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!

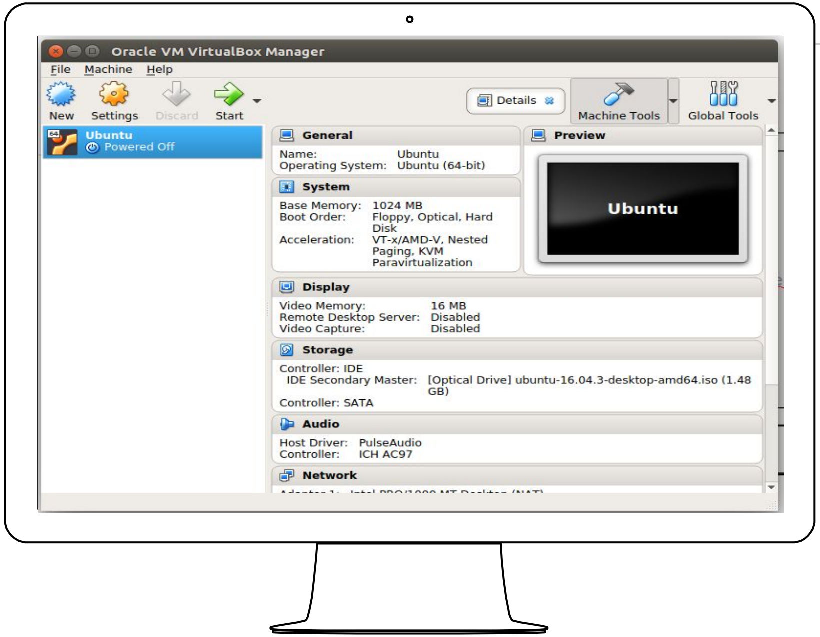
1

Installing Ubuntu 14.04

Quick and painless with Virtualbox

Open Virtualbox

- Install Virtualbox
- Download Ubuntu 14.04
- Install to a USB drive



General

Name: Ubuntu
Operating System: Ubuntu (64-bit)

System

Base Memory: 1024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE
IDE Secondary Master: [Optical Drive] ubuntu-16.04.3-desktop-amd64.iso (1.48 GB)
Controller: SATA

Audio

Host Driver: PulseAudio
Controller: ICH AC97

Network

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

2

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

3

ROS Concepts

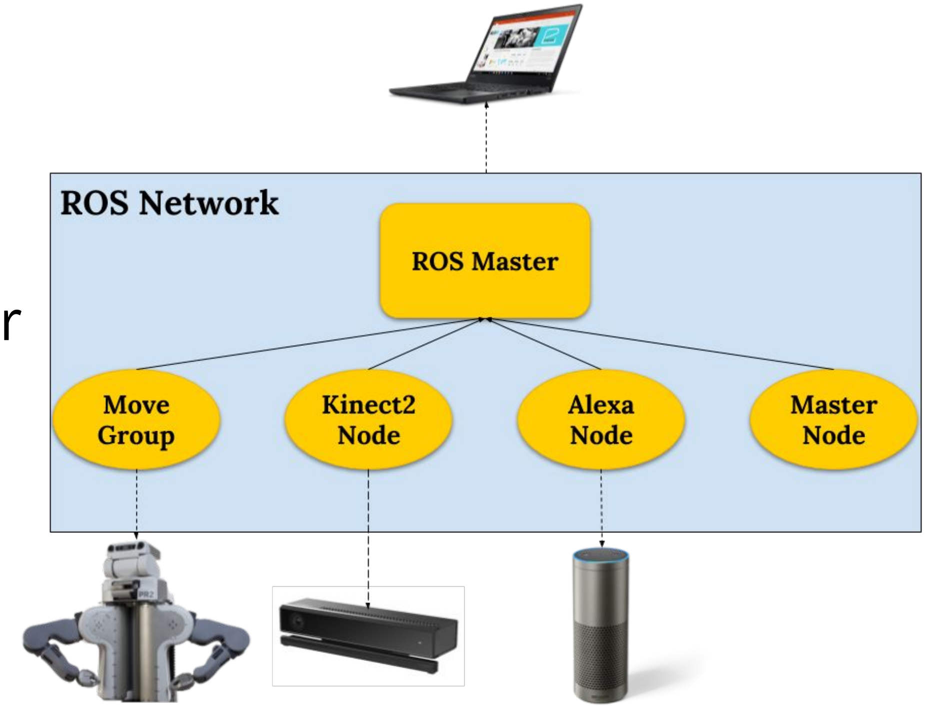
Like HTTP but with extra steps

ROS Nodes

- The ROS framework is a graph
- Each component is called a node
  - A node is a process
  - Nodes communicate through **topics**, **services**, and **actions**

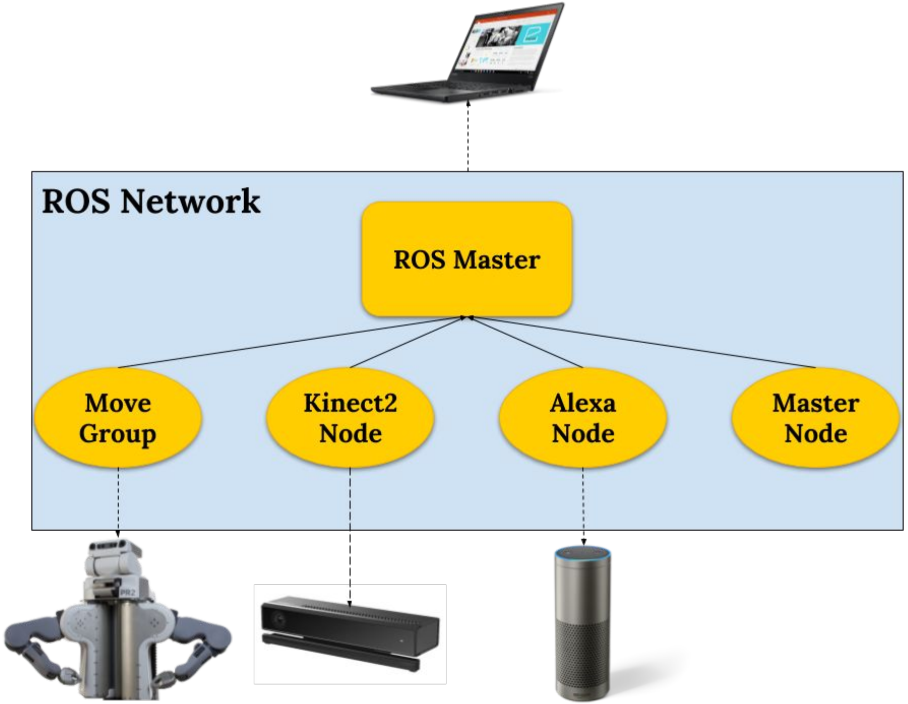
ROS as a framework

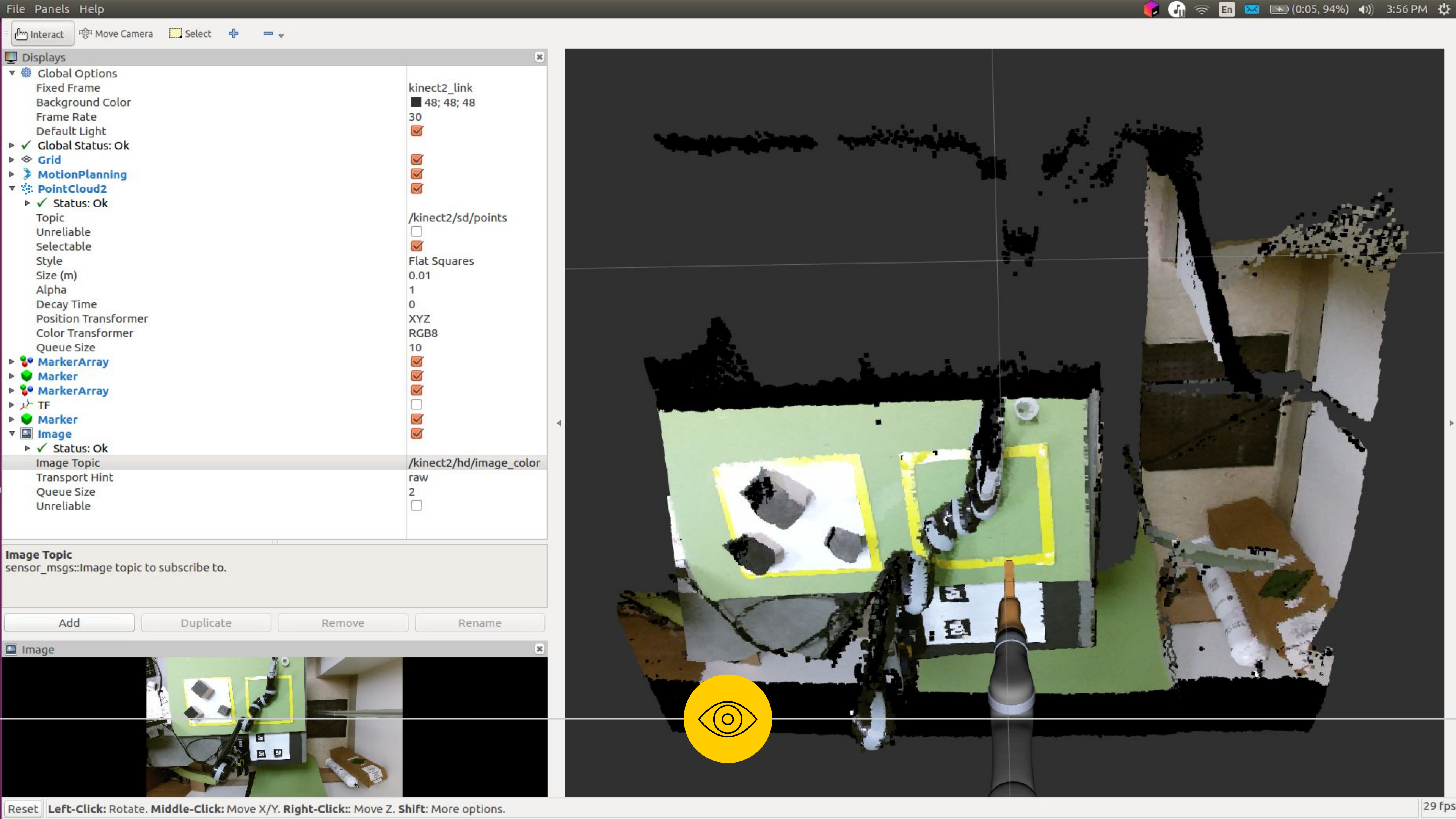
- ROS Master sends/receives
- Several nodes at once
- Whole network on your computer



ROS as a framework cont.

- `Kinect2 → /kinect2/images`
- Publishes image messages
- What are messages?





▾ Global Options	
Fixed Frame	kinect2_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	☑
▸ ✓ Global Status: Ok	
▸ Grid	☑
▸ MotionPlanning	☑
▾ PointCloud2	☑
▸ ✓ Status: Ok	
Topic	/kinect2/sd/points
Unreliable	□
Selectable	☑
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	RGB8
Queue Size	10
▸ MarkerArray	☑
▸ Marker	☑
▸ MarkerArray	☑
▸ TF	□
▸ Marker	☑
▾ Image	☑
▸ ✓ Status: Ok	
Image Topic	/kinect2/hd/image_color
Transport Hint	raw
Queue Size	2
Unreliable	□

Image Topic
sensor_msgs::Image topic to subscribe to.

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options.

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

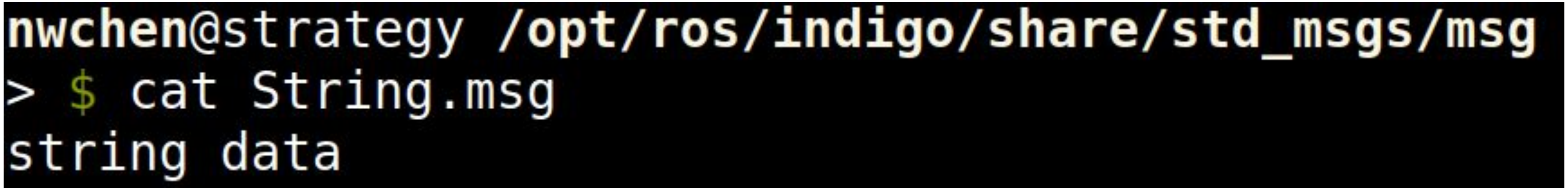
Topics

- 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



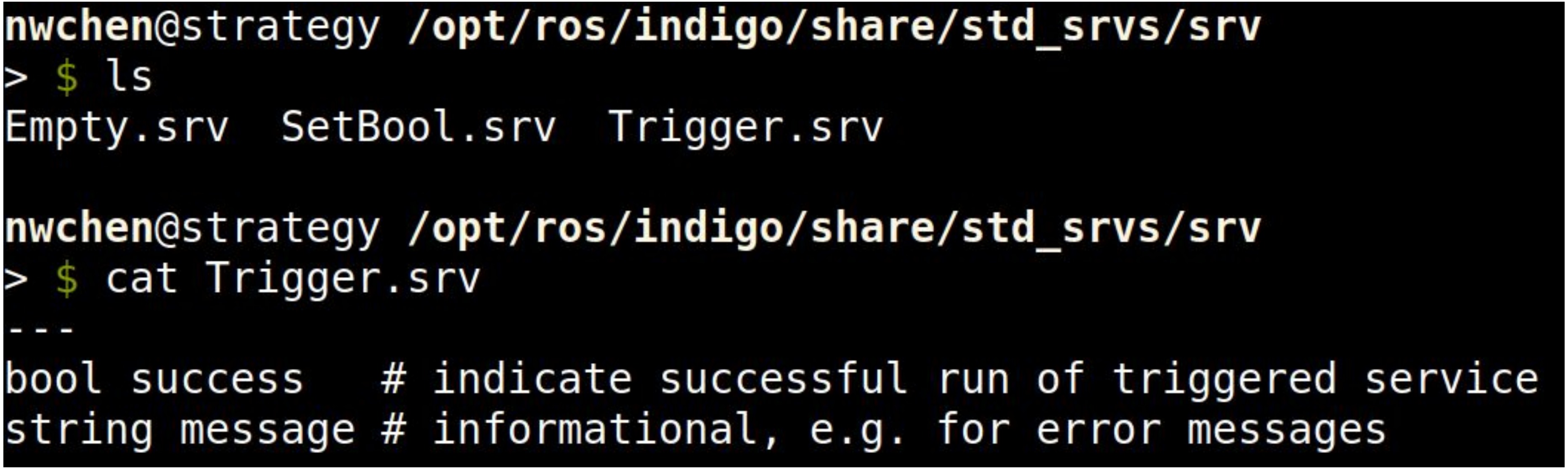
```
nwchen@strategy /opt/ros/indigo/share/std_msgs/msg
> $ cat String.msg
string data
```

Services

- A node can provide **services – synchronous** remote procedure calls
  - Request
  - Response
  -

```
Add.srv                         #Example Service
      float32 x
      float32 y
      ---     #Three dashes separate the request and response
      Float32 result
```

Can view all ROS services



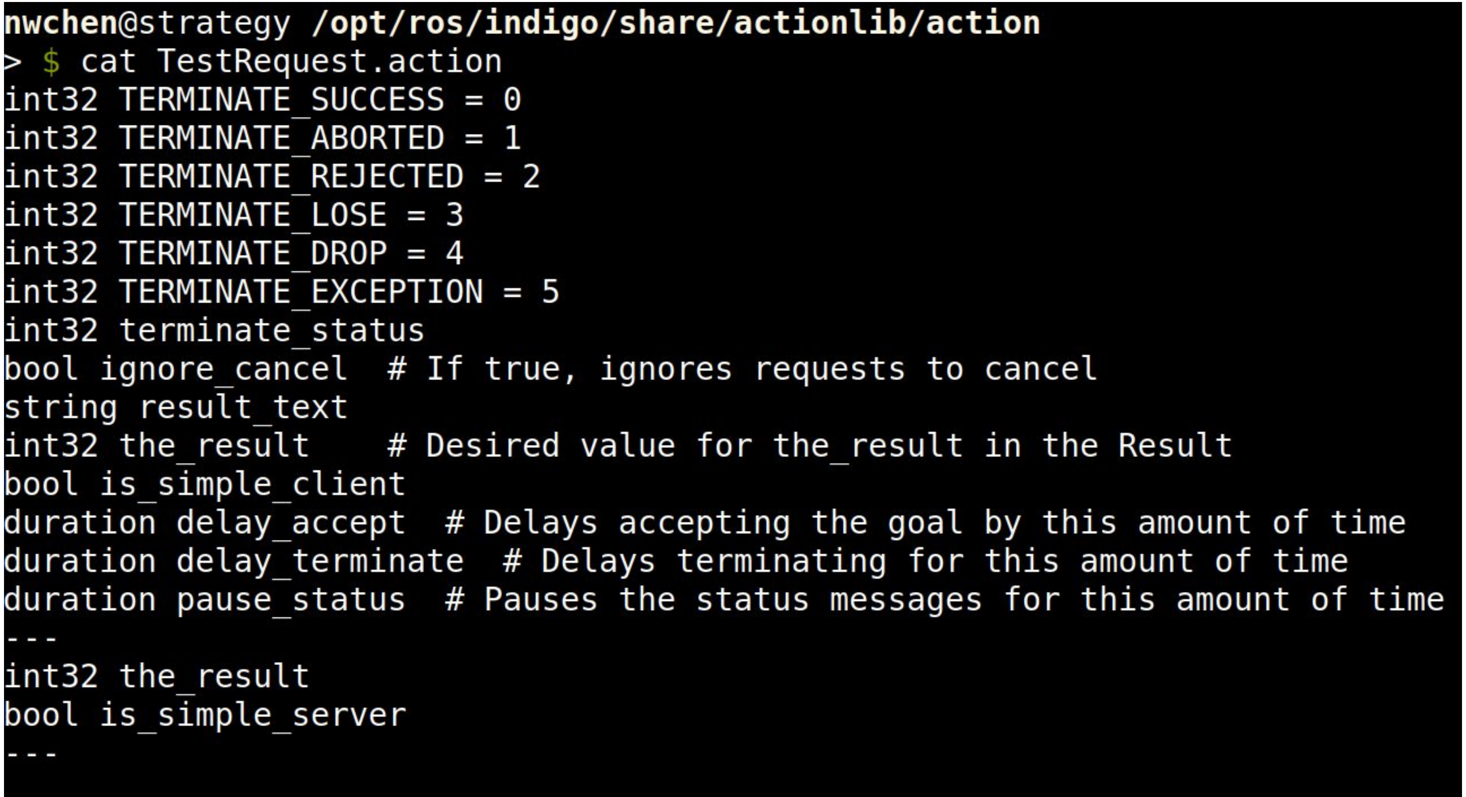
```
nwchen@strategy /opt/ros/indigo/share/std_srvs/srv
> $ ls
Empty.srv  SetBool.srv  Trigger.srv

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 (actionlib)

- Actions (asynchronous) are for long-running processes.
- They have a Goal, Result, and Feedback
  - **Navigation.action**   #Example Action

```
float32 dest_x
float32 dest_y
---
boolean success                   # Result
---
uint32 percent_complete           # Feedback
```

Can view all ROS actions



```
nwchen@strategy /opt/ros/indigo/share/actionlib/action
> $ 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
int32 terminate_status
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
---
```

Packages

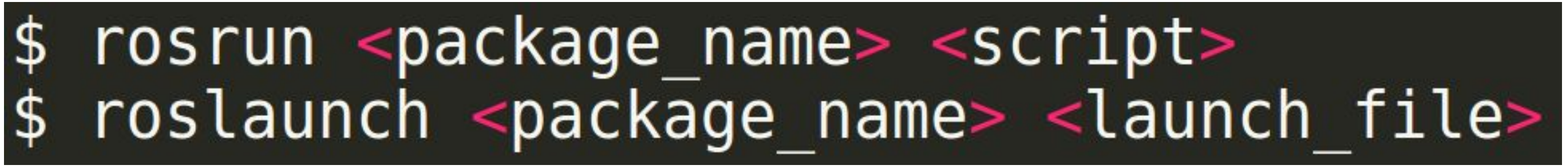
- ROS software is organized into **packages**
  - Each package contains some combination of code, data, and documentation

package_name/

- **package.xml** ← describes the package and its dependencies
- **CMakeLists.txt** ← Finds other required packages and messages/services/actions
- **src/** ← C++ source code for your node (includes go in **include/** folder)
- **scripts/** ← Python scripts for your node
- **msg/** ← ROS messages defined for your node (for topics)
- **srv/** ← ROS services defined for your node (for services)
- **launch/** ← The folder that contains `.launch` files for this package

Building/Running

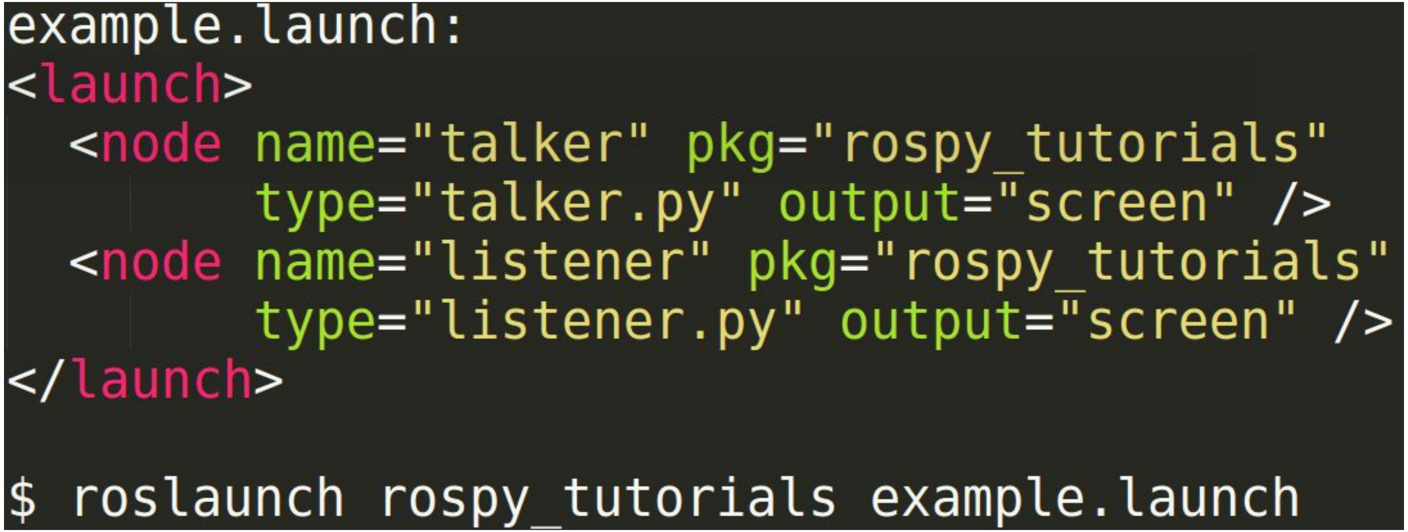
- **Catkin** is the official build system of ROS
  - Catkin combines Cmake macros and Python scripts to provide some functionality on top of Cmake's normal workflow
- Run ROS code



```
$ rosrun <package_name> <script>
$ roslaunch <package_name> <launch_file>
```

Launch Files

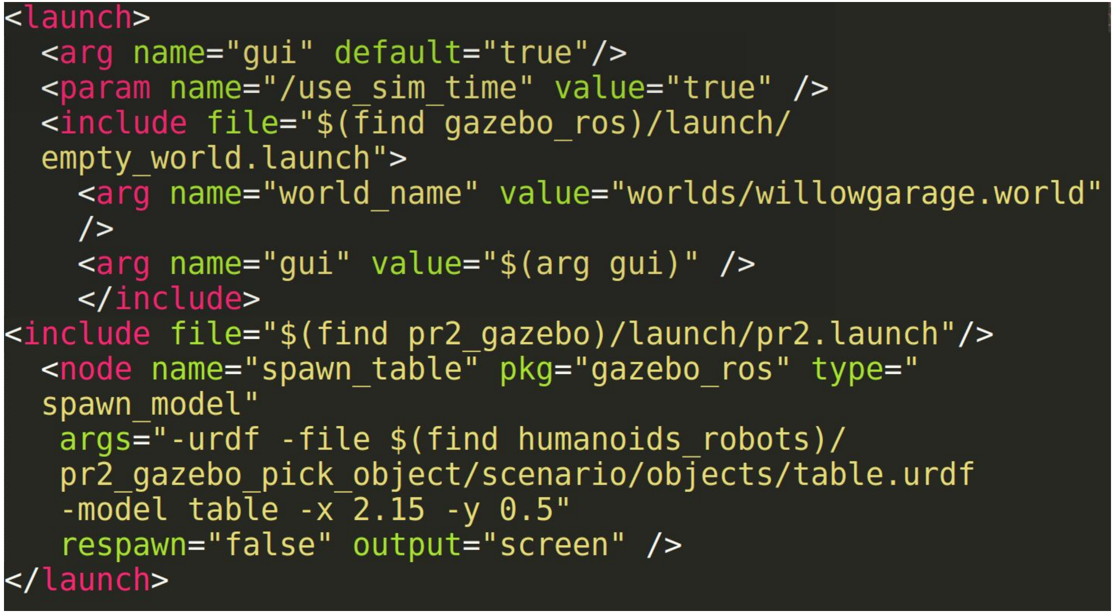
- Automate the launching of collections of ROS nodes via XML files and **roslaunch**



```
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
```

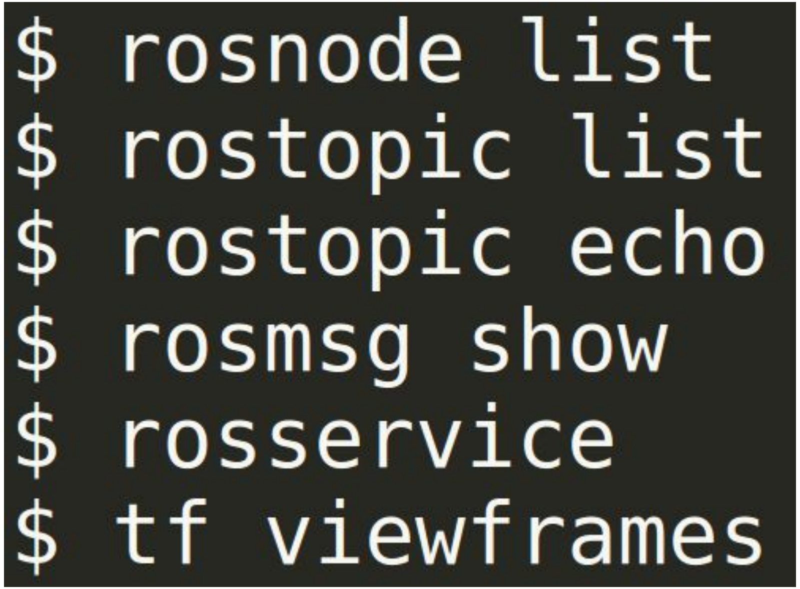
Launch Files

- You can also pass parameters via launch files



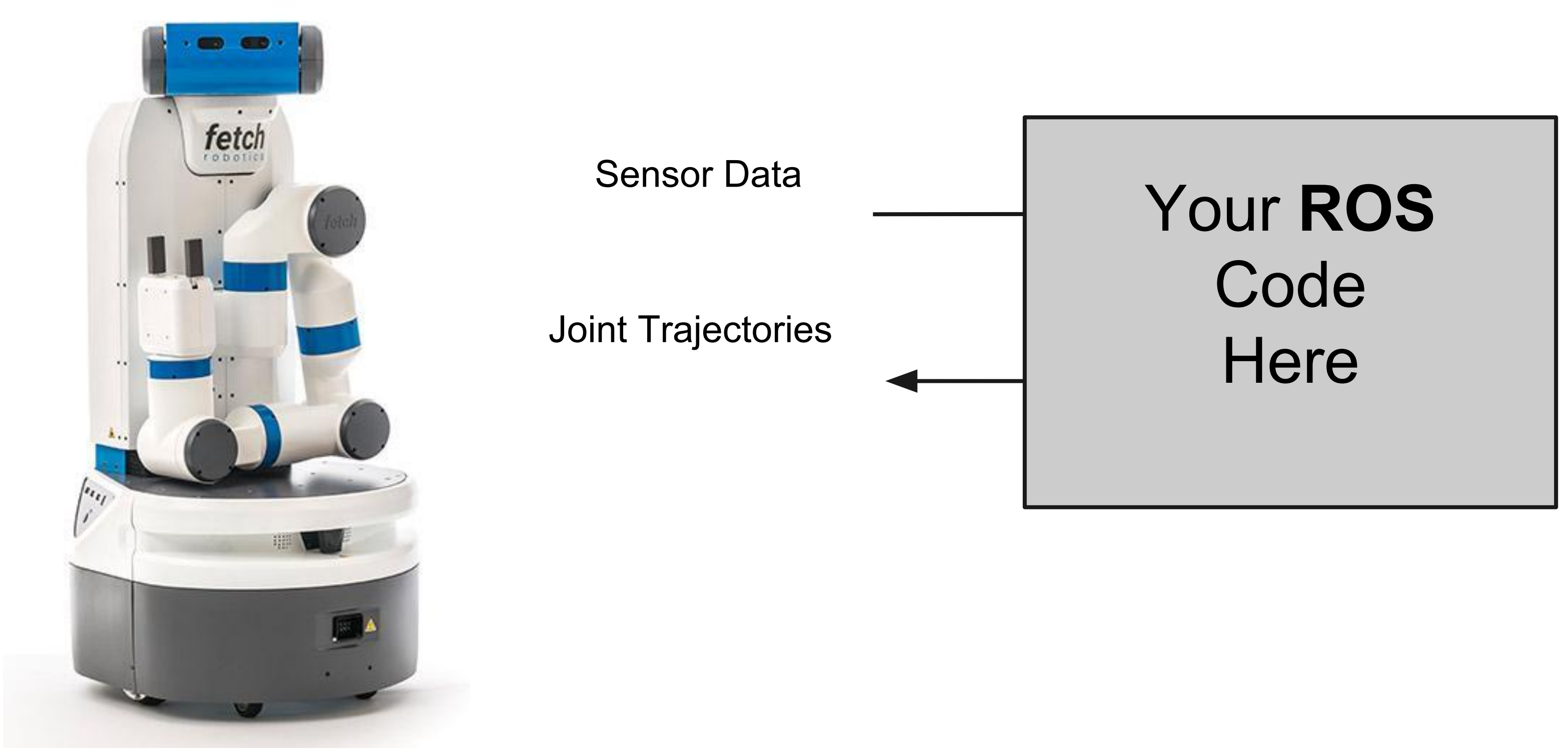
```
<launch>
  <arg name="gui" default="true"/>
  <param name="/use_sim_time" value="true" />
  <include file="$(find gazebo_ros)/launch/
  empty_world.launch">
    <arg name="world_name" value="worlds/willowgarage.world"
    />
    <arg name="gui" value="$(arg gui)" />
    </include>
<include file="$(find 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



```
$ 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

http://fetchrobotics.com/wp-content/uploads/2016/02/Fetch_spec_download_2016.pdf

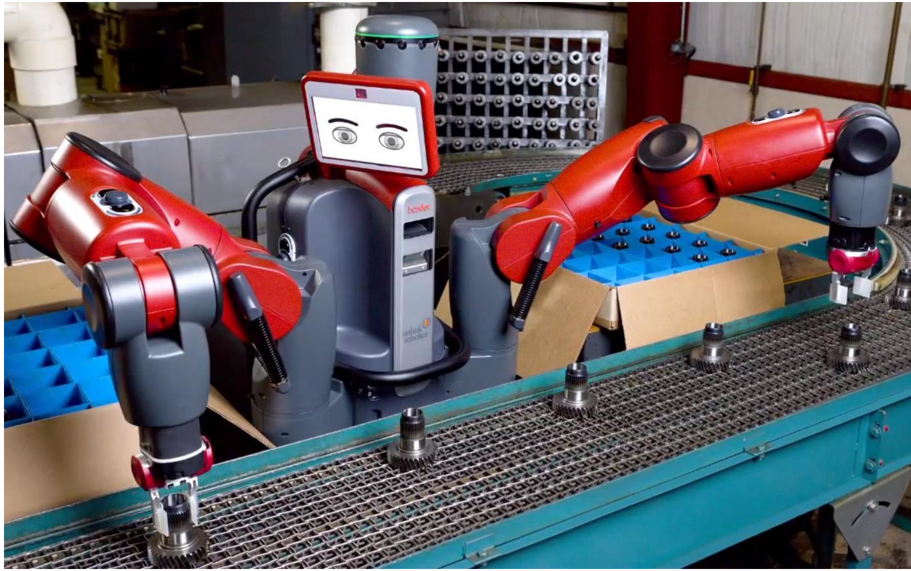
Robots available - PR2

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

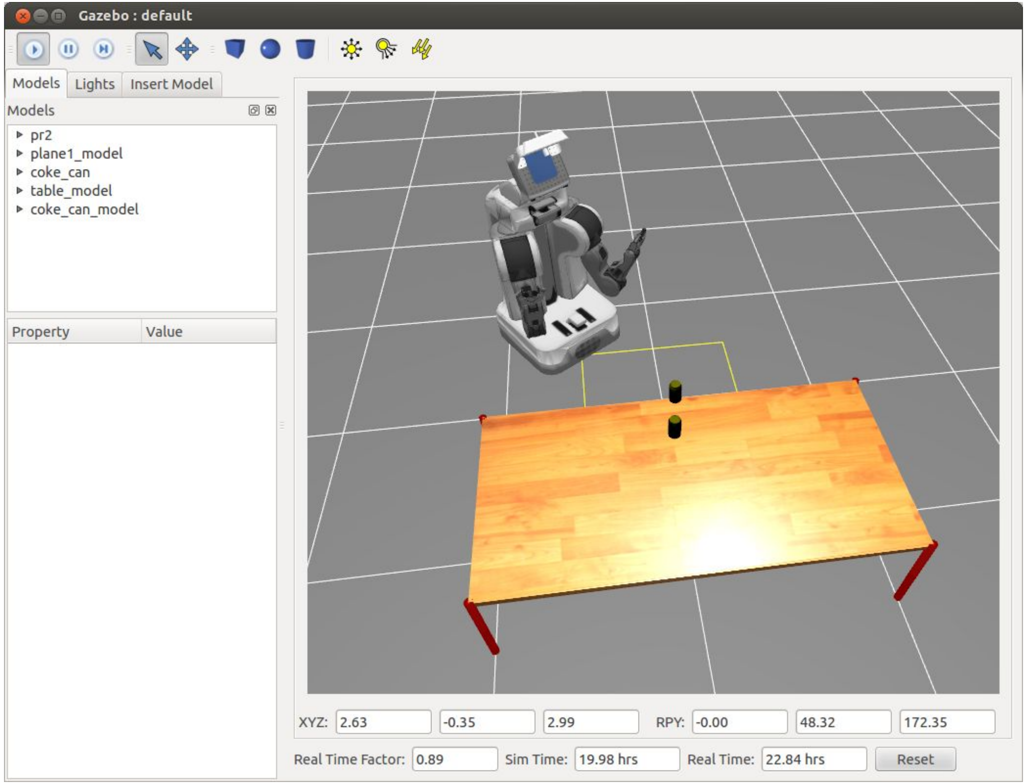
- More cost-effective
- Also has 2 arms
- Stationary base
- Sensors:
  - Sonar
  - Hand and head cameras
  - Hand rangefinders



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



- pr2
- plane1_model
- coke_can
- table_model
- coke_can_model

XYZ: 2.63 -0.35 2.99 RPY: -0.00 48.32 172.35

Real Time Factor: 0.89 Sim Time: 19.98 hrs Real Time: 22.84 hrs

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

Gazebo Simulator

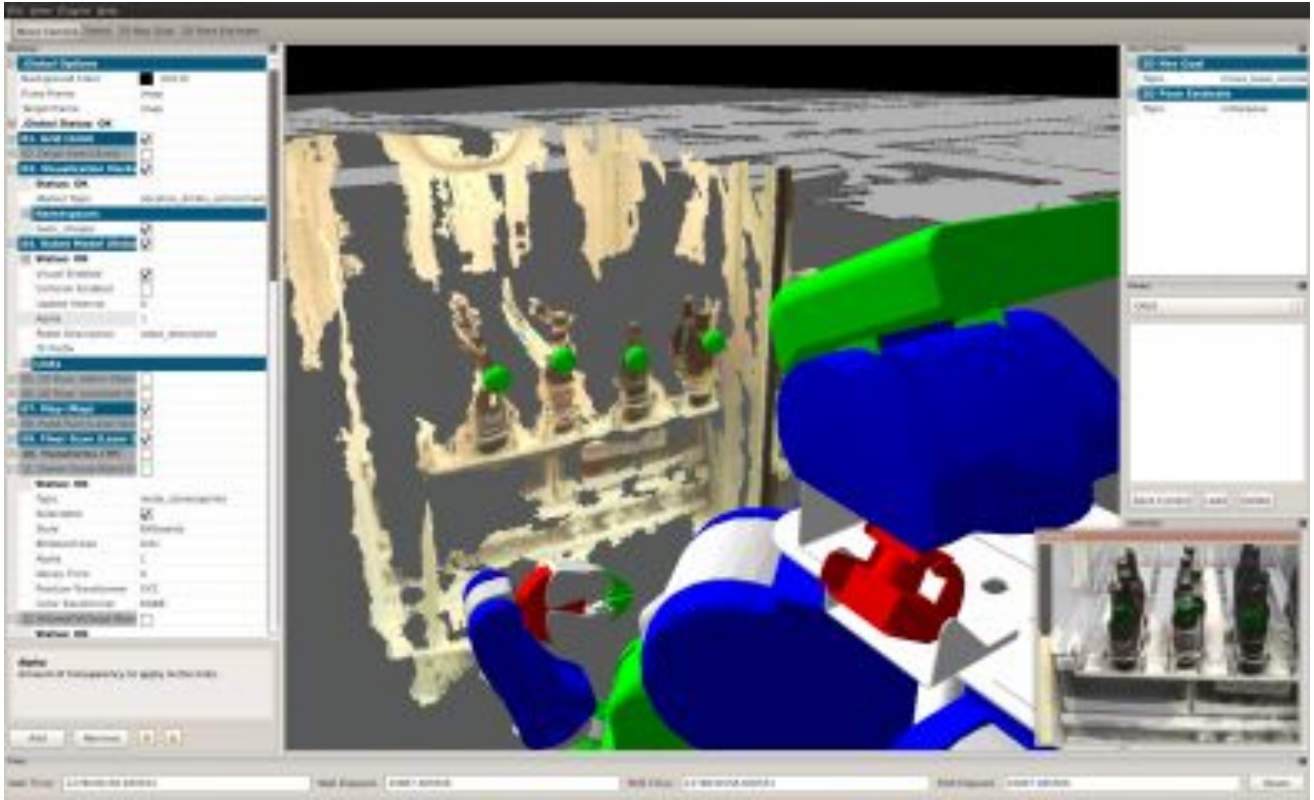
The organization that makes the robot often provides a Gazebo setup package for that robot.

For example: https://github.com/fetchrobotics/fetch_gazebo

Gazebo Simulator Demo

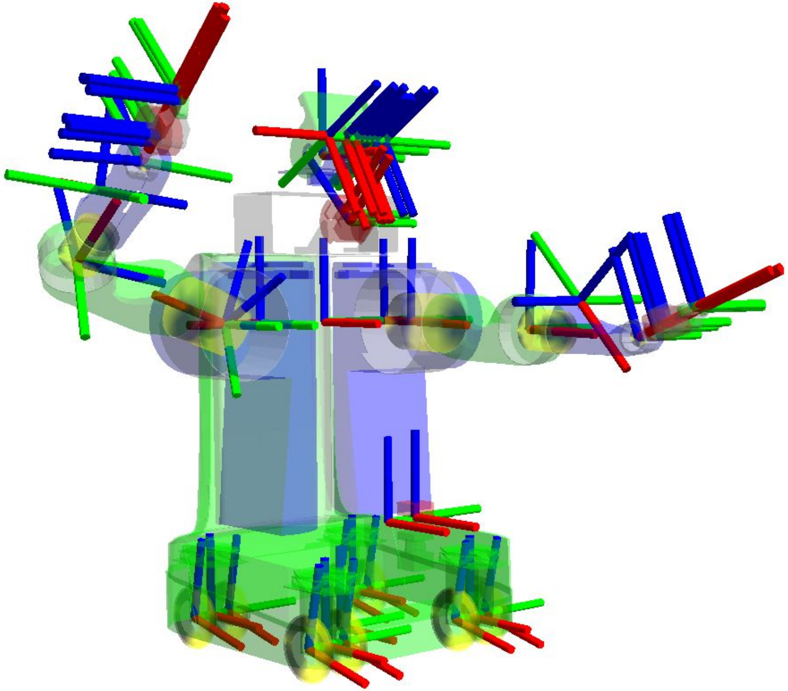
```
roslaunch fetch_gazebo playground.launch

rosrun applications keyboard_teleop.py

rostopic list | grep gazebo
```

Rviz: Robot Visualization



Moving the robot - TF

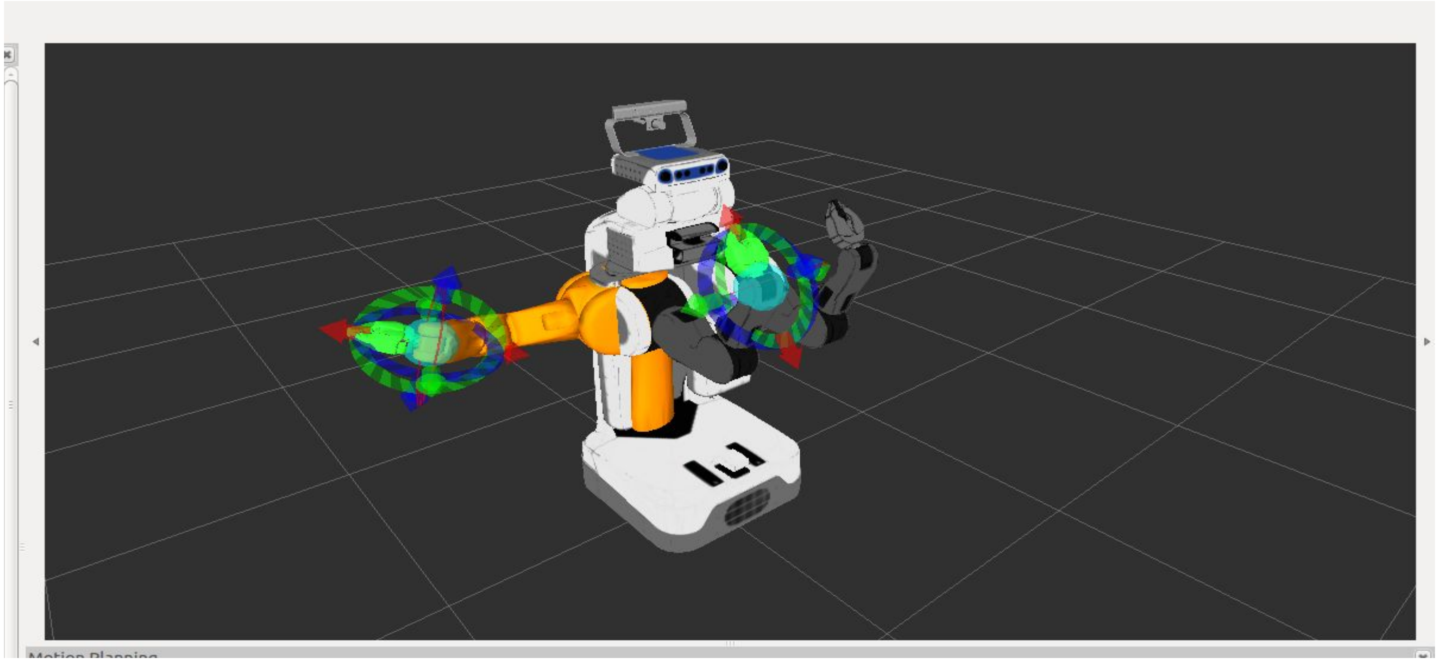
- A robotic system typically has many 3D coordinate frames that change over time.
- tf keeps track of all these frames over time.



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.

MoveIt! config

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

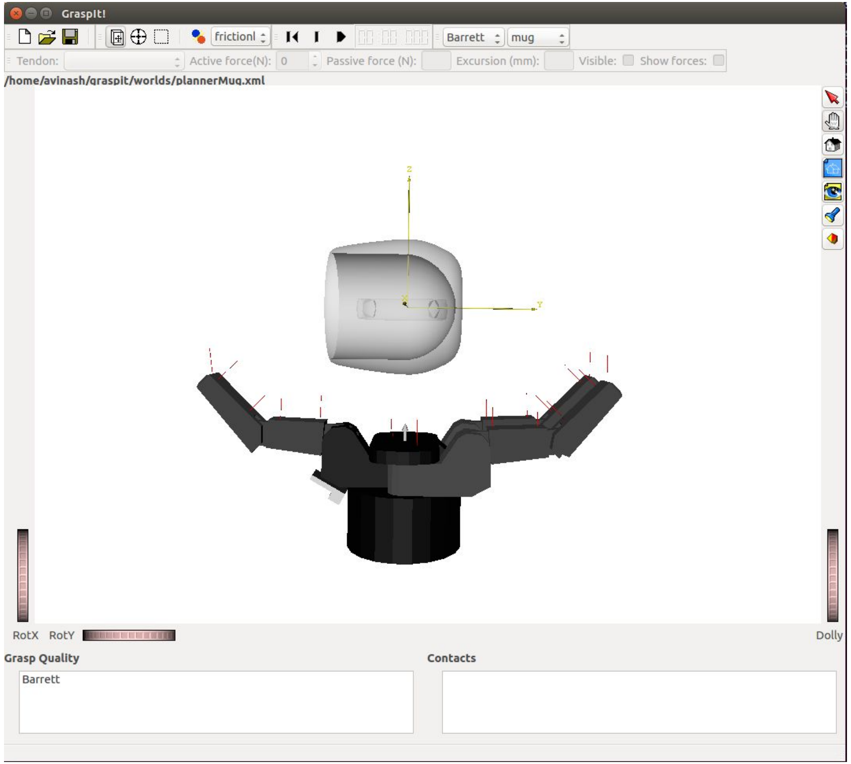
```
roslaunch fetch_moveit_config move_group.launch
```

MoveIt! Python interfaces

- moveit_commander
  - http://docs.ros.org/jade/api/moveit_commander/html/index.html
- moveit_python
  - https://github.com/mikeferguson/moveit_python

Graspit!

- Grasp planner
- Lots of robots and objects



GraspIt!

frictionl Barrett mug

Tendon: Active force(N): 0 Passive force (N): Excursion (mm): Visible: □ Show forces: □

/home/avinash/graspit/worlds/plannerMug.xml

Grasp Quality

Barrett

Contacts

Graspit! demo

```
roslaunch graspit_interface graspit_interface.launch
```

Python interface: graspit_commander
https://github.com/graspit-simulator/graspit_commander

Graspit! demo

```
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:
  - http://wiki.ros.org/ROS/Tutorials
- Reference class slides/codes provided
- Google it
- 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