Robot Alchemy Practice

RobotFlow

I. Introduction

Grasp synthesis is a task widely explored in the domain of robotic learning. It typically takes a 2D image or 3D point cloud as input and outputs a valid grasp pose with which the robot gripper could stably hold the object (as shown in Fig.1). Since it's crucial for further manipulation tasks, it's quite appropriate for junior students to start their robot learning journey with such a task.

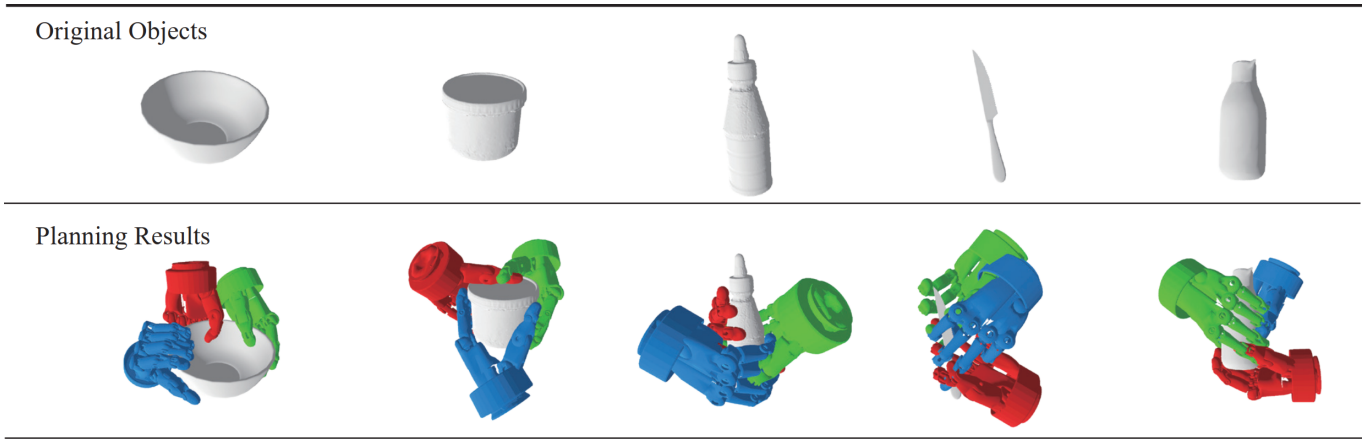

Fig. 1. Grasp Task

In robot learning tasks, the end-effector of a robot can be divided into two categories: the parallel gripper and the dexterous hand. The parallel gripper has only two opposing fingers, while the dexterous hand refers to those mechanical hands that have more than three fingers (some are shown in Fig.2). This leads to a higher degree of freedom (DoF), a more complex spatial structure, and a more intricate contact system. Compared to the grasp task using a parallel gripper, the dexterous grasp task requires the algorithm to not only focus on the spatial relation between the hand and the object, but also to handle the intricate mechanical structure and the much larger joint state space.

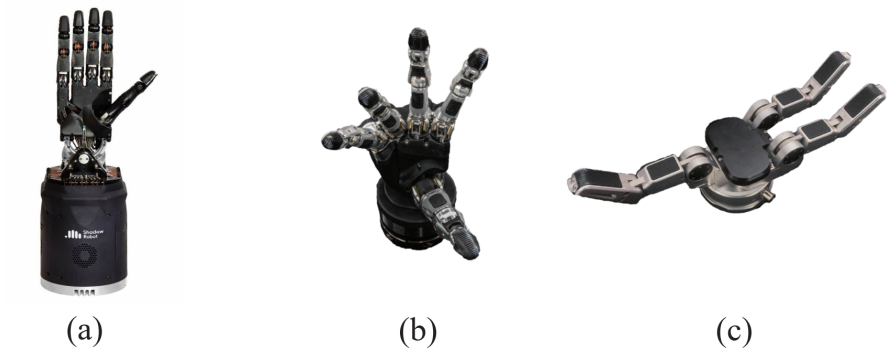

Fig. 2. (a) Shadow hand with 24-DoF (b) Schunk SVH hand with 9-DOF (c) Barrett hand with 4-DoF

Here, we design a dexterous grasp validation task. It takes an object's point cloud and a given grasp pose for the Shadow Hand as input, and outputs a binary value indicating whether the pose is feasible for grasping the given object. We will introduce the dataset in detail in Sec.II, and provide the code requirements you need to meet in Sec.III. **You can download the project here**.

II. Dataset

This project includes two datasets. One contains approximately 20 objects, and the other comprises grasp poses, which consist of a 6-D wrist pose, hand joint parameters, and a label indicating the quality of the grasp. Both datasets are located in the *asset/data* path. The structure of the grasp dataset is as follows:

```
grasp
|--train
|   |--obj1
|   |      |--pose.npy
|   |      |--joint_state .npy
|   |      |--label.npy
|   |--obj2
|    ...
|--test
     |--objx
      ...
```

The directory name of the description files corresponds to the object index for the grasps described by these files. You can find the object in the object dataset using this index. The files *pose.npy*, *joint_state.npy*, and *label.npy* contain numpy arrays with shapes $(N, 4, 4)$, $(N, 23)$, and (N) respectively. The *pose.npy* file contains transformation matrices, the *joint_state.npy* file contains joint angles measured in degrees, and the *label.npy* file contains labels indicating whether the grasps are valid.

The object dataset is a part of AKB48. We only use the pointcloud of its objects. As the code for the object dataset has already been implemented, we won't discuss it further.

The *test* dataset in the given project is, in fact, a part of the training dataset. You can use it to verify if your *test.py* executes correctly. During the formal testing process, we will replace this placeholder test dataset with the actual test dataset. The number of positive and negative samples will be equal in the test dataset. Furthermore, the test dataset will include some grasp poses on objects that were not seen in the training dataset.

III. Code Requirement

The code could be divided into three parts:

- **Data Processing**:All data processing code should be stored under the *data* directory. You need to complete the implementation of the grasp dataset in *data/dataset.py* by referring to the object dataset code. You may also find useful resources in *data/utils.py*.
- **Model**: All code related to neural network construction should be stored under the *model* directory. You can create new Python scripts to construct your own model.
- **Training**: All code related to model training should be written in the *model/dexGraspEvaluator.py* file. You need to implement all the functions in this file.

Apart from the three parts mentioned above, you can modify or add any other parts to support your project.

You may have noticed that the code framework uses Hydra library and Pytorch Lightning library to simplify program development. Hydra will dynamically create a hierarchical configuration by composition and override it through config files and the command line, while Pytorch Lightning is a deep learning framework designed for professional AI researchers and machine learning engineers who need maximum flexibility without sacrificing performance at scale. As the saying goes, *a beard well lathered is half shaved*, we hope that you can take advantage of these efficient tools. You can refer to this project and their official documents to get familiar with these two libraries.

IV. Scoring Standard

Your final score depends on three aspects of your works:

- **Code Quality(40%)**: Your source code should be well-organized. It should be easy to read, with appropriate variable and function names. Excellent code comments and hyper-parameter management are encouraged.
- **Report Quality(30%)**: Your idea should be presented with clear expression, a well-organized structure, and possibly some graphics. Your method should perform better than other baselines. This could mean having a smaller loss, higher accuracy, shorter training time, smaller parameter size, or other advantages.
- **Method Performance(30%)**: Your method will be tested on the test dataset, and the accuracy and average loss value will be taken into account.

Attention: Copying and plagiarism are strictly prohibited. All code should be original, except for code borrowed from open source projects (which should be properly cited in your report).

V. Environment Requirement

- **Operating System** Your project should be able to run on Ubuntu-20.04 or higher version. If you prefer using Windows, we strongly recommend you to leverage WSL2 to use Ubuntu environment.
- **Python Version** Your project should be able to run with Python 3.9 or higher version.
- **Dependent Packages** The *requirements.txt* file provides a list of dependent packages. If your project dependent on other packages, you need to submit an extended version of *requirements.txt*.

A. Python Environment Setting

You are recommended to use Miniconda to manage your python environments. To use Miniconda, install it by following the guidelines in the official documentation. Use the commands below to set up an environment with Python 3.9:

```
conda create –n py39 python=3.9
conda activate py39
```

B. Dependency Setting

Using *pip*, you can install the dependent packages quickly:

```
pip install –r requirements.txt
```

VI. Submission Requirement

Your submission should contain two parts:

- You need to submit a report in *pdf* format. This report should include, but is not limited to, your network structure, your training approach, and the results of your method. The report should not exceed 4 pages and must be written in English.
- You need to package all the source code along with your best checkpoint, excluding the dataset, and write a *README.md* file to explain how to set up the code environment and run the *test.py* file.

All submission files should be packaged into a *zip* file.

VII. Information Platform

All questions and discussions should take place on our forum. Informal questions are not allowed, and we won't respond to any questions privately. Any further supplementary files, such as hints or related papers about this project, will be posted on the forum's bulletin.