

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
```

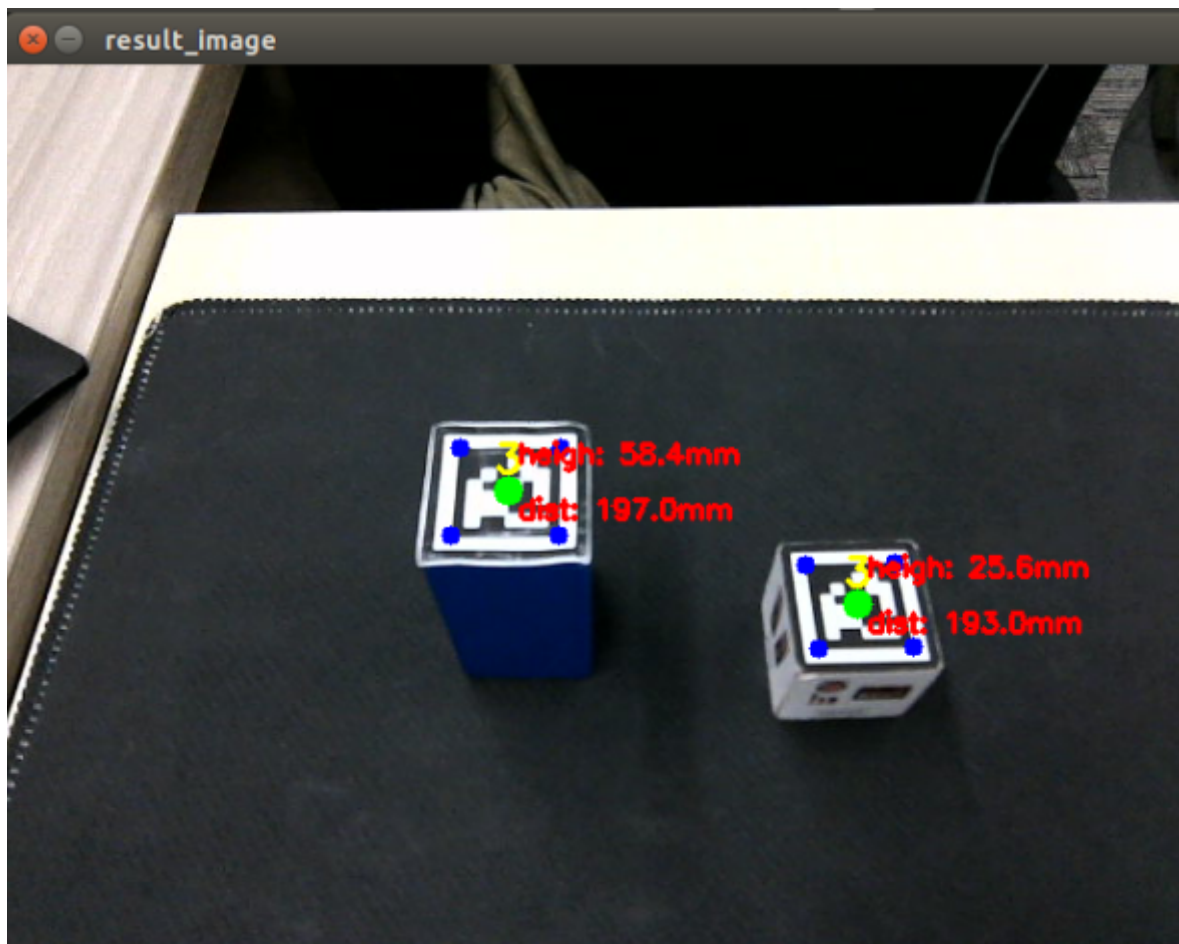
1. Function Description

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 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 kinematics_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
```



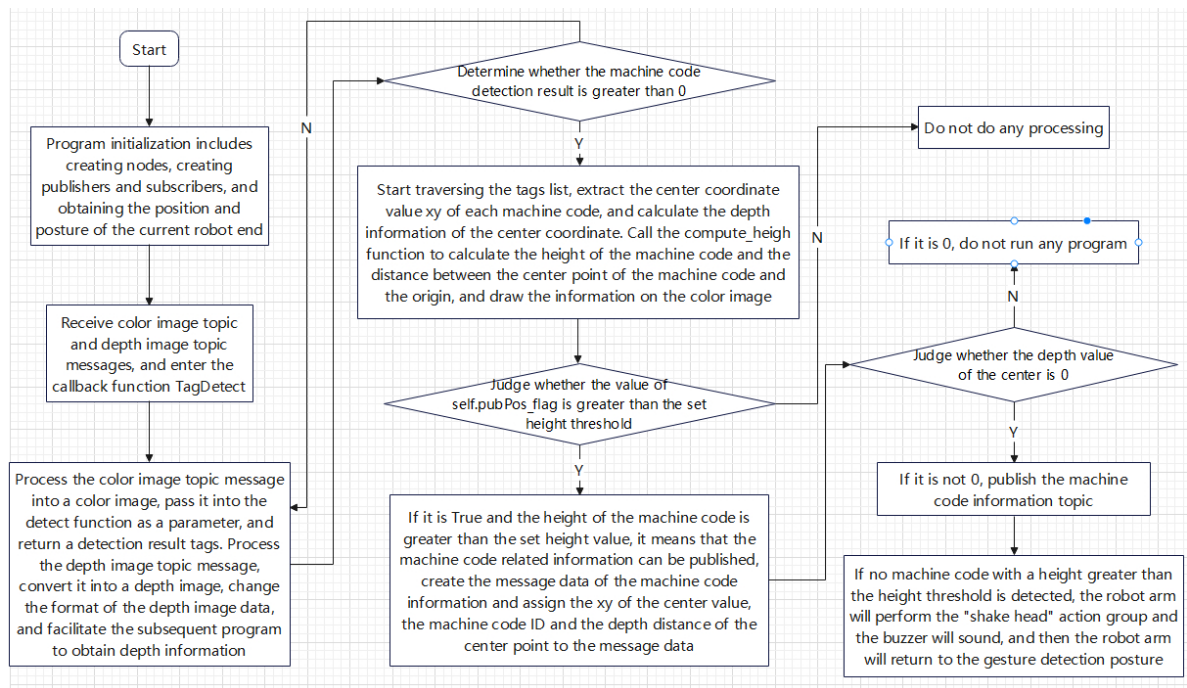
2.2. Operation

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 and the buzzer will sound, then the robotic arm returns to the gesture recognition posture.

Height threshold calculation: $30 + \text{gesture recognition result} * 10$

3. Program Flowchart

apriltag_list_Hight.py



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

```
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 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)
            #Scale up the height value, convert unit to millimeters
            heigh_detect = round(pose[2],4)*1000
            heigh = 'heigh: ' + str(heigh_detect) + 'mm'
            #Calculate distance value of AprilTag from base coordinate system,
scale up this value, convert 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'
            #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 height is greater than set height threshold
and self.pubPos_flag value is True and 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 self.detect_flag value, True means AprilTag higher than
        set threshold was recognized
        self.detect_flag = True
        #Assign values to message data, id value is AprilTag id, x and y
        are AprilTag center values, z is center point depth value, here scaled down by
        1000 times, 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 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 higher than height
        threshold was recognized and set height threshold is not 0 and AprilTag message
        publishing is enabled, meeting these three conditions means no AprilTag higher
        than height threshold was recognized.
        if self.detect_flag != True and self.set_height!=0 and
        self.pubPos_flag==True:
            print("-----")
            self.set_height!=0
            #Robotic arm performs "head shaking" action group
            self.shake()
            #time.sleep(2)
            #Return to gesture recognition posture, prepare for next gesture
            recognition
            self.pub_arm(self.search_joints)
            #Publish grasping completion topic for next gesture recognition node
            program to publish gesture recognition results
            grasp_done = Bool()
            grasp_done.data = True
            self.pubGraspStatus.publish(grasp_done)
            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 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].