Real Robot Challenge 2020 - Phase 2 Report

Team Name: sombertortoise

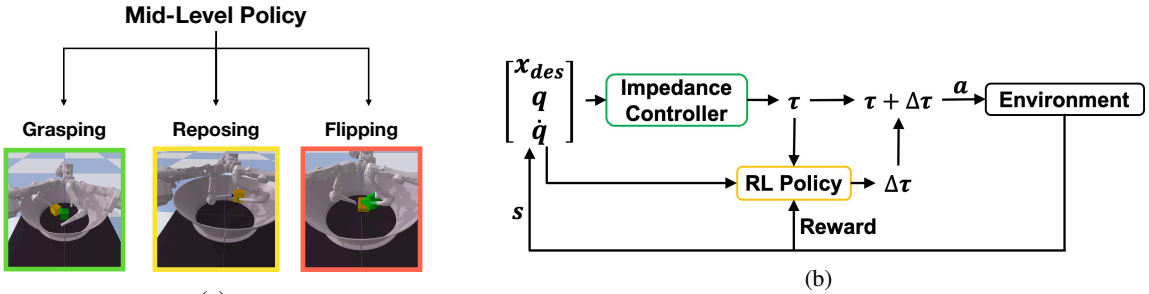
Abstract: In Phase 2, we solved the tasks of manipulating the cube to random goal positions with the real robot. Each task is decomposed into three stages: moving the fingers to the cube to grasp it, reposing the cube with three fingers, and flipping the cube with two fingers. These grasping, lifting, and flipping primitives are sequenced with a state-machine. As a baseline, we designed model-based controllers to execute each of these primitives. To improve performance of the controllers on the real robot, we trained a residual policy, which adds joint-torque corrections to torques output by the controller to mitigate discrepancies between the simulated and real robot.

1 Introduction

Our approach rests on two core hypotheses: (1) In-hand manipulation tasks can be achieved by sequencing manipulation primitives reaching, reposing, and flipping, and (2) combining model-based optimal control techniques with model-free reinforcement learning reduces the complexity of physical modeling and computational complexity of multi-contact control and planning while reducing sample complexity in policy learning. Driven by these hypotheses, our overall approach integrates these two methods. We decomposed the task of grasping and moving the cube into three primitives: moving the fingers to the cube to grasp it, reposing the cube with three fingers, and if required, flipping the cube with two fingers. These grasping, lifting, and flipping primitives are sequenced with a state-machine. As a baseline, we designed model-based controllers to do these tasks. To improve performance of these controllers on the real robot, we learned a residual policy [1], which adds joint-torque corrections to the controller commands to help account for model discrepancies between the simulated and real robot. Using this combined approach, we solve the tasks of manipulating the cube to random goal positions. Example videos can be viewed here.

2 Method

2.1 Overview



(a) (b)

Figure 1: (a) A hand-designed mid-level policy used for each task. (b) The residual reinforcement learning policy architecture

Fig 1 shows our overall approach in Phases 1 and 2. Using a hand-designed state machine, the hierarchical policy sequences grasping, reposing, and flipping primitives in order to move the cube to the goal position. For all the tasks, we assume prior knowledge of robot dynamics, physical parameters of the object, as well as

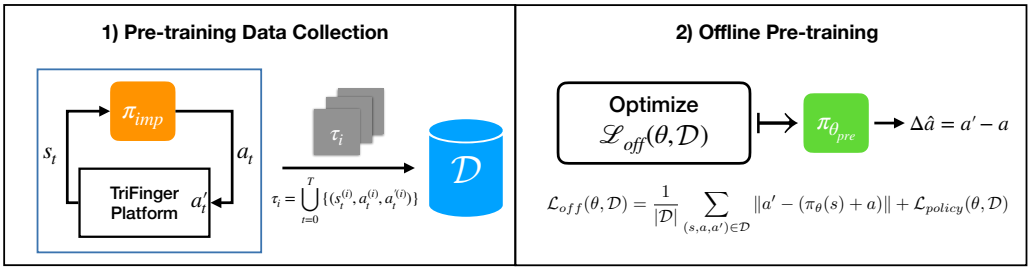


Figure 2: Offline residual policy learning algorithm

accurate state feedback on object pose and finger joints. With that, a model-based trajectory optimizer and controller is sufficient for implementing most of the primitives in simulation. However, unlike in simulation, the torques applied on the real robot will never match the desired torque actions output by the controller. Therefore, we use offline training to learn a residual policy that mitigates this sim-to-real action discrepancy.

To complete each task, the mid-level policy first determines whether the cube should be flipped before moving, so that it may be lifted directly to the goal orientation. If flipping is required, two fingers are used to grab two of the bottom corners and perform a lifting action that pivots the cube on one of the four edges resting on the table; this flips the cube onto a different face. Next, the policy plans and executes a reaching trajectory to move each fingertip to the centers of the closest faces of the cube, and then a reposing/lifting trajectory which lifts/reposes the cube from its current position to the goal position.

When interacting with the real robot, the learned residual policy, $\pi_\theta(s, a_{des}) = \Delta a$, acts as a corrective term designed to help bridge the sim-to-real gap, and is trained using both offline-collected data, as well as data on the real robot. The reward for the residual is computed using a combination of the object pose's distance to its goal pose, with reward bonuses for reaching waypoints on the desired trajectory.

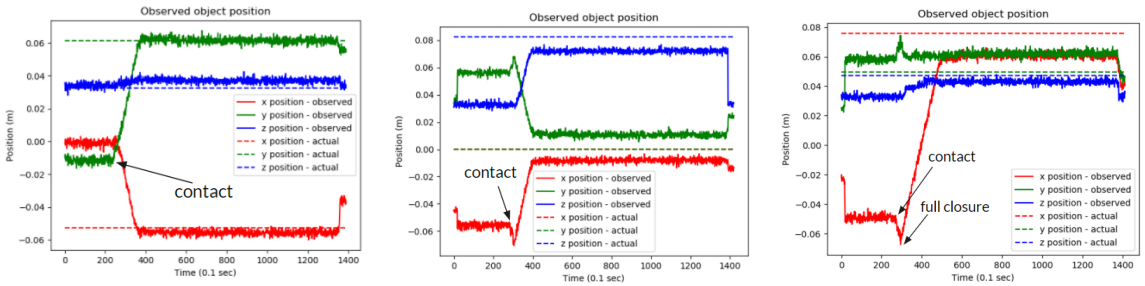
2.2 Residual Policy Training

Even though residual reinforcement learning dramatically reduces the training time needed on a real robot, the training required can still be significant. Given that we had already collected many runs of data when tuning the impedance controller policy on the real robot, we were able to also use this to pre-train a good initialization for the residual policy. Fig 2 shows the offline pre-training algorithm we designed. The offline dataset $\mathcal{D}$ contains trajectories τ composed of the state s, the action taken a', and the desired impedance controller action a, and the reward r. The residual policy is then trained to estimate the error between the desired and actual action. However, given the additional time it took to get the initial impedance controller working independently, we are still integrating this combined policy on the real robot, as we noted artifacts of overfitting to the training data hindering task performance in some runs.

2.3 Impedance controller

We compute the joint torques necessary for tracking the desired fingertip position and velocity trajectories in Cartesian space using an impedance controller similar to the one used in [2] with additional gravity compensation, shown in equation 1. For each finger j, $\Gamma_j \in \mathbb{R}^3$ is the output torques of one finger, f_j is the desired force to be applied by the finger, $J_{v,j}$ is the linear Jacobian of one fingertip, and G_j is the gravity compensation vector.

$$\Gamma_j = J_{v,j}^T(k_p(x_{d,j} - x_j) + k_v(\dot{x}_{d,j} - \dot{x}_j) + f_j) + G_j \tag{1}$$



(a) Task 1 (b) Task 2 (c) Task 3

Figure 3: Observed vs. desired object positions for tasks 1, 2, and 3 examples

2.4 Grasping

We formulate a direct collocation trajectory optimization problem to compute collision-free fingertip position and velocity trajectories to desired contact points on the cube. The optimization problem considers only the kinematics of the fingers and assumes that the pose of the cube is static. To prevent the fingers from colliding with the cube, we include a collision penalty term in the objective function.

2.5 Reposing

Assuming fixed contact point positions, we use direct collocation to compute an open-loop trajectory for the object as well as the associated forces that need to be applied at the contact points to move the object along this trajectory. In the optimization problem, we constrain the contact forces to lie within linear approximations of friction cones to prevent contact slippage and specify a target normal force that the fingers should apply to ensure sufficient internal forces on the cube.

3 Results and discussion

With our current method, the controller is able to grasp and repose the cube with the real robot, but struggles with the flipping task. The observed and desired object positions for example runs of tasks 1, 2, and 3 are shown in Figure 3. While the robot brings the cube very close to the goal, there are small steady state errors which can be decreased with higher gains in the impedance controller.

When transferring our flipping method from simulation to the real robot, there were significant challenges posed by attempting to fine-tune the controller gains and the flipping approach. For instance, while trying to execute the flipping trajectories on the real system, we observed that the fingers were unable to pinch the corners of the cube tightly enough to lift and rotate it. We hypothesize that this was the result of applying the appropriate contact forces at contact point locations being more critical when only two fingers are pinching the cube. Using what we've observed in this phase, we will re-design our flipping primitive for the next phase. Lastly, we are still in the process of integrating the residual policy, which needs significant fine-tuning on the real robot for each individual task phase after pre-training.

References

[1] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[2] M. Wüthrich, F. Widmaier, F. Grimminger, J. Akpo, S. Joshi, V. Agrawal, B. Hammoud, M. Khadiv, M. Bogdanovic, V. Berenz, et al. Trifinger: An open-source robot for learning dexterity. *arXiv preprint arXiv:2008.03596*, 2020.