Based on CNN convolutional neural network algorithm garbage sorting manipulator

Shen Qirong (12010311), Zhou Mingyu (12012725), Wang Zimeng (12011711)
Wen Junzhe (11910703), Liu Qian(11912312)

***Abstract*—This project aims to develop a new type of garbage sorting robot that can recognize the location of garbage through a camera and make corresponding posture changes so that it can reach the posture of picking up garbage. At the same time, it also needs to train a machine learning model to recognize different types of garbage and sort them to the correct location. This project will combine machine vision, deep learning systems, robotic arms, and the basic principles of robotic arms to achieve. By training a robotic arm to sort garbage, the efficiency and accuracy of garbage classification can be greatly improved.**

I. Introduction

Garbage classification is an important environmental issue, and machine learning technology can help us better solve this problem. By training a robotic arm to sort garbage, the efficiency and accuracy of garbage classification can be greatly improved. The background of this project is that we hope to develop a new type of garbage sorting robot that can recognize the location of garbage through a camera and make corresponding posture changes so that it can reach the posture of picking up garbage. At the same time, it also needs to train a machine learning model to recognize different types of garbage and sort them to the correct location. This project will combine machine vision, deep learning systems, robotic arms, and the basic principles of robotic arms to achieve.

We plan to complete this project through the following steps:

Collecting garbage image datasets: We need to collect garbage image datasets of different types to train the machine learning model.

Data preprocessing: For the collected image data, you need to do some preprocessing, such as image enhancement and image cropping.

Training the model: Train a machine learning model using a deep learning framework (such as TensorFlow or PyTorch) to recognize different types of garbage.

Deploying the model: Deploy the trained model to the robotic arm and write control programs so that it can sort garbage according to the recognition results.

II. Problem Statement

In this course project, we want to develop a robot who can distinguish between different kinds of garbage and finish the garbage sorting work by order. We divide our problem into two parts precisely: First, we need to train the robot into distinguish different kinds of garbage. Second, we need to implement the task of bin picking in simulation environment on the computer. The dataset we choose to train our robot is Dexterity Network (Dex-Net) and the simulation environment we choose is Robosuite. The result we expect is when given a heap of garbage, our robot can recognize different kinds of these garbage and successfully sort them out by grabbing the garbage and place it into the corresponding area in the simulation environment. The performance metric will be: accuracy of distinguishing, success rate in sorting task, efficiency of distinguishing, efficiency of sorting task.

III. Literature Review

In recent years, robotic grasping and manipulation have become important research areas in robotics. **Dex-Net2.0** is a state-of-the-art deep learning-based method that aims to provide high-accuracy grasping of objects. In this paper, we will discuss the relationship between Dex-Net2.0 and our project, which involves visual-based garbage classification and sorting using a robotic arm in the Robosuite simulation environment.

Dex-Net2.0 is a deep learning-based method that uses a 3D point cloud of an object obtained from an RGB-D sensor to predict the grasp quality and pose of the object. The method uses a two-step process to estimate the object's pose: First, a set of 6-DoF pose hypotheses are generated using a convolutional neural network (CNN) that takes the point cloud as input. Then, these pose hypotheses are refined using a second neural network that predicts the probability of success for each pose.

Dex-Net2.0 has several unique features that make it well-suited for robotic grasping tasks. First, it can handle highly cluttered scenes with multiple objects, which is a common scenario in our garbage sorting project. Second, it can estimate grasp stability, which is critical for ensuring that the robotic arm can securely grasp the object. Third, it can predict grasps for novel objects that were not present in the training dataset.

In our project, we plan to use Dex-Net2.0 to estimate the pose and grasp quality of garbage items using an RGB-D camera. This will enable our robotic arm to securely and accurately pick up the garbage and place it in the correct bin. Dex-Net2.0's ability to handle cluttered scenes and estimate grasp stability will be particularly useful in our project as the garbage items may be randomly scattered and have different shapes and sizes. Furthermore, Dex-Net2.0's ability to predict grasps for novel objects will allow us to easily add new types of garbage items to the system as needed.

IV. Technical Approach

For out project, the main two technical problems are how to classify different garbage and how to plan the trajectory. For garbage identification task, we are going to use convolution neural network. In class, we learned about multi-layer perceptron and some of deep learning. But the traditional neural network can not deal with as simple as a image with 1000*1000 pixels which have 1000000 dimensions. Training it directly with traditional neural network would easily cause overfitting and huge computation resources is acquired. To solve this problem, we are going to use Convolution Neural Network(CNN) for our computer vision part. Indeed, this is a optimization problem.

$$w, b = argmin_{w,b} L(w, b)$$

There are many encapsulated function in tensorflow to help you begin a garbage classification machine learning progress. We now have run a training progress in kaggle which have an accuracy up to 95 percent or more. For trajectory planning, we are going to use the library provided by the simulation environment since the DH matrix is tedious to solve by ourselves. The main problem is the Jacob matrix in velocity kinematics.

$$\dot{q} = J(q)^{-1}\dot{X}$$

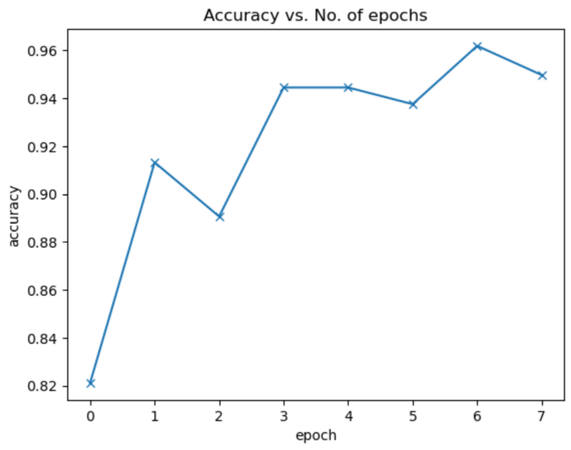
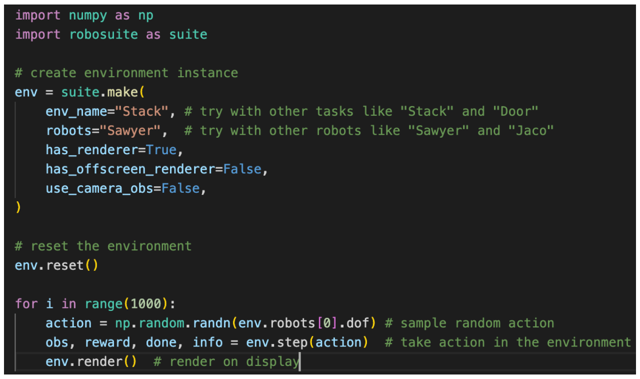


Fig. 1: Accuracy by ResNet

V. Intermediate/Preliminary Result

A. *Try and study the simulation environment–roboSuite*



```
import numpy as np
import robosuite as suite

# create environment instance
env = suite.make(
    env_name="Stack", # try with other tasks like "Stack" and "Door"
    robots="Sawyer",  # try with other robots like "Sawyer" and "Jaco"
    has_renderer=True,
    has_offscreen_renderer=False,
    use_camera_obs=False,
)

# reset the environment
env.reset()

for i in range(1000):
    action = np.random.randn(env.robots[0].dof) # sample random action
    obs, reward, done, info = env.step(action)  # take action in the environment
    env.render()  # render on display
```

Fig. 2: Code Structure

*1) **Code structure**:* Define environment and robot than choose the controller. There are many pre-defined environments and robots that we can use in our project but we still need to rewrite the environment file to adjust to our application environment like add some different shapes of items as rubbish to pick up. So, study the simulation code is pretty important for our project.

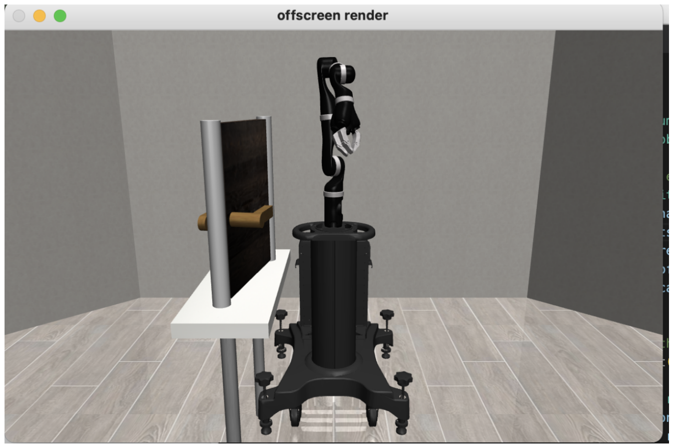


Fig. 3: Offscreen render

*2) **Controller**:* The simulation environment have different controllers. Some of them are used to control the gripper and the others are used to control the joints. More details as fellow:

OSC_POSE: Gripper moves sequentially and linearly in x, y, z direction, then sequentially rotates in x-axis, y-axis, z-axis, relative to the global coordinate frame

OSC_POSITION: Gripper moves sequentially and linearly in x, y, z direction, relative to the global coordinate frame

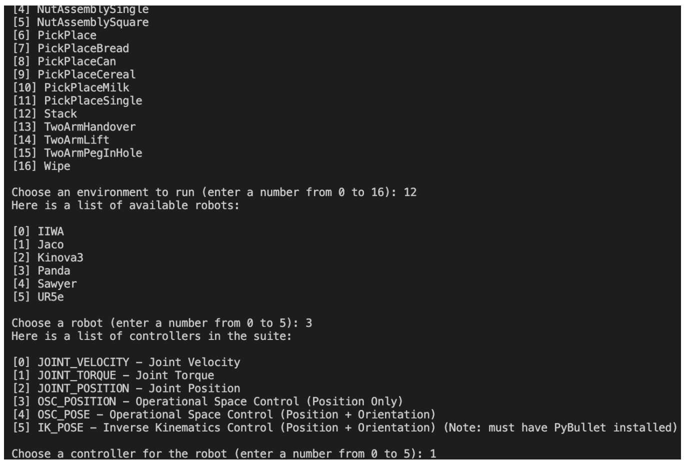
IK_POSE: Gripper moves sequentially and linearly in x, y, z direction, then sequentially rotates in x-axis, y-axis, z-axis, relative to the local robot end effector frame

JOINT_POSITION: Robot Joints move sequentially in a controlled fashion

JOINT_VELOCITY: Robot Joints move sequentially in a controlled fashion

JOINT_TORQUE: Unlike other controllers, joint torque controller is expected to act rather lethargic, as the “controller” is really just a wrapper for direct torque control of the mujoco actuators. Therefore, a “neutral” value of 0 torque will not guarantee a stable robot when it has non-zero velocity!

By using the demo_control.py code we can directly feel the differences between various controller.



```
[4] NutAssemblySingle
[5] NutAssemblySquare
[6] PickPlace
[7] PickPlaceBread
[8] PickPlaceCan
[9] PickPlaceCereal
[10] PickPlaceMilk
[11] PickPlaceSingle
[12] Stack
[13] TwoArmHandover
[14] TwoArmLift
[15] TwoArmPegInHole
[16] Wipe

Choose an environment to run (enter a number from 0 to 16): 12
Here is a list of available robots:

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

Choose a robot (enter a number from 0 to 5): 3
Here is a list of controllers in the suite:

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

Choose a controller for the robot (enter a number from 0 to 5): 1
```

Fig. 4: Demo_control

B. ***Future Work***

1) ***Build our own simulation world:*** Although there are so many pre-designed environments in the roboSuite, we also need to study how to build a new world for our project. This work related to study the model file of roboSuite.

2) ***Combine our code with the simulation code:*** It is obviously that we must ensure our controlling code can be combined with the simulation code perfectly because as we have seen that the simulation code also include controllers. So we intend to write our code in python and early apply it in the simulation environment to test the possibility.