ROS: "Robot Operating System"

6.141 / RSS Lecture 6
Monday, February 25th, 2013
Sudeep Pillai
MIT EECS PhD Student

1

3 Problems We Must Tackle in Developing Robot Software

(1) The world is asynchronous
Sequential programming paradigms are ill-suited

(2) Must manage significant complexity
Lots of moving parts, parameters, disturbances

(3) We want to abstract the details of specific robot hardware
Sensors, actuators

Consider Lab 2:

(1) Asynchronicity
E.g., odometry goal reached while motor is spinning

(2) Complexity
Motor encoders, PID controller, motion goals, …

(3) Hardware
Motors, encoders, wheels, wheelbase, Orcboard, …

Goal: Develop Big Software for Robots
Problem 1: Sequential Programming

How (some of) you are used to thinking about programs:

```
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 also wants the same data?

4

Alternative to Sequential Programming: *Callbacks*

Callback: Function that's called whenever data is available for processing.
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);
```

5

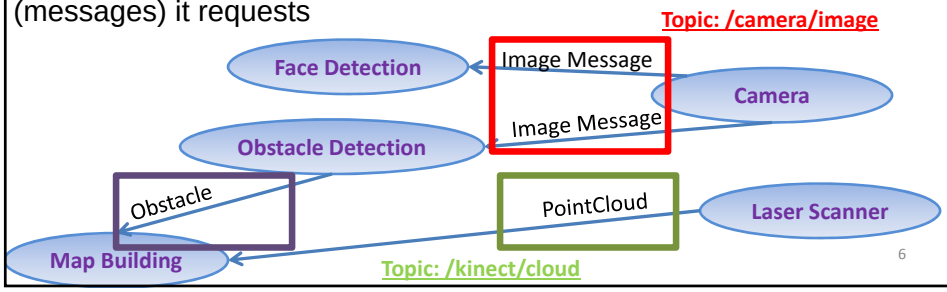
Goal: Develop Big Software for Robots
Problem 2: Complexity

How do we organize our code?

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

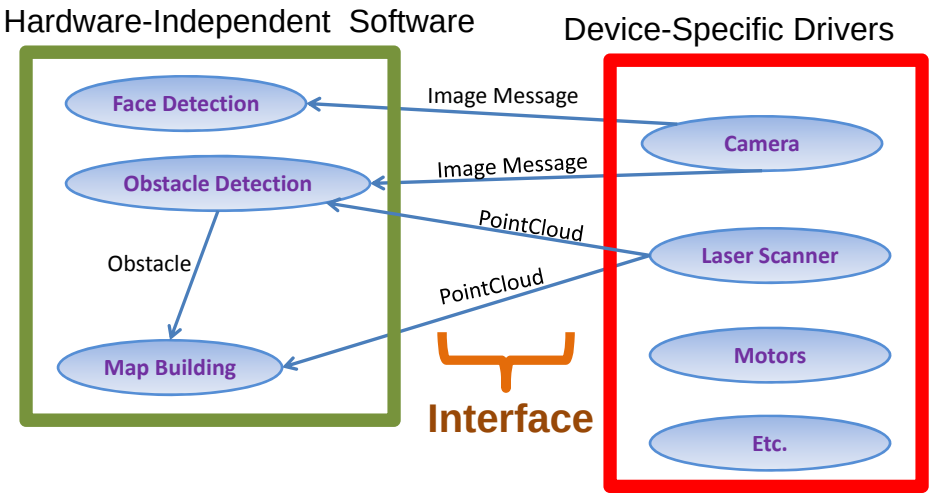
• *Interfaces*: Software processes ("nodes" in ROS) communicate about shared "topics" in ROS

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



6

Goal: Develop Big Software for Robots
Problem 3: Abstracting Hardware



7

Goal: Develop Complex Software for Robots
Problem 3: Abstracting Hardware

PR2

Roomba

Care-O-bot 3

REUSABLE CODE!

8

Goal: Develop Complex Software for Robots

MIT Urban Challenge Vehicle

9

Summary so Far

(1) ~~Sequential Programming~~
→ Callbacks

(2) ~~Complex, multi-function software~~
→ Separate processes that communicate only through a messaging interface

(3) ~~Hardware-dependent softwa~~re
→ Messaging interface helps avoid hardware dependencies

→ ROS : Supports this software structure for you.

10

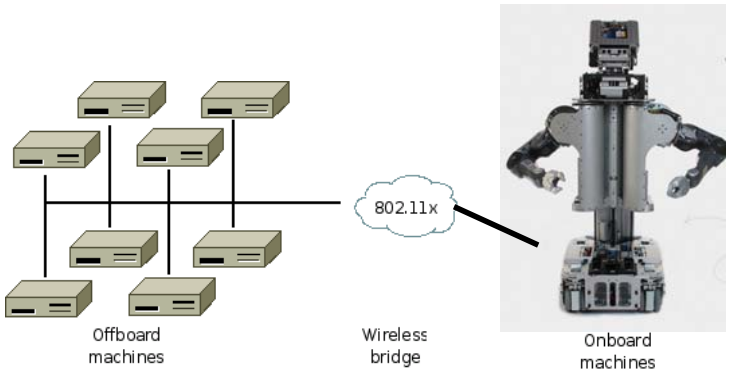
ROS : Robot "Operating" System

ROS Demo

http://youtu.be/UyLq4lfBsI0

11

ROS : Robot "Operating" System



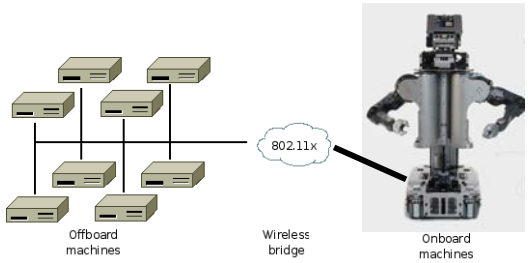
What is ROS?

- Message Passing
- Debugging tools
- Visualization tools
- Software Management (compiling, packaging)
- Libraries
- Nice abstraction for each of these elements

12

ROS : Goals for a Meta-Operating System

It is “Hardware Agnostic:”



- Peer-to-Peer
- Tools-based
- Multiple Languages (C++/Java/Python)
- Lightweight: Runs only at the edges of your modules
- Free & open-source
- Suitable for large-scale research

13

Outline

☑ **Introduction**
- ☑ 3 Software problems
- ☑ ROS Goals

☐ ***ROS Design*** ←
- ☐ Tools-Based
- ☐ Multiple Languages
- ☐ Lightweight
- ☐ Peer-to-peer
- ☐ Free & open-source

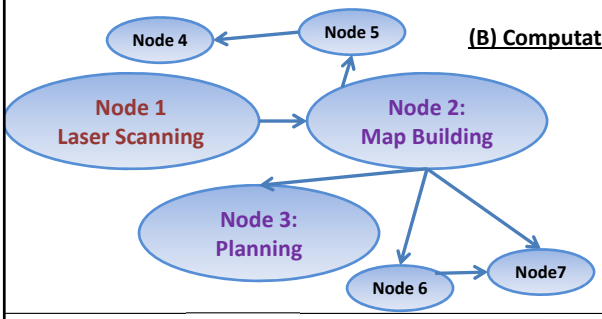
☐ **Developing Software with ROS**
- ☐ Debugging
- ☐ Visualizing
- ☐ Transforming Coordinate Frames

☐ **Packages : ROS and External**
- ☐ Perception
- ☐ Manipulation
- ☐ Navigation

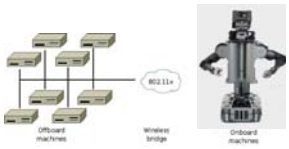
14

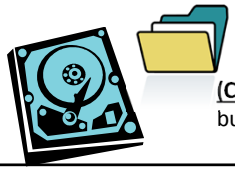
ROS Design : Conceptual Levels

(A) ROS Community: ROS Distributions, Repositories



(B) Computation Graph: Peer-to-Peer Network of ROS nodes (processes).





(C) File-system level: ROS Tools for managing source code, build instructions, and message definitions.

15

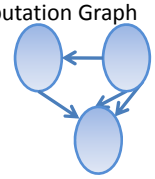
Tools-Based

- **Tools for:**
  - Building ROS nodes
  - Running ROS nodes
  - Viewing network topology
  - Monitoring network traffic

➔Not a single *monolithic program*
Instead: many cooperating *processes*

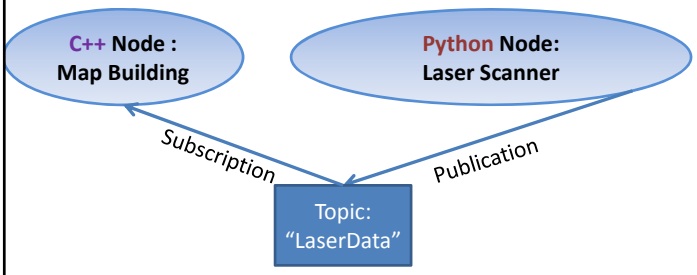
ROS Community

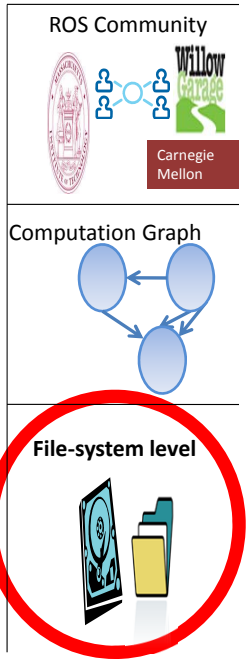
Computation Graph



File-system level

Multiple Languages





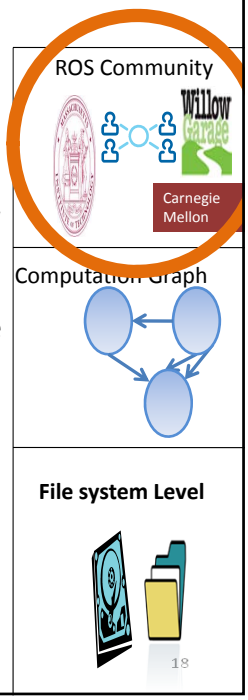
• ROS Implemented Natively In Each Language

• Quickly Define Messages in *Language-Independent* format:

File: PointCloud.msg

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

Lightweight



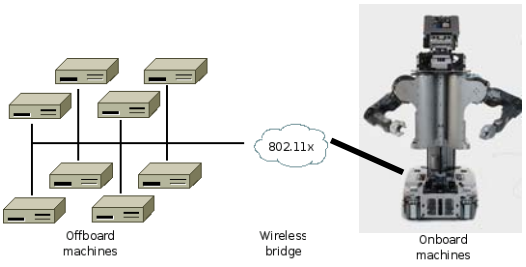
• Encourages standalone libraries with no ROS dependencies:
Don't put ROS dependencies in the core of your algorithm!

•Use ROS only at the *edges* of your interconnected software modules: Downstream/Upstream interface

• ROS re-uses code from a variety of projects:

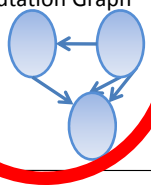
- •OpenCV : Computer Vision Library
- • Point Cloud Library (PCL) : 3D Data Processing
- • OpenRAVE : Motion Planning

Peer-To-Peer Messaging

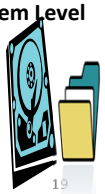


ROS Community

Computation Graph



File system Level



- 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

19

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.

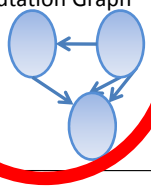
Delivery Guarantees: Specify a queue size for publishers: If publishing too quickly, will buffer a maximum of X messages before throwing away old ones

Transport Mechanism: TCPROS, uses TCP/IP

Bandwidth: Consider where your data's going, and how

ROS Community

Computation Graph



File-system level

20

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

Outline

☑ **Introduction**
- ☑ 3 Software problems
- ☑ ROS Goals

☑ **ROS Design**
- ☑ Tools-Based
- ☑ Multiple Languages
- ☑ Lightweight
- ☑ Peer-to-peer
- ☑ Free + Open Source

☐ **Developing Software with ROS** ←
- ☐ Debugging
- ☐ Visualizing
- ☐ Transforming Coordinate Frames

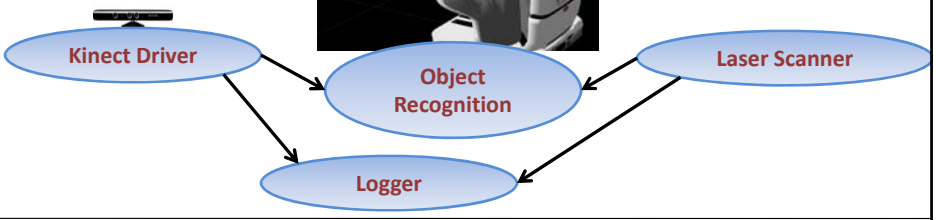
☐ **Packages : ROS and External**
- ☐ Perception
- ☐ Manipulation
- ☐ Navigation

22

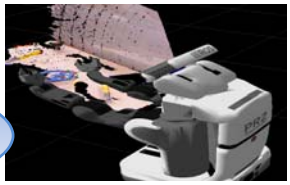
Development with ROS: Debugging

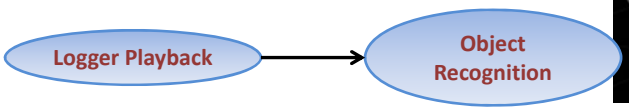
• Shutdown "Object" node → re-compile → restart : won't disturb system

• Logging (VIDEO)



• Playback (VIDEO)





Useful 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)

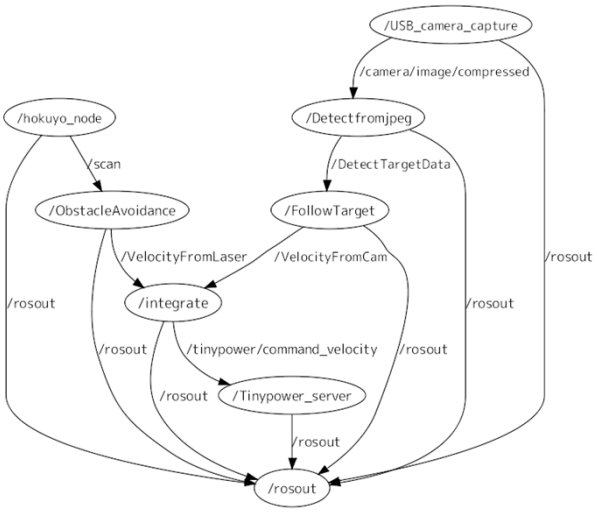
rxplot : Plot data from one or more ROS topic fields using matplotlib.

rxplot /turtle1/pose/x,/turtle1/pose/y → graph data from 2 topics in 1 plot

Useful quick reference:
http://mirror.umd.edu/roswiki/attachments/Documentation/ROScheatsheet.pdf

Useful Debugging Tools

rxgraph:



25

More Useful Development Tools: roslaunch

roslaunch : Used as a startup script. Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server

- Start Motor driver node
- Start Balanced Controller node
- Start Light sensor driver node
- More sensors…
- Start high-level navigation node

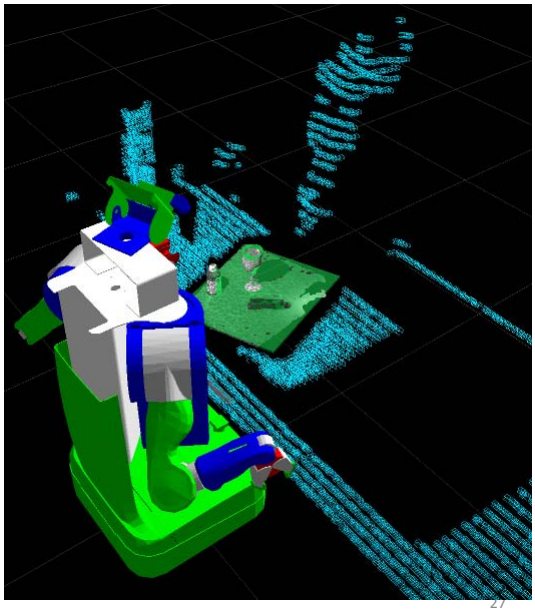
All these are encapsulated in a single roslaunch script

26

Development with ROS: Visualization

- Visualize:
  - Sensor Data
  - Robot Joint States
  - Coordinate Frames
  - Maps being built
  - Debugging 3D Markers

VIDEO



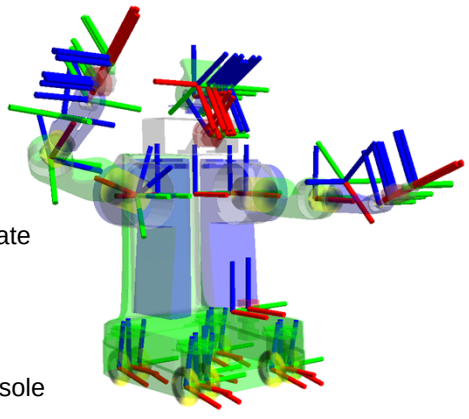
27

Development with 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]

28

Outline

☑ **Introduction**
- ☑ 3 Software problems
- ☑ ROS Goals

☑ **ROS Design**
- ☑ Tools-Based
- ☑ Multiple Languages
- ☑ Lightweight
- ☑ Peer-to-peer
- ☑ Free + Open Source

☑ **Developing Software with ROS**
- ☑ Debugging
- ☑ Visualizing
- ☑ Transforming Coordinate Frames

☐ **Packages : ROS and External** ←
- ☐ Perception
- ☐ Manipulation
- ☐ Navigation

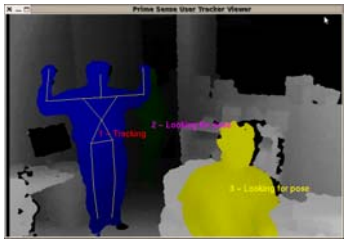
29

Packages: Perception

• Point Cloud Library (PCL)

• OpenCV

•Kinect / OpenNI :



30

Conclusion: tools to support development of complex software for robots

- *Reasons to use ROS:* Handling asynchronous world
  Managing complexity
  Abstracting various hardware
- *ROS Design:* Peer-to-Peer, Multiple Languages, Lightweight
- *Developing Software with ROS:* Debugging, Visualizing
- *Packages*

31

More Videos

Robotic Roommates Making Pancakes

32

References:

"ROS: an open-source Robot Operating System":
http://ai.stanford.edu/~mquigley/papers/icra2009-ros.pdf

www.ros.org **tutorials highly recommended**

33