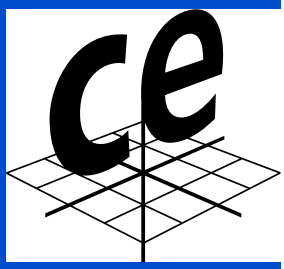
Simulation of GelSight Tactile Sensors for Reinforcement Learning

Simulation von taktilen GelSight Sensoren für Reinforcement Learning
Bachelor thesis in the field of study “Computational Engineering” by Duc Huy Nguyen
Date of submission: September 25, 2024

1. Review: Alap Kshirsagar, Ph.D.
2. Review: Tim Schneider, M.Sc.
3. Review: Guillaume Duret, M.Sc.
4. Review: Prof. Jan Peters, Ph.D.
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Duc Huy Nguyen, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Darmstadt, 25. September 2024

DH. Nguyen

Abstract

When we interact with our environment, our tactile sense plays a crucial role. With our sense of touch, we can infer object properties, such as shape, hardness, and surface texture. Among other things, we use it to grasp and lift objects reliably. Or to locate and identify objects in the dark. Similarly, robots can benefit from tactile sensing. One potential way to leverage tactile sensing in robotics is to use tactile sensors and Reinforcement Learning. Training Reinforcement Learning algorithms inside simulations is desirable and a common approach since training robots in the real world is time-consuming, costly, and potentially dangerous.

To simplify and further advance research on tactile sensing in robotics, we present a novel framework for simulating GelSight tactile sensors in a Reinforcement Learning setting. Compared to other tactile simulators, our simulation is built into a general-purpose robotics simulator. Additionally, our framework aims to be a uniform platform for tactile simulation. The framework is modular, extendable, and easy to use. One also has access to various powerful features, since the framework is built on NVIDIA's Isaac Sim and Isaac Lab.

The framework currently supports the simulation of the GelSight Mini sensor. For the physics simulation, we leverage the physics engine of Isaac Sim and our own GIPC simulation. For the sensor outputs, we use simulation approaches based on Taxim and FOTS. We demonstrate features of our framework and the behavior of our simulations with multiple examples. Moreover, our framework includes three Reinforcement Learning environments to leverage the tactile sensor simulation directly.

Code and documentation will be released as an open-source project.

Zusammenfassung

Wenn wir mit unserer Umwelt interagieren, spielt unser Tastsinn eine entscheidende Rolle. Mit unserem Tastsinn können wir Form, Härte und Oberflächenstruktur von Objekten bestimmen. Wir nutzen ihn unter anderem, um Gegenstände zuverlässig greifen und heben zu können. Oder um Objekte im Dunkeln zu finden und zu identifizieren. In ähnlicher Weise können auch Roboter von einem Tastsinn profitieren. Ein möglicher Weg, Tastsinn in der Robotik zu nutzen, ist die Verwendung von taktilen Sensoren und Reinforcement Learning. Es ist wünschenswert Reinforcement Learning Algorithmen in simulierten Umgebungen zu trainieren. Gründe dafür sind, dass das Training von Robotern in der Realität zeitaufwändig, kostspielig und potenziell gefährlich ist.

Um die Forschung auf dem Gebiet der taktilen Wahrnehmung in der Robotik zu vereinfachen und voranzutreiben, präsentieren wir deshalb ein neues Framework für die Simulation von taktilen GelSight Sensoren für Reinforcement Learning. Im Vergleich zu anderen existierenden taktilen Simulatoren ist unsere Simulation in einen Allzweck-Robotiksimulator integriert. Zusätzlich zielt unser Framework darauf ab, eine einheitliche Plattform für taktile Simulationen zu sein. Das Framework ist modular, erweiterbar und einfach zu benutzen. Außerdem hat man Zugang zu verschiedenen leistungsstarken Funktionen, da das Framework auf NVIDIAs Isaac Sim und Isaac Lab aufbaut.

Das Framework unterstützt derzeit die Simulation von GelSight Mini-Sensoren. Für die Physiksimulation nutzen wir sowohl die Physik-Engine von Isaac Sim als auch unsere eigene GIPC-Simulation. Um die Sensordaten zu simulieren, verwenden wir Ansätze die auf Taxim und FOTS basieren. Wir demonstrieren Eigenschaften und Verhalten unserer Simulationen anhand mehrerer Beispiele. Darüber hinaus verfügt unser Framework über drei Reinforcement Learning Umgebungen. Dies ermöglicht die direkte Nutzung unserer Simulation für taktilen Sensoren.

Der Code und die Dokumentation werden als Open-Source-Projekt veröffentlicht.

Acknowledgments

Grammarly was used for finding and fixing grammar issues, as well as to improve the overall quality of the writing.

I want to thank Alap Kshirsagar, Boris Belousov, Guillaume Duret and Tim Schneider for their continuous support and advice during the thesis. I additionally want to thank Fatma Miray Ural and Ngoc Anh Pham for helping me improve the writing.

And special thanks to Julia Bonk for inspiration, motivation and agitation.

Contents

Figures and Tables

List of Figures

List of Tables

1 Introduction

When we interact with our environment, our tactile sense plays a crucial role. Via our sense of touch, we can infer object properties, such as shape, hardness, and texture. Among other things, we use it to recognize objects, to locate objects, or to pick up objects reliably. Similarly, robots can benefit from tactile sensing. In settings where one cannot rely purely on vision, for example when occlusions appear, or in settings with bad illumination, having a sense of touch can be useful, or might even be crucial for the robot. Additionally, tactile sensing provides the robot with additional information that vision alone cannot provide. A tactile sensor can give a robot precise information about local contact geometry, object stiffness, and surface texture [1]. This can, for example, be used for slip detection [2], grasping of soft objects [3] or hardness estimation [4]. In general, using tactile sensors can aid in challenging manipulation tasks [5], but it can also be used for to improve locomotion [6]. Another use case lies in Human-Robot interaction [7].

One way to leverage tactile sensing in robotics is by using tactile sensors and Reinforcement Learning (RL). Reinforcement Learning has proven to be a promising method for solving challenging and complex tasks in various fields [8, 9], including robotics [10, 11]. RL algorithms rely on interactions with an environment. During Training, they collect a massive amount of data by trial and error. Since training robots in the real world is time-consuming, costly, and also potentially dangerous, a common approach is therefore to train RL algorithms inside simulated environments. Simulations enable extremely fast data collection without the potential to break expensive equipment. However, not only do the simulators need to be fast, they also need to be accurate. Otherwise, the Sim2Real gap becomes too large, resulting in the simulated training not being transferable to a real robot.

Naturally, researchers worked on simulating tactile sensors, due to their importance for research on tactile sensing in robotics and RL. Multiple different kinds of tactile sensors exist. Each has different working principles and designs. For instance, there are resistive, piezoresistive, piezoelectric, capacitance, and optical sensors [12]. Commonly used by

researchers are vision-based tactile sensors, which use cameras to generate high-resolution tactile readings. In this work, we focus on the simulation tactile sensors from the GelSight type [13]. GelSight sensors are one of the representative types of vision-based tactile sensors [14].

Nowadays, multiple simulators for GelSight tactile sensors exist [15–19]. They not only differ regarding their approaches for simulating the sensor output, but also in the way they simulate the physics. Additionally, most simulators use different backends, different tactile sensors, and different robots. This makes it extremely difficult to compare existing simulators and their approaches. Another problem is that these simulators often do not use general robotics simulators, which limits the usability for further research.

To address these issues, we developed a modular and easily extendable tactile simulation framework. By making it modular and easily extensible, developing and testing new simulation methods becomes easier than ever before. One could work on a new tactile simulation and directly use it for RL, as well as compare it to other existing methods. The effort for this is reduced considerably. Our framework currently supports three different kinds of physics simulation and uses state-of-the-art approaches for simulating the vision-based tactile sensors. Each approach uses the same top-level API, which means the usage of the tactile simulation is independent of the employed simulation approach.

To improve the usability for research, we built our framework on top of NVIDIA's Isaac Sim [20] and Isaac Lab [21]. Isaac Sim is a general-purpose robotics simulator with various powerful features, such as photorealistic rendering, ROS support, and GPU-accelerated physics simulation. Furthermore, it supports a variety of robots and sensors out of the box. By leveraging Isaac Sim, we gain access to these powerful features and can use them for tactile sensing in robotics. Isaac Lab was initially developed based on the Orbit [22] framework. It is a modular and extensible RL framework built on top of Isaac Sim. It provides many functionalities, such as support for teleoperation devices to collect demonstrations and GPU-accelerated multi-environment training. Additionally, it supports various RL libraries like RSL RL [23] and Stable-Baselines3 [24]. Furthermore, it contains multiple RL environments for direct usage. Integrating our tactile simulation inside Isaac Lab enables easy usage for RL.

We specifically simulate the commercially available GelSight Mini sensor [25], but our tactile simulation can easily be extended to simulate other types of tactile sensors. We use the built-in physics engine of Isaac Sim for rigid and soft body simulation. Additionally, we implement a custom soft body simulation based on GIPC [26]. GIPC is a GPU-based variant of the Incremental Potential Contact method. It allows for guaranteed intersection and inversion free simulation of soft bodies. For simulating the sensor output, we use

approaches based on Taxim [15] and FOTS [27]. We demonstrate the capabilities of our framework with various examples. Furthermore, we integrate three RL environments, which leverage tactile sensing.

Currently, our framework features:

- modular and extensible tactile sensor simulation
- GIPC-based soft body simulation for RL
- two robot configurations equipped with GelSight Mini sensors
- multiple examples for testing physics and sensor simulation
- three RL environments for tactile sensing
- the features of Isaac Sim and Isaac Lab

We hope our framework simplifies and further pushes research on simulation and usage of tactile sensors for robotics.

The rest of the thesis structure is as follows. To begin, we outline in chapter 2 the working principles of real GelSight sensors and introduce our used terminology regarding the sensors. Chapter 3 summarizes related works and their approaches to simulating tactile sensors. In 4 we describe our framework in detail. First, we give a general overview of the framework. Subsequently, we explain how exactly we simulate the GelSight Mini sensor. Chapter 5 describes our examples and showcases for demonstrating the capabilities of our framework. Finally, chapter 6 concludes the thesis and presents ideas for future work.

1.1 Design Goals

The core goal of our framework is to simplify research on tactile simulation methods, as well as tactile sensing for robotics in general. Our framework should ideally be a uniform platform for tactile simulations. For this, we aim to design a tactile simulation framework that is modular and easily extendable.

Modular here means, that one can easily switch out components of the tactile simulation with other approaches/implementations. On the one hand, extendable refers to the types of GelSight sensors that are simulated. It should be straightforward to integrate other GelSight sensors into our framework. On the other hand, extendable refers to the

simulation approaches, i.e., physics, optical, and marker simulation. Integrating other simulation approaches and sensor models into our framework should be simple.

Another design goal of ours is the ease of usage. Using tactile sensors in our framework should be the same as using other sensors in Isaac Lab. This enables the simple usage of tactile simulation for RL. Additionally, the usage should be independent of the underlying simulation approach. Users should be able to use different approaches for tactile simulation on the same RL tasks with minimal effort. For this, we aim to provide a workflow indifferent to the simulation approaches. Another benefit is the easy adoption of new tactile simulators even if they are developed in other backends. The need to reimplement the RL environments in their respective backends is omitted, instead one can update the tactile simulation of our framework and directly use it for RL.

Lastly, we aim to leverage Isaac Sim and Isaac Lab as much as possible, to benefit from their various features.

2 Foundations

Understanding the working principles of a real GelSight sensor is important for understanding how such a sensor can be simulated and what the challenges are. In this chapter, we therefore describe basic working principles behind a GelSight Type tactile sensor. For a more in-depth explanation of GelSight sensors, we refer to [28] and [13].

Additionally, we introduce the terminologies we use in the context of GelSight sensors, such as *tactile RGB image* and *marker motion field*.

2.1 GelSight Working Principles

Figure 2.1: A GelSight Mini sensor and its gelpads. Inside the sensor case are LEDs and a camera. There are two types of gelpads for the GelSight Mini, with markers and without.

GelSight sensors come in different variations and designs [25, 29–31], but the core components and working principles are the same. A GelSight sensor has a soft deformable elastomer, the *gelpad*. Inside its sensor case are LEDs and a camera. The LEDs illuminate the interior of the sensor for the camera. The GelSight Mini sensor can be seen in Figure 2.1. The camera itself generates images of the gelpad surface, which is inside the sensor case. These images are the tactile images.

When an object is pressed into the gelpad, the gelpad deforms. We call the object, which is pressed into gelpad, the *indenter*. And we use the term *indentation depth* to describe how deep the indenter is pressed into the gelpad.

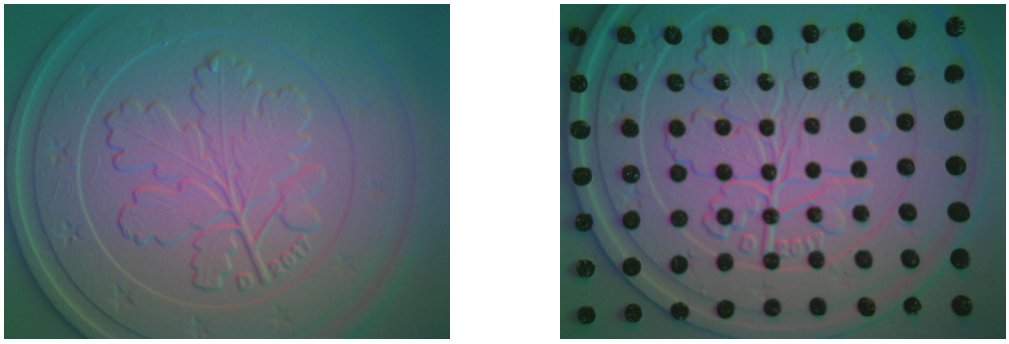


Figure 2.2: Tactile images from a real GelSight Mini sensor. A coin was pressed into the gelpad. On the left is a tactile RGB image and on the right an tactile image with markers.

The camera of the sensor captures the gelpad deformation, resulting in tactile images which can give information about the object's shape, hardness, texture, etc. There are two types of tactile images, *tactile RGB images* and images that showcase the *marker motion field*.

A tactile RGB image can be used to reconstruct the 3D surface of objects with extremely fine details [32], for example.

If the gelpad has markers embedded in it, then the markers are also visible in the tactile image, see Figure 2.2. When the gelpad deforms, the markers move. This movement can be tracked and displayed by drawing arrows from the initial marker position to the new position. This results in a tactile image with a marker motion field. The marker motion field can be used to estimate the normal and shear forces, which occur between sensor and indenter [13].

In general, we distinguish between three components, which make up the simulation of a GelSight tactile sensor. The first one is the physics simulation. Especially interesting here is how the gelpad is simulated since this has a great influence on the tactile images. Secondly, the optical simulation aims to create realistic tactile RGB images. And thirdly, the maker simulation deals with creating realistic marker motion field images.

3 Related Works

In recent years, plenty of research has been done on the simulation of GelSight tactile sensors. This led to the arrival of several simulators, which differ in their approaches to simulating output and physics of GelSight sensors. In the following, we describe the principles of these approaches and how they relate to each other in more detail. We will start with the different approaches used for simulating the physical properties of objects, i.e., the physics simulation. Then we give insights into the current approaches for the optical simulation of GelSight sensors. Last, we describe how marker motion field images are simulated.

Table 3.1 shows an overview of the tactile simulators and the key components of their approaches.

3.1 Physics Simulation

The physics simulation in a tactile simulator deals with the simulation of the physical properties of sensors and indenters. Especially interesting here is how the gelpad is simulated since this has a great influence on the tactile image simulation. An accurate gelpad simulation is needed to simulate the indentation depth correctly, which is important for tactile image generation. Additionally, some approaches for the marker simulation heavily rely on accurate simulation of the soft body gelpad.

Finite Element Methods (FEM) provide an accurate way to simulate the gelpad deformation. Chen et al. [17] and TacIPC [37] use incremental potential contact (IPC) [38] to simulate the gelpad deformation in a FEM-based manner. DiffTactile [18] employs various physics simulation methods to simulate objects. The gelpad deformation is also simulated with a FEM-based approach. Additionally, rigid, elastic, and elastoplastic objects are simulated with the Moving Least Square Material Point Method (MLS-MPM). For cable-like objects

Position Based Dynamics (PBD) is used. Contact dynamics between sensors and objects are simulated with a penalty-based contact model, which is similar to the one from Xu et al. [19].

Tacchi [34] uses the Material Point Method (MPM) to simulate the gelpad deformation. MPM is a particle-based method and requires a lot less computational resources compared to FEM approaches. In their following work [39], the physics simulation was extended by simulating contact between sensor and rigid bodies with the MLS-MPM method.

While FEM-based approaches can be accurate, they can also be quite slow. Therefore, some tactile simulators use rigid body simulation for the gelpad. Higuera et al. [35] and Kim et al. [36] use the rigid body simulation of PyBullet. Xu et al. [19] use a penalty-based contact model. The contact model approximates the soft body deformation of the gelpad with rigid body dynamics. Compared to FEM-based approaches, this allows for significantly faster simulation of the gelpad.

3.2 Optical Simulation

A common step in optical simulation approaches is to generate a height map, which approximates the gelpad deformation. This was first proposed by Gomes et al. [33]. For their optical simulation, a simulated depth camera is used to generate depth maps of the indenter surface. Bivariate Gaussian Filtering is applied to the depth map in non-contact areas to smooth sharp edges. This is done to approximate the deformation of the soft elastomer in a computational fast and efficient manner. After generating the gelpad height map, the RGB values for each pixel of the tactile image are computed by using Phong's model. With Phong's model one can compute how light is reflected from a given surface. For this, surface normals and knowledge regarding the light sources, such as emission direction and intensity of ambient light, are used. Surface normals are obtained from the gelpad height map. The light source knowledge is obtained manually before the simulation is run. For this, the LEDs of a real tactile sensor are observed. For example, what color a light source emits is determined by sampling at a corresponding brightness region in a real tactile image.

This optical simulation approach is used by Tacchi [34]. But here the gelpad deformation is simulated physically as a soft body. Instead of using the object's surface, the surface of the gelpad is used to generate height maps.

Taxim [15] uses a similar approach as Gomes et al. [33] for the optical simulation of GelSight sensors. But instead of using Phong's model, which requires prior knowledge regarding the light sources, it uses examples-based photometric stereo. First, a height map that approximates the gelpad deformation is generated. For the deformation approximation pyramid Gaussian kernels are used to smooth non-contact areas. Then, surface normals of the deformed gelpad are mapped to RGB values with a polynomial lookup table. This polynomial lookup table is created during a calibration process with real tactile images. The process consists of collecting data by pressing a small with a known radius inside the gelpad. Surface normals at each point in the contact area are then computed based on the ball's geometry. This way a dataset for intensity-shape-location pairs is created. To create the lookup table itself least squares is used for computing parameters. Interpolation is used for filling invalid values. An additional step in the Taxim simulation is the simulation of shadows inside the tactile RGB image. For this, they accumulate the shadows, a thin object would throw.

Similar to Taxim's optical simulation approach, is the one from the FOTS [27]. Like in Taxim, a height map, which is smoothed with pyramid Gaussian kernels, is used. Instead a polynomial lookup table, a multi-layer perceptron (MLP)is used to obtain the light intensities for tactile RGB image. The trained MLP maps surface normals of the height map to RGB values. Training happens with data collected in the same fashion as for Taxim. The MLP is only used to simulate tactile RGB images without the shadows. The shadows are simulated separately via a planar shadow generation method.

DiffTactile [18] also uses an MLP to learn the reflectance function. Different from FOTS, surface normals and viewing direction are the function inputs. Additionally, the input for the MLP is first augmented with positional encoding [40]. This enables the MLP to fit data with high-frequency variation better. Like in Taxim and FOTS, the deformation of the gelpad is approximated with pyramid Gaussian kernels.

TacIPC [37] approach mimics the working principles of a real GelSight sensor. The illumination inside the sensor case is simulated with physics-based ray tracing. For this, light sources are positioned inside Unity and the light parameters are adjusted corresponding to a real sensor. The gelpad deformation is simulated as a soft body. In Unity, the light inside the sensor case is reflected from the gelpad and then captured by a camera.

Higuera et al. [35] use a completely different approach to obtain the light intensities of a tactile RGB image. Here the problem of the optical simulation is seen as an image-to-image translation problem and a diffusion model is used. The diffusion model transforms a

grayscale depth image into a tactile RGB image. For the grayscale depth image, the height map of the deformed gelpad in the simulation is used.

3.3 Marker Motion Field Simulation

Taxim [15] uses the linear displacement relationship and superposition principle to simulate the Marker Motion Field. The marker motion simulation requires calibration, which is done with a FEM software.

Xu et al. [19] simulate the normal and shear tactile force fields. The force fields are computed with a penalty-based contact model. For this, multiple points are used to represent the gelpad. The shear and normal forces are computed via the contact model for each point. This results in a tactile force field. Such a force field is different from the vision-based marker field of a real sensor. To obtain a motion field, which one could obtain from a real sensor, the force field is normalized. The normalization leads to the maximal length of the marker displacements to be unit length. This normalized field matches the normalized marker motion field of a real sensor. So, to use this approach as a marker motion simulation, simulated tactile force fields and the marker motion fields of real sensors need to be normalized before they are used as input for an RL policy.

FOTS simulates the marker motion field with exponential functions, which model the marker displacement distributions. Specifically, one exponential function is used for each type of load: normal, shear, and twist. The marker motion is obtained by summing up the three functions and the initial marker positions. Each exponential function contains a coefficient that affects how the markers are displaced during contact. The values for the coefficients are obtained via a calibration process.

Chen et al. [17] use the soft body simulation of the gelpad for their marker motion simulation. Before the physics simulation starts, a mapping for the markers is computed. The mapping relates which marker belongs to which facet of the tetrahedra mesh. If the gelpad deforms, the new marker coordinates in the world can be computed according to the vertex positions of the corresponding facet. To create a marker image out of the marker world positions, the points are projected onto the image plane of a camera with a pinhole model. This way, the pixel coordinates of the markers in the image can be computed. The initial marker positions are obtained by computing the pixel coordinates while the gelpad is not deformed.

DiffTactile [18] uses a similar approach for the marker motion simulation. It relies on the simulated gelpad deformation and uses the positions of the soft body vertices to compute the world positions of the marker location. Likewise, the marker locations are projected to the image plane using the sensor's camera model. The core difference between this approach and the one of Chen et al. [17] lies in the soft body simulation.

A completely different approach is taken by Kim et al. [36]. Their method is based on a generative adversarial network (GAN). Simulated depth and real tactile images were collected for training the GAN: Depth images of objects were generated in a simulated environment. Additionally, a corresponding setup in the real world was used. In both setups, a robot presses on various objects with different indentation depths and contact positions. This data was then used to train the generator and discriminator in the GAN framework. The trained GAN takes a sequence of depth images as input and outputs a tactile RGB image with markers.

Simulator	Simulation Approach		
	Physics	Optical	Marker
Gomes et al. [33] (2021)	-	Phong's Model	-
Taxim [15] (2021)	-	Polynomial look up table	linear displacement relationship + superposition principle
Xu et al. [19] (2022)	penalty-based contact model	-	normalized force field
Tacchi [34] (2023)	MPM	Phong's Model	-
Higuera et al. [35] (2023)	rigid body (PyBullet)	diffusion model	-
Kim et al. [36] (2023)	rigid body (PyBullet)	GAN	GAN
TacIPC [37] (2024)	FEM (IPC)	camera captures light reflected by gelpad	-
DiffTactile [18] (2024)	FEM (Taichi) + penalty-based contact model	MLP with positional encoding	based on gelpad deformation
FOTS [27] (2024)	-	MLP	exp. functions to model marker displacement distributions
Chen et al. [17] (2024)	FEM (IPC)	-	based on gelpad deformation

Table 3.1: Overview of recently developed simulators for GelSight Type sensors and their core components. For each simulator, the core ideas and principles behind the simulation approaches are listed.

4 Methodology

In this chapter, we describe the framework and the employed methods for the tactile simulation in more detail. First, we give an overview of the framework: How is the tactile simulation achieved? Where lies the modularity? And how can it be used for Reinforcement Learning? Then we talk about the sensor setup, i.e., how we modeled the GelSight Mini inside Isaac Sim. After that, we describe the approaches for the physical, the optical, and last but not least, the marker simulation.

4.1 Framework Overview

The tactile sensor simulation has three core components: the physics simulation, the optical simulation, and the marker simulation. Physics simulation is responsible for the interaction between sensors and objects. Specifically important here is the simulation of the gelpad deformation. The optical simulation yields a tactile RGB image. The marker simulation deals with the simulation of the marker motion field. Our marker simulation directly returns the marker displacements, so that they can be directly used for Reinforcement Learning. But if required, a marker motion field image could (theoretically) be returned.

The framework allows us to use different simulation approaches for the core components easily. Changing simulation approaches can be done by changing the simulation configuration. Currently, our framework supports three different kinds of physics simulation for the tactile simulation. For this, we use three different sensor configurations, whose core differences lie in how the gelpad is simulated. The first configuration uses a rigid body gelpad with compliant contacts. Here we leverage the built-in physics engine of Isaac Sim, i.e., PhysX. The second one also uses PhysX, but instead of the rigid body simulation, the deformable body simulation is used. The third one uses our GIPC simulation, which adds simulation steps to the PhysX simulation. Our GIPC simulation is quite different from the pure PhysX simulation. We refer to subsection 4.3.1 for an in-depth explanation of our

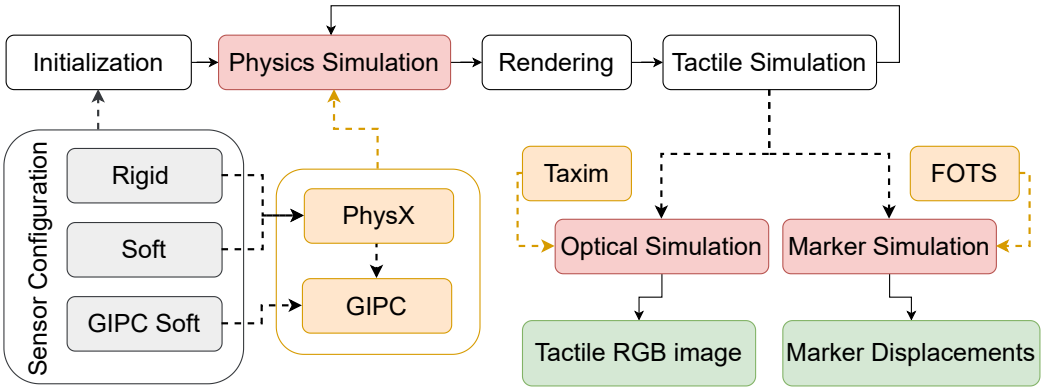


Figure 4.1: Overview of our tactile simulation pipeline. First, the simulation is initialized according to the configuration. Then the physics are simulated, followed by the scene rendering. In the end, the tactile sensor is simulated, i.e., the optical simulation and marker simulation approaches are used. This yields a tactile RGB image and marker displacements data. After this, the physics are simulated again and the process repeats. The core components of the simulation here are physics simulation, optical simulation, and marker simulation. Our framework enables the simple usage of different approaches for each component. One simply changes the configuration. Depending on the sensor configuration, the gelpad is either a rigid body or a soft body. Currently, the physics can be simulated either with PhysX and GIPC or only with PhysX. For the optical simulation, we use an approach based on Taxim. For the marker simulation an approach based on FOTS.

GIPC simulation. Rendering is done by Isaac Sim and identical for different simulation approaches.

In general, our tactile simulation pipeline looks as follows: First, the simulation is initialized. This includes spawning assets, initializing buffers, the physics simulation, etc. After that the physics are simulated and the scene is rendered. Then the tactile sensor output is simulated, i.e. the approach for optical and the one for the marker simulation is used. As a result, a tactile RGB image and the marker displacement data are obtained. If the simulation is not interrupted or aborted, the physics are simulated again and the process repeats. The general tactile simulation pipeline is visualized in Figure 4.1.

For our Reinforcement Learning tasks, we use two base robot configurations. In both

configurations, we use the Franka panda robot [41]. The first configuration has a Franka without a gripper. Instead, it contains a single GelSight Mini sensor, mounted on top of an adapter, which replaces the original panda hand. The other base robot configuration has two GelSight Mini sensors as fingers and corresponding adapters at the gripper. Each base configuration has three variants. One for each sensor configuration, i.e., the robot has GelSight Minis with rigid gelpads, or with soft body gelpads, or uses the GIPC simulation.

Reinforcement Learning is done using the direct RL workflow of Isaac Lab [42]. For the pure PhysX-based configurations, this is straightforward. Here one can proceed like one normally would in Isaac Lab. The only differences are the robot configurations and the tactile sensor. Since we implemented our simulation as an Isaac Lab sensor, using the tactile sensor is like any other sensor of Isaac Lab. Before the simulation is started, the tactile sensors need to be initialized (spawning it is not necessary, since we directly use the sensor models in the robot configuration). During the RL process, the tactile sensors are updated according to our tactile simulation pipeline, which is visualized in Figure 4.1. This is done automatically internally. One only needs to retrieve the desired tactile sensor data.

For the GIPC-based configuration, the default RL workflow cannot be used since we have additional physics steps and a separate scene creation. Therefore, we modify the default direct RL scripts and create a workflow, which mimics the default one. This is described in subsection 4.3.1.

4.1.1 Reasoning behind our Methods

For the physics simulation, we leverage the PhysX. Not only because it is the built-in physics engine of Isaac Sim, but also due its fast GPU-accelerated simulation. We simulate the gelpad as a rigid body with compliant contacts for extremely fast tactile simulation. This can be extremely useful for prototyping new RL environments and algorithms, for example. Besides that, we simulate the gelpad as a soft body to have an approach with accurate gelpad simulation. Additionally, Isaac Sims's soft body simulation has not been used for tactile simulation yet. In this way, our work also provides a first study of the capabilities of Isaac Sims soft body simulation for simulating GelSight sensors. Unfortunately, some of our initial experiments revealed that the built-in soft body simulation is currently lacking in some aspects. We tried to grasp and pick up objects with soft body gelpads, but the objects were constantly slipping away, see subsection 5.4.3. Even after extensive experimentation with different soft body parameters, this behavior did not change. One reason is the lack of static friction in their soft body simulation.

Therefore, we also wanted to integrate an external physics simulator into our framework. It would additionally serve as an example of the extensibility of our framework. But the question here is, which soft body simulation should we use? IPC [38] seemed to be a promising candidate. IPC is extremely robust. It guarantees intersection and inversion free simulation of soft bodies, regardless of material parameters and severity of deformation. This allows us to freely change parameters without worrying about unstable and inaccurate simulations. This is crucial for RL since one technique for closing the Sim2Real gap is domain randomization [43, 44]. Another benefit is, that the need to fine-tune simulation parameters till it behaves reasonably is omitted, or at least greatly reduced. IPC also simulates static and dynamic friction. Furthermore, it has already been shown that IPC can be used for accurate tactile simulation [17]. Instead of IPC, we use GIPC [26], a completely GPU-based variant of IPC with massive speedups.

For the optical simulation, we use the approach from Taxim [15] and for the marker simulation FOTS [27]. Both approaches are based on generating a height map, which approximates the gelpad deformation. As outlined in chapter 3, generating a height map is a common step for the optical simulation. By already having this step implemented, it is easier to integrate other optical simulation approaches. Additionally, both approaches do not rely on accurate simulation of the gelpad. Not only is this beneficial performance-wise but also for using different types of physics simulation. Compared to, for instance, the simulator from Chen et al. [17], we can simulate the marker motion field, even with a rigid body gelpad. Another benefit of Taxim and FOTS is that they can be easily adjusted to simulate other GelSight sensor models. Both use a relatively simple calibration process, which is also fairly similar.

4.2 Sensor Setup

To simulate and render our tactile inside Isaac Sim, we create a USD [45] asset for the GelSight Mini. Our model is based on the publicly available CAD files [46]. We separated the gelpad from the sensor case inside a 3D modeling software and imported them into Isaac Sim as USD assets. Separate meshes for gelpad and sensor case in Isaac Sim allow us to use different physics properties for these parts.

Inside Isaac Sim itself, we additionally added a transparent plate on top of the sensor case, to which we attach the gelpad. Attaching the gelpad happens either through a fixed joint, if we use a rigid body gelpad, or through a soft body attachment. Our model additionally contains a camera, which we place at the bottom of the sensor case. We use the camera

for the optical (section 4.4) and for the marker simulation (section 4.5). For actual usage of the sensors, we attach the sensor case to Franka panda robots [41] with a fixed joint. Then we attach the gelpad to the sensor case. For the rigid body gelpad, we use a fixed joint. For the soft body gelpad, a soft body attachment. For the GIPC-based gelpad, we place the gelpad at the sensor case, precompute the attachment data, and then save it as USD properties. For more details, we refer to subsection 4.3.1.

4.3 Physics Simulation

We use three different approaches to simulate the physical behavior of sensors and objects. The first one uses the PhysX rigid body simulation. Here we simulate the gelpad as a rigid body with compliant contact. For the second approach, we use the soft body simulation of PhysX for the gelpad. Since PhysX is the built-in physics engine of Isaac Sim, setting up these different physics approaches for tactile simulation was straightforward. One only has to apply the corresponding physic properties to the USD assets.

For our third approach, the GIPC-based one, we modified the open source code from GIPC [26] and created Python bindings. In the following subsection, we describe the inner workings of our GIPC simulation. Especially, how the coupling between GIPC and Isaac Sim works and how it can be used for RL.

4.3.1 GIPC Simulation in Isaac Sim

Our GIPC simulation is loosely coupled to Isaac Sim. Isaac Sim is used for setting up the scene, robot simulation, and rendering. GIPC objects, i.e., objects simulated by GIPC, do not interact with PhysX objects. For example, a GIPC object can simply go through a PhysX object. On the contrary, PhysX has some influence on the GIPC objects. Specifically, the robots are simulated by PhysX, and the gelpads are moved kinematically based on the robot's movement. For this, we attach the gelpad to the sensor case. Due to the attachment, the gelpad moves according to how the sensor case moves. Figuratively, we glue some vertices of the gelpad to the sensor case. These vertices are the attachment points and their position depends on the sensor case position. The positions of the other vertices are simulated by GIPC. The basic idea is that the robot in Isaac Sim moves. Then, the sensor case, which is attached to the robot, moves accordingly and obtains a new position. Based on this position, the attachment point positions of the GIPC gelpads are

updated. The GIPC solver then computes the new vertex positions for every GIPC object. As a result, the gelpad and the other GIPC objects move, deform, and interact.

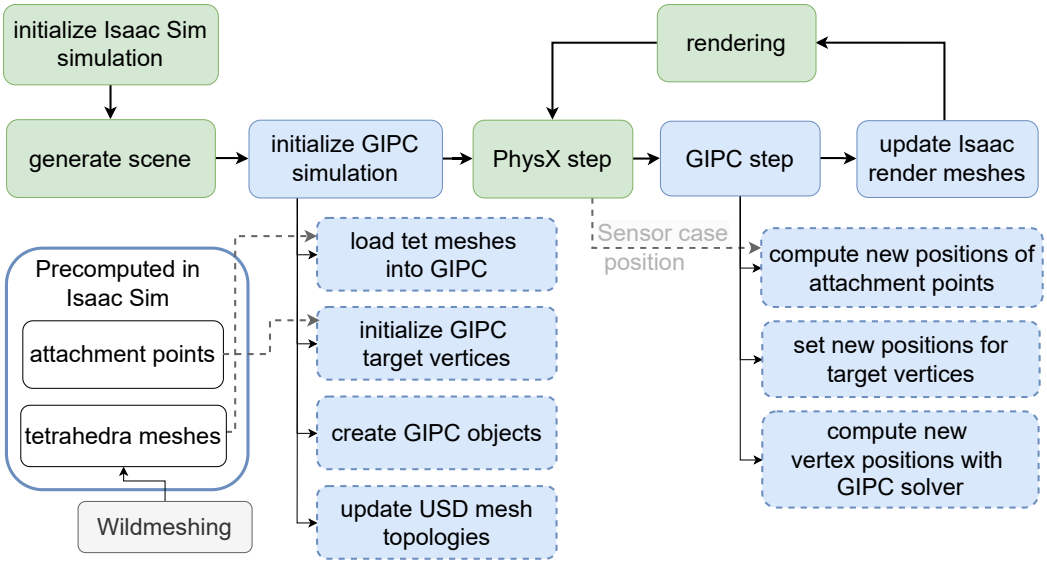


Figure 4.2: An overview of our GIPC simulation pipeline. First, the Isaac Sim simulation is initialized and the scene is created. Then the GIPC simulation is initialized. This includes, loading tetrahedra meshes into GIPC, creating GIPC objects, and updating the corresponding meshes in Isaac Sim. The tetrahedra (tet) meshes are computed with Wildmeshing. Attachment points for the gelpads are also precomputed. For the physics simulation, the simulation state after a time step is computed. First, with PhysX, which leads to, for example, the robot moving, and then with GIPC. Our GIPC simulation takes the newest position of the sensor case and uses it to compute the new positions of the attachment points. Then the GIPC solver computes the new vertex positions of every GIPC object. After that, the meshes in Isaac Sim are updated correspondingly and the scene is rendered.

General GIPC Simulation Pipeline

The general simulation pipeline with GIPC looks as follows. First, the GIPC simulation needs to be set up and initialized. This happens after initialization of the Isaac Sim simulation and generation of the scene. The scene generation involves spawning assets into Isaac Sim. For our GIPC simulation, we spawn assets without physical properties into Isaac Sim and then use them to create GIPC objects. To create a GIPC object out of an asset, we first extract the triangle mesh data of the corresponding USD mesh, i.e., world position and triangle indices of the mesh points. We then generate a tetrahedra mesh, which is required for the GIPC simulation. We use the Wildmeshing [47] python bindings for the tetrahedra generation. The topologies of the Isaac Sim USD meshes are updated according to the surface vertices and triangles of the tetrahedra meshes. This is necessary for the rendering of the objects.

After the initialization of the simulation, we compute the simulation state after a time step. For this, we first do a PhysX step, i.e., compute the new simulation state for objects simulated by PhysX. This, for instance, involves the simulation of the robot movement. The PhysX step is directly followed by a GIPC step. A step in our GIPC simulation consists of first computing the new positions of attachment points. These values are set as target vertex values for the vertices that are attachment points. This allows us to move the GIPC objects kinematically. At the end of the GIPC step, we compute the new vertex positions for all GIPC objects with the GIPC solver. Additionally, we update the object position data by computing the mean of the new vertex positions. The object position data is useful for RL environments.

To render the results of the GIPC simulations inside Isaac Sim, we update the position of the USD mesh vertices with the new computed vertex positions. For updating the USD meshes fast, we use the USDRT [48] API. The rest is done by Isaac Sim's rendering engine. The general simulation pipeline with GIPC and PhysX is visualized in Figure 4.2.

For RL, we also need to reset the GIPC simulation occasionally. Resetting means, bringing the scene back to the initial state. To achieve this, we save the initial positions of the GIPC vertices during the scene initialization. When the reset happens, we set the initial positions as the position of the vertices. The velocities are set to $(0, 0, 0)$.

GIPC Attachments

The two core questions regarding the attachment points are:

1. How do we find attachment points?
2. How do we compute the new vertex positions for the attachment points after each robot movement?

Finding attachment points means finding the IDs of vertices, that should be attachment points. Attachment points should be the vertices inside the sensor case or at least close to it. To attach a GIPC object to the sensor case, we first query world position and orientation of the sensor case. Secondly, we iterate through each point of the GIPC object's tetrahedra mesh (tet point) and do sphere ray casting in Isaac Sim with the PhysX scene query interface. A sphere with a specified radius is swept out from an origin point in a direction with the specified maximum. If the sphere hits a collider mesh, the impact point is returned. By using the world position of the tet points as the origin point, a very small sphere radius, and a very small maximum distance, we can check if a tet point is inside or close enough to a rigid body. This way, we find the attachment points.

For computing the new positions of the attachment points, we use that the relative positions of the attachment points to the sensor case position are constant. We precompute and save the offsets between attachment points and sensor case position. These offsets stay the same throughout the simulation. The attachment points positions are then updated by first querying the current pose, i.e., position and orientation, of the rigid body. Then the attachment offsets are transformed based on the current pose. The transformed offset points are the new attachment point positions.

Computing which vertices are attachment points and the corresponding offsets happens before the simulation is run. For this, we wrote a script, which can be used in the Isaac Sim GUI. The attachment data is saved as USD properties and retrieved during the GIPC initialization.

4.4 Optical Simulation

For the optical simulation, we use the approach from Taxim [15]. Specifically, a GPU accelerated version [49]. With its Torch GPU implementation, we can render tactile RGB images without shadows 21 and with shadows 25 times faster.

First, we generate a height map of the object's surface with the camera of our sensor model. Cameras inside Isaac Sim have multiple possible outputs, such as RGB image, normals, and motion vectors. To generate the object height map H_{obj}, we use *distance_to_image_plane*.

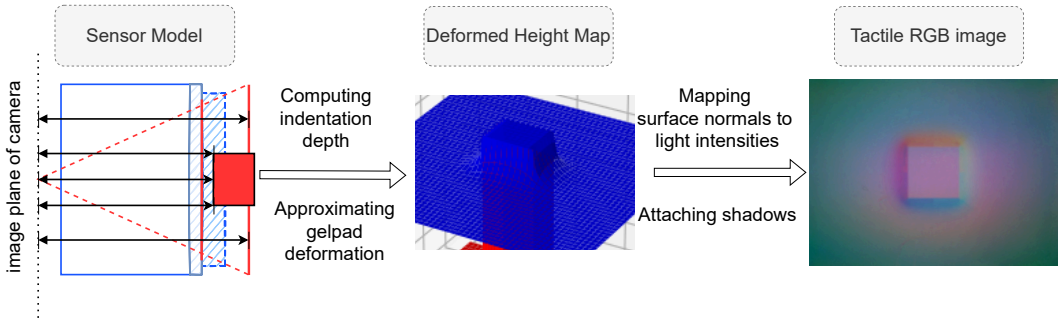


Figure 4.3: An overview of the optical simulation pipeline. First, a height map of the object's surface is generated by using the camera of the sensor model. Based on the distance of the object to the sensor case, the indentation depth is computed. The gelpad and the object height map are combined and smoothed with Gaussian pyramid kernels. Then the surface normals of the deformed height map are mapped to light intensities, resulting in a tactile RGB image. In the end, shadows are attached to the image to make it more realistic.

As the name implies, it gives us the distance of the objects in the camera view to the image plane at each pixel. The smaller the distance, the closer is the object to the camera. With our sensor base model, this means the smaller the distance, the closer to the sensor case. We want our object height map to have the value 0 if the object is directly at the sensor case. Therefore, we subtract the camera minimum distance from the *distance_to_image_plane* values.

The gelpad height map represents the gelpad surface. In our case, the gelpad is flat, therefore we use a simple flat plane. Object height map and gelpad height map are combined by shifting them according to the maximum indentation depth Δh_{max}. It can be computed with

$$\Delta h_{\text{max}} = \begin{cases} 0, & \min(H_{\text{obj}}) > h_{\text{gelpad}} \\ h_{\text{gelpad}} - \min(H_{\text{obj}}), & \text{otherwise} \end{cases}$$

where h_{gelpad} is the gelpad height at the location of $\min(H_{\text{obj}})$. In our case, we use a constant h_{gelpad}, because the GelSight Mini has a planar surface. In the combined height map the gelpad height map is on top of the object height map. We smooth the combined height map with pyramid Gaussian kernels to obtain an approximation for the gelpad deformation. Then we use the polynomial lookup table to map the surface normals of the

deformed height map to RGB values. As a final step, we attach shadows to the image. The simulation pipeline is visualized in Figure 4.3.

4.5 Marker Motion Simulation

To simulate the marker motion field, we use the marker motion simulation approach from FOTS [27]. With FOTS we can create a maker motion field based on a height map. Compared to relying on the gelpad deformation, this allows us to use different physics simulation methods for the gelpad easily.

The marker motion field shows the position changes of the markers after the gelpad deformation. For this, we first need to obtain the initial positions of the markers. For our marker simulation, we use a single pixel as a marker. Its initial position equals its pixel location in the tactile image when the gelpad is not deformed. We use a uniform grid to define the initial position based on the image resolution. The pixel coordinates are chosen so that we have N markers column-wise and M markers row-wise in the tactile image.

To obtain the marker motion field, we first generate a height map that approximates the gelpad deformation. This process identical to our optical simulation. Every time the sensor is updated, the marker positions are updated based on the height map and a trajectory. The trajectory describes the relative position and rotation of the indenter to the gelpad at every simulation step.

The position update is computed with three exponential functions that model the marker displacement motion under different loads. One function models the dilate motion which results from normal load and relative position between markers M and markers in contact C_i. Markers in contact are those whose values in the height map are > 0. The displacement due to dilate motion Δd_d is modeled by

$$\Delta d_d = \sum_{i=1}^{N} \Delta h_i \cdot (M - C_i) \cdot \exp(-\lambda_d \|M - C_i\|_2^2), \tag{4.1}$$

where N is the total number of markers C_i and Δh_i is the height map value of the marker C_i. Δh_i is obtained from the height map from the height map generation process described in section 4.4. These are the height values at the pixel coordinates of the marker C_i. $M - C_i$ is computed by subtracting the marker positions and the positions of the markers, which are part of the contact map.

One function is for the displacement due to shear motion. It is modeled by

$$\Delta d_s = \min\{\Delta s, \Delta s_{\max}\} \cdot \exp(-\lambda_s \|M - G\|_2^2), \tag{4.2}$$

where G is the position of the indenter relative to the gelpad surface. Δs is the distance between the current indenter position and the initial indenter position after contact. For G, we use the center of the contact area which appears during the first contact. We compute the contact center by taking the mean of the contact area, which appears in the height map. In other words, we take the mean of the pixel coordinates where $\Delta h_i > 0$. While object and gelpad are in contact, we keep track of the contact center. Δs is then computed by subtracting the initial contact center from the newest contact center.

And one function is for the displacement due to twist motion. It is modeled by

$$\Delta d_t = \min\{\Delta \theta, \Delta \theta_{\max}\} \cdot (M - G) \cdot \exp(-\lambda_t \|M - G\|_2^2), \tag{4.3}$$

where $\Delta\theta$ is the rotation between the current indenter position and the one from the initial contact. For $\Delta\theta$, we retrieve the position and rotation of the object as well as the ones for the gelpad from Isaac Sim. We then compute the relative rotation around the z-axis from the object to the gelpad. This is our θ Like for Δs, we keep track of the values while the object and gelpad are in contact. $\Delta\theta$ is then computed by substracting the inital θ from the current one. This only works for rigid body objects, since we cannot obtain information regarding the rotation from the soft bodies.

λ_d, λ_s and λ_t are coefficients which determine how the motion types influence the maker displacements. They are determined via a calibration process. The complete marker motion field is obtained by taking the sum of initial marker positions and the displacements caused by the three motion types. For our purposes, we only use the displacement data.

5 Demonstrations

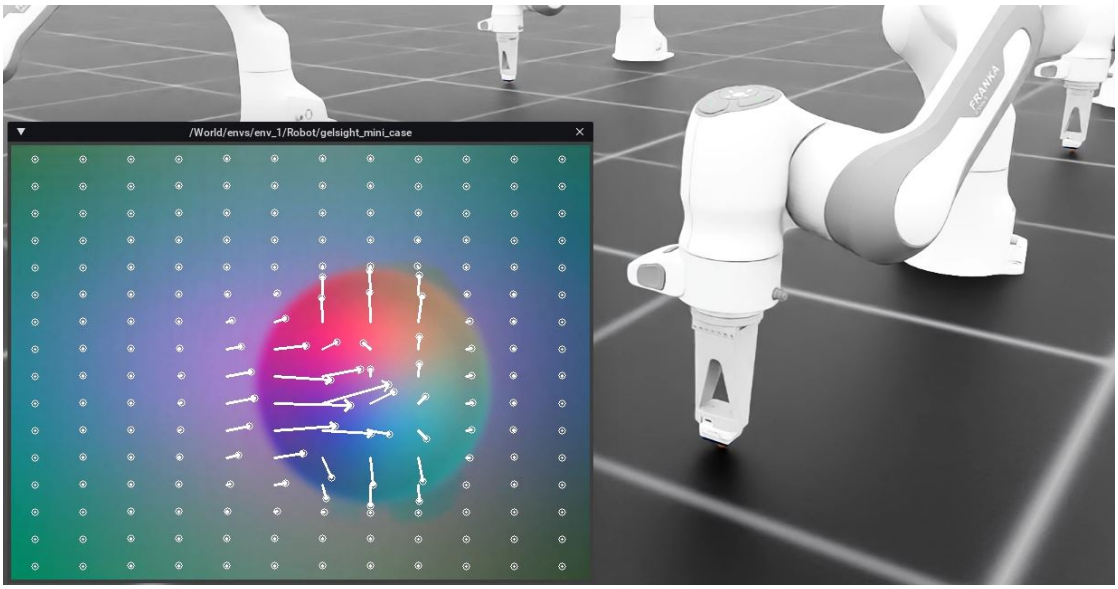


Figure 5.1: For showcasing a dynamic scene, we do a ball rolling experiment. The robots rolls a sphere around the ground with a single GelSight Mini sensor. We additionally retrieve the output of the tactile sensor, i.e., marker motion field and tactile RGB image.

We showcase the behavior and capabilities of our framework with multiple examples. For each example, we use all three sensor configurations. First, a ball rolling experiment as an example for a contact-rich scene with a single GelSight sensor. Secondly, an example for grasping of objects with soft body gelpads. And thirdly, one example which showcases the GIPC simulation under challenging conditions.
Furthermore, we include three Reinforcement Learning Tasks: object pushing, lifting, and

pole balancing. These environments demonstrate how and that our framework can be used for RL.

5.1 Ball Rolling

To showcase how the simulation behaves in a dynamic setting, we do a ball rolling experiment, similar to Xu et al. [19]. But unlike their experiment, we simulate a full robot to roll the ball. We use a setup, which is similar to our RL environments. The goal is to mimic the usage of the tactile sensor for manipulation of an object with a robot. This way, we gauge how our simulation performs in such a setting. Compared to actual usage in RL, we have more control about the movement and it is easier to investigate the tactile sensor performance.

For this showcase, we use a ball with a radius of $5\,\mathrm{mm}$ and place it on the ground. A robot with a single GelSight Mini sensor uses the gelpad to roll the ball around. For this, we define goal positions for the endeffector and compute the required joint values with differential inverse kinematics. The goal positions are computed based on the ball position. The scene is visualized in Figure 5.1. We can simulate about 20 robots at the same time with the rigid body configuration, till we hit the VRAM limits ($12\,\mathrm{GB}$) of our machine. The problem here is that cameras in Isaac Sim require a lot of VRAM. For the GIPC-based simulation, we can only simulate a single robot properly due to our VRAM limit.

5.2 Lift

In this example, we try to grasp and lift basic objects. We use the robot with the two GelSight sensors. The example reveals that the PhysX soft body setup cannot be used to grasp and lift objects. Even after various experiments regarding the soft body parameters, the objects always slip away. Also, using a single soft body gelpad and a rigid body gelpad does not work reliable for grasping and lifting. One reason for this is that the soft body simulation of PhysX currently does not support static friction.

This was the main reason for integrating a different kind of soft body simulation into our framework. Compared to the PhysX simulation, the GIPC soft bodies can be used to grasp and lift other objects. This is visualized in Figure 5.3.

Figure 5.2: The lifting example with soft body gelpads, which are simulated with PhysX. The robot is unable to lift objects. The rigid object just slips down.

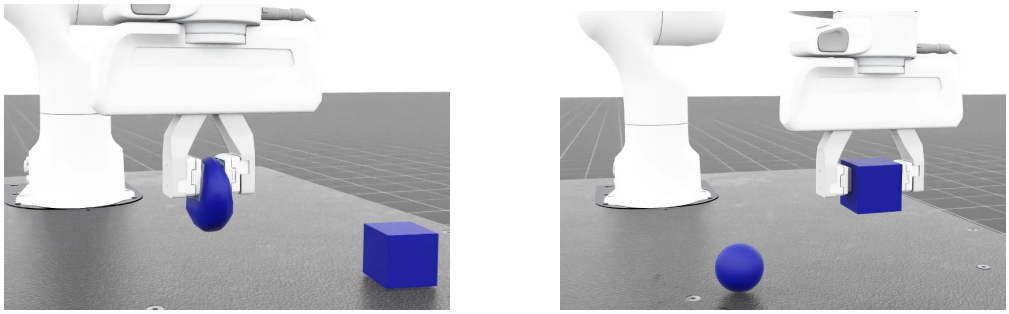


Figure 5.3: A robot grasping objects with soft body gelpads. The soft bodies are simulated with GIPC. On the left, the gripper additionally squeezes a sphere.

5.3 Twist

We use this example to showcase the capabilities of the GIPC soft body simulation. A beam is simulated as a soft body. It is attached to a plate. The robot grasps the top of the beam with two soft body gelpads, twists and stretches the beam till it snaps back. Three frames of the example are shown in Figure 5.4. The showcase demonstrates that the simulation stays stable even under extreme deformations. Additionally, it showcases that friction is reasonably simulated.

Figure 5.4: Three frames from the twist example. A robot with two soft body gelpads grasps a soft body beam, twists and stretches it. The soft bodies are simulated with GIPC. The simulation is stable even under large deformations and shows reasonable friction handling.

5.4 Tactile Reinforcement Learning Tasks

We implemented three RL environments and trained policies to validate that our framework can be used for Reinforcement Learning. In each environment, we use or tactile sensor. For training the RL policies, we used the PPO [50] implementation of [23]. Main reason being that PPO is used in the RL examples of Isaac Lab. Since our environments are similar to some of these RL environments, we used similar PPO parameters as they did. We additionally used similar reward functions as the environments, which are similar to ours (*Isaac-Lift-Cube-Franka-v0* and *Isaac-Cartpole-Direct-v0*). The environments are implemented as direct RL environments. This made it easier to use our GIPC simulation.

The observations for each environment include the marker displacements of the sensors. In the following, we describe setup, reset conditions, and rewards of the environments.

5.4.1 Object Pushing

This is the simplest of the three tasks. Here we use the robot with a single GelSight sensor. The goal is to push the cube to a target position. The cube is spawned on the ground around the robot base. An environment is reset when the endeffector is too far away from

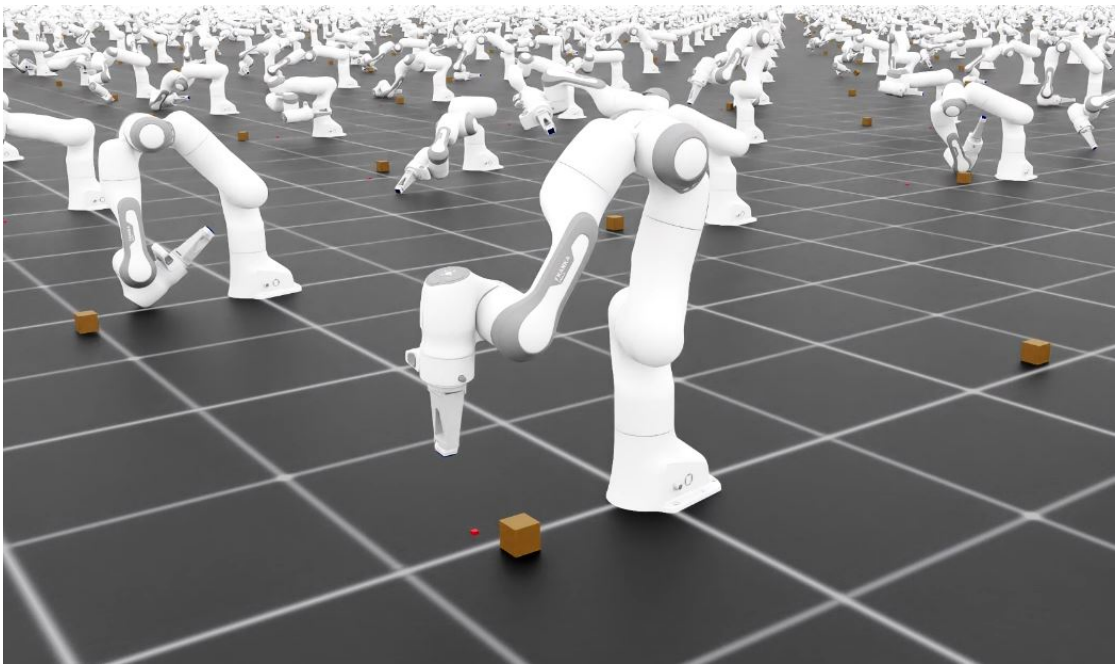


Figure 5.5: Screenshot of the object pushing RL environment with multiple robots. Without tactile simulation and soft bodies, we can train a massive amount of robots, which is ideal for prototyping. The initial robot position can be seen at the front robot. Target positions are visualized in red.

the cube, i.e., distance $> 1\,\mathrm{m}$. As a reward, we use a reaching reward, which rewards the agent based on a tanh kernel and the object distance to the endeffector. We also use a reward for tracking the target pose. It is like the reaching reward but uses the distance from object to target position.

For prototyping environments, one can use the rigid body configuration of the sensor, or even go without tactile simulation. Without tactile simulation, a massive amount of robots can be trained at the same time. This is visualized in Figure 5.5.

5.4.2 Pole Balancing

The goal here is to balance a thin pole for as long as possible. We use the robot configuration with a single sensor. The initial position of the robot can be seen at the front of Figure 5.6.

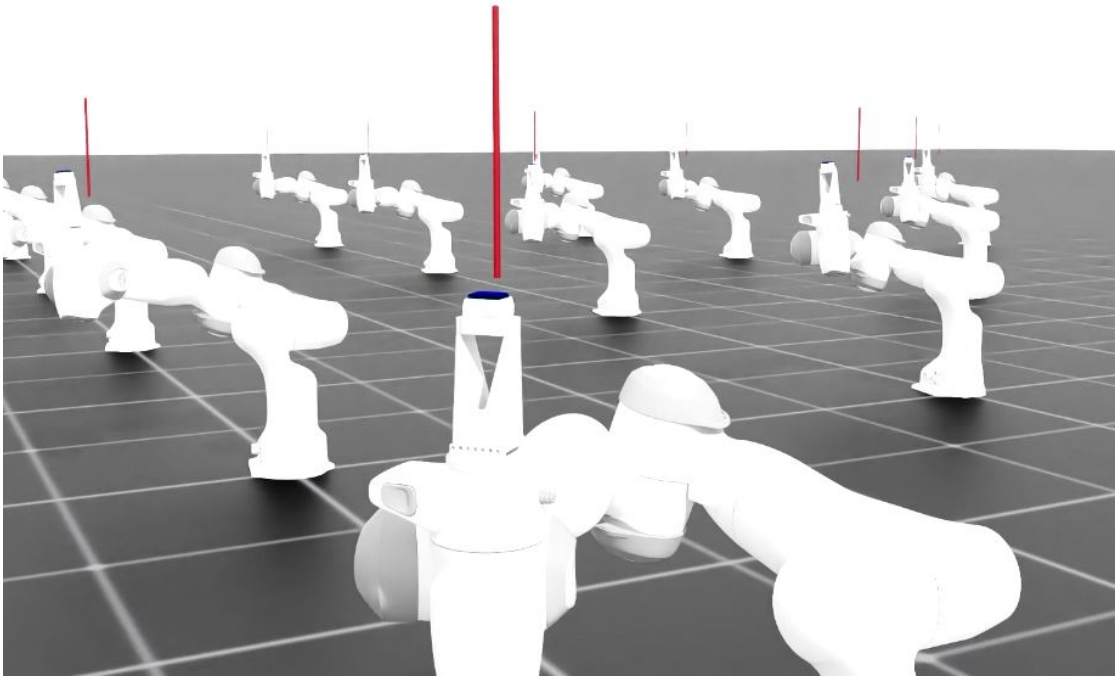


Figure 5.6: Screenshot of the RL training with multiple robots for the pole balancing task.

We use 12 markers per row and 16 markers per column for our marker grid pattern. An environment is reset when the pole is too far away from the gelpad, if the pole height is below $0.4\,\mathrm{m}$, or if the x and y angles of the pole are bigger than 90 degrees. Our reward function is similar to the Cartpole example of Isaac Lab (*Isaac-Cartpole-Direct-v0*).

5.4.3 Object Lifting

This environment is basically like the Lift example of Isaac Lab (*Isaac-Lift-Cube-Franka-v0*), but with tactile sensors. The robot has to pick up a cube and move it to a target position. We use the robot configuration with two GelSight sensors and spawn a cube on the ground. The environment is reset when the endeffector is too far away from the object, i.e., distance $> 1\,\mathrm{m}$. Our reward function here consists of three sub-rewards: A reaching reward, which rewards the agent based on the distance of the endeffector to the object with a tanh kernel. A lifting reward, which is applied when the cube is lifted above a minimal height. And a reward for moving the cube towards the target position, which

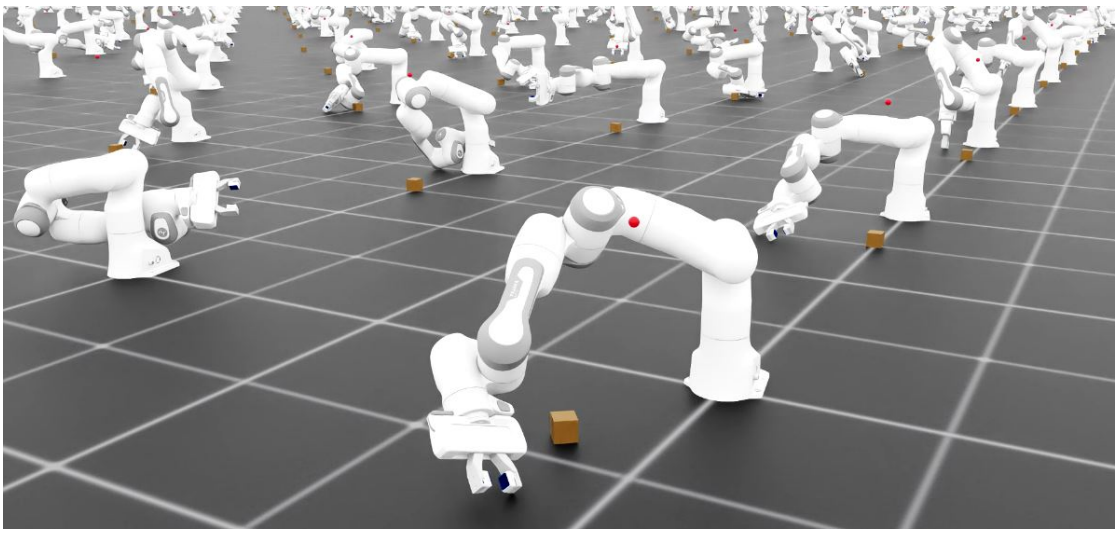


Figure 5.7: Screenshot of the object lifting environment. Target positions are visualized with a red sphere.

is added when the cube is above the minimal height. It is computed like the reaching reward, but instead of the object position, the target position is used.

6 Conclusion and Future Work

To sum it up, we developed a novel framework for simulating GelSight tactile sensors. The framework enables the usage of GelSight Mini sensors for Reinforcement Learning. It is built on top of Isaac Sim and Isaac Lab, which gives the user access to a wealth of features non-existing in current tactile simulators. We designed the framework to be modular, extendable, and easy to use. The framework integrates multiple different simulation approaches. The gelpad can either be simulated as a rigid body with compliant contact, or as a soft body. For the soft body simulation, one can either use PhysX or our GIPC simulation, which we integrated into Isaac Sim/Lab. To simulate the sensor output, we use height map based approaches. Specifically, Taxim [15] for the optical and FOTS [27] for the marker simulation. We demonstrated features of our framework and behavior of its simulations with multiple examples. Additionally, we provide three RL environments for RL with tactile sensing.

A limitation of our current work is that it only contains qualitative experiments. Furthermore, our tactile simulations, specifically the physics simulation approaches, lack experiments which investigate whether they can be used for Sim2Real or not. Therefore, we aim to do more quantitative experiments for comparing different tactile simulation approaches, as well as Sim2Real experiments. We also plan to extend our framework to include more RL environments and more tactile simulation approaches. Basically creating a benchmark for tactile simulators and RL algorithms for tactile sensing.

Future work could also be related to Reinforcement Learning. Naturally, our framework could be used for research on the manipulation of soft bodies. Besides that, our framework could also be used for research on fusing vision and tactile sensing. Since Isaac Sim features rendering that can be photorealistic, our framework seems to be suitable for this.

Another direction for future works lies in soft body simulation. One idea is to include a grasping setup similar to DefGraspSim [51], but with tactile sensors and different soft body simulation methods. This could, for example, be used to obtain a more detailed evaluation of different soft body simulation methods based on different metrics. It could

also be used for investigating how the accuracy of the soft body simulation affects the tactile simulation. This could be beneficial for finding reasonable trade-offs between simulation accuracy and speed.

One could also work on new methods for adaptive remeshing. Specifically, working on a GPU accelerated method for in-timestep remeshing for GIPC. In-Timestep Remeshing is an algorithm for adaptive remeshing in IPC [52]. The algorithm refines and coarsens tetrahedra meshes automatically based on how the physical energy changes. Such an algorithm would be highly beneficial for tactile simulation. Since the soft body simulation accuracy depends on the mesh resolution, this could lead to the soft bodies always being accurate enough. Another benefit would be, that a lot of manual tuning for the soft body simulation can be omitted. Problematic here is that the algorithm is too slow for usage in RL. A GPU accelerated algorithm might be the solution to this.

Another way to improve the GIPC simulation would be to extend it with rigid bodies. Currently, it only supports soft bodies. While, rigid bodies can be approximated by using a very high stiffness, this is computationally very expensive. The same problem was present in IPC and was addressed in [53, 54]. Integrating rigid bodies into GIPC would be highly beneficial for more general manipulation tasks.

Bibliography

[1] Qiang Li et al. “A Review of Tactile Information: Perception and Action Through Touch”. In: *IEEE Transactions on Robotics* 36.6 (Dec. 2020). Conference Name: IEEE Transactions on Robotics, pp. 1619–1634. ISSN: 1941-0468. DOI: 10.1109/TRO.2020.3003230. URL: https://ieeexplore.ieee.org/document/9136877/?arnumber=9136877 (visited on 08/24/2024).

[2] Wenzhen Yuan et al. “Measurement of Shear and Slip with a GelSight Tactile Sensor”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015 IEEE International Conference on Robotics and Automation (ICRA). Seattle, WA, USA: IEEE, May 2015, pp. 304–311. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139016.

[3] Yunhai Han et al. “Learning Generalizable Vision-Tactile Robotic Grasping Strategy for Deformable Objects via Transformer”. In: *IEEE/ASME Transactions on Mechatronics* (2024). Conference Name: IEEE/ASME Transactions on Mechatronics, pp. 1–13. ISSN: 1941-014X. DOI: 10.1109/TMECH.2024.3400789. URL: https://ieeexplore.ieee.org/document/10552075/?arnumber=10552075 (visited on 09/05/2024).

[4] Wenzhen Yuan et al. “Shape-independent Hardness Estimation Using Deep Learning and a GelSight Tactile Sensor”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 951–958. DOI: 10.1109/ICRA.2017.7989116. arXiv: 1704.03955[cs]. URL: http://arxiv.org/abs/1704.03955 (visited on 09/05/2024).

[5] Gaozhao Wang et al. “Visual-Tactile Perception Based Control Strategy for Complex Robot Peg-in-Hole Process via Topological and Geometric Reasoning”. In: *IEEE Robotics and Automation Letters* 9.10 (Oct. 2024), pp. 8410–8417. ISSN: 2377-3766. DOI: 10.1109/LRA.2024.3436334.

[6] J. Rogelio Guadarrama-Olvera et al. “Enhancing Biped Locomotion on Unknown Terrain Using Tactile Feedback”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids). Nov. 2018, pp. 1–9. DOI: 10.1109/HUMANOIDS.2018.8625024.

[7] Brenna D. Argall and Aude G. Billard. “A Survey of Tactile Human–Robot Interactions”. In: *Robotics and Autonomous Systems* 58.10 (Oct. 2010), pp. 1159–1176. ISSN: 09218890. DOI: 10.1016/j.robot.2010.07.002.

[8] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[9] JaeKwan Park et al. “Control automation in the heat-up mode of a nuclear power plant using reinforcement learning”. In: *Progress in Nuclear Energy* 145 (2022), p. 104107.

[10] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. eprint: https://doi.org/10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721.

[11] OpenAI et al. *Solving Rubik’s Cube with a Robot Hand*. Oct. 15, 2019. arXiv: 1910.07113 [cs, stat]. Pre-published.

[12] Mahmoud Meribout et al. “Tactile Sensors: A Review”. In: *Measurement* 238 (Oct. 2024), p. 115332. ISSN: 02632241. DOI: 10.1016/j.measurement.2024.115332.

[13] Wenzhen Yuan, Siyuan Dong, and Edward Adelson. “GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force”. In: *Sensors* 17.12 (Nov. 29, 2017), p. 2762. ISSN: 1424-8220. DOI: 10.3390/s17122762. URL: https://www.mdpi.com/1424-8220/17/12/2762 (visited on 09/04/2024).

[14] Shixin Zhang et al. “Hardware Technology of Vision-Based Tactile Sensor: A Review”. In: *IEEE Sensors Journal* 22.22 (Nov. 2022), pp. 21410–21427. ISSN: 1558-1748. DOI: 10.1109/JSEN.2022.3210210.

[15] Zilin Si and Wenzhen Yuan. *Taxim: An Example-based Simulation Model for GelSight Tactile Sensors*. Dec. 14, 2021. arXiv: 2109.04027 [cs]. Pre-published.

[16] Yijiong Lin et al. *Tactile Gym 2.0: Sim-to-real Deep Reinforcement Learning for Comparing Low-cost High-Resolution Robot Touch*. July 27, 2022. arXiv: 2207.10763 [cs]. Pre-published.

[17] Weihang Chen et al. “General-Purpose Sim2Real Protocol for Learning Contact-Rich Manipulation With Marker-Based Visuotactile Sensors”. In: *IEEE Transactions on Robotics* 40 (2024), pp. 1509–1526. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2024.3352969.

[18] Zilin Si et al. *DIFFTACTILE: A Physics-based Differentiable Tactile Simulator for Contact-rich Robotic Manipulation*. Mar. 13, 2024. arXiv: 2403.08716 [cs]. Pre-published.

[19] Jie Xu et al. “Efficient Tactile Simulation with Differentiability for Robotic Manipulation”. In: *6th Annual Conference on Robot Learning*. 2022.

[20] NVIDIA. *NVIDIA Isaac Sim*. URL: https://developer.nvidia.com/physx-sdk (visited on 09/09/2024).

[21] NVIDIA. *Isaac Lab*. URL: https://isaac-sim.github.io/IsaacLab/ (visited on 09/09/2024).

[22] Mayank Mittal et al. “Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments”. In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: 10.1109/LRA.2023.3270034.

[23] Lukas Schneider et al. *Learning Risk-Aware Quadrupedal Locomotion using Distributional Reinforcement Learning*. 2023. arXiv: 2309.14246 [cs.RO].

[24] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[25] Inc. GelSight. *GelSight Mini*. URL: https://www.gelsight.com/gelsightmini/ (visited on 09/09/2024).

[26] Kemeng Huang et al. “GIPC: Fast and Stable Gauss-Newton Optimization of IPC Barrier Energy”. In: *ACM Trans. Graph.* 43.2 (Mar. 2024). ISSN: 0730-0301. DOI: 10.1145/3643028. URL: https://doi.org/10.1145/3643028.

[27] Yongqiang Zhao et al. *FOTS: A Fast Optical Tactile Simulator for Sim2Real Learning of Tactile-motor Robot Manipulation Skills*. Apr. 30, 2024. arXiv: 2404.19217 [cs]. Pre-published.

[28] Alexander C. Abad and Anuradha Ranasinghe. “Visuotactile Sensors With Emphasis on GelSight Sensor: A Review”. In: *IEEE Sensors Journal* 20.14 (July 2020), pp. 7628–7638. ISSN: 1558-1748. DOI: 10.1109/JSEN.2020.2979662.

[29] Mike Lambeta et al. “DIGIT: A Novel Design for a Low-Cost Compact High-Resolution Tactile Sensor With Application to In-Hand Manipulation”. In: *IEEE Robotics and Automation Letters* 5.3 (July 2020), pp. 3838–3845. ISSN: 2377-3774. DOI: 10.1109/lra.2020.2977257. URL: http://dx.doi.org/10.1109/LRA.2020.2977257.

[30] Shaoxiong Wang et al. “GelSight Wedge: Measuring High-Resolution 3D Contact Geometry with a Compact Robot Finger”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021 IEEE International Conference on Robotics and Automation (ICRA). Xi’an, China: IEEE, May 30, 2021, pp. 6468–6475. ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9560783.

[31] Changyi Lin et al. *9DTact: A Compact Vision-Based Tactile Sensor for Accurate 3D Shape Reconstruction and Generalizable 6D Force Estimation*. Dec. 15, 2023. arXiv: 2308.14277 [cs]. Pre-published.

[32] Junyuan Lu, Zeyu Wan, and Yu Zhang. “Tac2Structure: Object Surface Reconstruction Only through Multi Times Touch”. In: *IEEE Robotics and Automation Letters* 8.3 (Mar. 2023), pp. 1391–1398. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2023.3238190. arXiv: 2209.06545 [cs].

[33] Daniel Fernandes Gomes, Paolo Paoletti, and Shan Luo. *Generation of GelSight Tactile Images for Sim2Real Learning*. Jan. 18, 2021. arXiv: 2101.07169 [cs]. Pre-published.

[34] Zixi Chen et al. “Tacchi: A Pluggable and Low Computational Cost Elastomer Deformation Simulator for Optical Tactile Sensors”. In: *IEEE Robotics and Automation Letters* 8.3 (Mar. 2023), pp. 1239–1246. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2023.3237042.

[35] Carolina Higuera, Byron Boots, and Mustafa Mukadam. *Learning to Read Braille: Bridging the Tactile Reality Gap with Diffusion Models*. Apr. 3, 2023. arXiv: 2304.01182 [cs]. Pre-published.

[36] Won Dong Kim et al. “Marker-Embedded Tactile Image Generation via Generative Adversarial Networks”. In: *IEEE Robotics and Automation Letters* 8.8 (Aug. 2023), pp. 4481–4488. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2023.3284370.

[37] Wenxin Du et al. “TacIPC: Intersection- and Inversion-Free FEM-Based Elastomer Simulation for Optical Tactile Sensors”. In: *IEEE Robotics and Automation Letters* 9.3 (Mar. 2024), pp. 2559–2566. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2024.3357030.

[38] Minchen Li et al. “Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics”. In: *ACM Transactions on Graphics* 39.4 (Aug. 31, 2020). ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3386569.3392425.

[39] Jianhua Shan et al. *Soft Contact Simulation and Manipulation Learning of Deformable Objects with Vision-based Tactile Sensor*. May 12, 2024. arXiv: 2405.07237 [cs]. Pre-published.

[40] Ben Mildenhall et al. “NeRF: representing scenes as neural radiance fields for view synthesis”. In: *Commun. ACM* 65.1 (Dec. 2021), pp. 99–106. ISSN: 0001-0782. DOI: 10.1145/3503250. URL: https://doi.org/10.1145/3503250.

[41] Franka Robotics. *Franka Panda Arm*. URL: https://franka.de/research (visited on 09/21/2024).

[42] The Isaac Lab Project Developers. *Creating a Direct Workflow RL Environment*. URL: https://isaac-sim.github.io/IsaacLab/source/tutorials/03_envs/create_direct_rl_env.html (visited on 09/21/2024).

[43] Bhairav Mehta et al. “Active Domain Randomization”. In: ().

[44] Josh Tobin et al. “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)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sept. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[45] Pixar Animation Studio. *Universal Scene Description (USD)*. URL: https://openusd.org/release/api/index.html (visited on 09/21/2024).

[46] Inc. GelSight. *GelSight Robotics Software*. Version 4.0. Nov. 2021. URL: https://github.com/gelsightinc/gsrobotics.

[47] Yixin Hu et al. “Fast Tetrahedral Meshing in the Wild”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392385. URL: https://doi.org/10.1145/3386569.3392385.

[48] NVIDIA. *USD, Fabric and USDRT*. URL: https://docs.omniverse.nvidia.com/kit/docs/usdrt/latest/docs/usd_fabric_usdrt.html (visited on 09/09/2024).

[49] Tim Schneider. *Taxim-GPU*. URL: https://git.ias.informatik.tu-darmstadt.de/tactile-sensing/taxim-gpu (visited on 09/09/2024).

[50] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[51] Isabella Huang et al. *DefGraspSim: Simulation-based Grasping of 3D Deformable Objects*. July 12, 2021. arXiv: 2107.05778 [cs]. Pre-published.

[52] Zachary Ferguson et al. “In-Timestep Remeshing for Contacting Elastodynamics”. In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–15. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3592428.

[53] Zachary Ferguson et al. “Intersection-Free Rigid Body Dynamics”. In: 40.4 ().

[54] Lei Lan et al. *Affine Body Dynamics: Fast, Stable & Intersection-free Simulation of Stiff Materials*. Jan. 31, 2022. arXiv: 2201.10022 [cs]. Pre-published.