人脸跟踪例程（rmtt_face_tracker.launch）

现在开始一步一步实现 TT 无人机对人脸的追踪。

根据之前的方法，在 ROS 中驱动 TT 无人机(drone_name:=rmtt_01)

打开新的终端，输入以下命令：

roslaunch rmtt_tracker rmtt_face_tracker.launch drone_name:=rmtt_01

正常启动后，我们可以看到如下图所示的返回信息，并会弹出一个 Frame 窗口

图 1：TT 无人系前摄画面

打开./rmtt_ros/rmtt_tracker/scripts/rmtt_face_tracker.py：

```python
#!/usr/bin/env python3

import rospy
import rospkg
import std_msgs.msg
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image
import cv2
from cv_bridge import CvBridge
import numpy as np

# resize the image to w*h
w = 360
h = 240

def callback(msg):

    img = bridge.imgmsg_to_cv2(msg)
    img = cv2.resize(img, (w, h))
    cv2.imshow('Frame', img)
    cv2.waitKey(1)

if __name__ == '__main__':

    rp = rospkg.RosPack()
    path = rp.get_path("rmtt_tracker")
    bridge = CvBridge()
    rospy.init_node('face_tracker', anonymous=True)
    sub = rospy.Subscriber("image_raw", Image, callback)
    rospy.spin()
```

可以看到该节点名为 face_tracker，订阅了/image_raw 话题，将图像比例固定为 360*240 然后利用 OpenCV 中的函数进行输出。由于我们目前只需要使用视频信息，为节省 TT 无人机电量也可以将上节课使用的 usb_cam package 拷贝至/tt_ws/src 中利用 catkin_make 进行编译，然后调用笔记本摄像头进行图像处理的算法实现，最后再和 TT 无人机进行联调。开启摄像头后改变名字空间即可利用 face_tracker 节点实现视频流的读取和输出：

roslaunch rmtt_tracker rmtt_face_tracker.launch drone_name:=usb_cam

人脸识别的模型我们已经放到了/tt_ws/src/rmtt_ros/rmtt_tracker/config 中我们在脚本中加载该模型：

```
46 if __name__ == '__main__':
47
48     rp = rospkg.RosPack()
49     path = rp.get_path("rmtt_tracker")
50     faceCascade = cv2.CascadeClassifier(path + '/config/haarcascade_frontalface_default.xml')
51     bridge = CvBridge()
52     rospy.init_node('face_tracker', anonymous=True)
53     sub = rospy.Subscriber("image_raw", Image, callback)
54     rospy.spin()
```

定义人脸检测函数 findFace，首先把彩色图像转换成灰度图像，然后调用人脸识别模型，再将检测到人脸的边界框依次绘制到图像上。每个边界框的中心和面积都进行存储，如果检测到人脸，选取最大面积边界框作为函数输出，这将是我们最终要跟踪的人脸。将下面一段代码增加到脚本中，并看看每一行实现了什么功能，在 callback 函数中调用人脸检测函数：

```
12 # resize the image to w*h
13 w = 360
14 h = 240
15
16 def callback(msg):
17
18     img = bridge.imgmsg_to_cv2(msg)
19     img = cv2.resize(img, (w, h))
20
21     img, info = findFace(img)
22
23     cv2.imshow('Frame', img)
24     cv2.waitKey(1)
25
26 def findFace(img):
27     imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
28     faces = faceCascade.detectMultiScale(imgGray, 1.1, 6)
29
30     # get the face of the largest bounding box
31     myFaceListC = []
32     myFaceListArea = []
33     for (x, y, w, h) in faces:
34         cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
35         cx = x + w // 2
36         cy = y + h // 2
37         area = w * h
38         myFaceListArea.append(area)
39         myFaceListC.append([cx, cy])
40     if len(myFaceListArea) != 0:
41         i = myFaceListArea.index(max(myFaceListArea))
42         return img, [myFaceListC[i], myFaceListArea[i]]
43     else:
44         return img, [[0, 0], 0]
```

保存并运行一下这个节点：roslaunch rmtt_tracker rmtt_face_tracker.launch drone_name:=usb_cam 看看是否能检测出人脸。

定义人脸跟踪函数 trackFace，利用最大检测框距离图像中心点的左右偏移、上下偏移、和检测框大小变化乘以某一系数（比例控制）分别输出偏航角速度、上下线速度、前后线速度的输出。因此我们有偏航、上下、前后三个通道的控制。我们这里给出偏航和前后两个通道的实现。首先在全局变量添加两个通道的控制参数和控制指令输出标志符（用来判断有无控制指令输出）：

无人系统技术研究院

```
# resize the image to w*h
w = 360
h = 240
pid_w = [0.5, 0, 0] # pid parameters of yaw channel
pid_f = [0.8, 0, 0] # pid parameters of forward channel
zero_twist_published = False
```

我们希望最终向/cmd_vel 发布控制指令，因此定义一个发布消息的对象，其数据格式是 Twist：

```
if __name__ == '__main__':

    rp = rospkg.RosPack()
    path = rp.get_path("rmtt_tracker")
    faceCascade = cv2.CascadeClassifier(path + '/config/haarcascade_frontalface_default.xml')
    bridge = CvBridge()
    rospy.init_node('face_tracker', anonymous=True)
    sub = rospy.Subscriber("image_raw", Image, callback)
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.spin()
```

增加 trackFace 函数，用来通过人脸检测框的位置和大小计算控制指令输出：

```
def trackFace(info, w, h, pid_w, pid_f):
    ## PID
    # yaw channel
    error_w = info[0][0] - w // 2
    speed_w = pid_w[0] * error_w
    speed_w = int(np.clip(speed_w, -100, 100)) / 100.0

    # forward channel
    error_f = np.sqrt(info[1]) - 70
    speed_f = pid_f[0] * error_f
    speed_f = int(np.clip(speed_f, -100, 100)) / 100.0

    if info[0][0] != 0:
        yaw_speed = speed_w
    else:
        yaw_speed = 0

    if info[1] != 0:
        forward_speed = speed_f
    else:
        forward_speed = 0

    return yaw_speed, forward_speed
```

在 callback 函数中调用 trackFace 获取偏航和前后两通道速度。定义 speed 这一 Twist 消息格式的对象，并给对应的线速度和角速度赋值。rospy.loginfo()用于在当前终端输出我们给定的信息，pub_publish(speed)用来发布控制指令：

```
def callback(msg):
    global zero_twist_published
    img = bridge.imgmsg_to_cv2(msg)
    img = cv2.resize(img, (w, h))

    img, info = findFace(img)
    yaw_speed, forward_speed = trackFace(info, w, h, pid_w, pid_f)
    rc = "left: " + "0" + " forward: " + str(forward_speed) + " up: " + str(0) + "  yaw: " +
  str(yaw_speed)
    speed = Twist()

    if yaw_speed != 0 or forward_speed != 0:
        speed.linear.x = -forward_speed
        speed.linear.y = 0.0
        speed.linear.z = 0.0 # add up channel here
        speed.angular.x = 0.0
        speed.angular.y = 0.0
        speed.angular.z = -yaw_speed
        rospy.loginfo(rc)
        pub.publish(speed)
        zero_twist_published = False
    else:
        if not zero_twist_published:
            pub.publish(speed)
            zero_twist_published = True
            rospy.loginfo("no face detected")

    cv2.imshow('Frame', img)
    cv2.waitKey(1)
```

无人系统技术研究院

保存并运行一下这个节点：roslaunch rmtt_tracker rmtt_face_tracker.launch drone_name:=usb_cam 看看检测到人脸之后在终端中是否有控制指令输出。前后左右移动人脸观察控制指令变化。

```
[1616465146.632140]: left: 0 forward: 0.19 up: -0.55  yaw: 0.19
[1616465146.715433]: left: 0 forward: 0.16 up: -0.56  yaw: 0.19
[1616465146.827269]: no face detected
[1616465211.523107]: left: 0 forward: -0.13 up: -0.74  yaw: 0.26
```

注意：由于我们还没有添加左右和上下控制通道，因此左右和上下控制指令实际应该为 0。左右方向的运动本次实验不需要进行添加。

现在我们连接 TT 无人机测试一下。

根据之前的方法，在 ROS 中驱动 TT 无人机(drone_name:=rmtt_01)

根据之前的方法，在 ROS 中启动键盘遥控(drone_name:=rmtt_01)

打开新的终端，输入以下命令：

roslaunch rmtt_tracker rmtt_face_tracker.launch drone_name:=rmtt_01

此时我们通过键盘控制将 TT 无人机的前置摄像头对准自己，在 Frame 窗口中可以看到人脸识别结果。

这时我们可以通过一下命令查看话题中就已经有了内容（geometry_msgs/Twist 消息）：

rostopic echo /rmtt_01/cmd_vel

注意：此时无人机不会起飞，也不会执行/rmtt_01/cmd_vel 中的控制命令

```
angular:
  x: 0.0
  y: 0.0
  z: 0.01
---
linear:
  x: -0.3
  y: 0.0
  z: 0.48
angular:
  x: 0.0
  y: 0.0
  z: 0.01
---
linear:
  x: -0.29
  y: 0.0
  z: 0.48
angular:
  x: 0.0
  y: 0.0
  z: 0.01
```

小贴士：我们可以使用 rqt_graph 可视化工具查看当前示例中运行的节点与话题的关系：

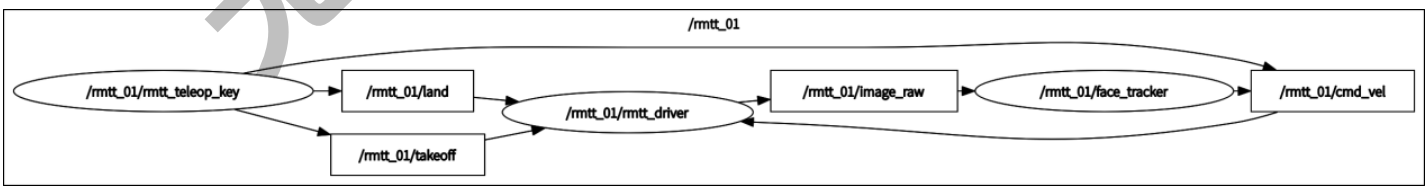

图 2：无人机人脸追踪节点话题关系图

完成 TT 无人机跟踪人脸示例的最后一步：

将启动键盘遥控的终端保持激活状态（切换到该终端），控制 TT 无人机起飞；

想办法出现在 TT 无人机前置摄像头的画面中；

TT 无人机将开始跟踪画面中出现的人脸，并保持一定距离。

请同学们自行添加上下控制通道！

无人系统技术研究院

QR code 二维码追踪

我们将在人脸追踪的基础上修改代码实现 QR code 二维码追踪，在~/tt_ws/src/rmtt_ros/rmtt_tracker/scripts 中有完整的 QR code 二维码检测 Python 脚本程序，（/tt_ws/src/rmtt_ros/rmtt_tracker/scripts/qrcode.py），请安装依赖库，并通过终端运行（例：python qrcode.py）。

```python
# coding=utf-9
# 环境准备: sudo apt-get install libzbar-dev
#           pip3 install pyzbar

import cv2
import pyzbar.pyzbar as pyzbar

def decodeDisplay(image):
    barcodes = pyzbar.decode(image)
    for barcode in barcodes:
        # 提取二维码的边界框的位置
        # 画出图像中条形码的边界框
        (x, y, w, h) = barcode.rect
        cv2.rectangle(image, (x, y), (x + w, y + h), (225, 225, 225), 2)

        # 提取二维码数据为字节对象，所以如果我们想在输出图像上
        # 画出来，就需要先将它转换成字符串
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type

        # 绘出图像上条形码的数据和条形码类型
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 5, (225, 225, 225), 2)

        # 向终端打印条形码数据和条形码类型
        print("[INFO] x: {} y:{} w:{} h:{} Found {} barcode: {}".format(x,y,w,h,barcodeType, barcodeData))
    return image


def detect():
    camera = cv2.VideoCapture(0)

    while True:
        # 读取当前帧
        ret, frame = camera.read()
        # 转为灰度图像
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        im = decodeDisplay(gray)

        cv2.waitKey(5)
        cv2.imshow("QRCode Detector", im)
        # 如果按键q则跳出本次循环
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    camera.release()
    cv2.destroyAllWindows()

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

注意：python 编程需要按照 4 个空格（或使用 tab）对齐同一程序块中的代码；
请按照代码中的提示安装指定的库；
看看能不能识别本手册最后一页的二维码？

小贴士：机器人是一个系统工程，通常一个机器人运行操作时要开启多个 node。ROS 提供了 launch 文件来避免每次启动某个 package 时依次开启其中每一个 node 的繁琐操作。launch 文件储存在 package 的/launch 目录下，每一个 launch 文件指定了对于这个 package 的一种启动方式，包括开启哪些节点、读取哪些配置文件、预置哪些参数等。roslaunch xxx yyy.launch 命令会自动启动 roscore，如果已启动了 roscore，roslaunch 就不会再启动 roscore。

接下来看看 QR code 二维码追踪和人脸追踪的实现上有什么异同。

无人系统技术研究院

```
#!/usr/bin/python3
# coding=utf-8
# 环境准备：sudo apt-get install libzbar-dev
#           pip install pyzbar
# 输入话题：image_raw
# 输出话题：cmd_vel

import rospy
import rospkg
import sensor_msgs.msg
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image
import cv2
from cv_bridge import CvBridge
import numpy as np
import pyzbar.pyzbar as pyzbar
```

安装和导入需要的函数库。

```
if __name__ == '__main__':

    bridge = CvBridge()
    rospy.init_node('qrcode_tracker')
    rospy.Subscriber("image_raw", Image, callback)
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.spin()
```

从主函数看起，与人脸追踪类似，CvBridge()类是用来将 ROS 的图像格式转换为 OpenCV 图像处理库可以读取并处理的图像格式。这里给 CvBridge()类声明了一个对象 bridge，bridge 将可以使用 CvBridge()类中所定义的变量和函数。接下来定义了这个节点的名称，这个节点所订阅以及发布的话题。rospy.spin()的功能是一旦从订阅的话题接收到图片就调用 callback 回调函数，并且程序在此处循环，不再向下执行。

```
fw = 360
fh = 240
pid_w = [0.8, 0, 0] # pid parameters of yaw channel
pid_h = [0.8, 0, 0] # pid parameters of up channel
pid_f = [0.8, 0, 0] # pid parameters of forward channel
pid_l = [2.0, 0, 0] # pid parameters of left channel
```

我们定义了如上几个全局变量，fw 和 fh 是将摄像头看到的图像缩放到对应的宽和高。在控制参数部分，和人脸跟踪相比，新增了 pid_l 通道。这个通道用于向无人机输出左右平移（滚转）控制量。**由于可以精确输出二维码的四角的像素坐标，可借此生成左右平移控制量，使得二维码原地转动时无人机可以跟随转动。**

```
def callback(msg):
    global zero_twist_published
    img = bridge.imgmsg_to_cv2(msg)
    img = cv2.resize(img, (fw, fh))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    zero_twist_published = False
```

现在开始定义 callback 回调函数的内容，注意和人脸追踪的不同，比如变量名等。

无人系统技术研究院

```
def callback(msg):
    global zero_twist_published
    img = bridge.imgmsg_to_cv2(msg)
    img = cv2.resize(img, (fw, fh))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    zero_twist_published = False

    # 从灰度图像中检测二维码
    barcodes = pyzbar.decode(gray)
    if barcodes :
        # 只提取识别到的第一个二维码
        barcode = barcodes[0]
        # 提取并绘制二维码的边界框的位置
        points =[]
        for point in barcode.polygon:
            points.append([point[0], point[1]])
        points = np.array(points,dtype=np.int32).reshape(-1,1, 2)
        cv2.polylines(img, [points], isClosed=True, color=(0,0,255),thickness=2)
        # 获取边界框对角线长度比值
        x1 = points[0,0,0]
        y1 = points[0,0,1]
        x2 = points[1,0,0]
        y2 = points[1,0,1]
        x3 = points[2,0,0]
        y3 = points[2,0,1]
        x4 = points[3,0,0]
        y4 = points[3,0,1]
        if x2 < x4:
            hl = np.sqrt(np.square((x1 - x2)) + np.square((y1 - y2)))
            hr = np.sqrt(np.square((x4 - x3)) + np.square((y4 - y3)))
        else:
            hl = np.sqrt(np.square((x1 - x4)) + np.square((y1 - y4)))
            hr = np.sqrt(np.square((x2 - x3)) + np.square((y2 - y3)))
        ratio = hl / hr * 100
```

紧接着，在 callback 回调函数中，利用 pyzbar 库，从转换后的灰度图像中检测二维码。由于可能检测到多个二维码，只利用检测到的第一个二维码进行追踪。代码中的(x1, y1)、(x2, y2)、(x3, y3)、(x4, y4)是二维码四角的像素坐标。**注意 QR code 二维码随着观测角度的变化，四个角的保存顺序可能发生变化，可以通过简单的判断，比如 x2 与 x4 的大小判定此时的保存顺序，用以计算生成左右平移控制量。**

```
        # 画出图像中二维码的外接矩形边界框
        (x, y, w, h) = barcode.rect
        cv2.rectangle(img, (x, y), (x + w, y + h), (225, 225, 225), 2)
        # 提取二维码数据为字节对象，所以如果我们想在输出图像上
        # 画出来，就需要先将它转换成字符串
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        # 绘出图像上条形码的数据和条形码类型
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(img, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
                    .5, (225, 225, 225), 2)
        # 向终端打印条形码数据和条形码类型
        print("[INFO] x: {} y:{} w:{} h:{} ratio:{} Found {} barcode: {}".format(x,y,w,h,ratio,
barcodeType, barcodeData))
        cx = x + w // 2
        cy = y + h // 2
        area = w * h
    else:
        cx = 0
        cy = 0
        area = 0
        ratio = 0.0001
    cv2.imshow("qrcode_detect_result", img)
    cv2.waitKey(1)
```

继续在回调函数中进行处理，代码中有详尽的中文注释，最终我们希望能够获取二维码的中心点位置、二维码像素面积、左右平移控制量。cv2.imshow 函数用来将图像以小窗口的形式进行显示，包括所有在图像上绘制的边框、显示的字符等。

无人系统技术研究院

```
83
84      # 获得各个通道的控制量并显示在终端
85      yaw_speed, up_speed, forward_speed, left_speed = pidtrack(cx, cy, area, ratio, fw, fh, pid_w,
    pid_h, pid_f, pid_l)
86      rc = "forward: " + str(forward_speed) + " up: " + str(up_speed) + "  yaw: " + str(yaw_speed) + "
    left: " + str(left_speed)
87
88      speed = Twist()
89      if yaw_speed != 0 or up_speed != 0 or forward_speed != 0 or left_speed != 0:
90          speed.linear.x = -forward_speed
91          speed.linear.y = -left_speed
92          speed.linear.z = -up_speed
93          speed.angular.x = 0.0
94          speed.angular.y = 0.0
95          speed.angular.z = -yaw_speed
96          rospy.loginfo(rc)
97          pub.publish(speed)
98          zero_twist_published = False
99      else:
100         if not zero_twist_published:
101             pub.publish(speed)
102             zero_twist_published = True
103             rospy.loginfo("no object detected")
```

将相应的变量输入到 pidtrack 函数，返回控制量，向控制指令相关话题发布四个通道的控制量，分别是：偏航、上下、前后和左右。rc 是一个长字符串用于利用 rospy.loginfo(rc)在终端显示。

```
105 def pidtrack(cx, cy, area, ratio, fw, fh, pid_w, pid_h, pid_f, pid_l):
106     ## PID
107     # yaw channel
108     error_w = cx - fw // 2
109     speed_w = pid_w[0] * error_w
110     speed_w = int(np.clip(speed_w, -100, 100)) / 100.0
111
112     # up channel
113     error_h = cy - fh // 2
114     speed_h = pid_h[0] * error_h
115     speed_h = int(np.clip(speed_h, -100, 100)) / 100.0
116
117     # forward channel
118     error_f = np.sqrt(area) - 70
119     speed_f = pid_f[0] * error_f
120     speed_f = int(np.clip(speed_f, -100, 100)) / 100.0
121
122     # left channel
123     if (ratio - 100) >= 0:
124         error_l = ratio - 100
125     else:
126         error_l = - 10000.0 / ratio + 100
127     speed_l = pid_l[0] * error_l
128     speed_l = int(np.clip(speed_l, -100, 100)) / 100.0
129
130     if cx != 0:
131         yaw_speed = speed_w
132     else:
133         yaw_speed = 0
134     if cy != 0:
135         up_speed = speed_h
136     else:
137         up_speed = 0
138     if area != 0:
139         forward_speed = speed_f
140     else:
141         forward_speed = 0
142     if ratio != 0.0001:
143         left_speed = speed_l
144     else:
145         left_speed = 0
146
147     return yaw_speed, up_speed, forward_speed, left_speed
```

pidtrack 函数与人脸追踪的 facetrack 类似，**注意最后增加了一个控制通道**。

即将大功告成，如果代码正确，可以通过 rosrun 命令来运行 QR code 二维码追踪节点。我们还可以在当前 package 下的 launch 文件夹中创建.launch 文件，这样我们便能够用 roslanuch 命令运行该节点。

无人系统技术研究院

```xml
<?xml version="1.0"?>

<launch>
  <arg name = "ns" default="$(optenv RMTT_NAMESPACE /)" />
  <group ns="$(arg ns)">
    <node pkg="rmtt_tracker" type="rmtt_qrcode_tracker.py" name="qrcode_tracker"
output="screen" respawn="true">
    </node>
  </group>
</launch>
```

通过编写 lanuch 文件，可以一次性运行多个节点。其中 arg 和 group 用来定义名字空间所使用的变量名（此处为 ns），node 中是需要运行的节点。在 node 中 pkg 是功能包名称、type 是节点程序、name 是该节点运行之后在系统中的名称。请同学们修改以上的 launch 文件，在 TT 无人机驱动正常启动后，一次性运行键盘控制和二维码追踪节点。