Imperial College London

PERSONAL ROBOTICS LABORATORY

imperial.ac.uk/PersonalRobotics

Intro to ROS

EE4-60 Human-Centred Robotics

ROS

Your Wonderful GTAs

Mark Zolotas

Rodrigo Chacon Quesada

Fan Zhang

Urbano Miguel Nunes

Vinicius Schettino

Outline

* What is ROS and why use it?
* Conceptual overview
* How do I use it? Sample code…
* Live demo + exercises
* Further resources

Taken from: http://wiki.ros.org/navigation

What is ROS?

* ROS is an open-source **Robot Operating System**
* **NOT** an actual operating system but instead a **software framework** that offers similar services:
  * Inter-process communication
  * Hardware abstraction
  * Low-level device control
* Collection of **libraries** & **tools** for common robot functionality

Why ROS?

* Widely used in the robotics community
* Works with C++, Python, Matlab etc.
* Sensors (http://wiki.ros.org/Sensors) and robots!

Taken from: https://robots.ros.org/

Conceptual Overview of ROS

ROS Master and Nodes

* **Nodes** are single-purpose, executable programs
* ROS **Master** manages the communication between nodes
* Every node registers a startup with the master

* Start a master with
  `roscore`
* Run a node with
  `rosrun package_name node_name`
* See active nodes with
  `rosnode list`
* Retrieve information about a node with
  `rosnode info node_name`

Topics

* Named bus for **one-to-many** communication
* Nodes exchange messages via topics
* Use **publish/subscribe** semantics for message transfer

* List active topics with
  `rostopic list`
* Subscribe and print topic contents with
  `rostopic echo /topic`
* Show information about topic with
  `rostopic info /topic`

Messages

* Data structures defining the **type** of a topic
* Can be comprised of a nested structure of types e.g. integers, floats, arrays of objects etc.
* Defined in ***.msg** files

* Display the type of a topic with
  `rostopic type /topic`
* Publish a message to a topic with
  `rostopic pub /topic type args`

More info
http://wiki.ros.org/Messages

Services

* **Request/response** communication paradigm
* **One-to-one:** client sends a request and server responds
* Defined in ***.srv** files

Use instead of topics for specific actions e.g. to rapidly query the state of the robot **once**

Packages

* Organise ROS-related resources into **packages**
* Contain nodes, libraries & associated files
* Every node *must* be inside a package
* Packages *must* be inside a **workspace**

"A ROS package is simply a directory which contains an XML file describing the package and stating any dependencies" — http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

Remember how to execute nodes with

```
rosrun package_name node_name
```

Launch Files

* Use a tool `roslaunch` to recall loads of individual processing units i.e. nodes

* **Single command** to launch multiple processes at once

* Written as XML files describing the computation graph

So… How do I use it?

* Learn how to **install** ROS and setup a **workspace**
* Walkthrough of a complete C++ tutorial on how to create/write a **…**
  * **… workspace**
  * **… package**
  * **… node**
  * **… publisher**
  * **… subscriber**
  * **… roslaunch file**
* **GitHub URL:** https://github.com/mazrk7/tutorial_pub_sub.git
* Full instructions for installation found at: http://wiki.ros.org/kinetic/Installation/Ubuntu

Installation

* **Configure** your Ubuntu repositories
  * `$ sudo sh -c 'echo "deb https://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`
  * `$ sudo apt-key adv —keyserver hip://ha.pool.sks-keyservers.net:80 —recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116`
* **Update** package manager and run a **full install**
  * `$ sudo apt-get update`
  * `$ sudo apt-get install ros-kinetic-desktop-full`
* Initialise `rosdep` and **setup** environment
  * `$ sudo rosdep init`
  * `$ rosdep update`
  * `$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc`
  * `$ source ~/.bashrc`
  * `$ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential`

Creating a Workspace

* Create the **workspace**
  * `$ mkdir -p ~/catkin_ws/src`
  * `$ cd ~/catkin_ws`
  * `$ catkin_make`
  * `$ echo “source ~/catkin_ws/devel/setup.bash” >> ~/.bashrc`
  * `$ source ~/.bashrc`
* When executed for the first time, `catkin_make` will create two other folders: `build` and `devel`
* If you execute `ls` command, you should have the following folders
  * `build` - compilation files
  * `devel` - executables, libraries, messages
  * `src` - packages, source files (.cpp, .h, .xml), launch files (.launch)

Running ROS

* Download the ROS tutorial software
  * `$ sudo apt-get install ros-kinetic-ros-tutorials`
* Startup our **ROS core**
  * `$ roscore`
* `roscore` is the node responsible for operating the basic processes required for any ROS system
* In another terminal, run the turtle demo
  * `$ rosrun turtlesim turtlesim_node`

Creating a Package

* Create a **package** to control the turtle position
  * `$ cd ~/catkin_ws/src`
  * `$ catkin_create_pkg tutorial_pub_sub std_msgs roscpp`
  * `$ source ~/.bashrc`
* Several files/folders will be created
  * `include/tutorial_pub_sub` — folder containing header files .h or .hpp
  * `src` — folder containing source files .cpp (can do the same for .py scripts instead)
  * `package.xml` — package description and dependencies
  * `CMakeLists.txt` — build instructions
* You have been provided with the `tutorial_pub_sub` package

package.xml

* Contains package **meta-data** (description, dependencies, …)
* Important fields
  * <name> package name </name>
  * <description> brief description </description>
  * <buildtool_depend> build tools packages </buildtool_depend>
  * <build_depend> packages needed at compile time </build_depend>
  * <exec_depend> packages needed at runtime </exec_depend>

CMakeLists.txt

* Contains the package **build instructions**
* Important fields
  * `cmake_minimum_required` — cmake version required
  * `project` — package name
  * `find_packages` — find necessary packages
  * `catkin_package` — package build info export
  * `include_directories` — included files `.h`
  * `add_executables` — node executable
  * `add_dependencies` — add necessary dependencies to library/executable
  * `target_link_libraries` — link libraries to other libraries/executable

Creating a Node

* **`hello_node.cpp`** — a “Hello ROS” node
* Important lines
  * `#include <ros/ros.h>` — includes main ROS headers
  * `ros::init(argc, argv, pkgName);` — initialise ROS node
  * `ros::NodeHandle nh;` — node handle to create ROS objects
  * `ROS_INFO(info);` — printing function (`ROS_DEBUG`, `ROS_ERROR`, …)
  * `ros::spin();` — awaits for stopping signal
* Build and run the package
  * `$ cd ~/catkin_ws`
  * `$ catkin_make --pkg tutorial_pub_sub`
  * `$ rosrun tutorial_pub_sub hello`

ROS Publishers

* **`tutorial_pub_node.cpp`** — a simple publisher
* Important lines
  * `#include <geometry_msgs/Twist.h>` — velocity cmds messages
  * `ros::Publisher pub=nh.advertise<msgType>(topic, bufferSize);`
  * `ros::Rate loop_rate(frequency);` — handle for keeping periodicity
  * `ros::ok()` — check if node is running fine
  * `pub.publish(msg);` — publish a message on topic advertised
  * `ros::spinOnce();` — check once for stopping signal
* Run the node
  * `$ rosrun tutorial_pub_sub tutorial_pub`

ROS Subscribers

* `tutorial_pub_sub_node.cpp` — a simple publisher/subscriber
* Important lines
  * #include <turtlesim/Pose.h> — pose messages
  * ros::Subscriber sub=nh.subscribe(topic, bufferSize, &callbackFunc);
  * void callbackFunc(msgType msg); — subscriber callback function
* Run the node with a remapping to subscribe to `/turtle1/pose`
  * `$ rosrun tutorial_pub_sub tutorial_pub_sub pose:=/turtle1/pose`

ROS Launch

* **`tutorial_pub_sub.launch`** — run everything with just one launch file
* Important lines
  * <node pkg="pkgName" name="uniqueROSNodeName" type="execName">
  * <remap from="topicFrom" to="topicTo"> — remaps topic's flow
  * <param name="paramName" value="paramValue">— node initial parameters
* Launch!
  * `$ roslaunch tutorial_pub_sub tutorial_pub_sub.launch`

Live Demo!

GitHub URL: https://github.com/mazrk7/p3at_teleop

joy

our node

p2os_driver

rviz + rqt_graph

rviz: visualise many robotic-related datatypes

rqt_graph: view nodes & topics

Logging Data

* **Bags** used to store ROS message data
* Record **all** data (`-a`) passing through topics
  * `$ rosbag record -a`
* **Playback** the recorded data with
  * `$ rosbag play bag_file`
* Services are not logged!
* Be weary of large files and expensive I/O

Additional Resources

- **ROS tutorials**
  - http://wiki.ros.org/ROS/Tutorials
- **ROS answers**
  - https://answers.ros.org/questions/
- **ROS paper**
  - http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf
- **Navigation stack**
  - http://wiki.ros.org/navigation
  - http://kaiyuzheng.me/documents/navguide.pdf
- **TF package**
  - http://wiki.ros.org/tf/Tutorials

Thanks for listening!