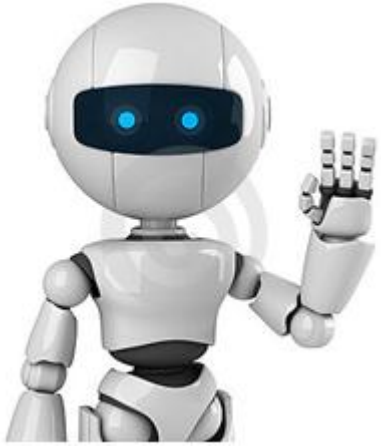
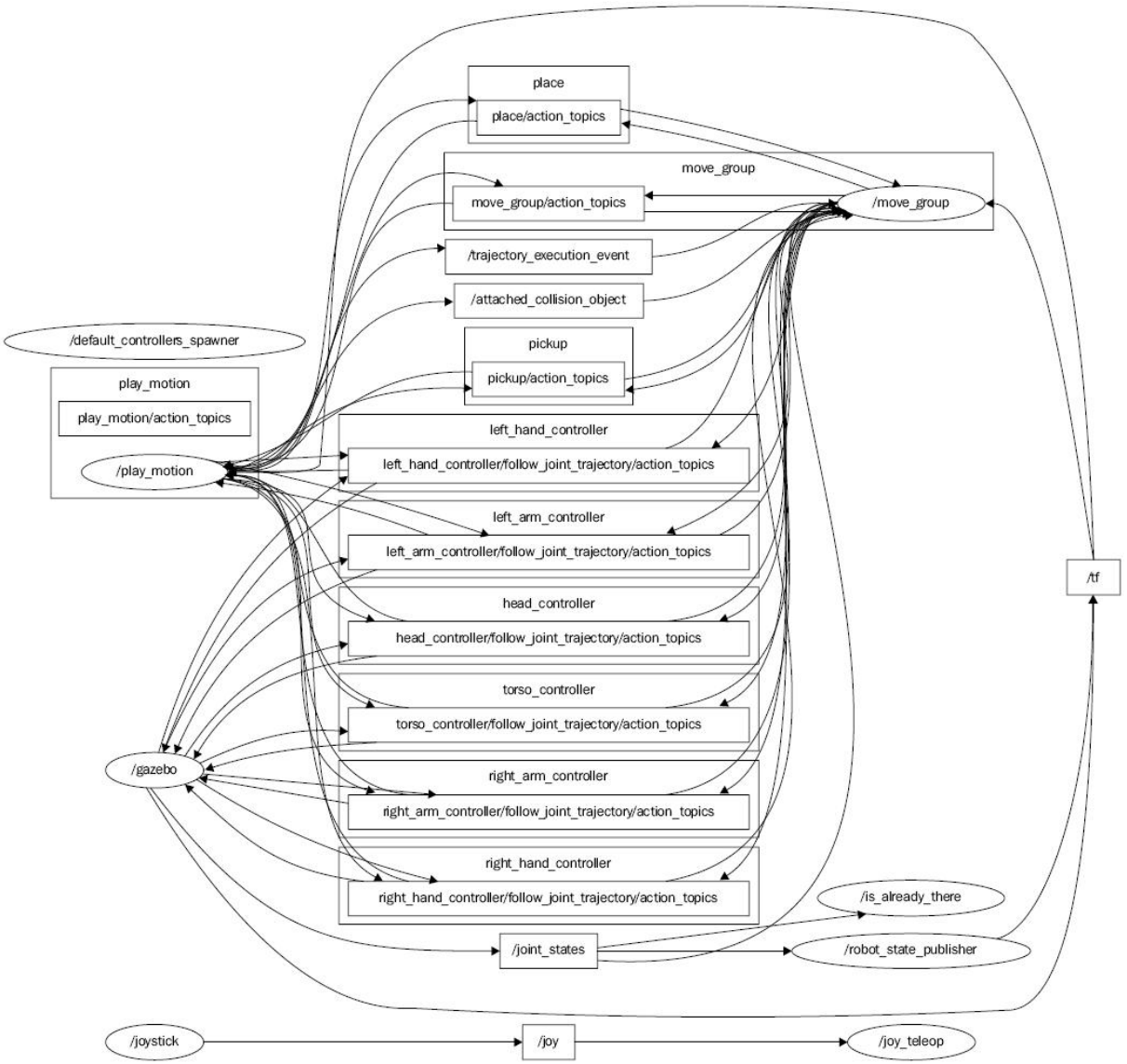
3(1)：ROS可视化和调试工具



东北大学 张云洲

内 容 提 纲

- gdb调试器
- 日志及调试信息
- 检测系统状态
- 绘制标量数据图
- 3D可视化（Rviz）
- 保存和回放数据


place
place/action_topics
move_group
move_group/action_topics
/move_group
/trajectory_execution_event
/attached_collision_object
/default_controllers_spawner
pickup
pickup/action_topics
play_motion
play_motion/action_topics
/play_motion
left_hand_controller
left_hand_controller/follow_joint_trajectory/action_topics
left_arm_controller
left_arm_controller/follow_joint_trajectory/action_topics
/tf
head_controller
head_controller/follow_joint_trajectory/action_topics
torso_controller
torso_controller/follow_joint_trajectory/action_topics
/gazebo
right_arm_controller
right_arm_controller/follow_joint_trajectory/action_topics
right_hand_controller
right_hand_controller/follow_joint_trajectory/action_topics
/is_already_there
/joint_states
/robot_state_publisher
/joystick
/joy
/joy_teleop

3.1 概述

ROS软件框架附带了大量功能强大的工具，帮助用户和开发人员检测软硬件问题并完成代码调试。

调试工具包括：

- 日志消息记录与展示工具
- 数据可视化工具
- 系统监测组件。

用户可以以自己喜欢的方式查看系统运行状态。

3.1 概述

在程序编译和运行时仍然可能会失败。

两个关键的问题：（1）与节点、主题、服务或任何其他ROS组件有关；（2）算法本身也可能导致问题。

ROS提供了一组功能强大的工具来监视系统状态：

- 节点状态图
- 通用绘图工具（时序，rviz，图像展示）

注意：数据相关于特定的传感器坐标空间，并通过坐标来呈现数据。一个机器人一般由多个坐标空间组成，坐标空间之间能够进行变换。

3.2 gdb调试

ROS 的灵活性允许使用GDB调试器对常规的C/C++程序进行调试，唯一要知道的是可执行文件的路径。

$ gdb example1

(gdb) r

一旦roscore已经运行，可以点击R键和Enter键从GDB启动节点；也可以用L键列出相关源代码，以及设置断点或使用任何GDB附带的功能。

gdb调试器

基本命令	描述
gdb	打开调试器
file FILE	装载指定可执行文件
r	代表run,从头开始运行程序直到断点。在一次debug中可以用 r 来多次重新运行程序，而不必重新rosrun或 roslaunch.
q	退出debug。
bt	列出调用堆栈。

gdb调试器

断点命令	描述
b	b即break。在当前行设置断点。
b 45	在某行设置断点。
b functionName	在某函数开始处设置断点。

常用：

b main 在程序开始设置断点。

watch x == 3	设置条件断点。
info break	查看当前存在的断点。每个断点都有自己的编号。
delete N	删除编号为N的那个断点。

gdb调试器

命令	描述
n	“next”。运行一行代码。相当于VS的step over。
s	“step”。运行一个指令。相当于VS的step in。 n和s都可以一次运行多行，比如n 5。
c	“continue”。继续运行直到下一个断点。
f	“finish”，运行完当前程序。

用launch文件启动gdb：**使用XML语法修改启动文件中的节点属性，在节点启动时调用GDB调试器。**

```
<?xml version="1.0" encoding="UTF-8"?>

<launch>
  <!-- Logger config -->
  <env name="ROSCONSOLE_CONFIG_FILE"
       value="$(find chapter3_tutorials)/config/chapter3_tutorials.config"/>

  <!-- Example 1 -->
  <node pkg="chapter3_tutorials" type="example1" name="example1"
        output="screen" launch-prefix="xterm -e gdb --args"/>
</launch>
```

*output=“screen”，可以让节点输出到屏幕。

运行：

~/dev/catkin_ws$ roslaunch chapter3_tutorials example1_gdb.launch

3.3 信息输出

- 有不同层级的调试信息输出，每条信息都有自己的名称，并根据相应条件输出消息，对性能没有任何影响。
- 这些工具无缝集成在并发执行的节点上，也就是说信息没有被打散，而是完全可以根据时间戳进行交叉展示。
- ROS自带大量的能够输出调试信息的函数和宏。这些信息包括错误、警告或简单的提示信息。它提供如信息（或日志）级别、条件触发消息和STL的流接口等诸多方式。

3.3 信息输出

想要输出任何提示信息，只需在代码中的任意位置插入：

ROS_INFO("My INFO message.");

注意，只要主要的ROS头文件被包括在内，在源代码中就不需要包括任何特定的库。但可以添加ros/console.h头文件：

```
#include <ros/console.h>
```

[INFO] [1356440230.837067170]: My INFO message.

所有输出的调试信息都附带其级别和当前时间戳。**时间戳以公历时间计时，代表着自1970年以来的秒和纳秒计数，**时间戳之后是具体的信息。

3.3 信息输出

ROS_INFO函数允许以C语言printf函数的方式增加参数，这意味着可以用printf格式为特殊字符传递数值。

例如，可以按照下面代码输出变量val对应的浮点数值：

```
float val = 1.23;
ROS_INFO("My INFO message with argument: %f", val);
```

或：

```
ROS_INFO_STREAM("My INFO message with argument: " << val);
```

3.3 信息输出

- ROS有五个标准信息级别，按照顺序排列分别是
- **DEBUG、INFO、WARN、ERROR、FATAL。**
- 这些名称是输出信息的函数或宏的一部分，遵循语法：
- ROS_<LEVEL>[_<OTHER>]
- DEBUG和INFO 信息输出到标准输出cout（或stdout）；WARN、ERROR、FATAL输出到错误信息输出cerr/stderr。
- 每一种信息会以特定的颜色输出在屏幕上：DEBUG为绿色，INFO为白色， WARN为黄色， ERROR为红色，以及FATAL为紫色。

3.3 信息输出

- **默认情况下，会显示INFO及更高级别的调试信息**，并使用ROS默认级别来过滤特定节点所需输出的信息。
- **可以在编译时动态地改变级别：**使用编译时宏记录去除器，通过设置CMakeLists.txt文件中ROSCONSOLE_MIN_SEVERITY宏实现。宏定义包括包括：

ROSCONSOLE_SEVERITY_DEBUG、
ROSCONSOLE_SEVERITY_INFO、
ROSCONSOLE_SEVERITY_WARN、
ROSCONSOLE_SEVERITY_ERROR、
ROSCONSOLE_SEVERITY_FATAL
ROSCONSOLE_SEVERITY_NONE。

3.3 信息输出

可以在同一节点的很多位置放置信息，ROS允许为每一条信息取名，从而知道每一条信息是来自于哪一段代码。实现这个功能，可使用<LEVEL>[_STREAM]_NAMED函数，例：

```
ROS_INFO_STREAM_NAMED(
"named_msg",
"My named INFO stream message; val = " << val
);
```

可以在config文件中为每个带有名称的信息设置不同调试级别。这允许在节点定义文件中使用信息的名称对信息进行配置。例：设定named_msg的信息只在ERROR及更高级别中显示：

```
log4j.logger.ros.chapter3_tutorials.named_msg=ERROR
```

3.3 信息输出

- 日志信息集成了一系列的窗口化显示及配置工具。

ROS 框架附带了名为 rosconsole的API。

- 除了能在节点中配置日志记录信息的 API之外，ROS 还在rqttools功能包下提供一种图形化工具：

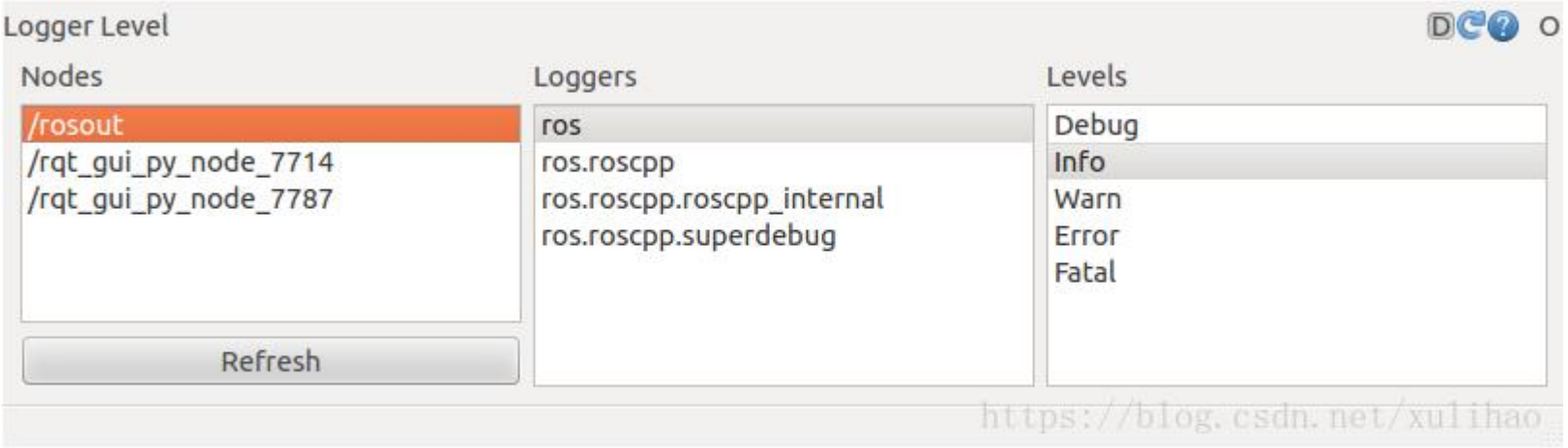
rqt_console——

图形界面允许查看、监视和配置所有正在运行的节点日志记录子系统。

3.3 信息输出

设置日志记录器的严重级别：

$ rosrun rqt_logger_level rqt_logger_level



改变严重级别后，rqt_console界面的信息输出实时反应。

使用rqt_console和rqt_logger_level:

rqt_console与ROS的日志框架（logging framework）有关，可以显示节点的输出

rqt_logger_level可以在节点运行的时候改变他们的verbosity level

在开始turtlesim之前，在两个terminal中运行rqt_console和rqt_logger_level

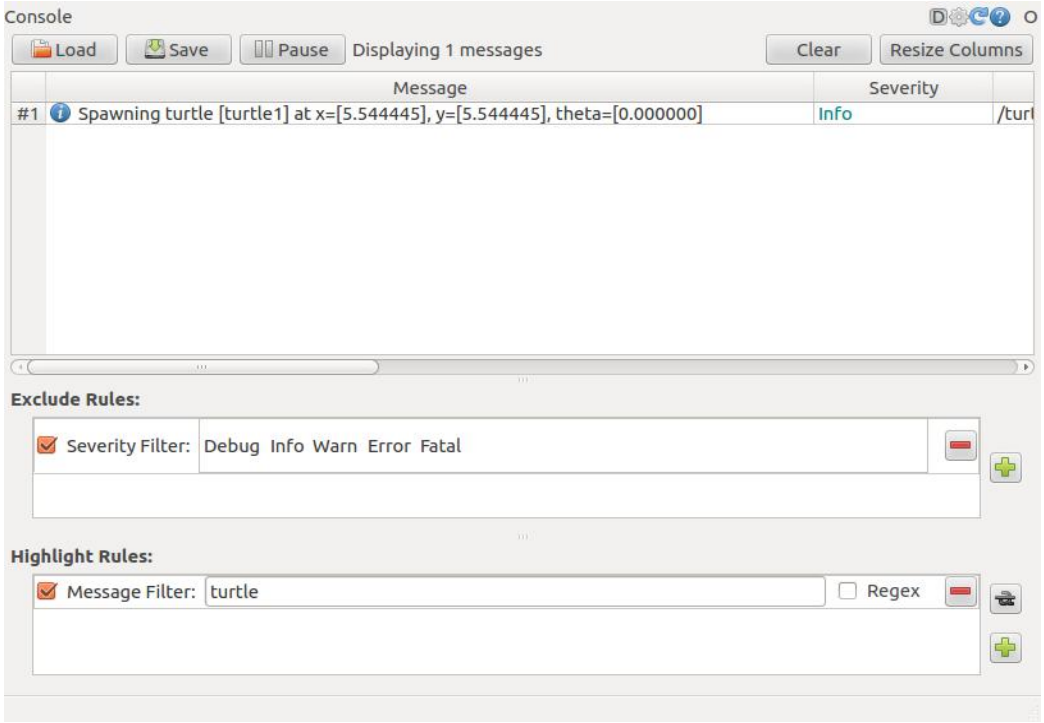
$ rosrun rqt_console rqt_console
$ rosrun rqt_logger_level rqt_logger_level

在新的terminal运行turtlesim：

$ rosrun turtlesim turtlesim_node

默认的logger level是INFO，

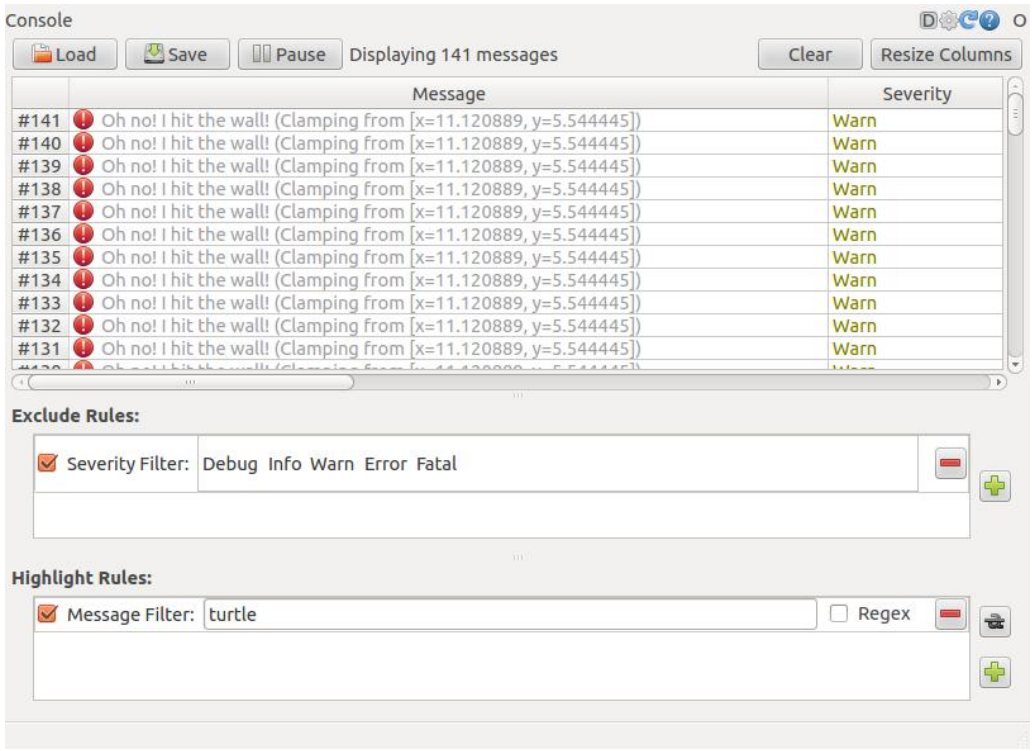
可以看到有关turtlesim发布的任何信息，如：



将logger level改成Warn，输入以下代码让乌龟碰上墙：

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: 0.0}}'

结果如图：



logger levels的优先级顺序

Fatal
Error
Warn
Info
Debug

logger level选了Warn之后会得到：

Warn, Error和 Fatal的logging messages.（相同以及更高）

3.4 监视系统状态

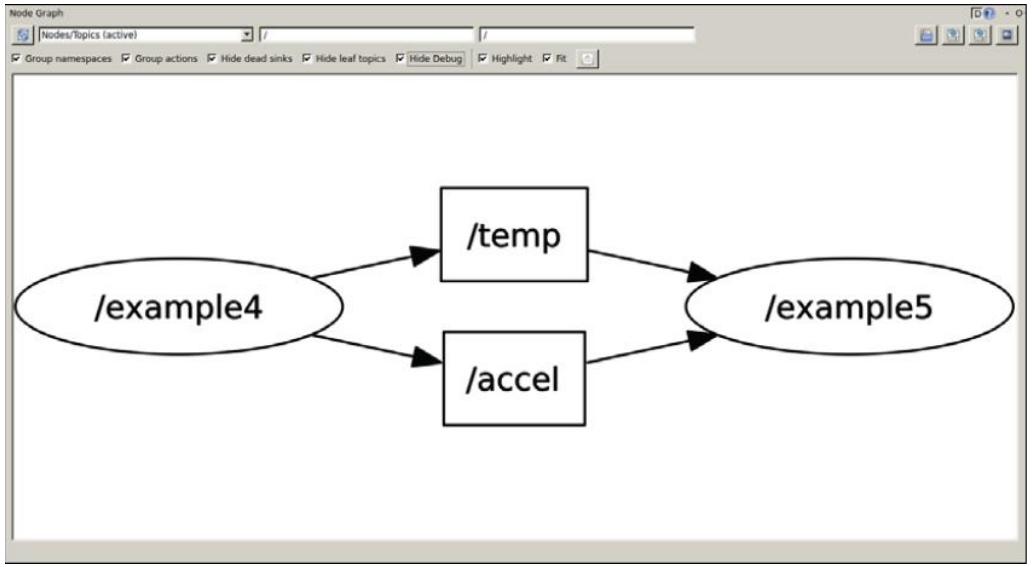
•当系统运行时，可能有数个节点和数十个主题在发布消息，通过订阅与其他节点连接。同时，有些节点可能也会提供服务。

•ROS提供了基本而又非常强大的工具，不仅能实现状态监视，还能**探测节点状态图中任何节点发生的功能失效。**简言之，使用主题来连接节点展现系统架构的状态。

•**能够说明ROS会话当前状态的最简单方法是用一个有向图**，它显示在系统中运行的节点和这些节点通过主题实现的发布者到订阅者的连接。

3.4 监视系统状态

ROS 框架提供了一些工具来生成此类节点状态图。其最主要工具是 rqt_graph，可以在系统执行过程中显示该节点状态图，并使我们看到一个节点是如何动态地出现或消失。



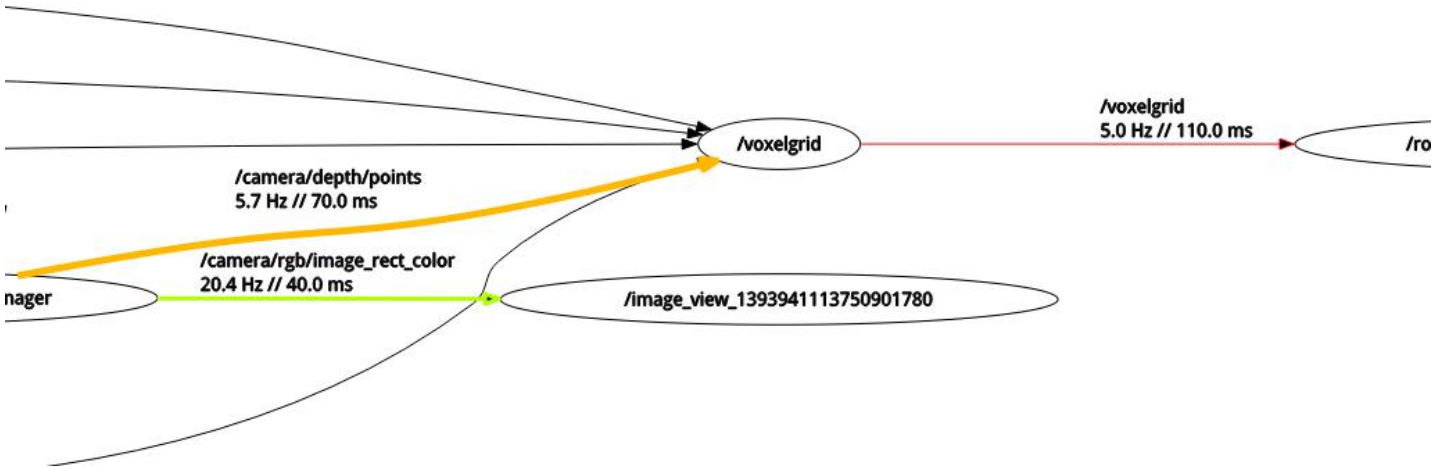
3.4 监视系统状态

添加统计信息：

在主题旁边显示信息速率、带宽、以及写入速率等信息。

$ rosparam set enable_statistics true

提示：在rqt_graph之前运行。

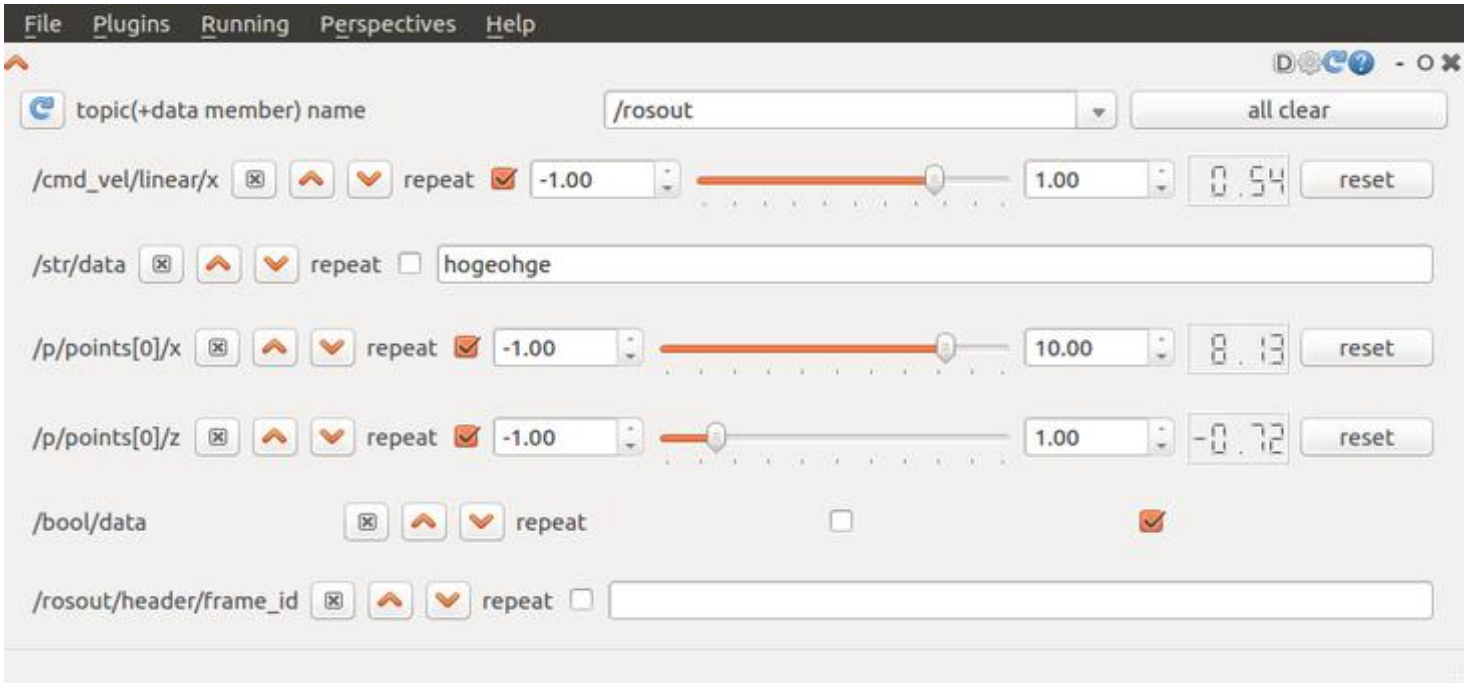


rqt-ez-publisher工具：

$ sudo apt-get install ros-kinetic-rqt-ez-publisher

$ rm ~/.config/ros.org/rqt_gui.ini

$ rosrun rqt_ez_publisher rqt_ez_publisher



example5节点

3.5 roswtf——检测错误

roswft可以检查系统是否存在错误：

$ roscd

$ roswtf

```
Stack: ros
================================================================================
Static checks summary:

No errors or warnings
================================================================================

Cannot communicate with master, ignoring graph checks
```

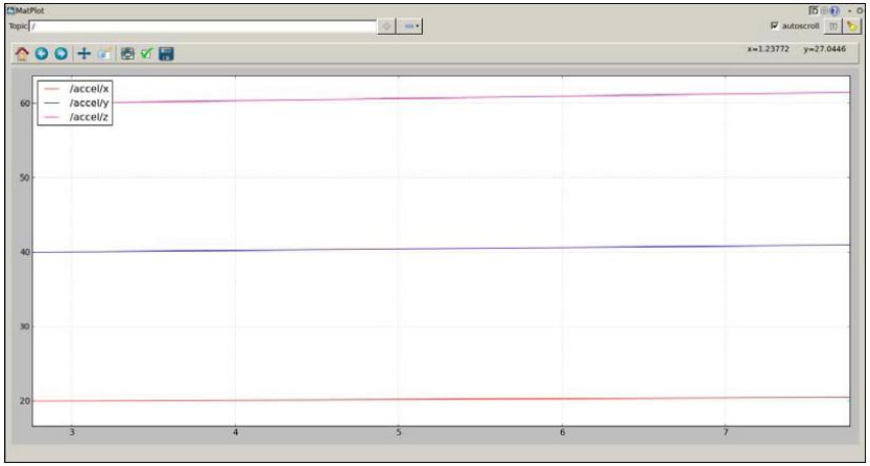
roswtf会警告那些看起来可疑但是可能在系统里是正常的内容，还会把它知道是错误的东西通过error报告出来。

例如：

$ ROS_PACKAGE_PATH=bad:$ROS_PACKAGE_PATH roswtf

3.6 绘制标量数据图

可以使用ROS中现有的一些通用工具，轻松地绘制标量数据图。它要求对每一个标量字段数据分别绘制成二维曲线。



实验：$ rosrun chapter3_tutorials example5　　（server）

$ rosrun chapter3_tutorials example4　　（client）

$ rosrun rqt_plot rqt_plot /temp/data

$ rosrun rqt_plot rqt_plot /accel/x:y:z

多窗：$ rosrun rqt_gui rqt_gui

rqt_plot不能运行时，可直接运行rqt。

3.7 图像可视化

在ROS系统中，可以创建一个节点，在节点中展示来自即插即用摄像头的图像。

example6节点：通过调用OpenCV库插入一段基本的摄像头捕捉程序，然后在ROS中将采集到的cv::Mat图像转换到ROS图像，就可以在主题中发布。该节点会在/camera主题里发布图像帧。

使用launch文件运行节点并进行图像捕捉和发布工作。

在节点运行起来后，可以列出主题列表(rostopic list)，查看是否包含/camera 主题。

查看是否正确捕捉到图像：

$rostopic hz /camera 在主题中收到的图像在哪个频率下，通常是30Hz。

3.7 图像可视化

在ROS系统中，可以创建一个节点，在节点中展示来自即插即用摄像头的图像。

example6节点：通过调用OpenCV库插入一段基本的摄像头捕捉程序，然后在ROS中将采集到的cv::Mat图像转换到ROS图像，就可以在主题中发布。该节点会在/camera主题里发布图像帧。

使用launch文件运行节点并进行图像捕捉和发布工作。

在节点运行起来后，可以列出主题列表(rostopic list)，查看是否包含/camera 主题。

3.7 图像可视化

查看是否正确捕捉到图像：

$rostopic hz /camera 在主题中收到的图像在哪个频率下，通常是30Hz。

测试代码：

roslaunch chapter3_tutorials example8.launch

rostopic hz /camera

显示单张图片：

rosrun image_view image_view image:=/camera //摄像头显示图像

rosrun rqt_image_view rqt_image_view //图形化GUI

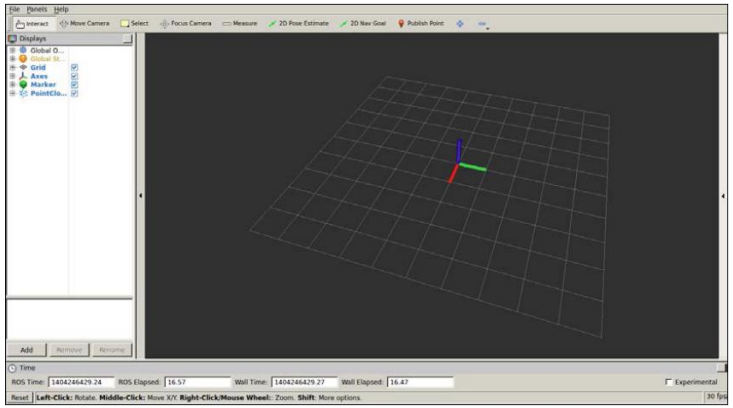
3.8 3D可视化

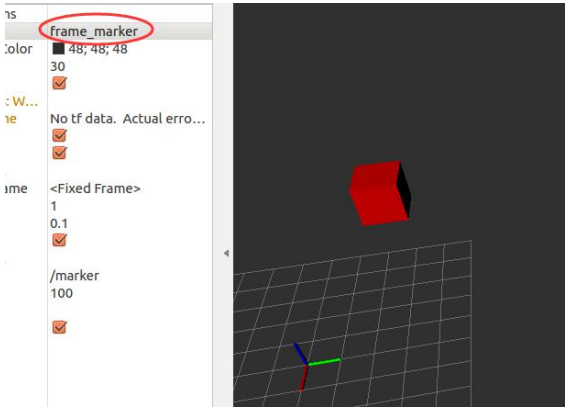
rviz工具集成了能够完成3D数据处理的OpenGL接口，能够将传感器数据在模型化世界（world）中展示。

运行：rviz

配置：config/example9.rviz

运行：roslaunch chapter3_tutorials example9.launch





3.9 保存与回放数据

ROS能够存储所有节点通过主题发布的消息。它能够创建一个消息记录包文件来保存消息，并包含消息的所有字段参数和时间戳。这允许离线回放实验并模拟真实的状态，包括消息的时间延迟。

- 消息记录包是一个包含各个主题所发消息的容器，用于记录机器人各个节点间的会话过程。
- 消息记录包中存储的数据使用二进制格式。
- 每个消息与发布它的主题都被记录下来。

$ rosbag record -a

$ rosbag record /temp /accel　　按Ctrl-C退出该命令，即结束数据记录。

[INFO] [1404248014.671339658]: Subscribing to /accel

[INFO] [1404248014.674950564]: Recording to 2014-07-01-22-54-34.bag.

3.9 保存与回放数据

有了一个消息记录包文件，可以使用它回放所记录主题发布的所有消息数据。只需要运行roscore，而不需要再运行其他任何节点。

$ rosbag play 2014-07-01-22-54-34.bag

——可以看到下列输出：

[INFO] [1404248314.594700096]: Opening 2014-07-01-22-54-34.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.

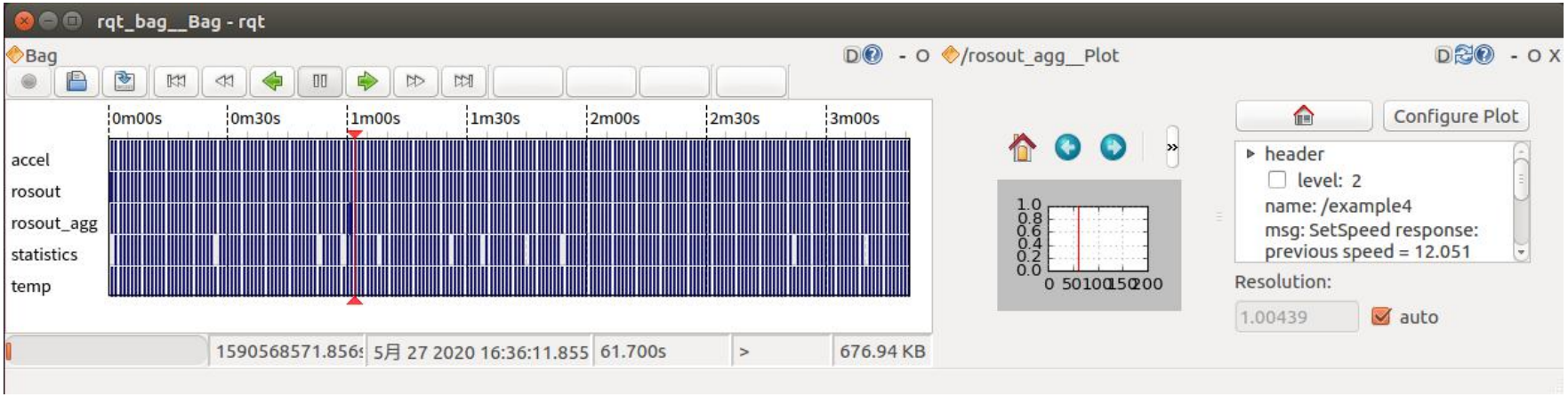
[RUNNING] Bag Time: 1404248078.757944 Duration: 2.801764 / 39.999515

图形化观察：**rqt_bag** **rqt_viz**

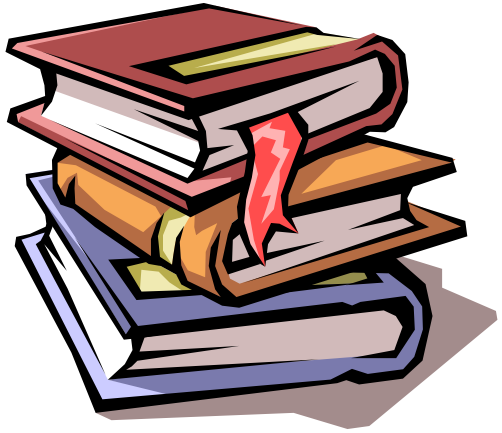
3.9 保存与回放数据

图形化观察：**rqt_bag**

右键：**view --> plot**



rqt_rviz：激光雷达数据。



Q & A
Thanks!