

ROS 实战演练

田雅夫

State Key Laboratory of Robotics and System

2017 年 10 月 26 日

目录

- ROS 简介
- 上手: 玩转小乌龟 (TurtleSim)
- 在模拟器中进行仿真
- ROS 机器人实例: 小白 (喷雾消毒机器人)

ROS 简介

概述

- ROS 是开源的, 生态环境良好的机器人元操作系统.
- ROS 的核心功能是松耦合的, 兼具同步与异步的进程网络.
- ROS 包含大量的, 经过充分测试的, 面向多种机器人应用场景的软件包.
- ROS 是现代机器人事实上的标准. 大多数机器人的控制程序都会开发面向 ROS 的接口.
- ROS 支持Linux, Unix, Mac OS 系统. 有限支持 Windows 系统.
- ROS 可以在普通计算机, 工控机, RaspberryPi 等Linux 开发板上运行. 也可以在虚拟机, docker 上运行. 甚至利用 ROS 的松耦合特性在多台电脑上构成的网络上运行一个项目.
- 你可以使用C++, Python, Common Lisp 😊 来构建程序的主要部分. 并使用任意语言 (Java, Javascript, Matlab, Ruby, Haskell 等) 构建独立的模块

ROS 简介

ROS 的优势: 多进程网络

- 一个完整的 ROS 程序由多个**节点**组成. 每个节点是一个**独立的进程**.
- ROS 支持节点间的**同步 (service)**, **异步 (topic)**通信.
- 每个 ROS 节点专注于机器人的一部分功能. 这些节点是可以独立运行的程序.
- 一个 ROS 工程中的不同节点可以运行在不同的系统, 甚至不同的机器上.
- 异步通信使得机器人的各个节点间**解耦合**. 一个甚至更多节点出现异常也能很好地处理.
- 通过动态启动/关闭节点的方式**配置机器人的行为**, 或是**解决出现的异常**.

ROS 的优势: 多进程网络

原始的机器人程序结构:

```
while(not robot.isShutdown()):  
    laserData, odomData, otherData = sensorHandler.readData()  
    decision = pathPlanner.makePlan( laserData, odomData, otherData )  
    motorHandler.move( decision )  
    sleep(1)
```

- 机器人的各个组件都有失效的可能性
- 机器人控制程序每一部分的时延是不确定的, 不方便控制.
- 机器人控制程序可能在某个位置被”阻塞”
- 机器人程序耦合性过高, 各个模块无法独立运行. 不利于二次开发与代码复用.

ROS 的优势: 多进程网络

```
# laserDriver
rate=Rate(0.5)
while( not node.shutdown()):
    data=collectLaserData()
    LaserScan(data)
    pub=Publisher("scan", LaserScan,
                  queue_size=5)
    pub.publish(data)
    rate.sleep()
```

```
# odomCalculator
rate=Rate(0.5)
while( not node.shutdown()):
    data=collectOdomData()
    Odom(data)
    pub=Publisher("odom", Odom, queue_size=5)
    pub.publish(data)
    rate.sleep()
```

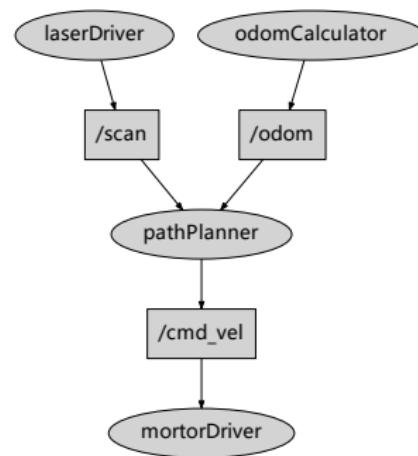


图 : ROS 的节点 - 话题结构示例

ROS 的优势: 多进程网络

```
# PathPlanner
rate=Rate(0.5)

def callback_recvLaserData(_laserScan):
    laserData = _laserScan.data

def callback_recvOdom(_odom):
    odom = _odom.data

while( not node.shutdown()):
    Subscriber("scan", LaserScan, callback=
        callback_recvLaserData)
    Subscriber("odom", LaserScan, callback=
        callback_recvOdom)

# rospy.spin()
plan = Twist( makePlan(laserData, odom) )
pub=Publisher("cmd_vel", Twist, queue_size=5)
pub.publish(plan)
rate.sleep()
```

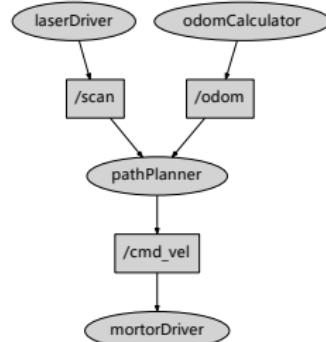
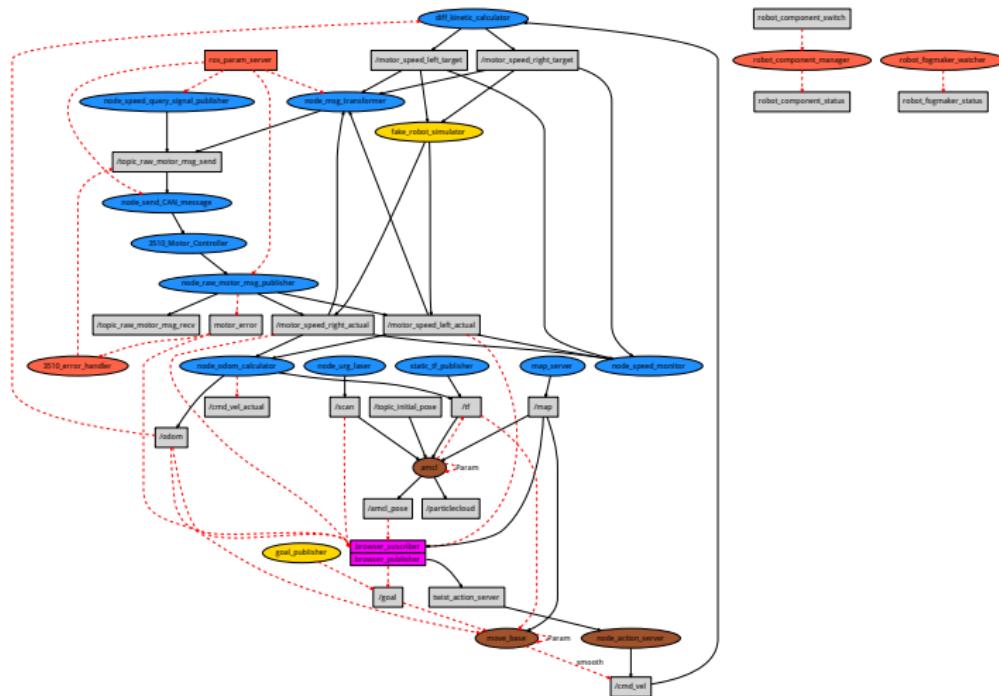


图 : ROS 的节点 - 话题结构示例

ROS 的优势: 多进程网络

小白机器人的节点拓扑图:



ROS 的优势: 多进程网络

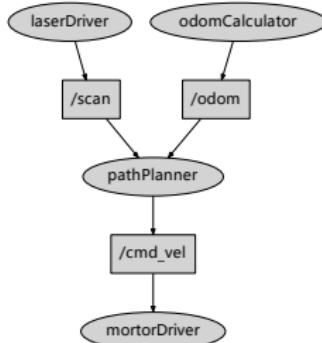
开放问题

开放问题：拓扑图的表示与可视化

- 已知有向有环图 $G = \langle V, E \rangle$
 $V = \{v_1, v_2, \dots, v_n\}$
 $E = \{\langle v_i, v_j \rangle \mid v_i, v_j \in V\}$
 - v_i 包括以下属性: id, label, size, shape
 - 集合 E 以邻接矩阵 A_{nn} 的形式给出,

$$A_{ij} = 1 \quad \text{when } \langle v_i, v_j \rangle \in E \quad (1)$$

- 用什么工具可以自动画出右图这样形式的拓扑图？
 - 如果要求自己实现一个这样的程序，要考虑那些情况？



图：一个拓扑图的实例

ROS 简介

ROS 的优势: 成熟的生态环境

ROS 的设计思想: 不要重复造轮子

- 别人做好的, 并且经过充分测试的功能模块可以拿来使用.
- 别人已经实现的, 成熟的机器人架构可以拿来借鉴.
- 每个功能模块的接口 (API) 只有以下三种, 非常方便使用. 无需非常复杂的依赖关系:
 - ① Topic 节点监听 topic 来获取数据, 向 topic 发送计算的结果.
 - ② Service 调整节点的功能, 更改可变参数
 - ③ 配置文件: 用来配置软件中的一些基本 (一般不会改变的) 参数.
- 使用别人的功能模块的方法: 确保数据格式正确且语义正确的情况下, 生成消息传给别人已经写好的节点.
- 成熟的生态环境: 文档相对规范, 方便上手. 出了问题 Google 一下基本都能找到答案. 不保证用百度能搜到正确答案 😊
- 拥抱开源: 你写好的功能模块, 遇到并解决了的问题回馈给社区. 帮助 ROS 家族发展壮大. 并证明你的能力, 找到更好的 offer 😊

ROS 简介

ROS 的优势: 成熟的生态环境

机器人运动学/运动规划:

- MoveIt (OMPL)
- ros-planning
- ROS navigation stack (一般用 move_base)
- 面向多自由度机械臂的套件

机器人 SLAM:

- Gmapping/Cartographer
- ROS navigation stack (mrpt-SLAM, AMCL)

底层硬件控制:

- hokuyo-node, urg-node (用这个)
- rosserial
- usb_cam, uvc_camera(安卓专用), opeeni2 (中间件, 包含 kinect 驱动)

常用工具

- RQT 工具包 (rqt-graph, rqt-ploth, rqt-tf-tree 等)
- rviz
- rosjslib, ros-bridge
- rosjava, ros-android

ROS 学习资源

- ① ROS tutorial <http://wiki.ros.org/ROS/Tutorials>

最重要的教程! 所有 ROS 教程的基础! 强烈建议从头到尾跟着做一遍

- ② 各个软件包的[官方文档](#). 强烈推荐

找不到的话可以跳转到该软件包的 github 页面, 在 readme 中一般都有文档的地址

- ③ ROS 探索总结 (古月居) <http://www.guyuehome.com/>

- ④ 知行合一博客 <http://blog.csdn.net/heyijia0327>

ROS 学习资源: 怎么看官方文档 -以 gmapping 为例

- 先看 **Nodes**: 该软件包提供几个节点, 分别做什么用? 该软件包提供一个节点 `slam_gmapping`
- 再看节点的 **Subscribed Topics** 和 **Published Topics** -> 这个节点的输入, 输出数据是什么样子的

Subscribed Topics: (要送到该节点里的数据)

- `tf` (`tf/tfMessage`) **topic 名称, 消息类型.** 这个消息类型的数据结构百度一下就有 `Transforms necessary to relate frames for laser, base, and odometry (see below)`
- `scan` (`sensor_msgs/LaserScan`) `Laser scans to create the map from`

Published Topics: (这个节点输出的数据)

- `map_metadata` (`nav_msgs/MapMetaData`) `Get the map data from this topic, which is latched, and updated periodically.`
- `map` (`nav_msgs/OccupancyGrid`) **这个是最感兴趣的, 其他两个可以忽略** `Get the map data from this topic, which is latched, and updated periodically`
- `entropy` (`std_msgs/Float64`) `Estimate of the entropy of the distribution over the robot's pose (a higher value indicates greater uncertainty). New in 1.1.0.`

ROS 学习资源: 怎么看官方文档 -以 gmapping 为例

- ① 其次看 Services, 它提供了一些额外的功能. 仔细看的话常有惊喜
- ② 最后看 Parameters, 这是该节点运行所需要的一些参数. 这些参数的意义要看明白, 调整参数让程序达到一个很好的效果

- particles (int, default: 30)
Number of particles in the filter
- xmin (float, default: -100.0)
Initial map size
- ymin (float, default: -100.0)
Initial map size
- xmax (float, default: 100.0)
Initial map size
- llsamplestep (float, default: 0.01)
Translational sampling step for the likelihood
-Gmapping 有几个参数, 找到其中相对重要的加以调节需要经验

上手: 玩转小乌龟 (TurtleSim)

开发 ROS 机器人的基本步骤

- ① 明确硬件类型, **为每个硬件 (传感器, 驱动器等) 写驱动程序**. 这些驱动程序以节点的形式出现, 并用 topic 方式呈递数据.
- ② 机器人正/逆运动学解算.
- ③ **明确机器人控制方式**
 - 开环控制
 - 单闭环控制 (力/电流/位置/速度等)
 - 多闭环控制 (位置/速度混合控制, 力位混合控制)
- ④ 写控制器 (前馈, 反馈) 节点 (收到传感器数据, 转化为驱动器指令)
- ⑤ 在仿真器中对机器人的工作状态进行仿真
- ⑥ 在每个节点中添加异常与错误的处理程序
- ⑦ 对每个节点实现**单元测试**
- ⑧ 对整个机器人系统进行冒烟测试, 集成测试
- ⑨ 开发用户界面. 写软件文档, 手册

这一节涉及上面的 3, 4, 5 三部分

上手: 玩转小乌龟 (TurtleSim)

目标

- 手动操作机器人前进/后退/转弯
- 在地图中给定一点, 让机器人到达该点 (地图中无障碍)
- 几个小乌龟编队行进

Turtlebot 介绍

从官方 wiki 上获得的信息:

- turtlesim 是一个入门用机器人模拟器
- 启动 turtlesim 的方法:

```
roscore &  
rosrun turtlesim turtlesim_node
```

- turtlesim 监听一个 (或者多个?) Twist 类型的消息 turtleX/cmd_vel 作为机器人速度的输入
- turtlesim 发布一个 (或者多个?) Pose 类型的消息 turtleX/pose 作为机器人的位姿
- turtlesim 提供一系列 service 来增加/干掉特定的一个乌龟

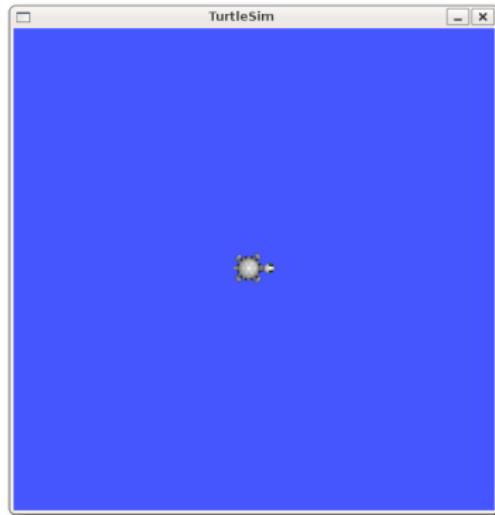


图 : turtlesim 界面

Twist 信息的结构

每个乌龟从相应的 turtleX/cmd_vel 话题
接收一个 Twist 信息作为速度输入. 从官方
wiki 上获得的信息:

```
geometry_msgs/Twist Message:  
  Vector3 linear  
    float64 x  
    float64 y  
    float64 z  
  Vector3 angular  
    float64 x  
    float64 y  
    float64 z
```

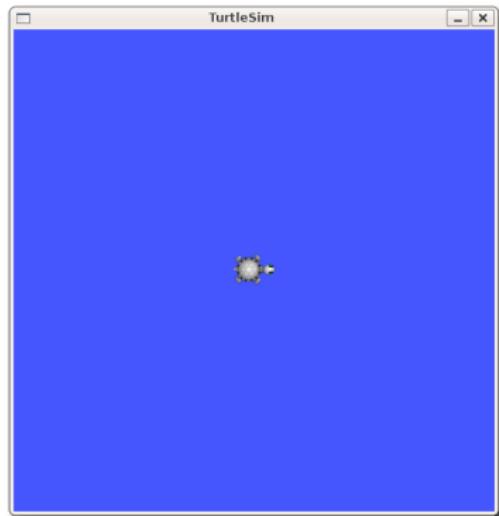
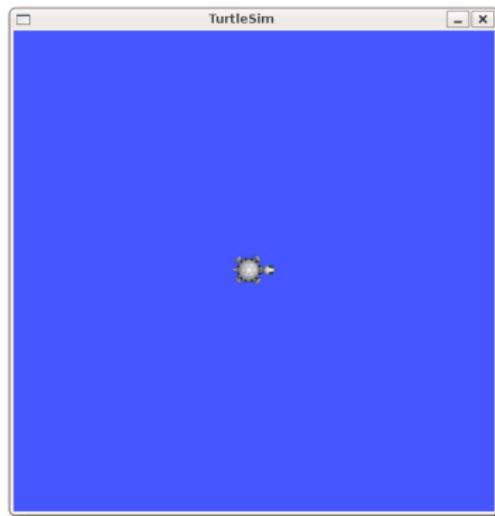


图 : turtlesim 界面

Twist 信息的结构

构造信息的方法:

```
msg = Twist()  
  
msg.linear.x, msg.linear.y, msg.linear.z =  
    (1, 0, 0)  
  
msg.angular.x, msg.angular.y, msg.angular.z =  
    (0, 0, 1)
```



- linear 对应的是机器人三个轴上的线速度, angular 对应的是 yaw, pitch, roll 三个角速度
- turtle 是两轮差分机器人, 所以只有 linear.x, angular.z 有意义
- 不支持 linear.y(乌龟不是螃蟹 😊), linear.z(空格起跳不可 😊), angular.x, angular.y(前滚翻和横滚翻做不到 😊)

图 : turtlesim 界面

新建工程

① 新建 catkin_workspace

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

② 配置环境变量 (这一步会经常用到的)

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

③ 在 catkin_ws 下新建软件包, 该软件包有依赖项 (rospy roscpp)

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg my_package  
    std_msgs rospy roscpp
```

⑤ (cpp 用户:) 正常编写软件包, 写好了用以下命令编译. 编译完了去 build 文件包的相应位置找可执行文件.

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

⑥ (python 用户: 无视上一步, 不过要在新建软件包时运行一次)

⑦ 注意 catkin_make 是在所有软件包的层级上进行的. 不过只会去编译改动了的部分. 代码没有修改的话是不会重新编译的

Catkin workspace 的结构:

catkin_ws:

- src 放源代码的文件夹
 - CMakeLists.txt
 - package1 一个软件包
 - include 一些第三方软件库可以通过直接拷贝的方式来使用
 - urdf 机器人描述文件
 - launch 存放工程的.launch 文件
 - config 一般在这个目录中保存所有配置文件
 - src 该软件的所有代码保存在这里
 - CMakeLists.txt package.xml 软件包级配置文件
 - package2 … package n 一个 catkin 中通常包含多 (hen) 个 (duo) 软件包
- build 代码生成的可执行文件会放在这个目录中
 - package1 … package n 使用 cpp 的话, 要在这里找编译出的可执行文件, 用 Python 的话无视它
- devel
- setup.sh 或 setup.bash, 出现“找不到东西”的错误时, 执行一次一般就好了

开始工作

启动 roscore, turtlesim 节点

```
roscore &  
rosrun turtlesim turtlesim_node
```

查看系统当前的状态, 显示当前可用的节点/话题/服务: (非常重要!)

```
rosnode list  
rostopic list  
rosservice list
```

查看一个 topic 的实时输出:

```
rostopic echo <topic name>
```

查看某个节点/话题/服务的详细信息:

```
rosnode info <node name>  
rostopic info <topic name>  
rosservice info <service name>
```

写一个节点向 turtle 发送速度信息

```
import multiprocessing, rospy
from geometry_msgs.msg import Twist

class velocityPublisher:
    def __init__(self): pass
    def procFunc(self):
        rospy.init_node("turtle_velocity_publisher")
        rospy.loginfo("Start node " + "turtle_velocity_publisher")
        pub = rospy.Publisher("turtle1/cmd_vel", Twist, queue_size=5)
        rate = rospy.Rate(5) # Hz
        while(not rospy.is_shutdown()):
            msg = Twist()
            msg.linear.x, msg.linear.y, msg.linear.z = (1, 0, 0)
            msg.angular.x, msg.angular.y, msg.angular.z = (0, 0, 1)
            pub.publish(msg)
            rate.sleep()
    def startNode(self):
        proc = multiprocessing.Process(target=self.procFunc)
        proc.start()
```

写一个节点接收速度信息

- 注意 `rospy.spin()`
- 注意 `callback` 机制

```
class velocityPublisher:  
    def __init__(self): pass  
    def callback_receiveTwist(_twist):  
        rospy.logdebug("Velocity:{0},{0}".format(_twist.linear.x, _twist.angular.  
            z))  
    def procFunc(self):  
        rospy.init_node("turtle_velocity_receiver")  
        rospy.loginfo("Start node " + "turtle_velocity_receiver")  
        pub = rospy.Subscriber("turtle1/cmd_vel", Twist, callback=self.  
            callback_receiveTwist)  
        rospy.spin()  
    def startNode(self):  
        proc = multiprocessing.Process(target=self.procFunc)  
        proc.start()
```

上手: 玩转小乌龟 (TurtleSim)

实现 ROS 节点

- 节点的名称
- 节点要实现的功能
- 明确节点的信息流向 (只接收/只发送/接收并发送)
 - 发送信息的节点注意发送频率, 在 `rospy.Rate` 里面修改
 - 接收消息的节点注意要用一个 `rospy.spin()` 或者一个无限循环 `hold` 住这个节点
 - 节点同时发送/接收消息时要用一个**线程安全**的数据结构来存放节点的状态信息. 推荐 `multiprocessing` 下的 `Value`, `Array` 等
- 保证节点在一个独立**进程**下运行
 - 一般会用`.launch` 文件启动所有节点
 - 我的做法是用**`multiprocessing`**包来独立的启动各个节点

多个乌龟

调用 turtlesim 的 service "spawn" 可以加入一个乌龟

```
rosservice call [service] [args]  
rosservice call /spawn 2, 2, 3.14, "Joy"
```

在程序中调用 service, 有两种方式:

- ① 官方方式: 用 serviceProxy, 见例程. 缺点是要找到 turtlesim.srv 以获取函数的相关定义
- ② 通过 os.popen(), subprocess 等模块以命令行的方式调用
如果你知道自己在干什么的话 😊 推荐第二种方式

每一个乌龟有自己的话题"turtle_i/cmd_vel", 所以要用 i 个 publisher 来给这些乌龟提供话题

TF

- TF: ROS 的坐标变换库
- 负责处理机器人各个组件坐标系间位置关系
- 以固定的频率自动计算/更新机器人内部各个组件的位置与姿态
- 负责计算机器人在环境中的位置
- 提供一系列函数来计算坐标变换
 - 机器人平移/旋转矩阵的计算
 - 四元数/欧拉角/绕某轴复合运动的相互转换
 - 运动学的反解 (有限)

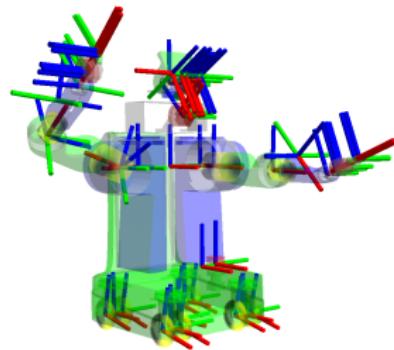


图 : ROS TF

发布 TF

- 发布 TF 的本质: 明确机器人各个部件/机器人与环境之间的相对位置
- fixed TF 与 non-fixed TF
 - fixed TF 可在.launch 文件中, 或是命令行中以固定频率发送
 - fixed TF 在机器人运行过程中是不变的
 - non-fixed TF 在机器人运行中是会变化的, 这种 TF 关系要在程序中进行发布
 - non-fixed TF 在移动机器人平台的应用: 处理 `odom`, `map`, `base_link` 之间的变换

```
$roslaunch turtle_tf turtle_tf_demo.launch
```

```
br = tf.TransformBroadcaster()
transform = tf.transform()
transform.setOrigin( tf.Vector3((msg.x), (msg.y), 0.0) );
q = tf.Quaternion()
q.setRPY(0, 0, msg.theta)
transform.setRotation(q)
now = Time.now()
br.sendTransform(tfStampedTransform(
    transform, now, "/world",
    turtle_name))
```

- <http://www.jianshu.com/p/cd6ba775639b>

.launch 文件

- launch 文件是xml格式的机器人配置文件.
- ros 核心系统读取 launch 文件来启动特定的节点并用给定的参数配置节点
- 当你明白你在做什么的时候 😊 可以放弃.launch 并用你自己的方式来配置节点
- launch 文件一般用来批量启动节点并设置这些节点的参数. 也就是把多条指令合并到一起执行

.launch 文件

- launch 文件是xml格式的机器人配置文件.
- ros 核心系统读取 launch 文件来启动特定的节点并用给定的参数配置节点
- 当你明白你在做什么的时候 😊 可以放弃.launch 并用你自己的方式来配置节点
- 一个工程可以有多个 launch 文件, 执行不同的任务
- .launch 文件的编写方式如实例:

在模拟器中进行仿真

ROS 平台上的仿真器:

- ① Stage
- ② Gazebo
- ③ MobileSim

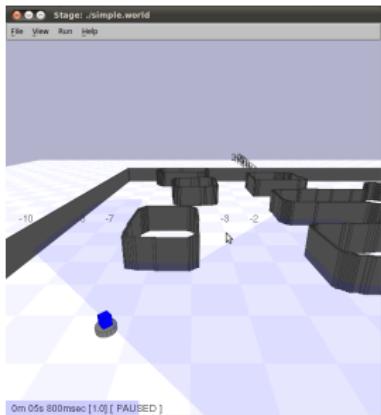


图 : Stage 仿真器

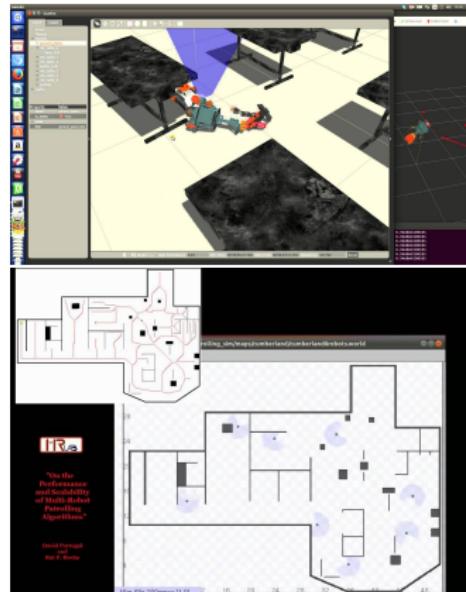


图 : (上) Gazebo (下) Stage

在模拟器中进行仿真

MobileSim 仿真平台

- MobileSim 是 Aria Mobile Robot (先锋机器人) 推出的一款 2D 机器人仿真软件.
- 该软件可对先锋机器人公司出品的多种机器人进行仿真
- 该软件可仿真机器人的运动学特性, 模拟机器人的激光雷达, 声呐, 防撞带, 电池容量等信息
- 该软件的接口与先锋机器人 (实际机器人) 的接口完全相同, 可以直接互换使用
- 该软件可以在 windows 平台使用



图 : mobilesim 机器人仿真器

在模拟器中进行仿真

MobileSim 仿真平台的使用

在官网上下载对应平台的 MobileRobot SDK 和 MobileSim, 并进行安装.

MobileRobot SDK:

- ① 包含上位机与先锋机器人的通信协议, 数据交换协议等核心程序. 以 dll 的形式打包
- ② 包含对 matlab, python, java 等编程语言的**接口**
- ③ 先锋机器人对 ros 的接口 (rosaria) 要自己编译安装.
 - ① 在 github 上 clone rosaria 的代码库到本地的 catkin_ws 上
 - ② 本地 catkin_make 编译代码
 - ③ 缺依赖库的话就装上
- ④ 在命令行中 rosrun rosaria RosAria 开启先锋机器人的 ROS 节点
- ⑤ 先锋机器人订阅 /RosAria/cmd_vel 话题作为机器人的控制方式
- ⑥ RosAria 节点发布 /scan, /sonar, /pose 等信息作为输出

小白机器人的结构



小白机器人的硬件结构

- 驱动器: 电机 + 码盘 + 驱动器的伺服装置
- 环境传感器:
 - 激光雷达 HOKUYO UST-10LX
 - 超声波传感器
 - 摄像机
- 内部装置:
 - 超声波雾化器, 风扇
 - 温度/液位传感器
 - ADAM6017 模拟量采集卡
 - ADAM6060 多路 IO 通道卡
 - 研华 CAN 总线服务器

机器人组件驱动 -激光雷达

激光雷达的工作方式如图: 激光雷达输出的数据: 一个长度为 n 的一维数组. 令扫描角度为 θ , 角分辨率 res . 则 $n = \theta \times res$ 数组里的每个元素是障碍物到激光雷达的距离 (没检测到障碍物的话是一个很大的值)

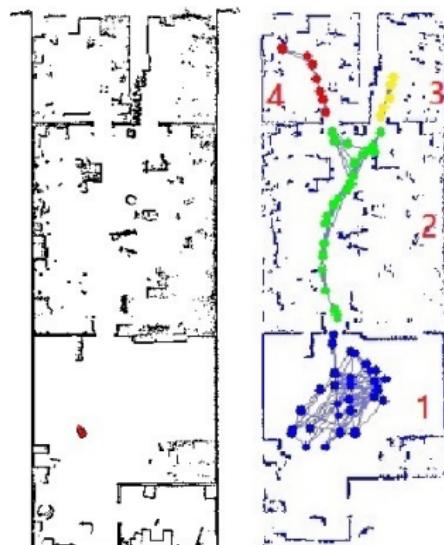
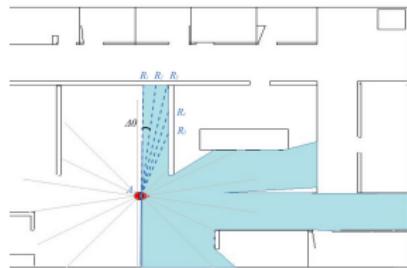


图 : 激光雷达扫描出来的环境示意图

机器人组件驱动 -激光雷达驱动

对于不常用的传感器, 需要自己写驱动. 但该激光雷达有别人写好的驱动可以用.

我写了一个简单的驱动, 并翻译了激光雷达的通信协议手册. 详见我的知乎专栏

现成的驱动可以使用 `hokuyo_node` 或 `urg_node`, 推荐使用后一个

机器人组件驱动 - 伺服电机驱动

伺服电机我们选用 3510 电机驱动器，3510 电机驱动器使用 CAN 总线协议通信。所以要使用一块 CAN 总线服务器卡，将计算机的 socket 协议通信 (ethernet) 转换成 CAN 总线协议。

CAN 总线协议文本见附文

小白机器人

开发 ROS 机器人的基本步骤

- ① 明确硬件类型, **为每个硬件 (传感器, 驱动器等) 写驱动程序**. 这些驱动程序以节点的形式出现, 并用 topic 方式呈递数据.
- ② 机器人正/逆运动学解算.
- ③ **明确机器人控制方式**
 - 开环控制
 - 单闭环控制 (力/电流/位置/速度等)
 - 多闭环控制 (位置/速度混合控制, 力位混合控制)
- ④ 写控制器 (前馈, 反馈) 节点 (收到传感器数据, 转化为驱动器指令)
- ⑤ 在仿真器中对机器人的工作状态进行仿真
- ⑥ 在每个节点中添加异常与错误的处理程序
- ⑦ 对每个节点实现**单元测试**
- ⑧ 对整个机器人系统进行冒烟测试, 集成测试
- ⑨ 开发用户界面. 写软件文档, 手册

这一节涉及上面的 3, 4, 5 三部分

小白机器人

小白机器人的节点拓扑图:

