ROS

Robot Operating System
Spring 2014

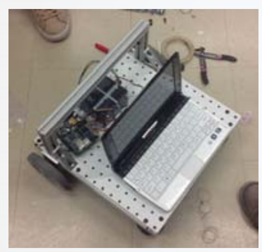
Alec Poitzsch
MIT EECS MEng Student (B.Sc '13)

Topics

I. Bridging the gap from simple to complex robotics
II. Introduction to ROS
III. ROS Software Development
IV. ROS in 6.141

Bridging the gap from simple to complex robotics

RSS Robot Hardware – Lab 2



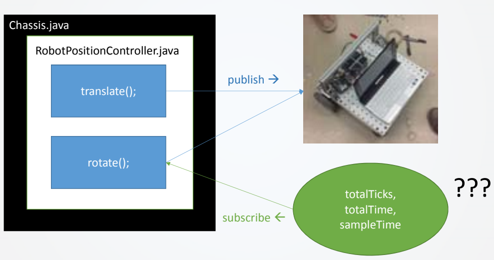
Actuators:

Motors

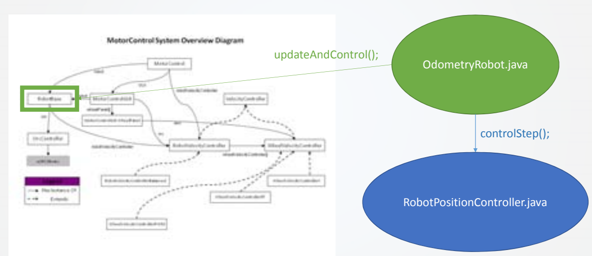
Sensors:

Encoders

RSS Robot Software – Lab 2



RSS Robot Software – Lab 2

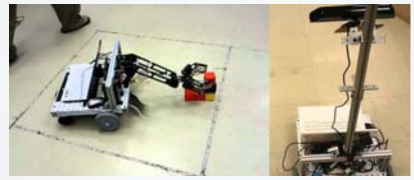


Problems with this implementation

Single pipeline bottleneck

This is an architecture only suitable for simple systems.

We will need an architecture which can support more complex hardware and software components.



Goal: Develop "big" software for robots

Challenges encountered in robotics

1. The world is asynchronous
2. Robots must manage significant complexity
3. Robot hardware requires abstraction

Problem 1: Sequential Programming

Lab 2 and conventional programming form:

```
goForward(1);
turnLeft(Math.PI/2);
Image image = camera.getImage();
double distance = computeDistanceToObject(image);
goForward(distance - 1);
(x, y) = getMyPositionFromTheEncoderCounts();
…
```

What happens if an obstacle appears while you are going forward?

What happens to the encoder data while you are turning?

What if some other module wants the same data?

Solution 1: "Callbacks"

Callback:

Function which is called whenever data is available for processing.

An *asynchronous callback* can happen at any time.

Examples:

Run the relevant *callback* function whenever:

- An image is read from the camera
- The odometry sensor reports new data

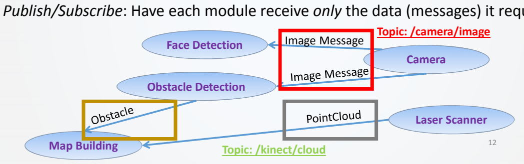
```
void imageCallback(ImageMessage image)
       // process the latest image

void odometryCallback(OdometryMessage data)
       // handle latest odometry data

void main()
       initialize();
       subscribe("image_msgs", imageCallback);
       subscribe("odometry_msgs", odometryCallback);
```

Problem 2: Complexity
Solution 2: Organizing code

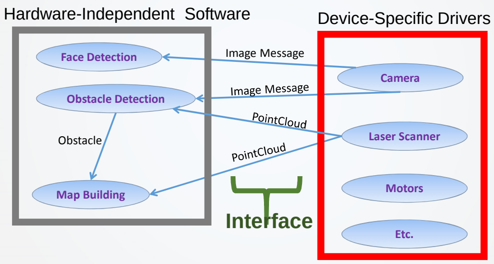
Separate processes: Cameras, Odometry, Laser Scanner, Map Building can all be separated out: they'll interact through an interface

Interfaces: Software processes ("nodes" in ROS) communicate about shared "topics" in ROS

Publish/Subscribe: Have each module receive *only* the data (messages) it requests



Problem 3: Hardware dependent code
Solution 3: Abstracting hardware



Result: Reusable code!

PR2

Roomba

Care-O-bot 3

MIT Urban Challenge Vehicle

Summary

We want:

- Callbacks
- Separate processes that communicate through a messaging interface
- A messaging interface that helps avoid hardware dependencies

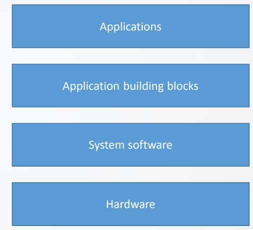
There's a software infrastructure out there that enables this (among **many** other things), and it's called ROS.

Introduction to ROS

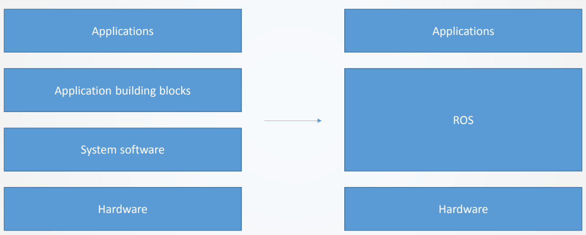
A meta-operating system for robots

Comparison: the PC ecosystem

- Standardized layers
- System software abstracts hardware
- Applications leverage other applications (such as database, web server).
- Widely existent sets of libraries



Comparison: the robotics ecosystem

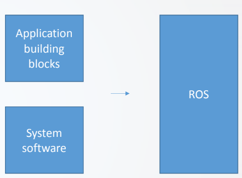


What is ROS?

- A “Meta” Operating System.
  - Open source
  - Runs in Linux (esp. Ubuntu)
  - Ongoing Windows implementation
- Agent based (nodes)
- Message passing
  - Publish
  - Subscribe
  - Services via remote invocation
- Supports numerous programming languages (C++, Python, Lisp, Java)

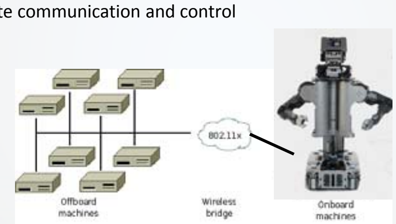
What is ROS?

- Low level device abstraction
  - Joystick
  - GPS
  - Camera
  - Controllers
  - Laser Scanners
  - …
- Application building blocks
  - Coordinate system transforms
  - Visualization tools
  - Debugging tools
  - Robust navigation stack (SLAM with loop closure)
  - Arm path planning
  - Object recognition
  - …



What is ROS?

- Software management (compiling, packaging)
- Remote communication and control

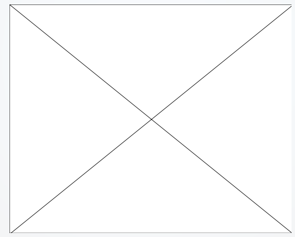


What is ROS?

- Founded by Willow Garage
- Exponential adoption
- Countless commercial, hobby, and academic robots use ROS (http://wiki.ros.org/Robots)

What is ROS?

- Software basis of Willow Labs's PR2

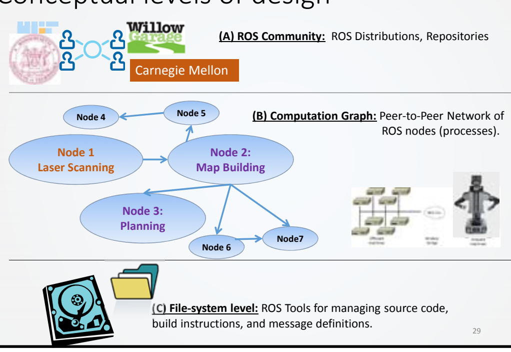


ROS Philosophical goals

- "Hardware agnosticism"
- Peer to peer
- Tools based software design
- Multiple language support (C++/Java/Python)
- Lightweight: runs only at the edge of your modules
- Free
- Open source
- Suitable for large scale research and industry

ROS software development

Conceptual levels of design



Tools-based software design

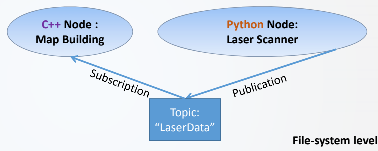
Tools for:

- Building ROS nodes
- Running ROS nodes
- Viewing network topology
- Monitoring network traffic

Many cooperating processes, instead of a single monolithic program.

Multiple language support

- ROS is implemented natively in each language.
- Quickly define messages in language-independent format.



File: PointCloud.msg

```
Header header
Points32[] pointsXYZ
int32 numPoints
```

File-system level

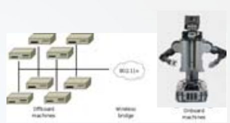
Lightweight

- Encourages standalone libraries with no ROS dependencies:
  Don't put ROS dependencies in the core of your algorithm!
- Use ROS only at the *edges* of your interconnected software modules: Downstream/Upstream interface
- ROS re-uses code from a variety of projects:
  - OpenCV : Computer Vision Library
  - Point Cloud Library (PCL) : 3D Data Processing
  - OpenRAVE : Motion Planning

ROS Community

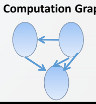
Carnegie Mellon

Peer to Peer Messaging

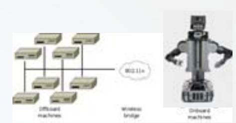


- No Central Server through which all messages are routed.
- "Master" service run on 1 machine for name registration + lookup
- Messaging Types:
  - Topics : *Asynchronous* data streaming
  - Parameter Server

Computation Graph

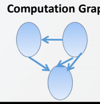


Peer to Peer Messaging



- *Master:* Lookup information, think DNS
  roscore command → starts master, parameter server, logging
- Publish: Will not block until receipt, messages get queued.
- Delivery Guarantees: Specify a queue size for publishers: If publishing too quickly, will buffer a maximum of X messages before throwing away old ones
- Transport Mechanism: TCPROS, uses TCP/IP
- Bandwidth: Consider where your data's going, and how

Computation Graph

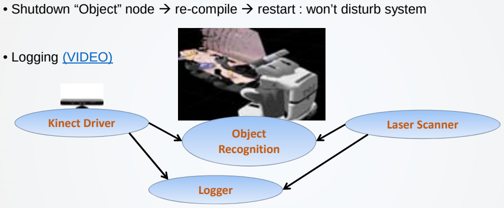


Free & Open Source

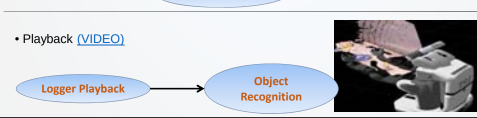
- BSD License : Can develop commercial applications
- Drivers (Kinect and others)
- Perception, Planning, Control libraries
- MIT ROS Packages : Kinect Demos, etc
- Interfaces to other libraries: OpenCV, etc

ROS Debugging

- Shutdown "Object" node → re-compile → restart : won't disturb system
- Logging (VIDEO)

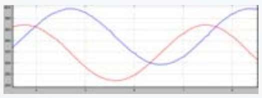


- Playback (VIDEO)

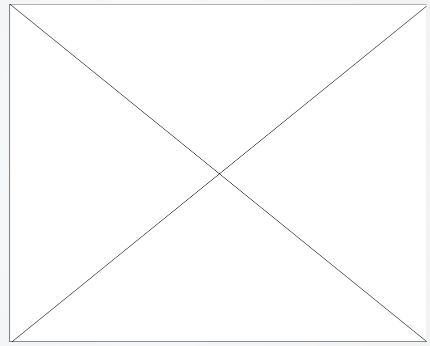


Useful ROS Debugging Tools

- `rostopic`: Display debug information about ROS topics: publishers, subscribers, publishing rate, and message content.
  - `rostopic echo [topic name]` → *prints messages to console*
  - `rostopic list` → *prints active topics*
  - … *(several more commands)*
- `rxplot` : Plot data from one or more ROS topic fields using matplotlib.
  - `rxplot /turtle1/pose/x,/turtle1/pose/y` → graph data from 2 topics in 1 plot

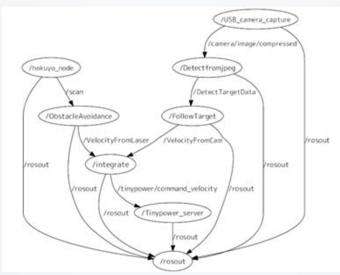


rosbag



Useful ROS Debugging Tools

`rxgraph`

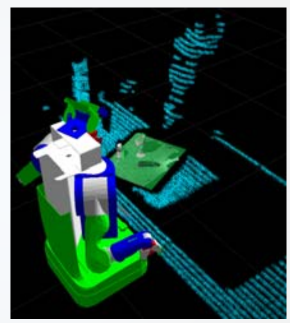


ROS Visualization

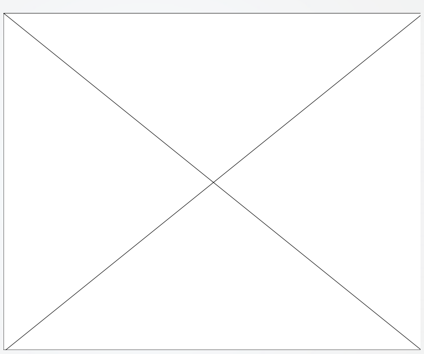
Visualize:

- Sensor data
- Robot joint states
- Coordinate frames
- Maps being built
- Debugging 3D markers

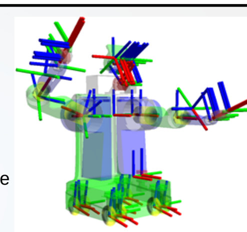
VIDEO



rviz



ROS Transformations



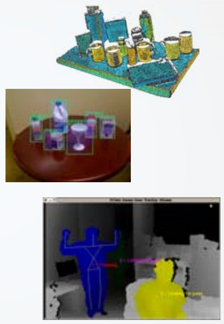
- "TF" = Name of Transform package
  "Tully Foote" == Person/Developer
- TF Handles transforms between coordinate frames : space + time
- tf_echo : print updated transforms in console

Example:

```
rosrun tf tf_echo [reference_frame] [target_frame]
```

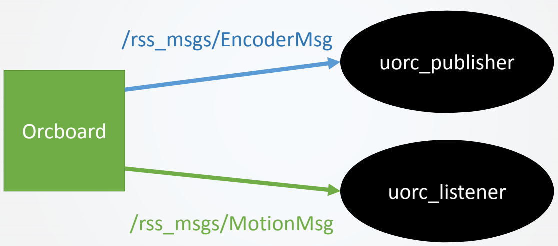
Packages

- Perception
  - Point Cloud Library (PCL)
  - OpenCV
  - Kinect/OpenNI

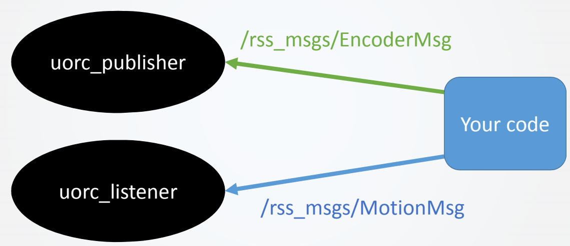


ROS in 6.141

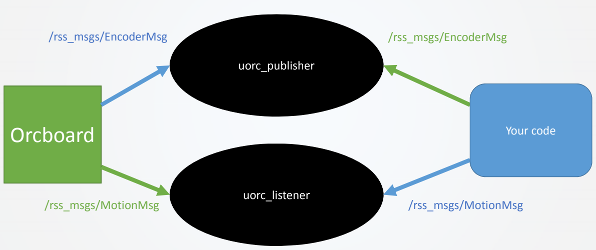
Reconciling Lab 2 with ROS



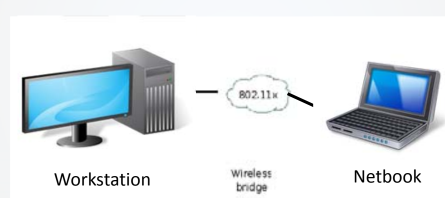
Reconciling Lab 2 with ROS



Reconciling Lab 2 with ROS



Peer to Peer

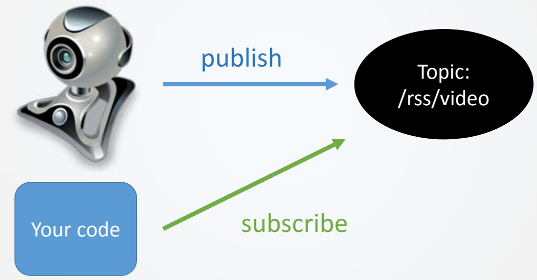


ROS code enviroment

Since our codebase is in JAVA, we use rosjava.

```
roscore → launch ROS host on netbook

rosmake → build a package (formerly ant)

roslaunch lab4 lab4.launch → launch package (formerly ant run)

*.launch file specifies additional parameters
```

Lab 3 – ROS and Visual Servoing



Lab 3 will contain a comprehensive overview of ROS

ROS Resources

- http://www.ros.org
- http://wiki.ros.org
- ROS Cheatsheet: http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf

Thank you!