TAMPERE UNIVERSITY OF TECHNOLOGY

Introduction to Linux & ROS

Mechatronics and Robot Programming - IHA-4206

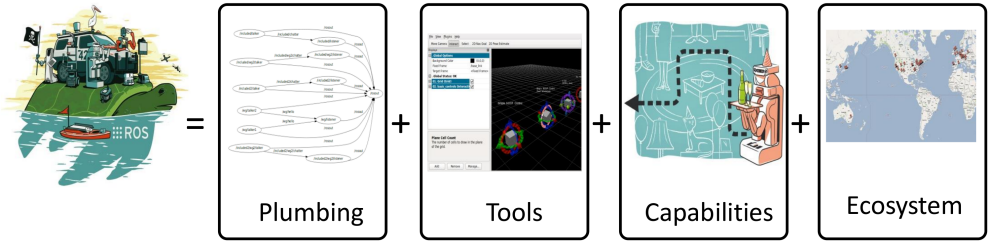
Roel Pieters

AUT

14 September 2018

Introduction

- **Short Linux intro**
- **Longer ROS intro**



Linux

- Open source and free operating system
- From phones to servers
- Many distributions: we will use Ubuntu
- Graphical User Interface
- Command Line Interface: Terminal

Help, howto:

https://help.ubuntu.com/community/

Terminal Commands

Too many to list:

- `ls` - list
- `cd` - change directory
- `cp, mv, rm`
  copy, move, remove
- `help, man` - help, manual
- `ctrl+c` - kill

Other tools:

- Synaptic: Package manager
- System Monitor
- Gedit, Nano
- Qt Creator
- Terminator

If not familiar, do some tutorial!

- http://linuxcommand.org
- https://help.ubuntu.com/community/UsingTheTerminal
- https://ycrc.github.io/PIL/

Terminal Commands

Secure Shell (SSH)

Remote access to a robot/computer

- Start/stop software remotely
- Eliminate latency due to network
- copy files to and from a remote server
- Windows: PuTTy
- Linux:

```
> sudo service ssh status
> sudo apt-get install openssh-server
> ssh username@IP
```

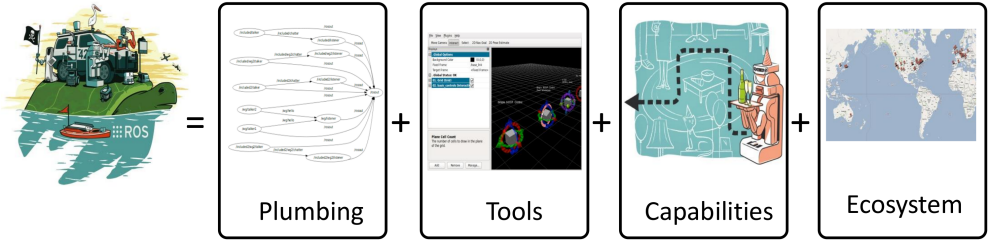
- ROS: distributed computing (one Master, multiple slaves)

https://help.ubuntu.com/community/SSH

Robot Operating System

Philosophy

"ROS is a flexible framework for writing robot software. It's a collection of tools, libraries and conventions aimed to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms."



History: http://www.ros.org/history/

Robot Operating System

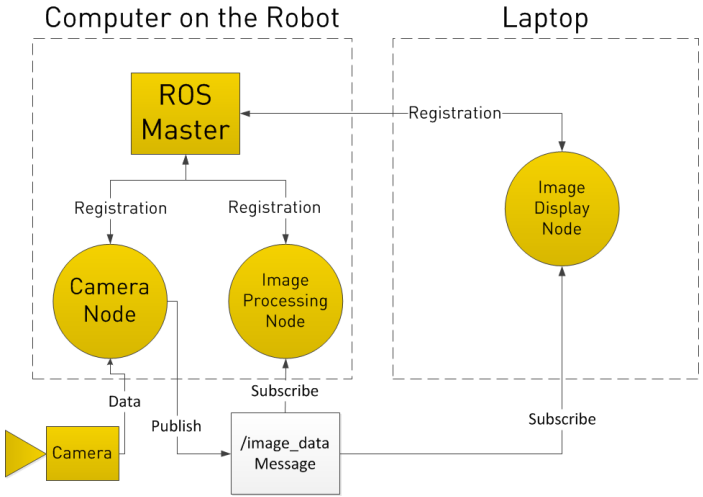
Philosophy

- Peer to peer communication over defined API (messages, service, etc.)
- Distributed nodes on multiple computers over a network
- Support for multiple languages (C++, Python, Java, Matlab, etc.)
- Light-weight wrapping of external libraries
- Open-source and free

Architecture

- Master: handles node communication
  `> roscore`
- Node: executable that uses ROS to communicate with other nodes
  `> rosrun package_name node_name`
- Nodes communicate with each other by publishing/subscribing messages over *topics*
  `> rostopic echo /topic`
- Messages: data structure defining the *type* of a topic
  primitives: int, float, bool, or *sensor_msg*: point, pose, image

Example


Computer on the Robot
Laptop
ROS Master
Registration
Registration
Registration
Image Display Node
Camera Node
Image Processing Node
Data
Publish
Subscribe
Subscribe
Camera
/image_data Message

Communication

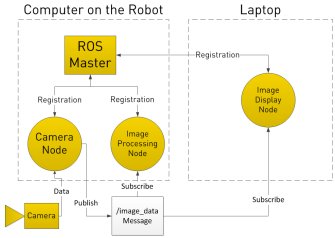
Nodes communicate with each other by publishing/subscribing messages over *topics*

- 1 publisher, n subscribers

```
> rostopic list
> rostopic echo /topic
> rostopic info /topic
```

- Message: data structure defining the *type* of a topic
  primitives: int, float, bool
  sensor_msg: point, pose, image, pointcloud
- Publish a message to a topic

```
> rostopic pub /topic type args
> rostopic pub my_topic std_msgs/String 'hello!'
> rostopic pub -r 10 /cmd_vel geometry_msgs/Twist
  '{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

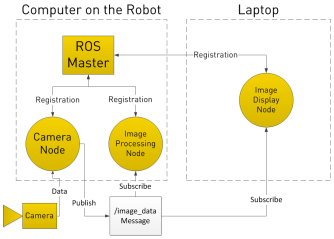


source: Clearpath Robotics Inc.

Execution

catkin is the ROS build system to generate executables, libraries, and interfaces

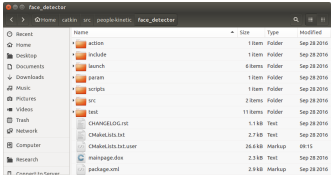
- `> catkin_make`
  `> source devel/setup.bash`
- Rosrun: run individual packages
  `> rosrun package_name node_name`
- Launch: start multiple nodes and set parameters
  `> roslaunch package_name file_name.launch`



source: Clearpath Robotics Inc.

ROS package structure

- src (code here!), dev, build (don't touch!)
  source code, launch files, config files, definitions, documentation, data, etc.
- `package.xml`: defines package properties
- `CMakeLists.txt`: input to CMakebuild system



C++ vs. Python

- Codability
- Readability
- Usability
- Debugging
- Speed
- Library-specific

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

talker.cpp

C++ vs. Python

- Codability
- Readability
- Usability
- Debugging
- Speed
- Library-specific

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

talker.py

vs. Matlab

- Codability
- Readability
- Usability
- Debugging
- Speed
- Library-specific

```
%Setting ROS_MASTER_URI
setenv('ROS_MASTER_URI','http://192.168.56.102:11311')
%Starting ROS MASTER
rosinit

%Creating ROS publisher handle
chatpub = rospublisher('/talker', 'std_msgs/String');
%This is to create the message definition
msg = rosmessage(chatpub);
%Inserting data to message
msg.Data = 'Hello World';
%Sending message to topic
send(chatpub,msg);
%Latching the message on topic
latchpub = rospublisher('/talker', 'IsLatching', true);
```

talker.m

Changing Parameters

ROS Parameter server

- Store and retrieve parameters at runtime
- Parameters defined in launch files or separate *YAML* files
- Terminal commands:

```
> rosparam list
> rosparam get param_name
> rosparam set param_name value
```

```
<launch>
  <node name="name" pkg="package" type="node_type">
    <rosparam command="load"
      file="\$(find package)/config/config.yaml" />
  </node>
</launch>
```

package.launch

```
camera:
    left:
        name: left_camera
        exposure: 1
    right:
        name: right_camera
        exposure: 1.1
```

config.yaml

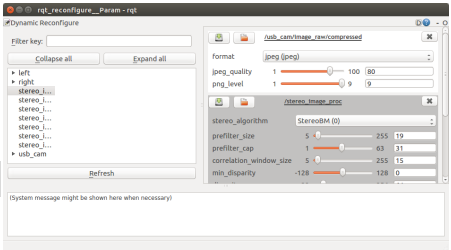
http://wiki.ros.org/parameter_server

Changing Parameters

Dynamic reconfigure

- similar commands for changing parameters
- Terminal commands:

```
> rosrun rqt_reconfigure rqt_reconfigure
```



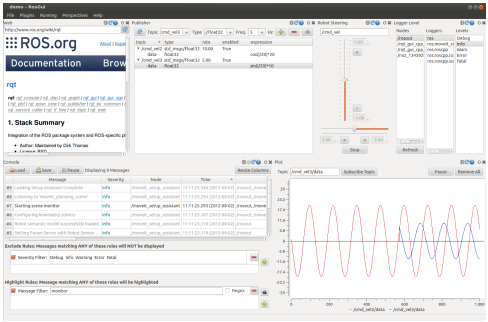
http://wiki.ros.org/rqt_reconfigure

http://wiki.ros.org/dynamic_reconfigure

Visualization

GUI development based on Qt

- Image_view
- Plot
- Graph
- Console
- All (and more) accessible from rqt
- Visualize, launch, log, etc.
- Develop: API in c++, python



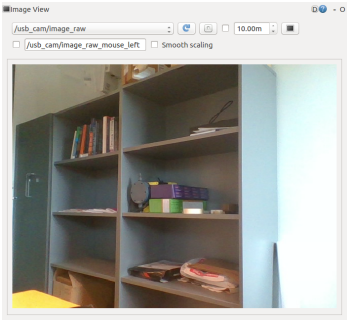

```
> rqt
```

http://wiki.ros.org/rqt

Visualization

GUI development based on Qt

- **Image_view**: display image topic
- Plot
- Graph
- Console



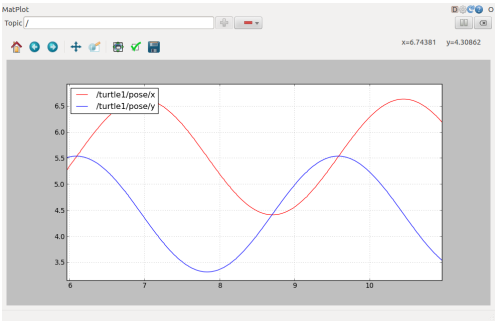

```
> rqt_image_view
```

http://wiki.ros.org/rqt_image_view

Visualization

GUI development based on Qt

- Image_view
- **Plot**: display data from topics
- Graph
- Console



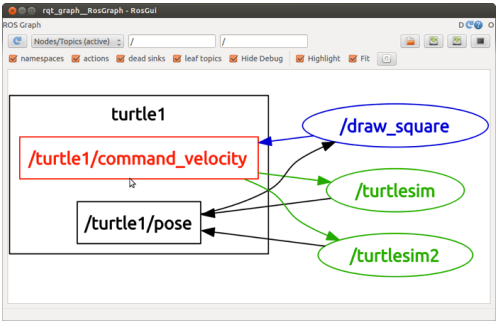

```
> rqt_plot
```

http://wiki.ros.org/rqt_plot

Visualization

GUI development based on Qt

- Image_view
- Plot
- **Graph**: Nodes/Topics
- Console



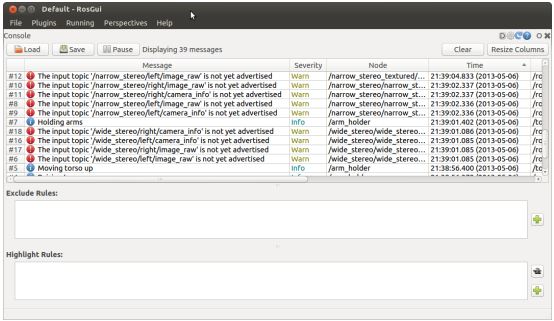

```
> rqt_graph
```

http://wiki.ros.org/rqt_graph

Visualization

GUI development based on Qt

- Image_view
- Plot
- Graph
- **Console**: Messages



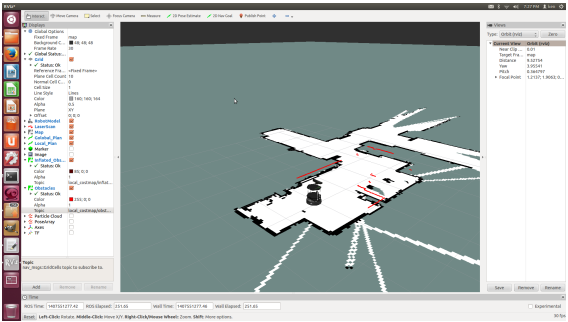

```
> rqt_console
```

http://wiki.ros.org/rqt.console

Visualization

Rviz: 3D visualization tool

- Subscribes to topics
- Visualizes message content
- 3D view
- User control interaction displaying, publishing, etc
- Extensible with plugins
- Save/load configuration



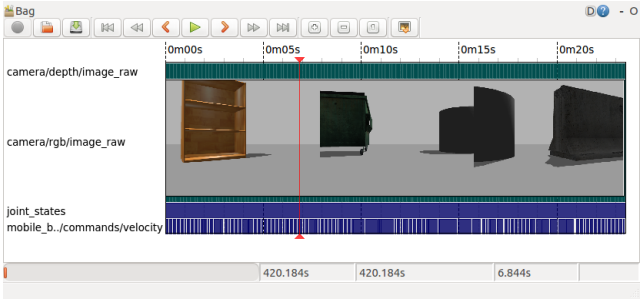
```
> rosrun rviz rviz -d my_config.rviz
```

http://wiki.ros.org/rviz

Logging

A *bag* is a file format in ROS for storing ROS *message* data.

- All messages can be recorded in 1 bagfile
- Tools to store, process, analyze, visualize
- Playback to simulate
- Commandline and GUI
- Import to Matlab



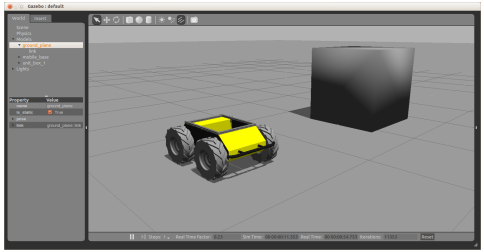

```
> rqt_bag
```

http://wiki.ros.org/bags

Simulation

Gazebo

- 3D rigid body dynamics
- Sensors + noise
- 3D visualize + interaction
- Database of worlds + robots
- SDF: XML description
- ROS interface

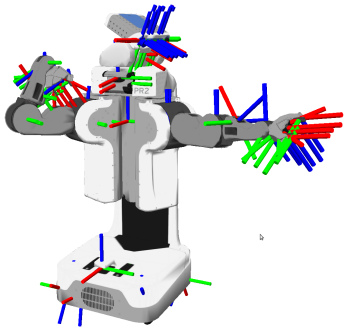


```
> rosrun gazebo_ros gazebo
```

http://wiki.ros.org/gazebo
http://gazebosim.org

Models & environments

- Unified Robot Description Format (URDF)
  XML format for representing a robot model
  Kinematic, dynamic, visual
- Simulation Description Format (SDF)
  XML format
  Environment (gravity), object (dynamic)
  sensor, robot
- TF: coordinate frame transform
  Keeping track of coordinate frames over time



http://wiki.ros.org/urdf
http://sdformat.org/
http://wiki.ros.org/tf2

Manipulator example

- Launch simulated Franka Panda robot
- Visualization

http://frankaemika.github.io/

MoveIt! tutorial

```
> roslaunch panda_moveit_config demo.launch rviz_tutorial:=true
```

Related to course

- Big picture of ROS
- Overload of information; much which is (too) advanced for the course
- Presented Linux, You can/will use Windows and Mac as well
- C++ or Python or Matlab
- Besides ROS: ROS 2.0 and ROS Industrial
- Own laptop/PC? Do tutorials! You don't need hardware.
- Go to SB202 (computer room), Matlab + ROS installed (VMware): see Moodle
- Version control: https://gitlab.tut.fi

Questions?

- Groups!
- Start with exercises!
- Slack: https://tut-robotics.slack.com channel: iha-4206-mepo
- Workstation booking
- Robolab safety session

References

- Further reading + courses:
  - http://www.rsl.ethz.ch/education-students/lectures/ros.html
  - http://www.clearpathrobotics.com/assets/guides/ros/index.html
  - https://github.com/ros-industrial/industrial_training/wiki
- Do tutorials:
  http://wiki.ros.org/ROS/Tutorials
- ROS cheat sheet:
  https://github.com/ros/cheatsheet/releases/download/0.0.1/ROScheatsheet_catkin.pdf