TACTO: A Simulator for Learning Control from Touch Sensing

Shaoxiong Wang, Mike Lambeta, Po-Wei Chou, Roberto Calandra
Facebook AI Research
{sxwang, lambetam, poweic, rcalandra}@fb.com

Abstract

Simulators perform an important role in prototyping, debugging and benchmarking new advances in robotics and learning for control. Although many physics engines exist, some aspects of the real-world are harder than others to simulate. One of the aspects that have so far eluded accurate simulation is touch sensing. To address this gap, we present TACTO – a fast, flexible and open-source simulator for vision-based tactile sensors. This simulator allows to render realistic high-resolution touch readings at hundreds of frames per second, and can be easily configured to simulate different vision-based tactile sensors, including GelSight, DIGIT and OmniTact. In this paper, we detail the principles that drove the implementation of TACTO and how they are reflected in its architecture. We demonstrate TACTO on a perceptual task, by learning to predict grasp stability using touch from 1 million grasps, and on a marble manipulation control task. We believe that TACTO is a step towards the widespread adoption of touch sensing in robotic applications, and to enable machine learning practitioners interested in multi-modal learning and control. TACTO is open-sourced at `https://github.com/facebookresearch/tacto`.

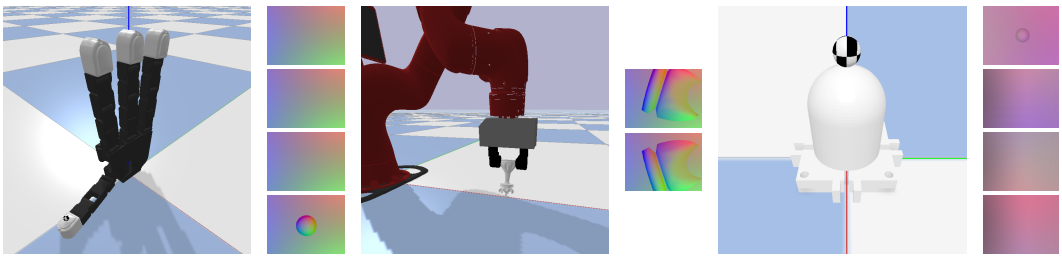

Figure 1: We open-source TACTO – a simulator of vision-based tactile sensors. TACTO produces high-resolution and high-fidelity reading from tactile sensors at high-frequency ($>$100 Hz). Its modular structure allows to model different vision-based tactile sensors and to be integrated with different physics engines. We believe that such tool can benefit the touch sensing and robotic community, as well as researchers in machine learning that can now access a new sensor modality.

1 Introduction

Simulators play an important role in prototyping, debugging and benchmarking new advances in robotics. One aspect that has proven so far to be difficult to simulate is tactile sensing, and in particular vision-based tactile sensors [Yuan et al., 2017, Padmanabha et al., 2020, Lambeta et al., 2020] which provide rich high-resolution measurements. This is because to accurately model this family of tactile sensors it is necessary not only to model the dynamics of the contact, but also to model the optical properties of the sensors and the corresponding illumination to obtain realistic

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

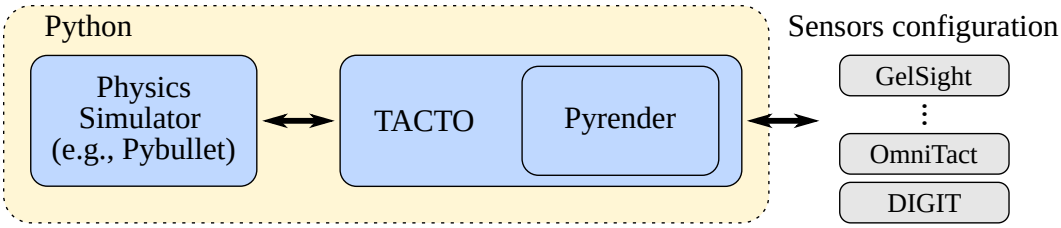

Figure 2: Software Architecture. TACTO bridges between physics simulator and back-end rendering engine, and can be configured to model different sensor designs through configuration files.

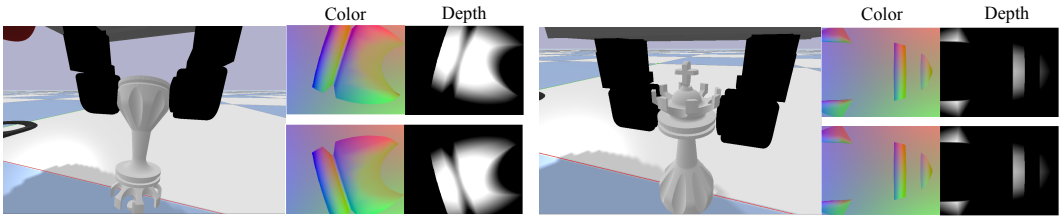

Figure 3: Example images of simulated DIGIT imprints. TACTO is able to generate color and depth images at the same time with details of the local geometry at high speed.

perceptual outputs. All of this while keeping the simulator flexible enough to implement various sensors with different form factors, and fast enough to be of practical use.

To fill this lack of touch sensing simulators, we introduce TACTO – a simulator of vision-based tactile sensors explicitly designed to be fast and flexible. Fig. 1 shows examples of TACTO in different scenarios.

We believe that TACTO can be of practical value for different communities: 1) To hardware designers, it provides a preliminary way of to simulate and evaluate their design choices for future sensors; 2) To the robotic community, it provides a way to simulate and study the integration of touch sensing into control scheme; 3) To the machine learning community, it can provide an easy-to-use tool to generate multi-modal inputs which would otherwise require real-world hardware.

2 Related Work

While there is a large number of physics engines available to robotic practitioners [Erez et al., 2015], the choice when simulating tactile sensors is more limited. This is mostly due to the difficulty of accurately and efficiently simulating touch.

Several traditional low-dimensional sensors have been simulated in the literature, including BioTac [Ruppel et al., 2018] and fabric-based tactile sensors [Melnik et al., 2019]. We aim to simulate high-resolution vision-based tactile sensors, which can possess millions of tactels. Exploiting the nature of high-resolution vision-based tactile sensors, it is possible to use ray-tracing models from computer graphics to render sensor output. Recent simulators for vision-based tactile sensors based on Unity [Ding et al., 2020], Gazebo [Gomes et al., 2019] and finite elements models [Sferrazza et al., 2020] are not openly available (e.g., open-source) to the community. With TACTO, we aim to provide an open-source simulator that is fast, flexible, and can be integrated with several physics engines for experimenting with different learning and control algorithms with touch modality.

3 Our Approach

3.1 Overview of the Software Architecture

Fig. 2 shows the overall software architecture of TACTO. TACTO leverages Pyrender [Matl], a python graphics library using OpenGL[Shreiner et al., 2013], to apply various ray-tracing techniques with minimal efforts. TACTO takes the responsibility of bridging between the physics simulator and the back-end rendering engine. To avoid I/O bottleneck of loading depth into renderer in real time,

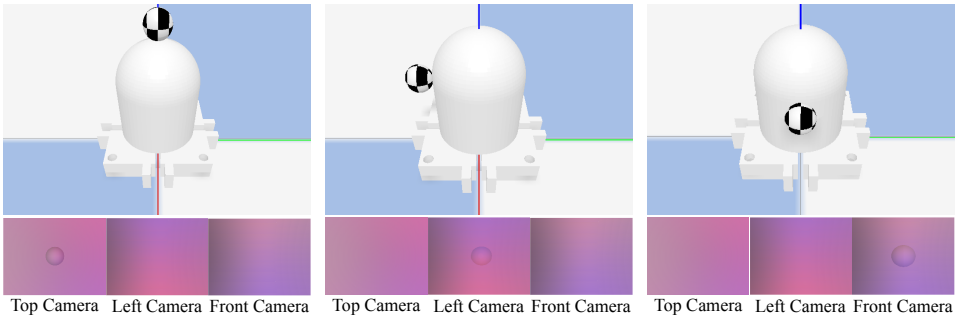

Figure 4: Example images of a simulated OmniTact [Padmanabha et al., 2020] touching a sphere. We show only 3 of the 5 cameras mounted on the sensor. It is visible how the specific light/camera placement in the OmniTact results in more pinkish images also in our simulator.

TACTO pre-loads the object meshes in the renderer and synchronizes the poses of objects and sensors during physics simulation.

TACTO includes three major phases (for simplicity, we here assume PyBullet as the underlying physics engine). 1) Initialize: TACTO loads the sensor configuration, and setup up the sensor (camera, lights, gel mesh) in the rendering engine. 2) Create scene: PyBullet loads object URDF into the scene, and TACTO parses the URDF (by urdfpy package in our case), and add the analyzed mesh into the rendering engine. 3) Step simulation: PyBullet first calculates physics simulation. Then TACTO loads the poses of each link from PyBullet, synchronizes the poses of objects and sensors in the rendering engine, and fetches the rendered tactile imprints.

3.2 Salient Features

Fast: TACTO is very fast for being a tactile sensor simulator. Table 1 shows the speed of TACTO. When interacting with an object mesh with 12K faces, TACTO is able to render a single DIGIT sensor at 200 frames per second on GPU with output resolution of 160x120 pixels. We optimized the TACTO so that only the number of objects in contact influences Pyrender's speed, making it maintain a high speed even in a cluttered environment.

	Resolution	# contact / # objets		
		1 / 1	1 / 100	10 / 100
GPU	160×120	220	200	80
	320×240	140	100	60
	640×480	90	90	50
CPU	160×120	60	60	40
	320×240	30	30	30
	640×480	10	10	10

Table 1: Frames per second rendered by TACTO (excluding physics simulation time) when simulating a DIGIT sensor with multiple objects (each mesh with 12K faces) in the scene. Appendix 6.5 shows breakdown time and machine specifications.

Flexible: TACTO can adapt to different sensor designs by changing configuration files easily. Besides rendering DIGIT [Lambeta et al., 2020] signals, as shown in Fig. 3, we also demonstrate the possibility to render OmniTact [Padmanabha et al., 2020] signals, as shown in Fig. 4, which has a substantially more complicated sensor structure including round surface, 5 cameras, and 11 light sources.

Force dependent: TACTO can simulate the dynamics range of the gel by mapping the force measured in the physics engine to different deformation of the gel. This means that light forces will yield to less deformations, and higher forces to more deformation of the gel. The forces are generated from rigid-body contact models currently for stability. It is also possible to set an lower threshold before forces results in deformations and a upper saturation limit over which increases in force will not results in more deformation.

See appendix for more features, including **Calibration from real sensors** (Sec. 6.1), and **Rendering from depth** (Sec. 6.2).

4 Results

We demonstrate TACTO on a perception task and a control task: 1) learning grasp stability from vision and touch readings. See results in Fig. 5, and details in Appendix 6.8. 2) learning to roll a marble to a target location in the sensor. See results in Fig. 6, and details in the Appendix 6.9.

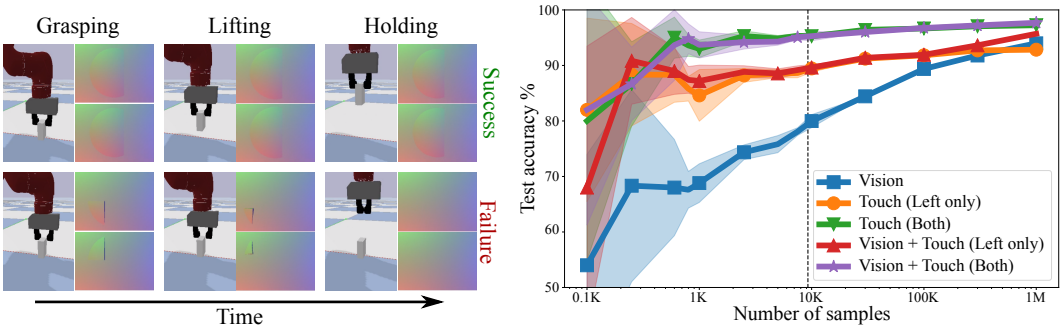

Figure 5: Learning Grasp Stability. (*Left*) Examples of a successful grasp and a failure grasp. In the failure grasp, the object is only grasped by the corner and begins to slip after being lifted. (*Right*) Median and 68% percentile of the learned models when varying the number of data used. We compare using only vision, only touch and both vision and touch as inputs of the models. Results show that learning grasp stability from touch needs significantly less amount of data to achieve relative high accuracy compared to vision, and that increasing the amount of data helps to improve performance. The vertical dashed line shows the dataset size of previous work on real robot [Calandra et al., 2017]. In simulation, we are able to experiment with a dataset more than two orders of magnitude larger.

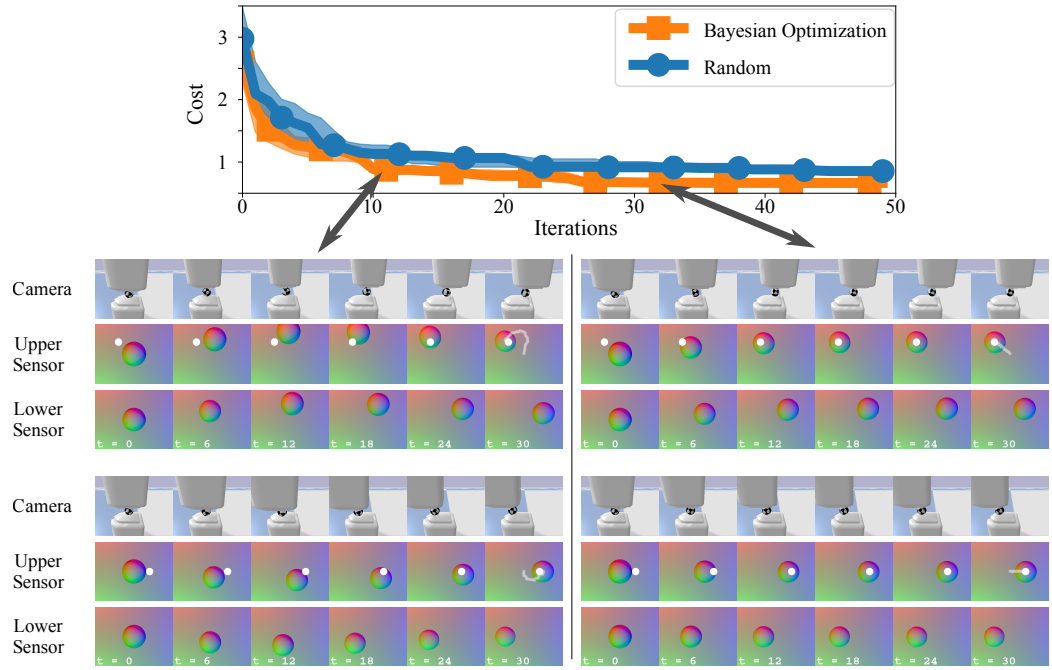

Figure 6: Learning in-hand marble manipulation. (*Upper*) The learning curve (median and 68th percentile over 30 experiments) for Bayesian optimization and random search. (*Lower*) Examples of marble rolling into different target locations within the fingers at early and later learning stage. At the later stages of learning, the controller is able to roll the marble between fingers with faster speed and higher accuracy, while avoid dropping the marble.

5 Conclusion

In this paper we introduce TACTO, a simulator for vision-based tactile sensors. TACTO is designed to provide an easy-to-use, fast and flexible simulator capable of generating realistic high-resolution readings. We demonstrate and validate TACTO on a perceptual task for learning grasp stability, and a control task for marble manipulation. To foster the tactile sensing community and to enable robotics and machine learning researchers to make use of touch in their simulations, we open-source TACTO. Future work will focus on improving the modeling of the effects of forces through the deformation of the elastomers, and ultimately on generating more realistic readings.

References

I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

R. Calandra, A. Owens, M. Upadhyaya, W. Yuan, J. Lin, E. H. Adelson, and S. Levine. The feeling of success: Does touch sensing help predict grasp outcomes? *Conference on Robot Learning (CORL)*, pages 314–323, 2017.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

Z. Ding, N. F. Lepora, and E. Johns. Sim-to-real transfer for optical tactile sensing. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

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 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1927–1934, 2018.

T. Erez, Y. Tassa, and E. Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404, 2015.

J. Geukes, M. Nakatenus, and R. Calandra. Gazebo plugin for the icub skin. `https://github.com/robertocalandra/icub-gazebo-skin`, 2017.

D. F. Gomes, A. Wilson, and S. Luo. Gelsight simulation for sim2real learning. *ICRA Workshop on ViTac: Integrating Vision and Touch for Multimodal and Cross-modal Perception*, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998, 2018.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

M. Lambeta, P.-W. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, D. Jayaraman, and R. Calandra. DIGIT: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters (RA-L)*, 5(3):3838–3845, 2020. doi: 10.1109/LRA.2020.2977257.

M. Matl. Pyrender. `https://github.com/mmatl/pyrender`.

A. Melnik, L. Lach, M. Plappert, T. Korthals, R. Haschke, and H. Ritter. Tactile sensing and deep reinforcement learning for in-hand manipulation tasks. In *IROS Workshop on Autonomous Object Manipulation*, 2019.

A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112, 2020.

A. Padmanabha, F. Ebert, S. Tian, R. Calandra, C. Finn, and S. Levine. Omnitact: A multi-directional high-resolution touch sensor. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

B. Romero, F. Veiga, and E. Adelson. Soft, round, high resolution tactile fingertip sensors for dexterous robotic manipulation. *arXiv preprint arXiv:2005.09068*, 2020.

P. Ruppel, Y. Jonetzko, M. Görner, N. Hendrich, and J. Zhang. Simulation of the syntouch biotac sensor. In *International Conference on Intelligent Autonomous Systems*, pages 374–387. Springer, 2018.

C. Sferrazza, T. Bi, and R. D'Andrea. Learning the sense of touch in simulation: a sim-to-real strategy for vision-based tactile sensing. *arXiv preprint arXiv:2003.02640*, 2020.

D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane. *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley, 2013.

A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2107–2116, 2017.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

S. Tian, F. Ebert, D. Jayaraman, M. Mudigonda, C. Finn, R. Calandra, and S. Levine. Manipulation by feel: Touch-based control with deep predictive models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 818–824, 2019.

W. Yuan, S. Dong, and E. H. Adelson. Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors*, 2017. doi: 10.3390/s17122762.

6 Appendix

6.1 Calibration from real sensors

The simulated images can be fine-tuned, as shown in Fig. 7, to be closer to real signals. Fig. 8 shows some comparison between fine-tuned simulated and real imprints. Furthermore, TACTO supports rendering shadows to match real signals, which is difficult to implement from scratch with the basic Phong's reflection model [Gomes et al., 2019] at high speed. Still, the major focus of TACTO at the current stage is to provide fast tactile simulation for testing different learning and control algorithms. The learned knowledge in simulation can inspire real-world applications. The simulated imprints still have distance from real signals, and we hope more calibrations and refinement like generative models [Hoffman et al., 2018, Shrivastava et al., 2017] can be applied in the future to make it more realistic.

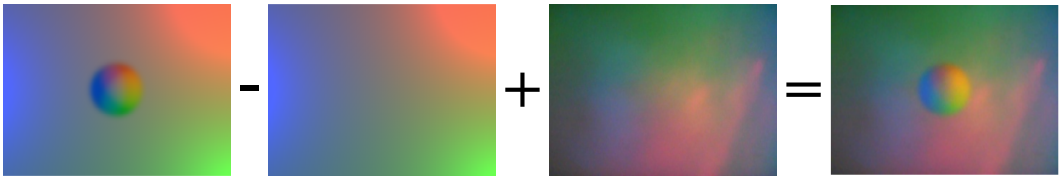

Figure 7: If readings from a real-world sensor are available, TACTO allows to fine-tune the simulator using the real-world data. This is achieved by calculating the pixel-wise difference of the simulated images with and without touch, and then adding the reference real-world image.

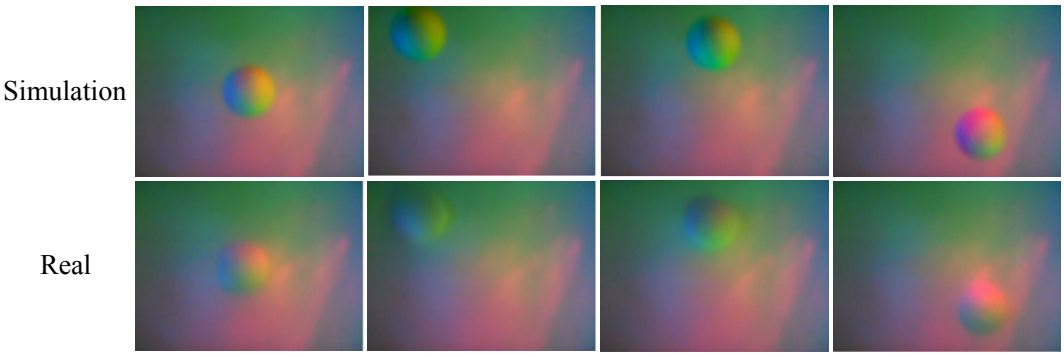

Figure 8: Examples of fine-tuned simulated reading compared to real-world readings. The real-world readings are collected from a DIGIT sensor [Lambeta et al., 2020] touching a ball of $5.3\,\mathrm{mm}$ diameter.

Figure 9: TACTO supports rendering shadows to obtain more realistic simulations. The real-world measurement is collected from a DIGIT sensor touching a ball of $3.7\,\mathrm{mm}$ diameter.

Fig. 10 shows more comparison results between simulated and real tactile readings. We press different shapes (spheres, corners, edges, planes) with different forces on the sensor. TACTO generates realistic tactile signals for both contact surfaces and shadows. We smooth the cube's mesh to better model the deformations on the contact boundaries, during pre-processing. A 15×15 Gaussian filter is applied on the rendered RGB image to make the rendering soft. Note that it takes only 2 extra milliseconds for rendering shadows (with meshs of 19k faces, output resolution of 640x480, GPU).

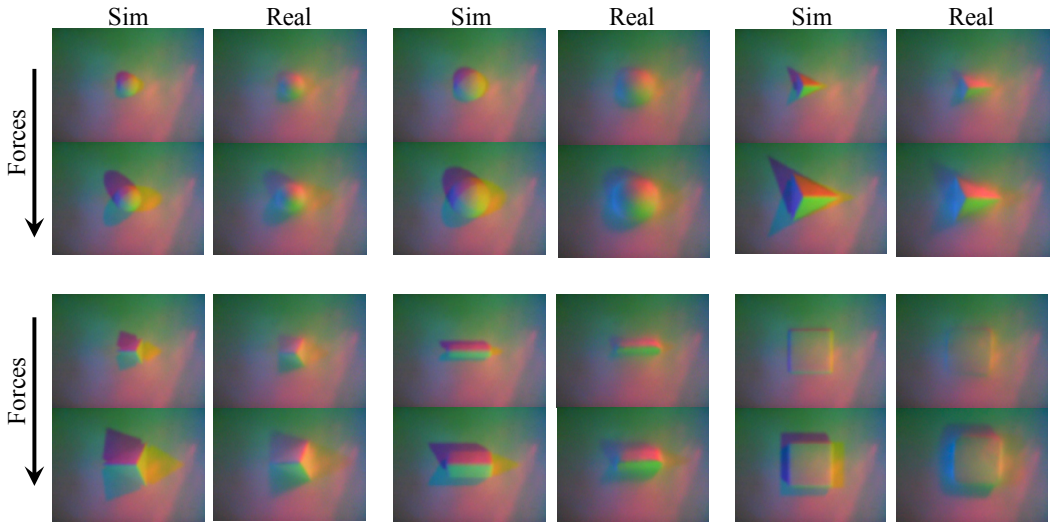

Figure 10: Comparison of simulated and real tactile readings. Real readings are collected from a DIGIT sensor touching objects with different shapes (spheres, corners, edges, planes), with different forces. TACTO generates realistic tactile signals for both contact surfaces and shadows. The simulated images are overlaid on a background real sensor image.

6.2 Rendering from depth

TACTO also supports rendering tactile imprints from depth image as an option. Given the depth image, it generates corresponding meshes to replace the original gel surface in the rendering engine. As mentioned in Section 6.4, the speed is limited due to I/O bottleneck, however, it can be helpful in some cases where it requires modification on depth images before rendering for more realistic tactile imprints.

6.3 Design Desiderata

High-throughput: for any simulator it paramount to be as fast as possible to reduce the real-world time of running simulations. Reproducing touch sensing, even for low-dimensional sensors, has traditionally been a very computationally intensive operation often leading to simulations barely faster than real-time [Geukes et al., 2017]. Obtaining a simulator that could perform hundreds of frames per second was, from the beginning, one of our most important design desiderata.

Flexible: since there are different sensor designs of vision-based tactile sensors, where some of them have complex geometry [Padmanabha et al., 2020], mirrors [Donlon et al., 2018], and transparent case for light piping [Romero et al., 2020], it is desirable for any simulator to be flexible and powerful enough to support a large choice of optical components, and mechanical designs.

Realistic: it is desirable that the simulator produces outputs as close as possible to real measurements, including illumination of the contact region, global lighting distribution, and details like shadows and deformation on the contact boundary.

Easy to Use: finally, it is from a practical point of view very important that the simulator is easy to install, use, modify and set up for different perception and control tasks.

6.4 Architectural Choices

Here we compare several architectural choices. The simulator needs to calculate the local contact geometry (Depth), and the corresponding rendering (RGB) that best meets the desiderata.

1. **Creating our own custom renderer:** PyBullet built-in camera can provide a depth map of the contact area. To render the RGB image from the depth map, researchers from [Gomes et al., 2019] implemented their own renderer based on the Phong's reflection model. It generated promising results and should be easy to use. However, it assumed that light only bounce once,

directly from gel surface to the camera. Hence, it is difficult to adapt to existing and future sensor designs that require advanced functionalities, like reflection, refraction, and shadows with fast speed. Although it can be extended to support more ray-tracing functionalities, this may require non-trivial engineering time to re-implement methods with GPU-acceleration which are already provided and tested by open-source software from computer graphics communities.

2. **Using OpenGL only for RGB:** Alternatively, we can leverage the power of OpenGL [Shreiner et al., 2013]. Besides the features in [Gomes et al., 2019], OpenGL also supports mirrors, transparent objects, shadow, GPU-acceleration, etc, which opens up the possibility for rendering advanced sensor design at high speed, with minimal effort. To use the power of OpenGL, one can get the depth image from PyBullet first, and pass the depth map to OpenGL for rendering. The rendering alone is fast and can be sped up in GPU, however, I/O speed becomes the bottleneck. In preliminary experiments, a significant amount of time was spent on loading the mesh generated by depth map into OpenGL, and this limited the overall speed of this method to only 20 frames per second, even on GPU.
3. **Using OpenGL for both Depth and RGB:** Our proposed system design directly renders both depth and RGB images in OpenGL to achieve high speed with powerful rendering functionalities. To avoid I/O bottleneck, it makes use of an observation that the contact surface is not arbitrary: it always consists of the gel surface and part of objects' surface. Although loading a mesh is slow, it is very fast to change their poses in OpenGL after preloading the gel and object meshes. So TACTO, duplicates the scene from the physics engine (e.g., PyBullet) into our OpenGL renderer. At every time step, instead of loading the mesh from depth repeatedly, it only needs to update the pose of each object. In this way, the system is able to render at very high speed (up to 200 frames per second in our experiments). One limitation of this method is that it is difficult to add deformation on the contact boundary, because the RGB and depth images are calculated at the same time. However, it can potentially be solved by adding data augmentation [Perez and Wang, 2017] or adding refinement afterward with generative models to make it more realistic [Shrivastava et al., 2017, Hoffman et al., 2018]. Overall, we think it is worthwhile to trade this for speed and flexibility. For implementation, we use Pyrender [Matl] as renderer in TACTO since it provides a light-weight python interface for deploying OpenGL with GPU support, making TACTO not only powerful, but also easy to use.

Based on extensive experiments and analysis of each method, we decided to use this third option. This results in TACTO being closely aligned to our ideal simulator, which is fast, flexible, and powerful. In Section 4, we will demonstrate that TACTO is also easy to set up for perception and control tasks.

6.5 Breakdown speed

Table 2 shows the breakdown speed of TACTO in conjunction with PyBullet. The physics simulation speed usually depends on the scenario. In our experiments, for 100 objects, we put 10×10 array of objects in the air, and central objects will contact the sensor after falling down. Note that the physics simulation can run asynchronously to speed up.

	Res	1 obj (1 in contact)			100 objs (1 in contact)			100 objs (10 in contact)		
		Step	Sync	Render	Step	Sync	Render	Step	Sync	Render
GPU	160×120	0.2	2.2	2.5	5.8	2.4	2.8	11.6	6.3	6.9
	320×240	0.2	2.2	5.0	5.8	2.5	7.6	11.5	6.3	9.5
	640×480	0.2	2.3	9.4	6.1	2.6	8.6	11.7	6.3	15.0
CPU	160×120	0.4	4.2	11.8	9.8	4.5	11.3	19.5	10.7	16.7
	320×240	0.4	4.2	25.8	9.6	4.6	26.1	18.0	9.8	26.2
	640×480	0.4	4.5	78.3	9.0	4.8	78.5	17.2	10.4	80.1

Table 2: Breakdown speed (in millisecond) for TACTO with PyBullet when simulating a DIGIT sensor with multiple objects (each mesh with 12K faces) in the scene. Step: PyBullet simulates physics; Sync: TACTO synchronizes the scene; Render: Pyrender renders the scene. Note that pybullet.stepSimulation can run asynchronously to speed up. CPU machine: Intel Core i7-6820HQ. GPU machine: Nvidia RTX 2080 Super GPU with Intel Core i9-9900K CPU.

6.6 Sensor configuration files

For the configuration file details, it currently requires parameters of the gel (pose, mesh), a list of camera(s) (pose, field of view, clipping plane), a list of lights (pose, color, intensity) and user-specific details like noise level, and the mapping from force to deformation. The example configuration files in the repository include the explanations for each parameter. The configuration file and renderer can be further extended to support more functionality of OpenGL/Pyrender and sensors.

6.7 Extension to different physics engines

We use an open-source physics engine, PyBullet, to demonstrate the framework. But TACTO can also support other physics engines. Specifically, there are two ways: 1) is to synchronize the scene with the functions provided by the selected physics engine. The core required functions include getting objects/links poses to synchronize the object pose and sensor pose, and getting contact forces to simulate deformation. 2) is rendering from depth as described in Sec. 3.2. Provided the depth information, TACTO can generate the mesh in the scene and render corresponding images.

6.8 Learning grasp stability

Setting Our setup consists of two DIGIT sensors [Lambeta et al., 2020] mounted on a WSG-50 parallel jaw gripper, and an external camera. We collected 1 million grasps of a box-shape object in a self-supervised manner by randomizing the position, orientation and force applied by the gripper. The ground-truth for each grasp was labeled depending on whether the object was still between the fingers after being lifted. Some examples are shown in Fig. 5.

Training procedure

Using the collected dataset, we trained ResNet-18 [He et al., 2016] neural network models that, given the raw vision and touch signals, would predict whether the grasp is stable or not after lift-off. The training procedure followed previous work [Calandra et al., 2017]. For details, to fuse vision and touch signals, the feature vectors produced by ResNet-18 were concatenated and fed to two fully connected layers, with 512 and 256 hidden units, to predict final results. To speed up training, we used ResNet-18 model pre-trained on ImageNet [Deng et al., 2009]. And we used vision and touch images with 160×120 pixels. The images were resized to 256×256 and random cropped to 224×224 for data augmentation.

To evaluate the performance of different dataset sizes, we used K-fold cross-validation and computed the median and 68% percentile of the classification accuracy. Due to computational limits we used $K = 10$ up to 1000 datapoints, $k = 5$ up to 10k datapoints, and above 10k datapoints we evaluated a single train/test split $80/20\%$. We trained 10 epochs for each dataset size using Adam optimizer [Kingma and Ba, 2014] with a learning rate of $5e - 4$ and batch size of 32.

Results The results shown in Fig. 5 suggests: (1) the model learned fast from touch with little amount of data, while vision requires 3 or 4 orders of magnitude more data to catch up; (2) single tactile sensor worked significantly worse than two tactile sensors, because the object may look stable from one tactile sensor while it unstably contact the other side; (3) on the low-data regime, the result agrees with previous real world experiments[Calandra et al., 2017]: combining vision and touch worked best in most of the cases. Although touch-only achieved comparable results to the combined model, we think the difference can increase with larger object set; on the high-data regime, we can evaluate with 2 orders of magnitude more data in simulation compared to [Calandra et al., 2017], and observe the trends that all the models keep improving still, and vision's potentials with more data.

6.9 Learning in-hand marble manipulation

The setup includes two DIGIT sensors as shown in Fig. 6. The lower sensor is fixed, while the upper sensor is controlled to roll the marble.

Setting We use position control with a maximal force for controlling the upper sensor. Specifically, we control the horizontal position of the upper sensor, and push the sensor vertically with maximal force to keep the marble in hand while rolling. We parameterize the controller as $\mathbf{u} = \mathbf{K}\bar{\mathbf{x}}$, where $\mathbf{u} \in \mathbb{R}^2$ is the desired velocity of the upper sensor in horizontal plane, $\bar{\mathbf{x}} \in \mathbb{R}^2$ is the error state between the current marble center location $\mathbf{x}$ and the goal location $\mathbf{x}^*$ in tactile space, and $\mathbf{K} \in \mathbb{R}^{2\times 2}$

is the parameter to learn. The cost is defined as cumulative error distance in tactile space $\sum_t \|\bar{\mathbf{x}}_\mathbf{t}\|$, and we set eight different target locations and take the average cost for robustness. We apply Bayesian optimization [Snoek et al., 2012] with upper confidence bound to optimize the parameter $\mathbf{K}$ automatically. Our main purpose here is to validate the simulation system, and provides benchmark experiments, however, the controller can be replaced by model predictive control and/or reinforcement learning to manipulate more complex objects [Tian et al., 2019, Akkaya et al., 2019] and for dexterous hand Lambeta et al. [2020], Nagabandi et al. [2020].

Results Fig. 6 shows the quantitative and qualitative results of rolling a marble into desired locations. The system is able to learn to roll the marble into different target locations with few iterations and roll faster with more iterations. During the experiments, we validate that both PyBullet and TACTO run as expected without abnormal situations. Because the simulation is fast, it only takes 8 minutes for Bayesian optimization to learn marble manipulation with 50 iterations. It includes 6 minutes for optimizing the acquisition function, and 2 minutes for simulation, where there are 50 iterations, and each iteration includes 50 steps for rolling into each of 8 directions, rendering 20,000 tactile imprints of 160×120 resolution in total.