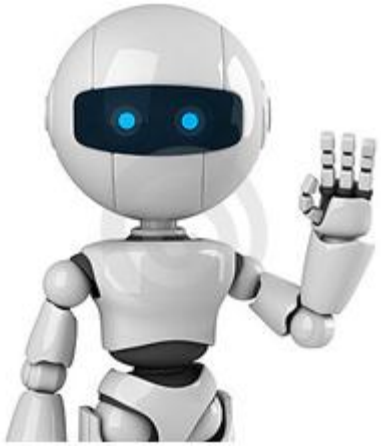
4. ROS环境下的传感器



东北大学 张云洲

作业：

以摄像头（USB或笔记本自带的）作为输入设备，识别你的手势并控制Turtlesim节点做不同的运动。

1. 至少支持3种手势，每个手势对应一个运动模式；
2. 提供一份文档，列出a.源码，b.各手势对应的效果截图；和c.你在调试时遇到的问题及解决过程。
3. 提交形式：压缩包，含源代码和文档。

三名同学为一个小组，自由组合。

评分：按组进行，组内依次差2分。

提示：实现方式不限，可以用OpenCV。

内 容 提 纲

4.1 使用游戏杆或游戏手柄

4.2 激光雷达——rplidar

4.3 RGB-D传感器——Kinect

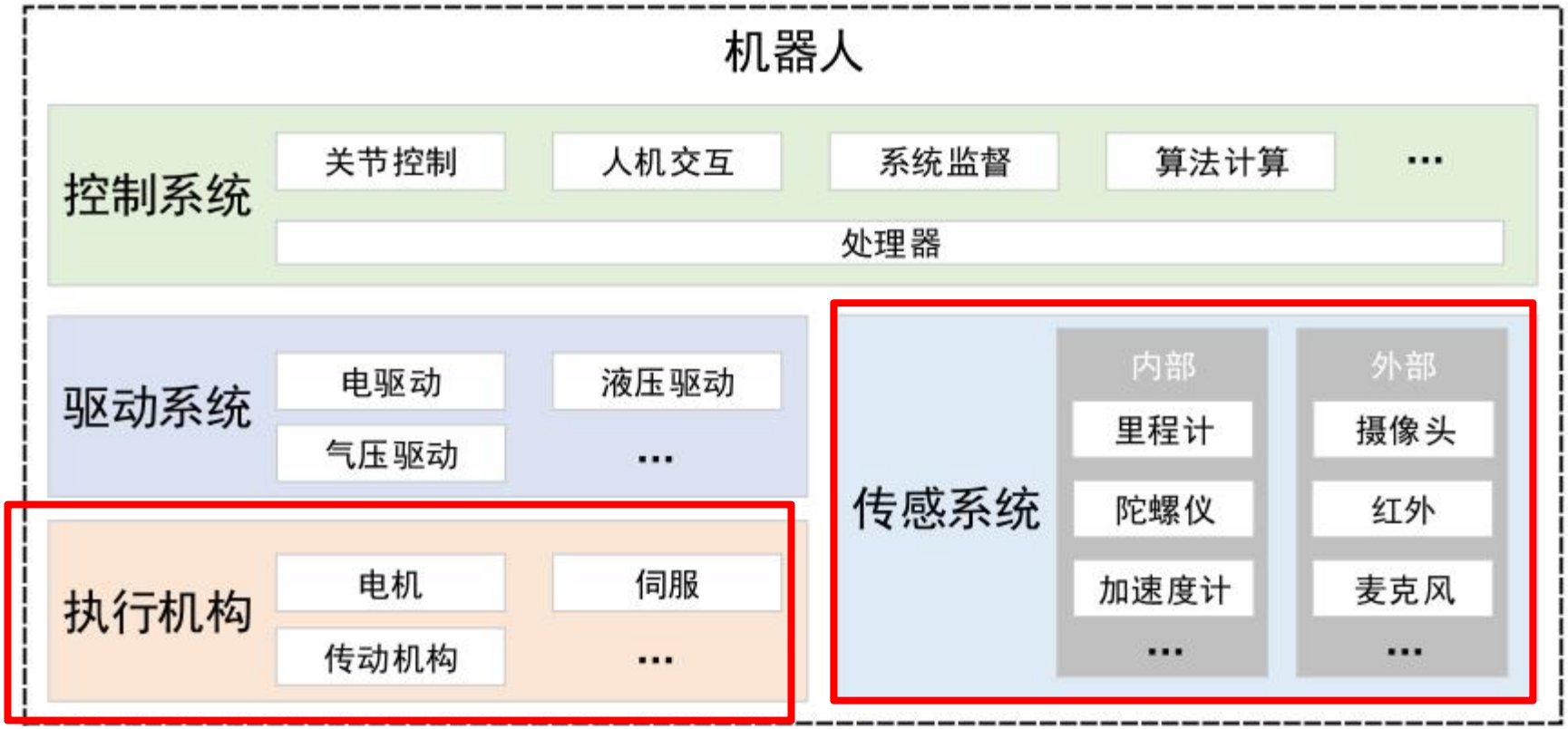
4.4 伺服电动机——Dynamixel

4.5 Arduino平台上使用超声波传感器

4.6 使用惯性测量模组IMU

4.7 通过Arbotix控制板控制电机

4.8 GPS的使用


机器人
控制系统
关节控制
人机交互
系统监督
算法计算
...
处理器
驱动系统
电驱动
液压驱动
气压驱动
...
执行机构
电机
伺服
传动机构
...
传感系统
内部
里程计
陀螺仪
加速度计
...
外部
摄像头
红外
麦克风
...

机器人的组成（控制角度）






机器人系统示例

4.1 使用游戏杆控制TurtleSim

通过输入设备实现某个对象的动作控制：

1. 被控对象接收什么数据？
2. 如何获得输入设备的数据？
3. 如何在输入设备和被控对象之间建立联系？

确定Turtlesim接收的数据类型

1、启动turtlesim节点，查看它订阅的话题。

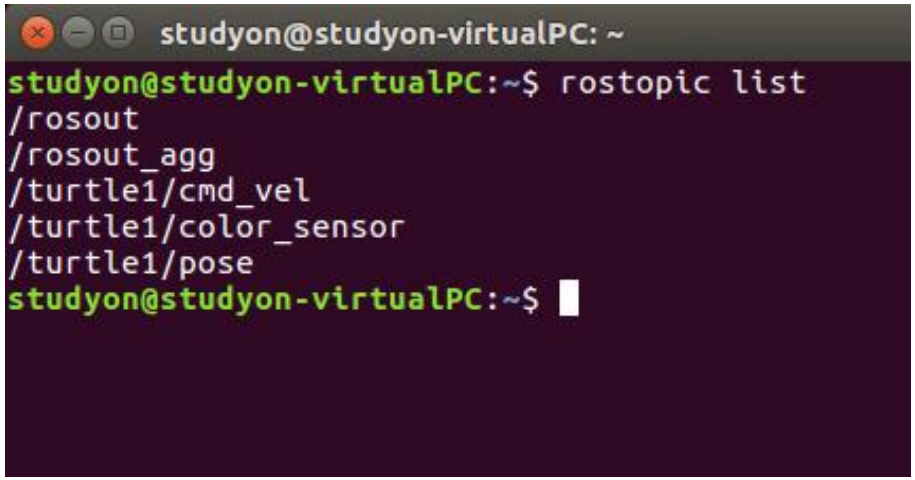
$ roscore

发布的消息所在主题的名称

$ rosrun turtlesim turtlesim_node

查看主题列表

$ rostopic list



确定Turtlesim接收的数据类型

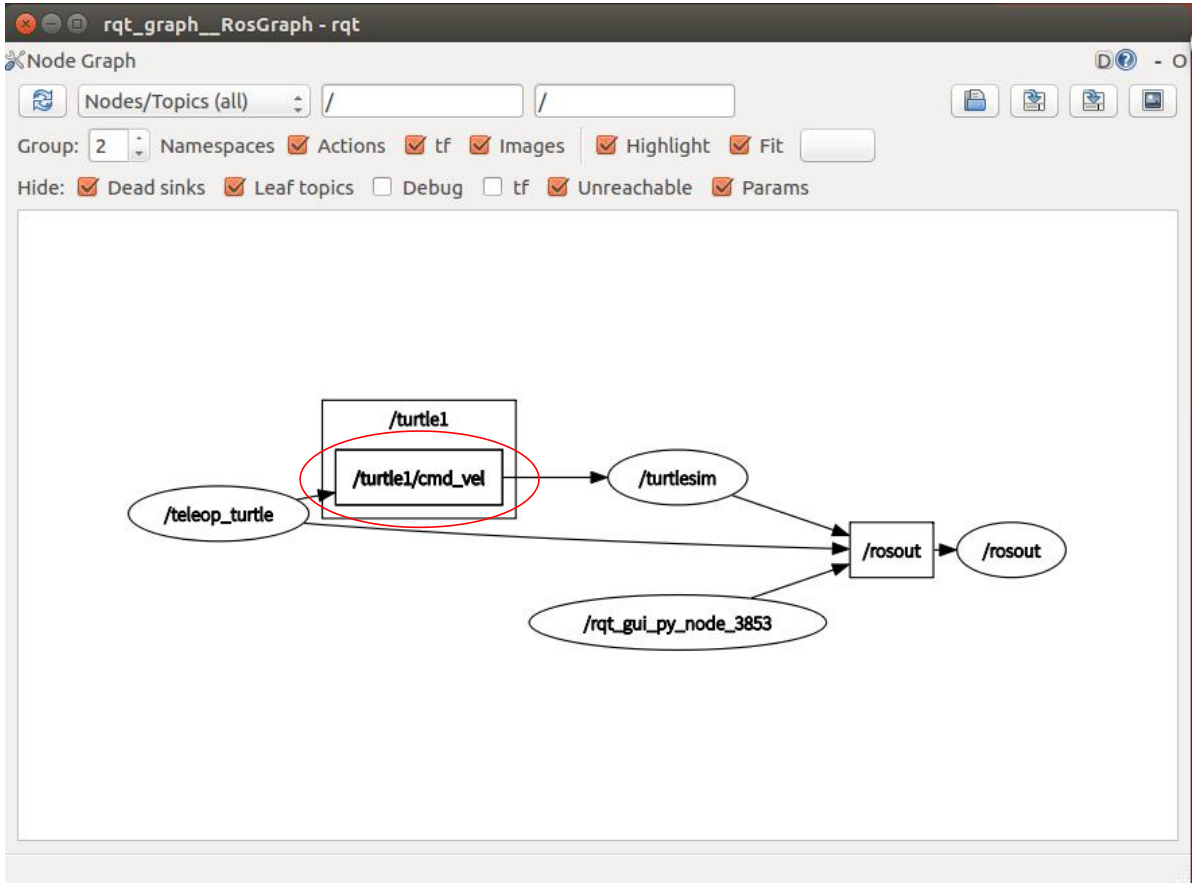
哪一个才是控制需要订阅的话题？

启动一个新的节点turtle_teleop_key，用键盘的方向键来控制乌龟的运动：

```
$ rosrun turtlesim turtle_teleop_key
```

确定Turtlesim接收的数据类型

$ rqt_graph　观察节点之间的topic联系。



确定Turtlesim接收的数据类型

查看主题类型

$ rostopic type turtle1/cmd_vel

查看消息内容

$ rosmsg show geometry_msgs/Twist

```
studyon@studyon-virtualPC: ~
studyon@studyon-virtualPC:~$ rosmsg show  geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

观察：键盘按下时，哪些数据在变化？

rostopic echo /turtle1/cmd_vel

```
studyon@studyon-virt
studyon@studyon-virtualPC:

linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -2.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
```

如何获得输入设备（游戏杆）的数据?

安装驱动

$ sudo apt-get install ros-kinetic-joystick-drivers

$ rosstack profile & rospack profile

检验手柄能否被识别

$ ls /dev/input/

```
by-id    event0  event2  event4  event6  event8  js0   mouse0
by-path  event1  event3  event5  event7  event9  mice
```

检验是否正常工作

$ sudo jstest 666 /dev/input/js0

```
Axes:  0:  0  1:  0  2:  0 Buttons:  0:off  1:off  2:off  3:off  4:off
5:off  6:off  7:off  8:off  9:off  10:off
```

ROS系统下测试

$ rosrun joy joy_node

```
[ INFO] [1357571588.441808789]: Opened joystick: /dev/input/js0.
deadzone_: 0.050000.
```

查看节点发布消息

$ rostopic echo /joy

输出结果：

```
header:
seq: 157
stamp:
 secs: 1357571648
 nsecs: 430257462
 frame_id: ''
axes: [-0.0, -0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

查看消息类型

$ rostopic type /joy

查看消息中使用的字段

$ rosmsg show sensor_msgs/Joy

输出结果：

```
std_msgs/Header
header
 uint32 seq
 time stamp
 string frame_id
float32[] axes
int32[] buttons
```

如何在输入设备和被控对象之间建立联系?

已有：

- joystick发布话题/joy
- 乌龟模拟器订阅的话题是/turtle1/cmd_vel

需要：

写一个中间节点——

1. 订阅/joy话题和接收发过来的数据；

2. 发布一个/turtle1/cmd_vel名称的话题让乌龟模拟器订阅它，从而把数据传给乌龟模拟器，实现joystick控制乌龟模拟器的demo。

编写程序从游戏杆中获取数据生成速度

代码部分：

```
#include<ros/ros.h>
#include<geometry_msgs/Twist.h>
#include<sensor_msgs/Joy.h>
#include<iostream>
using namespace std;
class TeleopJoy{
public:
    TeleopJoy();
private:
    void callBack(const sensor_msgs::Joy::ConstPtr& joy);
    ros::NodeHandle n;
    ros::Publisher pub;
    ros::Subscriber sub;
    int i_velLinear, i_velAngular;
};
```

发布者

订阅者

这些变量使用参数服务器中的数据进行填充
用于表示游戏杆轴向输入

```
TeleopJoy::TeleopJoy()
{
    n.param("axis_linear",i_velLinear,i_velLinear);
    n.param("axis_angular",i_velAngular,i_velAngular);
   pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel",1);
   sub = n.subscribe<sensor_msgs::Joy>("joy", 10, &TeleopJoy::callBack, this);
}
void TeleopJoy::callBack(const sensor_msgs::Joy::ConstPtr& joy)
{
   geometry_msgs::Twist vel;
   vel.angular.z = joy->axes[i_velAngular];
   vel.linear.x = joy->axes[i_velLinear];
   pub.publish(vel);
}
```

使用geometry_msgs::Twist类型发布主题

通过名为joy的主题得到数据

获取游戏杆轴向输入的值

发布主题

主程序：

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "teleopJoy");
    TeleopJoy teleop_turtle;
    ros::spin();
}
```

创建TeleopJoy类的一个实例

3、创建launch文件

为参数服务器声明一些变量
并启动joy和example1节点

代码部分：

```
<launch>
<node pkg="turtlesim" type="turtlesim_node" name="sim"/>
<node pkg="chapter4_tutorials" type="c4_example1"
name="c4_example1"
/>
<param name="axis_linear" value="1" type="int" />
<param name="axis_angular" value="0" type="int" />
<node respawn="true" pkg="joy"type="joy" name="teleopJoy">
 <param name="dev" type="string" value="/dev/input/js0" />
 <param name="deadzone" value="0.12" />
</node>
</launch>
```

axis_linear和axis_angular两个参数
将会用于配置游戏杆的轴向输入

dev参数用于配置游戏杆连接的端口
deadzone参数用于配置不能被设备识别的运动区域

launch文件运行

$ roslaunch chapter4_tutorials example1.launch

4.2 激光雷达（Hokuyo UST-10LX)

激光雷达 UST-10LX

性能参数：

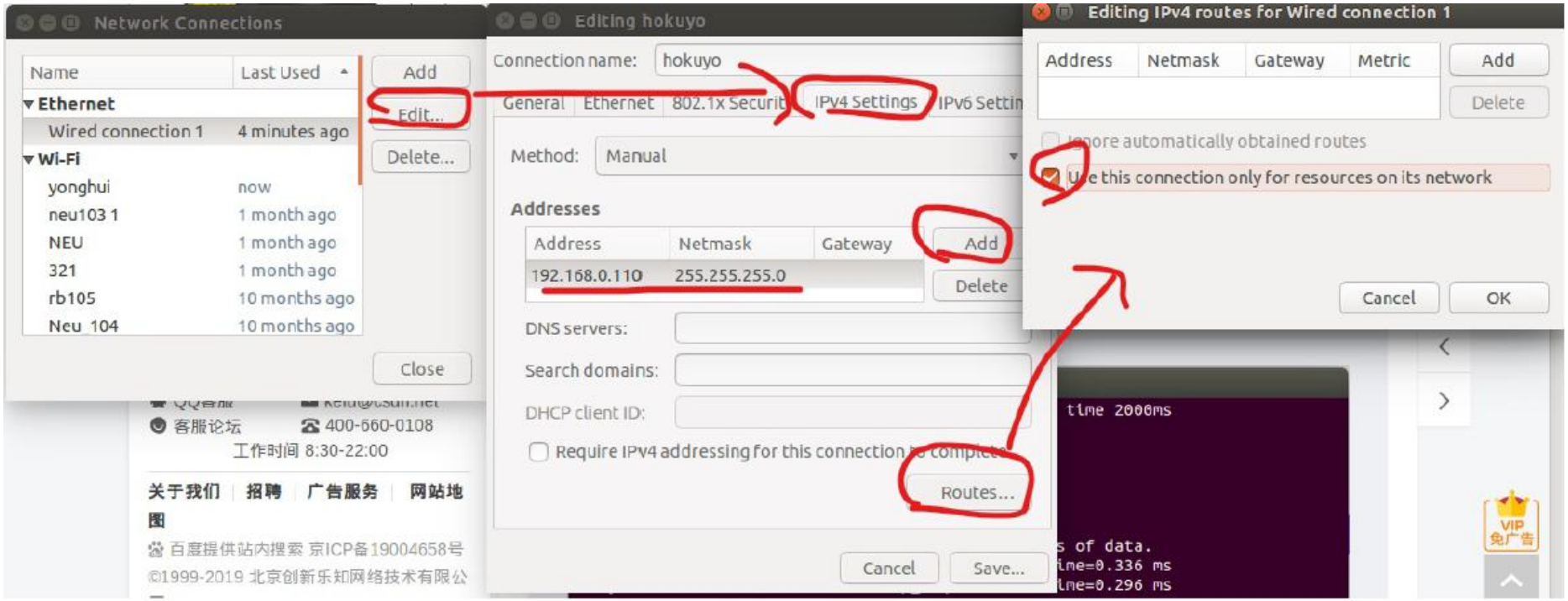
- 测距范围10m
- 测量范围270°
- DC12V/24V输入
- 扫描时间25ms
- IP65防护等级
- 非接触式测量。

1. 安装驱动

$ sudo apt-get install ros-kenetic-freenect-camera ros-kenetic-freenect-stack ros-kenetic-freenect-launch

2. 配置网络：

设置本机和激光雷达连接的固定ip，UST-10LX默认IP为192.168.0.10，因此设置本机IP为192.168.0.XXX， XXX 不是10就行。



3. 测试激光雷达的连接状态：

确保本机能够ping通雷达：

$ping 192.168.0.10

4. 连接正常：

```
PING 192.168.0.10 (192.168.0.10) 56(84) bytes of data.
64 bytes from 192.168.0.10: icmp_seq=1 ttl=64 time=0.126 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=64 time=0.197 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=64 time=0.197 ms
64 bytes from 192.168.0.10: icmp_seq=4 ttl=64 time=0.195 ms
64 bytes from 192.168.0.10: icmp_seq=5 ttl=64 time=0.202 ms
```

5. 启动激光雷达：

$roscore

$rosrun urg_node urg_node _ip_address:=192.168.0.10

* 雷达如何发送数据

$ rostopic list

$ rostopic type /scan

$ rosmsg show sensor_msgs/LaserScan

$ rostopic echo /scan

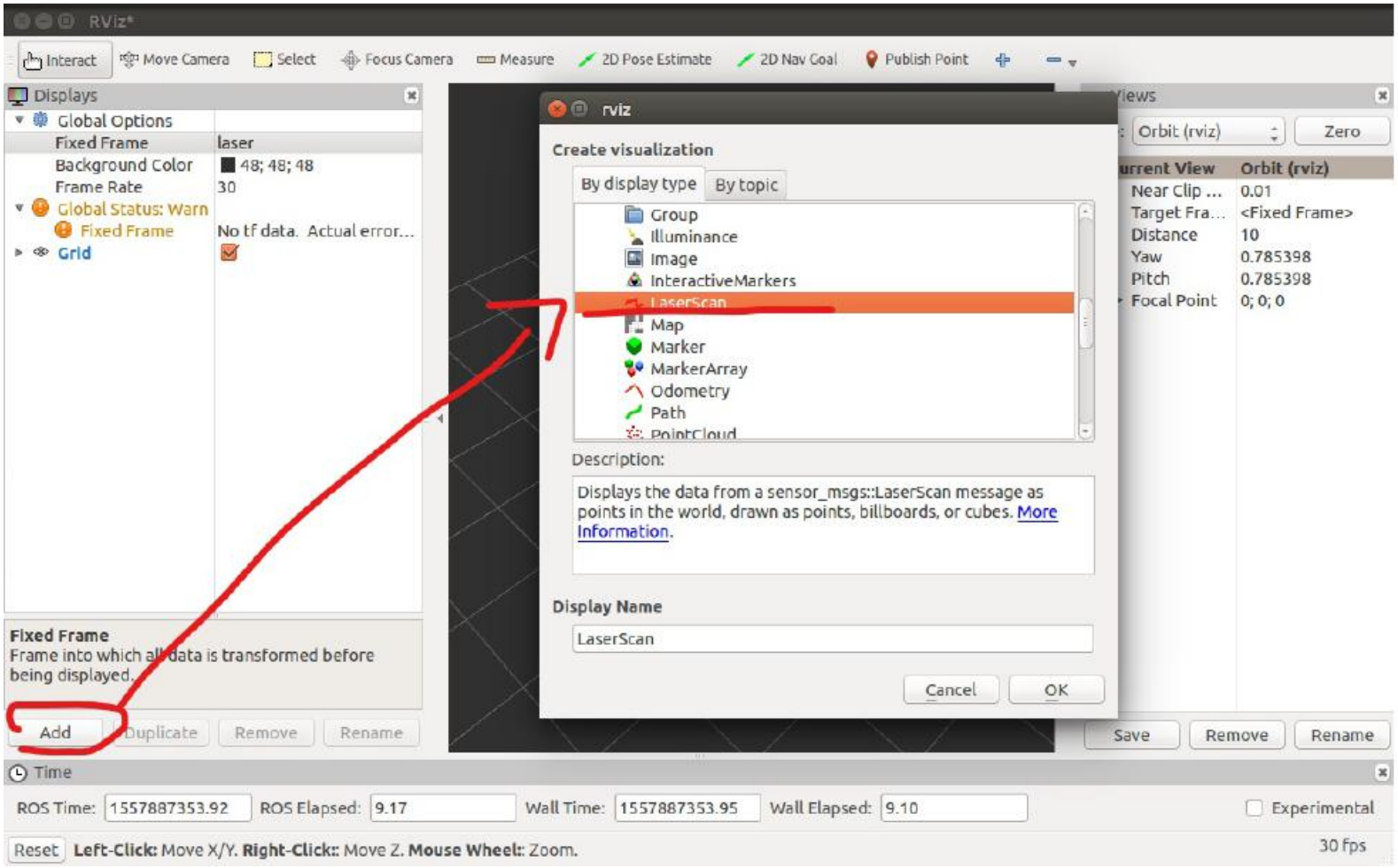
*** catkin_make编译指定的包**

$ catkin_make -DCATKIN_WHITELIST_PACKAGES="package1;package2"

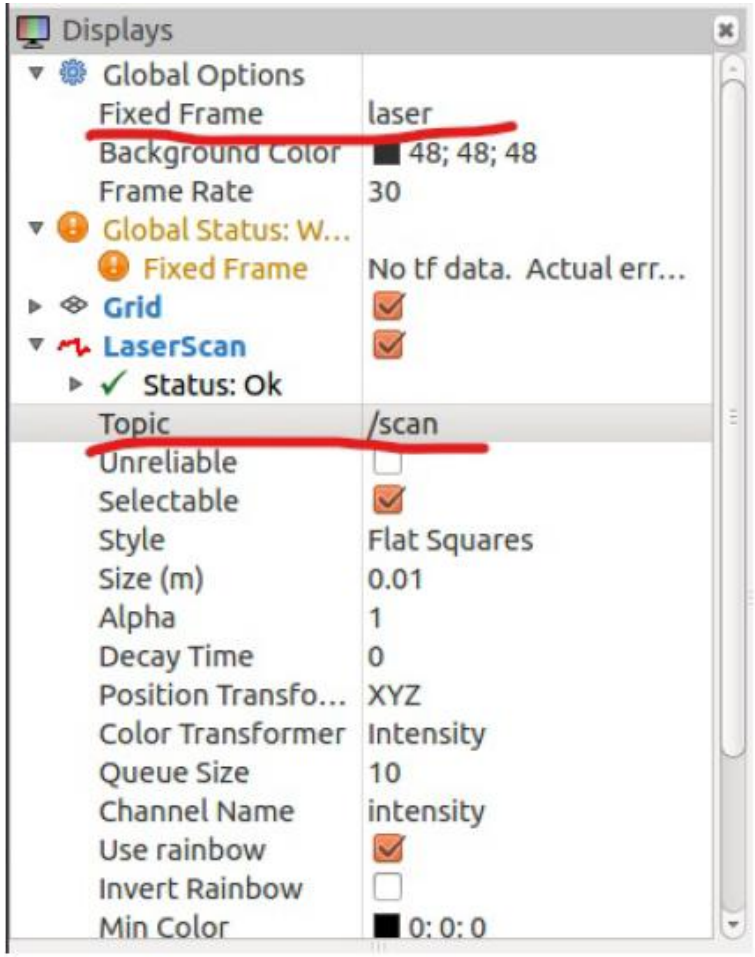
恢复编译所有的包

$ catkin_make -DCATKIN_WHITELIST_PACKAGES=""

6.在rviz中添加激光雷达消息： $rosrun rviz rviz



7.修改主题名：


Displays
Global Options
Fixed Frame
laser
Background Color
48; 48; 48
Frame Rate
30
Global Status: W...
Fixed Frame
No tf data. Actual err...
Grid
LaserScan
Status: Ok
Topic
/scan
Unreliable
Selectable
Style
Flat Squares
Size (m)
0.01
Alpha
1
Decay Time
0
Position Transfo...
XYZ
Color Transformer
Intensity
Queue Size
10
Channel Name
intensity
Use rainbow
Invert Rainbow
Min Color
0; 0; 0

访问和修改激光雷达数据：

```
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <sensor_msgs/LaserScan.h>
#include <stdio.h>
using namespace std;

class Scan2
{
public:
   Scan2() :
      nh_("~")
   {
      scan_pub_ = nh_.advertise<sensor_msgs::LaserScan>("/scan2", 1);
      scan_sub_ = nh_.subscribe<sensor_msgs::LaserScan>("/scan", 1,
&Scan2::ScanCB, this);
   }
```

```
private:
  void ScanCB(const sensor_msgs::LaserScanConstPtr &scan)
  {
    int ranges = scan->ranges.size();
    // 复制LaserScan消息
    sensor_msgs::LaserScan scan2;
    scan2.header.stamp = scan->header.stamp;
    scan2.header.frame_id = scan->header.frame_id;
    scan2.angle_min = scan->angle_min;
    scan2.angle_max = scan->angle_max;
    scan2.angle_increment = scan->angle_increment;
    scan2.time_increment = scan->time_increment;
    scan2.range_min = 0.;
    scan2.range_max = 100.;
    scan2.ranges.resize(ranges);
    // 修改LaserScan扫描点信息
    for (int i=0; i<ranges; i++)
    {
      scan2.ranges[i] = scan->ranges[i]+1;
    }
    // 发布修改后消息
    scan_pub_.publish(scan2);
  }

  ros::NodeHandle nh_;
  ros::Publisher scan_pub_;
  ros::Subscriber scan_sub_;
};
```

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "test_hokuyo_node");
  Scan2 scan2;
  ros::spin();
}
```

1. Main函数初始化一个节点，然后创建一个实例；

代码解析：

```
void ScanCB(const sensor_msgs::LaserScanConstPtr &scan2)
  {
    ……
    sensor_msgs::LaserScan scan;
    scan.header.stamp = scan2->header.stamp;

    ……
    ……
    scan.range_max = 100.;
    scan.ranges.resize(ranges);
    // 修改LaserScan扫描点信息

    for (int i=0; i<ranges; i++)
    {
      scan.ranges[i] = scan2->ranges[i]+1;
    }
    // 发布修改后消息
    scan_pub_.publish(scan);
  }
```

2. 在构造函数中创建两个主题，一个订阅另一个。第二个主题具有激光雷达的原始数据。

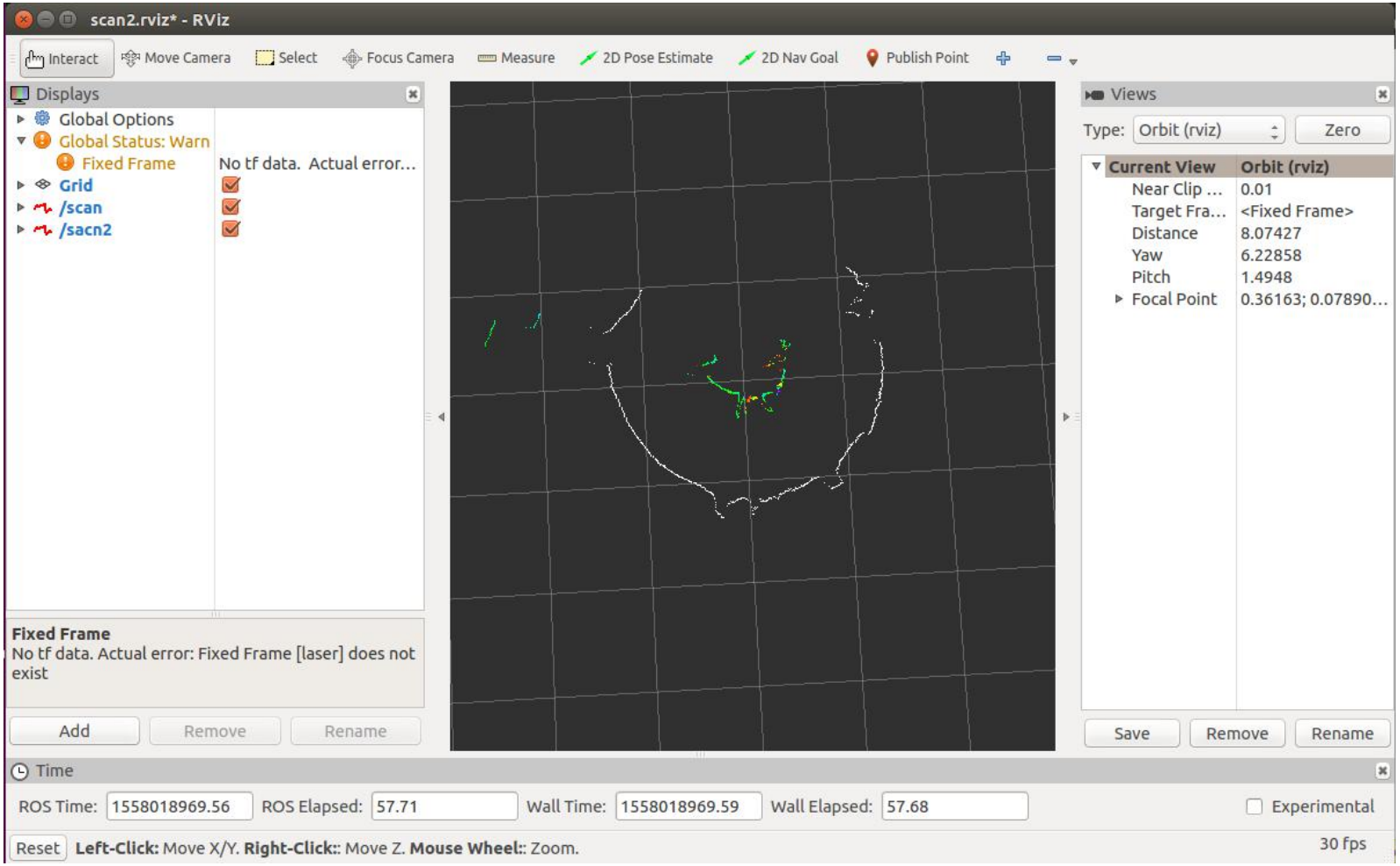
```
class Scan2
{
public:
  Scan2() :
    nh_("~")
  {
    scan_pub_ = nh_.advertise<sensor_msgs::LaserScan>("/scan2", 1);
    scan_sub_ = nh_.subscribe<sensor_msgs::LaserScan>("/scan", 1,
&Scan2::ScanCB, this);
  }
```

3. 对来自雷达主题的数据加1，然后重新发布它。

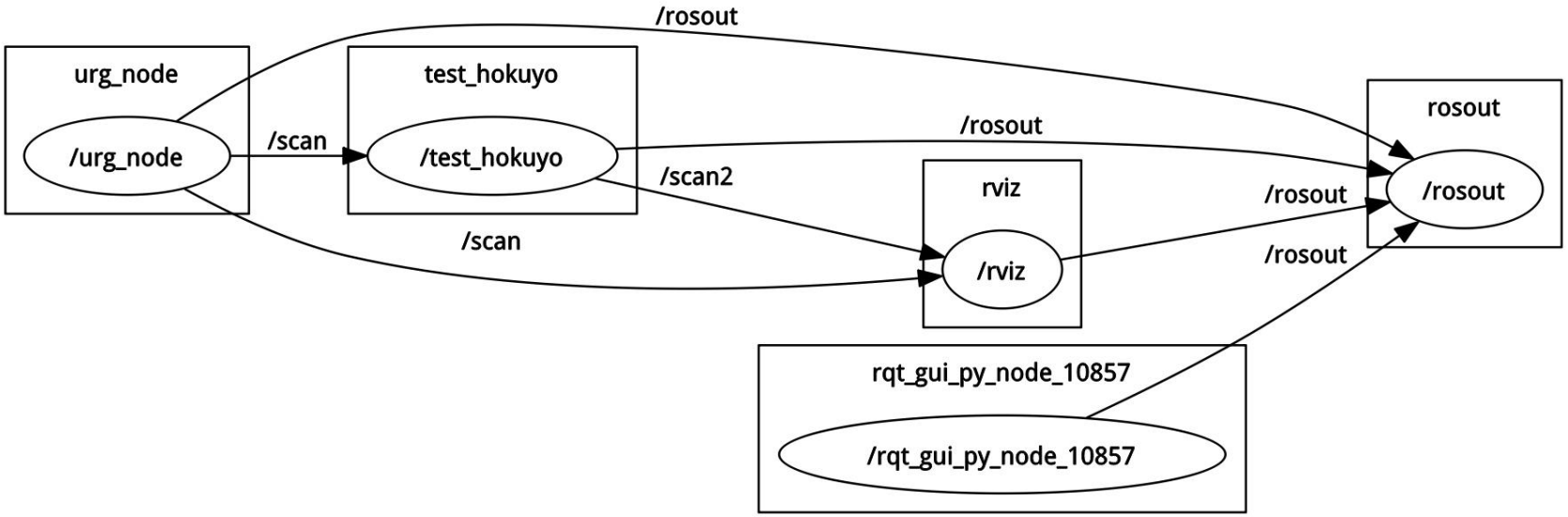
修改launch文件，运行测试程序：

```
<launch>
  <arg name="ip_address" default="192.168.0.10"/>

  <node pkg="urg_node" type="urg_node" name="urg_node"
    output="screen">
    <param name="ip_address" value="$(arg ip_address)"/>
  </node>

  <node pkg="test_hokuyo" type="test_hokuyo_node"
    name="test_hokuyo"
    output="screen"/>

  <node pkg="rviz" type="rviz" name="rviz" output="screen"
    args="-d $(find test_hokuyo)/rviz/scan2.rviz"/>
</launch>
```

运行测试程序：$roslaunch test_hokuyo test_hokuyo.launch



rqt_graph观察： urg_node，test_hokuyo……？



4.2 激光雷达（rplidar)

安装驱动

$ sudo apt-get install ros-kinetic-rplidar-ros

编译（catkin_ws/src目录下，catkin_ws为ROS包）

$ git clone https://github.com/ncnynl/rplidar_ros.git

$ cd ~/catkin_ws

$ catkin_make

更改串口权限

$ ls /dev/ttyUSB*

$ sudo chmod 666 /dev/ttyUSB0

运行

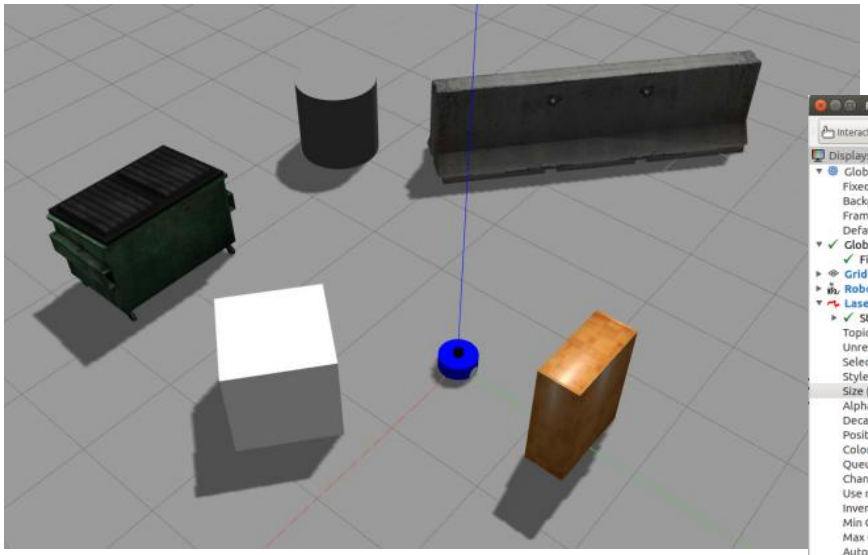
$ roslaunch rplidar_ros view_rplidar.launch

激光雷达 rplidar A1

性能简介：

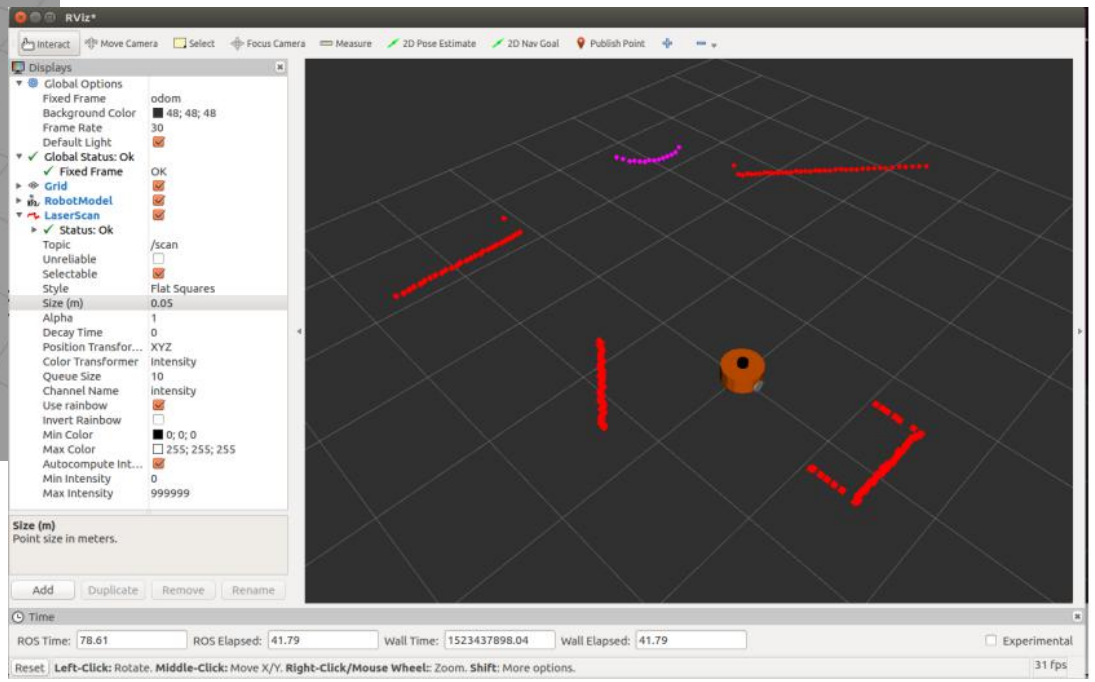
最快频率：10HZ
检测角度：360度
最远距离：6米

激光雷达在gazebo下的仿真演示



Gazebo模型空间

rviz查看



Gazebo：

一款3D动态模拟器,能够在复杂的室内和室外环境中准确有效地模拟机器人群

4.3 RGB-D传感器——Kinect

简介：

- 一种3D体感摄影机
- 即时动态捕捉、影像辨识、麦克风输入、语音辨识、社群互动等功能
- 获取颜色深度图像（包含颜色信息和距离信息），可用于进行机器人三维地图构建

Kinect V1

安装功能包

$ sudo apt-get install ros-kenetic-freenect-camera ros-kenetic-freenect-stack ros-kenetic-freenect-launch

$ rosstack profile && rospack profile

运行Kinect：

$ roslaunch freenect_launch freenect-registered-xyzrgb.launch

提示：如果一切正常，将不会有错误信息。

观察Kinect的Topic：

$rostopic list

```
exbot@studyon-ThinkPad-X250:~$ rostopic list
/camera/debayer/parameter_descriptions
/camera/debayer/parameter_updates
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/image_raw/compressed
/camera/depth/image_raw/compressed/parameter_descriptions
/camera/depth/image_raw/compressed/parameter_updates
/camera/depth/image_raw/compressedDepth
/camera/depth/image_raw/compressedDepth/parameter_descriptions
/camera/depth/image_raw/compressedDepth/parameter_updates
/camera/depth/image_raw/theora
/camera/depth/image_raw/theora/parameter_descriptions
/camera/depth/image_raw/theora/parameter_updates
/camera/depth_registered/camera_info
/camera/depth_registered/hw_registered/image_rect_raw
/camera/depth_registered/hw_registered/image_rect_raw/compressed
/camera/depth_registered/hw_registered/image_rect_raw/compressed/parameter_descr
iptions
/camera/depth_registered/hw_registered/image_rect_raw/compressed/parameter_updat
es
/camera/depth_registered/hw_registered/image_rect_raw/compressedDepth
/camera/depth_registered/hw_registered/image_rect_raw/compressedDepth/parameter_
descriptions
/camera/depth_registered/hw_registered/image_rect_raw/compressedDepth/parameter_
updates
/camera/depth_registered/hw_registered/image_rect_raw/theora
/camera/depth_registered/hw_registered/image_rect_raw/theora/parameter_descripti
ons
/camera/depth_registered/hw_registered/image_rect_raw/theora/parameter_updates
/camera/depth_registered/image_raw
/camera/depth_registered/image_raw/compressed
/camera/depth_registered/image_raw/compressed/parameter_descriptions
/camera/depth_registered/image_raw/compressed/parameter_updates
```

观察Kinect的消息结构：

$rostopic type /camera/depth_registered/points | rosmsg show

$rostopic type /camera/rgb/image_color | rosmsg show

观察Kinect的图像输出：

```
$ rosrun image_view image_view image:=/camera/rgb/image_raw
$ rosrun image_view image_view
image:=/camera/depth_registered/image_raw
```

```
$ rosrun rqt_image_view rqt_image_view
```

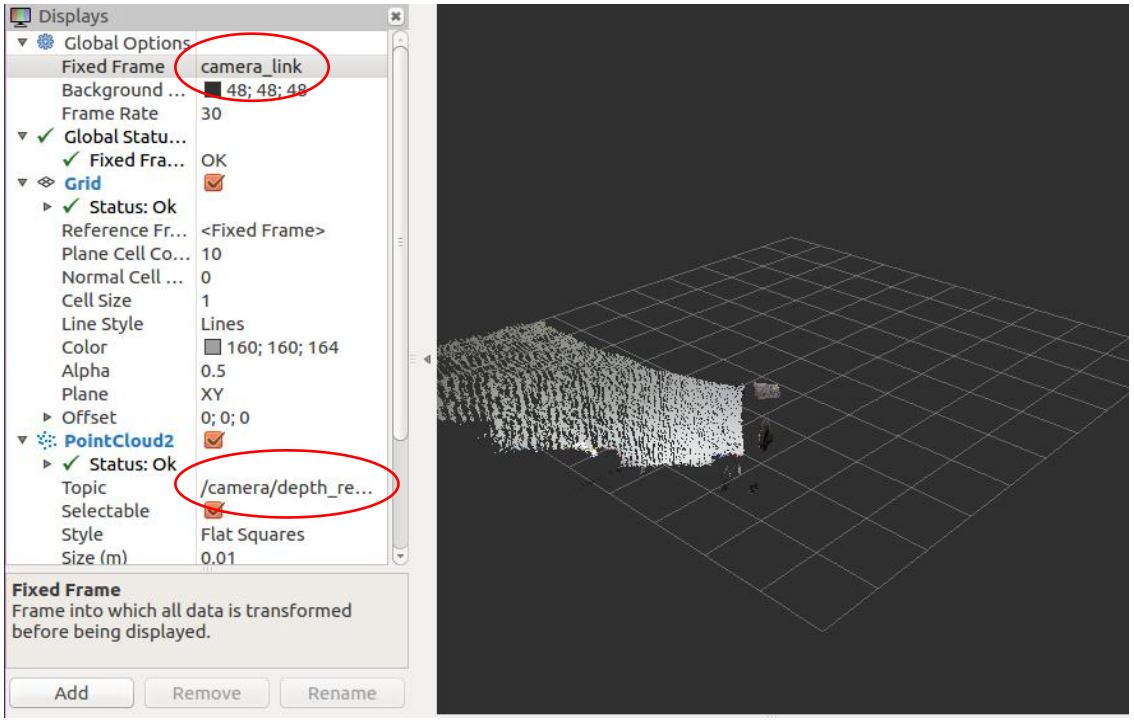
Kinect数据可视化：

$rosrun rviz rviz

操作：Add，按显示类型订阅主题，选择PointCloud2。

提示：a. 必须选择camera/depth_registered/points主题。

b. Fixed Frame选择camera_link



Kinect应用实例：

过滤来自Kinect的点云数量，应用过滤器减小原始数据中点云点的数量，从而减少采样的数据。

```
#include <ros/ros.h>
#include <sensor_msgs/PointCloud2.h>
// PCL specific includes
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/io/pcd_io.h>

ros::Publisher pub;
void cloud_cb (const pcl::PCLPointCloud2ConstPtr& input)
{
    pcl::PCLPointCloud2 cloud_filtered;
    pcl::VoxelGrid<pcl::PCLPointCloud2> sor;
    sor.setInputCloud (input);
    sor.setLeafSize (0.01, 0.01, 0.01);
    sor.filter (cloud_filtered);
    // Publish the dataSize
    ROS_INFO("[number points]before filtered: %d, after filtered: %d", input->data.size(),
cloud_filtered.data.size());
    pub.publish (cloud_filtered);
}
```

所有工作都在cb()函数中完成，当收到消息时会调用该函数。创建了一个VoxelGrid类型的变量sor，在sor.setLeafSize()中改变网格的大小。该数值会改变用于过滤器的网格参数，增大时将获得更低的分辨率和更少的点。

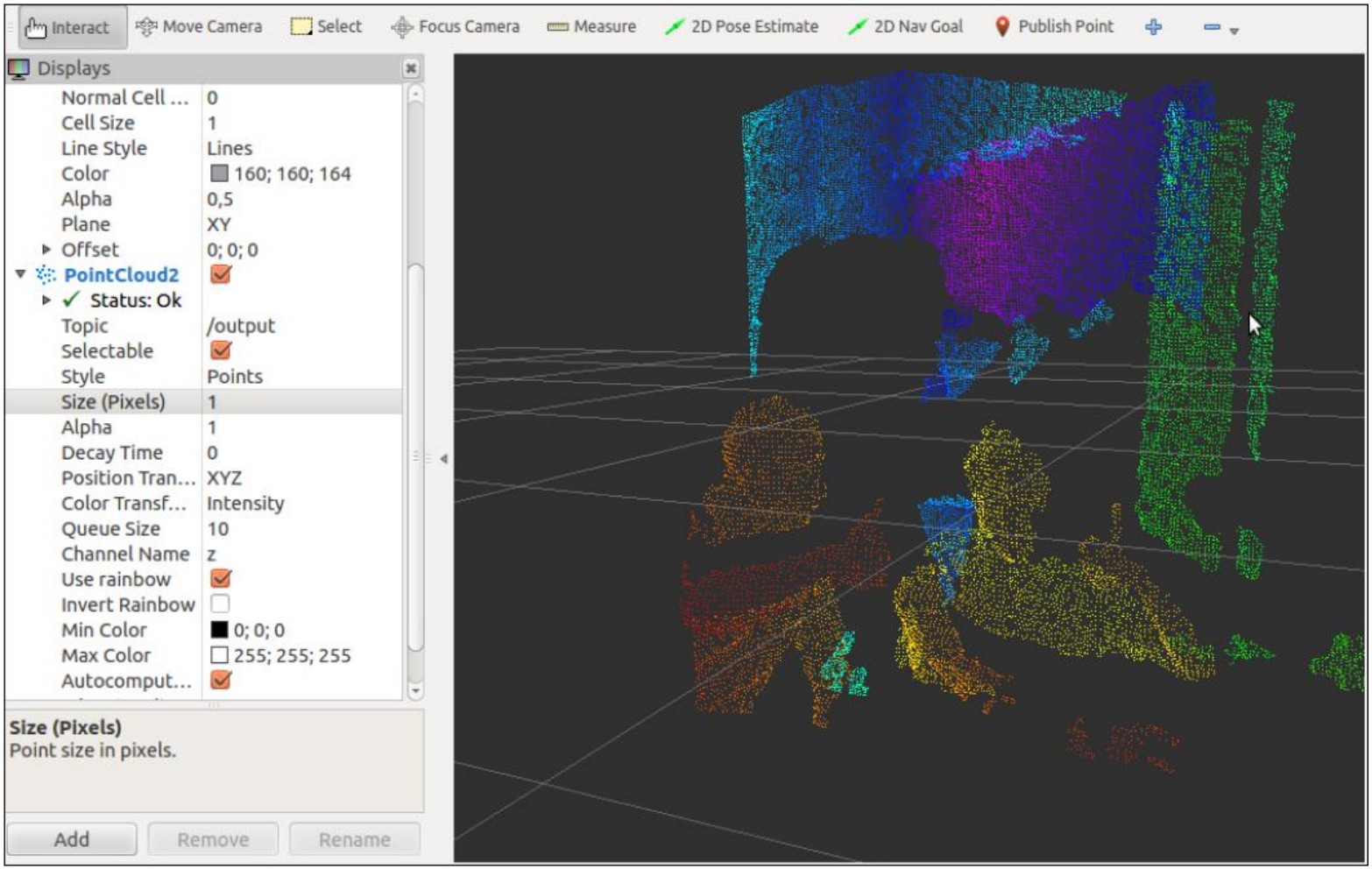
```
void cloud_cb (const pcl::PCLPointCloud2ConstPtr& input)
{
    ……
    pcl::VoxelGrid<pcl::PCLPointCloud2> sor;

    ……
    sor.setLeafSize (0.01, 0.01, 0.01);

    ……

    pub.publish (cloud_filtered);
}
```

```
int main (int argc, char** argv)
{
    // Initialize ROS
    ros::init (argc, argv, "my_pcl_tutorial");
    ros::NodeHandle nh;
    // Create a ROS subscriber for the input point cloud
    ros::Subscriber sub = nh.subscribe ("/camera/depth_registered/points", 1,
    cloud_cb);

    // Create a ROS publisher for the output point cloud
    pub = nh.advertise<sensor_msgs::PointCloud2> ("output", 1);
    // Spin

    ros::spin();
}
```

运行：

$rosrun test_kinect test_kinect_node

点云分辨率发生变化：


Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Normal Cell ... 0
Cell Size 1
Line Style Lines
Color 160; 160; 164
Alpha 0,5
Plane XY
Offset 0; 0; 0
PointCloud2
Status: Ok
Topic /output
Selectable
Style Points
Size (Pixels) 1
Alpha 1
Decay Time 0
Position Tran... XYZ
Color Transf... Intensity
Queue Size 10
Channel Name z
Use rainbow
Invert Rainbow
Min Color 0; 0; 0
Max Color 255; 255; 255
Autocomput...
Size (Pixels)
Point size in pixels.
Add
Remove
Rename

4.4 伺服电动机——Dynamixel

主要参数

AX-12		
重量（g） 55		
减速比 1/254		
输入电压 7V		10V
最大扭矩（kgf·cm） 12		16.5
转速（秒/60°） 0.269		0.196

通讯	半双工异步串口通讯（8bit、1 中断位，无奇偶校验）
最小角度	0.35°
波特率	7343 bps ~ 1M bps
ID 扩展	0 ~ 253
工作电压	7VDC ~ 10VDC（推荐 9.6V）

Dynamixel AX-12

Rostopic list可显示配置好的电动机状态：

/diagnostics

/motor_states/pan_tilt_port

/rosout

/rosout_agg

上述主题不能用于驱动电机，需运行：

$ roslaunch dynamixel_tutorials controller_spawner.launch

创建：

/tilt_controller/command

/tilt_controller/state

可通过指令驱动电机：

$ rostopic pub /tilt_controller/command std_msgs/Float64 -- 0.5

电动机应用实例的源代码

```
#include<ros/ros.h>
#include<std_msgs/Float64.h>
#include<stdio.h>

using namespace std;

class Dynamixel{
  private:
    ros::NodeHandle n;
    ros::Publisher pub_n;
  public:
    Dynamixel();
    int moveMotor(double position);
};

Dynamixel::Dynamixel(){
  pub_n = n.advertise<std_msgs::Float64>("/tilt_controller/command",1);
}
int Dynamixel::moveMotor(double position)
{
  std_msgs::Float64 aux;
  aux.data = position;
  pub_n.publish(aux);
  return 1;
}
```

```
int main(int argc,char** argv)
{
  ros::init(argc, argv, "example4_move_motor");
  Dynamixel motors;

  float counter = -180;
  ros::Rate loop_rate(100);
  while(ros::ok())
  {
    if(counter < 180)
    {
      motors.moveMotor(counter*3.14/180);
      counter++;
    }else
    {
      counter = -180;
    }
    loop_rate.sleep();
  }
}
```

4.5 Arduino平台上使用超声波传感器

什么是Arduino？

一种由灵活、易于开发的软硬件组成的开源电子设计平台，相当于一台没接输入装置与输出装置的电脑主机。

硬件上：

功能引脚引出方便使用，可以接很多外围设备（如GPS、伺服电动机）。

软件上：

专门的程序开发环境Arduino IDE，ROS能够通过rosserial功能包进行使用。

Arduino开发板

安装功能包：

```
$ sudo apt-get install ros-kinetic-rosserial-arduino
$ sudo apt-get install ros-kinetic-rosserial
```

编译（在 catkin_ws/src目录下）

```
$ cd ~/catkin_ws/src/
$ git clone https://github.com/ros-drivers/rosserial.git
$ cd ~/catkin_ws/
$ catkin_make
$ catkin_make_install
$ source install/setup.bash
```

Arduino IDE代码简洁

```
#include <ros.h>  //rosserial库
#include <std_msgs/Int32.h>

XXXXXX     //变量等.....

void setup()
{
  ......
}

void loop()
{
  ......
}
```

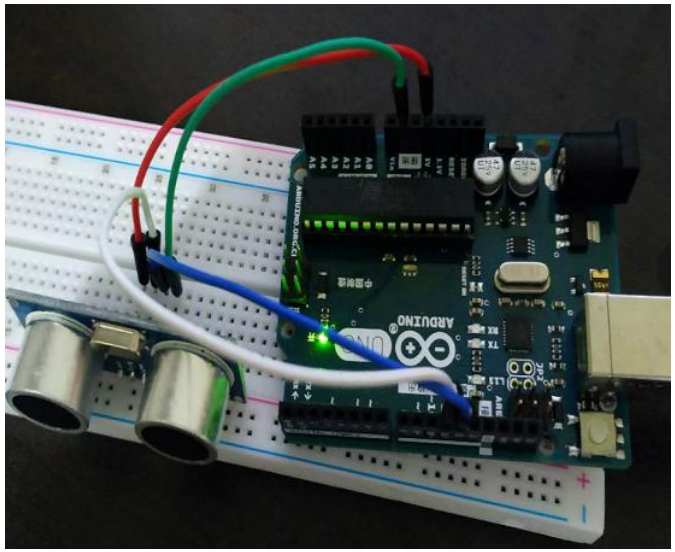
配置函数，
只执行一次

连续执行，内部一般有延时函数

超声波传感器

性能参数

电压:5V
探测距离：2cm-450cm
计算公式：高电平时间*声速
输出频率：40kHZ
精度：小于1cm



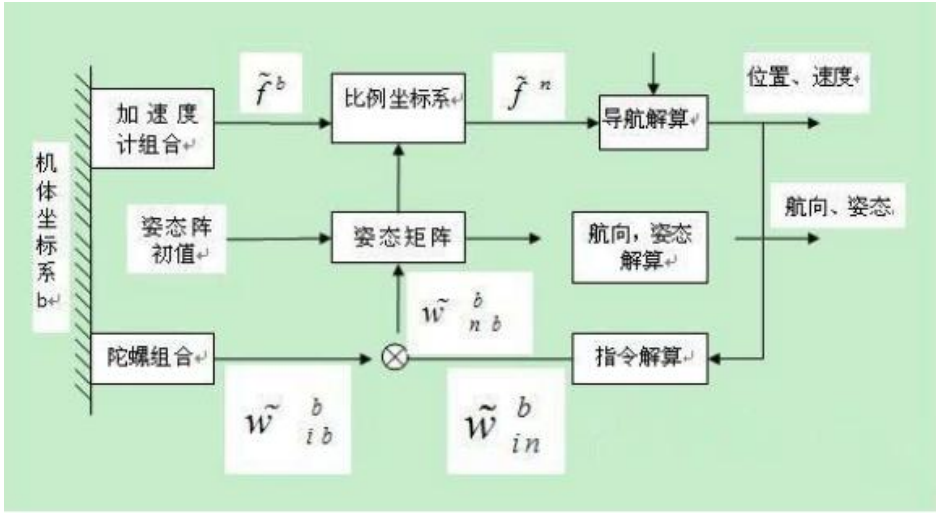
Arduino开发板和超声波进行连接

```
#include <ros.h>
#include <std_msgs/Int32.h>
.....
void setup()        {   .....   }
void loop()
{
 // 产生一个10us的高脉冲去触发TrigPin
 digitalWrite(TrigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(TrigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(TrigPin, LOW);
 // 检测脉冲宽度，并计算出距离
 distance.data = pulseIn(EchoPin, HIGH) / 58.00;
 chatter.publish( &distance);
 nh.spinOnce();
 delay(1000);
}
```

4.6 使用惯性测量模组IMU

惯性测量模组（IMU）：

- 一种测量物体速度、方向、和重力的电子设备
- 组合使用了加速度计和陀螺仪，甚至磁场强度计



IMU实现原理

Xsens MTi

输出信号：

- 3D加速度
- 3D转弯率
- 3D磁场
- 气压

1. 安装功能包和驱动

   $ sudo apt-get install ros-kinetic-xsens-driver

   $ rosstack profile

   $ rospack profile

2. 运行启动

   $ sudo chmod 777 /dev/ttyUSB*

   $ roslaunch xsens_driver xsens_driver.launch

避免出现/dev/ttyUSB permission denied的问题

3. 测试Xsens

   $ rostopic list

   $ rostopic echo /imu/data

```
header:
  seq: 1195
  stamp:
    secs: 1517380201
    nsecs: 206948995
  frame_id: /imu
orientation:
  x: -0.015730990377
  y: 0.00547921542695
  z: -0.464818446609
  w: 0.885249301515
orientation_covariance: [0.017453292519943295, 0.0, 0.0, 0.0, 0.0174532925199432
95, 0.0, 0.0, 0.0, 0.15707963267948966]
angular_velocity:
  x: 0.00389127200469
  y: -0.00115690415259
  z: 0.000154553708853
angular_velocity_covariance: [0.0004363323129985824, 0.0, 0.0, 0.0, 0.0004363323
129985824, 0.0, 0.0, 0.0, 0.0004363323129985824]
linear_acceleration:
  x: 0.0327020809054
  y: -0.321425765753
  z: 9.92385005951
linear_acceleration_covariance: [0.0004, 0.0, 0.0, 0.0, 0.0004, 0.0, 0.0, 0.0, 0
.0004]
---
```

Xsens控制Turtle移动

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/Imu.h>
#include <iostream>
#include <tf/LinearMath/Matrix3x3.h>
#include <tf/LinearMath/Quaternion.h>

using namespace std;

class Imu_Test
{
public:
        Imu_Test();
private:
        void CallBack(const sensor_msgs::Imu::ConstPtr& imu);
        ros::NodeHandle n;
        ros::Publisher pub;
        ros::Subscriber sub;
};

Imu_Test::Imu_Test()
{
        pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1);
        sub = n.subscribe<sensor_msgs::Imu>("imu/data", 10, &Imu_Test::CallBack, this);
}
```

Xsens控制Turtle移动

```
void Imu_Test::CallBack(const sensor_msgs::Imu::ConstPtr& imu)
{
        geometry_msgs::Twist vel;
        tf::Quaternion bq(imu->orientation.x, imu->orientation.y, imu->orientation.z, imu->orientation.w);
        double roll, pitch, yaw;
        tf::Matrix3x3(bq).getRPY(roll, pitch, yaw);
        ROS_INFO("%lf %lf %lf", roll, pitch, yaw);
        vel.angular.z = roll;
        vel.linear.x = pitch;
        pub.publish(vel);
}

int main(int argc, char **argv)
{
        ros::init(argc, argv, "IMU_Turtle");
        Imu_Test imu_test;
        ros::spin();
        return 0;
}
```

4.7 通过Arbotix控制板控制电机

Arbotix是什么？

- 一款控制电机、舵机的硬件控制板
- 提供了相应的ROS功能包
- 提供了一个差速控制器，通过接收速度控制指令，更新机器人的里程计状态

安装编译

$ git clone https://github.com/vanadiumlabs/arbotix_ros.git

$ catkin_make

4.8 GPS的使用

需要安装NMEA GPS 驱动包

在工作路径下

Ubuntu16.04版本

$ git clone https://github.com/ros-drivers/nmea_navsat_driver.git

$ cd libnmea_navsat_driver

$ ckdir build

$ cd build

$ cmake ..

$ make

$ sudo make install

Ubuntu14版本

$ sudo apt-get install ros-hydro-nmea-gps-driver

$ rosstack profile & rospack profile

EM-406a GPS

产品参数：

灵敏度： -159dbm

定位精度：10米/5米

尺寸：30mm*30mm*10.5mm

波特率：4800HZ

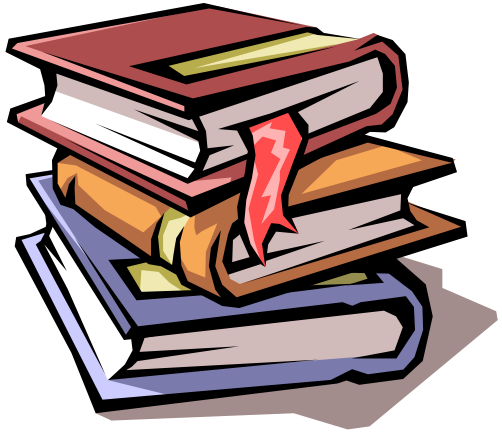
参考资料

ROS外设：

http://wiki.ros.org/Sensors

参考书籍

《ROS机器人开发实践》胡春旭 著



Q & A

Thanks !