Real Robot Challenge: Phase 3 Report

ardentstork

December 11, 2020

Abstract

Our approach combines motion planning with several scripted motion primitives to manipulate the cuboid. In the previous two phases we focused on learning corrections to our controller, but in this phase we did not have time to train and test the models. However, we find that our controller is able to complete the four tasks. A video of our approach can be found here.

1 Approach

In this phase of the competition, we are tasked with manipulating a small cuboid by picking it up and moving it to various goal poses using the TriFinger Platform developed by Wüthrich et al. [6]. The task has four different levels, the first three require reaching a goal position, while the fourth requires also achieving a desired orientation (i.e., a goal pose). These are challenging tasks that often require multiple grasps and manipulations to achieve the goal pose. We find that many feasible grasps of the object are not suitable for solving the task, especially when matching orientation is required. Therefore, a successful approach must be able to search or plan over the space of possible grasps in some way. The small size of the object also requires precise positioning of the fingertips, which can be challenging in the presence of observation noise.

The core of our approach is a grasp-planning loop in which feasible grasps are sampled until a path using that grasp can be found to a desired pose. The path is then executed by tracking target joint positions with a PD controller. This approach doesn't rely on feedback of the object's pose while the object is in motion, allowing us to be robust to inaccurate or missed object observations. We combine this with a set of motion primitives which grasp and align the object before attempting to move it to the goal pose. An overview of our approach can be found in Figure 1.

1.1 Grasp and Motion Planning

We sample grasps in two ways: by randomly sampling a set of three contact points on the surface of the object, and by selecting from a set of predefined heuristic contact points. A grasp is then generated by assigning a fingertip to each contact point. We reject grasps that don't exhibit the following two properties: the grasp is *feasible*, meaning the finger tips can reach the contact points and there are no collisions between the fingers (excluding the tips) and the cube, and the grasp is in *force closure*, meaning we can resist any external force applied to the object. The force closure check is computed by assuming a Coulomb friction model and making a linear approximation to the friction cone.

Once a grasp has been sampled, we run an RRT [3] in task space (i.e., the pose of the object) starting from the current object pose and planning to a desired object pose. We enforce that the grasp is maintained during planning by rejecting object poses if the corresponding grasp is not feasible. If no feasible path to the goal is found, we sample another grasp and try again. To ensure we eventually find a feasible path, we slowly increase the size of the goal set of the RRT.

We then use the resulting waypoints to control the joint positions of the fingers. During execution, we check to see if the object has been dropped. If so, we reset and plan again. Notably, the execution of this plan does not rely on feedback of the object's pose. This makes us robust to errors in the observations, which can be substantial when the cube is occluded by the fingers, including missed detections and occasional 180 degree flips in the detected orientation. In the previous two phases, we included force control to help

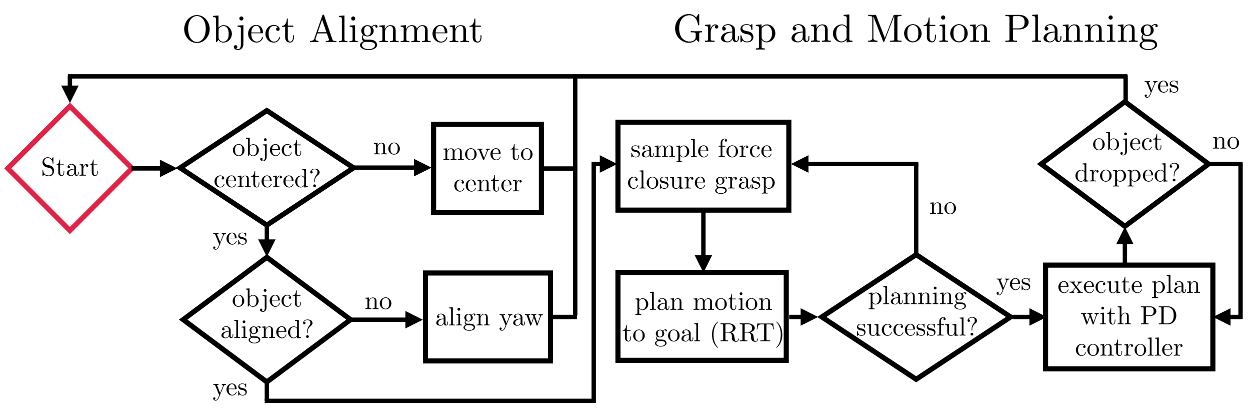

Figure 1: Structure of our approach. We first align the pose of the object by moving it to the center of the workspace and executing a yaw rotation to align the object with the goal orientation. We then find a grasp and motion plan which can move the object to the goal location. The object is then grasped and the plan is executed by tracking joint positions with a PD controller.

maintain grasping forces as the object moved, but the controller was not able to handle the increased observation noise of this phase.

In the cases in which the RRT generates a plan that does not move the object to the exact goal pose, we append a correction to the plan after execution that moves directly to the goal pose by translating and rotating the fingertip positions accordingly. In the previous phase, we also corrected for other errors such as slippage or changes in the grasp during execution of the plan, but we found the observation noise to be too large for this to work reliably.

1.2 Motion Primitives and Object Alignment

Many goal poses cannot be reached with a single grasp. Consequently, we use a set of scripted motion primitives to first align the object to the goal on the ground. These include: grasping the object, moving the object to the center of the workspace, and aligning the orientation of the object with the goal orientation. These primitives are combined as shown in Figure 1.

The *grasping* motion primitive moves the fingertips in a straight line to a predefined pregrasp position near the contact points, then grasps the object. This helps avoid accidentally hitting the object when approaching it. The smallest sides of the object are about the same size as the fingertips. As a result, grasping is prone to failure if there is noise in the object's observed pose. To combat this, we first move the fingers to a position that does not occlude the camera's view of the object and collect several observations. We then average and project those observations onto the ground plane by setting the height of the object's position and projecting its orientation into the subspace of yaw-only rotations. We then use the averaged observations for grasping.

The *move-to-center* motion primitive first grasps the object, then moves the fingertips along a straight-line path to the center of the workspace. We find that there is no need to use the above grasp-planning loop to orchestrate this motion ahead of time as it works well with almost all feasible force-closure grasps. The fingertips then release the object by moving away from and above it.

To make it easier to find a grasp and plan that can move the object to the goal, the *align-yaw* motion primitive rotates the object to be close to the goal orientation while remaining on the ground. This corresponds to projecting the goal-orientation onto the space of yaw-only rotations and rotating the object to the projected orientation. This primitive is used only for Level 4. In the previous phase, we used a scripted motion to rotate the object, but in this phase we found that the smaller object size often leads to collisions between the fingers. Instead, we use the grasp-planning loop defined above to find a feasible grasp and plan that achieve the rotation. We then grasp the cube, execute the planned motion, and release the grasp of the object.

Task	Mean	Median	Std. Dev.
Level 1	$-7,944$	$-8,104$	$4,214$
Level 2	$-4,039$	$-3,888$	446
Level 3	$-9,896$	$-6,046$	$10,066$
Level 4	$-24,755$	$-23,409$	$10,548$

Table 1: Unofficial performance of our approach showing the reward statistics for each level based on 10 episodes on the real system. Official results can be found on the leaderboard with username 'ardentstork' at `https://real-robot-challenge.com/leader-board`.

2 Discussion

We evaluate our approach by running 10 episodes for each level on the real robot. Table 1 presents the performance of each algorithm. We find that our approach is able to consistently solve the tasks, is robust to observation noise, and can recover from failures (i.e., dropping the object). However, there is still room for improvement. We often find that executing our planned motions only takes the object near the goal pose. This is because the grasp was invalid at the goal pose, planning failed or took too long, or the object slipped while executing the plan. Robustly correcting for these errors could improve our performance considerably. Additionally we accumulate a lot of negative reward while the RRT is planning, and failures that trigger another round of planning can lead to a large variance in rewards. Using a Probabilistic Roadmap (PRM) [2] would allow us to precompute the search space, and thereby speed up planning significantly.

Due to time constraints, we were unable to try everything we had planned for Phase 3. Specifically, we implemented a framework for residual policy learning [1, 4] with domain randomization [5] to help transfer from simulation to the real robot, but did not have time to train and test the learned policies with a finalized controller. In the previous two phases, our policies were able to learn useful corrections to our controllers, and we believe such an approach would be beneficial in this phase as well.

We thank the organizers for all the effort they put into creating the TriFinger platform and hosting this competition. We are excited to use this platform to extend this work or for other research projects related to dexterous manipulation.

References

[1] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2019.

[2] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

[3] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[4] Tom Silver, Kelsey R. Allen, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[5] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[6] Manuel Wüthrich, Felix Widmaier, Felix Grimminger, Joel Akpo, Shruti Joshi, Vaibhav Agrawal, Bilal Hammoud, Majid Khadiv, Miroslav Bogdanovic, Vincent Berenz, et al. Trifinger: An open-source robot for learning dexterity. *arXiv preprint arXiv:2008.03596*, 2020.