Optical Tactile Sim-to-Real Policy Transfer via Real-to-Sim Tactile Image Translation

Anonymous Author(s)
Affiliation
Address
email

Abstract: Simulation has recently become key for deep reinforcement learning to safely and efficiently acquire general and complex control policies from visual and proprioceptive inputs. Tactile information is not usually considered despite its direct relation to environment interaction. In this work, we present a suite of simulated environments tailored towards tactile robotics and reinforcement learning. A simple and fast method of simulating optical tactile sensors is provided, where high-resolution contact geometry is represented as depth images. Proximal Policy Optimisation (PPO) is used to learn successful policies across all considered tasks. A data-driven approach enables translation of the current state of a real tactile sensor to corresponding simulated depth images. This policy is implemented within a real-time control loop on a physical robot to demonstrate zero-shot sim-to-real policy transfer on several physically-interactive tasks requiring a sense of touch. Video results are available at *https://sites.google.com/view/tactile-sim-to-real*.

1 Introduction

Learning algorithms have an innate appeal for robotics to enable general, complex behaviours that would be difficult to achieve with classic control methods. To facilitate robot learning, large-scale data is often required, and physics-based simulation has a vital role to play for data collection. A common approach in learning based robotics is to simulate data that would be impractical to collect in the real world, learn control policies from this data, and then transfer learned skills to the physical system. Simulation also offers advantages such as avoiding damage during exploratory training, exploiting privileged information from the simulator, and use of open-sourcing for dissemination of research findings. However, physics engines necessarily approximate the real world to reduce computational costs, giving a 'sim-to-real gap' that impairs the performance of these policies applied to reality.

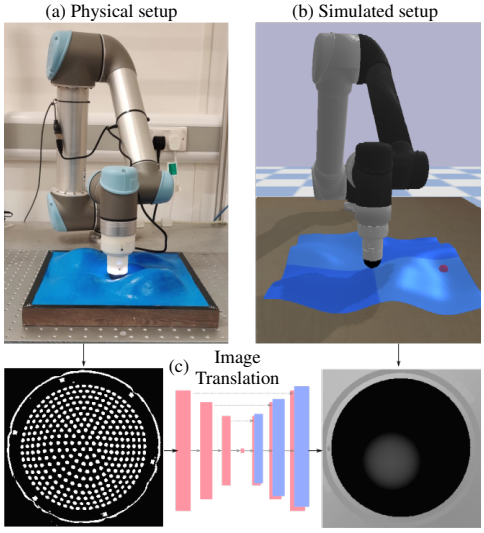

Figure 1: Sim-to-Real system used for policy transfer. (a) Real UR5 robot arm with marker-based optical tactile sensor. (b) Simulated UR5 with depth-based optical tactile sensor. RL policies are trained in simulation and policy transfer is enabled by Real-to-Sim image translation.

Reinforcement Learning (RL) robotics research is dominated by the use of proprioception and vision as the sensory inputs into the system. Whilst this has led to complex behaviours [1, 2, 3], other important sources of information have been underutilised. Specifically, humans use the sense of touch to accomplish complex manual tasks, utilising a granularity of detail unavailable to our other senses. Tactile data has several advantages as the main source of information for learning: it is less likely to be obscured, which is particularly useful in fine-grained manipulation tasks; information can be captured at finer detail than camera images of an entire scene; and the observation space is constrained more than external cameras, which simplifies the translation of real to simulated images and, likewise, from simulated to real policies.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

Here we make the following contributions aimed at bringing together reinforcement learning and tactile robotics: **1)** We introduce a fast method of simulating a high-resolution optical tactile sensor using depth imaging and approximation with rigid body dynamics. **2)** We provide an open-source suite of fast and parallelizable RL environments tailored to tactile robotics. **3)** We show that an RL method, Proximal Policy Optimisation (PPO) [4], acquires successful policies for all of these tasks. **4)** Finally, we demonstrate and validate a data-driven approach for zero-shot sim-to-real policy transfer via image translation between real and simulated tactile images on these tasks.

2 Related Work

The tools required to build this tactile simulation are available in most physics engines. Here we choose PyBullet [5], predominantly due to being open-source and having demonstrated sim-to-real success in robotics [6]. We provide further detail on this choice in Appendix A.

Tactile Simulators: A variety of simulation methods have been considered, making the common assumption that detailed simulations of the contact physics are required.

Contact detection is the simplest form of tactile data. It has the advantage of utilising the contact locations, contact forces and frictional forces available in most physics engines, giving computational efficiency at the cost of low-resolution data. Nonetheless, Melnik et al. [7] demonstrated in simulation that contact detection can improve the sample efficiency and performance of learning in-hand manipulation skills. Another method of modelling tactile sensors uses spring-mass-damper systems where components of skin are modelled as individual rigid bodies [8]; however, the increased computational cost has limited the scope to low-resolution tactile sensors. A more efficient approach considers a simulated sensor under elastic deformations [9, 10], but has not been used with deep RL.

Finite element simulations can model both high-resolution data and accurate contact dynamics, used for a shear-based optical tactile sensor [11] and the Syntouch Biotac [12]. This approach also has high computational cost but has been combined with deep RL to learn a difficult control task [13]. For computational efficiency this required a custom simulator with task-specific approximations.

Optical tactile sensors are compatible with image-rendering techniques for simulating high-resolution contact geometry. In [14], depth images are used to capture a heightmap of the elastic skin of a Gelsight sensor, followed by an illumination stage to make simulated images more like those from the real sensor. [15] extended this illumination stage using the rendering software OpenGL. Although illumination and rendering helps narrow the sim-to-real gap, there is still a gap remaining as is evident in the need for texture augmentation when evaluating on a real robot [14].

Our approach differs in that we do not attempt to bring the simulated tactile image closer to the real world, but instead use a data-driven *domain adaption* approach to map from the information-rich real world to the simplified simulated sensor. In consequence, we find a reduction of computation and complexity when simulating tactile sensors along with a reduced sim-to-real gap. The proposed methods work with a version of the TacTip optical tactile sensor [16] that was used here to demonstrate the approach, and we expect it to apply also to the Soft-Bubble sensor [17] and other camera-based optical tactile sensors [11, 18].

Sim-to-Real Transfer: For computational efficiency, physics engines necessarily approximate the real-world dynamics, leading to a sim-to-real gap where dynamics and visuals differ between simulation and reality. This gap can make it difficult to transfer skills learned in simulation to the real world. Several methods have been proposed to mitigate this issue, namely *Domain Randomisation*, *Network Distillation* and *Domain Adaption* [3].

In this work we mainly use *Domain Adaption*, following James et al. [6] to train an image-conditioned generator network that translates between real and simulated images. When using vision, *Domain Randomisation* was necessary on simulated images to train a generator robust enough to generalise to real visual images. Here we exploit the more constrained image space of optical tactile sensors to train generator networks using a more limited dataset. We perform some randomisations to make simulated tasks more difficult and RL policies more robust, aiming to keep within realistic conditions. Although privileged information from the simulator is used to construct a reward for improved learning, this information is not part of the observation in any task. As we perform zero-shot policy transfer from sim-to-real, *Network Distillation* is not required.

3 Tactile Simulation

In this work, we utilise PyBullet's synthetic camera rendering to capture depth images within a virtual optical tactile sensor, based on the CAD files used to 3D print a real TacTip (see e.g. [16, Fig. 4]). When gathering tactile images, we take the difference between the current depth image and a reference depth image taken from when the sensor is not in contact. This difference produces a penetration depth map that generalises to arbitrary sensor shapes. Noise is removed from the image by zeroing values below a set tolerance of 10^{-4}, followed by a re-scaling from $[0, \mathrm{max_penetration}]$ to $[0, 255]$. An artificial border is also overlaid onto the tactile image to bring it closer to real tactile images and to provide a reference point that transforms with augmentations.

In order to achieve the computational efficiency necessary to generate data at large scales, we approximate the soft tip of the tactile sensor with rigid body physics simulation. We limit the contact stiffness and damping used during collision detection as this allows penetration of objects into the simulated sensor tip in an approximate manner to the deformation of the real tip.

We describe our simulation method in relation to the list of desiderata proposed by Wang et al. [15].

High Throughput: We use PyBullet's GPU rendering functionality to offer fast simulation. On a PC with an Nvidia 2080Ti, we can achieve up to 1000 fps when rendering single tactile images at 128×128-resolution. Multiple PyBullet physics engines can also run in parallel, so during training we used 10 vectorised environments to increase throughput.

Flexible: Whilst this study is predominantly based on the TacTip sensor, we do not attempt to simulate images accurate to any specific tactile sensor. Instead, we simulate only useful tactile features, relying on a later image-translation stage to map from real to simulated images. We expect similar results are possible with a broad range of other high-resolution optical tactile sensors, including sensors of the Gelsight type [19, 20, 21], the Soft-Bubble [17] and optical shear-based sensors [22].

Realistic: Instead of aiming to realistically simulate the physical properties of any specific sensor, which is both difficult and computationally expensive, we simulate only the desired properties of an idealised tactile sensor. Currently, only contact geometry is considered, but this could be extended to include shear force, texture, etc. Our data-driven approach helps bridge the sim-to-real gap, so there is no need to generate synthetic images that match images from a real sensor to high precision.

Ease of use: The simulation suite and the sim-to-real translation tools will be open-sourced. The simulation only requires the commonly-used PyBullet, so our approach should have a low barrier of entry. Our approach should also extend readily to different tasks and other tactile sensors.

4 Reinforcement Learning Environments

Each task is provided with a set of observation spaces to allow for verification of the environments, comparisons between tasks, and an examination of multi-modal visuotactile control. Four standard types of observation space are considered:

Oracle: Comprises state information from the simulator, which is difficult information to collect in the real world. We use this to give baseline performance for a task that is expected to act as an upper limit. The information in this state varies between environments but commonly includes tool center point (TCP) pose, TCP velocity, goal locations and the current state of the environment.

Tactile: Comprises images (128×128) retrieved from the simulated optical tactile sensor attached to the end effector of the robot arm (Figure 2 right). Where tactile information alone is not sufficient to solve a task, this observation can be extended with oracle information retrieved from the simulator. This can only include information that could be easily and accurately captured in the real world, such as the TCP pose that is available on industrial robotic arms and the goal pose.

Visual: Comprises RGB images (128×128) retrieved from a static, simulated camera viewing the environment (Figure 2 left). Only a single camera was used, although this could be extended to multiple cameras. As the simulated environment differs visually from the real-world environment, sim-to-real using RGB observations is challenging, requiring an approach like that of [6, 23].

Visual + Tactile: Combines the RGB visual and tactile image observations to into a 4-channel RGBT image. This case demonstrates a simple method of multi-modal sensing.

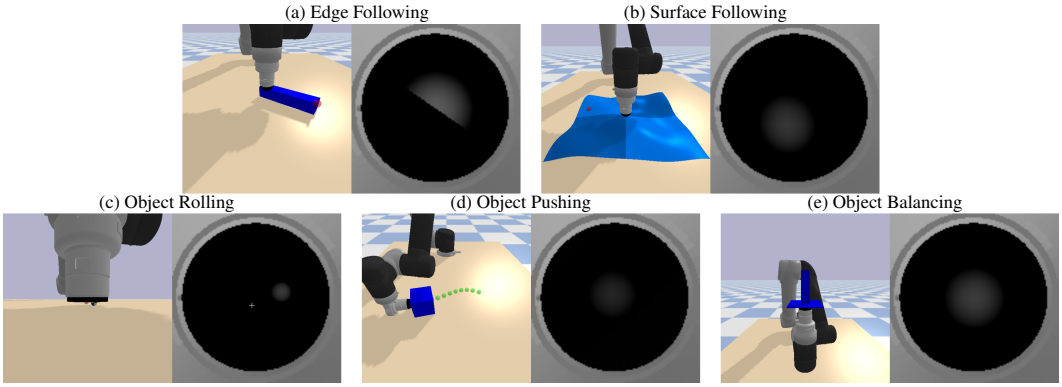

Figure 2: Tactile RL environments. (a) Traversing a randomly oriented edge while maintaining a set pose. (b) Traversing a randomly-generated 3D surface while maintaining a set penetration and normal orientation. (c) Manipulating a ball from a random initial position to a goal location. (d) Manipulating a cube along a randomly-generated trajectory. (e) Stabilising an object on the sensor tip.

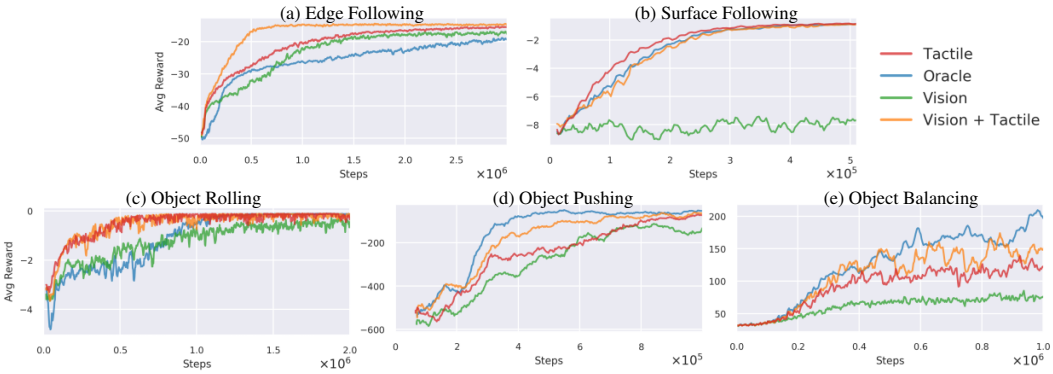

Figure 3: The DRL method PPO learns successful policies for the five considered RL environments. Panels show mean reward during training, smoothed with a window size of 50 and averaged over 3 seeds. Task reward commonly consists of negative distances between target and current poses, further details in Appendix C.

4.1 Tactile Exploration Environments

In this set of tasks, the robot interacts with a physical environment that stays static, with the objective to learn policies to safely explore physical areas. These policies can be used to report information about novel objects, simplify human-control methods and improve operational safety of the robots. Two exploration environments have been created, for edge and surface following. More complete details on these environments are given in Tables 5 and 6 of Appendix C.

Edge Following: The edge-following environment is used to train an agent that maintains its pose relative to an edge whilst traversing the edge towards a goal location. Previous work has demonstrated that robust 2D contour following can be achieved with supervised learning techniques [24]. Here we demonstrate this task can also be completed via sim-to-real reinforcement learning.

Surface Following: The surface-following environment is used to train an agent that maintains a set contact penetration of the sensor whilst orientating the TCP normal to an undulating 3D surface generated using OpenSimplex noise [25]. For ease of use, we automatically direct the sensor in the direction of the goal, analogous to the pose-based tactile servo control methods introduced in [26].

4.2 Non-prehensile Manipulation Environments

In this set of tasks, the objective is to learn policies that manipulate external objects in a desired manner. As our focus is on single sensors in this work, we consider non-prehensile manipulation. However, the simulation does support rendering of multiple tactile images, which could in principle be used simulate tactile grippers, manipulators or multiple robot arms in future work. Three manipulation tasks are proposed: object rolling, object pushing and object balancing. More details on the non-prehensile manipulation environments are given in Tables 7, 8 and 9 of Appendix C.

Object Rolling: The object-rolling environment requires the manipulation of small spherical objects into a goal position within the TCP coordinate frame. The agent must learn a policy to roll the object from a random starting position to a random goal position. In this environment, a flat tactile sensor tip is used to simplify the motion needed to maintain contact with a spherical object.

Object Pushing: In this task, the objective is to push an object to a goal location using the tactile image, as considered in previous work using supervised learning techniques [27]. Here we consider a cube pushed along a randomly-generated trajectory. The initial pose of the cube is randomised within limits and the trajectory generated using OpenSimplex noise.

Object Balancing: This task is analogous to the well-known 2D inverted pendulum problem [28], where an unstable pole with flat base is balanced on the tip of a sensor that points upwards. A random force perturbation is then applied to the object to cause instability. The objective is to learn a policy that applies planar actions to counteract the rotation of the balanced pole.

4.3 Reinforcement Learning Results

This work uses the Stable-Baselines-3 [29] implementation of PPO to train all tasks (further details in Appendix B). For an initial comparison, we train all tasks using all available observation spaces, averaged over 3 seeds. Training results are given in Figure 3, deterministic evaluation results and more detailed plots are given in Appendix D.

Oracle observations are used to verify that the environment leads to desired policies. As no image rendering is performed, this runs faster than tactile or visual observations. Even so, we find that in some cases tactile image observations lead to more sample-efficient and stable training when compared with oracle information, which is most clearly visible in the Edge-Following and Object-Rolling environments (Fig. 3a, 3c). In our view, this is likely to stem from less variation in the tactile images as each episode progresses, which could make the task easier to learn. For example, a tactile image taken when the sensor is positioned at the start, middle and end of the edge will look similar, whereas oracle information of the TCP pose will be different for each position.

Policies trained exclusively using visual observations tended to perform worst, which is likely due to most tasks being deliberately targeted towards tactile data. For example, details of the contact can be obscured when viewed from a single external camera. The best visual agent performance was found in the object pushing task, which is the coarsest manipulation challenge. Whilst the object rolling task appears to perform well with visual observations, there is a notable performance gap where visual polices do not accurately learn the desired behaviour. In alternative tasks, whilst there is likely to be a configuration that could lead to successful learning, this does demonstrate that other modalities can provide valuable information when training agents. When combining visual and tactile observations, successful learning can take place. Despite the additional complexity in the observations agents can learn well with notable improvements in sample efficiency for the Edge-Follow and Object-Push tasks (Fig. 3a, 3d).

5 Real-to-Sim Image Translation

An advantage of using tactile images as the main form of observation is that the image space is more simplistic (e.g. markers on a uniform background) compared with the variety of visual images from a scene. In particular, vision-based images are affected by features such as changing lighting conditions, shadows and texture that usually constitute a superfluous level of detail when solving a task. Including these visual features in simulation can make the learning more difficult because of the increased complexity of the observations and the additional computation to render these images; conversely, ignoring these visual features would make image translation more difficult because of the greater simulation and reality gap, as they are still prevalent in reality, necessitating techniques such as image randomisation or RL task-aware training [6, 23]. These complexities are

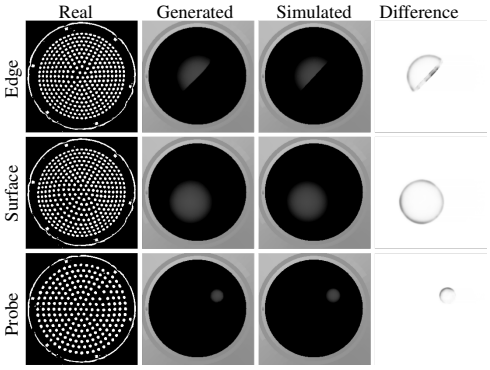

Figure 4: Image comparison between pairs of generated and simulated tactile images. Images are from validation sets for Real-to-Sim image translation. SSIM is used to create difference images.

not present for the real and simulated tactile images considered here because the camera is confined within the enclosure of the tactile sensor.

Conversely, a disadvantage of using tactile images as the main observations is that the tasks require the robot to be in close proximity, or touching, the environment. Whilst this contact may be necessary for some tasks, such as manipulation, it does make exploration in reality a challenge because of the self-inflicted damage that can occur. The approach proposed here is to use a separate data-collection stage, where the tactile sensor explores a series of configurations in a safe and controlled manner. As the tactile image space is relatively confined, an efficient exploration of a representative sample of configurations for a specific RL task becomes possible. Whilst some sensor configurations are not possible to sample in reality, such as large penetrations of the sensor that would cause damage, we aim to train a model that generalises to those unreachable configurations.

Specifically, in this work we treat the sim-to-real problem as a supervised image-to-image translation problem. The same data collection procedure is performed in both simulation and reality, producing a data set of real and simulated image pairs. Here we choose to transfer from real-to-sim images, because the real images are richer in information with details such as shear forces that we choose not to model in simulation. Generative Adversarial Networks (GANs) are the state of the art for realistic image generation. Here we use the pix2pix [30] architecture for image-to-image translation, which uses the U-net [31] architecture for the image conditioned generator and a standard convolutional network for the discriminator (Appendix E shows this architecture applied to tactile images).

5.1 Data Collection

Three data sets corresponding to distinct environments are collected for sim-to-real transfer, each with 5000 image pairs for training and 2000 pairs for validation. Following a previous investigation of contact-induced shear with the same optical tactile sensor [32], we deliberately induce shear perturbations by randomly sliding the sensor during data collection. Past work has shown this step is key to ensuring that the trained neural network outputs are insensitive to unavoidable motion-dependent shear during task performance (for more details, we refer to [26]). We do not model any sensor shear in simulation to ensure that the real-to-sim image generation is also insensitive to shear on the real sensor. During data collection, this shear perturbation is introduced by first moving the sensor to a randomly-selected prior location to establish contact with a stimulus, then moving from this pose to the target pose. Further details on data collection procedures are given in Appendix F.

5.2 Pix2Pix Training

The only change to the pix2pix architecture was to replace instance normalisation with spectral normalisation, with other default parameters sufficient for training on tactile images (full details in Appendix E). This change reduced droplet artefacts that were otherwise prevalent in our generated images. In addition, excluding the border from the simulated tactile images improved training, as otherwise the GAN focused on generating a realistic border instead of the tactile imprint. This is important because the imprint is the useful component when learning control policies. Instead, the border is re-added from a saved reference image after generation occurs.

Accurate tactile image generation is achieved with minimal difference between generated and target images (example images for each data set shown in Figure 4). For the edge, surface and probe datasets, mean SSIM scores across the full validation set are 0.9955, 0.9924 and 0.9942 respectively. The source of the small errors are highlighted in the shown SSIM image differences (Figure 4, right). The imprint borders in the generated images lack some sharp details generated in simulation, which will likely be due to a slight elastic stretching of the skin of the real tip that is not modelled in simulation. An approach such as Gaussian smoothing of the simulated images, as in [14], could reduce this effect, although we did not find it necessary to explore in the present work.

Crucially, the generator can interpret and generate image features useful to training RL policies. For example, edge orientation and position are accurately captured in the GAN. Also, the generator can generalise to images unseen during training. For example, all training data only had imprints from one source in each training image; however, during inference, multiple sources can be applied and the generator still produces realistic outputs (video demonstrations available *here*). Similarly, the generator appears to generalise outside the training data for penetration depth, which is important for helping to ensure that the sensor can avoid damage by not pressing too far into a surface.

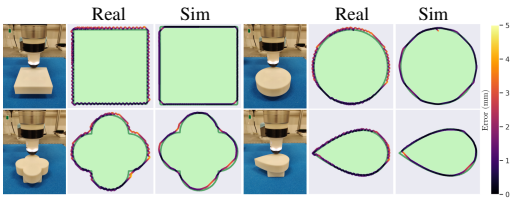

Figure 5: Comparison between sim and real performance for the Edge-Following RL environment. Using several flat shapes (circle, square, clover, foil) to evaluate the policy generalisation performance.

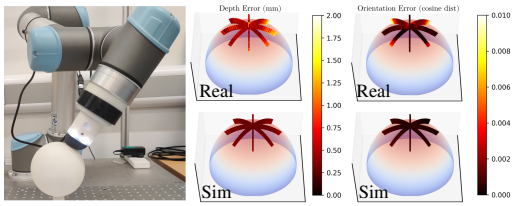

Figure 6: Comparison between sim and real performance for the Surface-Following RL environment. Positional error (left), Orientation error (right), Real data (top), simulated data (bottom).

6 Results: Sim-to-Real Policy Transfer

Successful sim-to-real policy transfer was achieved on 4 of the 5 proposed tasks. Qualitative demonstration videos are available at *https://sites.google.com/view/tactile-sim-to-real*

Edge Following: The learned edge-following behaviour is evaluated over several novel flat shapes in the environment (Figure 5). The learned policy successfully traverses all objects, despite not encountering them in training, including novel features such as curved edges and sharp internal or external corners. As these shapes are 3D printed, we have access to the ground truth in the CAD files (accurate to the precision of the 3D printer). We compare the trajectories taken in simulation to those in reality, reporting the distance from each point of the trajectory to the nearest ground truth node (Table 1). The results show that the sensor maintains close proximity to the target edge.

Table 1: Quantitative results for edge following task. Mean distance from ground truth shape taken over 1000 evaluation steps.

	Real	**Sim**
Square	1.47mm	0.63mm
Circle	1.50mm	0.80mm
Clover	1.58mm	1.38mm
Foil	1.09mm	0.47mm

The drop in performance on the physical robot seems related to an oscillating behaviour also present in simulation. The movements are exaggerated in reality due to increased latency between capturing an observation and predicting an action. A likely cause is the action-range clipping used in the PPO algorithm, which results in the action extremes being more prevalent. Approaches such as squashing functions [33], beta distributions [34] or constraint-based RL [35] could mitigate this artefact.

Surface Following: To evaluate the surface-following behaviour, a spherical object is traversed from its centre outwards in a set direction ranging over 360° in 45° intervals (Figure 6). As with the edge following task, we use a 3D-printed shape for which the ground truth is known from the CAD model. The sensor successfully and accurately traverses the object on the physical robot. Like the edge following task, there is a drop in accuracy between sim and real, although the evaluated policies still exhibited the desired behaviour. A more extensive qualitative test is in the video supplementary material for the complex undulating surface shown in Figure 1.

Table 2: Quantitative results for surface following task. Mean distance from ground truth evaluated over 1000 steps.

	Real	**Sim**
Depth Error	0.57mm	0.30mm
Cosine Error	0.00118	0.00054

Object Rolling: The object-rolling behaviour is tested by manipulating ball bearings (2-8mm diameter) from random initial positions to random goal locations. During the evaluation, the position of the ball bearing relative to the sensor is obscured, so the imprint of the object is tracked on the tactile image using basic computer vision (blob detection). The target is reached when the pixel distance is less than a 5-pixel threshold (approx 2mm). On the physical robot, this is performed for 25 trajectories with each bearing, resulting in 100 consecutive successful trials (subset shown in Figure 7). To compare with simulated results, we apply the same initial and target positions then perform the same pixel tracking. The sim and real results are similar, albeit with greater noise in the real trajectories.

Object Pushing: To evaluate this task we tracked the objects following the method given in [27]. Importantly Aruco markers were only used for obtaining quantitative results and not as part of the observation. This task was the most challenging to achieve accurate perfor-

Table 3: Quantitative results for object pushing task. Mean Euclidean distance from trajectory.

	Straight	**Curve**	**Sine**
Real Cube	11.5mm	11.7mm	13.6mm
Sim Cube	11.1mm	10.1mm	12.7mm
Real Tri	11.0mm	14.0mm	12.6mm
Sim Tri	21.8mm	10.2mm	11.0mm
Real Cyl	13.3mm	16.7mm	9.9mm
Sim Cyl	24.1mm	13.9mm	12.7mm

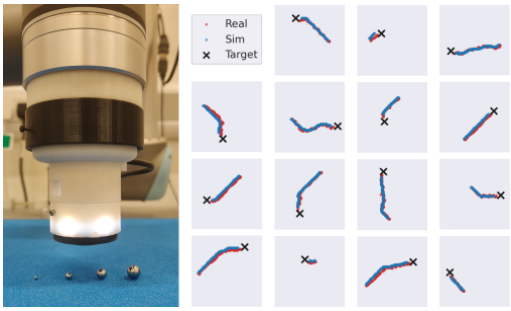

Figure 7: Real and simulated trajectories for policy evaluation of the object rolling task. A ball bearing is rolled an initial position to a goal location. Sizes range from 2-8mm diameter (columns: left to right).

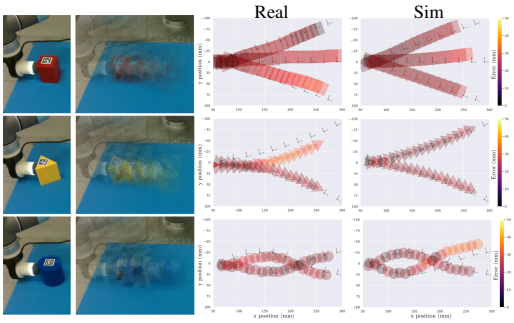

Figure 8: Evaluation of the pushing task. Trajectories show a cube (top), triangular prism (middle) and cylinder (bottom) pushed along straight, curved and sinusoidal trajectories respectively.

mance, likely due to the approximations of the simulated sensor tip, object being manipulated, and frictional interactions. Despite this, task success was achieved across several objects and trajectories when evaluated in reality (examples in Figure 8). Performance both in simulation and reality was sensitive to initial conditions, with compounding errors causing the object to veer off the trajectory. Future work could improve robustness to variations in the initial conditions by introducing additional randomisation in simulation and better matching the physical properties of the objects.

Object Balancing: The physical task remains an outstanding challenge, because the successful training in simulation required physics parameters outside the capabilities of our UR5 robot.

7 Discussion and Future Work

In this work, we demonstrated that zero-shot sim-to-real policy transfer is a viable approach for tactile-based RL agents. To learn the policy, we created a fast method for simulating tactile images based on contact geometry. From these images, distinct policies were learned for several physically-interactive tasks requiring a sense of touch. We demonstrated that a data-driven model for real-to-sim image translation can be embedded into the control loop for successful sim-to-real policy transfer.

There are several future directions for extending and improving this approach.

The tactile simulation of both the contact dynamics and captured information in the model could be improved. Our contact dynamics model used used rigid bodies with soft contacts as an approximation to the deformation of real tactile sensors. Extending this work with soft-body deformation should enable a more realistic simulation, which was not pursued in this initial study because of increased computational costs and instability issues with the simulators. The captured information in the present study focused exclusively on contact geometry, which in principle could be extended to include global shear information through contact data available in most physics simulators. Texture could be also included using rendering techniques common in photo-realistic image generation. Local shear of the sensor would be challenging to simulate, for example during incipient slip, due to the contact reduction commonly used in physics engines for computational efficiency; however, this capability may be possible in soft-body simulations where these local forces are required.

For translating real to simulated images, improvement could be made to both the GAN and training data. We used a conventional pix2pix [30] approach, which can be improved with extended methods [36]. In addition, a distinct data set was collected for training the GAN in each learning environment. Because of the constrained nature of the tactile image space, a general tactile data set could potentially be used train a single image-translation model for policy transfer over multiple tasks.

The environments considered here focused on skills achievable with only a single sensor and where the optimal behaviour is obvious for a human to interpret, as needed to verify that RL with a sim-to-real approach is a viable method. Future work could focus on tasks where ideal control policies are more complicated or unknown, so the RL framework can be fully exploited. An interesting topic would be to extend these methods to prehensile tasks such as grasping and dexterous manipulation, where tactile information will be valuable for learning desirable and robust policies. Although only one sensor was used in this current work, our tactile simulation does support multiple sensors, offering the opportunity to extend to more complex tasks involving hands with multiple tactile fingertips.

References

[1] 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:1910.07113*, 2019.

[2] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47), 2020.

[3] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *IJRR*, 2021.

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

[5] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. Available: http://pybullet.org, 2016–2019.

[6] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *CVPR*, pages 12627–12637, 2019.

[7] A. Melnik, L. Lach, M. Plappert, T. Korthals, R. Haschke, and H. Ritter. Tactile Sensing and Deep Reinforcement Learning for In-Hand Manipulation Tasks. *IROS Workshop on Autonomous Object Manipulation*, 2019.

[8] A. Habib, I. Ranatunga, K. Shook, and D. O. Popa. SkinSim: A simulation environment for multimodal robot skin. In *CASE*, pages 1226–1231. IEEE, 2014.

[9] Y. Wang, W. Huang, B. Fang, and F. Sun. Elastic interaction of particles for robotic tactile simulation. *arXiv:2011.11528*, 2020.

[10] Z. Ding, N. F. Lepora, and E. Johns. Sim-to-Real Transfer for Optical Tactile Sensing. *ICRA*, pages 1639–1645, 2020.

[11] C. Sferrazza, A. Wahlsten, C. Trueeb, and R. D'Andrea. Ground truth force distribution for learning-based tactile sensing: a finite element approach. *IEEE*, 7:173438–173449, 2019.

[12] Y. S. Narang, K. Van Wyk, A. Mousavian, and D. Fox. Interpreting and Predicting Tactile Signals via a Physics-Based and Data-Driven Framework. *arXiv:2006.03777*, 2020.

[13] T. Bi, C. Sferrazza, and R. D'Andrea. Zero-shot sim-to-real transfer of tactile control policies for aggressive swing-up manipulation. *arXiv:2101.02680*, 2021.

[14] D. F. Gomes, P. Paoletti, and S. Luo. Generation of gelsight tactile images for sim2real learning. *arXiv:2101.07169*, 2021.

[15] S. Wang, M. Lambeta, L. Chou, and R. Calandra. Tacto: A fast, flexible and open-source simulator for high-resolution vision-based tactile sensors. *arxiv:2012.08456*, 2020.

[16] B. Ward-Cherrier, N. Pestell, L. Cramphorn, B. Winstone, M. E. Giannaccini, J. Rossiter, and N. F. Lepora. The TacTip Family: Soft Optical Tactile Sensors with 3D-Printed Biomimetic Morphologies. *Soft Robotics*, 5(2):216–227, 2018. ISSN 2169-5172.

[17] N. Kuppuswamy, A. Alspach, A. Uttamchandani, S. Creasey, T. Ikeda, and R. Tedrake. Soft-bubble grippers for robust and perceptive manipulation. *arXiv:2004.03691*, 2020.

[18] K. Shimonomura. Tactile image sensors employing camera: A review. *Sensors*, 19(18), 2019.

[19] M. K. Johnson and E. H. Adelson. Retrographic sensing for the measurement of surface texture and shape. In *CVPR*, pages 1070–1077. IEEE, 2009.

[20] A. Padmanabha, F. Ebert, S. Tian, R. Calandra, C. Finn, and S. Levine. Omnitact: A multi-directional high-resolution touch sensor. In *ICRA*, pages 618–624. IEEE, 2020.

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

[22] C. Sferrazza and R. D'Andrea. Design, Motivation and Evaluation of a Full-Resolution Optical Tactile Sensor. *Sensors*, 19(4):928, 2019. ISSN 1424-8220.

[23] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. In *CVPR*, pages 11154–11163. IEEE, 2020.

[24] N. F. Lepora, A. Church, C. De Kerckhove, R. Hadsell, and J. Lloyd. From pixels to percepts: Highly robust edge perception and contour following using deep learning and an optical biomimetic tactile sensor. *RAL*, 4(2):2101–2107, 2019.

[25] S. Kurt and S. A. Opensimplex noise. https://github.com/lmas/opensimplex, 2021.

[26] N. F. Lepora and J. Lloyd. Pose-based servo control with soft tactile sensing. *arXiv:2012.02504*, 2020.

[27] J. Lloyd and N. F. Lepora. Goal-driven robotic pushing using tactile and proprioceptive feedback. *arXiv:2012.01859*, 2020.

[28] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE SMC*, 13(5):834–846, 1983.

[29] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. Available: https://github.com/DLR-RM/stable-baselines3, 2019.

[30] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[31] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241. Springer, 2015.

[32] N. F. Lepora and J. Lloyd. Optimal deep learning for robot touch: Training accurate pose models of 3d surfaces and edges. *RAM*, 27(2):66–77, 2020.

[33] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv:1812.05905*, 2018.

[34] P.-W. Chou. The beta policy for continuous control reinforcement learning. Master's thesis, Pittsburgh: Carnegie Mellon University, 2017.

[35] S. Bohez, A. Abdolmaleki, M. Neunert, J. Buchli, N. Heess, and R. Hadsell. Value constrained model-free continuous control. *arXiv:1902.04623*, 2019.

[36] T. Wang, M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, pages 8798–8807. IEEE, 2018.

[37] C. Chen, U. Jain, C. Schissler, S. V. A. Gari, Z. Al-Halah, V. K. Ithapu, P. Robinson, and K. Grauman. SoundSpaces: Audio-Visual Navigation in 3D Environments. In *ECCV*, volume 12351 LNCS, pages 17–36. Springer, 2019.

[38] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. RLBench: The Robot Learning Benchmark & Learning Environment. *RAL*, 5(2):3019–3026, 2019.

[39] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. *arXiv:2009.12293*, 2020.

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

[41] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme. NeuralSim: Augmenting Differentiable Simulators with Neural Networks. *arXiv:2011.04217*, 2020.

[42] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *CoRL*, pages 734–743. PMLR, 2018.

[43] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[44] I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv:2004.13649*, 2020.

[45] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, volume 33, pages 19884–19895. Curran Associates, Inc., 2020.

A Robotics Simulation

A.1 Physics Engine

Physics-based simulation has played a vital role in the field of robotics, enabling rapid prototyping, testing and experimentation. The accuracy and speed of simulation has scaled with available computation. Modern methods such as RL have leveraged this to simulate very large data sets that are otherwise impractical to collect from the real world, hence far more complex and general behaviours can be learnt. The majority of simulated robotics has focused on rigid-body dynamics and vision sensors. More recently, a range of specific environment suites have been introduced to bring simulation closer to reality or to facilitate research in an under-represented direction [37, 38, 39, 40, 41]. Collectively, these works highlight the importance of increasing the breadth and capabilities of simulation software available to researchers.

There are many choices for physics engines when developing a new suite. Here we choose the PyBullet [5] for its fast GPU rendering, support for deformable objects [42], fast and reliable kinematics and dynamics solvers, and its demonstrated sim-to-real success in robotics [6]. Importantly, PyBullet is open source and non-commercialised software which helps to improve accessibility and lower the barrier of entry for RL research. That said, the tools used here to build our tactile suite are available in other physics engines such as Mujoco, Gazebo, Nvidia-isaacgym and Unity ML agents.

A.2 Control

Throughout this work we use Cartesian velocity control where the control input is a desired velocity (twist) specified by a 6 DoF action constrained to the allowed modes of control of the task. A control rate specifies the frequency that new actions can be sent to the robot, with maximum velocity limits also imposed. For most of our experiments, we set a velocity control rate of 10 Hz, a maximum linear velocity of $10\,\mathrm{mm\,s^{-1}}$ and a maximum angular velocity of $5\,^{\circ}\,\mathrm{s}^{-1}$. When undergoing a series of predefined and random actions we notice no significant difference in Tool Centre Point (TCP) pose between simulation and the real robot.

Work-space coordinate frames are set in both simulation and reality, specific to each environment, with each action sent to the robot consisting of a Cartesian move relative to the work frame. Therefore, the learned policies can be transferred from sim-to-real without exactly replicating the simulated task; for example edges and surfaces can be placed in alternative locations provided the work frame is set correctly. As a consequence, the policy transfers even when there are notable differences in the simulation, such as mirrored arm configurations used in this work. Thus, in principle the policies could also be transferred to other robot arms, providing the same speed and frequency of control can be achieved.

B Reinforcement Learning Parameters

Near-default hyper-parameters are used in all training (full list in Table 4). Image-based observations use the Atari Nature [43] convolutional layers followed by two 256-node fully connected (FC) layers. Oracle observations use only the FC layers. For tasks that require both image and state data, the state data is passed through two 64-node FC layers and the output concatenated with the flattened output of the convolutional layers, which is then passed through the final FC layers for action and value prediction. The convolutional weights are shared for all policy and value networks. Small random image translation augmentations help to improve performance and stabilise training, as proposed in [44, 45].

Table 4: RL and network hyperparameters.

	Param	Value
Feature Extractor	Input dim	[128, 128, C]
	Conv filters	[32, 64, 64]
	Kernel widths	[8, 4, 3]
	Strides	[4, 2, 1]
	Pooling	None
	Output dim	512
	State Encoder	[64, 64]
	Activation	*ReLU*
	Initialiser	*Orthoganal*
RL Net	Policy	[256, 256]
	Value	[256, 256]
	Activation	*Tanh*
PPO parameters	Learning Rate	3×10^{-4}
	n/ Envs	10
	Epoch steps	2048
	Batch size	64
	n/ Epochs	10
	Discount (γ)	0.95
	GAE lambda	0.9
	Clip range	0.2
	Entropy coeff	0.0
	VF coeff	0.5
	Max grad norm	0.5
	KL limit	0.1
	Optimiser	*Adam*

C Reinforcement Learning Environment Details

Table 5: Edge Follow environment description.

Observation	• **Oracle:** {TCP pos, TCP lin vel, Goal pos, Edge ang} • **Tactile:** {Tactile Image} • **Vision:** {RGB Image} • **Vision + Tactile:** {RGB Image, Tactile Image}
Action Space	$\{x, y\}$
Reward	- (Euclidean distance from TCP to goal + perpendicular distance from TCP to edge)
Termination	• max episode length reached • euclidean distance from TCP to goal $< 1\text{cm}$
History	1 Frame.
Randomisation	• Edge randomly orientated through 360°. • Distance tip is embedded onto edge is randomly selected between 1.5mm and 3.5mm.

Table 6: Surface Follow environment description.

Observation	• **Oracle:** {TCP pos, TCP orn, TCP lin vel, TCP ang vel, Goal pos, Target surface height, Target surface normal} • **Tactile:** {Tactile Image} • **Vision:** {RGB Image} • **Vision + Tactile:** {RGB Image, Tactile Image}
Action Space	$\{z, Rx, Ry\}$
Reward	-(z difference between TCP and local surface index + cosine difference between TCP normal and local surface normal)
Termination	• max episode length reached • euclidean distance from TCP to goal $< 1\text{cm}$
History	1 Frame.
Randomisation	• Surface randomly generated w/ OpenSimplex noise. • Direction of goal randomly selected from $[0°, 360°]$.

Table 7: Object Roll environment description.

Observation	• **Oracle:** {TCP pos, TCP orn, TCP lin vel, TCP ang vel, Obj pos, Obj orn, Obj lin vel, Obj ang vel, Goal pos, Obj radius} • **Tactile:** {Tactile Image, Goal pos} • **Vision:** {RGB Image, Goal pos} • **Vision + Tactile:** {RGB Image, Tactile Image, Goal pos}
Action Space	$\{x, y\}$
Reward	-(euclidean distance from object to goal)
Termination	• max episode length reached • euclidean distance from object to goal $< 1\text{mm}$
History	1 Frame.
Randomisation	• Random starting position of object in TCP frame. • Random marble size between 5mm and 10mm diameter. • Random distance embedded into the sensor.

Table 8: Object Push environment description.

Observation	• **Oracle:** {TCP pos, TCP orn, TCP lin vel, TCP ang vel, Obj pos, Obj orn, Obj lin vel, Obj ang vel, Goal pos, Goal orn} • **Tactile:** {Tactile Image, TCP pos, TCP orn, Goal pos, Goal orn} • **Vision:** {RGB Image, TCP pos, TCP orn, Goal pos, Goal orn} • **Vision + Tactile:** {RGB Image, Tactile Image, TCP pos, TCP orn, Goal pos, Goal orn}
Action Space	$\{y, Rz\}$
Reward	-(Euclidean distance from object to goal + cosine distance from object orn to goal orn + cosine distance from TCP normal to object normal)
Termination	• max episode length reached • Euclidean distance from object to final goal $< 2.5\text{cm}$
History	1 Frame.
Randomisation	• Random trajectory of goals generated with OpenSimplex Noise.

Table 9: Object Balance environment description.

Observation	• **Oracle:** {TCP pos, TCP orn, TCP lin vel, TCP ang vel, Obj pos, Obj orn, Obj lin vel, Obj ang vel} • **Tactile:** {Tactile Image} • **Vision:** {RGB Image} • **Vision + Tactile:** {RGB Image, Tactile Image}
Action Space	$\{x, y\}$
Reward	+1 per step
Termination	• max episode length reached • object tilts passed set angle (35°)
History	2 Frames.
Randomisation	• Random external force perturbation applied at start of episode.

D Full Reinforcement Learning Results

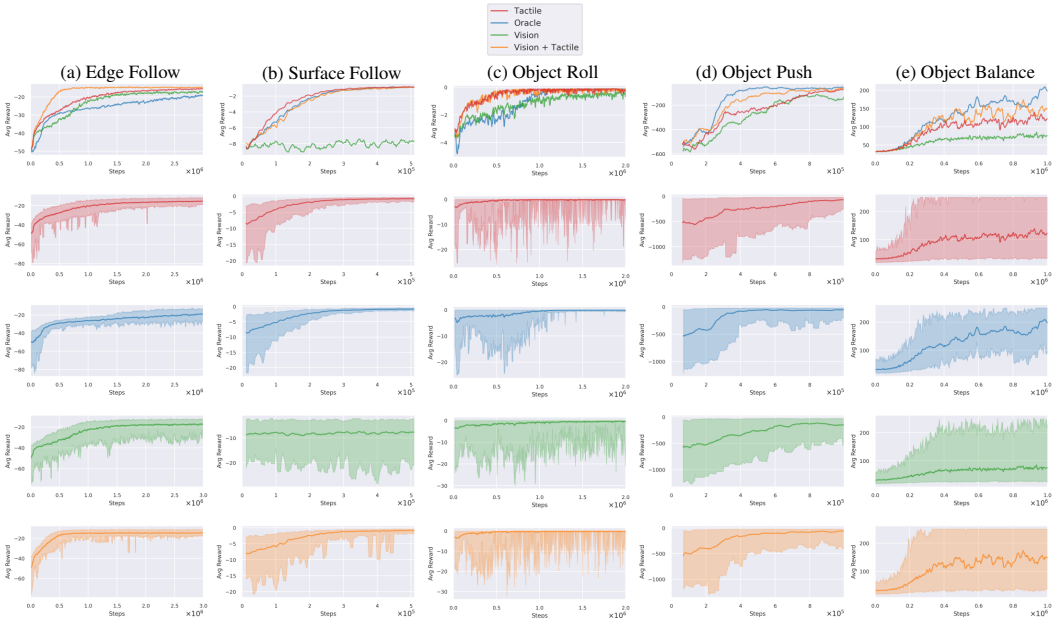

Figure 9: Full training results of reinforcement learning agents. Results are smoothed with a window size of 50 followed by averaging over 3 seeds. Shaded regions indicate maximum and minimum reward achieved over the 3 seeds (after smoothing).

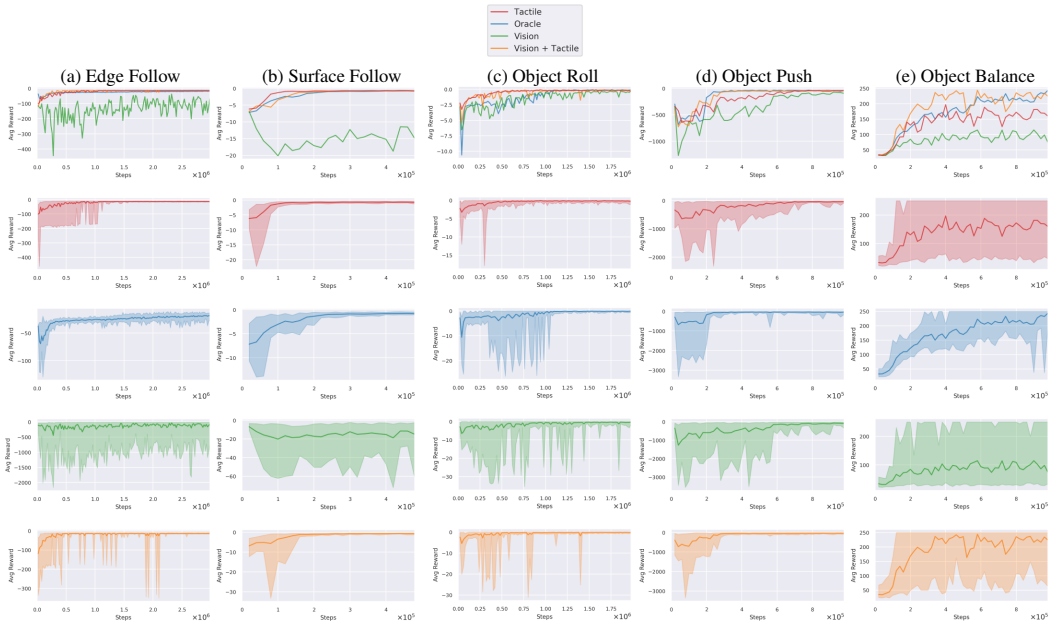

Figure 10: Full evaluation of reinforcement learning agents throughout training. 10 evaluation episodes occur every 20,000 steps, using deterministic actions. Results averaged over 3 seeds. Shaded regions indicate maximum and minimum reward achieved over the 3 seeds.

E Pix2Pix Architecture

Table 10: Pix2Pix architecture and parameters.

	Layer	Details			
Params	Batch Size	64			
	Learning Rate	0.0002			
	Image Norm	True			
	Image Trans	[2.5%, 2.5%]			
	Loss Weights	Wgan: 1.0, Wpix: 100.0]			
Generator	Input dim	[128, 128, 1]			
	Output dim	[128, 128, 1]			
		Input	**Output**	**Dropout**	**Norm**
	Down 1	1	64	None	False
	Down 2	64	128	None	True
	Down 3	128	256	None	True
	Down 4	256	512	0.5	True
	Down 5	512	512	0.5	True
	Down 6	512	512	0.5	True
	Down 7	512	512	0.5	True
	Up 1	512	512	0.5	True
	Up 2	1024	512	0.5	True
	Up 3	1024	512	0.5	True
	Up 4	1024	256	0.5	True
	Up 5	512	128	None	True
	Up 6	256	64	None	True
Discriminator	Input dim	[128, 128, 2]			
	Output dim	[16, 16, 1]			
		Input	**Output**	**Norm**	
	Disc 1	2	64	False	
	Disc 2	64	128	True	
	Disc 3	128	256	True	
	Disc 4	256	512	True	

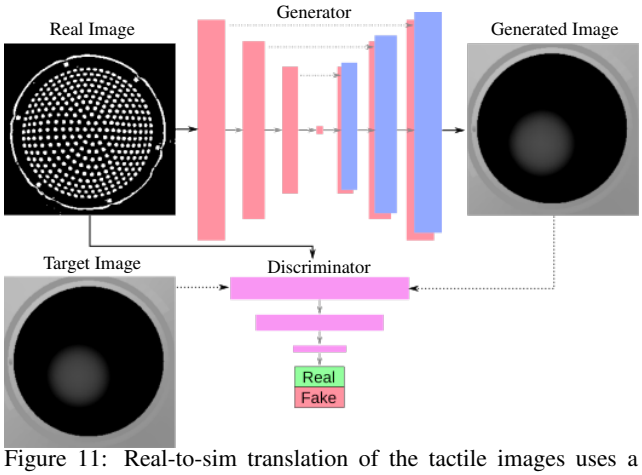

Figure 11: Real-to-sim translation of the tactile images uses a pix2pix-trained GAN. Real tactile images are processed by the generator to produce images that match the target simulated tactile images. The Discriminator is tasked with detecting whether an input tactile image pair is real or fake.

F Image Translation Data Collection

For the edge-following environment, we collect tactile images pressed onto a straight edge, varying the orientations, radial displacements and penetration of the sensor. A hemispherical sensor tip is used with the tool center point (TCP) located centrally at the end of the sensor. Relative to the TCP, the data is gathered over ranges: orientation $Rz \in [-179°, 180°]$, radial displacement $y \in [-6, 6]\text{mm}$, and penetration $z \in [3.5, 5.5]\text{mm}$.

For the surface-following and object-pushing environments, we collect tactile images pressed onto a flat surface, varying the orientations and penetration, also with a hemispherical tip. Relative to the TCP, data is gathered over ranges: orientation $\{Rx, Ry\} \in [-15°, 15°]$, and penetration $z \in [2, 5]\text{mm}$.

For the object-rolling environment, we collect tactile images pressed onto a spherical probe stimulus, using a flat sensor tip appropriate to this environment. 9 spherical probe stimuli are used, ranging over $[2, 6]\text{mm}$ radius in 0.5mm increments. The sensor is positioned to contact the probe at random placements within a 15mm disk surrounding the centre of the tip. In this case, shear is not introduced into the data collection because rolling objects induce negligible motion-dependent shear.

F.1 Full SSIM Scores

We measure the SSIM scores across the validation sets collected for each task, each consisting of 2000 image pairs. For the Edge, Surface and Probe datasets respectively, we found mean scores of $[0.995, 0.992, 0.994]$, min scores of $[0.985, 0.982, 0.974]$, and max scores of $[0.999, 0.999, 0.999]$. Whilst high scores are expected due to the relatively sparse target images, this indicates strong performance in all cases.