Mediapipe gesture recognition machine code distance 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
```

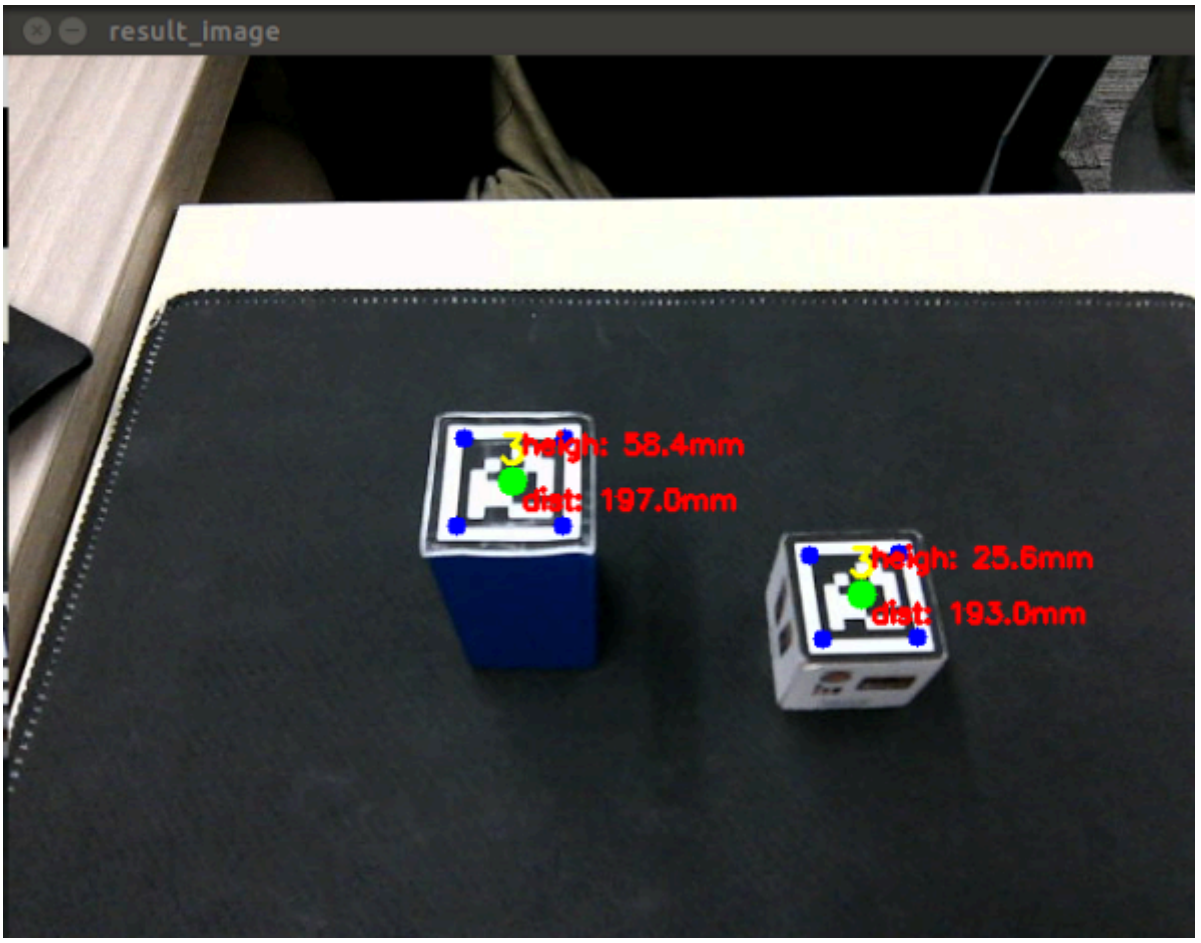
1. Functional description

After the program is started, the camera captures the image and recognizes the gesture. The gesture is 1 to 5. The distance threshold is calculated through the recognized gesture; the robot arm will change its posture to detect the machine code in the image and calculate their height and distance. If there is less than the distance threshold, the robot arm will clamp it and place it at the set position, and then return to the posture of detecting the machine code to continue recognition; if the machine code less than the distance threshold is not detected, the robot arm will perform the "shake head" action group and the buzzer will sound, and then the robot arm will return to the posture of recognizing gestures.

2. Start and operate

2.1. Start command

```
#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_apriltag msgToimg
#Start the machine code recognition program:
ros2 run dofbot_pro_apriltag apriltag_list_Dist
#Start the robot arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start the Mediapipe gesture recognition program:
ros2 run dofbot_pro_apriltag MediapipeGesture
```



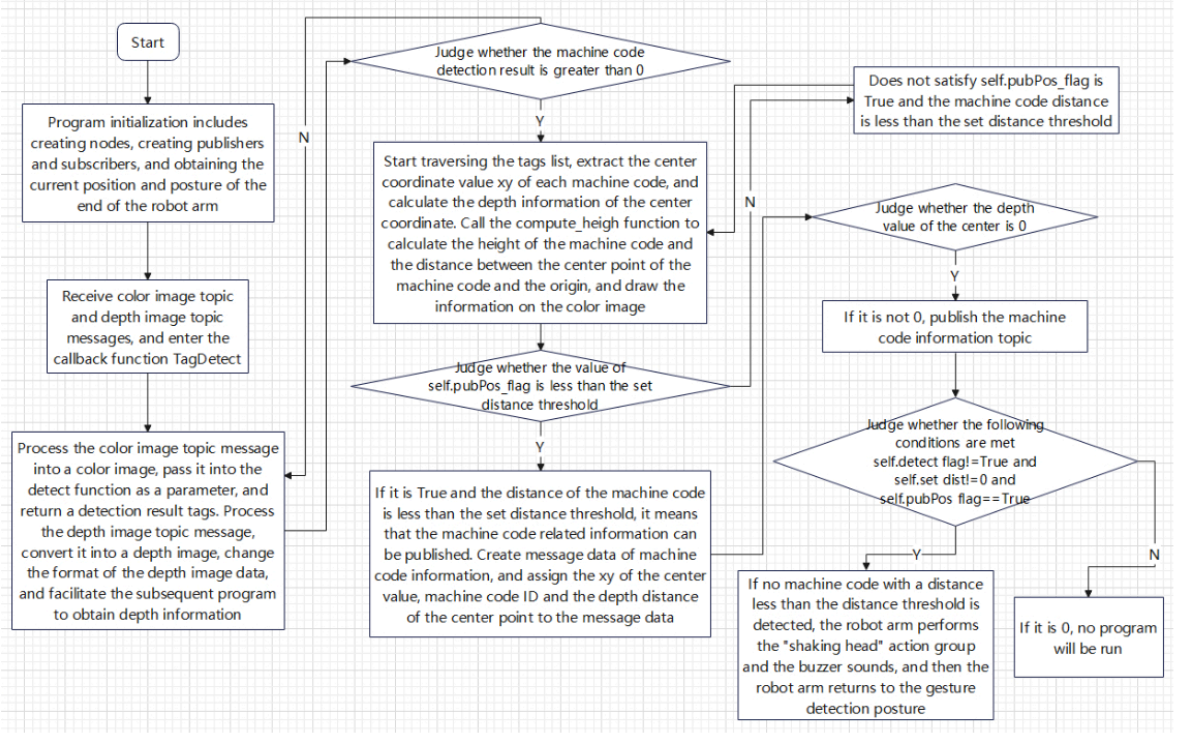
2.2、Operation

After the program is started, the robot arm will begin to show the gesture recognition posture. The recognizable gestures are one to five. The gesture recognition will wait for about 3 seconds, waiting for the machine code to change its posture and become the posture of detecting and recognizing the machine code. Press the space bar to start recognition; if the distance of the recognized machine code is less than the calculated distance threshold, the robot arm will clamp the machine code block with its lower claws and place it in the set position; after placement, the robot arm returns to the posture of recognizing the machine code, and the space bar needs to be pressed again for the next recognition. If the machine code less than the set distance threshold is not recognized, the robot arm will make a "shaking head" action group and the buzzer will sound, and then the robot arm returns to the gesture recognition posture.

Distance threshold calculation: 170+gesture recognition result*10

3. Program flow chart

apriltag_list_Dist.py



4. Core code analysis

4.1. MediapipeGesture.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/dofbot_pro_apriltag/MediapipeGesture.py
```

You can refer to the 4.1 part [MediapipeGesture.py] in the tutorial [3D space sorting and clamping\3. Mediapipe gesture recognition machine code ID sorting].

4.2、apriltag_list_Dist.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/dofbot_pro_apriltag/apriltag_list_Dist.py
```

Import the necessary libraries,

```
import cv2
import rclpy
from rclpy.node import Node
import numpy as np
from message_filters import ApproximateTimeSynchronizer, Subscriber
from sensor_msgs.msg import Image
from std_msgs.msg import Float32, Int8, Bool
from dt_apriltags import Detector
from dofbot_pro_apriltag.vutils import draw_tags
from cv_bridge import CvBridge
import cv2 as cv
from dofbot_pro_interface.srv import *
from dofbot_pro_interface.msg import ArmJoint, AprilTagInfo
```

```
import pyzbar.pyzbar as pyzbar
import time
import queue
import math
import os
encoding = ['16UC1', '32FC1']
import threading
#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
```

Initialize program parameters, create publishers and subscribers,

```
def __init__(self):
        rospy.init_node('apriltag_detect')
        super().__init__('apriltag_detect')

        #Initialization parameters
        self.detect_joints = [90.0, 150.0, 12.0, 20.0, 90.0, 30.0]
        self.init_joints = [90.0, 120.0, 0.0, 0.0, 90.0, 90.0]
        self.search_joints = [90.0, 120.0, 0.0, 0.0, 90.0, 30.0]
        self.Arm = Arm_Device()

        #ROS2 Posted by
        self.pubGraspStatus = self.create_publisher(Bool, "grasp_done", 1)
        self.tag_info_pub = self.create_publisher(AprilTagInfo, "PosInfo", 1)
        self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 1)

        # ROS2 subscribers (message synchronization)
        self.depth_image_sub = Subscriber(self, Image,
"/camera/color/image_raw", qos_profile=1)
        self.rgb_image_sub = Subscriber(self, Image, "/camera/depth/image_raw",
qos_profile=1)
        self.TimeSynchronizer =
ApproximateTimeSynchronizer([self.depth_image_sub,
self.rgb_image_sub],queue_size=10,slop=0.5)
        self.TimeSynchronizer.registerCallback(self.TagDetect)

        #ROS2 Other Subscribers
        self.grasp_status_sub = self.create_subscription(Bool, 'grasp_done',
self.GraspStatusCallback, 1)
        self.sub_targetID = self.create_subscription(Int8, "TargetId",
self.GetTargetIDCallback, 1)

        # Initialization tool
        self.rgb_bridge = CvBridge()
```

```
        self.depth_bridge = CvBridge()
        self.pubPos_flag = False
        self.done_flag = True

        self.set_height = 40.0
        self.set_dist = 0.0
        self.detect_flag = False

        # AprilTag Detector (configuration remains unchanged)
        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
        )
        self.target_id = 0
        self.pr_time = time.time()
        self.cnt = 0
        self.Center_x_list = []
        self.Center_y_list = []
        self.search_joints = [90.0,150.0,12.0,20.0,90.0,30.0]

        self.CurEndPos =
[-0.006,0.116261662208,0.0911289015753,-1.04719,-0.0,0.0]
        self.camera_info_K = [477.57421875, 0.0, 319.3820495605469, 0.0,
477.55718994140625, 238.64108276367188, 0.0, 0.0, 1.0]
        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]])
```

Mainly look at the image processing function TagDetect,

```
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 depth image topic message and convert the message data into
image data
    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)
    #调用detect函数，传入参数，
    #Call the detect function and pass in 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 posture 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.
    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) # 貌似出来就是升序排列的不需要手动
进行排列
    #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))
    key = cv2.waitKey(10)
    #定义self.Center_x_list和self.Center_y_list的长度
    #Define the length of self.Center_x_list and self.Center_y_list
    self.Center_x_list = list(range(len(tags)))
    self.Center_y_list = list(range(len(tags)))
    #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
    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以及夹取机器码完成的标识
    #Judge the length of tags. If it is greater than 0, it means that the machine
code has been detected and the machine code has been captured.
    if len(tags) > 0 and self.done_flag == True:
        #遍历机器码
        #Traverse the machine code
        for i in range(len(tags)):
            #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
            center_x, center_y = tags[i].center
            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_heigh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_heigh 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)
            heigh_detect = round(pose[2],4)*1000 - 12
            heigh = 'heigh: ' + str(heigh_detect) + 'mm'
            #计算机器码离基坐标系的距离值，这里采用了欧几里得距离计算公式计算xy与原点（0，0）
的距离，对该值进行放大运算，把单位换算成毫米，pose[1]表示中心点在世界坐标系下的y值，pose[0]表示
中心点在世界坐标系下的x值
            #Calculate the distance between the machine code and the base
coordinate system. Here, the Euclidean distance calculation formula is used to
calculate the distance between xy and the origin (0, 0). The value is enlarged
and the unit is converted to millimeters. Pose[1] represents the y value of the
center point in the world coordinate system, and pose[0] represents the x value
of the center point in the world coordinate system.
            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)
            #如果检测到的机器码距离基坐标系的距离小于设定的距离阈值且self.pubPos_flag 的值
为True且设定的距离阈值不为0
            #If the distance between the detected machine code and the base
coordinate system is less than the set distance threshold and the value of
self.pubPos_flag is True and the set distance threshold is not 0
            if dist_detect<=self.set_dist and self.set_dist!=0 and
self.pubPos_flag == True:
                print("self.set_dist: ",self.set_dist)
                print("dist_detect: ",dist_detect)
                #改变self.detect_flag的值，为True则表示识别到了有高于设定阈值的机器码
                #Change the value of self.detect_flag. If it is True, it means
that a machine code higher than the set threshold has been identified.
                self.detect_flag  = True
                tag = AprilTagInfo()
                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
                #如果深度信息不为0，说明为有效数据，然后发布机器码信息的消息
                #If the depth information is not 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)
                    self.pubPos_flag = False
                    self.done_flag = False
                else:
                    print("Invalid distance.")
        #如果self.detect_flag为False表示没有识别到小于距离阈值的机器码且设定的距离阈值不为0
以及使能发布机器码消息，符合三者条件说明没有识别到小于距离阈值的机器码。
```

```
        #If self.detect_flag is False, it means that no machine code less than
the distance threshold is recognized, and the set distance threshold is not 0,
and the release of machine code messages is enabled. If all three conditions are
met, it means that no machine code less than the distance threshold is
recognized.
        if self.detect_flag != True and  self.set_dist!=0 and
self.pubPos_flag==True:
            print("----------------------------------------")
            self.set_dist!=0
            #机械臂做出“摇头”的动作组且蜂鸣器响
            #The robot arm performs the "shaking head" action group and the
buzzer sounds
            self.shake()
            #time.sleep(2)
            self.pub_arm(self.search_joints)
            grasp_done = Bool()
            grasp_done.data = True
            #发布夹取完成的话题，以便于下次手势识别节点程序发布手势识别的结果
            #Publish the topic of completed clamping so that the gesture
recognition node program can publish the results of gesture recognition next time
            self.pubGraspStatus.publish(grasp_done)
            #time.sleep(2)
        self.pubPos_flag = False
        #如果按下空格后没有识别到任何机器码，则同样机械臂做出“摇头”的动作组，蜂鸣器响，然后回到
识别手势的姿态
        #If no machine code is recognized after pressing the space bar, the robot
arm will make the "shake head" action group, the buzzer will sound, and then
return to the gesture recognition posture
    elif self.pubPos_flag == True and len(tags) == 0:
        self.shake()
        self.pub_arm(self.search_joints)
        grasp_done = Bool()
        grasp_done.data = True
        self.pubGraspStatus.publish(grasp_done)
    result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
    cv2.imshow("result_image", result_image)
    key = cv2.waitKey(1)
```

The callback function GetTargetIDCallback of the gesture recognition result,

```
def GetTargetIDCallback(self,msg):
    print("msg.data: ",msg.data)
    #计算距离阈值单位是毫米mm，最小是160mm，最大是200mm
    #The distance threshold unit is mm, the minimum is 160mm and the maximum is
200mm
    self.set_dist = 150 + msg.data*10
    #计算高度阈值单位是毫米mm，最小是40mm，最大是80mm
    #The unit of height threshold calculation is millimeter, the minimum is 40mm
and the maximum is 80mm
    self.set_height = 30 + msg.data*10
    print("self.set_height: ",self.set_height)
    #接收到消息后，改变机械臂的姿态，呈现识别机器码的姿态
    #After receiving the message, change the posture of the robot arm to present
the posture of recognizing the machine code
    self.pub_arm(self.init_joints)
```

4.3、grasp.py

Please refer to the content of [grasp.py] in section 4.2 of the tutorial [Three-dimensional space sorting and gripping\1. Machine code ID sorting].