# KCF tracking and grabbing objects

# Tracking and grabbing machine code

Before starting this function, you need to close the process of the big program and APP. If you need to start the big program and APP again later, start the terminal,

```
bash ~/dofbot_pro/APP_DOFBOT_PRO/start_app.sh
```

## 1. Function description

After the program is started, use the mouse to frame the object to be tracked and clamped, and press the space bar to start tracking. The robot arm will adjust its posture so that the center of the tracked object coincides with the center of the image. After the robot arm is stationary, wait for 2-3 seconds. If the depth distance is valid (not 0) at this time, and the object is within the set clamping range at this time, the buzzer will sound, and then the robot arm will adjust its posture to clamp the object. After clamping, place it at the set position, and then return to the initial posture of the robot arm.

## 2. Start and operation

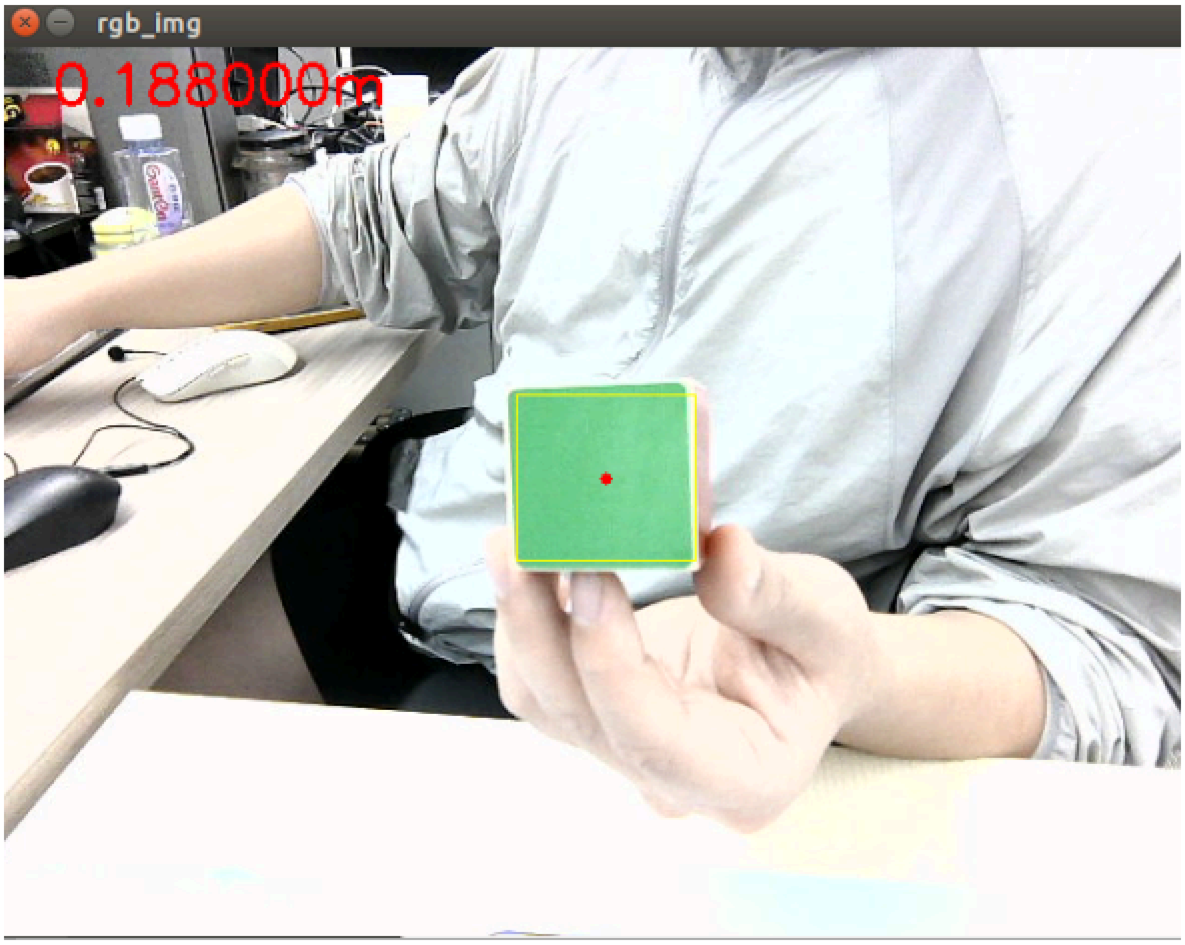
### 2.1. Start command

Enter the following command in the terminal to start

```
#Start the camera:
ros2 launch orbbec_camera dabai_dcw2.launch.py
#Start the underlying control:
ros2 run dofbot_pro_driver arm_driver
#Start the inverse solution program:
ros2 run dofbot_pro_info kinemarics_dofbot
#Start the KCF program:
ros2 run dofbot_pro_KCF KCF_Tracker
#Start the tracking and grabbing program:
ros2 run dofbot_pro_KCF KCF_TrackAndGrap
#Start the calculation of the gripper width program:
ros2 run dofbot_pro_KCF compute_width
```

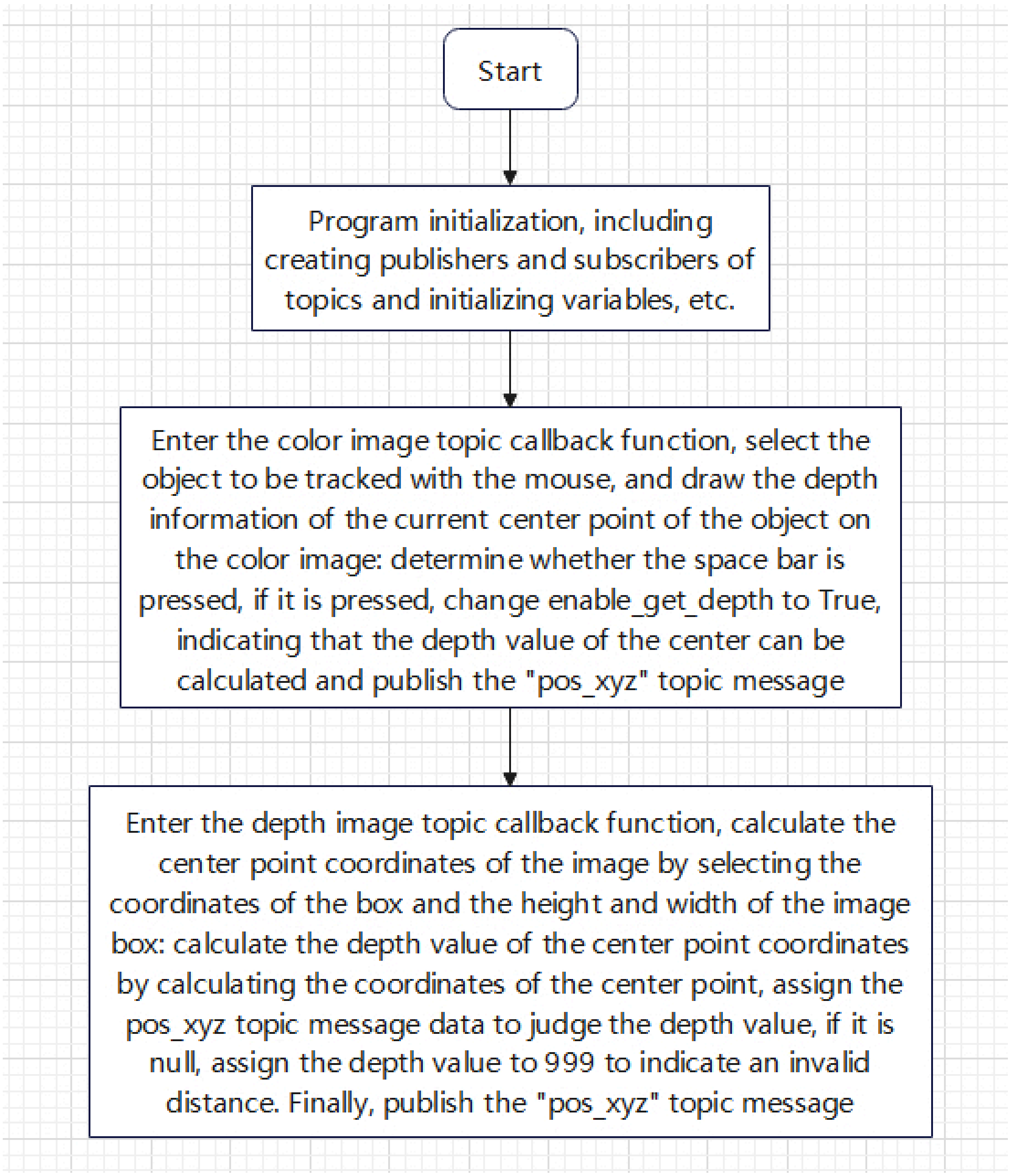
### 2.2, Operation

After the program is started, hold an object, click the mouse, and select the object in the image frame. The selection requirement is that the width of the selection frame is the same as the width of the object in the image. After releasing the mouse, the object will be framed in the image, and press the space bar to start tracking. Slowly move the object, and the robot will adjust its posture so that the center of the object always coincides with the center of the image. After the tracking action stops, wait for 2-3 seconds. If the depth distance is valid (not 0) and the object is within the set gripping range (greater than 18 cm and less than 30 cm), the buzzer will sound, and then the robot will adjust its posture to grip the object. After gripping, place it at the set position, and then return to the initial posture of the robot.

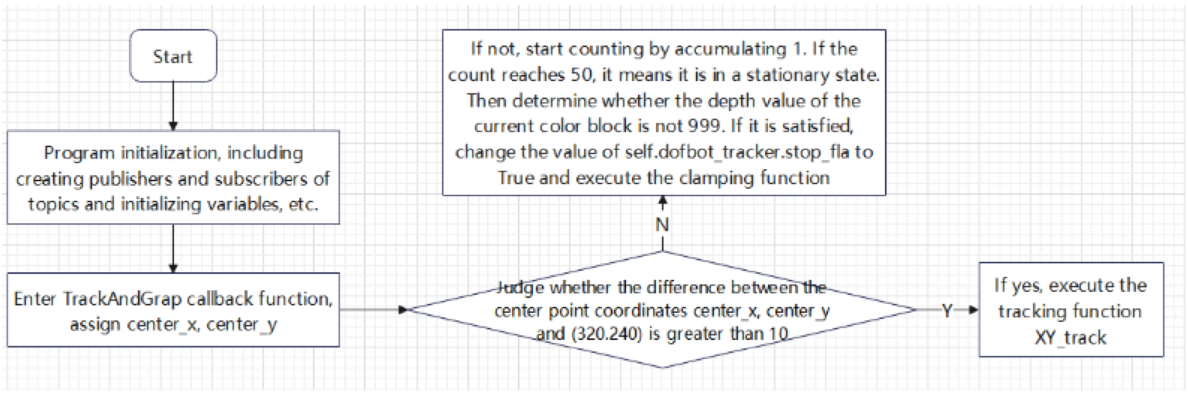


## 3. Program flow chart

KCF_Tracker.py


Start
Program initialization, including creating publishers and subscribers of topics and initializing variables, etc.
Enter the color image topic callback function, select the object to be tracked with the mouse, and draw the depth information of the current center point of the object on the color image: determine whether the space bar is pressed, if it is pressed, change enable_get_depth to True, indicating that the depth value of the center can be calculated and publish the "pos_xyz" topic message
Enter the depth image topic callback function, calculate the center point coordinates of the image by selecting the coordinates of the box and the height and width of the image box: calculate the depth value of the center point coordinates by calculating the coordinates of the center point, assign the pos_xyz topic message data to judge the depth value, if it is null, assign the depth value to 999 to indicate an invalid distance. Finally, publish the "pos_xyz" topic message

KCF_TrackAndGrap.py


Start
Program initialization, including creating publishers and subscribers of topics and initializing variables, etc.
Enter TrackAndGrap callback function, assign center_x, center_y
Judge whether the difference between the center point coordinates center_x, center_y and (320.240) is greater than 10
Y
If yes, execute the tracking function XY_track
N
If not, start counting by accumulating 1. If the count reaches 50, it means it is in a stationary state. Then determine whether the depth value of the current color block is not 999. If it is satisfied, change the value of self.dofbot_tracker.stop_fla to True and execute the clamping function

## 4. Core code analysis

### 4.1. KCF_Tracker.cpp

Code path:

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

Constructor, mainly to create 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 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 *
```

Initialize program parameters, 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!")
```

Mainly look at 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 the coordinates of the center point of the object and the center
point of the image (320, 240) are greater than 10, that is, they are not within
the acceptable range, the tracking program is executed and the state of the robot
arm is adjusted to make the center value of the object within the acceptable
range
        if abs(center_x-320) >10 or abs(center_y-240)>10 :
            #Execute the tracking program, input the center value of the current
object
            self.dofbot_tracker.XY_track(center_x,center_y)
        else:
            #If the coordinates of the center point of the machine code and the
coordinates of the center point of the image (320, 240) are less than 10, it can
be regarded that the center value of the object at this time is in the middle of
the image, and self.cnt is accumulated
            self.cnt = self.cnt + 1
            #When the cumulative number reaches 10, it means that the center
value of the machine code can be stationary in the middle of the image.
            if self.cnt == 10 :
                # Clear the count of self.cnt
                self.cnt = 0
                #Judge whether the center value of the object is not 999, 999
indicates an invalid distance
                if self.cur_distance!= 999:
                    self.dofbot_tracker.stop_flag = True

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