Machine code ID 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

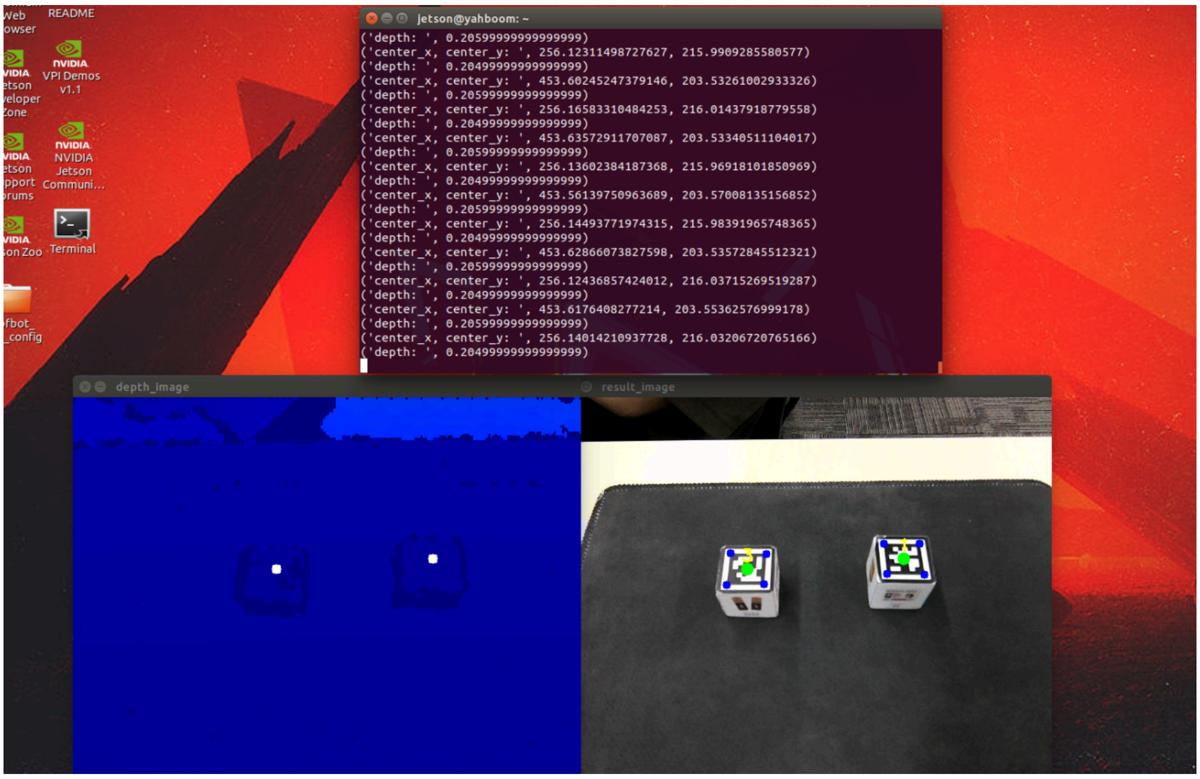
After the program is started, the camera recognizes the machine code, and the machine code will be framed in the screen and the corresponding ID value will be printed. Press the space bar to start the sorting program. The robot arm will grab the recognized machine code block and place the machine code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

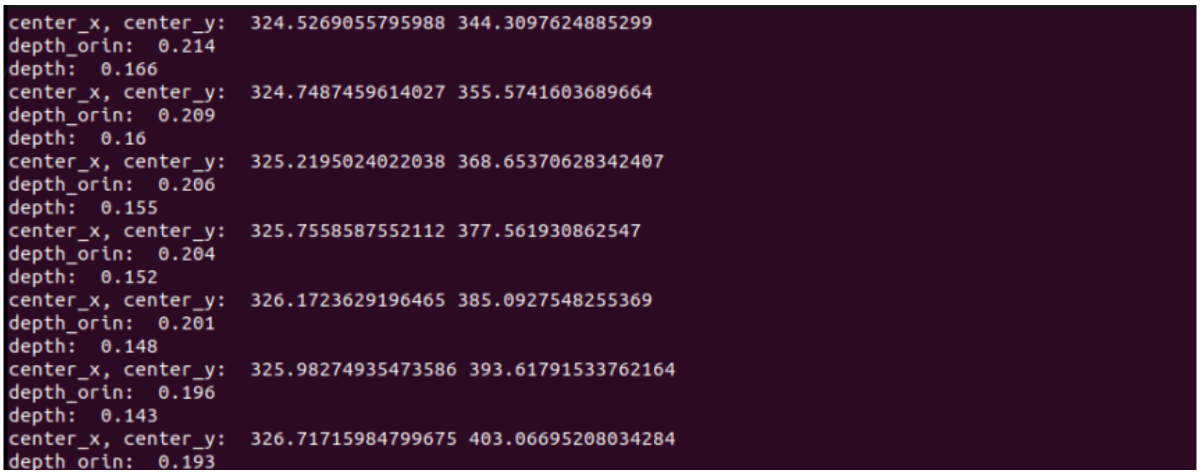
```
#Start the camera
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start the inverse program
ros2 run dofbot_pro_info kinemarics_dofbot
#Start the underlying control:
ros2 run dofbot_pro_driver arm_driver
#Machine code recognition
ros2 run dofbot_pro_driver apriltag_detect
#Grasp
ros2 run dofbot_pro_driver grasp
```



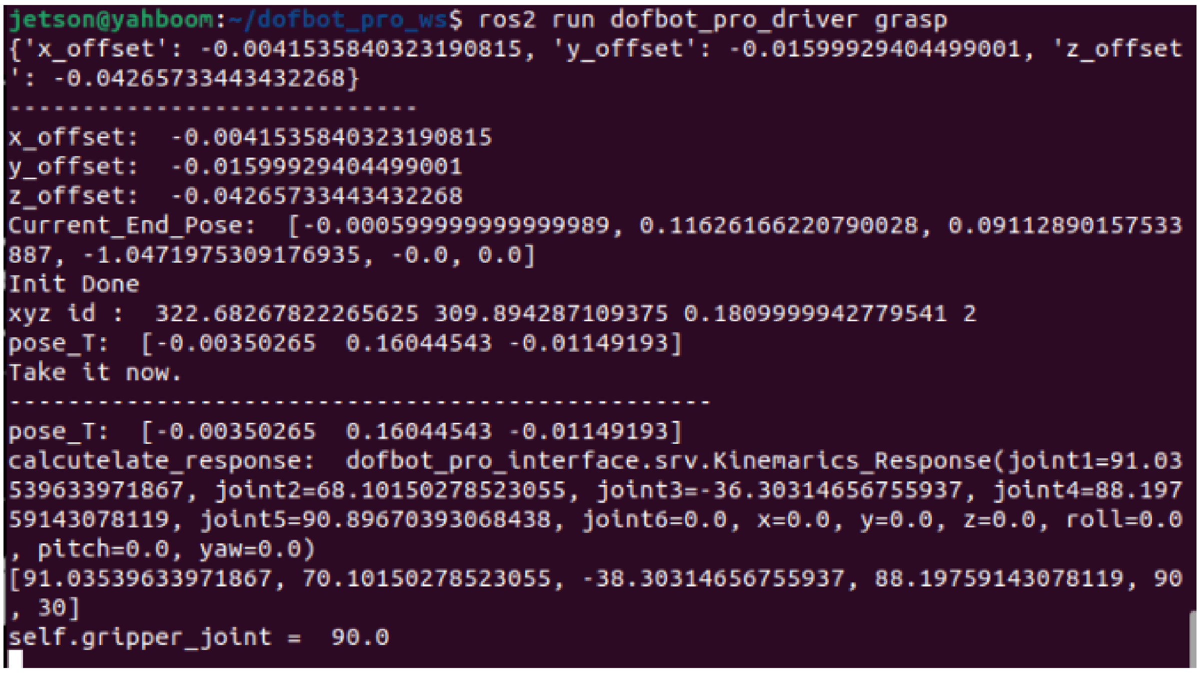
```
('depth: ', 0.20599999999999999)
('center_x, center_y: ', 256.12311498727627, 215.9909285580577)
('depth: ', 0.20499999999999999)
('center_x, center_y: ', 453.60245247379146, 203.53261002933326)
('depth: ', 0.20599999999999999)
('center_x, center_y: ', 256.16583310484253, 216.01437918779558)
('depth: ', 0.20499999999999999)
('center_x, center_y: ', 453.63572911707087, 203.53340511104017)
('depth: ', 0.20599999999999999)
('center_x, center_y: ', 256.13602384187368, 215.96918101850969)
('depth: ', 0.20499999999999999)
('center_x, center_y: ', 453.56139750963689, 203.57008135156852)
('depth: ', 0.20599999999999999)
('center_x, center_y: ', 256.14493771974315, 215.98391965748365)
('depth: ', 0.20499999999999999)
('center_x, center_y: ', 453.62866073827598, 203.53572845512321)
('depth: ', 0.20599999999999999)
('center_x, center_y: ', 256.12436857424012, 216.03715269519287)
('depth: ', 0.20499999999999999)
('center_x, center_y: ', 453.6176408277214, 203.55362576999178)
('depth: ', 0.20599999999999999)
('center_x, center_y: ', 256.14014210937728, 216.03206720765166)
('depth: ', 0.20499999999999999)
```

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 point 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.



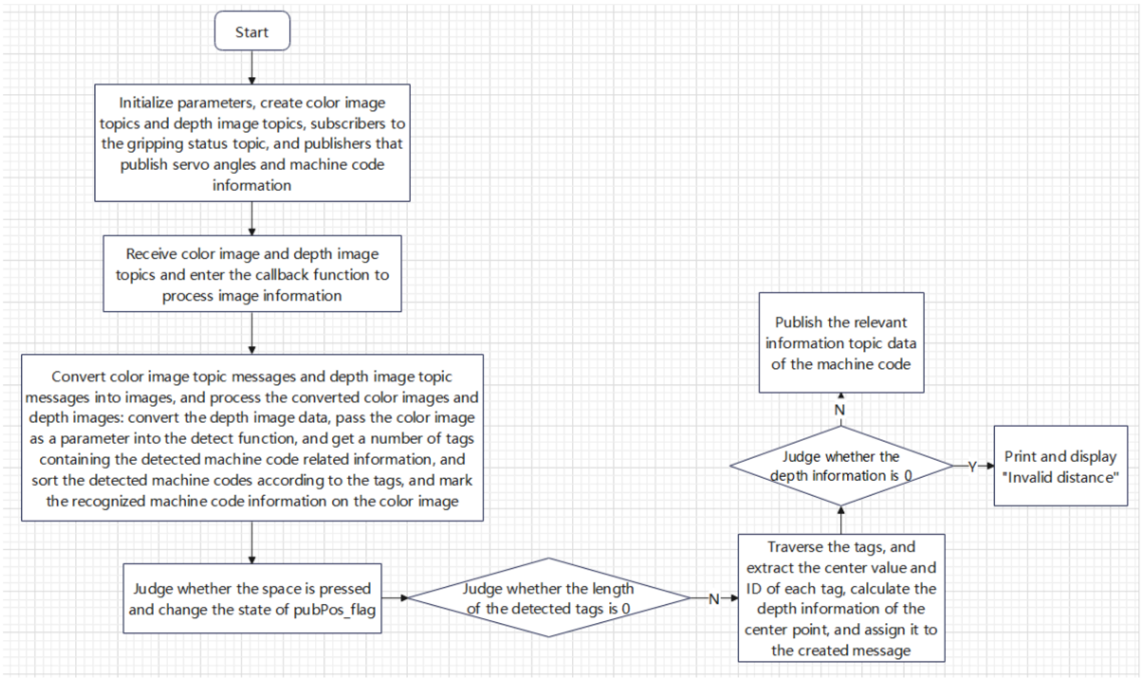
```
center_x, center_y:  324.5269055795988 344.3097624885299
depth_orin:  0.214
depth:  0.166
center_x, center_y:  324.7487459614027 355.5741603689664
depth_orin:  0.209
depth:  0.16
center_x, center_y:  325.2195024022038 368.65370628342407
depth_orin:  0.206
depth:  0.155
center_x, center_y:  325.7558587552112 377.561930862547
depth_orin:  0.204
depth:  0.152
center_x, center_y:  326.1723629196465 385.0927548255369
depth_orin:  0.201
depth:  0.148
center_x, center_y:  325.98274935473586 393.61791533762164
depth_orin:  0.196
depth:  0.143
center_x, center_y:  326.71715984799675 403.06695208034284
depth_orin:  0.193
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 run dofbot_pro_driver grasp
{'x_offset': -0.0041535840323190815, 'y_offset': -0.01599929404499001, 'z_offset
': -0.04265733443432268}
----------------------------
x_offset:  -0.0041535840323190815
y_offset:  -0.01599929404499001
z_offset:  -0.04265733443432268
Current_End_Pose:  [-0.0005999999999999989, 0.11626166220790028, 0.09112890157533
887, -1.0471975309176935, -0.0, 0.0]
Init Done
xyz id :  322.68267822265625 309.894287109375 0.1809999942779541 2
pose_T:  [-0.00350265  0.16044543 -0.01149193]
Take it now.
------------------------------------------------
pose_T:  [-0.00350265  0.16044543 -0.01149193]
calcutelate_response:  dofbot_pro_interface.srv.Kinemarics_Response(joint1=91.03
539633971867, joint2=68.10150278523055, joint3=-36.30314656755937, joint4=88.197
59143078119, joint5=90.89670393068438, joint6=0.0, x=0.0, y=0.0, z=0.0, roll=0.0
, pitch=0.0, yaw=0.0)
[91.03539633971867, 70.10150278523055, -38.30314656755937, 88.19759143078119, 90
, 30]
self.gripper_joint =  90.0
```

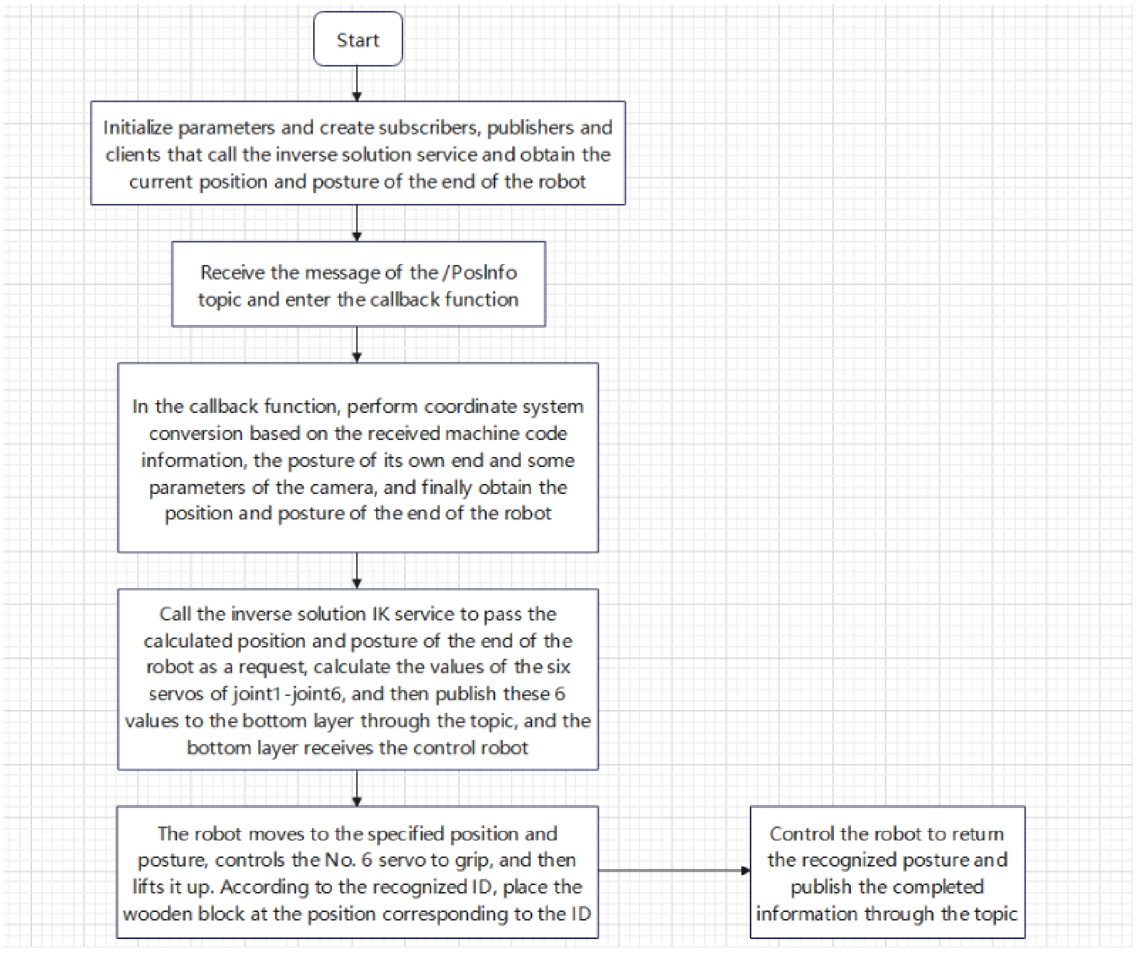
3. Program flow chart

apriltag_detect.py



Start
Initialize parameters, create color image topics and depth image topics, subscribers to the gripping status topic, and publishers that publish servo angles and machine code information
Receive color image and depth image topics and enter the callback function to process image information
Convert color image topic messages and depth image topic messages into images, and process the converted color images and depth images: convert the depth image data, pass the color image as a parameter into the detect function, and get a number of tags containing the detected machine code related information, and sort the detected machine codes according to the tags, and mark the recognized machine code information on the color image
Judge whether the space is pressed and change the state of pubPos_flag
Judge whether the length of the detected tags is 0
N
Traverse the tags, and extract the center value and ID of each tag, calculate the depth information of the center point, and assign it to the created message
Judge whether the depth information is 0
Y
Print and display "Invalid distance"
N
Publish the relevant information topic data of the machine code

grasp.py



Start
Initialize parameters and create subscribers, publishers and clients that call the inverse solution service and obtain the current position and posture of the end of the robot
Receive the message of the /PosInfo topic and enter the callback function
In the callback function, perform coordinate system conversion based on the received machine code information, the posture of its own end and some parameters of the camera, and finally obtain the position and posture of the end of the robot
Call the inverse solution IK service to pass the calculated position and posture of the end of the robot as a request, calculate the values of the six servos of joint1-joint6, and then publish these 6 values to the bottom layer through the topic, and the bottom layer receives the control robot
The robot moves to the specified position and posture, controls the No. 6 servo to grip, and then lifts it up. According to the recognized ID, place the wooden block at the position corresponding to the ID
Control the robot to return the recognized posture and publish the completed information through the topic

4. Core code analysis

4.1, apriltag_detect.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_driver/dofbot_pro_driver/apriltag_detec
t.py
```

Import necessary library files

```
import rclpy
from rclpy.node import Node
import cv2
import numpy as np
from sensor_msgs.msg import Image
from message_filters import ApproximateTimeSynchronizer, Subscriber
from std_msgs.msg import Float32, Bool
from cv_bridge import CvBridge
import cv2 as cv
from dt_apriltags import Detector
import threading
#Import drawing machine code library
from dofbot_pro_driver.vutils import draw_tags
#Import custom service data types
from dofbot_pro_interface.srv import Kinemarics
#Import custom message data types
from dofbot_pro_interface.msg import *
import pyzbar.pyzbar as pyzbar
from std_msgs.msg import Float32,Bool
import time
import queue
```

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]
        #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]
        #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 that publishes machine code information
        self.pos_info_pub = self.create_publisher(AprilTagInfo, "PosInfo",
qos_profile=10)
        #Create a publisher that publishes the target angle of the robot arm
        self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle",
qos_profile=1)
        #Time synchronization of color and depth image subscription messages
        self.TimeSynchronizer = 
message_filters.ApproximateTimeSynchronizer([self.rgb_image_sub,self.depth_image_
sub],1,0.5)
        #Create a subscriber that subscribes to the gripping results
        self.subscription = 
self.create_subscription(Bool,'grasp_done',self.GraspStatusCallback,qos_profile=1
)
        #Handle the callback function TagDetect for synchronization messages. The 
callback function is connected to the subscribed message so that it is 
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()
        #The flag for publishing machine code information. When it is True, it 
publishes /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 used, such as 'tag36h11'.
        nthreads: The number of threads for parallel processing to increase 
detection speed.
        quad_decimate: Reduce the resolution of the input image to reduce 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 to enhance 
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)
```

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_driver/dofbot_pro_driver/grasp.py
```

Import necessary library files

```
import rclpy
from rclpy.node import Node
import numpy as np
from std_msgs.msg import Float32, Bool
import time
import math
from dofbot_pro_interface.msg import *
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 # 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 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'))
```

Initialize program parameters, create publishers, subscribers and clients

```
def __init__(self):
    super().__init__('color_grap')
    #Create a subscriber to subscribe to the TagInfo topic, subscribe to the
machine code information message
    self.sub =
self.create_subscription(AprilTagInfo,'PosInfo',self.pos_callback,1)
    #Create a publisher to publish the topic of the servo target angle, and
publish the message of controlling the robot arm servo
    self.pub_point = self.create_publisher(ArmJoint, 'TargetAngle',1)
    #Create a publisher to publish the topic of the gripping result, and publish 
the message of the gripping result
    self.pubGraspStatus = self.create_publisher(Bool, 'grasp_done',1)
    #Create a client requesting the inverse solution service, which is used to 
calculate the current robot arm end position and posture and the servo value of 
the solution target
    self.client = self.create_client(Kinemarics, 'dofbot_kinemarics')
    #Initial gripping flag, True means grippable, False means non-gripable
    self.grasp_flag = True
    #Arm initialization 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
    #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+00,1.00000000e+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
    #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')
    #Print the current position and posture of the end of the robot arm
    print("Current_End_Pose: ",self.CurrentEndPos)
    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 pubTargetArm(self, joints, id=6, angle=180.0, runtime=2000):
        print(joints)
 self.Arm.Arm_serial_servo_write6(joints[0],joints[1],joints[2],joints[3],joints[
4],joints[5],2000)
```

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)

    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)
    self.pubArm(self.init_joints)
    time.sleep(5)
    self.grasp_flag = True
    grasp_done = Bool()
    grasp_done.data = True
    self.pubGraspStatus.publish(grasp_done)
```