Robotic Object Manipulation with Full-Trajectory GAN-Based Imitation Learning

Haoxu Wang

Master of Computer Science

School of Computer Science
McGill University
Montreal, Quebec, Canada

September 2020

A thesis submitted to McGill University in partial
fulfillment of the requirements of the degree of
Master of Computer Science

Abstract

Our work aims to approach the difficult problem of robotic imitation learning from human demonstrations. We propose a Generative Adversarial Network (GAN)-based manipulation trajectory planner, which was integrated into a prototype learning system, and deployed on a state-of-the-art commercial robotic arm. The system fulfills simple tasks in two phases: grasping and manipulation. We adapted an open-source deep learning planner for grasping, and developed a GAN-based neural network that directly outputs the complete manipulation trajectory. The networks were trained with very few demonstrations, and tested in untrained scenarios.

Experiments were conducted in both a simulated and a physical environment. The system extracts trajectories from the recorded demonstrations via classical computer vision methods, and then performs grasping and manipulation to complete the learned task. Our GAN-based approach and three baseline solutions all performed comparably on a simple object interaction task. For a complex task, namely pouring, our approach displayed better performance in comprehending the complicated manipulation motions, including plausible tilting and pausing.

Abrégé

Ce travail vise à traiter le sujet complexe de l'apprentissage par imitation des gestes humains par les robots. Il propose un plan de trajectoire de manipulation fondé sur un Generative Adversarial Network (GAN), qui a été intégré à un prototype de système d'apprentissage et déployé à l'aide d'un bras robotique commercial de pointe. Le système permet de réaliser des tâches simples en deux phases : il peut attraper des objets et les manipuler. Un plan d'apprentissage profond open-source a été adapté pour lui permettre d'attraper des objets, et un réseau neural basé sur un GAN a été développé pour produire directement la trajectoire de manipulation complète. Les réseaux ont été formés à l'aide de peu de gestes, et testés dans le cadre de scénarios sans entraînement.

Des expériences ont été réalisées dans des environnements simulés et physiques. Le système extrait les trajectoires des gestes enregistrés via des méthodes classiques de représentation assistée par ordinateur, puis réalise le geste de saisie et de manipulation pour ainsi effectuer la tâche apprise. Pour permettre des comparaisons, cette approche en GAN et trois solutions de référence ont été appliquées sur une simple tâche impliquant une interaction avec un objet. Pour une tâche complexe, dans le cas présent, le fait de verser un liquide, notre approche a montré de meilleurs résultats dans la compréhension de mouvements de manipulation compliqués, comprenant une inclinaison et des temps de pause plausibles.

Contributions

Under Professor David Meger's supervision, Haoxu Wang designed and prototyped the learning system, as well as conducted experiments to examine the viability of the system. Haoxu Wang wrote this manuscript in consultation with Professor David Meger.

Acknowledgements

I would like to thank my supervisor, Professor David Meger of the School of Computer Science at McGill University. Professor David Meger is very knowledgeable, patient, helpful and passionate in both researching and teaching. It was only with his continuous guidance that I was able to complete my Master's degree.

I would also like to thank my family and all my dear friends for being wonderful presences in my life.

Back in 2009, I journeyed to the opposite side of the world with a lone bag of clothing. A decade later, I have been gifted a home, a family, a few friends and a son. I would like to thank every person who supported me along the way. Life goes on without a moment of pause. Ultimately, the cornerstone of my life is not the things that kept me busy everyday, but the thoughts and stories of the people I care about.

Contents

1 Introduction **1**

2 Robotic Object Manipulation **5**

2.1 Formulation 6

2.2 Motion Planning 7

2.3 Grasping 9

3 Learning to Grasp **13**

3.1 Convolutional Neural Networks for Perception 15

3.2 Dex-Net 2.0 18

3.2.1 Custom Denoising Module 20

4 Learning to Manipulate by Imitation **22**

4.1 Reinforcement Learning and Markov Decision Process 23

4.2 Imitation Learning Algorithms 25

4.2.1 Policy Imitation 27

4.2.2 Learning Rewards 30

4.3 Generative Adversarial Algorithms 33

4.3.1 Generative Adversarial Networks 33

4.3.2 Generative Adversarial Imitation Learning 35

5 Method: Imitating Full Trajectories **38**

5.1 Method Overview 38

5.2 Architecture of the Networks 39

5.2.1 Generator Architecture 40

5.2.2 Discriminator Architecture 42

5.3 Training 44

6 Experimental Setup **48**

6.1 Tasks of Interest 49

6.2 Neural Network Graphs 51

6.3 Baseline Methods 53

6.4 Demonstration Process 54

6.5 Computer Vision Pipeline 56

6.5.1 RGBD Camera 56

6.5.2 Color Marker Annotated Objects 62

6.5.3 Filtering and K-means Clustering 62

6.6 The JACO2 Assistive Robot Arm 64

6.7 Trajectory Visualization 67

7 Results **73**

7.1 Viability of the System 73

7.2 Qualitative Comparison 75

7.3 Learning Curves and Progression of Predictions 77

7.3.1 Full-Trajectory GAN 78

7.3.2 State-to-State GAIL 82

7.3.3 Full-Trajectory BC 85

7.3.4 State-to-State BC 88

7.3.5 Without Long Skipping Edges 90

7.4 Visualized Trajectory Predictions 91

7.5 Hyperparameters Exploration 95

7.6 Manipulation Trajectories Performed in Simulation 101

8 Conclusions and Future Work **103**

8.1 Conclusions 103

8.2 Future Work 104

Bibliography **105**

A Supplemental Figures **124**

List of Figures

1.1 Executing the Learned Pick-and-Place Trajectory (Video available) 2

1.2 Our Workspace Setup 3

2.1 The Kinova JACO Robotic Arm (reprinted from [des19]) 5

2.2 Frame of Reference and Gripper Pose 6

2.3 *Form Closure* (left) and *Force Closure* (right) Grasping a Number "8". . . . 10

2.4 An Example of a Caging Grasp (reprinted from [DSFK08]) 11

3.1 Various Objects in the Grasping Database Proposed by [KBS15] (reprinted figure) 14

3.2 Architecture of a Convolutional Neural Network (reprinted from [CMM+11]) 16

3.3 Upside Down Mounted Tripod and Camera 18

3.4 Sampled Grasps [MLN+17] 19

3.5 GQCNN model architecture [MLN+17] 20

3.6 Dex-Net 2.0 Workspace [MLN+17] 21

5.1 Architecture of the Generator with Skip Edges (the circles in $\mathcal{P}$ denote repeated subgraphs) 42

5.2 Architecture of the Discriminator 44

6.1 System Flowchart 50

6.2 Network Architecture of the Generator (22M params) and Discriminator (4M params) 52

6.3 Demonstrating Using a Grabber 56

6.4 Kinect V2 Front Panel [WS16] 57

6.5 IAI Kinect2 calibration (photo reprinted from [Wie15]) 58

6.6 Camera Color Channel 60

6.7 PCD Visualization 60
6.8 The Kinova JACO2®Assistive Robot 64
6.9 A Snapshot of Demonstration Recording 68
6.10 Visualizing the Arm's Pose in a 3D Plot 69
6.11 Original Demonstration Trajectory for Pouring 70
6.12 Trajectory Visualization from Multiple Perspectives 71
6.13 Original Demonstration Trajectory for Pouring 72
6.14 Re-sampled Demonstration Trajectory for Pouring 72

7.1 Demonstration Trajectory 74
7.2 Predicted Trajectory 74
7.3 Another Example Executing the Learned Pick-and-Place Trajectory (Video Available) 75
7.4 Negative Rewards from *Train* 77
7.5 Negative Rewards from *Eval* 77
7.6 *Train* Accuracy of State-to-Full-Trajectory GAN 78
7.7 *Eval* Accuracy of State-to-Full-Trajectory GAN 78
7.8 Predictions From State-to-Full-Trajectory GAN (EvalSeed #0) 80
7.9 Predictions From State-to-Full-Trajectory GAN (EvalSeed #7) 81
7.10 *Train* Accuracy of State-to-State GAIL 82
7.11 *Eval* Accuracy of State-to-State GAIL 82
7.12 Predictions From State-to-State GAIL (EvalSeed #0) 83
7.13 Predictions From State-to-State GAIL (EvalSeed #7) 84
7.14 MSE Loss Learning curve of State-to-Full-Trajectory BC 85
7.15 Predictions From State-to-Full-Trajectory BC (EvalSeed #0) 86
7.16 Predictions From State-to-Full-Trajectory BC (EvalSeed #7) 87
7.17 MSE Loss Learning curve of State-to-Full-Trajectory BC 88
7.18 Predictions From State-to-State BC (EvalSeed #0) 89
7.19 Predictions From State-to-State BC (EvalSeed #7) 90
7.20 Progression of Model Predictions, for State-to-Full-Trajectory GAN Model Without Long Skipping Edges 91
7.21 State-to-Full-Trajectory GAN Prediction Compared to Demonstration 92

7.22 State-to-Full-Trajectory BC Prediction Compared to Demonstration 93
7.23 State-to-State GAIL Prediction Comparing to Demonstration 94
7.24 State-to-State BC Prediction Compared to Demonstration 95
7.25 Consistency of Training Evaluated by *Train* Dataset 96
7.26 Consistency of Training Evaluated by *Eval* Dataset 96
7.27 Training Consistency 97
7.28 Effect of Synthetic Dataset Size 98
7.29 Discriminator Steps Per Iteration 99
7.30 Generator Steps Per Iteration 100
7.31 One Example of the Final Predictions 101
7.32 Executing the Learned Pouring Trajectory in Gazebo Simulation 102

A.1 Proposed Generator Network Diagram 125
A.2 Proposed Discriminator Network Diagram 126
A.3 Baseline State to Action Generator Network Diagram 127
A.4 Baseline State to Action Discriminator Network Diagram 128
A.5 Baseline State to Action Behavior Cloning Network Diagram 129
A.6 Baseline Full Trajectory Behavior Cloning Network Diagram 130
A.7 Pick-and-Place Demonstration Trajectory #0 131
A.8 Pick-and-Place Demonstration Trajectory #3 132
A.9 Pick-and-Place Demonstration Trajectory #6 133
A.10 Pick-and-Place Demonstration Trajectory #9 134
A.11 Pouring Demonstration Trajectory #0 135
A.12 Pouring Demonstration Trajectory #3 136
A.13 Pouring Demonstration Trajectory #6 137
A.14 Pouring Demonstration Trajectory #9 138

List of Tables

4.1 Notations and Concepts 23

7.1 Comparison of Averaged Negative Rewards 77

1 Introduction

With the advent of data-rich deep learning, many successful applications emerged in the realm of Reinforcement Learning (RL). As a data-driven approach, both the training dataset's distribution and the algorithm's data efficiency are crucial factors to deep RL. Meanwhile, in the world of robotics, expert demonstrations can be in limited supply or expensive to produce. This research thesis addresses this challenge. We introduce an IL-based solution for building a teachable robotic system, which learns simple tasks from extremely few human demonstrations and was examined on a state-of-the-art commercial robotic arm. In particular, our research emphasizes building a teachable robot system, leveraging the existing programmable industrial robotic systems, which learns from extremely few demonstrations.

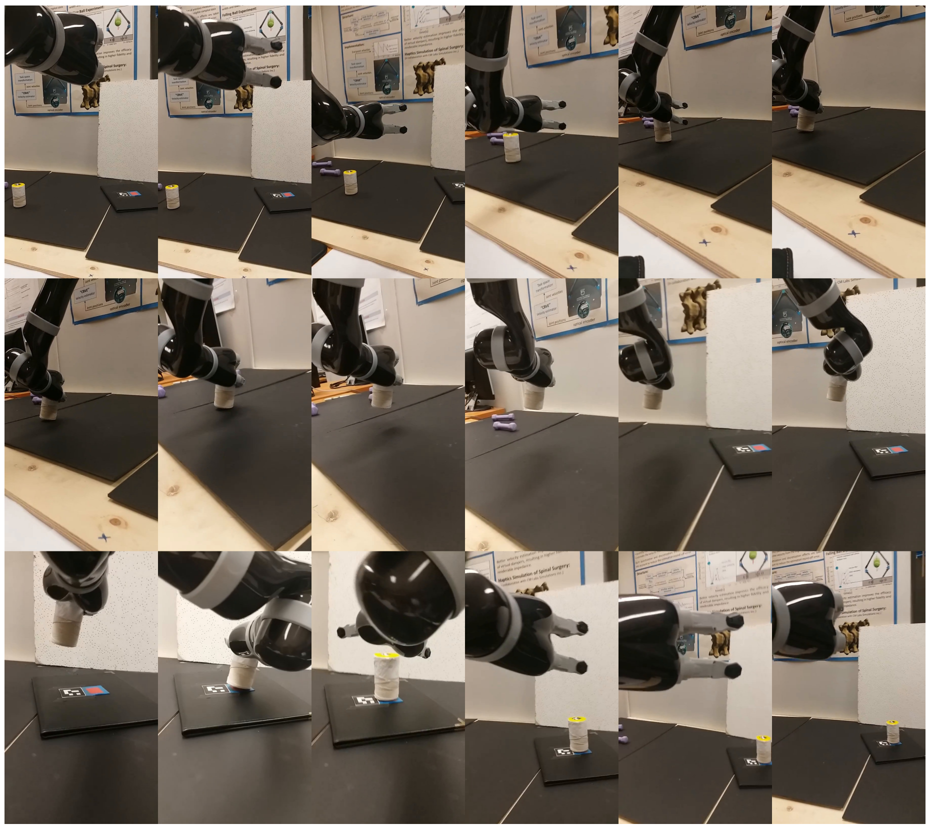


Figure 1.1: Executing the Learned Pick-and-Place Trajectory (Video available)

As the main technical contribution, we introduced a full-trajectory prediction deep neural network model, which implements a variation of the Generative Adversarial Network (GAN). Our network uses the initial positions of the object and the arm's end-effector as input, and outputs the full manipulation trajectory. Figure 1.1 shows the arm performing[1] the learned pick-and-place task. We believe directly predicting the complete trajectory contributes to a more coherent state-to-state transition. Compared to Generative Adversarial Imitation Learning (GAIL), our approach reduces the difficulty of training by not requiring estimated state-action value functions. During execution, we control the agent with existing joint-state-controller software solutions to follow the initially planned trajectory; however, this course correction feature sacrifices the potential capability for the agent to react to new information provided by real-time feedback. Overall, our method performed well on

[1]Video available at youtu.be/ywvo9tqk1q0 .

pouring, a complex real-world task.

We also implemented three baseline models. One of the most successful classical approaches for IL is behavior cloning (BC) implemented by supervised learning. While intuitively designed, this method imposes strict requirements for large and well-distributed training data. On the other hand, GAN relaxed those requirements by improving data efficiency, enabling the DNN to perform better on high-dimensional complex environments, which is a common characteristic of real world robotic problems. Compared to full trajectory prediction, state-to-state prediction is commonly used in the realm of reinforcement learning. To explore the aforementioned two aspects, our experiments compared our method with three different approaches, namely state-to-state BC, state-to-full-trajectory BC, and state-to-state GAIL.

To conduct real-world experiments, we constructed a prototype system using the Microsoft Kinect V2 camera as the eye of the system and the Kinova JACO2 Robotic Arm as our agent. The system was built upon the Robotic Operating System (ROS), bridging different infrastructural components of the system. This includes producing real-time point cloud data streamed from Kinect's RGBD video feed, recognizing affiliated objects in the scene, and parsing demonstration videos into coordinates of motion trajectories and points of interest.

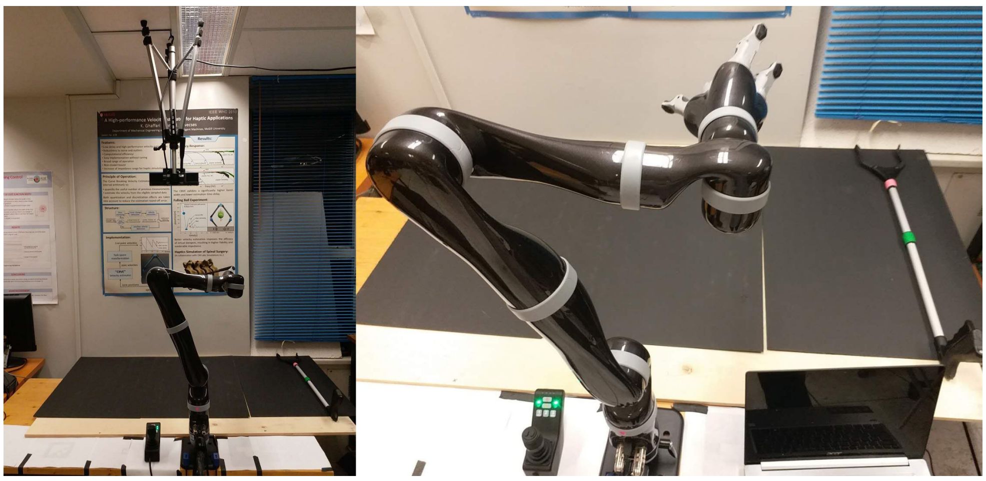


Figure 1.2: Our Workspace Setup

The manipulation trajectory planner of the system was built with a GAN-based DNN, which was able to imitate complex motions. The model exhibited improvements compared to the three baseline solutions. At execution time, the system first identifies relevant objects in the scene, then it utilizes a open-source grasp planner to acquire the object of interest in the gripper. After which, the deep neural network motion planner would suggest a manipulation trajectory by performing inference, using the position of the gripper and objects as input to the network. The trajectory will then be translated into 7 sets of joint angular positions and velocities using a joint-state controller, and finally executed by the JACO2 arm to complete the learned task.

Figure 1.2 shows the workspace used to conduct the experiments, highlighting a Kinect camera and tripod mounted against the ceiling, looking downwards, a JACO2 arm mounted on a table, a color-marker-annotated grabber and a laptop running the software.

Chapter 2 introduces the formulation of the robotic object manipulation problem and the classical solutions. Our work decomposes the fulfilment of a task into grasping and manipulation. Chapter 3 discusses the deep learning approach for the grasping problem, while Chapter 4 briefly discusses the theoretical foundations of deep imitation learning and its applications in object manipulation planning. Chapter 5 presents the main contributions of our research: a novel trajectory planning deep neural network network, as well as data engineering and training techniques.

Our experiments were conducted on both the simulated and the physical robot, with demonstrations recorded in the real world. Chapter 6 illustrates details regarding the experiments process, model parameters, baselines, the numerous software and hardware engineering challenges, as well as our solutions.

Finally, Chapter 7 attempts to scrutinize the results of our experiments, while Chapter 8 concludes this thesis.

2 Robotic Object Manipulation

A robotic arm functions by interacting with the world using its end-effector. This thesis studies the particular class of robotic arms which employ mechanical grippers as the end-effector. In this chapter, we will briefly introduce the classical approach to motion planning and grasping. In the next two chapters, we will discuss the motivation and advantages of the more modern, data-driven approach.

Figure 2.1: The Kinova JACO Robotic Arm (reprinted from [des19])

2.1 Formulation

Gripper-based robotic arms are designed to first grasp objects, and then perform the task of interest by executing manipulation trajectories, defined by

$$\tau = \{\vec{s}_{t_0}, \vec{s}_{t_1}, ..., \vec{s}_{t_H}\} \ , \tag{2.1}$$

where $\{t_0, t_1, ..., t_H\}$ is a sequence of timestamps, and $\vec{s}_t = [x, y, z, yaw, pitch, roll]$ defines the pose of the gripper at time t, as indicated in Figure 2.2. Meanwhile, the gripper is mobilized by the angles of joints, described by $\vec{\Theta} = \{\vec{\theta}_0, \vec{\theta}_1, ..., \vec{\theta}_{D-1}\}$[1]. The state of the arm can be described in *Cartesian space* as $\vec{s}$, or in *joint space* as $\vec{\Theta}$.

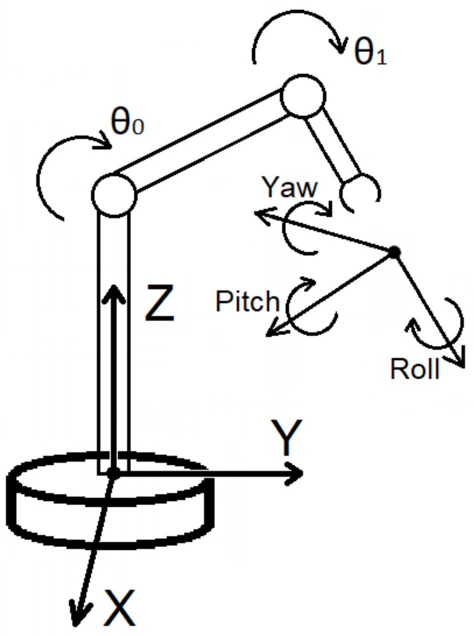


Figure 2.2: Frame of Reference and Gripper Pose

[1]The subscription D is the *degrees of freedom* of the robot.

Kinematics

Forward Kinematic (FK) studies the static geometrical problem of computing the position of the gripper from a given set of joint angles $\vec{\Theta}$ [Cra09]. Pose $\vec{s}$ of the gripper is calculated by a function denoted as FK

$$\vec{s} = FK(\vec{\Theta}) \ . \tag{2.2}$$

The concept of $workspace$ describe the physical space that the gripper can reach, which is the set of possible values of $\vec{s}$, defined as

$$\{\vec{s_j}, ...|\vec{s_j} = FK(\vec{\Theta})\} \ , \ s.t. \ \exists\vec{\Theta} \ . \tag{2.3}$$

Inverse Kinematics (IK), on the other hand, studies the problem of mapping back from a desired position of the gripper $\vec{s}$ to the set of all possible $\vec{\Theta}$ solutions of $FK(\vec{\Theta})$ by a solver function denoted as IK. The set is given by

$$IK(\vec{s}) = \{\vec{\Theta}_j, ...|\vec{s} = FK(\vec{\Theta}_j)\}. \tag{2.4}$$

Some modern industrial robots include high-level controller software solutions, which are facilitated by feedback sensors and actuators that are customized for their intended task domain [Cra09]. For example, the Kinova JACO Robotic Arm provides a driver capable of executing[2] a set of joint angles from time t_0 to t_H, given by

$$\{\vec{\Theta}_{t_0}, \vec{\Theta}_{t_1}, ..., \vec{\Theta}_{t_H}\}. \tag{2.5}$$

2.2 Motion Planning

Both FK and IK study the positioning and motion of robots that are constrained in a workspace. Meanwhile, they do not incorporate acceleration and force into consideration, which are crucial to practical robotic applications. Considering the following scenario, at each t, $\vec{\Theta}$ can be solved independently without considering the current joint angle states at

[2]The desired $\vec{\Theta}_{t+1}$ may not be possible to execute due to hardware limitation or the current $\vec{\Theta}_t$.

the previous timestamp. Trajectory τ is solved by

$$\begin{aligned}&\{\vec{\Theta}_{t_0}, \vec{\Theta}_{t_1}, ..., \vec{\Theta}_{t_H}\}\\ &s.t.\ \forall t\ ,\ \vec{\Theta}_t \in IK(\vec{s_t}),\ \vec{s_t} \in \tau\ .\end{aligned} \tag{2.6}$$

Because consecutive joint angles $\vec{\Theta}_{t_k}$ and $\vec{\Theta}_{t_{k+1}}$ could have large difference in value, the resulting motion would likely be incoherent and inefficient. A sophisticated joint state planner is required to plan smooth motion for τ while avoiding unintended contact with environmental obstacles. MOVEit is a popular motion planning framework in the open-source community, which provided motion planning, trajectory generation and environment monitoring [CSC12]. One naive way to plan a coherent motion trajectory τ could be

$$\begin{aligned}&\{\vec{\Theta}^*_{t_0}, \vec{\Theta}^*_{t_1}, ..., \vec{\Theta}^*_{t_H}\} = MP(\tau, Env)\\ &s.t.\ \forall\theta,\ \theta < \Psi,\ while\ \theta\ \in\ \{\theta_j, ...|\theta_j = (\vec{\Theta}_{t_{k+1}} - \vec{\Theta}_{t_k})\}\ 0 \leqslant k < H,\ .\end{aligned} \tag{2.7}$$

where MP is a motion planning function, Env denote relevant factors in the environment and Ψ is the threshold joint angular speed. The Equation constrains the first order derivative, velocity, under a certain threshold Ψ. A more fluent motion may constrain the second order derivative, acceleration.

Solving questions in Equation 2.7 requires *motion planning* in continuous state spaces. Given a desired trajectory τ, motion planning solves for a ideal sequence of joint angles

$$\{\vec{\Theta}^*_{t_0}, \vec{\Theta}^*_{t_1}, ..., \vec{\Theta}^*_{t_H}\} = MP(\tau, Env), \tag{2.8}$$

which can be executed collision free and low-cost. Our work considers the latter studies the higher level problem of coming up with the desired τ that fulfills some task of interest. Manipulation trajectory planning is discussed in Chapter 4.

In classical motion planning, both the arm and objects in the environment are explicitly modeled, often using polygons [AAS99], polyhedrons [SLG90] or voxels [LK02]. For simplicity of discussion, in this section, we refer to parts as all rigid bodies in the environment, which includes the objects and each mobile components of the robot arm.

With all parts modeled, the possible motions of the parts are reasoned with *rigid body* *transformations*, which can be loosely defined as a type of transformations that does not change the geometric integrity of the rigid object.

Meanwhile, not all parts of the environment can be freely transformed. For example, the links on robot arms are connected by joints with a limited range of angles. The possible transforms are studied by *configuration space*, which is formally defined by [LaV06] as

Definition 2.2.1. *The state space for motion planning is a set of possible transformations that could be applied to the robot.*

Sampling-Based Motion Planning

With the key concepts established, we introduce the *sampling-based motion planning* approach, which has been successfully applied to various real-world problems, including solutions implemented using classical software programming languages [BKV10] [KF10] [BB07] [PH10]. This approach also had groundbreaking achievements when implemented with deep learning models [IP19] [IHP18] [YA17].

Sampling-based motion planning attempts to find high-quality solutions by sampling and searching. The algorithm samples possible solutions Θ to individual desired states $\vec{s}$ in the *configuration space* denoted $\mathcal{C}$. *Metric Space* is established to measure the *distance* between different solutions $\Theta \in \mathcal{C}$. The algorithm then solves the optimization problem, the search for the optimal sequence, by minimizing total *distance*.

2.3 Grasping

Now that we briefly introduced motion planning for an end-effector, we may hope to use it to interact with objects. In this thesis, we will focus on robots with a mechanical gripper as the end-effector. Grasping is to firmly acquire the object of interest using the robotic arm's gripper. The fulfillment of many simple tasks can be decomposed into two phases: grasping, followed by manipulation. Firstly, grasp the object of interest, then manipulate the object to interact with the environment. Furthermore, many real-life single-hand chores can be accomplished by executing one or a sequence of those simple tasks. In this section, we briefly discuss the classical approaches for robotic grasping.

Form Closure and Force Closure

Mechanical grippers are designed to close on objects and grasp with surface contact. Two important properties of grasping, *form closure* and *force closure*, are defined as the capability of the robot to inhibit the motions of the object in spite of externally applied forces [Bic94]. *Form closure* defines the immobilization of objects using geometrics, such that the relative movement of the object requires bending the gripper. The latter prevents relative movement by equilibrium.

We define *wrench* as a vector of length 6 that describes the force and torque being applied on a rigid body, and *intensity vector* as a vector of contact force and moment components [PT16]. *Force closure* is achieved if, and only if, for any arbitrary external *wrench* $\vec{w}$ that the environment may apply, there exists an *intensity vector* λ satisfying the contact constraint inequalities [BK00], such that

$$W\lambda = \vec{w} \tag{2.9}$$

where W is the combined wrench. While both *form closure* and *force closure* are ideal conditions to facilitate object manipulation, the difference in the concepts is demonstrated in Figure 2.3.

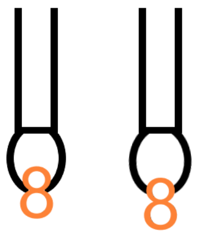


Figure 2.3: *Form Closure* (left) and *Force Closure* (right) Grasping a Number "8".

Grasping Quality and Grasping Planner

As important as the gripper's physical capabilities, the location on the object to grasp also greatly affects the outcome. Various *grasping quality* measurements has been established to quantify the preference of grasping configurations [FC92] [LS88] [HSSR05]. We use GQ to denote functions for calculating *grasping quality*, and μ to denote grasping configurations, which may include pose of the grasp, wrench vectors, and other relevant factors of the gripper.

Given an object described as Φ, a grasping planner function GP solves the optimal grasping configuration μ^* by maximizing *grasping quality*, given by

$$\mu^* = GP(\Phi) = argmax_\mu GQ(\Phi, \mu) \; , \tag{2.10}$$

where the candidate grasping configurations μ is proposed by a grasp sampler, often implemented as uniformly random or based on heuristics. The goal of a grasping planner is to suggest the ideal configuration μ^*.

Caging Configurations

Caging is an influential grasping planning method based on geometrics analysis [RMF12] [MKE12] [HNY13]. The work done by [DSFK08] defines[3] a *caging configuration* as

Definition 2.3.1. *Caging configuration is a grasp configuration μ that lies in a compact connected component of $M_f \bigcap M_\mu$, where M_f denotes the gripper's free movement space, and M_μ denotes the set of all configurations obtained by rigid body transformations from μ.*

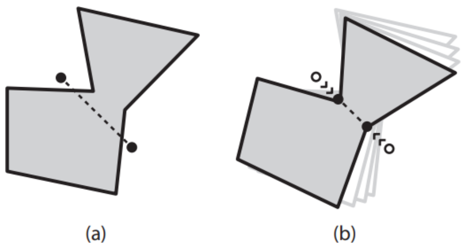


Figure 2.4: An Example of a Caging Grasp (reprinted from [DSFK08])

[3]The definition of 2.3.1 is based on [DSFK08]. However it has been rephrased, and the symbols are adapted to our notations.

Their work implements GP by identifying *caging configurations* as μ^*. An example is shown in Figure 2.4; on the left is a *caging configurations* μ, which leaves many possible configurations M_f. As the fingers on the gripper closes, using *rigid body transformations*, M_μ converges. Eventually $M_f \bigcap M_\mu$ becomes a set of configurations that achieves *form closure*.

Grasp Wrench Space Epsilon

In contrast, the work done by [WA12] uses Grasp Wrench Space Epsilon (ε_{GWS}) as *grasping quality* metric, which allows analysis of *force closure* solutions. Their work uses the widely cited ε_{GWS} metric [FC92], given by

$$\varepsilon_{GWS}(\mu) = argmax_{\varepsilon}\Big\{\{\ \vec{w}\ |\ \ ||\vec{w}||_2\ <\ \varepsilon\} \subseteq convexhull(\mu)\Big\}\ , \tag{2.11}$$

where $||\vec{w}||_2$ is the L2-norm of *wrench* vector $\vec{w}$, and grasp configuration μ is defined by the contact wrenches of the gripper. The optimal grasping configuration μ^* is then found by

$$\mu^* = argmax_{\mu}\big[\varepsilon_{GWS}(\mu)\big]\ . \tag{2.12}$$

Classical grasping methods often rely on explicitly modeled objects, and are computation intensive. Due to imperfect perception of the real world and limited computation power, the classical approach to grasping models real objects with oversimplified geometries and approximated material properties, and puts significant hardware constraints on real-time applications. Classical grasping analysis often requires controlled environments with prior knowledge [Sta90] [HPA+90].

This concludes our short discussion on the fundamentals of classical robotic motion planning and grasping. In the next two chapters, we will dive deep into learning-based grasping planning and manipulation planning. After that, we will be ready to present our proposed method.

3 Learning to Grasp

In the previous chapter, we discussed the challenges of classical grasping analysis, namely imperfect modeling of the real world and high computational cost. We will begin this chapter by introducing the data-driven approach to grasping, which has made groundbreaking progress in empowering practical grasping applications. Then, we will take a detour to introduce the relevant neural network layers used by our work, with an emphasis on Convolutional neural networks (CNN). After the necessary machine learning building blocks have been established, we will conclude this chapter by presenting a state-of-the-art open-source grasping planner, Dex-Net, as well as the custom modifications made to adapt it to our laboratory environment.

The Data Driven Approach

Grasping planners based on geometric analysis have proven successful with simple objects [Cra09], such as the work-pieces of an automated factory. However, everyday objects may have obscure geometry, either by design or due to damage. In order to build more versatile robots, more recent research has taken a data-driven approach. Those researchers aim to plan robust grasping on various objects and textures by learning from pre-constructed grasping databases. For example, [KBS15] proposed a large-scale database containing grasps that are applied to a large set of objects from numerous categories. The database contains objects that are represented by polyhedron models, and candidate grasps for those objects. Some examples in the database are shown in Figure 3.1.

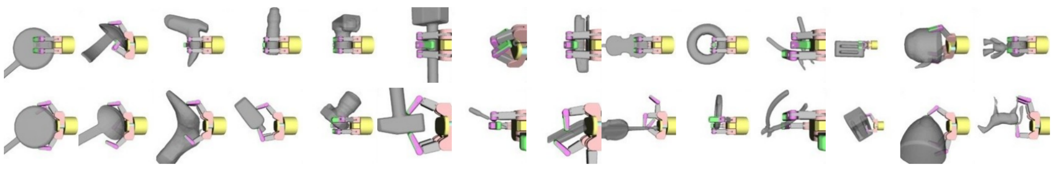


Figure 3.1: Various Objects in the Grasping Database Proposed by [KBS15] (reprinted figure)

A number of other researches have also proposed similar grasping databases, such as the popular *The Columbia Grasp Database* [GCDA09], *Human Grasping Database* [FB11], *Dex-Net* [MPH+16], and more [ZZMC13] [BL13] [LOJ14]. By providing a database of items and grasping quality estimations, those work paved the way for machine learning-based grasping planners, which have proven to be more effective for practical applications in complex environments [LPK+18] [MCL18] [JMS11] [JLD16]. A number of works combined learning-based grasping with one shot imitation learning [WD10] [VPSP17] [HEKL+13] [TTS+18] [CAGT18] [KW15].

The aforementioned digital databases were labeled by either the use of sophisticated physical simulations or human labelers. Instead of learning with an existing database, [PG16] learns to grasp in the real world using trial-and-error on real objects. Taking this approach farther, [LPK+18] conducted large-scale data collection using many robots experimenting in parallel and learning with the congregated dataset.

Deep learning-based grasping research solves Equation 2.12 by function approximation using neural networks. Given the object's feature vector, denoted by $\vec{\phi}$, the suggested grasping configuration μ is computed by

$$\mu = \mathcal{H}(\vec{\phi}), \tag{3.1}$$

where $\mathcal{H}$ is a function approximate maximizing $GQ(\vec{\phi}, \mu)$.

3.1 Convolutional Neural Networks for Perception

The field of robotic grasping has achieved groundbreaking success with the advent of deep learning [LPK+18] [JLD16] [FBH+18] [CAGT18]. Meanwhile, CNN have been pushing the limits of cognitive computing [SU15], especially for visual applications such as image classification [HHW+15], object detection [HCH+17] and semantic segmentation [CPK+17]. Moreover, CNN was been a key component of both the open-source grasping planner [MLN+17] used during our experiments, and our proposed manipulation planning network.

Standard Convolutional Layer

Perceptrons [MP17] are the most basic type of neurons in machine learning[1] networks.

$$\vec{y} = f(\vec{x}\,\vec{w} + \vec{b}), \tag{3.2}$$

where given $\vec{x}$ the input vector of the perceptron, the weight vector, the bias vector, and $\vec{y}$ is the output vector.

A fully connected layer (FC) [SU15] uses Perceptrons to connect each pixel of the previous layer to each pixel of the next. Compared to a FC, which perceives the information from the previous layer all at once, a convolutional layer introduces a shifting window, which sets a limit on the information available locally. By doing so, the model takes the underlying geometrical relationship of the input information into consideration. If applying a fully connected layer is to gaze into the night sky, applying a convolutional layer is comparable to scanning through the sky with a telescope.

A classical convolutional neural network is typically constructed by repetitively applying pairs of convolutional layers and pooling layers. As they get deeper down the network, those pairs of layers reduce the width and height of intermediate tensors' shape, while increasing the number of channels [MZZS18] [HMDC16] [TSM+16]. The design pattern of CNN is usually justified by the following reasoning: In the raw input image, information is

[1]To limit scope of the background chapter of this thesis, we will omit the introduction to the basics of machine learning.

geometrically structured on a 2D panel, and, as each layer is evaluated, low-level information in the tensors gets extracted and encoded into higher-level abstraction represented by each channel.

CNN has proven to be viable with 1D sequences [KIG15] [Mit15] [Kim14], 2D images [KSH12] [XRLJ14], while promising researches [MS15] [KCL+15] have also demonstrated its potential, classifying objects in scenery represented by 3D point clouds. This trend demonstrated CNN can comprehend various datasets by adapting the shapes and ranks of its kernels, which defines the behavior of the shifting window of perception.

In the following sections, we discuss popular variations of CNN in the context of 2D space within the image processing domain, as those are the most typical and successful use cases of CNN so far, in terms of both research activity and commercialized industrial applications. Note that 2D convolution takes 3D tensor as input, because each pixel at a 2D panel can have multiple channels, representing different signals, such as RGB color values or other abstract concepts.

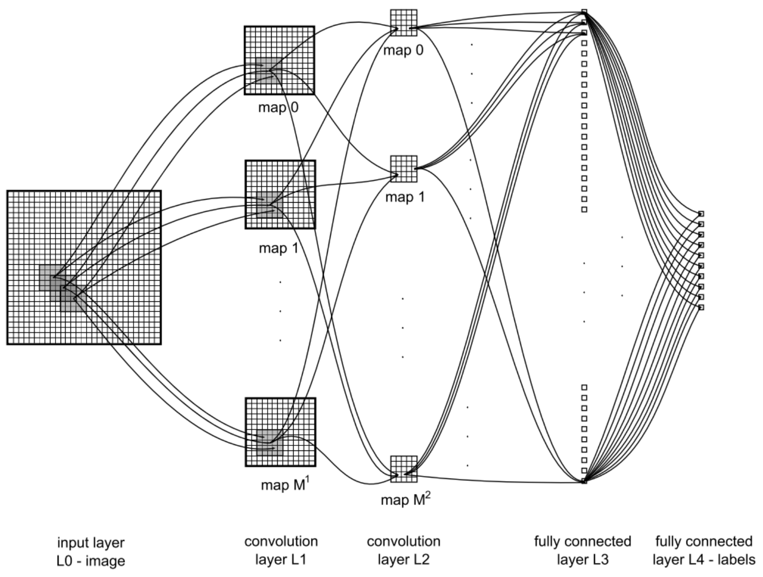


Figure 3.2: Architecture of a Convolutional Neural Network (reprinted from [CMM+11])

Convolutional layers are designed to extract high-level features from the input tensor

[LB+95]. At each location, feature extraction is done by computing the Frobenius inner product of a trainable weight tensor against the overlapped input tensor, which is called a kernel or a filter. The output is a measurement of how well local information in the input tensor matches the learned feature of interest. This is the basic unit of operation, which is called convolving. The operation is repeatedly performed for each 2D location, by shifting the observation window on the 2D panel of the input tensor, while the kernel remains the same in value. Convolving a 3D input tensor with a 3D kernel weight tensor produces a 2D output tensor. The results of multiple convolvings with multiple kernels are stacked together into a 3D output tensor. Convolutional layers have more expressive capacity compared to fully connected layers; the latter can be considered as a special case of the former. While Figure 3.2 shows one example of a Convolutional Neural Network, Listing 3.1 shows a C implementation of a vanilla 2D convolution layer, without padding nor striding.

Listing 3.1: Vanilla 2D Convolution Algorithm

```c
// in  : 3D input tensor of shape [inHeight, inWidth,
   inChannels]
// k   : 4D kernel tensor of shape [outChannels,
   kernelHeight, kernelWidth, inChannels]
// out : 3D output tensor of shape [(inHeight -
   kernelHeight + 1), (inWidth - kernelWidth + 1),
   outChannels]
for (int h = 0; h < inHeight - kernelHeight + 1; h++) {
  for (int w = 0; w < inWidth - kernelWidth + 1; w++) {
    for (int oc = 0; oc < outChannels; oc++) {
      out[h][w][oc] = 0;
      for (int kh = 0; kh < kernelHeight; kh++) {
        for (int kw = 0; kw < kernelWeight; kw++) {
          for (int ic = 0; ic < inChannels; ic++) {
            out[h][w][oc] +=
              in[h+kh][w+kw][ic] * k[oc][kh][kw];
          }
        }
      }
    }
  }
}
```

This concludes our introduction of the neural network layers most relevant to our work.

3.2 Dex-Net 2.0

With building blocks of neural networks established, we are ready to present the open-source grasping planner that we adapted to validate the viability of our learning system. Because we designed our learner to learn tasks in two separate phases, namely grasping and manipulation, a robust solution that enables the robot to acquire the object of interest is a prerequisite.

Dex-Net 2.0 is a state-of-the-art robotic object grasping research project [MLN+17]. Our work adopted a part of their grasping framework, as well as a pre-trained grasping planner network checkpoint, which is trained by [MLN+17] with the public Grasp Quality Convolutional Neural Network (GQCNN) database. In addition, we implemented a custom denoising module that adapted the pre-trained network to our laboratory environment.

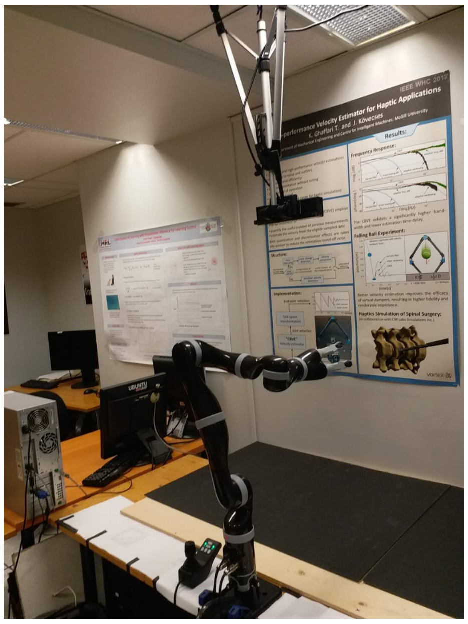


Figure 3.3: Upside Down Mounted Tripod and Camera

Dex-Net includes code, datasets, models and algorithms for robotic grasping. We leveraged its pre-trained grasping planner network, which is capable of predicting robust grasps from RGBD images of the object of interest. As a requirement of the framework, the direction of the view has to be the same as the direction of grasping. Although a single camera cannot guarantee a full view of every point of interest, to better the chance of having a clear view and to avoid unnecessary background noise, we set up the camera directly above the scene, as shown in Figure 3.3.

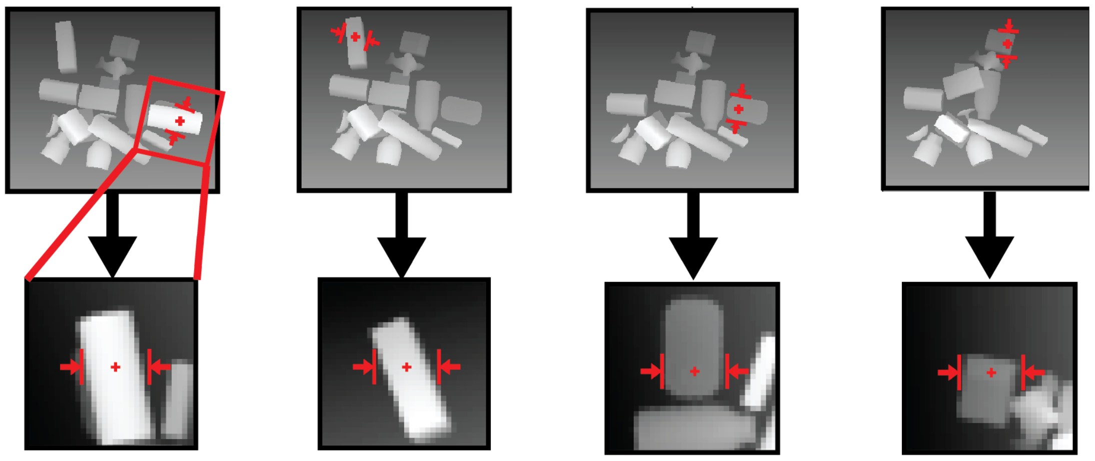


Figure 3.4: Sampled Grasps [MLN+17]

Initially, the camera takes a depth image snapshot of the scene, which is sent to the GQCNN pipeline. The pipeline includes multiple heuristic samplers, which pick potentially viable grasping points from the depth image. Each grasp is described by 4 values: 3D positions and the orientation of the parallel gripper. For each grasp, a smaller depth image would be produced by resizing the original image to the same scale required by GQCNN, rotated to align with the supposed gripper's direction, and finally clipped to the same size. As annotated by the red bounding boxes in Figure 3.4[2], each of the sampled images[3] would represent a candidate grasp.

[2]Figures for Dex-Net were copied from the original paper published by Berkeley Automation Lab.

[3]The grey images are depth maps, with lighter pixels closer to the camera, darker pixels farther. Red lines are not a part of the original image, but only visual annotations.

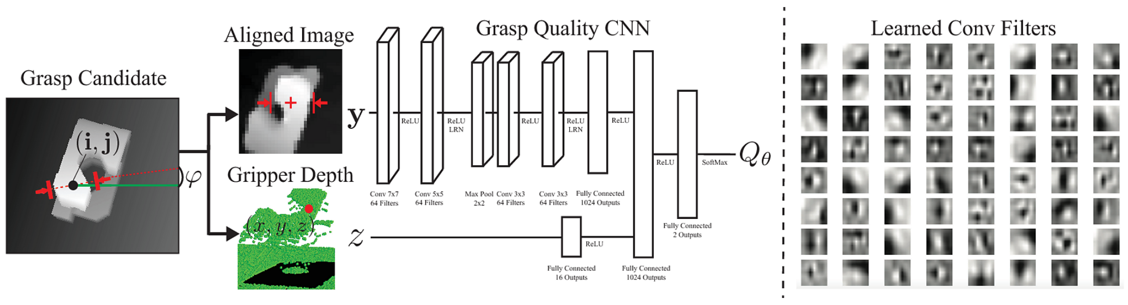


Figure 3.5: GQCNN model architecture [MLN+17]

Our work took a checkpoint from the open-sourced GQCNN model, which was pre-trained by a large set of labeled training images similar to our sampled grasp candidates. As shown in Figure 3.5, the model used a typical CNN architecture, which consists of a stack of convolution layers, fully connected layers, and pooling layers. At the very last layer, the signal was passed through a softmax function, which outputs the estimated quality of the grasp.

At last, the grasp candidate with the highest estimated grasp quality would be selected by the GQCNN package, which then would be executed by our high-level JACO2 arm controller. Interestingly, those training images, labeled with supposed ground truth, are synthetically produced by sophisticated simulations. The images are rendered depth maps of virtual 3D object models, while the labels are grasp-quality values estimated by a sophisticated physics simulator.

3.2.1 Custom Denoising Module

Backed by the experimental results of both the original Dex-Net research and ours, the model adapted very well from simulation to the real world. However, we found that, because the original model was trained with perfect depth images rendered from simulation, the model is extremely sensitive to background noise, including small cracks on the table, uneven surface of objects, and noise introduced by the depth sensor error. Since the original research was for grasping only, as shown in Figure 3.6, the visible area of the workspace is very limited and mostly free of visual distractions. However, our work needs a full view of the larger workspace, while working around the limitations of the Kinect V2 camera's 50cm minimum depth range. As a result, we also developed a lightweight, heuristic-based depth signal denoising project implemented with the Point Cloud Library (PCL). PCL uses a range of sophisticated filtering techniques [HJW+17], removing obvious sensor errors, as well as the nuances on object surface. Eventually, with the denoised depth image, the GQCNN package demonstrated ideal performance in our experimental environment.

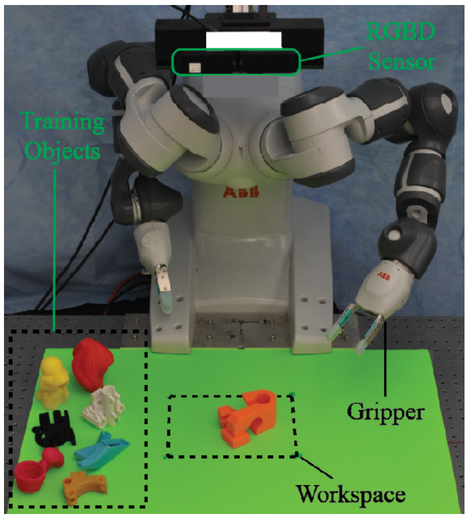


Figure 3.6: Dex-Net 2.0 Workspace [MLN+17]

Although Dex-Net requires grasps to be made by parallel-jaw, the gripper on the JACO2 arm has 3 fingers. We only used 2 of those to function together as a parallel-jaw. In practice, this was proven to be a reasonable approximation. The final software product was able to consistently suggest robust grasps, and to work in conjunction with the robotic arm to reliably grasp various objects. We will postpone the experimental verification of this unit until Chapter 7.

In this chapter, we presented the neural network layers relevant to our thesis, and also discussed data-driven grasping planners. Both are key components that made our experiments possible. In the next chapter, we are finally ready to discuss imitation learning, which is the main research field of this thesis.

4

Learning to Manipulate by Imitation

In the previous chapters, we discussed the machine learning approach for both robotic motion planning and grasping. Now that we can move the robot and acquire objects with its gripper, we have an opportunity to use the object as a tool to interact with the world.

In specific, Chapter 2 established the motion planning problem, where given the desired trajectory τ, we can control the robot using joint angles solved by

$$\{\vec{\Theta}_{t_0}, \vec{\Theta}_{t_1}, ..., \vec{\Theta}_{t_H}\} = MP(\tau, Env) \; . \tag{4.1}$$

In this chapter, we will discuss the problem of planning the trajectory τ to perform high-level tasks, such as pouring.

Most real-world object manipulation tasks can be performed by many equally viable trajectories. For example, there can be infinite ways to peel an orange, while the orange can also be presented in infinitely many different scenarios. Although software solutions such as MOVEit [SMK12] can solve joint motion for given trajectories, developing a rule-based planner for each manipulation scenario is not viable for complex environments and objects. In addition, the system cannot be easily expanded for new tasks; a new system has to be programmed again to peel bananas.

One way to build a versatile robotic system is to make it capable of performing imitation learning (IL) with a demonstration dataset that is easy to produce. In the same way a human apprentice learns, ideally, robots should be able to learn various tasks by simply

Env	Environment: Everything the robot can interact and observe
t	Time: The current timestamp
$\mathbb{S}$	State: Observed information of the agent
$\mathbb{A}$	Action: Instructions issued to the robot's controller
τ	Trajectory: A sequence of actions and their resulted states
$\mathbb{T}$	Transition Probability Matrix: Probabilities that models the environment dynamics
π	Policy: The strategy to decide the next action, given the current state
$\mathbb{R}$	Reward: A quantified feedback from Env at a certain timestamp
γ	Discount Factor: May weigh immediate reward to be more valuable than future reward

Table 4.1: Notations and Concepts

watching expert demonstrations, without requiring explicitly hard-coding the reaction for every possible situation.

This chapter begins with a brief illustration of the imitation learning algorithms that are fundamental or most relevant to our research. In later sections, we present the Generative Adversarial Networks (GAN), and its IL adaptation, Generative Adversarial Imitation Learning (GAIL).

4.1 Reinforcement Learning and Markov Decision Process

Alongside supervised learning and unsupervised learning, Reinforcement Learning (RL) is one of three main branches of machine learning, which attempts to deduce the strategies by learning from the agent's interactions with the environment. Table 4.1 summarizes key notations and concepts of RL. The overall goal is to find a learner policy π_l which maximizes the total reward over a limited time.

Markov decision processes (MDP) are the theoretical foundation for reinforcement learning algorithms. MDP has frequently been used as the go-to solution or baseline approach for optimization problems in many domains [HS15] [OVR14] [AET+14] [DFL14] [EDK14] [Ser09]. A MDP is a process which satisfies the Markov property, which is defined [OPN+18] as

Definition 4.1.1. *A process $s_0, ..., s_t$ has Markov property if at any time t, the future probability to arriving into states $s_{t+1}, s_{t+2}, ...$ depends on the history $s_0, ..., s_t$ only through the present state s_t.*

Many reinforcement learning researches use MDPs to model Env and solve for the optimal decision making policy function π. Reinforcement learning introduces actions and rewards to MDP, denoted by set $\mathbb{A}$ and set $\mathbb{R}$, respectively. At each timestamp t, an action $a_t \in \mathbb{A}$ is taken, causing the agent's state to transition from s_t to s_{t+1}, where $s \in \mathbb{S}$. Meanwhile, the agent is expected to be rewarded by

$$r^a_{s->s'} = \mathbb{E}[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'], \tag{4.2}$$

where $r \in \mathbb{R}$, $s \in \mathbb{S}$ and $a \in \mathbb{A}$.

In a stochastic environment, a given action may possibly bring the agent to different states. This is modeled by a transition probability matrix $\mathbb{T}$,

$$\mathbb{T}^a_{s->s'} = p(s_{t+1} = s'|s_t = s, a_t = a). \tag{4.3}$$

As time progresses, the agent collects rewards. The goal of the agent is to maximize the total reward accumulated under finite time [0, t_H]. To regulate the agents to complete tasks sooner rather than never, a discount factor γ is introduced to insert preferences to early rewards. The overall goal is measured by a goodness function,

$$\mathbb{J}(\pi) = \mathbb{E}_\pi\Big[\sum_{t=0}^{t_H}(\gamma^t r_t|s_t = s)\Big]. \tag{4.4}$$

The parameters of an MDP, such as $\mathbb{R}$ or $\mathbb{T}$, may be handcrafted, or learned using data-driven approaches, such as deep learning. Eventually, we aim to find an optimal policy π^* that maximizes total rewards,

$$\pi^* = argmax_\pi \mathbb{J}(\pi)\ . \tag{4.5}$$

With a parameterized MDP, we can compare if one state is preferable to another, using the State Value Function,

$$V_\pi(s) = \mathbb{E}_\pi\Big[\sum_{k=t}^{t_H} \gamma^{k-t} r_{k+1} \Big| s_t = s\Big]. \tag{4.6}$$

Given a state, we can also compare if taking one action is preferable to another, using the State-action Value Function,

$$Q_\pi(s,a) = \mathbb{E}_\pi\Big[\sum_{k=t}^{t_H} \gamma^{k-t} r_{k+1} \Big| s_t = s, a_t = a\Big]. \tag{4.7}$$

4.2 Imitation Learning Algorithms

With the help of motion planning, robots can be programmed to execute the trajectories with incredible efficiency and precision. However, in order to build a versatile robot, there is a gap between a desired high-level task and planning out a trajectory that fulfills the task. Machine learning algorithms use a data-driven approach to enable machines to bridge this gap. By generalizing the problem-solving strategies, those algorithms attempt to learn to perform similar tasks, without requiring explicit coding for each scenario.

As a branch of RL, IL also attempts to deduce the strategies by indirectly observing the expert agents' demonstrations. IL aims to open the possibility to demonstrate in heterogeneous space, where the robot agent could directly learn from humans, rather than learning from another agent controlled or programmed by humans. In addition, IL could enable robots to learn from observing, rather than deliberate teaching. Many researches have made significant progress with this approach [FYZ+17] [DAS+17] [YALF18] [ZWM+18] [YALF18] [JBD18] [ZMJ+18].

Value Iteration

With the MDP-based value functions defined, value iteration computes the optimal state value function by iteratively improving the estimate of $V_\pi(s)$. Eventually V and Q would converge to reach the optimal policy π^*,

$$V_\pi(s) = argmax_a \Big[Q_\pi(s) \Big]. \tag{4.8}$$

Policy Iteration

However, the state-action value function's space can be much larger than the policy function's space. To address this issue, we can directly optimize the parameterized policy function by updating at each state $s \in \mathcal{S}$ with

$$\pi^*(s) = argmax_a \Big\{ \mathbb{E}_\pi[r|s,a] + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \Big\}. \tag{4.9}$$

Trust Region Policy Optimization

With machine learning, the policy π is approximated by a differentiable function π_θ, which is implemented by a neural network parameterized with θ. Equation 4.9 is updated by taking a gradient step on the function evaluated on a batch of inputs. However, the best size of the gradient step is unclear. Errors made at each state accumulate, where taking a large step on a particular batch may regress the overall true objective of the policy function. As a result, the performance of the policy may oscillate during training.

Trust Region Policy Optimization (TRPO) seeks a more monotonic improvement of the policy by[1] performing

$$\theta_{k+1} = \arg\max_\theta \ \mathcal{L}(\theta_k, \theta) \text{s.t. } \vec{D}_{KL}(\theta || \theta_k) \leqslant \delta, \tag{4.10}$$

where $\mathcal{L}(\theta_k, \theta)$ is the surrogate advantage, a measure of how policy π_θ performs relative to the old policy π_{θ_k}, using data from the old policy,

[1]The equations were a summarization of the OpenAI Spinning Up's education material, which is available at https://spinningup.openai.com/en/latest/algorithms/trpo.html .

$$\mathcal{L}(\theta_k, \theta) = s, a \sim \pi_{\theta_k} \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \tag{4.11}$$

and, in order to measure the distance between policies across states visited by the old policy to the new, TRPO uses average KL-divergence [KL51] $\vec{D}_{KL}(\theta||\theta_k)$,

$$\vec{D}_{KL}(\theta||\theta_k) = s \sim \pi_{\theta_k} D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)\right). \tag{4.12}$$

4.2.1 Policy Imitation

While the core idea is to imitate various tasks from demonstrations, there are currently two main families of imitation learning algorithms – one directly optimizes the learner's policy function by imitating expert policy function, while the other, firstly, learns the underlying reward functions, then deduces the learner's policy function [OPN+18]. The two approaches are called behavior cloning and inverse reinforcement learning, respectively.

Policy imitation attempts to directly mimic the expert's reaction to each different situation, predicting what action would the expert take under a given state. Assuming the expert's behavior is based on a fixed policy $\pi(S|A) = \mathbb{P}[a|s, \theta]$, where θ represents a parameter carefully selected by the expert to perform the task. Policy imitation algorithms directly learn π, without reasoning about the underlying motivation.

Behavior Cloning

Learning expert policy π_e is simply learning its state-to-action mapping. Ideally, if we can sample a large enough amount of state-action pairs from π_e, where s sufficiently covers the whole state space, then the problem can be solved by supervised learning. As the size of this balanced training data increases, the learner should be able to clone the expert's behavior in any given situation. This is the core idea behind the widely successful behavior cloning (BC) algorithm. In recent years, various practical engineering problems have adopted machine learning-based solutions [CSKX15] [MDGK18] [CSLG19], rivaling or out-performing classical software engineering approaches in many cases. This approach is particularly successful in industrial applications where the ground truth solution can be easily labeled. In the simplest scenario, the model memorizes all the state-action pairs in the training dataset, and simply plays back the action during testing; however, this data distribution assumption is not practical.

While the overall goal is to learn a policy that completes a task by reproducing the expert's trajectory, given a current state, the problem can be further broken down into either a question of what is the desired next state, or what is the desired next action. The former, categorized as model-based BC [OPN+18], assumes the state transition probabilities are known, and then the desired action can be solved accordingly; meanwhile, the latter is categorized as model-free BC, which directly learns the expert's action [SB18]. As a trade-off for relaxing the assumption of known environment dynamics, model-free BC naturally requires more training data samples to implicitly estimate the transition probabilities.

In a practical application with human demonstrations, the expert's experience often progresses along the successful paths in the state space, where the agent follows ideal trajectories. Because the learner agent's data distribution depends on its past actions taken and states arrived, the learner's experience could be vastly different from the expert's. The naive version of BC suffers heavily from covariate shift in the training data distribution. To explain in detail, a different action chosen at an earlier time or an external random factor might cause a butterfly effect and lead the agent to a completely unfamiliar state space, and then the agent would essentially be operating in untrained space. Moreover, each error diverging from the ideal trajectories requires a correction; such recovery interactions are often never demonstrated by the expert, as those undesired situations were avoided by the expert in the first place. As the compounding error grows exponentially, the agent may be trapped in an untrained situation and never find the way back.

Although BC suffers from covariate shift [Pom89] [LMDBS18] [MBC+06], successful applications have been developed in demonstration rich scenarios, such as unmanned aerial vehicle (UAV) navigation [GGC+15]. For this particular problem, a large amount of demonstration data could be easily produced by a hiker with a mounted camera. Also, the navigation software of UVA provides an abstract interface, which reliably executes high-level movement instructions, essentially simplifying the problem to a model-based scenario. Our work also assumes a model-based scenario, taking advantage of the Kinova JACO2 robotic arm's high-level control interface.

Dataset Aggregation

For imitation learning problems involving a human expert, the state distribution from training data is not independent and identically distributed (i.i.d). As previously discussed, the supervised learning BC approach suffers heavily from covariate shift.

Dataset Aggregation (DAgger) [RGB11] addresses this issue by explicitly having the expert to demonstrate actions for states encountered by the learner. This is done by:

1. Training a learner's policy π_l on the original state-action pair dataset $\mathbb{D}$
2. Running π_l to produce a trajectory $\tau = \{(s_0 \rightarrow a_{l_0}), (s_1 \rightarrow a_{l_1}), (s_2 \rightarrow a_{l_2}), ...\}$
3. Running expert's policy π_e on $\{s_0, s_1, s_2, ...\}$, to produce $\{a_{e_0}, a_{e_1}, a_{e_2}, ...\}$
4. Aggregating $\{(s_0 \rightarrow a_{e_0}), (s_1 \rightarrow a_{e_1}, (s_2 \rightarrow a_{e_2}), ...\}$ to $\mathbb{D}$
5. Improving π_l with $\mathbb{D}$ using supervised learning
6. Repeating step 2 to 5 many times

In step 2, the learner policy has the opportunity to directly interact with the environment. In steps 3 to 5, the learner gets tailored feedback from the expert, guiding it to improve π_l towards mincing π_e. This makes policies trained by DAgger strictly better than naive supervised learning on the original $\mathbb{D}$. As shown by [RMBS+13], this algorithm can be effective in solving real-world UVA problems. However, both step 2 and step 3 could be very limiting when applying DAgger to solve large-scale problems.

Step 2 imposes a constraint on the viability of online demonstration. Considering an autonomous driving example, the online demonstration would require a human driver to pseudo-drive a car, but only record the state-action pairs $\{(s_0 \rightarrow a_{e_0}), (s_1 \rightarrow a_{e_1}, (s_2 \rightarrow a_{e_2}), ...\}$, while ignoring the expert's control and letting the learner π_l operate the car at the same time. Putting the learner policy π_l in control could be expensive and dangerous in practice, with undesired trajectories resulting in crashing into obstacles or pedestrians. This obvious issue of practicality imposes significant constraints in developing practical applications of DAgger.

Step 3 requires the expert to explore trajectories produced by the learner, which makes DAgger not directly applicable to another important subarea of imitation learning, where the robot and expert are in heterogeneous state and action spaces. An example could be when humanoid robots learning to perform a basic human movement from human demonstration. Industrial applications of this approach have been fruitful in animations, computer generated imagery, as well as in physical robot products. In this type of problem, the expert and the robot agent would have different skeletons, joint properties, and other physical properties. Meanwhile, because DAgger requires the expert to suggest actions at the learner's state spaces, a mapping between the two spaces has to be established, which, in itself, creates another research area.

4.2.2 Learning Rewards

This family of imitation learning algorithm attempts to deduce the underlying goals of the expert policy, rather than simply mimicking the behavior of the expert. Reward learning algorithms attempt to breakdown the overall task into quantifications of many internal objectives. The algorithm is implemented by first learning a set of reward functions which accumulatively contributes to the overall goal. The target policy is then inferred by finding the state action pairs which optimizes the previously learned reward functions.

Reward functions can be handcrafted equations or, more commonly in machine learning, learned by neural networks trained by demonstration datasets. In practice, those functions only imperfectly reflect the true causation connection between the current state and progress of the task: oversimplifying the system dynamics, while neglecting less notable factors relevant to the task. The nature of this data-driven approach dictates that, if the learned reward function model fails to keep generalization error under control, the reward function could display undesired behavior in untrained state space. Those issues are common in the realm of machine learning affecting the effectiveness of learning. After all, one can argue that humans would face similar challenges when attempting to master a new task.

Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL), also known as Inverse Optimal Control (IOC), attempts to learn from experts by first deducing their internal motives [NR+00]. To solve the original reinforcement learning task, we define an IRL problem which is the dual of the RL problem. In specific, the IRL problem here is to infer the hidden reward functions from demonstration trajectories produced by a fixed expert policy, which we assume is capable of always maximizing the reward functions. Then the remaining part of the puzzle becomes to deduce a learner policy which, in turn, maximizes those reward functions. This idea can be traced back from researches as early as the Linear Quadratic Regulator.

Similar to BC, IRL can also be classified into model-based or model-free. The former requires known state transition probabilities, which adds significant assumptions to the environment. In contrast, model-free IRL not only indirectly learns reward function, but also indirectly approximates the system dynamics. Both factors require a large size of training samples and intensive computation to learn. In practical robotic applications, it is crucial to properly model the uncertainty introduced by noise and stochastic factors in the environment. While intuitive and elegant, model-free learning requires a large amount of training data to neutralize noise and stochastic factors, as well as to improve generalization.

For real world IRL applications, the modeling of rewards using handcrafted formulas is often an oversimplification. Some notable variation of the IRL algorithm improved learning by explicitly modeling the decision making process, which indirectly models the reward mechanism. Two of the most notable approaches are introduced below.

Maximum Entropy Inverse Reinforcement Learning

Maximum Entropy IRL [ZMBD08] learns the internal reasoning of a Markov Decision Process, and solves the maximum likelihood of the reward function parameterizing the MDP. The deep learning adaptations of the algorithm have also achieved much success [WOP15] [GH18] [PSS+16] [HZAL18] riding the surge of deep learning. The algorithm, shown in Listing[2] 4.1, attempts to train neural networks to estimate the optimal policy $\pi(a|s)$, which is learned by taking gradient steps with

$$\nabla_\theta \mathcal{L} = \frac{1}{|D|} \sum_{\tau_d \in \mathcal{D}} f_{\tau d} + \sum_{s \in \mathcal{S}} p(s|\theta, T) f_s, \tag{4.13}$$

where $f_{\tau d}$ is the feature vector extracted from the batch of demonstration trajectories

[2]The pseudo code was replicated from a slide of UC Berkeley's online course, CS 294 episode 14 of Deep Reinforcement Learning.

and f_s is the feature vector extracted from $s \in \mathcal{S}$.

Listing 4.1: Maximum Entropy Inverse Reinforcement Learning Algorithm

```
Notation:
  $\tau$: trajectory = {(s_1,a_1), ..., (s_t,a_t), s_T}
Input:
  $\mathcal{D}_E$: demonstration dataset
  $T$, state transition dynamics
  $\theta$, parameters for the reward function
  $f$, feature vector extracted from trajectory
Algorithm:
Initialize $\theta$
Solve for optimal policy $\pi(a|s)$ with respect to $\theta$ with
    value iteration
Solve for state frequencies $p(s|\theta, T)$
Update $\theta$ with gradient step using Equation 4.13
```

Bayesian Inverse Reinforcement Learning

Bayesian IRL algorithm [RA07] models the decision making process with Gaussian Process (GP), which attempts to improve learning under imperfect environment modeling and possibly insufficient demonstration training data. Like Maximum Entropy IRL, Bayesian IRL also benefits from the advent of deep learning [OA16] [ZLN14] [CK14] [IBN18]. The algorithm [CK11] models the reward function by assuming a behavior previously defined as

$$P(R) = \prod_{s \in \mathbb{S}, a \in \mathbb{A}} P(R(s, a)). \tag{4.14}$$

The likelihood function was then defined as an independent exponential distribution with

$$P(X|R) = \prod_{m=1}^{M} \prod_{h=1}^{H} P(a_h^m, s_h^m, R). \tag{4.15}$$

Finally, the posterior $P(R|X)$ reward distribution can be deduced by combining the previous and the likelihood function using Bayes theorem, as

$$P(R|X) \propto P(X|R)P(R). \tag{4.16}$$

4.3 Generative Adversarial Algorithms

The field of robotic object manipulation has also achieved groundbreaking success with deep learning [LPK+18] [JLD16] [FBH+18] [CAGT18]. Many modern robotic systems are capable of performing various tasks [CHC+12] [AHS+09].

Meanwhile, many real-world robotic problems require unsupervised learning, because they often lack ground truth datasets, or a way to quantify the rewards. Generative adversarial algorithms made significant progress in studying those tasks, which supposedly require human-level intelligence and creativity.

Our work studied an IL problem, which is very suitable for generative adversarial algorithms. In this Section, we will first introduce GAN, and discuss how it could be applied to IL problems. Then, we will introduce Generative Adversarial Imitation Learning (GAIL), which is a successful adaptation of GAN into RL.

4.3.1 Generative Adversarial Networks

Generative Adversarial Networks [GPAM+14], introduced by Ian Goodfellow and his colleagues at the University of Montreal, has made a significant contribution to semi-supervised learning and reinforcement learning [KMWK17] [LSE17] [CCK+18] [LTH+17] [WZX+16] [SAS17]. GAN has seen remarkable success in learning tasks which intend to generate novel solutions that have the same distribution and characteristics as the training data. Its quintessential applications are in domains such as image generation [LTH+17], image translation [CCK+18] and 3D modeling [WZX+16]. The algorithm essentially constructs a zero-sum game between a Generator and a Discriminator. The Generator network learns to generate solutions by using random noise, or seeds, as the network's input. This is achieved by learning a Generator network, which was fitted by gradient defined by

$$\nabla_{\theta\mathcal{G}} \frac{1}{m} \sum_{i=1}^{m} log(1 - \mathcal{D}(\mathcal{G}(z^{(i)}))), \tag{4.17}$$

where notations will be defined in Listing 4.2.

Meanwhile, its counterpart, the Discriminator, learns to discriminate the Generator's solutions from a ground truth dataset sampled from a targeted distribution. This is achieved by learning a Discriminator network, which was fitted by gradient defined by

$$\nabla_{\theta\mathcal{D}} \frac{1}{m} \sum_{i=1}^{m} [log\mathcal{D}(x^{(i)}) + log(1 - \mathcal{D}(\mathcal{G}(z^{(i)})))], \tag{4.18}$$

where notations will be defined in Listing 4.2.

If the Discriminator has realized all its potential, but the Generator can still generate solutions that confuse the Discriminator, then it means learning is effective, and the generated solutions can approximate the targeted distribution. The algorithm is shown in Listing[3] 4.2.

Listing 4.2: Algorithm For Training Generative Adversarial Networks

```
Notation:
  $P_g$: Generator distribution
  $P_{data}$: Training data distribution, the ground truth
Input:
  $t$: The number of training iterations
  $k$: The number of steps to apply to the Discriminator
  $P_z(z)$: A pre-defined noise prior
  $\mathcal{X}, x$, Training data
  $\mathcal{Z}, z$, Noise, seed to the network
  $\mathcal{G}, \mathcal{D}$, Generator and Discriminator, both differentiable
     functions represented by multi-layer perceptrons
Algorithm:
for $t$ iterations do
  for $k$ steps do
    • Sample minibatch of m noise samples {$z^{(1)}$, ... ,$z^{(m)}$}
       from noise prior $P_g(z)$
    • Sample minibatch of m examples samples {$x^{(1)}$, ... ,
       $x^{(m)}$} from data generating distribution $P_{data}(x)$
    • Update the Discriminator by ascending its
```

[3]The pseudo code was replicated from the original GAN paper [GPAM+14].

```
          stochastic gradient with Equation 5.13
    end for
    • Sample minibatch of m noise samples {$z^{(1)}$, ... ,$z^{(m)}$}
      from noise prior $P_g(z)$
    • Update the Generator by descending its stochastic
      gradient with with Equation 5.15
end for
```

Attention from both academia and industry has been attracted by GAN's successful adaptation in various problem domains. GAN thrives with tasks involving mimicking training data distribution, and even exhibited some level of creativity. High-profile successes are often in fields such as visual imagery [DRG15] [LPMG17] [MJS+17] [OOS17], and time series synthesis [EHR17] [DMP18] [HSB18] [LCS+19]. The latter is closely related to the research of this thesis.

For the IL object manipulation problem that we studied in this thesis, we are given a small number of expert demonstration trajectories P_{data} that perform a certain task. Those trajectories are sampled from a much larger distribution P^* of all potentially viable trajectories that fulfill the task. If we can use the GAN Generator $\mathcal{G}$ to produce trajectory τ_g from P^*, then we can use $\mathcal{G}$ to be a manipulation trajectory planner for this task.

4.3.2 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning [HE16] is a more recent addition to the IRL family. As the name implies, GAIL is an adaptation of GAN, which was introduced in Section 4.3.1, into the realm of imitation learning. As a generative adversarial algorithm, the GAIL algorithm also consists of two sub-networks. One is a GAIL Generator that reassembles the learner's policy, which was trained by gradient descent on

$$\nabla\mathcal{G} = \hat{\mathbb{E}}_{\tau_i}[\nabla_\theta log(\pi_\theta(a|s)Q(s,a)] - \lambda\nabla_\theta H(\pi_\theta), \tag{4.19}$$

where $Q(\vec{s},\vec{a}) = \hat{\mathbb{E}}_{\tau_i}[log(\mathcal{D}_{w_{i+1}}(s,a))|s_0 = \vec{s}, a_0 = \vec{a}]$, and $H(\pi)$ defined by $\mathbb{E}_\pi[log\pi(a|s)]$

is the -discounted causal entropy [3] of the policy π.

The other is a GAIL Discriminator, which can be considered a trainable loss function of the Generator. It is trained by gradient descent on

$$\nabla \mathcal{D} = \hat{\mathbb{E}}_{\tau_i}[\nabla_w log(\mathcal{D}_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w log(1-\mathcal{D}_w(s,a))]. \tag{4.20}$$

The network learns by alternatively improving the two players. When the Discriminator can no longer distinguish generated distribution from the ground truth demonstrations, then the learner policy has learned to imitate the expert policy. The algorithm is shown below.

Listing 4.3: Generative Adversarial Imitation Learning Algorithm[4]

```
Notation:
  $\pi_E$: Expert policy
  $\tau$: trajectory
Input:
  $\tau_E$: Expert trajectories sampled by $\pi_E$
  $\mathcal{D}$: Discriminator with initial parameters $w_0$
  $\mathcal{G}$: Generator with initial parameters $\theta_0$
  $N$: Number of iterations
Algorithm:
for i from 0 to $N$ do
    • Sample trajectories $\tau_i$ by $\pi_{\theta_i}$
    • Update the parameters from $w_i$ to $w_{i+1}$ of $\mathcal{D}$ with
      gradient of Equation 4.20
    • Take a policy step for $\theta_i$ to $\theta_{i+1}$ of $\mathcal{G}$, `using the
      TRPO rule with cost function $log(\mathcal{D}_{w_{i+1}}(s,a))$.
      Specifically, take a KL-constrained natural
      gradient step with Equation 4.19
end for
```

Generative Adversarial Algorithms are infamously difficult to optimize [PV16], especially due to the high variance gradient descent step of the Generator. Tackling this issue is critical for applications in large environments. The original GAIL research integrated TRPO [SLA+15] to optimize backpropagation of the Generator. As mentioned in Section 4.3.2, taking a KL-constrained natural step contributes to preventing the policy from changing too much due to noise in the policy gradient update. Another work [BACM17] approached the same issue by using a forward model to make the computation fully differentiable, which enables training policies using the exact gradient of the Discriminator. Doing so essentially connects the two networks, updating the Generator's parameters directly using the Discriminator's feedback.

The effectiveness of GAIL largely benefited from the unsupervised learning nature of GAN, which empowered the algorithm to take on various real-world learning problems [BAM16] [TZLB18] [PV16] [SRSE18] [FDSF18]. Previously, classical imitation learning algorithms often relied on precisely modeled environment dynamics and well crafted reward functions. In solving real-world problems with intricate dynamics and complex objectives, either criterion could be difficult to fulfill. In contrast, GAIL relaxed those dependencies with a data driven approach that implicitly learns both.

We have previously discussed robotic motion planning in Chapter 2, and learning based grasping in Chapter 3. In this chapter, our discussion had led to GAN and GAIL, which are the state-of-the-art IR manipulation trajectory planning algorithms. In the following chapters, we will piece the different components together, and propose a GAN-based, teachable robotic system which is capable of learning simple tasks from human demonstrations.

5
Method: Imitating Full Trajectories

In this chapter, we will present our GAN-based deep learning manipulation trajectory planner that imitates full trajectories. With GAN's data efficiency, our method can perform IL from human demonstrations. In addition, instead of taking the MDP approach to predict state-action pairs, the planner's Generator network directly predicts the entire sequence of states. We believe, by incorporating the long-term dependency between the poses, the planner can output trajectories with more coherent motion.

5.1 Method Overview

Generative adversarial algorithms are suitable to address this imitating learning problem, which uses real-world expert demonstrations. For example, when teaching a robot to pour a cup of water into a container, demonstrations might differ in the pace of motion, pouring angles, container positioning, and more. At the same time, all those demonstrations do share key features that define them as pouring, from a essentialism perspective. To learn pouring is to learn those features. However, the manipulation trajectories extracted from the demonstrations may not be directly comparable. Simple loss functions, such as mean square error, cannot correctly compare time-series trajectories with possibly misaligned starts, different paces of motion, or different pouring angles. The GAN's deep learning-based Discriminator essentially serves as a learned loss function approximation by distinguishing authentic expert demonstrations from the Generator's prediction. In doing so, it guides the Generator to produce predictions close to the demonstration distribution.

We designed the planner's neural network to output trajectories that can be executed to perform the tasks of interest. Those planned trajectories have to start with the gripper's initial pose, and adapt to the positions of any objects relevant to the task. For simplicity of discussion, we will refer to the initial gripper pose and object position together as *initial pose*.

Given the desired *initial pose*, we designed the GAN Generator to predict full manipulation trajectories, which are sequences of temporal contiguous gripper poses. Consequently, the GAN Discriminator is also designed to distinguish full trajectories. Compared to GAIL, which predicts one pose at a time, we believe that, by processing the entire sequence of gripper poses at once, both of the GAN's networks can perceive the long-term dependency context more easily. Specifically, exposing long sequences of poses to the Discriminator makes the unsmooth transitions between consecutive states more obvious to detect, which, in turn, forces the Generator to output trajectories with coherent motion. GAIL is also a promising approach, which we used in our experiments as a baseline.

To summarize, we designed the planner using a GAN-based full-trajectory prediction imitation learning model. Because our experiments use very few expert demonstrations, we expanded our training dataset with data augmentation by applying rigid transformations.

5.2 Architecture of the Networks

We built the planner with a GAN-based deep imitation learning model. The Generator $\mathcal{G}$ of the GAN model is designed to preserve long term dependency by predicting full trajectories at once. For timestamps, $t_0, t_1, ..., t_H$, which monotonically increase in equal intervals. Given $\vec{s_z}$ as the input noise to $\mathcal{G}$, the predicted trajectory $\tau_{\mathcal{G}}$ is given by

$$\tau_{\mathcal{G}} = \mathcal{G}(\vec{s_z}) = \{\vec{s_{t_0}}, \vec{s_{t_1}}, ..., \vec{s_{t_H}}\} \ , \tag{5.1}$$

where $\forall t,\ \vec{s_t}$ is the gripper's pose at time t. To provide information regarding the environment of the task, the desired *initial pose* s_g is used as the input noise s_z. Meanwhile, to ensure the predicted trajectory starts at the gripper's initial position, the first pose $\vec{s_{t_0}}$

is set to be the initial pose of the gripper. Using our pouring experiment[1] to illustrate our methodology, input tensor $\mathcal{G}$ is set to be the *initial pose* s_g, which is both the current pose of the gripper and the starting pose of the predicted trajectory, given by

$$\vec{s_g} = \vec{s_z} = \vec{s_{t_0}}. \tag{5.2}$$

Finally, the desired trajectory prediction $\tau_{\mathcal{G}}$ that starts with the gripper's pose can be written as

$$\tau_{\mathcal{G}} = \mathcal{G}(\vec{s_g}) = \{\vec{s_g}, \vec{s'_{t_1}}, \vec{s'_{t_2}}, ..., \vec{s'_{t_H}}\}\ . \tag{5.3}$$

where $\vec{s_g}$ is hard-wired to the output of $\mathcal{G}$, and $\forall t,\ \vec{s'_t}$ is inferenced by the deep neural network layers of $\mathcal{G}$.

Meanwhile, the Discriminator is trained to distinguish predicted trajectories $\tau_{\mathcal{G}}$ from expert demonstration trajectories τ_E, given by

$$d = \mathcal{D}(\tau)\ , \tag{5.4}$$

where d is the binary classification of $\mathcal{D}$, τ is trajectory input from either predictions $\tau_{\mathcal{G}}$ or expert demonstration τ_E.

Inspired by the design pattern of several popular networks, such as MobileNet, SqueezeNet, and ResNet, both $\mathcal{G}$ and $\mathcal{D}$ were built by repeatedly stacking subgraphs of similar structures. Each of those subgraphs contains multiple towers of stacked layers. The rest of this chapter illustrates the details of the model architecture and training techniques.

5.2.1 Generator Architecture

To insert control over the output of our GAN model, $\mathcal{G}$ receives *initial pose*

$$\vec{s_g} = [x, y, z, yaw, pitch, roll] \tag{5.5}$$

[1]As would be discussed in Section 6.1, this experiment learns the gesture of pouring, where the expert demonstrates a pouring motion from various initial poses towards various unmarked target positions. In the pick-and-place, *initial pose* contains both the gripper's initial position and the target placing position.

of shape $[1, 6]$ as input, and output a tensor of shape $[64, 6]$ that describes a full trajectory $\tau_{\mathcal{G}}$, given by

$$\tau_{\mathcal{G}} = \{\vec{s_0}, \vec{s_1}, ..., \vec{s_{63}}\} = \mathcal{G}(\vec{s_g}). \tag{5.6}$$

The input $\vec{s_g}$ is sent to a CNN-based deep learning graph, denoted by $\mathcal{P}$, which produces an intermediate tensor of shape $[63, 6]$. At the very last layer of $\mathcal{G}$, $\vec{s_g}$ and $\mathcal{P}(\vec{s_g})$ are concatenated together, forming a complete trajectory, $\tau_{\mathcal{G}}$, of shape $[64, 6]$, given by

$$\begin{aligned} \tau_{\mathcal{G}} &= \mathcal{G}(\vec{s_g}) \\ &= \{\vec{s_g}, \mathcal{P}(\vec{s_g})\} \\ &= \{\vec{s_g}, \vec{s'_1}, \vec{s'_2}, ..., \vec{s'_{63}}\} \ . \end{aligned} \tag{5.7}$$

Skip Edges

Due to the Generator network's complex structure and large parameter space, gradient estimation suffers from high variance [BACM17]. We constructed $\mathcal{P}$ using residual neural networks [HZRS16], utilizing short skip edges, colored green in Figure 5.1. If a subgraph in $\mathcal{P}$ covered by a short skip edge does not contribute to reducing loss, backpropagation may silence the signal produced by that subgraph, and the short edge can forward the original signal, essentially bypassing the subgraph. $\mathcal{P}$ is constructed by *1D transposed convolution* layers and *fully connected* layers, and *elementwise add* layers. The short edges are shown in Figure 5.1, colored green.

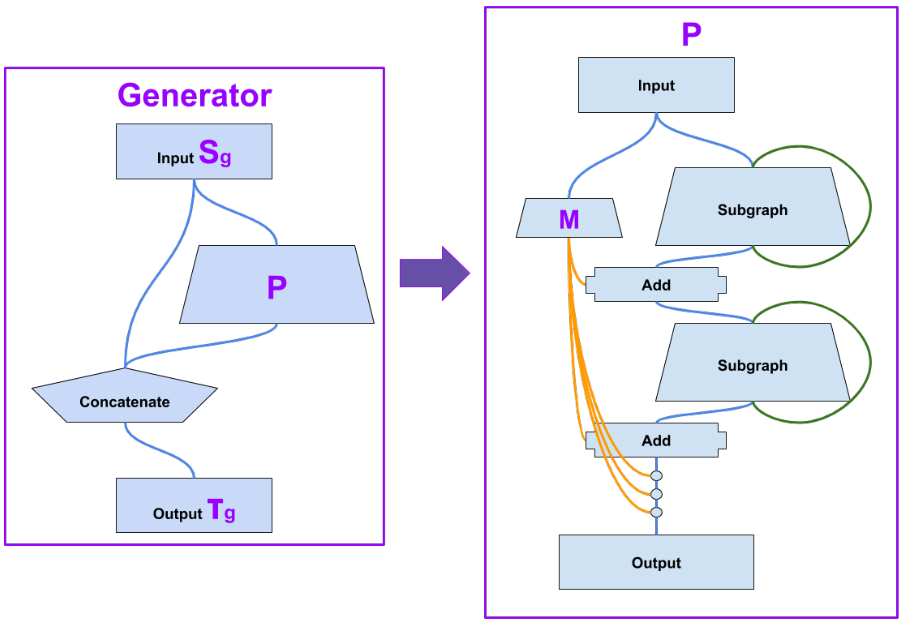


Figure 5.1: Architecture of the Generator with Skip Edges (the circles in $\mathcal{P}$ denote repeated subgraphs)

In addition, we introduce long skip edges to the graph. The input tensor was connected to a *multilayer perceptron* [SW89] $\mathcal{M}$. Its output, $\mathcal{M}(\vec{s_g})$, was then *elementwise added* with the output signal of each subgraph in $\mathcal{P}$. We believe the information provided by the long skip edges provides supplemental signals with low variance. The long edges are shown in Figure 5.1, colored orange. Experimental results regarding the effect of the long edges are presented in Section 7.3.5.

5.2.2 Discriminator Architecture

The Discriminator network $\mathcal{D}$ aims to distinguish authentic expert demonstrations τ_E from the Generator's prediction $\tau_{\mathcal{G}}$. It consumes input trajectory data τ of shape $[64, 6]$, and outputs a binary discrimination value d of shape $[1]$,

$$d = \mathcal{D}(\tau), \tag{5.8}$$

where τ is trajectory input from either predictions $\tau_{\mathcal{G}} = \mathcal{G}(s_g)$ or expert demonstrations τ_E.

At the very beginning of the Discriminator network, we handcrafted the network to append the first degree derivative of each of the original 6D poses to the channel dimension, i.e., forcing the network to compute the positional and angular velocity. The velocities are given by

$$\begin{aligned} \mathcal{V} &= \{\vec{v_1}, \vec{v_2}, ..., \vec{v_{63}}\} \\ &= \{(\vec{s_1} - \vec{s_0}), (\vec{s_2} - \vec{s_1}), ..., (\vec{s_{63}} - \vec{s_{62}})\}\ , \\ where\ &\{\vec{s_0}, \vec{s_1}, ..., \vec{s_{63}}\} = \tau\ . \end{aligned} \tag{5.9}$$

Then an intermediate tensor E with extended feature is produced by appending $\mathcal{V}$ to the input τ, given by

$$\begin{aligned} E &= \{\vec{s_0}, 0, \vec{s_1}, \vec{v_1}, \vec{s_2}, \vec{v_2}, ..., \vec{s_{63}}, \vec{v_{63}}\} \\ &where\ \{\vec{s_0}, \vec{s_1}, ..., \vec{s_{63}}\} = \tau\ , \\ &and\ \{\vec{v_1}, \vec{v_2}, ..., \vec{v_{63}}\} = \mathcal{V}\ . \end{aligned} \tag{5.10}$$

In practice, we found this simple feature engineering method can greatly improve the learning outcome, and reduce the complexity of the required network. $\mathcal{D}$ quickly learns that the demonstration trajectories have less extreme values in velocity.

Similar to the Generator's design, E is also fed to a CNN-based deep learning graph, denoted by $\mathcal{Q}$, constructed by *1D convolution* layers, *fully connected* layers, *pooling* layers, and *dropout* layers. The result of $\mathcal{Q}(E)$ is an intermediate scalar value, which is then regularized by a softmax activation function. The calculation for d is given by

$$d = \mathcal{D}(\tau) = softmax(\mathcal{Q}(E))\ . \tag{5.11}$$

Promote Network Generalization

As a classification network, the Discriminator $\mathcal{D}$ trades off between generalization and memorization. Because the expert demonstration dataset was sampled from a vastly large space, it is crucial to promote the generalization of $\mathcal{D}$. At the earlier stage of our research, we built a complex $\mathcal{D}$ network as deep and wide as $\mathcal{G}$. Pairing with the relatively small and less-diverse training dataset, the resulting models achieved extremely low training loss, but high evaluation loss. Those types of networks excelled in memorization, not generalization.

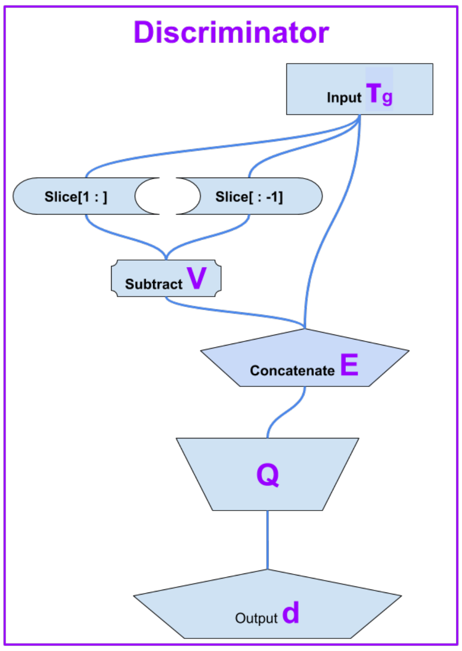


Figure 5.2: Architecture of the Discriminator

Essentially, $\mathcal{D}$ needs to learn the general characteristics of the desired trajectories, not memorize their specific values. In response, we especially promoted regularization of $\mathcal{D}$, including added *L1 kernel regularizers*, reduced the number of parameters in the network and inserted multiple *dropout* layers. All skip edges were also removed. The architecture design is shown in Figure 5.2.

5.3 Training

For each training iteration, we first created a minibatch of m noise $\mathcal{Z}$, which are randomly selected from the training dataset, denoted by

$$\begin{aligned} &\mathcal{Z} = \left\{z_1, ..., z_m\right\} \\ &where\ \forall z_i \in \mathcal{Z},\ z_i = \left(\vec{s_{t0}} | \vec{s_{t0}} \in \tau_{Ei}\right)\ , \end{aligned} \tag{5.12}$$

where τ_{Ei} is randomly sampled from the training dataset with replacement. Then $\mathcal{D}$ is updated by gradient computed with both τ_{E_i} and $\tau_{\mathcal{G}_i}$

$$\nabla_{\mathcal{D}} \frac{1}{m} \sum_{i=1}^{m} [log\mathcal{D}(\tau_{Ei}) + log(1 - \mathcal{D}(\tau_{\mathcal{G}_i})] \ , \tag{5.13}$$

where $\forall z_i \in \mathcal{Z}$, $\tau_{\mathcal{G}_i} = \mathcal{G}(z_i)$. Optionally, the update for $\mathcal{D}$ may be repeated multiple times with different samples of $\mathcal{Z}$.

Next, to update $\mathcal{G}$, we generate a minibatch of random noise $\mathcal{Z}'$, by

$$\mathcal{Z}' = \left\{z'_1, ..., z'_n\right\} \ , \tag{5.14}$$

where z follows a set of heuristics constraining the corresponding poses reachable by the robotic arm. We then update the Generator by taking a gradient computed by

$$\nabla_{\mathcal{G}} \frac{1}{n} \sum_{j=1}^{n} log(1 - \mathcal{D}(\mathcal{G}(z'_j))) \ , \tag{5.15}$$

where $z'_j \in \mathcal{Z}'$. Optionally, the update for $\mathcal{G}$ may also be repeated multiple times with different samples of $\mathcal{Z}'$.

By receiving the concatenated trajectory $\tau_{\mathcal{G}}$, $\mathcal{D}$ was designed to learn to penalize unsmooth transitions between the *initial pose* and the trajectory segment predicted by the neural network. In turn, this feedback would enforce the Generator $\mathcal{G}$ to output smooth trajectories that start at the gripper's *initial poses*.

We expected the accuracy of the Discriminator to start high and gradually decrease to 50%, at which point, we considered the model converged.

Loss Functions and Convergence

Accuracy metrics are a natural way to evaluate the performance of the binary classification outputs from $\mathcal{D}$. For simplicity, denote the evaluation accuracy of $\mathcal{D}$ as $ACC_{\mathcal{D}}$. At the beginning of GAN training, $ACC_{\mathcal{D}}$ is expected to be high, because the randomly initialized $\mathcal{G}$ would produce low quality $\tau_{\mathcal{G}}$. After multiple iterations, $ACC_{\mathcal{D}}$ is expected to oscillate around 0.5, which would indicate it is no longer able to distinguish expert demonstrations τ_E from predictions $\tau_{\mathcal{G}}$. If $ACC_{\mathcal{D}}$ remains high, it would indicate ineffective learning by $\mathcal{G}$. To summarize, effective training of GAN should have $ACC_{\mathcal{D}}$ that starts high and decreases to around 0.5, at which point, the model is considered converged.

Data Augmentation

Generative adversarial networks are infamously difficult to train, especially working with a small training dataset. In the case of GAN, training data for $\mathcal{D}$ contains both prediction $\tau_{\mathcal{G}}$, produced by $\mathcal{G}$, as well as ground truth trajectories τ_E from the demonstration dataset. While the supply of the former is limitless, the latter are in limited supply. Meanwhile, training would be more effective using a balanced and well-distributed dataset.

As will be discussed in Chapter 6, the number of demonstration trajectories in our experiments is extremely low. Due to the practical constraints of real robotic data collection, we expanded our demonstration dataset with synthetic variations of the original trajectories. This is done firstly with 2 random demonstrations as an evaluation dataset (*Eval*), while designating the remaining as training dataset (*Train*). It is important to not contaminate *Eval*. Then we expanded both the two datasets with rigid body transformations by adding randomly translated and rotated copies of the original trajectories from the same dataset. A synthetic trajectory $\tau_{E'}$ consists of

$$\begin{aligned} \tau_{E'} &= \{\vec{s'}_0, \vec{s'}_1, ..., \vec{s'}_{63}\} \ , \\ \forall t \ , \ \vec{s'}_t &= \mathbf{R}(\theta) \ (\vec{s}_t + \vec{v})^T \ , \end{aligned} \tag{5.16}$$

where $\forall t, \ \vec{s}_t \in \tau_E$, and $\mathbf{R}(\theta)$ is a random 3D rotation matrix capable of performing yaw rotation[2] of angle θ. Lastly, $\vec{v}$ is a random 3D translation vector. In practice, both θ and $\vec{v}$ are constrained with heuristic limits to prevent creating invalid trajectories outside of the robot's *workspace*.

In practice, our training dataset contains 10 demonstrations. For each of the demonstrations, we appended 999 additional variations using Equation 5.16. As a result, we intro-

[2]To be compatible with our learning scenario, the 3D rotation matrices are constrained to never introduce pitch and roll.

duced 9990 synthetic trajectories, expanding the training dataset into a size of 10000. We consider those synthetic trajectories as ground truth during training, not to be distinguished from the original expert demonstrations.

Avoid Training with Evaluation Dataset

One caveat of creating synthetic data is the possibility of accidentally creating training trajectories with *initial poses* equal to or very close to *initial poses* from *Eval*. Rigorous precautions were implemented to prevent training data from contaminating testing data. After the synthetic trajectories were created, we pruned any trajectory in *Train* with a starting pose too close to the starting pose of any trajectory in *Eval*, where the threshold distance is defined by the average distance of the starting poses across all original demonstration trajectories.

This concludes the discussion of our main contribution, the GAN-based IL manipulation trajectory planner that learns from a human expert. Therefore, the planner has to be examined in the real-world environment. In the next chapter, we will discuss in details of the experiments, including the experiment process, model parameters, baselines and the numerous software and hardware engineering challenges that we had to overcome.

6 Experimental Setup

To conduct experiments, the high-level process consists of the following 5 steps, summarized by the following table:

1. *Record Demonstrations* of simple tasks with an RGBD camera
2. Extract demonstration trajectories via *Computer Vision Pipeline*
3. *Train a Manipulation Planner* using Imitation Learning
4. *Plan and Execute Grasping* with the Dex-Net planner
5. *Plan and Execute Manipulation* motion with the GAN-based planner

Step 1 and 2 will be elaborated in Sections 6.4 and 6.5. Step 3 was discussed in Chapter 5, and supplemented in Section 6.2. For step 4, grasping, we adopted a state-of-the-art open-source grasping planner, Dex-Net 2.0 [MLN+17], and integrated the planner with ROS framework and arm controller provided by the open-source project kinova-ros [CLLL+19]. Dex-Net is denoted as DN. Given a depth image of the object of interest denoted by Φ, the suggested grasping configuration μ^* is given by

$$\mu^* = DN(\Phi) \ , \tag{6.1}$$

where DN is the grasping planner function implemented based on Equation 2.12. Subsequently, μ^* is executed by the arm controller interface to grasp the object. Denote the relevant states of the system by $\vec{s_g}$, which includes the initial pose of the gripper, and all other objects in the environment.

For step 5, after the gripper acquires the object, the second execution phase is to manipulate. Our custom trained deep neural network plans a manipulation trajectory τ using the Generator $\mathcal{G}$, given by

$$\tau_{\mathcal{G}} = \mathcal{G}(\vec{s_g}) \ . \tag{6.2}$$

The planned manipulation trajectory is then planned into joint angle space using motion planning software, given by

$$\{\Theta^*_{t_0}, \Theta^*_{t_1}, ..., \Theta^*_{t_H}\} = MP(\tau_{\mathcal{G}}, Env), \tag{6.3}$$

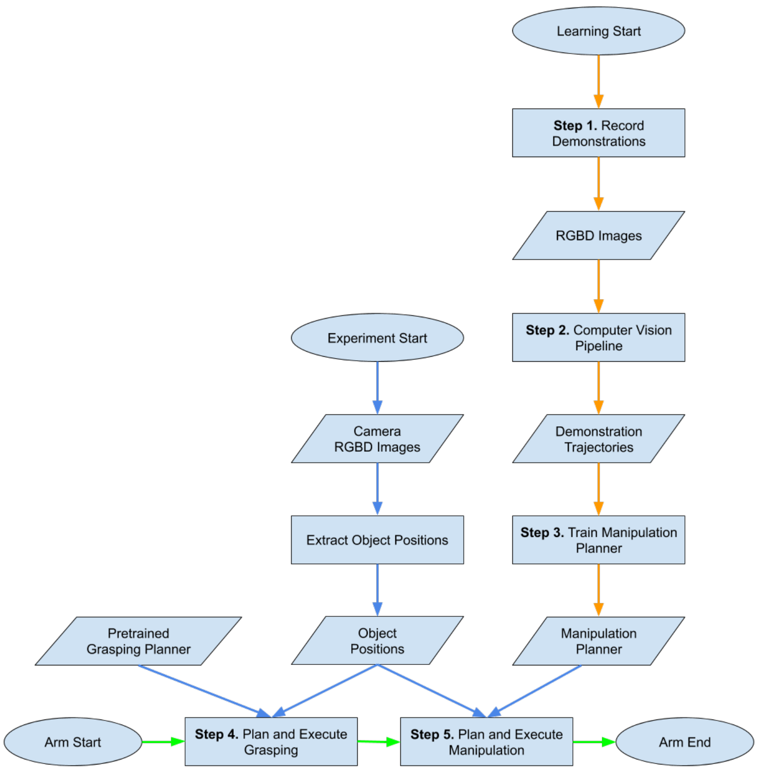
where MP is a motion planning function implemented by MOVEit [SMK12]. This sequence of Θ is finally executed by the robot. A flowchart of the experiment process is shown in Figure 6.1

In this chapter, we first illustrate details regarding our experiments: the tasks selected for learning, the specific neural network graph parameters used by our proposed GAN model, and the baseline models used to evaluate our contribution. In the rest of this chapter, we discuss the experiment process and the software engineering challenges, as well as our solutions. At last, we illustrate the trajectory visualization method that we used to present the results in the next chapter.

6.1 Tasks of Interest

Both the proposed method and the baselines were examined with experiments conducted on two tasks. The first task, pick-and-place, is to simply pick up an object and place it on a target location, where the initial pose of the arm, location of the object and the target location are all variables. This experiment is considered successful if the learning is able to grasp and place the object at the target location. The second task is pouring, where the initial pose of the arm and the demonstrated pouring location are both variables. The experiment is considered successful if there is a plausible pouring motion towards an arbitrary target location, followed by pose resetting. This simplified experiment can be considered as learning the gesture of pouring.

6.1 Tasks of Interest



Learning Start
Step 1. Record Demonstrations
RGBD Images
Step 2. Computer Vision Pipeline
Experiment Start
Camera RGBD Images
Demonstration Trajectories
Extract Object Positions
Step 3. Train Manipulation Planner
Pretrained Grasping Planner
Object Positions
Manipulation Planner
Arm Start
Step 4. Plan and Execute Grasping
Step 5. Plan and Execute Manipulation
Arm End

Figure 6.1: System Flowchart

For each task, we recorded 12 demonstration episodes, which were all made to follow a consistent behavior. Selected randomly, 10 episodes were used for training and 2 episodes were used for evaluation. Obvious and intentional distractions, such as pausing and ambiguous or turbulent motion, were avoided during demonstrations. Meanwhile, in order to introduce diversity, all demonstration episodes have different object positions and initial arm poses.

6.2 Neural Network Graphs

As the main contribution of our work, the proposed trajectory planner was implemented by a full-trajectory prediction deep learning model based on the original GAN paper [GPAM+14]. The network graphs follow the architecture design previously discussed in Chapter 5, and they are the results of many iterations of empirical results driven evolution.

Both the Discriminator $\mathcal{D}$ and Generator $\mathcal{G}$ networks were constructed with the popular strategy of repetitively stacking basic subgraph structure. In the simplified network diagrams shown in Figure 6.2, or full network diagrams in Appendix Figure A.1 and Figure A.2, $\mathcal{G}$ has 22,370,978 parameters, while $\mathcal{G}$ has 4,735,331 parameters. The two networks were constructed by repeating the subgraph pattern in the orange boxes for 9 and 6 times, respectively. Both networks were compiled using binary cross entropy as loss function [DC97] and Adam [KB14] as the optimizer. During initialization of the training process, all values in the trajectory datasets were normalized into the range between 0 and 1, and the parameters for both $\mathcal{D}$ and $\mathcal{G}$ were initialized with Gaussian noise.

Guided by empirical results, we gradually increased the number of parameters in $\mathcal{G}$ network. Notably, the long skipping edges introduced in Section 5.2.1 enabled us to largely increase the network's complexity without suffering heavily from exploding gradients. To upscale the 1D sequence, we used *transposed convolution*. Alternatively, we also experimented with nearest neighbors resizing[1], which significantly reduced the number of parameters, but performed poorly.

As discussed in Section 5.2.2, $\mathcal{G}$ network's complexity was deliberately constricted to

[1]Nearest neighbors resizing can be considered as a special case of *transposed convolution*, where all the kernel weights are constant values based on a fixed formula.

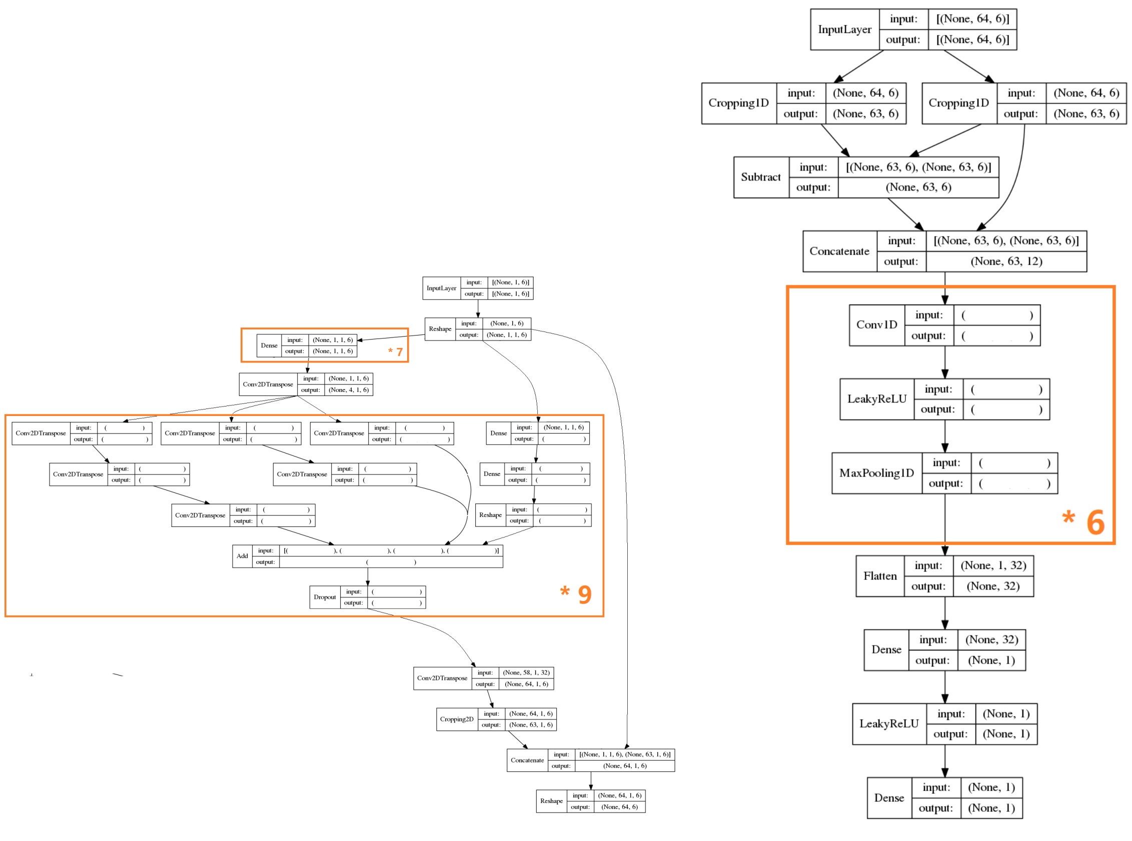


Figure 6.2: Network Architecture of the Generator (22M params) and Discriminator (4M params)

promote generalization. The initial input would pass through a stack of *1D convolutional* layers, *max pooling* layers, and *dropout* layers, which together act as an encoder, reducing the width of the input while increasing in depth. Finally, the feature vector was sent to a *sigmoid* activated *fully connected* layer, which quantifies the final verdict: whether or not the input trajectory is deemed authentic.

Also, our empirical results suggest that $\mathcal{G}$ performed better when the core layers are activated with *tanh*, while the Discriminator's core layers performed better with *leaky ReLU*. We believe this is likely because the former essentially serves as a regression network, while the latter acts as a classification network.

Depending on the model architecture and training configurations, training[2] time may range from hours to days. While some hyperparameters were explored in Section 7.5, many were not fine-tuned due to computation limitations.

6.3 Baseline Methods

We designed our experiments to examine the proposed method against reasonable baselines. To compare our approach with a classical solution of reinforcement learning, state-to-state Behavior Cloning (BC), we implemented 3 baseline models, which are based on state-to-state GAIL, state-to-state BC and full-trajectory BC.

We constructed the baseline state-to-state GAIL model with a Generator accepting 5 previous 6D poses as input seeds, and outputting 6 poses, which concatenate the 5 previous poses with the next pose prediction. $\mathcal{G}$ then accepts 6 poses and outputs the classification probability via a binary softmax function. The full model diagrams are shown in Figure A.3 and A.4 in the Appendix. For the BC models, we constructed networks with the same input and output shapes as the corresponding GAIL models[3]. Network diagrams are shown in Figure A.5 and A.6 in the Appendix.

To compare our method with the baselines, we defined a cost function, shown in Equation 6.4, which is the minimum of the mean square errors between the predicted trajectory

[2]Trained on a bare-metal PC with GeForce GTX 1080Ti GPU.

[3]The only exception is that the state-to-state BC model also outputs a termination signal, and trajectory inferencing was hardcoded to terminate at maximum length 64, preventing never-ending trajectories.

against all demonstration trajectories. Because the trajectories have different starting poses $\vec{s}_0$, we translated all trajectories to initiate from the origin by subtracting all poses in the sequence from the starting pose. The goal of learning is to minimize $\mathbb{J}$ between timestamps 0 to H. Although far from perfect, we will use this function to quantify the quality of predictions $\tau_{\mathcal{G}}$ against all demonstrations τ_E,

$$\mathbb{J} = \min(\sum_{t=1}^{t=H}((\vec{s}_{\mathcal{G}t} - \vec{s}_{\mathcal{G}0}) - (\vec{s}_{Et} - \vec{s}_{E0}))^2), \tag{6.4}$$

where $\forall t,\ \vec{s}_{\mathcal{G}t} \in \tau_{\mathcal{G}}$, and $\forall t,\ \vec{s}_{Et} \in \tau_E$.

6.4 Demonstration Process

Although the experiment was set up in a simplified laboratory scenario, we still encountered numerous engineering obstacles when building the prototype. In the rest of this chapter, we illustrate the challenges and our solutions.

Recent research work on imitation learning has had more success inside of simulations than in the real world. Notably, the OpenAI Gym [BCP+16] [Pal18] simulated environment using the MuJoCo [TET12] physics engine has been wildly used in research [CDFM18] [CG17] [PAR+18]. One significant reason is that deep learning algorithms usually require a large amount of data [FYZ+17], where massive amounts of ground truth demonstrations can be autonomously produced by other algorithms [AF17] [SVG+17].

Gathering a large training dataset is a different story in the real world. On one hand, getting one expert demonstration is easier; some cutting-edge imitation learning tasks are trivial for humans. For example, walking, pouring or making a sandwich. On the other hand, it is often infeasible to produce thousands, or even millions, of real-world demonstrations, which is not uncommon for researches conducted in simulation. The quality and quantity of demonstrations are immensely important factors for the effectiveness of imitation learning in general. Meanwhile, making demonstrations is tedious and tiresome, and people inevitably make mistakes, especially when performing long and repetitive work. While designing our pipeline, we kept in mind to reduce the effort and precision required by experts during demonstrations, and we promoted built-in error tolerance features in the system. In specific, we programmed software logic to address unintended pauses, turbulent motions, large chunks of missing motion segments due to various hardware limitations, and hindered or prematurely terminated recordings. Moreover, we simplified the laboratory scenario as discussed below.

Color-Coded Objects

To help ensure observations correctly reflect true states, all relevant objects are marked with color stickers, which can be reliably detected by the vision system. During both demonstration and execution, we deliberately removed all occupied colors in the scene, and ensured the color markers were visible by the camera at all times. Details regarding this design choice will be elaborated in Section 6.5.2.

Relying on Low-Level Drivers

The JACO2 commercial robot arm was equipped with sophisticated hardware sensors and joint controller drivers, supporting high accuracy positional and force feedback. Working in conjunction with the MOVEit [SMK12] joint motion planner, they are able to reliably bring the arm to the desired states.

Demonstrating with a Grabber

Instead of demonstrating directly by hand, we used a grabber to interact with the objects in our experiment. By doing so, we force the demonstrator's dexterity to match the JACO2 arm learner's. When demonstrating with real hands, subtle manipulating applied with our incredibly flexible fingers could be extremely difficult to identify, and potentially infeasible to imitate with the much simpler mechanical learner's arm. As shown in Figure 6.3, the grabber's gripper is attached to a rigid stick body, which has dexterity similar to the JACO2 arm's gripper. By demonstrating with a gripper, we brought the two agents into homogeneous action space, so the learner should be able to fulfill a task by executing the expert's trajectory. In other words, our learner's goal is to learn a policy with the same state-state pair in the expert's policy. As discussed later in Section 6.5.2, using a simplified end-effector also simplified the implementation of the color-based object tracking solution.

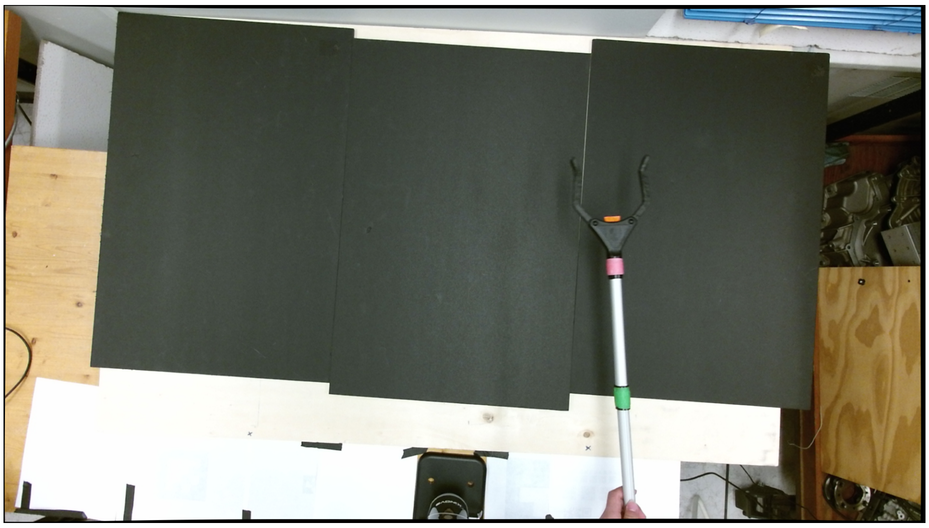


Figure 6.3: Demonstrating Using a Grabber

6.5 Computer Vision Pipeline

A computer vision pipeline was required to extract numerical trajectories from the raw camera feeds. In this section, we illustrate the engineering challenges and our solutions to this classical computer vision problem.

6.5.1 RGBD Camera

We deployed an RGBD sensor to capture the gripper motion and object position during both demonstration and execution. Our choice of hardware is the well supported and popular Kinect V2 sensor [But14] [FBR$^+$15].

The Kinect V2 camera captures color and depth with two separate sensors of different resolution and range. As shown in Figure 6.4, the color and depth cameras observe from different perspectives, which have per-device manufacturing-specific intrinsics [WS16]. As a result, each particular device requires software calibration to produce close-to-accurate point cloud data (PCD).

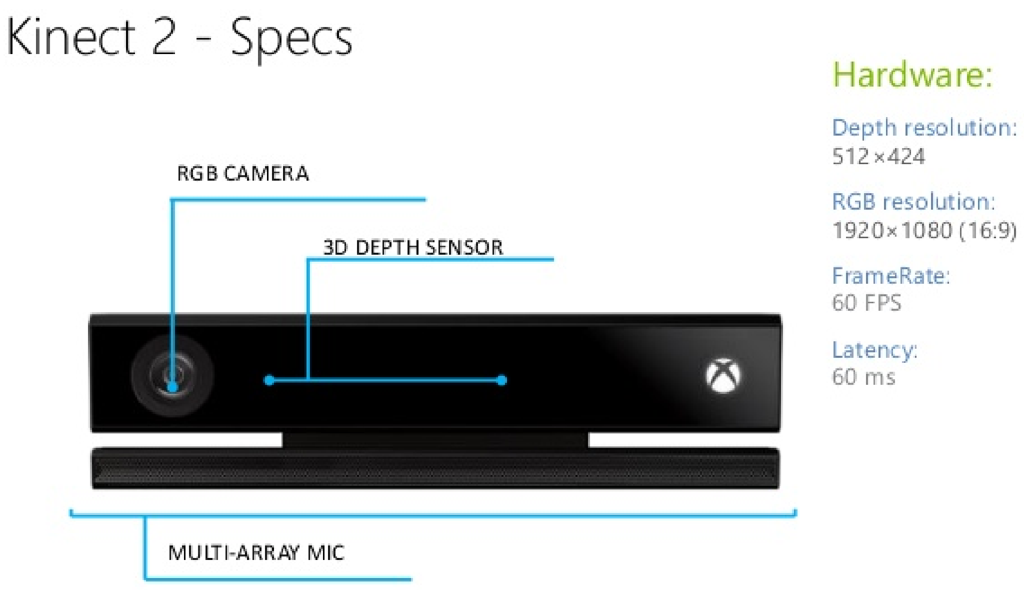


Figure 6.4: Kinect V2 Front Panel [WS16]

Calibration and Hardware Limitations

We employed a well-known open-source Kinect2 Calibration solution, IAI [Wie15]. It calculates the device-specific offsets and distortion intrinsics coefficients by analyzing camera feedback of known patterns. Shown in Figure 6.5, a pattern was printed on an A4 paper with precise ratio and pinged down to a ridged, flat board. Subsequently, we took snapshots of the pattern from various angles and distances, 300 for the color sensor and 300 for depth sensor. Finally, those snapshots were processed by the collaboration tool to calculate the estimated intrinsics of our particular device. However, even after multiple attempts, there are still significant errors in the estimated intrinsics matrices. By simple visual examination, notable mismatches between the PCD and reality were still pervasive. This causes issues later on during object tracking. Due to distortion, a misregistered color point could be mapped onto a neighbor point, which could have a significant difference in depth value.

The most common camera calibration error happens when the gripper or an object is moving through the air close to the depth camera, due to imperfect distortion estimation. In which case, the misregistered points may fall onto the next obstacle underneath, which is the table. To combat this issue, we implemented two strategies in the computer vision pipeline. We removed all points that are on or below the table surface; by doing so, we eliminated the majority of the misregistered points. Secondly, we added a sanity check mechanism using the relative distance between all the color markers attached to the gripper. Because those markers are tapped onto a ridged grabber, the relative distances between each marker are known. Therefore, when a heuristic tolerance threshold has been reached, we can deduce the readings are the culprit.

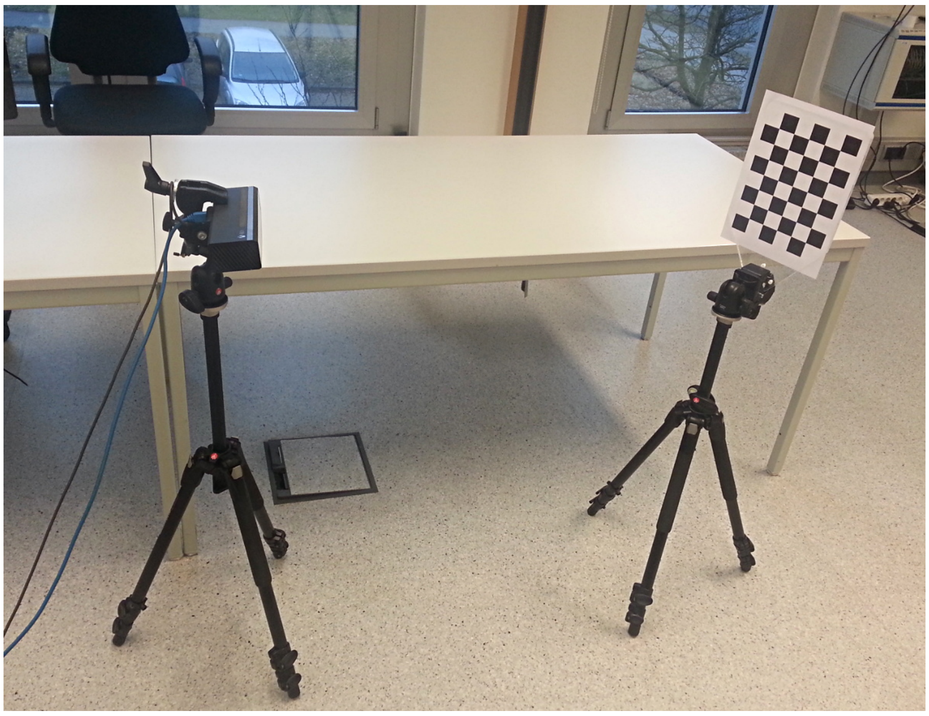


Figure 6.5: IAI Kinect2 calibration (photo reprinted from [Wie15])

Working Around Camera Minimal Distance

Another hardware limitation is the blind-spot in front of the depth camera. The sensor requires a minimal perception distance of 0.5 m. Meanwhile, due to the sensors' limited resolution, placing the camera too far from the demonstration scene would significantly affect the quality of PCD output. After all, the Kinect camera was designed for tracking body movements, not small objects. During demonstration recording, we had those limitations in mind and tried to carefully maneuver the grabber and objects to stay within the visible area of the workspace.

All the aforementioned solutions are a best-effort type of engineering approach. In practice, those heuristic-based sanity check mechanisms were developed with multiple iterations of trial and error.

Synchronizing Color and Depth Channels

The color camera sensor provides a relatively high frame rate of 30 Hz. However, the depth channel feed is not perfectly synchronized with the color channel feed, and, in practice, we often experience missing frames in the depth channel. Therefore, we sampled with a reduced frame rate of 2 Hz. For every 0.5 second interval, we picked the latest color image and depth image in the buffer as a matching pair, and then waited to pick the next new pair until the beginning of the next interval, while discarding the rest. Each pair is then computed into a PCD file using point registration. However, due to point registration and the hard disc writing speed limit, the final frame rate is only approximately 2 Hz, with large variance and frequent missing frames.

To summarize, the RGBD input was gathered as a pair of a color image, shown in Figure 6.6, and a depth image. Together they are computed into a PCD file via point registration, as visualized in Figure 6.7.

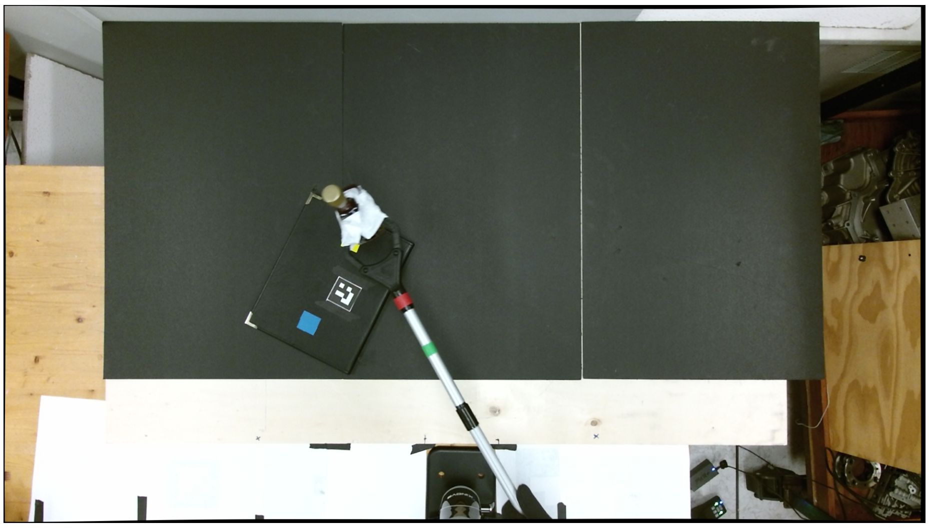


Figure 6.6: Camera Color Channel

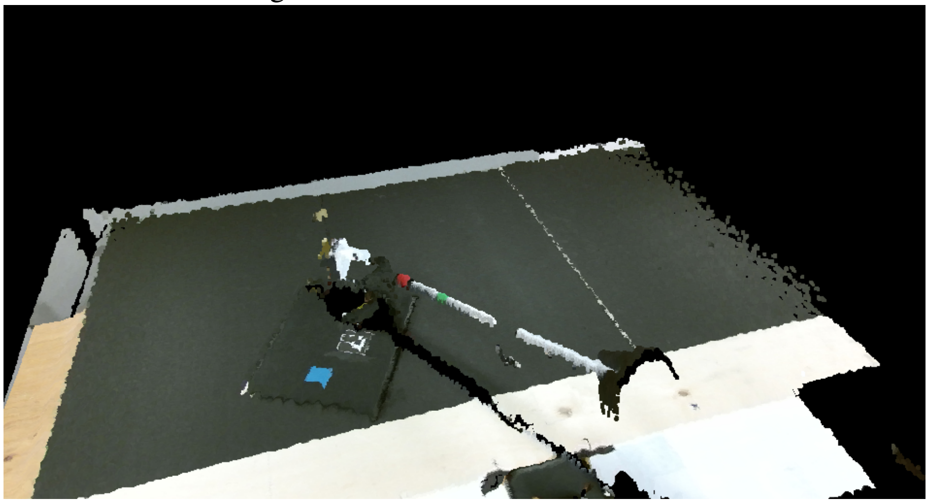


Figure 6.7: PCD Visualization

Alternative Methods for Perception

Utilizing visual signals for planning is a common approach to building a smart robotic system. Because humans rely heavily on vision to perceive the world, objects in our world are usually created and arranged to be visually distinguishable. This makes robotic vision sensors compatible with human-friendly environments.

Meanwhile, there are other options, such as miniature inertial sensors [RLS09], laser rangefinders [ZG00], or combining multiple types of sensors [MZB12] as the information source to perceive the surrounding environment. Those approaches usually work by attaching a sensor on the object of interest, which can produce precise and real-time positional feedback. Practical scenarios often involve robots operating within a precisely controlled and mapped factory environment. In other words, everything, including the robot, the objects and the environment, are all well-designed to facilitate the task. This line of ideas adepts very well in solving large scale industrial automation problems, where the goal is clear and the environment is well prepared and maintained, as there was little room for errors.

Our research addresses the polar opposite, where the environment is potentially unfamiliar, and the solution attempts to tackle a variety of tasks. To build a generalized operating system, we aim to have as few assumptions and modifications to the workspace as possible. Although we placed color markers on all the relevant objects as a proof-of-concept implementation, in a real-world application, the color tracker can be replaced by an object deception modulo, for example, utilizing the Tensorflow Object Detection API [Mus18].

Another advantage of a visual perception-based robotic system is the amount of potential demonstration data source. Although the theoretical foundation of machine learning has been laid decades ago, intelligent machine learning systems have historically not been viable for solving large-scale real-life problems. The relatively recent boom of machine learning progress is largely enabled by the recent boom of accessible data and computational power. Subsequently, imitation learning can also take advantage of the vast amount of easily accessible RGB videos on the internet, which can potentially be used as demonstration data for learning. With that motivation in mind, we decided to use visual feedback as the primary source of input.

6.5.2 Color Marker Annotated Objects

As object detection and pose estimation are not at the core of our research interest, our experimental scenario was designed to facilitate simple engineering solutions for extracting trajectory data from demonstrations. There are many well established technologies for annotating objects, for example, barcode, Bokode, QR code, Microsoft Tag and ArUco Markers. The ArUco Marker [dSCGF+15] was our initial choice for object annotation. It uses fixed-sized binary square fiducial markers to calibrate the marker's coordinate system with the camera's, capable of providing precise linear and angular real-time pose estimations.

While those sophisticated markers can provide rich information, one major issue of those barcode-like markers is that they require a full view of the marker without any significant parts obscured. This is not a reasonable assumption for our scenario, where the object and the grabber can overlap with each other from the camera's perspective, or rotate to a pose where the marker may no longer be clearly visible.

Instead, we used color-coded tags to annotate the relevant objects in the scene, as the blue tags shown in Figure 6.6. In addition, key parts of the grabber were also wrapped with colored tapes, so that the colored area would be visible from different angles. As previously shown in Figure 6.3, the pose of the gripper[4] can be calculated according to the coordinates of the green and pink color markers. An orange marker was attached to a part inside of the gripper, and it is only visible when the gripper is open. The markers enabled us to track the gripper's trajectory and its status.

6.5.3 Filtering and K-means Clustering

In practice, there were two main challenges we encountered when building an automated object and gripper tracker.

[4]Front end of the grabber, which grips objects.

Workspace and Lighting

Although the color tape is well-manufactured, with uniform color, and the workspace is intentionally organized to avoid clashing with the encoding colors, lighting can cause drastic differences in camera perception. The same strip of tape can present in a wide range of values when observed from different angles. Meanwhile, if we design the system to accept a large range of value for each color, significant noise can occur in unexpected ways. For example, the yellow-orange wooden table can cause pink glare when the image is taken from certain angles. Or, in other cases, the reflective metal surface of both the JACO2 arm and the gripper can display large blue point clouds, which turn out to be reflections of the blue Jalousie window in our lab.

Silent Errors

Incorrectly identified color markers can silently corrupt the gathered data. While the goal is to automate this tracking process for a large number of images, it is not ideal to manually verify results for every image. Therefore, the tracked data yield in this step is expected to be imprecise. This is a classic engineering problem to balance the cost of manual work and data accuracy. In our case, the deep imitation learning network does allow tolerance, and the quality of the tracked data only has to be reasonably accurate.

To address the aforementioned issues, firstly, we implemented a color range refiner tool with OpenCV, which allowed a trial-and-error type of manual tuning with Trackbar[5]. During our experiments, some 10 thousand color marker annotated images were captured and tracked. The color ranges were defined with hue, saturation and value (HSV), as opposed to RGB, due to its power to express concepts similar to how human defines a color: hue as the code of the color, saturation as the intensity of the color versus grey, and value as brightness. This step is crucial to the overall effectiveness of the color tracking solution.

Noisy color spots were inevitably included in the filtered PCD. In addition to color range filtering, we further refined the PCD by performing a statistical outlier removal process to eliminate smaller groups of noisy clouds. Both the filtering infrastructures were available as a part of the open-source Point Cloud Library (PCL) project.

[5]Trackbar is a popular OpenCV module for GUI numerical range selection

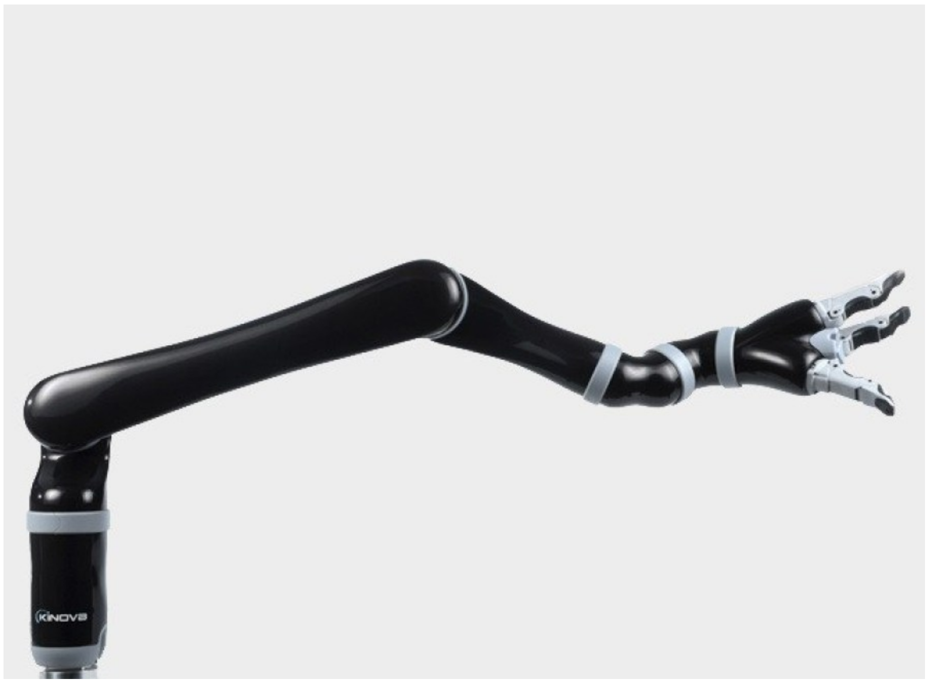


Figure 6.8: The Kinova JACO2®Assistive Robot

Even so, multiple clusters of colored points can present in the filtered scene, although often much smaller than the deliberately installed color taps. This makes simple K-mean clustering very effective [SLL11], where we simply pick the center of the largest cluster of color as the detected position of that color marker.

6.6 The JACO2 Assistive Robot Arm

The robot arm agent we employed is called the Kinova JACO2®Assistive Robot. Shown in Figure 6.8, it features six-axis movement that corresponding to shoulder, elbow and wrist, allowing us to mimic the smoothness and versatility of a fully functioning human arm. While the JACO2 arm was originally designed as a manually controlled assistive device for wheelchair users with upper extremity disabilities [MAFR11], our research explores its application to be interfaced with a machine learning task-level controller.

Our experiments were conducted in both simulations and the real world. The official kinova-ros driver project provided two vastly different low-level controller API for the two environments. To interface with them, we implemented two versions of controllers enabling smooth execution of high-level motion trajectory commands.

Controller for Gazebo Simulation

The Gazebo simulated environment was built upon ROS. The kinova-ros project also officially supports the MOVEit joint motion planning framework, which makes implementing a naive controller trivial. The controller simply needs to translate poses relative to the camera's reference frame into the arm's local reference frame, and then pass those parameters to MOVEit. The latter can be configured to plan joint-level instructions to complete the desired motion while avoiding collision with the arm itself or known objects, such as the table. Finally, kinova-ros provided joint controllers to change the arm model's states in ROS according to those instructions.

Initially, we built the system solely in Gazebo simulator. We implemented a high-level controller, which allows the demonstrator to joystick control the arm using a keyboard. After mastering controls, realistic demonstrations can be performed relatively easily on various 3D object models with the simulated JACO2 arm. However, the physics engine in ROS was not able to simulate contact between model surfaces. Most notably, the simulation could produce unexpected behavior or crashes when the gripper closes on rigid objects. Therefore, the simulation controller was also programmed to detect those abnormal states, to recover the robot to a normal state or otherwise restart the ROS world. In the pouring experiment, we only experimented with an empty gripper, instead of holding a cup as theatrical property.

In addition, the kinova-ros driver was not able to produce accurate motion. While the movement instructions produced by the kinova-ros joint controllers are described by position, velocity and acceleration, joint effort[6] is not explicitly modeled by the kinova-ros arm controller. In practice, if any instruction causes any part of the arm to go through another part of the arm, objects or other rigid surfaces, the offending model could be assigned with an unrealistic and undefined extreme velocity. For example, issues arise during grasping, where both sides of the gripper try to close on a solid object, or when the object gets pushed into the ground or a part of the arm. In both cases, the arm model or the object could be pro-

[6]Joint effort is a parameter to describe joint motions in ROS.

jected away with maximum speed. In a worse case scenario, eventually the joint controller would shut down, or the arm model could collapse into a non-recoverable state. Also, at the time, the kinova-ros driver implementation had several major known issues that often crashed the low-level joint controllers.

We eventually moved the experimental environment away from ROS Gazebo due to various limitations of the physical engine and the kinova-ros driver package.

Controller for the Physical Arm

The kinova-ros project also implemented controllers for the real arm. At the lowest level, each joint can be controlled by torque, in conjuncture with its per-joint position and force feedback. With those basic building blocks, kinova-ros also provided a variety of higher-level controllers. At the highest level, the end-effector's pose can be directly controlled in Cartesian space. For the interest of this research, we built a thin wrapper interfacing with this Cartesian pose controller, which takes a trajectory file and completes the described motion, including gripper's movements, closing, and opening. Similar functionality can be realized with per-joint velocity control, with the advantage of smoother motion and more control over the arm's pose.

Automated Reference Frames Calibration

When performing a task, the camera and the robot arm have to both calibrate into the same global coordinate reference frame. Throughout the span of many experiments, subtle displacement of the devices in the workspace often accumulate errors affecting the final test results. Subsequently, we implemented a robust and efficient solution to dynamically calibrate the different local coordinate systems against the global coordinate system.

As shown in Figure 1.2, the Kinect camera is installed on a tripod, which is strapped upside down from the ceiling, while the arm is fixed on the table. Both the devices have 6 degrees of freedom, 3D in spatial displacement, and 3D in orientation.

To effectively reference the two different local coordinate systems, we defined a global coordinate system using the user's perspective when sitting behind the table. Firstly, we established the global coordinate reference frame with a large Aruco marker with known geometry, which was fixed onto the table. The coordinate system of the arm can be visualized by putting a marker pen on the arm and drawing onto the table with the arm's controller interface. The broad base of the arm's foundation kept the Z-axis of the arm parallel to the global Z-axis. For simplicity, we rotated the arm until both the X and Y axis were also aligned with global. Once satisfied with the arm's pose, we fixed its foundation to the table with two heavy-duty mounting clamps across the base. This reliably prevents any significant displacements between the arm and the table. At last, a coefficient matrix was calculated to translate the arm's reference frame to the global frame.

Meanwhile, a small rotation of the camera's coordinate origin can significantly offset the coordinates of objects far away. To autonomously translate coordinates between the two systems, we implemented a calibration software solution using OpenCV Python package. At the beginning of each experiment, we took a snapshot image of the global Aruco marker from the camera's perspective. Using the pose estimation library provided by the Aruco OpenCV module, we could drive the 6 coefficients required to translate camera frame to global frame. Both sets of coefficients were written together to a yaml file, which gets loaded by the tracking and controller component during runtime.

Implemented in C++, the complete computer vision pipeline fetches inputs from the Kinect camera's image feed buffer, and outputs a series of tracked poses for the gripper and linear positions for the objects.

6.7 Trajectory Visualization

This section illustrates our trajectory visualization method as preparation to interpret the experimental results in the next chapter. To begin with, consider a trajectory τ_P extracted by the pipeline. An example of the snapshots taken during the pouring demonstration is shown in Figure 6.9.

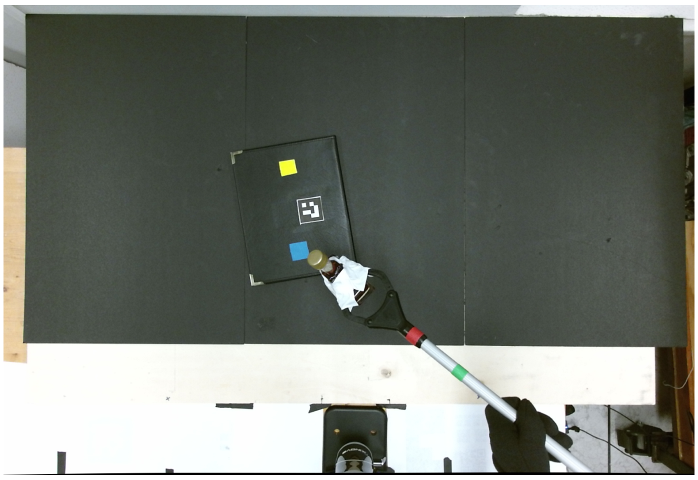


Figure 6.9: A Snapshot of Demonstration Recording

We first parse the demonstration recordings into features composed of a 6D-point fixed-length time-series of poses, which consist of 3D linear positions and 3D angular positions[7]. A trajectory τ can be represented by a 1D time series from t_0 to t_H,

$$\tau_P = \{\vec{s}_{t_0}, \vec{s}_{t_1}, ..., \vec{s}_{t_H}\}, \tag{6.5}$$

where $\vec{s}$ is a pose vector of $[x, y, z, yaw, pitch, roll]$, and $t_0, t_1, ..., t_H$ are variable timestamps which increase monotonically and have potentially uneven intervals. The arm's pose can be visualized by Figure 6.10, where the arrows in the figure represent the poses of the arm.

[7]Due to the limitation of our simple tracking system, *roll* was not possible to be detected, thus resulting in all *roll* values to be 0. However, we decided to overlook this application specific property of the training data in order to design a generic learning network.

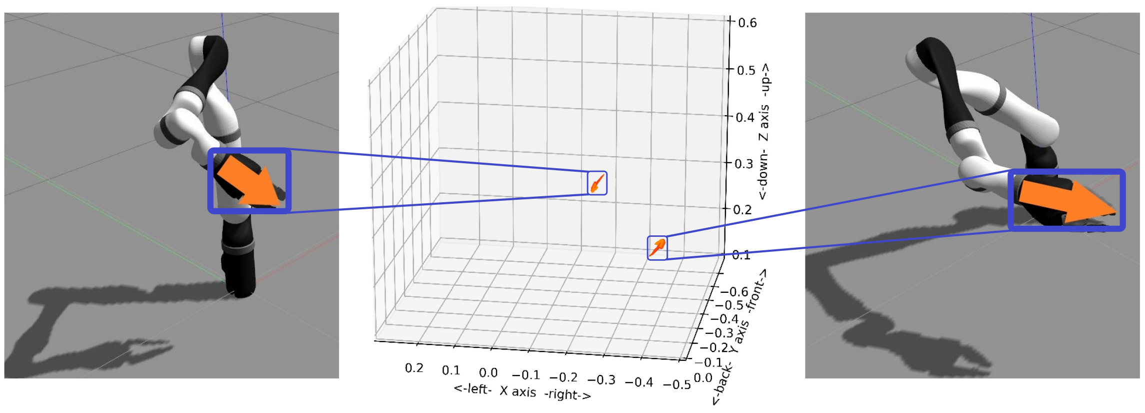


Figure 6.10: Visualizing the Arm's Pose in a 3D Plot

The full trajectories for pouring can then be visualized by examples in Figure 6.11. The plotted Euclidean space represents the real-world 3D space, with all axes using meters as units. Each arrow corresponds to the end-effector's poses sampled at a fixed frequency, while the color reflects time elapsed since the beginning of the trajectory. The initial pose was marked by the red arrow, and the subsequent poses were marked by arrows gradually fading into yellow.

For the rest of this thesis, we will use red to yellow for demonstration trajectories, and blue to cyan for predicted trajectories. As the elapsed time increases, the color of the arrows gradually fades from red into yellow or blue into cyan.

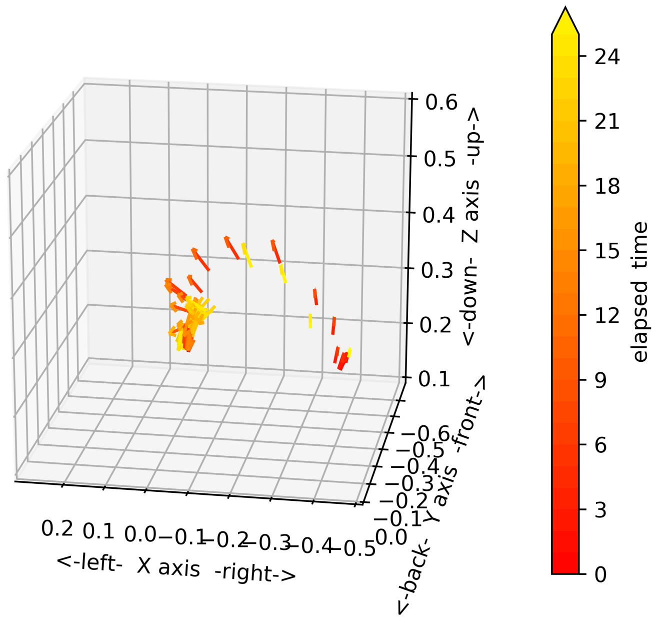


Figure 6.11: Original Demonstration Trajectory for Pouring

The origin of the coordinate system is collaborated to align with the arm's coordinate system. For better visualization, Figure 6.12 shows the same demonstration trajectory from multiple viewing angles.

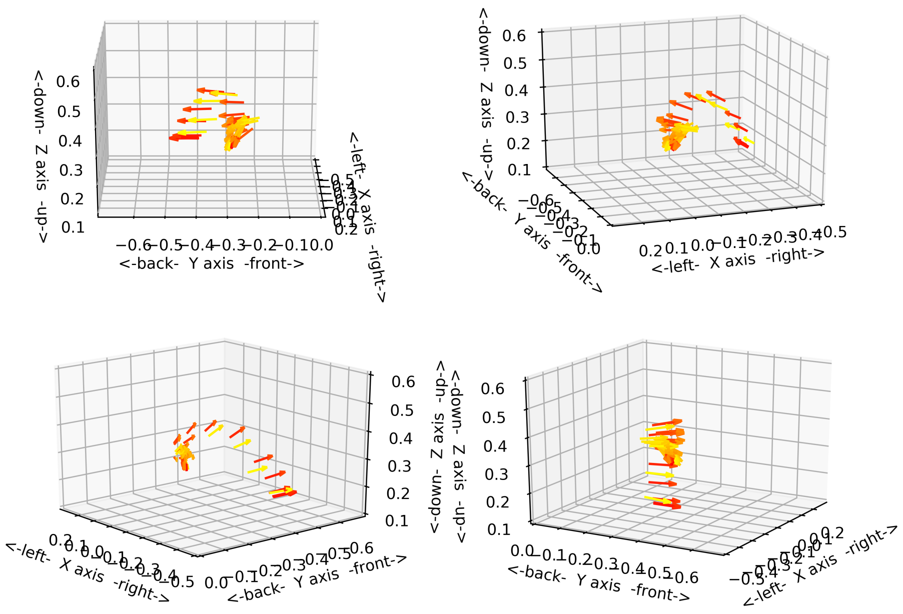


Figure 6.12: Trajectory Visualization from Multiple Perspectives

Due to hardware limitations during the recording and tracking process, some frames could be lost from the collected trajectories. Moreover, demonstration point cloud data were recorded at approximately 2 Hz, while duration varies from around 20 s to 30 s depending on the positioning of the objects. We first mirror padded a random number of points on both ends, and adjusted the original timestamps to ensure all episodes of demonstrations had a duration of 31.5 s.

Next, we re-sampled 64 points from the original trajectory at precisely 2 Hz, which conveniently makes the feature vectors have a fixed shape of $[64, 6]$, resulting in a re-sampled trajectory τ_E, given by

$$\tau_E = \{\vec{s'}_0, \vec{s'}_{0.5}, ..., \vec{s'}_{31.5}\}, \tag{6.6}$$

where $\vec{s_t}$ denotes the pose of the end-effector at timestamp t. Due to hardware limitations, which would be discussed in 6.5.1, evenly sampled points were calculated by performing linear interpolation on the original trajectory. For a given timestamp t, the re-sampled pose is calculated by

$$\vec{s'_t} = \frac{\vec{s_{t_k}} * (t - t_k) + \vec{s_{t_{k+1}}} * (t_{k+1} - t)}{t_{k+1} - t_k}, \tag{6.7}$$

where $t_k <= t$ and $t_{k+1} >= t$ such that $\vec{s_{t_k}}$ and $\vec{s_{t_{k+1}}}$ are valid points in τ_P. Consequentially, τ_E would have a fixed length of 64, while the ordering of each point implicitly indicates its elapsed time during the demonstration.

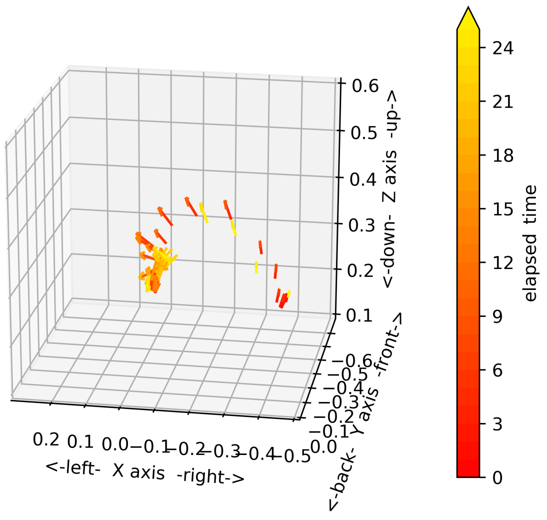


Figure 6.13: Original Demonstration Trajectory for Pouring

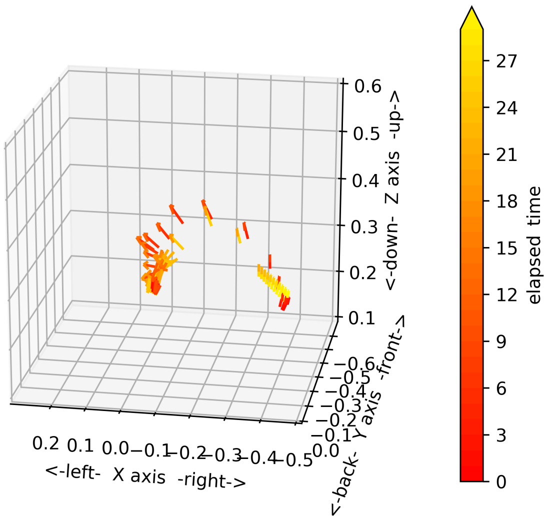


Figure 6.14: Re-sampled Demonstration Trajectory for Pouring

While Figure 6.11 shows one of the original demonstrations τ_P, Figures 6.13 and 6.14 compare the original and re-sampled trajectories of τ_E.

In the next chapter, we will illustrate the results of our experiments.

7 Results

Our experiments can be divided into two sets. The first set was conducted in the real world, aimed at examining the overall viability of the integrated system as a whole. The second set was conducted in simulation, aimed at investigating the performance of the proposed GAN-based manipulation trajectory planner.

7.1 Viability of the System

To examine the viability of the teachable robotics system, the goal of our first experiment was to evaluate our prototype system on learning one of the simplest tasks, pick-and-place.

Even the baseline models were able to consistently suggest viable pick-and-place manipulation trajectories. Figure 7.1 visualized one example of an expert demonstration, while Figure 7.2 illustrated one of the trajectories predicted by state-to-full-trajectory GAN.

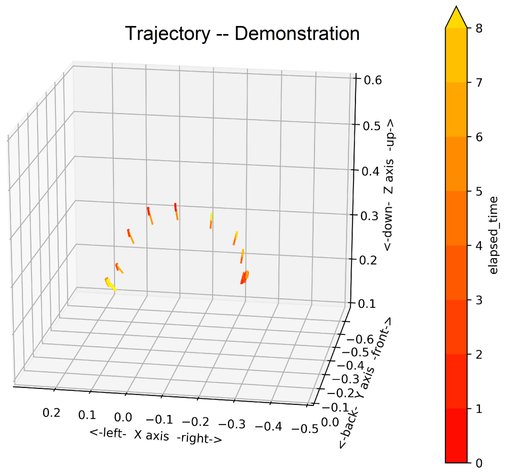


Figure 7.1: Demonstration Trajectory

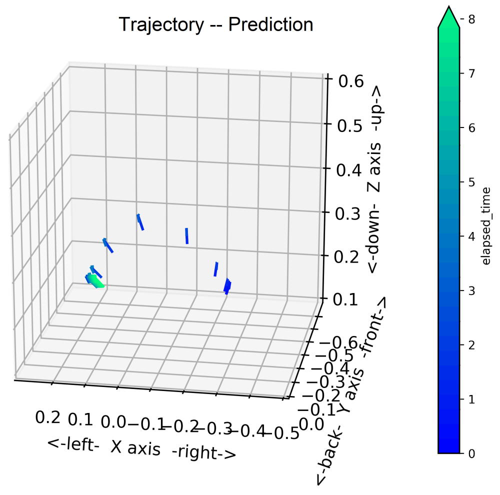


Figure 7.2: Predicted Trajectory

Through collaboration between the object tracking infrastructure, the adapted Dex-Net grasping planner, and the learned manipulation trajectory planner and the JACO2 arm, the system was able to consistently perform pick-and-place on scenarios with arbitrary starting and target positions.

A short example video demonstration is available at youtu.be/ywvo9tqk1q0. In the example, a color marked object was placed at an arbitrary location, the system locates the object using the camera, and finally performs pick-and-place using the grasping planner and the manipulation planner. Figure 7.3 animates one example of the experiment, where each snapshot was evenly sampled from the original video.

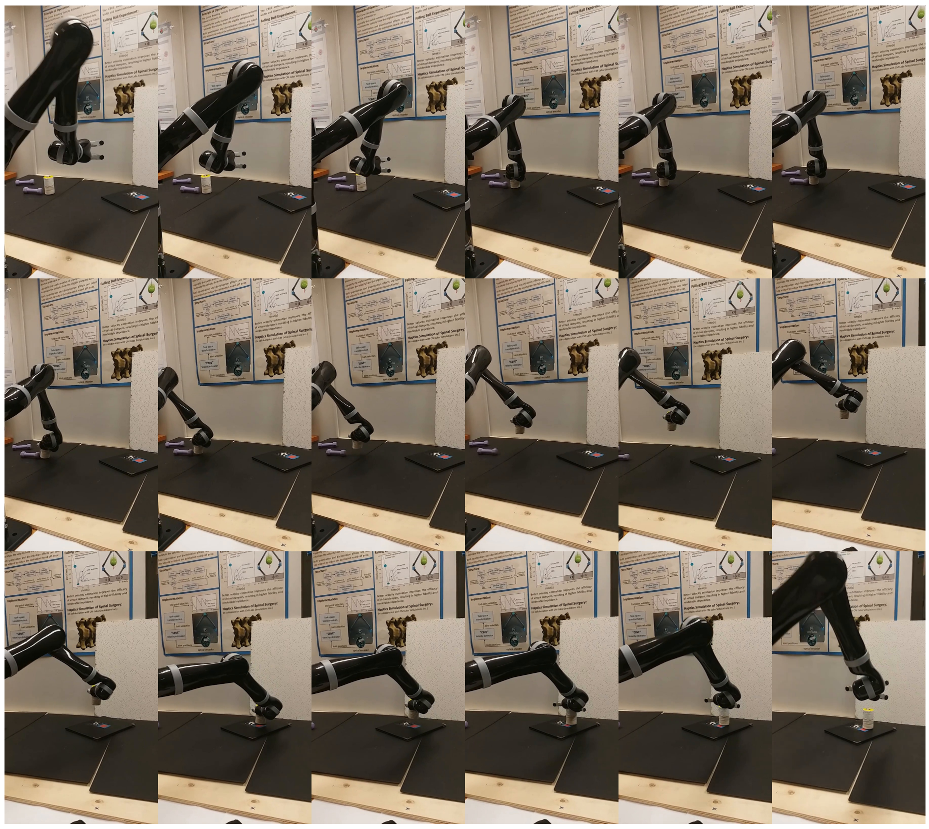


Figure 7.3: Another Example Executing the Learned Pick-and-Place Trajectory (Video Available)

7.2 Qualitative Comparison

In the rest of this chapter, we will compare the manipulation planners implemented by our proposed method against the baselines. While all four approaches succeeded at pick-and-place, only the proposed full-trajectory GAN-based model was able to reliably suggest a viable trajectory to complete pouring.

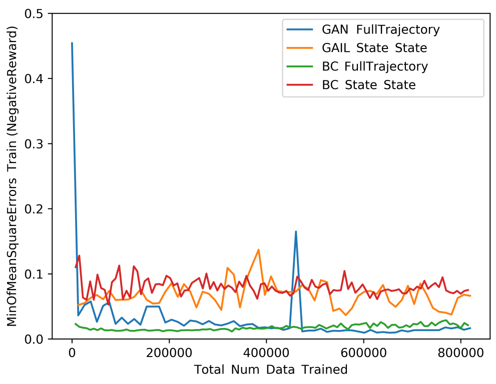
To quantifiably compare the four methods, Figures 7.4 and 7.5 visually illustrate the goodness values for each method, which was defined by Equation 6.4. To avoid terminology clashing with "cost" or "loss", which were also used during training, we will refer to this goodness value as Negative Rewards. Negative Rewards were plotted on the Y-axis, against the total amount of data trained plotted on the X-axis. The following two paragraphs clarify details regarding the figures.

Rewards were quantified on two sets of data, named *Train* and *Eval*. Our extremely small demonstration dataset only consists of 12 trajectories, which are divided into 10 for the training dataset (*Train*) and 2 for the evaluation dataset (*Eval*). Both datasets were expanded with synthetic trajectories by the procedure illustrated in Section 5.3. The expanded *Train* dataset was then used for model fitting, where 10% was designated for validation. We separately evaluated the models with seeds extracted from both *Train* and *Eval*.

Although trained with different algorithms, all four methods were plotted on the same x-axis. As a generative adversarial network, the generative adversarial models were trained with units of iterations, where each iteration uses a minibatch of seeds[1] randomly drawn from the training data, sampled with replacement. On the other hand, the BC models were trained with epochs of the entire training dataset. To provide a reasonable comparison, the GAN-based methods' total amount of data trained is calculated to be the product of iterations multiplied by the size of minibatch. For the BC-based methods, it is calculated to be the product of training data size, multiplied by epochs. Overall, the proposed method, plotted in blue, delivered significantly better performance.

For GAN, each minibatch was randomly sampled from all demonstrations in the training dataset; those demonstrations are unique, although including synthetic trajectories. Similarly for BC, each epoch was trained on the whole training dataset with every demonstration unique.

[1]The seed to the Full-Trajectory Prediction GAN is the initial pose of the gripper, while the seed to the State-to-State Prediction GAIL is the last 5 poses of the gripper.



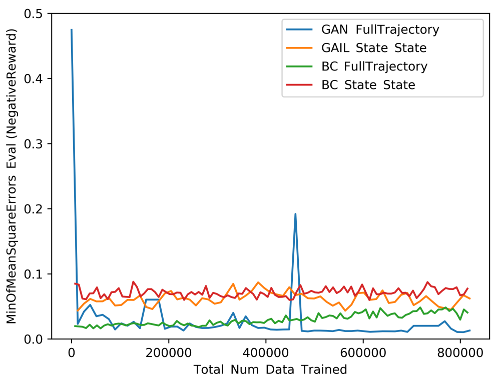


Figure 7.4: Negative Rewards from *Train*

Figure 7.5: Negative Rewards from *Eval*

Table 7.1 summarizes the quantitative results using average Negative Rewards measured from models trained with 650,000 and 750,000 trajectories. That particular range was chosen based on the respective learning curves, as will be discussed in Section 7.3. As the results show, state-to-full-trajectory prediction models performed significantly better than state-to-state prediction models, and GAN-based approaches performed better than BC-based approaches.

Method	Evaluated by *Train* Seeds	Evaluated by *Eval* Seeds
State-to-Full-Trajectory GAN	0.011	0.012
State-to-Full-Trajectory BC	0.021	0.038
State-to-State GAIL	0.068	0.065
State-to-State BC	0.072	0.070

Table 7.1: Comparison of Averaged Negative Rewards

7.3 Learning Curves and Progression of Predictions

We begin discussing the qualitative results by presenting learning curves.

During training, we also periodically evaluated the models with seeds from the *Eval* dataset, which were carefully prepared to avoid in training, as discussed in Section 5.3. Those evaluation trajectories are also visualized to show how predicted trajectories progress during training.

7.3.1 Full-Trajectory GAN

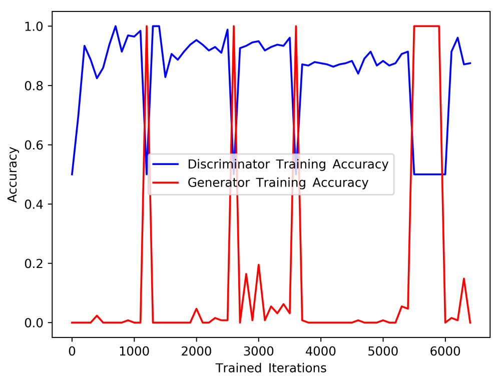


Figure 7.6: *Train* Accuracy of State-to-Full-Trajectory GAN

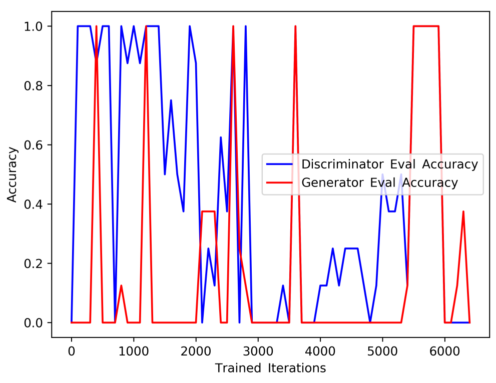


Figure 7.7: *Eval* Accuracy of State-to-Full-Trajectory GAN

The learning curves for state-to-full-trajectory GAN-based model agrees with the expectation of model convergence established in Section 5.3. This suggests the training was effective. The metrics in Figure 7.6 were measured against the *Train* dataset after each iteration. The Discriminator's post-fitting accuracy generally remained above 0.5 after each backpropagation, which means it was able to consistently distinguish the counterfeit trajectories. Although the Generator's post-fitting accuracy remained low, which reflects its hardship in learning to deceive its counterpart, there are interesting spikes where it learns to successfully generate plausible trajectories in the Discriminator's perspective. When the spikes occurred, the Discriminator's accuracy dropped to around 0.5 for a short period, during which it adapted and learned to distinguish the new characteristics of the predicted trajectories again.

The metrics in Figure 7.7 were measured against the *Eval* dataset after each iteration.

Near iteration $5,500$, both networks' accuracy overlapped at 1. This is because they are calculated from different inputs. Due to the particular nature of GANs, the Generator's output does not have ground truth labels. We measured its accuracy by feeding *Eval* seeds through the combined GAN. Therefore the correct prediction should all be 0s. The Discriminator's accuracy was measured by feeding *Eval* data to the standalone Discriminator network. Therefore, the correct prediction should all be 1s.

After iteration $3,000$, the *Eval* accuracy of both networks notably decreased to around 0.5[2]. This is the desired behavior indicating convergence of GAN training, where the Discriminator is no longer able to distinguish the Generator's predictions and the ground truth demonstrations.

[2]The learning curves, model performance largely vary between different experiments. In some experiments the *Eval* accuracies of the Discriminator were shown to consistently hover around 0.5; however, to avoid selection bias of machine learning, we did not deliberately pick the experiments with ideal learning curves.

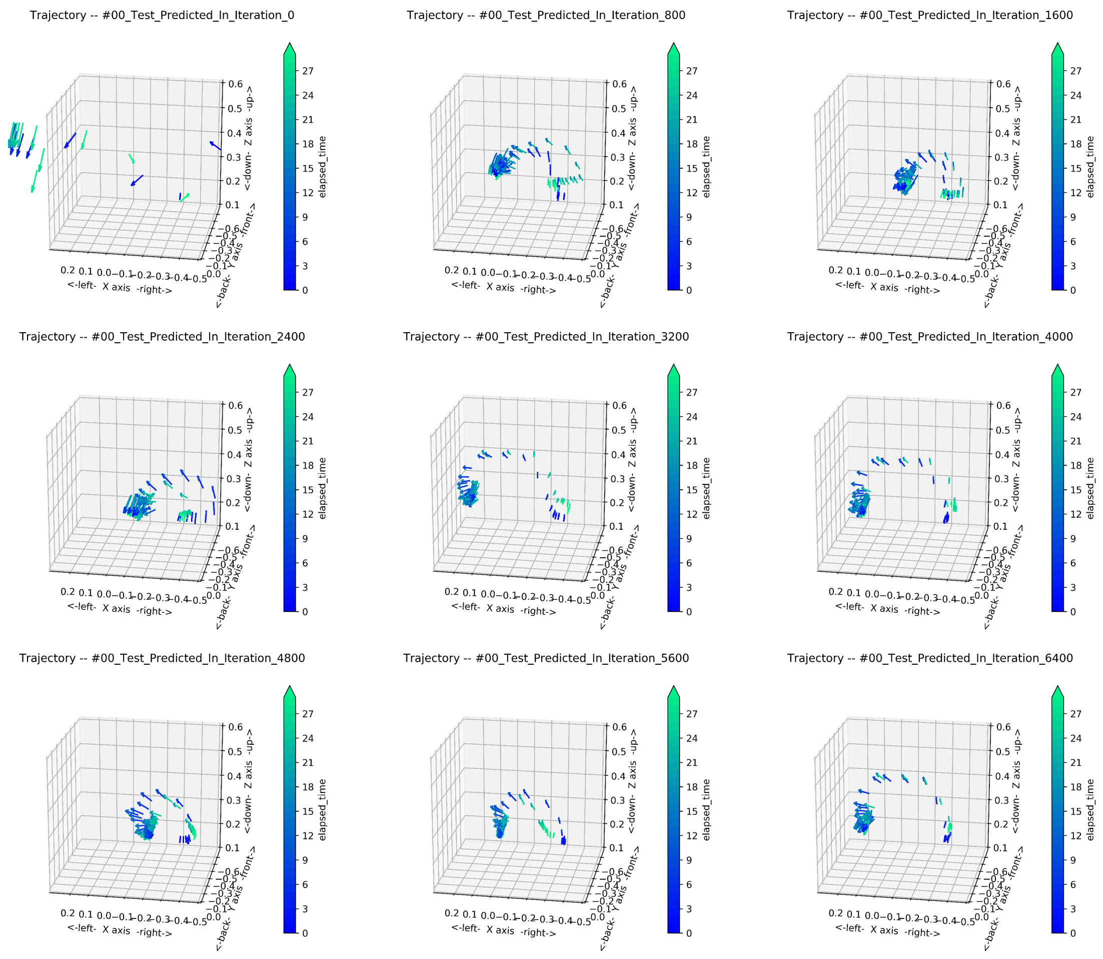


Figure 7.8: Predictions From State-to-Full-Trajectory GAN (EvalSeed #0)

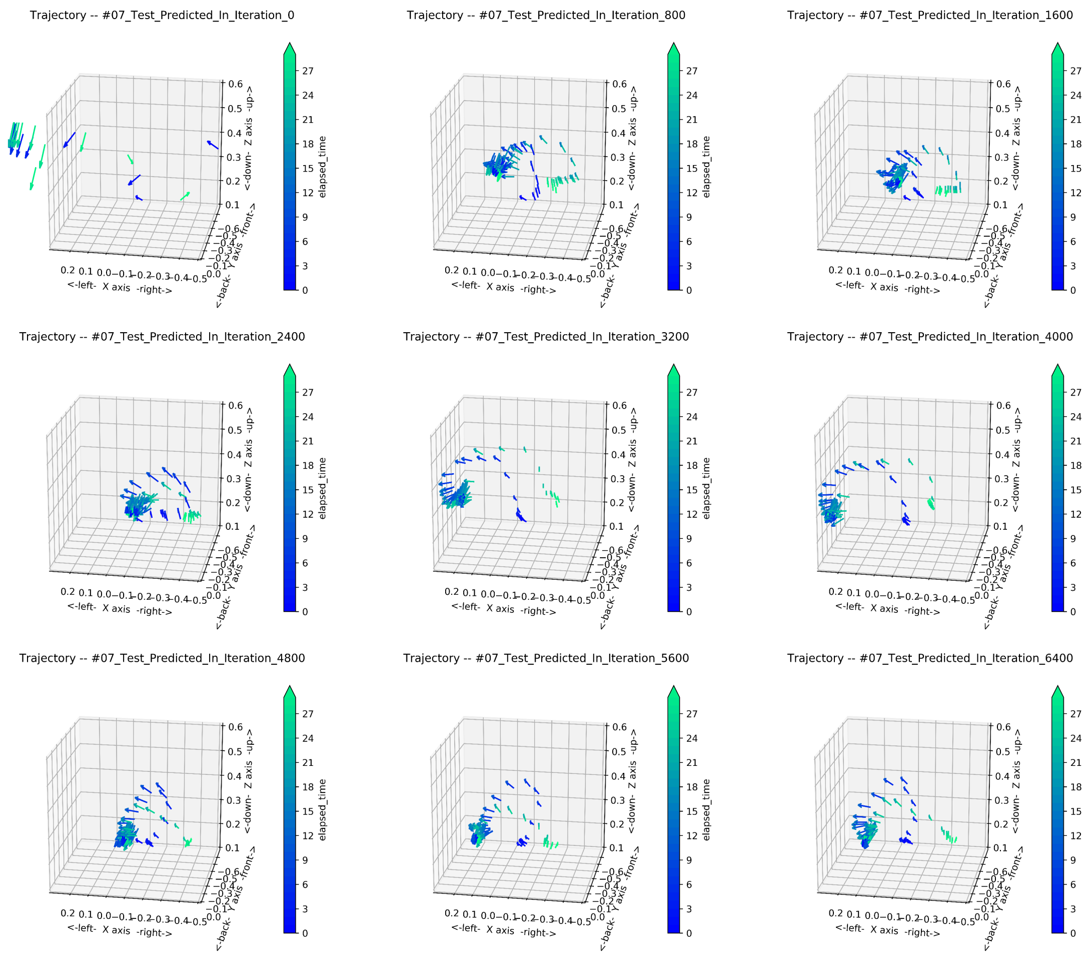


Figure 7.9: Predictions From State-to-Full-Trajectory GAN (EvalSeed #7)

Figures 7.8 and 7.9 shows how prediction improved as training took place, with two examples with different seed input (EvalSeed #0 and #7).

7.3.2 State-to-State GAIL

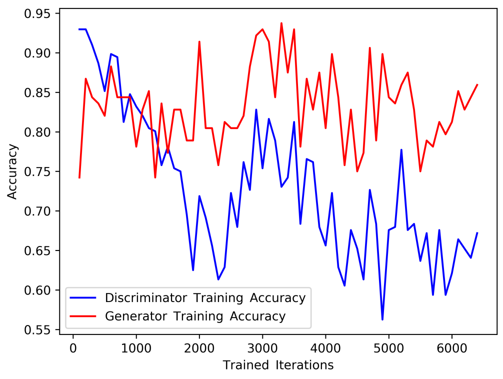


Figure 7.10: *Train* Accuracy of State-to-State GAIL

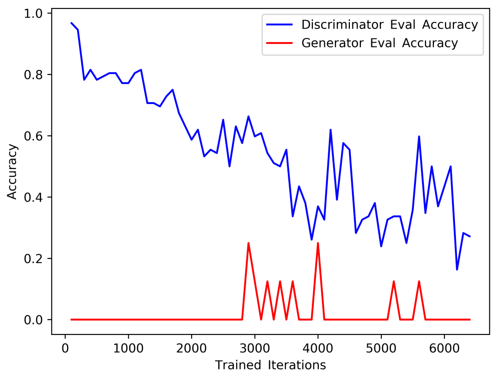


Figure 7.11: *Eval* Accuracy of State-to-State GAIL

In Figures 7.10 and 7.11, the learning curves for state-to-state GAIL-based model suggest that the training was also effective. Comparing to state-to-full-trajectory GAN, the curves had fewer sharp changes, as the networks had much fewer parameters and easier to train. Signaled by the *Eval* accuracy of the Discriminator, the model converged around iteration $4,000$.

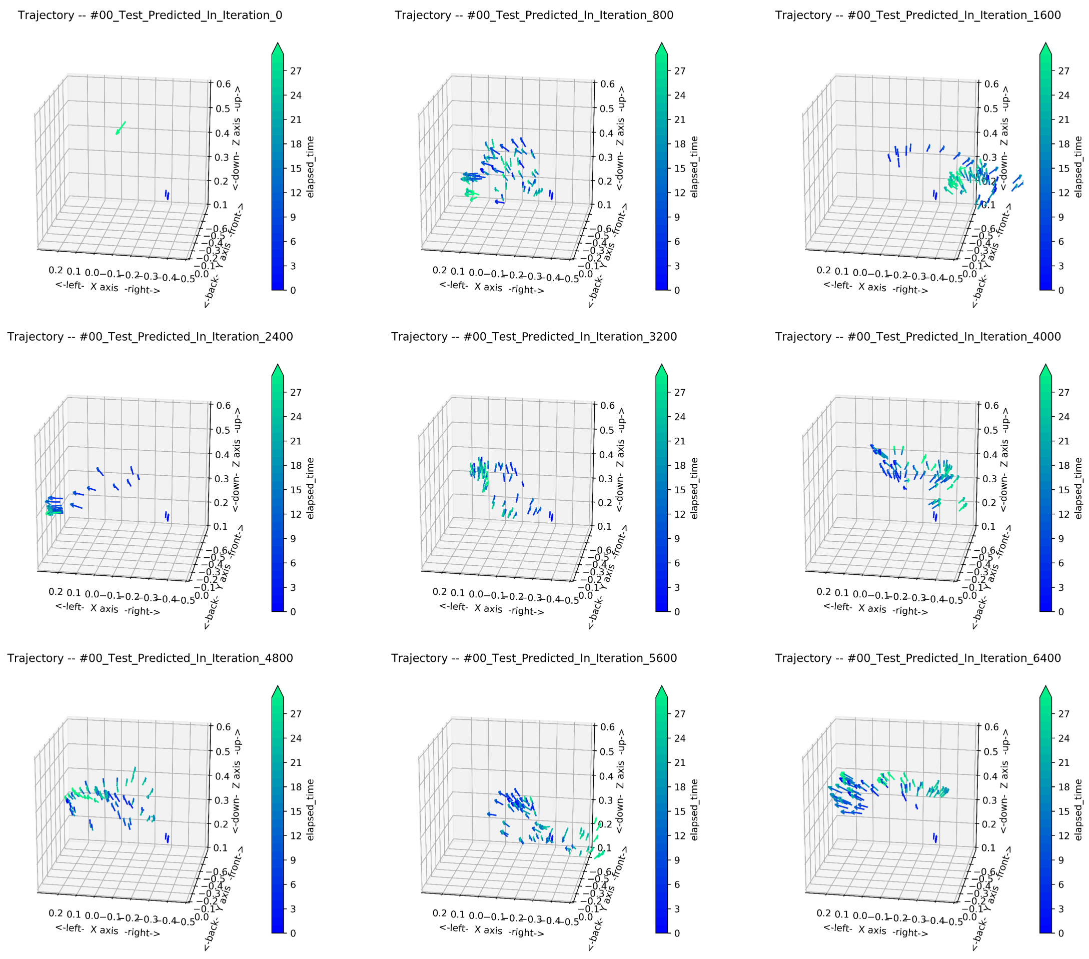


Figure 7.12: Predictions From State-to-State GAIL (EvalSeed #0)

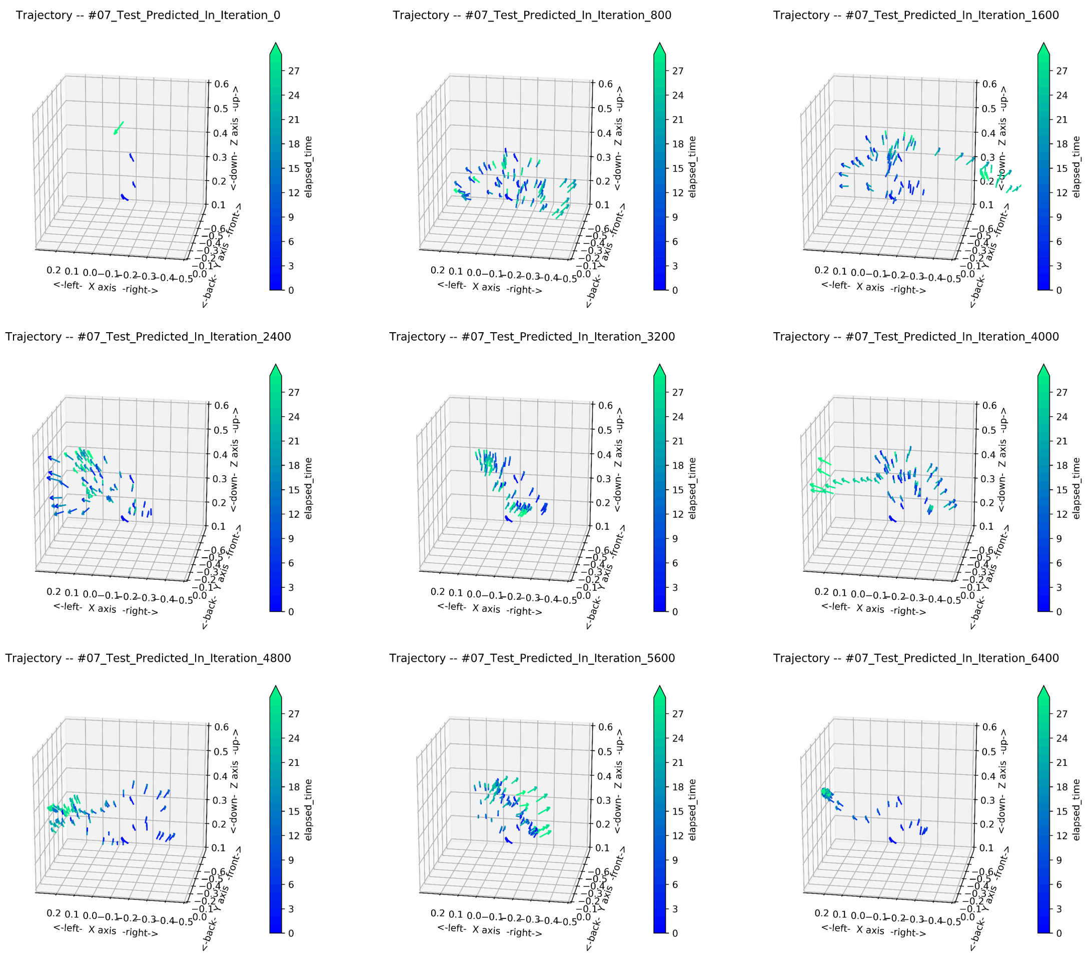


Figure 7.13: Predictions From State-to-State GAIL (EvalSeed #7)

Figures 7.12 and 7.13 show how prediction progressed as training took place, with two examples with different seed input (EvalSeed #0 and #7). The predicted pouring trajectories were not ideal, even after training converged.

7.3.3 Full-Trajectory BC

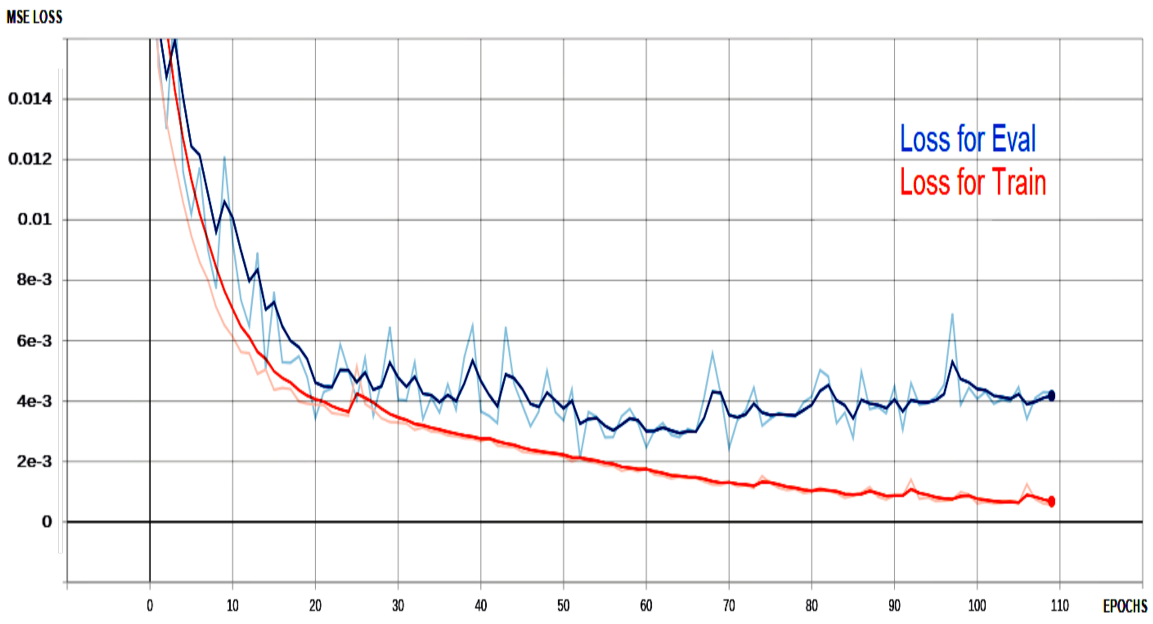


Figure 7.14: MSE Loss Learning curve of State-to-Full-Trajectory BC

Figure 7.14 shows that the state-to-full-trajectory BC-based model converged around epoch 25. Figures 7.15 and 7.16 show the progression. The predicted trajectories do not smoothly transition between the initial poses[3] and the predicted segments, nor resemble the behavior of pouring.

Although the seeds' values were not equal in the two figures, the predicted trajectory segments are almost identical[4]. Regardless of the requested starting pose, which was used as input to the network, the model always produces the same prediction as the rest of the trajectory. This means the model memorized solutions minimizing the loss function; however, it failed to learn the task.

[3]The initial 5 poses were used as seed input of the network. The final trajectory prediction is a concatenation of the initial points and outputs of the network.

[4]Both figures contain 9 trajectories visualized on a 3x3 grid, and the trajectories are compared against the trajectories of the other figure at the same position.

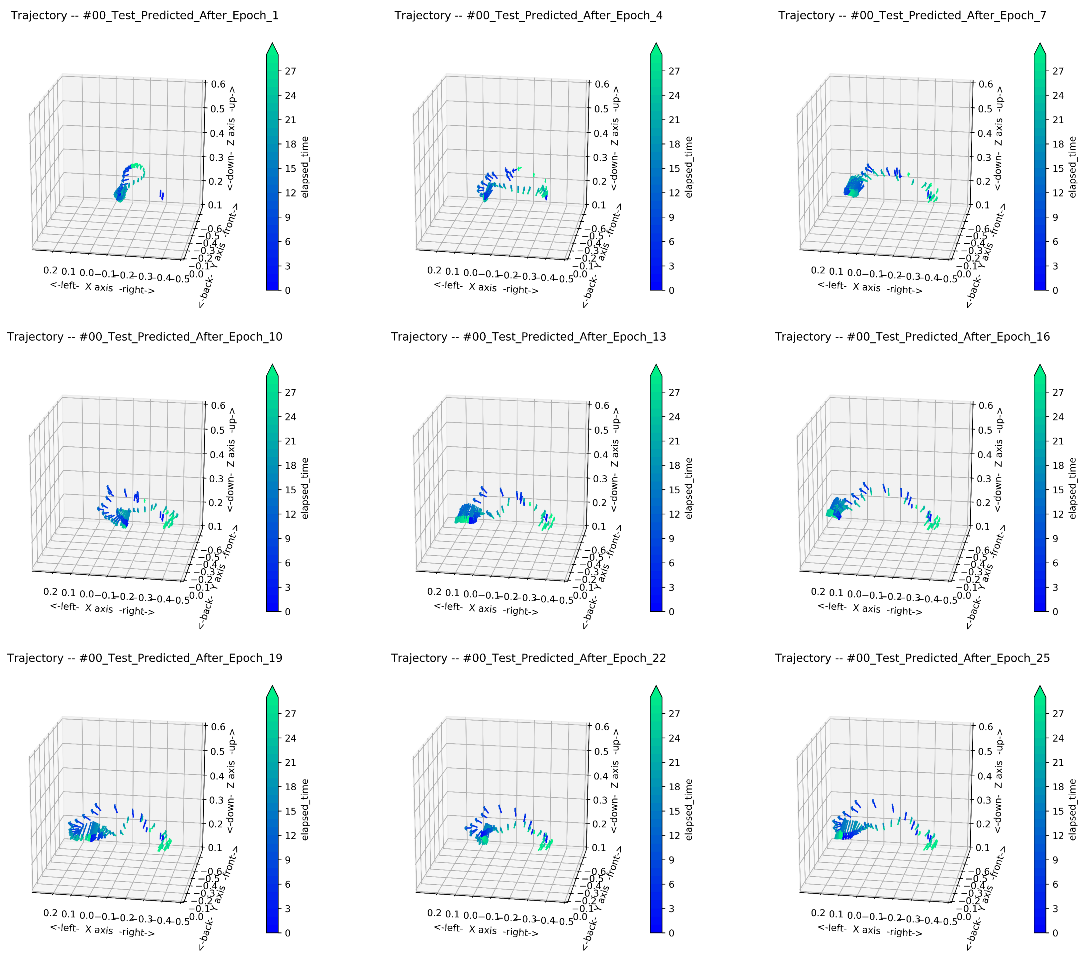


Figure 7.15: Predictions From State-to-Full-Trajectory BC (EvalSeed #0)

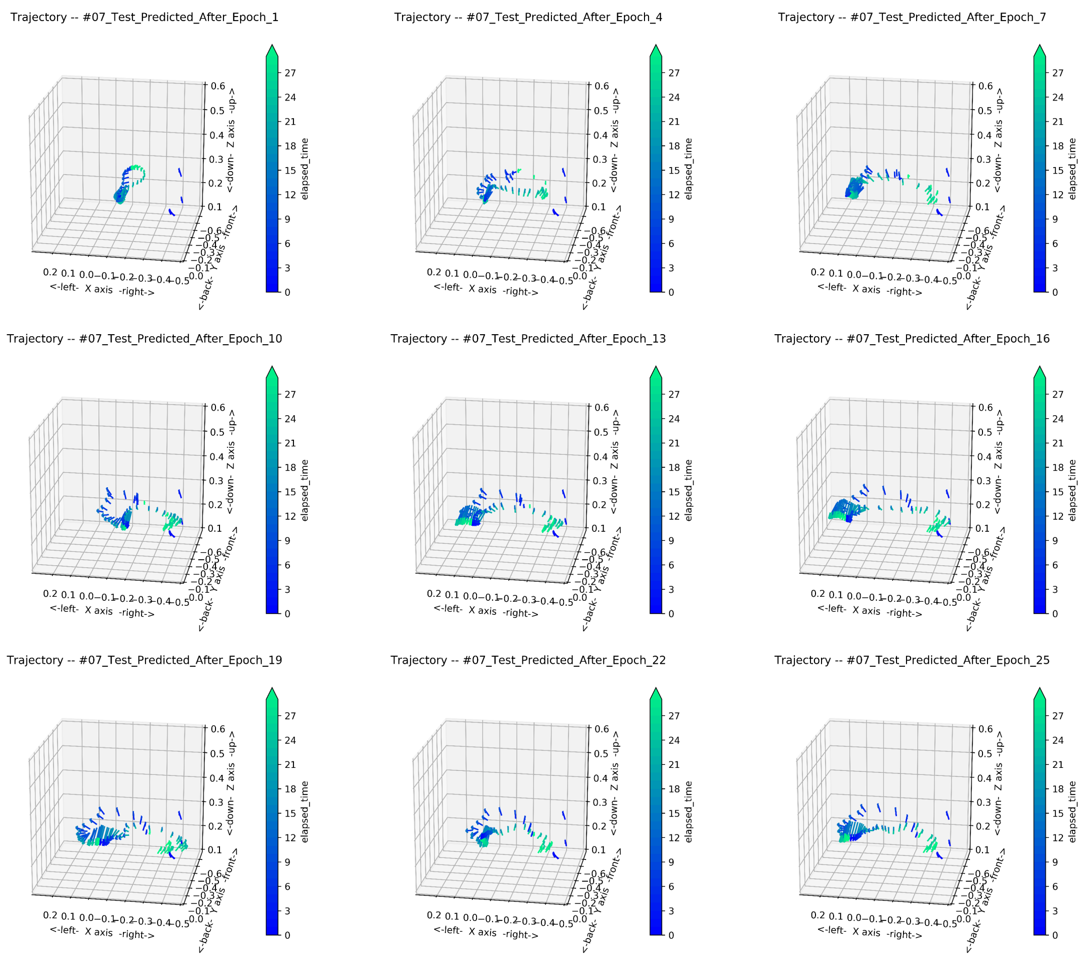


Figure 7.16: Predictions From State-to-Full-Trajectory BC (EvalSeed #7)

7.3.4 State-to-State BC

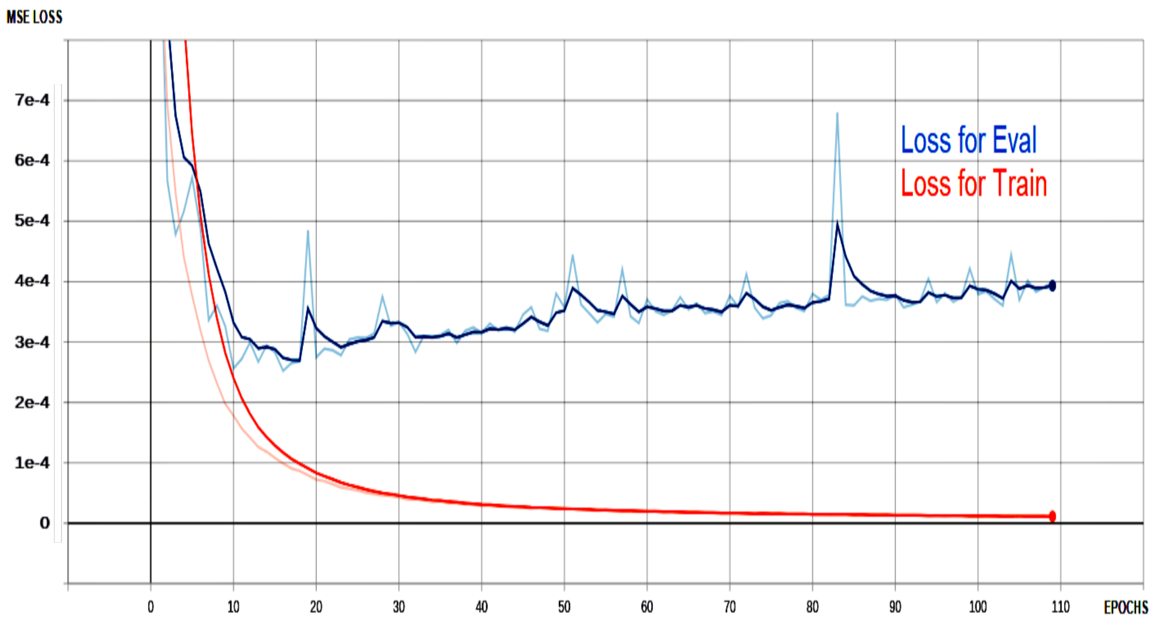


Figure 7.17: MSE Loss Learning curve of State-to-Full-Trajectory BC

Figure 7.17 shows that the state-to-full-trajectory BC-based model converged around epoch 15. As shown in Figures 7.18 and 7.19, while the network was able to output smooth and diverse predictions, the trajectories do not resemble the behavior of pouring.

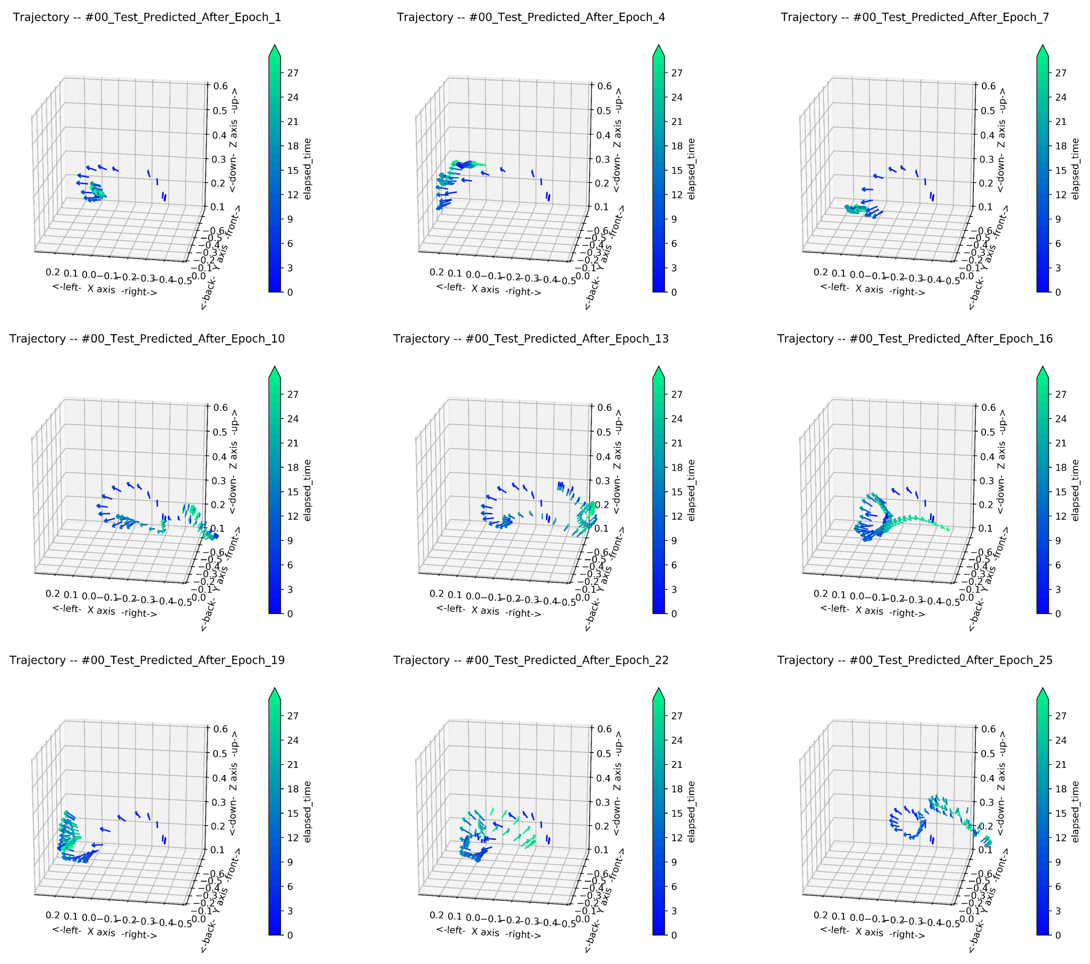


Figure 7.18: Predictions From State-to-State BC (EvalSeed #0)

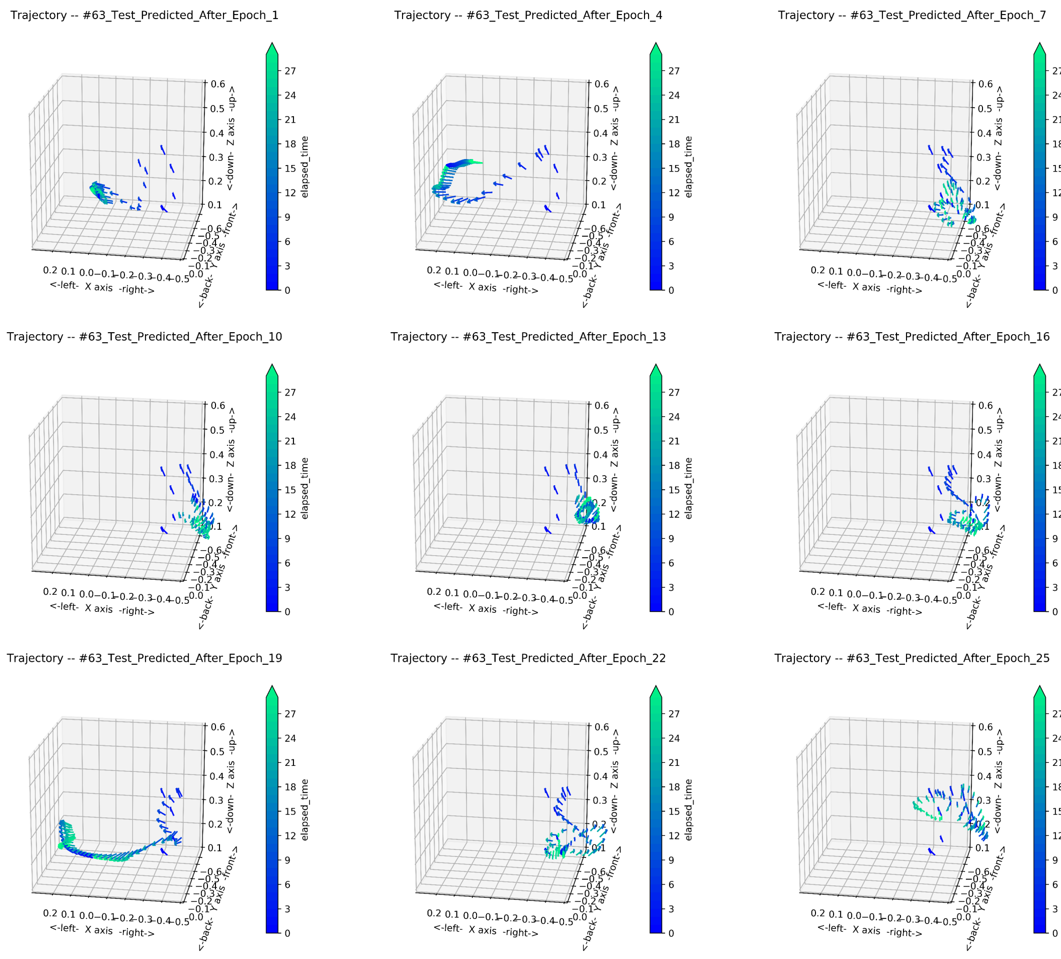


Figure 7.19: Predictions From State-to-State BC (EvalSeed #7)

7.3.5 Without Long Skipping Edges

In addition to comparing with baseline models, Figure 7.20 shows one example of learning progression of unsuccessful learning, where the proposed model was duplicated, except only the long edges were removed. The results suggest that the long skipping edges in the networks were crucial to the information flow.

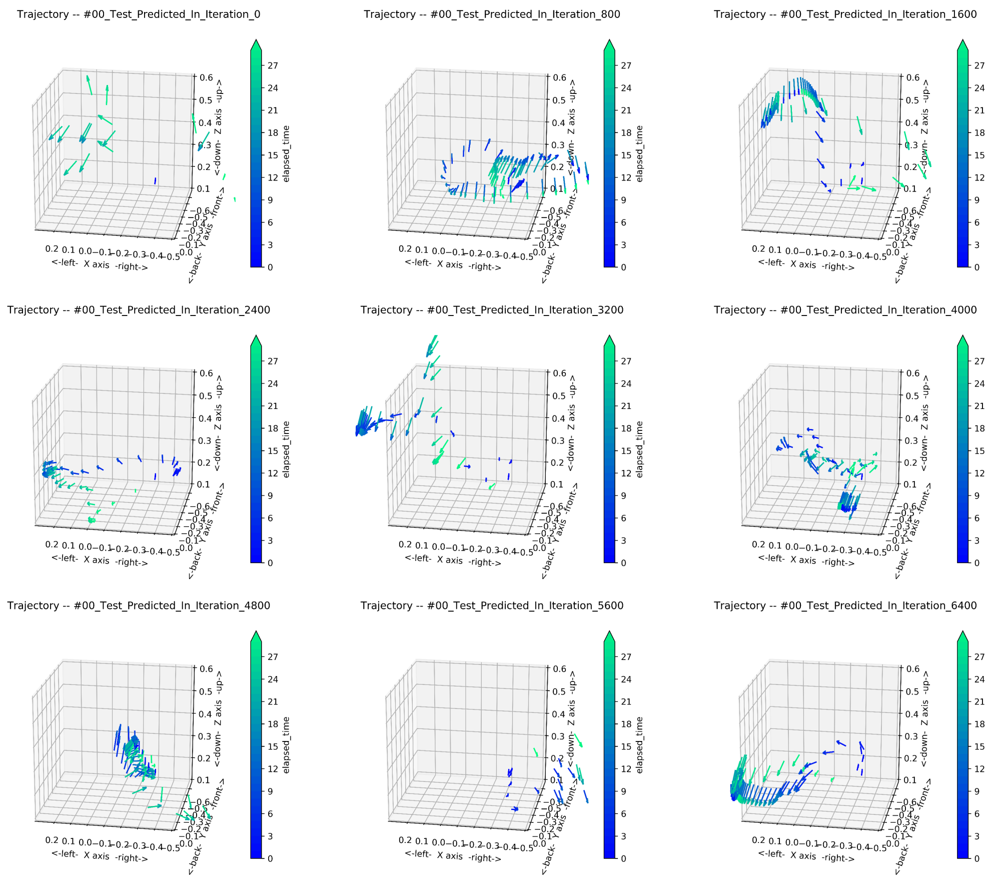


Figure 7.20: Progression of Model Predictions, for State-to-Full-Trajectory GAN Model Without Long Skipping Edges

To summarize, only the proposed methods were able to make observable improvement as training progressed.

7.4 Visualized Trajectory Predictions

We also visualized how final predictions compared to ground truth demonstrations[5] for each method.

[5]To avoid unclear visualization caused by overlapped starting points, we intentionally shifted the plotting of predicted trajectories via geometric translation toward the positive direction on the x-axis.

7.4 Visualized Trajectory Predictions

Unsurprisingly, learning curves and trajectory prediction can be drastically different when repeated with the same training configuration, influenced by the stochastic factors during training. In this section, we visualized example predictions of the same seed input (EvalSeed #0), hoping to reveal the common characteristics of the trajectories planned by each method.

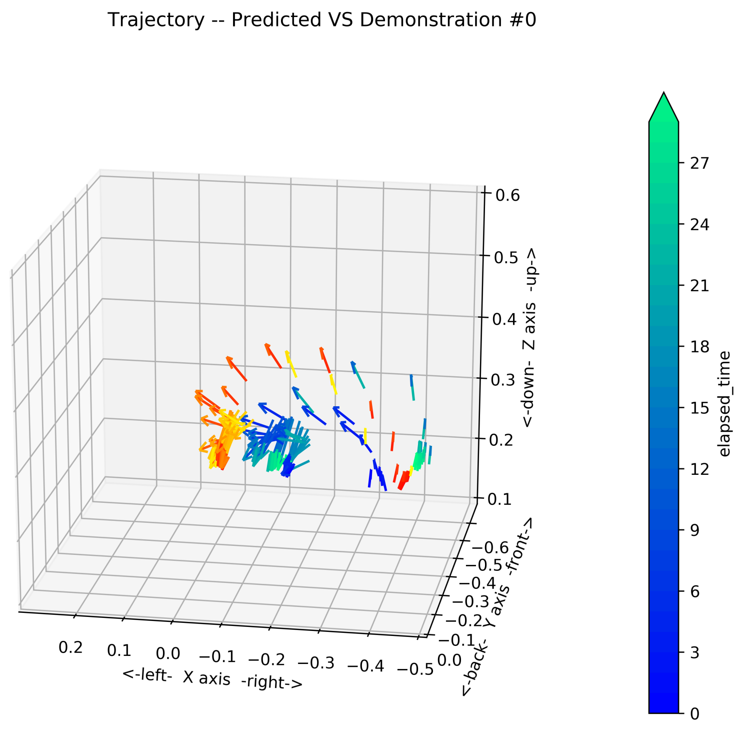


Figure 7.21: State-to-Full-Trajectory GAN Prediction Compared to Demonstration

The state-to-full-trajectory GAN-based model was able to produce fluent motions. Shown in Figure 7.21, the arm's pose starts from the deep blue colored arrows on the right side, gradually lifts and shifts leftwards, then slowly tilts downwards. Each arrow's pose would be smoothly executed at a fixed frequency, so the high density of arrows on the left side would lead to a pausing behavior during execution. Finally, the arm quickly lifts up and resets its pose to the right side. Although the predicted blue-cyan sequence does not closely follow the demonstrated red-yellow sequence, we can see that the overall trajectory indicates a plausible pouring motion. As the model was trained with a variety of demonstrations, with examples shown in Appendix Figures A.7 to A.14, we consider the result an indication of effective learning, because the GAN-based learner appeared to able to extract and recreate the essence of the pouring motion.

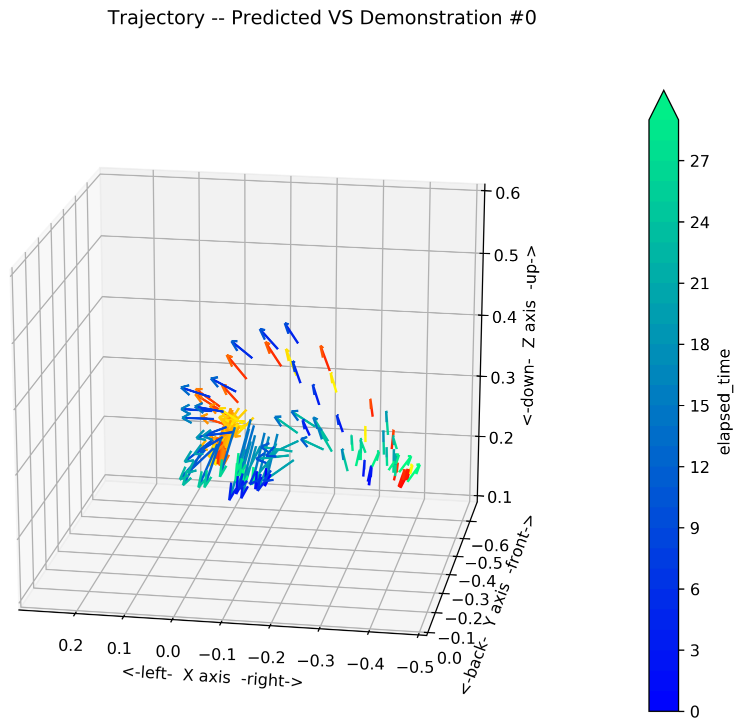


Figure 7.22: State-to-Full-Trajectory BC Prediction Compared to Demonstration

The state-to-full-trajectory BC-based model was also able to predict a plausible pouring motion. However, the large and irregular oscillations[6] indicates turbulent motion. Moreover, this BC-based model would often predict the exact same trajectories without adapting to different initial poses in untrained scenarios. As previously shown in Figures 7.15 and

[6]In particular, obvious oscillation occurred in the deep blue sequence before 3 seconds elapsed and the cyan sequence after 18 seconds had elapsed.

7.16, although the seeding input was different, prediction for the rest of trajectory was the same. This indicates memorization, rather than learning.

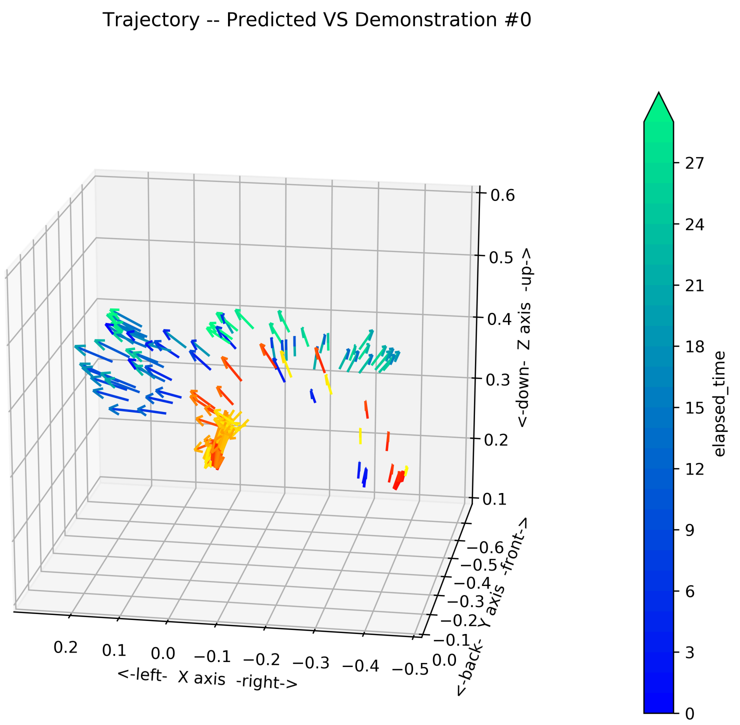


Figure 7.23: State-to-State GAIL Prediction Comparing to Demonstration

The state-to-state GAIL-based model produced relatively smooth motion in bringing the arm from the right side toward the left, and brought the arm back near the reset pose. However, tilting and pausing was crucial for a pouring motion, which was lacking in the predicted trajectory.

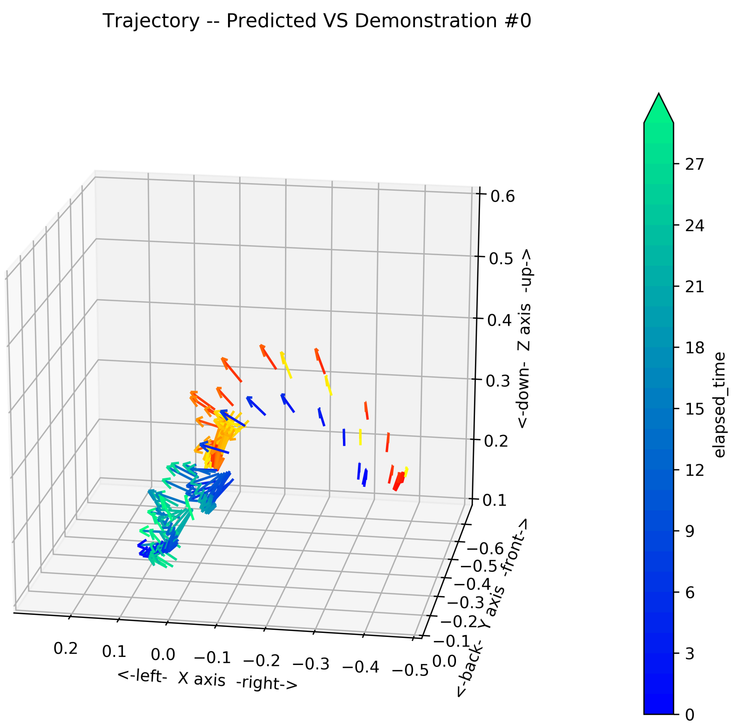


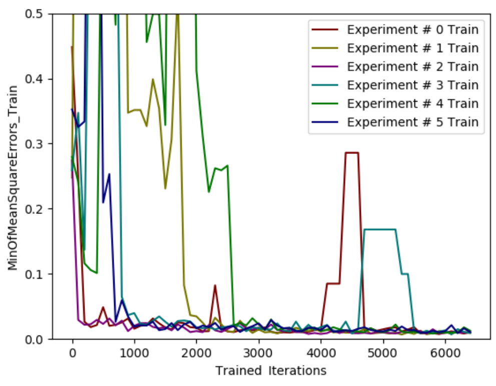
Figure 7.24: State-to-State BC Prediction Compared to Demonstration

Finally, the state-to-state BC-based model was able to produce smooth motion, but failed to recreate the overall behavior of tilting, pausing and reset.

Overall, the proposed method demonstrated more effective learning.

7.5 Hyperparameters Exploration

To search for better hyperparameters, we firstly establish the stability of model training. This is done by comparing the Negative Rewards, defined in Equation 6.4, of repetitive experiments conducted with the same training configuration.



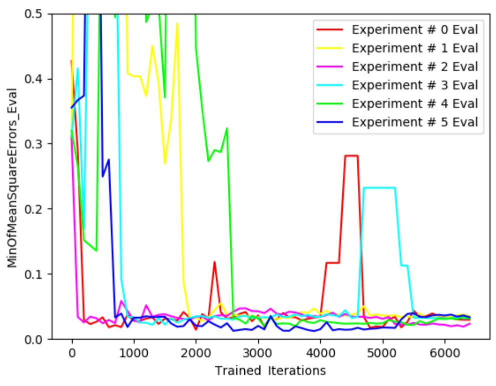


Figure 7.25: Consistency of Training Evaluated by *Train* Dataset

Figure 7.26: Consistency of Training Evaluated by *Eval* Dataset

The results are shown in Figures 7.25 and 7.26. Training progress can vary drastically between different runs of experiments, but eventually, they were able to consistently converge. Due to computation time constraints, those experiments were conducted with lite training configurations. In contrast, the experiments in the previous sections were conducted with the best known configurations.

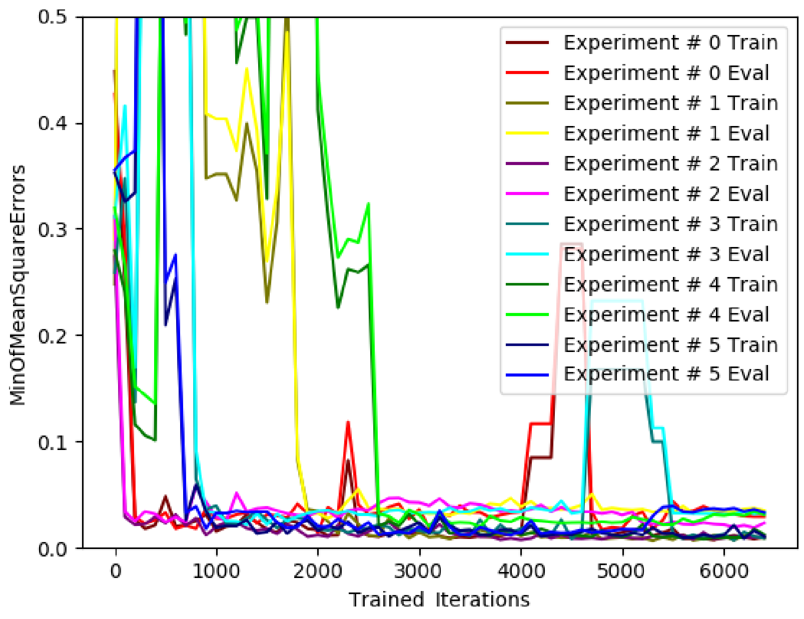


Figure 7.27: Training Consistency

Merging the two figures together, in Figure 7.27, we can observe that, although the Negative Rewards have large variance between experiments, given any single run of the experiments, the rewards measured by *Train* and *Eval* tightly correlate.

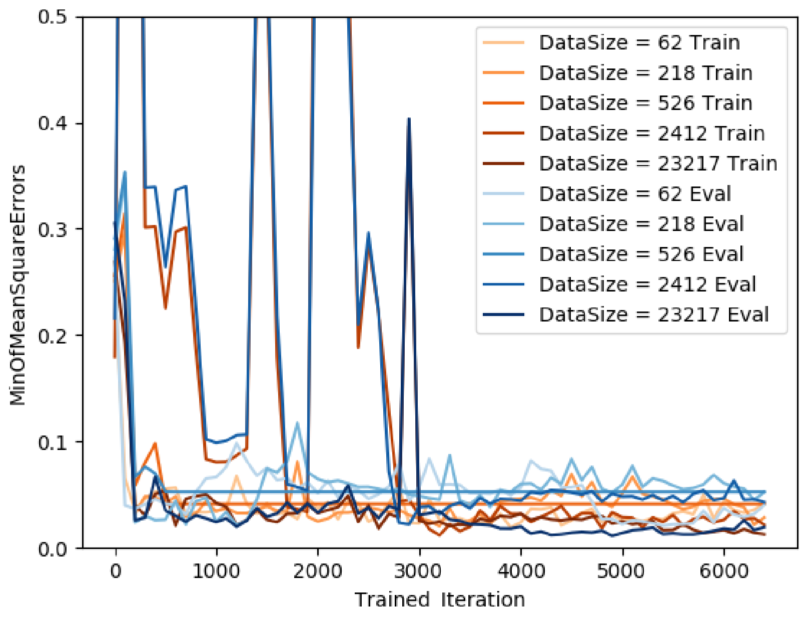


Figure 7.28: Effect of Synthetic Dataset Size

Figure 7.28 explores the relationship between the size of the training dataset and Negative Reward measured on both *Train* and *Eval*. From the original 12 demonstrations, we generated various sizes of synthetic trajectories, as described in Section 5.3. Due to the measurement to avoid mixing the *Eval* dataset in training, *Train* trajectories with seeds too close to any of *Eval*'s seeds were removed, with methodology illustrated in Section 5.3. The sizes shown in the figure describe the dataset after violating trajectories that were pruned. Overall, the learning benefited from larger dataset size, even though they are mostly synthetic.

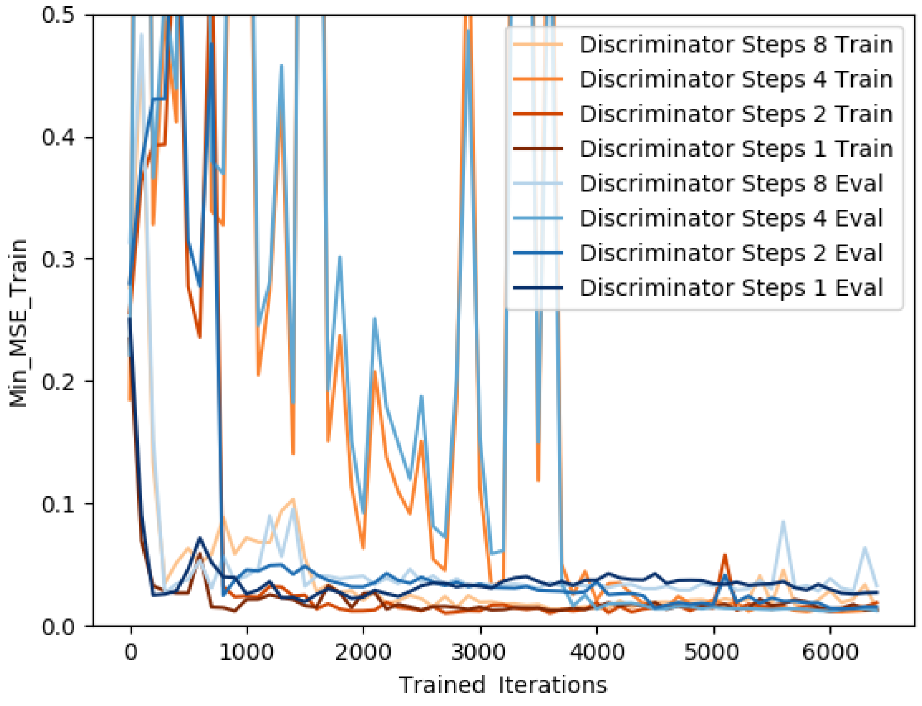


Figure 7.29: Discriminator Steps Per Iteration

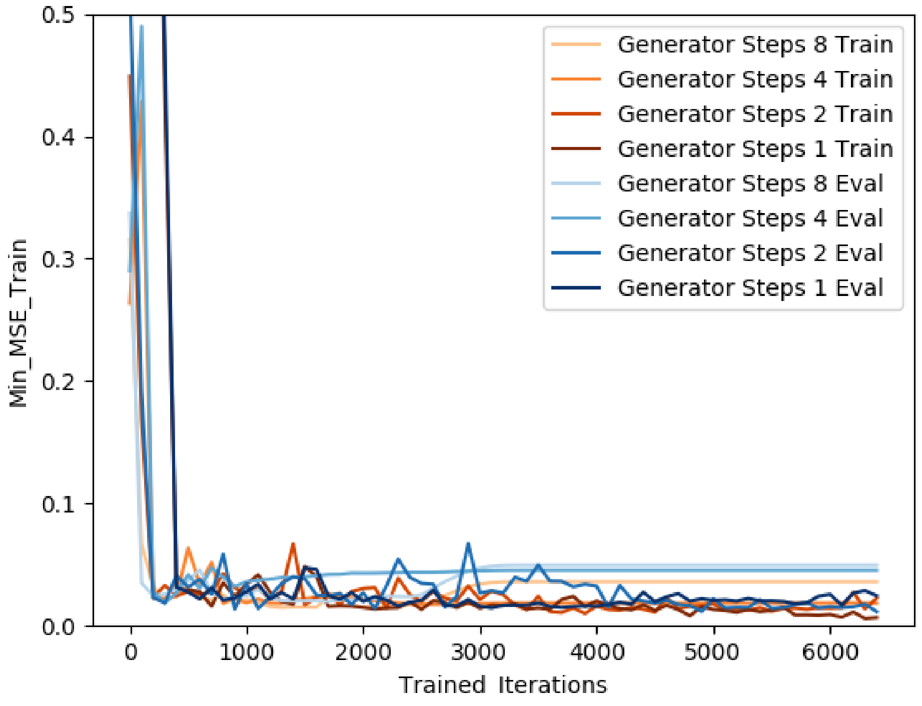


Figure 7.30: Generator Steps Per Iteration

Listing 4.2 presented the GAN training algorithm introduced by its original paper. In each iteration, the Discriminator takes multiple gradient steps, while the Generator only takes one step. With both networks compiled with the Adam optimizer[7] and fitted with mini-batches of size 128, we searched the hyperparameter space to optimize the number of gradient steps for both of the two networks. As shown in Figures 7.29 and 7.30, by comparing Negative Rewards at iteration $6,000$ for our particular experimental settings, taking 1 step for Discriminator and 2 steps for Generator appeared to be the best configuration.

[7]The Adam optimizer was configured with $learningRate = 0.0002$, $beta1 = 0.5$ and $beta2 = 0.999$.

7.6 Manipulation Trajectories Performed in Simulation

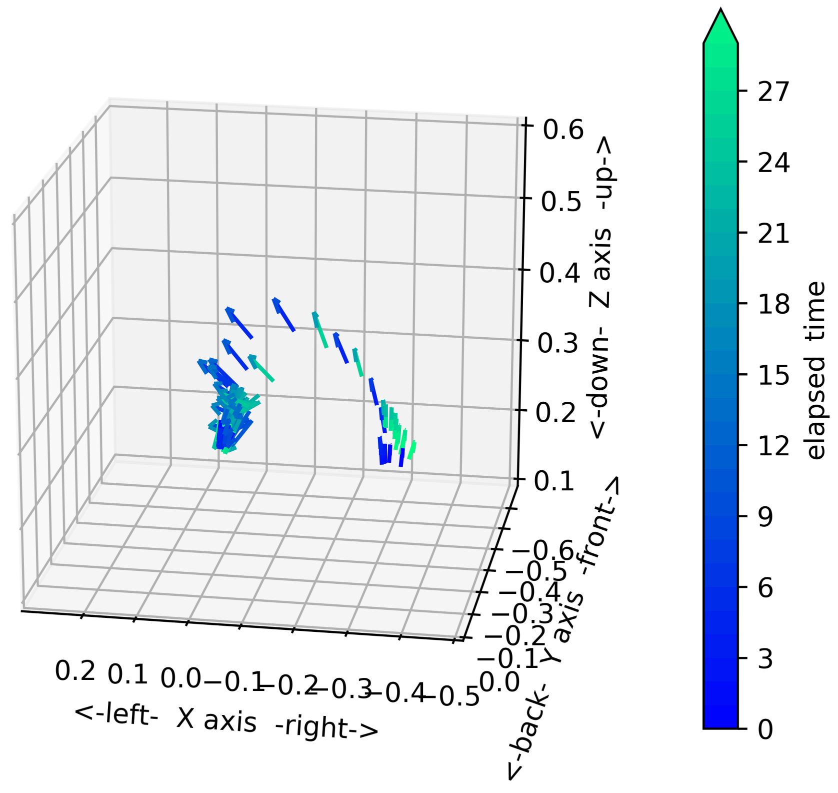


Figure 7.31: One Example of the Final Predictions

The final predicted trajectory is a sequence of 64 poses. The prediction would be denormalized and passed to the MOVEit motion planner and JACO arm controller for execution. Figure 7.31 presents one example of the predictions, while Figure 7.32 animated[8] the motion of that particular trajectory, when it was executed in the ROS Gazebo environment.

[8]Unlike the animation of Figure 7.3, snapshots from Gazebo simulation were not evenly sampled: lagging frames produced when waiting for MOVEit planning computation were manually pruned out.

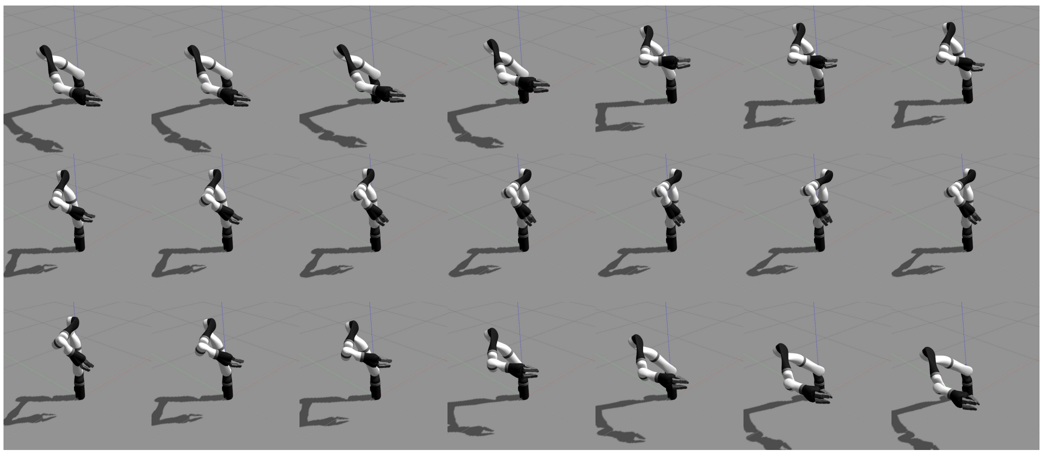


Figure 7.32: Executing the Learned Pouring Trajectory in Gazebo Simulation

8 Conclusions and Future Work

We proposed and prototyped a teachable robotic system, which learns simple tasks from demonstrations. The system was deployed on a state-of-the-art commercial robotic arm. The work overcame various practical engineering obstacles, examined various trade-offs when designing the system, explored various training strategies for our GAIL-based deep learning model, and eventually delivered a proof-of-concept teachable robotics system.

8.1 Conclusions

In experiments on the simple task of pick-and-place, the system was able to learn from raw human demonstration recordings, eventually performing the task under untrained scenarios. On the more complicated task of pouring, our approach outperformed the 3 baseline models. The learned trajectory planner was able to consistently produce plausible trajectories, which led to smooth motion and successful pouring behavior.

As the main contribution of the research, we proposed a generative, adversarial network-based full-trajectory prediction deep imitation learning model, which performed better than the baselines, both qualitatively and quantitatively. Measured by a goodness function defined by Equation 6.4, the quantitative results of our pouring imitation learning scenario displayed two trends. The state-to-full-trajectory prediction models performed significantly better than the state-to-state prediction models. We believe predicting full trajectory at once benefited the CNN-based networks in capturing longer intra-trajectory dependency, which empowered those models to produce more coherent long-term predictions. Meanwhile, results from the GAIL-based models performed better than the BC-based models: the former demonstrated better generalization, while the latter indicated memorization.

8.2 Future Work

Contemporary robots excel at following low-level instructions, often with precision and efficiency that far exceeds human capabilities. However, until we figure out an efficient way to enable robots to learn new tasks and adapt to new environments, those intelligent, loyal friends in science fiction have to remain fictional. While the ubiquitous presence of machine automation applications has been a major driving force of the modern industry, a major challenge when building a general purpose smart robot is that the traditional robotic systems require the agents to be explicitly programmed for each different circumstance and environment. Our work took a small step in addressing this era-defining challenge, but there is a long way to go.

Object Detection and Tracking

To simplify our experimental setting, we prearranged color-coded markers on parts of the gripper, as well as all relevant objects in the scene. For a practical real-world application, this unsaleable solution should be replaced with sophisticated object detection and tracking neural networks, and possibly with other types of sensor. An end-to-end solution may be viable with more demonstration data, or visual imagery transfer learning.

More Complex Tasks

The system could be scaled vertically to tackle more complicated tasks. The original complex task may be segmented into a sequence of simple tasks, which can be planned and executed separately. It is also possible to horizontally scale the system, where multiple agents can learn to operate in collaboration.

Real-Time Manipulation Trajectory Planning

Our teachable robotic system attempts to plan the complete manipulation trajectory before execution. This requires the robot to be operated in predictable and perfectly modeled environments. A real-world manipulation controller should be able to react to real-time feedback, with possibly the support of visual and force sensors. With robust hardware and sophisticated imitation learning methods, this could be done with explicitly modeled MDPs, or implicitly modeled high-level deep learning controllers.

Bibliography

[AAS99] Pankaj K Agarwal, Boris Aronov, and Micha Sharir. Motion planning for a convex polygon in a polygonal environment. *Discrete & Computational Geometry*, 22(2):201–221, 1999.

[AET$^+$14] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[AF17] Engr Akhtar and SM Farrukh. *Practical Reinforcement Learning: Develop self-evolving, intelligent agents with OpenAI Gym, Python and Java*. Packt Publishing, 2017.

[AHS$^+$09] Joe Augenbraun, Linda Hirschhorn, Pankaj Shah, Nick Donaldson, William Jayne, Juan B Gomez, and Joseph Pinzarrone. Multi-function robotic device, June 30 2009. US Patent 7,555,363.

[BACM17] Nir Baram, Oron Anschel, Itai Caspi, and Shie Mannor. End-to-end differentiable adversarial imitation learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 390–399. JMLR. org, 2017.

[BAM16] Nir Baram, Oron Anschel, and Shie Mannor. Model-based adversarial imitation learning. *arXiv preprint arXiv:1612.02179*, 2016.

[BB07] Brendan Burns and Oliver Brock. Sampling-based motion planning with sensing uncertainty. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3313–3318. IEEE, 2007.

[BCP$^+$16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Bic94] A Bicchi. On the form-closure property of robotic grasping. *IFAC Proceedings Volumes*, 27(14):219–224, 1994.

[BK00] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 348–353. IEEE, 2000.

[BKV10] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, pages 2689–2696. IEEE, 2010.

[BL13] Matteo Bianchi and Minas V Liarokapis. Handcorpus, a new open-access repository for sharing experimental data and results on human and artificial hands. In *IEEE World Haptics Conference (WHC)*, 2013.

[But14] Thomas Butkiewicz. Low-cost coastal mapping using kinect v2 time-of-flight cameras. In *2014 Oceans-St. John's*, pages 1–9. IEEE, 2014.

[CAGT18] Enric Corona, Guillem Alenyà, Antonio Gabas, and Carme Torras. Active garment recognition and target grasping point detection using deep learning. *Pattern Recognition*, 74:629–641, 2018.

[CCK$^+$18] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018.

[CDFM18] Maell Cullen, Ben Davey, Karl J Friston, and Rosalyn J Moran. Active inference in openai gym: A paradigm for computational investigations into psychiatric illness. *Biological psychiatry: cognitive neuroscience and neuroimaging*, 3(9):809–818, 2018.

[CG17] Robert Chuchro and Deepak Gupta. Game playing with deep q-learning using openai gym. *Semantic Scholar*, 2017.

[CHC$^+$12] Doo-jin Choi, Seong-jong Han, Yun-Seo Choi, Young-Jun Park, and Jae-hoon Kim. Multi-function robot for moving on wall using indoor global positioning system, July 3 2012. US Patent 8,214,081.

[CK11] Jaedeug Choi and Kee-Eung Kim. Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1989–1997, 2011.

[CK14] Jaedeug Choi and Kee-Eung Kim. Hierarchical bayesian inverse reinforcement learning. *IEEE transactions on cybernetics*, 45(4):793–805, 2014.

[CLLL$^+$19] Alexandre Campeau-Lecours, Hugo Lamontagne, Simon Latour, Philippe Fauteux, Véronique Maheu, François Boucher, Charles Deguire, and Louis-Joseph Caron L'Ecuyer. Kinova modular robot arms for service robotics applications. In *Rapid Automation: Concepts, Methodologies, Tools, and Applications*, pages 693–719. IGI Global, 2019.

[CMM$^+$11] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[CPK$^+$17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[Cra09] John J Craig. *Introduction to robotics: mechanics and control, 3/E.* Pearson Education India, 2009.

[CSC12] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

[CSKX15] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[CSLG19] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338, 2019.

[DAS$^+$17] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[DC97] Rob A Dunne and Norm A Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer, 1997.

[des19] Kinova's 'jaco' is a robotic arm designed to give independence to power wheelchair users, Jul 2019.

[DFL14] Stefanos Doltsinis, Pedro Ferreira, and Niels Lohse. An mdp model-based reinforcement learning approach for production station ramp-up optimization: Q-learning analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(9):1125–1138, 2014.

[DMP18] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 2018.

[DRG15] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

[dSCGF$^+$15] Diego Brito dos Santos Cesar, Christopher Gaudig, Martin Fritsche, Marco A dos Reis, and Frank Kirchner. An evaluation of artificial fiducial markers in underwater environments. In *OCEANS 2015-Genova*, pages 1–6. IEEE, 2015.

[DSFK08] Rosen Diankov, Siddhartha S Srinivasa, Dave Ferguson, and James Kuffner. Manipulation planning with caging grasps. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 285–292. IEEE, 2008.

[EDK14] Kyriakos Efthymiadis, Sam Devlin, and Daniel Kudenko. Knowledge revision for reinforcement learning with abstract mdps. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1535–1536, 2014.

[EHR17] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[FB11] T Feix and Otto Bock. Human grasping database. *Internet: http://web. student. tuwien. ac. at/˜ e0227312/,[Sep. 10, 2010]*, 2011.

[FBH$^+$18] Kuan Fang, Yunfei Bai, Stefan Hinterstoisser, Silvio Savarese, and Mrinal Kalakrishnan. Multi-task domain adaptation for deep learning of instance grasping from simulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3516–3523. IEEE, 2018.

[FBR$^+$15] Péter Fankhauser, Michael Bloesch, Diego Rodriguez, Ralf Kaestner, Marco Hutter, and Roland Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 388–394. IEEE, 2015.

[FC92] Carlo Ferrari and John F Canny. Planning optimal grasps. In *ICRA*, volume 3, pages 2290–2295, 1992.

[FDSF18] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. Learning temporal strategic relationships using generative adversarial imitation learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 113–121. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[FYZ$^+$17] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[GCDA09] Corey Goldfeder, Matei Ciocarlie, Hao Dang, and Peter K Allen. The columbia grasp database. In *2009 IEEE international conference on robotics and automation*, pages 1710–1716. IEEE, 2009.

[GGC$^+$15] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.

[GH18] Adam Gleave and Oliver Habryka. Multi-task maximum entropy inverse reinforcement learning. *arXiv preprint arXiv:1805.08882*, 2018.

[GPAM$^+$14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[HCH$^+$17] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip HS Torr. Deeply supervised salient object detection with short connections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3203–3212, 2017.

[HE16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[HEKL$^+$13] Bidan Huang, Sahar El-Khoury, Miao Li, Joanna J Bryson, and Aude Billard. Learning a real time grasping strategy. In *2013 IEEE International Conference on Robotics and Automation*, pages 593–600. IEEE, 2013.

[HHW$^+$15] Wei Hu, Yangyu Huang, Li Wei, Fan Zhang, and Hengchao Li. Deep convolutional neural networks for hyperspectral image classification. *Journal of Sensors*, 2015, 2015.

[HJW$^+$17] Xian-Feng Han, Jesse S Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, and Liping Xiao. A review of algorithms for filtering the 3d point cloud. *Signal Processing: Image Communication*, 57:103–112, 2017.

[HMDC16] Caner Hazirbas, Lingni Ma, Csaba Domokos, and Daniel Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Asian conference on computer vision*, pages 213–228. Springer, 2016.

[HNY13] Daichi Hirano, Kenji Nagaoka, and Kazuya Yoshida. Design of underactuated hand for caging-based grasping of free-flying object. In *Proceedings of the 2013 IEEE/SICE International Symposium on System Integration*, pages 436–442. IEEE, 2013.

[HPA$^+$90] Robert D Howe, Nicolas Popp, Prasad Akella, Imin Kao, and Mark R Cutkosky. Grasping, manipulation, and control with tactile sensing. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 1258–1263. IEEE, 1990.

[HS15] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[HSB18] Kay Gregor Hartmann, Robin Tibor Schirrmeister, and Tonio Ball. Eeg-gan: Generative adversarial networks for electroencephalograhic (eeg) brain signals. *arXiv preprint arXiv:1806.01875*, 2018.

[HSSR05] Robert Haschke, Jochen J Steil, Ingo Steuwer, and Helge Ritter. Task-oriented quality measures for dextrous grasping. In *2005 International Sym-*

posium on Computational Intelligence in Robotics and Automation, pages 689–694. IEEE, 2005.

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IBN18] Mahdi Imani and Ulisses M Braga-Neto. Control of gene regulatory networks using bayesian inverse reinforcement learning. *IEEE/ACM transactions on computational biology and bioinformatics*, 16(4):1250–1261, 2018.

[IHP18] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[IP19] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[JBD18] Stephen James, Michael Bloesch, and Andrew J Davison. Task-embedded control networks for few-shot imitation learning. *arXiv preprint arXiv:1810.03237*, 2018.

[JLD16] Edward Johns, Stefan Leutenegger, and Andrew J Davison. Deep learning a grasp function for grasping under gripper pose uncertainty. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4461–4468. IEEE, 2016.

[JMS11] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. Efficient grasping from rgbd images: Learning using a new rectangle representation. In *2011 IEEE International conference on robotics and automation*, pages 3304–3311. IEEE, 2011.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBS15] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. Leveraging big data for grasp planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311. IEEE, 2015.

[KCL$^+$15] Konstantinos Kamnitsas, Liang Chen, Christian Ledig, Daniel Rueckert, and Ben Glocker. Multi-scale 3d convolutional neural networks for lesion segmentation in brain mri. *Ischemic stroke lesion segmentation*, 13:46, 2015.

[KF10] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104(2), 2010.

[KIG15] Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Real-time patient-specific ecg classification by 1-d convolutional neural networks. *IEEE Transactions on Biomedical Engineering*, 63(3):664–675, 2015.

[Kim14] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[KMWK17] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE, 2017.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KW15] Marek Kopicki and Jeremy L Wyatt. One shot contact learning. In *Proceedings of the RSS Workshop on bridging the gap between data-driven and analytical physics-based grasping and manipulation, Rome, Italy*, 2015.

[LaV06] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[LB$^+$95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LCS$^+$19] Dan Li, Dacheng Chen, Lei Shi, Baihong Jin, Jonathan Goh, and See-Kiong Ng Mad-gan. Multivariate anomaly detection for time series data with generative adversarial networks. *arXiv preprint arXiv:1901.04997*, 2019.

[LK02] Orion Sky Lawlor and Laxmikant V Kalée. A voxel-based parallel collision detection algorithm. In *Proceedings of the 16th international conference on Supercomputing*, pages 285–293, 2002.

[LMDBS18] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[LOJ14] Richard Leslie, Christopher J O'Donnell, and Andrew D Johnson. Grasp: analysis of genotype–phenotype results from 1390 genome-wide association studies and corresponding open access database. *Bioinformatics*, 30(12):i185–i194, 2014.

[LPK$^+$18] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[LPMG17] Christoph Lassner, Gerard Pons-Moll, and Peter V Gehler. A generative model of people in clothing. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 853–862, 2017.

[LS88] Zexiang Li and S Shankar Sastry. Task-oriented optimal grasping by multi-fingered robot hands. *IEEE Journal on Robotics and Automation*, 4(1):32–44, 1988.

[LSE17] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3812–3822, 2017.

[LTH$^+$17] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[MAFR11] Veronique Maheu, Philippe S Archambault, Julie Frappier, and François Routhier. Evaluation of the jaco robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–5. IEEE, 2011.

[MBC$^+$06] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[MCL18] Douglas Morrison, Peter Corke, and Jürgen Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *arXiv preprint arXiv:1804.05172*, 2018.

[MDGK18] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[Mit15] Roni Mittelman. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015.

[MJS$^+$17] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. In *Advances in Neural Information Processing Systems*, pages 406–416, 2017.

[MKE12] Yusuke Maeda, Naoki Kodera, and Tomohiro Egawa. Caging-based grasping by a robot hand with rigid and soft parts. In *2012 IEEE international conference on robotics and automation*, pages 5150–5155. IEEE, 2012.

[MLN$^+$17] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[MP17] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[MPH$^+$16] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1957–1964. IEEE, 2016.

[MS15] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[Mus18] Pirkko Mustamo. Object detection in sports: Tensorflow object detection api case study. *no. January*, 2018.

[MZB12] B Mustapha, A Zayegh, and RK Begg. Multiple sensors based obstacle detection system. In *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, volume 2, pages 562–566. IEEE, 2012.

[MZZS18] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[NR+00] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[OA16] Billy Okal and Kai O Arras. Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2889–2895. IEEE, 2016.

[OOS17] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[OPN+18] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[OVR14] Ian Osband and Benjamin Van Roy. Near-optimal reinforcement learning in factored mdps. In *Advances in Neural Information Processing Systems*, pages 604–612, 2014.

[Pal18] Praveen Palanisamy. *Hands-On Intelligent Agents with OpenAI Gym: Your guide to developing AI agents using deep reinforcement learning*. Packt Publishing Ltd, 2018.

[PAR+18] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[PG16] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[PH10] Erion Plaku and Gregory D Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *2010 IEEE International Conference on Robotics and Automation*, pages 5002–5008. IEEE, 2010.

[Pom89] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[PSS$^+$16] Mark Pfeiffer, Ulrich Schwesinger, Hannes Sommer, Enric Galceran, and Roland Siegwart. Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2096–2101. IEEE, 2016.

[PT16] Domenico Prattichizzo and Jeffrey C Trinkle. Grasping. In *Springer handbook of robotics*, pages 955–988. Springer, 2016.

[PV16] David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*, 2016.

[RA07] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[RLS09] Daniel Roetenberg, Henk Luinge, and Per Slycke. Xsens mvn: Full 6dof human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV, Tech. Rep*, 1, 2009.

[RMBS$^+$13] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments.

In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.

[RMF12] Alberto Rodriguez, Matthew T Mason, and Steve Ferry. From caging to grasping. *The International Journal of Robotics Research*, 31(7):886–900, 2012.

[SAS17] Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *arXiv preprint arXiv:1703.01703*, 2017.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Ser09] Richard Serfozo. *Basics of applied stochastic processes*. Springer Science & Business Media, 2009.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[SLG90] Ching-Long Shih, T-T Lee, and William A Gruver. A unified approach for robot motion planning with moving polyhedral obstacles. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4):903–915, 1990.

[SLL11] Bao-Quan Shi, Jin Liang, and Qing Liu. Adaptive simplification of point cloud using k-means clustering. *Computer-Aided Design*, 43(8):910–922, 2011.

[SMK12] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[SRSE18] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 7461–7472, 2018.

[Sta90] Sharon A Stansfield. Knowledge-based robotic grasping. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 1270–1275. IEEE, 1990.

[SU15] Alexander G Schwing and Raquel Urtasun. Fully connected deep structured networks. *arXiv preprint arXiv:1503.02351*, 2015.

[SVG$^+$17] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3309–3318. JMLR. org, 2017.

[SW89] Sharad Singhal and Lance Wu. Training multilayer perceptrons with the extended kalman algorithm. In *Advances in neural information processing systems*, pages 133–140, 1989.

[TET12] 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*, pages 5026–5033. IEEE, 2012.

[TSM$^+$16] Andru P Twinanda, Sherif Shehata, Didier Mutter, Jacques Marescaux, Michel De Mathelin, and Nicolas Padoy. Endonet: A deep architecture for recognition tasks on laparoscopic videos. *IEEE transactions on medical imaging*, 36(1):86–97, 2016.

[TTS$^+$18] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*, 2018.

[TZLB18] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1111–1117. IEEE, 2018.

[VPSP17] Ulrich Viereck, Andreas ten Pas, Kate Saenko, and Robert Platt. Learning a visuomotor controller for real world robotic grasping using simulated depth images. *arXiv preprint arXiv:1706.04652*, 2017.

[WA12] Jonathan Weisz and Peter K Allen. Pose error robust grasping from contact wrench space metrics. In *2012 IEEE international conference on robotics and automation*, pages 557–562. IEEE, 2012.

[WD10] Yan Wu and Yiannis Demiris. Towards one shot learning by imitation for humanoid robots. In *2010 IEEE International Conference on Robotics and Automation*, pages 2889–2894. IEEE, 2010.

[Wie15] Thiemo Wiedemeyer. Iai kinect2. *Institute for artificial intelligence, University Bremen*, pages 2014–2015, 2015.

[WOP15] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[WS16] Oliver Wasenmüller and Didier Stricker. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Asian Conference on Computer Vision*, pages 34–45. Springer, 2016.

[WZX$^+$16] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*, pages 82–90, 2016.

[XRLJ14] Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. Deep convolutional neural network for image deconvolution. In *Advances in neural information processing systems*, pages 1790–1798, 2014.

[YA17] Gu Ye and Ron Alterovitz. Guided motion planning. In *Robotics research*, pages 291–307. Springer, 2017.

[YALF18] Tianhe Yu, Pieter Abbeel, Sergey Levine, and Chelsea Finn. One-shot hierarchical imitation learning of compound visuomotor tasks. *arXiv preprint arXiv:1810.11043*, 2018.

[ZG00] Li Zhang and Bijoy K Ghosh. Line segment based map building and localization using 2d laser rangefinder. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2538–2543. IEEE, 2000.

[ZLN14] Jiangchuan Zheng, Siyuan Liu, and Lionel M Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[ZMBD08] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.

[ZMJ$^+$18] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[ZWM$^+$18] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

[ZZMC13] Wenping Zhao, Jianjie Zhang, Jianyuan Min, and Jinxiang Chai. Robust realtime physics-based motion control for human grasping. *ACM Transactions on Graphics (TOG)*, 32(6):1–12, 2013.

A
Supplemental Figures

Supplemental Figures

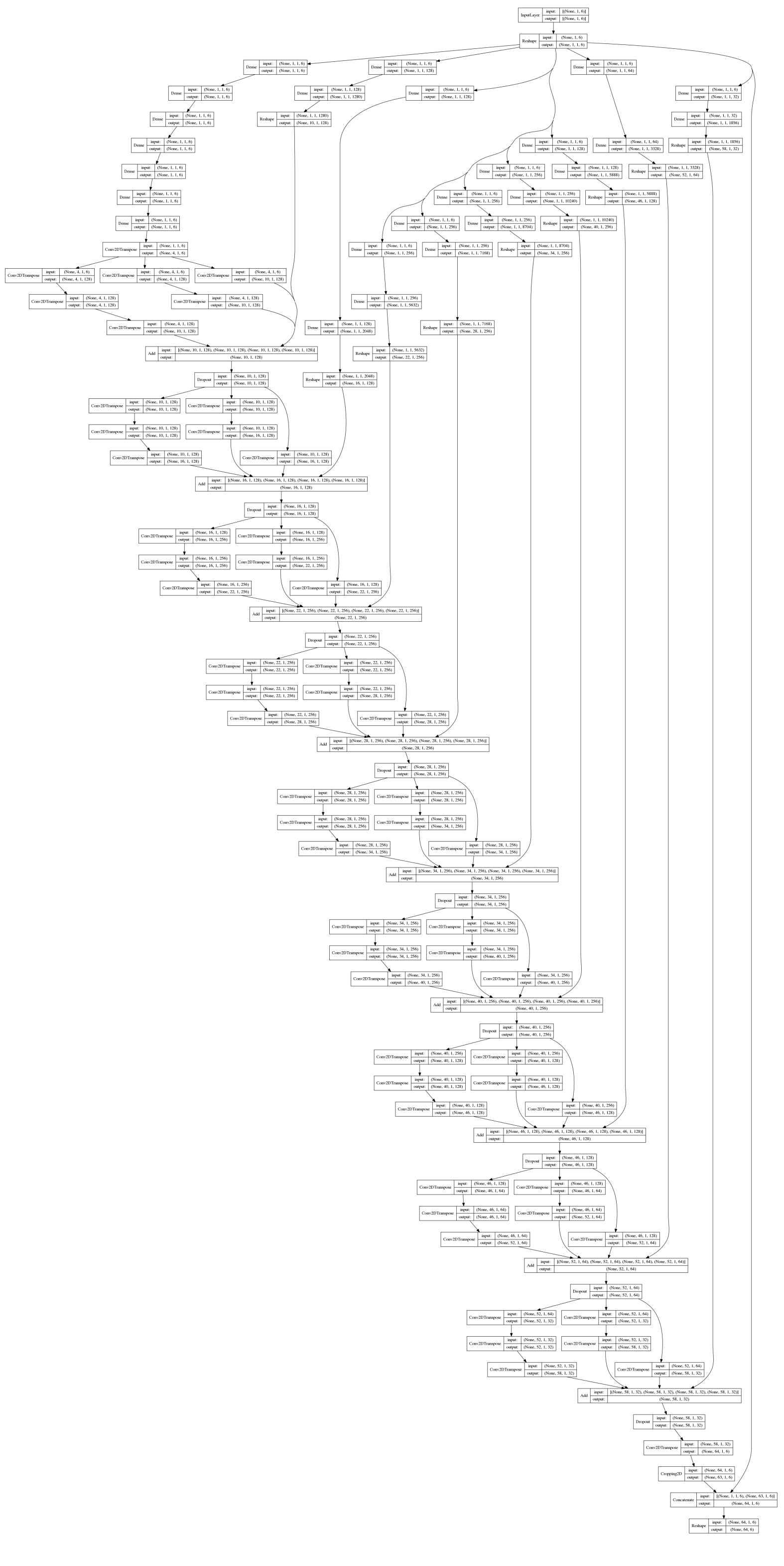
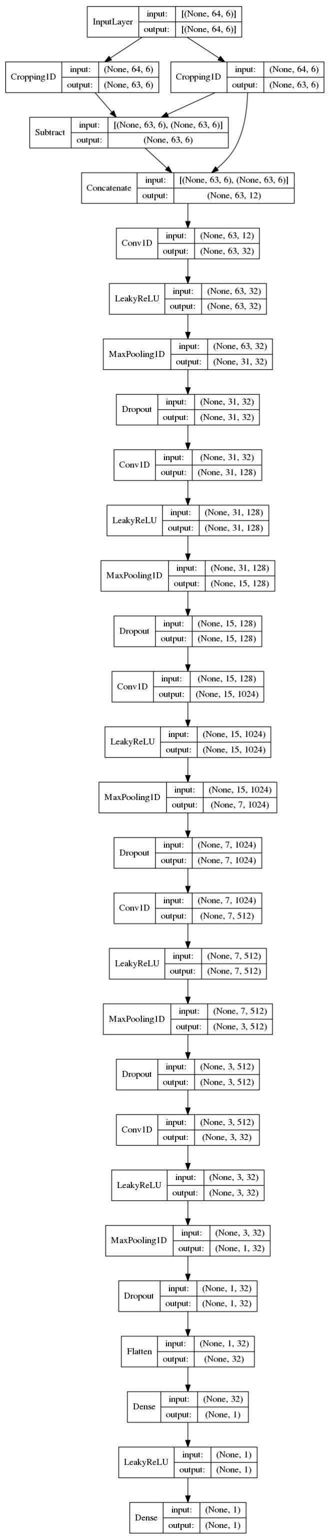


Figure A.1: Proposed Generator Network Diagram

Supplemental Figures



InputLayer input: [(None, 64, 6)] output: [(None, 64, 6)]
Cropping1D input: (None, 64, 6) output: (None, 63, 6)
Cropping1D input: (None, 64, 6) output: (None, 63, 6)
Subtract input: [(None, 63, 6), (None, 63, 6)] output: (None, 63, 6)
Concatenate input: [(None, 63, 6), (None, 63, 6)] output: (None, 63, 12)
Conv1D input: (None, 63, 12) output: (None, 63, 32)
LeakyReLU input: (None, 63, 32) output: (None, 63, 32)
MaxPooling1D input: (None, 63, 32) output: (None, 31, 32)
Dropout input: (None, 31, 32) output: (None, 31, 32)
Conv1D input: (None, 31, 32) output: (None, 31, 128)
LeakyReLU input: (None, 31, 128) output: (None, 31, 128)
MaxPooling1D input: (None, 31, 128) output: (None, 15, 128)
Dropout input: (None, 15, 128) output: (None, 15, 128)
Conv1D input: (None, 15, 128) output: (None, 15, 1024)
LeakyReLU input: (None, 15, 1024) output: (None, 15, 1024)
MaxPooling1D input: (None, 15, 1024) output: (None, 7, 1024)
Dropout input: (None, 7, 1024) output: (None, 7, 1024)
Conv1D input: (None, 7, 1024) output: (None, 7, 512)
LeakyReLU input: (None, 7, 512) output: (None, 7, 512)
MaxPooling1D input: (None, 7, 512) output: (None, 3, 512)
Dropout input: (None, 3, 512) output: (None, 3, 512)
Conv1D input: (None, 3, 512) output: (None, 3, 32)
LeakyReLU input: (None, 3, 32) output: (None, 3, 32)
MaxPooling1D input: (None, 3, 32) output: (None, 1, 32)
Dropout input: (None, 1, 32) output: (None, 1, 32)
Flatten input: (None, 1, 32) output: (None, 32)
Dense input: (None, 32) output: (None, 1)
LeakyReLU input: (None, 1) output: (None, 1)
Dense input: (None, 1) output: (None, 1)

Figure A.2: Proposed Discriminator Network Diagram

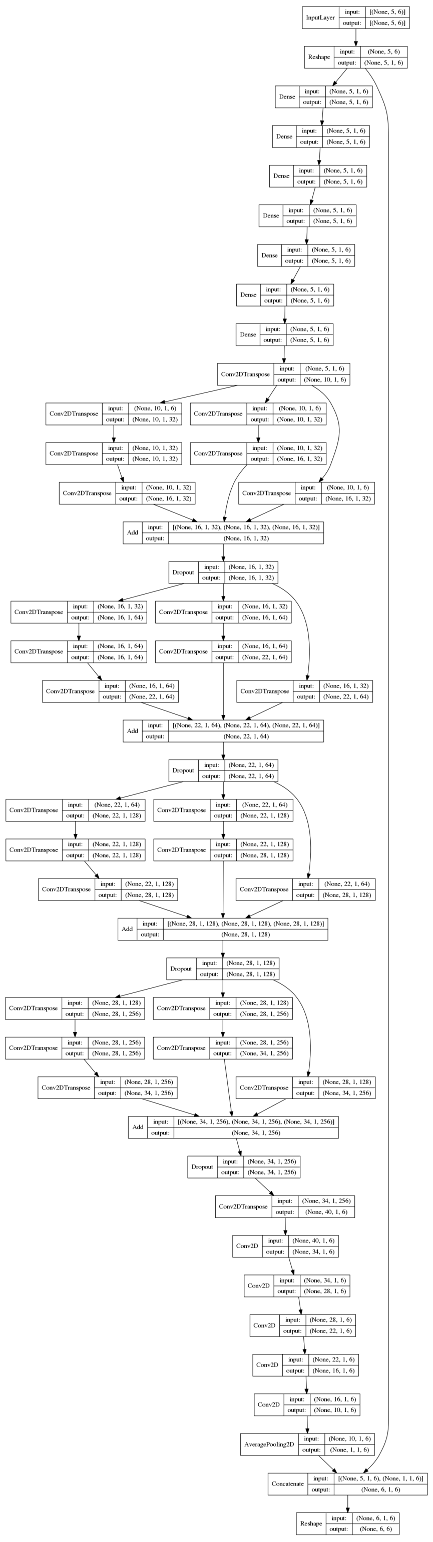

InputLayer | input: [(None, 5, 6)] | output: [(None, 5, 6)]
Reshape | input: (None, 5, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Dense | input: (None, 5, 1, 6) | output: (None, 5, 1, 6)
Conv2DTranspose | input: (None, 5, 1, 6) | output: (None, 10, 1, 6)
Conv2DTranspose | input: (None, 10, 1, 6) | output: (None, 10, 1, 32)
Conv2DTranspose | input: (None, 10, 1, 6) | output: (None, 10, 1, 32)
Conv2DTranspose | input: (None, 10, 1, 32) | output: (None, 10, 1, 32)
Conv2DTranspose | input: (None, 10, 1, 32) | output: (None, 16, 1, 32)
Conv2DTranspose | input: (None, 10, 1, 32) | output: (None, 16, 1, 32)
Conv2DTranspose | input: (None, 10, 1, 6) | output: (None, 16, 1, 32)
Add | input: [(None, 16, 1, 32), (None, 16, 1, 32), (None, 16, 1, 32)] | output: (None, 16, 1, 32)
Dropout | input: (None, 16, 1, 32) | output: (None, 16, 1, 32)
Conv2DTranspose | input: (None, 16, 1, 32) | output: (None, 16, 1, 64)
Conv2DTranspose | input: (None, 16, 1, 32) | output: (None, 16, 1, 64)
Conv2DTranspose | input: (None, 16, 1, 64) | output: (None, 16, 1, 64)
Conv2DTranspose | input: (None, 16, 1, 64) | output: (None, 22, 1, 64)
Conv2DTranspose | input: (None, 16, 1, 64) | output: (None, 22, 1, 64)
Conv2DTranspose | input: (None, 16, 1, 32) | output: (None, 22, 1, 64)
Add | input: [(None, 22, 1, 64), (None, 22, 1, 64), (None, 22, 1, 64)] | output: (None, 22, 1, 64)
Dropout | input: (None, 22, 1, 64) | output: (None, 22, 1, 64)
Conv2DTranspose | input: (None, 22, 1, 64) | output: (None, 22, 1, 128)
Conv2DTranspose | input: (None, 22, 1, 64) | output: (None, 22, 1, 128)
Conv2DTranspose | input: (None, 22, 1, 128) | output: (None, 22, 1, 128)
Conv2DTranspose | input: (None, 22, 1, 128) | output: (None, 28, 1, 128)
Conv2DTranspose | input: (None, 22, 1, 128) | output: (None, 28, 1, 128)
Conv2DTranspose | input: (None, 22, 1, 64) | output: (None, 28, 1, 128)
Add | input: [(None, 28, 1, 128), (None, 28, 1, 128), (None, 28, 1, 128)] | output: (None, 28, 1, 128)
Dropout | input: (None, 28, 1, 128) | output: (None, 28, 1, 128)
Conv2DTranspose | input: (None, 28, 1, 128) | output: (None, 28, 1, 256)
Conv2DTranspose | input: (None, 28, 1, 128) | output: (None, 28, 1, 256)
Conv2DTranspose | input: (None, 28, 1, 256) | output: (None, 28, 1, 256)
Conv2DTranspose | input: (None, 28, 1, 256) | output: (None, 34, 1, 256)
Conv2DTranspose | input: (None, 28, 1, 256) | output: (None, 34, 1, 256)
Conv2DTranspose | input: (None, 28, 1, 128) | output: (None, 34, 1, 256)
Add | input: [(None, 34, 1, 256), (None, 34, 1, 256), (None, 34, 1, 256)] | output: (None, 34, 1, 256)
Dropout | input: (None, 34, 1, 256) | output: (None, 34, 1, 256)
Conv2DTranspose | input: (None, 34, 1, 256) | output: (None, 40, 1, 6)
Conv2D | input: (None, 40, 1, 6) | output: (None, 34, 1, 6)
Conv2D | input: (None, 34, 1, 6) | output: (None, 28, 1, 6)
Conv2D | input: (None, 28, 1, 6) | output: (None, 22, 1, 6)
Conv2D | input: (None, 22, 1, 6) | output: (None, 16, 1, 6)
Conv2D | input: (None, 16, 1, 6) | output: (None, 10, 1, 6)
AveragePooling2D | input: (None, 10, 1, 6) | output: (None, 1, 1, 6)
Concatenate | input: [(None, 5, 1, 6), (None, 1, 1, 6)] | output: (None, 6, 1, 6)
Reshape | input: (None, 6, 1, 6) | output: (None, 6, 6)

Figure A.3: Baseline State to Action Generator Network Diagram

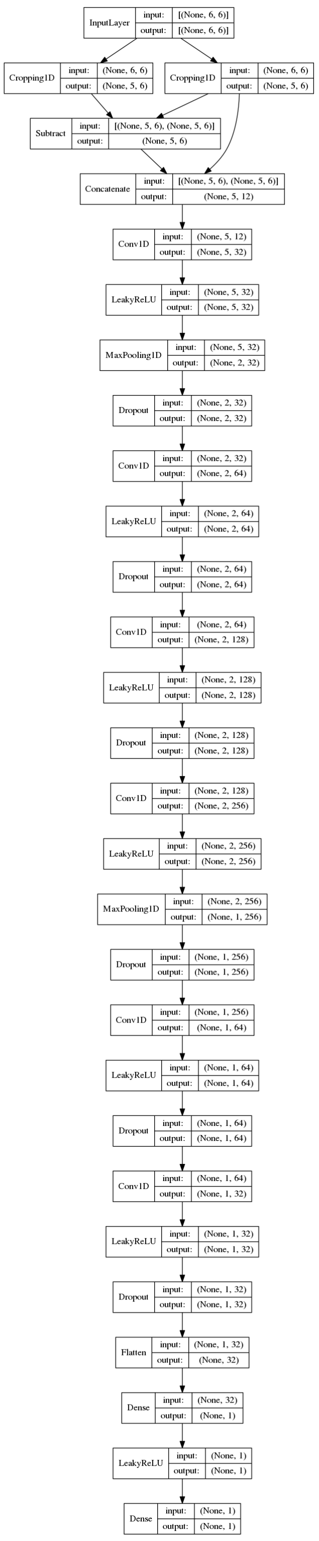

InputLayer input: [(None, 6, 6)] output: [(None, 6, 6)]
Cropping1D input: (None, 6, 6) output: (None, 5, 6)
Cropping1D input: (None, 6, 6) output: (None, 5, 6)
Subtract input: [(None, 5, 6), (None, 5, 6)] output: (None, 5, 6)
Concatenate input: [(None, 5, 6), (None, 5, 6)] output: (None, 5, 12)
Conv1D input: (None, 5, 12) output: (None, 5, 32)
LeakyReLU input: (None, 5, 32) output: (None, 5, 32)
MaxPooling1D input: (None, 5, 32) output: (None, 2, 32)
Dropout input: (None, 2, 32) output: (None, 2, 32)
Conv1D input: (None, 2, 32) output: (None, 2, 64)
LeakyReLU input: (None, 2, 64) output: (None, 2, 64)
Dropout input: (None, 2, 64) output: (None, 2, 64)
Conv1D input: (None, 2, 64) output: (None, 2, 128)
LeakyReLU input: (None, 2, 128) output: (None, 2, 128)
Dropout input: (None, 2, 128) output: (None, 2, 128)
Conv1D input: (None, 2, 128) output: (None, 2, 256)
LeakyReLU input: (None, 2, 256) output: (None, 2, 256)
MaxPooling1D input: (None, 2, 256) output: (None, 1, 256)
Dropout input: (None, 1, 256) output: (None, 1, 256)
Conv1D input: (None, 1, 256) output: (None, 1, 64)
LeakyReLU input: (None, 1, 64) output: (None, 1, 64)
Dropout input: (None, 1, 64) output: (None, 1, 64)
Conv1D input: (None, 1, 64) output: (None, 1, 32)
LeakyReLU input: (None, 1, 32) output: (None, 1, 32)
Dropout input: (None, 1, 32) output: (None, 1, 32)
Flatten input: (None, 1, 32) output: (None, 32)
Dense input: (None, 32) output: (None, 1)
LeakyReLU input: (None, 1) output: (None, 1)
Dense input: (None, 1) output: (None, 1)

Figure A.4: Baseline State to Action Discriminator Network Diagram

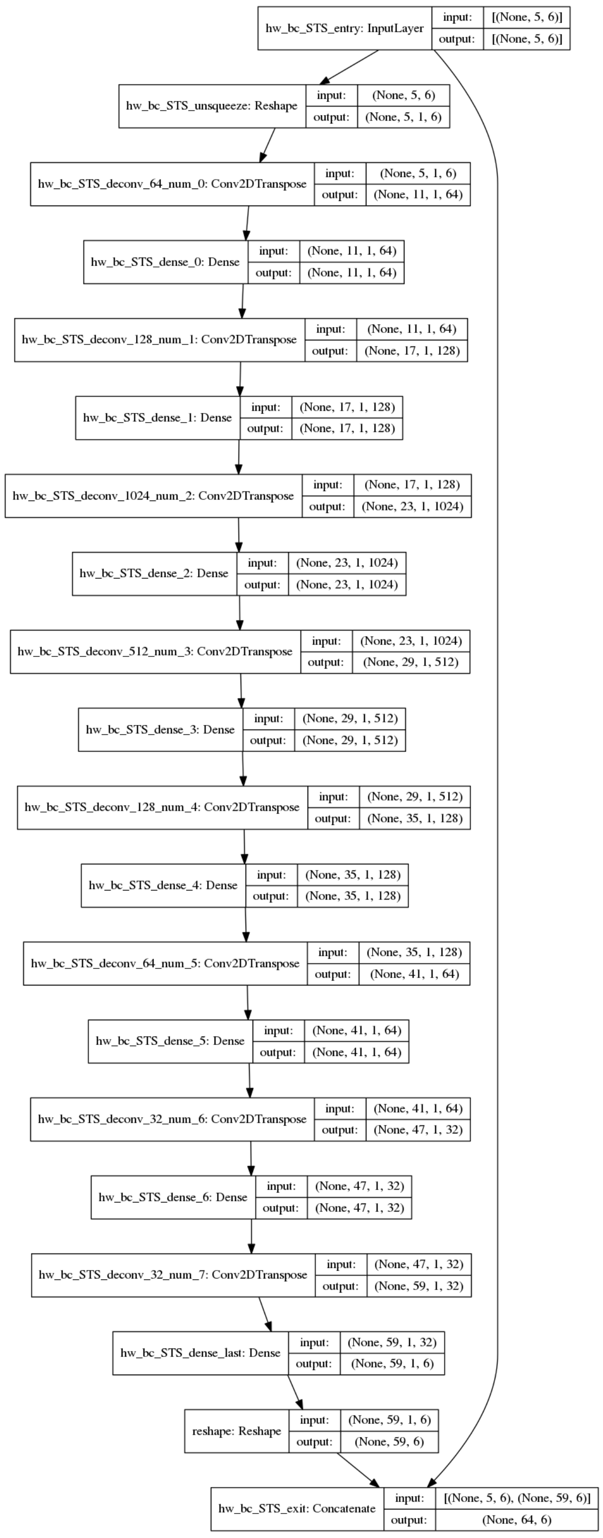

hw_bc_STS_entry: InputLayer
input: [(None, 5, 6)]
output: [(None, 5, 6)]
hw_bc_STS_unsqueeze: Reshape
input: (None, 5, 6)
output: (None, 5, 1, 6)
hw_bc_STS_deconv_64_num_0: Conv2DTranspose
input: (None, 5, 1, 6)
output: (None, 11, 1, 64)
hw_bc_STS_dense_0: Dense
input: (None, 11, 1, 64)
output: (None, 11, 1, 64)
hw_bc_STS_deconv_128_num_1: Conv2DTranspose
input: (None, 11, 1, 64)
output: (None, 17, 1, 128)
hw_bc_STS_dense_1: Dense
input: (None, 17, 1, 128)
output: (None, 17, 1, 128)
hw_bc_STS_deconv_1024_num_2: Conv2DTranspose
input: (None, 17, 1, 128)
output: (None, 23, 1, 1024)
hw_bc_STS_dense_2: Dense
input: (None, 23, 1, 1024)
output: (None, 23, 1, 1024)
hw_bc_STS_deconv_512_num_3: Conv2DTranspose
input: (None, 23, 1, 1024)
output: (None, 29, 1, 512)
hw_bc_STS_dense_3: Dense
input: (None, 29, 1, 512)
output: (None, 29, 1, 512)
hw_bc_STS_deconv_128_num_4: Conv2DTranspose
input: (None, 29, 1, 512)
output: (None, 35, 1, 128)
hw_bc_STS_dense_4: Dense
input: (None, 35, 1, 128)
output: (None, 35, 1, 128)
hw_bc_STS_deconv_64_num_5: Conv2DTranspose
input: (None, 35, 1, 128)
output: (None, 41, 1, 64)
hw_bc_STS_dense_5: Dense
input: (None, 41, 1, 64)
output: (None, 41, 1, 64)
hw_bc_STS_deconv_32_num_6: Conv2DTranspose
input: (None, 41, 1, 64)
output: (None, 47, 1, 32)
hw_bc_STS_dense_6: Dense
input: (None, 47, 1, 32)
output: (None, 47, 1, 32)
hw_bc_STS_deconv_32_num_7: Conv2DTranspose
input: (None, 47, 1, 32)
output: (None, 59, 1, 32)
hw_bc_STS_dense_last: Dense
input: (None, 59, 1, 32)
output: (None, 59, 1, 6)
reshape: Reshape
input: (None, 59, 1, 6)
output: (None, 59, 6)
hw_bc_STS_exit: Concatenate
input: [(None, 5, 6), (None, 59, 6)]
output: (None, 64, 6)

Figure A.5: Baseline State to Action Behavior Cloning Network Diagram

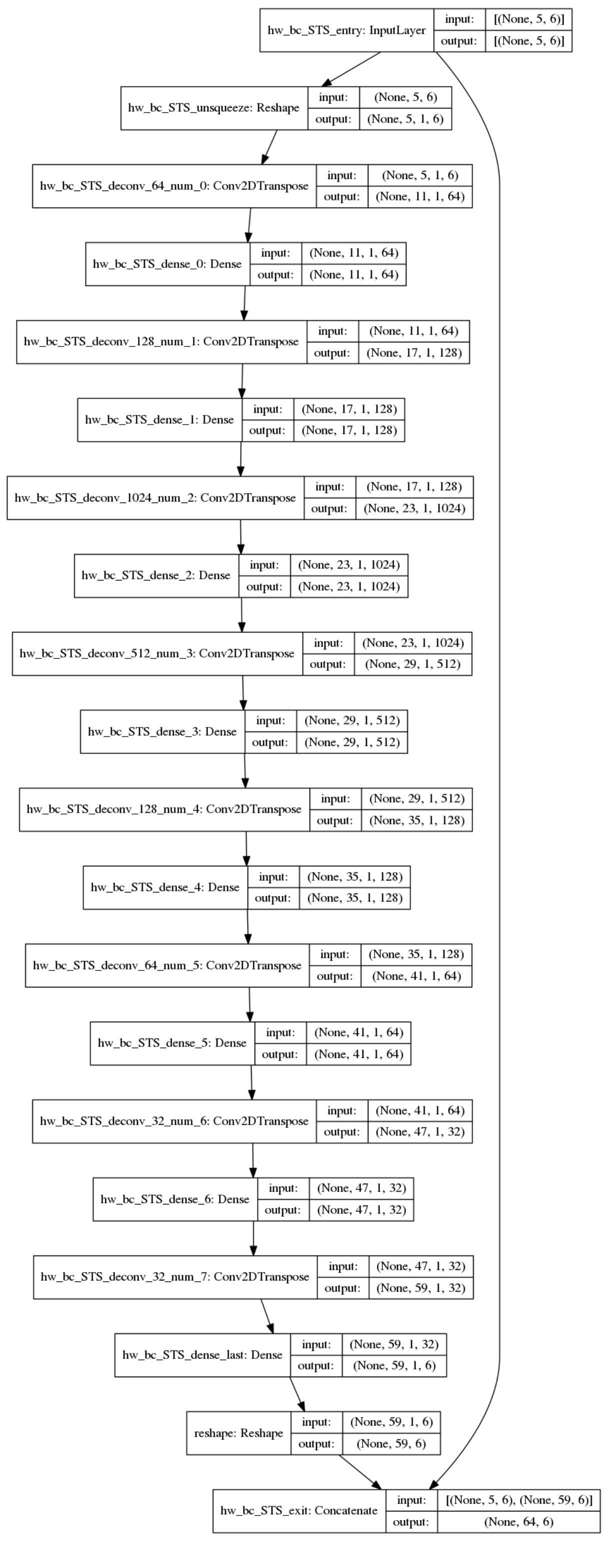

hw_bc_STS_entry: InputLayer | input: [(None, 5, 6)] | output: [(None, 5, 6)]
hw_bc_STS_unsqueeze: Reshape | input: (None, 5, 6) | output: (None, 5, 1, 6)
hw_bc_STS_deconv_64_num_0: Conv2DTranspose | input: (None, 5, 1, 6) | output: (None, 11, 1, 64)
hw_bc_STS_dense_0: Dense | input: (None, 11, 1, 64) | output: (None, 11, 1, 64)
hw_bc_STS_deconv_128_num_1: Conv2DTranspose | input: (None, 11, 1, 64) | output: (None, 17, 1, 128)
hw_bc_STS_dense_1: Dense | input: (None, 17, 1, 128) | output: (None, 17, 1, 128)
hw_bc_STS_deconv_1024_num_2: Conv2DTranspose | input: (None, 17, 1, 128) | output: (None, 23, 1, 1024)
hw_bc_STS_dense_2: Dense | input: (None, 23, 1, 1024) | output: (None, 23, 1, 1024)
hw_bc_STS_deconv_512_num_3: Conv2DTranspose | input: (None, 23, 1, 1024) | output: (None, 29, 1, 512)
hw_bc_STS_dense_3: Dense | input: (None, 29, 1, 512) | output: (None, 29, 1, 512)
hw_bc_STS_deconv_128_num_4: Conv2DTranspose | input: (None, 29, 1, 512) | output: (None, 35, 1, 128)
hw_bc_STS_dense_4: Dense | input: (None, 35, 1, 128) | output: (None, 35, 1, 128)
hw_bc_STS_deconv_64_num_5: Conv2DTranspose | input: (None, 35, 1, 128) | output: (None, 41, 1, 64)
hw_bc_STS_dense_5: Dense | input: (None, 41, 1, 64) | output: (None, 41, 1, 64)
hw_bc_STS_deconv_32_num_6: Conv2DTranspose | input: (None, 41, 1, 64) | output: (None, 47, 1, 32)
hw_bc_STS_dense_6: Dense | input: (None, 47, 1, 32) | output: (None, 47, 1, 32)
hw_bc_STS_deconv_32_num_7: Conv2DTranspose | input: (None, 47, 1, 32) | output: (None, 59, 1, 32)
hw_bc_STS_dense_last: Dense | input: (None, 59, 1, 32) | output: (None, 59, 1, 6)
reshape: Reshape | input: (None, 59, 1, 6) | output: (None, 59, 6)
hw_bc_STS_exit: Concatenate | input: [(None, 5, 6), (None, 59, 6)] | output: (None, 64, 6)

Figure A.6: Baseline Full Trajectory Behavior Cloning Network Diagram

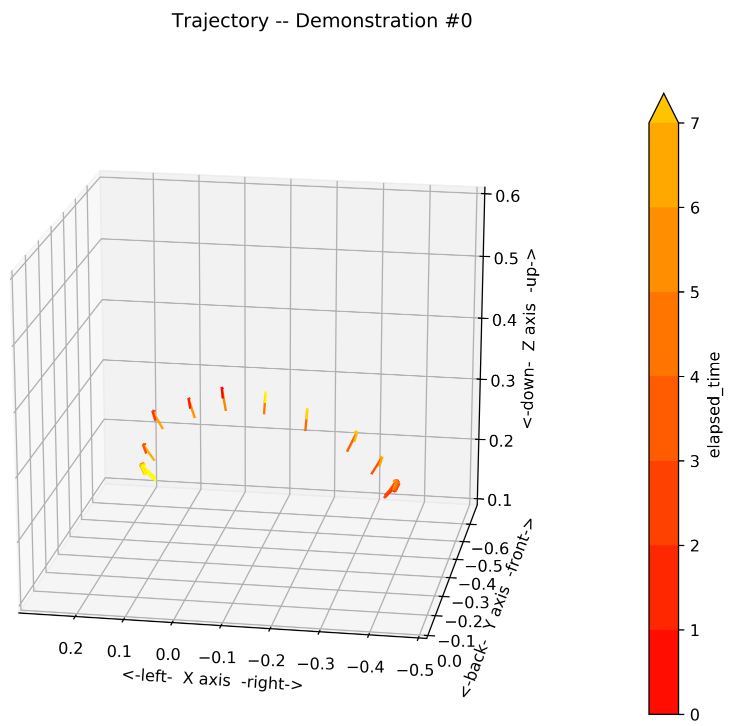


Figure A.7: Pick-and-Place Demonstration Trajectory #0

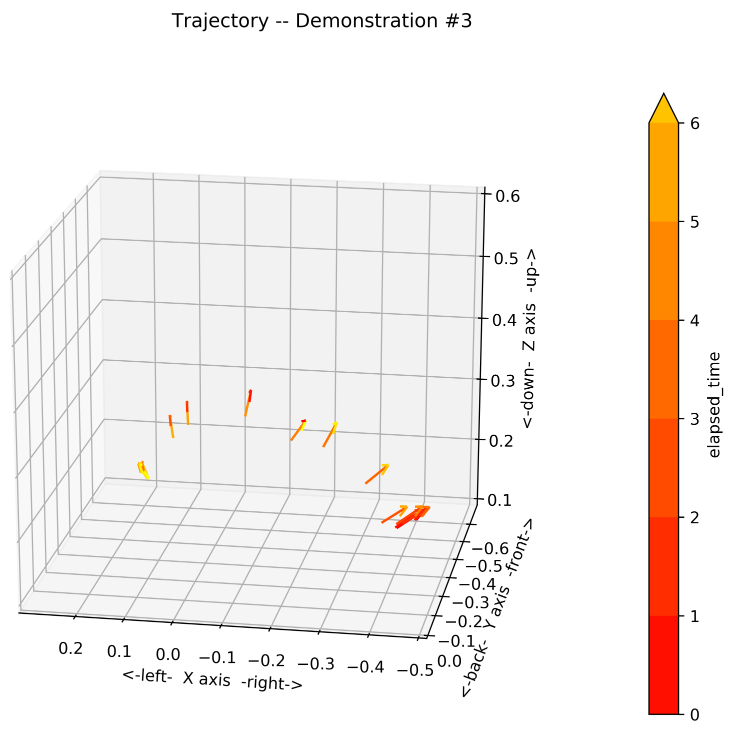


Figure A.8: Pick-and-Place Demonstration Trajectory #3

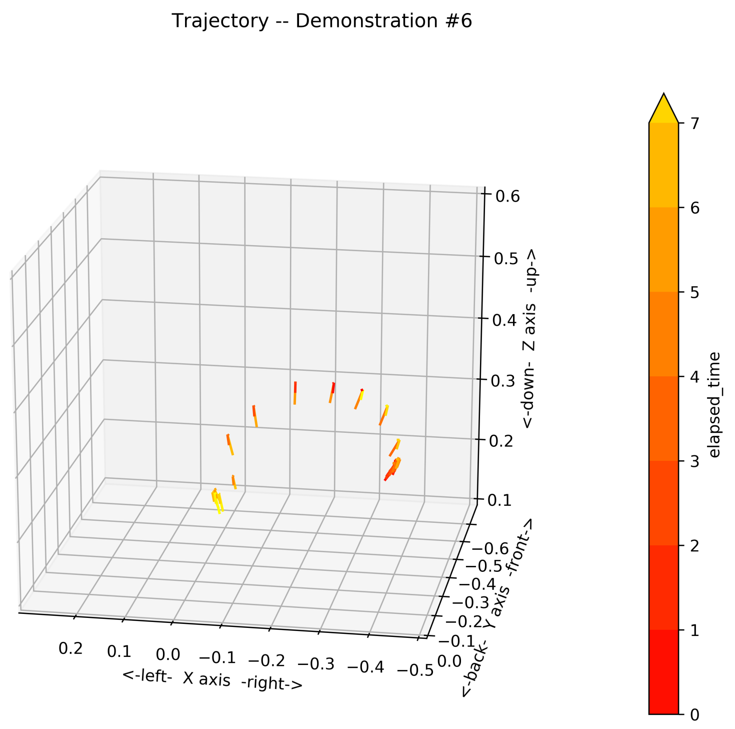


Figure A.9: Pick-and-Place Demonstration Trajectory #6

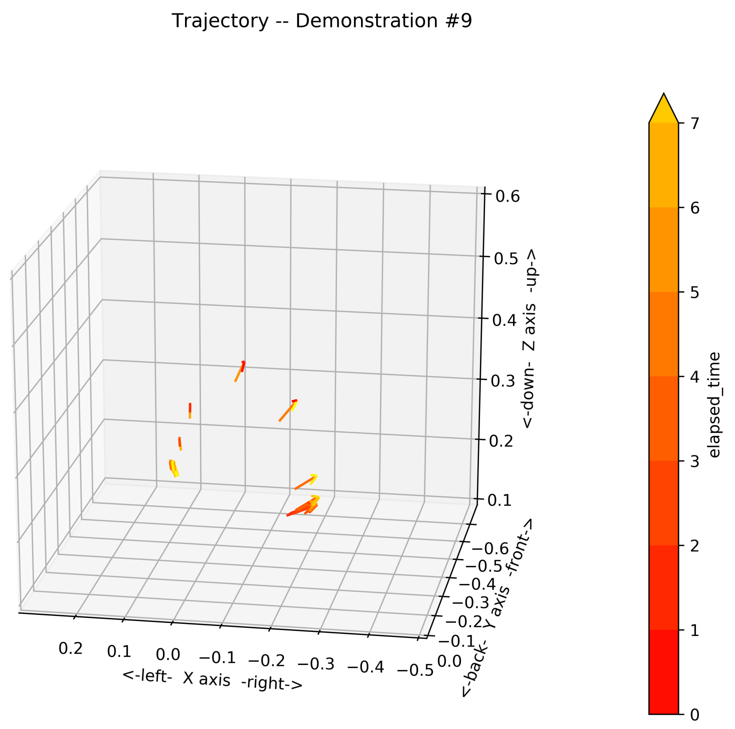


Figure A.10: Pick-and-Place Demonstration Trajectory #9

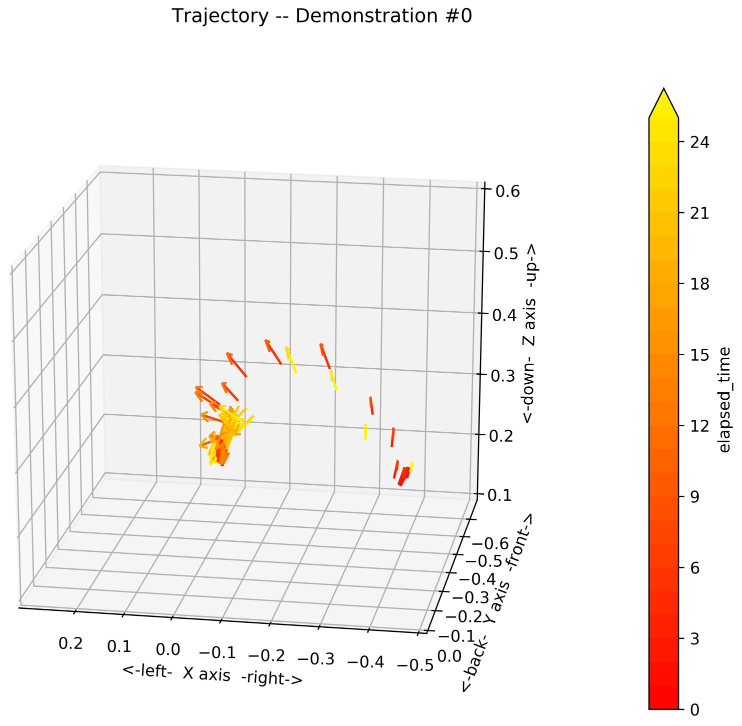


Figure A.11: Pouring Demonstration Trajectory #0

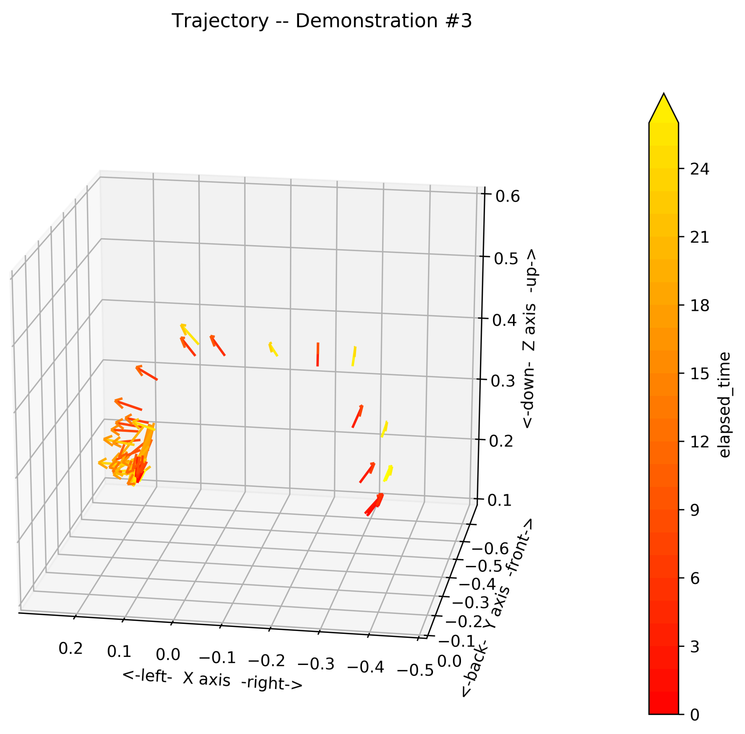


Figure A.12: Pouring Demonstration Trajectory #3

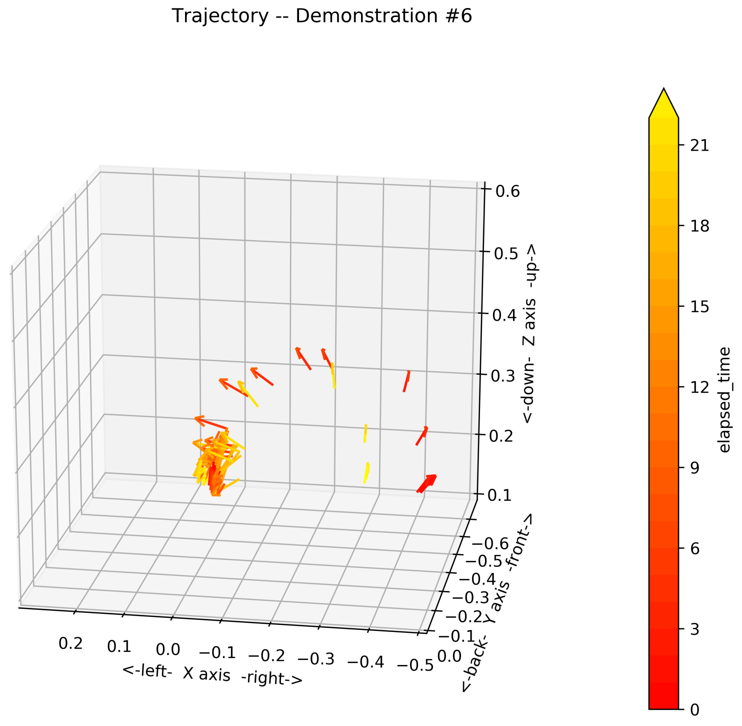


Figure A.13: Pouring Demonstration Trajectory #6

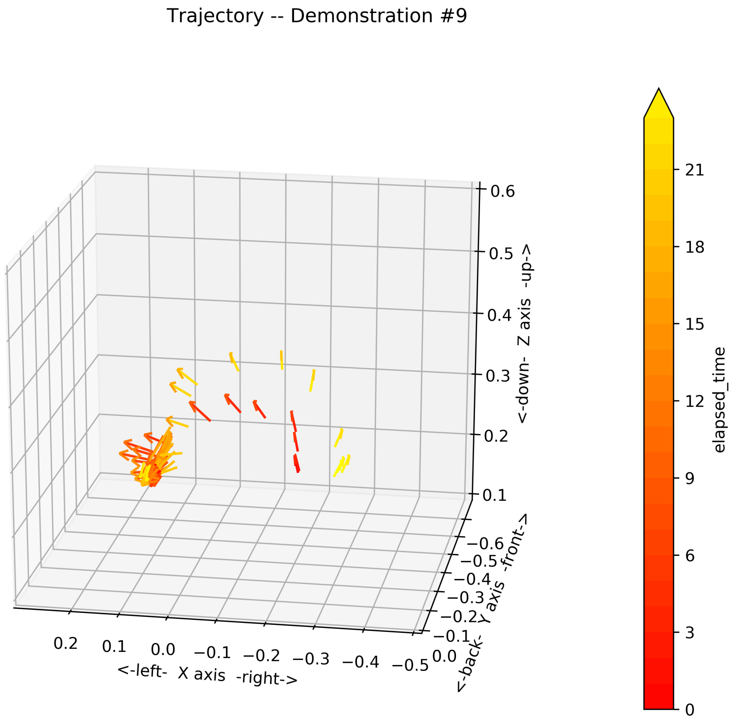


Figure A.14: Pouring Demonstration Trajectory #9