# Using Reinforcement Learning to Perform Manipulation Affordance Templates

Adam Pettinger
The University of Texas at Austin
Email: adam.pettinger@utexas.edu

***Abstract*—This project aims to perform contact tasks with a robotic manipulator using reinforcement learning techniques. Instead of learning to perform a task from scratch by rewards issued from environment/task states, the agent is explicitly told how to execute multi-step tasks via *Affordance Templates*. The goal is to make the cost of manually describing the task steps worth it by training an agent that is good at following general directions and avoid needing to retrain on new tasks. The project was more difficult than originally thought, and some paring down of the complexity was required to get reasonable results. In this report, I will present my efforts to accomplish this goal, discuss the results, and conclude with thoughts on interesting directions the work could go in the future.**

## I. Introduction

Being able to use a robotic manipulator to perform numerous unique, low-repetition tasks is paramount for expanding robotics outside of manufacturing and deploying them in a broader set of environments. There is a clear abundance of research dedicated to more complicated manipulation tasks in uncertain environments, and environments where industrial "put a big cage around the robot" safety practices are unacceptable. The research includes classical controls as well as machine learning approaches, see surveys for each respectively [11, 2].

In my work with the Nuclear & Applied Robotics Group at The University of Texas at Austin, we are interested in performing "complex contact tasks" with mobile manipulators. Complex contact tasks refer to tasks that involve manipulating the environment with uncertainty in the exact positioning of the robot or task, and may be unknown (in terms of the task geometry, forces required, and action required) until the mobile base arrives in position to perform the task. Our current work focuses on driving the 2-armed mobile manipulator shown in Figure 1 across a liquid natural gas facility and performing some maintenance tasks in the facility. Example tasks include turning rotary and wheeled valves, pushing buttons, flipping switches, opening doors, and releasing E-Stop buttons. For the time being, all of the operations are completed with a human in the loop, but we are working to push autonomy as far as possible with the eventual goal of just having the human operator make high level decisions about which maintenance tasks to perform and then observe the robot complete them.

To advance the autonomy goals, we are using *Affordance Templates* (AT's) [5, 6]. I will discuss more in Section II, but at a high level they allow the operator to set up a task and tell the robot how to kinematically perform the task. In our project, the current execution of the task is accomplished by sequentially moving to the waypoints given from the AT with MoveIt [1], which uses stochastic planning algorithms and is most useful for large movements in free space. The purpose of the work in this report is to explore alternatives to the execution–using the same waypoint inputs–that might be better for the constrained motion and contact forces of performing tasks. The project focused on using model-free reinforcement learning to train an agent that could perform AT tasks. My results were not perfect, but showed clear promise for applying machine learning approaches to this domain. I think further work to address the simplifications I made as part of the project could result in better than state-of-the-art AT execution.

Fig. 1. A mobile manipulation platform for performing contact tasks

The rest of this report is laid out as follows: Section II will discuss related work, Section III will formalize the problem, Section IV will give implementation details, Section V will discuss the experimentation results, and Section VI will wrap up with conclusions and a discussion of future work.

## II. Related Work

There is a plethora of research using machine learning to perform manipulation tasks. I will stick to model-free reinforcement learning as this is what I used for my project.

I chose to focus on model-free reinforcement learning mostly for its ease of setup. I think there is a strong argument that model-free approaches are sub-optimal for this kind of

problem, and I will discuss more in the conclusion. Primarily, I went with model-free methods (specifically TRPO [10] and SAC [3]) because they were well supported by Stable Baselines [7], which made implementing them quite plug-and-play.

An added benefit of the model-free approach is the ability to change the input to the agent without too much consideration. This allowed me to rapidly try new things and figure out what was going to work the best. Additionally, I really like the idea of using latent spaces or representation spaces besides the raw sensory data from the robot. Although I do not use camera images in this work, I think it is well suited for visual feedback and that using a latent space as in Dreamer [4] would be beneficial. The Dreamer work reduces image streams to a much smaller latent space which it then uses in an actor-critic scheme.

In a similar work, Lee *et. al.* [8] use neural networks to encode a representation space from visual feedback, robot state data, and force/torque (F/T) sensors. The representation mapping is trained using self-supervision and once trained, is used as the input to a model-free reinforcement learning algorithm (TRPO). I like this work because of the inclusion of the F/T data in the multi-modal representation. My previous work [9] with contact tasks has shown that considering the contact forces during task execution improves the task completion rate, so including load cell data is desirable. I especially like the way [8] uses causal convolution on multiple F/T readings spanning some amount of time, as both the raw forces and torques, as well as their change over time is important for controlling contact with the environment.

### A. Affordance Templates

An Affordance Template is semi-autonomous tool for performing robotic tasks. They were first introduced in 2014 as part of the DARPA Robotics Challenge [5, 6] as a way for an operator to control a robot through a task without direct teleoperation in the joint or operation space. A lot of the recent work has focused on the UI to allow users to more easily create ATs–including from teleoperated demonstration–and manually place them with camera overlays and using depth sensor information. At its heart, an AT is a tree of objects and waypoints, as shown in Figure 2. With *a priori* known transformations between the objects and waypoints, the user only needs to know the transform between the current robot position and one of the AT objects to be able to move the robot to each waypoint.

Each set of waypoints forms a trajectory, which is labeled as an ”action” for the robot to do. The idea is the only instruction from the user is to

1) select an AT they want performed
2) set the transformation from robot to the AT (there is no reason this can’t be automated, and doing so will likely be part of my research)
3) select a desired action (AT’s can have multiple actions e.g. pushing vs releasing an E-stop button)
4) hit ”go” and monitor the execution

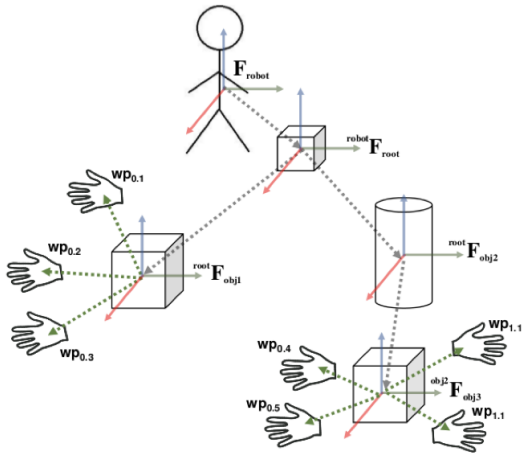


Fig. 2. An Affordance Template is stored as a series of frames and STL objects that allow the user to define waypoint trajectories in the object frames

Currently, users of ATs are mostly using MoveIt [1] to plan and execute between waypoints. This is still being used because it is easy, but there are definite shortcomings in MoveIt’s application to ATs. Firstly, MoveIt assumes there is no contact between the robot and environment, and that reaching the desired position is not prevented by the environment. Secondly, by default there are no guarantees about the continuity of the motion between subsequent waypoints. This means the manipulator may try a large joint reconfiguration partway through the action, which can cause the execution to fail and poses significant safety risks.

For these reasons, I think there is substantial improvement to be made in AT execution. In this project, I explore directly using reinforcement learning as an alternative.

## III. Problem Setup

This section formalizes my problem, starting with a high level overview and including theoretical details as well.

### A. Overview

The framework I used in this project was Robosuite [13] for the physics and simulation environment. The modularity of Robosuite let me try different robot controllers and use the standard environments. I also created a new environment, and was able to easily test different rewards and observations quickly.

For the reinforcement learning component, I went with Stable Baselines [7] for ease of use and because changing algorithms was very easy.

I focused on two tasks with this work: the locked door that comes standard in Robosuite, and a ball valve turn. The valve task required me to create a new environment in Robosuite. For the door task, I copied and modified the default to be able to change the inputs and outputs for testing. To complete the door task, the manipulator was required to turn the handle to unlock the door before pulling it open. The valve task was

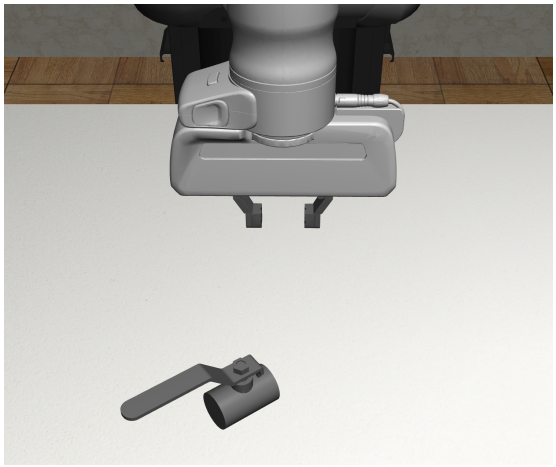

Fig. 3. A ball valve object jointed so the handle can rotate with respect to the main valve body

simpler and required the manipulator to turn the valve handle 90 degrees. Figure 3 shows a close up of the valve.

All of my work was performed with the Panda arm and gripper. The Panda arm was chosen it because it is 7 DoF and looks pretty, although there should be no reason why another arm would not work as well (with potential exceptions for 6 DoF arms).

For most of the project, I used the Operational Space Controller (OSC) in Robosuite. The input to the controller is a pose $\in \mathbb{R}^6$ with 3 positional and 3 rotational arguments that describe a pose change with respect to the current pose. I chose to use the OSC because it is similar to my research outside of this project, and intuitively fits with using waypoints (poses) as reference points. I did briefly experiment with joint velocity control, but ultimately decided against continuing with it. In addition to the arm controller, there is 1 more DoF in the action space for the gripper. I left this as the default, and continuous. The actual implementation seems to be discrete (with any negative number closing the gripper and any positive number opening it), but even the SAC agent with continuous only action space quickly learned to control the gripper.

### B. Task Descriptions

I did not want to go through the process of importing the real AT framework into my project, so I spoofed Affordance Templates for the given tasks. To do so, I described each task as a series of waypoints in the base frame. Each waypoint $\in \mathbb{R}^8$ is a 3-dimension position, a 4-element quaternion orientation, and a gripper position. To find suitable waypoints, I manually entered control commands to the manipulator until it was in a good position, and recorded the result. If I was going to do it again, I would definitely implement the teleoperation with SpaceNav mouse available in Robosuite, as I have a SpaceNav in my apartment... By repeating this process for multiple waypoints, I created a trajectory that would accomplish the task. Directly telling the robot what poses to go to in order to perform a task is an approach I think a lot of the robot learning community would cringe at, but is the essence behind Affordance Templates and is allowing for real-world task completion right now. Additionally, it is not unreasonable that human users will easily have a high-level understanding of the task, bypassing the need to learn some hierarchical structure. For example, the valve task is "naturally" (to me, a human) broken down as:

1) Move to a position above the valve, oriented correctly, with the gripper open
2) Move down to the grasp point
3) Close gripper
4) Turn valve by moving to the "valve closed" position
5) Open gripper
6) Move to a higher position to withdraw the gripper fingers from around the valve

With something like the above list of steps and their corresponding waypoints recorded, all that remains is getting the manipulator to move from one to the next, which I discuss in Section V.

### C. Waypoint Pose Error

This work involves the difference between the robot's current position and the target waypoint. For the gripper, this is trivially

$$\epsilon_{gripper} = q_{goal} - q_{current} \tag{1}$$

where I use $q_{goal}$ and $q_{current}$ as the goal and current gripper positions respectively. As a quick implementation note here, Robosuite reports the gripper position for each finger, so I actually use a $q_{goal}$ and $q_{current}$ for both fingers.

The position error is also easy, and is the Euclidean distance between the target and current positions

$$\epsilon_{pos} = \|\mathbf{x_{goal}} - \mathbf{x_{current}}\|_2 \tag{2}$$

with $\mathbf{x_{goal}}$ and $\mathbf{x_{current}} \in \mathbb{R}^3$ the goal and current position respectively as $[X, Y, Z]$. I will also note that the vector $\mathbf{x_{goal}} - \mathbf{x_{current}}$ gives the delta needed to reach the goal position, and will be important as the input to the reinforcement learning algorithm.

The rotational error is trickier. In the current pose provided by Robosuite, the rotation is ${}_BR_{EE} \in SO(3)$ or the rotation of the end effector (EE) with respect to the base frame (B). When I defined waypoints, I did so with respect to the base frame, so the goal orientations are ${}_BR_{goal}$. Of interest to us is the rotation from the current position to the goal

$${}_{EE}R_{goal} = {}_{EE}R_B \cdot {}_BR_{goal} = {}_BR_{EE}^{-1} \cdot {}_BR_{goal} \tag{3}$$

The above equation gives the rotation required to go from the current orientation to the goal orientation. Each $R \in SO(3)$ is represented as a quaternion with $q = [w, x, y, z] = w + x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}}$. With all rotations as unit quaternions $\|q\| = 1$ the inverse in Equation 3 is

$$q^{-1} = q^* = [w, -x, -y, -z] \tag{4}$$

and we can find the quaternion ${}_{EE}q_{goal}$ that is the rotation needed to reach goal, which is used in the observations. The "closeness" of the orientations can be measured in different ways. Simply, when the target and current orientation are aligned the difference should be the identity quaternion, so you can compare the difference between ${}_{EE}q_{goal}$ and identity with a 2-norm. In this work, I prefer to find the angle of the rotation if it were expressed in axis-angle format. Section IV discusses more, but I also tried using the axis-angle format as the observation. To convert from a quaternion in $[w, x, y, z]$ to axis-angle is

$$\theta = 2cos^{-1}(w) \tag{5}$$

$$\bar{\omega} = \frac{[x, y, z]}{\sqrt{1 - w^2}} \tag{6}$$

The $\theta$ above is used when comparing how close 2 rotations are to each other with $\theta = 0$ being perfectly aligned. $\bar{\omega}$ is the axis of rotation from axis-angle format.

## IV. Implementation Details

In this section, I will discuss the implementation of my project. This includes the details of what I did and a healthy dose of experimentation. The experimentation in this section will be more of what I tried in order to get the algorithm to behave, and Section V will be about the testing I did with the final implementation.

To start I decided to use the default locked door environment with no modifications because it is in the Robosuite benchmarks[1] using SAC. I thought it would be useful to compare to a Stable Baselines implementation run on my computer with the benchmarking runs using RLKit. I used the SAC hyperparameters from the benchmark (see table). Figure 4 compares the episode return during training for the runs. Note that I used the exact default for my run, so with Panda joint velocity controllers, shown in the green curve in the comparison.

| SAC Hyperparameter | Value |
|---|---|
| discount factor $\gamma$ | 0.99 |
| learning rate | 0.0008 |
| buffer size | 500,000 |
| learning starts | 3,300 |
| training frequency | 2 |
| batch size | 128 |
| soft update $\tau$ | 0.005 |
| target update interval | 5 |
| entropy coefficient | "auto" |

From Figure 4, the episode return after training is similiar between my run and the baseline. The difference however is is the benchmark was run for 500 epochs at 500 steps each $=$ 250K timesteps, and mine was run for 1500K timesteps (with episodes of 1000 steps). So while the end result was similar, my run took longer to reach it. I partially attribute this to the difference in epoch length as empirically there is a lot of "dead time" at the end episodes where the manipulator moves to a random position and moves very little afterward. I also attribute this to the difference in SAC implementations. I did my best to match the hyperparameters, but they weren't exactly the same set and some where labeled differently. I have included a video of a run of my agent performing the door task, where you can see that it unlocks the door but struggles to pull it open.

[1]at https://github.com/ARISE-Initiative/robosuite-benchmark

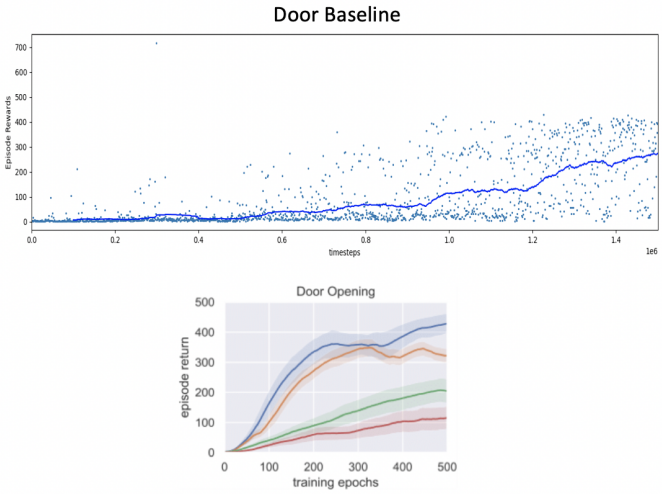


Fig. 4. Running the standard locked door environment (top) vs the results reported in the benchmark (bottom, green curve)

### A. Modified Environments

Convinced that I could get similar results to the benchmark, I moved on to modifying the door environment and creating the valve environment. With the standard issue locked door, the observation includes direct information about the environment such as the current position of the door and handle, the hinge angle of the door, and the displacement from the gripper to the door handle. The reward directly uses the distance to door handle (as in Equation 2) and the angle of the handle. My goal was to remove the environment-specific pieces and instead use target waypoints. Doing so allowed me to actually create a single environment for training both (or $>$ 2 in the future) tasks, as really the agent was learning to move to a specific waypoint.

As the observation and input to the reinforcement learning algorithm, I used

- Proprioceptive robot state including joint, EE, and gripper positions and velocities
- The current waypoint target pose, with the 3 DoF position, 1 DoF gripper goal, and orientation in either quaternion or axis-angle form, as discussed later.
- The difference between the goal waypoint and current position, as discussed in Section III.

As the reward function, I used 3 separate hyperbolic tangents for the position, orientation, and gripper, with the idea

to do a weighted sum of them for the final reward.

$$r_{gripper} = 1 - tanh(\alpha \epsilon_{gripper}) \tag{7}$$

$$r_{gripper} = 1 - tanh(\beta \epsilon_{pos}) \tag{8}$$

$$r_{orient} = 1 - tanh(\gamma \theta) \tag{9}$$

with $\epsilon_{gripper}$, $\epsilon_{pos}$, and $\theta$ defined in Equations 1, 2, 5 respectively, and $\alpha$, $\beta$, $\gamma$ as scaling constants for the $tanh$ functions.

### B. Training Individual Actions

In order to verify different parts of the project were working correctly and get an idea about what $[\alpha, \beta, \gamma]$ to use, I decided to test each reward function separately before combining them.

I started with the gripper because it was very simple. I treated the gripper as a discrete problem with "open" or "close" positions, but nothing in between. The goal positions of the gripper are given in the table

| Gripper State | Open | Close |
|---|---|---|
| Position (m) | 0.04 | 0.0 |

To train the gripper action, I simply used only the gripper reward from Equation 7, and during each episode randomly selected the goal state. As expected, the training of the agent using only the gripper reward was easy, as shown in Figure 5. Good results where achieved in under 50K timesteps. I use a scaling factor of $\alpha = 75$, but empirically I found that values above 25 all worked well.

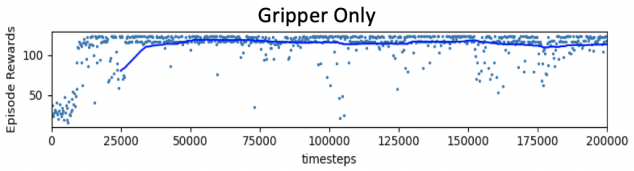


Fig. 5. Training only the gripper action

Next, I wanted to run the position only reward. I used only the reward in Equation 8 and trained the agent. In this case, I picked random positions as the goal points, but I made sure to pick positions near the manipulation space of the door and valve tasks. I think in future work, it would be beneficial to do longer training runs with positions from all over the reachable workspace in order to better generalize to new tasks. See the table for the trained workspace. In Figure 6, I show the rewards over a training run of 1000K timesteps, and the results look promising. For position, I used scaling factor $\beta = 20$.

| Coordinate | Min bound (m) | Max bound (m) |
|---|---|---|
| X | -0.14 | 0.03 |
| Y | -0.25 | 0.1 |
| Z | 0.88 | 1.16 |

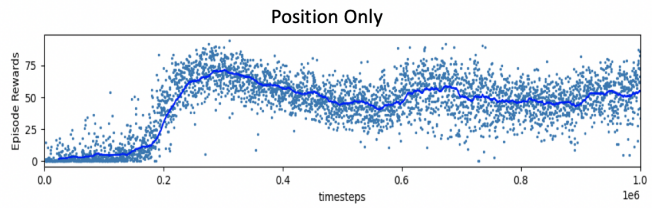


Fig. 6. Training with only the position reward

The tough one was the orientation. I initially tried to use the axis-angle format for the observation space because intuitively this maps well to the 3 DoF orientation input to the OSC controller. After struggling to get any good results, I also tried the quaternion format with no success. Figure 7 shows that neither orientation representation ever learned a good policy.

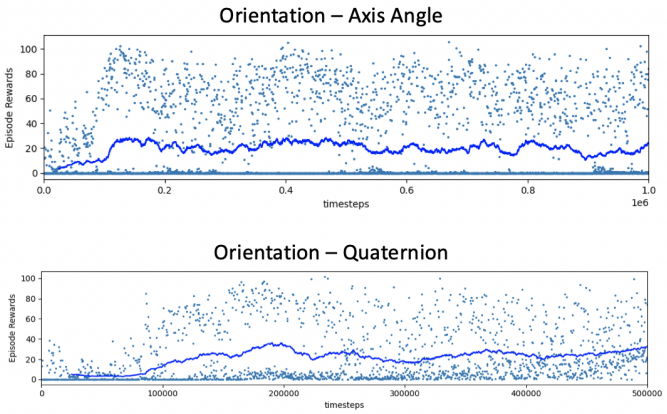


Fig. 7. Training the orientation with axis-angle representation (top) and quaternion representation (bottom)

I will note that in both parts of Figure 7, there is a pretty substantial split between episodes that got basically 0 reward, and episodes that got well above the mean. This is because I made sure that some goal orientations were close to the waypoints for the valve and door tasks, and the robot starting position was close to the first waypoint. So the episodes with high rewards were moving to a nearby orientation, which shows some promise but the inability to generalize to further away rotations.

Because of the poor performance of the orientation only training, I decided to move forward without considering the orientation. I think future work definitely should include getting good rotational results, perhaps following a 5th or 6th order rotation representation with continuous mappings between the representation and SO(3) rotation, as in [12].

### C. Combining Rewards

With the position and gripper actions successfully trained individually, it was time to combine them. I first attempted to average the gripper and position rewards that had previously

worked as

$$r = \frac{(1 - tanh(75\epsilon_{gripper})) + (1 - tanh(20\epsilon_{pos}))}{2} \quad (10)$$

Confidently, I queued up a big 1500K timestep training run and let it go, the results shown in Figure 8. As you can see, there was a brief jump in rewards at the start and then almost no change for the rest of the training. This figure looks very similar to Figure 5, and in fact an episode return of 250 is about half of the maximum for this setup. Upon further inspection, this was only really doing the gripper, with no effort made to close the distance to the target waypoint.

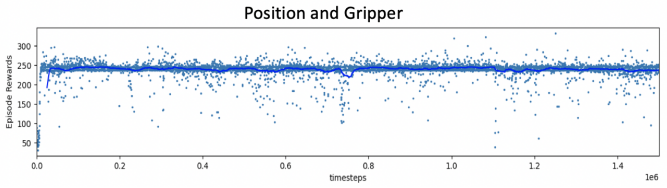


Fig. 8. Training with combined gripper and position rewards

To combat this, I decided to do 2 things. First, I reduced the scaling factor for the position reward to be $\beta = 10$, so that the derivative while very close to 0 reward was more distinct. I believe the noise in the gripper measurements was washing out the derivative signal in the position reward when it was small. Secondly, I reduced the overall weight of the gripper reward compared to the position by making the reward function

$$r = 0.25(1 - tanh(75\epsilon_{gripper})) + (1 - tanh(10\epsilon_{pos})) \quad (11)$$

I trained again with the new reward and got the results shown in Figure 9. These results look better, and on inspection the agent was moving to the correct position and controlling the gripper simultaneously. One note for the scale of the rewards in Figure 9 is that I was starting to test multi-waypoint rewards, and so divided the reward by the number of waypoints (3). Thus, the maximum reward in this case is around 155, which means the agent here is doing well as hitting 155 assumes the manipulator moves to the goal instantly and stays there for the episode.

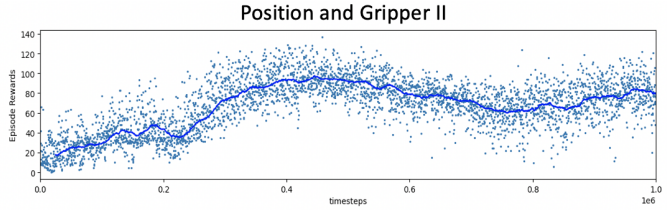


Fig. 9. Training with new gripper and position rewards

The agent trained during Figure 9 is the agent I used in the following section with no additional training. The intent was always to avoid retraining for individual tasks, so this was an effort at accomplishing that goal.

## V. Experimental Results

With the agent from the previous section trained, I next needed to create the validation environments. This is where the valve and door tasks diverged as they were separate tasks with their own waypoint descriptions. The difference between them is slight, however, just the varying waypoint targets and the correctly loaded and placed object of interest.

Next, I needed a way to sequentially move from waypoint to waypoint. In this case, I decided when change targets with the simple heuristic approach of allowing a transition when the manipulator was close enough to the target waypoint and had stopped moving. Thus, 2 parameters determined the transitions: the distance to target, and the allowable joint change between timesteps (as the 2-norm of the joint position change). For the project, I used the minimum distance as 5cm and the allowable joint change as 0.005 radians. For a variety of reasons, I think sequencing through waypoints while training the agent would have been beneficial. One of the reasons is that the agent could be trained to decide when it had satisfied a waypoint by itself, but that is left to future work.

Before I present the results of the trial tasks, I will briefly discuss the orientation again. The agent used does not include the orientation reward. However, it still has access to all 6 DoF of the OSC input. To some degree, the agent learned it could rotate the end effector to help achieve the desired position, but the orientation is clearly not maintained nor does it move to the rotational goal. This makes the tasks harder of course, as it would be nice to meet specific rotational goals while rotating the valve and the door handle. Additionally, the Panda gripper is fairly wide, so gripping the door handle can pose challenges under certain orientations of the grasp. Interestingly, intervening in the action to set the orientation input to the OSC to 0 does not effect the agent's performance much (but of course causes the orientation of the end effector to remain constant). I don't have quantitative results from doing this, but in the few runs I did the agent seemed to perform comparably to the un-meddled-with action.

### A. Task Results

In addition to grasping and turning the valve, I created another task with the valve: closing the gripper and using the fingertips to push the valve through an arc. As you may expect, this turned out to be much more forgiving of the lack of orientation control and performed much better than the grasp and turn.

Overall, both valve tasks seemed at least possible to complete. During the door task, the robot really struggled to effectively grasp the handle because the body of the gripper collided with the door, preventing the fingers from wrapping around the handle. That said, the door task looked similar to the baseline I started with. The valve push was quite effective at rotating the valve 90 degrees, while the grasp and turn struggled overall but succeeded in a few trials. Videos of all 3 tasks are included in the supplemental material. The below table shows the episodic return from trying each task 20 times.

| Task | Return Avg | Return Std |
|---|---|---|
| Door | 116.50 | 20.42 |
| Valve Grasp | 32.71 | 9.71 |
| Valve Push | 168.36 | 11.80 |

The tasks were not as successful as I had hoped, and Section VI discusses potential improvements in detail. From the table, you can clearly see the valve push task outperformed the others, while the grasp and turn task struggled more than the door.

## VI. Conclusion and Future Work

This is the most complex reinforcement learning challenge I have done to date, and it really opened my eyes to the difficulties in using machine learning with robotics. I ended up removing or simplifying large components of the project, namely controlling the orientation and using the F/T data in a latent space. I know that lacking in these areas caused the task performance to suffer, and I think with some fixes the method could be quite good at completing the given tasks.

Besides the orientation problem, learning from contact with the environment is something I wish I had done better. This includes using the F/T data, but also interacting better with the environment during training. During training, I made sure to include regions that would bring the robot into contact with the valve or table, but these weren't very intelligent. For instance, pushing against the table while trying to reach a point under it is not similar to the contact required to smoothly turn the valve or door handle. Additionally, the motion to turn the valve is different than required to open a sliding drawer, pick an object up, etc. Even when trying to train a generalized agent to perform a wide variety of tasks, I think there is benefit in trying to train with a lot of different task-like environmental interactions. Something that could have helped is training multiple sequential waypoints, and the transitions between them. This starts to get close to training for individual tasks, but having the agent try to move to specific waypoints while grasping an object could have helped.

Stepping back from the details, I think it is reasonable to question whether or not a model-free approach is the correct way to accomplish AT execution. Classical controls approaches exist to move a manipulator to a specific pose (in fact Robosuite's OSC is one such implementation). A very good method may be to use such a control law, with a higher level machine learning algorithm to set the impedance and motion parameters of the controller. This would likely result in a simpler system overall, that could be safer and faster to learn: perfect qualities for deploying on real hardware.

The work presented in this report represents a fairly vanilla reinforcement learning approach to solve the AT execution problem, but it was always my intent to use this project to explore machine learning approaches in this area and this project was helpful in doing so.

## References


[1] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[2] Bin Fang, Shidong Jia, Di Guo, Muhua Xu, Shuhuan Wen, and Fuchun Sun. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, pages 1–8, 2019.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[4] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[5] Stephen Hart, Paul Dinh, and Kim Hambuchen. Affordance templates for shared robot control. 2014.

[6] Stephen Hart, Paul Dinh, and Kimberly Hambuchen. The affordance template ros package for robot task programming. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6227–6234. IEEE, 2015.

[7] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[8] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

[9] Adam Pettinger, Cassidy Elliott, Pete Fan, and Mitch Pryor. Reducing the teleoperator's cognitive burden for complex contact tasks using affordance primitives. In *International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[11] Peng Song, Yueqing Yu, and Xuping Zhang. A tutorial survey and comparison of impedance control on robotic manipulation. *Robotica*, 37(5):801–836, 2019.

[12] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks, 2020.

[13] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.