# 实验一：ROS基础知识实验

**一 ROS基础知识**

ROS文件系统介绍

1.功能包集stack

ROS软件包集合，像Navigation Stack，属于导航软件包集合，包含了与导航有关的软件包，例如地图软件包，路径规划软件包，TF坐标软件包，move\_base移动控制软件包等，一整个stack下载后可以任意使用其中的软件包，每个软件包都实现其中一个或者好几个小功能，多个软件包组成集合实现一个完整的大的功能

http://wiki.ros.org/navigation

2.软件包package

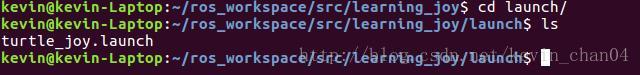
ROS程序的单元，每个软件包都包含程序库，可执行文件，脚本。图中列出的learning\_joy软件包，包含编译配置文件CMakeLists.txt，include头文件库，launch启动脚本文件夹，package.xml软件包概况描述文件，src程序源码库

软件包文件详解：

1）编译配置文件CMakeLists.txt， 使用cmake编译时要写的

2）include头文件库，如果只是使用ROS中附带有的头文件，include这个文件夹可能会是空的，但是系统会自动创建，当以后需要用到自己的头文件或者添加别人的头文件时，把头文件添加到该文件夹里便可正确使用

3）launch启动脚本文件夹，里边包含了后缀名为launch的启动脚本，功能是一次性启动多个节点



4）清单文件package.xml，包含软件包的相关信息的描述，包括版本号，开发者信息，各软件包之间的依赖关系

5）src程序源码库，存放编写的cpp或者python源码文件

ROS基本框架

1.节点node

节点是ROS机器人系统里任何操作的单元，可以理解为每一个节点可以实现一个小功能，一个机器人工程就是由多个节点组成的，节点与节点之间可以相互通信，相互协调完成工作，比如人手臂的运动，运动指令从大脑发出，经过肩膀，手肘，手腕，最后再到手指，触感的信息则是反过来传递，每个关节理解成一个节点，它们之间可能是相互传递消息，也可能是一个发送消息，一个接收消息；通常，我们在编程的时候就是编写一个个节点。

2.话题topic

话题是ROS里的一个通信方法，topic里包含了消息的名称，类型，内容，节点可以发布或者订阅某个或某几个话题，以此达到传递的目的；话题有两个重要的性质：a.话题的发布频率；b.话题的消息类型

3.发布者和订阅者publisher & subscriber

ROS里的节点node可能是发布者或者是订阅者，甚至两者都兼备，发布者就是发布topic的节点，订阅者就是订阅topic的节点，发布者以一定的消息类型格式在ROS服务器上发布消息，订阅者则根据话题名称在服务器上订阅消息，消息类型在ROSwiki上有详细定义

http://wiki.ros.org/std\_msgs

还有一些常用软件包内需要的消息类型，如：

common\_msgs 查看各类消息类型

geometry\_msgs 运动几何学相关的消息类型，包括速度，加速度等

sensor\_msgs 传感器相关的消息类型，包括激光雷达，点云，IMU惯性测量单元，JoySticks遥控手柄，图像等

4.理解ROS的通信流程

1.master 节点,管理所有的节点,每个节点都需要先向 master 节点注册, master 节点会让发布者和订阅

者进行匹配；所以每次启动ROS时，都需要先打开master节点

kevin@kevin-Laptop:~$ roscore

2.发布者和订阅者之间通过话题来通信时，信息不通过 master

3.发布者发布一个话题,订阅者进行订阅

4.流程:

1 )订阅者发送建立链接请求

2 )发布者进行反馈，确认链接

3 )订阅者请求 topic 的数据

4 )发布者发送 topic 数据

**创建和编译 ROS 软件包**

1.创建软件包

1）创建工作空间

在编写任何ROS程序之前，都应该先建立一个工作空间workspace，因为使用catkin环境，所以一般网上很多教程都会将工作空间命名为catkin\_ws，但其实可以任意命名，工作空间就是一个在~/home目录下的文件夹。工作空间下是存放软件包的，所以一般而言，工作空间不需要创建太多，除非某个工程特别庞大，才需要为它独立创建工作空间

创建工作空间目录：

查看文件夹目录：

编译软件包命令：

即使没有源码文件，工作空间也是可以通过编译的，编译命令要在工作空间文件夹目录下执行，系统会自动生成build文件夹和devel文件夹，devel文件夹内的文件与引导路径有关系，没有source配置正确的环境变量路径，即使catkin\_make编译通过，ROS也无法正确识别通过编译的节点

在工作空间目录下执行source命令：~/catkin\_ws$ source devel/setup.bash

但这个命令只能在该终端内source到正确的环境变量，每次新打开一个终端，仍需要再次配置，可以修改整个环境：

$ gedit ~/.bashrc

把编辑文本拉到最后，添加：

source ~/catkin\_ws/devel/setup.bash

就是添加工作空间下的/devel/setup.bash文件的路径，让终端每次打开时，都会自动执行source命令

配置完成后，按ctrl+alt+T打开终端，如果配置不正确，终端会提示找不到路径

2）创建软件包

软件包一般在工作空间的src文件夹下创建，否在ROS可能无法定位你的软件包，命名要避免与ROS的操作指令重复，不能以数字或者符号开头，并且不允许大写字母

创建软件包命令：

格式： catkin\_create\_pkg pkg\_name pkg\_depends

例子： catkin\_create\_pkg testing\_pkg roscpp rospy std\_msgs

1.pkg\_name是对软件包的命名；

2.pkg\_depends是软件包的依赖关系，如roscpp，rospy，std\_msgs；软件包的依赖关系类似依赖某个源代码或者头文件

3.依赖关系分为一级依赖和间接依赖，一级依赖是我们在创建软件包过程中添加的依赖项，间接依赖是各依赖项之间的关系，

$ rospack depends1 pkg\_name 查看一级依赖关系

$ rospack depends pkg\_name 查看所有的依赖关系

3）删除软件包

软件包删除命令：$ rm -rf 文件夹名称

再重新编译

2.编写测试代码步骤

1）先进入软件包目录下的src文件夹 $ cd catkin\_ws/src/testing\_pkg/src/

2）打开编辑器编写cpp或者python源码

$ gedit talker.cpp $ gedit listener.cpp

如果没有gedit编辑器，系统会提示安装gedit $ sudo apt-get install gedit

3）修改编译配置文件

4）返回工作空间目录，编译代码 $ catkin\_make

3.编写发布者代码

1）简单的发布者代码，发布者代码基本都是这个框架



4.编写订阅者代码

1）简单的订阅者代码



5.修改CMakeLists.txt文件



8）注意

launch 文件中,启动节点的时候给 node 重新指定了名称,会覆盖掉在源码文件中通过 ros::init 赋予节点的名称，所以在使用launch启动时要留意被修改的节点名

如果 output 没有定义到 screen ,节点的输出信息会在 log 日志文件下保存,查看方式:

$ ~/.ros/log/run\_id/node\_name-number-stour.log

拓展：<https://www.cnblogs.com/zjiaxing/p/5542614.html>

8.通过launch文件启动节点

1）

格式：$ roslaunch pkg\_name launch\_name

例子：$ roslaunch testing\_pkg testing\_pkg.launch

ROS基础工具

1.rqt\_graph

1）通过直观的图谱来查看当前正在运行的节点之间的关系

2）$ rosrun rqt\_graph rqt\_graph

3）graph图谱会列出节点，话题等信息

2.rqt\_robot\_steering

1）rqt\_robot\_steering是与机器人运动控制的密切相关的ROS工具，它相当于一个节点，可以发布/cmd\_vel类型的消息，/cmd\_vel类消息包括了机器人直线运动和转向运动的控制指令，通过滑动条可以十分方便地控制机器人运动

2）$ rosrun rqt\_robot\_steering rqt\_robot\_steering

3.rviz

1）rviz是一款2D&3D的数据可视化软件，提供了丰富的ROS接口，可以建立机器人运动模型，激光雷达地图等，并且进行仿真验证

2）rosrun rviz rviz

4.rqt\_logger\_level

1）ROS的一个log日志查看器，用于检查每个节点的日志信息，比如说什么时候哪个节点作出了更改，都会记录在log日志下

2）$ rosrun rqt\_logger\_level rqt\_logger\_level

3）log消息等级划分，从Debug到Fatal越来越严重

5.rqt\_console

1）用来查看节点输出信息，因为有很多消息并未通过ROS\_INFO()函数在终端显示出来，可以通过console查看是否有正确输出

2）rosrun rqt\_console rqt\_console

6.ROS文件系统工具

1）rospack 获取软件包相关信息,类似一个 find 的命令,只需要知道软件包的名称即可

1.rospack help 查看 rospack 相关功能的命令

2.rospack list 列出当前系统中安装了的软件包名称和路径信息

3.rospack list-names 只查看软件包名称

4.rospack list-names | wc -l 统计有多少个软件包，| 是一个符号,表示上一个命令的结果是下一个命令的输入

5.rospack find [package\_name] 查询软件包的路径信息

2）roscd 软件包名

直接切换工作目录到某个软件包中

3）rosls 软件包名

直接罗列该软件包的目录结构

4）rosed 软件包 可编辑文件名

直接编辑某个软件包的可编辑文件

5）roscp 软件包名 需要复制的文件名

直接把某个软件包下的文件复制到当前的命令行目录下

**实验二：ROS小海龟历程初探**

1. 启动ROS

2.启动小海龟节点

重新打开一个终端，输入



rosrun：启动一个新节点;

turtlesim：功能包

turtlesim\_node：功能包下的一个节点

如下图所示出来一个小海龟的界面

3.启动键盘控制节点

重新打开一个终端，输入

rosrun turtlesim turtle\_teleop\_key

如下图所示，将鼠标放在这个终端上，就可以用键盘的上下左右控制小海龟移动了。

4.可视化工具rqt

（1） rqt\_graph

可以看到当前正在运行的节点关系图，如下：

1. rqt\_plot

可以用来查看数据输出波形，在 topic 栏目中输入 / ，系统会自动弹出各种信息

我们选择小海的 /turtle1/pose/x, 点击+， /turtle1/pose/y，点击+，再用键盘控制小海龟移动

5.使用ROS节点（rosnode   <...>）

rosnode分别有以下几个用法：

rosnode list：获取正在运行的节点

rosnode info /turtlesim   输出此节点信息

rosnode ping  /turtlesim

rosnode machine  /turtlesim

rosnode kill  /turtlesim   杀死/结束此节点

rosnode cleanup  /turtlesim

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

rosnode list：获取正在运行的节点

如下图所示，和上面一样，一共3个节点：

/rosout：这个节点始终是有的，随着roscore的运行而运行

/teleop\_turtle：键盘控制节点

/turtlesim：小海龟节点

rosnode info node1：查看节点“node1”的信息

如输入：



6.使用主题与节点交互（rostopic）

常用rostopic有以下几个用法

rostopic list 查看主题清单

rostopic echo TOPIC 将主题发布的消息输出到屏幕

rostopic type TOPIC 输出主题类型

rostopic pub TOPIC  将数据发布到主题

（1）查看主题清单，终端输入：



可以看到正在运行的 /turtlesim所有的主题如下图：

（2）现在查看上面的两个节点，分别运行：





键盘节点的发布者：Publications 中有一个主题：/turtle1/cmd\_vel [geometry\_msgs/Twist]

小海龟节点的订阅者：Subscriptions 中有一个主题：/turtle1/cmd\_vel [geometry\_msgs/Twist]

这说明第一个节点发布了 /turtle1/cmd\_vel [geometry\_msgs/Twist] 这个主题，第二个节点订阅了 /turtle1/cmd\_vel [geometry\_msgs/Twist] 这个主题。

（3）现在我们用 echo 来查看节点通过话题发出的消息，终端输入：



然后我们再用键盘控制小海龟移动，就可以看到如下输出，包含线速度linear和角速度anular：

（4）使用pub发布主题

为了发布话题，我们要知道我们需要发送什么类型的消息

终端输入：



会显示消息类型为：geometry\_msgs/Twist

现在我们重新打开一个小乌龟节点，并且不开启键盘控制哪个节点，然后用pub直接发布主题，输入如下命令：

rostopic pub /turtle1/cmd\_vel geometry\_msgs/Twist "linear:

x: 5.0

y: 5.0

z: 0.0

angular:

x: 0.0

y: 0.0

z: 1.0"

这时可以看到小海龟就可以按照我们发布的消息来运动了，如下图

当然，如果想让小海龟不停的运动，可用如下命令：

rostopic pub /turtle1/cmd\_vel geometry\_msgs/Twist -r 1 "linear:

x: 2.0

y: 0.0

z: 0.0

angular:

x: 0.0

y: 0.0

z: 1.0"

可看到小海龟一直在转圈

6.如何使用服务（rosservice）

（1）查看正在运行的 /turtlesim 节点服务清单，终端输入：

rosservice list

显示如下各种服务：

（2）查看服务

如要查看/ spawn 服务，终端输入：



Node: /turtlesim  指这个服务属于哪个节点的

URI: rosrpc://Linux:35059

Type: turtlesim/Spawn 服务类型

Args: x y theta name 服务参数

（3）调用服务

# 实验三：创建工作空间、创建ROS消息和ROS服务

**创建工作空间**

1.创建工作空间 catkin\_ws

创建src文件，放置功能包源码：



进入src文件夹：



初始化文件夹：



这样就在src文件中创建了一个 CMakeLists.txt 的文件，目的是告诉系统，这个是ROS的工作空间。

2.编译工作空间 catkin\_make

所有编译工作都要在catkin\_ws文件夹下编译：

 编译，编译完成后，会发现catkin\_ws中多了两个文件 build 和 devel

 3.设置环境变量

我们需要把我们工作空间的环境变量设置到bash中。



 让上面的配置在当前的终端生效：

 我们用 vi 编辑器打开   ~/.bashrc 文件，就可以看到最后两行，第一行是我们第1篇添加的ros系统的环境变量，第2行是我们创建的catkin\_ws工作空间的环境变量：

如果想要查看环境变量：

终端会输出：

/home/wdd/catkin\_ws/src : /opt/ros/kinetic/share 第一个是我们刚才创建的catkin\_ws的，第二个是ros系统的。

4.创建功能包

在src中创建：

功能包格式：

catkin\_create\_pkg   package\_name   depend1 depend2 depend2

package\_name：功能表名称

depend1、2、3：依赖项

创建功能包：



std\_msgs：包含常见消息类型

roscpp：使用C++实现ROS各种功能

rospy：使用python实现ROS各种功能

5.编译功能包



显示如下说明编译成功：

**创建ROS消息和ROS服务**

1. 创建一个自定义消息 msg

（1）在 catkin\_ws/src/learning\_communication 文件中，创建一个文件夹，命名为 msg，在这个msg文件夹里面，创建一个 .msg 文件,命名为msg.msg

（2）打开msg.msg，复制如下内容，当然，你可以仿造上面的形式多增加几行以得到更为复杂的消息

int64 num

string first\_name

string last\_name

uint8 age

uint32 score

（3）编辑 package.xml

目的是确保msg文件被转换成为C++，Python和其他语言的源代码：

打开 catkin\_ws/src/learning\_communication 下的 package.xml

删除：

  <!--    <build\_depend>message\_generation</build\_depend> -->

  <!--   <exec\_depend>message\_runtime</exec\_depend> -->

这两行的  <!-- 、--> ，如下图所示：

（4）编辑CMakeLists.txt

设置find\_packag函数：

增加对 message\_generation 的依赖，这样就可以生成消息了

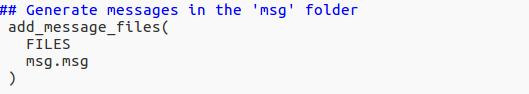
打开 catkin\_ws/src/learning\_communication 下的CMakeLists.txt

在find\_package（）函数中加入  message\_generation

设置运行依赖：

找到如下行，并加入 message\_runtime ，如下图：

加入新消息名称：



1. 编译



（6）使用 rosmsg

显示如下：

消息类型包含两部分：

* learning\_communication/msg-- 消息所在的位置
* int64 num

string first\_name

string last\_name

uint8 age

uint32 score                               -- 消息名.

2. 创建一个服务 srv

（1）在 catkin\_ws/src/learning\_communication 文件中，创建一个文件夹，命名为 srv，在这个srv文件夹里面，创建一个 .srv 文件,命名为 AddTwoInts.srv

（2）打开 AddTwoInts.srv，复制如下内容

srv文件分为请求和响应两部分，由'---'分隔，其中 a 和 b 是请求, 而sum 是响应。

int64 a

int64 b

---

int64 sum

（3）编辑 package.xml

       同创建msg

（4）编辑CMakeLists.txt

设置find\_packag函数：    同创建msg

设置运行依赖：同创建msg

加入新消息名称：

msg和srv都需要的步骤

去掉注释并附加上所有你消息文件所依赖的那些含有.msg文件的package

（这个例子是依赖std\_msgs,不要添加roscpp,rospy)，结果如下:

（5）编译



（6）使用 rossrv



# 实验四：话题（topic）编程与服务（service）编程

# 话题（topic）编程

# 本次创建两个节点，一个收，一个发。

# 先在 catkin\_ws/src/learning\_communication/src 功能包下建两个 .cpp 文件，分别命名talker.cpp 、listener.cpp

# （1）编写发布器节点

# （2）编写订阅器节点

# /\*\*

# \* 该例程将订阅chatter话题，消息类型String

# \*/

# 

# #include "ros/ros.h"

# #include "std\_msgs/String.h"

# （3）编译节点

# 在 learning\_communication 文件夹里面的 CMakeLists.txt 文件末尾加入几条语句:

# include\_directories(include ${catkin\_INCLUDE\_DIRS})

# 

# add\_executable(talker src/talker.cpp)

# target\_link\_libraries(talker ${catkin\_LIBRARIES})

# 

# add\_executable(listener src/listener.cpp)

# target\_link\_libraries(listener ${catkin\_LIBRARIES})

# 这会生成两个可执行文件, talker 和 listener,具体在：~/catkin\_ws/devel/lib/learning\_communication 中

# 编译节点：

# 

# 

# （4）运行节点

# 启动ROS

# 在不同终端中输入：

# 

# 

# 服务（service）编程

# 本次我们将创建一个简单的service节点("add\_two\_ints\_server")，该节点将接收到两个整形数字，并返回它们的和。

# 先在 catkin\_ws/src/learning\_communication/src 功能包下建两个 .cpp 文件，分别命名add\_two\_ints\_server.cpp 、add\_two\_ints\_client.cpp

# （1）编写Service节点

# （2）编写Client节点

# 打开add\_two\_ints\_client.cpp文件，复制如下内容

# （4）运行节点

# 启动ROS

# 

# 运行Service



# 运行Client

# 实验五：常用工具掌握与URDF文件学习

1.启动文件 roslaunch

roslaunch可以用来启动定义在launch文件中的多个节点。

用法：

例如：

在 ~/catkin\_ws/src/learning\_communication 目录下，建立一个名为 launch 的文件夹

然后在这个文件夹里面建一个名为 learning\_communication.launch 的文件

打开文件输入：

<launch> ：以launch标签开头以表明这是一个launch文件

<node pkg="learning\_communication" name="talker1" type="talker"/>

pkg：节点所在功能包

type：原节点名称

name：节点运行时的新名称

</launch>：launch文件的结束标签

这样的话，我们就可以用一行指令同时启动两个节点了：



启动后，我们可以用 rosnode list  或者 rqt\_graph 来查看正在运行的两个节点

但这里并没有看到 talker 和 listener 输出的信息，这是因为程序用了 ROS\_INFO 输出消息的原因

我们可以用：



来查看消息：

2.QT工具箱

前面都有用到，这就不再介绍了

（1）rqt\_console 日志输出工具箱

（1）rqt\_graph 计算图可视化工具

（1）rqt\_plot 计算图可视化工具

（1）rqt\_reconfigure 参数动态配置工具

3.Rviz可视化平台

（1）启动rqt\_rviz



显示如下

4.Gazebo物理仿真环境

（1）启动Gazebo



# ---URDF文件学习

1.URDF定义

标准机器人描述（URDF）是种用于描述机器人机器结构、关节、自由度等的XML格式文件。

2.创建一个URDF文件

下面创建一个简单的四轮驱动小车模型

（1）首先创建一个机器人建模功能包：



 返回 catkin\_ws 编译功能包

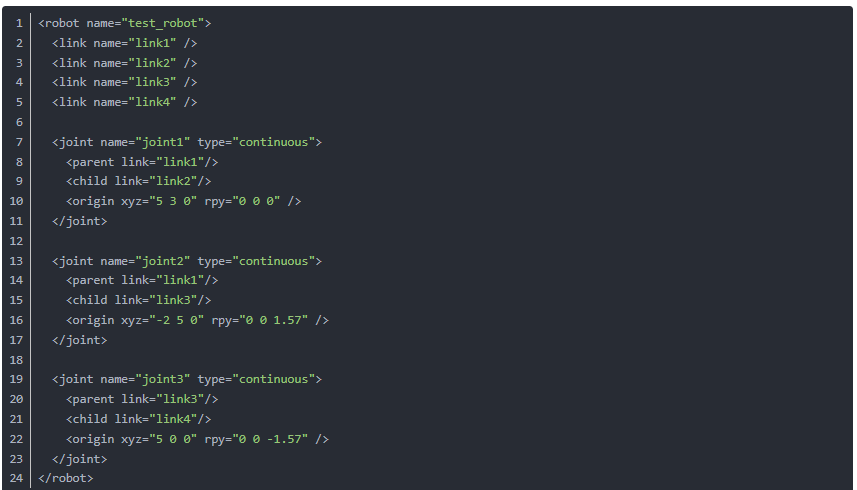
 （2）进入 testbot\_description 文件夹，建一个名为 urdf 的文件夹，在urdf文件夹内，建名为： my\_robot.urdf 的文件

3.创建树结构



4.添加关节空间位置

打开 my\_robot.urdf增加如下内容：



5.添加关节的旋转轴

 打开 my\_robot.urdf 增加如下内容：

<robot name="test\_robot">

<link name="link1" />

<link name="link2" />

<link name="link3" />

<link name="link4" />

<joint name="joint1" type="continuous">

<parent link="link1"/>

<child link="link2"/>

<origin xyz="5 3 0" rpy="0 0 0" />

<axis xyz="-0.9 0.15 0" />

</joint>

<joint name="joint2" type="continuous">

<parent link="link1"/>

<child link="link3"/>

<origin xyz="-2 5 0" rpy="0 0 1.57" />

<axis xyz="-0.707 0.707 0" />

</joint>

<joint name="joint3" type="continuous">

<parent link="link3"/>

<child link="link4"/>

<origin xyz="5 0 0" rpy="0 0 -1.57" />

<axis xyz="0.707 -0.707 0" />

</joint>

</robot>

只是加了3行

<axis xyz="0 1 0" />   表示joint绕y轴正方向旋转

 <axis xyz="-0.707 0.707 0" />  表示joint1围绕以x、y轴旋转

6.如果你想以图形方式来查看

进入~/catkin\_ws/src/testbot\_description/urdf

在此目录下会生成一个PDF文件，打开如下：

# 实验六：ROS 运动规划 (Motion Planning): MoveIt! 与 OMPL

1. 基础概念   
   首先，我们要了解一些基础的概念，了解各个名词的意义和区别。   
   1.1. 运动规划 (Motion Planning)   
   我们这里讲的运动规划，有别于轨迹规划 (Path Planning)。一般来说，path planning用于无人车/无人机领域，而motion planning主要用于机械臂，类人机器人领域。当然了，这两者没有本质的区别，理论上说MoveIt!和OMPL同样可以用于无人车无人机的规划，但不免有些杀鸡用牛刀的感觉。两者规划的空间维度不同，导致他们的难易程度不同。举例说明，如果不考虑速度加速度，只考虑位置的话，无人车轨迹规划维度是3 (x,y,和角度), 无人机是6 (x,y,z,和另外3个量确定空间的旋转角度)。确定3D空间的一个姿势(pose)需要6个变量，而对于关节数大于6的机械臂结构，它的规划空间维度就大于6，成为冗余系统(redundant system)，从而使规划问题变得更为复杂。所谓冗余系统，就是说，存在多种关节角度配置能够使得终端达到相同的位姿，存在无数的解。这是达到的最终姿势有无数个解，那么如何到达这个最终姿势，整个运动的轨迹，更是存在无数个解。   
   既然存在无数的解，那么问题来了。很明显，存在两种不同的方向，一种是找到最好的那个解，另一种是快速的找到一个有效的解。前者，大部分算法使用最优规划 (Optimization-based Planning)，后者使用采样规划 (Sampling-based Planning)。具体的区别和算法，不在这里赘述。

1.2. 开源运动规划库 ([OMPL](http://ompl.kavrakilab.org/)).

接上文，而OMPL (Open Motion Planning Library)， 开源运动规划库，就是一个运动规划的C++库，其包含了很多运动规划领域的前沿算法。虽然OMPL里面提到了最优规划，但总体来说OMPL还是一个采样规划算法库。而采样规划算法中，最出名的莫过于 Rapidly-exploring Random Trees ([RRT](http://ieeexplore.ieee.org/ielx5/6794/18246/00844730.pdf?tp=&arnumber=844730&isnumber=18246)) 和 Probabilistic Roadmap ([PRM](http://ieeexplore.ieee.org/ielx4/70/11078/00508439.pdf?tp=&arnumber=508439&isnumber=11078&tag=1))了, 当然，这两个是比较老的，还有很多[其他](http://ompl.kavrakilab.org/planners.html)新算法。

OMPL能做什么？ 简单说，就是提供一个运动轨迹。给定一个机器人结构(假设有N个关节)，给定一个目标(比如终端移到xyz)，给定一个环境，那么OMPL会提供给你一个轨迹，包含M个数组，每一个数组长度是N，也就是一个完整的关节位置。沿着这个轨迹依次移动关节，就可以最终把终端移到xyz，当然，这个轨迹应当不与环境中的任何障碍发生碰撞。

为什么用OMPL？ 运动规划的软件库和算法有很多，而OMPL由于其模块化的设计和稳定的更新，成为最流行的规划软件库之一。很多新算法都在OMPL开发。很多其他软件（包括ROS/MoveIt）都使用OMPL做运动规划。

1.3. 逆运动学 (Inverse Kinematics)

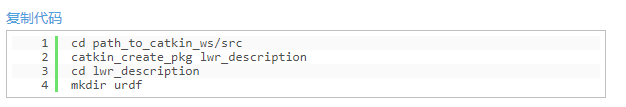
什么是逆运动学(IK)？简单说，就是把终端位姿变成关节角度，q=IK(p)。p是终端位姿(xyz)，q是关节角度。

为什么要用IK？OMPL是采样算法，也就是要在关节空间采样。 这与无人车的规划有一个最明显的区别，无人车的目标就是在采样空间， e.g. 目标是(x,y), 采样空间也是(x,y). 但是对于机械臂，目标是终端空间位置(xyz), 但采样空间却是关节空间(q0,q1,...qN)。有了IK之后，我们就可以把三维空间的目标p转化为关节空间的目标q。那么这样就会让采样算法能算的更快，具体方法不赘述，这样的算法有RRT-Connect，BKPIECE等等双向采样算法。

1.4. [MoveIt!](http://moveit.ros.org/)   
问：我不想看也看不懂OMPL和各种算法，但是我想让机械臂动起来，怎么办？   
答：那这正是MoveIt!的设计初衷。Move It！让它动起来！   
OMPL是运动规划的“规划”部分，而MoveIt!是OMPL的ROS接口。当然这不完全准确，OMPL有单独的ROS接口，但依旧很繁杂，而MoveIt是OMPL ROS接口的接口。而且MoveIt!还结合了其他一些功能，总之MoveIt!就是个大接口。

MoveIt!能做什么？一句话，MoveIt!就是一个模块化的接口，让你在最短时间内，不用自己写太多代码，就能配置出一个ROS Package来为你的机械臂做运动规划。

创建MoveIt! Package   
  
2.1 准备URDF package   
首先我们要准备一个机械臂的urdf，如果你已有URDF，可以使用自己的urdf模型。若手头没有现成的URDF，可以从[此处](https://github.com/openhumanoids/exotica)<https://github.com/ipab-slmc/exotica>下载一个库卡LWR简化模型URDF，这是一个固定底座7自由度的机械臂。   
从该连接处依次进入examples/sovlers/ik\_solver\_demo/resources，下载里面的lwr\_simplified.urdf。

  
将下载好的lwr\_simplified.urdf放入urdf文件夹中，这样一个urdf package便创建好了。   
  
  
2.2 MoveIt!配置助手 (MoveIt! Setup Assistant)   
2.2.1 打开MoveIt! Setup Assistant   
  
  
复制代码

|  |  |
| --- | --- |
| 1 | roslaunch moveit\_setup\_assistant setup\_assistant.launch |

MoveIt! Setup Assistant 是一个图形界面，可以让我们不用写代码看代码，直接用鼠标点击就可以配置机器人的运动规划所需要的信息。点击Create New MoveIt Configuration Package来创建新的配置包，选择刚刚下载的urdf，然后点击Load Files 载入文件。   
  
2.2.2 创建碰撞免检矩阵(ACM)   
  
点击Setup Assisant的左边第二项'Self-Collisions'，在这里我们将创建碰撞免检矩阵(Avoid Collision Matrix, ACM)。再次强调，怎么创建很简单，点击一下'Regenerate Default Collision Matrix'就可以了，问题是，为什么？ACM是做什么的？   
我们知道，碰撞检测是非常复杂的运算过程。对于多关节机械臂或者类人机器人来说，机械结构复杂，肢体多，碰撞检测需要涉及很多的空间几何计算。但是对于刚体机器人来说，有些肢体之间是不可能发生碰撞的，比如原本就相邻的肢体，比如类人机器人的脚和头。这里生成的ACM就是告诉我们，这个URDF所描述的机器人，哪些肢体之间是不会发生碰撞的。那么在之后的碰撞检测算法中，我们就可以略过对这些肢体之间的检测，以提高检测效率。   
  
  
2.2.3 创建虚拟关节 (Virtual Joints)   
在Setup Assistant 第三项Virtual Joints里面，我们要创建所谓的虚拟关节。这个虚拟关节，可以理解为一个连接机器人和世界的关节。

一般来说，Virtual Joint Name我们命名为‘world\_joint’，而'Child Link'指的是我们要把‘世界’和机器人的那个部位连接，那么很显然我们选择基座'base'。‘Parent Frame Name’，是世界坐标的名字，在ROS中一般叫'world\_frame'。关节类型 Joint Type, 很显然这里我们选择固定Fixed. 代表机器人相对于世界是固定的。而另外两种, Planar指的是平面移动底座(xy平面+角度)，用于移动机器人比如PR2；

还有一种Floating, 指的是浮动基座(xyz position+orientation)，比如类人机器人。

2.2.4 创建规划群 (Planning Groups)

创建Planning Group是MoveIt的核心之一。首先，点击Add Group, 我们会看到一个界面，如下图，

那么这些都是什么呢？

Group Name: 不用多说，名字。我们就叫Arm。

Kinematic Solver: 运动学求解工具，这个就是负责求解正向运动学(Forward Kinematics)和逆运动学(IK,)的。 一般我们选用KDL， [The Kinematics and Dynamics Library](http://www.orocos.org/kdl)。这是一个运动学与动力学的库，可以很好的解决6自由度以上的单链机械结构的正逆运动学问题。当然你也可以用其他IK Solver, 比如SRV或者[<span \"="" style="box-sizing: border-box;">IK\_FAST](http://docs.ros.org/indigo/api/moveit_ikfast/html/doc/ikfast_tutorial.html)，甚至你可以自己开发新的Solver然后插入进来。

Kin. Search Resolution: 关节空间的采样密度

Kin. Search TImeout: 求解时间

Kin. Solver Attempts: 求解失败尝试次数，一般来说这三项使用默认值就可以。你也可以根据具体需要做出适当调整。

接下来，我们要正式创建这个组群，有很多不同的方法，Add Joints, Add Links, Add Kin. Chain, Add Subgroups。我们这里选择'Add Kin. Chain'，这样我们可以清楚的看到整个机器人的机械机构，

在正中方我们可以看到这个机械臂的结构，一个link接着一个link。下方我们可以看到有'Base Link'和'Tip Link'，我们选择'lwr\_arm\_0\_link'作为Base，选择'lwr\_arm\_7\_link'作为Tip. 然后点击Save，这样一个规划组群就创建好了。同样的，我们可以再创建一个手的组群(Hand)，这一次我们用Add Links，然后选择'lwr\_arm\_7\_link'。

2.2.5 创建机器人预设位姿 (Robot Poses)

在Setup Assistant 第五项, ‘Robot Poses’，我们创建预设的机器人位姿。点击‘Add Pose’，我们为机械臂创建一个向上直立的位姿UpRight，选择Planning Group为Arm。可以看到很多滚动条，全设为0就是垂直向上的位姿。然后点击保存。当然，你可以根据需要设置其他不同位姿。



2.2.6 配置终端控制器(End Effectors)

终端控制器，就是机械臂的手，以后用来在工作环境中直接控制的部位。我们添加一个叫做HandEff的终端控制器，End Effector Group选择之前创建好的Hand，Parent\_Link选择机械臂的最后一个肢体lwr\_arm\_7\_link。Parent Group选择Arm。



2.2.7 配置被动关节(Passive Joints)   
所谓被动关节，就是指现实中不配置电机的关节，也就是不会出现在机器人的Joint State Msg里，以避免MoveIt与JointState出现匹配错误。这里我们的LWR机械臂并没有此类被动关节，所以可以直接跳过。   
  
2.2.8 生成配置文件(Configuration Files)   
最后一步，在Configuration Package Save Path里面选择一个保存地址，一般我们把他放在path\_to\_catkin\_ws/src/lwr\_moveit\_config然后点击Generate Package，这样一个完整的MoveIt Configuration Package就创建好了！先不要急着运行，我们先来看看都生成了哪些东西，还有一些重要的配置参数都是在哪定义的。  
  
三. MoveIt 配置包详解   
打开刚刚创建好的lwr\_moveit\_config文件夹，我们发现有config和launch两个文件夹。3.1 MoveIt! 配置文件先看config，里面有

fake\_controllers.yaml：这是虚拟控制器配置文件，方便我们在没有实体机器人，甚至没有任何模拟器（如gazebo）开启的情况下也能运行MoveIt。

joint\_limits.yaml：这里记录了机器人各个关节的位置速度加速度的极限，这些都会被用于以后的规划中。

kinematics.yaml：这里就是上一章2.2.4里面设置的东西，用于初始化运动学求解库

lwr.srdf：这个是一个重要的MoveIt配置文件。

ompl\_planning.yaml：这里是配置OMPL各种算法的各种参数。

3.2 SRDF文件   
SRDF是moveit的配置文件，配合URDF使用。打开lwr.srdf，

我们可以看到这是一个xml格式的配置文件，根是robot，并有一个属性值name='lwr'。下面各个项目应该很明显，就是我们刚刚在Setup Assistant里面所设置的东西，包含了组群，位姿，终端控制器，虚拟关节，以及碰撞免测矩阵ACM的定义。理论上，只要有了srdf和urdf，我们就可以完全定义一个机器人moveit信息。

3.3 Launch文件

下面，我们看看launch文件夹，我们来看看几个重要的文件。   
3.3.1 demo.launch

demo是运行的总结点，打开我们可以看到他include了其他的launch文件。其中第14行说，如果有需要，发布静态的tf。比如说，你的机器人基座不在世界坐标的原点，你可以发布一个静态tf来描述机器人在世界坐标中的位置。第17-21行，就是我们发布虚拟机器人状态的地方了，当然，如果你有实体机器人或者有gazebo之类的模拟器，你需要去掉这一部分，有其他相应的节点来发布机器人状态。26-32行运行了另一个moveit重要的节点，move group。   
  
  
3.3.2 move\_group.launch   
顾名思义，move group的功能是让一个规划组群动起来。怎么动，那就要做运动规划了，在move\_group.launch第24-26行定义了运动规划库的使用，我们可以看到，默认的是使用ompl运动规划库。同样的，如果以后有时间，我会发帖详解如何创建新的运动规划库插件并让moveit使用其他的运动规划算法。其他的都是设置一些基本参数，暂时可以略过。   
  
  
3.3.3 planning\_context.launch   
这里我们可以看到，定义了所使用的urdf和srdf文件，以及运动学求解库。不建议手动更改这些，但是如果你需要使用不同的urdf，srdf，可以在这里更改。   
  
  
3.3.4 setup\_assistant.launch   
如果你需要更改一些配置，那么可以直接运行

  
这样就可以基于当前设置做更改，而不是重新设置。   
  
四. 运行MoveIt!   
  
4.1 Launch Demo   
现在我们可以来尝试运行moveit了！

  
等待几秒，当看到 All is well! Everyone is happy! You can start planning now! 的时候，就代表启动成功了。我们可以看到一个Rivz窗口，左下角有一个运动规划MotionPlanning模块。

 第一个进入视野的就是Planning Library, OMPL。没错，这里告诉你当前用的是OMPL运动规划算法。在中间的下来菜单里面有很多的具体算法，之后你可以尝试不同的算法，看看他们的区别。

4.2 选择目标位姿

如果如上文第二章中设置，你会在rviz主窗口中看到一个互动标记位于机械臂终端位置。移动这个标记到另外一个地方，你可以看到一个橙色的目标位姿(每一次移动标记，就运行了一次逆运动学IK求解过程)。

同样的，你也可以在MotionPlanning模块下的Planning子模块写的Query子模块里面设置随机的或者预设的目标位置。

4.3到了运动规划的时候了。在Planning子模块中单击Plan，一个运动轨迹就会出现与Rviz窗口中并循环播放。你可以在Display->MotionPlanning->Planned Path里面设置各种显示参数。

4.4 避障运动规划 Collision Free Motion Planning

上面这个简单的运动规划，环境中并没有障碍物。下面我们来为环境中增加几个物体。创建一个demo.scene文件，将以下代码复制进去（你可以自己设计不同的scene）

(noname)+++++

\* obs1.dae

1

mesh

24 12

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 0.0315993

0 1 2

1 3 2

4 5 6

4 7 5

8 9 10

11 8 10

12 13 14

12 14 15

16 17 18

19 16 18

20 21 22

20 22 23

0.7 0 0.8

0.706825 0 0 0.707388

0.5 0 0 1

\* obs2.dae

1

mesh

24 12

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 0.0315993

0 1 2

1 3 2

4 5 6

4 7 5

8 9 10

11 8 10

12 13 14

12 14 15

16 17 18

19 16 18

20 21 22

20 22 23

0.6 0.5 0.8

0.706825 0 0 0.707388

0.5 0 0 1

\* obs3.dae

1

mesh

24 12

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 0.0315993

0 1 2

1 3 2

4 5 6

4 7 5

8 9 10

11 8 10

12 13 14

12 14 15

16 17 18

19 16 18

20 21 22

20 22 23

0.3 0 0.8

0.999784 0 0 0.0207948

0.5 0 0 1

\* table.dae

1

mesh

24 12

-0.420619 0.0420619 -0.841237

-0.420619 -0.0420619 -0.841237

-0.420619 0.0420619 0.841237

-0.420619 -0.0420619 0.841237

0.420619 0.0420619 0.841237

0.420619 -0.0420619 -0.841237

0.420619 0.0420619 -0.841237

0.420619 -0.0420619 0.841237

-0.420619 0.0420619 0.841237

-0.420619 -0.0420619 0.841237

0.420619 -0.0420619 0.841237

0.420619 0.0420619 0.841237

0.420619 -0.0420619 -0.841237

-0.420619 -0.0420619 -0.841237

-0.420619 0.0420619 -0.841237

0.420619 0.0420619 -0.841237

-0.420619 -0.0420619 0.841237

-0.420619 -0.0420619 -0.841237

0.420619 -0.0420619 -0.841237

0.420619 -0.0420619 0.841237

0.420619 0.0420619 -0.841237

-0.420619 0.0420619 -0.841237

-0.420619 0.0420619 0.841237

0.420619 0.0420619 0.841237

0 1 2

1 3 2

4 5 6

4 7 5

8 9 10

8 10 11

12 13 14

15 12 14

16 17 18

19 16 18

20 21 22

20 22 23

0.7 0 0.63

0.706825 0 0 0.707388

0 0.5 0.5 1

\* target

1

mesh

24 12

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

-0.0315993 -0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 -0.126397 0.0315993

-0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 -0.0315993

0.0315993 -0.126397 0.0315993

0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 -0.0315993

-0.0315993 0.126397 0.0315993

0.0315993 0.126397 0.0315993

0 1 2

1 3 2

4 5 6

4 7 5

8 9 10

11 8 10

12 13 14

12 14 15

16 17 18

19 16 18

20 21 22

20 22 23

0.6 0.2 0.8

0.706825 0 0 0.707388

0 1 0 1

.

然后在SceneObjects模块中点击ImportFromText，选择刚刚创建的demo.scene文件，一个简单的桌面环境就被导入进了rviz。你可以通过选择各个物体，来调整他们的位置。

回到Context子模块，点击Publish Current Scene，将当前的环境发布出去。

然后再次点击Plan，你会看到一条不同的轨迹，这一轨迹应该绕过所有障碍物并且达到目标位姿。

因为OMPL是采样算法，由于其随机采样的特性，每次的路径是不同的，如下图。而且有可能失败。