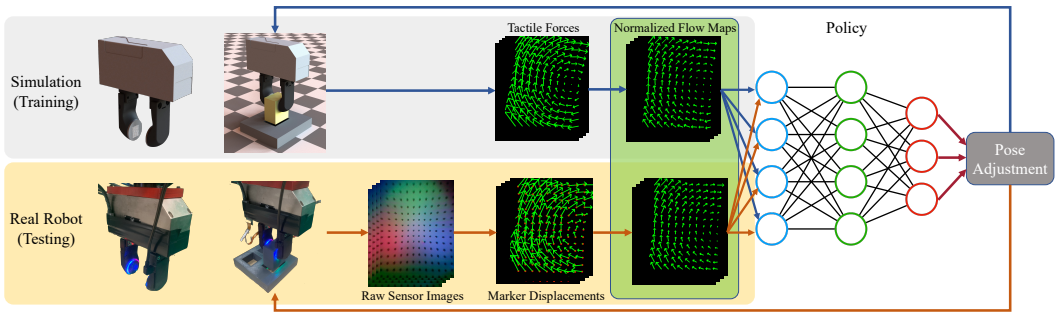


Figure 2: Sim-to-Real Pipeline for Insertion Task (§4.5). *Gray Box:* During training, we convert the tactile force output from the simulator into the normalized flow map representation (shaded in green). *Yellow Box:* When executing the policy on a real system, we convert the sensor output into the same normalized flow map. This intermediate representation is then treated as the observation input to a neural network policy to output the pose adjustment for the next attempt. Here we only visualize the tactile output from one tactile sensor pad.

which includes the simulation dynamics and tactile derivatives, is derived analytically, as shown below. The second gradient, $\partial \boldsymbol{u}_t / \partial \theta$, is computed by PyTorch's auto-differentiation.

We compute the gradients in reverse order. At each time step t, we derive $\partial \mathcal{L} / \partial \boldsymbol{u}_t$ given the analytically computed gradients with respect to the system states ($\partial \mathcal{L} / \partial \boldsymbol{q}_t$), auxiliary variables ($\partial \mathcal{L} / \partial \mathbf{v}_t$), and tactile forces ($\partial \mathcal{L} / \partial T_t$) (we drop the terms related to the time integrator here for brevity; see the full derivation in §A.2-A.3)):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} = \underbrace{\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{u}_t}}_{\boldsymbol{a}} + \underbrace{\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{v}_t} \frac{\partial \mathbf{v}_t}{\partial \boldsymbol{q}_t} + \frac{\partial \mathcal{L}}{\partial T_t} \left(\frac{\partial T_t}{\partial \boldsymbol{q}_t} + \frac{\partial T_t}{\partial \dot{\boldsymbol{q}}_t} \frac{\partial \dot{\boldsymbol{q}}_t}{\partial \boldsymbol{q}_t} \right) \right)}_{\boldsymbol{b}} \underbrace{\frac{\partial \boldsymbol{q}_t}{\partial \boldsymbol{u}_t}}_{-A^{-1}D} . \tag{4}$$

The right-most derivative can be computed by applying the implicit function theorem on Eq. 17b, which gives us $\partial \boldsymbol{q}_t / \partial \boldsymbol{u}_t = -(\partial g / \partial \boldsymbol{q}_t)^{-1} (\partial g / \partial \boldsymbol{u}_t)$. Writing this as $\partial \boldsymbol{q}_t / \partial \boldsymbol{u}_t = -A^{-1} D$ and combining with Eq. 4, we first solve the linear system $A^\top \boldsymbol{c} = \boldsymbol{b}^\top$ for $\boldsymbol{c}$, and then we compute the final gradient as $\partial \mathcal{L} / \partial \boldsymbol{u}_t = \boldsymbol{a} - \boldsymbol{c}^\top D$ using the adjoint approach (more details in §A.3).

3.4 Normalized Tactile Flow Map for Sim-to-Real

We use the GelSlim 3.0 sensor [4], which utilizes small markers to track motions in the shear direction, to demonstrate the sim-to-real capability (§4.5). There is an unavoidable sim-to-real gap between our simulator that emulates tactile forces and the physical GelSlim sensor that relies on imaging. In this section, we demonstrate how we overcome this gap by constructing a common intermediate tactile representation for policy input observation. We assume that the stiffness of the sensor along different shear directions is isotropic and that there exists a linear relationship between the displacement and the contact *shear* forces (T_{sx} and T_{sy} from Eq. 8) at each tactile point. We connect these two different sensor output formats via a unitless normalized tactile flow map representation.

Specifically, we use the raw tactile sensor images from the past k steps from the n tactile sensor pads on a real robot as our policy observation $T_{\text{image}}^{\{1:k,1:n\}}$. As shown in Fig. 2, we first detect and identify the marker positions in each image, and obtain the marker displacement field $T_{\text{displacement}}^{\{1:k,1:n\}} \in \mathbb{R}^{r \times c \times 2}$ by subtracting the marker positions in the rest configuration from their positions in the deformed configuration, where r and c are the rows and columns of the tactile marker array on each sensor pad, with each marker giving us the x and y displacement information. Then we normalize the displacement field so that the maximal length of the marker displacement across all tactile points and images is of unit length; *i.e.,*

$$T_{\text{normalized}}^{\{1:k,1:n\}} = \frac{T_{\text{displacement}}^{\{1:k,1:n\}}}{\max\left(\max_{k,n,r,c}(\| T_{\text{displacement}}^{\{k,n\}}(r,c) \|), \epsilon \right)}, \tag{5}$$

where ϵ ensures that the output is zero when there is no any displacement on the markers (*i.e.,* no contact). We assemble these flow maps into a single tensor $T_{\text{normalized}} \in \mathbb{R}^{k \times n \times r \times c \times 2}$, which is

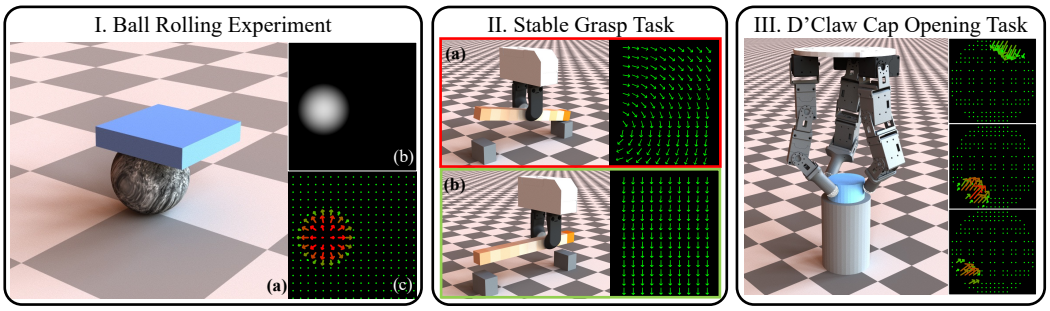


Figure 3: (I) Ball rolling experiment: The tactile sensors are installed on the lower surface of the pad. The depth map of the tactile normal forces is shown in (b). The tactile force field is shown in (c) with the arrow denoting the shear forces and the color denoting the magnitude of the normal force. **(II) Stable Grasp Task:** The bar is composed of 11 blocks with random densities (the deeper the color, the heavier the block). (a) An unsuccessful grasp results in rotational patterns in the tactile force field and (b) a successful grasp requires the gripper to adjust the grasp location to the center of mass of the bar. **(III) D'Claw Cap Opening Task:** The tactile sensors (white dots) are installed at the three hemisphere fingertips of the hand. We map each tactile point at one fingertip onto a 2D image plane and visualize the tactile forces field of three fingertips on the right.

our normalized tactile flow map representation. For the tactile shear forces $\{T_{sx}^{\{1:k,1:n\}}, T_{sy}^{\{1:k,1:n\}}\}$ acquired from the simulation, we conduct the same normalization process as in Eq. 5.

Intuitively speaking, the normalized tactile flow map provides directional information about the relative motion of the markers induced by the contact forces, and it also keeps the relative tactile load magnitude relationships among different sensors and different time steps so as to preserve meaningful spatial and temporal information about the contact. For our sim-to-real experiments, we only use the shear directional information from the sensor, but the same technique can also be applied to the normal directional information by normalizing the depth map of the contact surface reconstructed from the GelSlim image [4] across different frames and different sensor pads.

4 Experiments

We conduct extensive tactile-based experiments to demonstrate the capability of our approach.[1] We investigate the following questions: (§4.1, §4.2) Can our simulator reliably simulate the high-resolution tactile force field at a high speed for RL algorithms? (§4.3) Does the differentiability of our simulator provide advantages in policy learning? (§4.4) Is our tactile sensor representation flexible enough for sensors with arbitrary geometrical layouts? (§4.5) How does our simulated tactile force field compare to the tactile feedback from real sensors, and does our normalized tactile flow map representation help to transfer the policies learned in the simulator to a real robot?

4.1 Speed and Reliability: High-Resolution Tactile Ball Rolling Experiment

We design a ball rolling experiment to show the efficacy of the tactile force field generated by our simulator and to test the simulation speed. The simulation setup is shown in Fig. 3(I). A high-resolution tactile pad (200×200 markers) touches the marble ball and moves it around. The simulation step size $h = 5$ ms, and we compute the tactile force field every 5 steps (*i.e.,* 40 Hz). Fig. 3(I) also shows the normal tactile force (represented by a depth map) and the tactile shear forces acquired from our simulator. For this example, our simulation runs at 1050 frames per second (FPS) on a single core of Intel Core i7-9700K CPU. The simulation speed can be further accelerated by simply parallelizing it across multiple CPU cores, as we do in the RL experiments.

4.2 RL Training: Tactile-Based Stable Grasp Task

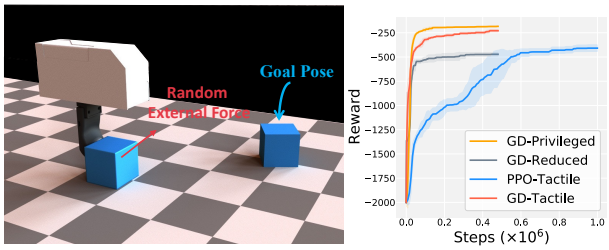
Our tactile simulator provides the shear force information on the contact surfaces, which is critical for many manipulation tasks. Inspired by the setup in [9], we show the usage of shear force information for control and the effectiveness of our tactile simulator in a parallel-jaw bar grasping task.

[1] See also the supplementary video.

As shown in Figure 3(II), the task requires a WSG-50 parallel-jaw gripper to stably grasp a bar with *unknown mass distribution* in fewer than 10 attempts. The gripper has two tactile sensors with a tactile marker resolution of 13×10. The bar is composed of 11 blocks where the density of each block is randomized. The total mass of the bar is in the range $[51, 120]$ g. We consider a grasp to be a failure if the bar tilts more than 0.02 rad after the gripper grasps a bar.

We use RL to train a control policy that determines the grasp location. The initial grasp location is the geometric center of the bar. Based on the tactile sensor readings (the only observation input to the policy), the policy outputs a delta change in the grasping location. The policy is a shallow CNN (*Conv-ReLU-MaxPool-Conv-ReLU-FC-FC*) that takes as input the two tactile sensor readings. We train the policies with PPO [37] using 32 parallel environments with 20K environment steps in total. We train the policies with 3 different random seeds and test them 320 times. The success rate is $98.5 \pm 1.8\%$. The average number of attempts taken to stably grasp the bars is 2.1, meaning the policy can compute the correct grasping location after a single failed attempt in most cases.

4.3 Differentiability: Tactile-Based Box Pushing Task



(a) (b)

Figure 4: Tactile-based box pushing task. (a) The goal of the gripper policy is to use its tactile feedback to push a box to a randomized target position/orientation. A time-varying external force is randomly applied to the box during the task. **(b)** the training curve for each policy variation is averaged from the five independent runs with different random seeds.

	Pos. Error	Ori. Error
GD-Privileged	$0.037 \pm 0.002m$	$0.043 \pm 0.003°$
GD-Reduced	$0.126 \pm 0.009m$	$0.255 \pm 0.021°$
PPO-Tactile	$0.123 \pm 0.034m$	$0.241 \pm 0.123°$
GD-Tactile	$\mathbf{0.058 \pm 0.003}m$	$\mathbf{0.074 \pm 0.020}°$

Table 1: Metrics comparison on box pushing task. We compute the final position/orientation errors of the best policy in each run and average the metrics from five runs for each policy variation. *GD-Privileged* gives a reference of the best possible metrics, and without the privileged state information of the box, our *GD-Tactile* achieves much better position error and rotation error than the other two variations.

In this experiment, we design a box pushing task similar to [5] to demonstrate how we can leverage the provided analytical gradients to help learn tactile-based control policies better and faster.

Task Specification As shown in Fig. 4(a), the task here is to use the same WSG-50 parallel jaw gripper as §4.2 (with only one finger kept) to push the box to a randomly sampled goal location and orientation. The initial position of the box is randomly disturbed. A random external force is applied continually on the box, which changes every 0.25 s. More details of the task are in §B.3.

Comparing Policy Learning Algorithms We train the control policies through four different combinations of learning algorithms and observation spaces. *GD-Privileged*: This variation uses the gradient-based optimizer Adam by utilizing the analytical policy gradients computed from our differentiable simulation. The policy observation contains all the privileged state information of the gripper, the box, and the goal. This policy provides an upper-bound performance reference. *GD-Reduced*: Similar to *GD-Privileged*, except that the observation space only contains the state information that can be acquired on a real system such as the gripper state and the goal. *GD-Tactile*: Other than the state information used in *GD-Reduced*, we also include the tactile sensor readings in the policy input. The policy is trained using the analytical policy gradients. *PPO-Tactile*: Similar to *GD-Tactile*, but the policy is trained by PPO.

All the policies are trained to maximize the same reward function (§B.3). We run each variation five times with different random seeds, and plot their training curves averaged from five runs in Fig. 4(b). We also randomly sample 300 goal poses and measure the final position and orientation errors between the box and the goal pose of the best policy from each run, and report the average metrics across five seeds in Table 1. The results show that when neither state information of the box nor tactile information is available (*i.e., GD-Reduced*), the policy cannot reliably push the box to the target location since the gripper has no clue when the box goes outside of the control of the gripper

due to the random initial box position perturbation and the random external forces. With tactile information feedback (*i.e., PPO-Tactile* and *GD-Tactile*), the gripper has tactile information to keep the gripper touching the box, allowing it to push the box to the goal effectively. However, the high dimensional tactile observation space results in higher computational costs with PPO, which relies on stochastic samples to estimate the policy gradients. In contrast, with the help of our differentiable tactile simulation, *GD-Tactile* makes use of the analytical policy gradients and leads to faster policy learning and better policy performance.

4.4 Flexibility: Tactile Sensor Simulation on Curved Surfaces

To demonstrate that our method supports tactile sensors on curved surfaces, we train a D'Claw [38] tri-finger hand to open a cap on a bottle. We put the tactile sensors on the three rounded fingertips as shown in Fig. 3(III). The sensor layout on each fingertip is a hemisphere, and we use 302 evenly-spaced tactile markers. We build a coordinate mapping to project the marker positions on the 3D surface into a 20×20 2D array (with some empty values around the boundary). Fig. 3(III) shows that our simulation can produce reliable and realistic tactile sensor readings on a rounded fingertip.

The task is to open a cap using the D'Claw hand. The position and the radius of the cap are randomized and unknown. There is also unknown random damping between the cap and the bottle. The task is considered a success if the cap is rotated by $45°$. The only observation data that the policy gets are the angles of each joint, fingertip positions, and tactile sensor readings. This task is similar to how we open caps by just using proprioception sensory data and tactile feedback on the fingers without knowing the exact size and location of the cap. The policy outputs the delta change on the joint angles. We again use PPO to train the policy (a shallow CNN) using 32 parallel environments. To show that the tactile sensors are useful in this task, we also train a baseline policy (a simple MLP policy) where the policy only takes as input the joint angles and fingertip positions. With tactile sensor readings, the policies learn significantly faster and achieve an 87.3% success rate, while the policies only achieve a 59.7% success rate when tactile sensor information is unavailable. More details about task setup and results are provided in §B.4.

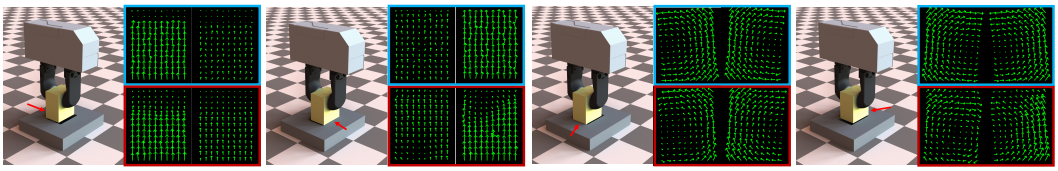
4.5 Zero-Shot Sim-to-Real: Tactile RL Insertion Task

In this experiment, we show the quality of the simulated tactile feedback when compared to the tactile sensing obtained from the real system, and demonstrate how to do zero-shot transfer for the policies learned in simulation to the real robot via our normalized tactile flow map representation.

Task Specification We experiment on the tactile-RL insertion task similar to [10]. In this task, a gripper (same as in §4.2) is controlled to insert a cuboid object into a rectangle-shaped hole with a random initial pose misalignment. The insertion process is modeled as an episodic policy that iterates between open-loop insertion attempts followed by insertion pose adjustments (shown in Fig. 2). The robot has up to 15 pose correction attempts, and the robot only has access to tactile feedback from the sensors installed on both gripper fingers. For the real robot system, we use a 6-DoF ABB IRB 120 robot arm with a WSG-50 parallel jaw gripper. On each side of the gripper finger, we mount a GelSlim 3.0 tactile sensor that captures the tactile interaction between the fingers and the grasped object as a high-resolution tactile image. More details are in §B.5.

This task is more challenging than the stable grasp task (§4.2) because it not only needs to recognize the rotational pattern of the tactile field when the object contacts the front/back edges of the hole, but also needs to leverage the different magnitude relationship of two sensors' outputs to tell whether the object hits the left or the right hole edge (Fig. 5). It becomes even more challenging when the object hits the hole at four corners, because the robot must recognize nuances in the tactile pattern to decide the insertion pose adjustment. Therefore, this task requires a high-quality simulated tactile force field in order to transfer the learned policy to the real system successfully.

Policy Learning via RL We train the control policies with PPO [37] for three types of misalignments. *Rotation*: The object is initialized at the hole's center and has random rotation misalignment around the vertical axis. The action of the policy is the angle adjustment in the next insertion attempt. *Translation*: The object has randomly initialized translation misalignment to the hole and no rotation misalignment. The action space in this case is two dimensional for the translational correction on the x-y plane. *Rotation & Translation*: The object has both rotation and translation initial misalignments. The action space of the policy is three dimensional.



(a) contact at the left edge (b) contact at the right edge (c) contact at the back edge (d) contact at the front edge

Figure 5: Comparison of the normalized tactile flow maps. The flow maps in the top blue boxes are from simulation (with noise added), while the flow maps in the bottom red boxes are produced from the real GelSlim sensor. In each box, the two flow maps (left and right) are for the two tactile pads on the two gripper fingers.

During training, we convert the simulated tactile force field into our normalized tactile flow map representation (§3.4), and treat the resulting tactile flow map as a 13×10 flow "image" with 4 channels (2 sensors and 2 shear components of tactile forces). We model the policy by a convolutional RNN to leverage more information from previous attempts. For better sim-to-real performance, we also apply the domain randomization technique [39] on contact parameters, tactile sensor parameters, grasp forces, grasp height and tactile readings, to increase the robustness of the learned policies. More details of policy learning are provided in §B.5.

Experiment Results We first qualitatively compare the normalized tactile flow maps generated by simulation and by real GelSlim sensors. We plot the normalized tactile flow maps at four representative contact configurations (*i.e.,* object contacts at different edges of the hole) [10] in Fig. 5, which shows that our simulation is able to produce highly realistic patterns in those contact configurations. We then deploy the policies learned in simulation on real hardware and quantitatively test its zero-shot performance by conducting 100 insertion experiments under different initial pose misalignments. As reported in Table 2, our zero-shot policy transfer achieves 100% success rates on *Rotation* and *Translation* tasks. We also calculate the average number of pose corrections for successful experiments. The average number of pose corrections is 1.53 for the *Rotation* task and 2.33 for the *Translation* task, which means that the policy is able to successfully infer the pose misalignment after just one or two failed attempts in most experiments. Given that the policies are purely trained with simulated tactile data, the high success rates indicate that our simulation is able to produce normalized tactile flow maps with highly realistic tactile patterns and magnitudes to help the gripper to infer the exact adjustment. For challenging *Rotation & Translation* task, our zero-shot transferred policy also achieves 83% success rate and 4.81 pose corrections on average. For comparison, Dong et al. [10] achieve 89.6% success rate and 5.42 times pose adjustments for the cuboid object, with a policy trained directly on the real hardware from a pre-trained policy and with a carefully designed task curriculum. On the other hand, our policy is trained from scratch only in simulation without observing any real-world data.

TASK	SUCCESS	ATTEMPTS
R	100%	1.53
T	100%	2.33
R&T	83%	4.81

Table 2: Zero-shot sim-to-real performance of the tactile RL insertion policies.

5 Limitations and Future Work

We presented an efficient differentiable simulator that can handle dense tactile force fields with both normal and shear components. When the tactile pad is *very* soft (*e.g.,* TacTip), its dynamics cannot be well approximated by our penalty-based approach. An interesting direction to explore is how to efficiently simulate such soft tactile sensors. We demonstrated with the box pushing task (§4.3) the potential advantage of differentiable simulators. However, how to effectively leverage analytical gradients for more complex tactile-based tasks is still an open question and it may require more advanced policy learning algorithms [40]. Furthermore, in the sim-to-real experiment (§4.5), the zero-shot success rate of *Rotation & Translation* is not perfect. This is probably due to some intricacies of the real hardware that are difficult to model in our simulation. We believe that further fine-tuning the learned policies with a few shots on the real hardware will likely lead to improved performance.

Acknowledgments

We thank the anonymous reviewers for their helpful comments in revising the paper. We also thank Toyota Research Institute (TRI), ONR MURI (N00014-22-1-2740), and the National Science Foundation (CAREER-1846368) for providing funds to support this work.

References

[1] E. Donlon, S. Dong, M. Liu, J. Li, E. Adelson, and A. Rodriguez. Gelslim: A high-resolution, compact, robust, and calibrated tactile-sensing finger. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1927–1934. IEEE, 2018.

[2] S. Sundaram, P. Kellnhofer, Y. Li, J.-Y. Zhu, A. Torralba, and W. Matusik. Learning the signatures of the human grasp using a scalable tactile glove. *Nature*, 569(7758):698–702, 2019.

[3] M. Lambeta, P.-W. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, et al. Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters*, 5(3):3838–3845, 2020.

[4] I. Taylor, S. Dong, and A. Rodriguez. Gelslim3. 0: High-resolution measurement of shape, force and slip in a compact tactile-sensing finger. *arXiv preprint arXiv:2103.12269*, 2021.

[5] A. Church, J. Lloyd, R. Hadsell, and N. F. Lepora. Optical tactile sim-to-real policy transfer via real-to-sim tactile image translation. *arXiv preprint arXiv:2106.08796*, 2021.

[6] M. Bauza, A. Bronars, and A. Rodriguez. Tac2pose: Tactile object pose estimation from the first touch. *arXiv preprint arXiv:2204.11701*, 2022.

[7] E. Smith, R. Calandra, A. Romero, G. Gkioxari, D. Meger, J. Malik, and M. Drozdzal. 3d shape reconstruction from vision and touch. *Advances in Neural Information Processing Systems*, 33: 14193–14206, 2020.

[8] S. Suresh, Z. Si, J. G. Mangelson, W. Yuan, and M. Kaess. Shapemap 3-d: Efficient shape mapping through dense touch and vision.

[9] R. Kolamuri, Z. Si, Y. Zhang, A. Agarwal, and W. Yuan. Improving grasp stability with rotation measurement from tactile sensing. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6809–6816. IEEE, 2021.

[10] S. Dong, D. K. Jha, D. Romeres, S. Kim, D. Nikovski, and A. Rodriguez. Tactile-rl for insertion: Generalization to objects of unknown geometry. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6437–6443. IEEE, 2021.

[11] S. Kim and A. Rodriguez. Active extrinsic contact sensing: Application to general peg-in-hole insertion. *arXiv preprint arXiv:2110.03555*, 2021.

[12] W. Yuan, R. Li, M. A. Srinivasan, and E. H. Adelson. Measurement of shear and slip with a gelsight tactile sensor. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 304–311. IEEE, 2015.

[13] S. Dong, W. Yuan, and E. H. Adelson. Improved gelsight tactile sensor for measuring geometry and slip. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 137–144. IEEE, 2017.

[14] F. Veiga, H. Van Hoof, J. Peters, and T. Hermans. Stabilizing novel objects by learning to predict tactile slip. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5065–5072. IEEE, 2015.

[15] M. Li, Y. Bekiroglu, D. Kragic, and A. Billard. Learning of grasp adaptation through experience and tactile sensing. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3339–3346. Ieee, 2014.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[17] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[18] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[19] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[20] O. . M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[21] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[22] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018.

[23] J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, and W. Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[24] Z. Ding, Y.-Y. Tsai, W. W. Lee, and B. Huang. Sim-to-real transfer for robotic manipulation with tactile sensory. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6778–6785. IEEE, 2021.

[25] A. Melnik, L. Lach, M. Plappert, T. Korthals, R. Haschke, and H. Ritter. Using tactile sensing to improve the sample efficiency and performance of deep deterministic policy gradients for simulated in-hand manipulation tasks. *Frontiers in Robotics and AI*, page 57, 2021.

[26] S. Wang, M. Lambeta, P.-W. Chou, and R. Calandra. Tacto: A fast, flexible, and open-source simulator for high-resolution vision-based tactile sensors. *IEEE Robotics and Automation Letters*, 7(2):3930–3937, 2022.

[27] Y. S. Narang, K. Van Wyk, A. Mousavian, and D. Fox. Interpreting and predicting tactile signals via a physics-based and data-driven framework. *arXiv preprint arXiv:2006.03777*, 2020.

[28] Y. Narang, B. Sundaralingam, M. Macklin, A. Mousavian, and D. Fox. Sim-to-real for robotic tactile sensing via physics-based simulation and learned latent projections. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6444–6451. IEEE, 2021.

[29] Z. Ding, N. F. Lepora, and E. Johns. Sim-to-real transfer for optical tactile sensing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1639–1645. IEEE, 2020.

[30] Z. Si and W. Yuan. Taxim: An example-based simulation model for gelsight tactile sensors. *IEEE Robotics and Automation Letters*, 2022.

[31] D. Ma, E. Donlon, S. Dong, and A. Rodriguez. Dense tactile force estimation using gelslim and inverse fem. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5418–5424. IEEE, 2019.

[32] T. Bi, C. Sferrazza, and R. D'Andrea. Zero-shot sim-to-real transfer of tactile control policies for aggressive swing-up manipulation. *IEEE Robotics and Automation Letters*, 6(3):5761–5768, 2021.

[33] A. Habib, I. Ranatunga, K. Shook, and D. O. Popa. Skinsim: A simulation environment for multimodal robot skin. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1226–1231. IEEE, 2014.

[34] S. Moisio, B. León, P. Korkealaakso, and A. Morales. Model of tactile sensors using soft contacts and its application in robot grasping simulation. *Robotics and Autonomous Systems*, 61(1):1–12, 2013.

[35] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi:10.15607/RSS.2021.XVII.008.

[36] Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda. REDMAX: Efficient & flexible approach for articulated dynamics. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi:10.1145/3306346.3322952. URL https://doi.org/10.1145/3306346.3322952.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[38] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar. Robel: Robotics benchmarks for learning with low-cost robots. arxiv e-prints, page. *arXiv preprint arXiv:1909.11639*, 2019.

[39] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[40] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.

[41] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

A Penalty-based Tactile Simulation and Derivatives

In this section, we give the details of the forward dynamics and backward gradient derivation of our differentiable penalty-based tactile simulation. In §A.1, we give the formulation of the equations of motion for forward dynamics with the BDF1 time stepping scheme. In §A.2, we introduce our penalty-based tactile model and derive its analytical derivatives, which are necessary for our implicit forward time integration and backward gradients computation. In §A.3, we show in detail how we compute the analytical gradients of the whole simulation through reverse-mode backward differentiation.

A.1 Equations of Motion

We give the formulation of equations of motion $g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t)$ here. We follow the reduced-coordinate rigid-body dynamics formulation of DiffRedMax [35] and use the BDF1 implicit time integration scheme [41] with step size h to step forward the simulation. Mathematically, at each time step t, we take a state in reduced coordinate representation $(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1})$ and the joint-space action $\boldsymbol{u}_t$, and get the state $(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$ by solving the following equation with Newton's Method:

$$\left.\begin{aligned} \boldsymbol{q}_t &= \boldsymbol{q}_{t-1} + h\dot{\boldsymbol{q}}_t \\ \dot{\boldsymbol{q}}_t &= \dot{\boldsymbol{q}}_{t-1} + h\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_t) \end{aligned}\right\} \Rightarrow \underbrace{\boldsymbol{q}_t - \boldsymbol{q}_{t-1} - h\dot{\boldsymbol{q}}_{t-1} - h^2\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_t)}_{g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t)} = 0 \tag{6}$$

with

$$\ddot{\boldsymbol{q}}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u}_t) = \mathsf{M}_r^{-1}(\boldsymbol{q}_t)\Big[\mathsf{f}_r(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) + \mathsf{J}^\top(\boldsymbol{q}_t)\mathsf{f}_m(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) + \mathsf{f}_{QVV}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t) + \boldsymbol{u}_t\Big], \tag{7}$$

where M_r is the generalized mass matrix in reduced coordinates, J is the Jacobian, f_r is the generalized force vector generated by joint-space effects (*e.g.,* joint damping), f_m is the maximal wrench (*e.g.,* gravity, Coriolis forces, contact forces, external forces, etc.), f_{QV} is the quadratic velocity vector, and $\boldsymbol{u}_t$ is the joint-space action. Whenever we need the velocity, we compute it from the positions: $\dot{\boldsymbol{q}}_t = (\boldsymbol{q}_t - \boldsymbol{q}_{t-1})/h$. We will not go into details of Eq. 7 since it can be found in many rigid body dynamics tutorials.

A.2 Penalty-based Tactile Model and the Derivatives

As introduced in §3.2, our penalty-based tactile model contains two parts: first, we compute the contact forces (*i.e.,* normal force and friction force) at the tactile point's location with a penalty-based approach; then, we project the contact force into the local coordinate frame of the tactile point to acquire the desired *shear* and *normal* tactile force magnitudes:

$$\boldsymbol{f}_n = (-k_n + k_d\dot{d})d\boldsymbol{n}, \quad \boldsymbol{f}_t = -\frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|}\min(k_t\|\boldsymbol{v}_t\|, \mu\|\boldsymbol{f}_n\|) \tag{8a}$$

$$\boldsymbol{f} = \boldsymbol{f}_n + \boldsymbol{f}_t \tag{8b}$$

$$T_{sx} = \boldsymbol{f}^\top \boldsymbol{x}, \qquad T_{sy} = \boldsymbol{f}^\top \boldsymbol{y}, \qquad T_n = \boldsymbol{f}^\top \boldsymbol{z}. \tag{8c}$$

In order to complete the backward gradient computation, we need to compute the derivatives of the tactile forces with respect to the state of the simulation (*i.e.,* $\frac{\partial T_{\{sx,sy,n\}}}{\partial \mathbf{q}_t}$ and $\frac{\partial T_{\{sx,sy,n\}}}{\partial \dot{\mathbf{q}}_t}$). We drop the subscript t for brevity. Let $\mathbf{q}_m$ be the state of the simulation in maximal coordinates (which can be converted to reduced coordinates through the Jacobian [36]). We have

$$\frac{\partial T_{\{sx,sy,n\}}}{\partial \boldsymbol{q}} = \{\boldsymbol{x}^\top, \boldsymbol{y}^\top, \boldsymbol{z}^\top\}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} \tag{9a}$$

$$= \{\boldsymbol{x}^\top, \boldsymbol{y}^\top, \boldsymbol{z}^\top\}\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}_m}\mathsf{J} + \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{q}}_m}\frac{\partial \dot{\boldsymbol{q}}_m}{\partial \boldsymbol{q}}\right) \qquad \left(\frac{\partial \boldsymbol{q}_m}{\partial \boldsymbol{q}} = \mathsf{J}\right) \tag{9b}$$

$$\left(\frac{\partial \dot{\boldsymbol{q}}_m}{\partial \boldsymbol{q}} = \frac{\partial \mathsf{J}}{\partial \boldsymbol{q}} \otimes \dot{\boldsymbol{q}}, \text{ which is provided by DiffRedMax.}\right)$$

and

$$\frac{\partial T_{\{sx,sy,n\}}}{\partial \dot{\boldsymbol{q}}} = \{\boldsymbol{x}^\top, \boldsymbol{y}^\top, \boldsymbol{z}^\top\}\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{q}}} \tag{10a}$$

$$= \{\boldsymbol{x}^\top, \boldsymbol{y}^\top, \boldsymbol{z}^\top\}\left(\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{q}}_m}\mathsf{J}\right). \qquad \left(\frac{\partial \dot{\boldsymbol{q}}_m}{\partial \dot{\boldsymbol{q}}} = \mathsf{J}\right) \tag{10b}$$

Let $\mathbf{B}_1$ denote the body the tactile point attaches to, and $\mathbf{B}_2$ be the body the tactile point has contact with. We first derive the gradients of the contact normal force $\boldsymbol{f}_n$.

$$\frac{\partial \boldsymbol{f}_n}{\partial \boldsymbol{q}_m} = d\mathbf{n}\frac{\partial\big(-k_n + k_d\dot{d}\big)}{\partial \boldsymbol{q}_m} + (-k_n + k_d\dot{d})\boldsymbol{n}\frac{\partial d}{\partial \boldsymbol{q}_m} + (-k_n + k_d\dot{d})d\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{q}_m} \tag{11a}$$

$$= k_d d\mathbf{n}\frac{\partial \dot{d}}{\partial \boldsymbol{q}_m} + (-k_n + k_d\dot{d})\boldsymbol{n}\frac{\partial d}{\partial \boldsymbol{q}_m} + (-k_n + k_d\dot{d})d\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{q}_m} \tag{11b}$$

and

$$\frac{\partial \boldsymbol{f}_n}{\partial \dot{\boldsymbol{q}}_m} = k_d d\mathbf{n}\frac{\partial \dot{d}}{\partial \boldsymbol{q}_m}. \tag{12}$$

The values d, $\dot{d}$, $\boldsymbol{n}$ are only related to the bodies $\mathbf{B}_1, \mathbf{B}_2$. We assume that there is a distance function of body $\mathbf{B}_2$ (can be either an analytical function if $\mathbf{B}_2$ is a primitive shape or a signed distance field if $\mathbf{B}_2$ is an arbitrary shape), so their corresponding derivatives can be computed easily from the distance function.

Next, we derive the gradients for the contact friction force $\boldsymbol{f}_t$. Since the contact friction force is composed of the static friction force ($\boldsymbol{f}_s = -k_t\boldsymbol{v}_t$) and dynamic friction force ($\boldsymbol{f}_d = -\frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|}\mu\|\boldsymbol{f}_n\|$), we derive the derivatives for each of them separately.

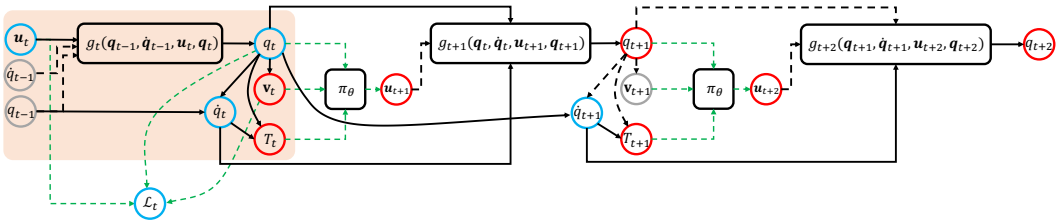


Figure 6: Computation graph of the simulation with BDF1 time stepping around time step t. We illustrate the computation graph for gradient derivations of $\partial\mathcal{L}/\partial\boldsymbol{q}_t$ and $\partial\mathcal{L}/\partial\boldsymbol{u}_t$. The boxes (*e.g.,* g, π_θ) represent functions, and the circles represent data/values. The grey circles are the data unrelated to the gradient derivation at step t. The red circles are the data $(\cdot)$ that we already have the gradient $\partial\mathcal{L}/\partial(\cdot)$ for when we arrive at step t during backward propagation. The blue circles are the data related to the gradient computation at step t. The green arrows are the data flows computed by PyTorch, and the black arrows are the data flows computed by our simulator. The dashed arrows are the data flows whose gradients computations are not handled by the simulator or are not related to the derivation at the current step. The orange-shaded part is our simulation layer for step t in the PyTorch computation graph.

For the static friction force, the derivatives are:

$$\frac{\partial \boldsymbol{f}_s}{\partial \boldsymbol{q}_m} = -k_t \frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{q}_m} \tag{13}$$

and

$$\frac{\partial \boldsymbol{f}_s}{\partial \dot{\boldsymbol{q}}_m} = -k_t \frac{\partial \boldsymbol{v}_t}{\partial \dot{\boldsymbol{q}}_m}, \tag{14}$$

where the derivatives of $\boldsymbol{v}_t$ can also be acquired from the distance function.

For the dynamic friction force, the derivatives are:

$$\frac{\partial \boldsymbol{f}_d}{\partial \boldsymbol{q}_m} = -\mu \frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|} \frac{\partial \|\boldsymbol{f}_n\|}{\partial \boldsymbol{q}_m} - \mu \|\boldsymbol{f}_n\| \frac{\partial \left(\boldsymbol{v}_t / \|\boldsymbol{v}_t\|\right)}{\partial \boldsymbol{q}_m} \tag{15a}$$

$$= -\mu \frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|} \frac{\boldsymbol{f}_n^\top}{\|\boldsymbol{f}_n\|} \frac{\partial \boldsymbol{f}_n}{\partial \boldsymbol{q}_m} - \mu \|\boldsymbol{f}_n\| \left(\frac{1}{\|\boldsymbol{v}_t\|} \frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{q}_m} - \frac{\boldsymbol{v}_t \boldsymbol{v}_t^\top}{\|\boldsymbol{v}_t\|^3} \frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{q}_m} \right) \tag{15b}$$

and

$$\frac{\partial \boldsymbol{f}_d}{\partial \dot{\boldsymbol{q}}_m} = -\mu \frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|} \frac{\boldsymbol{f}_n^\top}{\|\boldsymbol{f}_n\|} \frac{\partial \boldsymbol{f}_n}{\partial \dot{\boldsymbol{q}}_m} - \mu \|\boldsymbol{f}_n\| \left(\frac{1}{\|\boldsymbol{v}_t\|} \frac{\partial \boldsymbol{v}_t}{\partial \dot{\boldsymbol{q}}_m} - \frac{\boldsymbol{v}_t \boldsymbol{v}_t^\top}{\|\boldsymbol{v}_t\|^3} \frac{\partial \boldsymbol{v}_t}{\partial \dot{\boldsymbol{q}}_m} \right). \tag{16}$$

A.3 Backward Gradients Computation through Adjoint Method

Since we use an implicit time integration scheme for forward dynamics, the core step of gradient computation is to differentiate through the nonlinear equations of motion. We re-write the finite-horizon tactile-based policy optimization problem here for convenience.

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \;\; \mathcal{L} = \sum_{t=1}^{H} \mathcal{L}_t\big(\boldsymbol{u}_t, \boldsymbol{q}_t, \mathbf{v}_t(\boldsymbol{q}_t)\big) \tag{17a}$$

$$\text{s.t.} \;\; g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0 \qquad \text{(Equations of Motion)} \tag{17b}$$

$$\boldsymbol{u}_t = \pi_\theta\big(\tilde{\boldsymbol{q}}_{t-1}, \tilde{\mathbf{v}}_{t-1}(\boldsymbol{q}_{t-1}), T_{t-1}(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1})\big). \qquad \text{(Policy Execution)} \tag{17c}$$

Here, H is the task horizon, $\mathcal{L}_t$ is a step-wise task-dependent reward function, $\boldsymbol{u}$ is the action (*e.g.,* joint torques), $\boldsymbol{q}$ is the simulation state (*i.e.,* joint angles), and $\mathbf{v}$ is the derived auxiliary simulation variables (*e.g.,* fingertip positions) which themselves are a function of $\boldsymbol{q}$. Eq. 17b describes the nonlinear equations of motion (§A.1). Eq. 17c represents the inference of the control policy π_θ to obtain the desired action given the partial observation of the simulation state $\tilde{\boldsymbol{q}}$, partial observation of the simulation computed variables $\tilde{\mathbf{v}}$, and the tactile force values T from Eq. 8.

We embed our simulator as a differentiable layer into the PyTorch computation graph and use reverse mode differentiation to backward differentiate through dynamics time integration. To illustrate the gradient derivation, we draw the computation graph in Fig. 6. The computation steps such as

loss/reward computation and policy inference (*i.e.,* green arrows in the figure) are computed by PyTorch, and the dynamics-related computation (*i.e.,* black arrows) are processed by our simulator in C++. Each step of our simulation can be regarded as a function (shown in the orange shaded box in Fig. 6) in the computation graph:

$$(\boldsymbol{q}_t, \mathbf{v}_t, T_t) = \text{Sim}(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t). \tag{18}$$

We compute the gradients $d\mathcal{L}/d\theta = \sum_t (\partial\mathcal{L}/\partial\boldsymbol{u}_t)(\partial\boldsymbol{u}_t/\partial\theta)$ for policy optimization. The first gradient, $\partial\mathcal{L}/\partial\boldsymbol{u}_t$, which includes the simulation dynamics and tactile derivatives, is derived analytically. The second gradient, $\partial\boldsymbol{u}_t/\partial\theta$, is computed by PyTorch's auto-differentiation.

Now we show how to compute $\partial\mathcal{L}/\partial\boldsymbol{u}_t$. We compute $\partial\mathcal{L}/\partial\boldsymbol{u}_t$ in reverse order, starting from the last time step. At time step t, we assume that we have the following gradients computed: $\partial\mathcal{L}/\partial\mathbf{v}_{t,t+1,\dots}$, $\partial\mathcal{L}/\partial T_{t,t+1,\dots}$, and $\partial\mathcal{L}/\partial\boldsymbol{q}_{t+1,t+2,\dots}$ (red circles in Fig. 6). To compute the gradient backpropagation at step t, we need to compute $\partial\mathcal{L}/\partial\boldsymbol{q}_t$ and $\partial\mathcal{L}/\partial\boldsymbol{u}_t$:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_{t+1}}\left(\frac{\partial\boldsymbol{q}_{t+1}}{\partial\boldsymbol{q}_t} + \frac{\partial\boldsymbol{q}_{t+1}}{\partial\dot{\boldsymbol{q}}_t}\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t}\right) + \left(\frac{\partial L}{\partial T_{t+1}}\frac{\partial T_{t+1}}{\partial\dot{\boldsymbol{q}}_{t+1}} + \frac{\partial L}{\partial\boldsymbol{q}_{t+2}}\frac{\partial\boldsymbol{q}_{t+2}}{\partial\dot{\boldsymbol{q}}_{t+1}}\right)\frac{\partial\dot{\boldsymbol{q}}_{t+1}}{\partial\boldsymbol{q}_t} \\ + \frac{\partial\mathcal{L}}{\partial\mathbf{v}_t}\frac{\partial\mathbf{v}_t}{\partial\boldsymbol{q}_t} + \frac{\partial\mathcal{L}}{\partial T_t}\left(\frac{\partial T_t}{\partial\boldsymbol{q}_t} + \frac{\partial T_t}{\partial\dot{\boldsymbol{q}}_t}\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t}\right),\end{aligned} \tag{19a}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{u}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{u}_t} + \frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_t}\frac{\partial\boldsymbol{q}_t}{\partial\boldsymbol{u}_t}. \tag{19b}$$

The derivatives $\partial\mathbf{v}_t/\partial\boldsymbol{q}_t$ can be computed from the functions $\mathbf{v}(\boldsymbol{q})$ easily. The derivatives $\partial T_t/\partial\boldsymbol{q}_t$, $\partial T_t/\partial\dot{\boldsymbol{q}}_t$, and $\partial T_{t+1}/\partial\dot{\boldsymbol{q}}_{t+1}$ have been shown in §A.2. The derivatives of $\partial\dot{\boldsymbol{q}}_t/\partial\boldsymbol{q}_t$ and $\partial\dot{\boldsymbol{q}}_{t+1}/\partial\boldsymbol{q}_t$ can be computed from the BDF1 equations (Eq. 6).

$$\frac{\partial\dot{\boldsymbol{q}}_t}{\partial\boldsymbol{q}_t} = \frac{1}{h}\mathbf{I} \tag{20a}$$

$$\frac{\partial\dot{\boldsymbol{q}}_{t+1}}{\partial\boldsymbol{q}_t} = -\frac{1}{h}\mathbf{I}. \tag{20b}$$

To computing the remaining derivatives $\partial\boldsymbol{q}_{t+1}/\partial\boldsymbol{q}_t$, $\partial\boldsymbol{q}_{t+1}/\partial\dot{\boldsymbol{q}}_t$, $\partial\boldsymbol{q}_{t+2}/\partial\dot{\boldsymbol{q}}_{t+1}$, and $\partial\boldsymbol{q}_t/\partial\boldsymbol{u}_t$ we must differentiate through the implicit function $g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0$. We show the derivation for $\partial\boldsymbol{q}_t/\partial\boldsymbol{u}_t$ and how to compute Eq. 19b efficiently through the adjoint method; the same approach can be used for computing others and Eq. 19a.

We apply the implicit function theorem on $g(\boldsymbol{q}_{t-1}, \dot{\boldsymbol{q}}_{t-1}, \boldsymbol{u}_t, \boldsymbol{q}_t) = 0$:

$$\frac{dg}{d\boldsymbol{u}_t} \equiv 0 \Rightarrow \frac{\partial g}{\partial\boldsymbol{q}_t}\frac{\partial\boldsymbol{q}_t}{\partial\boldsymbol{u}_t} + \frac{\partial g}{\partial\boldsymbol{u}_t} \equiv 0 \Rightarrow \frac{\partial\boldsymbol{q}_t}{\partial\boldsymbol{u}_t} = -\left(\frac{\partial g}{\partial\boldsymbol{q}_t}\right)^{-1}\frac{\partial g}{\partial\boldsymbol{u}_t}. \tag{21}$$

Plugging this into Eq. (19b):

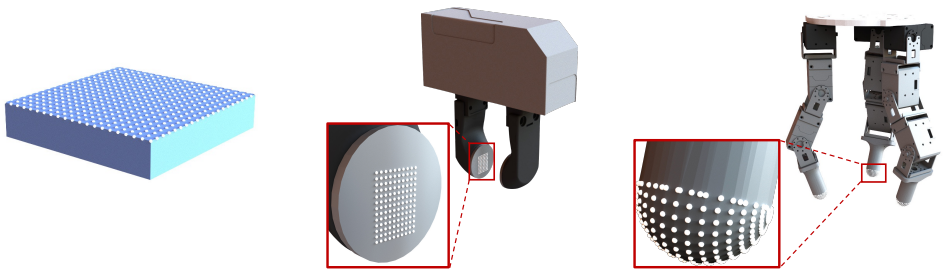
$$\frac{d\mathcal{L}}{d\boldsymbol{u}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{u}_t} - \underbrace{\frac{\partial\mathcal{L}}{\partial\boldsymbol{q}_t}}_{\boldsymbol{b}}\left(\underbrace{\frac{\partial g}{\partial\boldsymbol{q}_t}}_{A}\right)^{-1}\frac{\partial g}{\partial\boldsymbol{u}_t}. \tag{22}$$

To efficiently calculate Eq. (22), we apply the adjoint method to first solve $\boldsymbol{c}$ from the linear equation $A^\top\boldsymbol{c} = \boldsymbol{b}^\top$ and then compute Eq. (19b) as

$$\frac{d\mathcal{L}}{d\boldsymbol{u}_t} = \frac{\partial\mathcal{L}_t}{\partial\boldsymbol{u}_t} - \boldsymbol{c}^\top\frac{\partial g}{\partial\boldsymbol{u}_t}. \tag{23}$$

B Detailed Experiment Setups and More Results

In this section, we give the details of the 5 experiments in §4 of the main text. The subsections correspond between this appendix section and the main experiment section, so that §4.1 ↔ §B.1, §4.2 ↔ §B.2, etc.



(a) Flat Tactile Pad (§4.1) (b) WSG-50 (§4.2, §4.3, §4.5) (c) D'Claw (§4.4)

Figure 7: Visualization of the tactile sensor layouts. We visualize the tactile sensor layouts of the three tactile manipulators we used in the experiments: (a) The flat tactile pad used in the ball rolling experiment (§4.1) (for better visualization, we only render tactile resolution of 20×20); (b) The WSG-50 gripper with GelSlim sensor used in the stable grasp task (§4.2), box pushing task (§4.3) and tactile RL insertion task (§4.5); (c) The D'Claw tri-finger hand used in the rotating cap task (§4.4).

B.1 High-Resolution Tactile Ball Rolling Experiment

In §4.1, we use a ball rolling experiment to show the efficacy of the tactile force field generated by our simulator and to test the simulation speed. The resolution of the tactile marker points is 200×200, and the simulation step size $h = 5$ ms. Here we visualize the lower surface of the pad in Fig. 7(a). The speed of the simulation varies with the resolution of the tactile marker points and the frequency of the tactile force field computation. We report in Table 3 the speeds of the simulation under different tactile marker resolutions and different frequencies of tactile force field acquisition. All the experiments run on a single core of an Intel Core i7-9700K CPU. The simulation speed can be further accelerated by simply parallelizing different environments across multiple CPU cores, and the speed of each individual simulation can be significantly accelerated by computing the tactile force at each tactile marker point in parallel through GPU programming since all the tactile marker points are independent of each other.

Frequency / Resolution	40 Hz (5 steps)	10 Hz (20 steps)
10×10	3477 FPS	3562 FPS
50×50	3167 FPS	3482 FPS
200×200	1050 FPS	2360 FPS

Table 3: Simulation speeds at different tactile points resolutions and the frequencies of the tactile force computation.

B.2 Tactile-Based Stable Grasp Task

In §4.2, we show the usage of shear force information for control and the effectiveness of our tactile simulator in a parallel-jaw bar-grasping task.

Task Specification The task requires a WSG-50 parallel-jaw gripper to stably grasp a bar with *unknown mass distribution* in fewer than 10 attempts. The gripper has two tactile sensors with a tactile marker resolution of 13×10 with 1.5 mm space between adjacent markers (shown in Fig. 7(b)). The bar is composed of 11 blocks. The total mass of the bar is in the range $[51, 120]$ g. Directly randomizing the density of each block results in the configuration where the center of mass of the bar is located near the geometric center in most cases. Therefore, to generate the bar with a uniform distribution of the center of mass, we randomize the center of mass location first, and then adjust the density of the blocks to meet the requirement of the center of mass location. We consider a grasp to be a failure if the bar is not lifted up or tilts more than 0.02 rad after the gripper grasps a bar.

The initial grasp location is the geometric center of the bar. The policy executes in an episodic process iterating between open-loop grasp attempts followed by grasp position adjustments. Specifically, the policy observes the tactile sensor readings at the frame the gripper lifts up the bar. Based

on this observation input only, the policy outputs a delta change in the grasping location. A scripted grasping controller will then be executed to grasp the bar in the predicted grasping location.

Reward Function The reward function is defined as:

$$\mathcal{R}_t = \begin{cases} 100 & |\alpha| \leq 0.2 \text{ rad and } h > 0.5 \text{ cm (success)} \\ -10|\alpha| & \text{otherwise,} \end{cases} \tag{24}$$

where α is the tilting angle of the bar and h is the lifted height of the bar.

Policy Learning We treat the tactile force field as a multi-channel "image" (resolution is 13×10 and each channel is for the force component in each axis). The policy is modeled as a shallow CNN (*Conv-ReLU-MaxPool-Conv-ReLU-FC-FC*) that takes as input the tactile sensor readings from two tactile pads. We train the policies with PPO [37] using 32 parallel environments with 20K environment steps in total. The PPO parameters are reported in Table 4.

Parameter names	**Value**
learning rate	$3e^{-4}$ (with *linear decay* schedule)
number of rollouts per iteration	32
entropy coefficient	0.01
value loss coefficient	0.5
batch size	256
discount factor γ	0.99
GAE λ	0.95
PPO clip range	0.2

Table 4: The hyperparameter setting of PPO on the tactile-based stable grasp task.

Training Results We train the policies with 3 different random seeds. The training curve is plotted in Fig. 8. We test the trained policies for 320 times, and the success rate is $98.5 \pm 1.8\%$ (the success rate is updated from the one reported in §4.2). The average number of attempts taken to stably grasp the bars is 2.1. We further test the learned policies on grasping the bar with a different number of blocks. While the policies are trained with 11 blocks, it achieves 93.3% success rate on 7-block bars and 99.8% on 13-block bars. The generalizability of the learned policies comes from the tactile-based observation, since the rotation pattern of the tactile force field remains consistent no matter how long the bar is.

B.3 Tactile-Based Box Pushing Task

In §4.3, we design a box pushing task similar to [5] to demonstrate how we can leverage the provided analytical gradients to help learn tactile-based control policies better and faster.

Task Specification The task is to use the same WSG-50 parallel jaw gripper as the one in the stable grasp task (with only one finger kept) to push the box to a randomly sampled goal location and orientation. The ranges of the goal location coordinates are $x_g \in [0.15 \text{ m}, 0.25 \text{ m}]$, $y_g \in [-0.2 \text{ m}, 0.2 \text{ m}]$. The range of the goal orientation is $\alpha_g \in [y_g\pi - \pi/16, y_g\pi + \pi/16]$. The initial position of the box is randomly disturbed ($[-0.02 \text{ m}, 0.02 \text{ m}]$ along the direction of the finger surface). A random external force f_{ext} (with $f^x_{\text{ext}}, f^y_{\text{ext}} \in [-1, 1]$ N) is applied continually on the box, which changes every 0.25 s. The control frequency is 40 Hz.

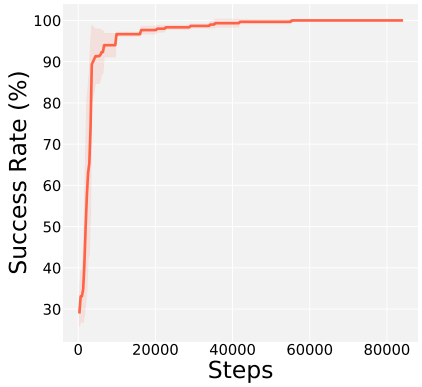


Figure 8: Training Curve of Tactile-based Stable Grasp Task. The curve is averaged from three random seeds. The shaded area is the standard deviation.

Reward Function The reward function is defined at each control step as

$$\mathcal{R}_t = \mathcal{R}_{\text{pos}} + \mathcal{R}_{\text{rot}} + \mathcal{R}_{\text{touch}} + \mathcal{R}_{\text{u}} \tag{25a}$$

$$\mathcal{R}_{\text{pos}} = -0.01\Big(\frac{\|p_{xy} - [x_g, y_g]\|}{\sigma_{\text{pos}}}\Big)^2, \quad \sigma_{\text{pos}} = 0.01 \text{ m} \tag{25b}$$

$$\mathcal{R}_{\text{rot}} = -0.1\Big(\frac{\alpha - \alpha_g}{\sigma_{\text{rot}}}\Big)^2, \quad \sigma_{\text{rot}} = \frac{\pi}{36} \text{ rad} \tag{25c}$$

$$\mathcal{R}_{\text{touch}} = -\Big(\frac{\|p_{\text{finger}} - p_{\text{box}}\|}{\sigma_{\text{touch}}}\Big)^2, \quad \sigma_{\text{touch}} = 0.02 \text{ m} \tag{25d}$$

$$\mathcal{R}_{\text{u}} = -0.1\|u\|^2, \tag{25e}$$

where p_{xy} is the position of the box in the x-y plane, α is the rotation angle of the box around the vertical axis (z axis), p_{finger} is the position of the center of the gripper finger, p_{box} is the position of the center of the box surface closest to the finger, and u is the policy action (normalized to $[-1, 1]$).

Policy Learning In this task, we explore another tactile observation representation. We flatten the whole tactile force field into a vector and model the policy as an MLP with 2 fully connected hidden layers of 64 units. For the *PPO* policies, we use the PPO parameters reported in Table 5. For the *GD* policies, we use Adam as our optimizer with $\beta_1 = 0.7, \beta_2 = 0.95$. The learning rate of Adam starts from 0.005 and follows a *linear decay* schedule over the episodes.

Parameter names	**Value**
learning rate	$3e^{-4}$ (with *linear decay* schedule)
number of rollouts per iteration	80
entropy coefficient	0
value loss coefficient	0.5
batch size	128
discount factor γ	0.99
GAE λ	0.95
PPO clip range	0.2

Table 5: The hyperparameter setting of PPO on the tactile-based box pushing task.

Experiment Results More visual results comparing different policies are provided in the supplemental video.

B.4 D'Claw Rotate Cap

In §4.4, we train a D'Claw tri-finger hand to open a cap on a bottle, to demonstrate that our method supports tactile sensors on curved surfaces. We put the tactile sensors on the three rounded fingertips in a hemisphere layout, and we use 302 evenly-spaced tactile markers. A close-up view of the tactile sensors is provided in Fig. 7(c).

Task Specification The task is to open a cap using the D'Claw hand. The position (randomized in a $0.04 \times 0.04\,\mathrm{m}^2$ region on the horizontal plane) and the radius (randomized in the range of $[0.2, 0.8]$ m) of the cap are unknown. There is also unknown damping (randomized in the range of $[0.01, 0.7]$) between the cap and the bottle. The task is considered a success if the cap is rotated by $\alpha_g = \pi/4$ rad. The only observation data that the policy gets are the angles of each joint, fingertip positions, and tactile sensor readings. This task is similar to how we open caps by just using proprioception sensory data and tactile feedback on the fingers without knowing the exact size and location of the cap. The control frequency is 40 Hz.

Reward Function The reward function is defined at each control step as

$$\mathcal{R}_t = \mathcal{R}_{\text{touch}} + \mathcal{R}_{\text{rot}} + \mathcal{R}_{\text{u}} + \mathcal{R}_z + \mathcal{R}_{\text{success}} \tag{26a}$$

$$\mathcal{R}_{\text{touch}} = -0.5 N_{\text{no touch}} \tag{26b}$$

$$\mathcal{R}_{\text{rot}} = -\min(\alpha - \alpha_g, 0)^2 \tag{26c}$$

$$\mathcal{R}_{\text{u}} = -0.005 \|u\|^2 \tag{26d}$$

$$\mathcal{R}_z = -50 \cdot \mathbb{1}_{p_{i,z} > c_z \text{ for } i \in \{1,2,3\}} \tag{26e}$$

$$\mathcal{R}_{\text{success}} = 50 \cdot \mathbb{1}_{\alpha > \alpha_g}, \tag{26f}$$

where $N_{\text{no touch}}$ is the number of fingers not touching the cap, α is the rotating angle of the cap, and u is the action of the policy, $p_{i,z}$ is the z coordinate of i^{th} fingertip position, and c_z is the z coordinate of the cap's top surface. An episode is terminated when the task is successfully completed, the number of steps exceeds the maximum episode length, or any of the fingertips is above the cap.

Policy Learning The policy takes the hand joint angles, fingertip positions, and the tactile sensor readings as input, and outputs the delta change on the joint angles. We use PPO to train the policy (a shallow CNN) using 32 parallel environments. The PPO parameters are shown in Table 6. To show that the tactile sensors are useful in this task, we also train a baseline policy (a simple MLP policy) where the policy only takes as input the joint angles and fingertip positions.

Parameter names	**Value**
learning rate	$3e^{-4}$ (with *linear decay* schedule)
number of rollouts per iteration	32
entropy coefficient	0.01
value loss coefficient	0.5
batch size	256
discount factor γ	0.99
GAE λ	0.95
PPO clip range	0.2

Table 6: The hyperparameter setting of PPO on the D'Claw rotating cap task.

Experiment Results We run each policy type three times with different random seeds, and plot the averaged training curves in Fig. 9. With tactile sensor readings, policies learn significantly faster and achieve an 87.3% success rate, while policies only achieve a 59.7% success rate when tactile sensor information is unavailable.

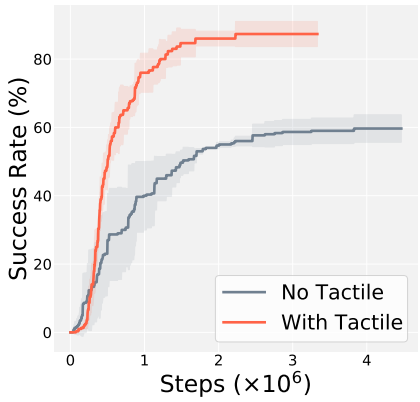


Figure 9: Training Curves of D'Claw Rotating Cap Task. The curves are averaged from three random seeds. The shaded area is the standard deviation.

B.5 Zero-Shot Sim-to-Real: Tactile RL Insertion Task

In §4.5, we conduct a sim-to-real experiment on the tactile-RL insertion task.

Task Specification In this task, a gripper (same as the one in the stable grasp task) is controlled to insert a cuboid object into a rectangle-shaped hole with a random initial pose misalignment (up to 6 mm for translation error and 10° for rotation error). The insertion process is modeled as an episodic policy that iterates between open-loop insertion attempts followed by insertion pose adjustments. The robot has up to 15 pose correction attempts, and the robot only has access to tactile feedback from the sensors installed on both gripper fingers.

Real Robot Setup We use a 6-DoF ABB IRB 120 robot arm with a WSG-50 parallel jaw gripper. On each side of the gripper finger, we mount the GelSlim 3.0 tactile sensors that capture the tactile interaction between the fingers and the grasped object as a high-resolution tactile image. The rectangle-shaped object and hole are 3-D printed. The clearance between the object and hole is 2.25 mm and the initial misalignment between them is randomly sampled within the maximum value of (6 mm, 6 mm, 10°), which is identical to the previous work [10]. We also vary the grasping force between 10$\sim$15 N and the grasping height between 42$\sim$57 mm. To extract the marker tracking information from the raw image tactile measurement, we use a marker detection algorithm from [4]. The speed of the gripper moving down during the insertion attempt is 0.5 mm/s and we capture tactile images every 80 ms.

Reward Function The reward is defined for each insertion attempt as follows:

$$\mathcal{R}_t = \mathcal{R}_{\text{pos}} + \mathcal{R}_{\text{rot}} \tag{27a}$$

$$\mathcal{R}_{\text{pos}} = 10^4 \times \|\text{error}_{\text{position}}\|^2 \tag{27b}$$

$$\mathcal{R}_{\text{rot}} = 20 \times \text{error}^2_{\text{rotation}}. \tag{27c}$$

Policy Learning We train the control policies with PPO [37] for three types of misalignments as mentioned in §4.5. During training, we convert the simulated tactile force field into our normalized tactile flow map representation, and treat the resulting tactile flow map as a 13×10 flow "image" with 4 channels (2 sensors and 2 shear components of tactile forces). For *Rotation* and *Translation* tasks, we sample five tactile frames during the insertion and stack them to obtain the corresponding normalized tactile flow maps to form the policy observation. For *Rotation & Translation* task, we only use the last frame during the insertion attempt as the observation since we observed that the policy tends to overfit to the simulation when more frames are provided. Specifically, when we input more frames of tactile fields, the policy tends to learn to leverage some simulation-only unnoticeable patterns of the tactile field to complete tasks in a tricky way; however those patterns do not sometimes exist in the real robot, and thus we input fewer tactile information to prevent such overfitting. For all three tasks, we model the policy by a convolutional RNN to leverage more information from

previous attempts. For better sim-to-real performance, we also apply the domain randomization technique to increase the robustness of the learned policies. Specifically, we randomly change various simulation parameters such as contact parameters, tactile sensor parameters, grasp forces, and grasp height within some ranges, and we also apply random noise to each value in the tactile force observation. The ranges of the parameter randomization are provided in Table 7.

	Parameter names	**Range**
Contact Parameters	k_n	$[2e^3, 1.4e^4]$
	k_t	$[20, 140]$
	μ	$[0.5, 2.5]$
Tactile Parameters	k_n	$[50, 450]$
	k_t	$[0.2, 2.3]$
	μ	$[0.5, 2.5]$
	k_d	$[0, 100]$
Other Parameters	grasp force	$[2.5, 16]$ N
	grasp height	$[-10, 5]$ mm
	tactile force noise	$[-1e^{-5}, 1e^{-5}]$

Table 7: The ranges of the domain randomization parameters.