KCF object tracking

1. Introduction

KCF stands for **Kernel Correlation Filter**. It was proposed by Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista in 2014. It caused a sensation after it came out. This algorithm has very impressive performance in both tracking effect and tracking speed. It belongs to discriminative tracking, which mainly uses the given samples to train a discriminative classifier to determine whether the tracked object is the target or the surrounding background information. It mainly uses the rotation matrix to collect samples and uses the fast Fourier transform to accelerate the algorithm.

2. Start

2.1 Preparation

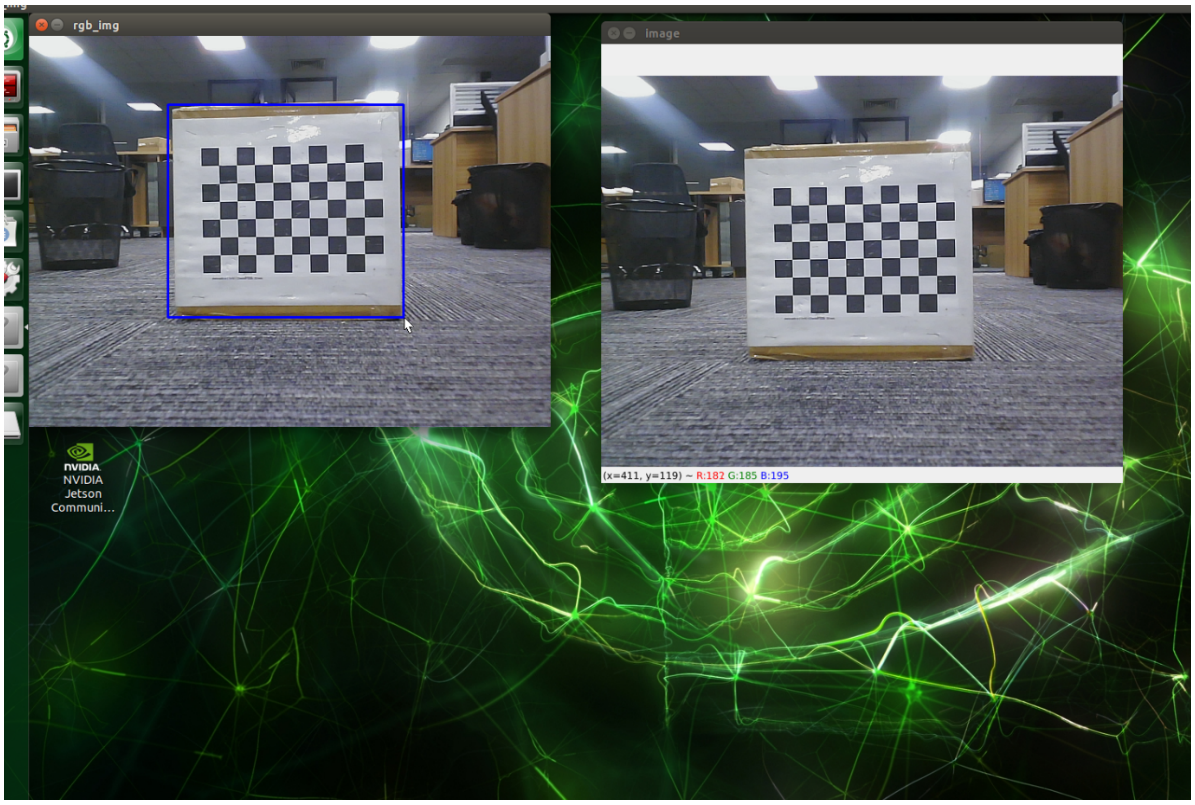
You need to prepare an object that is significantly different from the background as the object to be tracked by KCF.

2.2 Program description

After the program is started, the camera will capture an image. In the camera screen, press the s key (if you are logging in remotely via VNC, you need to press and hold it), and the screen will remain still.

Use the mouse to select the object to be tracked, select it with a blue box, press the space bar or the Enter key, the robot arm will enter the tracking mode, the blue box will turn into a yellow box, and you can move the object slowly, and the robotic arm will track the object.

Note: The object cannot be moved too fast to prevent the robotic arm from being unable to keep move.



2.3 Start program程序启动

- If you are using Jetson Orin NX/Jetson Orin Nano board. You need to enter the Docker environment using the following command.

```
sh ~/start_docker.sh
```

- Input following command to start the program

```
roslaunch jetcobot_advance kcf_tracking.launch
```

2.4 About code

Code path: ~/jetcobot_ws/src/jetcobot_advance/scripts/kcf_tracking.py

```
#!/usr/bin/env python3
# coding: utf8
import os
import cv2
import rospy
import queue
import numpy as np
from sensor_msgs.msg import Image
import fps
from pymycobot.mycobot import MyCobot
from simple_pid import PID


class KCFTrackingNode:
    def __init__(self):
        rospy.init_node('kcf_tracking')
        self.target_servox=0
        self.target_servoy=-90
        self.xservo_pid = PID(2, 0.05, 0.05)
        self.yservo_pid = PID(2, 0.05, 0.05)

        self.tracker = None
        self.enable_select = False
        self.fps = fps.FPS() # Frame rate counter
        self.mc = MyCobot(str(os.getenv('MY_SERIAL')), 1000000)
        self.mc.send_angles([0, 0, -90, 90, 0, -45], 50)
        print(cv2.__version__)

        # Subscribe to the Camera Image Topic
        source_image_topic = rospy.get_param('~source_image_topic',
'/camera/color/image_raw')
        rospy.loginfo("source_image_topic = {}".format(source_image_topic))
        self.image_sub = rospy.Subscriber(source_image_topic, Image,
self.image_callback, queue_size=2)


    def image_callback(self, ros_image):
        # Convert the image to opencv format
        rgb_image = np.ndarray(shape=(ros_image.height, ros_image.width, 3),
dtype=np.uint8, buffer=ros_image.data)
        result_image = np.copy(rgb_image)
        factor = 4
        rgb_image = cv2.resize(rgb_image, (int(ros_image.width / factor),
int(ros_image.height / factor)))

        try:
            if self.tracker is None:
                if self.enable_select:
                    roi = cv2.selectROI("image", cv2.cvtColor(result_image,
cv2.COLOR_RGB2BGR), False)
                    roi =  tuple(int(i / factor)for i in roi)
                    if roi:
                        self.tracker = cv2.TrackerKCF_create()
                        self.tracker.init(rgb_image, roi)
            else:
                status, box = self.tracker.update(rgb_image)
                if status:
                    p1 = int(box[0] * factor), int(box[1] * factor)
                    p2 = p1[0] + int(box[2] * factor), p1[1] + int(box[3] *
factor)
                    cv2.rectangle(result_image, p1, p2, (255, 255, 0), 2)
                    center_x, center_y = (p1[0] + p2[0]) / 2, (p1[1] + p2[1]) /
2
                    rospy.loginfo("center_x = {}, center_y =
{}".format(center_x,center_y))
                    center_x = center_x / result_image.shape[1]
                    if abs(center_x - 0.5) > 0.02: # If the difference is less
than a certain value, there is no need to move it.
                        self.xservo_pid.setpoint = 0.5 # Our goal is to make the
color block in the center of the picture, which is 1/2 of the pixel width of the
entire picture.
```

```
                        output = self.xservo_pid(center_x, dt=0.1)
                        self.target_servox = min(max(self.target_servox +
output, -160), 160)
                    else:
                        self.yservo_pid.reset()

                    center_y = center_y / result_image.shape[0]
                    if abs(center_y - 0.5) > 0.02:
                        self.yservo_pid.setpoint = 0.5
                        output = self.yservo_pid(center_y, dt=0.1)
                        self.target_servoy = min(max(self.target_servoy +
output, -160), 0)
                    else:
                        self.yservo_pid.reset()

                    joints_0 = [self.target_servox, 0, self.target_servoy, -
self.target_servoy, 0, -45]
                    rospy.loginfo("joints_0 = {}".format(joints_0))
                    self.mc.send_angles(joints_0, 50)

        except Exception as e:
            rospy.logerr(str(e))


        self.fps.update()
        self.fps.show_fps(result_image)
        result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
        cv2.imshow("image", result_image)

        key = cv2.waitKey(1)
        if key == ord('s'): # Press s to start selecting the tracking target
            self.tracker = None
            self.enable_select = True


if __name__ == '__main__':
    try:
        kcf_tracking = KCFTrackingNode()
        print("Please switch to the camera window, press s, select with the
mouse, and then press the space bar or Enter key to track the target.")
        rospy.spin()
    except Exception as e:
        rospy.logerr(str(e))
```