• ROS

• Design goals

• Installation

• Core components
  – Packages, Meta Packages, Node, ROS Master, Parameter Server, Messages, Topics, Services, ActionClients
  – Client libraries: roscpp, rospy, roslisp, rosjava*, roslua*, rosc#*, …

• Tools
  – catkin_make (rosmake), catkin_create_pkg, rospack, roscd
  – Rosrun/launch, rostopic/node/param/srv/msg, rviz, rxbag, rxplot

• Example
ROS design goals

• ROS is thin – won't wrap main()
  – Suitable for software frameworks and libraries
• ROS-agnostic libraries with clear interfaces
• Language independence – c++, python, lisp …
  – Through client libraries: roscpp, rospy, roslisp, rosjav*, roslua*, rosc#*
• Easy testing – unit testing using rostest (gtest)
• Scaling – suitable for large runtime systems
ROS Installation

- Follow instructions at: http://wiki.ros.org/ROS/Installation

- On ubuntu
  - Add package repositories
  - Install debian packages

- Other systems are supported (experimental):
  - Windows, Ubuntu ARM, OS X, ...
ROS core components
ROS core components

- Catkin workspaces
- Meta Packages
  - Group together multiple packages
- Packages
  - Contain code for nodes, configuration files, build scripts, launch files, ...
- Node
  - Executable or library
ROS core components

- **Messages**
  - Data structures that can be sent to other nodes

- **Topics**
  - Publish/subscribe to a topic to send/receive
  - Non blocking, regularly updated

- **Services**
  - Request-response architecture
  - Blocking, on demand data

- **ActionClients**
  - Request-monitor-response architecture
  - Non blocking, on demand data
ROS core components

- ROS Master
  - Registers nodes, topics, services, ...
- Parameter Server
  - Stores master-wide parameters (e.g. robot model)
ROS core components
ROS Tools
ROS tools – creating and compiling packages

- **catkin_init_workspace**
  - Initialize a catking workspace
    http://wiki.ros.org/catkin/Tutorials/create_a_workspace

- **catkin_create_pkg**
  - Creates new ROS package

- **catkin_make**
  - Compile catking workspace or package
    http://wiki.ros.org/catkin/commands/catkin_make

- **rospack**
  - Search for packages, display package info
    http://wiki.ros.org/rospack

- **rosccd**
  - Change directory into a ROS package
ROS tools – running code

- rosrunc
  - Runs a ROS node

- roslaunch
  - Runs a launch script
    - May run several nodes
    - Set parameters
    - Remap topics
    - ...

ROS tools – debug

- rostopic
  - Print topic info: value, update rate, type, publisher, subscribers, list topics
- rosnode
  - Print node info and list nodes, kill, ping, ...
- rosparam
  - Get/set value, list parameters, dump to file
- rossrv
  - List services, print service info
- rosmsg
  - Print message info
ROS tools - rqt_bag
ROS tools - rviz
Example
c++ guidelines
C++

- Directory structure
- CMAKE or catkin build systems
- Google c++ style guide
  [http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml](http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml)
- Write code for standard, high-performance c++
  + on modern processors
  (avoid embedded c hacks)
c++ style

• Don't use tabs
  – Indent with 2 spaces
  – Configure your editor (Eclipse, NetBeans, ...)

c++ keywords used incorrectly

• unsigned
  - Is appropriate for code that performs bitwise operations, that needs unsigned overflow semantics, and potentially for storage types.
  - It is not suitable for asserting that a variable is never negative, loop indexes, etc.

• volatile
  - Never use this in c++
c++ namespaces

Don't import an entire namespace. You can import individual members of the namespace if you need to.

- using namespace std; // BAD
- using std::vector; // Good
c++ unused code

Don't comment out code to disable it!
• If you think you will need to go back to refer to the old code in the future, Git will have a copy
• If you have good reason to need to disable part of the code on occasion, use the preprocessor:
  • #if 0
  • do_thинг();
  • #endif
• You can still have code in comments to document intended usage.
  You can comment out unused function arguments as documentation.
c++ reusing code

Don't copy-paste code around!

• If you're copy-pasting because you're making a new version of something:

• To replace an old one (mynode.cpp, mynode_v2.cpp)? The old version will still be around in Git.

• To test side-by-side? Use compiler flags, parameters, or config files.
Git repositories

Remote server (GitHub, Bitbucket, ...)

Original repository (read only)
Git repositories

Remote server (GitHub, Bitbucket, ...)

Original repository (read only)

Fetch

Push

Local clone of original repo. (read write)

Local machine
Local machine

Remote server (GitHub, Bitbucket, ...)

Original repository (read only)

Pull request

Fork

Private fork (read write)

Fetch

Push

Local clone of original repo. (read write)
Git repositories

Remote server (GitHub, Bitbucket, ...)

Original repository (read only)

Pull request

Fork

Private fork (read write)

Fetch

Push

Local machine

Local clone of original repo. (read write)

Fork

Push

Fetch

Local clone of private fork (read write)
Git repositories

Remote server (GitHub, Bitbucket, ...)

Original repository (read only)

Pull request

Fork

Private fork (read write)

Local machine

Local clone of original repo. (read write)

Push

Stage & commit

New code

Local clone of private fork (read write)

Fetch

Push

Fetch
Links

- ROS:
  - http://www.ros.org
  - http://wiki.ros.org/ROS/Installation
  - http://wiki.ros.org/ROS/Tutorials

- GIT:
  - https://www.atlassian.com/git/tutorials/

- Coding:
  - http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml