Mediapipe Gesture-AprilTag ID 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

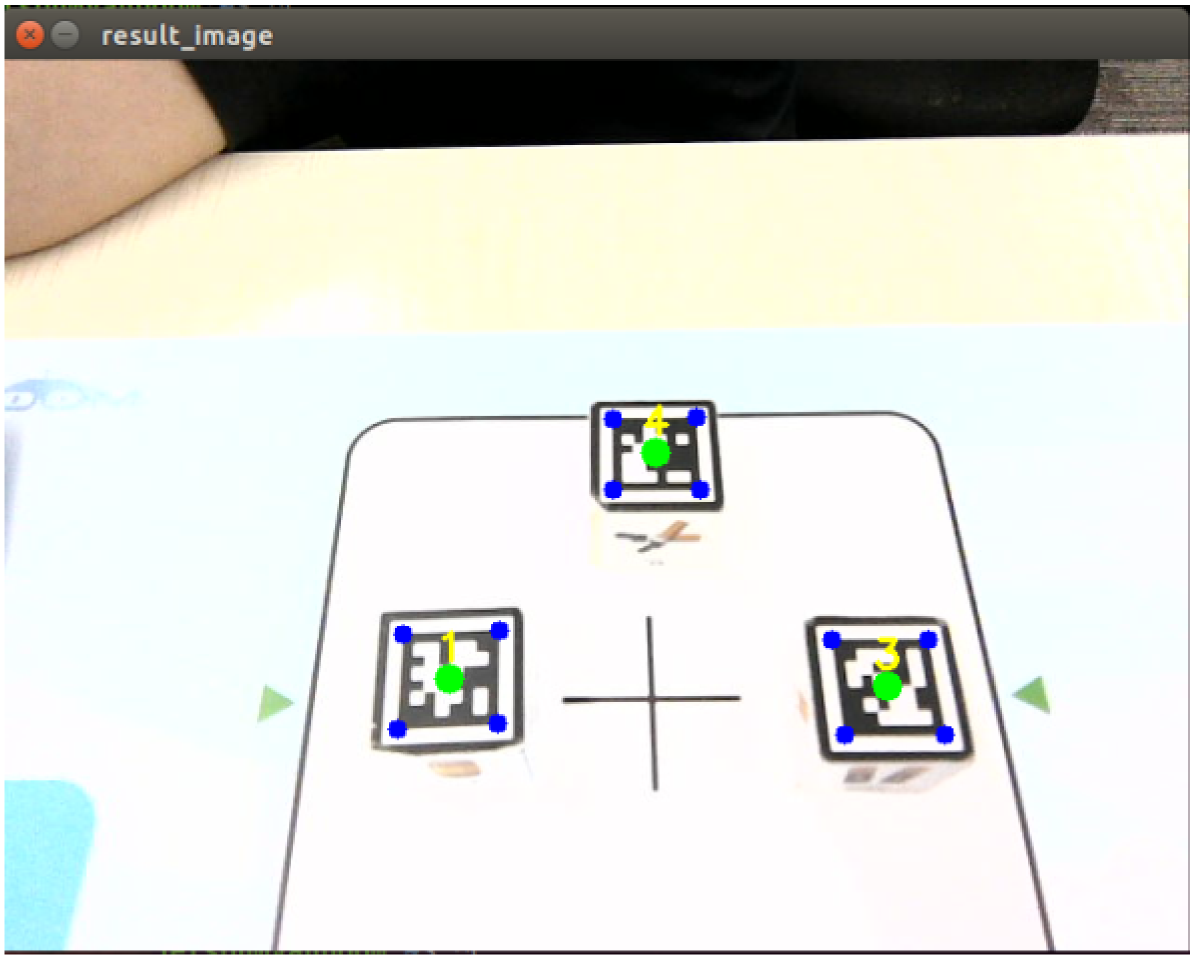
After the program runs, the camera captures images and recognizes gestures. Gestures range from 1 to 4. Through the recognized gesture, the target AprilTag ID value is given; then the robotic arm will change its posture to search for the target ID AprilTag. If it is detected and recognized, it will grasp and place it at the set position, then return to the recognition posture to continue recognition; if the target AprilTag is not detected, the robotic arm returns to the gesture recognition posture.

2. Startup and Operation

2.1. Startup Commands

Enter the following commands in the terminal:

```
#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 apriltagID_finger_detect
#Start robotic arm grasping program:
ros2 run dofbot_pro_driver grasp
#Start Mediapipe gesture recognition program:
ros2 run dofbot_pro_apriltag MediapipeGesture
```

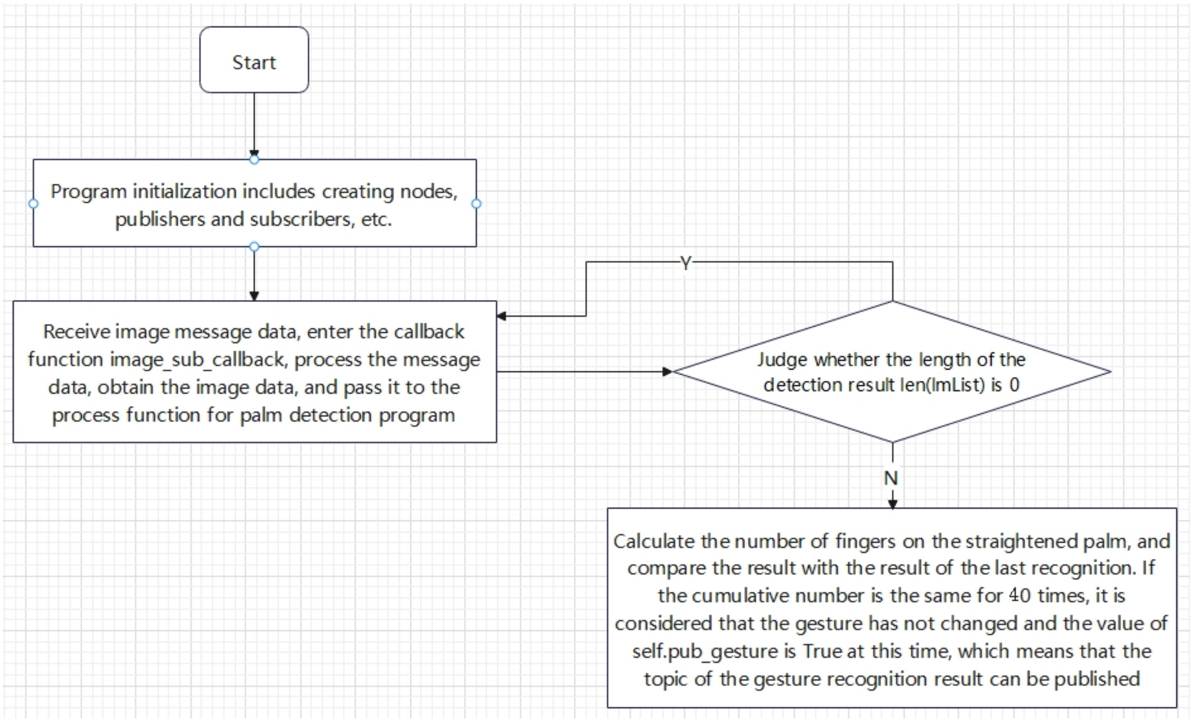


2.2. Operation

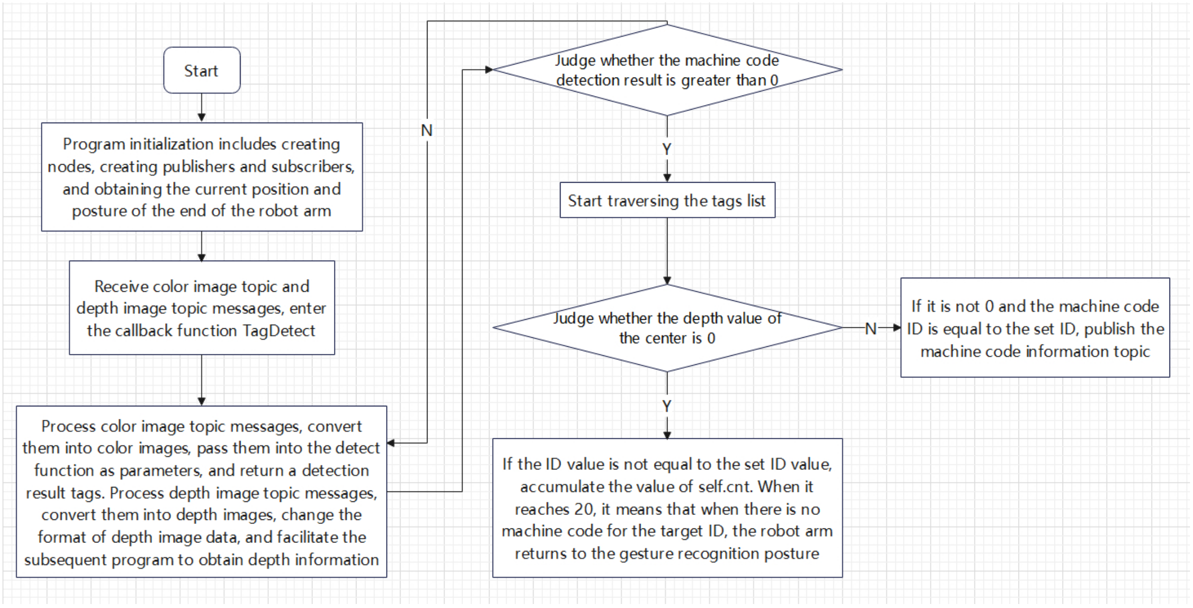
After the program starts, the robotic arm will initially present a gesture recognition posture. The recognizable gestures range from one to four, with corresponding gestures as follows; gesture recognition waits for about 3 seconds, waiting for the AprilTag to change posture to the AprilTag detection and recognition posture, press the spacebar to start recognition; if the target AprilTag is recognized, it will lower the gripper to grasp, then place it at the set position; if no AprilTag matching the target ID value is recognized, the robotic arm returns to the initial gesture recognition posture.

3. Program Flowchart

MediapipeGesture.py


Start
Program initialization includes creating nodes, publishers and subscribers, etc.
Receive image message data, enter the callback function image_sub_callback, process the message data, obtain the image data, and pass it to the process function for palm detection program
Judge whether the length of the detection result len(lmList) is 0
Y
N
Calculate the number of fingers on the straightened palm, and compare the result with the result of the last recognition. If the cumulative number is the same for 40 times, it is considered that the gesture has not changed and the value of self.pub_gesture is True at this time, which means that the topic of the gesture recognition result can be published

apriltagID_finger_detect.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, enter the callback function TagDetect
Process color image topic messages, convert them into color images, pass them into the detect function as parameters, and return a detection result tags. Process depth image topic messages, convert them into depth images, change the format of 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
Judge whether the depth value of the center is 0
N
If it is not 0 and the machine code ID is equal to the set ID, publish the machine code information topic
Y
If the ID value is not equal to the set ID value, accumulate the value of self.cnt. When it reaches 20, it means that when there is no machine code for the target ID, the robot arm returns to the gesture recognition 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
```

Import necessary library files

```
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
import numpy as np
from dofbot_pro_apriltag.media_library import *
from collections import defaultdict
import threading
from std_msgs.msg import Float32, Int8, Bool
from dofbot_pro_interface.msg import *
from time import sleep, time
```

Program parameter initialization, create publishers and subscribers

```
def __init__(self):
    super().__init__('detect_gesture_node')
   self.hand_detector = HandDetector()
        self.pub_gesture = True

        # ROS2 subscribers
        qos = QoSProfile(depth=10)
        self.img_sub = self.create_subscription(ImageMsg,
"/image_data",self.image_sub_callback, qos)
        self.grasp_sub=
self.create_subscription(Bool,'grasp_done',self.GraspStatusCallback, qos)
        # ROS2 publishers
        self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", qos)
        self.pub_targetID = self.create_publisher(Int8, "TargetId", qos)
        self.pTime = self.cTime = 0
        self.cnt = 0
        self.last_sum = 0
        self.detect_gesture = Int8()
        self.pTime = 0
        # Initialize other variables
        self.detect_gesture_joints = [90.0, 150.0, 12.0, 20.0, 90.0, 30.0]
        self.img = np.zeros((480, 640, 3), dtype=np.uint8)
        self.pubTargetArm(self.detect_gesture_joints)
```

Image processing function image_sub_callback

```
def image_sub_callback(self,msg):
    # Convert custom image message to image
    image = np.ndarray(shape=(msg.height, msg.width, msg.channels),
dtype=np.uint8, buffer=msg.data)
    # Convert rgb to opencv's bgr order
    self.img[:,:,0],self.img[:,:,1],self.img[:,:,2] =
image[:,:,2],image[:,:,1],image[:,:,0]
    frame = self.img.copy()
    #Pass the converted image to the process function for image processing and
recognition
    self.process(frame)
```

Image processing function process

```
def process(self, frame):
    #Pass the image to the findHands function of the created hand_detector
object. This function is used for real-time hand tracking and recognition
detection
    frame, lmList, bbox = self.hand_detector.findHands(frame)
    #If the detection result is not 0, it means a hand is detected. Pass the
detection result to the Gesture_Detect_threading thread function and start the
thread
    if len(lmList) != 0:
        threading.Thread(target=self.Gesture_Detect_threading, args=
(lmList,bbox)).start()
    #Calculate frame rate
    self.cTime = time()
    fps = 1 / (self.cTime - self.pTime)
    self.pTime = self.cTime
    text = "FPS : " + str(int(fps))
    #Draw the frame rate on the color image
    cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255),
1)
    if cv.waitKey(1) & 0xFF == ord('q'):
        cv.destroyAllWindows()
        rospy.signal_shutdown("exit")
    cv.imshow('frame', frame)
```

Gesture recognition thread function Gesture_Detect_threading

```
def Gesture_Detect_threading(self, lmList,bbox):
    #Pass the detected hand result as a parameter to the fingersUp function. This
function returns a list of extended fingers. The list index corresponds to 5
fingers. Which fingers are extended will have a value of 1 at the corresponding
array index, otherwise 0
    fingers = self.hand_detector.fingersUp(lmList)
    #Calculate the sum of the fingers list values to determine how many fingers
are in the extended state, and assign this value to self.last_sum to compare with
the previous sum of extended fingers
    self.last_sum = sum(fingers)
    print(self.pub_gesture)
    if sum(fingers) == self.last_sum:
        print("---------------------------")
        self.cnt = self.cnt + 1
        print("cnt: ",self.cnt)
        #If the current finger sum equals the previous one, accumulate the
count. When the same count reaches 30 times, it means the gesture has not
changed, and if self.pub_gesture is True at this time, you can assign a value to
the message and publish the /TargetId topic
        if self.cnt==30 and self.pub_gesture == True:
            print("sum of fingers: ",self.last_sum)
            #Change self.pub_gesture state value to prevent topic publishing
after misrecognition
            self.pub_gesture = False
            #Assign value to the created message data. The value of
self.last_sum corresponds to the AprilTag ID
            self.detect_gesture.data = self.last_sum
            self.pub_targetID.publish(self.detect_gesture)
            #Restore data for next recognition
            self.last_sum = 0
```

4.2. apriltagID_finger_detect.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/dofbot_pro_apriltag/apriltagI
D_finger_detect.py
```

Import necessary library files

```
import cv2
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
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 *  # Assume services have been migrated
from dofbot_pro_interface.msg import ArmJoint, AprilTagInfo
import pyzbar.pyzbar as pyzbar
import time
import queue
import os
encoding = ['16UC1', '32FC1']
```

Program parameter initialization, create publishers and subscribers

```
def __init__(self):
    super().__init__('apriltag_detect_node')  # ROS2 node initialization

        # 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]



        # 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

        # 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
        )

        # State variables
        self.pr_time = time.time()
        self.target_id = 0
        self.cnt = 0
        self.detect_flag = False
        self.pub_arm(self.detect_joints)
```

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))
    #Wait for keyboard input, 32 means spacebar pressed, after pressing change
self.pubPos_flag value, indicating AprilTag related information can be published
    key = cv2.waitKey(10)
    if key == 32:
        self.pubPos_flag = True
    if self.target_id!=0:
        print("Get th target id,start to search it.")
        self.pub_arm(self.search_joints)
        #time.sleep(0.5)
        #Check the length of tags, greater than 0 means AprilTag is detected
        if len(tags) > 0 :
            #Traverse AprilTags
            for i in range(len(tags)):
                #self.tag_info_pub.publish(tag)
                if self.pubPos_flag == True:
                    center_x, center_y = tags[i].center
                    cv.circle(result_image, (int(center_x),int(center_y)), 10,
(0,210,255), thickness=-1)
                    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)
                    #Create AprilTag information message data
                    tag = AprilTagInfo()
                    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
                    #If the current AprilTag center point depth distance is
greater than 0 and the current AprilTag ID equals our target ID value, then
publish topic
                    if tag.z>0 and tag.id == self.target_id:
                        self.tag_info_pub.publish(tag)
                        self.pubPos_flag = False
                        self.cnt = 0
                    else:
                        if tag.z!=0:
                            print("Invalid distance.")
                        if tag.id != self.target_id:
                            print("Do not find the target id tag.")
                            self.cnt = self.cnt + 1
                            #If it's not our target ID, accumulate. When
accumulation reaches 20, it means no matching ID was detected, return to gesture
recognition posture
                            if self.cnt == 20:
                                self.cnt = 0
                                #Return to gesture recognition posture
                                self.pub_arm(self.detect_joints)
                                self.target_id = 0
        else:
            self.cnt = self.cnt + 1
            #If no AprilTag is detected, accumulate count. When count reaches 
100, robotic arm returns to gesture recognition posture
            if self.cnt == 100:
                self.cnt = 0
                self.pub_arm(self.detect_joints)
                self.target_id = 0
    result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
    cv2.imshow("result_image", result_image)
    key = cv2.waitKey(1)
```

Look at the two callback functions

```
#Grasping result callback function
def GraspStatusCallback(self,msg):
    #Change self.pubPos_flag so that topic messages can be published next time
    if msg.data == True:
        self.pubPos_flag = True
#Get target ID callback function
def GetTargetIDCallback(self,msg):
    #Change self.target_id value
    self.target_id = msg.data
    print("Get th traget is ",self.target_id)
```

4.3. grasp.py

You can refer to section 4.2 [grasp.py] in tutorial [12. 3D Sorting and Grasping Course\1. AprilTag ID sorting].