University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Benchmarking Imitation learning algorithms for robotic manipulation

Gunraj Gulati

March 24, 2023

Abstract

This dissertation compares three imitation learning algorithms, Behavioural Cloning (BC), Behavioral Cloning with Recurrent Neural Networks (BC-RNN) and Hierarchical Behavioral Cloning (HBC), for robotic manipulation tasks. Furthermore, extensive datasets from robomimic by Mandlekar et al. (2021b), and robosuite by Zhu et al. (2020) were used, which provide extensive examples of each algorithm's performance. With these comprehensive datasets, an accurate evaluation of each algorithm can be made.

This research examines the strengths and weaknesses of each algorithm, as well as their suitability for various manipulation tasks, by comparing BC, BC-RNN, and HBC's performance on different types of datasets. This provides valuable insights into robotic manipulation and emphasizes the significance of selecting an appropriate imitation learning algorithm based on task requirements.

Education Use Consent

Contents

1 Introduction

This chapter provides the motivation and the aims to do this project.

1.1 Motivation

Robotic manipulation is a crucial area in robotics research, with applications ranging from manufacturing and assembly to housework and healthcare cite4141029. According to Ross et al. (2011), creating algorithms that enable robots to learn from human demonstrations to complete complex tasks more quickly and precisely is one of the critical challenges in robotic manipulation.

According to Ho and Ermon (2016b), imitation learning algorithms have demonstrated great promise. These methods enable robots to gain complex abilities from observing human behaviours without explicit task programming, leading to intense debate in robotics research over recent years Yu et al. (2018).

Though imitation learning may have several advantages, several issues must be addressed before being effectively utilized. According to Goodman et al. (2016), Designing algorithms that can learn from various demos with noisy or partial information and flexibility to generalize to new circumstances must be paramount. Furthermore, effective methods must be found for accurately assessing and contrasting various imitation learning algorithms to pinpoint their advantages and drawbacks.

1.2 Aims

This project seeks to benchmark imitation learning algorithms for robotic manipulation. According to Fang et al. (2019), Imitation learning is a subfield of machine learning that involves learning from examples, wherein an algorithm models how to replicate specific actions or events. This project seeks to evaluate the performance of different imitation learning algorithms on a robotic manipulation task by training them on datasets with experts' demonstrations and comparing their success in performing it in simulation tasks. The objectives are:

1. To assess the accuracy and speed of learning using various imitation learning algorithms.

2. To test the robustness of algorithms

3. To test the scalability of our algorithms on more challenging datasets.

4. Acquaint the limitations and potential drawbacks of imitation learning for robotic manipulation tasks.

Benchmarking the performance of these algorithms will provide valuable insight into the strengths and shortcomings of different approaches, helping to inform the development of more efficient methods for imitation learning in robotics.

2 Background

This chapter will discuss the essential methods, tools and research related to the topic.

2.1 Materials and Methods

This section explains the material and methods used for training algorithms.

2.1.1 Panda robot

The panda robotic arm, developed by Franka Emika, is a versatile and sophisticated platform with much research attention. It features 7 DOF(Degrees of freedom) and can perform several manipulation tasks with unmatched precision. According to Emika (2021), the arm also comes with torque sensors at each joint, enabling it to interact safely with its environment. Its modular structure, open-source software, and compatibility with the Robot Operating System (ROS), according to Emika (2021), make it the ideal platform for various tasks, including assembly, manipulation, and interaction with people. The Figure 2.1 shows how the panda robotic arm looks like

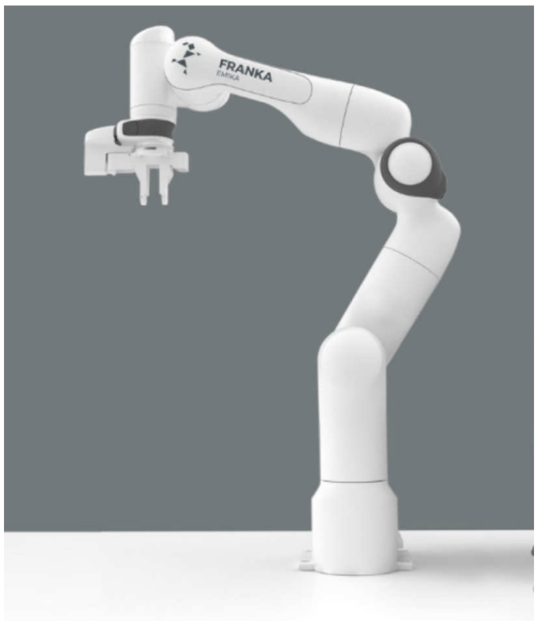


Figure 2.1: *Franka Emika robotic arm - Panda. Image taken from London*

2.1.2 Gaussian Mixture Model (GMM)

According to Riaz et al. (2020), Gaussian Mixture Model (GMM) is a probabilistic model used for modelling continuous data by assuming it comes from multiple Gaussian distributions. Under GMM, each data point is generated from one of K's different Gaussian distributions with different mean and variance values.

The GMM consists of two major components: the mixture model and Gaussian distribution, as explained by Chellappa et al. (2009). The mixture model specifies which data point belongs to each K Gaussian distribution, while each Gaussian component's probability density function defines its probability density function.

The parameters of a GMM are the means, variances and mixing coefficients of K Gaussian distributions. These parameters are typically estimated from data using an Expectation-Maximization (EM) algorithm, which iteratively updates them until convergence according to Moon (1996).

According to Tian (2018), the GMM has numerous applications in fields such as image processing, computer vision, speech recognition and bioinformatics. In image processing the GMM assists with segmentation and object tracking, while computer vision uses it for face recognition, object recognition and scene analysis. Speech recognition uses GMM for speaker identification and segmentation, while bioinformatics uses it to identify gene expression patterns and analyse DNA sequences.

One advantage of the GMM is its versatility in modelling complex data distributions. Unlike other models such as linear regression and logistic regression, the GMM does not assume a linear relationship between features and response variables. Furthermore, it can capture non-linear relationships between features and responses by applying non-linear transformations to the features.

However, GMM has several drawbacks. According to Moon (1996), one issue is that its EM algorithm is highly sensitive to initial parameters, leading to different local maxima. Furthermore, it assumes data comes from a mixture of Gaussian distributions which may not be appropriate for all types of data sets.

Finally, the Gaussian Mixture Model (GMM) is an effective tool for modelling complex data distributions and has numerous applications across different fields. The GMM's capacity to model non-linear relationships between features and the response variable makes it a popular choice for many data analysis tasks.

As part of the study, the BC(Behavioural cloning) algorithm is trained with the GMM model to see if the algorithm benefits from GMM enabled. The success rate is compared to when it is not enabled, as shown in the evaluation section.

2.1.3 Recurrent Neural Network (RNN)

According to Lipton et al. (2015), Recurrent Neural Network (RNN) is an artificial neural network that processes sequential data. Unlike traditional feedforward neural networks, which only process input data in one direction, RNNs maintain an internal state that allows them to process inputs in a temporal sequence. This capacity to remember past inputs and use them to influence future processing makes RNNs ideal for tasks such as natural language processing, speech recognition, and time series prediction.

An RNN's primary characteristic is its recurrent connection, which allows information to flow between steps in a sequence. To accomplish this, engineers add a feedback loop into the network architecture, which feeds back each output as an input for subsequent processing. In essence, this feedback loop creates a memory within the network by preserving knowledge about past inputs and using that insight to shape future processing decisions.

RNNs come in various architectures and training methods. According to Graves (2012), one popular type is the Long Short-Term Memory (LSTM) network utilises gated cells to control information flow through it. On the other hand, Dey and Salem (2017) says that Gated Recurrent Unit (GRU) networks utilize a simplified version of LSTM architecture.

One challenge in training RNNs is the vanishing gradient problem, which occurs when gradients become very small as they move through the network's feedback loop. This makes it difficult for the network to learn long-term dependencies among data. Various techniques have been developed to combat this issue, such as gradient clipping and alternative activation functions.

RNNs have proven to be highly effective in a variety of applications. They excel at tasks involving sequential data and have achieved state-of-the-art results in many natural languages processing tasks such as language translation and sentiment analysis. As more data becomes available and more sophisticated architectures are developed, RNNs will likely continue to play an increasingly significant role in AI research.

RNN has been used as part of the study combined with Behavioural cloning(BC), as an algorithm in the form of BC-RNN to see if it is a better alternative to BC. The success rates are compared between these algorithms, as mentioned in the evaluation section.

2.1.4 Variational Autoencoder (VAE)

According to Odaibo (2019), Variational Autoencoder (VAE) is a type of generative model that utilizes neural networks and probability theory to generate a latent space that accurately depicts the distribution of input data. VAEs have become widely used in deep learning applications such as unsupervised learning, image generation, and data compression.

VAE stands out because it learns a probability distribution over input data instead of just mapping from input to output. To do this, the VAE introduces a latent variable z, representing a lower-dimensional input data representation. The VAE then maps input x onto this latent variable z and back again using two neural networks - encoder and decoder - parameterized by two neural networks according to Zhang et al. (2019).

Training an encoder network involves mapping input data to a latent variable z, while decoding, it returns it to the output data. The aim is to minimize the difference between output data and reconstructed values; however, since VAE is a probabilistic model, it's not just about minimizing reconstruction errors but also learning a distribution over z itself.

To achieve this goal, the VAE introduces a regularization term that encourages the learned distribution over z to be close to a predefined prior distribution (usually a normal distribution). According to Johnson and Sinanovic (2001), this regularization term is known as Kullback-Leibler (KL) divergence, and it ensures that the learned distribution over z is both close to the prior and accurately captures the structure of input data.

One key characteristic of VAEs is their capacity for creating new samples from the learned distribution over a latent variable, z. This is accomplished by sampling from this distribution and feeding it into a decoder network in order to generate fresh data samples.

VAEs have been employed in various applications, such as image generation, data compression and speech recognition. Their effectiveness lies in their capacity for capturing the underlying structure of input data and producing new samples from it. With the increasing availability of data and advances in neural network architectures, VAEs will likely continue to play an increasingly significant role in deep learning and artificial intelligence research.

As part of the study, the HBC algorithm is trained with the VAE model and hence explained in this section of the study.

2.1.5 Markov decision process

The Markov Decision Process (MDP) is a mathematical model used to study sequential decision-making in scenarios where the outcomes are partly random and partly controlled by the decision-maker. According to Banerjee (2021) For, reinforcement learning (RL) it is a foundational element and is used to formalize decision-making problems.

According to Shin and Lee (2015) MDP requires the following specifications:

-State variables, s, and a finite state space S

-Decision variables, a, and a finite action space A

-Exogenous information variables, Wk, and a finite exogenous space Ω

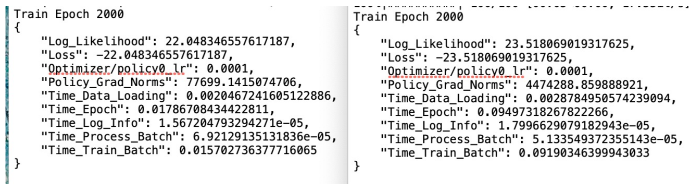
-Stage-wise cost function, c(s, a)

-Transition function, s+1 = SM(s, a, Wk+1)

Imagine a Markov Decision Process that simulates robot manipulation in discrete time with an indefinite horizon (MDP) (explained in the background section). $M = (S, A, T, R, \gamma, \rho_0)$, where S is the state space, A is the action space, $T(\cdot|s, a)$ is the state transition distribution, $R(s, a, s')$ is the reward function, $\gamma \in [0, 1)$ is the discount factor, and $\rho_0(\cdot)$ is the initial state distribution, are used to describe the MDP.

In each step, an agent observes the current state s_t decides on an action $a_t = \pi(s_t)$ in accordance to a policy π, monitors the next state $s_{t+1} \sim T(\cdot|s_t, a_t)$ and the reward $r_t = R(s_t, a_t, s_{t+1})$ is given. The goal is to maximise the expected return : $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})]$, by learning a policy π as described in Mandlekar et al. (2021a). The Algorithm BC works in accordance to MDP as a policy π needs to be learned to clone the actions of the provided demonstrations in the dataset it is trained on.

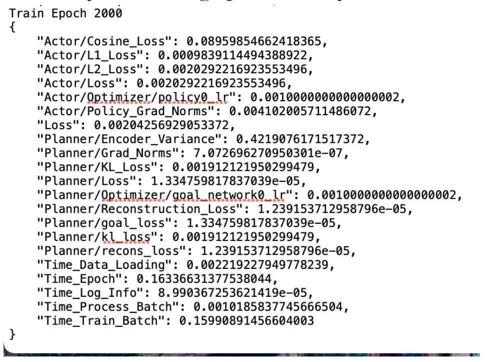
2.1.6 Other terms used during epoch training



```
Train Epoch 2000
{
    "Log_Likelihood": 22.048346557617187,
    "Loss": -22.048346557617187,
    "Optimizer/policy0_lr": 0.0001,
    "Policy_Grad_Norms": 77699.1415074706,
    "Time_Data_Loading": 0.0020467241605122886,
    "Time_Epoch": 0.01786708434422811,
    "Time_Log_Info": 1.567204793294271e-05,
    "Time_Process_Batch": 6.92129135131836e-05,
    "Time_Train_Batch": 0.015702736377716065
}
```

```
Train Epoch 2000
{
    "Log_Likelihood": 23.518069019317625,
    "Loss": -23.518069019317625,
    "Optimizer/policy0_lr": 0.0001,
    "Policy_Grad_Norms": 4474288.859888921,
    "Time_Data_Loading": 0.0028784950574239094,
    "Time_Epoch": 0.09497318267822266,
    "Time_Log_Info": 1.7996629079182943e-05,
    "Time_Process_Batch": 5.133549372355143e-05,
    "Time_Train_Batch": 0.09190346399943033
}
```

(a) Screenshot from training epoch 2000 of BC and BC-RNN



```
Train Epoch 2000
{
    "Actor/Cosine_Loss": 0.08959854662418365,
    "Actor/L1_Loss": 0.0009839114494388922,
    "Actor/L2_Loss": 0.0020292216923553496,
    "Actor/Loss": 0.0020292216923553496,
    "Actor/Optimizer/policy0_lr": 0.0010000000000000002,
    "Actor/Policy_Grad_Norms": 0.004102005711486072,
    "Loss": 0.00204256929053372,
    "Planner/Encoder_Variance": 0.4219076171517372,
    "Planner/Grad_Norms": 7.072696270950301e-07,
    "Planner/KL_Loss": 0.001912121950299479,
    "Planner/Loss": 1.334759817837039e-05,
    "Planner/Optimizer/goal_network0_lr": 0.0010000000000000002,
    "Planner/Reconstruction_Loss": 1.239153712958796e-05,
    "Planner/goal_loss": 1.334759817837039e-05,
    "Planner/kl_loss": 0.001912121950299479,
    "Planner/recons_loss": 1.239153712958796e-05,
    "Time_Data_Loading": 0.002219227949778239,
    "Time_Epoch": 0.16336631377538044,
    "Time_Log_Info": 8.990367253621419e-05,
    "Time_Process_Batch": 0.0010185837745666504,
    "Time_Train_Batch": 0.15990891456604003
}
```

(b) Screenshot from training epoch 2000 of HBC

Figure 2.2: *Varius statistics and values associated with training the algorithms. As explained below the figure*

Log_Likelihood: The product of probabilities of each point is the likelihood of that data set. It is represented in the logarithmic state as a result, is usually in an exponential state, and log simplifies it. A higher log-likelihood would mean better model compatibility with the data.

Loss: According to Wang et al. (2020), the loss function is simply a mathematical expression signifying the difference between predicted and actual values. In machine learning, the lower the loss function is, the higher the prediction accuracy. It can also be written as the negative of log-likelihood.

Optimizers: They are the adjustments done to the first parameters, especially the initial ones, to reduce the loss function.

Gradient Norms: It is the magnitude of the gradient vector of a function such as the loss function. Adjusting or improving this increases the general performance of the deep learning model. Usually higher gradient signifies that the training process is not stable enough.

Actor/Cosine_Loss: The cosine loss is a loss function in the actor-network, acting as an optimizer to increase the actor's performance in the training process.

2.2 Important terms related to project

2.2.1 Machine learning

The background section explains machine learning, as the key research topics are part of machine learning. According to of Leeds, a field of computer science and artificial intelligence(AI) known as machine learning focuses on using data and algorithms to mimic how humans learn, with the goal of continuously increasing the accuracy of simulation. Machine learning mainly has three types: supervised learning(Task-driven), unsupervised learning(Data-driven), and reinforcement learning(Learning from errors), as stated by Analytics (2019).

2.2.2 Robotic manipulation

Robotic manipulation is the way that robots interact with the things around them. This includes tasks like picking up an object, opening a door, folding a piece of clothing, and placing an order in a box. To do all these things, robotic hands and arms must be planned and controlled smartly according to university of Leeds. The ability to manipulate objects makes robots different from most automated or electronic systems. Robots' power to interact physically with and influence their surroundings opens up various potential uses. The field of manipulation, however, is still in its development. In research labs, impressive manipulator speed, dexterity, and autonomy have been shown, but before commercialisation is practical, these skills will need years of further development Man (2010).

2.2.3 Reinforcemennt learning

Reinforcement learning, often known as RL, is one of the fundamental subfields of machine learning. In reinforcement learning, an agent interacts with the environment by following a particular set of rules or policies. Furthermore, it works on a reward-based strategy where when the agent acts in each state of the environment based on a policy, it receives an award and obtains a new form. Learning an optimum policy that maximises the long-term accumulated rewards is the main objective of Reinforcement Learning (A brief overview of Imitation Learning, 2019). To do this, there are multiple RL algorithms and methods, most of which employ incentives as the primary tactic to approximate the optimal policy. The benefits of adopting these tactics are generally pleasant. But, in other instances, the process of teaching might be challenging. This may be especially true in environments where incentives are less accessible (e.g. a game where we only receive a reward when the game is won or lost). To remedy this issue, we may provide the agent with more frequent incentives by manually constructing the rewards functions. This will help. In addition, this manual method is necessary since a predefined reward function is not always accessible, such as when teaching an autonomous vehicle how to drive itself. Manually constructing a reward function that matches the desired behaviour may be complex and time-consuming (A brief overview of Imitation Learning, 2019). Hence, imitation learning is the solution to this specific problem.

2.2.4 Imitation Learning

Imitation learning is a subfield of machine learning that involves training models to perform a task by mimicking the behaviour of an expert. In robotics, imitation learning algorithms allow robots to learn from human demonstrations to perform complex tasks. In Inverse reinforcement learning, the goal is to learn the reward function that the expert is optimizing. Once the reward function is learned, the robot can perform the task more efficiently. Abbeel and Ng (2004) proposed an algorithm for apprenticeship learning via inverse reinforcement learning.

In Behavioral cloning, the robot learns a policy that maps observations to actions by directly mimicking the expert's behaviour. Ross et al. (2010) showed that behavioural cloning could be reduced to online learning without regret, meaning future observations do not have to depend on previous observations. Generative adversarial imitation learning is another type of IL. In this approach, the robot learns a policy by playing a game with a discriminator that distinguishes between the expert's demonstrations and the robot's actions. ? proposed an algorithm for generative adversarial imitation learning. However, in One-shot imitation learning, the robot learns to perform a task from a single demonstration. Yu et al. (2018) proposed an algorithm for one-shot imitation learning via domain-adaptive meta-learning.

Imitation learning has shown great promise in various robotic manipulation tasks, including grasping, pushing, and object manipulation. Sasaki et al. (2020) survey reinforcement learning in robotics, including imitation learning. In today's times, Imitation learning algorithms have become an active area of research in robotics, with significant progress being made in recent years. These algorithms enable robots to learn from human demonstrations, which can significantly improve the performance and capabilities of robotic systems.

Figure 2.3 show the functioning of imitation learning where according to Fang et al. (2019) representation, demonstration, and the algorithm are three main parts.

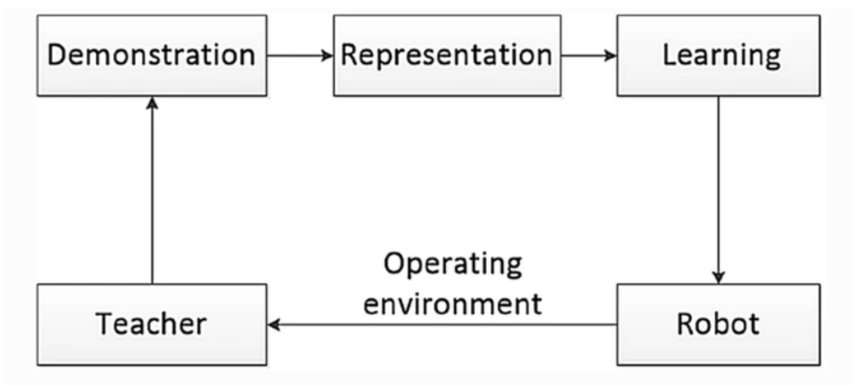


Figure 2.3: *This figure shows how imitation learning works Fang et al. (2019)*

2.3 Imitation learning and Benchmarking in-context to robotic manipulation

Imitation learning is a type of machine learning in which a task is learned by an agent by monitoring the activities of another person or agent (Ho and Ermon 2016a). When discussing robotic manipulation, imitation learning refers to teaching a robot so it can imitate the actions of a human demonstrator or another robot.

The technique to measure the performance of an algorithm in relation to a set of predefined benchmarks or metrics is known as benchmarking (Schulman et al. 2017). When discussing robotic manipulation, benchmarking is the process in which established benchmarks or metrics evaluate various robotic manipulation algorithms.

2.4 Previous research and its limitations

Benchmarking is an essential part of imitation learning in robotic manipulation. It helps in comparing different algorithms against a set of actions giving us the best algorithm for further use. It can be applied to robots for multiple different fields as well. But it also has its limitations, some of which are the absence of actual human expert demonstration from which the robot may learn, or the sheer number of diverse actions there are and setting benchmarks for each of them, all of which are highly challenging tasks, ultimately acting as a limiter according to Codevilla et al. (2019); Sun et al. (2021).

2.4.1 Addressing these limitations

This study aims to address these limitations in the context of robotic manipulation. Three imitation learning algorithms were used for this purpose: Behavioural Cloning (BC), Behavioural Cloning with a Recurrent Neural Network (BC-RNN), and Hierarchial Behavioural Cloning (HBC) on robosuite by Zhu et al. (2020) and robomimic by Mandlekar et al. (2021b). PyTorch Library and Tensorboard were utilised for visualisation. Ultimately this study aims to evaluate these algorithms under standardised benchmarks further to enable research in imitation learning and robotic manipulation.

2.5 Common metrics used in imitation learning

According to Alissandrakis et al. (2006), the metrics used to evaluate the action and states play an essential role in imitation learning and can be mathematically determined. Metrics that are commonly used for imitation learning algorithms are:

Success rate: Measures the percentage of tasks the agent completes out of 100. It tells how well the robot performs on the dataset.

Accuracy: Measures the similarity between the expert's demonstration and the agent's performance. It helps to evaluate how close the agent's performance is to the expert's demonstration.

Error rate: This metric measures the difference between the expert's demonstration and the agent's performance. It explains the agent's deviation from the expert's demonstration.

For this research, the success rate was used as the primary metric for comparing three algorithms: BC, BC-RNN and HBC success rate is the best effective way to tell how well an algorithm has performed on the dataset given.

2.6 Description of Imitation learning algorithms

Three Imitation learning algorithms were used in the research. Although other imitation learning algorithms are available, these algorithms were used because for this research, Mandlekar et al. (2021b) has been used as the base setup for experimentation and Mandlekar et al. (2021a) provide an extensive comparison of these algorithms. These are Behavioral Cloning (BC), Recurrent Behavioral Cloning (BC-RNN), and Hierarchical Behavioral Cloning (HBC), and promising results were seen from the use of these algorithms.

2.6.1 Behavioral Cloning(BC)

Behavioural cloning(BC) plays a vital role in robotic manipulation. According to Choi et al. (2020), the main idea behind behavioural cloning is to understand and learn a policy for a Markov Decision Process (MDP) through expert demonstrations and then perform an action from these demonstrations. BC also comes under imitation learning, where the main goal is reproducing actions from a learned policy.

Recently 'zero-trial' imitation learning was introduced as a concept within behavioural cloning, according to Wu et al. (2020) where a policy is learned only from human samples without any robot trials and can be an advantageous technique when there is limited interaction with the environment.

Behavioural cloning has some disadvantages as well. According to Sasaki et al. (2020) apprenticeship learning (AL), another imitation learning strategy requires a large number of training time and extensive environmental attractions.

2.6.2 Recurrent Behavioral Cloning (BC-RNN)

Impressive performance has been demonstrated using neural networks, such as RNNs on various computing science fields, such as language processing and speech recognition, according to Brusaferri et al. (2020). It hence can be used with behavioural cloning (BC) as well.

According to Mandlekar et al. (2021a), BC-RNN (Behavioral cloning with Recurrent Neural Network (RNN)) supports decision-making by allowing the modelling of temporal correlations.

BC-RNN can work better than BC because it can record the various actions performed over the training and how these actions relate to each other.

2.6.3 Hierarchical Behavioral Cloning (HBC)

Future predictions are made in HBC, by training a high-level subgoal planner. Action sequences are predicted by conditioning a low-level recurrent policy on future observations according to Mandlekar et al. (2021a).

HBC is also a better way to learn from offline human demonstrations because it seeks to learn from hierarchical policies that encourage temporal abstraction according to Tung et al. (2020).

HBC achieves its goals by learning a sequence of actions and predicting future observations, making it a better alternative to BC.

2.7 Significance of this research

According to gema.parreno.piqueras (2020), benchmarking imitation learning is to evaluate different imitation learning algorithms against specific tasks to determine which one is most suitable. For robotic manipulation, it can help us to determine the pros and cons of the algorithms we deploy for the training process.

The goal of imitation learning in robotic manipulation is to train the robots to perform different tasks. Setting benchmarks will clearly describe which algorithm would be best for it. In addition, it can also help differentiate which algorithm to use for which action depending upon the sector for which the robot is being trained, such as industrial automation, manufacturing, medical robots, etc.

For example, even in the agricultural sector, robots can be manipulated and trained for harvesting and grafting. Adding in imitation learning makes them perform complex tasks by learning and adapting to new situations and environments according to Fang et al. (2019).

Another example is in the medical field to as described in this article Collaborative Robot-Assisted Endovascular Catheterization with Generative Adversarial Imitation Learning. It mentions making the robot first learn endovascular catheterization by assisting a human operator and performing the same operation. It produced encouraging results and showcased the potential of imitation learning in medical robots.

Memmesheimer et al. (2019)suggest that for advancing imitation learning in robotics, benchmarking is necessary as it identifies strengths and weaknesses of algorithms and paves the way for more efficient methods to come forth with better results.

3 Design

This chapter includes the requirements, the approach taken towards the project and the implementation.

3.1 Requirements

This section of the Design chapter lists the essential requirements for this project and explains these requirements.

3.1.1 Robomimic

According to Mandlekar et al. (2021b)"*robomimic is a framework for robot learning from demonstration. It offers a broad set of demonstration datasets collected on robot manipulation domains and offline learning algorithms to learn from these datasets."* Robomimic includes multiple datasets such as proficient human, Multi-human, and machine-generated datasets. It also has datasets like Roboturk, which includes about 1000 demonstrations collected from several human operators by Mandlekar et al. (2018). Figure 3.1 shows robomimic having a modular design with its features.

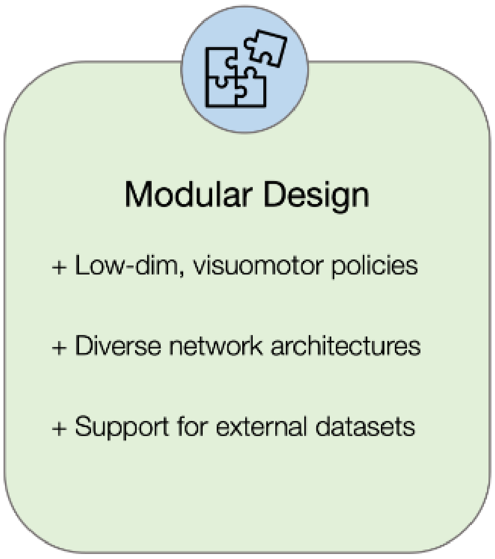
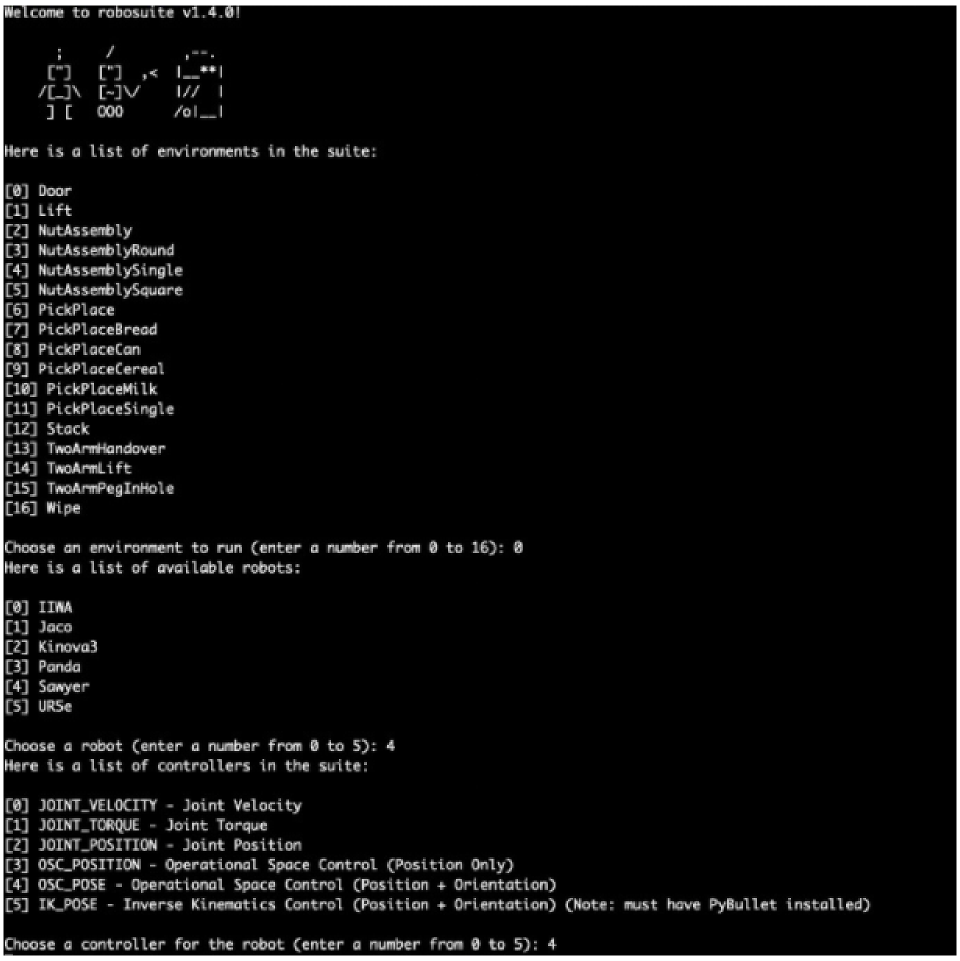

Modular Design
+ Low-dim, visuomotor policies
+ Diverse network architectures
+ Support for external datasets

Figure 3.1: *Robomimic had a modular design with a lot of features Mandlekar et al. (2021b)*

3.1.2 Robosuite

According to Zhu et al. (2020) -"*robosuite is a simulation framework powered by the MuJoCo physics engine for robot learning. It also offers a suite of benchmark environments for reproducible research*

". Robosuite also provides a collection of benchmarks that are uniformly used for rigorous assessment and algorithm development, high-quality implementation of off-the-shelf learning algorithms and robot controllers to minimise entrance barriers, and a flexible modular structure that allows for the creation of new robot simulation scenarios Zhu et al. (2020). Figure 3.2 shows the robosuite environment in the command prompt.



```
Welcome to robosuite v1.4.0!

      ;     /        ,--.
     ["]   ["]  ,<   |__**|
    /[_]\  [~]\/     |//  |
     ] [   OOO      /o|__|

Here is a list of environments in the suite:

[0] Door
[1] Lift
[2] NutAssembly
[3] NutAssemblyRound
[4] NutAssemblySingle
[5] NutAssemblySquare
[6] PickPlace
[7] PickPlaceBread
[8] PickPlaceCan
[9] PickPlaceCereal
[10] PickPlaceMilk
[11] PickPlaceSingle
[12] Stack
[13] TwoArmHandover
[14] TwoArmLift
[15] TwoArmPegInHole
[16] Wipe

Choose an environment to run (enter a number from 0 to 16): 0
Here is a list of available robots:

[0] IIWA
[1] Jaco
[2] Kinova3
[3] Panda
[4] Sawyer
[5] UR5e

Choose a robot (enter a number from 0 to 5): 4
Here is a list of controllers in the suite:

[0] JOINT_VELOCITY - Joint Velocity
[1] JOINT_TORQUE - Joint Torque
[2] JOINT_POSITION - Joint Position
[3] OSC_POSITION - Operational Space Control (Position Only)
[4] OSC_POSE - Operational Space Control (Position + Orientation)
[5] IK_POSE - Inverse Kinematics Control (Position + Orientation) (Note: must have PyBullet installed)

Choose a controller for the robot (enter a number from 0 to 5): 4
```

Figure 3.2: *Figure showing how robosuite provides different environments- a total of seventeen environments, six different robots, and a list of available controllers.* Image is taken from the personal laptop.

3.1.3 Tensorboard

Tensorboard is used in machine learning experiments to do visualisation tasks. Some tools included are visualizing metrics such as accuracy and loss, viewing success rates at various epochs, viewing the weights or biases as they change over time, and showing audio and image data. It can be seen in Figure 3.3 how the Tensorboard dashboard looks like, where different results can be found, such as the Time series, Scalars, images, and graphs from the various experiments that are performed on machine learning models.

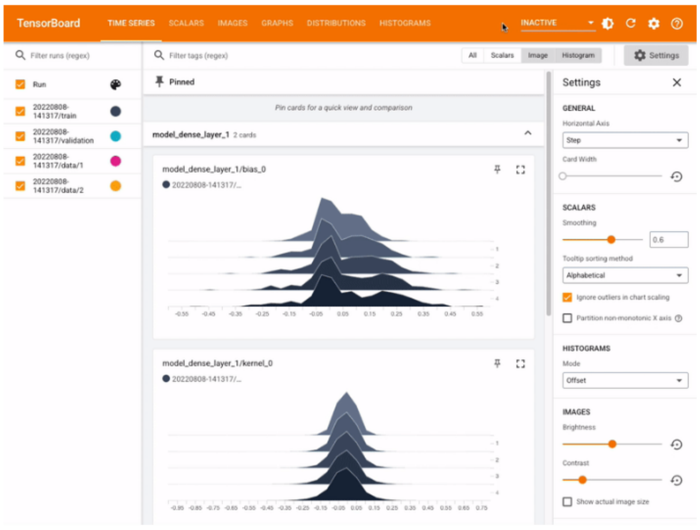


Figure 3.3: Tensorboard Dashboard PyT Fea

3.1.4 PyTorch

According to PyT"*PyTorch is a fully featured framework for building deep learning models, which is a type of machine learning that's commonly used in applications like image recognition and language processing*". Pytorch offers many features, as described in the image below. The different features of PyTorch according to Fea are:

- PyTorch is production ready, which means it provides all the necessary tools in one hand.

- PyTorch also provides a simple tool torchverse which helps with the deployment process. Example: Deploying the model to the cloud.

- PyTorch has various other features, such as it a robust ecosystem consisting of various essential tools for development.

3.2 Approach

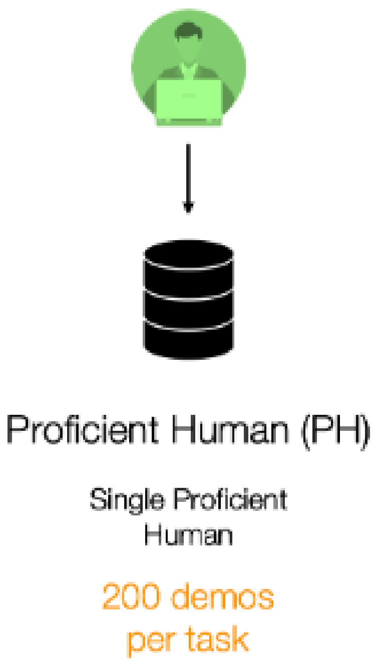
As part of this research, the main aim was to benchmark imitation learning algorithms for robotic manipulation tasks. Imitation learning mainly uses algorithms such as Behavioural cloning(BC), Inverse reinforcement learning, and Generative adversarial imitation learning(GAIL). The algorithms that are focused on in this study are Behavioral Cloning (BC), Behavioral Cloning with Recurrent Neural Networks (BC-RNN), and Hierarchical Behavioral Cloning (HBC). For the training of algorithms, the learning rate (the value which helps in the performance improvement of models) was set to 1e-4. Mandlekar et al. (2021a) suggests that with a lower learning rate the performance improves.

3.2.1 Software selection

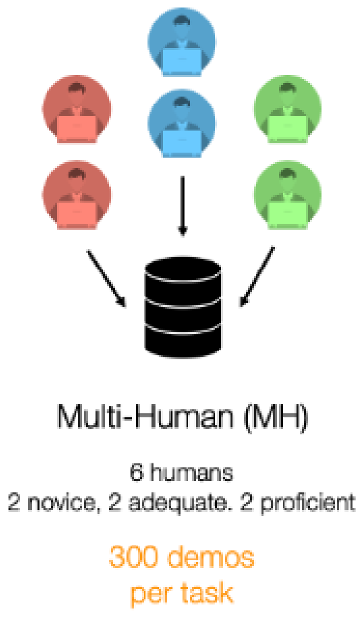
Robomimic library Mandlekar et al. (2021b) offers an easy interface for training and evaluating imitation learning algorithms for robotic manipulation, so the algorithms were implemented based on robomimic. Robosuite Zhu et al. (2020), which is based on mujoco engine, was used as the simulation environment.

3.2.2 Dataset and task selection

The chosen datasets are the proficient human dataset, which includes a demonstration from a single human being, and the Multi-Human dataset, which includes demonstrations from multiple human demonstrators of varying quality. The tasks chosen are the lift and can tranfer from one table to some other location task. So four datasets had to be downloaded, which are Lift(PH), Can(PH), Lift(MH), Can(MH) from the robomimic frameworkMandlekar et al. (2021b).



(a) Single human Dataset.



(b) Multi-Human Dataset.

Figure 3.4: *The datasets used in the study on which the imitation learning algorithms were trained.Mandlekar et al. (2021b)*

3.2.3 Summary

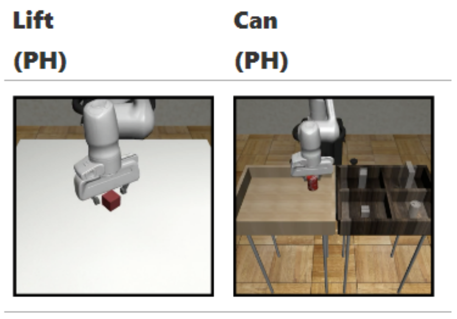
This study will benchmark the performance of the chosen imitation learning algorithms for robotic manipulation tasks. We will utilise PyTorch, the RoboMimic library, the RoboSuite simulation environment, and other required hardware and software to construct and test the chosen algorithms. Finally, the experiment's findings will be carefully examined and debated.

3.3 Implementation

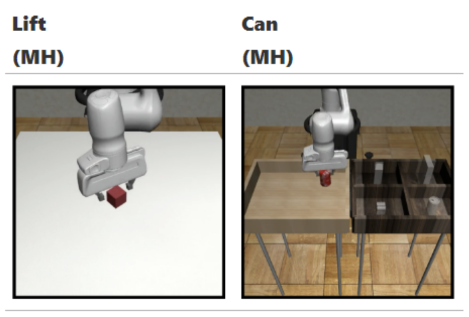
This research section explains how benchmarking was implemented on imitation learning algorithms for robotic manipulation. Robomimic and robosuite were used as the environments to perform the experiments. The algorithms were evaluated using the results obtained from TensorFlow. The source code had been taken from robomimic by Mandlekar et al. (2021b) and robosuite by Zhu et al. (2020) as part of the research.

3.3.1 Environment and algorithms

In this research, the manipulation tasks were performed using the environments provided by robosuite. The tasks Lift(PH), Lift(MH), Can(PH), and Can(MH) were the main priorities. As described in the robomimic documentation, three imitation learning algorithms were constructed and trained for this research: Behavior Cloning (BC), Behavior Cloning with Recurrent Neural Networks (BC-RNN), and Hierarchy of Behavior Cloning (HBC).



(a) Lift and Can task using proficient human dataset



(b) Lift and Can task using proficient human dataset

Figure 3.5: *Here we can see in the images sourced from Mandlekar et al. (2021b) how task performed look like in simulation using robosuite Zhu et al. (2020)*

3.3.2 Algorithm implementation

For this research, the algorithms were implemented using the guidelines provided in the robomimic documentation. After training the three algorithms: BC, BC-RNN, HBC, the graphs for the different traing results were obtained from Tensorboard for better understanding of the results and to evaluate their performance. Figure 3.7 shows a screenshot of how the results look like obtained from tensorboard.

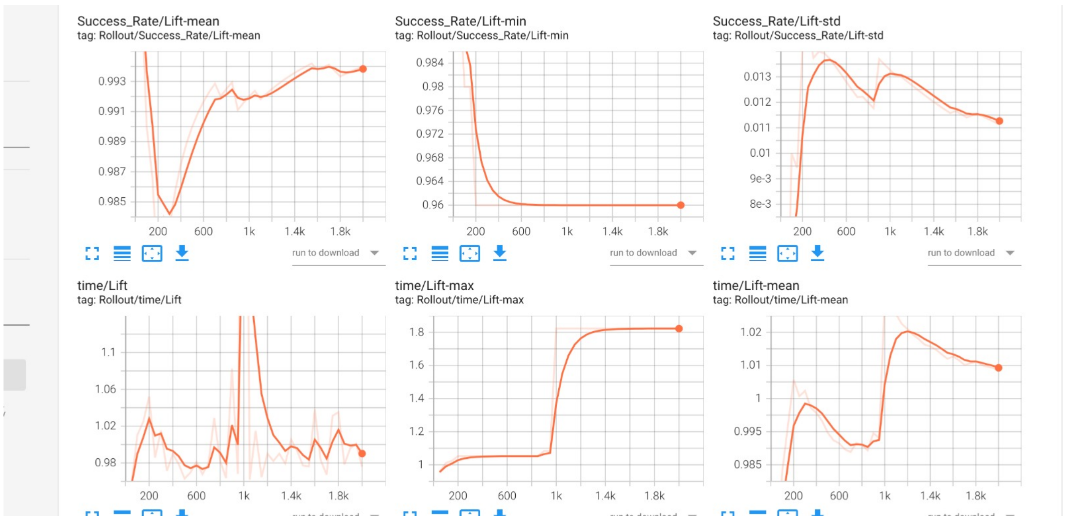


Figure 3.6: Results from training 1 of the algorithms

3.3.3 Observation space used

The low-dimensional observation space was used for this research which contains proprioception observations that consist of an end effector position which is 3-dimensional, a quaterion which is 4-dimensional, and gripper finger positions which are 2- dimensional.Mandlekar et al. (2021a)

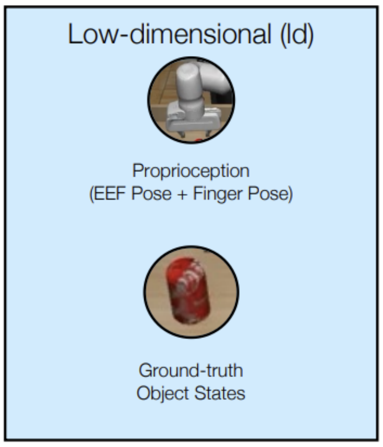


Figure 3.7: Low-dim observational space Mandlekar et al. (2021a)

3.3.4 Setup for training

Adam(Adaptive moment estimation) optimizer Kingma and Ba (2017) is used to train the neural networks. Each agent's maximum training success rate is recorded, and the outcomes are averaged over three seeds. For this research, the agents trained with low-dim observations in the following settings were specified: $N = 2000$, $M = 100$, and $E = 50$, where N being the total number of epochs, M being the number of gradient steps taken during each epoch. Every E epochs the effectiveness of agents is assessed by doing 50 rollouts in the environment and reporting the success rate accross a maximum horizon. These settings were chosen because robomimc results Mandlekar et al. (2021a) were selected as the base for comparison.

3.3.5 Algorithm setup

This section explains how BC, BC-RNN, and HBC function.

Behavioral Cloning (BC) is a method for learning a policy from demonstrations. It trains a policy $\pi_\theta(s)$ to imitate the actions in the dataset with the following objective:

$$\arg\min_{\theta} \mathbb{E}(s, a) \sim D\|\pi\theta(s) - a\|^2. \tag{3.1}$$

BC-RNN is a variant of BC that uses a Recurrent Neural Network (RNN) as the policy network, allowing it to model temporal dependencies in the data.

Hierarchical Behavioral Cloning (HBC) trains hierarchical policies, consisting of a low-level policy $\pi_\theta^L(s, sg)$ conditioned on subgoals sg and a high-level policy $\pi_\theta^H(s)$ predicting future subgoals. The low-level policy is similar to BC-RNN, with the addition of subgoal conditioning. The high-level policy is often a conditional Variational Autoencoder (cVAE) that learns a conditional distribution $\pi^H(s_{t+T}|s_t)$.

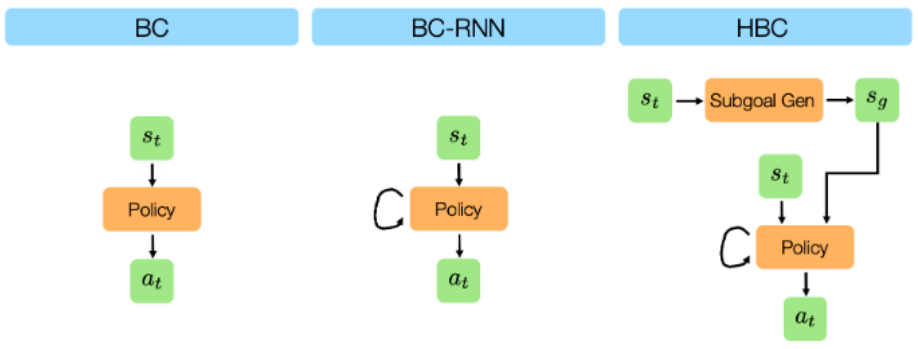


Figure 3.8: How the algorithms function Mandlekar et al. (2021a)

3.3.6 Dataset downloads and training run

These python scripts were obtained from Mandlekar et al. (2021b) as part of the study. The datasets were downloaded using download_datasets.py, a python script that provides a way to install all the required datasets. For example, to download the low_dim multi-human dataset for the lift and can tasks, the following command was used:

```
python download_datasets.py –tasks can lift –dataset_types mh –hdf5_types low_dim
```

Then generate_paper_configs.py script was used to generate training config json files to reproduce experimental results. These were generated using the command:

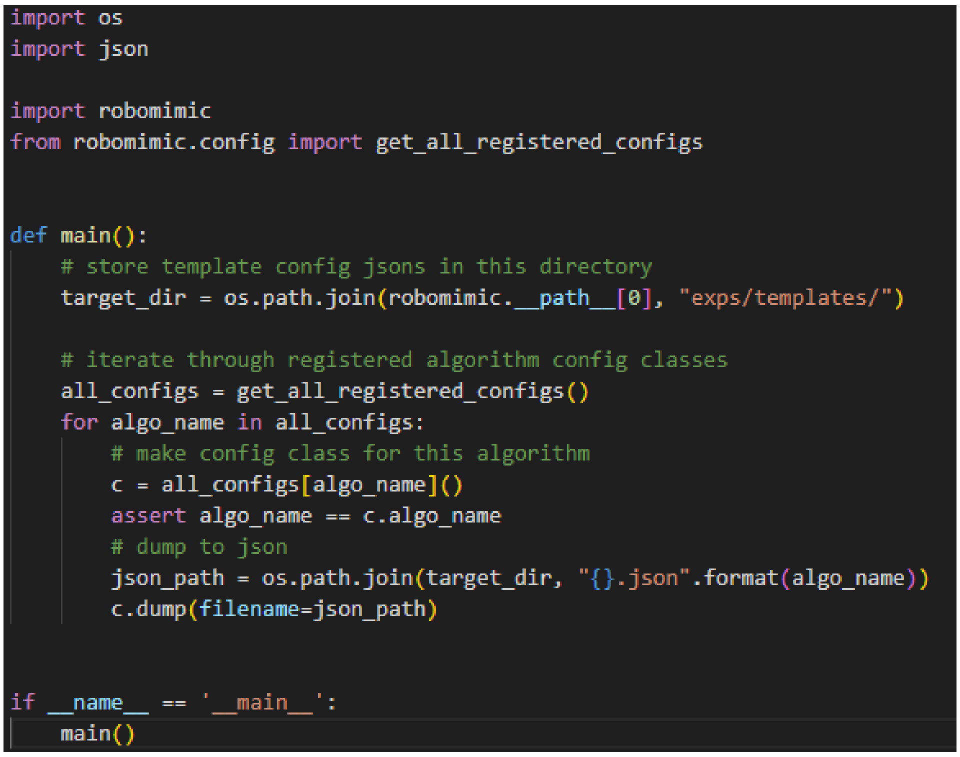
```
python generate_paper_configs.py –output_dir /tmp/experiment_results
```

Then the train.py script was run to initiate the training process. Figure 3.9 shows the code snippet for config generation. To run BC for example on Low dim Lift(MH) dataset, the following command was run:

```
python train.py –config ../exps/paper/core/lift/mh/low_dim/bc.json
```

These results were then obtained in the form of graphs by running Tensorboard. this line of command below will show the outputs from training HBC on a multi-human dataset.

```
tensorboard –logdir /Users/rutvik/Desktop/trained_models/HBC_mh/20230309014608 –
bind_all
```

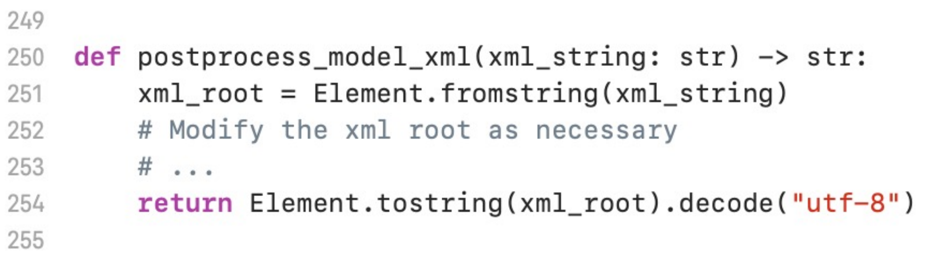


```python
import os
import json

import robomimic
from robomimic.config import get_all_registered_configs


def main():
    # store template config jsons in this directory
    target_dir = os.path.join(robomimic.__path__[0], "exps/templates/")

    # iterate through registered algorithm config classes
    all_configs = get_all_registered_configs()
    for algo_name in all_configs:
        # make config class for this algorithm
        c = all_configs[algo_name]()
        assert algo_name == c.algo_name
        # dump to json
        json_path = os.path.join(target_dir, "{}.json".format(algo_name))
        c.dump(filename=json_path)


if __name__ == '__main__':
    main()
```

Figure 3.9: Code snippet screenshot showing the generate config script Mandlekar et al. (2021b)

3.3.7 Compatibility issues between robomimic and robosuite

For this research, some issues and errors kept coming up during the training. Upon further research, it was clear that there were compatibility issues with robomimcs' latest version and robosuite, which was not mentioned in the documentation. To address this issue, a small change had to be made in the robosuites mjcf.utils. For this, an import was made in the file from xml.etree.ElementTree import Element the following code was added:

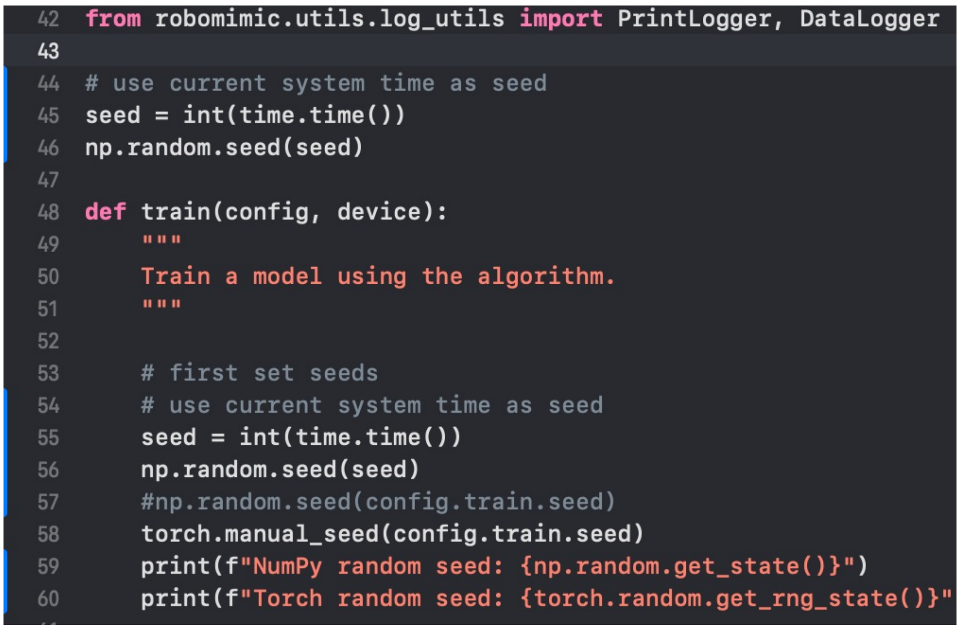


```python

def postprocess_model_xml(xml_string: str) -> str:
    xml_root = Element.fromstring(xml_string)
    # Modify the xml root as necessary
    # ...
    return Element.tostring(xml_root).decode("utf-8")

```

Figure 3.10: code snippet showing the added code in mjcf.utils

3.3.8 Issue with random seed generation

To ensure the research was done correctly, the seeds had to be randomly generated to compare results properly and see the effectiveness of the algorithms on the datasets. A small line of code had to be added to ensure the seeds were randomly generated.



```python
from robomimic.utils.log_utils import PrintLogger, DataLogger

# use current system time as seed
seed = int(time.time())
np.random.seed(seed)

def train(config, device):
    """
    Train a model using the algorithm.
    """

    # first set seeds
    # use current system time as seed
    seed = int(time.time())
    np.random.seed(seed)
    #np.random.seed(config.train.seed)
    torch.manual_seed(config.train.seed)
    print(f"NumPy random seed: {np.random.get_state()}")
    print(f"Torch random seed: {torch.random.get_rng_state()}"
```

Figure 3.11: Code snippet showing the added code in train.py and the current seed time being used as seed.

4 Evaluation

This section shows the results obtained and the base experimental results and compares them.

4.1 Training and metrics

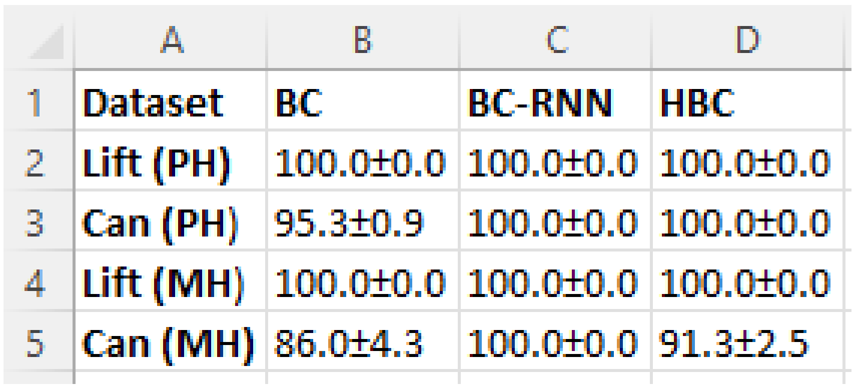
For the research, the training was done on a MacBook pro. The training time for each algorithm on a dataset for every task was around 4-5 hours. The mean and std deviation of success rates over three avg seeds were compared for every algorithm trained on every dataset, and the results were compared.

4.2 Comparison of results

The average and std deviation from success rates obtained as part of the training done for the study over three seeds using the low-dim datasets Lift(PH), Lift(MH), Can(PH), and Can(MH) from the research were compared to results obtained in the robomimic paper Mandlekar et al. (2021b) and was set as the base experimental result.

4.2.1 Original results

Figure 4.1 shows a table made from the original values of results obtained from Mandlekar et al. (2021a) as part of the study.

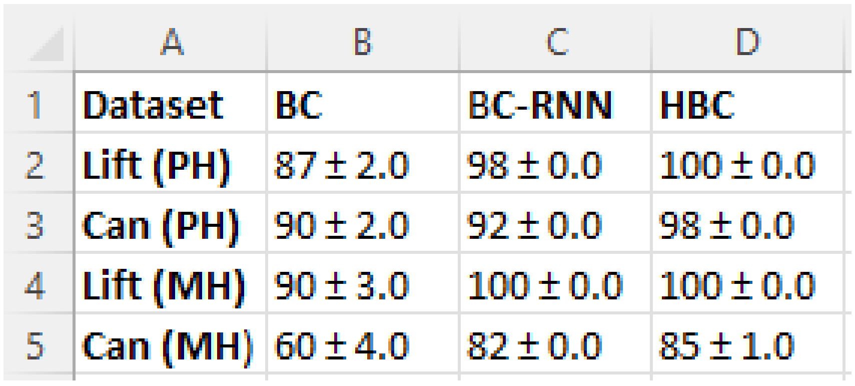


	A	B	C	D
1	Dataset	BC	BC-RNN	HBC
2	Lift (PH)	100.0±0.0	100.0±0.0	100.0±0.0
3	Can (PH)	95.3±0.9	100.0±0.0	100.0±0.0
4	Lift (MH)	100.0±0.0	100.0±0.0	100.0±0.0
5	Can (MH)	86.0±4.3	100.0±0.0	91.3±2.5

Figure 4.1: *These are the original results in percentage out of 100 obtained from the robomimic paper and set as the base for experimenting. Mandlekar et al. (2021a)*

4.2.2 Results obtained

The table in Figure 4.2 below shows the mean and st deviations of success rates of 3 seeds for each algorithm on the low-dim Proficient-Human (PH) and Multi-Human (MH) datasets to train the lifting task and the can transfer task obtained as part of the research.



	A	B	C	D
1	Dataset	BC	BC-RNN	HBC
2	Lift (PH)	87 ± 2.0	98 ± 0.0	100 ± 0.0
3	Can (PH)	90 ± 2.0	92 ± 0.0	98 ± 0.0
4	Lift (MH)	90 ± 3.0	100 ± 0.0	100 ± 0.0
5	Can (MH)	60 ± 4.0	82 ± 0.0	85 ± 1.0

Figure 4.2: These are the self obtained results in terms of percentage out of 100

4.2.3 Comparison

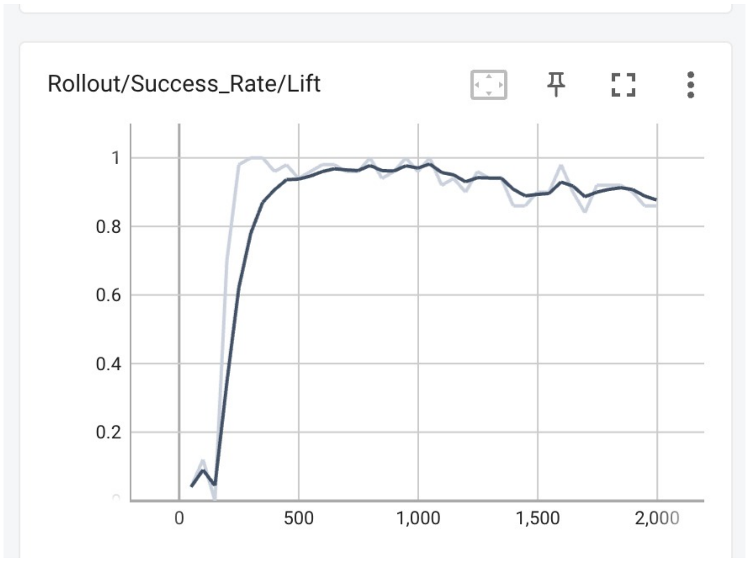
From the tables shown above (Figure 4.1, Figure 4.2), it can be seen that HBC performed better than the results obtained for Can(MH). Still, BC-RNN performed better for the actual results from robomimic, suggesting that both BC-RNN and HBC are better alternatives to BC and hence suggesting that adding different policies with behavioural cloning (BC) can improve the performance of the algorithm. These slight changes in the performance of BC and BC-RNN could be because of the random seeds generated for the study for a better understanding.

4.3 Graphs showing a comparison of algorithms based on success rate

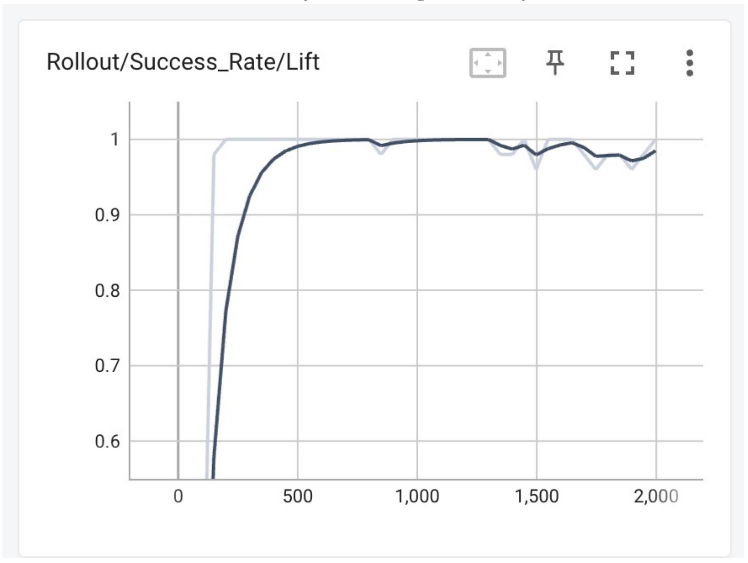
In this section, we compare the graphs of success rates obtained from Tensorboard for the Lift(PH), and Lift(MH).

4.3.1 Comparison of algorithms on Lift(MH) dataset

For the graphs below the horizontal axis is the success rate ranging from 0 to 1, and the vertical axis represents the epoch trained from 0 to 2000. Figure 4.3(a) represents the success rate for Behavioural cloning(BC). Figure 4.3(b) represents the success rate for BC-RNN and Figure 4.4 represents the graph for Heirarichal Behavioural cloning(HBC). The graph of HBC also shows how it learns from the previous observations as well and keeps on improving.



(a) Success rate from training BC on Lift(MH)



(b) Success rate from training BC-RNN on Lift(MH) dataset

Figure 4.3: *The success rate graphs obtained from Tensorboard.*

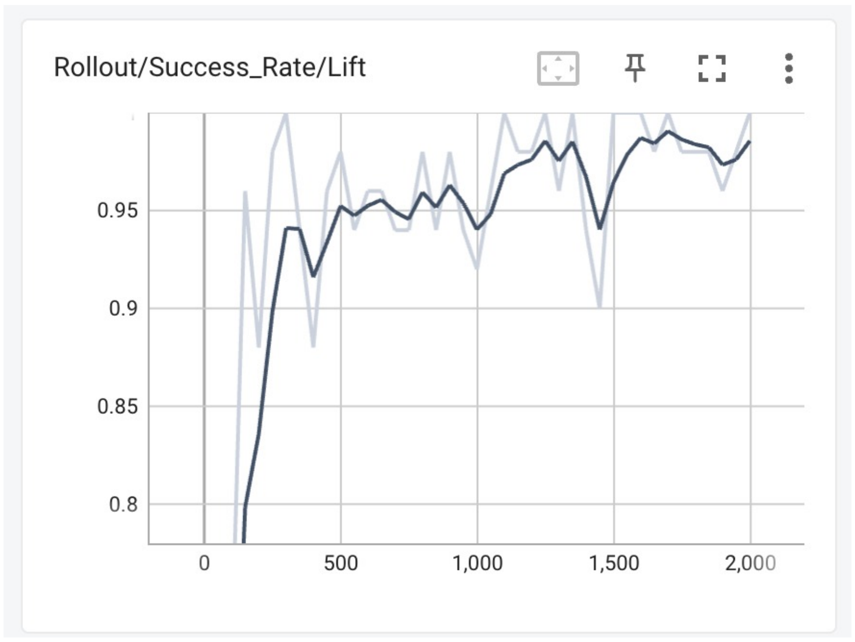


Figure 4.4: *Success rate from training HBC on Lift(MH). The graph obtained from Tensorboard*

Summary These graphs suggest that the success rates are better overall for the training epochs of BC-RNN and HBC than that of BC and also show that BC-RNN has a much stable success rate curve than that of BC and HBC showing that it performed well on Lift(MH) dataset to do the lift task. For HBC it also shows that

4.3.2 Comparison of algorithms on Lift(PH) dataset

For the graphs below, the horizontal axis is the success rate ranging from 0 to 1, and the vertical axis represents the epoch trained from 0 to 2000. Figure 4.5 represents the success rate graph for Behavioural cloning(BC). Figure 4.6 represents the success rate graph for BC-RNN, and Figure 4.7 represents the success rate graph for Heirarichal Behavioural cloning(HBC).

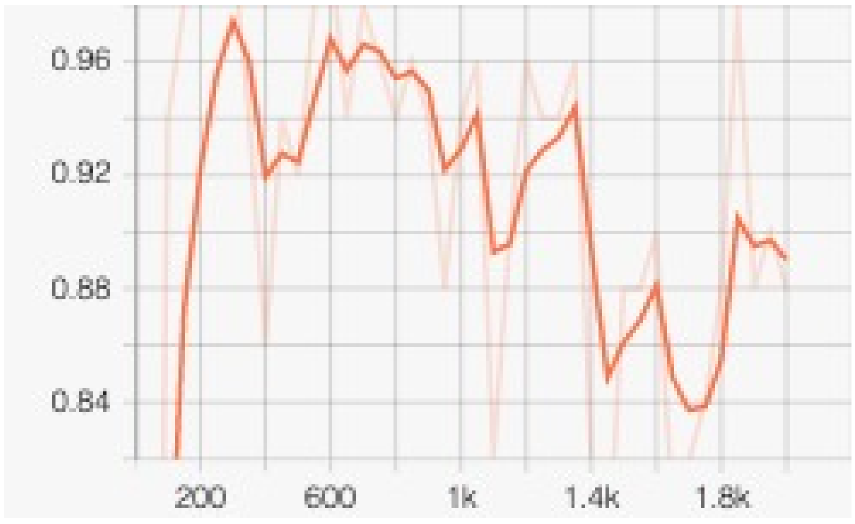


Figure 4.5: *Success rate from training BC on Lift(PH)*

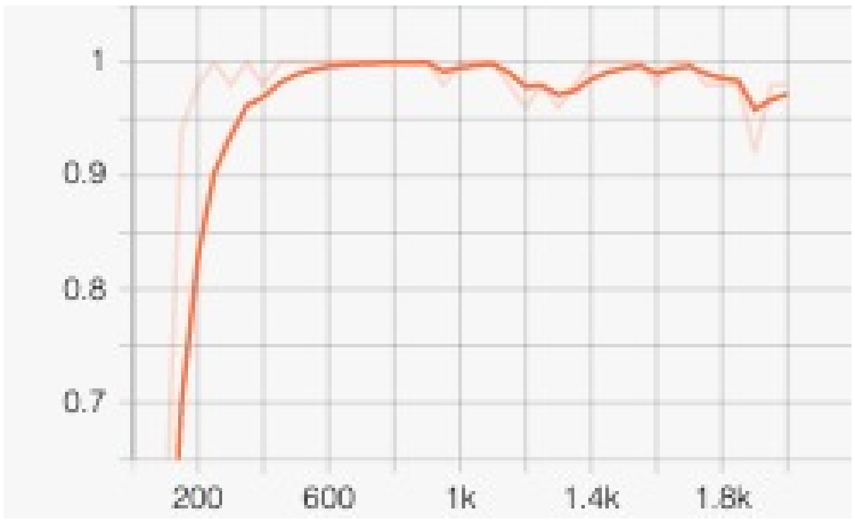


Figure 4.6: *Success rate from training BC on Lift(PH)*

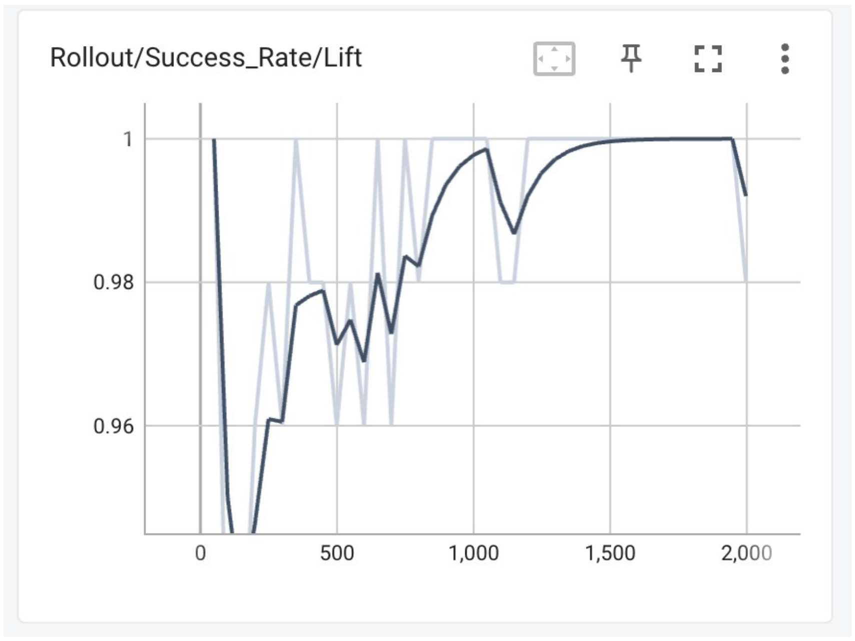


Figure 4.7: Success rate from training BC on Lift(PH)

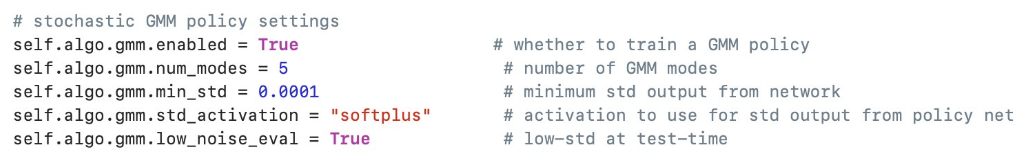
Summary These graphs suggest that the success rates are better overall for the training epochs of BC-RNN and HBC than that of BC, with BC performing well only for a few of the epochs and also show that BC-RNN has a much more stable success rate curve than that of BC and HBC showing that it performed well on Lift(Ph) dataset to do the lifting task. Still, HBC also shows excellent performance, and its graph improves with more epochs trained, which is what it defines.

4.4 Training BC with GMM enabled

For this section of study, the GMM policy was enabled (explained in section 2.1.2) for the traditional BC(Behavioural cloning) algorithm to see if it has any benefits and improvements.

4.4.1 Enabling the GMM policy

To enable GMM for this research, it had to enable from bc_config.py file provided by the Mandlekar et al. (2021b) library, as it can be seen from the code snippet in figure 4.8.



```
# stochastic GMM policy settings
self.algo.gmm.enabled = True                    # whether to train a GMM policy
self.algo.gmm.num_modes = 5                      # number of GMM modes
self.algo.gmm.min_std = 0.0001                   # minimum std output from network
self.algo.gmm.std_activation = "softplus"        # activation to use for std output from policy net
self.algo.gmm.low_noise_eval = True              # low-std at test-time
```

Figure 4.8: *hraph showing the success rate curve for BC run with enabled GMM to train lift task on Lift(PH) dataset*

4.4.2 Graph comparison

In this section, the graph in figure 4.5, which shows the success rate when BC is performed on Lift(PH), is compared with Figure 4.9, which shows the graph of the success rate when BC with enabled GMM is trained on Lift(PH) to perform the lifting task to see if Bc performs better with the GMM policy enabled. The vertical axis for Figure 4.9 is again the success rates and the horizontal axis is the epochs.

The comparison of both graphs shows that BC enabled with GMM performs better than BC without GMM, as can be seen from the learning curve in the graph. Better success rates are achieved throughout training the epochs and hence show better task performance which is the lift task.

This also suggests that adding a policy like GMM can improve the performance of imitation learning algorithms like BC and hence provide better success over task completion and robotic manipulation.

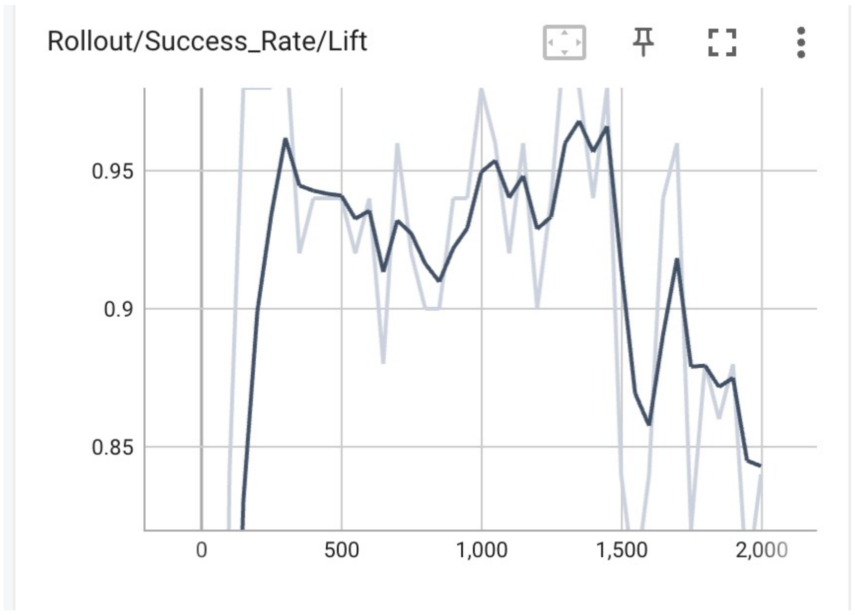


Figure 4.9: *Setting self.algo.gmm.enabled to True so a GMM policy can be trained.*

4.5 Conclusion

This chapter summarises the study done for this project and provides future work.

4.5.1 Summary

This dissertation explored and evaluated imitation learning algorithms for robotic manipulation tasks and benchmarked these algorithms. The concepts of behavioural cloning, BC-RNN, and HBC were reviewed, and their requirements, advantages, and limitations were analysed. Through experiments using the RoboMimic and RoboSuite simulation environments, these algorithms were trained and evaluated on four datasets, the Lift(PH and MH) and Can(PH and MH) datasets, to perform two tasks, lifting and can transfer.

4.5.2 Analysis

Analysis of these algorithms showed that HBC outperformed the other algorithms regarding success rates and task completion times, suggesting that the newer algorithms like BC-RNN and HBC are better than traditional approaches like BC. The study also suggested that using a policy like GMM can perform the performance of these traditional approaches. The performance of the algorithms varied depending on the complexity of the task and the type of the dataset used for training.

In conclusion, the study also demonstrated the potential of imitation learning algorithms for robotic manipulation tasks. It highlighted the importance of benchmarking and evaluating these algorithms using diverse datasets and tasks. Findings from this research can inform the development of more effective algorithms for real-world robotic manipulation tasks and contribute to advancing robotics and AI research.

4.5.3 Future work

The future work that can be done related to research is:

1. Proposing methods that build on the strengths of BC, BC-RNN, HBC. This might be adding techniques like meta-learning to the current algorithms mentioned.

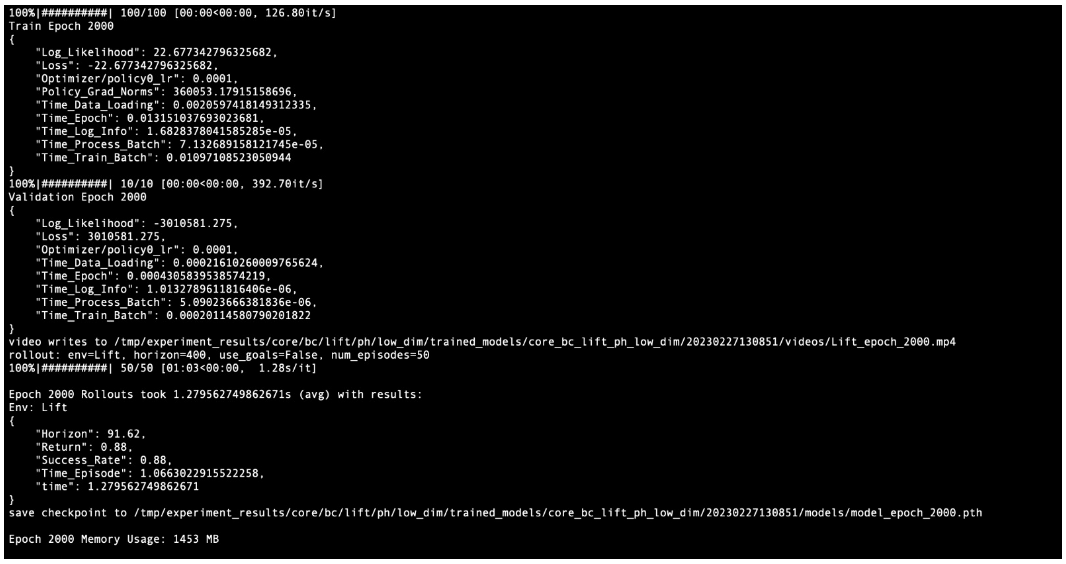
2. To allow algorithms to adapt to a new environment and generalize better the possibility of transferring the learned skills from one robot to the other should be considered.

3. New robot models, sensors, and algorithms can be added to the robomimic and robosuite framework to expand the capability to address a wide range of robot manipulation tasks.

4. To adapt to the requirements of individual users, new approaches should be considered to personalize imitation learning algorithms, which can also help robots to better learn from specific user-based preferences.

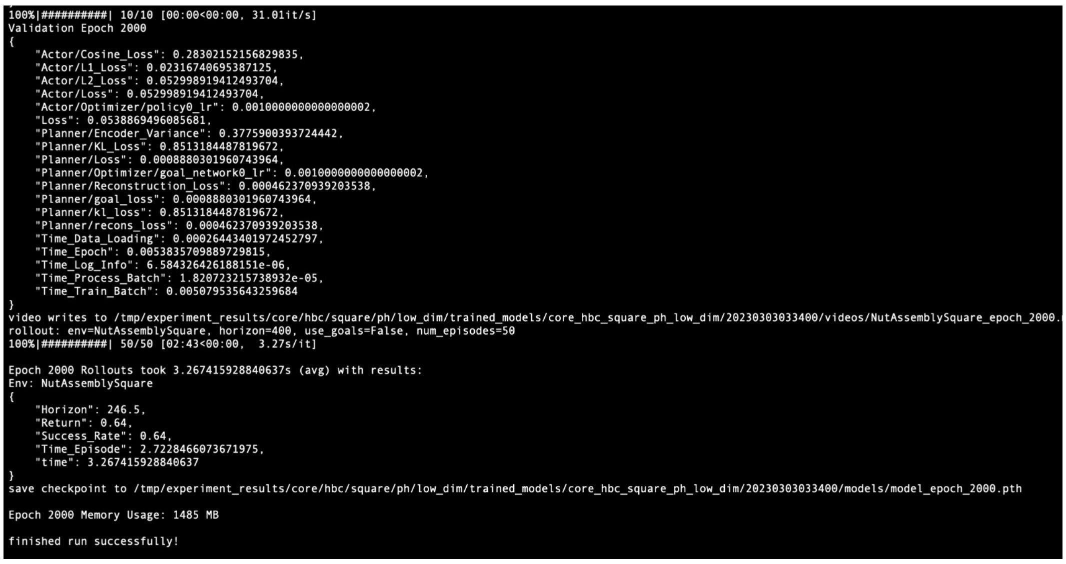
A Appendices

A.1 Log outputs from some results



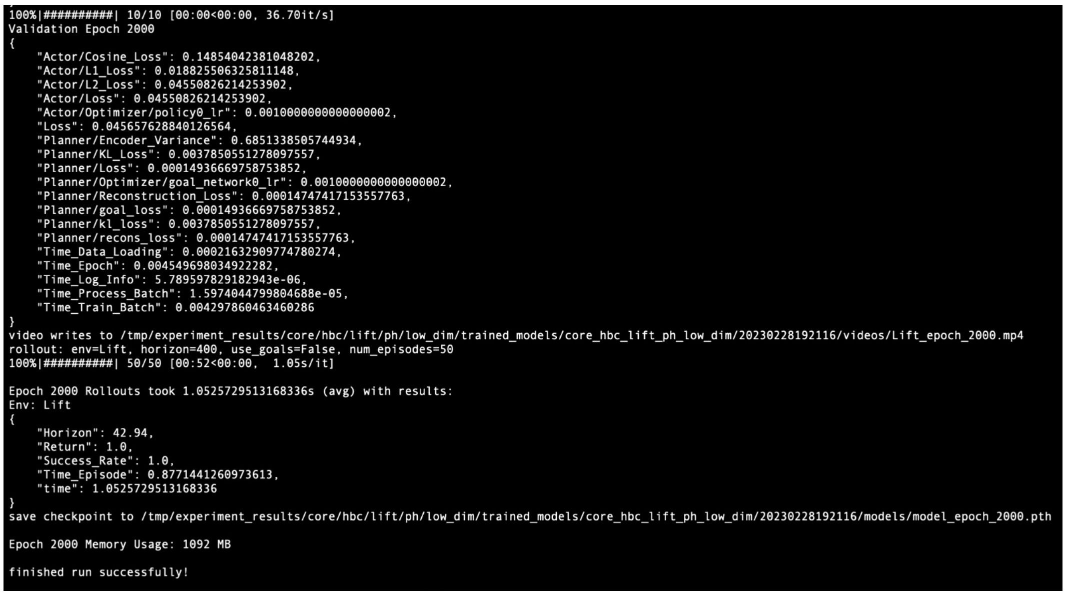
```
100%|##########| 100/100 [00:00<00:00, 126.80it/s]
Train Epoch 2000
{
    "Log_Likelihood": 22.677342796325682,
    "Loss": -22.677342796325682,
    "Optimizer/policy0_lr": 0.0001,
    "Policy_Grad_Norms": 360053.17915158696,
    "Time_Data_Loading": 0.0020597418149312335,
    "Time_Epoch": 0.013151037693023681,
    "Time_Log_Info": 1.6828378041585285e-05,
    "Time_Process_Batch": 7.132689158121745e-05,
    "Time_Train_Batch": 0.01097108523050944
}
100%|##########| 10/10 [00:00<00:00, 392.70it/s]
Validation Epoch 2000
{
    "Log_Likelihood": -3010581.275,
    "Loss": 3010581.275,
    "Optimizer/policy0_lr": 0.0001,
    "Time_Data_Loading": 0.00021610260009765624,
    "Time_Epoch": 0.0004305839538574219,
    "Time_Log_Info": 1.0132789611816406e-06,
    "Time_Process_Batch": 5.09023666381836e-06,
    "Time_Train_Batch": 0.00020114580790201822
}
video writes to /tmp/experiment_results/core/bc/lift/ph/low_dim/trained_models/core_bc_lift_ph_low_dim/20230227130851/videos/Lift_epoch_2000.mp4
rollout: env=Lift, horizon=400, use_goals=False, num_episodes=50
100%|##########| 50/50 [01:03<00:00,  1.28s/it]

Epoch 2000 Rollouts took 1.279562749862671s (avg) with results:
Env: Lift
{
    "Horizon": 91.62,
    "Return": 0.88,
    "Success_Rate": 0.88,
    "Time_Episode": 1.0663022915522258,
    "time": 1.279562749862671
}
save checkpoint to /tmp/experiment_results/core/bc/lift/ph/low_dim/trained_models/core_bc_lift_ph_low_dim/20230227130851/models/model_epoch_2000.pth

Epoch 2000 Memory Usage: 1453 MB
```

Figure A.1



```
100%|##########| 10/10 [00:00<00:00, 31.01it/s]
Validation Epoch 2000
{
    "Actor/Cosine_Loss": 0.28302152156829835,
    "Actor/L1_Loss": 0.02316740695387125,
    "Actor/L2_Loss": 0.052998919412493704,
    "Actor/Loss": 0.052998919412493704,
    "Actor/Optimizer/policy0_lr": 0.0010000000000000002,
    "Loss": 0.0538869496085681,
    "Planner/Encoder_Variance": 0.3775900393724442,
    "Planner/KL_Loss": 0.8513184487819672,
    "Planner/Loss": 0.0008880301960743964,
    "Planner/Optimizer/goal_network0_lr": 0.0010000000000000002,
    "Planner/Reconstruction_Loss": 0.000462370939203538,
    "Planner/goal_loss": 0.0008880301960743964,
    "Planner/kl_loss": 0.8513184487819672,
    "Planner/recons_loss": 0.000462370939203538,
    "Time_Data_Loading": 0.00026443401972452797,
    "Time_Epoch": 0.005383570988972981 5,
    "Time_Log_Info": 6.584326426188151e-06,
    "Time_Process_Batch": 1.820723215738932e-05,
    "Time_Train_Batch": 0.005079535643259684
}
video writes to /tmp/experiment_results/core/hbc/square/ph/low_dim/trained_models/core_hbc_square_ph_low_dim/20230303033400/videos/NutAssemblySquare_epoch_2000.
rollout: env=NutAssemblySquare, horizon=400, use_goals=False, num_episodes=50
100%|##########| 50/50 [02:43<00:00,  3.27s/it]

Epoch 2000 Rollouts took 3.267415928840637s (avg) with results:
Env: NutAssemblySquare
{
    "Horizon": 246.5,
    "Return": 0.64,
    "Success_Rate": 0.64,
    "Time_Episode": 2.7228466073671975,
    "time": 3.267415928840637
}
save checkpoint to /tmp/experiment_results/core/hbc/square/ph/low_dim/trained_models/core_hbc_square_ph_low_dim/20230303033400/models/model_epoch_2000.pth

Epoch 2000 Memory Usage: 1485 MB

finished run successfully!
```

Figure A.2



```
100%|##########| 10/10 [00:00<00:00, 36.70it/s]
Validation Epoch 2000
{
    "Actor/Cosine_Loss": 0.14854042381048202,
    "Actor/L1_Loss": 0.018825506325811148,
    "Actor/L2_Loss": 0.04550826214253902,
    "Actor/Loss": 0.04550826214253902,
    "Actor/Optimizer/policy0_lr": 0.0010000000000000002,
    "Loss": 0.045657628840126564,
    "Planner/Encoder_Variance": 0.6851338505744934,
    "Planner/KL_Loss": 0.0037850551278097557,
    "Planner/Loss": 0.00014936669758753852,
    "Planner/Optimizer/goal_network0_lr": 0.0010000000000000002,
    "Planner/Reconstruction_Loss": 0.00014747417153557763,
    "Planner/goal_loss": 0.00014936669758753852,
    "Planner/kl_loss": 0.0037850551278097557,
    "Planner/recons_loss": 0.00014747417153557763,
    "Time_Data_Loading": 0.00021632909774780274,
    "Time_Epoch": 0.004549698034922282,
    "Time_Log_Info": 5.789597829182943e-06,
    "Time_Process_Batch": 1.5974044799804688e-05,
    "Time_Train_Batch": 0.004297860463460286
}
video writes to /tmp/experiment_results/core/hbc/lift/ph/low_dim/trained_models/core_hbc_lift_ph_low_dim/20230228192116/videos/Lift_epoch_2000.mp4
rollout: env=Lift, horizon=400, use_goals=False, num_episodes=50
100%|##########| 50/50 [00:52<00:00,  1.05s/it]

Epoch 2000 Rollouts took 1.0525729513168336s (avg) with results:
Env: Lift
{
    "Horizon": 42.94,
    "Return": 1.0,
    "Success_Rate": 1.0,
    "Time_Episode": 0.8771441260973613,
    "time": 1.0525729513168336
}
save checkpoint to /tmp/experiment_results/core/hbc/lift/ph/low_dim/trained_models/core_hbc_lift_ph_low_dim/20230228192116/models/model_epoch_2000.pth

Epoch 2000 Memory Usage: 1092 MB

finished run successfully!
```

Figure A.3

4 Bibliography

URL `https://www.nvidia.com/en-us/glossary/data-science/pytorch/`.

URL `https://www.nvidia.com/en-us/glossary/data-science/pytorch/`.

Oct 2010. URL `https://www.roboticsbusinessreview.com/legal/manipulation-key-functionality-and-leading-opportunity/`.

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015430. URL `https://doi.org/10.1145/1015330.1015430`.

A. Alissandrakis, C. Nehaniv, and K. Dautenhahn. Action, state and effect metrics for robot imitation. pages 232 – 237, 10 2006. doi: 10.1109/ROMAN.2006.314423.

P. Analytics. What is machine learning: Definition, types, applications and examples, Dec 2019. URL `https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/`.

S. Banerjee. Real world applications of markov decision process (mdp), Jan 2021. URL `https://towardsdatascience.com/real-world-applications-of-markov-decision-process-mdp-a39685546026`.

A. Brusaferri, M. Matteucci, S. Spinelli, and A. Vitali. Learning behavioral models by recurrent neural networks with discrete latent representations with application to a flexible industrial conveyor. *Computers in Industry*, 122:103263, 2020. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2020.103263.

R. Chellappa, A. Veeraraghavan, N. Ramanathan, C.-Y. Yam, M. S. Nixon, A. Elgammal, J. E. Boyd, J. J. Little, N. Lynnerup, P. K. Larsen, et al. Gaussian mixture models. *Encyclopedia of Biometrics*, 2009(2):659–663, 2009.

J. Choi, H. Kim, Y. Son, C.-W. Park, and J. H. Park. Robotic behavioral cloning through task building. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1279–1281, 2020. doi: 10.1109/ICTC49870.2020.9289148.

F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

R. Dey and F. M. Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1597–1600, 2017. doi: 10.1109/MWSCAS.2017.8053243.

F. Emika. Franka emika: World's first sensitive robot, 2021. URL `https://www.franka.de/`.

B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, 3(4):362–369, Dec 2019. ISSN 2366-598X. doi: 10.1007/s41315-019-00103-5.

gema.parreno.piqueras. Mempathy: Benchmarking imitation and reinforcement learning for npc players in serious video..., Nov 2020. URL https://gema-parreno-piqueras.medium.com/mempathy-benchmarking-imitation-and-reinforcement-learning-for-npc-players-in-se

J. Goodman, A. Vlachos, and J. Naradowsky. Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing. In *Proceedings of the 54th annual meeting of the association for computational linguistics*, volume 1, pages 1–11. ACL, 2016.

A. Graves. *Supervised Sequence Labelling*, pages 5–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24797-2. doi: 10.1007/978-3-642-24797-2_2. URL https://doi.org/10.1007/978-3-642-24797-2_2.

J. Ho and S. Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016a. URL http://arxiv.org/abs/1606.03476.

J. Ho and S. Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016b. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

D. Johnson and S. Sinanovic. Symmetrizing the kullback-leibler distance. *IEEE Transactions on Information Theory*, 2001.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

I. C. London. Franka emika panda. https://www.imperial.ac.uk/a-z-research/robot-intelligence/robots/franka-emika-panda/. Research Themes.

A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *CoRR*, abs/2108.03298, 2021a. URL https://arxiv.org/abs/2108.03298.

A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021b.

R. Memmesheimer, I. Mykhalchyshyna, V. Seib, and D. Paulus. Simitate: A hybrid imitation learning benchmark. *CoRR*, abs/1905.06002, 2019. URL http://arxiv.org/abs/1905.06002.

T. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6): 47–60, 1996. doi: 10.1109/79.543975.

S. G. Odaibo. retina-vae: Variationally decoding the spectrum of macular disease. *arXiv preprint arXiv:1907.05195*, 2019.

U. of Leeds. Robotic manipulation: Robotics at leeds. URL https://robotics.leeds.ac.uk/research/ai-for-robotics/robotic-manipulation/.

F. Riaz, S. Rehman, M. Ajmal, R. Hafiz, A. Hassan, N. R. Aljohani, R. Nawaz, R. Young, and M. Coimbra. Gaussian mixture model based probabilistic modeling of images for medical image segmentation. *IEEE Access*, 8:16846–16856, 2020. doi: 10.1109/ACCESS.2020.2967676.

S. Ross, G. J. Gordon, and J. A. Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL `http://arxiv.org/abs/1011.0686`.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `https://proceedings.mlr.press/v15/ross11a.html`.

F. Sasaki, T. Yohira, and A. Kawaguchi. Adversarial behavioral cloning. *Advanced Robotics*, 34 (9):592–598, 2020. doi: 10.1080/01691864.2020.1729237. URL `https://doi.org/10.1080/01691864.2020.1729237`.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

J. Shin and J. H. Lee. *MDP formulation and solution algorithms for inventory management with multiple suppliers and supply and demand uncertainty*, volume 37 of *12 International Symposium on Process Systems Engineering and 25 European Symposium on Computer Aided Process Engineering*, page 1907–1912. Elsevier, Jan 2015. doi: 10.1016/B978-0-444-63576-1.50012-1. URL `https://www.sciencedirect.com/science/article/pii/B9780444635761500121`.

M. Sun, A. Mahajan, K. Hofmann, and S. Whiteson. Softdice for imitation learning: Rethinking off-policy distribution matching. *arXiv preprint arXiv:2106.03155*, 2021.

D. Tian. Gaussian mixture model and its applications in semantic image analysis. *J. Inf. Hiding Multim. Signal Process.*, 9(3):703–715, 2018.

A. Tung, J. Wong, A. Mandlekar, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese. Learning multi-arm manipulation through collaborative teleoperation. *CoRR*, abs/2012.06738, 2020. URL `https://arxiv.org/abs/2012.06738`.

Q. Wang, Y. Ma, K. Zhao, and Y. Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020.

A. Wu, A. Piergiovanni, and M. S. Ryoo. Model-based behavioral cloning with future image similarity learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1062–1077. PMLR, 30 Oct–01 Nov 2020. URL `https://proceedings.mlr.press/v100/wu20b.html`.

T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning, 2018.

T. Zhang, E. Zhang, H. Zhang, F. Shen, D. Guo, and K. Lin. An encoder-decoder neural network for indefinite length digit sequences in natural scene recognition. *Journal of Physics: Conference Series*, 1345(2):022025, nov 2019. doi: 10.1088/1742-6596/1345/2/022025. URL `https://dx.doi.org/10.1088/1742-6596/1345/2/022025`.

Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.