Learning from Demonstration for Waste Picking*

Qiu Nuofan[1#], Xiao Yang[1#], Hu Bowen[1#], Liu Xin[1#], Li Yifei[1#], Xu Ronghan[1#], Dong Yujian[1#], Xiang Yanzhen[1#], Guo Shangkun[1#], Dr. Wan Fang[2#], Prof. Song Chaoyang[2#]

***Abstract*— In the context of robotics and automation, learning from demonstration(LfD) is the paradigm in which robots acquire new skills by learning to imitate an expert. In this article, we will show our work to attempt Learning from demonstration by passive observation, including data collection, trajectory reproducing in both real and simulation environments and using ML-Agent to build the framework of learning.**

***Index Terms*— learning from demonstration(LfD), passive observation, simulation**

CONTENTS

I. INTRODUCTION

Decision making in robotics often involves computing an optimal action for a given state, where the space of actions under consideration can potentially be large and state dependent. It is also very common for humans to make decisions for different situations and humans are good at making decisions. So learning from human demonstrations is a powerful approach to reduce the space of actions to help robotics to make decisions.

In the context of robotics and automation, learning from demonstration is the paradigm in which robots acquire new skills by learning to imitate an expert. According to the technique by which demonstrations will be performed, learning from demonstration can be divided into three categories: kinesthetic teaching, teleoperation,and passive observation[1]. In this project, we chose to use passive observation, which means that we performed tasks using our own body with a pair of modified kitchen tongs. Learning from demonstration by passive observation is easy for demonstrator, requiring almost no training to perform.

In this report, we will introduce our work about learning from demonstration for generating trajectory to approach a target, including data collection, reproducing trajectory in both real and simulation environment.

*This work was supported by Southern University of Science and Technology and AncoraSpring Inc. for the ME336 Collaborative Robot Learning (me336.ancorasir.com) under the supervision of Prof. Song Chaoyang songcy@ieee.org

#Equal contribution as co-first authors.

[1]All authors[1#] are with Department of Mechanical and Energy Engineering, Southern University of Science and Technology, Shenzhen, Guangdong 518055, China. 11713013, 11811019, 11811406, 11710913, 11812814, 11810502, 11811403, 11811417, 11810904@mail.sustech.edu.cn

[2]Author Dr.Wan Fang is with SUSTech Institute of Robotics, Southern University of Science and Technology, Shenzhen, Guangdong 518055, China. wanf@mail.sustech.edu.cn

II. System Design

A. Data Collection

1) Platform introduction : In this project, we directly use Wang Haokun 's data collection platform, DeepClaw 2.0. It consists of a visual sensor, Intel RealSense D435i and a working platform. The camera is fixed on a frame above the working platform. The working platform contains a calibration grid for localization, a pair of modified kitchen tongs for demonstrations and objects to be manipulated. In this platform a human teacher operates kitchen tongs to perform object manipulation. On each tip of tong, there is a soft finger network attached to it, which can be used to detect the force application during grabbing by visual method, while in this course we didn't use this function. Fig. 1

2) Collecting data: In working, Intel RealSense D435i will take color and depth images at 30fps. These raw sensor data are sequential images recorded with times. Everything in the working platform is marked by AprilTags. By analysing each recorded images with the AprilTag detector in ViSP, we can estimate their 6D positions. These pose data will be further used to extract action by taking their time derivatives. Using this kind of data, we can simulate the movement of each one in the Unity and direct the robot arm to grab the bos in the reality. The final goal of our group is to use the data to do the deep learning and direct the robot arm to move more like human. During this course, we use the camera to video each teammates grabbing the box with marker in 16 different positions. The working platform is divided into 4×4 blocks. Every position randomly distributes on these blocks and the object to be grabbed also has random pose. These preparations are to imitate the unpredictable conditions in real work. When the demonstration begins, the experimenter moves the tongs starting from an initial point into the working platform and slowly forward the object until grabbing it. Then one collecting period ends. All the data will be collected and position of whole tongs will be replaced by the center point calculating from the position of two sides of tongs. Fig. 2

3) Shortcoming: One limitation is the performance of the camera used (Intel RealSense D435i). During the process of collecting the data, the frame has high delay. For the better recording, systems such as Photoneo MotionX could be a potential alternative. Future work of this study could be aimed at a more comprehensive system design towards a low-cost, open-sourced platform for robot manipulation learning research, recommended by Wang Haokun. What's more, recent research by the Berkley Open Arm project proposed a promising design paradigm in this direction for robot hardware that "enables useful automation in unconstrained real-world human environments at low cost."

B. Data processing and Trajectory Reproducing

1) Data processing: **Preliminary analysis of data:** The collected data file contains the time stamp, each frame of RGB-D image, 6D posture information of the AprilTag on the gripper fingers and objects, and some information calculated from the data set, like the distance between the left and right gripper fingers.The part of data we will mainly use in trajectory reproduction is the 6D posture information of the center of the gripper tool during the entire grasping process. However, these data sets have some problems and cannot be used directly, so we need data cleaning to make these data usable. If our final project has a longer period, we will make full use of the data collected. Through RGB-D images and finger distance data, it's possible to control the robotic arm to throw the object to a certain area with it's gripper opening for a certain distance, which can be realized through machine learning.

Data cleaning: There are mainly two problems with the data sets we collected. The first is that some of the data collecting processes have already started before the gripper enters the view field of the RealSense camera yet, which results in the 6D gripper posture information corresponding to some early moments in the data set being empty. In order to ensure that the trajectory of the robotic arm is continuous, we decided to delete those lines whose 6D posture information of the tool center is empty in the data set which means the kitchen tongs haven't started the task.

The second is that while we are collecting the data, when the gripper moves to a position close to the vertical state, the QR code on the gripper fingers will have the problem of coordinate axis recognition error affected by indoor lighting. As a result, the three-dimensional posture information of the gripper in the data will have some sign errors. At the same time, when collecting data, due to the low frame rate of our depth camera, we have to slow down the speed of hand movement in order to ensure that the point interval is as small as possible, which also caused a sudden change in the collected jaw posture information. Fig. 18

We set a threshold artificially according to the actual situation, and deal with the points where the change exceeds the threshold. The left picture above is the scatter plot before processing, the right picture is the scatter plot after processing, and the following is the data after processing by 5th degree polynomial fitting. Fig. 4 The data after these two steps can be used to reproduce the trajectory.

Visualization of data collected: In order to observe the effect of data cleaning, we decided to visualize the data using the plotting tools in matplot library of Python. Fig. 5

Advantages of data: By comparing the visual image of the collected data with the trajectory video during the collection, we found that their feature similarity is relatively high, which means that the data we collected can well restore and represent the actual movement posture of the person while picking the target object using gripper.

Shortcomings of data: The density of the data points we collected is low, and the distance between two points is a little far.The trajectories fluctuate relatively greatly.

Data points are not uniform, the middle part is sparse and both ends are dense. Because in the whole process of gripping the target, the speed is slow at the beginning of moving the gripper and finally gripping the object, while the speed is faster when the middle gripper moves towards the target.

The initial position of the gripper we specified starts from the desktop plane and ends when the object is clamped. This means that the end point of the trajectory is usually higher than the starting point. In the actual robotic arm trajectory reproduction, the position of robotic arm will be close to the desk plane, which might lead to collisions with the desk while the robotic arm is moving.

Thinking and solutions: If we were to do the project again, we would replace the RealSense D435i camera with one with higher FPS, so that we can grip the target object with a more natural posture and speed. And we would specify a higher starting position to prevent the robotic arm from colliding with the desktop during the process of reproducing the trajectory. In this experiment, in order to overcome the above shortcomings in the data as much as possible, we decided to use the combination of polynomial fitting and interpolation method for trajectory planning to make the trajectory of robotic arm smoother and denser, which will be talked about in the trajectory planning part in detail.

2) The coordinate transformation from the data collection platform to the robotic arm: In order to make the reproduced trajectory and the gripped trajectory as similar as possible, we hope that the ratio of the two trajectories is equal. So we set a ruler on the data acquisition platform and compare it with the actual data to determine the scaling factor of the coordinate transformation matrix. And also, we set the first point of each trajectory as the origin point. To ensure the robotic arm grip the target on the horizontal tabletop smoothly, and make the entire trajectory in the middle part of the base under the condition that the robot arm can move normally, we offset the data coordinates according to the actual situation.

3) Trajectory planning: In order to realize the multi-point path planning of the robotic arm, we need to ensure that the motion trajectory is smooth enough and write the trajectory in the form of a parametric equation to facilitate the subsequent programming of the robotic arm motion control interface.

We first use the $polyfit$ function in python's $numpy$ toolkit to fit the 6D coordinates of the collected scattered points by polynomial. In the specific operation, we fit the xyz three-axis coordinates of the collected data to the corresponding picture frame number n, and then use the $polyval$ function of the $numpy$ toolkit to output the corresponding equation, and integrate the results of the three-axis fitting to get The parameter equation corresponding to the frame number n (that is, time) of the entire motion track. And later, the parameter equation is derived to plan the speed and acceleration curve of the reproduced trajectory of the manipulator.

We use the $matplot$ toolkit again to visualize the trajectory. By connecting the scatter plots of the data in sequence and plot the fitted curve in the same picture, we analyze the effect of curve fitting by comparison. In the whole curve fitting process, we tried to use the 2nd- order polynomial to the 6th-order polynomial for fitting. Through comparison and trade-offs, we believe that the 5th-order polynomial has a better fit for 3D position. The fitting of polynomials below 5th-order will result in lacking some information in the collected data set, and some characteristics of the motion of human hand will miss. Those polynomials above 5th-order will lead to the entire trajectory to be not smooth enough, which is not only due to slowing down the movement while collecting data, but also not good for the motors of the robotic arm. In the final analysis, the fifth-order polynomial trajectory fitting is a balance of our evaluation of the performance of the data collection platform and the actual operation of the robotic arm system as shown in the following figures. Fig. 6

Regarding how to implement our trajectory planning program on the control code of the robotic arm, we read some control example C/C++ programs in Franka FCI library, and also inquired about some related robotic arm control blogs, but we found it beyond our ability. After that, we asked our teaching assistant to help write the interface of Franka's multi-point planning. In actual use, the speed and acceleration are not continuous errors. Under the instruction of the teaching assistant, we thought that the speed in the trajectory planning might exceed the upper limit of the speed of the robotic arm, so we tried to make the entire robotic arm finish the trajectory longer by changing the timestamp frequency corresponding to the interpolation, thereby reducing the speed of robotic arm. However, there are still discontinuous speed and acceleration errors. Due to the time relationship, we gave up and used the $move\_p()$ single-point motion function to test our trajectory reproduction.

C. Construction of simulation environment

1) Simulation software——Unity: Unity is a real-time 3D interactive content creation and operation platform with rich community resources. The simulation of our project requires 3D rendering, model motion control, model and trajectory-based learning process, and the Unity community provides rich resources for this, so we use Unity software to build the simulation and learning environment.

2) Simulation model construction: Robotic arm model We first need to complete the reproduction of the Franka manipulator in the simulation environment. We hope to construct it through the connection of joints, and control its movement by setting the joint angle or solving inverse kinematics. Fig. 7

3) Simulation model construction: Conveyor model

We need to add collision and friction properties to the conveyor belt and apply a translation script to it, which can

	Online Reproduction	Offline Reproduction
Pros	Easy to ensure real-time control	Data exchange between real and simulation can be easily realized by file IO operations
Cons	Hard to implement communication between real and simulation	Hard to ensure real-time control

TABLE I: Pros Cons of Online Reproduction and Offline Reproduction

bring the animal block to move when its surface is in contact with the block. Fig. 8

4) Simulation model construction: Other models Models used to support and enrich the environment, such as robotic arm support bases, tables, shelves, cameras, etc. The overall environment is shown in the figure below.

D. Reproduction and Exploration in Simulation

Basically there are three tasks to conduct in simulation:

Task 1: To realize control of the arm in simulation in joint space: To input a certain trajectory containing position and velocity information at every time step, and find out how accurate can the arm track the trajectory.

Task 2: To realize control of the arm in simulation in workspace: A certain trajectory in workspace is given, and joint positions at every time step are calculated using Inverse Kinematics(IK) of the arm. The calculated joint positions are input to the arm.

Task 3: To find a certain area in which the IK results are relatively accurate, and use this area as the target area to which the sampled trajectories of human are transformed (This transformation should be improved compared to previous coordinate transformation because avoidance of large IK errors is taken into consideration), and thus finding a proper way to transform the sampled data into the data of trajectory which the robot arm should traverse.

Prior to all of these, we should choose between online reproduction and offline reproduction. Both of them have their own pros and cons(Table I). Considering our practical situation, offline production is chosen.

1) Task 1: Control of Arm in Joint Space: A test trajectory is input into the joint space of the robot arm: $q_2 = q_4 = 90 - 45cos(0.8count)$ where q_2 and q_4 are respectively joint positions of joint 2 and joint 4, and $count$ is the number of counts in the program loop, which is positively proportional to time.

Note that the derivative of the trajectory above, which are the joint velocities of the arm are:

$$\dot{q_2} = \dot{q_4} = 36sin(0.8count)$$

where $\dot{q_2}$ and $\dot{q_4}$ are respectively joint velocities of joint 2 and joint 4. So not only the joint positions but also joint velocities are both in the shape of the sine wave. If the arm can nicely track such a trajectory, it means that the position control and velocity control of the arm in simulation in joint space can be realized.

	Write IK code from scratch	Using IK function provided by the simulation Franka model	Using MATLAB API of IK of Franka arm
Pros	Easy to extend to robotic arms of different kinds	The result is more accurate	Easy to realize offline control
Cons	Difficult and too time-consuming to implement	Hard to realize offline IK solving	The result is less accurate because of the shortcoming of the numerical optimal method applied in the API

TABLE II

Setting the maximum joint velocity of the arm to 40 rad/s and the updating frequency to 100 Hz in Unity and running the simulation of this trajectory, the result is obtained as is shown in Fig. 10. The figure shows that the actual trajectory is quite close to the expected trajectory except for some special cases(e.g. When the simulation has not started for long, the arm needs time to run to the expected position or velocity. And joint 4 reaches its joint limits when its position is close to the lowest of the theoretical position, and it cannot exceed its limit.) This indicates that the arm simulation behaves well in position and velocity control in joint space.

2) Task 2: Control of Arm in Workspace: This task involves the IK solution of the arm. Since each joint of the arm has its own position limits, the optimization algorithm with constraints are applied to find the optiamal joint position solution at every time step:

$$\begin{aligned} &min\ ||\mathbf{x} - \hat{\mathbf{x}}|| \\ &s.t.\ \mathbf{x} = f(\mathbf{q}) \\ &\quad q_{imin} \leq q_i \leq q_{imax}, i = 1, 2, ..., 7 \end{aligned}$$

where

$$\mathbf{x} = [x, y, z, yaw, pit, rol]^{\mathrm{T}}$$

is the actual position and attitude of the end effector. And $\hat{\mathbf{x}}$ is the theoretical position and attitude. Since the end effector seldom rotates in the process, for simplification its 3D attitude is set to fix ($yaw = 180^o, pit = -30^o, rol = 0^o$). The joint position at each time step is calculated and then stored into a file.

To solve the IK problems, basically there are 3 choices, and each choice has its own pros and cons(Table II).Considering our practical situation, we choose to use MATAB API of IK.

For test, we input a certain trajectory in workspace into IK to calculate the trajectory in joint space and save the results in the file. Then the results are input into Unity simulation, and the actual end effector position of the arm in simulation are recorded. Fig. 11 left part shows the expected and actual trajectory, and Fig. 11 right part shows the error of the end effector's cartesian coordinate at every time step.

It is shown that the maximum tracking error will be no more than 0.1m, indicating the tracking of such a trajectory

is relatively good. However, not every trajectory has such low tracking error. In task 3 the area where tracking error is relatively small will be found out.

3) Task 3: Finding an Improved Relationship Between Testee's Workspace in Data Sampling and Workspace of Robotic Arm, Using Results of Simulation: In order to find out an area where the IK error is relatively small, a series of points in the arm's workspace are generated (points seperate each other 0.1m away, either in x, y or z direction.). Using IK and input its results into Unity, the error at each point is calculated, which is depicted in Fig.12

It is shown that in the area not far above the conveyor and moderately far away from the arm's base the errors are relatively small, while in the area near the margin of the workspace or very close to the arm's base the errors are relatively large. Based on this, a cuboid area which can contain the normalized sampled trajectory and has relatively low IK error can be found.

To find this area, the mean trajectories of each person picking the object at the same position are calculated (since totally there are 16 different positions of the target object, there are totally 16 mean trajectories). This is shown in Fig.13.Also, the standard derivations on each direction are calculated (Before calculating standard derivation, the trajectories are all trimmed to the same starting point to eliminate the influence of discrepancy in starting points. Also, splining is applied to ensure the moving speeds of different testees are all the same). The maximum standard derivations for each of 16 target positions are shown in Fig.Fig.14 It is shown that the maximum standard derivations on x-axis and z-axis are all very small, while those on y-axis vary much. To further find out some patterns of maximum standard derivations on y-axis, the bar chart(Fig.18) is plotted((1,1) is position 1, (1,2) is position 2, (1,3) is position 3, ..., (2,1) is position 5, ..., (4,4) is position 16).

It is shown that the trajectories targeting at position 1-4 and 13-16 have relatively low standard derivation, while those targeting at position 5-12 have relatively high one. This is consistent with common sense: In picking based on human's own intuition and when the target is too close or too far, usually there are not many only ways to reach the target object. But when the target is neither very close nor far, usually there are multiple ways to reach the target.

Referencing to Fig.15 and taking the standard derivations into account, finally a cuboid area whose length and width are both 0.6m and whose height is 0.05m is chosen as the target working area of the robotic arm to which the sampled trajectories are transformed to. By referencing to Fig.15, the target working area of the robotic arm is finally chose: [0.1, 0.7] in x, [-0.7, -0.1] in y and [0.1, 0.15] in z. This is the area with relatively small error in IK solution which contains almost all of the sampled trajectories considering mean and standard derivation.

Plus, to tackle the problem of different heights of the testees, the positions of the sampled trajectories in z-direction should be normalized into the chosen area. Thus, either the sampled data of a very tall adult testee or a young kid testee can be properly mapped to the chosen working area of the robotic arm. The mapping can be expressed as:

$$
\begin{aligned}
x_{new} &= x - \frac{0.7 + 0.1}{2} = x - 0.4 \\
y_{new} &= y - \frac{-0.7 + (-0.1)}{2} = y + 0.4 \\
z_{new} &= (z - \frac{0.1 + 0.15}{2}) \frac{0.15 - 0.1}{z_{max} - z_{min}} \\
&= (z - 0.125) \frac{0.05}{z_{max} - z_{min}}
\end{aligned}
$$

E. Imitation Learning

1) Principle: It is often more intuitive to simply demonstrate the behavior we want an agent to perform, rather than attempting to have it learn via trial-and-error methods. Imitation learning can either be used alone or in conjunction with reinforcement learning. If used alone it can provide a mechanism for learning a specific type of behavior (i.e. a specific style of solving the task). If used in conjunction with reinforcement learning it can dramatically reduce the time the agent takes to solve the environment.

GAIL (Generative Adversarial Imitation Learning): GAIL, or Generative Adversarial Imitation Learning, uses an adversarial approach to reward your Agent for behaving similar to a set of demonstrations. GAIL can be used with or without environment rewards, and works well when there are a limited number of demonstrations. In this framework, a second neural network, the discriminator, is taught to distinguish whether an observation/action is from a demonstration or produced by the agent. This discriminator can then examine a new observation/action and provide it a reward based on how close it believes this new observation/action is to the provided demonstrations.

At each training step, the agent tries to learn how to maximize this reward. Then, the discriminator is trained to better distinguish between demonstrations and agent state/actions. In this way, while the agent gets better and better at mimicking the demonstrations, the discriminator keeps getting stricter and stricter and the agent must try harder to "fool" it.

This approach learns a policy that produces states and actions similar to the demonstrations, requiring fewer demonstrations than direct cloning of the actions. In addition to learning purely from demonstrations, the GAIL reward signal can be mixed with an extrinsic reward signal to guide the learning process.

Behavioral Cloning (BC): BC trains the Agent's policy to exactly mimic the actions shown in a set of demonstrations. The BC feature can be enabled on the PPO or SAC trainers. As BC cannot generalize past the examples shown in the demonstrations, BC tends to work best when there exists demonstrations for nearly all of the states that the agent can experience, or in conjunction with GAIL and/or an extrinsic reward.

2) Imitation Learning tool: For silmulation and imitation learning tool, we choose Unity and it's ML-Agtnts package. The Unity Machine Learning Agents Toolkit (ML-Agents) is

an open-source project that enables games and simulations to serve as environments for training intelligent agents.It provide implementations (based on PyTorch) of state-of-the-art algorithms to easily train intelligent agents for 2D, 3D. Researchers can also use the provided simple-to-use Python API to train Agents using reinforcement learning, imitation learning, neuroevolution, or any other methods. Especially, it supports for learning from demonstrations through two Imitation Learning algorithms (BC and GAIL) and Deep Reinforcement Learning algorithms (PPO, SAC, MA-POCA, self-play).

Last but not least, ML-Agents package provides a very detailed hand-on documentation with 18+ examples. It's more friendly for a fresh man in Machine Learning and Unity. Fig. 16

3) Demonstration record: In unity, it's easy to record actions and state of an agent. We only need to add a record component, and choose the observer vector. In this project, we add a scripts to record the actions and state of Franka Panda. Fig. 17

4) Imitation Learning config: Set-up:A panda robot agent with seven free joints in working environment with a target cube and convey platform obstrcal. Goal:The end-effector of panda robot reaches the position of target cube. Agents: The environment contains one agent. Agent Reward Function:

- 0.01 for previous distance larger than current distance.

+1.0 of the end effector reaches the target cube's position.

- 2.0 for collision happends(episode ends)

Behavior Parameters:

Vector Observation space:14 variables corresponding to 7 joints state of panda robot, position of end-effector, position of target cube, distance between cube and end-effector.

Actions: 7 continuous actions, with the value increasing or decreasing of 7 joints value within -1 to 1.

5) Result: We build a physics simulation environment and find a good tool for training our robot arm reach a point with learning from demonstration. We also figure out suitable config for our training, such as agent, action, reward, observation. However, these are still at basic and low level of imitation learning. Besides, there are still some problems in our learning environment. For example, the simulation of forward kinematic and inverse kinematic is not fluent, which affects our training process. These problems are mainly because of the controller code of Franka Panda, it uses velocity and position controller, which needs a lot of computation resources. If we do this project again, we will build a robot arm model by ourselves, and use position controlor code directly, which will save lots of computation resources and the simulation will be more fluent.

III. Final Remark

In this project, we have collected data form learning from demonstration by passive observation, reproduced trajectory in both real and simulation environments and explored the potential of ML-Agent to do learning from demonstration. Although using the data collection platform producing by Haokun Wang makes it easier to collect task trajectories performed by demonstrators and avoid mapping human's joints to robot joints, there are still some unsolved problems in this project. For example, reproducing trajectories in Franka Arm smoothly and producing learning from demonstration in simulation environment.

Our work also shows the potential to make learning from demonstration easier by mapping the end of modified kitchen tongs to the end effector of robot arm and transfer the result trajectories produced in simulation environment to real robot arm.

IV. Contribution

Here are our group division in this project:

- Collecting data: Xiao Yang & Guo Shangkun
- Data processing trajectory reproducing in Franka: Xu Ronghan, Hu Bowen & Qiu Nuofan
- Construction of simulation environment: Dong Yujian
- Reproduction and exploration in simulation: Xiang Yanzhen
- Imitation learning in simulation: Liu Xin & Li Yifei

And they respectively wrote their own part in this essay.

References

[1]Ravichandar, Harish and Polydoros, Athanasios S. and Chernova, Sonia and Billard, Aude, "Recent Advances in Robot Learning from Demonstration" in Annual Review of Control, Robotics, and Autonomous Systems volume 3, 2020, pp.297-330,doi = 10.1146/annurev-control-100819-063206

[2]Ratliff, Nathan and Bagnell, J. Andrew and Srinivasa, Siddhartha S., "Imitation learning for locomotion and manipulation," in 2007 7th IEEE-RAS International Conference on Humanoid Robots, pp.392-397, doi.10.1109/ICHR.2007.4813899

[3]Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents

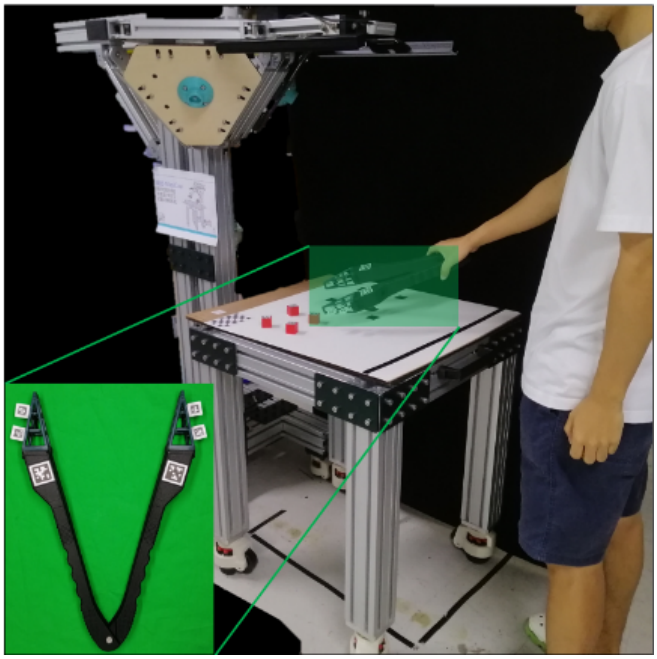


Fig. 1: Platform

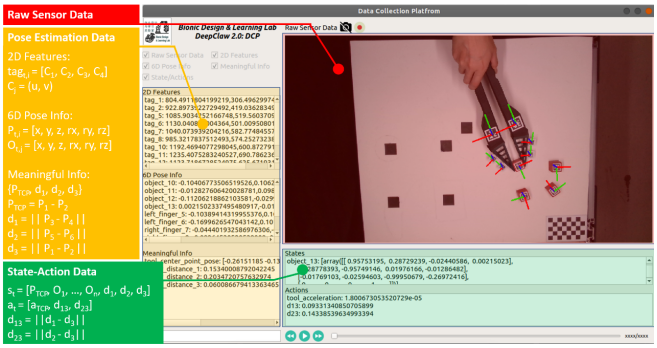


Fig. 2: Data collection

(a) Before cleaning (b) After cleaning

Fig. 3: Data cleaning Process

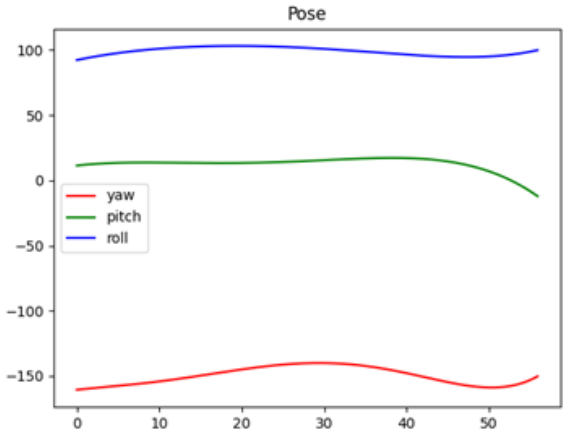


Fig. 4: 5th polynomial fitting pose data

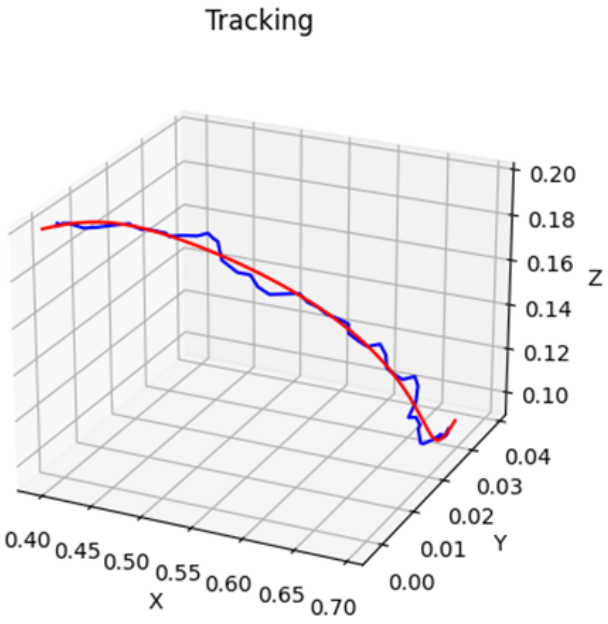
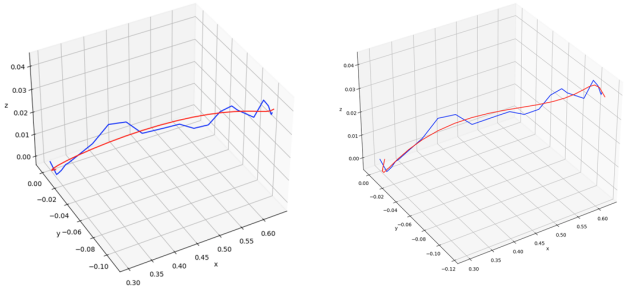


Fig. 5: Data visualization by Matplot



(a) 4th polynomial (b) 5th polynomial

Fig. 6: Polynimial fitting

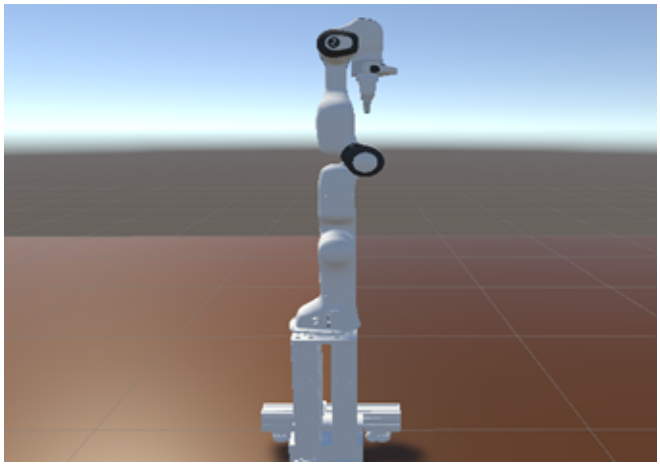


Fig. 7: Robotic arm model

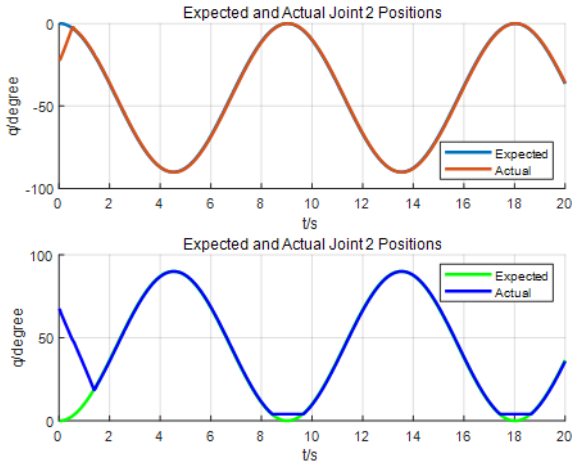


Fig. 10: Control of Arm in Joint Space

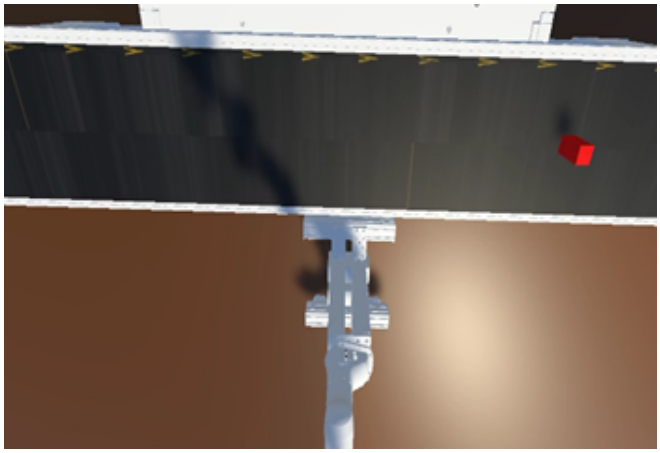


Fig. 8: Conveyor model

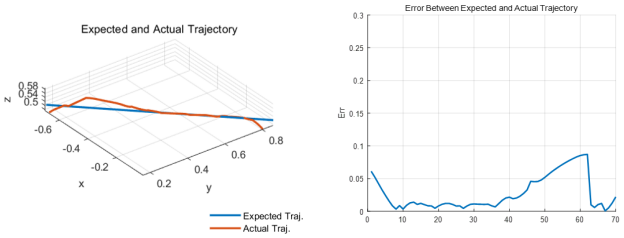


Fig. 11: Control of Arm in Workspace

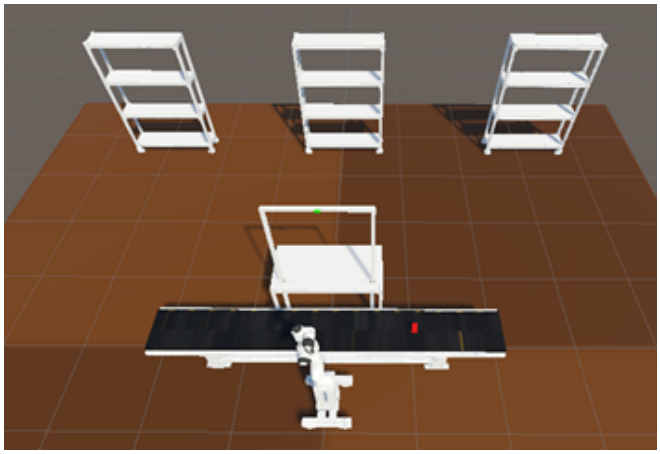
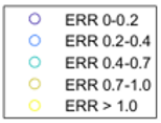


Fig. 9: Other models



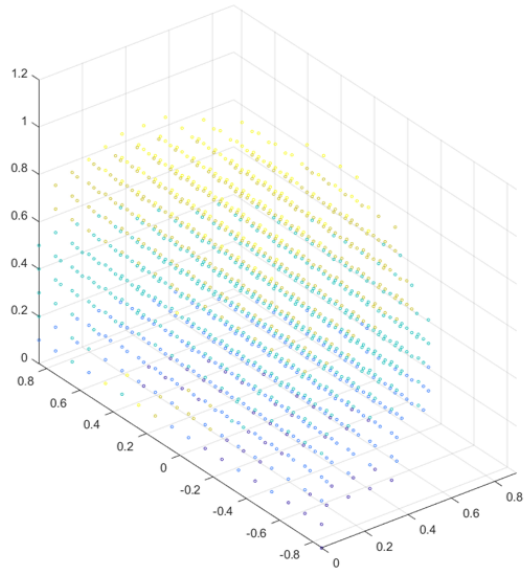


Fig. 12: Errors on Each Points in Workspace

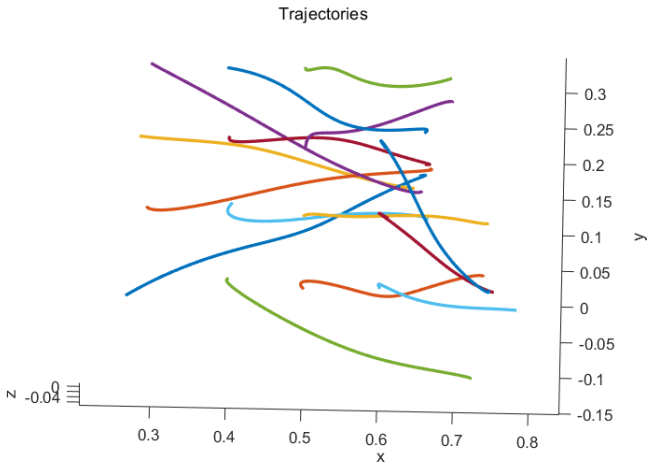


Fig. 13: Mean Sampled Trajectories

	1	2	3	4
x	0.0404	0.046	0.0692	0.0431
y	0.0344	0.038	0.0696	0.0812
z	0.0211	0.0226	0.02	0.039
	5	**6**	**7**	**8**
x	0.0169	0.01867	0.01245	0.0296
y	0.1405	0.1474	0.1004	0.109
z	0.0174	0.0125	0.0153	0.015
	9	**10**	**11**	**12**
x	0.0133	0.014	0.0208	0.0341
y	0.1037	0.138	0.1598	0.159
z	0.02625	0.0097	0.01966	0.0276
	13	**14**	**15**	**16**
x	0.0255	0.0398	0.009	0.049
y	0.0095	0.0499	0.018	0.022
z	0.024	0.011	0.009	0.02

Fig. 14: Max Standard Derivation on Each Direction at Each of 16 Targeting Positions

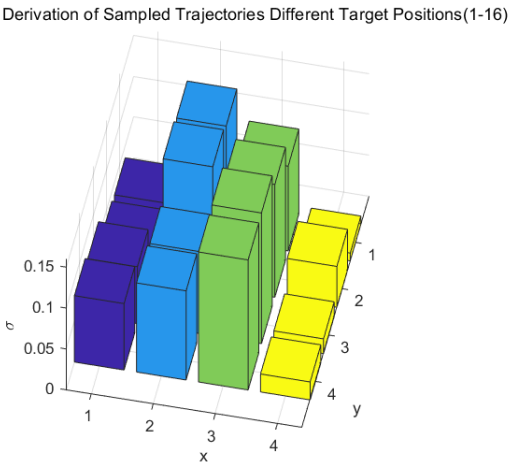


Fig. 15: Maximum Standard Derivation on y-axis Direction at Each of 16 Targeting Positions

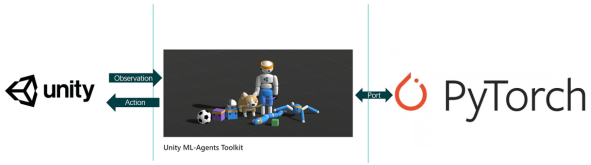


Fig. 16: ML-agent pipeline

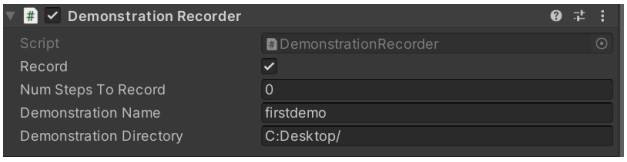
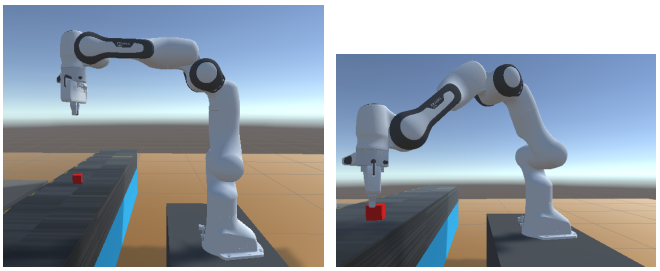


Fig. 17: Demonstration record



(a) Initial State (b) Reward State

Fig. 18: Inmitation Learning config