Mediapipe Gesture-AprilTag Distance 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
```

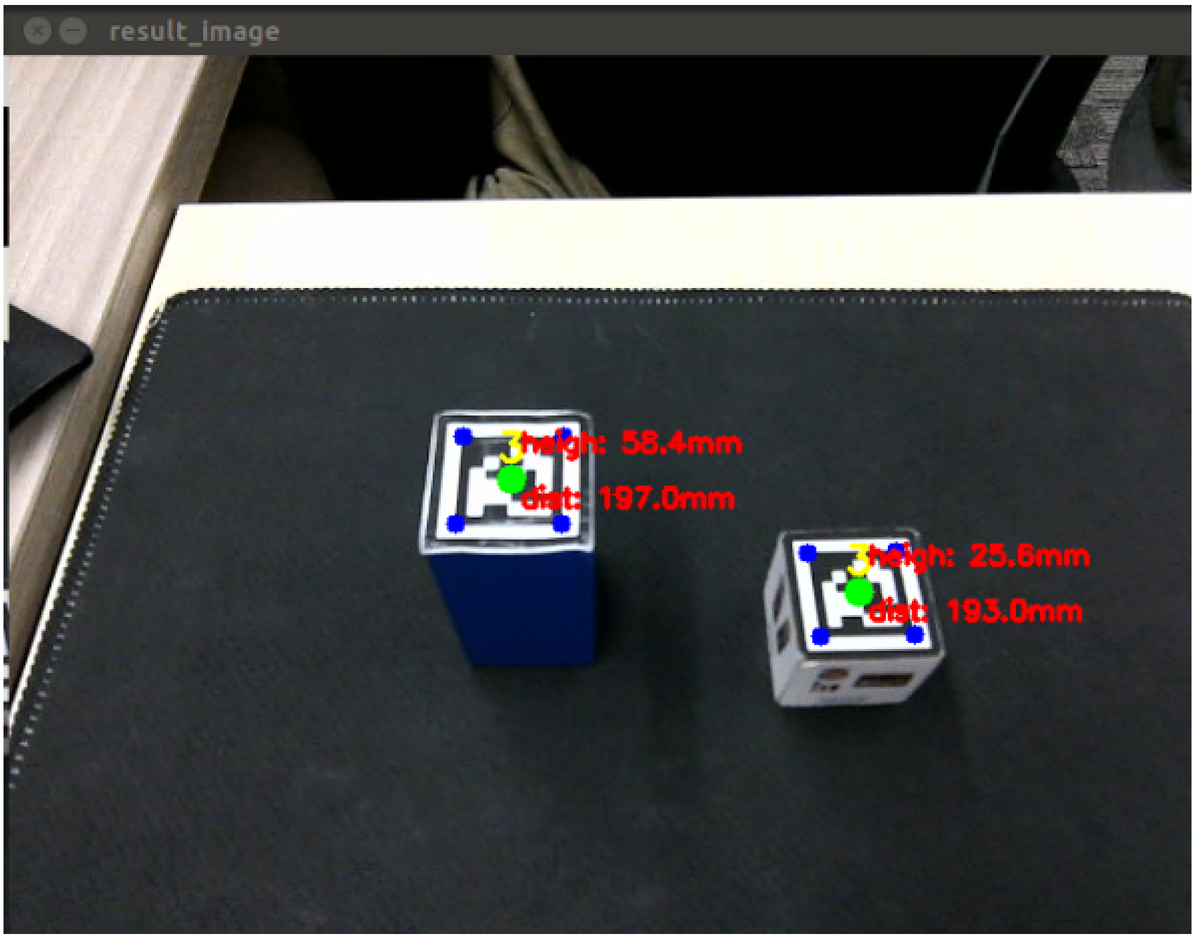
1. Function Description

After the program starts, the camera captures images and recognizes gestures. Gestures range from 1 to 5. Through the recognized gesture, the distance threshold is calculated; the robotic arm will change its posture to detect AprilTags in the image and calculate their height and distance. If any are smaller than the distance 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 smaller than the distance threshold is detected, the robotic arm will perform a "head shaking" action group and the buzzer will sound, 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_Dist
#Start robotic arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start Mediapipe gesture recognition program:
ros2 run dofbot_pro_apriltag MediapipeGesture
```



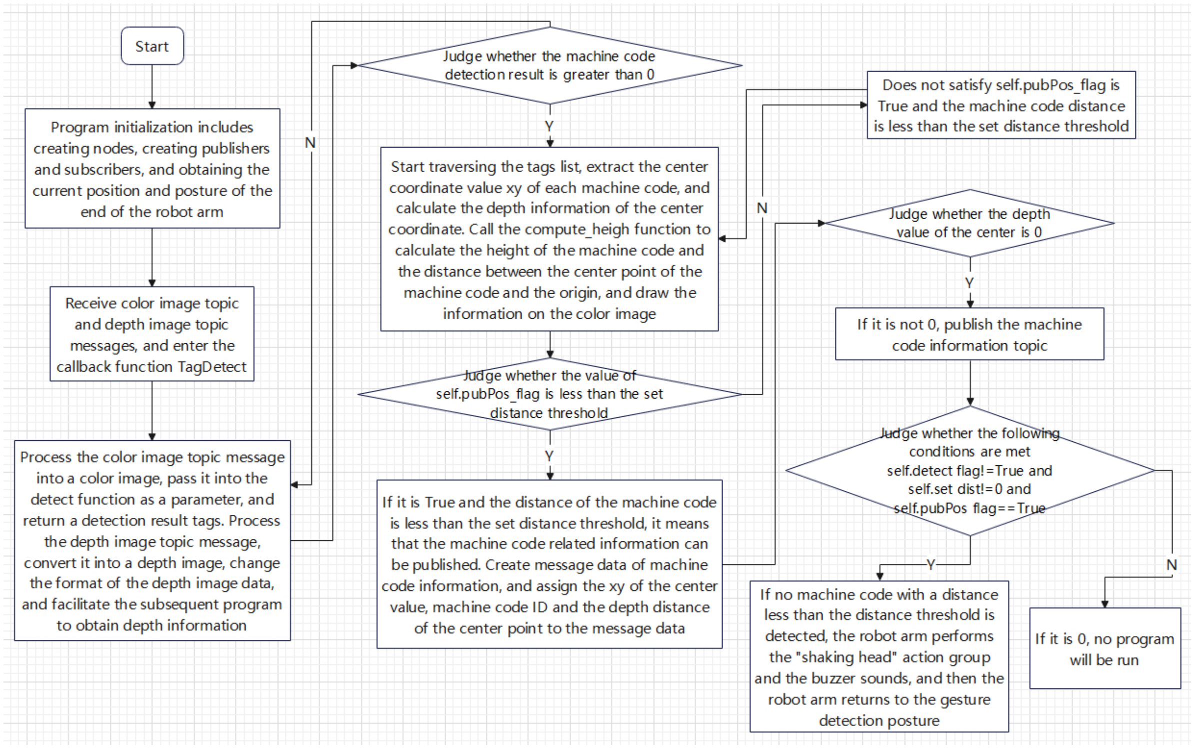
2.2. Operation

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 distance smaller than the calculated distance 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 smaller than the set distance threshold is recognized, the robotic arm will perform a "head shaking" action group and the buzzer will sound, then the robotic arm returns to the gesture recognition posture.

Distance threshold calculation: 170 + gesture recognition result * 10

3. Program Flowchart

apriltag_list_Dist.py


Start
Program initialization includes creating nodes, creating publishers and subscribers, and obtaining the current position and posture of the end of the robot arm
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
Judge 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 less than the set distance threshold
Y
If it is True and the distance of the machine code is less than the set distance threshold, it means that the machine code related information can be published. Create message data of machine code information, and assign the xy of the center value, machine code ID and the depth distance of the center point to the message data
Does not satisfy self.pubPos_flag is True and the machine code distance is less than the set distance threshold
N
Judge whether the depth value of the center is 0
Y
If it is not 0, publish the machine code information topic
Judge whether the following conditions are met self.detect flag!=True and self.set dist!=0 and self.pubPos flag==True
Y
N
If no machine code with a distance less than the distance threshold is detected, the robot arm performs the "shaking head" action group and the buzzer sounds, and then the robot arm returns to the gesture detection posture
If it is 0, no program will be run

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_Dist.py

Code path:

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

Import 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
#Import transforms3d library for handling transformations in 3D space, performing
conversions between quaternions, rotation matrices and Euler angles, supporting
3D
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
from Arm_Lib import Arm_Device
```

Program parameter initialization, create publishers and subscribers

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

         # Initialize 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 publishers
        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)

        # Initialize tools
        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]])
```

Main image processing function TagDetect

```
def TagDetect(self,color_frame,depth_frame):
    #rgb_image
    #Receive color image topic message, convert message data to image data
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'rgb8')
    result_image = np.copy(rgb_image)
    #depth_image
    #Receive depth image topic message, convert message data to 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 detect function, pass parameters,
    '''
    cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY): Convert RGB image to grayscale
image for tag detection.
    False: Indicates not estimating tag pose.
    None: Indicates no camera parameters provided, may only perform simple
detection.
    0.025: May be the set tag size (unit is usually meters), used to help
detection algorithm determine tag size
    Returns a detection result, including information such as 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 each tag in tags, non-essential step
    tags = sorted(tags, key=lambda tag: tag.tag_id) # Seems to be already in
ascending order, no need for manual sorting
    #Call draw_tags function, which draws recognized AprilTag related information
on the color image, including corners, 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 spacebar pressed, after pressing change
self.pubPos_flag value, indicating AprilTag related information can be published
    if key == 32:
        self.pubPos_flag = True
    #Check the length of tags, greater than 0 means AprilTag is detected and
AprilTag grasping completion flag
    if len(tags) > 0 and self.done_flag == True:
        #Traverse AprilTags
        for i in range(len(tags)):
            #AprilTag center xy values are stored in Center_x_list list and
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
            #Calculate depth value of center coordinates
            cz = depth_image_info[int(cy),int(cx)]/1000
            #Call compute_heigh function to calculate AprilTag height,
parameters passed are AprilTag center coordinates and center point depth value,
returns a position list, pose[2] represents z value, which is height value
            pose = self.compute_heigh(cx,cy,cz)
            heigh_detect = round(pose[2],4)*1000 - 12
            heigh = 'heigh: ' + str(heigh_detect) + 'mm'
            #Calculate distance value of AprilTag from base coordinate system,
here uses Euclidean distance formula to calculate distance from xy to origin
(0,0), scale up this value, convert unit to millimeters, pose[1] represents
center point y value in world coordinate system, pose[0] represents center point
x value in 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'
            #Draw height and distance values on 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 detected AprilTag distance from base coordinate system is
smaller than set distance threshold and self.pubPos_flag value is True and 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)
                #Change self.detect_flag value, True means AprilTag smaller than
set threshold was recognized
                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
                #if depth information is not 0, it means it's valid data, then
publish AprilTag information message
                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 no AprilTag smaller than
distance threshold was recognized and set distance threshold is not 0 and
AprilTag message publishing is enabled, meeting these three conditions means no
AprilTag smaller than distance threshold was recognized.
        if self.detect_flag != True and  self.set_dist!=0 and
self.pubPos_flag==True:
            print("----------------------------------------")
            self.set_dist!=0
            #Robotic arm performs "head shaking" action group and buzzer sounds
            self.shake()
            #time.sleep(2)
            self.pub_arm(self.search_joints)
            grasp_done = Bool()
            grasp_done.data = True
            #Publish grasping completion topic for next gesture recognition node
program to publish gesture recognition results
            self.pubGraspStatus.publish(grasp_done)
            #time.sleep(2)
        self.pubPos_flag = False
        #If no AprilTag is recognized after pressing spacebar, robotic arm also
performs "head shaking" action group, buzzer sounds, then returns to 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 200mm
    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].