Machine code ID sorting

Before starting this function, you need to close the process of the big program and APP. Enter the following program in the terminal to close the process of the big program and APP.

```
sh ~/app_Arm/kill_YahboomArm.sh
sh ~/app_Arm/stop_app.sh
```

If you need to start the big program and APP again later, start the terminal.

```
sudo systemctl start yahboom_arm.service
sudo systemctl start yahboom_app.service
```

1. Function description

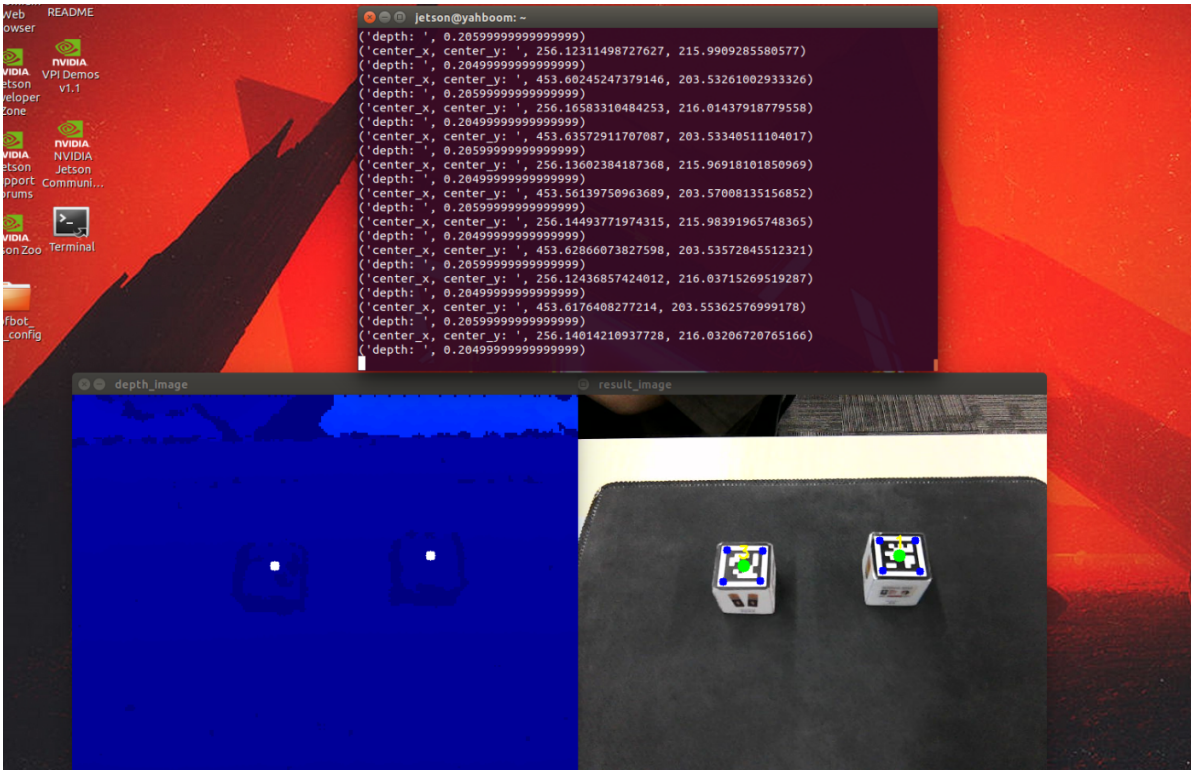
After the program is started, the camera recognizes the machine code, and the machine code will be framed in the screen and the corresponding ID value will be printed. Press the space bar to start the sorting program. The robot arm will clamp the recognized machine code block and place the machine code in the specified position according to the recognized ID.

2. Start and operate

2.1. Start command

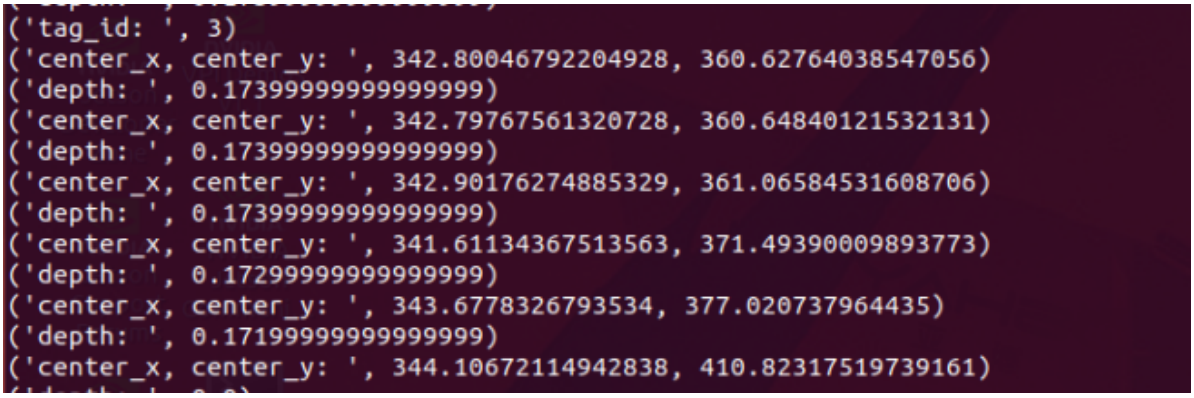
After opening the terminal, enter the following command,

```
#Start the camera
roslaunch orbbec_camera dabai_dcw2.launch
#Start the underlying control robot
rosrun dofbot_pro_info arm_driver.py
#Start the inverse solution program
rosrun dofbot_pro_info kinemarics_dofbot_pro
#Start the detection machine code recognition program
rosrun dofbot_pro_apriltag apriltag_detect.py
#Start the robot gripping program
rosrun dofbot_pro_info grasp.py
```



2.2, Operation

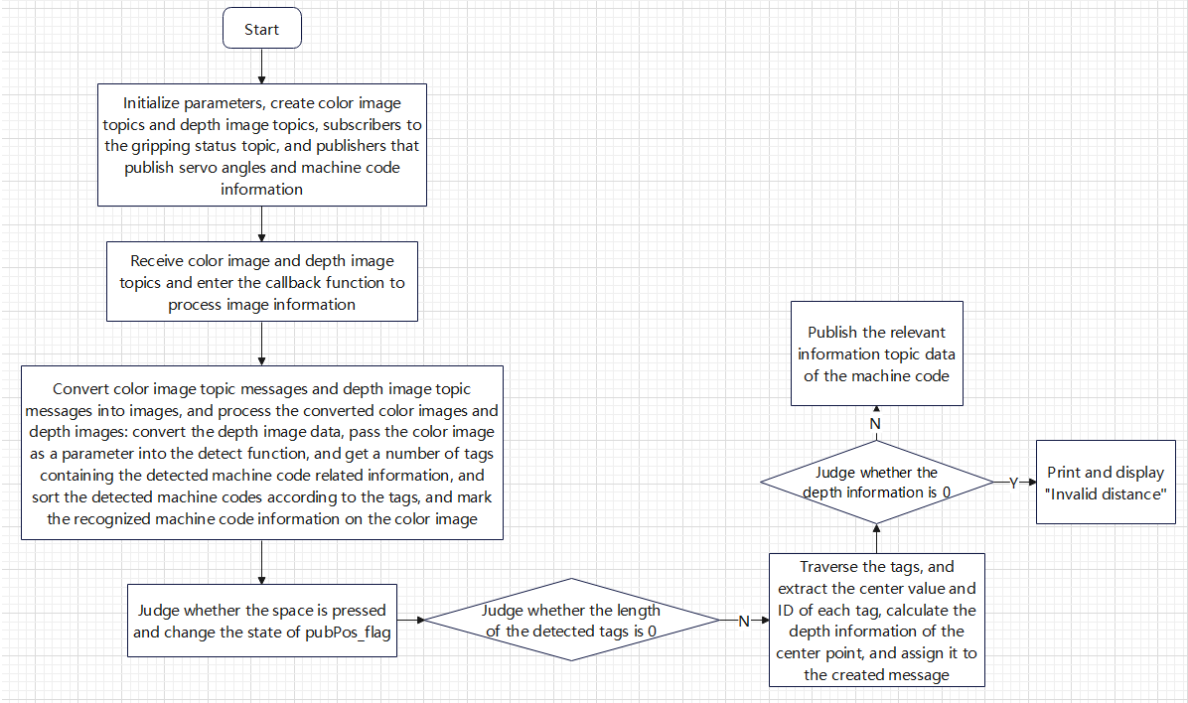
Use **3*3 square wooden block**, click the image frame with the mouse, and then press the space bar on the keyboard. The robot arm will perform a series of calculations based on the center coordinates of the machine code wooden block and the depth value of the xy coordinates of the center point. Then, according to the calculation results, the lower claw will clamp the recognized machine code wooden block; after clamping, the machine code will be placed at the set position according to the ID value of the clamped machine code; after placement, it will return to the recognition posture and recognize the next machine code. Two terminals print out the center value of the recognized machine code and the depth information of the center value, and the other will print the real-time calculation process.



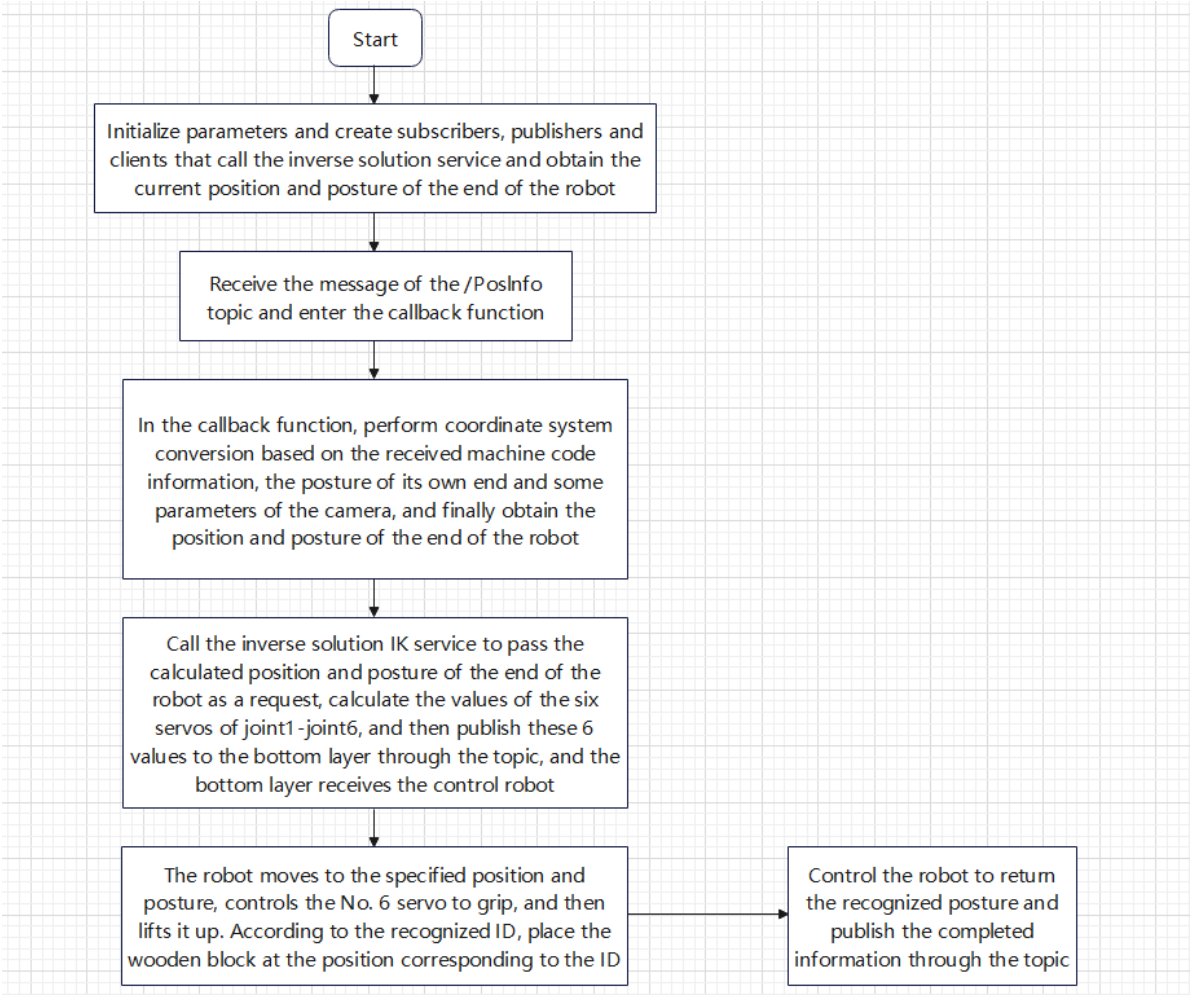
```
Take it now.
------------------------------------------------
('pose_T: ', array([ 0.0079323 ,  0.15784294,  0.02849252]))
('calcutelate_request: ', tar_x: -0.00206770085958
tar_y: 0.172842936318
tar_z: 0.0319493800691
Roll: -1.04719753092
Pitch: 0.0
Yaw: 0.0
cur_joint1: 0.0
cur_joint2: 0.0
cur_joint3: 0.0
cur_joint4: 0.0
cur_joint5: 0.0
cur_joint6: 0.0
kin_name: "ik")
('compute_joints: ', [90.486102912174, 83.7127367211455, -13.726199738519027, 50
.01256880572116, 90, 30])
('self.gripper_joint = ', 90)
```

3. Program flow chart

apriltag_detect.py



grasp.py



4. Core code analysis

4.1、apriltag_detect.py

Code path:

`/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/scripts/apriltag_detect.py`

Import necessary library files

```
import os
import rospy
import numpy as np
from sensor_msgs.msg import Image
import message_filters
#Import drawing machine code library
from vutils import draw_tags
#Import machine code library
from dt_apriltags import Detector
from cv_bridge import CvBridge
import cv2 as cv
#Import custom service data types
from dofbot_info.srv import kinemarics, kinemaricsRequest, kinemaricsResponse
#Import custom message data types
from dofbot_pro_info.msg import ArmJoint
from dofbot_pro_info.msg import AprilTagInfo
from std_msgs.msg import Float32,Bool
encoding = ['16UC1', '32FC1']
```

```
import time
```

Initialize program parameters, create publishers and subscribers

```
def __init__(self):
    rospy.init_node('apriltag_detect')
    #self.init_joints = [90.0, 120, 0, 0.0, 90, 90]
    #发布机械臂的初始姿态，同样也是识别的姿态
    #Publish the initial posture of the robot arm, which is also the recognized
posture
    self.init_joints = [90.0, 120, 0.0, 0.0, 90, 30]
    #创建两个订阅者，订阅彩色图像话题和深度图像话题
    #Create two subscribers to subscribe to the color image topic and the depth
image topic
    self.depth_image_sub =
message_filters.Subscriber('/camera/depth/image_raw',Image)
    self.rgb_image_sub =
message_filters.Subscriber('/camera/color/image_raw',Image)
    #创建发布机器码信息的发布者
    #Create a publisher that publishes machine code information
    self.tag_info_pub = rospy.Publisher("TagInfo",AprilTagInfo,queue_size=1)
    #创建发布机械臂目标角度的发布者
    #Create a publisher that publishes the target angle of the robotic arm
    self.pubPoint = rospy.Publisher("TargetAngle", ArmJoint, queue_size=1)
    #将彩色和深度图像订阅的消息进行时间同步
    #Synchronize the time of color and depth image subscription messages
    self.TimeSynchronizer =
message_filters.ApproximateTimeSynchronizer([self.rgb_image_sub,self.depth_image
_sub],1,0.5)
    #创建订阅夹取结果的订阅者
    #Create a subscriber to subscribe to the fetch results
    self.grasp_status_sub = rospy.Subscriber('grasp_done', Bool,
self.GraspStatusCallback, queue_size=1)
    #处理同步消息的回调函数TagDetect，回调函数与订阅的消息连接起来，以便在接收到新消息时自动调
用该函数
    #The callback function TagDetect that handles the synchronization message is
connected to the subscribed message so that it can be automatically called when a
new message is received
    self.TimeSynchronizer.registerCallback(self.TagDetect)
    #创建彩色和深度图像话题消息数据转图像数据的桥梁
    #Create a bridge for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
    #发布机器码信息的标识，为True时发布/TagInfo话题数据
    #The flag for publishing machine code information. When it is True, it will
publish the /TagInfo topic data.
    self.pubPos_flag = False
    #创建机器码的对象，设定一些参数，参数如下，
    #Create a machine code object and set some parameters. The parameters are as
follows:
    '''
    searchpath: specifies the path to search for the tag model.
    families: sets the tag family to be used, for example 'tag36h11'.
    nthreads: the number of threads for parallel processing to increase detection
speed.
```

```
    quad_decimate: reduces the resolution of the input image to reduce the amount
of calculation.
    quad_sigma: the standard deviation of the Gaussian blur, which affects image
preprocessing.
    refine_edges: whether to refine the edges to improve detection accuracy.
    decode_sharpening: the sharpening parameter during decoding to enhance the
label contrast.
    debug: the debug mode switch to facilitate viewing of information during the
detection process
    '''
    self.at_detector = Detector(searchpath=['apriltags'],
                                families='tag36h11',
                                nthreads=8,
                                quad_decimate=2.0,
                                quad_sigma=0.0,
                                refine_edges=1,
                                decode_sharpening=0.25,
                                debug=0)
    #通过调用相机服务，设置相机的曝光值为50
    #By calling the camera service, set the camera exposure value to 50
    exit_code = os.system('rosservice call /camera/set_color_exposure  50')
```

Mainly look at the TagDetect callback function,

```
def TagDetect(self,color_frame,depth_frame):
    #rgb_image
    #接收到彩色图像话题消息，把消息数据转换成图像数据
    #Receive the color image topic message and convert the message data into
image data
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'rgb8')
    result_image = np.copy(rgb_image)
    #depth_image
    #接收到深度图像话题消息，把消息数据转换成图像数据
    #Receive the deep image topic message and convert the message data into image
data
    depth_image = self.depth_bridge.imgmsg_to_cv2(depth_frame, encoding[1])
    #把深度图像转换成伪彩色图像
    #Convert the depth image into a pseudo-color image
    depth_to_color_image = cv.applyColorMap(cv.convertScaleAbs(depth_image,
alpha=0.03), cv.COLORMAP_JET)
    frame = cv.resize(depth_image, (640, 480))
    depth_image_info = frame.astype(np.float32)
    #调用detect函数，传入参数，
    #Call the detect function and pass in the parameters
    '''
    cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY): Converts an RGB image to a
grayscale image for label detection.
    False: Indicates that the label's pose is not estimated.
    None: Indicates that no camera parameters are provided, and only simple
detection may be performed.
    0.025: It may be the set label size (usually in meters), which is used to
help the detection algorithm determine the size of the label.
    1.Returns a detection result, including information such as the location, ID,
and bounding box of each label.
    '''
    tags = self.at_detector.detect(cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY),
False, None, 0.025)
```

```
    #给tags里边的各个标签进行排序，非必须步骤
    #Sort the tags in tags, not a necessary step
    tags = sorted(tags, key=lambda tag: tag.tag_id)
    #调用draw_tags函数，作用是在彩色图像上描绘出识别的机器码相关的信息，包括角点，中心点和id值
    #Call the draw_tags function to draw the information related to the
recognized machine code on the color image, including corner points, center
points and id values
    draw_tags(result_image, tags, corners_color=(0, 0, 255), center_color=(0,
255, 0))
    #等待键盘的输入，32表示空格按下，按下后改变self.pubPos_flag的值，表示可以发布机器码相关
信息了
    #Wait for keyboard input, 32 means the space key is pressed, after pressing
it, the value of self.pubPos_flag is changed, indicating that the machine code
related information can be released
    key = cv2.waitKey(10)
    if key == 32:
        self.pubPos_flag = True、
    #判断tags的长度，大于0则表示有检测到机器码
    #Judge the length of tags. If it is greater than 0, it means that the machine
code has been detected.
    if len(tags) > 0 :
        #遍历机器码 Traversing the machine code
        for i in range(len(tags)):
            if self.pubPos_flag == True:
                #获取识别机器码的中心值
                #Get the center value of the recognition machine code
                center_x, center_y = tags[i].center
                #在伪彩色图像上标记机器码木块的中心点
                #Mark the center point of the machine code block on the pseudo-
color image
                cv.circle(result_image, (int(center_x),int(center_y)), 10,
(0,210,255), thickness=-1)
                #创建机器码信息的消息数据
                #Create message data for machine code information
                tag = AprilTagInfo()
                #给消息数据赋值，id值为机器码的id，x和y为机器码的中心值，z为中心点的深度值，
这里做了按比例缩小1000倍数，单位是米
                #Assign values ••to the message data. The id value is the id of
the machine code. x and y are the center values ••of the machine code. z is the
depth value of the center point. Here, it is scaled down by 1000 times. The unit
is meter.
                tag.id = tags[i].tag_id
                tag.x = center_x
                tag.y = center_y
                tag.z = depth_image_info[int(center_y),int(center_x)]/1000
                #打印识别机器码信息的信息
                #Print information to identify machine code information
                print("tag_id: ",tags[i].tag_id)
                print("center_x, center_y: ",center_x, center_y)
                print("depth:
",depth_image_info[int(center_y),int(center_x)]/1000)
                #判断如果机器码的距离大于0，说明为有效数据，然后发布机器码信息的消息
                #If the machine code distance is greater than 0, it means it is
valid data, and then publish the message of machine code information
                if tag.z>0:
                    self.tag_info_pub.publish(tag)
                    #改变self.pubPos_flag状态，防止多次发布信息，等待夹取完成后再改变状态
```

```
                    #Change the state of self.pubPos_flag to prevent multiple
publishing of information, and wait until the clamping is completed before
changing the state
                    self.pubPos_flag = False
                else:
                    print("Invalid distance.")
    #转换彩色图的颜色空间，把RGB转换成BGR
    #Convert the color space of the color image, convert RGB to BGR
    result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
    #显示图像  Display the image
    cv2.imshow("result_image", result_image)
    cv2.imshow("depth_image", depth_to_color_image)
    key = cv2.waitKey(1)
```

Callback function for gripping status

```
def GraspStatusCallback(self,msg):
    #接收到消息数据如果为真，说明夹取完成，改成self.pubPos_flag，表示可以发布下一帧的消息
    #If the received message data is true, it means that the clamping is
completed, and it is changed to self.pubPos_flag, indicating that the message of
the next frame can be released
    if msg.data == True:
        self.pubPos_flag = True
```

4.2、grasp.py

Code path:
`/home/jetson/dofbot_pro_ws/src/dofbot_pro_info/scripts/grasp.py`

Import necessary library files

```
import rospy
import numpy as np
from std_msgs.msg import Float32,Bool
import time
import math
from dofbot_pro_info.msg import *
from dofbot_pro_info.srv import *
#导入transforms3d 库用于处理三维空间中的变换，执行四元数、旋转矩阵和欧拉角之间的转换，支持三维
几何操作和坐标转换
#Import the transforms3d library to handle transformations in three-dimensional
space, perform conversions between quaternions, rotation matrices, and Euler
angles, and support three-dimensional geometric operations and coordinate
conversions
import transforms3d as tfs
#导入transformations处理和计算三维空间中的变换，包括四元数和欧拉角之间的转换
#Import transformations to process and calculate transformations in three-
dimensional space, including conversions between quaternions and Euler angles
import tf.transformations as tf
import threading
import rospkg
import yaml
```

Open the offset parameter table,

```
offset_file = rospkg.RosPack().get_path("dofbot_pro_info") +
"/param/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'))
class TagGraspNode:
```

Initialize program parameters, create publishers, subscribers, and clients

```
def __init__(self):
    nodeName = 'apriltag_grap'
    rospy.init_node(nodeName)
    #创建订阅TagInfo话题的订阅者，订阅机器码信息的消息
    #Create a subscriber to subscribe to the TagInfo topic and subscribe to
messages about machine code information
    self.tag_info_sub = rospy.Subscriber("PosInfo", AprilTagInfo,
self.tag_info_callback, queue_size=1)
    #创建发布舵机目标角度的话题的发布者，发布控制机械臂舵机的消息
    #Create a publisher for the topic of the target angle of the servo, and
publish messages to control the robotic arm servo
    self.pubPoint = rospy.Publisher("TargetAngle", ArmJoint, queue_size=1)
    #创建发布夹取结果的话题的发布者，发布夹取结果的消息
    #Create a publisher for the topic of clamping results and publish the message
of clamping results
    self.pubGraspStatus = rospy.Publisher("grasp_done", Bool, queue_size=1)
    #创建请求逆解算服务的客户端，用于计算当前机械臂末端位置和位姿以及解算目标的舵机值
    #Create a client requesting the inverse solution service to calculate the
current end position and posture of the robotic arm and the servo value of the
solution target
    self.client = rospy.ServiceProxy("get_kinemarics", kinemarics)
    #初始夹取的标识，True为可夹取，False为不可夹取
    #Initial gripping flag, True means grippable, False means not grippable
    self.grasp_flag = True
    #机械臂初始化姿态
    # Initialize the robot arm posture
    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.gripper_joint = 90
    #初始化当前位置位姿，对应的是x、y、z、roll、pitch和yaw
    #Initialize the current position and posture, corresponding to x, y, z, roll,
pitch and yaw
    self.CurEndPos = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    #深度相机的内参
    #Internal parameters of the depth camera
    self.camera_info_K = [477.57421875, 0.0, 319.3820495605469, 0.0,
477.55718994140625, 238.64108276367188, 0.0, 0.0, 1.0]
    #机械臂末端与相机的旋转变换矩阵，描述了两者之间的相对位置和位姿
    #The rotation transformation matrix of the end of the robotic arm and the
camera describes the relative position and posture between the two
    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]])
    #获取当前的机械臂末端的位置和位姿，会改变self.CurEndPos的值
    #Get the current position and posture of the end of the robot arm, which will
change the value of self.CurEndPos
    self.get_current_end_pos()
    #定义当前的id值，后边根据该值把机器码放置在对应的位置
    #Define the current id value, and then place the machine code in the
corresponding position according to the value
    self.cur_tagId = 0
    #读取偏移量参数表内容，赋值给偏移量参数
    #Read the contents of the offset parameter table and assign them to the
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')
    #打印当前机械臂末端的位置位姿
    #Print the current position and posture of the end of the robot arm
    print("Current_End_Pose: ",self.CurEndPos)
    print("Init Done")
```

Machine code information callback function tag_info_callback,

```
def tag_info_callback(self,msg):
    pos_x = msg.x
    pos_y = msg.y
    pos_z = msg.z
    self.cur_tagId = msg.id
    #判断如果接收到的中心点的深度信息不为>0,说明为有效数据
    # If the received center point depth information is not > 0, it means it is
valid data
    if pos_z!=0.0:
        print("xyz id : ",pos_x,pos_y,pos_z,self.cur_tagId)
        #第一次坐标系转换，由像素坐标系转换到相机坐标系下
        #The first coordinate system conversion, from the pixel coordinate
system to the camera coordinate system
        camera_location = self.pixel_to_camera_depth((pos_x,pos_y),pos_z)
        #print("camera_location: ",camera_location)
        #第二次坐标系转换，由相机坐标系转换到机械臂末端的坐标系下
        #The second coordinate system conversion, from the camera coordinate
system to the coordinate system of the end of the robotic arm
        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()
        #第三次坐标系转换，由机械臂末端的坐标系转换到基座标系下，得到的worldPose（旋转变换矩
阵）就是机器码的中心相对应机械臂基座标系的位置和姿态
        #The third coordinate system conversion is to convert the coordinate
system of the end of the robot arm to the base coordinate system. The obtained
worldPose (rotation transformation matrix) is the position and posture of the
center of the machine code corresponding to the base coordinate system of the
robot arm.
```

```
        WorldPose = np.matmul(EndPointMat, PoseEndMat)
        #WorldPose = np.matmul(PoseEndMat,EndPointMat)
        #把旋转变换矩阵转换成xyz和欧拉角
        #Convert the rotation transformation matrix into xyz and Euler angles
        pose_T, pose_R = self.mat_to_xyz_euler(WorldPose)
        #加上偏移量参数，补偿由于舵机数值差异导致的偏差
        #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
        print("pose_T: ",pose_T)
        #判断夹取的标识，为True则表示下爪夹取
    if self.grasp_flag == True :
        print("Take it now.")
        #改变夹取的标识，防止在夹取过程中识别到再次执行夹取
        #Judge the gripping flag. If True, it means the lower claw is gripping.
        self.grasp_flag = False
        #开启一个线程，线程执行grasp的程序，参数是刚才计算的到的pose_T,也就是xyz的值，表示机
械臂末端的目标位置
        #Start a thread, the thread executes the grasp program, the parameter is
the pose_T just calculated, that is, the value of xyz, indicating the target
position of the end of the robot arm
        grasp = threading.Thread(target=self.grasp, args=(pose_T,))
        #执行线程 Thread of execution
        grasp.start()
        grasp.join()
```

The function grasp of the robot arm's lower claw

```
def grasp(self,pose_T):
    print("--------------------------------------------------")
    print("pose_T: ",pose_T)
    #调用逆解算的服务，调用的是ik服务内容，把需要的request参数赋值进去
    #Call the inverse solution service, call the ik service content, and assign
the required request parameters
    request = kinemaricsRequest()
    #机械臂末端的目标x值，单位是m
    #The target x value at the end of the robotic arm, in m
    request.tar_x = pose_T[0] -0.01
    #机械臂末端的目标y值，单位是m
    #The target y value at the end of the robotic arm, in m
    request.tar_y = pose_T[1]
    #机械臂末端的目标z值，单位是m，0.2为缩放系数，根据实际情况进行微小的调整
    #The target z value at the end of the robot arm, in meters, 0.2 is the
scaling factor, and small adjustments are made based on actual conditions
    request.tar_z = pose_T[2] +
(math.sqrt(request.tar_y**2+request.tar_x**2)-0.181)*0.2
    #指定服务的内容为ik
    #Specify the service content as ik
    request.kin_name = "ik"
    #机械臂末端的目标Roll值，单位是弧度，该值为当前的机械臂末端的roll值
    #The target Roll value at the end of the robot arm, in radians, is the
current roll value at the end of the robot arm
    request.Roll = self.CurEndPos[3]
    print("calcutelate_request: ",request)
```

```
    try:

        response = self.client.call(request)
        #print("calcutelate_response: ",response)
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        #调用服务返回的joint1-joint6值赋值给joints
        #Assign the joint1-joint6 values ••returned by the call service to
joints
        joints[0] = response.joint1 #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)
        #执行pubTargetArm函数，把计算得到的joints值作为参数传入
        #Execute the pubTargetArm function and pass the calculated joints value
as a parameter
        self.pubTargetArm(joints)
        time.sleep(3.5)
        #执行move函数，夹取木块根据机器码的id值放置在设定的位置
        #Execute the move function, grab the block and place it at the set
position according to the machine code ID value
        self.move()

        except Exception:
            rospy.loginfo("run error")
```

Publish the robot arm target angle function pubTargetArm

```
def pubArm(self, joints, id=1, angle=90, run_time=2000):
    armjoint = ArmJoint()
    armjoint.run_time = run_time
    if len(joints) != 0: armjoint.joints = joints
    else:
    armjoint.id = id
    armjoint.angle = angle
    self.pubPoint.publish(armjoint)
```

Grab and place function move

```
def move(self):
    print("self.gripper_joint = ",self.gripper_joint)
    self.pubArm([],5, self.gripper_joint, 2000)
    time.sleep(2.5)
    self.pubArm([],6, 140, 2000)
    time.sleep(2.5)
    self.pubArm([],2, 120, 2000)
    time.sleep(2.5)
    #根据id值，改变self.down_joint的值，该值表示机器码木块放置的位置
    #According to the id value, change the value of self.down_joint, which
indicates the position of the machine code block.
    if self.cur_tagId == 1:
```

```
            self.down_joint = [150.0, 30, 70, 5, 90.0,140]
        elif self.cur_tagId == 2:
            self.down_joint = [180.0, 35, 60, 0, 90.0,140]
        elif self.cur_tagId == 3:
            self.down_joint = [28.0, 30, 70, 2, 90.0,140]
        elif self.cur_tagId == 4:
            self.down_joint = [0.0, 43, 48, 6, 90.0,140]
        self.pubArm(self.down_joint)
        time.sleep(2.5)
        self.pubArm([],6, 90, 2000)
        time.sleep(2.5)
        self.pubArm([],2, 90, 2000)
        time.sleep(2.5)
        #放置完成后，回到初始位置
        #After placement is completed, return to the initial position
        self.pubArm(self.init_joints)
        time.sleep(5)
        #等待机械臂回到初始位置后，发布抓取完成的消息，改变抓取的标识值以便于下一次符合条件后下爪夹
取
        #After waiting for the robot arm to return to the initial position, publish
the message that the grasping is completed, and change the grasping
identification value so that the next time the conditions are met, the lower claw
will grasp
        self.grasp_flag = True
        grasp_done = Bool()
        grasp_done.data = True
        self.pubGraspStatus.publish(grasp_done)
```