AprilTag Tracking and Grasping

Before starting this function, you need to close the main program and APP processes. If you need to restart the main 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, it recognizes the AprilTag and tracks it, making the AprilTag center coincide with the image center. After the robotic arm is stationary for 2-3 seconds, if the depth distance is valid (not 0), the buzzer will sound once, and the robotic arm will adjust its posture to grasp the AprilTag. After grasping, it places it at the designated position, then returns to the AprilTag recognition posture.

2. Startup and Operation

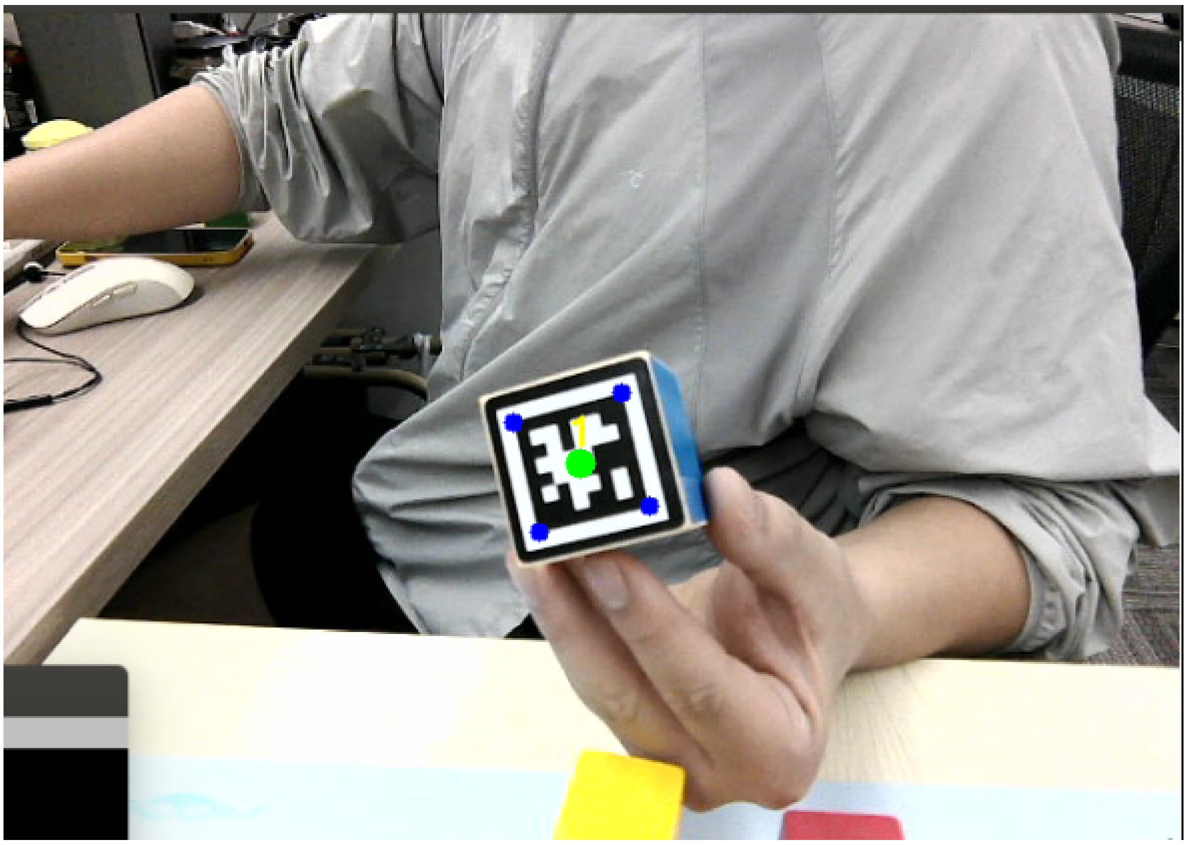
2.1. Startup Commands

Enter the following commands in the terminal to start:

```
# Start camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
# Start low-level control:
ros2 run dofbot_pro_driver arm_driver
# Start inverse kinematics program:
ros2 run dofbot_pro_info kinemarics_dofbot
# Start AprilTag tracking and grasping program:
ros2 run dofbot_pro_apriltag apriltag_follow
```

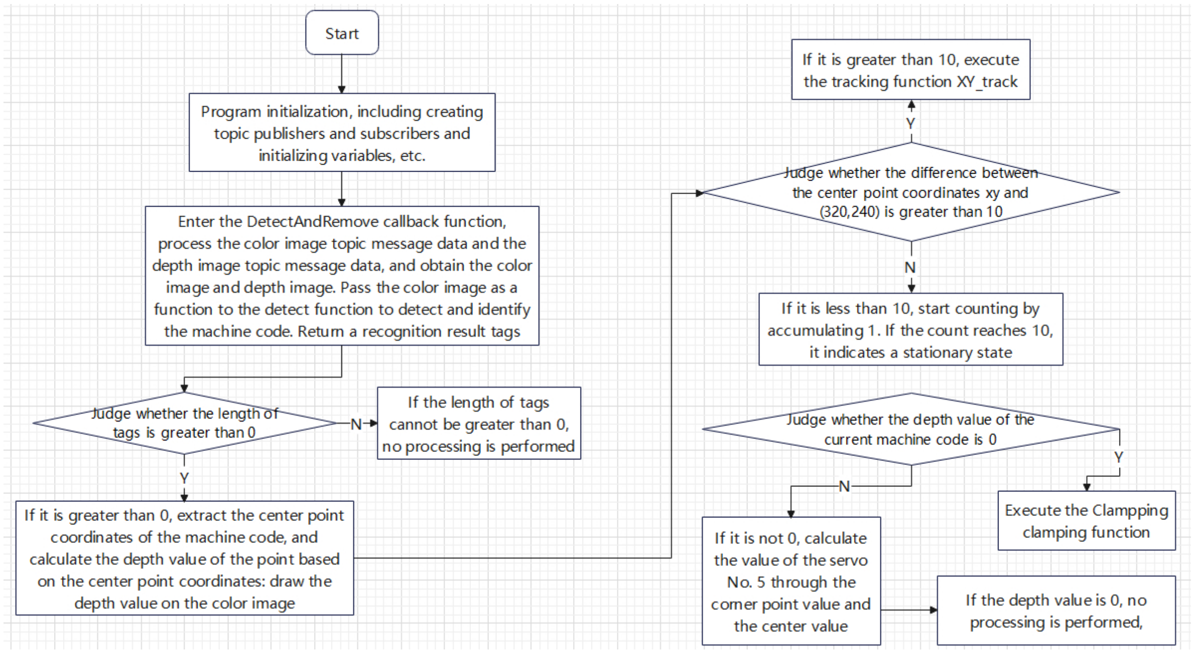
2.2. Operation

After the program starts, hold a 4cm*4cm AprilTag wooden block in front of the camera. The robotic arm will adjust its posture to make the AprilTag center coincide with the image center. Slowly move the AprilTag, and the robotic arm will follow the AprilTag movement, continuously adjusting its posture. After the AprilTag center coincides with the image center, if the depth information in the upper left corner of the image is not 0 and the distance from the AprilTag to the base is less than 30cm, it means the AprilTag center coordinate depth value is valid and within the robotic arm's grasping range. The buzzer will sound once, and the robotic arm will adjust its posture to grasp the AprilTag based on its position. After grasping, it places it at the designated position and finally returns to the recognition posture. If the robotic arm cannot meet the conditions (depth information not 0 and distance less than 30cm, greater than 18cm), you need to move the AprilTag back and forth again to make it stationary after tracking to meet the grasping conditions.



3. Program Flowchart

apriltag_follow.py


Start
Program initialization, including creating topic publishers and subscribers and initializing variables, etc.
Enter the DetectAndRemove callback function, process the color image topic message data and the depth image topic message data, and obtain the color image and depth image. Pass the color image as a function to the detect function to detect and identify the machine code. Return a recognition result tags
Judge whether the length of tags is greater than 0
N
If the length of tags cannot be greater than 0, no processing is performed
Y
If it is greater than 0, extract the center point coordinates of the machine code, and calculate the depth value of the point based on the center point coordinates: draw the depth value on the color image
Judge whether the difference between the center point coordinates xy and (320,240) is greater than 10
Y
If it is greater than 10, execute the tracking function XY_track
N
If it is less than 10, start counting by accumulating 1. If the count reaches 10, it indicates a stationary state
Judge whether the depth value of the current machine code is 0
Y
Execute the Clampping clamping function
N
If it is not 0, calculate the value of the servo No. 5 through the corner point value and the center value
If the depth value is 0, no processing is performed,

4. Core Code Analysis

Code path:

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

Import necessary libraries:

```
import rclpy
```

```
from rclpy.node import Node
import cv2
import numpy as np
import os
import time
from message_filters import ApproximateTimeSynchronizer, Subscriber
import cv2 as cv
from sensor_msgs.msg import Image
from std_msgs.msg import Float32
from cv_bridge import CvBridge

from dofbot_pro_apriltag.vutils import draw_tags
from dt_apriltags import Detector
from dofbot_pro_apriltag.Dofbot_Track import *

encoding = ['16UC1', '32FC1']
```

Program parameter initialization, create publishers, subscribers, etc.:

```
def __init__(self):
    super().__init__('apriltag_tracking')
    #Create a tracking and grasping object
    self.dofbot_tracker = DofbotTrack()
    #Create AprilTag object
    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)
    #Count variable, used to record the number of times conditions are met
    self.cnt = 0
    #Current depth value of AprilTag center
    self.cur_distance = 0.0
    #Create bridges for converting color and depth image topic message data to
image data
    self.rgb_bridge = CvBridge()
    self.depth_bridge = CvBridge()
     # 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.DetectAndRemove)
```

Focus on the DetectAndRemove callback function:

```
def DetectAndRemove(self,color_frame,depth_frame):
    #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)
    #Receive depth image topic message, convert message data to image data
    depth_image = self.depth_bridge.imgmsg_to_cv2(depth_frame, encoding[1])
    frame = cv2.resize(depth_image, (640, 480))
    depth_image_info = frame.astype(np.float32)
    #Call detect function with parameters
    '''
    cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY): Convert RGB image to grayscale
image for tag detection.
    False: Indicates not to estimate tag pose.
    None: Indicates no camera parameters provided, may only perform simple
detection.
    0.025: May be the set tag size (unit usually meters), used to help detection
algorithm determine tag size
    Returns detection result, including position, ID, and bounding box
information for 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)
    #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))
    #Check if the length of detection result tags is greater than 0, indicating
AprilTag was detected
    if len(tags) > 0 :
        #print("tag: ",tags)
        #Get AprilTag center point coordinates
        center_x, center_y = tags[0].center
        #Calculate depth value of AprilTag center coordinates
        self.cur_distance = depth_image_info[int(center_y),int(center_x)]/1000.0
        dist = round(self.cur_distance,3)
        dist = 'dist: ' + str(dist) + 'm'
        #Draw center point depth value on color image
        cv2.putText(result_image, dist,  (30, 30), cv2.FONT_HERSHEY_SIMPLEX,
1.0, (255, 0, 0), 2)
        print("cur_distance: ",self.cur_distance)
        #If the AprilTag center point coordinates differ from image center point
(320, 240) by more than 2, meaning not within acceptable range, execute tracking
program to adjust robotic arm state to bring AprilTag center value within
acceptable range
        if abs(center_x-320) >2 or abs(center_y-240)>2:
            #Execute tracking program, input is current AprilTag center value
            self.dofbot_tracker.XY_track(center_x,center_y)
        print("=====================================")
        #If AprilTag center point coordinates differ from image center point
(320, 240) by less than 2, can be considered that AprilTag center value is in the
middle of the image
        if abs(center_x-320) <2 and abs(center_y-240)<2:
            print("center_x: ",center_x)
            print("center_y: ",center_y)
            #Under satisfying conditions, accumulate self.cnt
            self.cnt = self.cnt + 1
            #When accumulated count reaches 50, it means AprilTag center value
is stationary in the middle of the image
            if self.cnt==50:
                #Clear self.cnt count
                self.cnt = 0
                print("take it now!")
                print("last_joint: ",self.dofbot_tracker.cur_joints)
                #Check if current depth value is 0, non-zero means the value is
valid
                if self.cur_distance!=0:
                    #Calculate servo 5 value through corner coordinates
                    vx = int(tags[0].corners[0][0]) - int(center_x)
                    vy = int(tags[0].corners[0][1]) - int(center_y)
                    angle_radians = math.atan2(vy, vx)
                    angle_degrees = math.degrees(angle_radians)
                    print("angle_degrees: ",angle_degrees)
                    if abs(angle_degrees) >90:
                        compute_angle = abs(angle_degrees) - 45
                    else:
                        compute_angle = abs(angle_degrees)
                    set_joint5 = compute_angle*2
                    if set_joint5>135:
                        print("--------------------------------------")
                        self.dofbot_tracker.set_joint5 = set_joint5 - 90
                    else:
                        self.dofbot_tracker.set_joint5 = set_joint5
                    #Execute grasping program, calling the Clamping function of
the created dofbot_tracker object, input parameters are AprilTag center value and
center point depth value

 self.dofbot_tracker.Clamping(center_x,center_y,self.cur_distance)

    result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
    cv2.imshow("result_image", result_image)
    key = cv2.waitKey(1)
```

Let's look at the implementation process of the Clamping function of the created dofbot_tracker object. This function is located in the Dofbot_Track library, Dofbot_Track library location

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_apriltag/dofbot_pro_apriltag/Dofbot_Tr
ack.py
```

```
def Clamping(self,cx,cy,cz):
    #Get current robotic arm end position and pose
    self.get_current_end_pos(self.cur_joints)
    #Start coordinate system conversion, finally get AprilTag center coordinates
position in world coordinate system
    camera_location = self.pixel_to_camera_depth((cx,cy),cz)
    PoseEndMat = np.matmul(self.EndToCamMat,
self.xyz_euler_to_mat(camera_location, (0, 0, 0)))
    EndPointMat = self.get_end_point_mat()
    WorldPose = np.matmul(EndPointMat, PoseEndMat)
    pose_T, pose_R = self.mat_to_xyz_euler(WorldPose)
    # Add offset parameters to compensate for deviations caused by servo value
differences
    pose_T[0] = pose_T[0] + self.x_offset
    pose_T[1] = pose_T[1] + self.y_offset
    pose_T[2] = pose_T[2] + self.z_offset
    print("pose_T: ",pose_T)
    #Call inverse kinematics IK algorithm, input parameter is pose_T obtained
through coordinate system conversion, which is AprilTag center coordinates
position in world coordinate system, calculate 6 servo values through IK
    request = kinemaricsRequest()
    #Target x value of robotic arm end, unit is m
    request.tar_x = pose_T[0]
    #Target y value of robotic arm end, unit is m
    request.tar_y = pose_T[1]
    #Target z value of robotic arm end, unit is m, 0.2 is scaling factor, make
minor adjustments based on actual situation
    request.tar_z = pose_T[2]
    #Specify service content as ik
    request.kin_name = "ik"
    #Target Roll value of robotic arm end, unit is radians, this value is current
robotic arm end roll value
    request.Roll = self.CurEndPos[3]
    print("calcutelate_request: ",request)
    try:
        response = self.client.call(request)
        joints = [0.0, 0.0, 0.0, 0.0, 0.0,0.0]
        #Assign service returned joint1-joint6 values to joints
        joints[0] = response.joint1 #response.joint1
        joints[1] = response.joint2
        joints[2] = response.joint3
        if response.joint4>90:
            joints[3] = 90
        else:
            joints[3] = response.joint4
        joints[4] = 90
        joints[5] = 30
        print("compute_joints: ",joints)
        #Calculate distance from AprilTag to robotic arm base coordinate system
        dist = math.sqrt(request.tar_y ** 2 + request.tar_x** 2)
        #If distance is between 18cm and 30cm, control robotic arm to move to
grasping point
        if dist>0.18 and dist<0.30:
            self.Buzzer()
            print("Clamp Mode.")
            #Execute pubTargetArm function, pass calculated joints values as
parameters
            self.pub_arm(joints)
            time.sleep(3.5)
            #Execute grasping action and movement, placement actions
            self.move()
        else:
            print("Too far or too close.")
    except Exception:
       rospy.loginfo("run error")
```