Mediapipe Gesture-AprilTag Height Sorting

Before starting this function, you need to close the large program and APP processes. If you need to restart the large program and APP later, start them from the terminal:

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

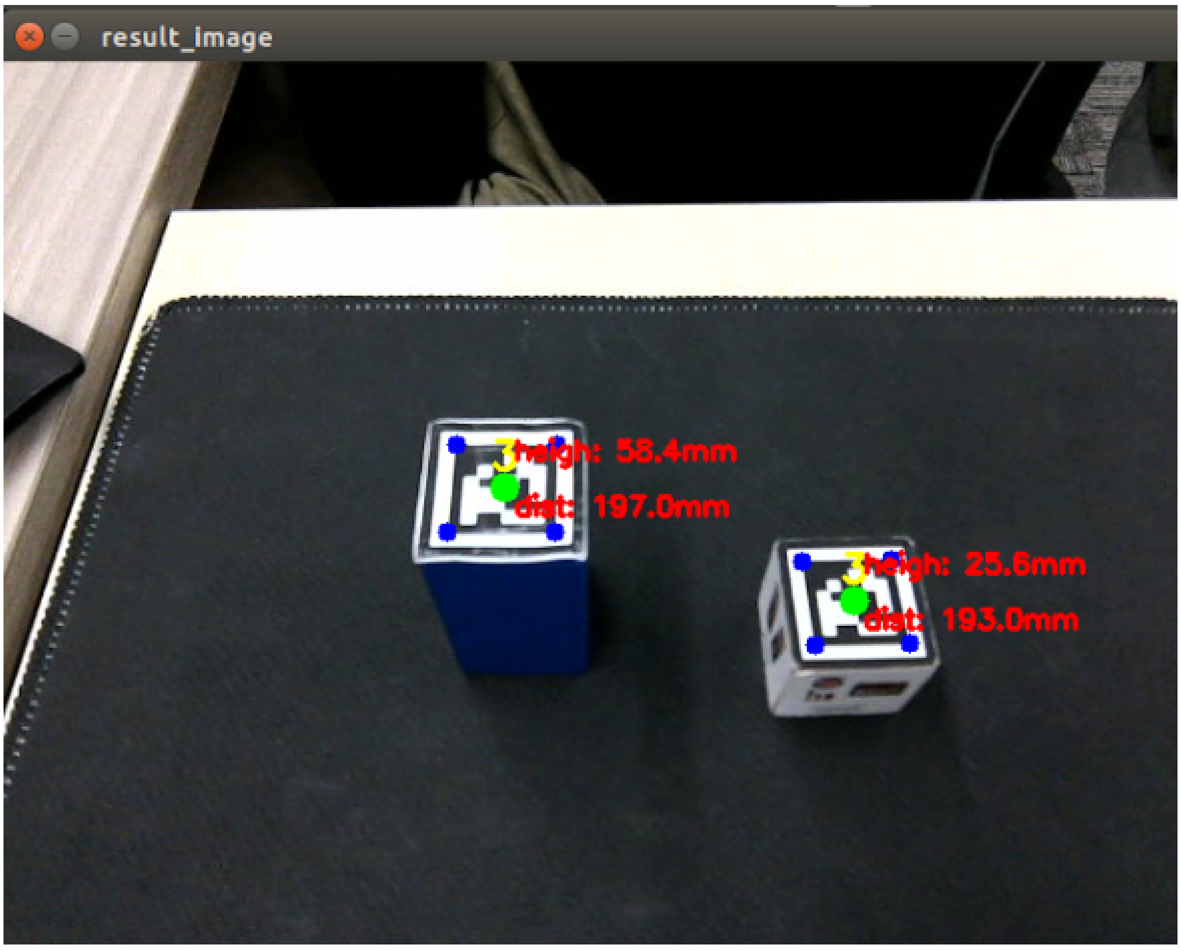
1. Function Description

After the program starts, the camera captures images and recognizes gestures. Gestures range from 1 to 5. Through the recognized gesture, the height threshold is calculated; the robotic arm will change its posture to detect AprilTags in the image and calculate their height. If any exceed the height threshold, the robotic arm will lower the gripper to grasp and place them at the set position, then return to the AprilTag detection posture to continue recognition; if no AprilTag exceeding the height threshold is detected, the robotic arm will perform a "head shaking" action group , then the robotic arm returns to the gesture recognition posture.

2. Startup and Operation

2.1. Startup Commands

```
#Start camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start underlying control:
ros2 run dofbot_pro_driver arm_driver
#Start inverse kinematics program:
ros2 run dofbot_pro_info kinemarics_dofbot
#Start image conversion program:
ros2 run dofbot_pro_apriltag msgToimg
#Start AprilTag recognition program:
ros2 run dofbot_pro_apriltag apriltag_list_Hight
#Start robotic arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start Mediapipe gesture recognition program:
ros2 run dofbot_pro_apriltag MediapipeGesture
```



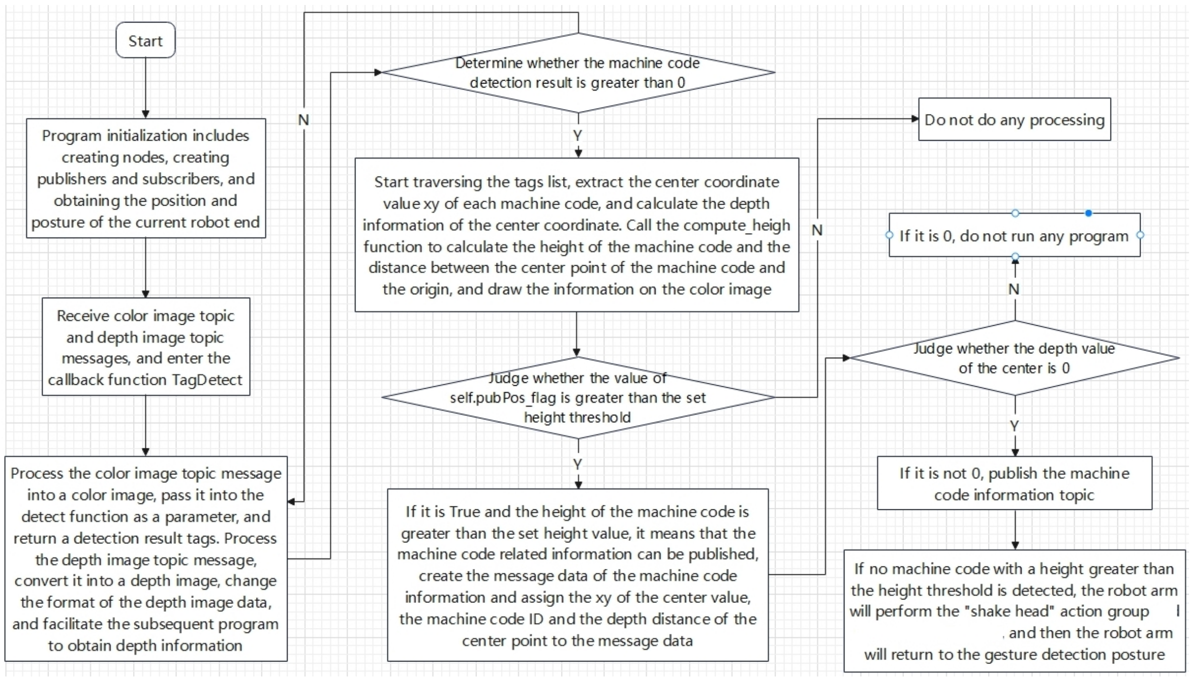
2.2. Operation

After the program starts, the robotic arm will initially present a gesture recognition posture. The recognizable gestures range from one to five. Gesture recognition waits for about 3 seconds, waiting for the AprilTag to change posture to the AprilTag detection and recognition posture. Press the spacebar to start recognition; if a AprilTag with height higher than the calculated threshold is recognized, the robotic arm will lower the gripper to grasp that AprilTag block and place it at the set position; after placement is complete, the robotic arm returns to the AprilTag recognition posture. The next recognition requires pressing the spacebar again. If no AprilTag higher than the set height threshold is recognized, the robotic arm will perform a "head shaking" action group, then the robotic arm returns to the gesture recognition posture.

Height threshold calculation: 30 + gesture recognition result * 10

3. Program Flowchart

apriltag_list_Hight.py


Start
Program initialization includes creating nodes, creating publishers and subscribers, and obtaining the position and posture of the current robot end
Receive color image topic and depth image topic messages, and enter the callback function TagDetect
Process the color image topic message into a color image, pass it into the detect function as a parameter, and return a detection result tags. Process the depth image topic message, convert it into a depth image, change the format of the depth image data, and facilitate the subsequent program to obtain depth information
N
Determine whether the machine code detection result is greater than 0
Y
Start traversing the tags list, extract the center coordinate value xy of each machine code, and calculate the depth information of the center coordinate. Call the compute_heigh function to calculate the height of the machine code and the distance between the center point of the machine code and the origin, and draw the information on the color image
Judge whether the value of self.pubPos_flag is greater than the set height threshold
Y
If it is True and the height of the machine code is greater than the set height value, it means that the machine code related information can be published, create the message data of the machine code information and assign the xy of the center value, the machine code ID and the depth distance of the center point to the message data
N
Do not do any processing
If it is 0, do not run any program
N
Judge whether the depth value of the center is 0
Y
If it is not 0, publish the machine code information topic
If no machine code with a height greater than the height threshold is detected, the robot arm will perform the "shake head" action group and then the robot arm will return to the gesture detection posture

4. Core Code Analysis

4.1. MediapipeGesture.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/dofbot_pro_apriltag/Mediapipe
Gesture.py
```

You can refer to section 4.1 [MediapipeGesture.py] in tutorial [3D Space Sorting and Grasping\3. Mediapipe Gesture-AprilTag ID Sorting].

4.2. apriltag_list_Hight.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/dofbot_pro_apriltag/apriltag_
list_Hight.py
```

Import necessary library functions

```python
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
from Arm_Lib import Arm_Device
#Import transforms3d library for handling transformations in 3D space, performing
conversions between quaternions, rotation matrices and Euler angles, supporting
3D geometric operations and coordinate transformations
import transforms3d as tfs
#Import transformations for handling and calculating transformations in 3D space,
including conversions between quaternions and Euler angles
import tf_transformations as tf
```

Program parameter initialization, create publishers and subscribers

```
def __init__(self):
    super().__init__('apriltag_detect')
    #Robotic arm AprilTag recognition posture
    self.init_joints = [90.0, 120, 0, 0.0, 90, 90]
    #Create two subscribers, subscribe to color image topic and depth image
topic
    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)
    #Create subscriber for publishing grasping results
    self.pubGraspStatus = self.create_publisher(Bool, "grasp_done", 1)
    #Create publisher for buzzer topic
    self.pub_buzzer = rospy.Publisher("Buzzer", Bool, queue_size=1)
    #Create publisher for AprilTag information
    self.tag_info_pub = self.create_publisher(AprilTagInfo, "PosInfo", 1)
    #Create publisher for robotic arm target angle
    self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 1)
    #Create subscriber for gesture recognition results
    self.sub_targetID = self.create_subscription(Int8, "TargetId",
self.GetTargetIDCallback, 1)
    #Time-synchronize color and depth image subscription messages
    self.TimeSynchronizer = ApproximateTimeSynchronizer([self.depth_image_sub,
self.rgb_image_sub],queue_size=10,slop=0.5))
    #Create subscriber for grasping results
    self.grasp_status_sub = self.create_subscription(Bool, 'grasp_done',
self.GraspStatusCallback, 1)
    #Callback function TagDetect for handling synchronized messages, connecting
callback function with subscribed messages to automatically call this function
when receiving new messages
    self.TimeSynchronizer.registerCallback(self.TagDetect)
    #Create bridges for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
    #Flag for publishing AprilTag information, when True publish /TagInfo topic
data
    self.pubPos_flag = False
    self.done_flag = True
    #Initialize set height threshold to 0.0
    self.set_height = 0.0
    #Initialize set distance threshold to 0.0
    self.set_dist = 0.0
    self.detect_flag = False
    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 = 31
    self.cnt = 0
    self.Center_x_list = []
    self.Center_y_list = []
    #Robotic arm gesture recognition posture
    self.search_joints = [90,150,12,20,90,30]
    #Current robotic arm end position and pose
    self.CurEndPos = [-0.006,0.116261662208,0.0911289015753,-1.04719,-0.0,0.0]
    #Camera built-in parameters
    self.camera_info_K = [477.57421875, 0.0, 319.3820495605469, 0.0,
477.55718994140625, 238.64108276367188, 0.0, 0.0, 1.0]
    #Rotation transformation matrix between robotic arm end and camera,
describing the relative position and pose between them
    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]])
    exit_code = os.system('rosservice call /camera/set_color_exposure  50')
```

Main image processing function TagDetect

```
def TagDetect(self,color_frame,depth_frame):
    #rgb_image
    #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
    #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)
    #Call the detect function and pass in parameters,
    '''
    cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY): Convert the RGB image to a
grayscale image for tag detection.
    False: Indicates that the pose of the tag is not estimated.
    None: Indicates that no camera parameters are provided, and only simple
detection may be performed.
```

```
    0.025: May be the set tag size (the unit is usually meters) to help the
detection algorithm determine the size of the tag
    Returns a detection result, including the position, ID, and bounding box of
each tag.
    '''
    tags = self.at_detector.detect(cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY),
False, None, 0.025)
    #Sort the tags in tags, not a necessary step
    tags = sorted(tags, key=lambda tag: tag.tag_id) # It seems that the output
is already sorted in ascending order, so manual sorting is not required
    #Call the draw_tags function to draw the information related to the
recognized machine code on the color image, including corner points, center point
and id value
    draw_tags(result_image, tags, corners_color=(0, 0, 255), center_color=(0,
255, 0))
    key = cv2.waitKey(10)
    #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)))
    #Wait for keyboard input, 32 means space is pressed, after pressing, change
the value of self.pubPos_flag, which means that machine code related information
can be published
    if key == 32:
        self.pubPos_flag = True
    #Judge the length of tags, if it is greater than 0, it means that the machine
code has been detected and the identification of the completion of grasping the
machine code
    if len(tags) > 0 and self.done_flag == True:
        #遍历机器码
        for i in range(len(tags)):
            #The center xy value of the machine code exists in the Center_x_list
and Center_y_list lists
            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
            #Calculate the depth value of the center coordinate
            cz = depth_image_info[int(cy),int(cx)]/1000
            #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 return is a position list,
pose[2] represents the z value, which is the height value
            pose = self.compute_heigh(cx,cy,cz)
            #Magnify the height value and convert the unit to millimeters
            heigh_detect = round(pose[2],4)*1000
            heigh = 'heigh: ' + str(heigh_detect) + 'mm'
            #Calculate the distance value of the machine code from the base
coordinate system, magnify the value, and convert the unit to millimeters
            dist_detect = math.sqrt(pose[1] ** 2 + pose[0]** 2)
            dist_detect = round(dist_detect,3)*1000
            dist = 'dist: ' + str(dist) + 'mm'
            #The 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)
```

```
            #If the detected machine code height is greater than the set height threshold and the value of self.pubPos_flag is True and the set height threshold is not 0
            if heigh_detect>=self.set_height and self.set_height!=0 and self.pubPos_flag == True:
                print("self.set_height: ",self.set_height)
                print("heigh_detect: ",heigh_detect)
                #Change the value of self.detect_flag to True, which means that a machine code higher than the set threshold has been identified
                self.detect_flag  = True
                #Assign a value 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, and z is the depth value of the center point. Here, it is reduced by a factor of 1000, and the unit is meters.
                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
                #If the depth information is not 0, it means that the data is valid, and then the message of the machine code information is published
                if tag.z!=0 :
                    self.tag_info_pub.publish(tag)
                    self.pubPos_flag = False
                    self.done_flag = False
                else:
                    print("Invalid distance.")
        #If self.detect_flag is False, it means that no machine code higher than the height threshold is recognized, and the set height threshold is not 0, and the machine code message is enabled to be published. If the three conditions are met, it means that no machine code higher than the height threshold is recognized.
        if self.detect_flag != True and  self.set_height!=0 and self.pubPos_flag==True:
            print("---------------------------------------")
            self.set_height!=0
            #The robotic arm makes a "head shaking" action group
            self.shake()
            #time.sleep(2)
            #Answer the posture of recognizing the gesture, and prepare to recognize the next gesture
            self.pub_arm(self.search_joints)
            #Publish the topic of grasping completion, so that the next gesture recognition node program can publish the result of gesture recognition
            grasp_done = Bool()
            grasp_done.data = True
            self.pubGraspStatus.publish(grasp_done)
        self.pubPos_flag = False
#If no machine code is recognized after pressing the space bar, the robotic arm will also make a "head shaking" action group, 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)
```

Gesture recognition result callback function GetTargetIDCallback

```
def GetTargetIDCallback(self,msg):
    print("msg.data: ",msg.data)
    #Calculate distance threshold unit is millimeters mm, minimum is 160mm,
maximum is 20mm
    self.set_dist = 150 + msg.data*10
    #Calculate height threshold unit is millimeters mm, minimum is 40mm, maximum
is 80mm
    self.set_height = 30 + msg.data*10
    print("self.set_height: ",self.set_height)
    #After receiving message, change robotic arm posture to present AprilTag
recognition posture
    self.pub_arm(self.init_joints)
```

4.3. grasp.py

You can refer to section 4.2 [grasp.py] in tutorial [3D Space Sorting and Grasping\1. AprilTag ID Sorting].