语音巡逻

通过前面的课程讲解，我们理解了整个实现语音自定义交互的过程。接下来我们讲解的是通过语音交互功能来控制我们的机器人进行巡逻。而我们的语音导航也是基于前面讲解的自定义交互模式设计的一种功能实现。

整体效果

默认demo通过语音下达开始巡逻或者结束巡逻指令，可以控制机器人的运行在设定的点位进行巡逻。

实现过程

首先我们得通过唤醒词将机器人唤醒，让他进入唤醒状态。所以我们的前面讲解的语音离线命令词检测和 VAD检测得先打开。

```
roslaunch handsfree_voice offline_interactive_ros.launch//语音交互功能节点
rosrun handsfree_voice vad_record.py//VAD检测节点
```

根据语音第一节的内容,我们可以先使用仿真环境进行测试，先看一下效果 ，这里我们也提供了相应的仿真环境。

```
roslaunch handsfree_stage demo_handsfree_room_stage.launch
roslaunch handsfree_stage demo_handsfree_xuda_stage.launch
```

这两个都是仿真环境，在空间不够空旷的情况下或者还没熟练的情况下，可以执行其中任意一个就能进行仿真测试。通过仿真环境，用我们的语音交互的功能去看执行效果，进行驱动功能的调试。

打开第二个终端，开启激光节点

```
roslaunch rplidar_ros rplidar.launch
```

打开第三个终端，开启move_base节点

```
roslaunch handsfree_2dnav move_base_amcl.launch map_name:=my_map
```

打开第四个终端，开启RVIZ可视化工具

```
rosrun rviz rviz -d `rospack find handsfree_bringup`/rviz/navigation.rviz
```

初始位置的标定 通过`2D Pose Estimate`标定初始化位置，直到机器人在地图中的位置和机器人实际的位置一样。

然后我们运行

```
rosrun handsfree_voice set_goal.py//语音导航节点
```

控制核心

```
    XL_goal=0
    XL_finish=0
    rospy.init_node('smach_serial_demo2',log_level=rospy.DEBUG)
    server_movebase = actionlib.SimpleActionClient('/move_base', move_bas
e_msgs.msg.MoveBaseAction)
    connect_state = server_movebase.wait_for_server()
    points = [[11.3,50.1],  # target point 1  only need x and y
              [14.7,41.9],  # target point 2
              [16.9,18.2],  # target point 3
              [10.4,25.4]]  # target point 4
```

这里修改我们的巡逻点位的设置。

```
----------------------------------
def speech_callback(msg):
        #rospy.loginfo(msg)
        global XL_goal
        global XL_finish
        if msg.data == 'patrol':
            XL_goal=1
        elif msg.data == 'stoppatrol':
            XL_goal=0
        elif msg.data == 'shutdown':
            XL_finish=1
----------------------------------
 def execute(self, ud):
        goal = gen_new_movebase_goal('map',
                                     self.__target_x,
                                     self.__target_y,
                                     self.__next_target_x,
                                     self.__next_target_y)
        if XL_goal==1:
            server_movebase.send_goal(goal)
        # todo: what should robot do when failed to finish the target poi
nt?
        is_server_availabel = server_movebase.wait_for_result()
        if is_server_availabel is False:
            return 'error_exit'
        else:
```

```
            if XL_finish==1:
               return 'error_exit'
            else:
               return 'next_point'
-----------------------------------
    with smach_serial_demo2:
        while XL_goal==0:
              b=0
        if XL_goal==1 and a==0:
                    a+1
        # to point in points list into state machine
                    for target_point in range(0, size_points, 1):
                        next_target_point = (target_point+1) % size_point
s
                        name_target_point = 'POINT%d' % target_point
                        name_next_target_point = 'POINT%d' % next_target_
point
                        smach.StateMachine.add(name_target_point,
                                   MoveToPoint(target_x=points[target_poi
nt][0],
                                               target_y=points[target_poi
nt][1],
                                               next_target_x=points[next_
target_point][0],
                                               next_target_y=points[next_
target_point][1]),
                                   transitions={'next_point': name_next_t
arget_point})
        smach_serial_demo2.execute()
```

修改我们的SMACH状态机的状态，当开始巡逻指令发出，SMACH状态机开始工作。

关于smach状态机

smach提供了actionlib整合和smach viewer两大组件。samch viewer可以实时查看任务执行当前的状态结点位置，便于开发调试。smach还整合了动作状态，例如定点导航，可以把topic，service转化为状态，也可以把一个类转化为状态结点。smach还可以实现状态的并发执行，任务的重复执行，层次嵌套复杂的状态机。
SMACH有利于控制机器人复杂动作的具体实现、在SMACH中,所有可能的状忘和状态转移都能被明确定义。SMACH有以下特点：

快速原型:利用基于Python的状态机语法,能够快速构建可执行的状态机。
复杂状态机:SMACH支持设计、维护和调试大型的复杂的层次状态机。
自省:SMACH可以创建状态机的自省程序,包括状态转移、用户数据等要素的初始化。
由于SMACH旨在提供任务级的状态机描述，因此并不适用于无结构性的任务，以及需要较高执行效率的底层系统。

这里就是一直在判断当前的状态如果到了新的节点，就会进行下一个动作状态的任务，开始巡逻循环。

当结束巡逻指令发出后，SMACH状态机因为我们设计的一个条件判断状态，当满足巡逻结束条件满足时，我们的状态机到了新的状态的时候，会因为不满足条件，而不发送下一个节点的信息给`/move_base`。实现一个停止巡逻的状态。

当退出语音指令发出的时候，满足了完成任务的标志位信息，会切换SMACH自身的一个状态为结束状态，将进程杀死。

下面是一个科大讯飞的错误码信息查询，如果在调用科大讯飞的时候报错了相关的错误码，可以通过这个链接查询。
科大讯飞错误码信息查询