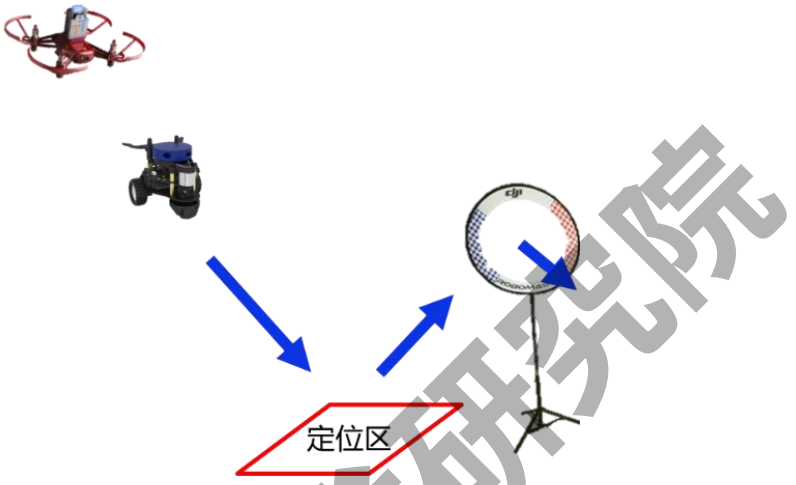
本节课我们将使无人机全自主完成以下几个任务：1. 通过任意跟踪方法将无人机引导到定位板上方；2. 无人机识别到定位板并原地上升到一定的高度；3. 无人机上升到指定高度后原地旋转搜索圆环；4. 无人机识别并跟踪圆环；5. 稳定跟踪后给定前向速度穿越圆环。



本次实验课首先介绍“有限状态机”，让大家了解如何通过编写任务控制节点设计不同任务（状态）之间的切换。然后，将已经实现的圆环跟踪节点加入状态机，实现上述任务流程。我们还给出了通过手势识别控制无人机运动的完整代码，大家可以进行测试。

本节课使用的代码及框架已经在/rmtt_ros/rmtt_tracker/src/中。如果没有下载过无人机驱动包，在工作空间/src 文件夹下开启终端进行下载：

```
git clone https://github.com/cavayangtao/rmtt_ros.git
```

或者：

```
git clone https://gitee.com/cavayangtao/rmtt_ros.git
```

提示：由于TT无人机电量有限，在进行视觉算法调试时可以先调用笔记本电脑摄像头，最后再和无人机进行联调。我们已经在 rmtt_ros 中下载好了笔记本电脑摄像头驱动包 usb_cam package，在工作空间中进行编译后启动摄像头：

```
roslaunch usb_cam usb_cam-test.launch
```

此时图像话题应为：/usb_cam/image_raw

有限状态机

有限状态机（finite-state machine）可以使用状态图（或状态转移图）来表示。下面展示最常见的表示：当前状态（B）和条件（Y）的组合指示出下一个状态（C）。

状态转移表

当前状态→ 条件↓	状态A	状态B	状态C
条件X	...	...	...
条件Y	...	状态C	...
条件Z	...	...	...

在复杂的机器人任务，任务中包含多个状态模块，而这些状态模块之间在某些情况下会发生跳转，这就是有限状态机可以发挥作用的地方。Python 已经实现了一个功能强大且易于扩展的有限状态机程序库 smach。smach 本质上并不依赖于 ROS，可以用于任意 Python 项目，不过在 ROS 中元功能包 executive_smach 将 smach 和 ROS 很好的集成在了一起，可以为机器人复杂应用开发提供任务级的状态机框架。

可以在新的终端中直接使用如下命令安装：

```
sudo apt-get install ros-noetic-executive-smach
sudo apt-get install ros-noetic-executive-smach-visualization
```

前者是 ROS 中的 smach 库，后者用于可视化状态及其转移规则，但该可视化工具在 ROS noetic 存在兼容性问题。

我们已经给出了一个实现好的有限状态机代码框架，该框架可实现以下功能：1. 通过二维码跟踪将无人机引导到定位板上方；2. 无人机识别到定位板并原地上升到一定的高度；3. 无人机上升到指定高度后原地旋转。

首先运行一下，随后我们进行代码讲解。

1、连接无人机到计算机，在新的终端启动无人机驱动：

```
roslaunch rmtt_driver rmtt_bringup.launch
```

2、在新的终端启动无人机的描述文件：

```
roslaunch rmtt_description rmtt_description.launch
```

3、在新的终端启动二维码检测节点：

```
roslaunch rmtt_apriltag detection.launch
```

4、在新的终端启动二维码跟踪节点：

```
roslaunch rmtt_tracker rmtt_tag_tracker.launch
```

5、在新的终端启动任务控制节点(有限状态机)：

```
rosrun rmtt_tracker rmtt_smach.py
```

6、在新的终端启动遥控程序

```
roslaunch rmtt_teleop rmtt_teleop_key.launch
```

现在，起飞 TT 无人机，通过二维码将无人机引导到 1 号定位板上方，看看会发生什么，rmtt_smach.py 任务控制节点的终端是如何显示的？

```
[INFO] [1649472142.147753]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.149020]: Executing state Tag_track
[INFO] [1649472142.150351]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.151694]: Executing state Tag_track
[INFO] [1649472142.153103]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.154609]: Executing state Tag_track
[INFO] [1649472142.155866]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.157204]: Executing state Tag_track
[INFO] [1649472142.158482]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.159706]: Executing state Tag_track
[INFO] [1649472142.160999]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.162434]: Executing state Tag_track
[INFO] [1649472142.163741]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.165005]: Executing state Tag_track
[INFO] [1649472142.166414]: State machine transitioning 'TAG_TRACK':'stay'-->'TAG_TRACK'
[INFO] [1649472142.167694]: Executing state Tag_track
```

一起来看看 rmtt_smach.py 任务控制节点中究竟做了什么。

```
78 if __name__ == '__main__':
79     mission = 1
80     pad_id = 0
81     vel_tag = Twist()
82 
83     rospy.init_node('state_machine')
84 
85     rospy.Subscriber('/mission_pad_id', UInt8, callback_mission_pad)
86     rospy.Subscriber('/cmd_vel_tag', Twist, callback_cmd_tag)
87     rospy.Subscriber('/tof_btm', Range, callback_tof_btm)
88 
89     pub = rospy.Publisher("/cmd_vel", Twist, queue_size=1)
```

从程序主函数看起。mission 表示当前执行的任务号，随后我们会看到它的作用。pad_id 记录无人机下视摄像头所识别到的定位板序号，我们赋初值 0。vel_tag 记录二维码追踪节点生成的速度控制指令。我们订阅了 3 个话题，/mission_pad_id 传递 UInt8 消息类型，是无人机下视摄像头所识别到的定位板序号，/tof_btm 传递 Range 消息类型，是无人机下视红外距离传感器测量的离地距离，这两个话题是固化在无人机驱动中的，可以直接订阅读取。/cmd_vel_tag 订阅二维码追踪节点发布的控制消息（需要将二维码追踪节点发布的速度话题改为/cmd_vel_tag），/cmd_vel 话题用来发布无人机的实际控制指令。

```
65 def callback_cmd_tag(msg):
66     if mission == 1:
67         vel_tag = msg
68         pub.publish(vel_tag)
69 
70 def callback_mission_pad(msg):
71     global pad_id
72     pad_id = msg.data
73 
74 def callback_tof_btm(msg):
75     global height
76     height = msg.range
```

在话题/mission_pad_id 和/tof_btm 对应的回调函数中，获取的定位板序号以及无人机高度分别赋给了全局变量 pad_id 和 height。在话题/cmd_vel_tag 对应的回调函数中，只有当 mission 的值为 1 时才会将二维码追踪节点发布的控制指令发布给/cmd_vel 话题。

```
94     # Create a SMACH state machine
95     sm = smach.StateMachine(outcomes=['end'])
96 
97     # Open the container
98     with sm:
99         # Add states to the container
100        smach.StateMachine.add('TAG_TRACK', Tag_track(),
101                               transitions={'next':'RISE', 'stay':'TAG_TRACK'})
102        smach.StateMachine.add('RISE', Rise(),
103                               transitions={'next':'TURN', 'stay':'RISE'})
104        smach.StateMachine.add('TURN', Turn(),
105                               transitions={'stay':'TURN'})
```

这一段程序首先创建了一个状态机，当状态返回“end”时将结束状态机的工作，然而随后会发现结束标志我们并没有实际使用。然后在状态机中定义了 3 个状态分别是 TAG_TRACK、RISE、TURN，对应了三个类 Tag_track()、Rise()、Turn（）。以 TAG_TRACK

状态为例，当处于 TAG_TRACK 状态时将执行“Tag_track()”类中的函数。我们还在这一状态中定义了跳转条件，当返回“next”时跳转到 RISE 状态，当返回“stay”时跳转到 TAG_TRACK 状态，后者跳转回了当前状态，并未发生改变。

```
111     while not rospy.is_shutdown():
112         outcome = sm.execute()
113         # rate.sleep()
114 
115     # Wait for ctrl-c to stop the application
116     rospy.spin()
117     # sis.stop()
```

在 while not rospy.is_shutdown()循环中执行定义的有限状态机。rospy.spin()用来等待话题消息，一旦接收到消息，将调用对应话题的回调函数。

```
13 # define state Tag_track
14 class Tag_track(smach.State):
15     def __init__(self):
16         smach.State.__init__(self,
17                              outcomes=['next', 'stay'])
18 
19     def execute(self, userdata):
20         global pad_id, mission
21         rospy.loginfo('Executing state Tag_track')
22         if pad_id == 1:
23             zero_twist = Twist()
24             pub.publish(zero_twist)
25             mission = 2
26             return 'next'
27         else:
28             return 'stay'
```

现在看定义的 TAG_TRACK 状态。在状态初始化阶段我们需要定义好两个状态跳转条件“next”和“stay”。在执行阶段首先获取了两个全局变量 pad_id 和 mission。当 pad_id 为 1 时我们将给无人机 0 速度并给 mission 赋值为 2，并返回“next”跳转条件，状态机将跳转到下一阶段。否则，将停留在当前阶段。

```
30 # define state Rise
31 class Rise(smach.State):
32     def __init__(self):
33         smach.State.__init__(self,
34                              outcomes=['next', 'stay'])
35 
36     def execute(self, userdata):
37         global height, mission
38         rospy.loginfo('Executing state Rise')
39         if height > 1.5:
40             zero_twist = Twist()
41             pub.publish(zero_twist)
42             mission = 3
43             return 'next'
44         else:
45             vel_rise = Twist()
46             vel_rise.linear.z = 0.2
47             pub.publish(vel_rise)
48             return 'stay'
```

在 RISE 状态执行阶段，首先获取两个全局变量 height 和 mission。当升高到 1.5 米时我们将给无人机 0 速度并给 mission 赋值为 3，并返回“next”跳转条件，状态机将跳转到下一阶段。否则，无人机原地上升，停留在当前状态。

```
50 # define state Turn
51 class Turn(smach.State):
52     def __init__(self):
53         smach.State.__init__(self,
54                              outcomes=['stay'])
55 
56     def execute(self, userdata):
57         global mission, rad
58         rospy.loginfo('Executing state Turn')
59 
60         vel_rise = Twist()
61         vel_rise.angular.z = 0.2
62         pub.publish(vel_rise)
63         return 'stay'
```

在 TURN 状态中我们目前只定义了一个跳转条件“stay”，之后请同学们继续扩展。在

该状态中无人机将原地转圈。需要注意的是，无人机的二维码跟踪指令我们是在回调函数中发布的，但原地上升和原地转圈指定我们直接在状态机进行了发布。原因有限状态机的执行效率要低于回调函数，在状态机中发布二维码跟踪指令会导致跟踪失败，而原地上升和原地转圈只有恒定速度的发布。

圆环跟踪及穿越

圆环的检测通过霍夫变换进行，我们直接使用 OpenCV 中封装好的圆环检测函数。

1、连接无人机到计算机，在新的终端启动无人机驱动：

roslaunch rmtt_driver rmtt_bringup.launch

2、在新的终端中，运行圆环检测节点：

rosrun rmtt_tracker rmtt_circle_detection.py

运行代码后将无人机摄像头对准穿越圆环，会看到在画面中绘制出了圆心和圆轮廓。


```
69 if __name__ == '__main__':
70
71     bridge = CvBridge()
72     rospy.init_node('circle_tracker', anonymous=True)
73     sub = rospy.Subscriber('/image_raw', Image, callback)
74     pub = rospy.Publisher('/circle_msg', ROI, queue_size=1)
75
76     rospy.spin()
```

rmtt_circle_detection.py 节点的主函数中订阅了无人机摄像头的视频流，圆环识别将在/image_raw 话题对应的回调函数中实现。最终 ROI 消息发布到/circle_msg 话题中。ROI（RegionOfInterest）消息用来存储感兴趣区域框的坐标和边长，其包含：

uint32 x_offset

uint32 y_offset

uint32 height

uint32 width

bool do_rectify

offset、y_offset 用来存储图像中矩形框的左上顶点坐标，height、width 用来存储矩形框

的高和宽。在这里用 ROI 消息存储检测圆形的中心坐标和半径。

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

    img, info = findCircle(image)
    circle_msg = trackCircle(info)

    rc = 'cx:' + str(circle_msg.x_offset) + 'cy:' + str(circle_msg.y_offset) + 'radius:' +
  str(circle_msg.width)
    rospy.loginfo(rc)
    pub.publish(circle_msg)

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

callback()回调函数中，首先利用 cv_bridge 将 ROS 的图片消息格式转成 OpenCV 支持的图片格式。findCircel()函数用于读入图片，并使用 OpenCV 库检测圆，trackCircle()用于将检测到圆环的圆心和半径写入 ROI 消息。rospy.loginfo()用来在终端中显示相关信息，这在 ROS 程序调试中经常用到。最后我们将圆环的信息进行了发布，并在图像中显示该圆环及其圆心。

```
def findCircle(img):
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgGray = cv2.medianBlur(imgGray,5)
    circles = cv2.HoughCircles(imgGray, cv2.HOUGH_GRADIENT, dp=1, minDist=70, param1=50, param2=50,
  minRadius=60, maxRadius=120)
    myCircleListC = []
    myCircleListArea = []
    if circles is not None:
        circles = np.uint16(np.around(circles))

    # get the circle of the largest bounding box
        for (x,y,r) in circles[0,:]:
            cv2.circle(img,(x,y),r,(0,255,0),2)
    #画个圆心
            cv2.circle(img,(x,y),2,(0,255,255),2)
            area = r*r*pi
            myCircleListArea.append(area)
            myCircleListC.append([x, y])
        i = myCircleListArea.index(max(myCircleListArea))
        return img, [myCircleListC[i], myCircleListArea[i]]
    else:
        return img, [[0, 0], 0]
```

可以看到，检测圆环的流程和 rmtt_face_tracker.py 人脸跟踪例程类似，这里调用了 cv2.HoughCircles 函数进行圆环检测。注意到有一些参数可以设定调整。霍夫检测圆是基于梯度的，梯度良好的情况下 dp=1 没有问题，当梯度周围或者圆上有干扰的时候，应当是当减小累计值，所以需要把 dp 值适度增加，这样会提高霍夫圆的检测能力。

假如检测到多个圆环，只返回面积最大的圆环圆心及面积。

```
def trackCircle(info):
    circle_msg = ROI()
    if int(info[1]) == 0:
        circle_msg.x_offset = 0
        circle_msg.y_offset = 0
        circle_msg.width = 0
        circle_msg.height = 0

    else:
        circle_msg.width = int(np.sqrt(info[1]/pi))
        circle_msg.x_offset = int(info[0][0])
        circle_msg.y_offset = int(info[0][1])
        circle_msg.height = 0
    return (circle_msg)
```

trackCircle()函数将圆心坐标和半径赋给了 ROI 消息对象 circle_msg 中。

请同学们自行尝试实现以下功能：

1. **编写节点订阅/circle_msg 话题，根据圆心距离画面中心的位置生成升降及偏航（左右转动）通道控制指令，根据圆的半径与参考长度（如 80）的比较生成俯仰（前后）**

通道控制指令。该节点可在上节课通用目标跟踪器的控制节点基础上简单修改，参考 rmtt_tracker 程序包中 rmtt_face_tracker.py 节点。

2. 在 rmtt_smach.py 节点中增加两个状态 TRACK_CIRCLE 和 CROSS_CIRCLE。一旦识别到圆（可订阅/circle_msg 话题）状态从 TURN 转移到 TRACK_CIRCLE。
3. 在 TRACK_CIRCLE 中一旦实现稳定圆环跟踪（可通过检测的半径是否在指定范围，以及圆环跟踪的速度控制量是否小于阈值判断），状态转移到 CROSS_CIRCLE，否则状态停留在 TRACK_CIRCLE。
4. 在 CROSS_CIRCLE 中，发布俯仰控制指令使无人机持续向前运动，可以增加一个定时器使得无人机运动一段时间后停止。一旦进入状态 CROSS_CIRCLE，将在该状态停留。