Robot Learning in Simulation for Grasping and Manipulation

Beatrice Liang
Department of Computer Science
Columbia University
Email: bsl2127@columbia.edu

***Abstract*—Teaching a robot to acquire complex motor skills in complicated environments is one of the most ambitious problems facing roboticists today. Grasp planning is a subset of this problem which can be solved through complex geometric and physical analysis or computationally expensive data driven analysis. As grasping problems become more difficult, building analytical models becomes challenging. Consequently, we aim to learn a grasping policy through a simulation-based data driven approach. In this paper, we create and execute tests to evaluate simulator's suitability for manipulating objects in highly constrained settings. We investigate methods for creating forward models of a robot's dynamics, and apply a model free reinforcement learning approach with the goal of developing a grasping policy based solely on proprioception.**

I. Introduction

Expanding capabilities and complexity of robots has led to a greater need to explore adaptive control. Creating analytical models of has proven to be difficult; hand tuning policies is not viable due to the intricacy of environments. Creating an analytical model to explore the complex physical interaction between a robot and the space of variable environments is especially difficult. As a result, researchers are increasingly looking towards data driven approaches. However, collecting a sufficient amount data to produce an unbiased model of the system's dynamics or to produce a robust policy for interacting with complex environments on a physical robot proves to be a limiting factor.

Simulators can provide efficiency that a physical robot cannot afford. When developing control methods, either through model based design or reinforcement learning, researchers need to collect large and comprehensive datasets. The robot needs to repeatedly perform a task or some variation of the same task to collect data. Before each iteration of task, the researcher has to set or reset the task scene. If researchers were to use a physical robot to collect data, the initialization of the scene must be performed manually, and the scene could not be precisely reproduced. Using simulation, the initialization of each scene is easily automated and can be precisely reproduced. Additionally, with a physical robot, collecting thousands of data points is either prohibitively expensive or time intensive. We would have to choose either for one robot to collect all of the data, or for multiple robots to simultaneously collect the data. Moreover, with a physical robot, we need to consider the physical limitations such as failing hardware. In simulation, we can explore more of the state space with relative ease and in a shorter amount of time, and data collection is easily parallelizable.

For a simulator to be considered valuable for creating grasping policies, it must have robust contact generation under many constraints. Without robust contact generation, the simulated objects can exhibit artifacts such as jitter, divergence, or phantom impulses as described in [1]. We need to ensure that we can achieve stable equilibrium under applied frictional and normal forces. In this project, we develop and execute tests of increasing complexity to evaluate the robustness of the simulator. The simulator must allow for the robot to have articulations with externally applied joint torques and position control, so we can evaluate a variety of robots and robot control methods. For this project, we collect joint position, motor position, and joint torque data to derive a grasping policy, therefore the simulator must also implement joint position and joint torque sensors. Other simulator requirements we have for the task of grasping include non-dynamic objects, such as floors or walls, and general mesh objects.

Once we establish that the simulator is suitable for our task, we embark on developing a policy to grasp and manipulate an object. We explore the creation of dynamics models built by data-driven analysis, and apply a model free reinforcement learning method to develop a grasping policy. We aim to learn a control policy that relies solely on proprioception, where the input is composed of only of features the robot itself can feel, such as the joint positions, joint torques, motor positions, and joint velocities of a robots.

II. Related Work

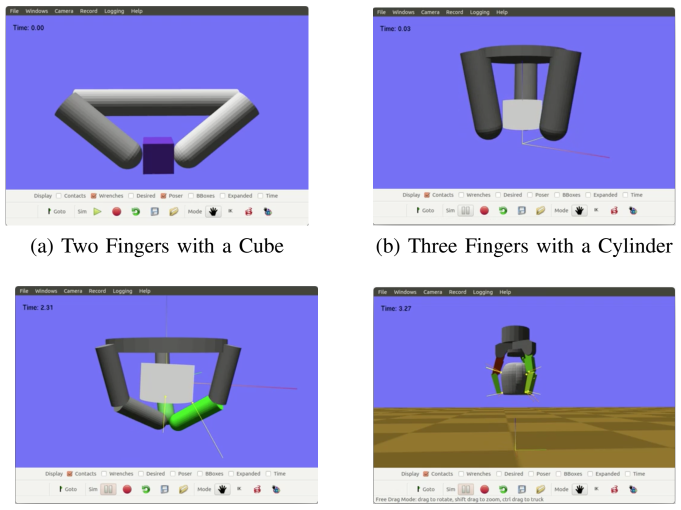
Reinforcement Learning has been used successfully in various tasks including navigating through complex environments and manipulating objects. Heess et. al deployed a large scale deep reinforcement learning algorithm [2] based on [3] to teach different torque controlled bodies to steer themselves through a terrain with various obstacles such as hurdles, gaps, variable terrain, walls to go around, and platforms in simulation. Gu et. al train local linear models to reduce the sample complexity required by model free reinforcement learning and accelerate learning in [4] ot apply the reinforcement learning method of real robots. They also extended the work to add synthetic on-policy examples, generated by the local models, which augment the samples from which the algorithm can learn. They later show in [5] that the deep reinforcement learning algorithm from [4] can be used to learn how to manipulate objects by using it to learn how to open a door. Boularias, Bagnell, and Stentz in [6] tackle the problem of grasping from a dense clutter by combining a reinforcement learning framework for object manipulation and object detection using computer vision. Levine et. al developed a grasping method that trains a large convolutional neural network to predicts grasp success from visual input and uses the network to servo the gripper in real time to grasp objects [7]. In [8], Peng et. al use a policy gradient method to train a policy in simulation for a pushing task where the state space is composed of joint positions and velocities of the robot's arm, the position of the robot's gripper, and the object's position, orientation, linear and angular velocities. They randomized the dynamics parameters of the simulation and transferred the learned policy to the real world robot.

Others have been able to evaluate grasps and perform grasps without using visual servoing. In [9], Hao, Weisz, and Allen trained a Support Vector Machine using hand kinematic and simulated tactile feedback data to estimate the stability of a given robotic grasp. Madry et. al in [10] extract features from sequences of tactile data in a process called Spatio-Temporal Hierarchical Matching Pursuit to add an additional time dimension to the data. Chebotar et. al use the spatio-temporal tactile features introduced to identify failed grasps and adjust the grasp using the feedback if the grasp is unstable [11]. Chen and Ciocarlie show in [12] that joint position and torque sensing is sufficient for a multi-input multi-output proportional-integral controller to produce stable finger tip grasps.

We seek to learn a grasping policy solely using proprioceptive features of joint position, joint torque, motor position, and joint velocity, without information external to the robot such as object detection using computer vision and tactile sensors. We develop models of the dynamics grasping that that map current state (joint position, joint torque, motor position, joint velocity) and action (motor position command), to subsequent states. We also implement and apply Proximal Policy Optimization, a model-free Policy Gradient Reinforcement Learning method introduced in [3], to develop a grasping policy based solely on proprioception.

III. Testing Simulators

The most important tool for this project is a robust simulator that can simulate articulated joints, non-dynamic objects, and prescribed base motion for the robot. The primary requirement for a simulator is its ability to maintain stable equilibrium under applied forces for an extended period of time. We evaluated the simulator Klamp't [13], which uses Open Dynamics Engine (ODE) [14] as its backend. ODE uses an iterative Linear Complementarity Problem solver to solve contact constraints, to make certain that interpenetration does not occur. However, in highly constrained problems, the solver is not guaranteed to find a solution to the contact generation problem and will simply error out. Klamp't attempts to avoid this issue by adding a thin boundary layer over the underlying meshes as described in [1]. This method helps the simulator steer clear of discontinuities in contact generation, caused by collision. We sought to verify that the simulator robustly produces contacts.



(a) Two Fingers with a Cube

(b) Three Fingers with a Cylinder

(c) Multi-Joint Fingers with a Cylinder

(d) Barrett Hand with a Mug

Fig. 1. Each of these tested configurations were able to hold on to the object for at least a few minutes

We test the robustness of contact generation of the simulator by creating various robotic configurations of increasing complexity, as seen in Fig. 1. With the increasing complexity, we add more constraints to the system for the simulator to solve. If the configuration is has too many hard constraints for the simulator to solve, then the simulation could produce unrealistic simulation artifacts or simply fail.

A. Friction at Contacts

We evaluate the friction at the contacts to assess the contact dynamics. We are interested in determining whether the simulator would model friction correctly via critical torque. We define critical torque as the minimum torque at which an object of a specified mass can be held in equilibrium by equally and symmetrically spaced fingers given a specific configuration and coefficient of friction. Critical torque τ is given by

$$\tau = \tau_f + \tau_b \tag{1}$$

$$\tau_f = \frac{m_f g R}{2}\cos(\theta) \tag{2}$$

$$\tau_b = \frac{m_b g R}{n}\left(\cos(\theta) + \frac{\sin(\theta)}{\mu}\right) \tag{3}$$

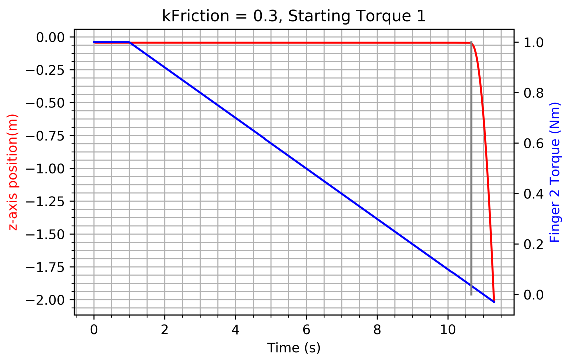


Fig. 2. An example of a plot produced by decreasing torque. The gray line denotes the point at which the block began to slip

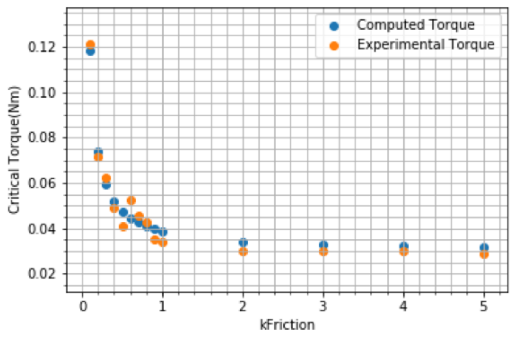


Fig. 3. Critical Torque vs. Coefficient of Friction, comparing experimental data averaged over 3 trials and the theoretical computed critical torque

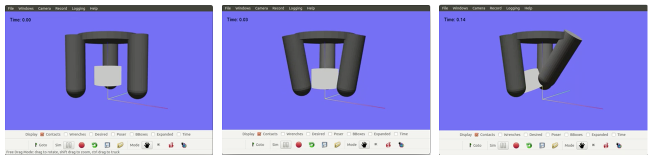


Fig. 4. Three Finger Asymmetry

where

$$
\begin{aligned}
\tau_f &= \text{minimum torque to hold the finger at angle } \theta \\
\tau_b &= \text{minimum torque to prevent the object from slipping} \\
m_f &= \text{mass of the finger} \\
m_b &= \text{mass of the object} \\
g &= \text{gravitational acceleration} \\
R &= \text{distance between joint and point of contact with object} \\
\theta &= \text{joint angle of the fingers} \\
\mu &= \text{friction coefficient} \\
n &= \text{number of fingers}
\end{aligned}
$$

We simulate the configuration at Fig. 1a using torque control, apply the same magnitude torque to both fingers, and decrease the torque at the joints over time until the block slips out of the robot's grasp. We plot the block's vertical position and the applied torque over time to determine the torque at which the block began to slip, as seen in Fig. 2. If the position of the block changes by more than a millimeter in a tenth of a second, we consider the corresponding torque to be the critical torque for a specific coefficient of friction.

We repeat this procedure for multiple coefficients and initial block positions to see if the simulator agreed with the critical torques we computed using (1). We found that the simulator produced physically realistic results (Fig. 3) for the two fingered model.

B. Asymmetry

When we increase complexity by adding a third finger, simulating Fig. 1b, we encounter asymmetry in the grasp, despite applying the same magnitude of torque at each of the joints. This asymmetry (Fig. 4) is a cause for concern as it may indicate there are undesired artifacts in the simulation.

We need to establish whether the configuration itself is in stable equilibrium or not. A system is in stable equilibrium if the second derivative of the potential energy is greater than zero. This means that if the system is disturbed, it will tend to return to its original position. We simplify the problem to a two dimensional problem, focusing on position disturbances of the block on the x-y plane. We compute the potential energy U as a function of displacement $\vec{x}$ from the central axis of the palm. See Appendix A for details. We then plot the potential energy against displacement $\vec{x}$ from the central axis of the palm. In Fig. 5, we can see that Potential Energy vs $\vec{x}$ is concave down, meaning that the second derivative of the potential energy is not greater than zero. The system is not in stable equilibrium. Since the system is not in stable equilibrium, any small disturbances in position could displace the block and cause dramatic asymmetry. However, in this simulation, we ensured that the forces applied to each joint was the same and that the fingers were all oriented the same way. The block is not displaced from the central axis of the base of the gripper, so the block is in unstable equilibrium. Without any external forces or other instability, the asymmetry should not have occurred.

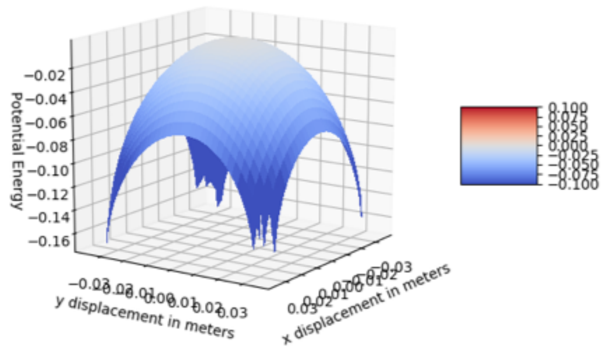


Fig. 5. Potential Energy vs. 2D Displacement of the Three Finger Configuration

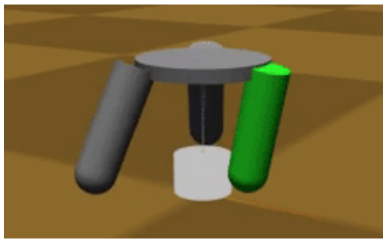
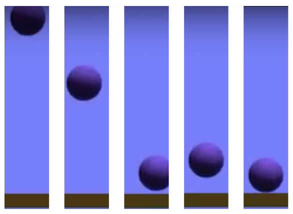
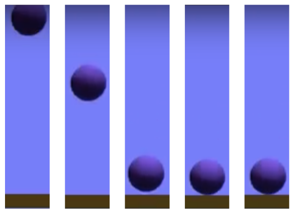


Fig. 6. Gripper fingers bounced off of the block even though the Coefficient of Restitution = 0, the applied forces are all equal, and the fingers are oriented the same way.



(a) Stiffness Coefficient=∞ and Damping Coefficient=∞



(b) Stiffness Coefficient=800,000 Damping Coefficient = 200,000

Fig. 7. These are screenshots over time of identical simulations of a sphere colliding with a plane differing only in the stiffness and damping coefficients. The Coefficient of Restitution is 0, so the ball should not bounce.

The asymmetry is still unexplained until we look into another simulation artifact, bouncing due to numerical instability. The fingers bounce off the block unevenly despite the Coefficient of Restitution being 0 and the applied torques being identical, as in Fig. 6, causing the block to be displaced from the center of the 3 fingers. As a result the system was no longer in unstable equilibrium.

C. Numerical Instability

We find that when the stiffness and damping coefficients are set to infinity and the Coefficient of Restitution is 0, the simulation becomes unstable, like the asymmetry in Fig. 4 and Fig. 6. However, when we reduce the stiffness coefficient and damping coefficient, this asymmetry was resolved. This is due to instability in the collision handling in Open Dynamics Engine, as shown by Fig. 7. In ODE, there are parameters called Constraint Force Mixing (CFM) and Error Reduction Parameter (ERP) for each contact. The amount a constraint is allowed to be violated is proportional CFM times the restoring force needed to enforce that constraint. Thus, when CFM = 0, the constraints are hard and when CFM is larger, the constraints are softer and the collisions become spongier. Another important feature of CFM is that greater CFM leads to fewer numerical errors.

ERP defines the proportion of error that will be corrected in the next time step for a contact. Ideally, ERP lies between 0.1 and 0.8, so that some errors are corrected. Typically, ERP is not should not be set to 1 due to ODE's inability to resolve all errors because of numerical approximations. If ERP is too small, then the simulator will accumulate an abundance of errors.

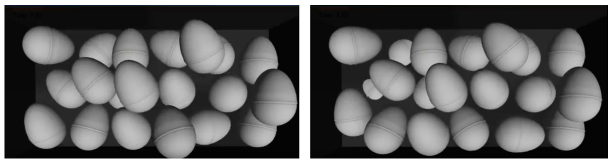


Fig. 8. A simulated box of 20 eggs over time where the coefficient of stiffness = 800,000 and the coefficent of damping = 200,000

CFM and ERP are defined as follows:

$$ERP = \frac{hk_p}{hk_p + k_d} \tag{4}$$

$$CFM = \frac{1}{hk_p + k_d} \tag{5}$$

where

$$h = \text{the stepsize}$$
$$k_p = \text{the stiffness coefficient}$$
$$k_d = \text{the damping coefficient}$$

We can compare the CFM and ERP between Fig. 7a and Fig. 7b to demonstrate why the simulation in Fig. 7a had noticeable numerical instability, while the simulation in Fig. 7b did not. In simulation, infinity is approximated by a large real number. So, in the case where $k_p = \infty$ and $k_d = \infty$, $k_p = k_d$ where k_p is represented by a large number. In both simulations in Fig. 7 we used a time step $h = 0.03$. As a result,

$$ERP_{7a} = \frac{hk_p}{hk_p + k_p} = \frac{h}{h+1} \approx 0.029 \tag{6}$$

$$CFM_{7a} = \frac{1}{hk_p + k_p} \approx 0 \tag{7}$$

In the case where $k_p = 800,000$ and $k_d = 200,000$

$$ERP_{7b} = \frac{800000h}{800000h + 200000} = \frac{4h}{4h+1} \approx 0.107 \tag{8}$$

$$CFM_{7b} = \frac{1}{800000h + 200000} = 4.46 \times 10^{-6} \tag{9}$$

Comparing equations 6 and 8, we can see that changing k_p to 800,000 and k_d to 200,000 increased ERP. Likewise, comparing equations 7 and 9, we know that we have increased CFM as well. By increasing the CFM, we can increase the amount the constraints can be violated and reduce the numerical errors in the simulation. By increasing the ERP when we change the damping and stiffness coefficients, more of error is corrected. Softened constraints lead to more stable simulations.

D. Interpenetration

By softening the constraints, the simulations overall become more stable than with hard constraints. However, with softer constraints, other simulation artifacts such as interpenetration will occur in highly constrained scenarios. For example, when we simulate clutter, we discover numerical instability. We find that when the constraints are soft and objects are stacked on top of each other and confined in a box, there is a tendency for the objects to sink through the bottom of the box, as seen in Fig. 8. This behavior does not occur when there are no eggs placed on top of one another. If we reduced the number of eggs such that the eggs could distribute themselves on the base of the box without constant contact with each other, then the simulation is stable and does not feature sinking or interpenetration. Having soft constraints allows for the constraints to be violated by a certain extent, and constant forces encouraging two objects that are persistently in contact to get closer together exacerbates interpenetration.

On the other hand, if the constraints are hard, the objects jitter and pop instead of exhibiting interpenetration and sinking behavior. After a few minutes, the numerical instability accumulates and the eggs explode out of the box. As explored in the previous section, hard constraints result in numerical instability. In highly constrained simulations, collisions are numerically complex to solve and will eventually become unstable.

E. Comparison with Other Simulators

Despite the inaccuracies in the clutter simulation, we still believe that Klamp't is an appropriate open source simulator for this project. Other open source physics engines that we explored, Bullet Physics [15], Moby [16] and GraspIt! [17], were unable to produce simulations that were as physically accurate as Klamp't. Bullet Physics was unable to simulate friction properly; the critical torque was frequently too high given a specified coefficient of friction. Moby and GraspIt! failed after simulating a gripper holding an object for a few seconds of simulation time due to increasing interpenetration between the gripper's links and the object. Klamp't was able to produce accurate contact dynamics and stably simulated a gripper holding an object for a few minutes. Furthermore, we require several other features which Klamp't is able to suitably provide, such as simulated contact sensors, joint position sensors, prescribed base motion for a gripper to lift objects off a surface. In this project we will use Klamp't, since it is the most robust simulator we have examined.

IV. Learning Forward Dynamics Models

We aim to learn a policy to grasp an object solely on the proprioceptive features of the Series-Elastic-Actuated (SEA) gripper, where the physical robot is described in [12]. The SEA gripper, Fig. 9, is composed of two fingers which each have two links, one distal and one proximal. For model based approaches to learn control policies, we need to develop a forward model of the system that maps current state and action to the next state. We can use the learned forward model to develop a policy that finds the action a_t, for a given state s_t, that maximizes the reward of the subsequent state s_{t+1}. In the case of the SEA gripper, the state is composed of joint position, joint torque, motor position, and motor command for each joint and the action is motor command. We want to predict the next joint position, joint torque, motor position, and motor command for each joint given the current state and a motor command.

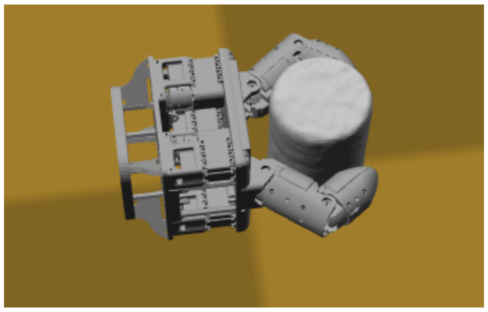


Fig. 9. Series Elastic Actuated Gripper gripping a cylinder

A. Data Collection

We collected state and action pairs for the purpose of creating a forward model by repeating the following procedure 900 times in Klamp't and storing the state at 100Hz in simulated time. We first randomly placed a random object between the fingers of the gripper and assigned random masses and coefficients of friction to the object. The objects were either from the YCB object set [18] or were a randomly generated ellipsoid, box, or cylinder. Next, we command the fingers to a fully closed position until the velocity of the fingers is less than 0, so we know that the fingers are touching the object. This is the initial touch position. We select one of six types of commands: commanding all the joints to random positions, commanding one joint to a random position at a time, commanding the proximal joints to the same random position so that they directly oppose each other and squeeze the object, commanding the distal joints to the same random position so that they directly oppose each other and squeeze the object, commanding the proximal joints to the opposite positions so that they move parallel each other and move the object side to side, or commanding the distal joints to the opposite positions so that they move parallel each other and move the object side to side. We also choose whether the joints return to their initial touch position every few time steps or not for a total of 12 types of motion. We terminated the data collection when the object was no longer in reach of the object or after simulating 200 seconds. Additionally, we collected an hour of simulated data at 100 Hz where there was no object between the fingers of the gripper and commanded a random number of joints to random positions.

B. Forward Model Results

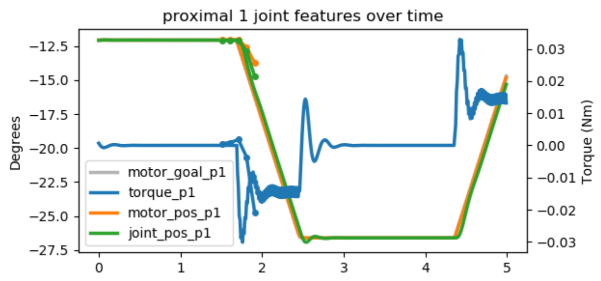
In this section, we evaluate two regression methods, Linear Ridge Regression and a Deep Neural Network, trained on data where there was no object between the fingers of the gripper. When there is an object between the fingers, we do not know the contacts between the robot and the object nor do we know the location of the object. These features can greatly affect the dynamics of the forward model and make the problem much more difficult to solve. Whereas when there is no object between the fingers, we have full knowledge of the state and we can accurately evaluate the forward model. In particular, we know that the torque $\tau_{t+1} = k_p(\theta_{\text{motor}_t} - \theta_{\text{joint}_t})$, where τ_{t+1} is the torque at time step $t+1$, k_p is proportional gain (a constant), θ_{motor_t} is the motor position at time step t and θ_{joint_t} is the joint position at time step t. We evaluate the model's ability to predict multiple time steps into the future. The first state is the observed state, and every subsequent state is generated by supplying the previous state and an action to the forward model.

1) Linear Ridge Regression: Linear Ridge Regression is an augmentation of Ordinary Least Square regression by adding a term α that minimizes the sum of squared errors to prevent multicollinearity. The larger α is, the more robust the regression is to multicollinearity. Multicollinearity occurs when features are highly linearly related. In our problem, the features are linearly related, exemplified by the linear relationship between torque and the difference between the motor and joint positions. We create a linear ridge regression model that takes in the states and actions (joint position, motor position, joint torque, and motor command for each joint) and outputs the state at the next time step (joint position, motor position, joint torque). We perform a line search over various values of α with cross validation to find the ideal value for α. We can see in Fig. 10a that the learned model correctly predicts that there is a change in the joint torques, joint positions, and motor position and the correctly predicts the direction of the change.

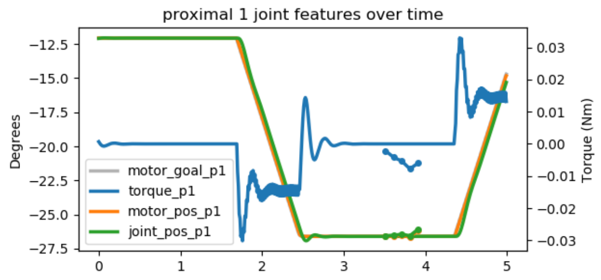
However, the model is not perfectly accurate. In Fig. 10b, the observed state (the timestep 0.1 seconds before the first predicted state) has a motor position that is equal to joint position, but the predicted torque for the next time step is non-zero. We were unable to learn that $\tau_{t+1} = k_p(\theta_{\text{motor}_t} - \theta_{\text{joint}_t})$ using a linear ridge regressor. The forward model also predicted changes in the state when the motor command did not change in Fig. 10b. We do not expect a linear model to be able to learn a perfect forward model, since the mapping from current state to next state is non-linear.

2) Deep Neural Network: Deep Neural Networks are designed for the purpose of modeling non-linear systems. We construct a neural network with 2 hidden layers with 100 hidden nodes each and ReLU activation between the layers. Similar to the linear ridge regressor, the input to the network were the state features and actions, and the output of the network were state features of the next timestep. We train the neural network for 15 epochs using batch sizes of 100, mean squared error loss, and an Adam optimizer with a learning rate of 0.001.

After training, we find again that while the observed states in both plots in Fig. 11 had motor position equal joint position, the first predicted torques are not 0. The forward model learned by the deep neural network is also unable to learn the simple relationship between motor position, joint position, and torque. We have found that constructing a forward model of the system's dynamics to be more difficult than expected, despite having a large amount of training data.

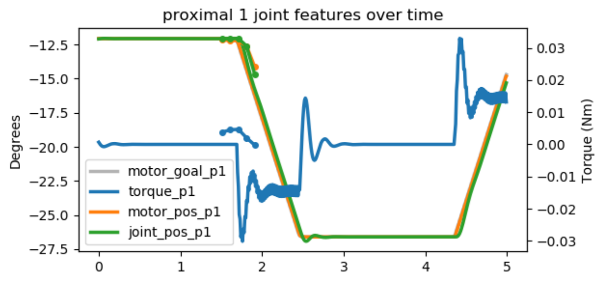


(a) Prediction of the next five time steps given an initial state at 1.5 seconds

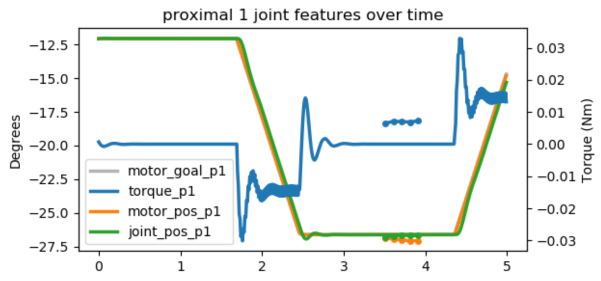


(b) Prediction of the next five time steps given an initial state at 3.5 seconds

Fig. 10. These are plots of features for one of the proximal links over time in seconds, where the lines without dots represent observed states and the dotted lines represent the predicted states. We feed the Linear Ridge Regressor model an initial observed state s_t and an action a_t to obtain the first predicted state s_{t+1}. We then repeat the process with the predicted state s_{t+1} and a given action a_t to obtain the next predicted state s_{t+2}. We keep repeating the process for a total of 5 predicted time steps. Here, a single time step is 0.1 seconds and motor position is motor command (denoted as motor_goal) offset by 0.01 seconds



(a) Prediction of the next five time steps given an initial state at 1.5 seconds



(b) Prediction of the next five time steps given an initial state at 3.5 seconds

Fig. 11. These plots are identical to the ones in Fig. 10, except we use a Deep Neural Network instead of a Linear Ridge Regressor

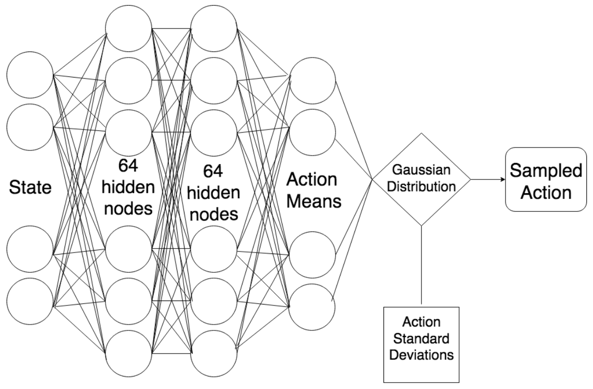


Fig. 12. The architecture of the policy $\pi(a_t|s_t)$

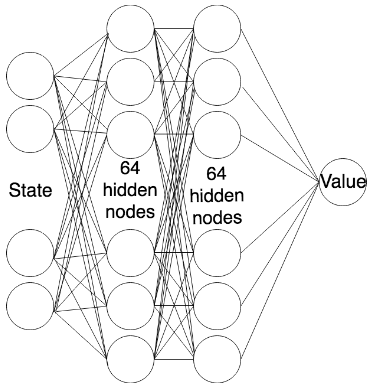


Fig. 13. The architecture of the state value function $V(s_t)$

V. Model Free Reinforcement Learning

We have thus far been unsuccessful in creating forward models to use for model-based control methods. Instead, we embark on the task of using Model Free Reinforcement Learning to learn a policy for grapsing using proprioceptive features and evaluate the practicality of Model Free Learning.

A. Defining the Problem as a Markov Decision Process

We want to derive a policy to stably grasp an object, where our reward indicates how good the grasp is. In reinforcement learning, we formulate the problem as a Markov Decision Process to learn a policy π that maximizes total expected reward. At each time step t, the agent chooses an action a_t from its current policy $\pi(a_t|s_t)$, where s_t is the current state, and determines the reward $r(s_t, a_t)$. Total reward from a time step t is $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, t_i)$ where T is the horizon, and γ is a discount factor that prioritizes early reward. The optimal policy maximizes $\mathbb{E}(R_1)$. In our grasping problem, actions are defined by motor commands and states are solely defined by proprioceptive features: joint position, joint torque, and motor position.

B. Policy Gradient Method

Once we have defined the problem as a Markov Decision Process, we need to learn the policy. We learn the policy by directly optimizing the policy itself through policy gradient method. For policy gradient method, we want to maximize $\mathbb{E}(\pi_\theta(a_t|s_t)A_t)$, where θ are the parameters of the policy and A_t is the advantage of an state and action pair.

1) Policy: The policy in a policy gradient method is a neural network takes states as input and outputs an estimated mean of the actions. These means and user defined standard deviations of the actions then form the Gaussian distribution from which actions are sampled, creating a stochastic policy. The specific network architecture we used is in Fig. 12. The policy network $\pi(a_t|s_t)$ is trained by maximizing the objective function $\mathbb{E}(\pi_\theta(a_t|s_t)A_t)$ using gradient descent.

2) Advantage Function: Advantage A_t is the difference between the state-action value and the state value. The state-action value Q is the expected reward from a state s_t after taking action a_t. The state value V is the expected reward from a state s_t regardless of which action is taken.

$$Q(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \ldots}[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})] \tag{10}$$

$$V(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \ldots}[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})] \tag{11}$$

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{12}$$

We estimate the state value $V(s_t)$ using a neural network (Fig. 13). We estimate the state action value $Q(s_t, a_t)$ as the observed total rewards R_t for a state s_t. The value function network $V(s_t)$ is trained by minimizing

$$(Q(s_t, a_t) - V(s_t))^2 \tag{13}$$

via gradient descent, since we want the estimated value of the state $V(s_t)$ to be close to the observed value of the state $Q(s_t, a_t)$.

C. Proximal Policy Optimization

By directly optimizing the policy network's objective function through gradient descent, the steps can be too large and overshoot the local optimum leading to divergence. Instead, we use Proximal Policy Optimization introduced in [3] by Schulman et. al, which modifies the objective function. They prevent divergence by encouraging the new learned policy $\pi_{\theta_{\text{new}}}$ to stay near the old policy $\pi_{\theta_{\text{old}}}$. The new objective function to train the policy is

$$\min\Bigg(fA_t, \text{clip}\Big(1-\epsilon, 1+\epsilon, f\Big)A_t\Bigg) \tag{14}$$

$$f = \frac{\pi_{\theta_{new}}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{15}$$

where ϵ is the clipping parameter (Fig. 14). By clipping (15), they do not reward new policies for significantly diverging from old policies. See Algorithm 1 for the Proximal Policy Optimization algorithm.

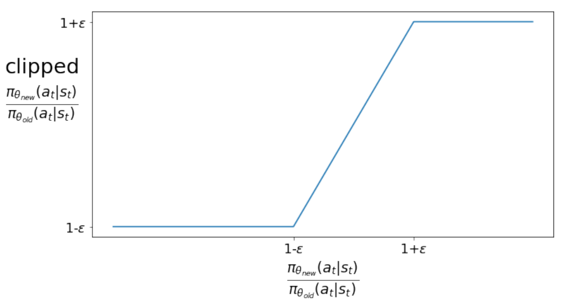


Fig. 14. Clipped $\frac{\pi_{\theta_{\text{new}}}}{\pi_{\theta_{\text{old}}}}$ vs. $\frac{\pi_{\theta_{\text{new}}}}{\pi_{\theta_{\text{old}}}}$

Algorithm 1: Proximal Policy Optimization

```
foreach iteration do
    foreach actor do
        for $t := 1$ to $T$ do
            Execute policy $\pi_{\theta_{\text{old}}}$ in the environment
        end
    end
    Compute estimates of $A_1, A_2, ..., A_{T-1}, A_T$
end
Maximize policy objective function (14) and minimize
value objective function (13) through gradient descent
over $K$ epochs with a batchsize of $M$.
```

VI. Experiments and Results

In this section, we demonstrate the ability of our simulated robot to learn reach a given position given a simple reward function in our toy problem, so we can establish how well the Proximal Policy Optimization algorithm performs for simple problems and conceptualize how this might extend to a more complex problem. We also initiate work on developing a policy for grasping using proprioception.

A. Toy Problem

Our objective is to create a policy that will command the SEA gripper's distal joint on the right finger to 45 degrees. The state comprises the position of the joint of interest and the action is the motor command for the same joint. Joint position ranges from 0 to 90 degrees, where 0 is the fully extended position. We establish the reward function for this problem to be $r(s_t) = \left(1 - \frac{45-s_t}{45}\right)^2$. We trained the robot for 12,000 iterations (see Appendix B-A for hyperparameters) and were able to achieve convergence to a reward of 1.0 after 12,000 iterations (Fig. 15). Training for 12 thousand iterations and collecting over 12 million state and action pairs took 18 hours on commodity hardware with 4 cores, demonstrating the sample inefficiency of model free reinforcement learning methods. However, the learned policy is robust and the joint will reach a position of 45 degrees no matter the initial joint position, as seen in Fig. 16.

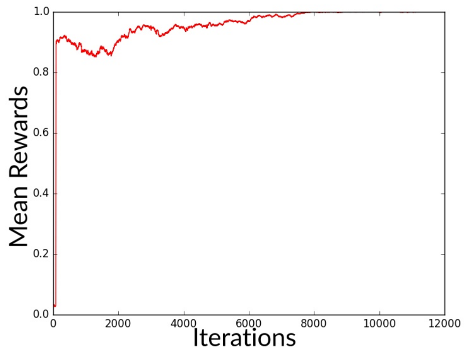
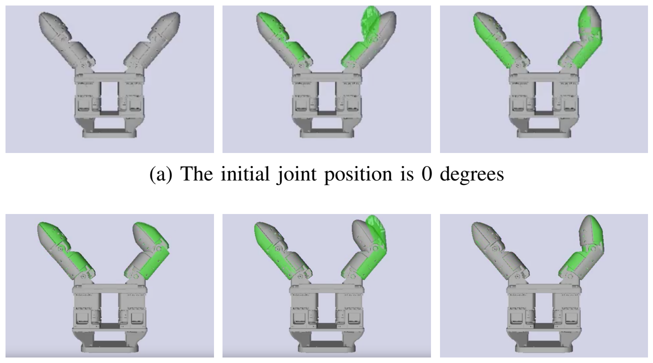


Fig. 15. Learning Curve for the Toy Problem trained for 12,000 iterations where the maximum possible reward is 1



(a) The initial joint position is 0 degrees

(b) The initial joint position is 80 degrees

Fig. 16. We execute the policy for the toy problem. The green represents the motor command

B. Grasping a Box

In this experiment, our goal is to create a policy that will stably grasp an object. The state s_t is composed as previously defined: joint postion θ_{joint}, joint torque τ, motor position θ_{motor} for each joint. The action a_t is composed of the motor commands for each joint. Our reward function is

$$r(s_t) = 2 - ||\tau - 0.2||) - ||\theta_{\text{joint}} - \theta_{\text{closed}}||) \quad (16)$$

$$\theta_{\text{closed}} = [45, 90, 45, 90] \quad (17)$$

where θ_{closed} is the vector of joint positions when the gripper is fully closed. Distal joint positions range from 0 to 90 degrees, while proximal joint positions range from -45 to 45 degrees. We want to encourage high torques and the joints to be as fully closed. We trained the robot by randomly placing a box between the fingers of the robot and executing the Proximal Policy Optimization algorithm for 18 thousand iterations, collecting over 18 million state and action pairs. If the box pushed out of the gripper's reach, we reset the scene and place the box in a new random location between the fingers of the robot.

The policy does not improve much over the iterations as demonstrated by Fig. 17. However, we know that this problem is much more complex that the Toy Problem in the previous section, so it may be that we need to train the policy for more

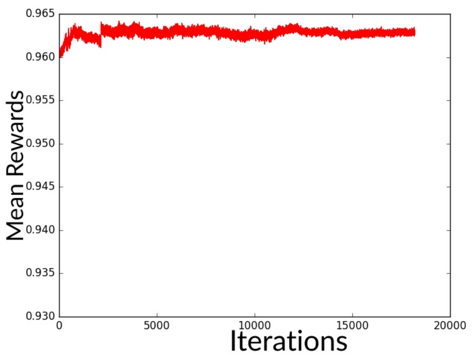
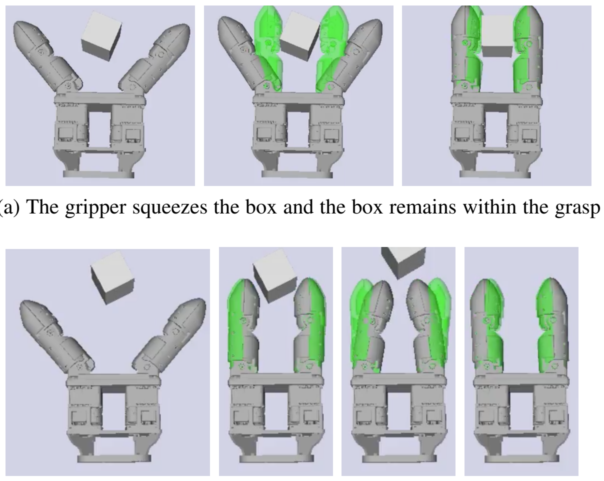


Fig. 17. Learning Curve for the Grasping problem trained for 18,000 iterations where the maximum possible reward is 2



(a) The gripper squeezes the box and the box remains within the grasp

(b) The gripper squeezes the box and the box is pushed out of the grasp

Fig. 18. Here we have a top down view of the grasping policy being executed with a box between the fingers. The box is supported by a surface.

iterations. As shown in Fig. 18, we learned a policy to close the fingers to the width of the box and to squeeze the box. Even when the box is pushed out of the grasp, the policy continues to command a motor position that corresponds with the width of the box. We, nevertheless, were not able to derive a policy to stably grasp an object, since the block can slip out of the grasp.

VII. Conclusion and Future Work

Simulation is a promising method for robot skill learning. The quantity of data required for data driven learning can be a limiting factor on a physical robot. Simulated robots are faster to set up than a physical robot aiding faster prototyping, and due to parallelization, we can collect data faster in simulation than in the real world. Having an accurate and robust simulator is vital for these kinds of tasks, so we developed tests to evaluate the robustness of a simulator, which involve increasing the number of constraints the simulator had to solve and investigating any simulation artifacts that occur.

We evaluated the simulator Klamp't, and through our critical torque test, we found that Klamp't can accurately model friction. Additionally, Klamp't is able to simulate highly constrained scenarios, like a three finger gripper holding a glass, for extended periods of time indicating its ability to avoid interpenetration between objects. We investigated the source of Klamp't numerical instability, as it was unable to accurately simulate dense clutter for sufficiently long periods of time. Despite this numerical instability, we found that Klamp't is more robust to physical inaccuracies and interpenetration than other simulators we examined, as all simulators will have some amount of error.

We used Klamp't to collect hundreds of thousands of data points and found building forward models to be difficult even with large datasets. We proceeded to investigate a model free reinforcement learning technique, Proximal Policy Optimization. We found that while model free methods can be simpler to implement than model based methods, they are extremely sample inefficient and require extraordinarily large training sets. We have so far only shown the model free method to work for our toy problem and have yet to show it to work on a more realistic problem, where we do not have complete state information.

We have shown that learning in simulation is a valuable exercise that can provide great insight to the problem of grasping and manipulation. Future work includes improving upon our forward dynamics models by investigating different neural network architectures and hyperparameters, using the improved forward dynamics models to develop control methods through Model Predictive Control or Model-Based Reinforcement Learning, transfer learning – applying dynamics models or controller methods learned in simulation onto a real robot, and employing Model-Free Reinforcement Learning to more difficult tasks, such as grasping from clutter.

Appendix A
Potential Energy as a Function of Block Displacement

By definition, potential energy is the negative of total work done, so

$$U = -\sum_{i=1}^{3} W_i(\vec{x}) \tag{18}$$

where $W_i(\vec{x})$ is work done by finger i given displacement $\vec{x}$. Work done is the integral of force with respect to displacement; in this case the force is the contact normal force N, since the displacement is along the x-y plane. We can compute N_i from the torques applied to finger i. At a given finger, there are three torques to consider: $\tau =$ the torque applied at the joint (which is user defined), $\tau_f =$ the torque due to gravity on the finger, and τ_c the torque due to the contact force with the block. Let

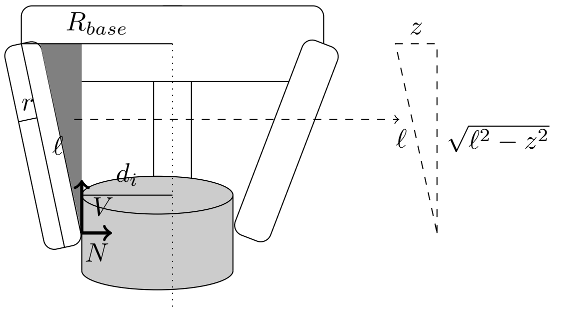


Fig. 19. We want to solve for the normal contact force N so we can compute the total work done. In this diagram, $V =$ the force required to hold the block, $R_{base} =$ the radius of the base of the gripper, $r =$ the radius of a finger, $\ell =$ the length of a finger, $d_i =$ the the distance between the central axis of the base and point of contact between the finger i and the block, and $z = R_{base} - r - d_i$.

M_f be the mass of a finger and g be gravitational acceleration. From Fig. 19, we know

$$\tau_f = (\frac{z}{2}, -\frac{\sqrt{\ell^2 - z^2}}{2}) \times (0, M_f g) = -\frac{z}{2} M_f g \tag{19}$$

$$\begin{aligned}\tau_c &= (R_{base} - d_i, -\sqrt{\ell^2 - z^2}) \times (-N, -V) \\ &= -V(R_{base} - d_i) - N\sqrt{\ell^2 - z^2}\end{aligned} \tag{20}$$

Since τ is user defined, we set $\tau + \tau_f + \tau_c = 0$ and solve for N.

$$\begin{aligned}0 &= \tau + \tau_f + \tau_c \\ 0 &= \tau - \frac{z}{2} M_f g - V(R_{base} - d_i) - N\sqrt{\ell^2 - z^2} \\ N &= \frac{\tau - \frac{z}{2} M_f g - V(R_{base} - d_i)}{\sqrt{\ell^2 - z^2}}\end{aligned} \tag{21}$$

Let us assume that each finger i supports 1/3 of the block's weight, so $V = \frac{M_b}{3}$. Then, every variable defining N in (21), except for d_i and z which is a function of d_i, is known variable that is independent of $\vec{x}$. For us to ultimately understand whether the system is in stable equilibrium or not, we need to define d_i as a function of $\vec{x}$.

As previously discussed, d_i is the distance between the central axis of the base of the gripper and the point of contact between a finger and the block. Looking at a top down view of the gripper holding a block in Fig. 20, we can see that

$$R_{block} = ||d_i \vec{f_i} - \vec{x}|| \tag{22}$$

where f_i is the unit vector in the direction of the finger from the central axis of the base of the gripper. Thus,

$$R_{block}^2 = (d_i \vec{f_i})^2 - 2 d_i \vec{f_i} \vec{x} + \vec{x}^2 \tag{23}$$

By the quadratic formula

$$d_i = \vec{f_i} \cdot \vec{x} \pm \sqrt{(\vec{f_i} \cdot \vec{x})^2 + (R_{block}^2 - \vec{x}^2)} \tag{24}$$

Since distances must be positive

$$d_i = \vec{f_i} \cdot \vec{x} + \sqrt{(\vec{f_i} \cdot \vec{x})^2 + (R_{block}^2 - \vec{x}^2)} \tag{25}$$

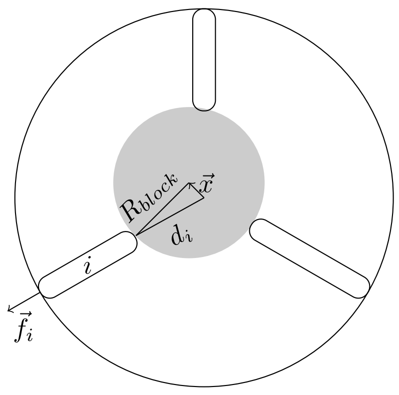


Fig. 20. Top Down View of the Three Finger gripper holding a block slightly displaced from the center of the hand. R_{block} is the radius of the block, $\vec{x}$ is the displacement of the block from the central axis of the base of the gripper,d_i is the distance between the central axis of the base of the gripper and the point of contact between finger i and the block,and f_i is the unit vector from the center of the base of the gripper to the finger.

We can substitute (25) into (21) to define N as a function of $\vec{x}$.

Work done by a finger i is the integral of force with respect to displacement of the finger.

$$W_i = \int_{R_{base} - R_{block} - r}^{R_{base} - d_i - r} N dz \tag{26}$$

$$= \int_{R_{base} - R_{block} - r}^{R_{base} - d_i - r} \frac{\tau - \frac{z}{2} M_f g - V(R_{base} - d_i)}{\sqrt{\ell^2 - z^2}} dz \tag{27}$$

Solving the integral,

$$\begin{aligned}W_i &= (\tau - rV)\Big[\arcsin\Big(\frac{D}{\ell}\Big) - \arcsin\Big(\frac{R}{\ell}\Big)\Big] \\ &+ \Big(\frac{M_f g}{2} + V\Big)\Big[\sqrt{\ell^2 - D^2} + \sqrt{\ell^2 - R^2}\Big]\end{aligned} \tag{28}$$

where

$$\begin{aligned}D &= R_{base} - d_i - r \\ R &= R_{base} - R_{block} - r\end{aligned}$$

We substitute this back into (18) and plug in all of the parameters defined by the configuration of the hand and the block. We then can plot the potential energy of the system as a function of x and y displacement, as we do Fig. 5.

Appendix B
Hyperparameters for Proximal Policy Optimization

A. Toy Problem

learning rate	0.01
number of actors	8
number of timesteps	128
batch size	256
discount factor γ	0.99
clip parameter ϵ	0.01
optimizer	Adam
time step size	0.01 seconds

B. Grasping a Box

learning rate	0.0003
number of actors	8
number of timesteps	128
batch size	256
discount factor γ	0.99
clip parameter ϵ	0.01
optimizer	Adam
time step size	0.01 seconds

ACKNOWLEDGEMENTS

I am grateful to my advisor Professor Matei Ciocarlie for his invaluable guidance and support through these past two years. I would like to express my thanks to members of my thesis committee Professor Peter Allen and Professor Eitan Grinspun for their encouragement of my work. I would also like to thank the members of the Robotic Manipulation and Mobility Lab at Columbia University for the stimulating conversations we had. Finally, I would like to express my gratitude to my parents for their unwavering support.

REFERENCES

[1] K. Hauser, "Robust Contact Generation for Robot Simulation with Unstructured Meshes," Apr. 2016, pp. 357–373.

[2] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of Locomotion Behaviours in Rich Environments," *arXiv:1707.02286 [cs]*, Jul. 2017, arXiv: 1707.02286. [Online]. Available: http://arxiv.org/abs/1707.02286

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Jul. 2017, arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347

[4] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," *arXiv:1603.00748 [cs]*, Mar. 2016, arXiv: 1603.00748. [Online]. Available: http://arxiv.org/abs/1603.00748

[5] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," *arXiv:1610.00633 [cs]*, Oct. 2016, arXiv: 1610.00633. [Online]. Available: http://arxiv.org/abs/1610.00633

[6] A. Boularias, J. A. Bagnell, and A. Stentz, "Learning to Manipulate Unknown Objects in Clutter by Reinforcement," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 1336–1342. [Online]. Available: http://dl.acm.org/citation.cfm?id=2887007.2887192

[7] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," *arXiv:1603.02199 [cs]*, Mar. 2016, arXiv: 1603.02199. [Online]. Available: http://arxiv.org/abs/1603.02199

[8] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," *arXiv:1710.06537 [cs]*, Oct. 2017, arXiv: 1710.06537. [Online]. Available: http://arxiv.org/abs/1710.06537

[9] H. Dang, J. Weisz, and P. K. Allen, "Blind grasping: Stable robotic grasping using tactile feedback and hand kinematics," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 5917–5922.

[10] M. Madry, L. Bo, D. Kragic, and D. Fox, "ST-HMP: Unsupervised Spatio-Temporal feature learning for tactile data," May 2014, pp. 2262–2269.

[11] Y. Chebotar, K. Hausman, Z. Su, G. S. Sukhatme, and S. Schaal, "Self-supervised regrasping using spatio-temporal tactile features and reinforcement learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 1960–1966.

[12] T. Chen and M. Ciocarlie, "Grasping Unknown Objects with Proprioception Using a Series-Elastic-Actuated Gripper," *arXiv:1803.09674 [cs]*, Mar. 2018, arXiv: 1803.09674. [Online]. Available: http://arxiv.org/abs/1803.09674

[13] "Intelligent Motion Laboratory at Duke University." [Online]. Available: http://motion.pratt.duke.edu/klampt/

[14] "Open Dynamics Engine - home." [Online]. Available: http://www.ode.org/

[15] "Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning." [Online]. Available: https://pybullet.org/wordpress/

[16] "GitHub - PositronicsLab/Moby: The Moby rigid body dynamics simulator." [Online]. Available: https://github.com/PositronicsLab/Moby

[17] "GraspIt!" [Online]. Available: http://graspit-simulator.github.io/

[18] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and Model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, Jul. 2015, pp. 510–517.