Robotics programming with ROS

Introduction and installation

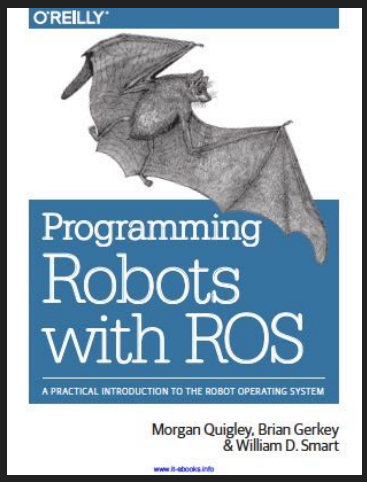
Adopted from Justin Huang
https://www.youtube.com/c/JustinHuang101/

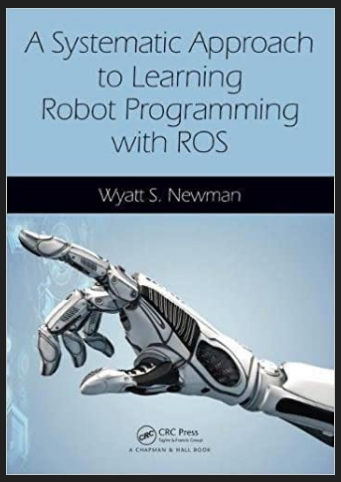
Google Self-Driving Car

Amazon Prime Air

Savioke Relay

Robotics in the real world





Good references

What is ROS?

- A software framework for programming robots
- Prototypes originated from Stanford AI research, officially created and developed by Willow Garage starting in 2007
- Currently maintained by Open Source Robotics Foundation
- Consists of infrastructure, tools, capabilities, and ecosystem

Advantages and Disadvantages of ROS

Advantages

- Provides lots of infrastructure, tools, and capabilities
- Easy to try other people's work and share your own
- Large community
- Free, open source, BSD license

Great for open-source and researchers

Disadvantages

- Approaching maturity, but still changing
- Security and scalability are not first-class concerns
- OSes other than Ubuntu Linux are not well supported

Not great for mission-critical tasks

About these tutorials

- Focus is on ROS concepts and software engineering
- Mostly developing with a Turtlebot in simulation
- A mix of explanation and coding
- Recommend checking out the official ROS tutorials and documentation

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

Turtlebot

Prerequisites

- Ubuntu Linux 14.04+
- A fairly recent/fast computer
  - Mostly to run the simulation
- Familiarity with developing in a Linux environment
- Knowledge of programming in C++ and/or Python

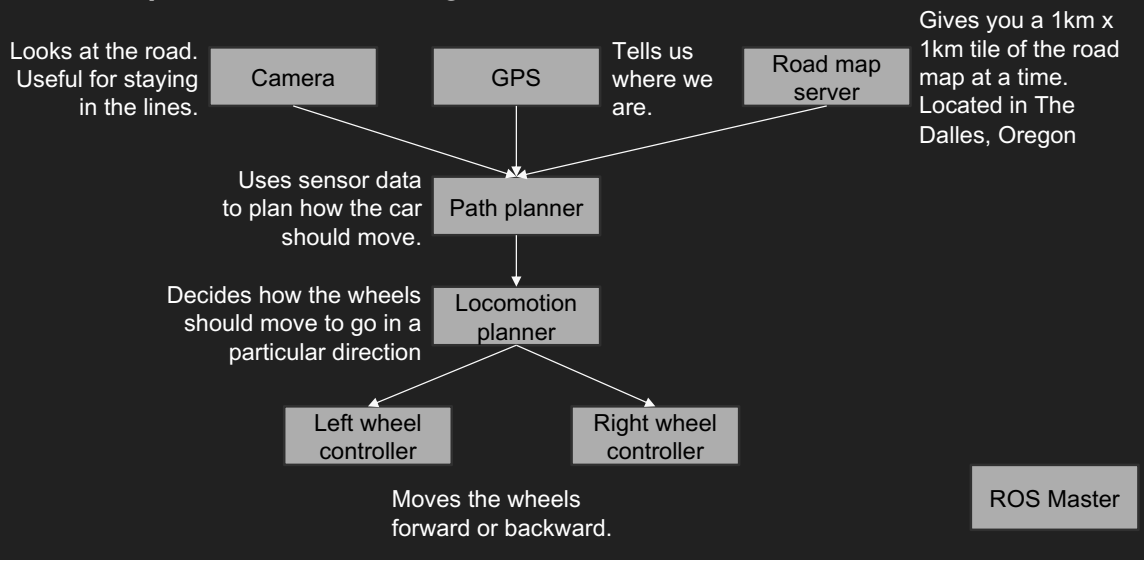
Architecture

- Organized as a network of *nodes*.
- Many nodes run on a computer, but the network can span across many computers.
- Each node performs a single task.
- Nodes coordinate with each other through *topics* and *services*.

ROS Master

- The ROS Master is a program that stores information about the network.
- Nodes register themselves with the Master on startup.
- Nodes ask the Master where to find other nodes. After that, nodes establish peer-to-peer communication with each other.

Example: self-driving car



Topics

- Topics are *streams* of data.
- Data can come consistently, many times a second (e.g., video)
- Data can come intermittently (e.g., email)
- A topic is uniquely identified by its name
- Nodes can publish data to a topic, or subscribe to data published onto it.

Services

- Think of it as a function call or an RPC.
- A server provides a service.
- A client calls the service, waits, then gets a response back.
- Also uniquely identified by name.

API design

Camera

Topics in: None
Topics out:
/camera_image
Services: None

GPS

Topics in: None
Topics out:
Publishes location to /gps
Services: None

Road map server

Topics in: None
Topics out: None
Services:
Tile get_tile(Location loc)

Path planner

Topics in:
/camera_image, /gps
Topics out:
/direction
Services: None

Locomotion planner

Topics in: /direction
Topics out:
/left_direction,
/right_direction
Services: None

Wheel controller

Topics in: /left_direction
Topics out: None
Services: None

Messages

- Main languages for ROS are C++ and Python.
- Messages are a language-agnostic way to represent data.

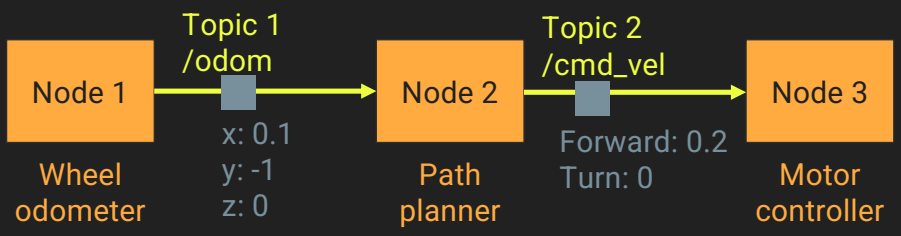
ROS Installation

- ROS.org
  - http://wiki.ros.org/melodic/Installation/Ubuntu
- Guides (dated, but still valuable)
  - http://file.ncnynl.com/ros/ros_by_example_v1_indigo.pdf
  - http://file.ncnynl.com/ros/ros_by_example_v2_indigo.pdf

Publishing and subscribing

ROS tutorial #2

ROS computation graph



Publishing and subscribing

- Any node can publish a message to any topic
- Any node can subscribe to any topic
- Multiple nodes can publish to the same topic
- Multiple nodes can subscribe to the same topic
- A node can publish to multiple topics
- A node can subscribe to multiple topics

Publish/subscribe tools

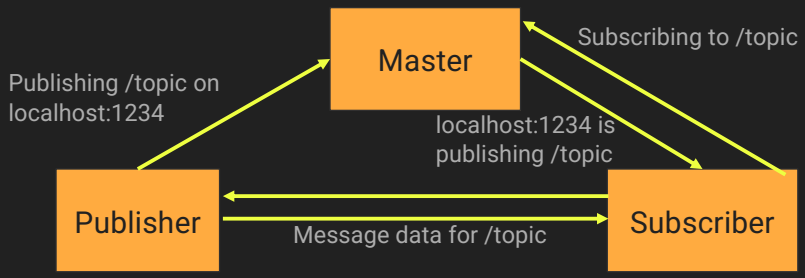
- rosnode list
- rosnode info /some_node
- rostopic list
- rostopic info /some_topic
- rostopic echo /some_topic
- rostopic pub /some_topic msg/MessageType "data: value"

Messages

- A serialization format for structured data
- Allows nodes written in C++ and Python to communicate with each other
- Defined in a .msg file
- Must be compiled into C++ / Python classes before using them

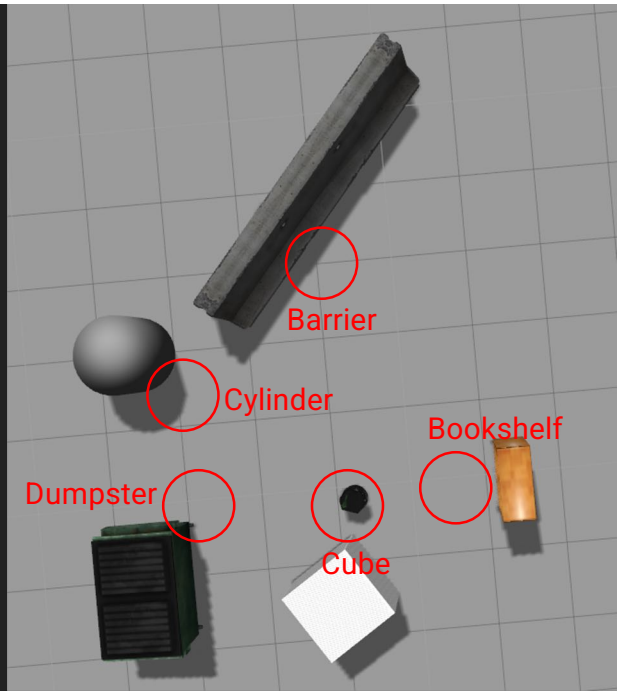
What is the ROS master?

- A server that tracks the network addresses of all other nodes
  - Also tracks other information like parameters
- Informs subscribers about nodes publishing on the same topic
- Publisher and subscriber establish a peer-to-peer connection
- Nodes must know network address of master on startup (ROS_MASTER_URI)
- Can be started with roscore or roslaunch



Are we there yet?

- Robot publishes the name and distance to the closest landmark
- Make the robot print out "I'm near the {landmark}" when it enters one of the circled zones
  - E.g., "I'm near the cube"
- Zones are circles with 1 meter diameter

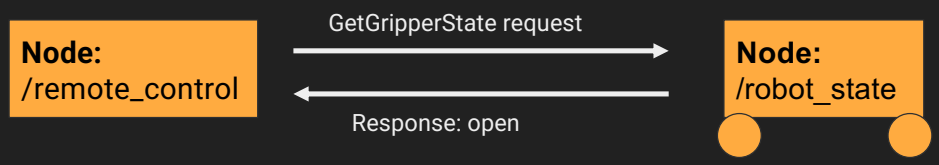


Services

ROS tutorial #3

What are services?

- ROS's remote procedure call (RPC)
- A node can implement one or more services (server)
- Any node can call a service (client)
- Calls are synchronous / blocking
  - Actions are preferred for long-running tasks



Service messages

- Must define a .srv message that defines a Request message and a Response message

Example: GetDistance.srv

```
string name

---

float64 distance
```

Service command-line tools

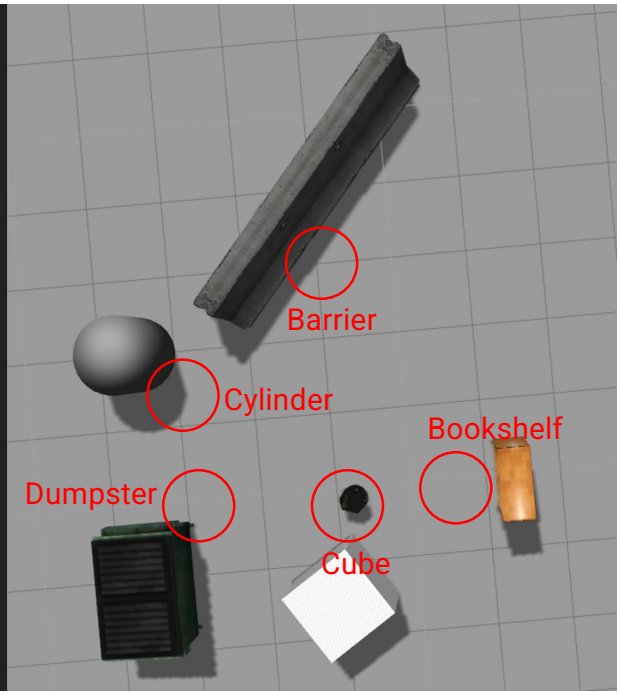
- rosservice list
- rosservice info /some_service
- rosservice call /some_service "param1: 0.0"
- rossrv show my_msgs/ServiceName

When to use services?

- In many cases, writing a library function is better
- Quick operations
  - For long running functions, use Actions instead
- C++ / Python interoperability
- When one computer is connected to real hardware

GetDistance and GetClosest

- We will create two services
- **GetDistance** will given in the name of a landmark and return the distance to the landmark
- **GetClosest** will return the name of the closest landmark



ROS1 vs ROS2

- Linux 14.04-20.04 (OS X?)
- TCP/IP
- C++ (and Python 2 (and some 3)
- Build System – Cmake
- One Node – One Process
- Launchfile (XML)
  - Fixed ROS Graph (in theory)
- No direct support
- Add-on
- Specialized Msg Format
- Programmer has responsibility
- Parameters in Launchfile
- Crawl file space

- HW
- IPC
- Language
- Build
- Nodes/Thread
- Orchestration

- Real-Time
- Security
- Middleware
- Namespaces
- Parameters
- Resource lookup

- Linux, OSX, Win 10, … (full CI)
- DDS: TCP, SHM, …
- C++ and Python 3.5+
- Python/Ament (build, install, test, …)
- One Node – One/Many Processes
- Flexible orchestration
  - ROS Graph is a true graph / dynamic
- Full RT support
- Integral to design
- Through DDS
- Each Node has its own namespace
- Separate parameter server
- Index server

More information

- Justin Huang YouTube channel (C++/Python node start)
- Wiki.ros.org & www.ros2.org
- Numerous hands-on guides to get started
- https://www.jetsonhacks.com/2019/10/23/install-ros-on-jetson-nano/