2020 春交叉综合训练项目-Ros Programming

陈昱宏

2020.03

目录

实验目的

1. 掌握从零新建 ROS Package 的能力
2. 熟悉面向对象的 ROS 程序的编写
3. 熟练使用基本的 ROS 调试工具

实验环境

双系统 +Ubuntu16.0.4+ROS+gazebo

实验内容

命令行新建一个 package

先执行 roscore，

```
$ roscore
```

移动到工作空间下的 src 文件夹，

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

创建 package（1），

```
$ catkin_create_pkg husky_rcl_controller std_msgs roscpp nav_msgs
geometry_msgs
```

```
yuhung@yuhung-ThinkPad-T480:~/catkin_ws/src$ catkin_create_pkg husky_rcl_controller std_msgs roscpp n
av_msgs geometry_msgs
Created file husky_rcl_controller/package.xml
Created file husky_rcl_controller/CMakeLists.txt
Created folder husky_rcl_controller/include/husky_rcl_controller
Created folder husky_rcl_controller/src
Successfully created files in /home/yuhung/catkin_ws/src/husky_rcl_controller. Please adjust the valu
es in package.xml.
```

图 1: 创建名为 husky_rcl_controller 的包

重新编译工作空间，

```
$ catkin build husky_rcl_controller
```

更新环境变量，

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

搜索包并查找直接依赖项和全部依赖项（2），

```
$ rospack find husky_rcl_controller
$ rospack depends1 husky_rcl_controller
$ rospack depends husky_rcl_controller
```

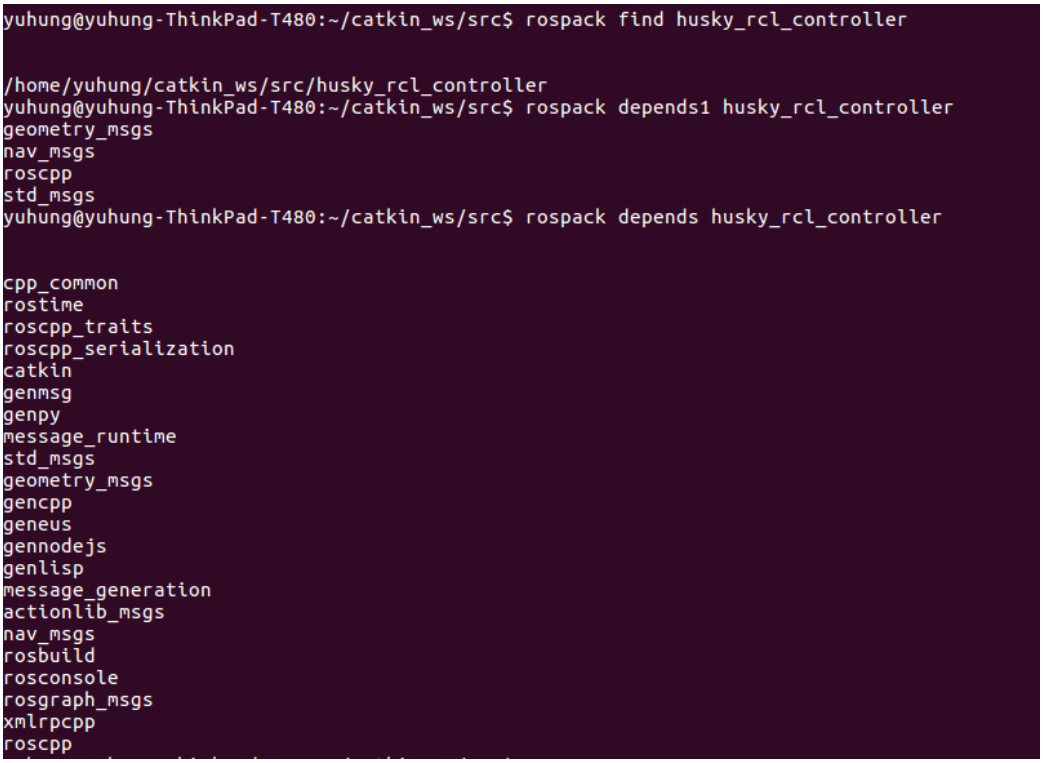


图 2: 执行 rospack 相关命令

面向对象的 ROS 编程

1. 查看/odometry/filtered 消息的接收格式：

先执行 husky_empty_world 仿真环境，

```
$ roslaunch husky_gazebo husky_empty_world.launch
```

利用 rostopic 命令行工具查看消息类型，

```
$ rostopic type /odometry/filtered
```

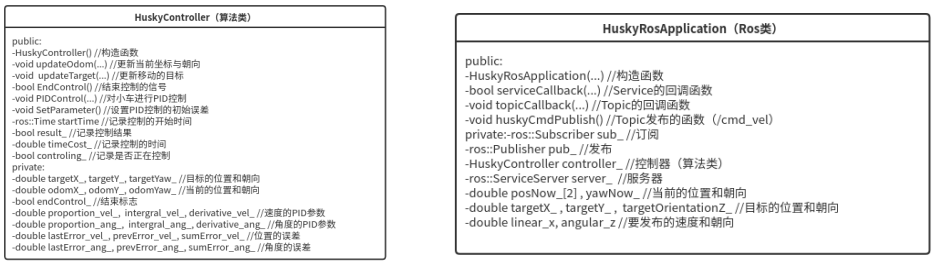
可以看到/odometry/filtered 的消息类型是 nav_msgs/Odometry (3)。

```
yuhung@yuhung-ThinkPad-T480:~$ rostopic type /odometry/filtered
nav_msgs/Odometry
```

图 3: 执行 rostopic 查看消息类型

2. 仿照 teleop_twist_keyboard 发布消息的模式，在算法类编写小车的控制算法，并在 ROS 类将控制的输出发送到话题:/cmd_vel：

这里我们创建了算法类和 Ros 类，其各自的结构图 (4) 如下：



(a) (b)

图 4: 算法类 (a) 和 Ros 类 (b) 的类结构图

控制器的部分采用 PID 控制，虽然只使用了 PD 的参数，但算法类中依然提供了 PID 的三个参数可以调整，具体的控制框图 (5) 如下：

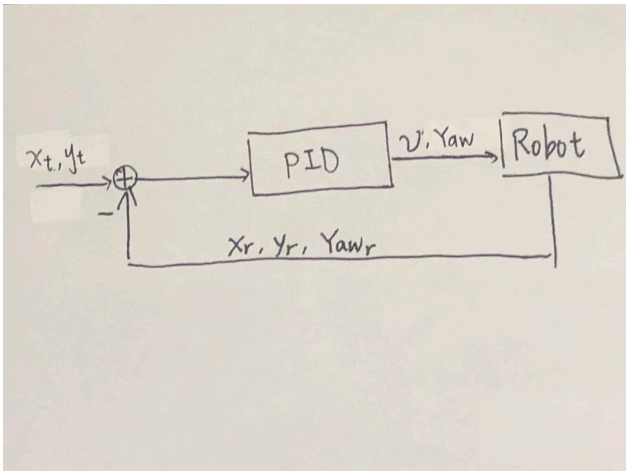


图 5: 系统控制框图

其中 x_t, y_t 代表目标的位置（由于没有做挑战内容，所以没有目标的转向），v, Yaw 代表发布的速度和角速度，x_r, y_r, Yaw_r 代表实际机器人的位置和转向，透过反馈控制，控制机器人到达指定位置。

检查 rqt_graph(6) 可以看到，我们自己设计的控制器，的确满足要求向机器人发送/cmd_vel 的消息并接收/odometry/filtered 的消息。

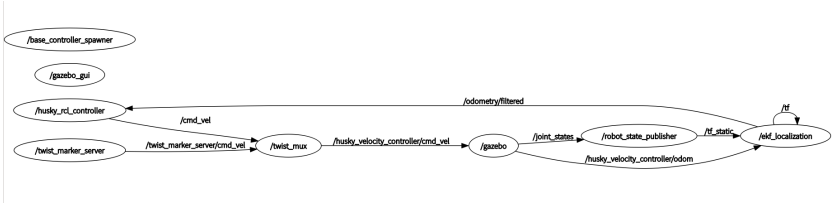


图 6: rqt_graph 结果

3. 创建 ROSServer：

按照前一节的创建 ROS package 的方法，创建一个名为 husky_rcl_controller_srv 的包，在 husky_rcl_controller_srv 的目录下，创建 srv 文件夹，

```
$ cd ~/catkin_ws/src/husky_rcl_controller_srv
$ mkdir srv
$ cd srv
```

创建.srv 文件，输入目标的消息格式 (7)，

```
$ vim husky_rcl_controller_srv.srv
```

```
float64 targetX
float64 targetY
float64 targetOrientationZ
---
bool result
float64 timeCost
```

图 7: Srv 文件中规定的图片格式

分别修改 package.xml(8) 和 CMakeLists.txt(9) 两个文件 [1]，

```
$ cd ..
$ gedit package.xml
$ gedit CmakeLists.txt
```

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

图 8: 修改 package.xml 的内容，添加图中两行

```
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  nav_msgs
  roscpp
  std_msgs
  message_generation
)
```

(a)

```
## Generate services in the 'srv' folder
 add_service_files(
   FILES
   husky.srv
 )
```

```
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES husky_rcl_controller_srv
   CATKIN_DEPENDS message_runtime
#  DEPENDS system_lib
)
```

(b)

图 9: 修改 CMakeLists.txt 的内容

更新环境变量，

```
$ cd source ~/catkin_ws/devel/setup.bash
```

检查 server 的消息格式 (10),

```
$ rossrv show husky_rcl_controller_srv
```

```
yuhung@yuhung-ThinkPad-T480:~/catkin_ws/src/husky_rcl_controller_srv/srv$ rossrv show husky_rcl_contro
ller_srv
[husky_rcl_controller_srv/husky_rcl_controller_srv]:
float64 targetX
float64 targetY
float64 targetOrientationZ
---
bool result
float64 timeCost
```

图 10: 检查 server 消息格式

4. 使用命令行语句，手动输入参数，调用编写好的 Service 接口，完成让小车移动到指定位置的任务：

首先我们要修改 husky_rcl_controller 下的 CMakeLists.txt 和 package.html 文件，找到对应内容并修改成以下形式，

CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8.3)
project(husky_rcl_controller)
find_package(catkin REQUIRED COMPONENTS
    geometry_msgs
    nav_msgs
    roscpp
    std_msgs
    husky_rcl_controller_srv
)
catkin_package(
    INCLUDE_DIRS include
    CATKIN_DEPENDS geometry_msgs nav_msgs roscpp
     std_msgs husky_rcl_controller_srv
```

```
)
include_directories(
    include
    ${catkin_INCLUDE_DIRS}
)
add_executable(${PROJECT_NAME}
    src/HuskyRosApplication.cpp
    src/HuskyController.cpp
    src/main.cpp
)
target_link_libraries(husky_rcl_controller
    ${catkin_LIBRARIES}
)
```

package.xml 添加以下几行:

```
<build_depend>husky_rcl_controller_srv</build_depend>
<build_export_depend>husky_rcl_controller_srv</build_export_depend>
<exec_depend>husky_rcl_controller_srv</exec_depend>
```

对 husky_rcl_controller 包进行编译,

```
$ catkin build husky_rcl_controller
```

接下来开启仿真环境并运行自定义的 Ros 节点,

```
$ roslaunch husky_gazebo husky_empty_world.launch
$ rosrun husky_rcl_controller husky_rcl_controller
```

利用 rosservice 发布目标命令, 并观察结果 (11),

```
$ rosservice call husky_rcl_controller/husky_controller_server
1 1 0
```

```
yuhung@yuhung-ThinkPad-T480:~/catkin_ws$ rosservice call husky_rcl_controller/husky_controller_server
1 1 0
result: False
timeCost: 10.0
yuhung@yuhung-ThinkPad-T480:~/catkin_ws$ rosservice call husky_rcl_controller/husky_controller_server
1 1 0
result: False
timeCost: 10.0
yuhung@yuhung-ThinkPad-T480:~/catkin_ws$ rosservice call husky_rcl_controller/husky_controller_server
1 1 0
result: True
timeCost: 2.0
^Cyuhung@yuhung-ThinkPad-T480:~/catkin_ws$ rosrun husky_rcl_controller husky_rcl_controller
[ INFO] [1584098216.925667214, 76.927000000]: Result: Failed
Time Cost: 10.000000
[ INFO] [1584098237.145921060, 93.666000000]: Result: Failed
Time Cost: 10.000000
[ INFO] [1584098242.787910022, 98.341000000]: Result: Finished
Time Cost: 2.000000
```

图 11: 执行三次目标命令分别的结果

利用 rqt_plot 显示控制结果 (12),

```
$ rqt_plot
```

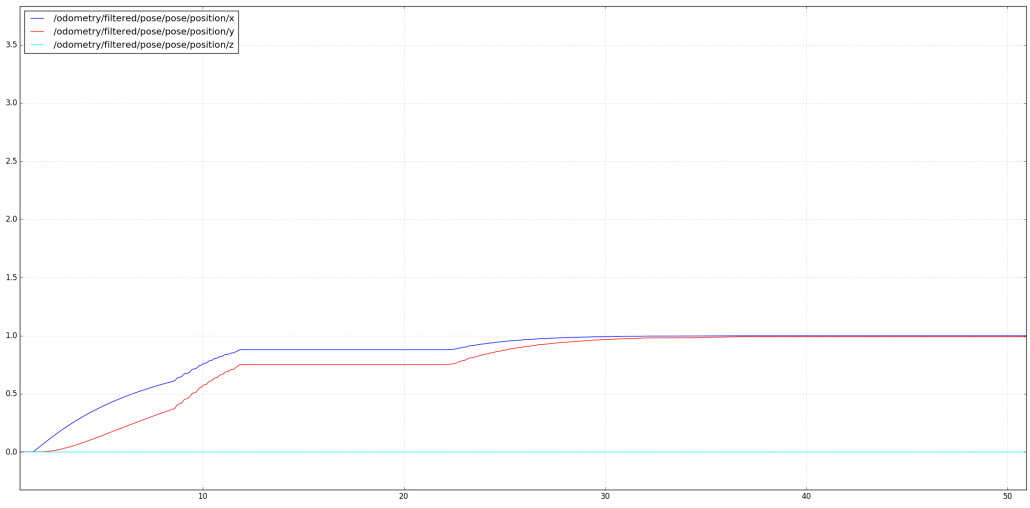


图 12: rqt_plot 的控制效果曲线

对于 Ros 消息机制的理解

在 Ros 中, 提供了两种不同的消息机制, 分别是 Topic 和 Service, Topic 是透过发布者 (Publisher) 和订阅者 (Subscriber) 来进行通信, 任何节点都可以是一个 Topic 的发送者或订阅者所以 Topic 的消息机制是一个多对多

的机制，在 Publisher 中，只要根据指定 Topic 的数据格式定义要发送的数据，就可以发送一个 Topic。而 Subscriber 是透过回调函数 (Callback) 接收想要订阅的 Topic，将得到的数据再进行处理。

Service 机制则是一个一对一的消息机制，可以由用户定义发布的消息类型和回复的消息类型，利用客户端 (Client) 接收消息传送给服务端 (Server)，并由服务端经过自己内部的运算 (回调函数 Callback)，返回结果给客户端。

在本次的实验中，我们综合利用了两种消息机制，透过 Service 接收控制台的目标命令，在利用 Topic 机制来接收当前位置和控制结果 (速度和角速度)。

参考文献

[1]https://blog.csdn.net/u010647296/article/details/83143946?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task

[2]https://www.jianshu.com/p/ed1d9a490bd1

[3]https://github.com/leggedrobotics/ros_best_practices

[4]https://blog.csdn.net/lin5103151/article/details/89893380