

# 智能技术与系统综合实验

## 实验手册

### 第 1 讲：ROS 基础与 Turtlebot2 基础部分

## 1 实验内容

### 1.1 基础知识

- (1) 了解什么是工作空间和功能包。（[对应 2.1 和 2.2](#)）
- (2) 了解什么是节点、消息、话题（发布/订阅模型）。（[对应 2.3](#)）
- (3) 了解什么是 Turtlebot2。（[对应 2.4 和 2.5](#)）
- (4) 掌握工作空间和功能包的创建步骤，创建本课程的工作空间，以及本课程的第一个功能包。（[对应 3 和 4.1](#)）
- (5) 掌握话题发布者的创建步骤。（[对应 4.2](#)）
- (6) 掌握话题接收者的创建步骤。（[对应 4.3](#)）
- (7) 掌握节点创建、编译和运行方法。（[对应 4.4-4.6](#)）
- (8) 了解什么是 launch 文件，学习如何使用 launch 文件。（[对应 4.6](#)）

### 1.2 仿真内容

- (1) 在 gazebo 仿真环境中，使用键盘控制 Turtlebot2 移动。（[对应 2.6](#)）
- (2) 在 gazebo 仿真环境中，编程控制 Turtlebot2 移动。（[对应 4.2 和 4.4-4.6](#)）
- (3) 在 gazebo 仿真环境中，编程输出 Turtlebot2 的位姿信息（根据里程计消息转换）。（[对应 4.3 和 4.4-4.6](#)）
- (4) 在 gazebo 仿真环境中，编程控制 Turtlebot2 跟随预设轨迹（如圆形、矩形、五角星形、正多边形等）移动，学习使用 launch 文件修改参数，可视化实时跟随轨迹。（[对应 5](#)）

## 2 ROS 基础与 Turtlebot2 基础

### 2.1 工作空间介绍

工作空间（workspace）是一个存放工程开发相关文件的文件夹。ROS kinetic 默认使用的是 Catkin 编译系统，一个典型 Catkin 编译系统下的工作空间结构如图 2 所示。

典型的工作空间中一般包括以下四个目录空间。

- (1) src: 代码空间（Source Space），开发过程中最常用的文件夹，用来存储所有 ROS 功能包的源码文件。（**重点关注**）
- (2) build: 编译空间（Build Space），用来存储工作空间编译过程中产生的缓存信息和中间文件。
- (3) devel: 开发空间（Development Space），用来放置编译生成的可执行文件。
- (4) install: 安装空间（Install Space），安装空间并不是必需的，很多工作空间中可能并没有该文件夹。

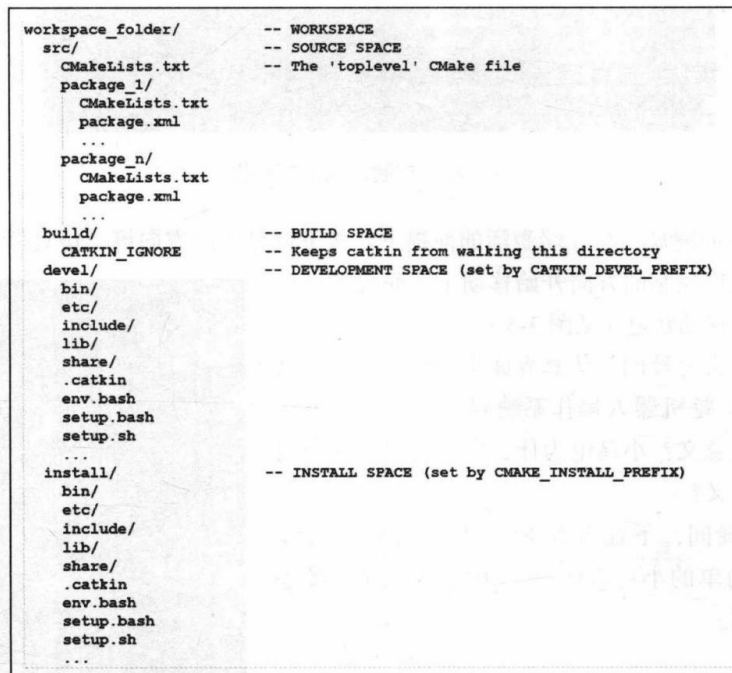


图 1 ROS 工作空间的典型结构

## 2.2 功能包介绍

类似于操作系统，ROS 将所有文件按照一定的规则进行组织，不同功能的文件被放置在不同的文件夹下。功能包（Package）是 ROS 软件中的基本单元，包含 ROS 节点、库、配置文件等。

一个功能包的典型文件结构如图 2 所示。

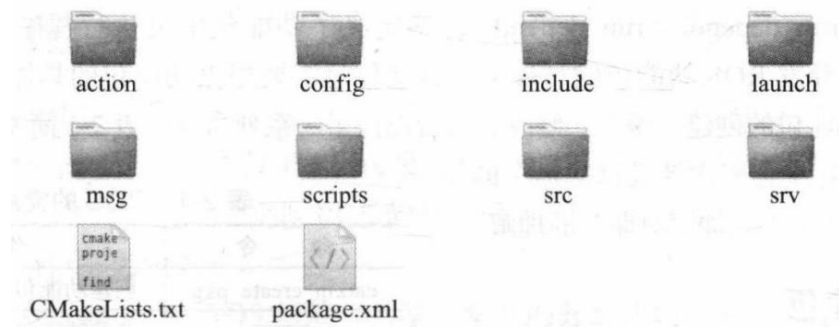


图 2 ROS 功能包的典型结构

一个功能包中的文件（夹）的主要功能如下：

- (1) config: 放置功能包中的配置文件，由用户创建，文件名可以不同。
- (2) include: 放置功能包中需要用到的头文件。（本课程使用，自动创建）
- (3) scripts: 放置可以直接运行的 Python 脚本。
- (4) src: 放置需要编译的 C++代码。（本课程必备，自动创建）
- (5) launch: 放置功能包中的所有启动文件。（本课程使用）
- (6) msg: 放置功能包自定义的消息类型。
- (7) srv: 放置功能包自定义的服务类型。
- (8) action: 放置功能包自定义的动作指令。
- (9) CMakeLists.txt: 编译器编译功能包的规则。（必备，自动创建）
- (10) package.xml: 功能包清单。（必备，自动创建）

每一个 ROS 功能包都是一个独立的功能，其中可能包含一个或者多个节点，这些功能对外使用话题、服务、参数等作为接口。其他开发者在使用这个功能包时，可以不用关注内部的代码实现，只需要知道这些接口的类型和作用，就可以集成到自己的系统中。

## 2.3 计算图介绍

从计算图的角度来看，ROS 系统软件的功能模块以节点为单位独立运行，可以分布于多个相同或不同的主机中，在系统运行时通过端对端的拓扑结构进行连接。

### 2.3.1 节点介绍

节点(Node)就是一些执行运算任务的进程，一个系统一般由多个节点组成，也可以称为“软件模块”。节点概念的引入使得基于 ROS 的系统在运行时更加形象：当许多节点同时运行时，可以很方便地将端对端的通信绘制成如图 3 所示的节点关系图，在这个图中进程就是图中的节点，而端对端的连接关系就是节点之间的连线。

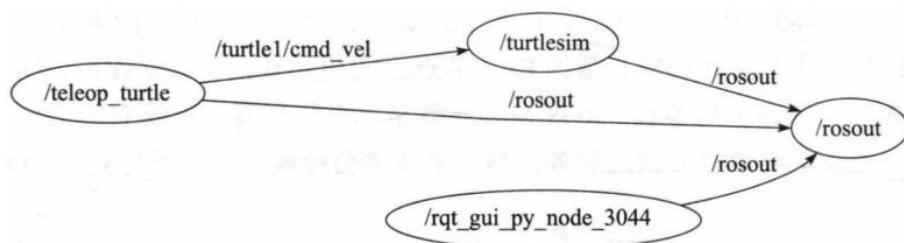


图 3 ROS 中的节点关系图

### 2.3.2 消息介绍

节点之间最重要的通信机制就是基于发布/订阅模型的消息(Message)通信。每一个消息都是一种严格的数据结构，支持标准数据类型（整型、浮点型、布尔型等），也支持嵌套结构和数组（类似于 C 语言的结构体 struct），还可以根据需求由开发者自定义。

### 2.3.3 话题介绍

消息以一种发布/订阅(Publish/Subscribe)的方式传递，如所示。一个节点可以针对一个给定的话题(Topic)发布消息（称为发布者/Talker），也可以关注某个话题并订阅特定类型的数据（称为订阅者/Listener）。发布者和订阅者并不了解彼此的存在，系统中可能同时有多个节点发布或者订阅同一个话题的消息。

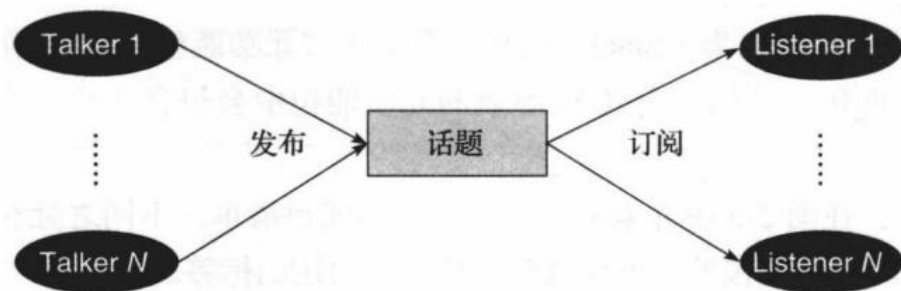


图 4 ROS 中基于发布/定义模型的话题通信

## 2.4 Turtlebot2 介绍

Turtlebot 系列机器人 Willow Garage 公司开发的低成的机器人,如图 5 所示。Turtlebot 的目的是给入门级的机器人爱好者或从事移动机器人编程的开发者提供一个基础平台,让他们直接使用 Turtlebot 自带的软硬件,专注于应用程序的开发,避免了设计草图、购买、加工材料、设计电路、编写驱动、组装等一系列工作。借助该机器人平台,可以省掉很多前期工作,只要根据平台的软硬件接口,就能实现所需的功能。

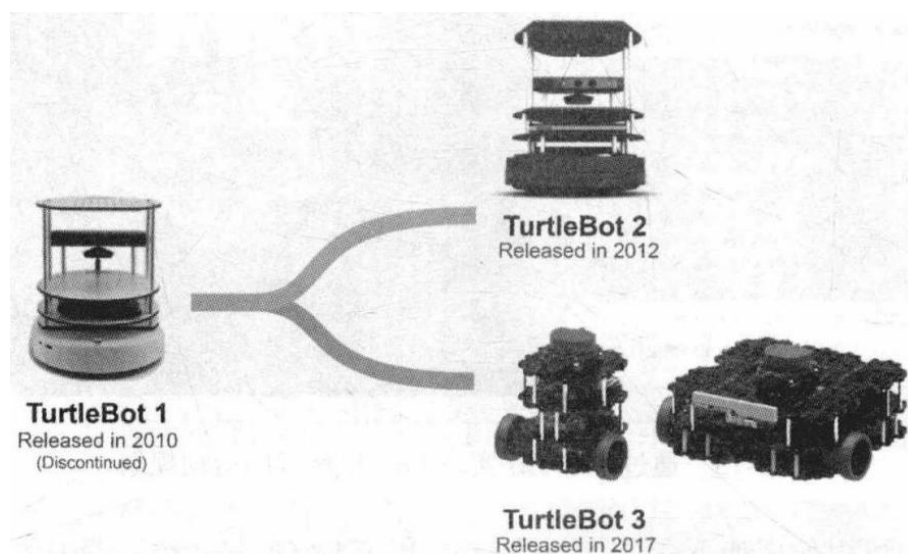


图 5 Turtlebot 机器人

Turtlebot 第一代发布于 2010 年,两年后发布了第二代产品。前两代 Turtlebot 使用 iRobot 的机器人作为底盘,在底盘上可以装载激光雷达、Kinect 等传感器,使用 PC 搭载基于 ROS 的控制系统。

Turtlebot2 模型如图 6 所示,  $x_m$  即为 Turtlebot2 的正前方。

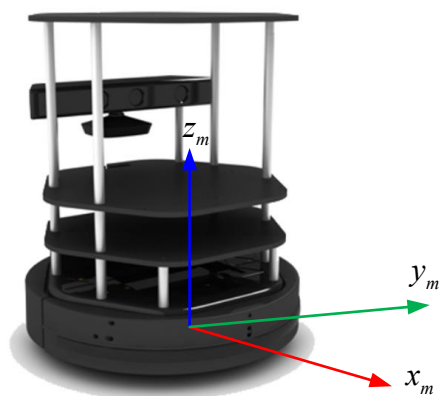


图 6 Turtlebot2 模型

## 2.5 移动机器人模型

移动机器人模型如图 7 所示。

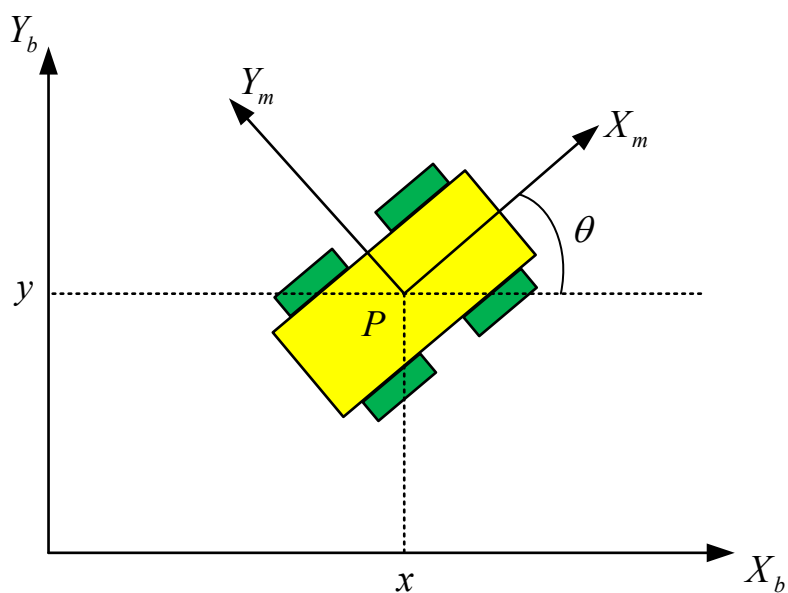


图 7 移动机器人模型

移动机器人的模型定义如下：

移动机器人的位姿通常由三个变量表示：位置  $[x \ y]^T$  和方向  $\theta$ 。

基坐标系表示为  $\{X_b, Y_b\}$ ，移动机器人坐标系表示为  $\{X_m, Y_m\}$ 。

移动机器人在基坐标系下的位姿为  $P = [x \ y \ \theta]^T$ 。

## 2.6 简单控制 Turtlebot2 移动

打开一个终端运行：

**roslaunch turtlebot\_gazebo turtlebot\_world.launch**

如果运行成功，系统自动打开 gazebo 仿真界面，如图 8 所示。

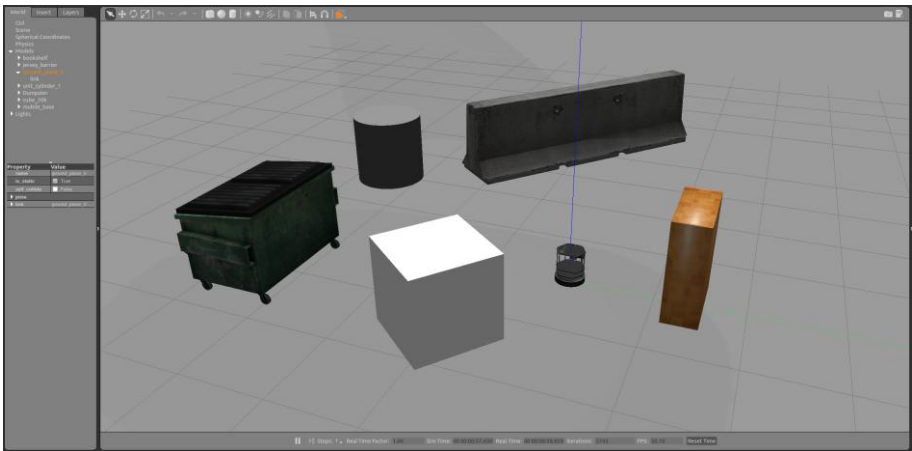


图 8 gazebo 仿真界面

打开一个新的终端运行：

**roslaunch turtlebot\_teleop keyboard\_teleop.launch**

如果运行成功，终端如图 9 所示，按键说明如表 1 所示。

```
Control Your Turtlebot!
-----
Moving around:
  u      i      o
  j      k      l
  m      ,      .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2      turn 1
```

图 9 键盘控制 Turtlebot2 移动的终端

表 1 按键使用说明

u（向左前移动）	i（向前移动）	o（向右前移动）
j（原地向左旋转）	k（暂停）	l（原地向右旋转）
m（向左后移动）	,（向后移动）	.（向右后移动）
q/z（增大或减小最大线速度和最大角速度）		
w/x（增大或减小最大线速度）		

e/c（增大或减小最大角速度）

### 3 创建工作空间

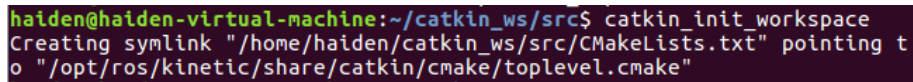
创建工作空间的命令比较简单，首先使用系统命令创建工作空间目录，然后运行 ROS 的工作空间初始化命令即可完成创建过程：

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/src
```

```
catkin_init_workspace
```

如果初始化成功，则可以在终端中看到如图 10 所示的输出信息。



```
haiden@haiden-virtual-machine:~/catkin_ws/src$ catkin_init_workspace
Creating symlink "/home/haiden/catkin_ws/src/CMakeLists.txt" pointing to
"/opt/ros/kinetic/share/catkin/cmake/toplevel.cmake"
```

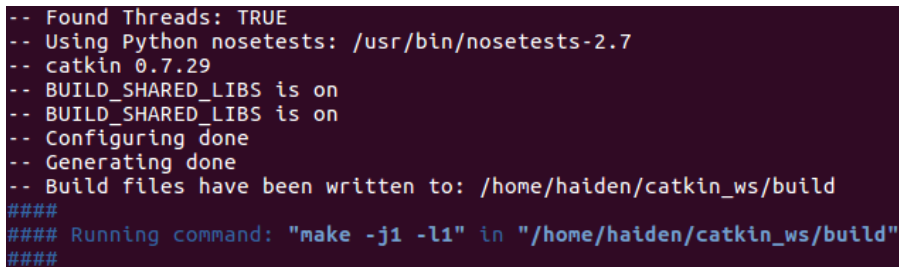
图 10

创建完成后，可以在工作空间的根目录下使用 `catkin make` 命令编译整个工作空间：

```
cd ~/catkin_ws
```

```
catkin_make
```

如果编译成功，则可以在终端中看到如图 11 所示的输出信息。



```
-- Found Threads: TRUE
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.29
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/haiden/catkin_ws/build
####
#### Running command: "make -j1 -l1" in "/home/haiden/catkin_ws/build"
####
```

图 11

编译完成后，在 `devel` 文件夹中已经产生几个 `setup.*sh` 形式的环境变量设置脚本，将它添加到 `.bashrc` 文件：

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

为了确保环境变量已经生效，可以使用如下命令进行检查：



## echo \$ROS\_PACKAGE\_PATH

如果打印的路径中已经包含当前工作空间的路径（红框处），则说明环境变量设置成功（见图 12）。

```
haiden@haiden-virtual-machine:~$ echo $ROS_PACKAGE_PATH
/home/haiden/catkin_ws/src:/home/haiden/turtlebot/src:/home/haiden/kobuki/src:/home/haiden/rocon/src:/opt/ros/kinetic/share
```

图 12

## 4 话题发布和订阅基础

### 4.1 创建 learning\_topic 功能包

首先进入代码空间，使用 `catkin create pkg` 命令创建功能包：

```
cd ~/catkin_ws/src
```

```
catkin_create_pkg learning_topic roscpp std_msgs geometry_msgs nav_msgs
```

如果创建成功，则可以在终端中看到如图 13 所示的输出信息，代码空间 `src` 中会生成一个 `learning_topic` 功能包，其中已经包含 `package.xml` 和 `CMakeLists.txt` 文件。

```
haiden@haiden-virtual-machine:~/catkin_ws/src$ catkin_create_pkg learning_topic
roscpp std_msgs geometry_msgs nav_msgs
Created file learning_topic/CMakeLists.txt
Created file learning_topic/package.xml
Created folder learning_topic/include/learning_topic
Created folder learning_topic/src
Successfully created files in /home/haiden/catkin_ws/src/learning_topic. Please
adjust the values in package.xml.
```

图 13

然后回到工作空间的根目录下进行编译：

```
cd ~/catkin_ws
```

```
catkin_make
```

如果编译成功，则可以在终端中看到如图 14 所示的输出信息。

```
-- catkin 0.7.29
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~ traversing 1 packages in topological order:
-- ~~~~ - learning_topic
-- ~~~~
-- +++ processing catkin package: 'learning_topic'
-- ==> add_subdirectory(learning_topic)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- Configuring done
-- Generating done
-- Build files have been written to: /home/haiden/catkin_ws/build
####
#### Running command: "make -j1 -l1" in "/home/haiden/catkin_ws/build"
####
```

图 14

## 4.2 创建 Publisher 发布速度消息

Publisher 的主要作用是针对指定话题发布特定数据类型的消息。我们尝试使用代码创建一个节点 `velocity_publisher`，节点中创建一个 Publisher 并向话题 `/mobile_base/commands/velocity` 发布速度消息。实现一个 Publisher 的主要流程如下：

- (1) 初始化 ROS 节点。
- (2) 向 ROS Master 注册节点信息，包括发布的话题名和话题中的消息类型。
- (3) 按照一定频率循环发布消息。

源码 `learning_topic/src/velocity_publisher.cpp` 的详细内容如图 15 所示。

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>

int main(int argc, char **argv)
{
    // ROS节点初始化
    ros::init(argc, argv, "velocity_publisher");

    // 创建节点句柄
    ros::NodeHandle nh;

    // 创建一个Publisher, 发布名为/mobile_base/commands/velocity的topic
    // 消息类型为geometry_msgs::Twist, 队列长度10
    ros::Publisher vel_pub = nh.advertise<geometry_msgs::Twist>("mobile_base/commands/velocity", 10);

    // 设置循环的频率
    ros::Rate loop_rate(10);

    while (ros::ok())
    {
        // 初始化geometry_msgs::Twist类型的消息
        geometry_msgs::Twist vel_msg;
        vel_msg.linear.x = 0.5;
        vel_msg.linear.y = 0.0;
        vel_msg.angular.z = 0.2;

        // 发布消息
        vel_pub.publish(vel_msg);
        ROS_INFO("Publish Turtlebot2 velocity command[%0.2f m/s, %0.2f rad/s]",
                 vel_msg.linear.x, vel_msg.angular.z);

        // 按照循环频率延时
        loop_rate.sleep();
    }

    return 0;
}
```

图 15 velocity\_publisher.cpp

### 4.3 创建 Subscriber 订阅里程计消息

接下来，我们尝试创建一个使用代码创建一个节点 odom\_subscriber，节点中创建一个 Subscriber 以订阅话题/odom 的里程计消息。实现一个 Subscriber 的主要流程如下：

- (1) 初始化 ROS 节点。
- (2) 订阅需要的话题。
- (3) 循环等待话题消息，接收到消息后进入回调函数。
- (4) 在回调函数中完成消息处理。

实现源码 learning\_topic/src/odom\_subscriber.cpp 的详细内容如图 16 所示。

```

#include <ros/ros.h>
#include <nav_msgs/Odometry.h>
#include <tf/tf.h>

// 接收到订阅的消息后, 会进入消息回调函数
void odomCallback(const nav_msgs::Odometry::ConstPtr& msg)
{
    // 获取Turtlebot2的位置(x, y)
    double x = msg->pose.pose.position.x;
    double y = msg->pose.pose.position.y;

    // 四元数转换为欧拉角 (获取Turtlebot2的朝向角theta = yaw)
    tf::Quaternion quat;
    tf::quaternionMsgToTF(msg->pose.pose.orientation, quat);
    double roll, pitch, yaw;
    tf::Matrix3x3(quat).getRPY(roll, pitch, yaw);

    // 将接收到的消息打印出来
    ROS_INFO("Turtlebot2's odom: [x:%.2f, y:%.2f, theta:%.2f]", x, y, yaw);
}

int main(int argc, char **argv)
{
    // 初始化ROS节点
    ros::init(argc, argv, "odom_subscriber");

    // 创建节点句柄
    ros::NodeHandle nh;

    // 创建一个Subscriber, 订阅名为/odom的topic, 注册回调函数odomCallback
    ros::Subscriber pose_sub = nh.subscribe("/odom", 10, odomCallback);

    // 循环等待回调函数
    ros::spin();

    return 0;
}

```

图 16 odom\_subscriber.cpp

#### 4.4 编译功能包

节点的代码已经完成, C++是一种编译语言, 在运行之前需要将代码编译成可执行文件。

ROS 中的编译器使用的是 CMake, 编译规则通过功能包中的 CMakeLists.txt 文件设置, 使用 catkin 命令创建的功能包中会自动生成该文件, 已经配置多数编译选项, 并且包含详细的注释, 我们几乎不用查看相关的说明手册, 稍作修改就可以编译自己的代码。

打开 learning\_topic/CMakeLists.txt 文件, 在文件末尾添加:

```

add_executable(velocity_publisher src/velocity_publisher.cpp)
target_link_libraries(velocity_publisher ${catkin_LIBRARIES})

add_executable(odom_subscriber src/odom_subscriber.cpp)
target_link_libraries(odom_subscriber ${catkin_LIBRARIES})

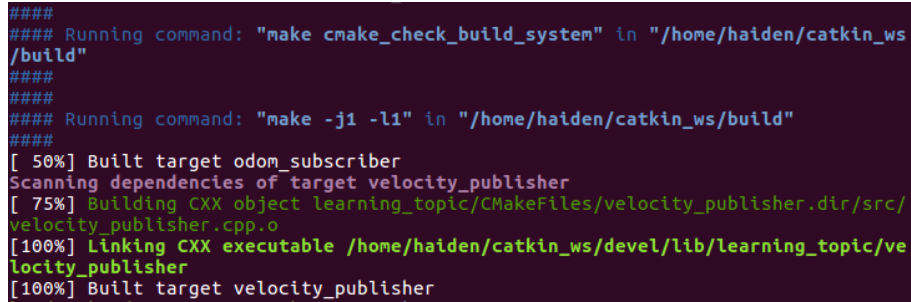
```

CMakeLists.txt 修改完成后，在工作空间的根路径下开始编译（**注意：每次修改代码后都需要重新编译，编译成功后才能运行**）：

```
cd ~/catkin_ws
```

```
catkin_make
```

如果编译成功，则可以在终端中看到如图 17 所示的输出信息。



```
#####  
#### Running command: "make cmake_check_build_system" in "/home/haiden/catkin_ws/build"  
#####  
####  
#### Running command: "make -j1 -l1" in "/home/haiden/catkin_ws/build"  
#####  
[ 50%] Built target odom_subscriber  
Scanning dependencies of target velocity_publisher  
[ 75%] Building CXX object learning_topic/CMakeFiles/velocity_publisher.dir/src/velocity_publisher.cpp.o  
[100%] Linking CXX executable /home/haiden/catkin_ws/devel/lib/learning_topic/velocity_publisher  
[100%] Built target velocity_publisher
```

图 17

#### 4.5 运行 Publisher 与 Subscriber

将提供的 **worlds** 文件夹复制到 **/home** 目录下（也即与 catkin\_ws、turtlebot 等文件夹同一级）。

在三个终端按顺序运行（**注意：如果不是使用本课程提供的虚拟机，请将第一句命令标红处修改为虚拟机的用户名**）：

```
roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/haiden/worlds/empty.world
```

```
roslaunch learning_topic velocity_publisher
```

```
roslaunch learning_topic odom_subscriber
```

第一句命令如果运行成功，系统自动打开 gazebo 仿真界面，如图 18 所示；第二句命令如果运行成功，gazebo 仿真环境中，Turtlebot2 开始移动；第三句命令如果运行成功，效果如图 19 所示。

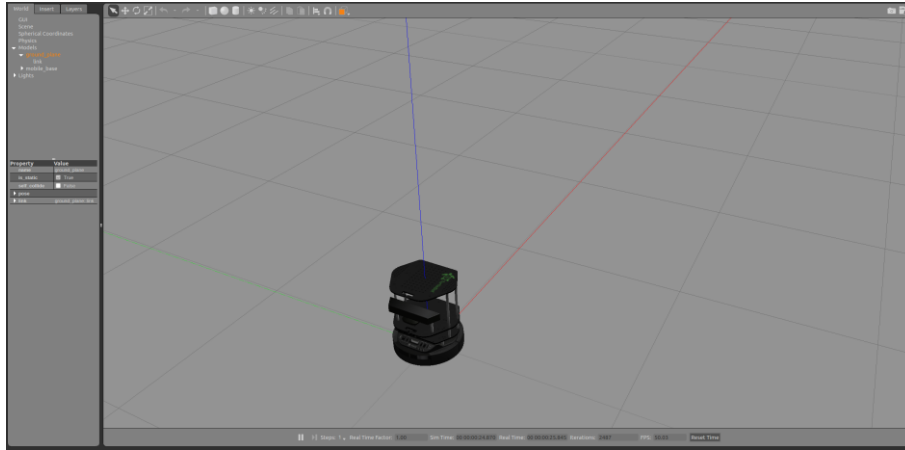


图 18

```
[INFO] [1627482056.555650485, 269.140000000]: Turtlebot2's odom: [x:3.02, y:3.94, theta:2.18]
[INFO] [1627482056.555743566, 269.140000000]: Turtlebot2's odom: [x:3.01, y:3.95, theta:2.18]
[INFO] [1627482056.555827139, 269.140000000]: Turtlebot2's odom: [x:3.01, y:3.95, theta:2.18]
[INFO] [1627482056.555907651, 269.140000000]: Turtlebot2's odom: [x:3.01, y:3.96, theta:2.19]
[INFO] [1627482056.599505166, 269.180000000]: Turtlebot2's odom: [x:3.01, y:3.96, theta:2.19]
[INFO] [1627482056.599588766, 269.180000000]: Turtlebot2's odom: [x:3.00, y:3.96, theta:2.19]
[INFO] [1627482056.599675457, 269.180000000]: Turtlebot2's odom: [x:3.00, y:3.97, theta:2.19]
[INFO] [1627482056.599747258, 269.180000000]: Turtlebot2's odom: [x:3.00, y:3.97, theta:2.19]
[INFO] [1627482056.599804377, 269.180000000]: Turtlebot2's odom: [x:2.99, y:3.98, theta:2.20]
[INFO] [1627482056.610491558, 269.200000000]: Turtlebot2's odom: [x:2.99, y:3.98, theta:2.20]
[INFO] [1627482056.653735509, 269.240000000]: Turtlebot2's odom: [x:2.99, y:3.98, theta:2.20]
[INFO] [1627482056.653845630, 269.240000000]: Turtlebot2's odom: [x:2.99, y:3.99, theta:2.20]
[INFO] [1627482056.653907715, 269.240000000]: Turtlebot2's odom: [x:2.98, y:3.99, theta:2.20]
[INFO] [1627482056.653964775, 269.240000000]: Turtlebot2's odom: [x:2.98, y:4.00, theta:2.21]
[INFO] [1627482056.697867130, 269.280000000]: Turtlebot2's odom: [x:2.98, y:4.00, theta:2.21]
[INFO] [1627482056.698021685, 269.280000000]: Turtlebot2's odom: [x:2.97, y:4.00, theta:2.21]
[INFO] [1627482056.698172096, 269.280000000]: Turtlebot2's odom: [x:2.97, y:4.01, theta:2.21]
[INFO] [1627482056.698294077, 269.280000000]: Turtlebot2's odom: [x:2.97, y:4.01, theta:2.21]
[INFO] [1627482056.742353851, 269.310000000]: Turtlebot2's odom: [x:2.96, y:4.02, theta:2.22]
[INFO] [1627482056.742601537, 269.310000000]: Turtlebot2's odom: [x:2.96, y:4.02, theta:2.22]
[INFO] [1627482056.743685067, 269.310000000]: Turtlebot2's odom: [x:2.96, y:4.02, theta:2.22]
[INFO] [1627482056.769462688, 269.320000000]: Turtlebot2's odom: [x:2.96, y:4.03, theta:2.22]
```

图 19

#### 4.6 launch 启动文件

到目前为止，每当需要运行一个 ROS 节点或工具时，都需要打开一个新的终端运行一个命令。当系统中的节点数量不断增加时，“每个节点一个终端”的模式会变得非常麻烦。

启动文件（Launch File）便是 ROS 中一种同时启动多个节点的途径，它还可以自动启动 ROS Master 节点管理器，并且可以实现每个节点的各种配置，为多个节点的操作提供很大便利。

创建 `learning_topic/launch/learning_topic.launch`，详细内容如图 20 所示（**注意：如果不是使用本课程提供的虚拟机，请将红框处修改为虚拟机的用户名**）。

```
<launch>

  <include file="$(find turtlebot_gazebo)/launch/turtlebot_world.launch">
    <arg name="world_file" value="/home/haiden/worlds/empty.world" />
  </include>

  <node name="velocity_publisher" pkg="learning_topic" type="velocity_publisher" output="screen" />

  <node name="odom_subscriber" pkg="learning_topic" type="odom_subscriber" output="screen" />

</launch>
```

图 20 learning\_topic.launch

**注意：创建或修改 launch 文件后不需要重新编译功能包，可以直接运行 launch 文件。**

打开一个终端运行：

```
roslaunch learning_topic learning_topic.launch
```

如果运行成功，效果与 4.5 相同，所有的输出均在一个终端中。

## 5 简单轨迹跟踪

### 5.1 功能包 show\_trajectory

功能包 show\_trajectory 提供了根据里程计消息可视化移动机器人移动轨迹的功能。**将提供的 show\_trajectory 文件夹复制到 catkin\_ws/src 目录下**（也即与 learning\_topic 文件夹同一级）。

### 5.2 功能包 trajectory\_following

功能包 trajectory\_following 是轨迹跟随功能包，已经实现了矩形轨迹跟随。**将提供的 trajectory\_following 文件夹复制到 catkin\_ws/src 目录下**（也即与 learning\_topic 文件夹同一级）。

下面以矩形轨迹跟随为例，简单介绍实现步骤。

#### 5.2.1 创建并编写代码文件

源码 trajectory\_following/src/rectangle.cpp 的核心内容是矩形轨迹跟随，可以将矩形轨迹拆分为 8 段：长边->转弯->短边->转弯->长边->转弯->短边->转弯。gazebo 仿真环境是理想的，假设 Turtlebot2 匀速运动，则对于每一段有：

$$\text{时间} = \text{距离} / \text{速度}$$
$$\text{循环次数} = \text{时间} * \text{循环频率}$$

如图 21 所示。

```

// 矩阵轨迹跟随
int count = 0;
while (ros::ok())
{
    int time = 0;
    if (count % 2 == 0) // 长边
    {
        double seconds = length / SPEED;
        time = int(seconds * RATE);
    }
    else // 短边
    {
        double seconds = width / SPEED;
        time = int(seconds * RATE);
    }

    // 前进
    ROS_INFO("Going Straight");
    for (int i = 0; i < time; i++)
    {
        vel_pub.publish(vel_msg_move);
        rate.sleep();
    }

    // 转弯
    ROS_INFO("Turning");
    for (int i = 0; i < 2 * RATE; i++)
    {
        vel_pub.publish(vel_msg_turn);
        rate.sleep();
    }

    count = (count + 1) % 4;
    if (count == 0)
    {
        ROS_INFO("Turtlebot2 should be close to the original starting position (but it's probably way off)");
    }
}

```

图 21 rectangle.cpp 核心

另外，获取参数的代码如图 22 所示，此处需要与 rectangle.launch 对应。

```

// 获取参数（与rectangle.launch文件中对应）
nh.getParam("/length", length);
nh.getParam("/width", width);

```

图 22 rectangle.cpp 参数

### 5.2.2 编译

打开 trajectory\_following/CMakeLists.txt 文件，在文件末尾添加：

```

add_executable(rectangle src/rectangle.cpp)
target_link_libraries(rectangle ${catkin_LIBRARIES})

```

**注意：**如果代码文件名不同，最简单的修改方式是将上方所有标红处修改为代码文件名（不包括后缀）。

CMakeLists.txt 修改完成后，在工作空间的根路径下开始编译（**注意：**每次修改代码后都需要重新编译，编译成功后才能运行）：

```
cd ~/catkin_ws
```

```
catkin_make
```

如果编译成功，则可以在终端中看到如图 23 所示的输出信息。



```

####
### Running command: "make cmake_check_build_system" in "/home/haiden/catkin_ws/build"
###
###
### Running command: "make -j1 -l1" in "/home/haiden/catkin_ws/build"
###
[ 25%] Built target odom_subscriber
[ 50%] Built target velocity_publisher
[ 75%] Built target show_trajectory
Scanning dependencies of target rectangle
[ 87%] Building CXX object trajectory_following/CMakeFiles/rectangle.dir/src/rectangle.cpp.o
[100%] Linking CXX executable /home/haiden/catkin_ws/devel/lib/trajectory_following/rectangle
[100%] Built target rectangle

```

图 23

### 5.2.3 创建并编写 launch 文件

创建 `trajectory_following/launch/rectangle.launch`，详细内容如图 24 所示，包括启动可视化实时轨迹的节点和矩形轨迹跟踪的节点，另外，还包括参数的设置（红框处），此处设置的矩形轨迹为 1.2m\*0.6m。

```

<launch>

  <arg name="length" default="1.2" />
  <arg name="width" default="0.6" />

  <include file="$(find show_trajectory)/launch/show_trajectory_rviz.launch" />

  <param name="length" type="double" value="$(arg length)" />
  <param name="width" type="double" value="$(arg width)" />

  <node name="rectangle" pkg="trajectory_following" type="rectangle" output="screen" />

</launch>

```

图 24 rectangle.launch

**注意：创建或修改 launch 文件后不需要重新编译功能包，可以直接运行 launch 文件。**

## 5.3 仿真

### 5.3.1 矩形轨迹跟随

打开一个新终端运行（**注意：如果不是使用本课程提供的虚拟机，请将标红处修改为虚拟机的用户名**）：

```
roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/home/haiden/worlds/empty.world
```

等待 gazebo 仿真世界加载完成后，打开一个新终端运行：

```
roslaunch trajectory_following rectangle.launch
```

如果运行成功，gazebo 仿真环境中，Turtlebot2 跟随矩形轨迹移动，在 rviz 界面中可视化轨迹，如图 25 所示。

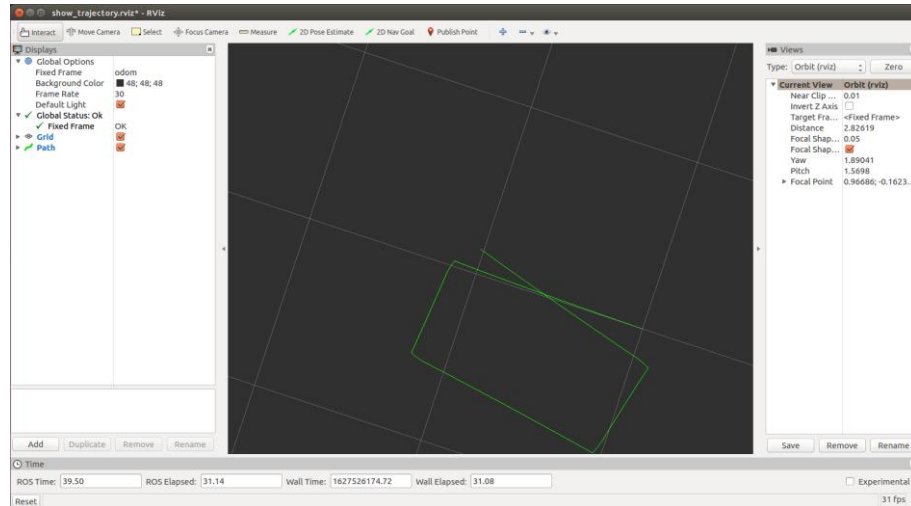


图 25 rviz 可视化轨迹

### 5.3.2 修改参数的方法

(1) 方法一：直接在 launch 文件中修改参数（见图 26 中红框处）。

```
<launch>
  <arg name="length" default="1.2" />
  <arg name="width" default="0.6" />
  <include file="$(find show_trajectory)/launch/show_trajectory_rviz.launch" />
  <param name="length" type="double" value="$(arg length)" />
  <param name="width" type="double" value="$(arg width)" />
  <node name="rectangle" pkg="trajectory_following" type="rectangle" output="screen" />
</launch>
```

图 26 rectangle.launch 参数修改

(2) 方法二：运行 roslaunch 时设置参数。

如果预设的矩形轨迹为 1.6m\*0.8m，等待 gazebo 仿真世界加载完成后，打开一个新终端运行：

```
roslaunch trajectory_following rectangle.launch length:=1.6 width:=0.8
```

尝试修改矩形轨迹的长宽，可视化 Turtlebot2 的跟随轨迹。

## 第 1 讲：ROS 基础与 Turtlebot2 基础拔高部分

### 6 拔高实验

**注意：**上述所有实验完成后，如果选择继续完成拔高实验，请在课堂时间内，选择其中 1 项进行实现即可。

- (1) 编程实现圆形轨迹跟随，可调整的参数至少包括圆形轨迹的半径，可视化 Turtlebot2 的跟随轨迹。
- (2) 编程实现五角星形轨迹跟随，可调整的参数至少包括五角星形轨迹的边长，可视化 Turtlebot2 的跟随轨迹。
- (3) 编程实现正多边形轨迹跟随，可调整的参数至少包括正多边形轨迹的边长，可视化 Turtlebot2 的跟随轨迹。
- (4) 编程实现其它轨迹跟随，可视化 Turtlebot2 的跟随轨迹。

**要求：**新建一个 .cpp 文件，命名为对应轨迹的英文名称（如圆形轨迹：circle.cpp），完成节点的编译，新建一个 launch 文件，命名为对应轨迹的英文名称（如圆形轨迹：circle.launch），使用该 launch 文件控制 turtlebot2 运动并且可视化 Turtlebot2 的跟随轨迹。

-----

### 7 实验提交形式

- (1) 如果课上已经完成，直接让助教或老师现场登记；
- (2) 如果课上来不及完成，课后将上述文件和视频整理打包到下面的邮箱里面：[1582536225@qq.com](mailto:1582536225@qq.com)，命名形式：姓名+学号+班级+组号+彭键清老师第 1 次作业\_基础+拔高（小组确定由谁来交，例如：1 班\_第 1 组\_张同学\_20222023\_彭键清老师第 1 次作业\_基础\_拔高）