ROS

...

Resources

Official ROS Tutorials

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

“A Gentle Introduction to ROS” - Jason M. O’Kane

https://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf

Robohub’s ROS 101 Series

https://robohub.org/ros-101-intro-to-the-robot-operating-system/

Background

ROS- “Robot Operating System”

Initial roots at Stanford in mid-2000s as software projects for robotic development

Willow Garage heavily pushed forward the program

PR2 was the motivating platform

Continued open-source development


PR2

Background

Intent: A middleware for controlling robots

Robots need a distributed control system for handling joint actuation, sensors, high-level processing, etc.

ROS is a framework that helps this system communicate well

Installing

1. Choose a version
2. Setup keys
3. Install using apt-get
4. Initialize rosdep
5. Source appropriate installation

http://wiki.ros.org/melodic/Installation/Ubuntu

Versions

New version released every year

Versions typically matched with Ubuntu versions, so the “big” releases are every 2 years (Ubuntu LTS releases)

Moving to ROS 2: https://pythonclock.org/

Using ROS

Two client libraries: rospy & roscpp

Catkin workspaces for creating and building packages

Functionality available from sourcing built packages

Setting Up a ROS Workspace

Layout:

```
catkin_ws/
    src/
        CMakeLists.txt
        package1/
            CMakeLists.txt
            package.xml
    devel/
    build/
```

Setting Up a ROS Workspace

After creating catkin_ws and src directory…

catkin_create_pkg <package name> [depend 1] [depend 2] …

This generates package.xml file for you

To compile: “catkin_make” in workspace home directory

Don’t forget to source!

ROS Concepts

Overview of Concepts

- Master
- Nodes
- Messages
- Topics
- Services
- Rosbag
- Roslaunch
- Gazebo

ROS Master

Provides communication between all other nodes

Allows for distributed processing

Parameter server

Overview of Concepts

- ~~Master~~
- Nodes
- Messages
- Topics
- Services
- Rosbag
- Roslaunch
- Gazebo

Nodes

An individual computational process

Individual pieces of your control system should be created as individual nodes

e.g., How many nodes for a robot that uses a camera to move toward a goal object while avoiding obstacles?

Creating a New Node

In rospy:

```
rospy.init_node('node name here')
```

Command line arguments:

```
rospy.myargv(argv=sys.argv)
```

Spinning:

```
rospy.spin()
```

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- Messages
- Topics
- Services
- Rosbag
- Roslaunch
- Gazebo

Messages

Used for communication between nodes

Specific message types have to be used

Many predefined, can make custom ones as well

Messages

Example creation of a point message:

```
import geometry_msgs.msg

msg = geometry_msgs.msg.Point()

msg.x = 0
```

http://docs.ros.org/api/geometry_msgs/html/msg/Point.html

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- ~~Messages~~
- Topics
- Services
- Rosbag
- Roslaunch
- Gazebo

Topics

Named pathways for communication between nodes

Specific message type per topic

Asynchronous & unidirectional

Multiple publishers -- multiple subscribers

Creating and Publishing to a New Topic

To create a topic, initialize a publisher to a new named topic:

```
pub = rospy.Publisher('topic name', msg_type, queue_size)

...

pub.publish(my_msg)
```

Listening to a Topic

Creating a subscriber:

```
def callback(msg):
        print(msg.data)
rospy.Subscriber('topic name', msg type, callback)
```

Other options (single message, etc.)

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- ~~Messages~~
- ~~Topics~~
- Services
- Rosbag
- Roslaunch
- Gazebo

Services

Allows request-response calls instead of many-many pub/sub model

A service is defined through two messages: a request message and a response message

Creating a New Service

To create a service, we first have to defined a .srv file:

List of request message(s) type(s)

List of response message(s) type(s)

Creating a New Service

Example “AddTwoInts.srv” service that adds two ints:

```
int64 a

int64 b

---

int64 sum
```

Writing the Server

Assuming we have a .srv file ‘AddTwoInts’:

```
def server_callback(request):
    return AddTwoIntsResponse(request.a + request.b)

def add_two_ints_server():
    s = rospy.Service(‘add_two_ints’, AddTwoInts, server_callback)
    rospy.spin()
```

Using a Service

Wait for service to initialize, then call using a proxy:

```
rospy.wait_for_service('add_two_ints')

add_two_ints = rospy.ServiceProxy('add_two_ints',
AddTwoInts)

response_data = add_two_ints(4,5)

print(response_data.sum)
```

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- ~~Messages~~
- ~~Topics~~
- ~~Services~~
- Rosbag
- Roslaunch
- Gazebo

Rosbag

Stores ROS message data from ~~any~~ all topics

Can be used for debugging, to playback an episode, data recording for an experiment, etc.

Basic format:

```
rosbag record command_out control_signal sensor1
```

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- ~~Messages~~
- ~~Topics~~
- ~~Services~~
- ~~Rosbag~~
- Roslaunch
- Gazebo

Roslaunch

Used to start up multiple nodes, services, etc.

Timing can be an issue

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- ~~Messages~~
- ~~Topics~~
- ~~Services~~
- ~~Rosbag~~
- ~~Roslaunch~~
- Gazebo

Gazebo

Simulation tool for ROS

Useful for testing code without breaking expensive robots

Many robots have meshes predefined for use with Gazebo

Overview of Concepts

- ~~Master~~
- ~~Nodes~~
- ~~Messages~~
- ~~Topics~~
- ~~Services~~
- ~~Rosbag~~
- ~~Roslaunch~~
- ~~Gazebo~~

Example Robot Control