Fetchit! Object Picking Using RL

COMS 6731 Final Report, May 2019

Lucas Schuermann, lvs2124@columbia.edu

1 Introduction

As a part of the Fetchit! challenge, a number of pick and place tasks must be executed: a set of bolts from a resting bin into the competition bin, a set of bolts placed into a machine, and so on. Rather than individually/manually optimize grasping code for each, we decided to explore the viability of training a model to do so. Specifically, we generate demonstrations from the OpenAI gym for competition-like objects, train an RL policy to act on simulated perception data, and measure error in a simple goal pick and place task.

2 Related Past Work

The current state-of-the-art for sparse reward continuous state space environment reinforcement learning is Hindsight Experience Replay (HER) [1], developed by OpenAI, and building upon work from Google Deepmind. Learning from demonstration was determined to be our most reasonable approach in the timelines of the Fetch competition, as we do not have enough computing power, nor the time to otherwise train grasping policies for use in the competition. It is our hope to validate that learning from demonstration is a reasonable way to create a convergent model for this relatively difficult (general) task.

Whilst a variety of open-source projects and academic papers have attempted to re-implement HER [3], very little work has been available on extensions, as well as tests beyond the core set of OpenAI gyms. A key contribution of this work is testing the efficacy of demonstration modifications in new gym environments that more closely replicate the Fetchit! challenge.

3 Implementation Details

3.1 High-level Model Design

We structured a Deep Deterministic Policy Gradient [4] to maintain 2 separate actor and critic networks. The actor network gives behavior, while the critic attempts to estimate the return on an action given a state. HER-like techniques are great for solving sparse reward problems–we are additionally hobbled by sparse rewards just in the ability of the agent to explore the state space, given a constrained number of iterations. Its been observed that DDPG+HER works well for training these grasping policies (e.g. in the OpenAI baselines, with small changes), but we follow Nair et al. [5] to make the improvements as follows to reduce training convergence time.

Firstly, in addition to the normal HER replay buffer, we include second buffer containing the demonstration data. Each batch draws a number of samples from this buffer to update both the actor and critic network. Secondly, to integrate, a loss is applied to the actor based upon deviation from the demonstration examples. Lastly, a Q-value filter is applied to account for possible suboptimality of demonstrations–the aforementioned loss is only applied when the critic estimates the state reward for the demonstrator action would be better than the actor's action.

3.2 Environment

For training and testing, we use a Ubuntu 18.04.2 LTS server running python3 with tensorflow 1.13.1 and numpy 1.16.3. We obtained a student trial version of mujocopy and have installed the OpenAI gym and baseline packages from the most recent tip of tree as of April 2019.

3.3 Architecture

Specifically, the core of this code is split into three parts: a set of modified OpenAI gyms, building upon the fetch pick and place suite; a set of manually-coded solutions to the gyms which can be used to generate demonstration data files; and a modified DDPG+HER implementation built to use the demonstration data as described in section 3.1.

3.3.1 Modified Gyms

The robotics OpenAI gyms are built essentially as small wrappers on top of mujoco scenes [2]. Mujoco scenes are specified by XML files labeling hierarchies of bodies in a scene, each with a label and comprised of primitives, meshes, joints, and so on. The pick and place demo code, for example, loads a common robot.xml file, which has meshes and primitives for the fetch robot, as well as initial positions and joints to properly link up kinematics. Additionally, the pick and place demo code includes a simple table with a box primitive, and an attached *site* for monitoring or referencing the box position in the gym. The gym code itself simply defines init, observation, and step methods that place the box at a random location, give the position and velocity of the gripper and box etc at each step, and move the objects to a new position over time by setting a target position for the joints in mujoco given the user-supplied action. Figure 1 shows the simplest pick and place configuration, as well as the bolt STL mesh object that we will be replacing the box with.

Two modified gyms were built. One uses a modified scene to pick and place a bolt mesh to better simulate the competition objects and environment. The other uses an entirely modified gym implementation to vary the z-axis orientation of an elongated box (along its length axis) and take inputs of the gripper orientation in the action vector. Each presented individual challenges. For the former, loading and defining the collision properties of the mesh in mujoco took a large amount of manual fine-tuning, as well as simply loading it in at a reasonable scale and position relative to the other objects in the scene. The mesh object was eventually specified with a contact resolution solver margin etc. similar to how the Fetch robot was constructed in the scene. A cylinder was used to track the mesh reference as a *site* object, which was manually placed and tuned as well. For the latter, switching to an elongated box in the XML file was quite simple, but modifying the Fetch environment such that the other environments did not break proved to be a bit more of a challenge. The eventual solution was found by appending an optional flag to the initialization of scenes which were to respect rotation, and using some of the handy provided *rotations* utility library to generate a quaternion given a randomly chosen theta along the z-axis. Further integration between the action state vector and mujoco was relatively straightforward.

3.3.2 Demonstration Solutions

Broadly speaking, the approach to gathering demonstration data was to (1) load a standardized OpenAI environment, which could also be fed to the RL framework, (2) define a function to produce winning actions given the observation state vectors, (3) test and dump (observation, action) pairs to a file so that they can be loaded into the demonstration replay buffer during model training.

For the default (Fetch pick and place) OpenAI gym, the solution was given by comparing the distance between the end gripper and the object of interest (both given by the observation state). If the distance was greater than some epsilon value, the action (in this case simply a gripper target position coordinate vector and the gripper open amount) would be set to the position of the object, plus a vertical offset, moving the gripper above and opening. Once this distance was satisfied, the gripper would close and the vertical offset would be removed. Lastly, while the normed delta between the object and the goal was above an epsilon, the gripper would remain closed and be moved towards the goal position. Once this was satisfied, only an instruction to keep the gripper closed would continue to be sent. In all cases, a slight overshoot of the target position was used to instruct the joints to move more quickly, accomplished by just subtracting to get the vector between the current and desired position, and then scaling by a constant. It should be noted that the OpenAI gym constraint setup handles the kinematics of moving the Fetch arm in a valid way, and the action is always given simply by a set of new target coordinates. This approach was quite effective in creating (observation, action) pairs which were written to a file and consumed by the model.

For the custom gyms, more care had to be taken in creating solutions. Obviously, this creates difficulty in scaling the approach to new objects, but time is drastically saved in training and convergence of the policy. The same steps as above could generally be followed for picking up the bolt, but the vertical offset, grip closing speed, and distance epsilons to transition between steps had to be manually tuned to avoid edge cases such as the target being low to the ground and the gripper dragging the bolt along the table, causing it to be torn out of grasp. For the gym with randomly seeded rotation, a new set of rules were developed that included an initial observation made of the target's orientation, and setting the gripper goal orientation to be matching before approach. A number of manual parameter tweaks were also made before high-quality data could be gathered, as well as modifications to the gym random sampling to attempt to keep the rotated object placements and goal placements in a viable range.

3.3.3 Model

The model is built from open-source code given by OpenAI baselines, Nair, and Jangir, implementing the same framework in section 3.1. Specifically, the OpenAI DDPG implementation is mostly reused, so DDPG and actor-critic code are relatively barebones. Modifications have been made there and in a HER implementation to support the replay buffer, as well as integrate the demonstration loss function and normalization using the Q-value filter. Tensorflow was used (as always) to implement these extensions, and OpenMPI was used to support multi-core parallelism. Numpy was heavily relied upon for quick loading of demonstration data etc. Pickled policy files were generated by a training script and could be run in the gym by evaluating against new random states using a play script. Other command-line configuration niceties were added, as well as some documentation and a relatively extensible (observation, action) state vector since an initial goal of this project was to eventually extend to the gym-gazebo and deploy on the Fetch robot for the challenge.

It should be mentioned that in the OpenAI environments, playing with the kinematics solver, friction parameters, and so on did not substantially affect the convergence or training outcomes. This came as a relative surprise, since usually the accuracy and integration scheme are to blame for sim-to-real transfer difficulties. As we did not yet have time to implement gym-gazebo testing for a different simulation and transfer to real-world robot, we lack data on how well these models would perform in the real world.

3.4 Steps taken

- Create an OpenAI gym with mujoco physics installation to simulate Fetch grasp task.
- Generate set of demonstrations from OpenAI environment, pick and place block. Compress action set as examples for policy initial demonstration. See figure 1.
- Configure DDPG actor/critic 3 layers, 256 hidden, learning rates 0.001, averaging 0.8, buffer size 1E6, l2 rescaling of 1. Configure HERrandom exploration 20% of the time, 0.1 standard dev of gaussian noise applied to nonrandom actions taken (helps exploration). Support batching and normalization.
- Train DDPG+HER given demonstrations following above outlined approach. Record best policy over time, support arbitrary demonstration data and other training configurations. See figure 2.
- Create modified gym including mesh object (bolt) and debug mujoco physics with collision detection and joint placement.
- Optimize demonstration solution for mesh object (bolt), collect demonstration data. See figure 3.
- Visualize policy/measure accuracy in new environment by evaluating best model from training.
- Pleased with results from real-world bolt object, create modified gym allowing for varied rotations. Modify core OpenAI gym code to support specification of gripper target rotation in action.
- Optimize demonstration solution for rotation gym, collect demonstration data. See figures 4 and 5.
- Begin work on architecture training model to handle new action space and inputs, contemplate viability for Fetchit! competition. See figure 6.

- *Concurrently:* explore gym-gazebo, create Dockerized environment for Fetch gym, attempt to link server to running gym code.
- *Concurrently:* Optimize model and document code, package for further and future work, potentially in the context of the competition.

4 Conclusions

Conclusions of this project must be largely a discussion of its limitations. This was intended to be more an exploration of the viability of RL for grasping in the challenge. Our current work has moved to (1) a navigation stack, (2) an OpenCV masking and blob-based perception platform (object poses in camera coordinates), and (3) hard-coded and manually tested approach logic for different stations in the competition. This work is not well-suited in its current state to be deployed for the Fetchit! challenge, especially since the challenge objects are themselves relatively easy to grasp with conventional axis-aligned means and are always nicely posed and segmented. That said, this work provides a great starting point for further use in the coming years, by (a) proving the viability (at least in sim) of training RL models with demonstration for much more complex tasks than the conventional gyms, and (b) giving a starting point in implementation for a pipelined automated policy production system given competition object meshes.

5 Future Work

Whilst we see substantial progress and quickly-converging results with this modified demonstration-based RL grasping system, there are still a number of caveats. Firstly, there is the question/notion of sim-to-real transfer loss. The state in the OpenAI gym environment is provided as the object position, gripper position, gripper (open) state, and the object relative position. In more complex examples, we have rotation and velocity available as well. Certainly, using segmentation and depth estimation with OpenCV and the Fetch's perception system, we could reasonably recognize and establish object locations, but for this type of RL system, we'd expect a decent amount of sensitivity to both initial and ongoing position estimations. In testing, this could have been mitigated by introducing some stochastic jitter process in the observed object positions, but then we would expect convergence to significantly suffer as reward would always be measured from the absolute.

Secondly, there is the the question of generating high-quality training data. For speed, this initial testing was done with the OpenAI gym, but we began work on integration with ROS/gazebo. Some open source code exists in this space, but it is far from complete. Ideally, we would like to train in the same provided competition simulation environment, but this will take substantially more code. Similarly, attempting grasping (and even coding slightly modified reward functions for orientation-focused picks/places) of different competition objects will take further pipelining to both generate data and handle either separate network training or batching training/testing with some split of a group of object models. Time did not allow for sanity checks such as testing the resiliency of the gym specification and demonstration solutions to different meshes, but it would be my intuition that further tricks would have to be developed to speed up the process of generating gym specifications and training data given a new set of competition objects.

Again, in the interest of time, we have taken a different approach for Fetchit! this year, but we have made substantial and significant progress towards our stated project goals, and have demonstrated the viability (with time caveats) of a modified DDPG+HER demonstration-based RL system for pick/place, which is exciting! This was a tremendous learning opportunity for every aspect of the stack, and I am looking forward to returning to finish this work after we perfect our manual approaches to win the Columbia University Robotics Group a new Fetch robot! :)

References

[1] Andrychowicz, Marcin, et al. "Hindsight experience replay." Advances in Neural Information Processing Systems. 2017.

[2] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[3] Dhariwal, Prafulla, et al. "Openai baselines." GitHub, GitHub repository (2017).

[4] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[5] Nair, Ashvin, et al. "Overcoming exploration in reinforcement learning with demonstrations." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.

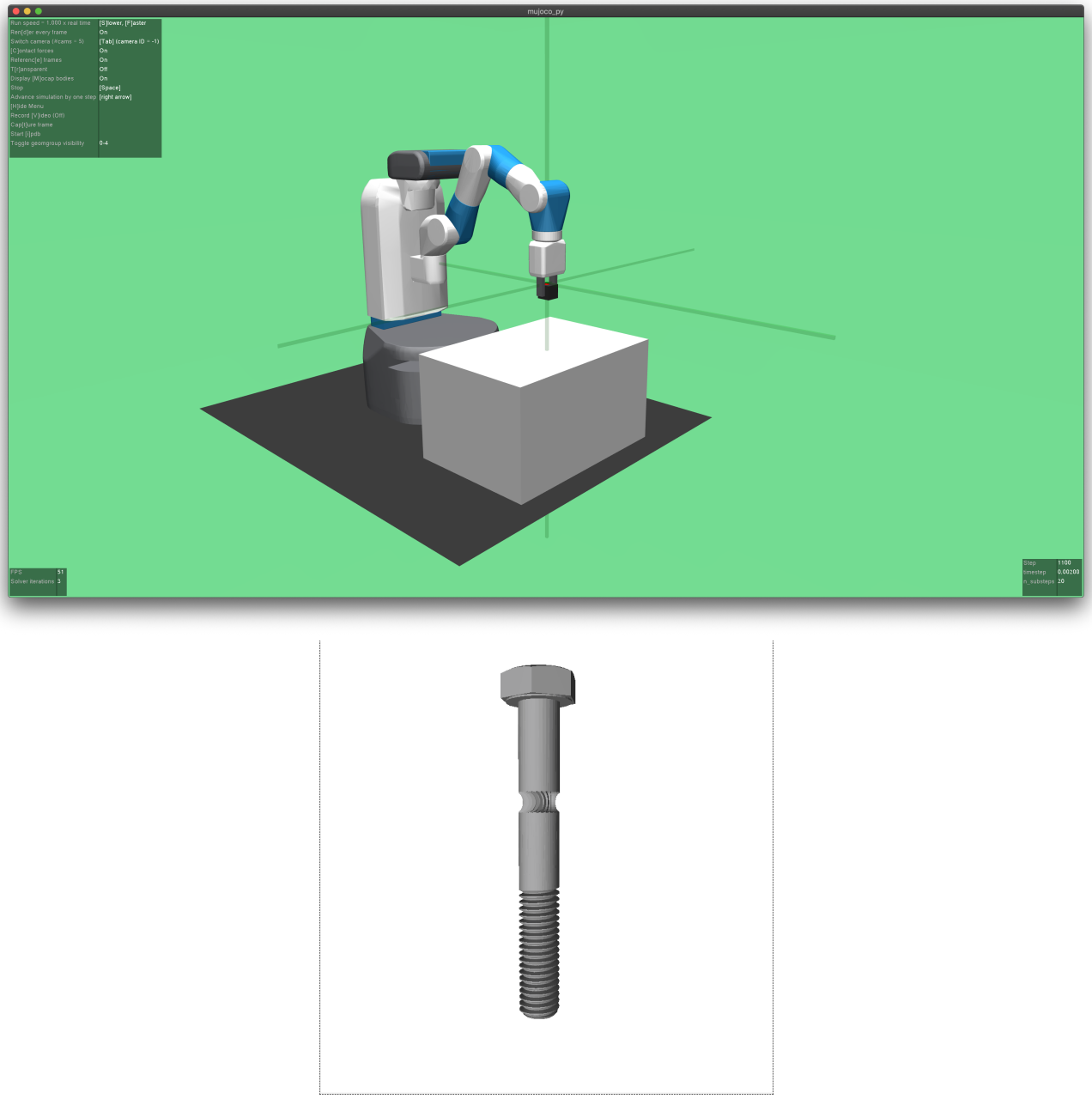

Figure 1: Top: Block pick and place OpenAI gym environment with Fetch+MuJoCo. Bottom: bolt STL model used to replicate competition objects.

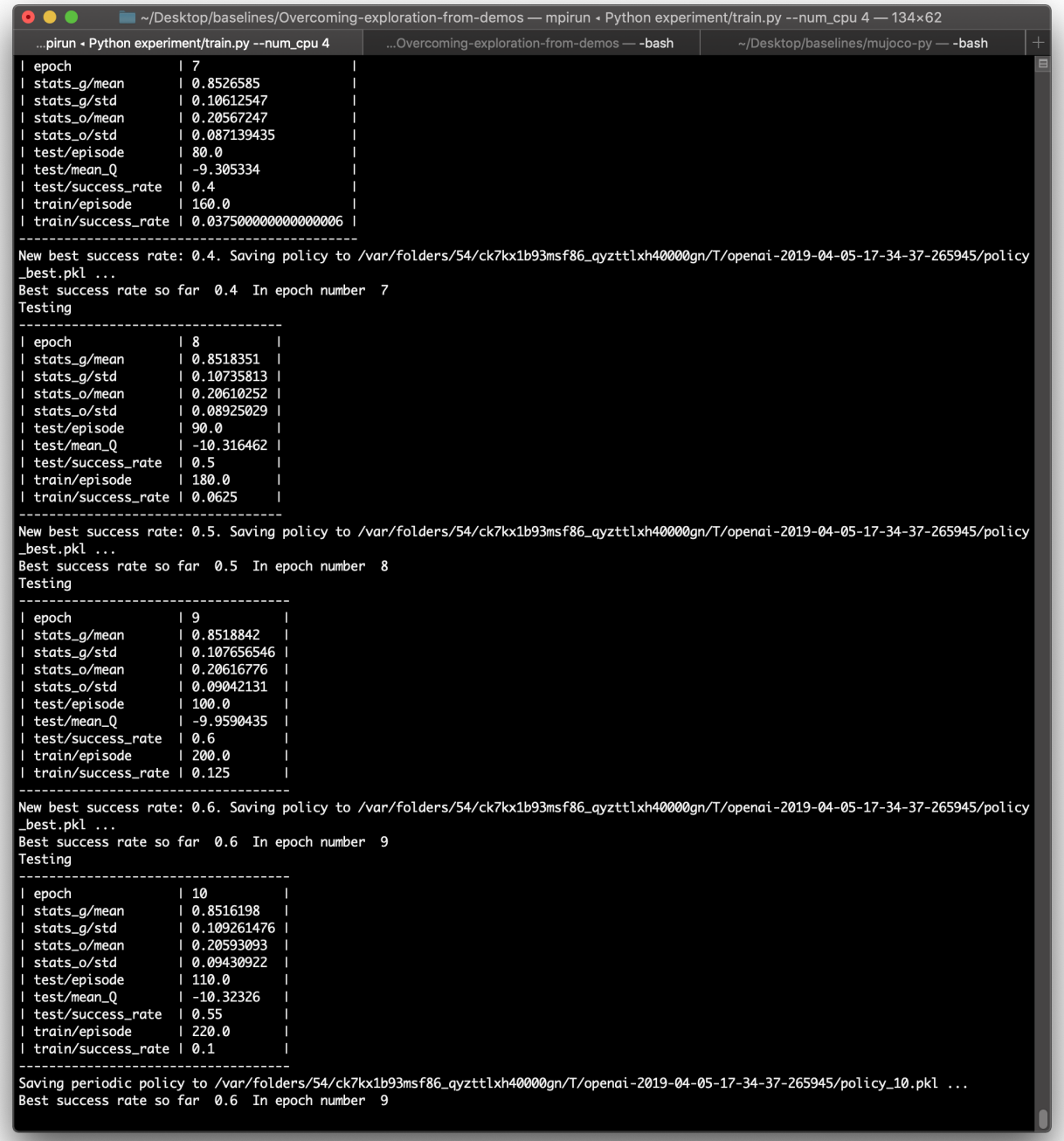

Figure 2: Training DDPG+HER with demonstration data leads to relatively rapid learning, contrasted with baseline more linear growth from low success rate.

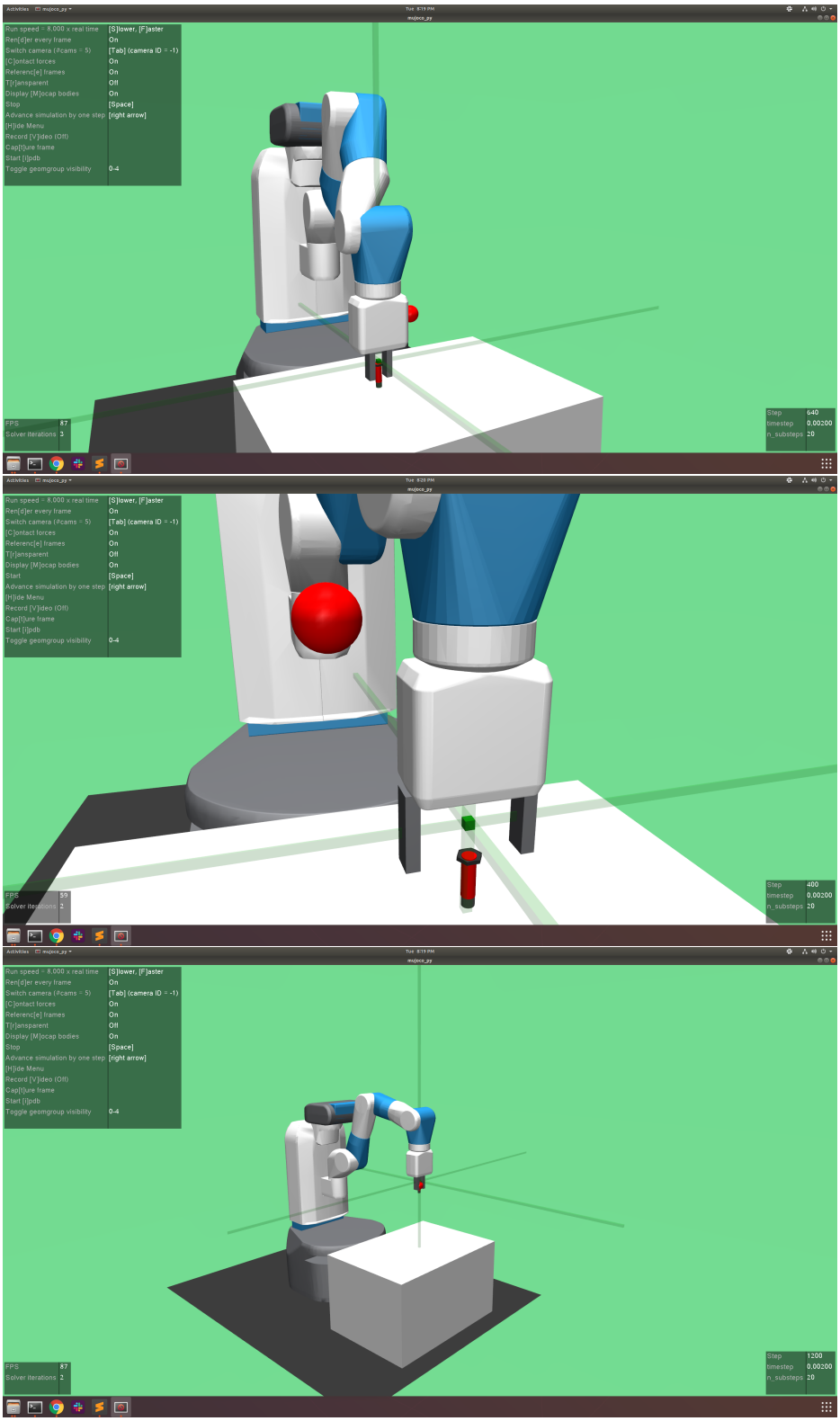

Figure 3: Top-to-bottom: competition pick and place task simulated using OpenAI gym with custom mesh, placement, and collision objects. Wide angle pick, closeup of pick (with red target), and wide angle final goal place.

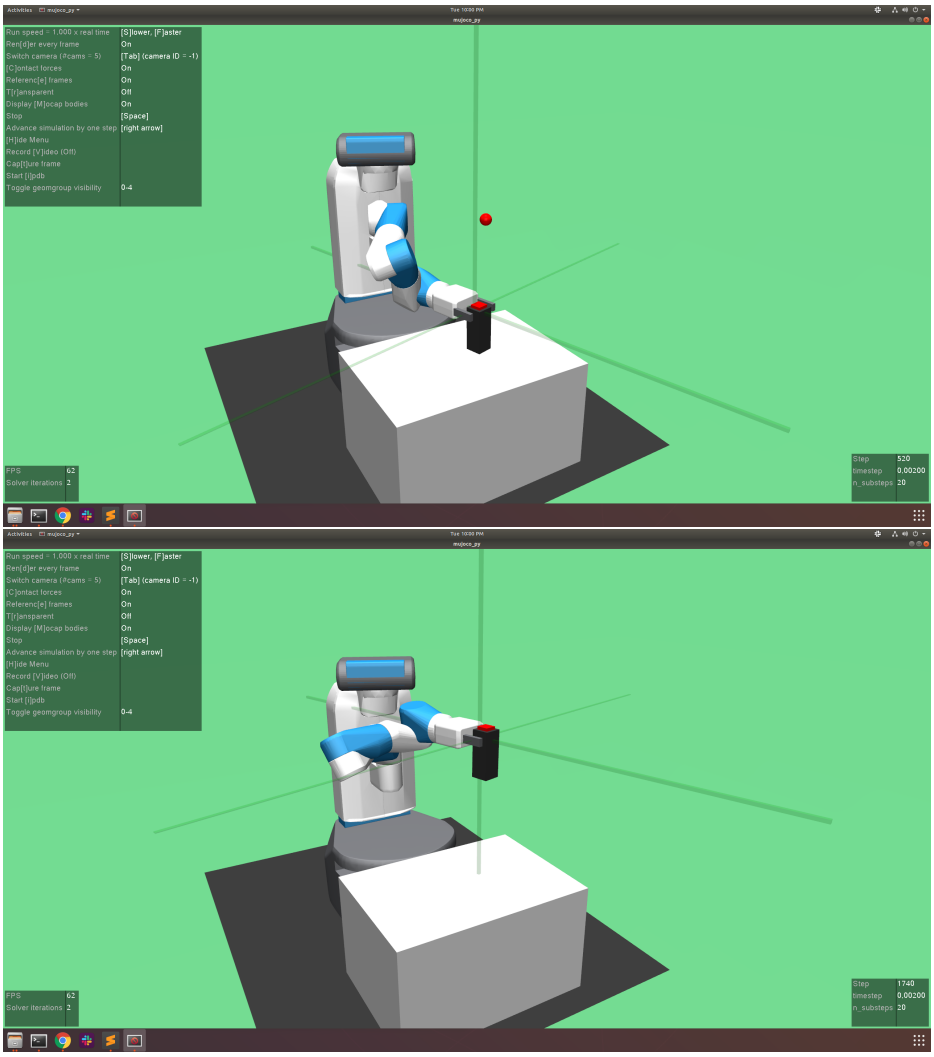

Figure 4: In the rotation modified OpenAI gym, the model must correct for the box not being parallel to the gripper at init, and therefore grasps with a more complex movement. Top: ending of approach, bottom: final movement towards place goal.

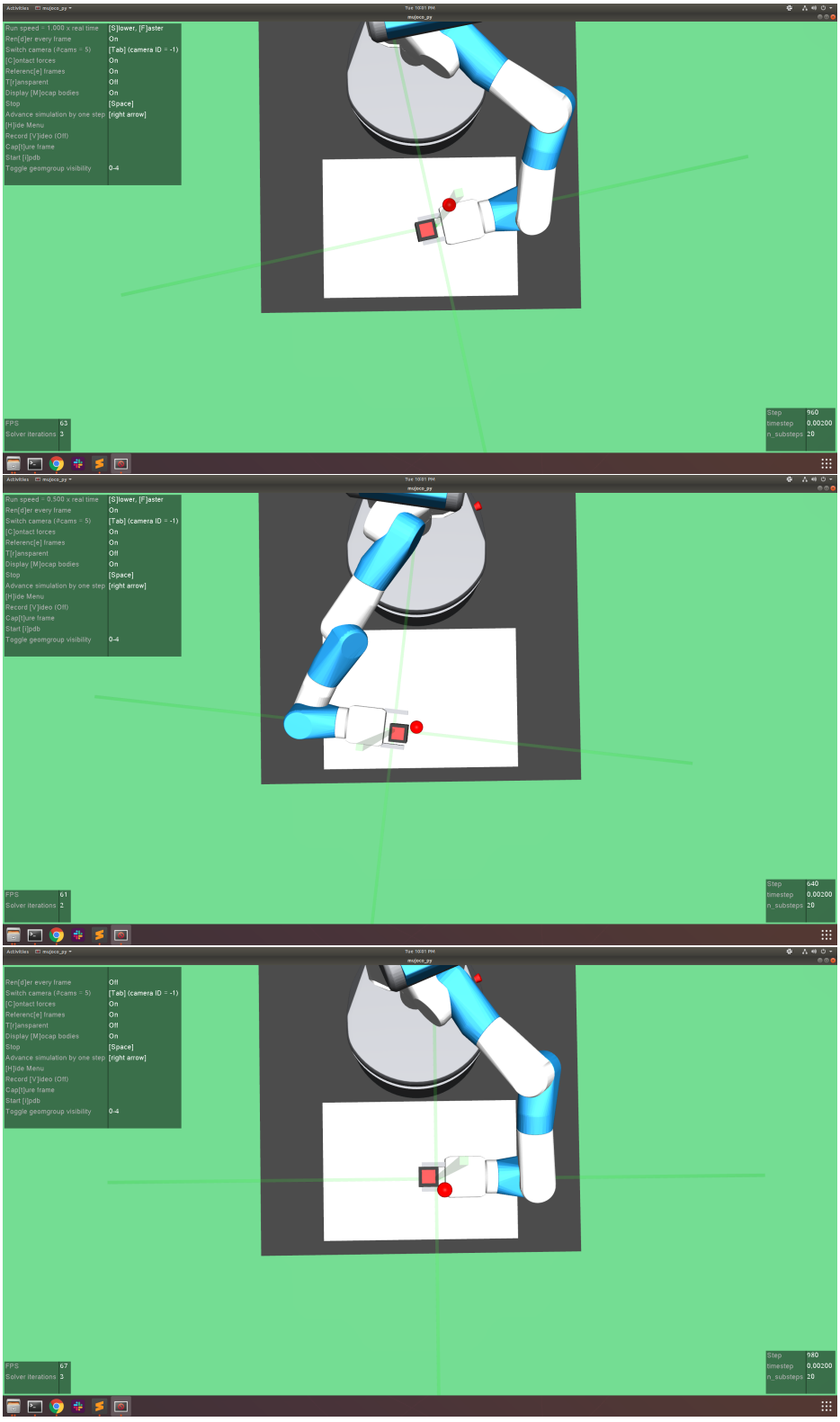

Figure 5: Left-to-right: Three random example rotations and resultant grasps from the modified rotated pillar pick environment.

Figure 6: Our results (see figure 2) are inline with baseline implementations from Nair et al.