Sorting of abnormal machine code height

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

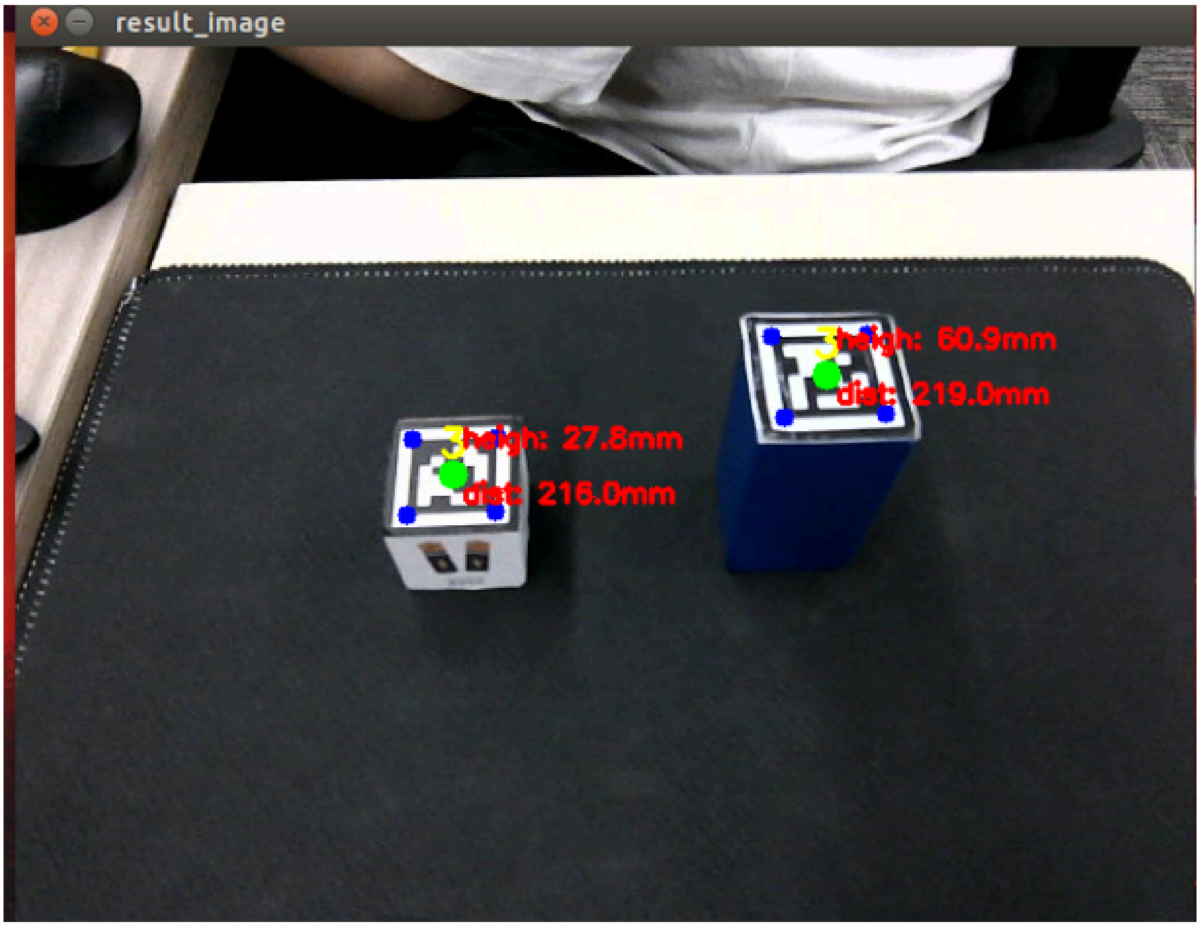
After the program is started, the camera recognizes the machine code, and it will clamp the machine code with a height higher than the set height and place it in the set position.

2. Start and operate

2.1. Start command

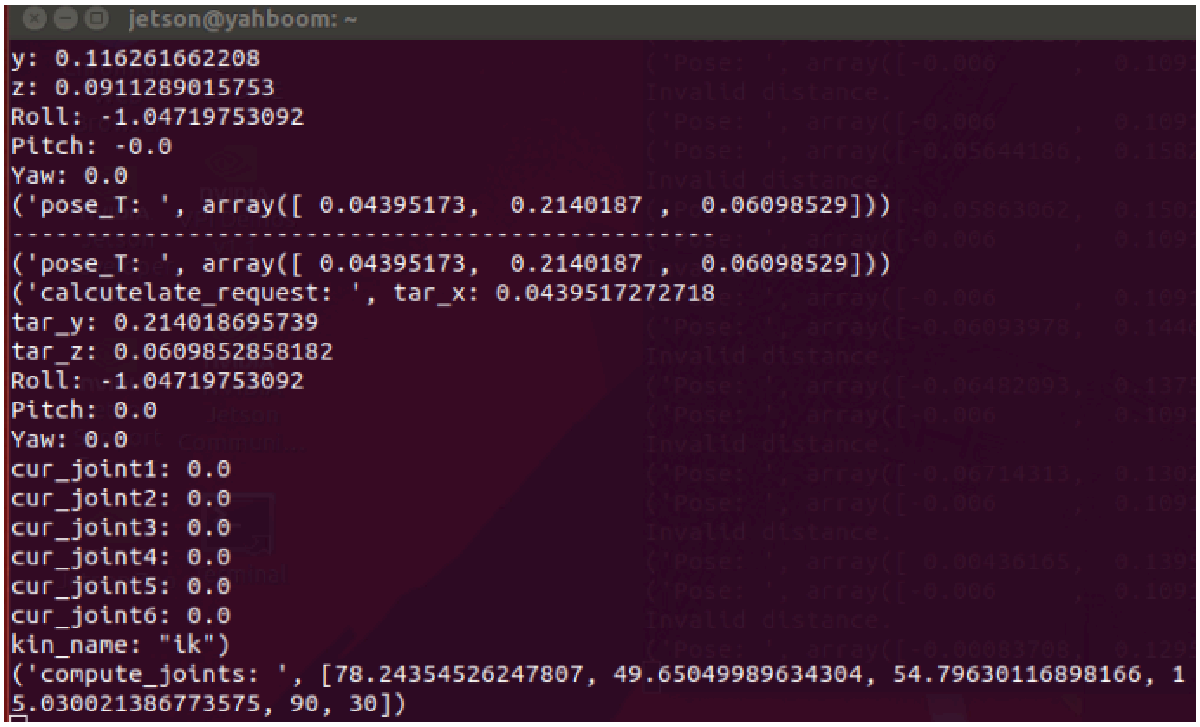
Terminal input,

```
#Start the camera
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start the inverse solution program
ros2 run dofbot_pro_info kinemarics_dofbot
#Start the underlying control
ros2 run dofbot_pro_driver arm_driver
#Start the detection machine code recognition program
ros2 run dofbot_pro_driver apriltag_list
#Start the robot arm to grab the machine code program
ros2 run dofbot_pro_driver apriltag_remove_higher
```



2.2. Operation

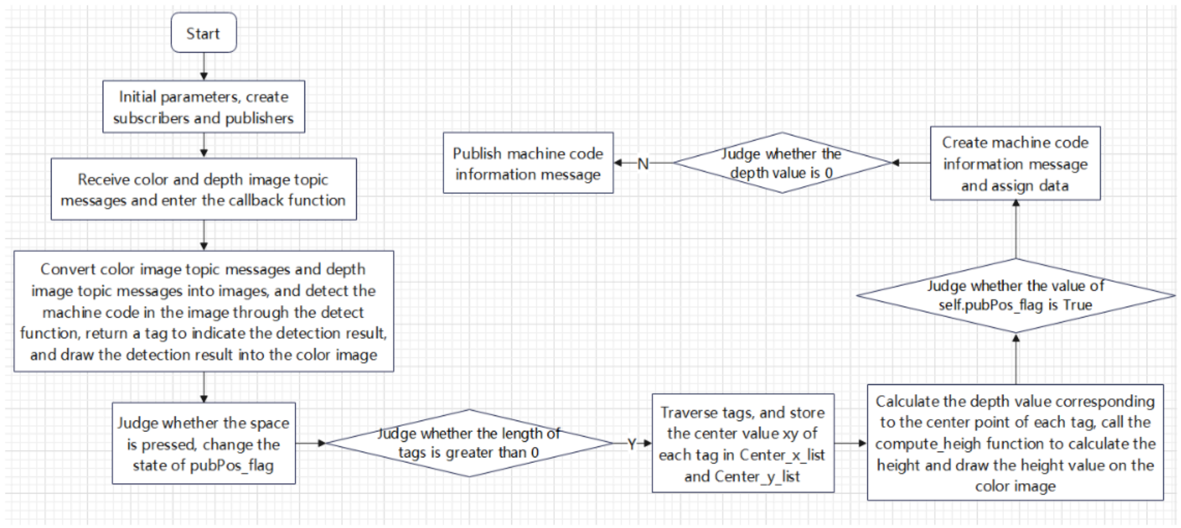
Click the image frame with the mouse, and then press the space bar on the keyboard. The robot arm will grab the machine code wooden block that is higher than the set height, and then place it in the set position. After the placement is completed, it will return to the recognition posture, continue to recognize the machine code and detect whether there is a machine code wooden block with abnormal height. The second time the wooden block with abnormal height is recognized, there is no need to press the space bar to grab it. The terminal will print the clamping information.



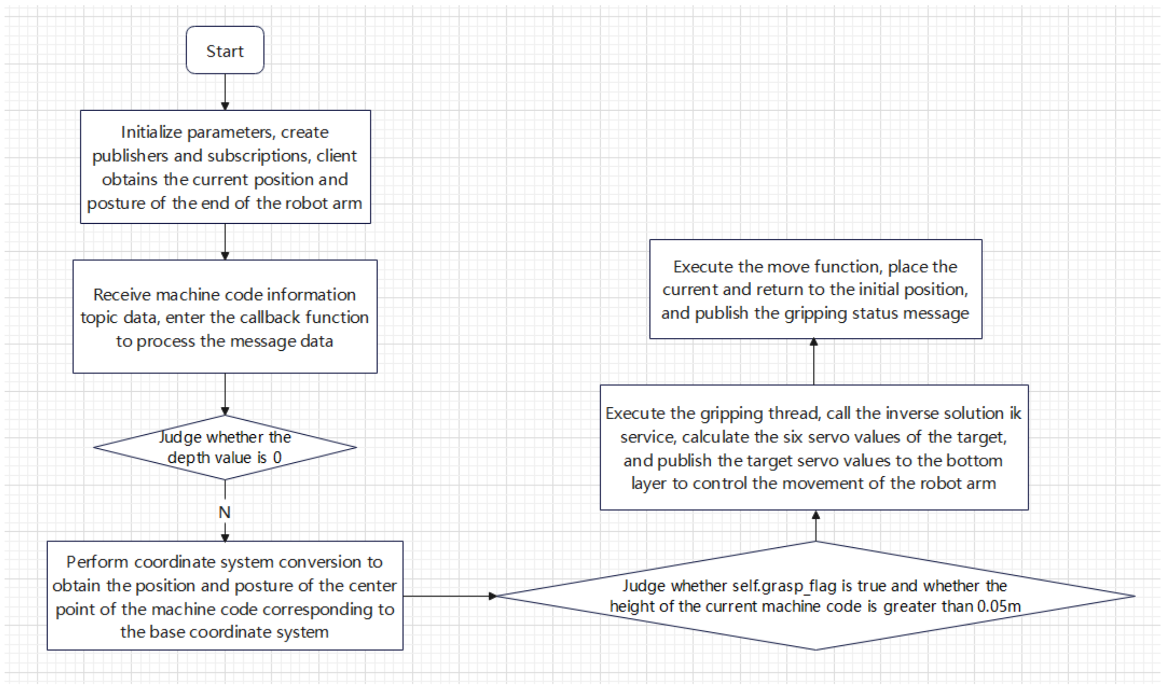
```
y: 0.116261662208
z: 0.0911289015753
Roll: -1.04719753092
Pitch: -0.0
Yaw: 0.0
('pose_T: ', array([ 0.04395173,  0.2140187 ,  0.06098529]))
-------------------------------------------------------
('pose_T: ', array([ 0.04395173,  0.2140187 ,  0.06098529]))
('calcutelate_request: ', tar_x: 0.0439517272718
tar_y: 0.214018695739
tar_z: 0.0609852858182
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: ', [78.24354526247807, 49.65049989634304, 54.79630116898166, 1
5.030021386773575, 90, 30])
```

3. Program flow chart

3.1. apriltag_list.py


Start
Initial parameters, create subscribers and publishers
Receive color and depth image topic messages and enter the callback function
Convert color image topic messages and depth image topic messages into images, and detect the machine code in the image through the detect function, return a tag to indicate the detection result, and draw the detection result into the color image
Judge whether the space is pressed, change the state of pubPos_flag
Judge whether the length of tags is greater than 0
Y
Traverse tags, and store the center value xy of each tag in Center_x_list and Center_y_list
Calculate the depth value corresponding to the center point of each tag, call the compute_heigh function to calculate the height and draw the height value on the color image
Judge whether the value of self.pubPos_flag is True
Create machine code information message and assign data
Judge whether the depth value is 0
N
Publish machine code information message

3.2. apriltag_remove_higher.py


Start
Initialize parameters, create publishers and subscriptions, client obtains the current position and posture of the end of the robot arm
Receive machine code information topic data, enter the callback function to process the message data
Judge whether the depth value is 0
N
Perform coordinate system conversion to obtain the position and posture of the center point of the machine code corresponding to the base coordinate system
Judge whether self.grasp_flag is true and whether the height of the current machine code is greater than 0.05m
Execute the gripping thread, call the inverse solution ik service, calculate the six servo values of the target, and publish the target servo values to the bottom layer to control the movement of the robot arm
Execute the move function, place the current and return to the initial position, and publish the gripping status message

4. Core code analysis

4.1. apriltag_list.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/apriltag_list.
py
```

Import necessary library files

```
import rclpy
from rclpy.node import Node
import numpy as np
from sensor_msgs.msg import Image
from message_filters import ApproximateTimeSynchronizer, Subscriber
#Import drawing machine code library
from dofbot_pro_driver.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_pro_interface.srv import Kinemarics
#Import custom message data types
from dofbot_pro_interface.msg import *
from std_msgs.msg import Float32,Bool
encoding = ['16UC1', '32FC1']
import time
#Import transforms3d The library is used to process 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
#Import transformations to process and calculate transformations in three-
dimensional space, including conversions between quaternions and Euler angles
import tf_transformations as tf
import math
```

程序参数初始化，创建发布者、订阅者和客户端

```
def __init__(self):
super().__init__('apriltag_detect')
    #Publish the initial posture of the robot arm, which is also the recognized
posture
    self.init_joints = [90.0, 120, 0, 0.0, 90, 90]
    #Create two subscribers to subscribe to the color image topic and the depth
image topic
    self.depth_image_sub = Subscriber(self, Image, '/camera/depth/image_raw')
    self.rgb_image_sub = Subscriber(self, Image, '/camera/color/image_raw')
    #Create a publisher to publish machine code information
    self.pos_info_pub = self.create_publisher(AprilTagInfo, "PosInfo",
qos_profile=10)

    #Time synchronize the messages subscribed to color and depth images
    self.ts = ApproximateTimeSynchronizer([self.rgb_image_sub,
self.depth_image_sub],queue_size=10,slop=0.5)
    #Create a subscriber to subscribe to the grab result
    self.subscription =
self.create_subscription(Bool,'grasp_done',self.GraspStatusCallback,qos_profile=1
)
    #Handle the callback function TagDetect for synchronous messages. The
callback function is connected to the subscribed message so that the function can
be automatically called when a new message is received
    self.ts.registerCallback(self.TagDetect)
    #Create a bridge for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
    #Publish the flag of machine code information. When it is True, publish the
/TagInfo topic data
    self.pubPos_flag = False
    #Create a machine code object and set some parameters. The parameters are as
follows,
    '''
    searchpath: Specify the path to find the tag model.
    families: Set the tag family to be used, such as 'tag36h11'.
    nthreads: the number of threads for parallel processing, which improves the
detection speed.
    quad_decimate: reduces the resolution of the input image and reduces the
amount of calculation.
    quad_sigma: the standard deviation of Gaussian blur, which affects image
preprocessing.
    refine_edges: whether to refine the edges to improve detection accuracy.
    decode_sharpening: the sharpening parameter during decoding, which enhances
the label contrast.
    debug: debug mode switch, convenient for viewing information during detection
    '''
    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)
    #List of machine code center x values
    self.Center_x_list = []
    #List of machine code center y values
    self.Center_y_list = []
    self.heigh = 0.0
    #Position and posture of the end of the robot arm
    self.CurEndPos = [-0.006,0.116261662208,0.0911289015753,-1.04719,-0.0,0.0]
    #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]
    #Rotation transformation matrix between the end of the robotic arm and the
camera, describing 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]])
```

Machine code information callback function tag_info_callback,

```
def TagDetect(self,color_frame,depth_frame):
    #rgb_image
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'rgb8')
    result_image = np.copy(rgb_image)
    #depth_image
    depth_image = self.depth_bridge.imgmsg_to_cv2(depth_frame, encoding[1])
    frame = cv.resize(depth_image, (640, 480))
    depth_image_info = frame.astype(np.float32)
    tags = self.at_detector.detect(cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY),
False, None, 0.025)
    tags = sorted(tags, key=lambda tag: tag.tag_id) # 貌似出来就是升序排列的不需要手动
进行排列  It seems that the output is in ascending order and does not need to be
sorted manually
    draw_tags(result_image, tags, corners_color=(0, 0, 255), center_color=(0,
255, 0))
    key = cv2.waitKey(10)
    self.Center_x_list = list(range(len(tags)))
    self.Center_y_list = list(range(len(tags)))
    if key == 32:
        self.pubPos_flag = True
    if len(tags) > 0 :
        for i in range(len(tags)):
            center_x, center_y = tags[i].center
            #机器码的中心xy值存在Center_x_list列表和Center_y_list列表中
            #The center xy values ••of the machine code are stored in the
Center_x_list list and the Center_y_list list
            self.Center_x_list[i] = center_x
            self.Center_y_list[i] = center_y
            cx = center_x
            cy = center_y
            #计算中心坐标的深度值
            #Calculate the depth value of the center coordinate
            cz = depth_image_info[int(cy),int(cx)]/1000
            #调用compute函数，计算机器码的高度，传入的参数是机器码的中心坐标和中心点的深度值，
返回的是一个位置列表，pose[2]表示z值，也就是高度值
            #Call the compute function to calculate the height of the machine
code. The parameters passed in are the center coordinates of the machine code and
the depth value of the center point. The returned value is a position list.
pose[2] represents the z value, which is the height value.
            pose = self.compute_heigh(cx,cy,cz)
            #对高度值进行放大运算，把单位换算成毫米
            #Enlarge the height value and convert the unit into millimeters
            heigh = round(pose[2],4)*1000
            heigh = 'heigh: ' + str(heigh) + 'mm'
            #计算机器码离基坐标系的距离值，对该值进行放大运算，把单位换算成毫米
            #Calculate the distance between the machine code and the base
coordinate system, enlarge the value, and convert the unit into millimeters
            dist_detect = math.sqrt(pose[1] ** 2 + pose[0]** 2)
            dist_detect = round(dist_detect,3)*1000
            dist = 'dist: ' + str(dist_detect) + 'mm'
            #高度和距离值使用opencv绘制在彩色图像上
            #Height and distance values ••are drawn on the color image using
opencv
            cv.putText(result_image, heigh, (int(cx)+5, int(cy)-15),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
            cv.putText(result_image, dist, (int(cx)+5, int(cy)+15),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
            print("Pose: ",pose)
        for i in range(len(tags)):
            if self.pubPos_flag == True:
                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 = self.Center_x_list[i]
                tag.y = self.Center_y_list[i]
                tag.z = depth_image_info[int(tag.y),int(tag.x)]/1000
                #判断如果机器码的距离不等于0，说明为有效数据，然后发布机器码信息的消息
                #If the distance of the machine code is not equal to 0, it means
it is valid data, and then publish the message of the 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)
    key = cv2.waitKey(1)
```

Calculate the height value function compute_heigh

```
def compute_heigh(self,x,y,z):
    #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((x,y),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)
    #获取当前的机械臂末端位置和位姿
    #Get the current end position and posture of the robot arm
    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)
    #返回表示位置的pose_T
    #Return pose_T representing the position
    return pose_T
```

3.2, apriltag_remove_higher.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/apriltag_remov
e_higher.py
```

Import necessary library files

```
import math
import rclpy
from rclpy.node import Node
import numpy as np
from std_msgs.msg import Float32,Bool
import time
from dofbot_pro_interface.msg import * # Need to confirm whether the ROS2 message
package name is consistent
from dofbot_pro_interface.srv import Kinemarics
#Import transforms3d library is used to process 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
#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
```

Initialize program parameters, create publishers, subscribers, and clients

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

        #Create a subscriber to subscribe to the TagInfo topic and subscribe to
messages about machine code information
        self.sub =
self.create_subscription(AprilTagInfo,'PosInfo',self.pos_callback,1)
        #Create a publisher to publish the topic of servo target angle and
publish messages to control the robotic arm servo
        self.pub_point = self.create_publisher(ArmJoint, 'TargetAngle',1)
        #Create a publisher to publish the topic of gripping results and publish
messages about gripping results
        self.pubGraspStatus = self.create_publisher(Bool, 'grasp_done',1)
        #Create a client requesting inverse solution service, used to calculate
the current end position and posture of the robot arm and the servo value of the
solution target
        self.client = self.create_client(Kinemarics, 'dofbot_kinemarics')
        #Initial gripper flag, True means grippable, False means non-grippable
        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.gripper_joint = 90
        #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]
        #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]
        #Rotation transformation matrix between the end of the robot arm and the
camera, describing 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+00,1.000000000e+00]])
        #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()
        #Define the current id value, and then place the machine code at the
corresponding position according to the value
        self.cur_tagId = 0
        #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):
    #print("msg: ",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)
        #获取当前的机械臂末端位置和位姿
        #Get the current end position and posture of the robot arm
        self.get_current_end_pos()
        #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)))
        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)
        #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)
        print("pose_T: ",pose_T)
        #判断夹取的标识，为True且高度大于0.05时，也就是高于0.05m时候，执行下爪夹取
        #Judge the gripping flag. When it is True and the height is greater than
0.05, that is, higher than 0.05m, the lower claw gripping is executed.
        if self.grasp_flag == True and (pose_T[2])>0.05:
            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,))
            grasp.start()
            grasp.join()
```

机械臂下爪夹取的函数grasp

```
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，0.01是x轴方向（左右）偏移量参数，由于舵机的微小差异性，无法保证计算出来的位置于实际的一样，根据实际情况进行微小的调整
    #The target x value at the end of the robot arm, in meters, 0.01 is the offset parameter in the x-axis direction (left and right). Due to the slight differences in the servos, it is impossible to guarantee that the calculated position is the same as the actual one. Make slight adjustments based on the actual situation.
    request.tar_x = pose_T[0]  - 0.01
    #机械臂末端的目标y值，单位是m，0.015是y轴方向（前后）偏移量参数，由于舵机的微小差异性，无法保证计算出来的位置于实际的一样，根据实际情况进行微小的调整
    #The target y value at the end of the robot arm, in meters. 0.015 is the offset parameter in the y-axis direction (front and back). Due to the slight differences in the servos, it is impossible to guarantee that the calculated position is the same as the actual one. Make slight adjustments based on the actual situation.
    request.tar_y = pose_T[1]  + 0.015
    #机械臂末端的目标z值，单位是m，0.02是z轴方向（高低）偏移量参数由于舵机的微小差异性，无法保证计算出来的位置于实际的一样，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, 0.02 is the offset parameter in the z-axis direction (high and low). Due to the slight differences in the servos, it is impossible to guarantee that the calculated position is the same as the actual one. Make slight adjustments based on the actual situation.
    request.tar_z = pose_T[2] + request.tar_y* 0.02
    #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, 135, 2000)
    time.sleep(2.5)
    self.pubArm([],2, 135, 2000)
    time.sleep(2.5)
    if self.cur_tagId == 1:
        self.down_joint = [130.0, 55.0, 34.0, 16.0, 90.0,135]
    elif self.cur_tagId == 2:
        self.down_joint = [170.0, 55.0, 34.0, 16.0, 90.0,135]
    elif self.cur_tagId == 3:
        self.down_joint = [50.0, 55.0, 34.0, 16.0, 90.0,135]
    elif self.cur_tagId == 4:
        self.down_joint = [10.0, 55.0, 34.0, 16.0, 90.0,135]
    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)
        self.pubArm(self.init_joints)
        time.sleep(5)
        self.grasp_flag = True
        grasp_done = Bool()
        grasp_done.data = True
        self.pubGraspStatus.publish(grasp_done)
```