KCF Object 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, use the mouse to select the object to be tracked and grasped, press the spacebar to start tracking. The robotic arm will adjust its posture to make the tracked object's center coincide with the image center. After the robotic arm is stationary for 2-3 seconds, if the depth distance is valid (not 0) and the object is within the set grasping range, the buzzer will sound once, and the robotic arm will adjust its posture to grasp the object. After grasping, it places it at the designated position, then returns to the robotic arm's initial posture.

2. Startup and Operation

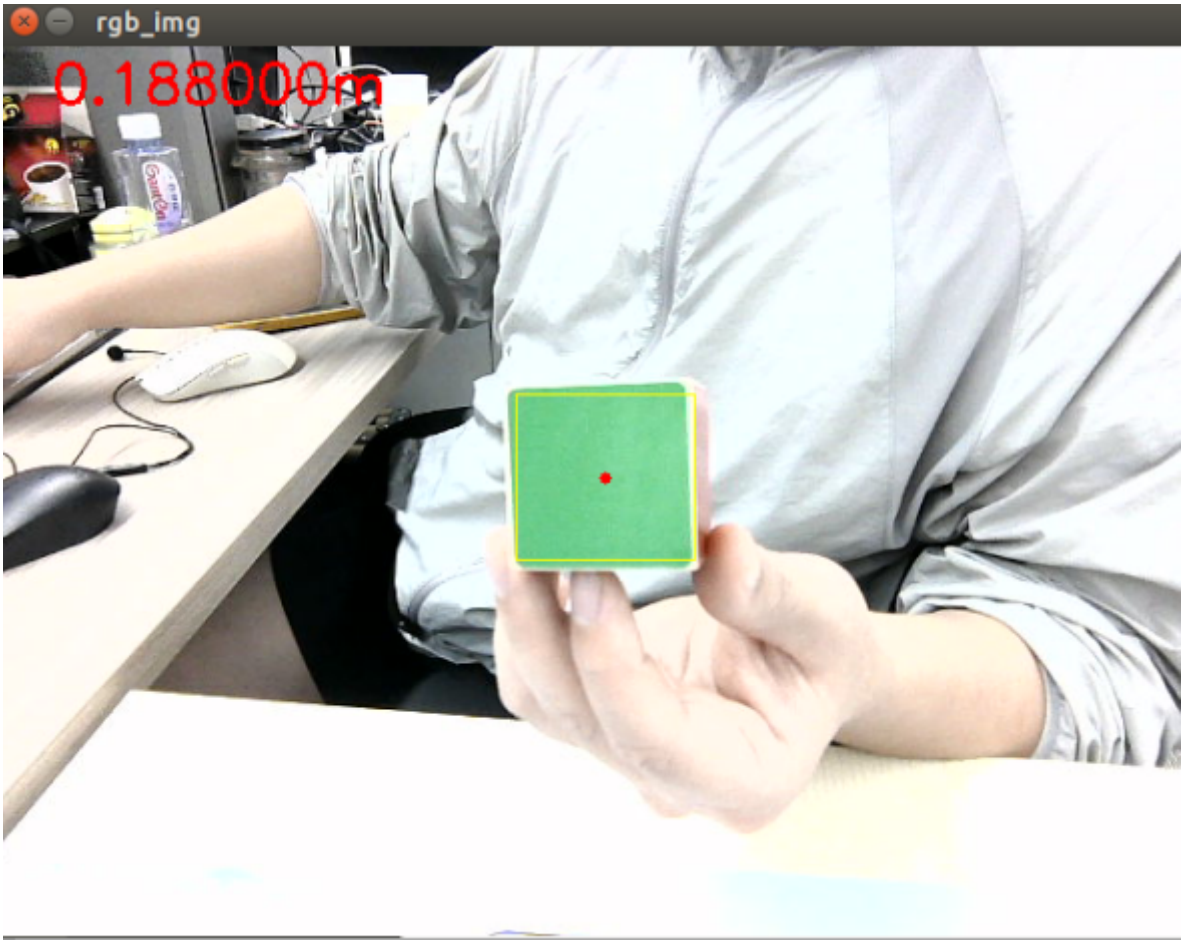
2.1. Startup Commands

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 KCF program:
ros2 run dofbot_pro_KCF KCF_Tracker
# Start tracking and grasping program:
ros2 run dofbot_pro_KCF KCF_TrackAndGrap
# Start gripper width calculation program:
ros2 run dofbot_pro_KCF compute_width
```

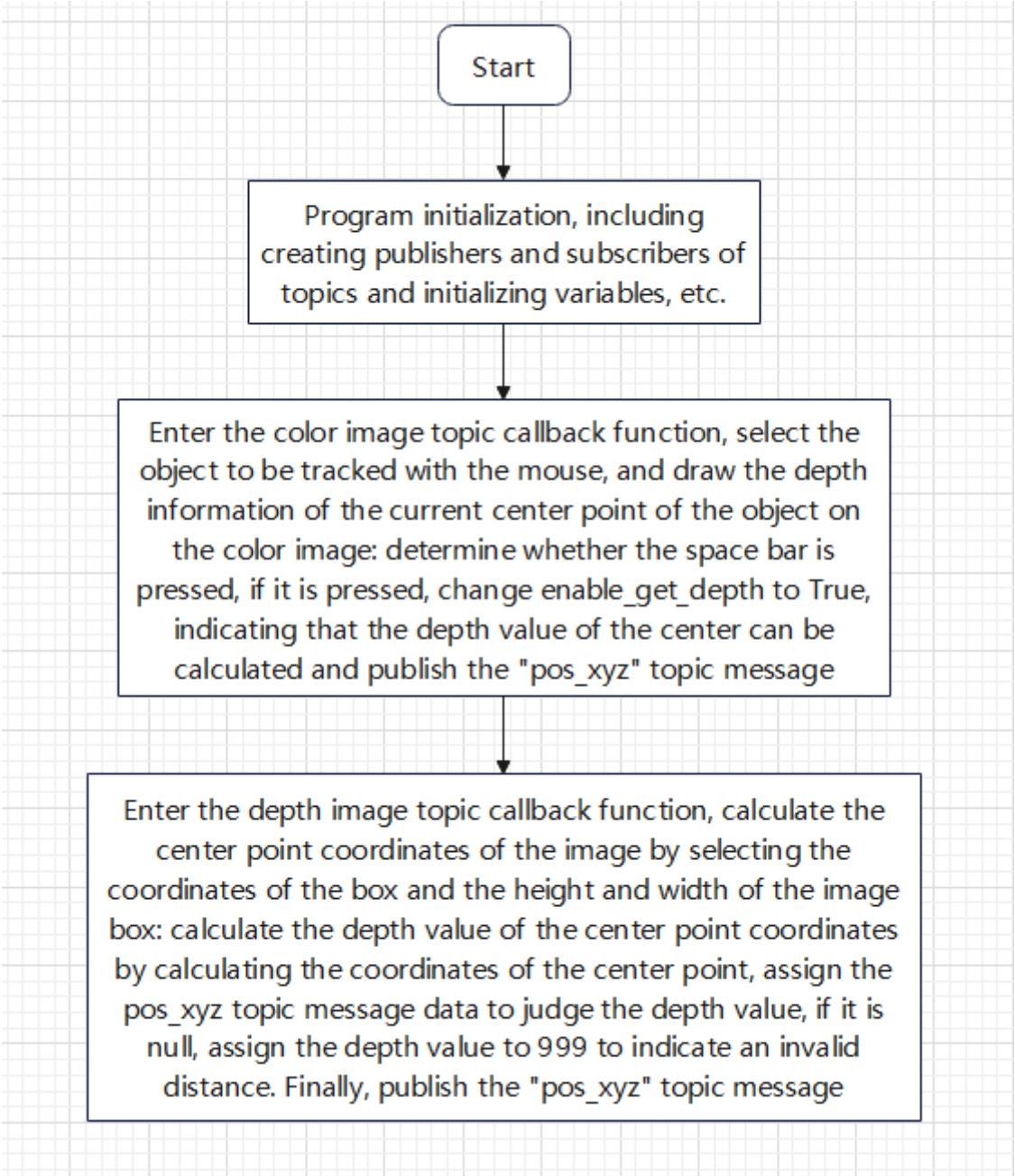
2.2. Operation

After the program starts, hold an object and click the mouse to select the held object in the image frame. The selection requirement is that the width of the selection box matches the object's width in the image. After releasing the mouse, the object will be framed in the image. Press the spacebar to start tracking. Slowly move the object, and the robotic arm will adjust its posture to keep the object's center coincident with the image center. After the tracking action is stationary, wait 2-3 seconds. If the depth distance is valid (not 0) and the object is within the set grasping range (greater than 18cm and less than 30cm), the buzzer will sound once, and the robotic arm will adjust its posture to grasp the object. After grasping, it places it at the designated position, then returns to the robotic arm's initial posture.

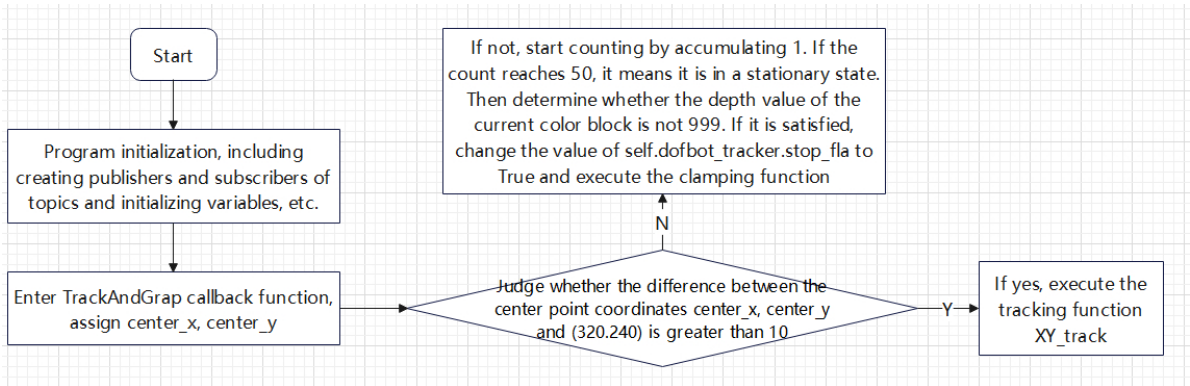


3. Program Flowchart

KCF_Tracker.py



KCF_TrackAndGrap.py



4. Core Code Analysis

4.1. KCF_Tracker.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_KCF/dofbot_pro_KCF/KCF_Tracker.py
```

Constructor function, mainly creates publishers and subscribers:

```
super().__init__('monoIdentify')

        self.point_pose = (0, 0, 0)
        self.circle = (0, 0, 0)
        self.hsv_range = ()
        self.circle_r = 0
        self.dyn_update = True
        self.select_flags = False
        self.gTracker_state = False
        self.windows_name = 'frame'
        self.init_joints = [90.0, 150.0, 12.0, 20.0, 90.0, 30.0]
        self.cols, self.rows = 0, 0
        self.Mouse_XY = (0, 0)
        self.end = 0
        self.cx = 0
        self.cy = 0

        self.rgb_bridge = CvBridge()
        self.depth_bridge = CvBridge()

        self.pubPoint = self.create_publisher(ArmJoint, "TargetAngle", 10)
        self.pub_pos = self.create_publisher(Position, "/pos_xyz", 10)
        self.joint6_pub = self.create_publisher(Float32,'joint6',1)

        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.kcfTrack)

        self.tracker_type = 'KCF'
        self.VideoSwitch = True
        self.img_flip = False
```

Color and depth image topic callback function:

```
def kcfTrack(self,color_frame,depth_frame):
        #rgb_image
        rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'bgr8')
        result_image = np.copy(rgb_image)
        #depth_image
        depth_image = self.depth_bridge.imgmsg_to_cv2(depth_frame, encoding[1])
```

```
        depth_to_color_image =
cv2.applyColorMap(cv2.convertScaleAbs(depth_image, alpha=0.03),
cv2.COLORMAP_JET)
        frame = cv.resize(depth_image, (640, 480))
        depth_image_info = frame.astype(np.float32)
        action = cv.waitKey(10) & 0xFF
        result_image = cv.resize(result_image, (640, 480))

        result_frame, binary = self.process(result_image,action)


        if self.cx!=0 and self.cy!=0 and self.circle_r>10 :
            center_x, center_y = self.cx,self.cy
            cv2.circle(depth_to_color_image,(int(center_x),int(center_y)),1,
(255,255,255),10)
            self.cur_distance =
depth_image_info[int(center_y),int(center_x)]/1000.0
            print("self.cur_distance: ",self.cur_distance)
            dist = round(self.cur_distance,3)
            dist = 'dist: ' + str(dist) + 'm'
            cv.putText(result_frame, dist,  (30, 30), cv.FONT_HERSHEY_SIMPLEX,
1.0, (255, 0, 0), 2)
            pos = Position()
            pos.x = center_x
            pos.y = center_y
            pos.z = self.cur_distance
            self.pub_pos.publish(pos)


        cur_time = time.time()
        fps = str(int(1/(cur_time - self.pr_time)))
        self.pr_time = cur_time
        cv2.putText(result_frame, fps, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 2)
        if len(binary) != 0: cv.imshow(self.windows_name, ManyImgs(1,
([result_frame, binary])))
        else:
            cv.imshow(self.windows_name, result_frame)
        cv2.imshow("depth_image", depth_to_color_image)
```

4.2. KCF_TrackAndGrap.py

Code path:

```
/home/jetson/dofbot_pro_ws/src/dofbot_pro_KCF/dofbot_pro_KCF/KCF_TrackAndGrap.py
```

Import necessary libraries:

```
import rclpy
from rclpy.node import Node
import numpy as np
from message_filters import ApproximateTimeSynchronizer, Subscriber
from dofbot_pro_KCF.Dofbot_Track import *
from dofbot_pro_interface.msg import *
```

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

```
def __init__(self):
    super().__init__('KCF_tracking')

    self.dofbot_tracker = DofbotTrack()

    self.pos_sub = Subscriber(self,Position,'/pos_xyz')
    self.time_sync =
ApproximateTimeSynchronizer([self.pos_sub],queue_size=10,slop
=0.5,allow_headerless=True)
    self.time_sync.registerCallback(self.TrackAndGrap)

    self.cur_distance = 0.0
    self.cnt = 0
    print("Init done!")
```

Main focus on the callback function:

```
def TrackAndGrap(self,position):

    if position.z != 0 :
        center_x, center_y = position.x,position.y
        self.cur_distance = position.z
        #If object's center point coordinates differ from image center point
(320, 240) by more than 10, meaning not within acceptable range, execute tracking
program to adjust robotic arm state to bring object's center value within
acceptable range
        if abs(center_x-320) >10 or abs(center_y-240)>10 :
            #Execute tracking program, input is current object's center value
            self.dofbot_tracker.XY_track(center_x,center_y)
        else:
            #If object's center point coordinates differ from image center point
(320, 240) by less than 10, can be considered that object's center value is in
the middle of the image, accumulate self.cnt
            self.cnt = self.cnt + 1
            #When accumulated count reaches 10, it means object center value is
stationary in the middle of the image
            if self.cnt == 10 :
                #Clear self.cnt count
                self.cnt = 0
                #Check if object's center value is not 999, 999 indicates invalid
distance
                if self.cur_distance!= 999:
                    self.dofbot_tracker.stop_flag = True

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