Quick Introduction to ROS
ROS is huge

ROS is an open-source, meta-operating system for humanoid robots
What can ROS do?

- Hardware abstraction
- Low-level device control
- Message passing between nodes
- Sophisticated build environment
- Libraries
- Debugging and Visualization Tools
What are the major concepts?

- ROS packages
- ROS messages
- ROS nodes
- ROS services
- ROS action servers
- ROS topics
- ...and many more!
Installing Ubuntu 14.04
Quick and painless with Virtualbox
Open Virtualbox

- Install Virtualbox
- Download Ubuntu 14.04
- Install to a USB drive
Time for a demo!
Let’s install Ubuntu on a USB drive. I’ve uploaded a video of this to Youtube here: https://youtu.be/UGl0x2ZT_cI
What is ROS?

Getting started with the concepts
What can ROS do?

- Research development
  - Fast prototyping easier in a simulated world
- Transferring from simulated robot to real robot takes a bit of effort
ROS Concepts
Like HTTP but with extra steps
The ROS framework is a graph
Each component is called a node
- A node is a process
- Nodes communicate through topics, services, and actions
ROS as a framework

- ROS Master sends/receives
- Several nodes at once
- Whole network on your computer
ROS as a framework cont.

- Kinect2 → /kinect2/images
- Publishes image messages
- What are messages?
So what does this mean?

- Hardware talks to drivers, which then talk to nodes, which then talks to ROS
- Nodes can run any software you want as long as it is a language ROS supports
Topics

- Each node can listen on or publish messages to topics
  - Built in message types (std_msgs)
  - User defined messages

**Complex.msg**

- `float32 real`
- `float32 imaginary`
All ROS messages are viewable
A node can provide services – synchronous remote procedure calls

- Request
- Response

Add.srv

```plaintext
float32 x
float32 y
---
#Three dashes separate the request and response
Float32 result
```
Can view all ROS services

nwchen@strategy /opt/ros/indigo/share/std_srvs/srv
> $ ls
Empty.srv  SetBool.srv  Trigger.srv

nwchen@strategy /opt/ros/indigo/share/std_srvs/srv
> $ cat Trigger.srv
---
bool success  # indicate successful run of triggered service
string message  # informational, e.g. for error messages
Actions (asynchronous) are for long-running processes.

They have a Goal, Result, and Feedback

- **Navigation.action**
  ```
  float32 dest_x
  float32 dest_y
  ```
  # Example Action

  ```
  boolean success
  ```
  # Result

  ```
  uint32 percent_complete
  ```
  # Feedback
Can view all ROS actions

cat TestRequest.action
int32 TERMINATE_SUCCESS = 0
int32 TERMINATE_ABORTED = 1
int32 TERMINATE_REJECTED = 2
int32 TERMINATE_LOSE = 3
int32 TERMINATE_DROP = 4
int32 TERMINATE_EXCEPTION = 5

bool ignore_cancel  # If true, ignores requests to cancel
string result_text
int32 the_result    # Desired value for the_result in the Result
bool is_simple_client

duration delay_accept  # Delays accepting the goal by this amount of time
duration delay_terminate  # Delays terminating for this amount of time
duration pause_status  # Pauses the status messages for this amount of time

int32 the_result
bool is_simple_server

...
ROS software is organized into **packages**
- Each package contains some combination of code, data, and documentation

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name/</td>
<td></td>
</tr>
<tr>
<td>package.xml</td>
<td>describes the package and its dependencies</td>
</tr>
<tr>
<td>CMakeLists.txt</td>
<td>Finds other required packages and messages/services/actions</td>
</tr>
<tr>
<td>src/</td>
<td>C++ source code for your node (includes go in include/ folder)</td>
</tr>
<tr>
<td>scripts/</td>
<td>Python scripts for your node</td>
</tr>
<tr>
<td>msg/</td>
<td>ROS messages defined for your node (for topics)</td>
</tr>
<tr>
<td>srv/</td>
<td>ROS services defined for your node (for services)</td>
</tr>
<tr>
<td>launch/</td>
<td>The folder that contains .launch files for this package</td>
</tr>
</tbody>
</table>
Building/Running

- **Catkin** is the official build system of ROS
  - Catkin combines Cmake macros and Python scripts to provide some functionality on top of Cmake’s normal workflow
- Run ROS code

```bash
$ rosrun <package_name> <script>
$ rosrun <package_name> <launch_file>
```
Launch Files

- Automate the launching of collections of ROS nodes via XML files and `roslaunch`

```xml
example.launch:
<launch>
  <node name="talker" pkg="rospy_tutorials"
       type="talker.py" output="screen" />
  <node name="listener" pkg="rospy_tutorials"
       type="listener.py" output="screen" />
</launch>
```

```
$ roslaunch rospy_tutorials example.launch
```
Launch Files

You can also pass parameters via launch files

```xml
<launch>
  <arg name="gui" default="true"/>
  <param name="/use_sim_time" value="true" />
  <include file="$(find_gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="worlds/willowgarage.world" />
    <arg name="gui" value="$(arg gui)" />
  </include>
  <include file="$(find pr2_gazebo)/launch/pr2.launch"/>
  <node name="spawn_table" pkg="gazebo_ros" type="spawn_model"
    args="-urdf -file $(find humanoids_robots)/pr2_gazebo_pick_object/scenario/objects/table.urdf
    -model table -x 2.15 -y 0.5"
    respawn="false" output="screen" />
</launch>
```
Command Line Tools

```bash
$ rosnode list
$ rostopic list
$ rostopic echo
$ rosmsg show
$ rosservice
$ tf viewframes
```
Robots + ROS

- Sensor Data
- Joint Trajectories

Your ROS Code Here
Robots available - Fetch

Provides Data From (sensors):
- Depth camera
- Laser scanner
- Head camera
- Current Joint States
Robots available - PR2

Provides Data From (sensors):
- Kinect
- Two Laser Scanners
- Multiple Cameras (head and hand cameras)
- Fingertip pressure sensor arrays (gripper)
- Current Joint States
Robots available - Baxter

- More cost-effective
- Also has 2 arms
- Stationary base
- Sensors:
  - Sonar
  - Hand and head cameras
  - Hand rangefinders
Robots in the wild - Problems

- I don’t have a Robot in front of me
- I want to try something that may break my Robot
- Setting up the Robot takes too much time, I want to test changes to my code quickly
Gazebo Simulator
Gazebo Simulator

- Same interface as real Fetch, PR2 or Baxter
- Add/remove items in environment
- Physics engine to simulate effects of motor commands and provide updated sensor feedback
The organization that makes the robot often provides a Gazebo setup package for that robot.

For example:  
https://github.com/fetchrobotics/fetch_gazebo
Gazebo Simulator Demo

roslaunch fetch_gazebo playground.launch

rosrun applications keyboard_teleop.py

rostopic list | grep gazebo
A robotic system typically has many 3D coordinate frames that change over time.

tf keeps track of all these frames over time.
RViz and tf Demo

rosrun rviz rviz
Moveit!
Moveit!

- **Given:**
  - Current State of Arm
  - Desired End Effector Pose
  - Scene
- **Returns:**
  - Trajectory to Move End Effector to Desired Pose
Moveit!

- Provides a common interface to several different planners
- Probabilistic Planners: will not return the same path every time and may not even find a path reliably.
The organization that makes the robot often provides a MoveIt! config package.

These will provide information about robot joints, links, control information. For example:
1. joints of each group
2. end-effector of each group
3. joint limits
4. default planners

For example: https://github.com/fetchrobotics/fetch_ros
Moveit! demo

roslaunch fetch_moveit_config move_group.launch
MoveIt! Python interfaces

- moveit_commander

- moveit_python
  - [https://github.com/mikeferguson/moveit_python](https://github.com/mikeferguson/moveit_python)
Graspit!

- Grasp planner
- Lots of robots and objects
Graspit! demo

roslaunch graspit_interface graspit_interface.launch

Python interface: graspit_commander
https://github.com/graspit-simulator/graspit_commander
import graspit_commander
gc = graspit_commander.GraspitCommander()
gc.clearWorld()
gc.importRobot('fetch_gripper')
gc.importGraspableBody("longBox")

grasps = gc.planGrasps()
grasps = grasps.grasps
If you have a question

- Look in Tutorials:
- Reference class slides/codes provided
- Google it
- [http://answers.ros.org/questions/](http://answers.ros.org/questions/)
- Ask a TA
Some project tips

- Get going early.
- Start from a simple prototype.
- Seek help.
- Several robot platforms available (Fetch, PR2, and Baxter)
Homework are out!

- Homework deadline: Feb. 26
- Start early, last part is a little bit annoying
- Paper choice deadline: Feb. 5
- Walk-through