Robot Operating System

Prof. Tiago Vieira, PhD

Universidade Federal de Alagoas

tvieira@ic.ufal.br

January 23, 2018

Contents

Introduction

Robot Operating System (ROS)

Preliminaries

The “fetch and item” task

The “fetch and item” task

Figure: Rosie. *Technische Universität München*.

Building a Robot is HARD!!

- Real-world variations.
- Different HW/SW.
- Interact with human intervention.

Robot Operating System (ROS)

Framework for writing robot software.

Motivation:

- Wide variaty of robots.
- Building robust, general-purpose software for robots is hard (picking something up is simple for humans)!

ROS composition

1. Drivers.
   - Read data from sensors.
   - Actuate motors.
2. Algorithms.
   - Navigation.
   - Data interpretation.
   - Environment mapping.
3. Framework.
   - Connection between different modules.
   - Incorportation of algorithms.
   - Distributed computing.

ROS composition

4. Tools
   - Visualization.
   - Debugging.
   - Data record.
5. Resources.
   - Community.
   - Documentation.

Requirements

- Python.
  - Programming and basic algorithms.
- Linux.
  - Filesystem.
- Robotics.
  - Coordinate transform.
  - Kinematic chains.

History

- Conceived from a necessity for collaboration between research centers.
- Mid-2000s Stanford projects.
  - STanford AI Robot (http://stair.stanford.edu/index.php) (STAIR).
  - Personal Robots (PR) program (http://personalrobotics.stanford.edu/).
- Created in 2007 stimulated by Willow Garage.

History

- BSD[1] open-source permissive license.
- Widely used in the robotics research community.
- In the beginning: Multiple institutions and multiple robots.
- Afterwards: This became a ROS strength.

[1] Berkeley Software Distribution (BSD)

Philosophy

ROS follows the UNIX philosophy.

- **Peer to peer**.
  - Numerous small programs communicating through *messages*.
  - No central routing.
- **Tools based**.
  - No canonical integrated development.
  - No runtime environment.
  - Tools can be exchanged.

Philosophy

ROS follows the UNIX philosophy.

- **Multi-lingual**.
  - High productivity scripting languages (Python, Ruby).
  - C++ for performance.
  - *Client* libraries allow the use of C++, Python, Ruby, Java, JavaScript, LISP and MATLAB.

Philosophy

ROS follows the UNIX philosophy.

- **Thin**.
  - ROS conventions encourage contributors to create stand-alone libraries and then wrap those libraries so they send and receive messages to/from other ROS modules.
- **Free and open-source**.
  - BSD license (commercial and non-commercial use).
  - ROS passes data between modules using inter-process communication (IPC).
  - Commercial systems, for example, often have several closed-source modules communicating with a large number of open-source modules.

Installation

- Ubuntu 16.04 LTS.
- ROS Kinetic.
- http://wiki.ros.org/kinetic/Installation/Ubuntu.

The ROS Graph

- *Computation graph*. Each program perform some piece of computation, passing data and results to other programs, arranged in a pre-defined network.

Fetch a Stapler Problem

Robot's task: Navigate an environment, find a stapler, and deliver it to some individual.

Relatively large and complex robot:

- Manipulator arm.
- Wheeled base.
- Several cameras.
- Laser scanners.

The task can be decomposed into many independent subsystems:

- Navigation.
- Computer vision.
- Grasping, etc;

It should be possible to:

- Re-purpose these subsystems for other tasks:
  - Security patrols.
  - Cleaning.
  - Delivering mail.

And: **The vast majority of the software should be able to run on any robot**.

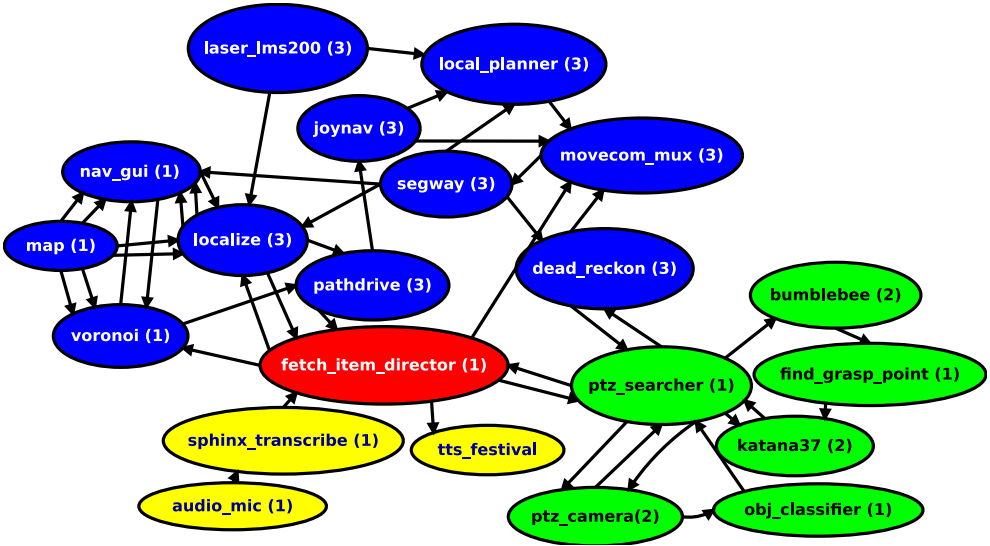
A ROS Graph

Many different programs running simultaneously and communicating through *messages*.

- Programs are *nodes*.
- *Edges* represent a stream of messages between two nodes.
- Typically, *nodes* are POSIX[2] processes and *edges* are TCP connections.

[2]Portable Operating System Interface (POSIX)

The Fetch a Stapler's Graph



Implementation of messages:

- Every program has a stdout[3].
- ROS extends this concept – arbitrary number of streams.

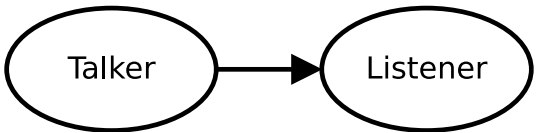
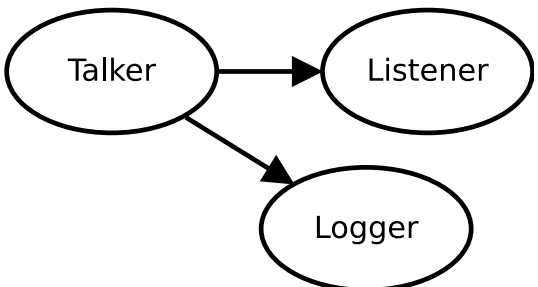


Figure: The simplest possible ROS graph: one program is sending messages to another program.

[3] "Standard Output"

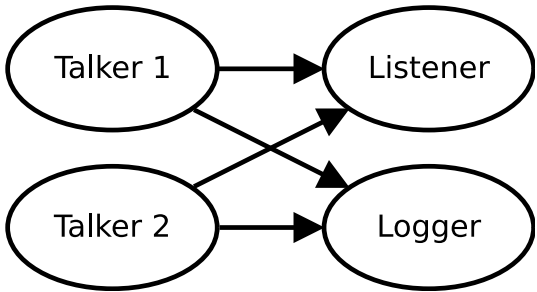
Implementation of messages:

- In ROS, nodes have no idea who they are connected to.
- A generic logger program writes all incoming strings to disk.



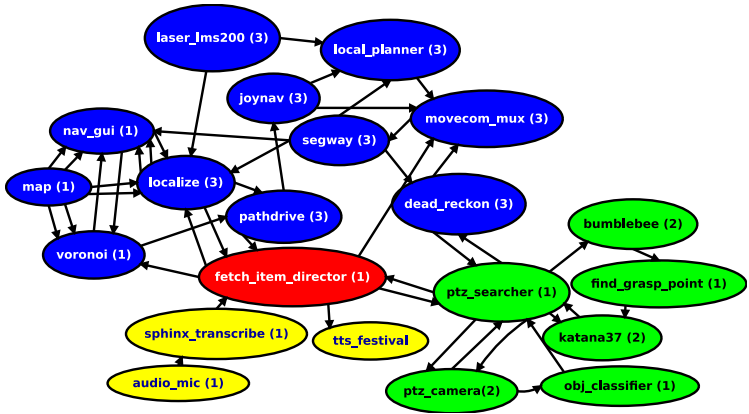
Implementation of messages:

- More complex topologies:



More complex topologies:

- 22 programs.
- 4 computers.



The Fetch a Stapler's Graph

- The graph is *sparse* (most nodes connect to small number of other nodes).
- Re-assess the flow of data if the graph is looking like a star.
- Separate the functions into smaller pieces.

The Fetch a Stapler's Graph

- Often, each graph node is running in its own POSIX process.
  - This offers additional fault tolerance: a software fault will only take down its own process.
  - The rest of the graph will stay up, passing messages as functioning as normal.
  - The circumstances leading up to the crash can often be re-created by logging the messages entering a node, and simply playing them back at a later time inside a debugger.

Benefits of graph-based architecture:

- Ability to rapid-prototype complex systems with little or no software "glue" required.
- Single nodes, can be swapped by simply launching an entirely different process.
- Not only can a single node be swapped, but entire chunks of graph can be torn down and replaced, even at runtime.
- And more:
  - Real-robot hardware drivers can be replaced with simulators.
  - navigation subsystems can be swapped.
  - algorithms can be tweaked and recompiled.

ROS creates all required network backend on-the-fly, the entire system is interactive and designed to encourage experimentation!

roscore

- roscore is a broker that provides connection information to nodes so that they can transmit messages to each other.
- Nodes register details of the messages they provide, and those that they want to subscribe to, and this is stored in roscore.
- Every ROS system needs a running roscore.
- Without it, nodes cannot find other nodes to get messages from or send messages to.

roscore

When a ROS node starts up:

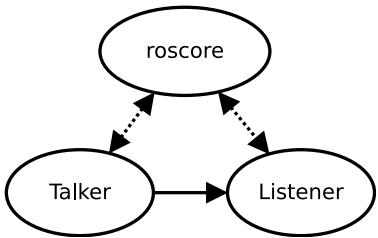
- It expects its process to have a POSIX environment variable named ROS_MASTER_URI.
- This is expected to be of the form http://hostname:11311/
- It must point to a running instance of roscore somewhere on the network.

roscore

- Different ports can be specified to allow multiple ROS systems to co-exist on a single LAN.
- With knowledge of the location of roscore on the network, nodes register themselves at startup with roscore, and can then query roscore to find other nodes and data streams by name.

roscore

- Each ROS node tells roscore which messages it provides, and which it would like to subscribe to.
- roscore then provides the addresses of therelevant message producers and consumers in Viewed in a graph form.
- Every node in the graph can periodically call on services provided by roscore to find their peers.



roscore

- Holds a parameter server which is used extensively by ROS nodes for configuration.
- The parameter server allows nodes to store and retrieve arbitrary data structures, such as;
  - descriptions of robots;
  - parameters for algorithms, and so on.
- simple command-line tool to interact with the parameter server: rosparam.

rosrun

- ROS software is organized into *packages*[4].
-

How does it work?

- In UNIX, every program has a stream called "STanDard OUTput".
- In ROS, this concept is extended so that programs can have an arbitrary number of streams.

[4] A collection of resources that are built and distributed together