Overview

- Which ROS version should I use and why?
  - Windows?
- Write as little code as possible!
  - stealing re-use & configuration, not creation
- If you really have to: how to write nodes and build them dependencies?
- Application design
  - mapping requirements to packages
- Where to find information and get help

Maybe: demo UR5
Which ROS version and why?
REP-003

- Newest: Kinetic Kame (LTS, 2016-2021)
  - Binaries: Ubuntu Xenial (x86, x64, ARM), Debian Jessie (x64, ARM64)
  - Source: OSX, Yocto, etc
  - C++11

- Most used: Indigo Igloo (LTS, 2014-2019)
  - Binaries: Ubuntu Saucy, Trusty (x86, x64, ARM)
  - Source: Debian, OSX, Yocto, Arch, Slackware, Android, etc

- Older: Hydro, Groovy, Fuerte, Electric, Diamondback, C, Box
Demo

Installation & setup
Write as little code as possible!

- Component based software system
- High degree of modularity:
  - Separation of concerns
  - Locality of change
  - Location independence
  - Decoupling: time, space & synchronisation

→ result: compose applications, don’t write them
Demo

Zero-code “applications”
Demo – zero-code “applications” (1)
Demo – zero-code “applications” (1)
Demonstration – zero-code "applications" (2)
Demo – zero-code “applications” (2)
How to write nodes and build them
If you really have to

- Writing nodes (programs)
  - creating a package
  - hello ROS: talker, listener
  - service client, service server
  - dependencies: CMakeLists & package manifests

- Building packages
  - catkin, catkin tools (really: CMake)
  - dependencies: rosdep
How to write nodes and build them
Creating a workspace

```
user@machine:~$ source /opt/ros/indigo/setup.bash
user@machine:~$ mkdir -p catkin_ws/src
user@machine:~$ cd catkin_ws
user@machine:~$ catkin_make   ('catkin build' if you have catkin_tools)
```
How to write nodes and build them
Creating a package

```
user@machine:~/catkin_ws$ cd src/
user@machine:~/catkin_ws/src$ catkin_create_pkg my_pkg roscpp std_msgs
Created file my_pkg/CMakeLists.txt
Created file my_pkg/package.xml
Created folder my_pkg/include/my_pkg
Created folder my_pkg/src
Successfully created files in /home/user/catkin_ws/src/my_pkg. Please
    adjust the values in package.xml.
user@machine:~/catkin_ws/src$
```
How to write nodes and build them

Building the workspace (and the pkg)

```
user@machine:~/catkin_ws/src$ cd..
user@machine:~/catkin_ws$ catkin_make
Base path: /home/user/catkin_ws
Source space: /home/user/catkin_ws/src
...
...
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/catkin_ws/build
```
How to write nodes and build them
CMakeLists.txt

```bash
# CMakeLists.txt

cmake_minimum_required(VERSION 2.8.3)
project(my_pkg)
find_package(catkin REQUIRED roscpp std_msgs)
include_directories(${catkin_INCLUDE_DIRS})
catkin_package(CATKIN_DEPENDS roscpp std_msgs)
add_executable(my_node node.cpp ..)
target_link_libraries(my_node ${catkin_LIBRARIES})
```
How to write nodes and build them
package manifest

<package format="2">
  <name>fanuc_resources</name>
  <version>0.4.1</version>
  <description>This is my pkg</description>
  <author>Me</author>
  <maintainer email="me@me.me">Me</maintainer>
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>std_msgs</exec_depend>
</package>
How to write nodes and build them
Checking dependencies

```
user@machine:~/catkin_ws/src$ cd ..
user@machine:~/catkin_ws$ rosdep check --from-paths src/ --ignore-src
All system dependencies have been satisfied
user@machine:~/catkin_ws$
```
Application design
I did the tutorials, now what?

- Top-down approach
- Basic idea:
  1. Gather requirements: functional and non-functional
  2. Group functional requirements into cohesive clumps (*name* them)
  3. Determine what kind of data *would be* needed by those clumps, and what *should be* produced
  4. Determine how input is *transformed* into output (*transfer functions*)
  5. Find nodes (packages) that can perform those transformations *and* that fulfil your non-functional requirements
Example: PR2 “fetch me a sandwich”

Requirements, grouping & naming

- Robot needs to get from where it is to the sandwich shop
- Needs to plan a route
- Needs to know where elevators are (or: how floors are connected)
- Needs to know where it started
- Needs to know where sandwich shop is
- More importantly: it needs to know where *it* is

→ the robot needs (self)-localisation
Example: PR2 “fetch me a sandwich”
Data: consumed, produced

Consumed:
- A map
- Measurements of the ‘world’:
  - Laser scans
  - Radar
  - (3D) Camera
  - GPS
  - Bar codes / markers / rfid / artags / qr codes
  - Etc

Produced:
- Position of robot in map
Example: PR2 “fetch me a sandwich”
Transformation(s)

Data IN:
- Map (image) + measurements (msgs / image frames / GPS coordinate)

Data OUT:
- Position in map (WGS84 referenced fi)
Example: PR2 “fetch me a sandwich”
Map needed transforms to nodes/pkgs

- AMCL: Adaptive Monte Carlo Localization
  wiki.ros.org/amcl
- MRPT: Mobile Robot Programming Toolkit
  wiki.ros.org/mrpt_localization
- RTAB-Map: RGB-D SLAM & visual odometry
  wiki.ros.org/rtabmap_ros
- robot_localization: no, does not take a map
- hector_localization: no, idem
Where to find info and get help

- **Main entry point**: ros.org → nice to look at, once
- **Main wiki**: wiki.ros.org → packages, tutorials
- **Main ‘forum’**: answers.ros.org → StackOverflow for ROS
- **Discussion forum**: discourse.ros.org → for ‘complex’ questions
- **Main mailing list**: lists.ros.org → ros-users@
- **Other**: IRC, Slack
- **Books**: - A Gentle Introduction to ROS (free)
  - ROS Robotics By Example
  - Mastering ROS for Robotics Programming
  - Programming Robots with ROS

wiki.ros.org/Books
Dos, Don’ts & Info

- Prefer pre-built binary pkgs: look for
  `sudo apt-get install ros-$distro-abc`
- There’s more than C++: Python, Java, Lua, Go, LISP, C#, Haskell, Smalltalk, R, Ruby, Matlab, LABView, etc
- Use TF (sensor data)
- Use standard messages as much as possible (RViz, re-use)
- Catkin == CMake (+ some ROS macros)
- Embedded: setup cross-compilation (avoid on-target)
- Use RViz as a debugging tool
- Add lots of tests, it helps
- source /path/to/catkin_ws/devel/setup.bash
- rospack profile
Catkin: additional material

- Catkin: http://jbohren.com/articles/gentle-catkin-intro/
- Catkin documentation: http://docs.ros.org/jade/api/catkin/html/index.html
- Packages, libraries, catkin: http://jbohren.com/articles/modular-ros-packages/