Introduction to ROS with Python

Evan Krell

Texas A&M University - Corpus Christi

November 2018

Outline

Introduction

Getting Started

Writing ROS Programs

Log Messages

Graph Resources

Launch Files

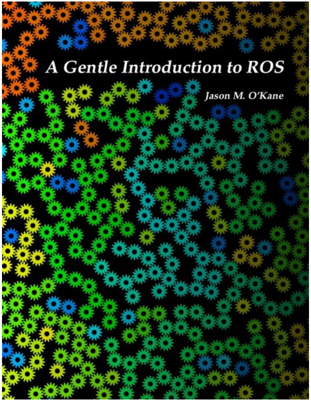
Parameters

Services

Recording & Replaying Messages

References

Sources



Jason M. O'Kane
`cse.sc.edu/~jokane/agitr`

Structure Python-based ROS Package
Simon Birrel
`artificialhumancompanions.com`

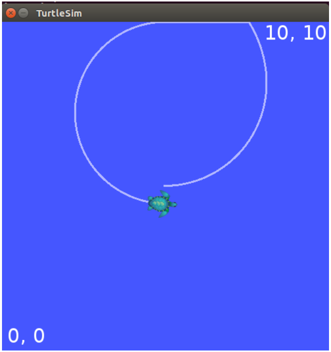
Package for this Tutorial

The content presented is demonstrated in an ROS package written in Python.

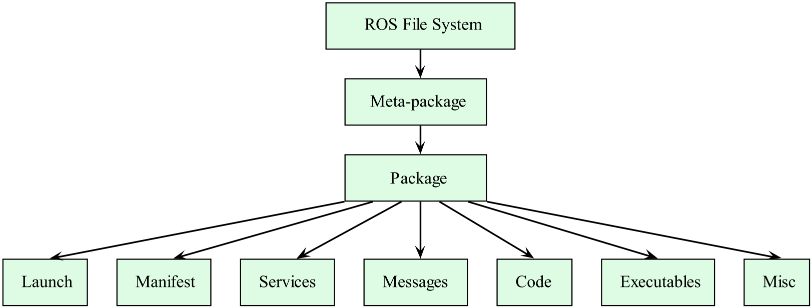
Location

`https://github.com/ekrell/ros_python_workshop`

Turtlesim Environment



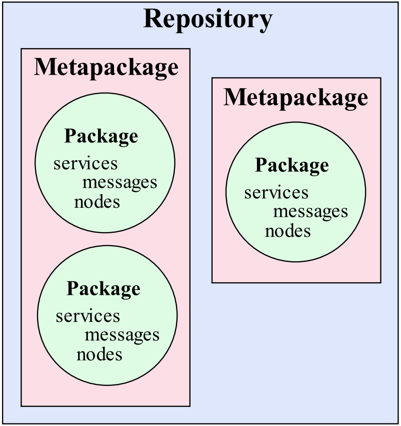
Packages


ROS File System
Meta-package
Package
Launch
Manifest
Services
Messages
Code
Executables
Misc

Getting Started

Packages

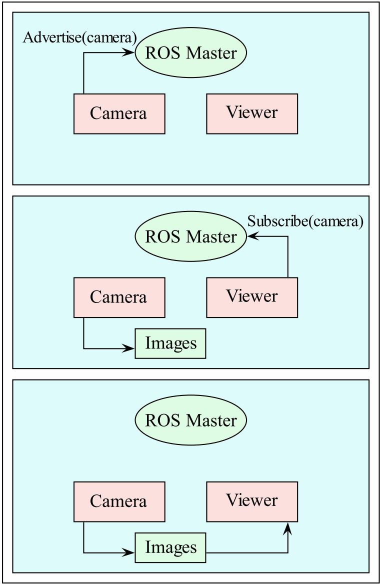
- Collection of files that fulfill single purpose (code, executables, etc)
- Simply a directory with **manifest** file called `package.xml`
- **Manifest** file has package definition, with name, version, dependencies
- Facilitates organization, sharing



Getting Started

ROS Master

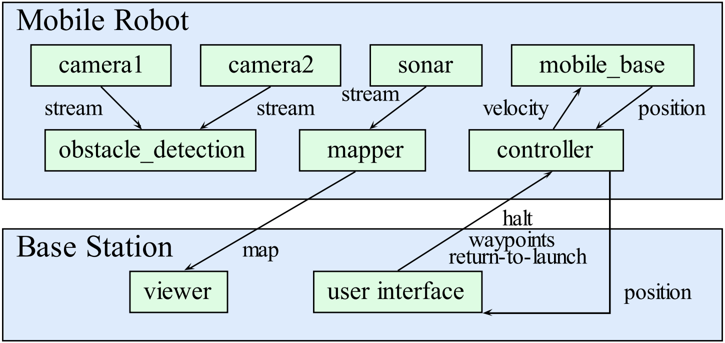
- Maintains directory of nodes, messages, services, parameters, etc
- Enables communication among nodes
- **Parameter server**: directory of parameters and values



Getting Started

Nodes

- Single executable using ROS
- Communicate over **topics** (publish, subscribe)



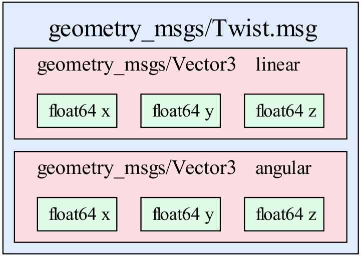
Getting Started

Topics

- Named buses for node communication
- Each has a specific message **type**
- Types are integers, floats, strings, and composite structures

Messages

- Units of communication
- Each message is of a specific **type**



geometry_msgs/Twist.msg

geometry_msgs/Vector3 linear

float64 x float64 y float64 z

geometry_msgs/Vector3 angular

float64 x float64 y float64 z

Writing ROS Programs

Install ROS

Installation guide: `wiki.ros.org/ROS/Installation`

Setup Catkin

catkin: build system for ROS `/wiki.ros.org/catkin`

ROS for Python

rospy: ROS Python library `wiki.ros.org/rospy/Tutorials`

Writing ROS Programs

Install ROS Package

```
cd ~/catkin_ws/src
git clone https://github.com/ekrell/ros_python_workshop.git
cd ~/catkin_ws
catkin_make
```

Execute ROS package

```
roscore
rosrun PACKAGE_NAME SCRIPT.py
```

Log Messages

ROS Logging

View log in console: `rostopic echo /rosout`

View log in GUI: `rqt_console`

Log Message Severity

Debug: `rospy.logdebug(msg, *args)` Lowest severity

Warn: `rospy.logwarn(msg, *args)`

Info: `rospy.loginfo(msg, *args)` ...

Error: `rospy.logerr(msg, *args)`

Fatal: `rospy.logfatal(msg, *args)` Highest severity

Log Messages

Python Example

```
rospy.loginfo_throttle(10, status2str(pose, params["goal"]))
```

Result

```
rostopic echo /rosout
```

```
    level: 2
    name: "/purepursuit"
    msg: "Position:␣(x:5.5,␣y:5.5,␣theta:0.0),␣Goal:␣(x:9,␣y:9)"
```

Graph Resources

Naming Scheme

- ROS organizes nodes, topics, services, parameters in graph
- Thus, elements are called graph resources
- Flexible naming scheme for referencing these resources
- Facilitates modularity and existence of duplicate executions of same node
- But can be difficult to find where resources come from at first

/turtle1	+	cmd_vel	=>	/turtle1/cmd_vel
current namespace		relative name		global name

Launch Files

Launch Multiple Nodes

- Launch files setup and run multiple nodes
- Relieves burden of opening multiple terminals, executing each node in order, remembering all parameters, etc
- **Modular:** launch files can call launch files
- `roslaunch ros_python_workshop ros_python_workshop`
- **Ctrl-C** will (ideally) gracefully shut down each node

Example

`ros_python_workshop/launch/ros_python_workshop.launch`

`rosnode list`

```
/purepursuit
/rosout
/turtlesim_node
```

Parameters

Parameter Server

- Handled within ROS Master
- Dictionary shared by nodes
- Setting & getting inside and outside node
- Just **strings**, not ROS message types
- **Caution:** Node must manually check for param changes

Console Basics

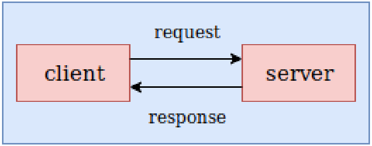
```
rosparam list

rosparam get goal

rosparam get roslaunch

rosparam set goal "[9, 9]"

rosparam dump testdump.txt roslaunch

rosparam load testdump.txt roslaunch
```

Services

Services

Publishing and subscribing to topics are not the only way to handle messages in ROS. Services are similar to the message passing you may be familiar with in MPI. Content of messages specified by *service data type*.

- Service calls are bi-directional: a sender expects a response
  - With topics: you just send messages. No idea if any nodes are subscribed
  - With services: send to a specific node and wait for response
- Service calls are one-to-one:
- With topics: arbitrary number of opt-in recipients.
- With services:
  - A sends *request* to B: A → B
  - B sends *respond* to A: A ← B



Services

Command Line Service Management

List all services: `rosparam list`

```
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

List all node-specific services: `rosnode info turtlesim`

```
/turtle1/teleport_absolute
/turtlesim/get_loggers
/turtlesim/set_logger_level
/reset
/spawn
/clear
```

Services

Command Line Service Management

Find a service's host node: `rosservice node /spawn`

```
    /turtlesim
```

Find a service's data type: `rosservice into /spawn`

```
    Node: /turtlesim
    URI: <your URI>
    Type: turtlesim/Spawn
    Args: x y theta name
```

Inspect a service's data type: `rossrv show turtlesim/Spawn`

```
float32 x
float32 y
float32 theta
string name
---
string name
```

Call a service: `rosservice call /spawn 5 5 0 Sally`
Adds turtle named Sally at with position (x:5, y:5, theta:0)
The */spawn* service was used within the turtlesim code.

Recording & Replaying Messages

Message-based Architecture

- Core to ROS: nodes act upon information on topics & services
- Nodes should not care *who* sends that information
- Example: turtlebot should not know if move commands come from command line, keyboard, or joystick

rosbag

- rosbag allows you to record messages and replay them
- Start recording:
  ```
  rosbag record -O filename.bag topic-names
  ```
  All messages published on the topic-names will be recorded to file filename.bag
- Replay recording:
  ```
  rosbag play filename.bag
  ```
  Those messages will be republished on their original topics.
  Original timing is preserved!

Recording & Replaying Messages

Bags in Launch Files

- A launch file *record* node

```
<node
pkg="rosbag"
name="record"
type="record"
args="-O filename.bag topic-names"
/>
```

- Launch file *play* node

```
<node
pkg="rosbag"
name="play"
type="play"
args="filename.bag"
/>
```

Image Sources

Slide 5	Mastering ROS for Robotics Programming
Slide 7	`wiki.ros.org/Master`
Slide 8	ASV C-Worker USV