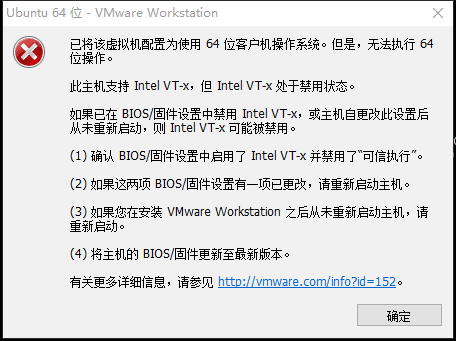
Slam流程

一 安装虚拟机vmware workstation14（百度教程<https://jingyan.baidu.com/article/c275f6ba07e269e33d756714.html>）

二 在虚拟机中安装Ubuntu 16.04（https://www.ubuntu.com/download/desktop）（要在电脑磁盘中独立分出大约50G的空间出来：电脑/管理/磁盘管理/压缩卷/50000/新建简单卷）

创建新的虚拟机/自定义/下一步/稍后安装操作系统/ubuntu64/linux/位置是独立的安装磁盘/创建新的虚拟磁盘/存储为单个文件/完成—编辑虚拟机设置/ cd/dvd选择下载的镜像文件 ，显示器选项中不要勾选3D加速效果，除非是有好显卡的电脑，否则可能在用可视化插件如gazebo时会出现死机现象。

若安装后出现以下情况



请参考<https://jingyan.baidu.com/article/c275f6ba07e269e33d756714.html>

（关机，开机时按F2（win8/10命令行进入 cmd shutdown.exe -r -o -t 0），进入BIOS设置，将Intel Virtual Technology从Disable改为Enable，按F10保存并退出，电脑会重启，然后再打开VMware，运行虚拟机）

然后打开该虚拟机安装ubuntu就可以了。

注意： 用虚拟机安装的可能不能使用复制粘贴，即不能从windows中复制到虚拟机，那是因为vmware tools没有安装。

三 若直接在电脑上安装Ubuntu系统，下载Ubuntu系统。下载地址<https://www.ubuntu.com/download/desktop>

然后安装一个ultralso软件（腾讯管家上就可以安装），然后用U盘在ultralso中制作一个启动盘（<https://jingyan.baidu.com/article/a378c960630e61b329283045.html>）

以管理员身份运行ultralso安装系统。（详情参考教程）

**四 安装ROS系统**

在ubuntu中打开一个新终端，输入脚本安装

$ wget https://raw.githubusercontent.com/oroca/oroca-ros-pkg/kinetic/ros\_install.sh && chmod 755 ./ros\_install.sh && bash ./ros\_install.sh catkin\_ws kinetic

或参考<https://www.ncnynl.com/archives/201707/1817.html>

**然后安装turtlebot3及依赖项**

$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation

**安装turtlebot3源码**

这里首先需要了解如何创建一个新的名为catkin\_ws工作空间

1 $ mkdir -p ~/catkin\_ws/src

解析：在 /home/你的用户名目录/ 下创建一个ROS工作空间，即文件夹catkin\_ws,  
然后在工作空间 catkin\_ws 下创建一个文件夹 src  
你也可以将工作空间 catkin\_ws 起成其他名字，例如：mkdir -p ~/tutorial/src ，那么下面的所有命令中的catkin\_ws都要对应换成 tutorial

2 $ cd ~/catkin\_ws/src

3 $ catkin\_init\_workspace

解析：

执行完该命令后，src目录下会多出一个 CMakeLists.txt 文件

4 $ cd ~/catkin\_ws

5 $ cakin\_make

解析：

执行完该命令后，发现工作空间catkin\_ws中有三个目录： build devel src

其中，src是我们创建工作空间时创建的目录，另外两个是执行完 catkin\_make 后生成的。

**下载turtlebot3的代码包**

$ cd ~/catkin\_ws/src/

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3\_msgs.git

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git

$ cd ~/catkin\_ws && catkin\_make

然后将环境设置一下:

$ echo "source ~/catkin\_ws/devel/setup.bash" >> ~/.bashrc

(这里也可以直接打开 .bashrc文件，输入source ~/catkin\_ws/devel/setup.bash，保存，再在终端输入 $ source ~/.bashrc)

最后再查看路径$ echo $ ROS\_PACKAGE\_PATH

**更改工作空间路径**

Ctrl+h 选择 .bashrc，打开中有一行

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

将这一行中catkin\_ws改为你的工作空间名字

$ source ~/.bashrc

再查看echo $ ROS\_PACKAGE\_PATH

**安装goolgle cartographer算**法

首先新建一个工作空间，记得添加路径后查看路径

参考<https://www.ncnynl.com/archives/201801/2230.html>具体安装过程可能还会出现其他问题。安装好后可能要对启动文件做小改动。

Cartographer主要理论是通过闭环检测来消除构图过程中产生的累积误差。用于闭环检测的基本单元是submap。一个submap是由一定数量的laser scan构成。将一个laser scan插入其对应的submap时，会基于submap已有的laser scan及其它传感器数据估计其在该submap中的最佳位置。submap的创建在短时间内的误差累积被认为是足够小的。然而随着时间推移，越来越多的submap被创建后，submap间的误差累积则会越来越大。因此需要通过闭环检测适当的优化这些submap的位姿进而消除这些累积误差，这就将问题转化成一个位姿优化问题。当一个submap的构建完成时，也就是不会再有新的laser scan插入到该submap时，该submap就会加入到闭环检测中。闭环检测会考虑所有的已完成创建的submap。当一个新的laser scan加入到地图中时，如果该laser scan的估计位姿与地图中某个submap的某个laser scan的位姿比较接近的话，那么通过某种 scan match策略就会找到该闭环。Cartographer中的scan match策略通过在新加入地图的laser scan的估计位姿附近取一个窗口，进而在该窗口内寻找该laser scan的一个可能的匹配，如果找到了一个足够好的匹配，则会将该匹配的闭环约束加入到位姿优化问题中。Cartographer的重点内容就是融合多传感器数据的局部submap创建以及用于闭环检测的scan match策略的实现。

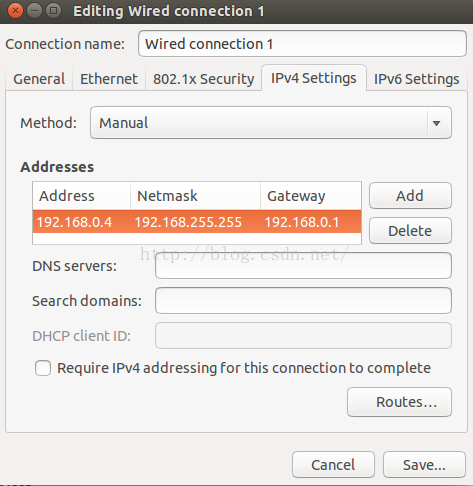
**lms1xx在ros上的安装驱动**

1, 下载驱动到你想要放置的工作空间的src文件夹下（该工作空间不是安装cartographer的工作空间）

$ cd catkin\_ws/src

$ git clone <https://github.com/clearpathrobotics/lms1xx.git>

2,点击wifi图标，点击编辑链接，点击以太网(ethernet)，然后添加，新建，链接名称随意，我的取名lmsxxx,然后IPv4设置，配置IP： 192.168.0.4 掩码：255.255.255.0 网关：192.168.0.1



3.进入第一步下载的驱动包，打开launch文件夹中的launch文件，修改default="192.168.0.1" 保存.

或者IP一劳永逸修改，找到catkin\_ws/src/LMS1xx-master/src/LMS1xx\_node.cpp文件，gedit打开，第50行，将IP改成激光默认的就行了

4.然后再source一下工作空间

$ source catkin\_ws/deve/setup.bash

5.然后运行

$ roslaunch lms1xx LMS1xx.launch或rosrun lms1xx LMS1xx\_node \_host:=192.168.0.1

然后用 $ rostopic list 查看是否有/scan话题

$ rostopic echo /scan 查看激光头数据

如果想用可视化软件RVIZ查看激光头数据 $ rosrun rviz rviz 添加ADD/by topic/LaserScan，再将FixedFrame后面的值改为laser即可

如果想连接串口，还需要再launch文件中添加串口的初始化，参考turtlebot3\_core的激光头启动文件，将里面的节点全部复制到lms.launch文件中，再启动就行

6.想要使用cartographer算法建图，直接启动激光头，再启动算法建图(启动launch文件)

Slam流程

1 $ roscore 启动ros

2 $ roslaunch lms1xx LMS1xx.launch 启动激光头

3 $ export TURTLEBOT3\_MODEL=burger

4 $ roslaunch cartographer\_turtlebot turtlebot3\_lidar.launch 启动建图算法

5 $ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch 键盘控制机器人扫描地图

6 $ rosrun map\_server map\_saver -f ~/map 建完地图保存地图

7 $ export TURTLEBOT3\_MODEL=burger

8 $ roalaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=$HOME/map/map1234.yaml 启动导航，加载地图

9 $ rosrun rviz rviz -d ‘rospack find turtlebot3\_navigation’/rviz/turtlebot3\_nav.rviz

在开始导航之前，TurtleBot3应该知道它的位置和姿势。

要给出初始数据，请按照说明进行操作。

点击2D Pose Estimate按钮

通过单击并拖动地图上的方向来设置地图上的大致位置。

箭头的每个点意味着TurtleBot3的预期姿势。激光扫描仪将在近似位置绘制线条，如地图上的墙壁。

如果图形没有显示线条，请重复上述过程。

当TurtleBot3已经定位，它将自动计划路径。

要发送目标位置： 点击 2D Nav Goal按钮

点击地图上你想要的TurtleBot驱动和拖动方向TurtleBot应该指向地方

如果目标位置的路径被阻止，这可能会失败。

要在机器人到达目标位置之前停止机器人，请发送TurtleBot3的当前位置(如何发送？)

也可以通过程序控制多点的导航https://www.ncnynl.com/archives/201704/1544.html

**常见指令**

cd 定位文件夹

rosrun roslaunch 运行启动文件

rostopic list 查看话题列表

rostopic type +话题名 查看话题类型

romsg show 终端查看数据

rostopic pub 可以把数据发布到当前某个正在广播的话题

rostopic pub -1 /cmd\_vel geometry\_msgs/Twist ‘{linear:{x:0.1,y:0,z:0},angular:{x:0,y:0,z:0}}’

-1表示消息只发送一次，若换成-r 10 表示频率hz

若想让机器人停止运行，可以发布一条空的消息。

rostopic pub -1 /turtlel/command\_velocity turtlesim/Velocity -- 2.0 1.8

命令诠释：

rostopic pub －－－在给定主题上发布消息

-1 －－－此选项表示rostopic只发布一个消息，然后退出

/turtle1/command\_velocity －－－给定的主题名

turtlesim/Velocity －－－在给定主题上发布的消息类型

－－－双负号，表示词法选择器在此后的参数都不可用。这种情况发生在错误参数比如负

数

2.0 1.8 －－－当前主题消息的类型的数据值

rostopic echo /scan 查看消息

source ~/.bashrc

ROS笔记(一)xxx.launch文件详解

.launch文件是ROS中用于同时启动多个节点的重要文件,在大型的ROS项目中使用频繁,所以掌握其主要元素与属性对ROS系统的应用至关重要：

launch标签(元素)说明

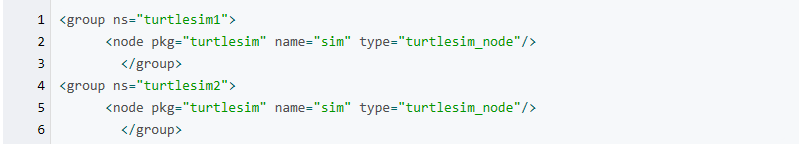
launch拓展说明

parameter说明

1. 标签(元素)说明

- 以launch标签开头以表明是launch文件.

- group标签定义命名空间.



在这里我们创建了两个节点分组并以命名空间（namespace)标签来区分，其中一个名为turtulesim1，另一个名为turtlesim2，两个组里面都使用相同的turtlesim节点并命名为’sim’。 这样可以让我们同时启动两个turtlesim模拟器而不会产生命名冲突。(当然如果直接用不同的命名也是可以的,比如一个叫sim\_1,一个叫sim\_2,这样也可以同时多次启动同一节点).

1. if 或unless 属性

组(groups)可以通过 判别条件 来启用或禁用节点(nodes)：

<group

if="0-or-1" />

</group>

如果 if 属性的值是 1 ， 标签内封闭的元素(elements)会被包含。如果 if 属性 值是 0 ，则 标签内包含的元素会被忽略。 unless 属性的工作方式类似 if 属性，但是含义颠倒：

<group

unless="1-or-0" />

</group>

当然了，通常我们不会给这些属性使用简单的 0 或 1 这样的赋值。建议：结合 arg 的 $() 技术，它们会将你的launch文件的配置变得非常的强大。

- node标签

任何一个launch文件的重点都是：节点(node)元素的集合。启动的每一个节点(node)都要有自己独一无二的名字(name)。如果node(节点)元素有children，就需要显式标签来定义，children比如是：remap 元素或者 param 元素。

显式关闭标签的形式：

<node pkg="" type="" name=""></node>

1

pkg 和 type 它们分别是：程序包名字和可执行文件的名字。 ros::init() 函数 提供的 name 信息将会全面的覆盖命名信息（launch文件中node标签里面的name 属性）

拓展属性:

output = “screen”

Q： 如何将标准输出信息显示在终端(console)上？

A： 在 node 元素中使用 output 属性：

output=”screen”

respawn=”true”

启动完所有请求启动的节点之后，roslaunch 监测每一个节点，让它们保持正常的运行状态。对于每一个节点，当它终止时，我们可以要求 roslaunch 重新启动它.

required=”true”

当一个必需的节点终止时，roslaunch会做出响应，终止其他所有的节点并退出它自己。

注意： 由于 required 属性和 respawn 属性的含义，所以如果你给单个的一个节点同时设置了这2个属性，roslaunch 命令会抱怨。所以不要这样设置。

launch-prefix = “command-prefix”

roslaunch 命令 的一个潜在的缺点：相比我们原来对每个节点在单独的终端使用 rosrun 命令启动的做法，roslaunch 则是让所有的节点共享同一个终端。 那些只需要生产简单的日志消息文件而不需要终端(console)输入的节点是容易管理的，而那些依赖终端输入的节点，比如 turtle\_teleop\_key 节点，它可能要优先的保留在独立的终端上。

在例子launch文件中，我们给 teleoperation 节点使用了这个属性：launch-prefix=”xterm -e”.因为这个属性，启动这个 node 元素的 rosrun 命令大致相当于：xterm -e rosrun turtlesim turtle\_teleop\_key.

xterm 命令会开一个新的终端窗口。 -e 参数告诉 xterm ：执行其命令行剩余部分(rosrun turtlesim turtle\_teleop\_key)。

ns = “namespace”

在例子launch文件中，使用这个属性创建了两个无关的 turtlesim 模拟器。

turtlesim 话题名字(turtle1/cmd\_vel、turtle1/color\_sensor 和 turtle1/pose )被从全局命名空间移动 到 /sim1 和 /sim2 的单独命名空间里。

args属性

传递参数到节点. 个人理解,arg参数只在launch文件中合法(相当于局部变量),不直接传给节点,所以需 要通过node中的args属性进行传递.

- remap标签

在launch文件中重新命名：使用 remap 元素：

<remap from="original-name" to="new-name" />

1

如果这个 remap 是 launch 元素的一个child（子类），与 node 元素同一层级， 并在 launch 元素内的最顶层。那么这个 remapping 将会作用于后续所有的节点。

- include标签

file属性

<include file="path-to-launch-file" />

包含其他文件.这个 file属性期望我们添加想要包含的文件的完整路径。但是大多数时候，include 元素使用一个 find 命令来搜索一个程序包，代替一个明确的完整路径：

<include file="$(find package-name)/launch-file-name" />

这样做是正确的：

<include file = "find learning\_tutrols"/launch/start\_demo.launch" / >

这样做是错误的：

<include file = "find learning\_tutrols"/start\_demo.launch" />

ns属性

include 元素也支持 ns 属性，可以让这个文件里的内容推送到一个命名空间里面：

<include file=". . . " ns="namespace" />

一般我们都会给 include 元素设置一个 ns 属性。

- arg标签

在launch文件中，要声明一个参数的存在，我们使用arg元素：

<arg name="arg-name" />

声明里面只有一个 name 是起不上什么作用的，这就像是你在程序中定义了一个 int 类型的变量，但 是你并没有使用它一样。(你至少还需要给 arg 元素分配 default 属性或 value 属性)

给arg赋值

在launch文件中使用的每一个argument必须给它分配一个 value（赋值）。有以下两种实现方法:

在命令行中你可以这样做：给 roslaunch 提供一个 value :

roslaunch package-name launch-file-name arg-name:=arg-value

另外，在launch文件中，你可以提供一个 value（赋值） 作为 arg 声明的一部分，使用下面的两种语法之一就可以：

<arg name="arg-name" default="arg-value" />

<arg name="arg-name" value="arg-value" />

这两种语法的唯一不同是：命令行可以覆盖default的值，但是不能覆盖 value 。在例子launch文件 中，use\_sim3 节点的 default 值为 0，所以它可以通过命令行改变值，就像下面这样：

$ roslaunch agitr triplesim.launch use\_sim3:=1

如果我们修改了这个例子launch文件：使用value替换default。那么上面这个命令执行的时候会出 现错误，因为使用 value属性 配置的argument的值是不允许改变的。

获取argument的数值

$(arg arg-name)

$() 这个符号出现的任何地方，roslaunch命令都将会把它替换成给定argument 的值(value)。

关于argument的继承

在argument的传递上有一个限制，就是argument不能传递给 include 元素里包含的子launch文件 使用。这个问题非常重要，因为这个 argument 就像是一个局部变量，它不能被包含的launch文件 所 “继承” 。

解决这个问题的方法：在 include 元素中插入 arg 元素作为 include 元素的子类(children)，就像是这样：

<include file="path-to-launch-file">

<arg name="arg-name" value="arg-value"/>

</include>

注意，这里的 arg 元素不同于我们已经知道的 arg 声明，在 inchude 标签内的arguments是给包含 (included) 的launch文件提供的arguments，不是为本launch文件提供的。

一种常见的情况是，被包含(included)的launch文件和本launch文件会有共同的参数。在这种情况下，我们希望这些值(values)永远不变。像这样的元素，在这两个地方使用相同的argument name （参数名），要这样做：

<arg name="arg-name" value="$(arg arg-name)" />

在这种情况下，第一个 arg-name 和往常一样。第二个 arg-name 是launch文件中提供的。结果是，这两个launch文件中给定的argument具有相同的值(value)。

拓展说明

使用 roslaunch 命令 和 使用 rosrun 命令 单独运行每个节点之间的重要区别

默认情况下，roslaunch 命令 从启动节点开始，标准输出信息会重定向到一个日志文件中，而不会像 rosrun 命令那样，将 log 信息显示在终端(console)上.

日志文件所在路径： ∼/.ros/log/run\_id/node\_name-number-stdout.log

Q： 如何将标准输出信息显示在终端(console)上？

A： 在 node 元素中使用 output 属性：

output=”screen”

带这个属性启动的节点会将标准输出信息显示在终端的窗口中，而不会保存在日志文件中。这也 解释 了为什么这个带有output=”screen”的节点（node） 的日志文件在上面日志文件列表中丢失的原因。

扩展： node 元素的 output 属性只能影响这个节点自己。除了 output 属性，我们可以使用 roslaunch命令行工具的 –screen 命令行选项强制性的在终端的窗口中显示所有节点的输出信息。

$ roslaunch --screen package-name launch-file-name

重映射名字

在启动一个节点的时候，有两种方法创建重映射：

在终端命令行中启动一个节点时，要重新给这个节点命名：给出一个节点原来的名字和新的名字， 中间用:=分开。 original-name:=new-name

例如，在运行turtlesim实例时，我们现在想把发布姿态数据的话题/turtle1/pose名称改 为：/tim，那么命令就 是这样的：$ rosrun turtlesim turtlesim\_node turtle1/pose:=tim

在launch文件中重新命名：使用 remap 元素：

<remap from="original-name" to="new-name" />

如果这个 remap 是 launch 元素的一个child（子类），与 node 元素同一层级， 并在launch 元素内的最顶 层。 那么这个 remapping 将会作用于后续所有的节点。

这个 remap 元素也可以作为 node 元素的一个child（子类）出现。下面这个就是使用模板：

<node node-attributes >

<remap from="original-name" to="new-name" />

</node>

例如：上面命令行命令如果在launch文件中，就是下面这个样子的：

<node pkg="turtlesim" type="turtlesim\_node"

name="turtlesim" >

<remap from="turtle1/pose" to="tim" />

</node>

argument和parameter有区别

尽管术语argument和parameter在许多计算机环境中稍微可以互换使用，它们的含义在ROS中有很大的不同。Parameters（参数）在一个运行的ROS系统中是变量(values)，它被存储在parameter服务器中，活动（或者叫：运行）的节点通过ros::param::get()函数访问它，并且用户可以通过 rosparam 命令行工具使用它。相比之下，arguments只有在launch文件里合法，它们的值不是直接提供给节点。

parameter说明

All parameters are “owned” by the parameter server rather than by any particular node. This means that parameters—even those created with private names—will continue to exist even after the node they’re intended for has terminated.

在C++文件中,set或者get处理parameter参数

r o s : : param : : s e t ( ” background\_r ” , 2 5 5 ) ;

b o o l ok = r o s : : param : : g e t (PARAM\_NAME, maxVel ) ;

在launch文件中设置parameter,使用param标签:

<param name="param-name" value="param-value" />

group标签中的param标签的作用等同于rosparam set命令.

node标签中的param标签设置为该节点的子元素.

Parameter names given in param elements that are children of node elements are always resolved as private names, regardless of whether they begin with ∼ or even / .

在launch文件中也支持等同与rosparam load 功能的rosparam标签

用于一次性加载大量的参数

<rosparam command="load" file="path-to-param-file" />

这里的file是.yaml类型.

文件介绍来自于

（<https://blog.csdn.net/qq_33444963/article/details/77893881>）

具体launch文件的含义需要多看一些例程才能慢慢了解。

**Slam过程中三个重要的算法**

Acml 自适应蒙特卡罗定位算法 是一个针对在二维移动的机器人的基于概率定位系统。它实现了自适应蒙特卡罗滤波的定位方法，并使用粒子滤波器去跟踪在已知地图中机器人的位置

Move\_base (DWA算法) 路径规划

Cartographer 地图构建算法 主要理论是通过闭环检测来消除构图过程中产生的累积误差。用于闭环检测的基本单元是submap。一个submap是由一定数量的laser scan构成。将一个laser scan插入其对应的submap时，会基于submap已有的laser scan及其它传感器数据估计其在该submap中的最佳位置。submap的创建在短时间内的误差累积被认为是足够小的。然而随着时间推移，越来越多的submap被创建后，submap间的误差累积则会越来越大。因此需要通过闭环检测适当的优化这些submap的位姿进而消除这些累积误差，这就将问题转化成一个位姿优化问题。当一个submap的构建完成时，也就是不会再有新的laser scan插入到该submap时，该submap就会加入到闭环检测中。闭环检测会考虑所有的已完成创建的submap。当一个新的laser scan加入到地图中时，如果该laser scan的估计位姿与地图中某个submap的某个laser scan的位姿比较接近的话，那么通过某种 scan match策略就会找到该闭环。Cartographer中的scan match策略通过在新加入地图的laser scan的估计位姿附近取一个窗口，进而在该窗口内寻找该laser scan的一个可能的匹配，如果找到了一个足够好的匹配，则会将该匹配的闭环约束加入到位姿优化问题中。Cartographer的重点内容就是融合多传感器数据的局部submap创建以及用于闭环检测的scan match策略的实现。

Cartographer主要有三个文件夹：

1 cartographer\_ros cartographer在ros平台的接口

2 cartographer 核心slam算法

3 ceres-solver google 的开源优化库

还有其他的建图算法如：

1 gmapping

目前激光2Dslam用得最广的方法，gmapping采用的是RBPF的方法. 必须得了解粒子滤波(利用统计特性描述物理表达式下的结果)的方法.

粒子滤波的方法一般需要大量的粒子来获取好的结果,但这必会引入计算的复杂度;粒子是一个依据过程的观测逐渐更新权重与收敛的过程,这种重采样的过程必然会代入粒子耗散问题(depletion problem), 大权重粒子显著,小权重粒子会消失(有可能正确的粒子模拟可能在中间的阶段表现权重小而消失).

自适应重采样技术引入减少了粒子耗散问题 , 计算粒子分布的时候不单单仅依靠机器人的运动(里程计),同时将当前观测考虑进去, 减少了机器人位置在粒子滤波步骤中的不确定性. (FAST-SLAM 2.0 的思想，可以适当减少粒子数)

2 Hector-slam

要求: 高更新频率小测量噪声的激光扫描仪. 不需要里程计,使空中无人机与地面小车在不平坦区域运行存在运用的可能性.

利用已经获得的地图对激光束点阵进行优化, 估计激光点在地图的表示,和占据网格的概率.

其中扫描匹配利用的是高斯牛顿的方法进行求解. 找到激光点集映射到已有地图的刚体转换(x,y,theta).

( 接触的匹配的方法还有最近邻匹配的方法(ICP) ,gmapping代码中的scanmatcher部分有两种方法选择. )

为避免局部最小而非全局最优的(类似于多峰值模型的,局部梯度最小了,但非全局最优)出现,地图采用多分辨率的形式.

导航中的状态估计可以加入惯性测量，进行ＥＫＦ滤波。

3 karto

图优化的核心思想我认为主要就是 矩阵的稀疏化与最小二乘..参见graphslam学习

　　KartoSLAM是基于图优化的方法，用高度优化和非迭代 cholesky矩阵进行稀疏系统解耦作为解．　图优化方法利用图的均值表示地图，每个节点表示机器人轨迹的一个位置点和传感器测量数据集，箭头的指向的连接表示连续机器人位置点的运动，每个新节点加入，地图就会依据空间中的节点箭头的约束进行计算更新．

KartoSLAM的ROS版本，其中采用的稀疏点调整（the Spare Pose Adjustment(SPA)）与扫描匹配和闭环检测相关．landmark越多,内存需求越大,然而图优化方式相比其他方法在大环境下制图优势更大.在某些情况下KartoSLAM更有效,因为他仅包含点的图(robot pose),求得位置后再求map.

4 coreslam

为了简单和容易理解最小化性能损失的一种slam算法.将算法简化为距离计算与地图更新的两个过程, 第一步,每次扫描输入,基于简单的粒子滤波算法计算距离,粒子滤波的匹配器用于激光与地图的匹配,每个滤波器粒子代表机器人可能的位置和相应的概率权重,这些都依赖于之前的迭代计算. 选择好最好的假设分布,即低权重粒子消失,新粒子生成..在更新步骤,扫描得到的线加入地图中,当障碍出现时,围绕障碍点绘制调整点集,而非仅一个孤立点.

5.lagoslam

基本的图优化slam的方法就是利用最小化非线性非凸代价函数.每次迭代, 解决局部凸近似的初始问题来更新图配置,过程迭代一定次数直到局部最小代价函数达到. (假设起始点经过多次迭代使得局部代价函数最小). LagoSLAM 是线性近似图优化,不需要初始假设. 优化器的方法可以有三种选择 Tree-based netORK Optimizer(TORO), g2o,LAGO

以上参考博客（<https://blog.csdn.net/hjwang1/article/details/62236432>）

Slam算法讲解还可以参考博客

（<https://blog.csdn.net/tiandijun/article/details/51221090>）

无法获取锁 /var/cache/apt/archives/lock

$ sudo rm /var/lib/apt/lists/lock

$ sudo killall apt-get

(有时候出现这种情况，把终端关了，再重新打开，就没有问题了)

参考资料

1 Ros官网

2创客智造 这里面有大量的资料和教程可供参考，但都基本是教学性质的，偏应用，不涉及底层具体程序的实现

3 古月居博客

4 相关资料还可以在博客园，csdn中搜索得到。

**安装ros2**

1. Linux安装（debians）

设置软件源，如下：

$ sudo apt update && sudo apt install curl

$ curl http://repo.ros2.org/repos.key | sudo apt-key add -

$ sudo sh -c 'echo "deb [arch=amd64,arm64] http://repo.ros2.org/ubuntu/main xenial main" > /etc/apt/sources.list.d/ros2-latest.list'

使用下面命令安装ROS 2 ardent全部包，注意，ros-ardent-ros1-bridge和ros-ardent-turtlebot2-\*需要ROS 1的支持（需安装ROS Kinetic，需要请点击链接：http://blog.csdn.net/zhangrelay/article/details/51364622）:

$ sudo apt install `apt list ros-ardent-\* 2> /dev/null | grep "/" | awk -F/ '{print $1}' | grep -v -e ros-ardent-ros1-bridge -e ros-ardent-turtlebot2- | tr "\n" " "`

保持足够耐心等待安装结束

3. 环境设置

这个也和ROS 1是保持一致的，如下：

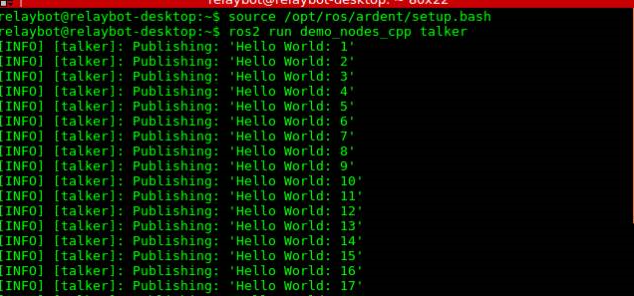
打开 .bashrc文件，写入下面这句

source /opt/ros/ardent/setup.bash

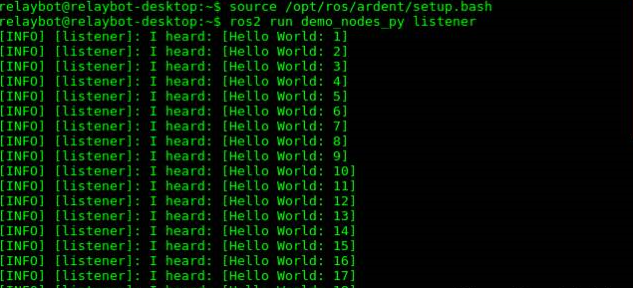
然后$ source ~/.bashrc

在打开终端的时候可能会有一个提示： ROS\_DISTRO was set to ‘ardent’ before.please make sure that the environment does not mix paths from different distributions.(可以不用管)

最后打开终端运行 $ ros2 run demo\_nodes\_cpp talker



打开另外一个终端： $ ros2 run demo\_nodes\_py listener



5. 使用ROS 1功能包的附加包

在ROS 2和ROS 1都安装配置完成的系统，可以通过ros1\_bridge同时使用，如果使用Docker可以避免环境之间的切换等配置，安装如下包：

$ sudo apt install ros-ardent-ros1-bridge ros-ardent-turtlebot2-\*

备注：不要使用下载二进制包的方法来安装，我试过了，节点运行不了。