

一. 教学目标

- (1) 熟悉机器人操作系统（ROS）的基本概念；
- (2) 掌握 ROS 的操作流程及常用工具箱的使用；
- (3) 编写 ROS 节点实现旋翼无人机人脸追踪。

二. 教学重难点

教学重点：ROS 的基本概念、操作流程及工具箱的使用；

教学难点：编写 ROS 节点，实现无人机人脸追踪。

三. 教学流程



四. 教学设计

ROS2GO 准备工作

安装 `rmmtt_ros` 系列功能包：

下载 `rmmtt_ros.zip`: <https://pan.baidu.com/s/12uFC6IxU2eceLSddreZBuA> 提取码: pikd;

解压 `rmmtt_ros.zip` 压缩包到 `~/tt_ws/src/` (自行创建文件夹);

在该工作空间 (`~/tt_ws/`) 下打开终端;

编译工作空间: `catkin_make`;

运行功能包中的节点前, 请在该终端及新终端中输入: `source ./devel/setup.bash`;

或在 `home` 中打开终端, 输入: `gedit ~/.bashrc`, 将以上命令粘贴入文档并保存;

关闭所有终端, 再次打开新终端可直接运行功能包中的节点。

TT 无人机 EXT 模块安装

安装完成后的效果：



图 1：Robomaster TT 旋翼无人机

TT 无人机电池充电

充电逻辑说明：

每节电池依次充电，并不是同时充电；

绿色闪烁为充电中，绿色常亮为充满，黄色常亮为等待充电。

注意事项：

随时保持电池处于充电状态，方便使用。

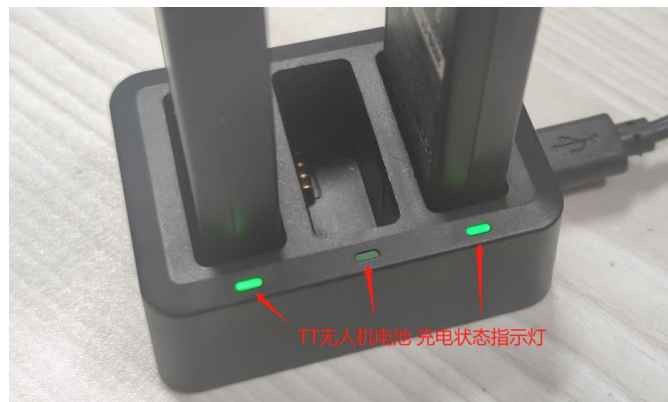


图 2：充电指示

激活 TT 无人机

通过 Tello 手机客户端对 TT 无人机进行激活：

开机和关机均通过短按 TT 无人机的电源键，稍等片刻；

将手机通过 wifi 连接至 TT 无人机，将 EXT 模块调至 AP 模式；

打开 Tello 手机客户端，全新的 TT 无人机需要首先连接客户端进行激活，才能继续后续任务；

看看 Tello 手机客户端都提供了哪些功能，飞一下吧；

每块电池满电的飞行时间在十分钟左右，记得时刻给不使用的电池充电；

之后我们将通过 ROS 对 TT 无人机进行控制和编程。

TT 网络模式配置

TT 无人机具有 STA（路由）模式与 AP（热点）模式。STA 模式可以在通过一台电脑连接和控制多台无人机，本节课我们只使用 AP 模式。

TT 进入 AP 模式：

将 EXT 模块上的“模式开关”切换到“AP”

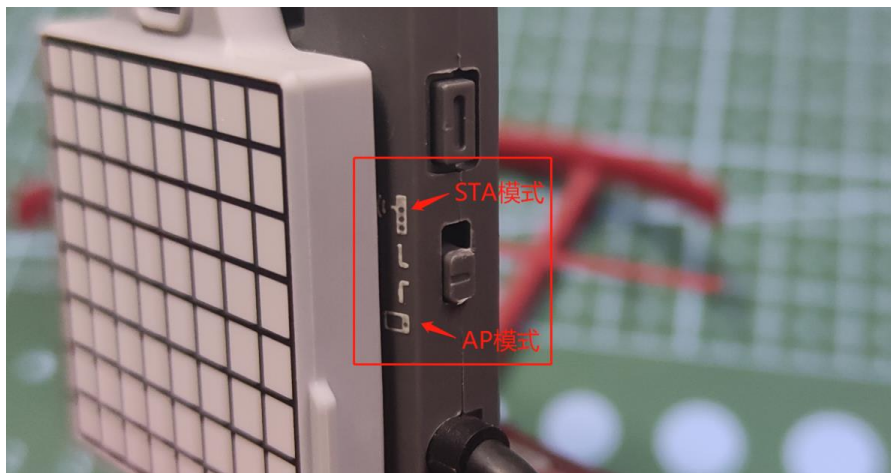


图 3：切换路由和热点模式

短按电源按钮，启动 TT 无人机，无人机的 wifi 名称于 LED 模块背面：RMTT-xxx。
稍等片刻(TT 启动后)，使用 PC 连接到 RMTT-xxx 网络。



图 4：连接到 RMTT-xxx 网络

在 ROS 中驱动 TT 无人机

硬件驱动启动文件：rmtt_bringup.launch。

参数说明：

drone_ip: TT 无人机的 IP 地址；

local_ip: PC 的 IP 地址；

ns: 名字空间，用于区分多台 TT 无人机（在后面的应用中，对 ns 会有更感性的认识，来理解 ns 的作用）

local_ip 通过在新的终端执行 ifconfig 来获取。

在新的终端执行以下命令：

```
roslaunch rmtt_driver rmtt_bringup.launch
```

或者：

```
roslaunch rmtt_driver rmtt_bringup.launch ns:=rmtt_01
```

注意：由于目前我们只连接一台无人机，无需设定 `drone_ip` 和 `local_ip`。
执行以上命令后，我们可以看到如下图所示的返回信息：

```
setting /run_id to eca810c8-8af6-11eb-910f-e5c311262486
process[rosout-1]: started with pid [10759]
started core service [/rosout]
process[rmtt_driver-2]: started with pid [10766]
[INFO] [1616407871.568722]: RoboMaster TelloTalent Initializing...
[INFO] [1616407871.901708]: drone sn: 
[INFO] [1616407871.912234]: start camera video stream!
[INFO] [1616407871.943901]: battery level: 100
█
```

在返回信息中，我们可以看到 TT 无人机的序列号、视频流以及电量信息。看到以上信息时，表示 TT 无人机已经被正确的启动，并在 ROS 中驱动成功。

TT 无人机 ROS 驱动的话题简要说明

首先，根据上节的方法，在 ROS 中驱动 TT 无人机。

使用 `rostopic list` 命令查看 ROS 驱动发布的话题列表（看看有无 `ns:=rmtt_01` 话题名称的区别）：

```
tianbot@ros2go:~$ rostopic list
/altitude
/battery
/camera_info
/cmd_vel
/flip
/image_raw
/image_raw/compressed
/imu_data
/land
/led
/mission_pad_id
/mled
/pose
/rosout
/rosout_agg
/takeoff
/tof_btm
/tof_ext
tianbot@ros2go:~$ █
```

使用 `rostopic info xxx` 查看 xxx 话题的基本信息：

```
/tof_ext
tianbot@ros2go:~$ rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
* /rmtt_driver (http://ros2go:44015/)
```

图中以 `/cmd_vel` 话题为例，我们可以看到发布者：None，订阅者：`/rmtt_driver`。

如果在驱动 TT 无人机时使用了 `ns` 参数，假设 `ns:=rmtt_01`，那么我们将看到如下的话题列表：

```

tianbot@ros2go:~$ rostopic list
/rmtt_01/altitude
/rmtt_01/battery
/rmtt_01/camera_info
/rmtt_01/cmd_vel
/rmtt_01/flip
/rmtt_01/image_raw
/rmtt_01/image_raw/compressed
/rmtt_01/imu_data
/rmtt_01/land
/rmtt_01/led
/rmtt_01/mission_pad_id
/rmtt_01/mled
/rmtt_01/pose
/rmtt_01/takeoff
/rmtt_01/tof_btm
/rmtt_01/tof_ext
/rosout
/rosout_agg
tianbot@ros2go:~$ 

```

接下来我们一一介绍上图中所示的话题的具体功能：

/rmtt_01/altitude	#TT 无人机内置气压计海拔数据
/rmtt_01/battery	#TT 无人机电池电量百分比
/rmtt_01/camera_info	#TT 无人机前置摄像头参数信息
/rmtt_01/cmd_vel	#控制 TT 无人机运动
/rmtt_01/flip	#控制 TT 无人机翻转
/rmtt_01/image_raw	#TT 无人机前置摄像头画面
/rmtt_01/image_raw/compressed	#TT 无人机前置摄像头画面(压缩)
/rmtt_01/imu_data	#TT 无人机内置 IMU 数据
/rmtt_01/land	#控制 TT 无人机降落
/rmtt_01/led	#EXT 模块顶端全彩 LED 的颜色
/rmtt_01/mission_pad_id	#DJI 定位毯坐标 id
/rmtt_01/mled	#控制 EXT 模块全彩 LED 点阵显示字符串
/rmtt_01/pose	#TT 无人机相对于 DJI 定位毯的姿态
/rmtt_01/takeoff	#控制 TT 无人机起飞
/rmtt_01/tof_btm	#TT 无人机本体底部的 tof 传感器距离值（范围：0.3-1.5）
/rmtt_01/tof_ext	#EXT 模块前端 tof 传感器距离值（范围：0.01-1.5）

在新的终端部利用 `rostopic echo xxx` 命令查看 xxx 话题的数据，和发布频率（回忆上节课内容）。用手移动无人机查看诸如 `/rmtt_01/imu_data`、`/rmtt_01/tof_btm`、`/rmtt_01/tof_ext` 等话题的数据及其意义。

小贴士：在使用 `rostopic list` 命令的时候加入 `-v` 指令，可以快速地区分哪些话题是被发布的与订阅的。


```
Published topics:
* /rosout_agg [rosgraph_msgs/Log] 1 publisher
* /rosout [rosgraph_msgs/Log] 1 publisher
* /rmtt_01/imu_data [sensor_msgs/Imu] 1 publisher
* /rmtt_01/pose [geometry_msgs/PoseStamped] 1 publisher
* /rmtt_01/mission_pad_id [std_msgs/UInt8] 1 publisher
* /rmtt_01/image_raw/compressed [sensor_msgs/CompressedImage] 1 publisher
* /rmtt_01/image_raw [sensor_msgs/Image] 1 publisher
* /rmtt_01/camera_info [sensor_msgs/CameraInfo] 1 publisher
* /rmtt_01/tof_btm [sensor_msgs/Range] 1 publisher
* /rmtt_01/tof_ext [sensor_msgs/Range] 1 publisher
* /rmtt_01/altitude [std_msgs/Float32] 1 publisher
* /rmtt_01/battery [std_msgs/Float32] 1 publisher

Subscribed topics:
* /rosout [rosgraph_msgs/Log] 1 subscriber
* /rmtt_01/cmd_vel [geometry_msgs/Twist] 1 subscriber
* /rmtt_01/takeoff [std_msgs/Empty] 1 subscriber
* /rmtt_01/land [std_msgs/Empty] 1 subscriber
* /rmtt_01/flip [std_msgs/Empty] 1 subscriber
* /rmtt_01/led [std_msgs/ColorRGBA] 1 subscriber
* /rmtt_01/mled [std_msgs/String] 1 subscriber
```

查看/rmtt_01/image_raw 话题的视频画面：

打开终端，输入以下命令： rqt

（首次我们打开 rqt 工具，会看到一个空白的窗口，不要慌，这是正常的）

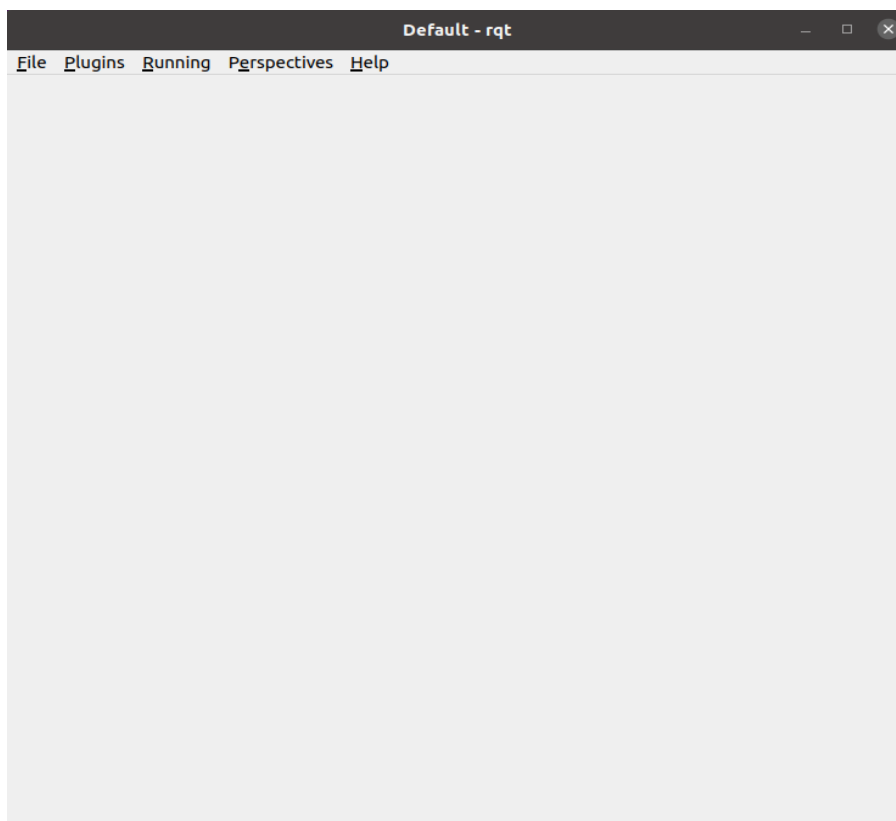


图 5：rqt 工具界面

点击 Plugins - Visualization - Image View，添加视频画面查看组件

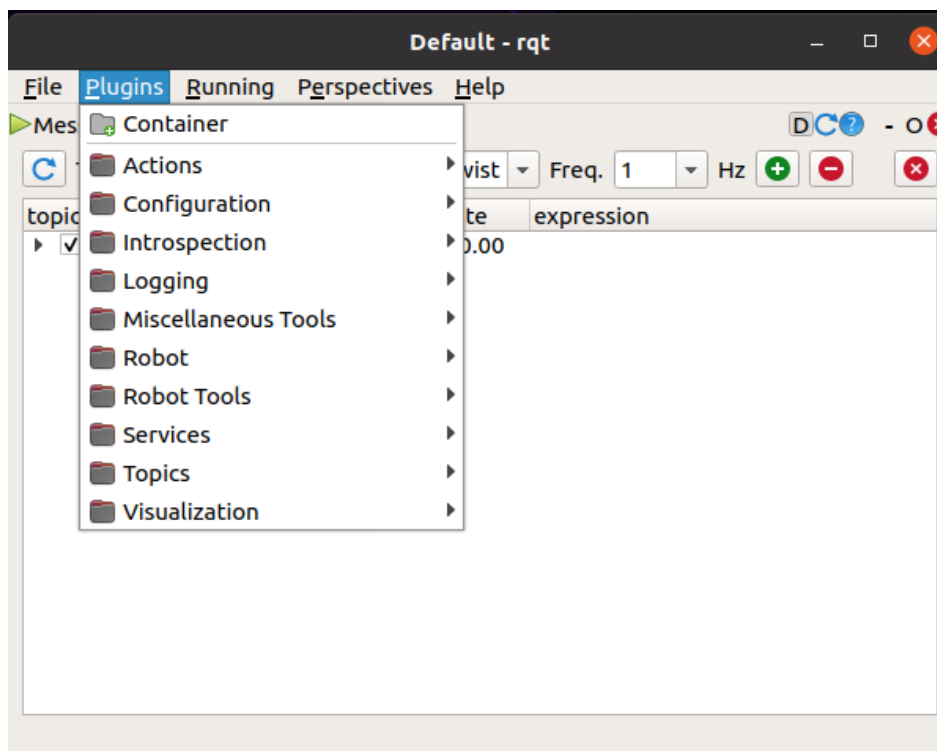


图 6：添加组件菜单栏

在打开的 Image View 插件中，我们选择/rmtt_01/image_raw 话题，就可以看到画面了。还记不记得在 rviz 里怎么看到画面？尝试一下。



图 7：在 rqt 中显示 TT 无人机摄像头画面

LED 字符显示

默认状态下，LED 显示屏的最下方一行指示前方障碍物的距离，看看测距传感器在哪？

在一个新的终端运行：

`rostopic pub /rmtt_01/mled std_msgs/String 'nwpu'。`

也可以利用 `rqt` 工具选择对应话题，Plugins - Topics - Message Publisher，选择 `/rmtt_01/mled`。按下拉箭头，在 `expression` 处输入要显示的字符（比如 “nwpu”）然后打勾发布：

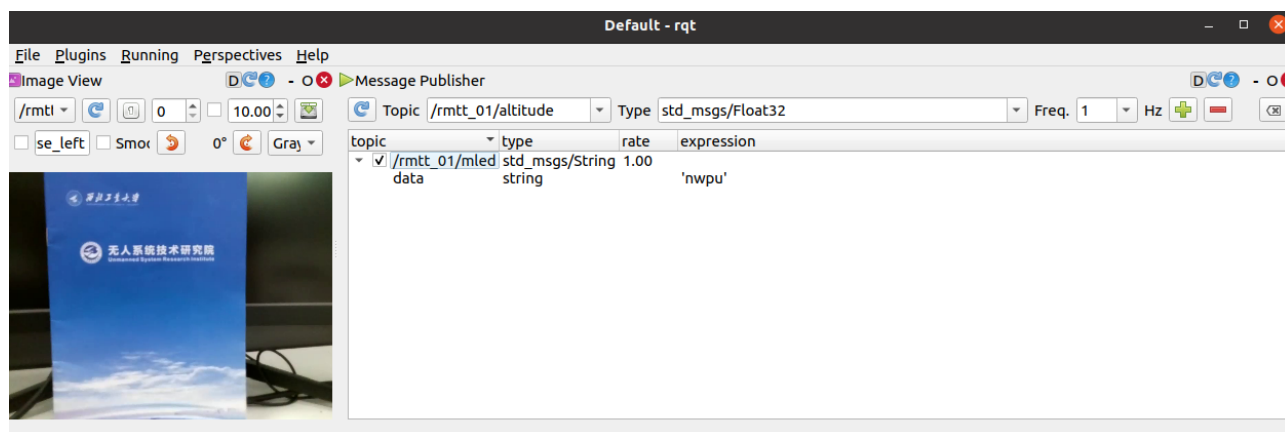


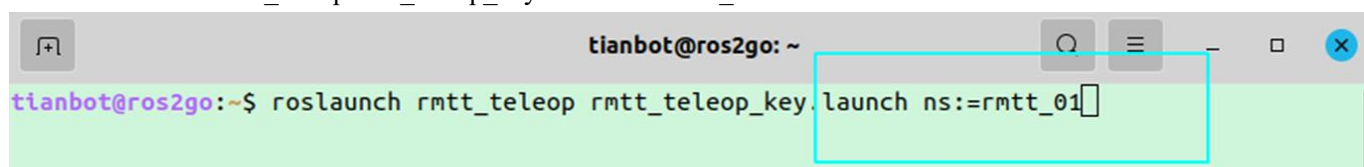
图 8：在 rqt 中发布话题消息

在 ROS 中使用键盘遥控 TT 无人机

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

打开一个新的终端，输入如下指令：

`roslaunch rmtt_teleop rmtt_teleop_key.launch ns:=rmtt_01`



在本例中，我们依旧以 `ns:=rmtt_01` 为例，在启动 `teleop` 遥控程序时，也加入 `ns` 参数，如上所示：

```
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
mode two
LEFT HAND:
    w
  a      d
    x
RIGHT HAND:
  u  i  o
  j  k  l
  m  ,  .
- : to takeoff
= : to land
(need to hold SHIFT)
>/< : increase/decrease linear speed by 10%
X/Z : increase/decrease angular speed by 10%
anything else : stop
CTRL-C to quit

currently:      speed 0.2      turn 0.2
```

遥控程序启动后，我们在终端中可以看到如上返回信息。

保持此终端激活状态，就可以按照信息提示的方法遥控 TT 无人机：

起飞 键盘 -

降落 键盘 =

小贴士：什么是保持终端激活状态？当执行命令之后，不要将光标（鼠标）移动到其他窗口，此时光标所在的终端就是激活状态，在这个状态中的终端可以采集键盘的输入，用来控制机器人。

本例中的话题关系图：我们可以看出，在启动键盘遥控节点后，键盘遥控与 TT 无人机驱动通过/rmtt_01/cmd_vel 联系了起来，这也是键盘遥控可以控制 TT 无人机运动的机制。

```
tianbot@ros2go:~$ rostopic info /rmtt_01/cmd_vel
Type: geometry_msgs/Twist

Publishers:
* /rmtt_01/rmtt_teleop_key (http://ros2go:45851/)

Subscribers:
* /rmtt_01/rmtt_driver (http://ros2go:36255/)

tianbot@ros2go:~$
```

以可视化的方式查看话题关系图：（在新的终端中输入：rqt_graph）。

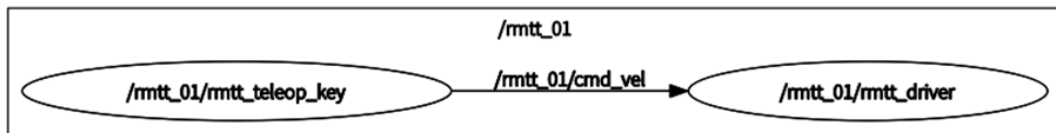


图 9：键盘遥控 TT 无人机节点话题关系图

异常处理（失控、无法降落）

关闭 bringup.launch 所在终端、30s 后 TT 无人机会自动降落；

或者抓住 LED 模块，将 TT 无人机翻转。

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(ns:=rmtt_01)

打开新的终端，输入以下命令：

```
roslaunch rmtt_tracker rmtt_face_tracker.launch ns:=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 10：TT 无人系前摄画面

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

```
1 #!/usr/bin/env python3
2
3 import rospy
4 import rospkg
5 import std_msgs.msg
6 from geometry_msgs.msg import Twist
7 from sensor_msgs.msg import Image
8 import cv2
9 from cv_bridge import CvBridge
10 import numpy as np
11
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     cv2.imshow('Frame', img)
21     cv2.waitKey(1)
22
23 if __name__ == '__main__':
24
25     rp = rospkg.RosPack()
26     path = rp.get_path("rmtt_tracker")
27     bridge = CvBridge()
28     rospy.init_node('face_tracker', anonymous=True)
29     sub = rospy.Subscriber("image_raw", Image, callback)
30     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 ns:=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 ns:=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）：

```

12 # resize the image to w*h
13 w = 360
14 h = 240
15 pid_w = [0.5, 0, 0] # pid parameters of yaw channel
16 pid_f = [0.8, 0, 0] # pid parameters of forward channel
17 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:

```

92 if __name__ == '__main__':
93
94     rp = rospkg.RosPack()
95     path = rp.get_path("rmtt_tracker")
96     faceCascade = cv2.CascadeClassifier(path + '/config/haarcascade_frontalface_default.xml')
97     bridge = CvBridge()
98     rospy.init_node('face_tracker', anonymous=True)
99     sub = rospy.Subscriber("image_raw", Image, callback)
100     pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
101     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:

```

68 def trackFace(info, w, h, pid_w, pid_f):
69     ## PID
70     # yaw channel
71     error_w = info[0][0] - w // 2
72     speed_w = pid_w[0] * error_w
73     speed_w = int(np.clip(speed_w, -100, 100)) / 100.0
74
75     # forward channel
76     error_f = np.sqrt(info[1]) - 70
77     speed_f = pid_f[0] * error_f
78     speed_f = int(np.clip(speed_f, -100, 100)) / 100.0
79
80     if info[0][0] != 0:
81         yaw_speed = speed_w
82     else:
83         yaw_speed = 0
84
85     if info[1] != 0:
86         forward_speed = speed_f
87     else:
88         forward_speed = 0
89
90     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:

```

19 def callback(msg):
20     global zero_twist_published
21     img = bridge.imgmsg_to_cv2(msg)
22     img = cv2.resize(img, (w, h))
23
24     img, info = findFace(img)
25     yaw_speed, forward_speed = trackFace(info, w, h, pid_w, pid_f)
26     rc = "left: " + str(0) + " forward: " + str(forward_speed) + " up: " + str(0) + " yaw: " +
    str(yaw_speed)
27     speed = Twist()
28
29     if yaw_speed != 0 or forward_speed != 0:
30         speed.linear.x = -forward_speed
31         speed.linear.y = 0.0
32         speed.linear.z = 0.0 # add up channel here
33         speed.angular.x = 0.0
34         speed.angular.y = 0.0
35         speed.angular.z = -yaw_speed
36         rospy.loginfo(rc)
37         pub.publish(speed)
38         zero_twist_published = False
39     else:
40         if not zero_twist_published:
41             pub.publish(speed)
42             zero_twist_published = True
43             rospy.loginfo("no face detected")
44
45     cv2.imshow('Frame', img)
46     cv2.waitKey(1)

```

保存并运行一下这个节点：`roslaunch rmtt_tracker rmtt_face_tracker.launch ns:=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(`ns:=rmtt_01`)

根据之前的方法，在 ROS 中启动键盘遥控(`ns:=rmtt_01`)

打开新的终端，输入以下命令：

```
roslaunch rmtt_tracker rmtt_face_tracker.launch ns:=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：

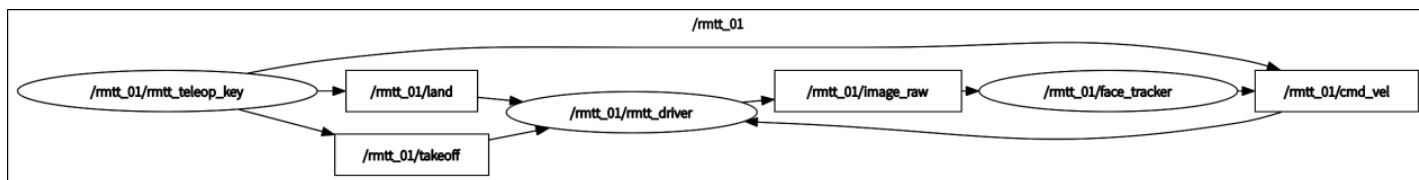


图 11：无人机人脸追踪节点话题关系图

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！

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`。

