Yolov11 garbage identification and sorting

Before starting this function, you need to close the process of the big program and APP. If you need to start the big program and APP again later, start the terminal,

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

1. Function description

After the program is started, the camera captures the image, places the garbage label code block in the image, the robot arm recognizes the category of the garbage label code block, and grabs the garbage label code block with the lower claw. According to the category of the garbage label code, it is placed in the set position. After the placement is completed, it returns to the recognized posture.

2. Start and operate

2.1. Start command

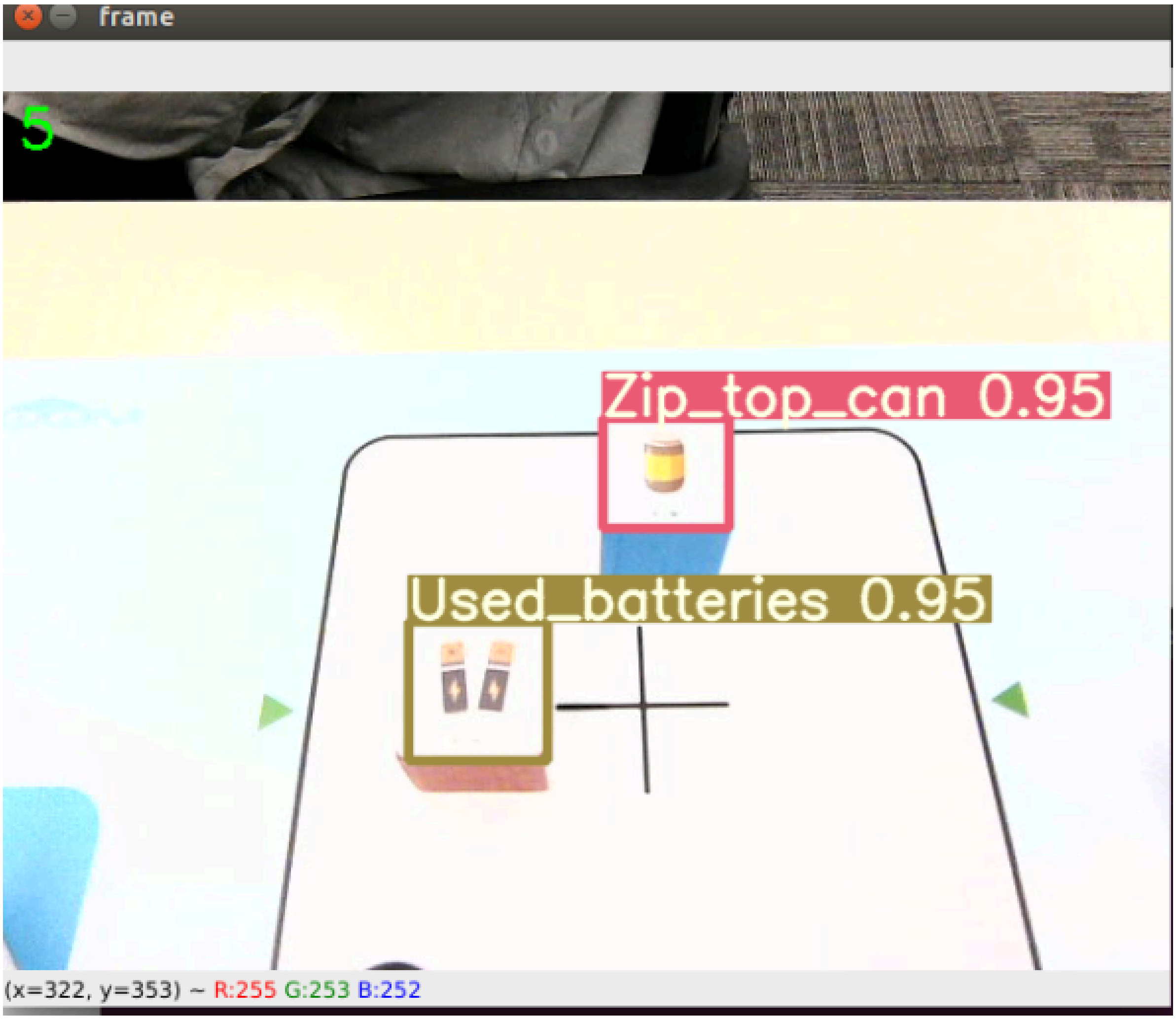
Enter the following command in the terminal to start,

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

Due to the performance differences of the motherboard, the time it takes to load the Yolov11 recognition program on different motherboards is different. You need to wait patiently for a while.

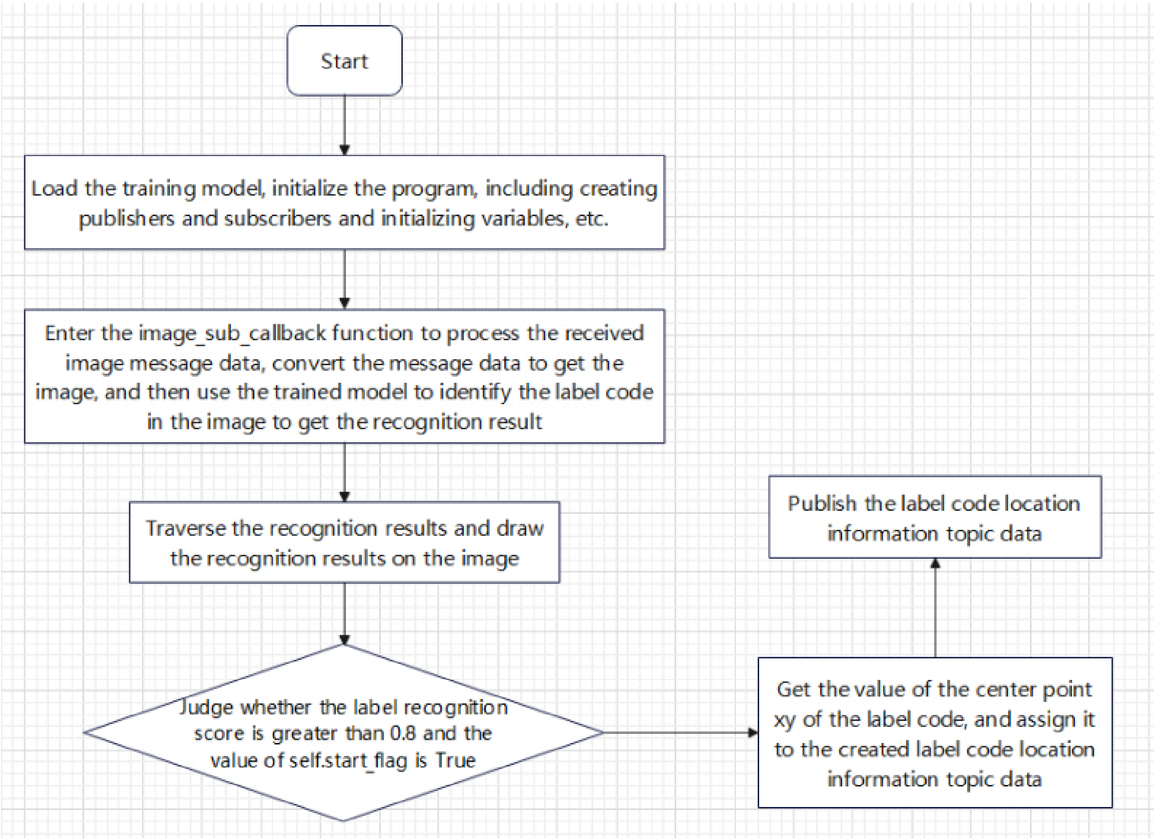
2.2. Operation process

After the program starts, place the wooden block with the garbage label code in the middle of the image. The wooden block needs to be straightened and the icon needs to be in the same direction as the robot arm (forward, Y axis direction). Press the space bar to start recognition. The robot arm will grasp the wooden block and place it in the corresponding position according to the type of garbage identified.

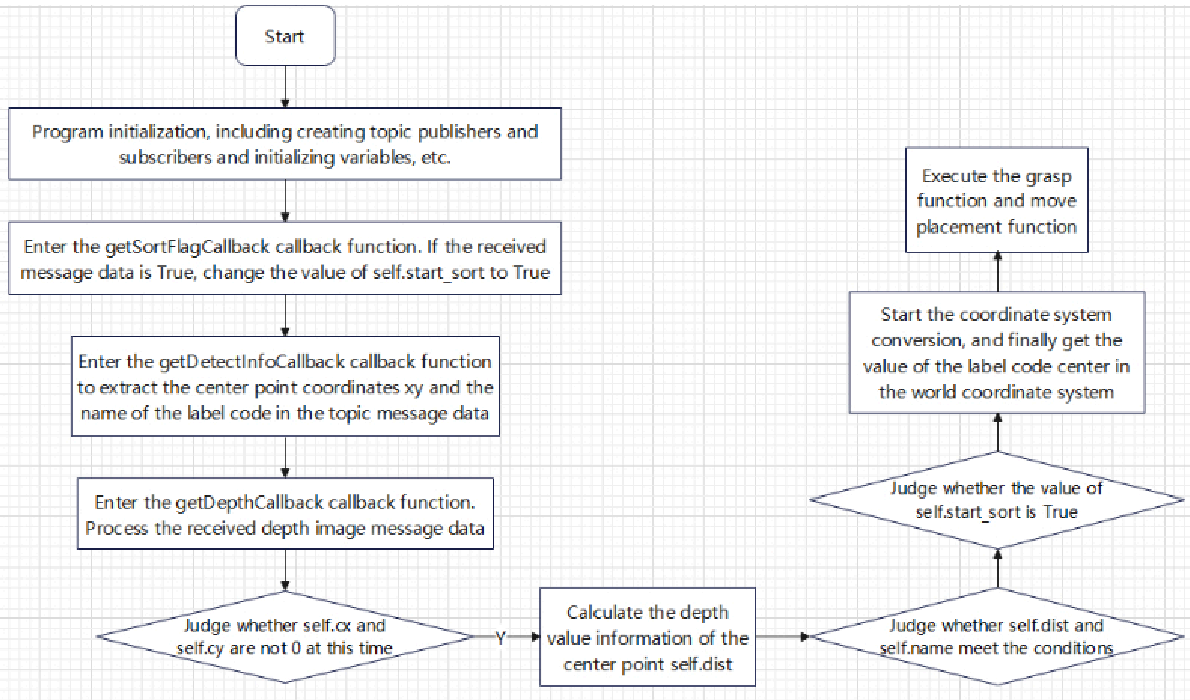


3. Program flow chart

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
```

Initialize program parameters, create publishers and subscribers,

```
def __init__(self):
    #Create a bridge for color image topic message data to image data
    self.bridge = CvBridge()
    #Create a color image topic subscriber
    self.image_sub =
rospy.Subscriber("/camera/color/image_raw",Image,self.image_sub_callback)
    #Create an 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 the message object of the image data
    self.image = Image_Msg()
```

image_sub_callback callback function,

```
def image_sub_callback(self, data):
    #Receive a color image topic message and convert the message data into image
data
    self.img = self.bridge.imgmsg_to_cv2(data, "bgr8")
    #Get the length and width of the image
    size = self.img.shape
    #Assign values ••to the data in the image message object
    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 an 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']
```

Initialize program parameters, 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_profi
le=1)
    self.img = np.zeros((480, 640, 3), dtype=np.uint8)
    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.e
ngine", task='detect')
    self.fps = FPS()
```

The callback function for subscribing to the image topic,

```
 def image_sub_callback(self,data):
        image = np.ndarray(shape=(data.height, data.width, data.channels),
dtype=np.uint8, buffer=data.data)
        self.img[:,:,0],self.img[:,:,1],self.img[:,:,2] =
image[:,:,2],image[:,:,1],image[:,:,0] #
        # self.img = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)


        results = self.yolo_model(self.img, save=False, verbose=False)
        annotated_frame = results[0].plot(
            labels = True,
            conf = False,
            boxes = True,
        )
        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}"
```

```
                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)
                cv2.putText(annotated_frame, label, (x_min, y_min - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

                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_sort
ation.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
import threading

from ament_index_python import get_package_share_directory
import yaml
import os
```

```
from Arm_Lib import Arm_Device
```

Open the 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']
```

Initialize program parameters, 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 modification
     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, qos_profile=1)
    self.sub_SortFlag = self.create_subscription(Bool, 'sort_flag',
self.getSortFlagCallback, qos_profile=10)
    # Service client modification
    self.client = self.create_client(Kinemarics, "dofbot_kinemarics")
    # Initialize the grip flag. When the value is True, it means that gripping 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 built-in 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 the relative position of the camera and
the end of the robotic arm
```

```
    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 the position and posture information of the current end of the robot
    self.get_current_end_pos()
    #Current label center coordinate value
    self.cx = 320
    self.cy = 240
    #Garbage label name
    self.name = None
    #Read the content of the offset parameter table and assign it to the offset
parameter
    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")
```

The callback function getDetectInfoCallback of the identified spam tag result,

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

The callback function getDepthCallback of the depth image topic,

```
def getDepthCallback(self,msg):
    #Process the received deep 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)
    #Judge whether the values ••of self.cy and self.cx are both not 0
    if self.cy!=0 and self.cx!=0:
        #Get the depth value of the 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)
        #Judge whether the depth value of the center point is not 0 and the value
of self.name is not none
        if self.dist!=0 and self.name!=None:
```

```
            #Judge whether self.start_sort is True
            if self.start_sort == True:
                #Start coordinate system conversion, and finally output the
position of the center point in the 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 the offset parameter to compensate for the deviation caused
by the difference in servo values
                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 the clamping thread, the parameter passed in is the
position of the label just calculated in the world coordinate system
                grasp = threading.Thread(target=self.grasp, args=(pose_T,))
                grasp.start()
                grasp.join()
```

The callback function getSortFlagCallback that starts sorting topics.

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

Grasp function grasp,

```
def grasp(self,pose_T):
    print("------------------------------------------------")
    print("pose_T: ",pose_T)
    #Call the ik algorithm in the inverse solution service to calculate the
values ••of the six servos
    request = kinemaricsRequest()
    #The target x value at the end of the robot arm, in m
    request.tar_x = pose_T[0]
    #The target y value at the end of the robot arm, in m
    request.tar_y = pose_T[1]
    #The target z value at the end of the robot arm, in m, 0.2 is the scaling
factor, make slight adjustments based on actual conditions
    request.tar_z = pose_T[2] +
(math.sqrt(request.tar_y**2+request.tar_x**2)-0.181)*0.2
    #Specify the service content as ik
    request.kin_name = "ik"
    #The target Roll value at the end of the robot arm, in radians, this value is
the current roll value at the end of the robot arm
    request.Roll = self.CurEndPos[3]
```

```python
    print("calcutelate_request: ",request)
    try:
        response = self.client.call(request)
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        #Assign the joint1-joint6 values returned by the call service 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 grabbing, call the move function, determine which garbage list it belongs to based on the value of self.name, and place it in the set position
        self.move()
    except Exception:
       rospy.loginfo("run error")
```