13. KCF object tracking

13.1. 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.

13.2. Startup

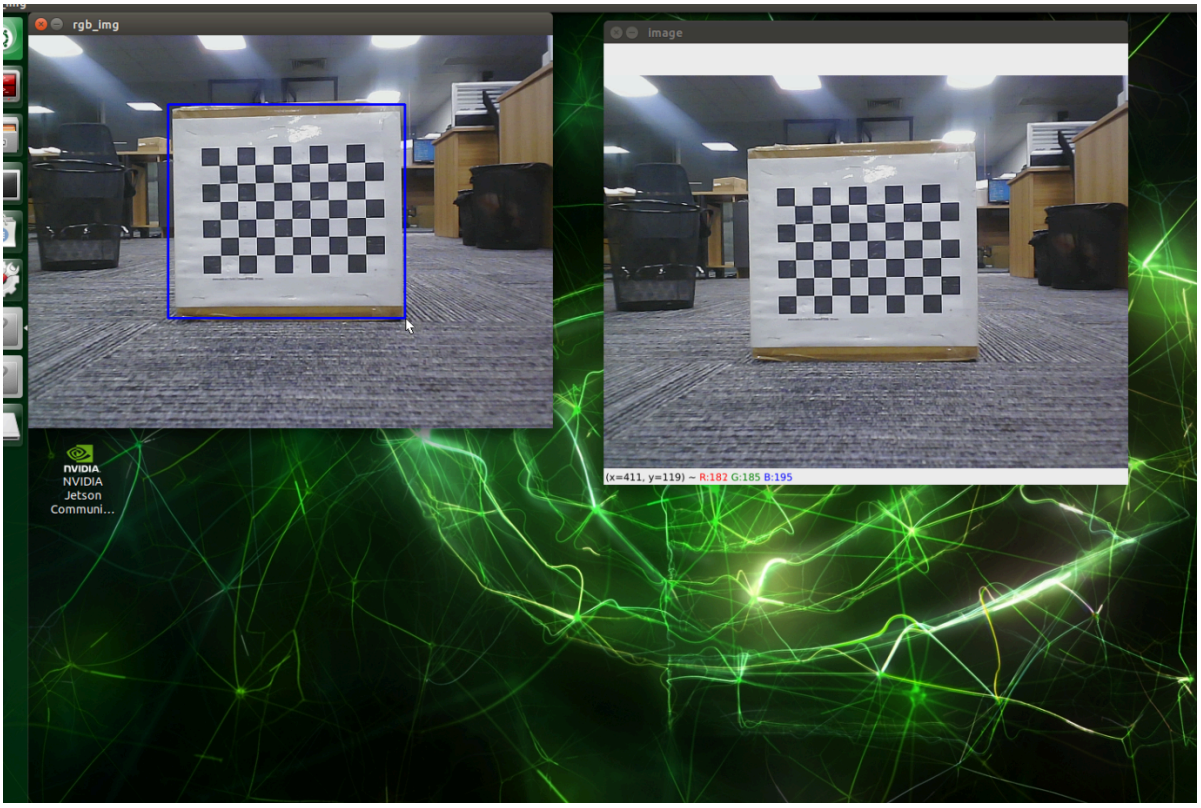
13.2.1. Preparation before starting the program

You need to prepare an object that is significantly different from the background as the object of KCF tracking.

13.2.2, Program Description

After the program is started, the camera captures the image. In the camera screen, press the s key (if you log in remotely via VNC, you need to press and hold it). The screen is still. Use the mouse to select the object to be tracked. Select the blue box. Press the space bar or the Enter key. The robot arm enters the tracking mode. The blue box turns into a yellow box. At this time, you can slowly move the object and the robot arm will track the object.

Note: The object cannot move too fast to prevent the robot arm from not keeping up.



13.2.3, Program startup

- Enter the following command to start the program

```
ros2 run jetcobot_advance kcf_tracking
```

13.2.4, Source code

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

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


class KCFTrackingNode(Node):
    def __init__(self):
        super().__init__('kcf_tracking')
        self.factor = 1.0
        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.identify = False
        self.tracker = None
        self.enable_select = False
        self.mc = MyCobot('/dev/ttyUSB0', 1000000)
        self.mc.send_angles([0, 0, -90, 90, 0, -45], 50)
        self.fps = 0
        print(cv2.__version__)

        self.cap = cv2.VideoCapture(0, cv2.CAP_V4L2)
        if not self.cap.isOpened():
            self.get_logger().error("Unable to open camera")
            return

        # Initialize roi_selected and roi
        self.roi_selected = False
        self.roi = None
        self.roi_start = None
        self.roi_end = None
        cv2.namedWindow("image")
        cv2.setMouseCallback("image", self.on_mouse)

        self.frame_count = 0
        self.start_time = time.time()
```

```
        self.timer = self.create_timer(0.03, self.process_frames_loop)
        #self.process_frames_loop()
    def on_mouse(self, event, x, y, flags, param):
        if self.tracker is not None:
            return

        if event == cv2.EVENT_LBUTTONDOWN:
            if self.roi_selected:
                self.roi_selected = False
                self.roi_start = None
                self.roi_end = None
                self.roi = None
            self.roi_start = (x, y)
        elif event == cv2.EVENT_MOUSEMOVE and self.roi_start is not None and not
self.roi_selected:
            self.roi_end = (x, y)
        elif event == cv2.EVENT_LBUTTONUP:
            if self.roi_start is not None:
                self.roi_end = (x, y)
                self.roi = (self.roi_start[0], self.roi_start[1],
                            self.roi_end[0]-self.roi_start[0],
                            self.roi_end[1]-self.roi_start[1])
                self.roi_selected = True

    def process_frame(self):
        ret, frame = self.cap.read()
        if not ret:
            self.get_logger().error("Unable to read frame")
            return

        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        result_image = np.copy(rgb_image)

        try:
            if self.tracker is None and self.roi_selected and self.identify:
                roi = tuple(int(i / self.factor) for i in self.roi)
                if roi:
                    #self.tracker = cv2.TrackerMIL_create()
                    self.tracker = cv2.TrackerKCF_create()
                    self.tracker.init(rgb_image, roi)
                    self.roi_selected = False
                    self.identify = False
                    self.roi_start = None
                    self.roi_end = None
            elif self.tracker is not None:
                status, box = self.tracker.update(rgb_image)
                if status:
                    p1 = int(box[0] * self.factor), int(box[1] * self.factor)
                    p2 = p1[0] + int(box[2] * self.factor), p1[1] + int(box[3] *
self.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
                    self.get_logger().info("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:
```

```
                        self.xservo_pid.setpoint = 0.5
                        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()

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

            except Exception as e:
                self.get_logger().error(str(e))

            if self.roi_start is not None and self.roi_end is not None:
                cv2.rectangle(result_image, self.roi_start, self.roi_end, (0, 0,
255), 2)

            self.frame_count += 1
            elapsed_time = time.time() - self.start_time
            if elapsed_time > 0.3:
                self.fps = self.frame_count / elapsed_time
                self.frame_count = 0
                self.start_time = time.time()
            cv2.putText(result_image, f"FPS: {self.fps:.1f}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

            result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
            cv2.imshow("image", result_image)


def main(args=None):
    rclpy.init(args=args)
    kcf_tracking = KCFTrackingNode()
    rclpy.spin(kcf_tracking)
    kcf_tracking.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```