Yolov11 Garbage Recognition and Sorting

Before starting this function, you need to close the main program and APP processes. If you need to restart the main program and APP later, start them from the terminal:

```
bash ~/dofbot_pro/APP_DOFBOT_PRO/start_app.sh
```

1. Function Description

After the program starts, the camera captures images. Place garbage label code blocks in the image. The robotic arm will recognize the category of garbage label code blocks, lower its gripper to pick up the blocks, and place them at designated positions based on the identified garbage category. After placement, it returns to the recognition pose.

2. Startup and Operation

2.1. Startup Commands

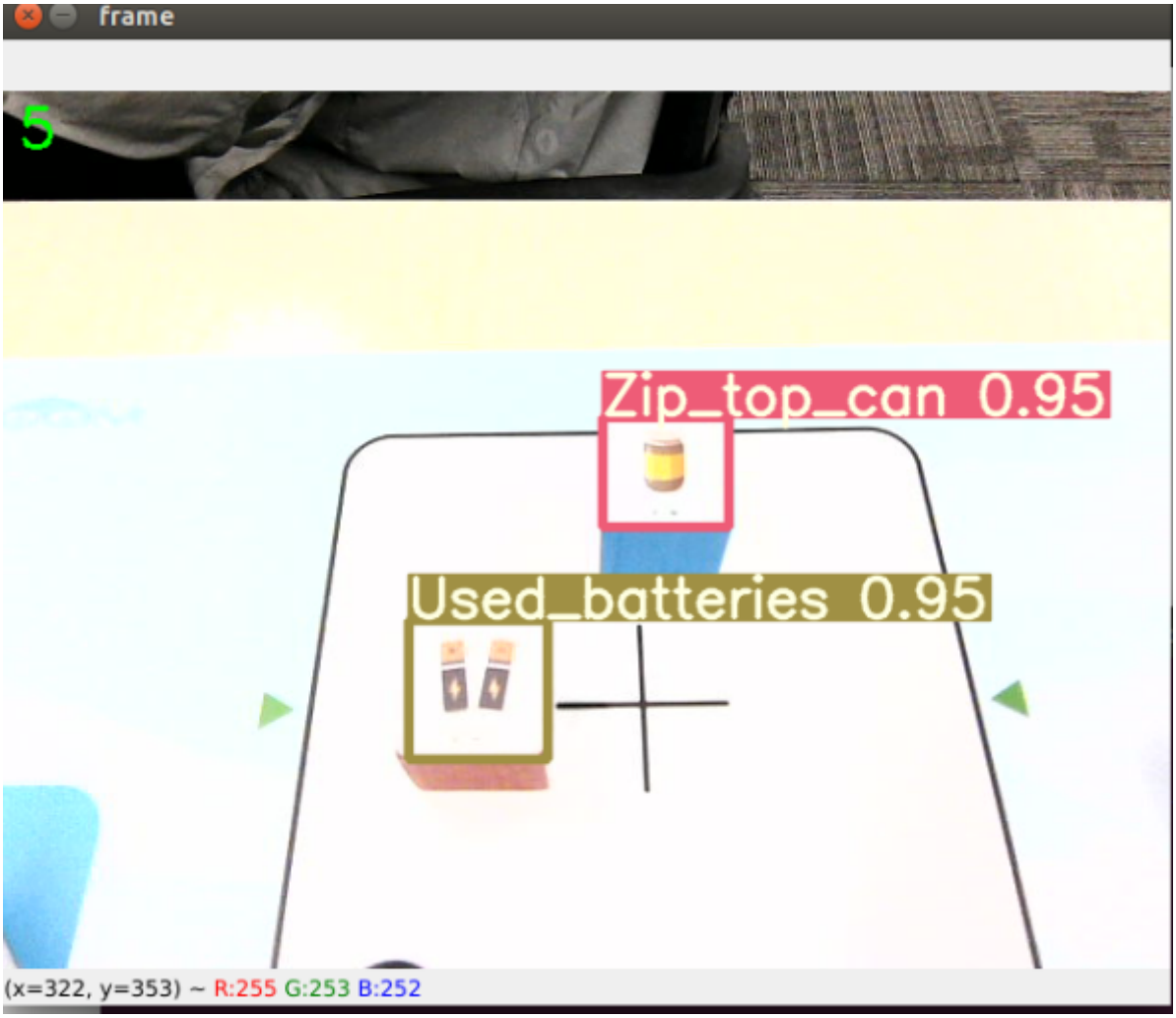
Enter the following commands in the terminal to start:

```
# Start camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
# Start low-level control:
ros2 run dofbot_pro_driver arm_driver
# Start inverse kinematics program:
ros2 run dofbot_pro_info kinemarics_dofbot
# Start image conversion program:
ros2 run dofbot_pro_yolov11 msgToimg
# Start Yolov11 recognition program:
python3 ~/dofbot_pro_ws/src/dofbot_pro_yolov11/dofbot_pro_yolov11/yolov11.py
# Start robotic arm garbage sorting program:
ros2 run dofbot_pro_yolov11 yolov11_sortation
```

Due to performance differences of mainboards, the loading time for Yolov11 recognition program varies across different mainboards. Please wait patiently.

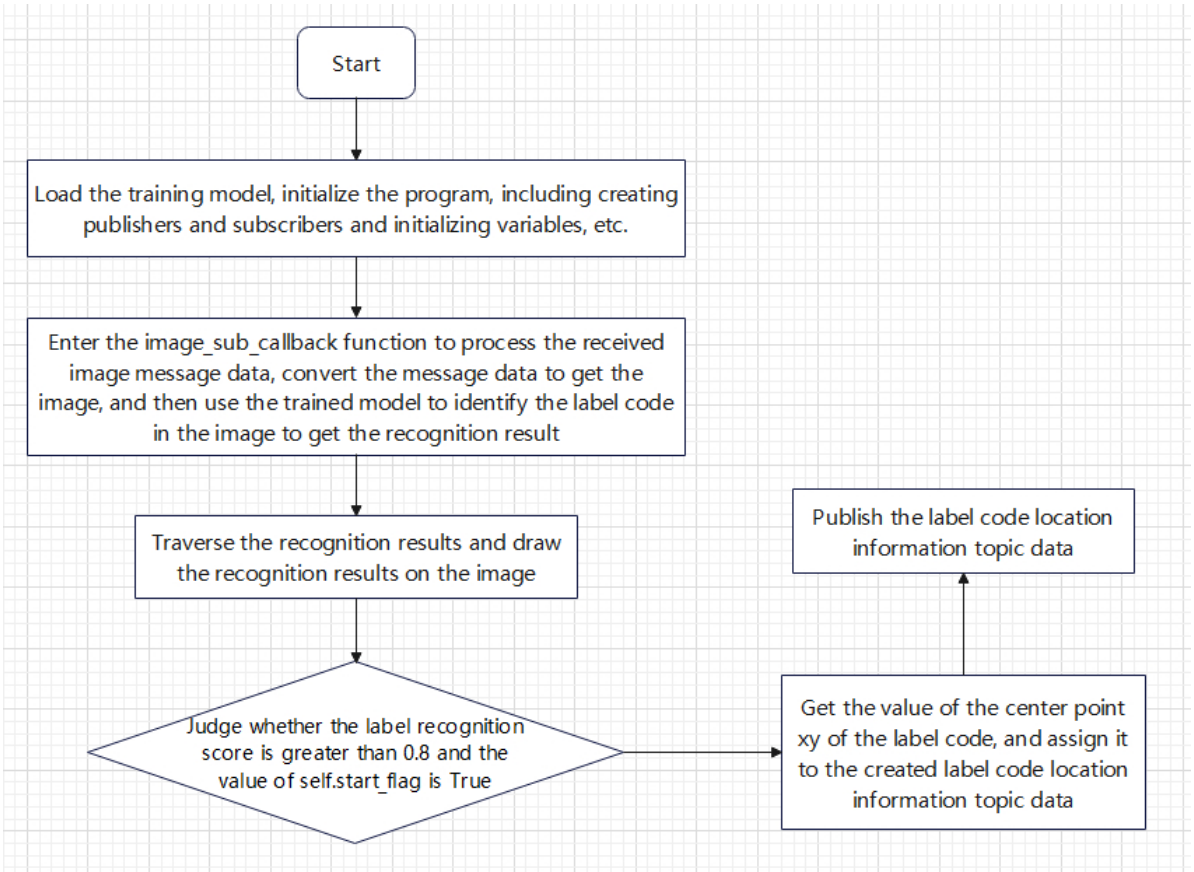
2.2. Operation Flow

After the program starts, place wooden blocks with garbage label codes in the center of the image. The blocks should be positioned upright with the icon facing the same direction as the robotic arm (forward, Y-axis direction). Press the spacebar to start recognition. The robotic arm will lower its gripper to pick up the blocks and place them at corresponding designated positions based on the identified garbage category.

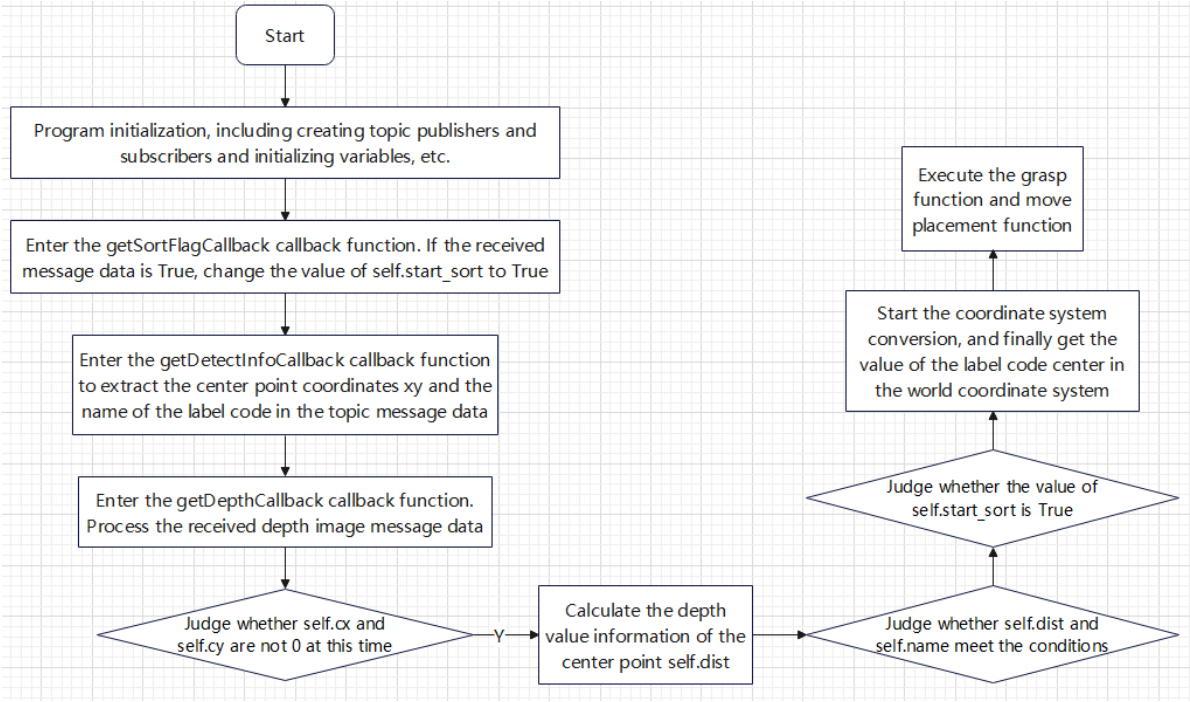


3. Program Flowchart

yolov11.py



yolov11_sortation.py



4. Core Code Analysis

4.1. msgToimg.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_yolov11/dofbot_pro_yolov11/msgToimg.py
```

Import necessary libraries:

```
import sys
import rospy
import numpy as np
import os
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from dofbot_pro_info.msg import Image_Msg
```

Program parameter initialization, create publishers and subscribers:

```
def __init__(self):
    #Create bridge for converting color image topic message data to image data
    self.bridge = CvBridge()
    #Create color image topic subscriber
    self.image_sub =
rospy.Subscriber("/camera/color/image_raw",Image,self.image_sub_callback)
    #Create image data publisher
    self.image_pub = rospy.Publisher('/image_data', Image_Msg, queue_size=1)
    self.img = np.zeros((480, 640, 3), dtype=np.uint8)  # Initial image
    self.yolov5_img = np.zeros((480, 640, 3), dtype=np.uint8)  # Initial image
    self.img_flip = rospy.get_param("~img_flip", False)
    #Initialize image data message object
    self.image = Image_Msg()
```

image_sub_callback callback function:

```
def image_sub_callback(self, data):
    #Receive color image topic message, convert message data to image data
    self.img = self.bridge.imgmsg_to_cv2(data, "bgr8")
    #Get image dimensions
    size = self.img.shape
    #Assign values to image message object data
    self.image.height = size[0] # 480
    self.image.width = size[1] # 640
    self.image.channels = size[2] # 3
    self.image.data = data.data # image_data
    #Publish image topic
    self.image_pub.publish(self.image)
```

4.2. yolov11.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_yolov11/dofbot_pro_yolov11/yolov11.py
```

Import necessary library files:

```
import rclpy
from rclpy.node import Node
import Arm_Lib
import os
import time
import cv2
import cv2 as cv
import numpy as np
import threading
from time import sleep
import ipywidgets as widgets
from std_msgs.msg import Float32, Bool
from IPython.display import display
from dofbot_pro_yolov11.fps import FPS
from ultralytics import YOLO
from dofbot_pro_yolov11.robot_controller import Robot_Controller
from dofbot_pro_interface.msg import *
encoding = ['16UC1', '32FC1']
```

Program parameter initialization, create publishers and subscribers:

```
def __init__(self):
     super().__init__('detect_node')

    self.pr_time = 0
    self.image_sub =
self.create_subscription(ImageMsg,"/image_data",self.image_sub_callback,qos_prof
ile=1)
    self.img = np.zeros((480, 640, 3), dtype=np.uint8)  # Initial image
    self.init_joints = [90.0, 120.0, 0.0, 0.0, 90.0, 90.0]
```

```
    self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 10)
    self.pubDetect = self.create_publisher(Yolov11Detect, "Yolov11DetectInfo",
10)
    self.pub_SortFlag = self.create_publisher(Bool, 'sort_flag', 10)
    self.grasp_status_sub =
self.create_subscription(Bool,'grasp_done',self.GraspStatusCallback,qos_profile=
1)
    self.start_flag = False
    self.yolo_model =
YOLO("/home/jetson/dofbot_pro_ws/src/dofbot_pro_yolov11/dofbot_pro_yolov11/best.
engine", task='detect')
    self.fps = FPS()
```

Callback function for subscribing to image topics:

```
 def image_sub_callback(self,data):
        image = np.ndarray(shape=(data.height, data.width, data.channels),
dtype=np.uint8, buffer=data.data) # Convert custom image message to image
        self.img[:,:,0],self.img[:,:,1],self.img[:,:,2] =
image[:,:,2],image[:,:,1],image[:,:,0] # Convert rgb to opencv's bgr order
        # self.img = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)


        results = self.yolo_model(self.img, save=False, verbose=False)  # Use
YOLO11 for object detection
        annotated_frame = results[0].plot(
            labels = True, # Show labels
            conf = False,  # Show confidence
            boxes = True,  # Draw bounding boxes
        )
        boxes = results[0].boxes
        key = cv2.waitKey(10)

        if boxes != [None] and self.start_flag == True:
            for box in boxes:  # detections per image
                x_min, y_min, x_max, y_max = map(int, box.xyxy[0])
                class_id = int(box.cls)
                confidence = float(box.conf)
                label = f"{self.yolo_model.names[class_id]} {confidence:.2f}"
                # Calculate center position
                center_x = (x_min + x_max) // 2
                center_y = (y_min + y_max) // 2

                center = Yolov11Detect()
                center.centerx = float(center_x)
                center.centery = float(center_y)
                center.result = str(self.yolo_model.names[class_id])

                # cv2.circle(annotated_frame, (center_x, center_y), 5, (0, 0,
255), -1)  # Draw red dot
                cv2.putText(annotated_frame, label, (x_min, y_min - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)  # Display in upper left corner

                self.pubDetect.publish(center)
                self.start_flag = False
        cur_time = time.time()
        fps = str(int(1/(cur_time - self.pr_time)))
```

```
        self.pr_time = cur_time
        cv2.putText(annotated_frame, fps, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 2)
        cv2.imshow("frame", annotated_frame)
```

4.3. yolov11_sortation.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_yolov11/dofbot_pro_yolov11/yolov11_sor
tation.py
```

Import necessary libraries:

```
import rclpy
import cv2
from rclpy.node import Node
import numpy as np
from std_msgs.msg import Float32,Bool,Int8
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
import cv2 as cv
import time
import math

from message_filters import ApproximateTimeSynchronizer

from dofbot_pro_interface.msg import *
from dofbot_pro_interface.srv import *

import transforms3d as tfs
import tf_transformations as tf         # ROS2 uses tf_transformations
import threading

from ament_index_python import get_package_share_directory
import yaml
import os
from Arm_Lib import Arm_Device
```

Open offset parameter table:

```
pkg_path = get_package_share_directory('dofbot_pro_driver')
offset_file = os.path.join(pkg_path,'config', 'offset_value.yaml')
with open(offset_file, 'r') as file:
    offset_config = yaml.safe_load(file)
print(offset_config)
print("----------------------------")
print("x_offset: ",offset_config.get('x_offset'))
print("y_offset: ",offset_config.get('y_offset'))
print("z_offset: ",offset_config.get('z_offset'))
encoding = ['16UC1', '32FC1']
```

Program parameter initialization, create publishers, subscribers, etc.:

```
def __init__(self):
```

```
    super().__init__('yolov11_grap')
    self.cx = 0
    self.cy = 0
    self.Arm = Arm_Device()
    self.sub_joint5 = self.create_subscription(Float32,
"adjust_joint5",self.get_joint5Callback,qos_profile=1)
     # Publisher modifications
    self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle",
qos_profile=10 )
    self.pubGraspStatus = self.create_publisher(Bool, "grasp_done",
qos_profile=10)
    self.pub_playID = self.create_publisher(Int8, "player_id", qos_profile=10)
    # Other subscribers
    self.subDetect = self.create_subscription(Yolov11Detect,
"Yolov11DetectInfo",self.getDetectInfoCallback,qos_profile=10)
    self.depth_image_sub =
self.create_subscription(Image,'/camera/depth/image_raw',self.getDepthCallback,q
os_profile=1)
    self.sub_SortFlag =
self.create_subscription(Bool,'sort_flag',self.getSortFlagCallback,qos_profile=1
0)
    # Service client modifications
    self.client = self.create_client(Kinemarics, "dofbot_kinemarics")
    #Initialize grasp flag, True means grasping is possible
    self.grasp_flag = True
    self.init_joints = [90.0, 120, 0.0, 0.0, 90, 90]
    self.down_joint = [130.0, 55.0, 34.0, 16.0, 90.0,125]
    self.set_joint = [90.0, 120, 0.0, 0.0, 90, 90]
    self.gripper_joint = 90
    self.depth_bridge = CvBridge()
    self.start_sort = False
    self.CurEndPos = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    #Camera intrinsic parameters
    self.camera_info_K = [477.57421875, 0.0, 319.3820495605469, 0.0,
477.55718994140625, 238.64108276367188, 0.0, 0.0, 1.0]
    #Rotation transformation matrix of relative pose between camera and robotic
arm end
    self.EndToCamMat =
np.array([[1.00000000e+00,0.00000000e+00,0.00000000e+00,0.00000000e+00],
                                  [0.00000000e+00,7.96326711e-04,9.99999683e-
01,-9.90000000e-02],
                                  [0.00000000e+00,-9.99999683e-01,7.96326711e-
04,4.90000000e-02],

[0.00000000e+00,0.00000000e+00,0.00000000e+00,1.00000000e+00]])
    #Get current robotic arm end position and pose information
    self.get_current_end_pos()
    #Current label center coordinate values
    self.cx = 320
    self.cy = 240
    #Garbage label name
    self.name = None
    #Read offset parameter table content, assign values to offset parameters
    self.x_offset = offset_config.get('x_offset')
    self.y_offset = offset_config.get('y_offset')
    self.z_offset = offset_config.get('z_offset')

    self.play_id = Int8()
```

```
    #List of four types of garbage
    self.recyclable_waste=['Newspaper','Zip_top_can','Book','Old_school_bag']
    self.toxic_waste=
['Syringe','Expired_cosmetics','Used_batteries','Expired_tablets']
    self.wet_waste=['Fish_bone','Egg_shell','Apple_core','Watermelon_rind']
    self.dry_waste=
['Toilet_paper','Peach_pit','Cigarette_butts','Disposable_chopsticks']
    print("Current_End_Pose: ",self.CurEndPos)
    print("Init Done")
```

Callback function for recognized garbage label results getDetectInfoCallback:

```
def getDetectInfoCallback(self,msg):
    #Assign label center coordinate values and label name
    self.cx = int(msg.centerx)
    self.cy = int(msg.centery)
    self.name = msg.result
```

Depth image topic callback function getDepthCallback:

```
def getDepthCallback(self,msg):
    #Process received depth image topic message
    depth_image = self.depth_bridge.imgmsg_to_cv2(msg, encoding[1])
    frame = cv.resize(depth_image, (640, 480))
    depth_image_info = frame.astype(np.float32)
    #Check if self.cy and self.cx values are not 0
    if self.cy!=0 and self.cx!=0:
        #Get depth value of center point
        self.dist = depth_image_info[self.cy,self.cx]/1000
        print("self.dist",self.dist)
        print("get the cx,cy",self.cx,self.cy)
        print("get the detect result",self.name)
        #Check if center point depth value is not 0 and self.name value is not
None
        if self.dist!=0 and self.name!=None:
            #Check if self.start_sort is True
            if self.start_sort == True:
                #Start coordinate system conversion, finally output center point
position in world coordinate system
                camera_location =
self.pixel_to_camera_depth((self.cx,self.cy),self.dist)
                PoseEndMat = np.matmul(self.EndToCamMat,
self.xyz_euler_to_mat(camera_location, (0, 0, 0)))
                #PoseEndMat = np.matmul(self.xyz_euler_to_mat(camera_location,
(0, 0, 0)),self.EndToCamMat)
                EndPointMat = self.get_end_point_mat()
                WorldPose = np.matmul(EndPointMat, PoseEndMat)
                #WorldPose = np.matmul(PoseEndMat,EndPointMat)
                pose_T, pose_R = self.mat_to_xyz_euler(WorldPose)
                #Add offset parameters to compensate for deviations caused by
servo value differences
                pose_T[0] = pose_T[0] + self.x_offset
                pose_T[1] = pose_T[1] + self.y_offset
                pose_T[2] = pose_T[2] + self.z_offset
                #Start grasping thread, pass parameter is the label position in
world coordinate system just calculated
```

```
                grasp = threading.Thread(target=self.grasp, args=(pose_T,))
                grasp.start()
                grasp.join()
```

Start sorting topic callback function getSortFlagCallback:

```
def getSortFlagCallback(self,msg):
    #Check if received value is True, if so modify self.start_sort value to True
    if msg.data == True:
        self.start_sort = True
```

Grasping function grasp:

```
def grasp(self,pose_T):
    print("--------------------------------------------------")
    print("pose_T: ",pose_T)
    #Call ik algorithm in inverse kinematics service to calculate 6 servo values
    request = kinemaricsRequest()
    #Target x value of robotic arm end, unit is m
    request.tar_x = pose_T[0]
    #Target y value of robotic arm end, unit is m
    request.tar_y = pose_T[1]
    #Target z value of robotic arm end, unit is m, 0.2 is scaling factor, make
minor adjustments based on actual situation
    request.tar_z = pose_T[2] +
(math.sqrt(request.tar_y**2+request.tar_x**2)-0.181)*0.2
    #Specify service content as ik
    request.kin_name = "ik"
    #Target Roll value of robotic arm end, unit is radians, this value is current
robotic arm end roll value
    request.Roll = self.CurEndPos[3]
    print("calcutelate_request: ",request)
    try:
        response = self.client.call(request)
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        #Assign service returned joint1-joint6 values to joints
        joints[0] = response.joint1
        joints[1] = response.joint2
        joints[2] = response.joint3
        if response.joint4>90:
            joints[3] = 90
        else:
            joints[3] = response.joint4
        joints[4] = 90
        joints[5] = 30
        print("compute_joints: ",joints)
        self.pubTargetArm(joints)
        time.sleep(2.5)
        #After grasping, call move function, determine which garbage list it
belongs to based on self.name value, place at designated position
        self.move()
    except Exception:
       rospy.loginfo("run error")
```