Comparing Model Based and Model Free techniques for Underactuated In-hand Manipulation

Pratik Gyawali

A thesis

submitted in the in partial fulfilment of the

requirements for the degree of

Master of Science in Mechanical Engineering

University of Washington

2021

Committee:

Siddhartha Srinivasa

Santosh Devasia

Xu Chen

Program Authorized to Offer Degree:

Department of Mechanical Engineering

University of Washington

Abstract

Comparing Model Based and Model Free techniques for Underactuated In-hand Manipulation

Pratik Gyawali

Chair of the Supervisory Committee:

Professor Siddhartha Srinivasa

Computer Science and Engineering

Compared to their fully actuated counterparts underactuated hands are cheap, lighter and provide stable grasp across variety of objects without feedback. However, underactuated hands are less dexterous for in-hand manipulation task due to the limited range of motion in their configuration space. Brake Assisted Tendon Actuator (BATA) is a novel mechanism to enhance dexterity in underactuated hand. This work aims to implement a controller framework for BATA and asses it's in-hand manipulation capabilties. Control is challenging due to contacts, under-actuation and model uncertainty. We develop a simulation environment in MuJoCo and formulate the underlying discrete Markov Decision Process. Model based and model free reinforcement learning methods are implemented to learn a policy for a specific type of in-hand manipulation task: rolling. Simulation results with objects of varying mass and radius suggest Model Predictive Path Integral (MPPI) is more generalizable compared to model free, Proximal Policy Optimization (PPO).

to my family and friends

Acknowledgments

I would like to thank my mentors Patrick Lancaster and Dr. Christoforos Mavrogiannis for their continuous guidance over the last year. Moreover, I would like to thank Prof. Srinivasa and Prof. Smith for giving me the opportunity to work with them, and for their crucial feedback. Thanks to my committee, Prof. Devasia, and Prof. Chen for their support.

Thanks to my friends from 2019-2021 MSME cohort, Miles Matsen, Krishna Balasubramanian who contributed in my education and helped me to improve my research.

I would like to thank my friends, Jayanth Kumar, Uttam Bhetuwal, Tanka Rana, Milan Gurung and Biswa Aryal for being the best company and for giving me great advices about my life.

Finally, I would like to thank baba, mommy and baini for their endless love, and their continuous support at every step of my life.

Table of Contents

List of Tables

List of Figures

Chapter1

Introduction

Object and tool manipulation is a fundamental problem in robotics. Dexterous manipulation is an exciting area of robotics in which multiple manipulators, or fingers, cooperate to grasp and manipulate objects [1]. Dexterous manipulation, requiring precise control of forces and motions, cannot be accomplished with a conventional robotic gripper; fingers or specialized robotic hands must be used [2]. In this section we explore fully actuated dexterous hands, their limitations and the motivation for developing dexterous underactuated hand.

1.1 Background

Dexterity has been commonly addressed in robotics using rigid, fully actuated, multi-fingered robot hands[3, 4, 5]. Existing dexterous robot hands, which are able to re-grasp and manipulate objects within the hand, are usually complex systems with multiple actuators and sophisticated sensing elements[6]. These types of hands are very bulky, expensive and difficult to build and maintain. From controls perspective, these system rely on complicated control laws because of rich contact patterns and high-dimensional action and state spaces.

Underactuated hands are devices with less control authority over the joints i.e. they have fewer actuators than the degrees of freedom (DOF). Underactuated hands are often equipped with flexure or spring loaded pin joints, under-actuated fingers, and differential mechanisms, which provides their adaptive nature. These hands are capable of exhibiting enveloping or force closure grasp by which they grasp objects of a variety of shapes and sizes by wrapping

the fingers around it [7].

Most applications of underactuated hands involve the use of enveloping grasp, although some prior works have demonstrated use of a special case of form closure known as pinch grasp for doing in hand manipulation using underactuated hands [8]. Overall advantages of underactuated hands over fully actuated counterparts can be summarized as:

- Lighter and smaller than fully actuated hands due to the reduced number of sensors and actuators
- Self adapting nature due to enveloping provides stable grasp across wide range of objects without feedback
- Easy to manufacture and maintain using rapid prototyping techniques

Despite these advantages, the general application of underactuated hands is limited particularly by their range of motion. These hands are mostly actuated with cable as a result of which only a small subset of poses in the configuration space can be reached.

1.2 Related Work

By holding an object between the fingertips rather than wrapping the fingers around it, some work have been reported in the literature for underactuated in-hand manipulation. Odhner and Dollar demonstrated that an underactuated hand is able to move cylindrical objects in the horizontal plane [7]. Their underactuated hand prototype comprises a pair of 3-DOF 1-actuator fingers with a passive distal joint made of flexible material. Even though the hand has two actuators, the experimental results show only one-dimensional motion mainly in the direction transversal to the fingers.

Authors in [9], developed a flip-and pinch o that enables a pair of 2-DOF 1-actuator fingers to pick up a thin object from a flat surface and securely hold it between the fingertips without tactile sensing or force control.Calli and Dollar [10] demonstrated that a robust, planar in

hand manipulation is possible, without detailed hand or object models, using basic visual servoing techniques and no additional sensing. Likewise in [11] they leverage this robust and simple control for data-driven approaches to improve understanding of the hand-object system, especially for difficult-to-sense modes of operation during in hand manipulation.

1.3 Motivation

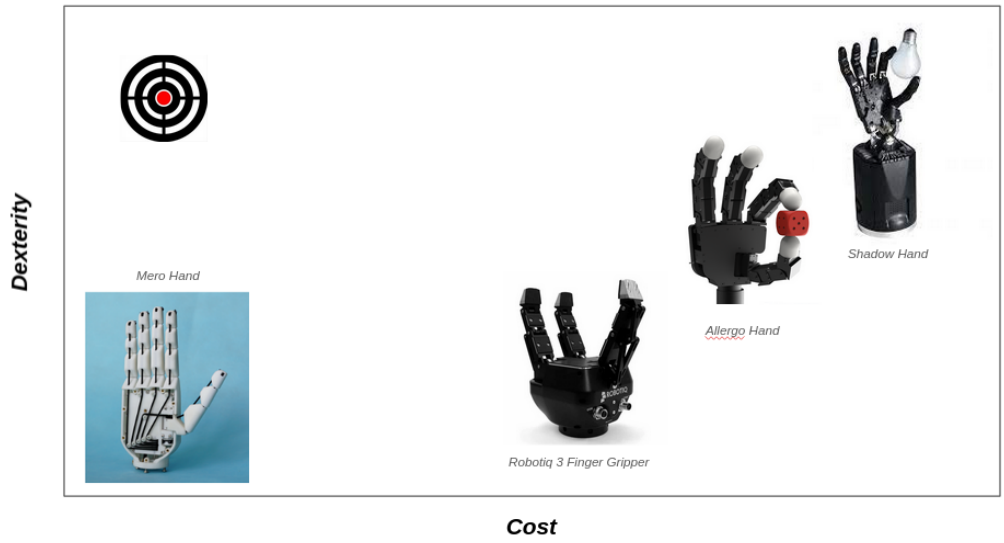


Figure 1.1: Cost-Dexterity spectrum of robotics hands

The cost-dexterity spectrum for the robotic hand is highly skewed. As shown in Figure 1.1, there are highly dexterous fully actuated systems [12] on the right end, whose high cost limits their use across the general robotics community. Likewise, there are underactuated hands on the lower left end which are cheap [13] but are limited by their sensing and range of motion capabilities.

Novel sensing and actuation can be incorporated into existing underactuated hands to develop next generation of highly dexterous underactuated hands. Towards this bigger goal of hitting the sweet spot in the cost-dexterity spectrum, researchers in the Sensor Systems Lab and Personal Robotics Lab at the University of Washington are developing Brake Assisted

Tendon Driven Actuator (BATA).

BATA, as shown in Figure 1.2 is a two finger 3DOF tendon driven underactuated gripper. The flexion of fingers is achieved by pulling the tendons through the motors, whereas extension is achieved through counter-torque provide by the linear spring acting on the rack and pinion mechanism on the joints. Each joints of BATA is equipped with electrostatic brakes. Activating the brakes provide a counter torques on the joint, thereby locking the joint position. This locking phenomenon increases the range of motion, and the fingers can reach arbitrary points in the reachable workspace. In this way by using braking technology, BATA significantly improves the dexterity of underactuated hands.

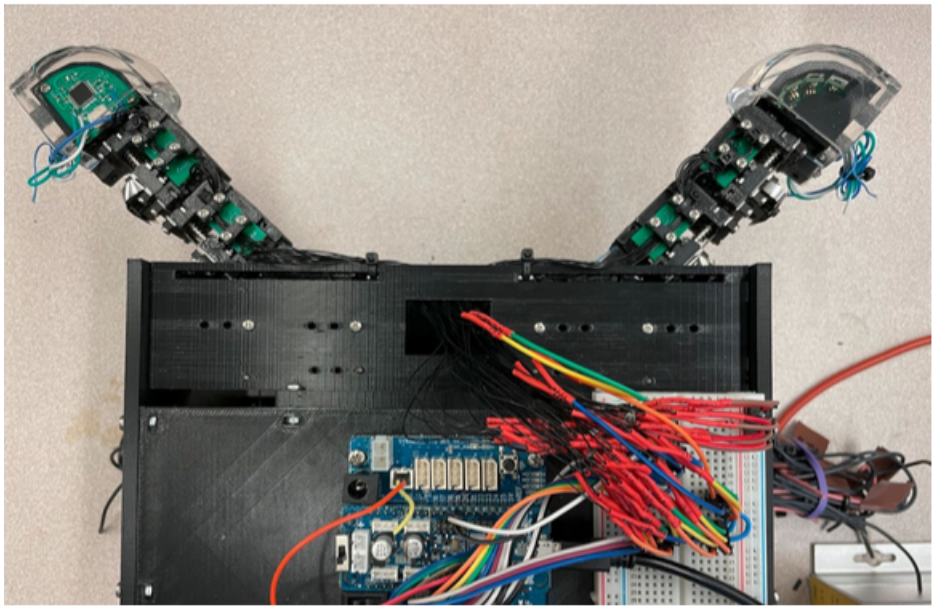


Figure 1.2: Early Prototype of BATA

Unlike most underactuated hands, BATA is equipped with joint encoders in each of the joints for position feedback. Likewise, the fingertips are equipped with proximity sensors that facilitates "pre-touch sensing". These sensing capabilities enhances the efficiency of underactuated system.

BATA is under heavy development, and this thesis utilizes BATA in its current stage: without electrostatic brakes, to design and evaluate controllers for underactuated in-hand manipulation.

1.4 Problem Statement

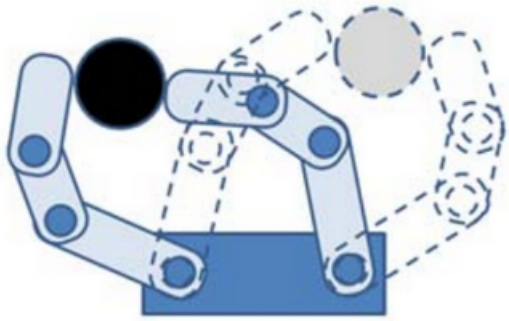


Figure 1.3: In Grasp manipulation task

The primary goal of this thesis is to develop and evaluate controllers for in hand manipulation using BATA. In particular, this thesis will address a specific instance of in-hand manipulation known as in-hand grasping or rolling as shown in Figure 1.3. This fits into the bigger scope of BATA by 1) Laying a framework to test new controllers, 2) Providing a baseline for comparing the effectiveness of electrostatic brakes in future work. By the end of the thesis we seek to achieve the following,

- A two-fingered, tendon actuated gripper must move a cylindrical object to a desired goal location along a path perpendicular to the gripper's plane of symmetry (rolling motion)
- Perform the rolling task robustly in a previously unobserved object size, and object weight, to enhance generalization and facilitate sim2real transfer

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 discusses the modelling of the BATA actuator, Chapter 3 focuses on the theory and the implementation details of the

controllers used, Chapter 4 presents the results of the simulations while Chapter 5 concludes the thesis and discusses future directions.

Chapter2

Modelling

As we proceed to control BATA, we need a model that abstractly describes the behaviour of the system. This representation explains how the system evolves as a function of time and input. In the chapter, we discuss how we developed the BATA model which forms the basis for control in the later chapter.

2.1 Kinematic Modelling

A kinematic model describes the motion of a robot in mathematical form without considering the forces that affect motion and concerns itself with the geometric relationship between elements.In the kinematic model of a robot, the connection of different manipulator joints is known as link, and the integration of two or more links is called a joint. Using the rigid transformation [14] between the links, the kinematic equation for a serial manipulator can be developed.

Once, this transformation is formulated the Forward kinematics uses this transformation to compute the position of the end-effector from specified values for the joint parameters. Likewise, the inverse problem of computing the joint parameters that achieve a specified position of the end-effector is known as inverse kinematics, and can be computed geometrically, algebraically or numerically [15].

Denavit-Hartenberg (DH) convention is a well-known minimal parameter convention for 3D serial robot kinematics [16]. In this convention, joint axes are always aligned to the z axis of each child link, and the offset between joints is always pointing along the x axis of the

N	α_{N-1}	a_{N-1}	d_N	θ_N
1_r	0	-0.05144	0.09795	θ_{1r}
2_r	0	0.04214	0	θ_{2r}
3_r	0	0.03714	0	θ_{3r}
R_{tip}	0	0.03451	-0.00275	0
1_l	0	0.05144	0.09795	θ_{1l}
2_l	0	0.04214	0	θ_{2l}
3_l	0	0.03714	0	θ_{3l}
L_{tip}	0	0.03451	-0.00275	0

Table 2.1: BATA DH Parameters

parent link. Using the Link frame assignment as shown in Figure 2.1, the DH parameters for BATA is formulated in Table 2.1

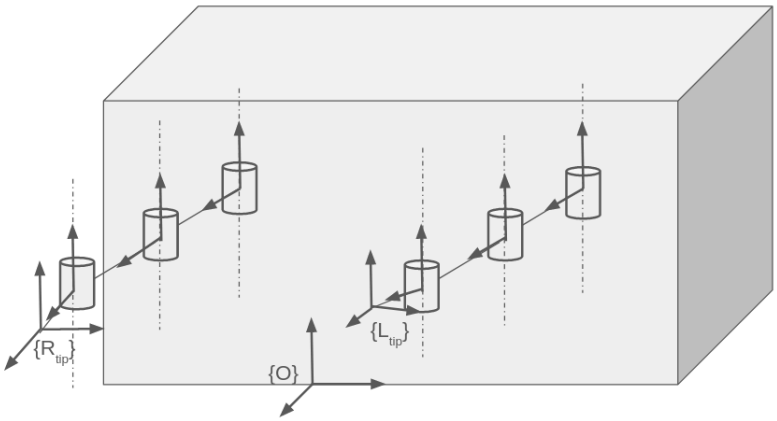


Figure 2.1: Frame Assignment for Forward Kinematics

The DH formulation of BATA allows us to use the equation 2.1 which essentially represents the transformation from the previous link to the current link [15].

$$
{}^{N-1}_{N}T = \begin{bmatrix} c\theta_N & -s\theta_N & 0 & a_{N-1} \\ s\theta_N c\alpha_{N-1} & c\theta_N c\alpha_{N-1} & -s\alpha_{N-1} & -s\alpha_{N-1}d_N \\ s\theta_N s\alpha_{N-1} & c\theta_N s\alpha_{N-1} & c\alpha_{N-1} & c\alpha_{N-1}d_N \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}
$$

Locating the finger tips with respect to the base is therefore just aggregating these trans-

formations from the DH parameters in sequence,

$$\overset{O}{R_{tip}}T = \overset{O}{1_r}T\ \overset{1_r}{2_r}T\ \overset{2_r}{3_r}T\ \overset{3_r}{R_{tip}}T \tag{2.2}$$

Although we explore the Universal Robot Description Format (URDF) like file format in the subsequent subsection, this formulation of the kinematic model is particularly important to estimate end effector embedded proximity sensor readings with respect to the base link.

2.2 Dynamic Modelling

Dynamic models explain the relation between the applied forces/torques and the resulting motion in a robot. Typically for fully actuated robot, the standard approach is to develop the dynamic equations from first principles using techniques like Newton-Euler algorithm (RNEA), articulated-body algorithm (ABA),composite-rigid-body algorithm (CRBA) etc [15, 14, 16].

In our application of BATA, modelling from first principles is a particularly difficult due to under actuation, tendon interaction and contacts with the manipulated object. More recently, use of robotic simulators with high fidelity physics engine has been explored and demonstrated impressive results [17, 18]. Following the same philosophy, we develop a simulation model of BATA with accurate contact simulations in MuJoCo.

2.2.1 MuJoCo Model

MuJoCo stands for Multi-Joint dynamics with Contact [19]. It is a general purpose physics engine that has been finding extensive use in research and development in robotics, biomechanics, graphics and animation, machine learning, and other areas which demand fast and accurate simulation of articulated structures interacting with their environment.

MuJoCo models are XML based MJCF file format. This is similar to the more extensively used URDF file format. We develop a gray box model for the BATA actuator in MuJoCo,

meaning certain parameters of the model are based on the real robot. In particular, the same STL files used for 3D printing is used to define the links as shown in Figure 2.2. The link inertia is inferred from geometry where as the actual masses of the links used. Like the real robot, a tendon passing through the idlers is implemented, and the actuator exerts pulling force on the tendon. Ray-traced simulation of proximity sensors is also included in the XML format.

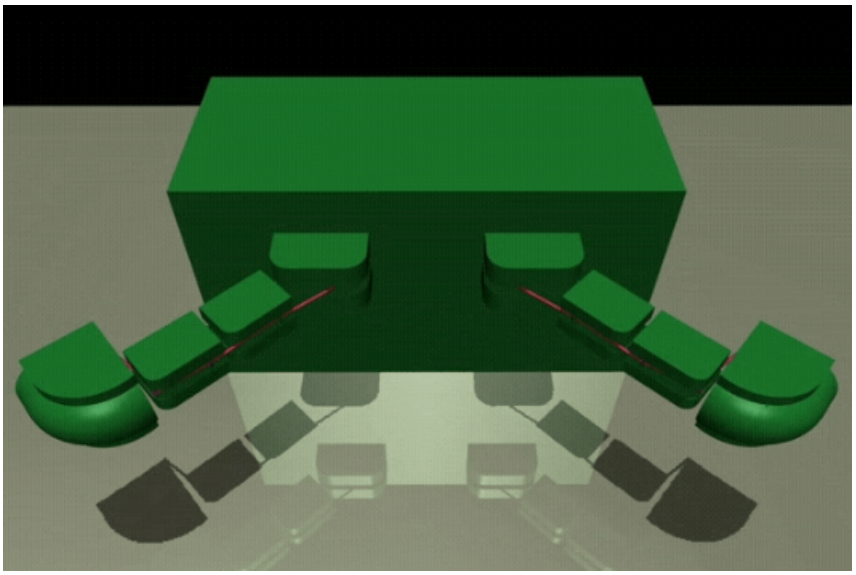


Figure 2.2: BATA MuJoCo Model

2.2.2 Tuning the model

As explained in the previous section, the flexion is provided by the tendons whereas the extension is provided by the springs. Two springs resist the motion, thereby exerting the torque on the rack-pinon mechanism which forces the fingers to return to the spring rest i.e. the home position. Without modelling this behaviour in simulation, BATA control in simulation wouldn't encode the behaviour of the actual robot hand.

Modelling this spring-gear behaviour would unnecessarily complicate the model, so we propose a hypothetical spring that provides equivalent return torque as the rest of the mechanism. This virtual spring is implemented as a torsional spring with predefined stiffness and

home position in the MJCF file format.

θ	l
0°	13.625 mm
90°	4.20 mm

Table 2.2: Experimental Joint-spring length data

Given the actual robot data about the joint angle and spring length as shown in Table 2.2, a linear interpolation can be performed to get a relationship between the joint angle and the spring length,

$$l = 13.625 - 0.104722\theta$$

From the manufacturer specs, $k = 0.04lbs/mm = 177.93N/m$, and the uncompressed length of the spring is 19.8mm. The change in length from the uncompressed position for a given θ by,

$$\Delta x = 19.8 - (13.625 - 0.104722\theta)$$

Since two springs act on the pinion, the total torque exerted on the pinion is, $\tau = 2k\Delta xr$. Replacing the numerical values we get,

$$\tau = 2135.16 \times 10^{-6} \times (6.175 + 0.104722\theta)$$

At 0°, $\tau = 0.0132Nm$ Likewise, at 90°, $\tau = 0.0333Nm$ When the spring is uncompressed, the hypothetical angle that would be the spring reference would be;

$$19 = 13.625 - 0.104722\theta \Longrightarrow \theta = -58.97^\circ = -1.029rad$$

Hence, for the virtual torsional spring;

$$\tau = K\Delta\theta \;\; or \; 0.0132 = K \times (0 - (-1,029)) \Longrightarrow K = 0.01281Nm/rad$$

Chapter3

Approach

3.1 Notion and Definitions

Throughout this thesis we consider the world as a finite horizon, discounted Markov Decision Process (MDP) defined by a state-space $S \in \mathbb{R}^n$, action-space $A \in \mathbb{R}^m$, reward function $R : S \times A :\rightarrow \mathbb{R}$, a state transition function $T : S \times A \rightarrow S$, and a discount factor $\gamma \in [0, 1)$ We define the cost function $C : S \times A \rightarrow R$ as the negative of the MDPs reward function: $c(s, a) = -r(s, a)$

3.1.1 BATA MDP

The MuJoCo model of BATA integrated with the OpenAI Gym framework [20] gives us a discrete formulation of the MDP in simulation. The task is an episodic task which ends after 1000 steps, each time step being 0.01 s in real world clock. The state S is described as $S \in \mathbb{R}^{25}$. MuJoCo simulator directly gives us access to all of these states which include the object position, orientation, linear and angular velocity, joint positions and velocity.

The action A is described as $A \in \mathbb{R}^2$, which corresponds to the tendon force applied on each finger. The action is normalized i.e. $A \in [-1, 1]$, so as to ensure efficient sampling from the Gaussian. This normalized action maps to [-10,0] as the pulling force in the tendon. Negative sign meaning tendons can only be pulled. The state transition function, $T : S \times A \rightarrow S$ is given by the MuJoCo Physics engine.

Based on this setting of the MDP, the goal is to develop a policy that manipulates the

object from the start location to the goal location taking into account the following,

- Large distance to the goal pose must be penalized
- Unnecessary and expensive actions should be penalized
- The object should stay upright throughout the manipulation
- The object must be in contact with both the finger tips i.e. the condition of in-hand manipulation has to be met throughout

Based on these qualitative requirements we formulate the reward function $R : S \times A :\rightarrow \mathbb{R}$, as:

$$- w_1 \| O_{goal} - O_{object} \|_2 - w_2 \sqrt{u_1^2 + u_2^2} - w_3 (1 - {q_{obj}}^T q_{goal}) - w_4 1_{!contact} \tag{3.1}$$

The first term in equation 3.1, penalizes the euclidean distance to the goal, second term penalizes unnecessary action, the third term penalizes the dot-product of the current orientation and the desired orientation. The fourth term is interesting as in it is an indicator function which is 1 when there is no contact between the object and the fingertips, else it's 0. This essentially enforces the in-hand manipulation constraint in the manipulation task.

3.2 Model Free Reinforcement Learning

Model of the environment in RL is the transition function T, which predicts state transitions and rewards. A model-free algorithm is an algorithm that estimates the optimal policy without using or estimating the dynamics (transition and reward functions) of the environment [21]. In practice, a model-free algorithm either estimates a value function or the policy directly from experience i.e. the interaction between the agent and environment.

3.2.1 Policy Gradient Algorithms

The goal of reinforcement learning is to find an optimal behavior strategy for the agent to obtain optimal rewards. This is mathematically expressed as finding a policy $\pi : S \rightarrow A$, which maximizes the sum of discounted future rewards or expected return J as shown in equation 3.2. Policy gradient methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to θ as $\pi_\theta(a|s)$ The value of the reward function depends on this policy and then various algorithms can be applied to optimize for the best reward. [21]

$$J = \underset{\theta}{\text{maximize}}\, \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right] \tag{3.2}$$

The objective above can be expressed in a trajectory setting where $\tau \sim \pi_\theta$ and the reward of the trajectory is given by $R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t$. Using policy gradient theorem and the log-likelihood ratio trick, the gradient of the objective can be calculated as [22, 23];

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau) \cdot \nabla_\theta \left(\sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)\right)\right] \tag{3.3}$$

A baseline is included in equation 3.3 to reduce the high variance in the gradient estimates. This baseline which is often a value function assists in the policy updates, hence the two main components in policy gradient algorithms are the policy model and the value function. Actor Critic methods [24] learn the value function in addition to the policy, reducing gradient variance in vanilla policy gradients. Actor-critic methods thus consists of two models, which may optionally share parameters

- Critic updates the value function parameters w which depending on the algorithm it could be action-value, state-value or advantage estimate.
- Actor updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the

critic.

3.2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a trust region based on-policy actor-critic method [25]. On policy in the sense that the behaviour policy which is the policy used for data generation is used for action selection as well. Trust region methods [26] improve training stability, by avoid parameter updates that change the policy too much at one step. This is implemented by enforcing a KL divergence constraint on the size of policy update at each iteration. PPO simplifies the trust region approach by removing the KL penalty and need to make adaptive updates, while simultaneously retaining similar if not better performances compared to TRPO.

$$J^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a))] \tag{3.4}$$

PPO optimizes a clipped surrogate objective [25] as in equation 3.4 to do a trust region update update which is compatible with Stochastic Gradient Descent. It imposes the constraint by forcing the ratio between the new and old policy known as policy ratio, $r(\theta)$ to stay within a small interval around 1, precisely $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter known as the clipping ratio. The function $clip(r(\theta), 1-\epsilon, 1+\epsilon)$ clips the ratio to be no more than $1+\epsilon$ and no less than $1-\epsilon$. The PPO objective function takes the minimum of the original value and the clipped version therefore discouraging large policy change if it is outside our comfortable zone.

3.2.3 BATA-MuJoCo PPO Implementation

Stable Baselines3 (SB3) [27], a set of reliable implementations of reinforcement learning algorithms in PyTorch is used to train the BATA-MuJoCo model based OpenAI Gym environment. A full connected neural network with 19 input observations, 2 hidden layers

of size 32 each and a output layer of 2 dimension is initialized. This policy network gives the distribution over actions given the current observation. Firstly, it collects a set of trajectories for each epoch by sampling from the latest version of the stochastic policy. Then, the rewards-to-go and the advantage estimates are computed in order to update the policy and fit the value function. The policy is updated via a stochastic gradient ascent optimizer, while the value function is fitted via some gradient descent algorithm. This iterative process is applied for many epochs until a stable policy with minimum cost and terminal error in the final state is acheived. Below, we explain the PPO applied to the BATA-MuJoCo simulation:

Algorithm 1: PPO applied to BATA-MuJoCo simulation environment

Data: Initial Policy parameters θ_0, value function parameters ϕ_0,clipping ratio ϵ, discount γ, GAE factor $\lambda \in [0, 1]$

for $K = 0, 1, 2, ...$ **do**

1. Collect set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in simulation
2. Compute the rewards-to-go $\hat{R}_t$
3. Estimate the advantages $\hat{A}_t$ using Generalized Advantage Estimate (GAE)

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V; \quad where \quad \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

4. Compute the policy update

$$\theta_{k+1} = \arg\max_{\theta} J^{\text{CLIP}}(\theta)$$

by taking K steps of minibatch Stochastic Gradient Descent (via Adam)

5. Fit value function by regression on mean squared error:

$$\theta_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

via gradient descent

end

3.3 Model Based Reinforcement Learning

Model-based reinforcement learning differs from it's model-free counterpart in the adoption of a dynamics model, which significantly effects the decisions making process downstream [28]. With access to a dynamics model, the agent can decide how to act by predicting into the future. With those actions, the agent collects more data, improves said model, and hopefully improves future actions. Model-based RL has a strong advantage of being sample efficient [29]. Many models behave linearly at least in the local proximity [30]. This requires very few samples to learn them. Likewise, once the model and the cost function are known, optimal controls can be generated without further sampling.

3.3.1 Model Predictive Control

Given the dynamics model along with physical constraints, model predictive control (MPC) is a useful technique for planning. MPC was initially developed as control stragey for controlling process industry with a slow plant dynamics [31]. But with the development of high performance computing hardware MPC has seen substantial application in robotics to control autonomous vehicles, quadrotors, and humanoids [32, 33, 34]

MPC seeks to compute a locally policy using knowledge of system dynamics. Consider π_{MPC} be the policy found by running some MPC algorithm for a N horizon starting at initial state s_0. If the stage cost and the terminal costs are given by q and ϕ respectively, the MPC policy can be expressed as: [35]

$$\pi_{MPC} = \underset{\pi_{\mathbf{0:N-1}}}{\arg\min}\, \mathbb{E}\left[\sum_{t=0}^{N-1} \gamma^t q(s_t, a_t) + \gamma^N \phi(s_N)\right] \tag{3.5}$$

A trajectory can be found by solving Equation 3.5 at successive time steps. Starting at the initial state s_0, one iteration of MPC can be run in order to calculate a policyπ_{MPC}. But rather than executing the entire actions for N steps from the policy only the first actiona_0 is

applied. Based on this action the MDP transitions to a new state s_1 following the transition model T. Subsequently, the previous equation can be solved to get the next controls. This process repeats until a series of actions defining a trajectory $\tau = \{st, at\}_{t=0}^{N-1}$ is generated.

Model-based MPC approaches remain relatively inflexible because they require carefully tuned cost functions, along with detailed comprehension of system dynamics [36].Despite these limitations, MPC remains one of the standard techniques in optimal control and trajectory optimization. Various flavour of MPC based optimization controllers are found in the literature. [37] In this thesis, we utilize the popular sampling based method MPC known as MPPI, which will be explained in further detail in the following sub-sections

3.3.2 Path Integral Control

Stochastic optimal control deals with controlling systems as shown in Equation 3.6, with both uncertainty in the action and sensor models, and the resultant state uncertainty [38].

$$\mathbf{dx} = \mathbf{f}(x_t, t)dt + \mathbf{G}(x_t, t)\mathbf{u}(x_t, t)dt + \mathbf{B}(x_t, t)dw \tag{3.6}$$

The sequence of control commands is found by minimising an integral of running cost along a given trajectory.

$$\min_{\mathbf{u}} \mathbb{E}_{\mathbb{Q}} \left[\phi(\mathbf{x_T}) + \int_t^T \left(q(\mathbf{x_t}, t) + \frac{1}{2}\mathbf{u}^T \mathbf{R}(\mathbf{x_t}, t)\mathbf{u} \right) dt \right] \tag{3.7}$$

Mathematically, it is defined by a Hamilton-Bellman- Jacobi (HJB) partial differential equation (PDE) corresponding to the system to be controlled. The stochastic Hamilton-Jacobi-Bellman equation for system in equation 3.6 with the cost function as expressed in

equation 3.7 is given as [39];

$$\begin{aligned} -\partial_t V = q(\mathbf{x_t}, \mathbf{t}) + \mathbf{f}(\mathbf{x_t}, t)^T V_x \\ - \frac{1}{2} V_x^T \mathbf{G}(\mathbf{x_t}, \mathbf{t}) \mathbf{R}(\mathbf{x_t}, t)^{-1} \mathbf{G}(\mathbf{x_t}, \mathbf{t})^T V_x \\ + \frac{1}{2} tr \left(\mathbf{B}(\mathbf{x_t}, \mathbf{t}) \mathbf{B}(\mathbf{x_t}, \mathbf{t})^T V_{\mathbf{xx}} \right) \end{aligned} \tag{3.8}$$

This can be solved numerically back- wards in time, given the system's initial and target configurations. For linear systems with quadratic costs this is relatively easy [40], but non-trivial for non-linear systems. However, by using the Feynman-Kac theorem, a non-linear HJB can be converted into a linear PDE, which can be solved via forward sampling of trajectories.[41, 42] This formulation can cope with arbitrary state costs that need not be differentiable, and is applicable to a wide range of non-linear systems.

3.3.3 Model Predictive Path Integral Control (MPPI)

Traditional path integral optimal control approaches struggle with respect to two aspects. The first has to do with the taking of expectations with respect to the uncontrolled dynamics of the system, which in high dimensional cases means that the probability of sampling a low cost trajectory can be incredibly small. Second, in systems with nonlinear dynamics, sampled trajectories run the risk of getting stuck in undesirable parts of the search space.

Williams et al [43] develops an approach that solves these problems, enabling modulation of both the mean and the variance of the sampling distribution without violating the fundamental assumptions in the path integral derivation. Their method, called model predictive path integral (MPPI) control, converges fast enough despite being applied in an MPC setting. [44]

The core idea in MPPI is to maintain a nominal trajectory and improve it through sampling. The sampling is done by simulating thousands of rollouts into the future. As shown in Figure 3.1, MPPI takes some large number N of trajectories—randomly sampled

off of some given state and perturbations in the controls. Each of these trajectories are scored based on their expected cost. MPPI then updates its control sequence by taking a softmin weighted average over these sampled and scored trajectories [45].

$$\mathbf{u_i} \leftarrow \mathbf{u_i} + \left[\sum_{k=1}^{K} \left(\frac{exp(\frac{-1}{\lambda} C(\tau_{i,k}))\delta u_{i,k}}{\sum_{k=1}^{K} exp(\frac{-1}{\lambda} C(\tau_{i,k}))} \right)\right] \tag{3.9}$$

The negative exponential term as shown in equation 3.9, gives trajectories with lower cost higher weights. In practice, this biases the updated control sequence toward those sampled trajectories with lowest cost.

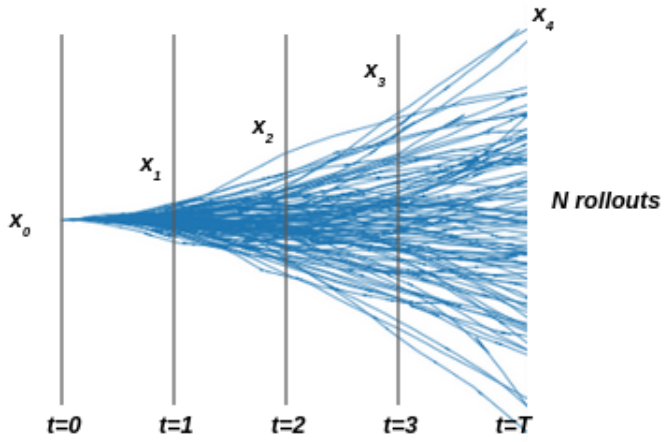


Figure 3.1: MPPI Rollouts of N trajectories over T horizon

We provide pseudocode of the MPPI algorithm (Algorithm 2) as found in both [43] and [46] for our application of in-hand manipulation using BATA.

3.3.4 BATA-MuJoCo MPPI Implementation

MPPI requires a dynamics model to rollout the N trajectories T steps into the future. In our work, this dynamics model is given by the MuJoCo physics engine. We generate 12 simulated SubProcVecEnv, which are essentially 12 different threads of model instances in the CPU cores. These are the hallucinated environments in addition to the simulation environment (sim-env) where the MPPI control policy will be applied. Each of these hallu-

cinated environment rollout 3 trajectories. At every control sampling time i.e 0.05 s, a total of 36 trajectories are rolled out for 20 timesteps in the future. The cost for each of these trajectory is calculated based on the previously defined cost function.

The weighted average of the controls is taken and added to the nominal controls to get the new control signal. This control signal is then applied to the sim-env. For the next time step, all the hallucinated environments are set to the state of sim-env and then again rolling out, evaluating the cost and generating a weighted average control signal. This iterative process continues till the end of the episode.

Algorithm 2: MPPI applied to BATA-Mujoco

Data: N, T, $(\mathbf{u_0}, \mathbf{u_1}, \mathbf{u_2}.....\mathbf{u_{N-1}})$, Σ, ϕ, q, λ
while *task not completed* **do**
  $\mathbf{x_0} \leftarrow GetState()$
  for $K \leftarrow 0$ **to** $K-1$ **do**
    $\mathbf{x} \leftarrow \mathbf{x_0}$;
    Sample δ;
    for $t \leftarrow 1$ **to** T **do**
      $x_{t+1} = x_t + ForwardModel(u_i + \delta u_{i,k})$;
      $C(\tau_{i,k})$= CostTrajectory() ;
    end
  end
  for $t \leftarrow 0$ **to** $T-1$ **do**
    $\mathbf{u_i} \leftarrow \mathbf{u_i} + \left[\sum_{k=1}^{K} \left(\frac{exp(\frac{-1}{\lambda}C(\tau_{i,k}))\delta u_{i,k}}{\sum_{k=1}^{K} exp(\frac{-1}{\lambda}C(\tau_{i,k}))}\right)\right]$;
  end
  send to actuators u_0;
  for $t \leftarrow 0$ **to** $T-2$ **do**
    $\mathbf{u_i} = \mathbf{u_{i+1}}$;
  end
  $\mathbf{u_{N-1}} = \mathbf{u_{init}}$;
  $UpdateState()$;
end

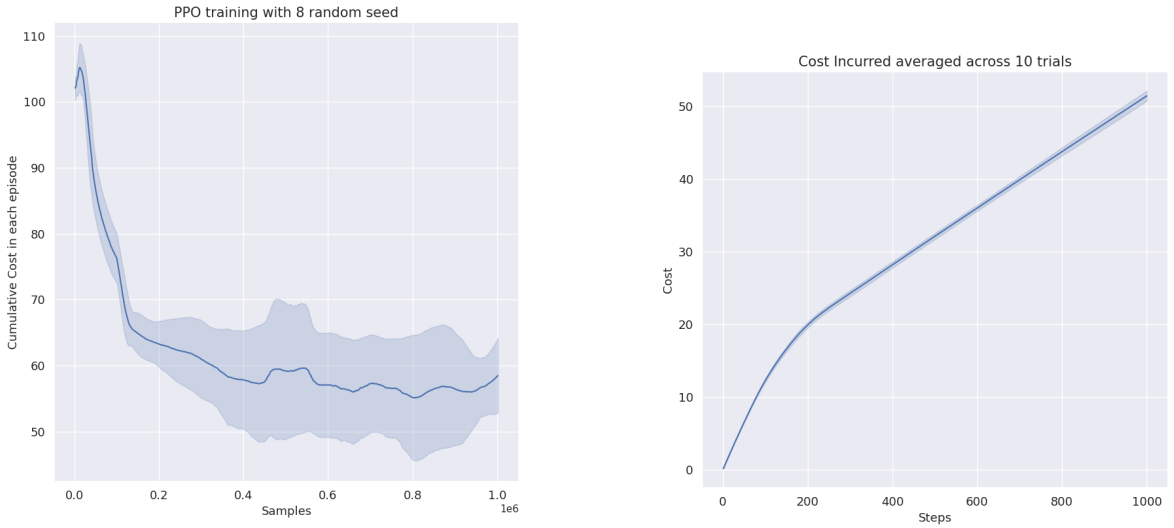
Chapter4

Results

4.1 PPO Results

4.1.1 Training

PPO is trained on the object with 100gm mass and 4cm radius, which we refer to as baseline trained condition. Figure 4.1 a) shows the PPO training with 8 random seeds. At about 400k timesteps the policy reaches the cumulative cost incurred at about 59. Even after a significant samples after this point, the policy doesn't show vast improvements hence the training is stopped at 1 million time steps.



(a) PPO Policy training over 8 trails

(b) Cumulative Cost Incurred by PPO Policy

Figure 4.1: PPO Performance in training and execution

After the policy has been trained, the policy is deployed in the simulation framework using the OpenAI gym like environment. Figure 4.1 b) shows the performance of the policy in the baseline trained condition. The policy incurs a total cumulative cost of 51.08 over the entire episode. Figure 4.2 shows the key frames of the executed PPO policy in MuJoCo. At the end of the episode the terminal error in the baseline trained condition is 3.8 cm.

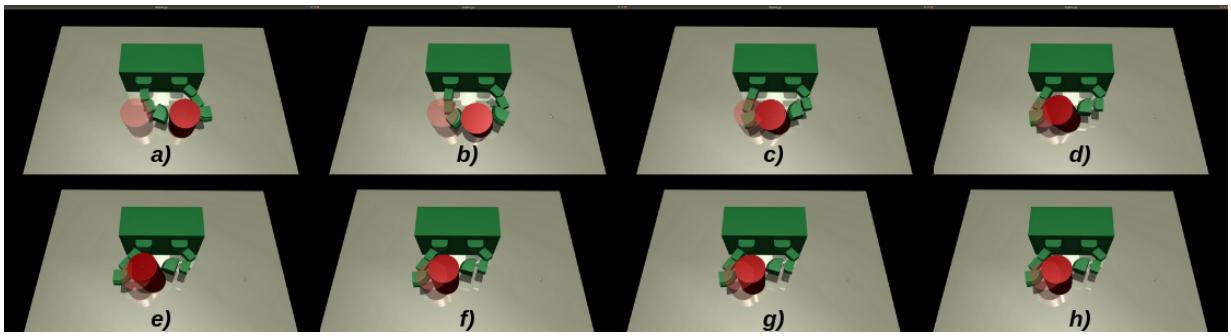


Figure 4.2: PPO Policy on object with 100gm and 4 cm

4.1.2 Evaluation in unseen conditions

In order to test how well PPO generalizes over untrained objects two experiments: one with varying mass and other with varying radius were performed. It is found that PPO generalizes well over the heavier objects and particularly fails in lighter object. The intuition here is since PPO is just trained on the baseline train distribution, the neural network fits this aggressive policy of rolling using both fingers. When using a lighter object this aggressive policy performs the same maneuver as a result of which the stable pinch grasp is disturbed and hence the object is out of grasp and falls down as shown in Figure 4.3

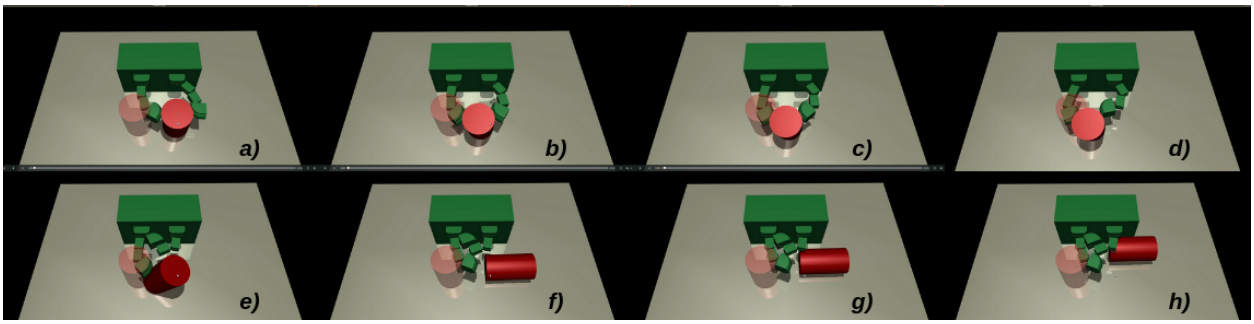
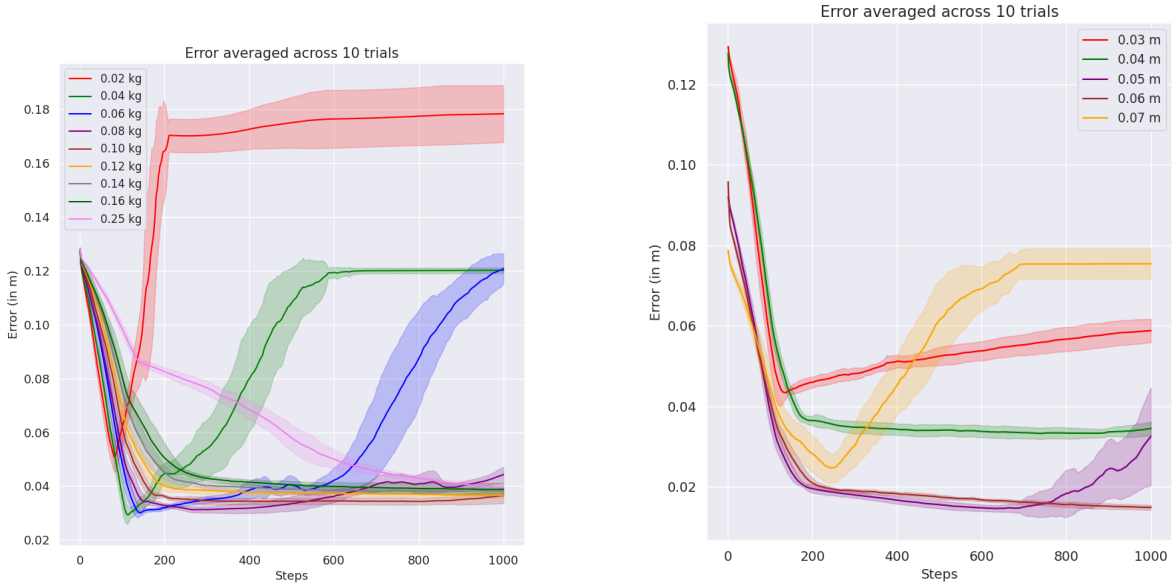


Figure 4.3: PPO Policy on object a lighter object with mass 20 gm

For object with heavier mass, the same aggressive policy is executed but the speed at which the policy is executed depends on the object mass as evident from the Figure 4.4 a). The heavier the mass after the trained baseline condition, the slower the response.



(a) Varying Mass (b) Varying Radius

Figure 4.4: PPO Simulation Results for unseen objects

Similarly, while testing the policy on object with different radius, the PPO generalization was found to be good only if the test object radius is comparable to the baseline trained radius. In particular, for object with smaller radius as shown in Figure 4.5, the object is manipulated by the links rather than the finger tips, thereby violating the in-grasp manip-

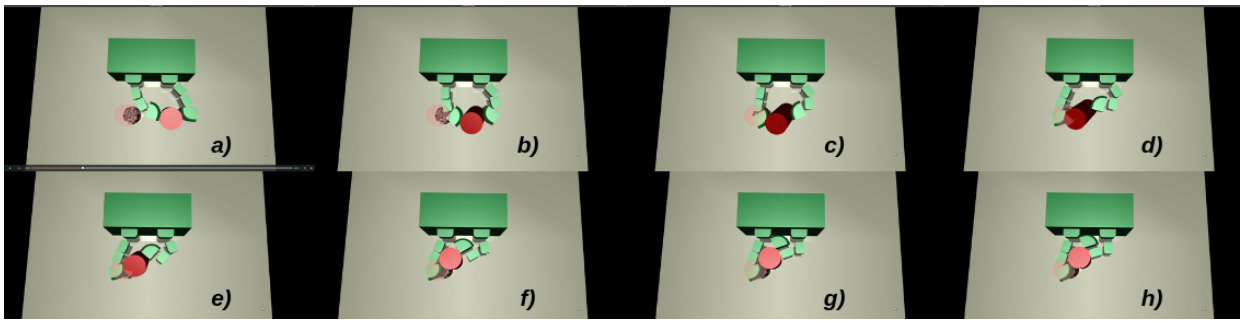


Figure 4.5: PPO Policy on object a lighter object with radius 3 cm

ulation which we enforced in the reward function. Likewise, for larger objects, the pinch grasp is disturbed and the object is ejected out of the reachable workspace.

Another interesting test we put the PPO policy is the symmetric initial pose and goal test. In this test, rather than asking the policy to move the object from left to right on which the baseline policy was trained we ask the policy to move the object from right to left. All other parameters of the test are kept same as the baseline training case except for the start and the goal location flipped.

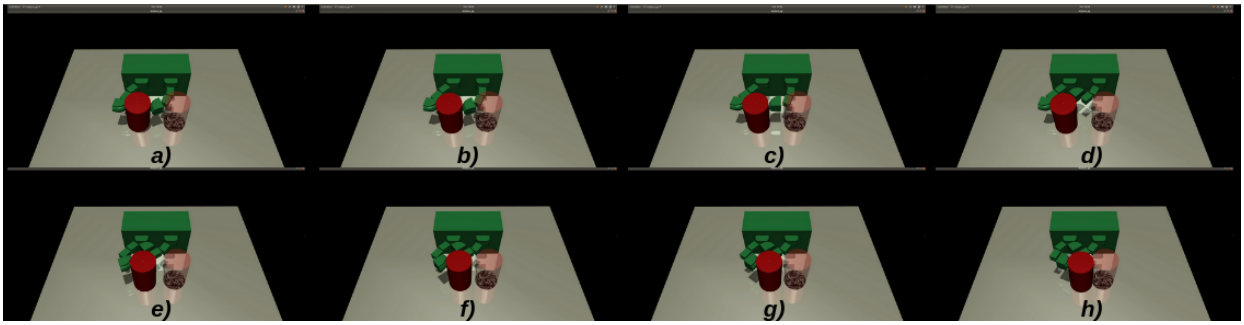


Figure 4.6: PPO Policy with initial and goal pose swapped

It doesn't come as a surprise as shown in Figure 4.6, the PPO policy failing to execute the task. The training distribution comes from just one instance of the pose and the neural network has never seen this symmetric goal configuration. Due to this, the PPO policy executes the same maneuver that it has "memorized", and there's no way it can achieve the goal with the same set of control actions that it had been using for the left-right combination.

4.2 MPPI Results

4.2.1 Training

Unlike PPO, where the policy is trained offline and then deployed to evaluate the accuracy metrics, MPPI is an online algorithm. Rather than training the policy, training in MPPI involves tuning the weights of the reward function to a particular baseline train condition i.e. reward shaping to attain the goal in a particular test case. For training the MPPI we use the same object parameters: 100 gm mass and 4 cm radius, as with PPO.

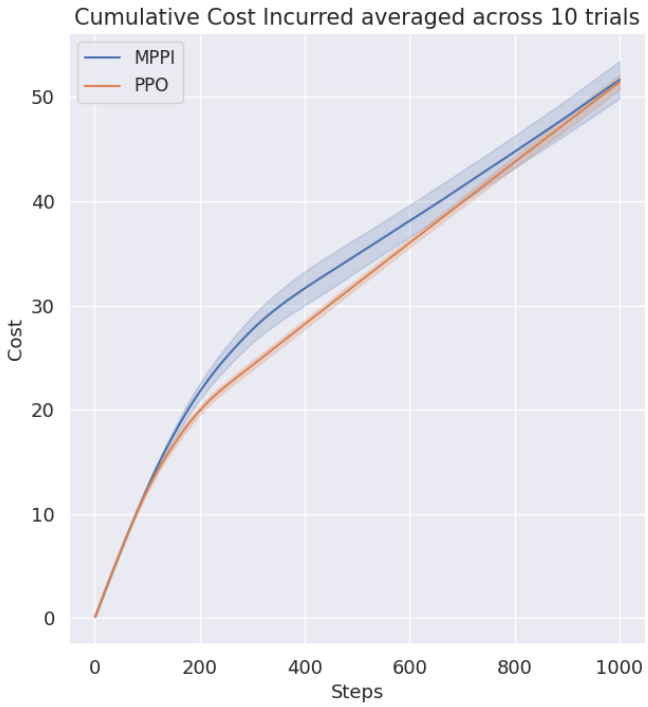


Figure 4.7: Cumulative Cost Incurred by MPPI against PPO

As shown in Figure 4.7, both MPPI and PPO incurs similar costs over the entire episode. MPPI in particular, being an online algorithm and the way the initial perturbation in the actions are selected shows more variance than the PPO, which is essentially executing the same policy again and again.

4.2.2 Evaluation in unseen conditions

Despite incurring similar costs in the test condition, we wanted to test how well MPPI adapts to change in the object mass and radius. We test MPPI with the cost function optimized for the base train condition with objects of different mass and radius.

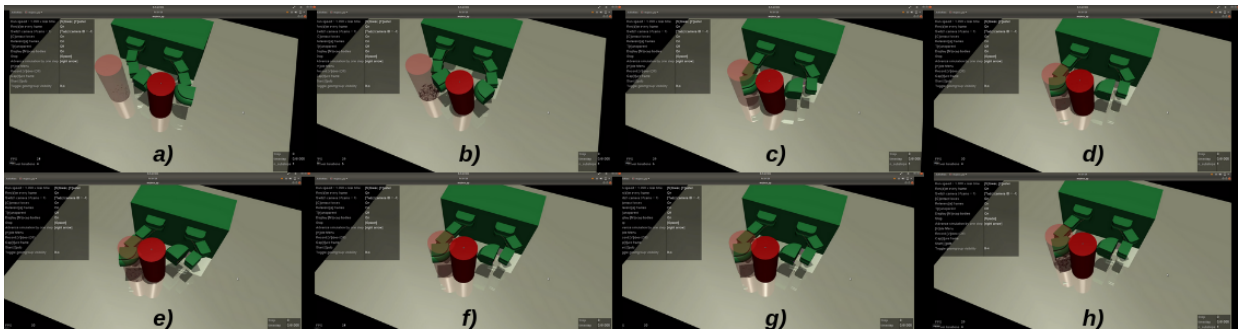


Figure 4.8: MPPI Policy on object a lighter object with radius 3 cm

Interestingly, MPPI retains similar performance even in varying radius condition. For object with 3 cm radius, PPO dragged the object aggressively, which caused the fingertips to lose contact with object, thus losing the in-hand manipulation. MPPI on the other hand, maintains the contact of the object with both the fingers as specified in the reward function. As shown in Figure 4.9, MPPI policy is able to complete the task with a terminal error of 4.8 cm, while maintaining the grasp until the end of the episode.

As shown in Figure 4.9, we get small terminal errors in object with radius 5cm. This turns out to be the optimized object radius to be manipulated with our current BATA robot kinematics. The finger always maintain the stable pinch grasp in this configuration, thus MPPI is consistently able to perform well across several trails, sometimes even hitting less than 15 mm errors. The box plots and the error-steps graph also show as the object radius increases after this 5 cm threshold, the terminal error and it's variance across the trails increases. The insight here is that, as the object radius increases significantly, the fingertips lose that grasping ability therefore ejecting the object out of the reach.

(a) Distance error along the timesteps

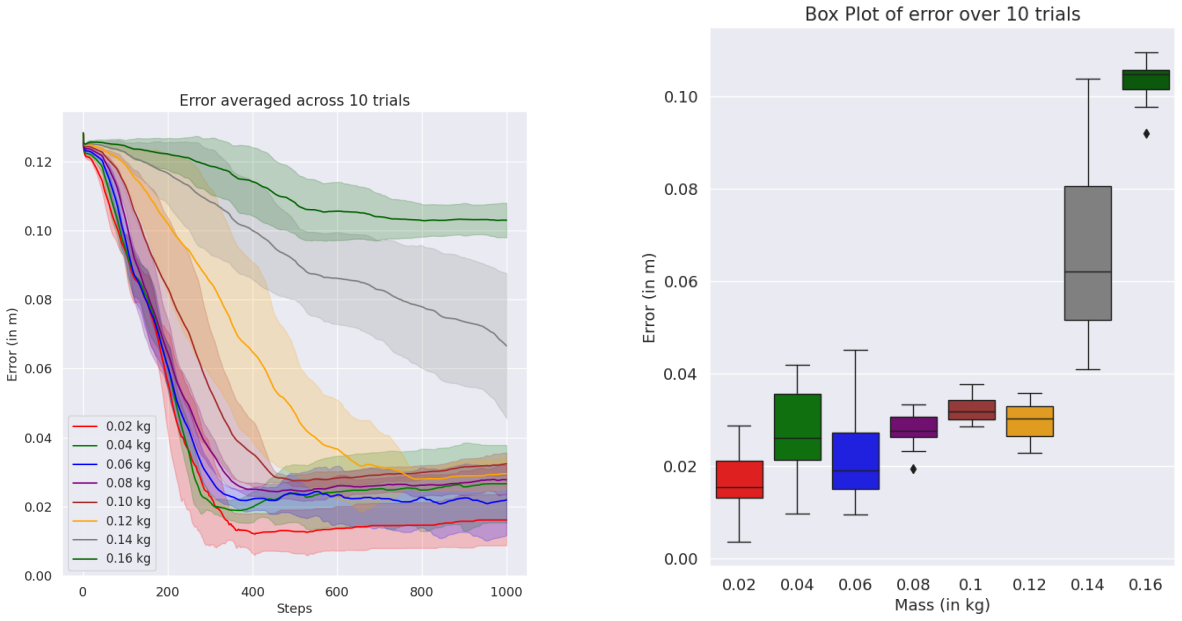
(b) Variance in terminal error

Figure 4.9: MPPI Simulation Results for object with varying radius

When MPPI was tested in it's current setting for objects with smaller masses, the terminal error was comparable and often less than condition in which the cost was optimized for. This indicates the generalization of MPPI for range of diverse masses. For objects with smaller weights, an interesting emergent behaviour of using one of the finger tips for fine manipulation was observed. This behaviour gives smaller terminal errors as the episode ends as is evident from Figure 4.10.

For heavier objects, MPPI's policy doesn't seem aggressive enough. Although we see a decreasing trend in the error metric, 1000 timesteps is just not enough for the MPPI policy to bring the object to a decent terminal error. Throughout the simulation we set up this artificial constraint that MPPI has to give a new control command every 0.05 s i.e 20 Hz frequency at which the actual robot control will be executed. In our current setting, this constraint forced us to limit the horizon to 20 timesteps into the future rolling out 36 trajectories. It seems that with this particular setup, MPPI is very short-sighted and doesn't

take into account the rewards of aggressive action into the future for heavier objects.



(a) Distance error along the timesteps

(b) Variance in the terminal error

Figure 4.10: MPPI Simulation Results for object with varying mass

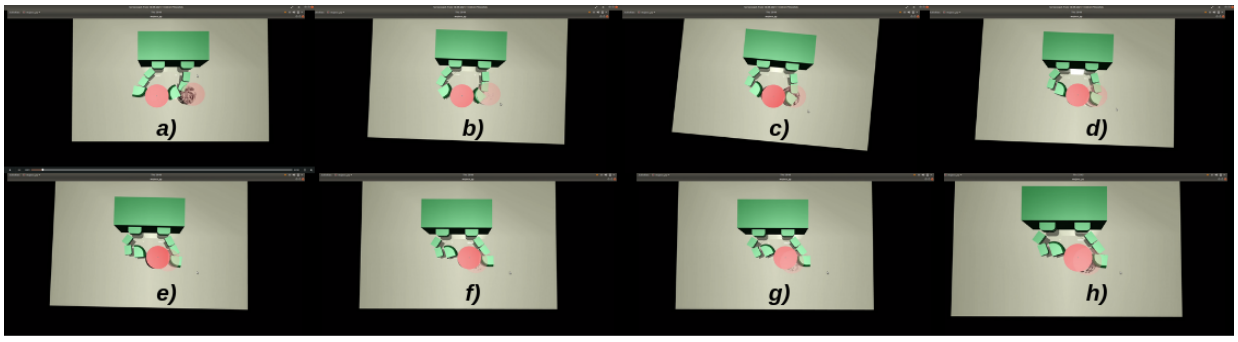


Figure 4.11: MPPI Policy with initial and goal pose swapped

Likewise in the symmetric case where PPO fails miserably, MPPI is able to generalize well. Since the model and the cost function encodes the intrinsic of the problem being solved, MPPI performs the symmetric goal swap test just as well as the baseline train condition as shown in Figure 4.11

Chapter5

Conclusion

We first introduced Brake Assisted Tendon Actuator (BATA) as a novel gripper towards dexterous underactuated in hand manipulation. Kinematic model based on the DH formulation and a dynamic gray box model based on the actual hardware was developed in MuJoCo. A discrete MDP was setup using the OpenAI gym framework and detail investigation of the model based and model free techniques for underactuated in hand manipulation was performed. The current work lays a framework for controller implementation in the novel BATA gripper.

Testing in several untrained cases, we were able to demonstrate the robustness of the developed policy for in hand manipulation. In particular we demonstrated, MPPI, a model based technique was able to generalize across wide range object masses and radius compared PPO, which always executed to a memorized aggressive policy. Results from the experiments without electrostatic braking from this work will serve as baseline to evaluate the advantages of electrostatic braking for underactuated in hand manipulation in future work.

5.1 Limitations

The initial pose of the object and the finger configuration were chosen manually in the current work. In absence of an oracle to asses the grasp stability, we had to resort to qualitative assessment to see if the pinch grasp was stable and whether in grasp manipulation would be feasible for the a given combination of initial pose and goal pose. This severely limits the data-set generation, hence the training distribution on which PPO is trained on

is limited.

The constraint of control frequency of 20 Hz in the real robot, forced us our MPPI implementation to complete the entire rollout, cost evaluation and generation of new controls using softmin weighted average in under 0.05 seconds. With the current framework and CPU based MuJoCo model, we are limited by the number of cores in the machine on which MPPI is being executed. Sampling based MPC like MPPI work best when the number of rollouts are in the order of thousands. These computationally intensive rollouts therefore sets an immediate need for GPU based parallelization for superior performance.

5.2 Future Work

The limitation of initial conditions can be addressed using a grasp planner like GraspIt! [47] This will help generate a range of stable grasps for a given object pose and current hand configuration. Using an external grasp planner will significantly increase the library of the initial and goal poses to train the reinforcement learning algorithm in. We believe PPO can be made robust enough by generating a wide distribution of initial poses on which it is trained.

Likewise, rather than relying on CPU cores to rollout the policies, GPUs can be utilized to achieve massive parallelization. Recent works [48] have shown GPU based Physics simulators demonstrating complex RL training process in simulation to be orders of magnitude faster than CPU based models. MuJoCo doesn't support GPU physics simulation, therefore considering the scope of the project the current MCJF model can be migrated to the Nvidia Issac Gym environment. Stable implementations of both model based RL and model free RL is available in the Nvidia Omniverse ecosystem. [49, 50]

On the hardware side, we have started some work recently. Two pipeline compose the overall hardware implementation, the first is the state estimation and second is the controller implementation. The simulation experiments relied on the fact that object pose and joint

position are always available. The later is easily provided by the encoders, but the former is the major challenge. As a first step to validate the simulation results, we have begun work on setting up the Vicon motion capture system to query for the object position and orientation in the space. But as the project progresses, state estimation using embedded proximity sensors remains the major challenge. Our initial work has shown some interesting results using non-least squares methods to estimate the cylinder pose, however we believe a Bayesian filter like a particle filter is the key to robust state estimation in BATA.

Bibliography

[1] A.M. Okamura, N. Smaby, and M.R. Cutkosky. "An overview of dexterous manipulation". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 255–262 vol.1. DOI: 10.1109/ROBOT.2000.844067.

[2] Raymond R. Ma and Aaron M. Dollar. "On dexterity and dexterous manipulation". In: *2011 15th International Conference on Advanced Robotics (ICAR)*. 2011, pp. 1–7. DOI: 10.1109/ICAR.2011.6088576.

[3] Hong Liu et al. "Multisensory five-finger dexterous hand: The DLR/HIT Hand II". In: *2008 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2008, pp. 3692–3697.

[4] XH Gao et al. "The HIT/DLR dexterous hand: Work in progress". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*. Vol. 3. IEEE. 2003, pp. 3164–3168.

[5] Antonio Bicchi, J Kenneth Salisbury, and David L Brock. "Experimental evaluation of friction characteristics with an articulated robotic hand". In: *Experimental Robotics II*. Springer, 1993, pp. 153–167.

[6] OpenAI: Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[7] Lael U Odhner et al. "A compliant, underactuated hand for robust manipulation". In: *The International Journal of Robotics Research* 33.5 (2014), pp. 736–752.

[8] Berk Calli and Aaron M. Dollar. "Vision-based model predictive control for within-hand precision manipulation with underactuated grippers". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2839–2845. DOI: 10.1109/ICRA.2017.7989331.

[9] Raymond R Ma, Lael U Odhner, and Aaron M Dollar. "Dexterous manipulation with underactuated fingers: Flip-and-pinch task". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3551–3552.

[10] Berk Calli and Aaron M. Dollar. "Vision-based precision manipulation with underactuated hands: Simple and effective solutions for dexterity". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1012–1018. DOI: 10.1109/IROS.2016.7759173.

[11] Berk Calli et al. "Learning Modes of Within-Hand Manipulation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3145–3151. DOI: 10.1109/ICRA.2018.8461187.

[12] Anna Kochan. "Shadow delivers first hand". In: *Industrial robot: an international journal* (2005).

[13] Huan Liu et al. "The mero hand: A mechanically robust anthropomorphic prosthetic hand using novel compliant rolling contact joint". In: *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE. 2019, pp. 126–132.

[14] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[15] John J Craig. *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[16] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

[17] Tom Erez, Yuval Tassa, and Emanuel Todorov. "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 4397–4404.

[18] Lenka Pitonakova et al. "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators". In: *Annual Conference Towards Autonomous Robotic Systems*. Springer. 2018, pp. 357–368.

[19] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[20] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[23] Shalabh Bhatnagar et al. "Natural actor–critic algorithms". In: *Automatica* 45.11 (2009), pp. 2471–2482.

[24] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[25] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[26] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[27] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[28] Lukasz Kaiser et al. "Model-based reinforcement learning for atari". In: *arXiv preprint arXiv:1903.00374* (2019).

[29] Athanasios S Polydoros and Lazaros Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics". In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173.

[30] Yuval Tassa, Nicolas Mansard, and Emo Todorov. "Control-limited differential dynamic programming". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1168–1175.

[31] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[32] Moses Bangura and Robert Mahony. "Real-time model predictive control for quadrotors". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 11773–11780.

[33] Tom Erez et al. "An integrated system for real-time model predictive control of humanoid robots". In: *2013 13th IEEE-RAS International conference on humanoid robots (Humanoids)*. IEEE. 2013, pp. 292–299.

[34] Mario Zanon et al. "Model predictive control of autonomous vehicles". In: *Optimization and optimal control in automotive systems*. Springer, 2014, pp. 41–57.

[35] Kendall Lowrey et al. "Plan online, learn offline: Efficient learning and exploration via model-based control". In: *arXiv preprint arXiv:1811.01848* (2018).

[36] Carlos E Garcia, David M Prett, and Manfred Morari. "Model predictive control: Theory and practice—A survey". In: *Automatica* 25.3 (1989), pp. 335–348.

[37] James B Rawlings. "Tutorial overview of model predictive control". In: *IEEE control systems magazine* 20.3 (2000), pp. 38–52.

[38] Wendell H Fleming and Raymond W Rishel. *Deterministic and stochastic optimal control*. Vol. 1. Springer Science & Business Media, 2012.

[39] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[40] D Subbaram Naidu. *Optimal control systems*. CRC press, 2002.

[41] Hilbert J Kappen. "Path integrals and symmetry breaking for optimal control theory". In: *Journal of statistical mechanics: theory and experiment* 2005.11 (2005), P11011.

[42] Satoshi Satoh, Hilbert J Kappen, and Masami Saeki. "An iterative method for nonlinear stochastic optimal control based on path integrals". In: *IEEE Transactions on Automatic Control* 62.1 (2016), pp. 262–276.

[43] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. "Model predictive path integral control: From theory to parallel computation". In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357.

[44] Grady Williams et al. "Aggressive driving with model predictive path integral control". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1433–1440.

[45] Mohak Bhardwaj, Sanjiban Choudhury, and Byron Boots. *Blending MPC Value Function Approximation for Efficient Reinforcement Learning*. 2021. arXiv: 2012.05909 [cs.LG].

[46] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. "Model predictive path integral control using covariance variable importance sampling". In: *arXiv preprint arXiv:1509.01149* (2015).

[47] Andrew T Miller and Peter K Allen. "Graspit! a versatile simulator for robotic grasping". In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122.

[48] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021.

[49] Nikita Rudin et al. *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*. 2021.

[50] Arthur Allshire et al. "Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger". In: *arXiv preprint arXiv:2108.09779* (2021).