工程实践与科技创新 IV-E

Lecture 2: ROS 入门实验

授课老师：张晗、王景川 *2023* 年春季

1 分组情况

2 超算平台配置与 ROS 基本指令

2.1 超算平台配置

请按照配置指南搭建好自己的环境

2.2 小海龟仿真器的运行与实践

2.2.1 小海龟仿真器

安装完成后可以使用 ROS 自带的小海龟例程进行验证。首先打开一个终端运行 roscore，启动 rosmaster；再打开一个终端启动小海龟仿真器，最后再打开一个终端启动海龟控制节点。此时可以使用方向键控制小海龟运动即可说明 ROS 安装成功。注意控制小海龟时要注意输入焦点在控制节点对应的终端中。

1. roscore 指令启动：

   roscore

   roscore 是基于 ROS 的系统的先决条件的节点和程序的集合，必须运行 roscore 才能使 ROS 节点进行通信。

2. 启动小海龟节点：

   rosrun turtlesim turtlesim_node

   启动成功后会打开一个可视化终端，效果如图 Fig. 1。注意：其中小海龟的图样是随机的，会有所不同。

3. 启动小海龟键盘控制节点：

   rosrun turtlesim turtle_teleop_key

   如图图 Fig.2，打开键盘节点后，使用键盘的方向键即可控制小海龟向对应方向运动。注意，要保证输入焦点在 turtle_teleop_key 的终端才能正常使用。

2.2.2 查看 ROS 计算图

1. rqt_graph 提供了一个 GUI 插件，用于可视化 ROS 计算图。可以使用 rqt_graph 命令打开可视化的计算图：

   rqt_graph

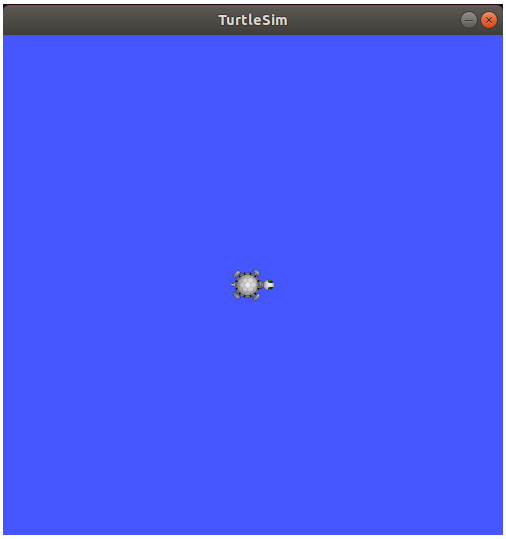

Figure 1: turtlesim_node

Figure 2: turtle_teleop_key

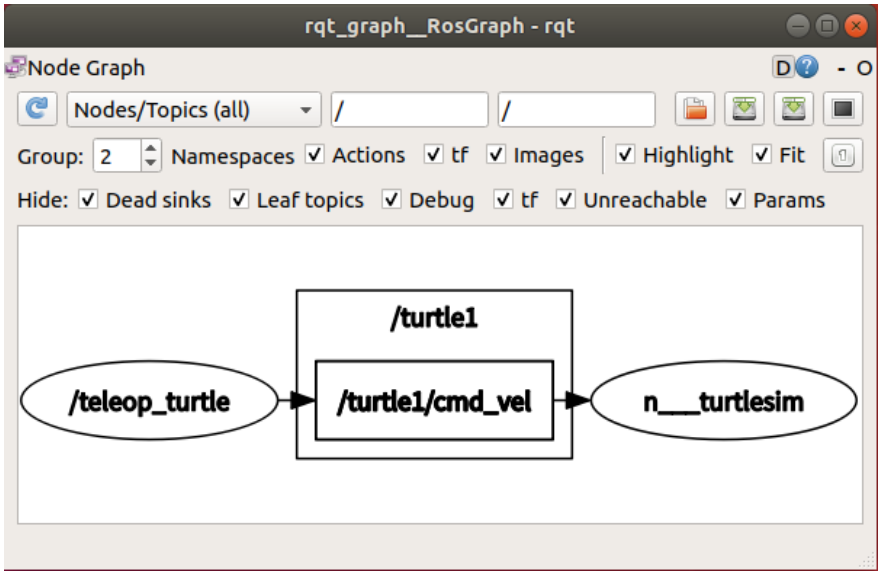

Figure 3: rqt_gragh

2. 打开后即可看到如图 Fig.3所示的可视化弹窗。其中显示了当前所开启的全部节点，包括 turtlesium 和 turtle_teleop_key 及其对应的话题。

2.2.3 ROS 的常用命令

rosnode 是一个命令行工具，用于显示有关 ROS 节点的调试信息，包括发布，订阅和连接。

1. 其中 rosnode info 可以显示节点的相关信息，包括发布和订阅:

```
$ rosnode info /turtlesim
--------------------------------------------------------------------------------
Node [/turtlesim]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/color_sensor [turtlesim/Color]
 * /turtle1/pose [turtlesim/Pose]
Subscriptions:
 * /turtle1/cmd_vel [unknown type]
Services:
 * /clear
 * /kill
 * /reset
 * /spawn
 * /turtle1/set_pen
 * /turtle1/teleport_absolute
 * /turtle1/teleport_relative
 * /turtlesim/get_loggers
```

```
 * /turtlesim/set_logger_level
contacting node http://192.168.3.154:45825/ ...
Pid: 2406
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (36671 - 192.168.3.154:35814) [24]
    * transport: TCPROS
```

2. rosnode list 可以显示当前节点的列表:

```
$ rosnode list
/rosout
/teleop_turtle
/turtlesim
```

3. rostopic 包含 rostopic 命令行工具，用于显示有关 ROS 话题的调试信息，包括发布者，订阅者，发布率和 ROS 消息。rostopic list 可以显示当前话题列表:

```
$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_cel
/turtle1/color_sensor
/turtle1/pose
```

4. rostopic info 可以打印话题有关的信息:

```
$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://192.168.3.224:42663/)

Subscribers:
 * /turtlesim (http://192.168.3.224:40187/)
```

5. rostopic echo 可以显示话题发布的消息。例如查看小海龟的位姿数据:

```
$ rostopic echo /turtle1/pose
x: 5.544444561
y: 5.544444561
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
```

6. rostopic pub 可以发布话题数据，可以通过此命令发布小海龟的速度消息，使小海龟进行圆周运动:

```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist
"linear:
x:10.0
y:0.0
z:0.0
angular:
x:0.0
y:0.0
z:10.0"
```

其中可以增加 -r 参数，表示速率，默认为 10hz。

rosservice 包含用于列出和查询 ROS 服务的命令行工具，它包含一个 Python 库，用于检索有关服务的信息并动态调用它们。

7. rosservice list 显示活动中的服务的信息：

```
$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

8. rosservice info 可以打印服务有关的信息：

```
$ rosservice info /turtle1/set_pen
Node: /turtlesim
URI: rosrpc://nvidia:35261
Type: turtlesim/SetPen
Args: r g b width off
```

9. rosservice call 可以用输入的参数调用服务：

```
$ rosservice call /turtle1/set_pen "{r:255,g:0,b:0,width:5,'off':0}"
```

课堂示例是请求/turtle1/set_pen 服务的命令。所使用的“255 0 0 5 0”是对应于用于/turtle1/set_pen 服务的参数 (r，g，b，width，off) 的值。红色的 r 的最大值是 255，因为 g 和 b 都是 0，所以笔的颜色是红色的。width 设置为 5，off 为 0（假）。

3 ROS 例程：Publisher / Subscriber

本节将通过例程介绍，如何从搭建 catkin 工作空间开始，编写你自己的 ROS package，自定义一个 message 实例，并实现一对发布者（Publisher）/订阅者（Subscriber），最后通过 roslaunch 启动他们。简便起见，本课程主要使用 python2 作为编程语言，这里不涉及 C++ 的部分。

3.1 Catkin 工作空间

Catkin 是 ROS 官方推荐的编译系统，结合了 CMake 和 Python，对 CMake 的工作流进行封装，使用更简便。相比旧的 rosbuild，catkin 提供了更好的移植性。更具体的描述可参阅`http://wiki.ros.org/catkin/conceptual_overview`。

Catkin 工作空间（Catkin Workspace），简单的说，是你编辑和编译代码的文件夹。只要搭好规范的文件目录结构，并在文件（如 CMakeLists.txt）中给出要求的编译规则，catkin 可以帮你“一键”编译好目录下的所有项目！关于 catkin workspace 更详细的说明可查阅`http://wiki.ros.org/catkin/workspaces`。但现在可以先跳过这些细节，开始搭建你的 catkin 工作空间。在终端运行以下命令

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin_make
```

catkin_make 是 catkin workspace 下的编译指令。初次运行时，目录下会生成 build、devel 两个文件夹，并在 src 文件夹生成 CMakeLists.txt 文件。

要使用这个新工作空间下的功能包（package），必须给终端添加一些环境变量。catkin 刚刚已经帮你准备好了脚本，继续输入命令 (以 bash 为例)：

```
$ source ./devel/setup.bash
```

检查一下环境变量是否正确：

```
$ echo $ROS_PACKAGE_PATH
```

预期大概会得到：

```
/home/ubuntu/catkin_ws/src:/opt/ros/kinetic/share
```

这是当前终端所知的所有 ros 功能包的路径。路径之间由冒号分隔。

3.2 创建新功能包（package）

接下来，在工作空间里创建一个新功能包：

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tutorials std_msgs rospy
# catkin_create_pkg <package_name> [depend1] [depend2]
```

这里用到了新命令：catkin_create_pkg，它创建了一个名叫 tutorials 的新功能包，这个包依赖于已有的 ros 功能包：std_msgs, rospy。运行这个命令之后，你会看到新文件夹 catkin_ws/src/tutorials。其中 package.xml 文件中是该功能包的基本数据，包括名称描述、作者、证书、依赖项等。其中最重要的是 51 行开始的这一部分：

```
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
```

接下来编译这个新功能包

```
$ cd ~/catkin_ws
$ catkin_make
```

这样就生成了一个新功能包。例如，可以直接使用 roscd 命令访问，不必输入绝对路径:

```
$ roscd tutorials
# it is equivalent to
# cd ~/catkin_ws/src/tutorials
```

3.3 创建一对发布者 (Publisher)/订阅者 (Subscriber) 节点 (Node)

简单的说，节点 (Node) 就是 ROS 功能包中的一个可执行文件。Topic 是节点之间信息交流的方式之一。例如，对一个 Topic，节点 A，B 注册成为该 Topic 的发布者 (Publisher)，他们都可以不定时的向 Topic 发送格式化的消息 (message)；节点 C，D 可以订阅 (Subscribe) 该 Topic，每当 Topic 上出现新 message，C，D 都会收到通知，并调用各自的回调函数 (Callback Function) 处理新 message。

接下来我们将在刚刚创建的功能包中编写一对发布者/订阅者节点。

3.3.1 发布者节点 (Publisher Node)

```
$ roscd tutorials
$ mkdir scripts
$ cd scripts
$ touch str_talker.py
```

接下来编辑 str_talker.py，代码如下。也可以在例程中找到。

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5
6  def talker():
7      pub = rospy.Publisher('chatter',\
8          String, queue_size=10)
9      rospy.init_node('talker', anonymous=True)
10     rate = rospy.Rate(10) # 10hz
```

```
11      while not rospy.is_shutdown():
12          hello_str = "hello world %s"%rospy.get_time()
13          rospy.loginfo(hello_str)
14          pub.publish(hello_str)
15          rate.sleep()
16
17  if __name__ == '__main__':
18      try:
19          talker()
20      except rospy.ROSInterruptException:
21          pass
```

逐行解释一下代码的含义，

```
3   import rospy
4   from std_msgs.msg import String
```

调用 ROS 提供的 python 库，所有用 python 实现的 ROS Node 都会调用该库。std_msgs.msg 提供了一些标准的 message 格式，这里使用 std_msg/String 格式的 message。

```
7       pub = rospy.Publisher('chatter',\
8           String, queue_size=10)
```

接下来进入主函数 talker()，这段代码表示这个节点向名叫 “chatter” 的 Topic 发送 message，message 格式是 “String”，queue_size 表示队列长度，当订阅者反映太慢时，队列不会堆积超过 10.

```
9       rospy.init_node('talker', anonymous=True)
```

上面这条命令初始化了这个节点，他的名字定为 talker，anonymous=True 通过给节点加随机后缀名的方式保证节点名的唯一性。

```
10      rate = rospy.Rate(10) # 10hz
11      while not rospy.is_shutdown():
12          hello_str = "hello world %s" % rospy.get_time()
13          rospy.loginfo(hello_str)
14          pub.publish(hello_str)
15          rate.sleep()
```

第 10 行和第 15 行的命令合起来保证该循环以 10hz 的频率运行（按照 ROS 的虚拟时钟）。while 循环检查程序是否退出，若未退出则执行循环。rospy.loginfo(hello_str) 将信息打印在终端中，写入该节点的 log file，同时发给 rosout 节点。rosout 节点可以配合 rqt_console 工具，方便开发者 debug。pub.publish(hello_str) 则是向 “chatter” Topic 发布 message。

```
17  if __name__ == '__main__':
18      try:
19          talker()
20      except rospy.ROSInterruptException:
21          pass
```

当运行节点时，调用主函数 talker()。异常 rospy.ROSInterruptException 可能由 rospy.sleep() 或 rospy.Rate.sleep() 抛出，表示节点关闭（例如你按下 Ctrl+C）。

3.3.2 订阅者节点 (Subscriber)

接下来创建订阅者节点。

```
$ roscd tutorials/scripts
$ touch str_listener.py
```

编辑 str_listener.py 如下，例程中也可以找到：

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String
4
5  def callback(data):
6      rospy.loginfo(rospy.get_caller_id() \
7          + "I heard %s", data.data)
8
9  def listener():
10     rospy.init_node('listener', anonymous=True)
11     rospy.Subscriber("chatter", String, callback)
12     rospy.spin()
13
14 if __name__ == '__main__':
15     listener()
```

订阅者的代码中只有两行是新功能。

```
11     rospy.Subscriber("chatter", String, callback)
```

该语句表示该节点订阅 “chatter” Topic，message 类型是 String，当收到 message 时，调用 callback 函数，并以收到的 message 作为 callback 的第一个参数。

```
12     rospy.spin()
```

该语句保证订阅者节点在后台持续运行并接受处理 message。

3.3.3 编译节点并测试

写好发布者/订阅者的程序，先修改文件权限，再编译工作空间。

```
$ chmod +x ./str_listener.py     # 给文件以执行权限
$ chmod +x ./str_talker.py
$ cd ~/catkin_ws
$ catkin_make
```

打开 3 个终端窗口，分别在里面运行：

```
$ ## ===== Terminal No.1 =======
$ cd ~/catkin_ws
$ source ./devel/setup.bash
$ roscore
```

```
$ ## ===== Terminal No.2 =======
$ cd ~/catkin_ws
$ source ./devel/setup.bash
$ rosrun tutorials str_talker.py
```

```
$ ## ===== Terminal No.3 =======
$ cd ~/catkin_ws
$ source ./devel/setup.bash
$ rosrun tutorials str_listener.py
```

你会在终端 No.2 No.3 中分别看到发布者和订阅者输出的 helloworld。

3.3.4 定义自己的 message 格式

有时，ROS 提供的 message 格式并不能满足我们的需求，我们需要定义自己的 message 格式。
首先创建一个.msg 文件

```
$ roscd tutorials
$ mkdir msg
$ touch ./msg/person.msg
```

编辑该文件如下：

```
string name
uint8 age
```

文件每行是一个成员，前半段是成员类型，后半段是成员名。msg 支持的成员类型有：

- int8, int16, int32, int64 (还有 uint*)
- float32, float64
- string
- time, duration
- other msg files(其他文件定义的 msg 类型)
- array[](变长数组), array[C](定长数组)

修改功能包的 package.xml 文件（tutorials/package.xml），添加依赖：

```
  <build_depend>message_generation</build_depend>
  <exec_depend>message_runtime</exec_depend>
```

修改 package 的 CMakeLists.txt(tutorials/CMakeLists.txt)，修改 find_package 部分，改成这样：

```
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  message_generation
)
```

修改 catkin_package 部分，改成这样：

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

找到 add_message_files 部分，取消注释，改成这样：

```
add_message_files(
  FILES
  person.msg
)
```

最后找到 generate_messages 部分，取消注释，变成这样：

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

这样就完成了。你可以使用 rosmsg 命令查询一下，确认你的 message 类型成功创建了。

```
$ rosmsg show person
```

3.4 使用 launch 文件快速启动多个节点

回忆刚刚测试发布者/订阅者节点时，我们启动了三个终端分别运行 roscore 和两个节点，非常繁琐。ROS 项目中常常有许多节点需要启动，这时我们可以编写 launch 文件，一次性启动所有节点（也会自动启动 roscore）。

顺便，我们可以测试一下刚刚定义的 msg 类型。创建文件：

```
$ touch ./scripts/per_talker.py
$ touch ./scripts/per_listener.py
```

编辑 per_talker.py 如下，例程中也可以找到：

```
1 #!/usr/bin/env python
2
3 import rospy
4 from tutorials.msg import person
5 def talker():
6     pub = rospy.Publisher('people', \
```

```
        person, queue_size = 10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
        msg = person()
        msg.name = "Ice"
        msg.age = 18
        rospy.loginfo("send Name=Ice, age=18.")
        pub.publish(msg)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

编辑 per_listener.py 如下，例程中也可以找到：

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

import rospy
from tutorials.msg import person

def callback(data):
    rospy.loginfo(rospy.get_caller_id() +\
        "heard Name=%s"%data.name)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber('name', person, callback)
    rospy.spin()

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

注意到，per_talker 发布 Topic 名为 people，而 per_listener 订阅 Topic 名为 name。这个问题我们可以在 launch 文件中解决。

首先创建 launch 文件：

```
$ roscd tutorials
$ mkdir launch
$ touch ./launch/nodes.launch
```

编辑 nodes.launch 如下，例程中也可以找到

```
<launch>
```

```
  <group ns="namespace1">
    <node pkg="tutorials" name="talker" type="str_talker.py"/>
    <node pkg="tutorials" name="listener" type="str_listener.py"/>
  </group>

  <group ns="namespace2">
    <node pkg="tutorials" name="talker" type="per_talker.py">
    <remap from="people" to="name"/>
    </node>
    <node pkg="tutorials" name="listener" type="per_listener.py"/>
  </group>

</launch>
```

最后重新编译所有节点，并测试：

```
$ roscd tutorials
$ chmod +x ./scripts/per_listener.py # 给文件以执行权限
$ chmod +x ./scripts/per_talker.py
$ cd ~/catkin_ws
$ catkin_make
$ roslaunch tutorials nodes.launch # 使用roslaunch启动
```

4 （练习）基于消息发布的小乌龟运动编程

4.1 开环控制

在阅读完上述内容后，尝试按照以下流程完成小乌龟的控制，并且实现两个目标。

1. 打开一个终端，并且在终端中运行以下命令。

   ```
   $ mkdir -p ~/catkin_ws/src
   $ cd ~/catkin_ws
   $ catkin_make
   ```

2. 进入 /catkin_ws/src 文件夹, 并且在工作空间中创建功能包。在终端中运行以下命令。

   ```
   $ cd ~/catkin_ws/src
   $ catkin_create_pkg learning_topic roscpp rospy std_msgs \\
   geometry_msgs turtlesim
   ```

   完成后会生成文件夹/catkin_ws/src/learning_topic

3. 进入文件夹 /catkin_ws/src/learning_topic/src

   ```
   $ touch velocity_publisher.py
   ```

   编辑 velocity_publisher.py, 控制小海龟按照圆形和方形进行运动如图 Fig. 4和图 Fig. 5（仿照例程尝试补全代码实现功能） 程序有以下要点：

   (a) Ros 节点初始化

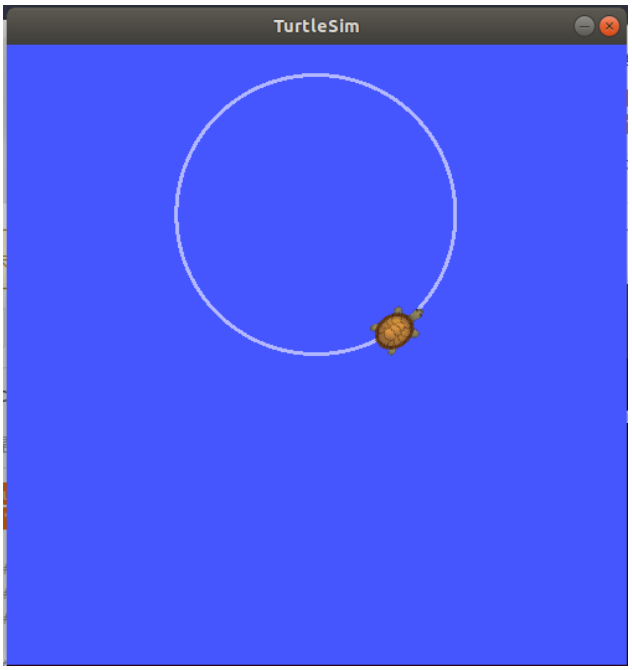

Figure 4: 按照圆形进行运动

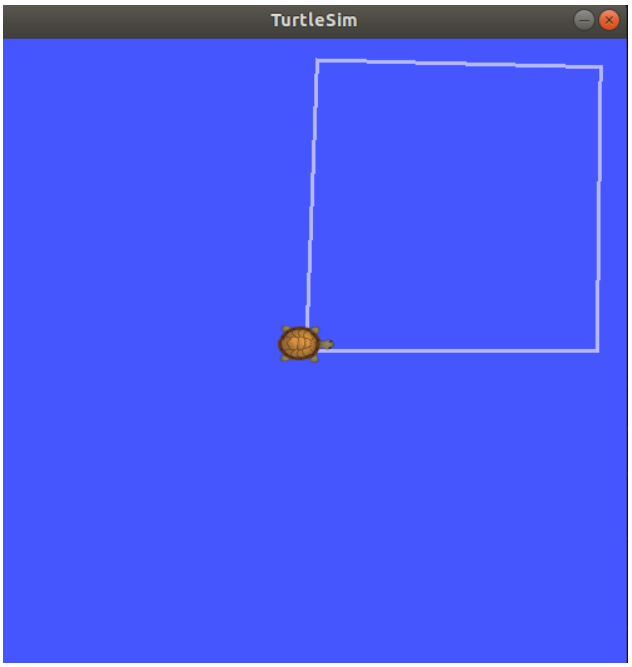

Figure 5: 按照方形进行运动

(b) Publisher 的创建，消息类型为 geometry_msgs::twist, 发布名为/turtle1/cmd_vel

(c) 设置循环的频率

(d) 编写循环：初始化消息类型、发布消息、设置延时

完成 py 文件编写后在文件所在目录下运行

```
$ chmod +x velocity_publisher.py        #chmod为权限控制指令
```

将该 py 文件更改为“允许作为程序执行文件”。

4. 对功能包进行编译并且运行 开启一个新的终端，并且运行

```
$ cd ~/catkin_ws
$ catkin_make                #确认没有报错信息后再进行下一步
$ source devel/setup.bash    #设置当前文件夹为ros工作空间
$ rosrun learning_topic velocity_publisher.py
```

运行之前先确保终端中完成了小海龟节点启动即进行到 2.2.1 的第二个步骤。

4.2 闭环控制

在实现开环控制的基础上，考虑在发出控制指令时订阅小乌龟状态信息，并根据反馈值调整控制量实现闭环。