Algorithmic Human-Robot Interaction

Hands-on with ROS: Quadcopter Control

CSCI 7000

Prof. Brad Hayes

Computer Science Department

University of Colorado Boulder

Literature Review (Due 4/2)

What other work exists in this space?

How do people solve the problem you’re solving?

(If nobody is attacking the same problem, what’s the closest thing to it?)

Where does your approach fit in the landscape of this existing work?

What technical gap are you addressing that others don’t? How do they fall short?

Looking Ahead

3/26	**Tuesday:**	**Spring Break**
3/28	**Thursday:**	**Spring Break**
4/2	**Tuesday:**	**ROS, Computer Vision and Robot Control**
4/4	**Thursday:**	**HRI 2019 Papers, Evaluation Workshop**
4/9	**Tuesday:**	**Explainable AI and In-progress Project Presentations**
4/11	**Thursday:**	**Explainable AI and XAI Papers**
4/16	**Tuesday:**	**(Inverse) Reinforcement Learning**
4/18	**Thursday:**	**(Inverse) Reinforcement Learning and RL Papers**
4/23	**Tuesday:**	**Guest Lecture – Dr. Alessandro Roncone**

…

Papers for Thursday:
HRI 2019

Transfer depends on Acquisition: Analyzing Manipulation Strategies for Robotic Feeding by Gallenberger et al.

Pro: Shivendra Agrawal
Con: Karthik Palavalli

Balanced Information Gathering and Goal-Oriented Actions in Shared Autonomy by Brooks et al.

Pro: Matthew Luebbers
Con:

Homework Due Next Thursday

- May 2nd is our last class
- At most 8 paper presentations left!
- **Each project group should nominate two papers (relevant to their project) to discuss in class.**
  - **Papers should be submitted via Moodle**

Final Project Due 5/7 AoE

Format: AAAI Author Kit
https://www.aaai.org/Publications/Templates/AuthorKit18.zip

Deliverables:

- 6-8 page research article about your work
  - This should read like a research paper, not a typical class project report!
- All code and data required for replication of results

Demo Prerequisites

Parrot AR SDK	::	Network Protocols
Bebop Autonomy Library	::	ROS Wrapper
cv_bridge	::	OpenCV ⇔ Image Msgs
dlib	::	Powerful Vision Library

Installing Prerequisites

Install the base SDK

> sudo apt-get install ros-kinetic-parrot-arsdk

Install some useful computer vision libraries

> pip install dlib

> pip install cv_bridge

Install the ROS Wrapper from https://github.com/AutonomyLab/bebop_autonomy
While in your ROS workspace/src directory:

> git clone https://github.com/AutonomyLab/bebop_autonomy.git

Compile the bebop_autonomy package from your catkin workspace root (it might take a few tries):

> catkin_make

ROS: Robot Operating System

Available at http://www.ros.org/

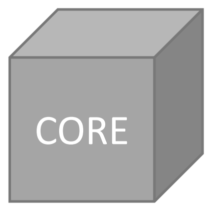
- Current Version on Lab Machines: Kinetic Kame
- Download Ubuntu 16.04 LTS image and install on VM
- http://wiki.ros.org/kinetic/Installation
- Tutorials will get you up to speed quickly!
  - http://wiki.ros.org/ROS/Tutorials

ROS is a Messaging System

ROS serves to pass information between programs

ROS includes a bunch of tools to help debug your system

ROS Components

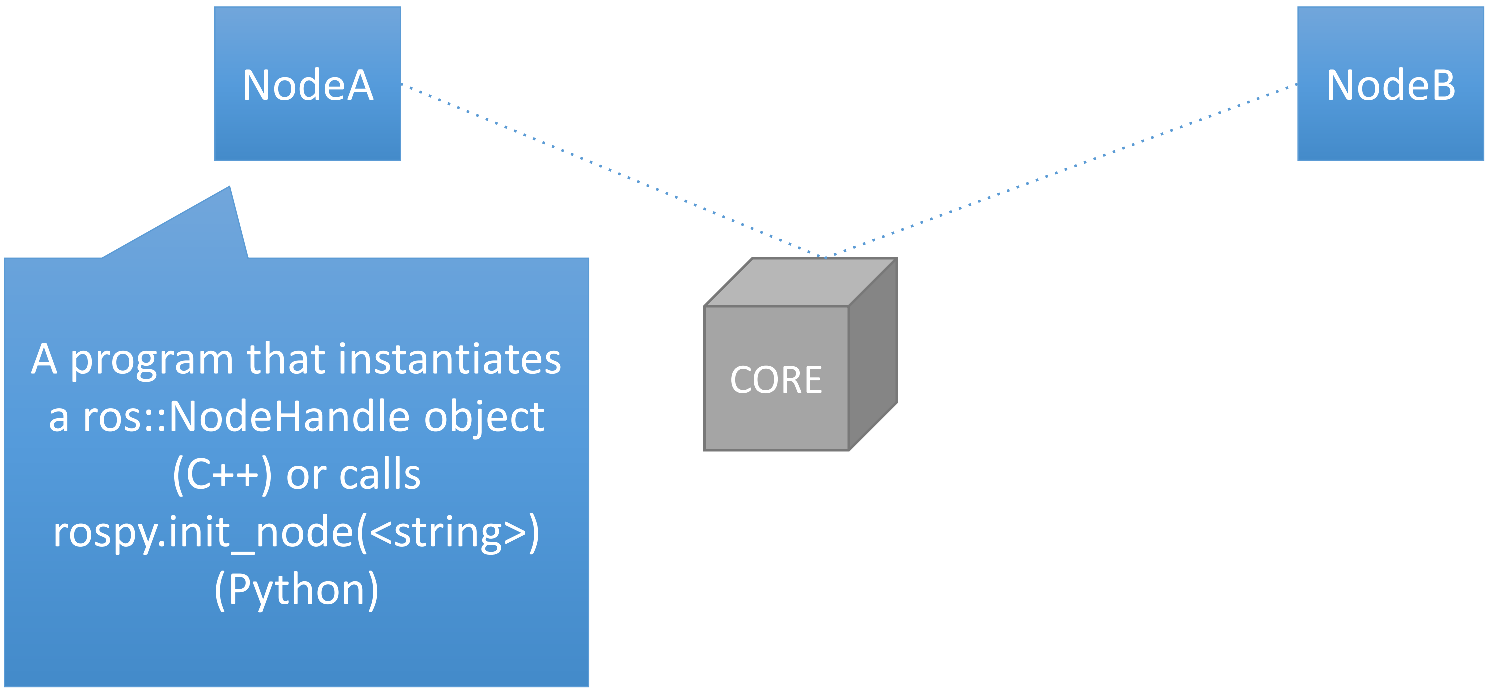


Central messaging hub.

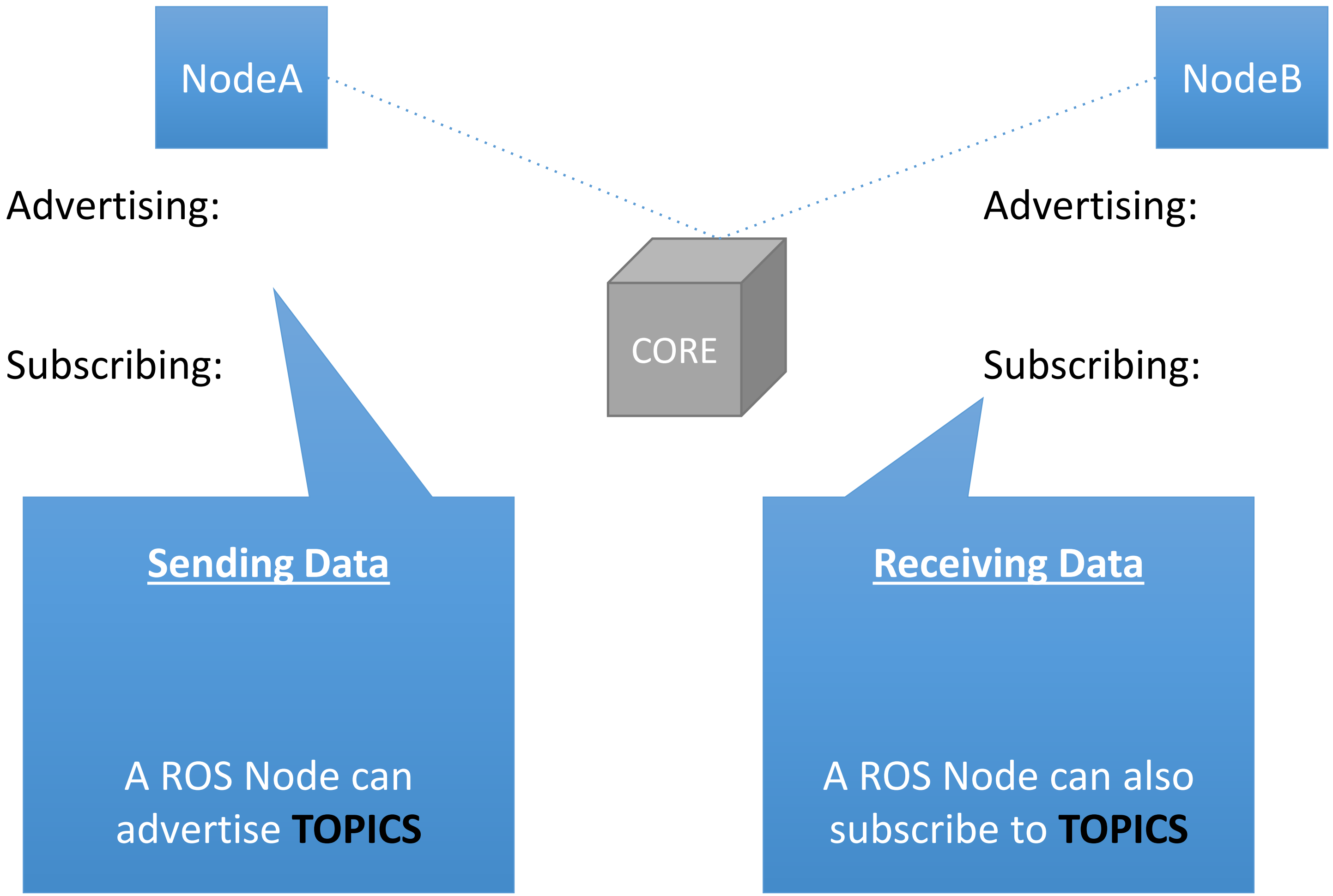
Does all of the bookkeeping.

Responsible for keeping Nodes in sync.

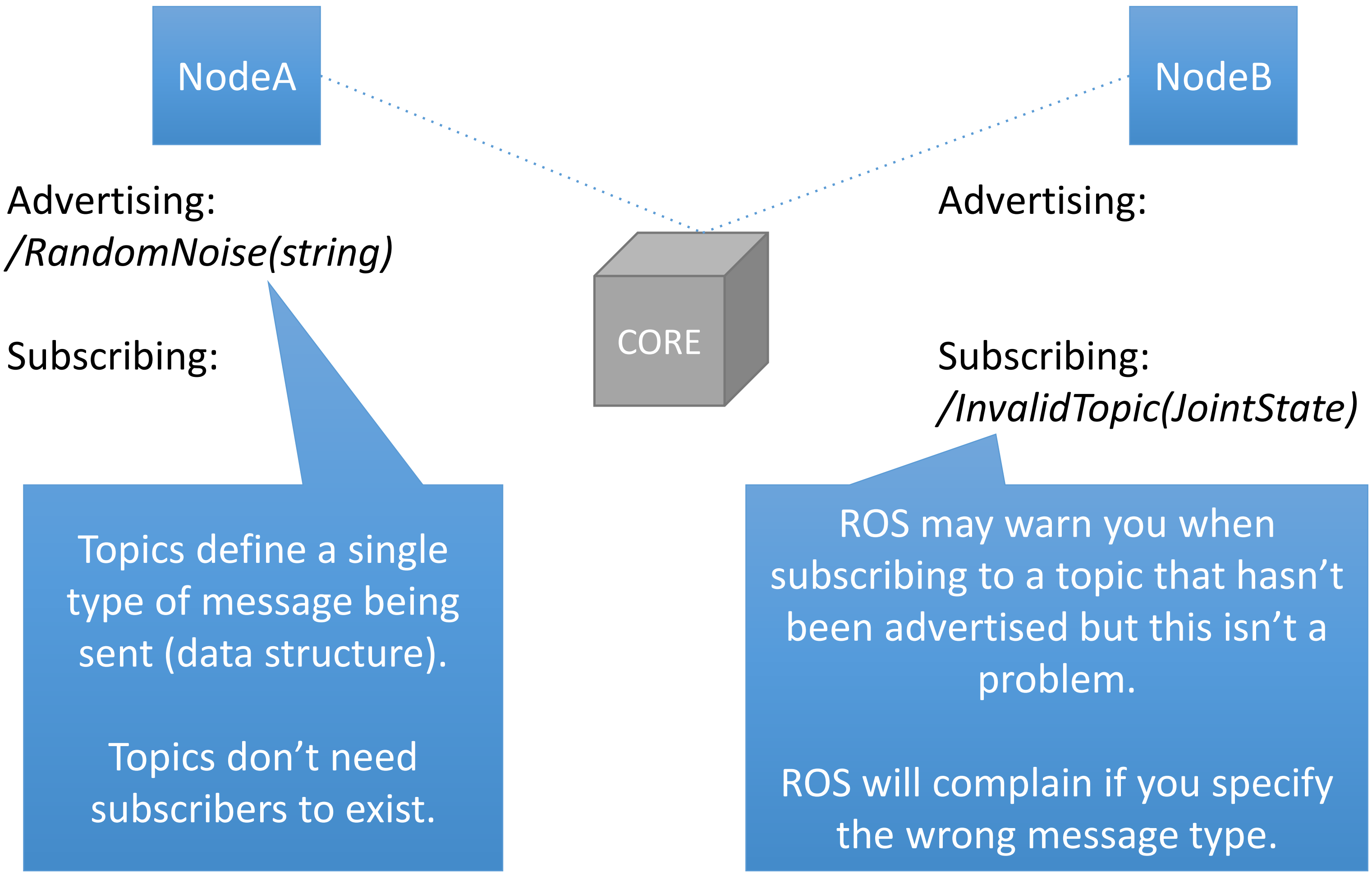
ROS Components



ROS Components



ROS Components



ROS Components

NodeA

Advertising:
/RandomNoise(string)

Subscribing:

CORE

NodeB

Advertising:

Subscribing:
/InvalidTopic(JointState)

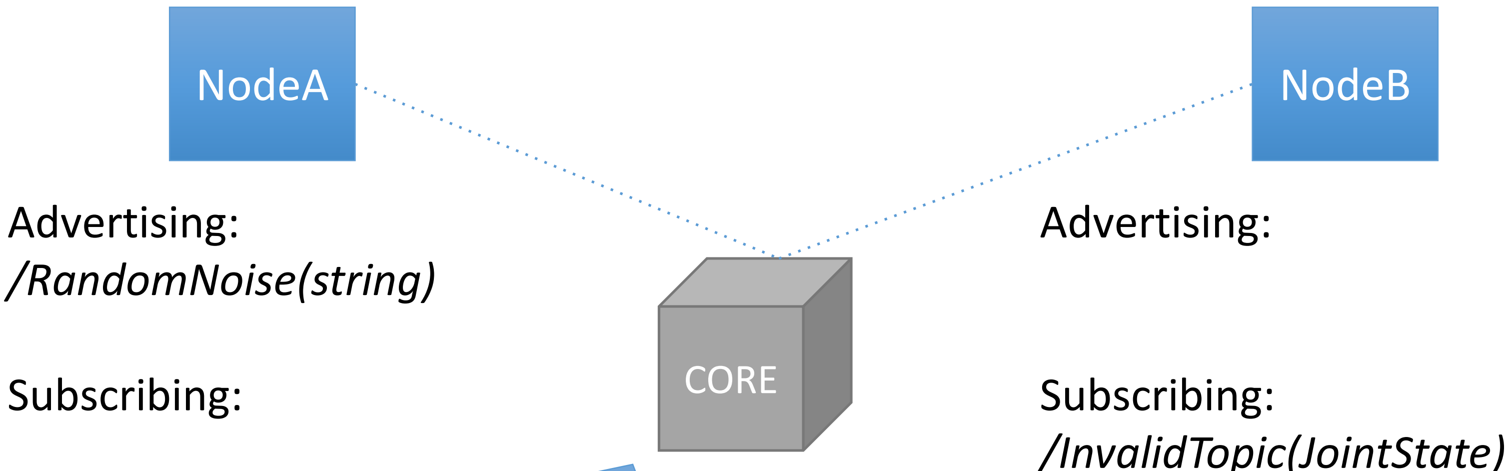
Message types are specified by .msg files.

.msg files follow a simple format and get turned into C++ and Python classes by ROS.

SimpleType.msg:

string my_data
bool isReliable

ROS Components



Message Broadcasting:

1. NodeA broadcasts a message over *RandomNoise*
2. The message goes to CORE
3. CORE checks which nodes are subscribed to *RandomNoise*
4. CORE sends the message directly to those nodes

ROS Components

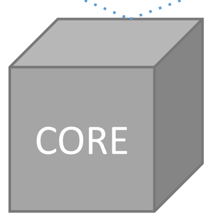
NodeA

Advertising:
/RandomNoise(string)
[Type1,Type2] /ImportantService(string,MyType)

Subscribing:

SERVICES can have arbitrarily many arguments and return values.

Treat them as regular function calls.



NodeB

Advertising:

Subscribing/ServiceProxy:
[Type1,Type2] /ImportantService(string,MyType)

You cannot utilize services that haven't been advertised.

You can wait for them to become available using waitForService(address).

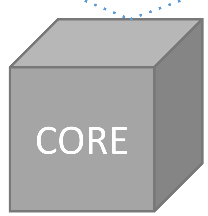
ROS Components

NodeA

NodeB

Advertising:
/RandomNoise(string)
[Type1,Type2] /ImportantService(string,MyType)

Subscribing:



Advertising:

Subscribing/ServiceProxy:
[Type1,Type2] /ImportantService(string,MyType)

MyService.srv:
int a
int b
---
int sum
int product

SERVICES are defined by .srv files.

.srv files specify arguments and return values for a service.

ROS Components

NodeA

NodeB

Advertising:
/RandomNoise(string)
[Type1,Type2] /ImportantService(string,MyType)

Subscribing:

CORE

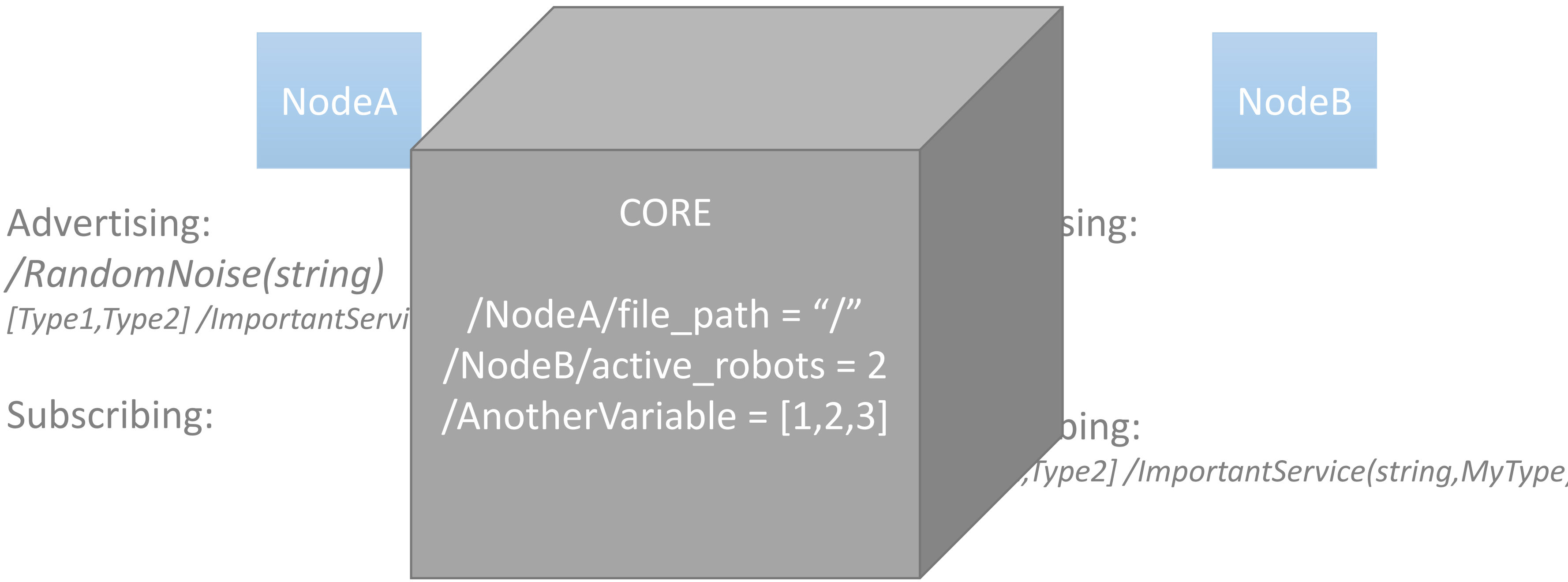
Advertising:

Subscribing:
[Type1,Type2] /ImportantService(string,MyType)

Service Calls:

1. NodeB makes a call to /ImportantService
2. The request goes to CORE
3. CORE checks which node is advertising /ImportantService
4. CORE sends the request directly to NodeA
5. NodeA executes the function it's advertising at that address
6. NodeA sends the result of the function back to CORE
7. CORE sends the result to the caller, NodeB.

ROS Components



CORE also keeps track of **PARAMETERS**.

Parameters are variables defined by nodes or launch files that are analogous to environment variables, but just for ROS.

Parameters can have scope, and use the same address naming convention as services and topics.

Running ROS Programs

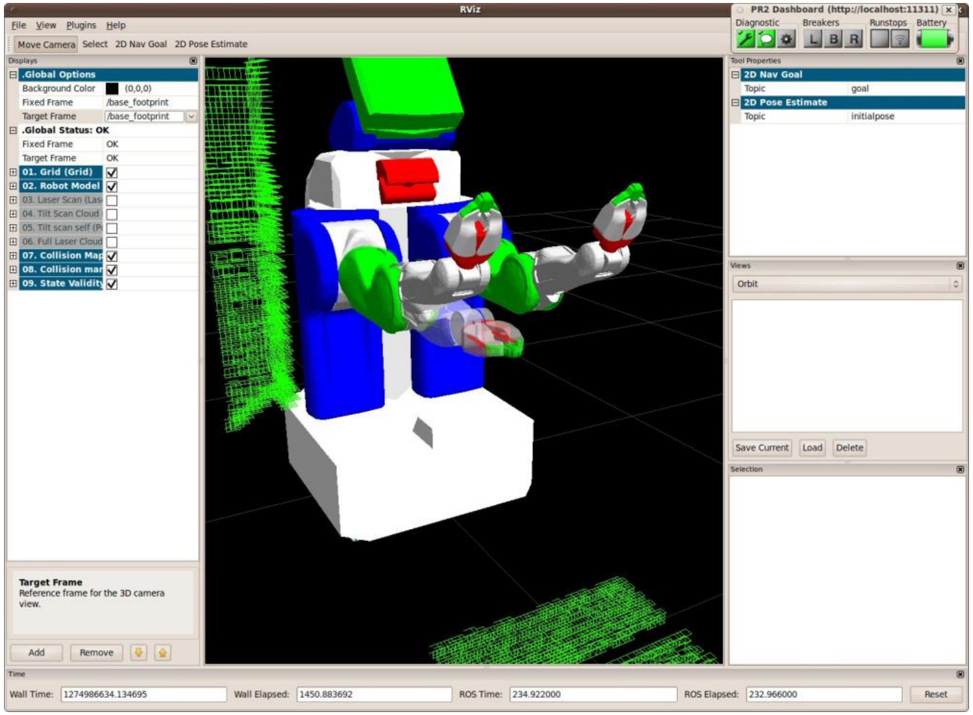
- rosrun <package> <executable>
  - Runs <executable> within <package>
- roslaunch <package> <launch file>
  - Parses the <launch file> in <package>
  - Launch files can
    - Define parameters
      - Which robot to use, Server IP addresses, etc.
    - Can specify nodes to run
      - Launch multiple nodes simultaneously
    - Specify namespaces for node groups or parameters
    - Remap parameter or service names

Looking into a ROS System

- Some powerful terminal commands:
  - rostopic and rosservice
    - Can list all advertised topics or services
    - Can listen or broadcast to topics, or call services
    - **Try "rostopic list"**
  - rosmsg, rossrv and rosparam
    - Can list all defined message types or parameters
    - Can get information about each message type or parameter value
    - **Try "rosmsg show sensor_msgs/Image"**

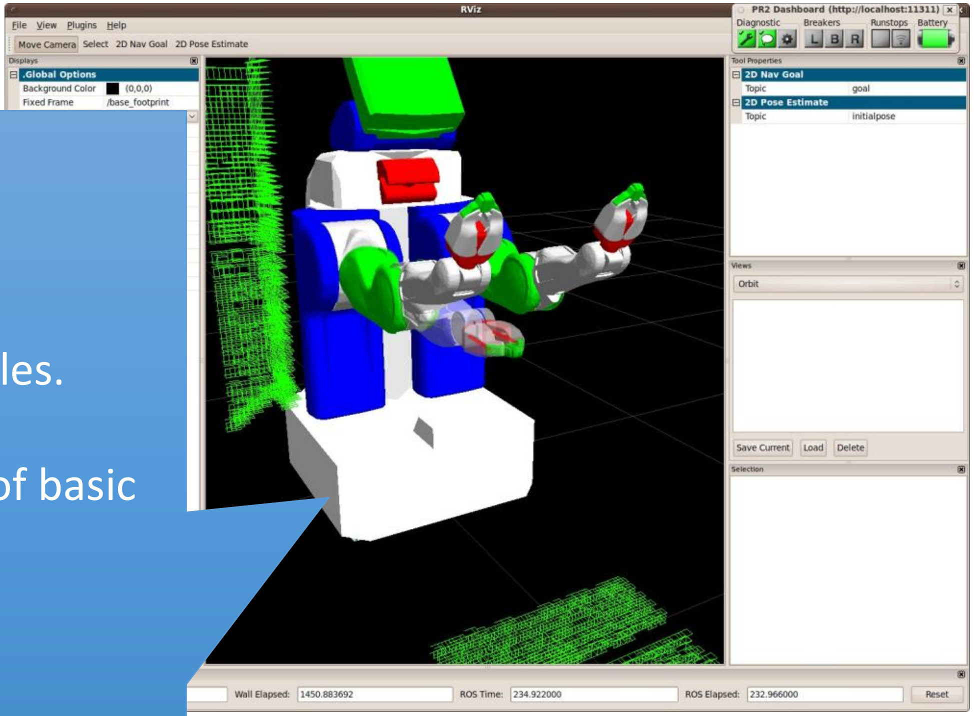
Some Important ROS Tools

- RViz
  - Full simulation environment
  - Environment Visualization
  - Many, many plugins available
  - Can make virtual objects for real system to interact with



Some Important ROS Tools

- RViz
  - Full simulation

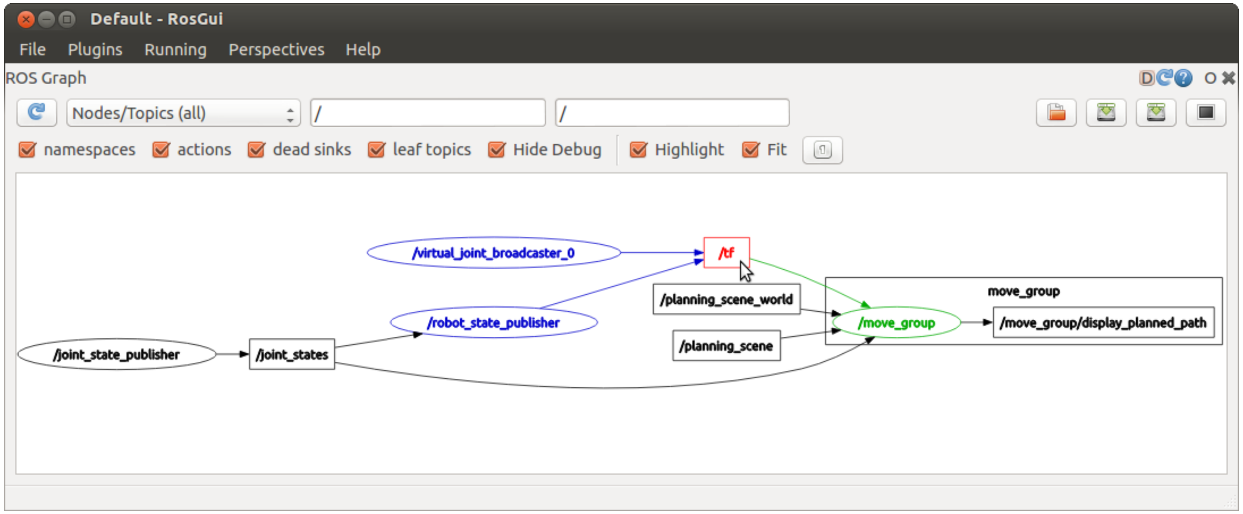


3D models are defined by URDF files.

URDF files contain XML descriptions of basic shapes and their relationships.

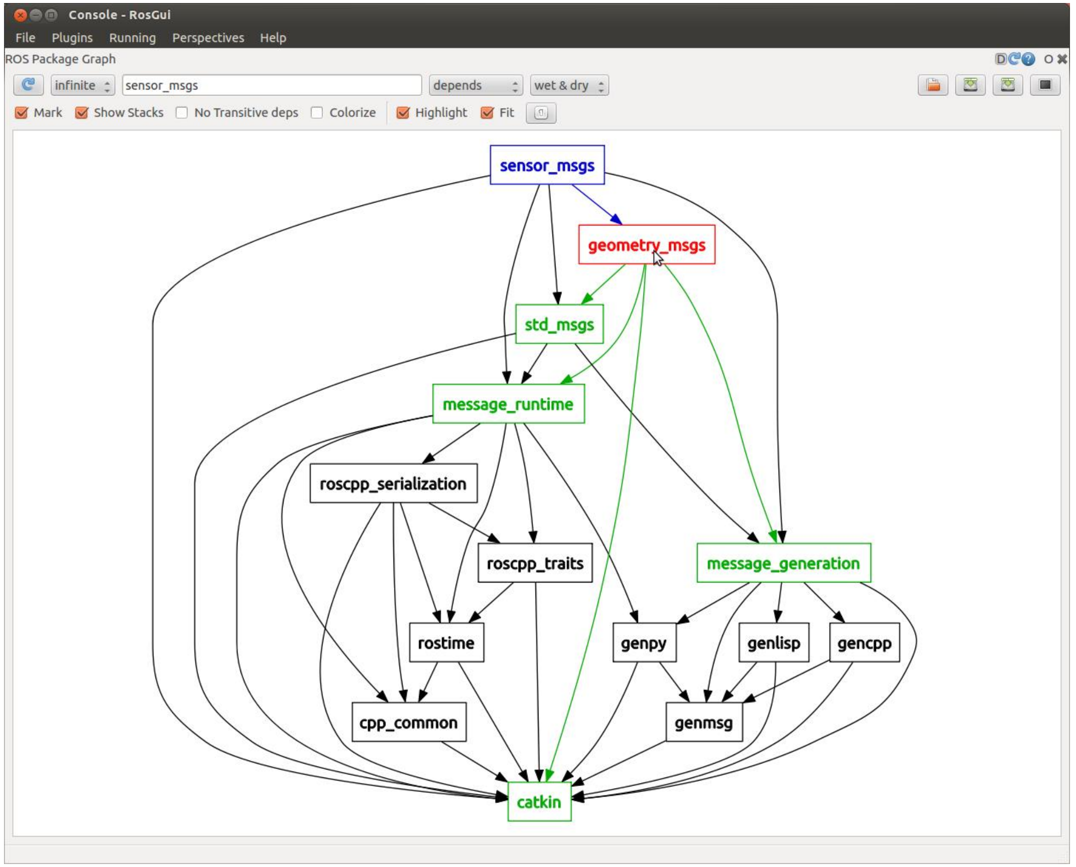
Some Important ROS Tools

- Rqt_graph
  - Network visualization tool
  - Shows relationships between nodes
    - Topics
    - Services
    - Namespaces



Some Important ROS Tools

- Rqt_dep
  - Package dependency graph visualization tool



Some Important ROS Tools

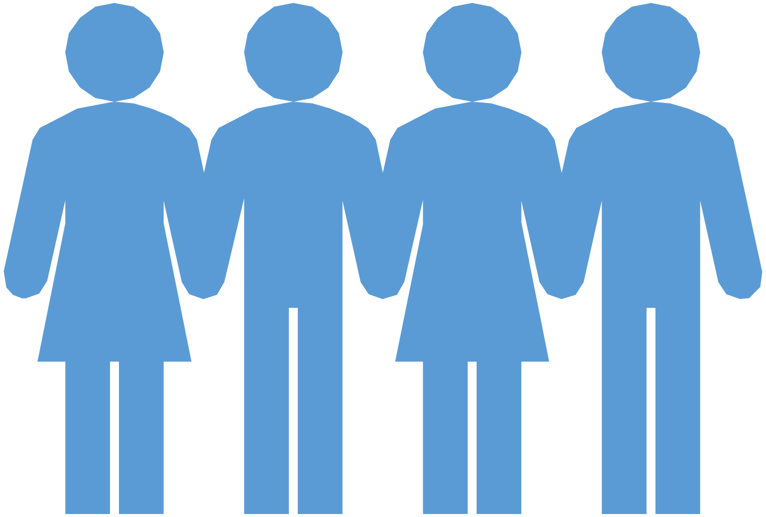
- roswtf
  - General purpose debugging tool
  - Provides checks for common sources of errors after analyzing your ROS node graph

```
Stack: ros
================================================================================
Static checks summary:

No errors or warnings
================================================================================
Beginning tests of your ROS graph. These may take awhile...
analyzing graph...
... done analyzing graph
running graph rules...
... done running graph rules

Online checks summary:

Found 1 warning(s).
Warnings are things that may be just fine, but are sometimes at fault

WARNING The following node subscriptions are unconnected:
 * /rosout:
   * /rosout
```

Team Formation

Break into 3 Groups

One member per group **must** have all prereqs installed!



Starting up ROSCore

>roscore

Central messaging handler for ROS nodes

Starting the Drone driver

Connect to your Drone's Wi-fi network first!

> roslaunch bebop_driver bebop_nodelet.launch

> rostopic list

> rosrun image_view image_view image:=/bebop/image_raw

Moving the Camera

> rostopic pub /bebop/camera_control geometry_msgs/Twist '{linear: {x: 0, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'

Publish a geometry_msgs/Twist message to **/bebop/camera_control** to "move" the camera

Angular y: [-90, 90]   -90 = Down, +90 = Forward, (0 = Mostly Forward)

Moving the Drone

http://bebop-autonomy.readthedocs.io/en/latest/piloting.html

Taking Off: Publish std_msgs/Empty to **/bebop/takeoff**

Landing: Publish std_msgs/Empty to **/bebop/land**

Moving: Publish geometry_msgs/Twist to **/bebop/cmd_vel**

linear.x:	**Forward (+) / Backward (-)**
linear.y:	**Left (+) / Right (-)**
linear.z:	**Up (+) / Down (-)**
angular.z:	**Counter-Clockwise (+) / Clockwise (-)**

```
roll_degree       = linear.y  * max_tilt_angle
pitch_degree      = linear.x  * max_tilt_angle
ver_vel_m_per_s   = linear.z  * max_vert_speed
rot_vel_deg_per_s = angular.z * max_rot_speed
```

Other Useful Tools

pdb (Python Debugger)

- Add “pdb.set_trace()” inside your program to trigger a breakpoint
- Run your Python script with
  python –m pdb my_script.py
  to get access to the PDB post-mortem prompt

Useful pdb prompt information:

‘h’: Help

‘l’: Show code around current breakpoint

‘u’: Go up one stack frame

‘d’: Go down one stack frame

‘bt’ / ‘w’: Backtrace – shows stack frames

Making Your ROS Node

```
import rospy
from std_msgs.msg import Empty
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image

rospy.init_node("my node name")
```

Communication: Publishers

```
takeoff_pub = rospy.Publisher("/bebop/takeoff", Empty, queue_size=1)

land_pub = rospy.Publisher("/bebop/land", Empty, queue_size=1)

control_pub = rospy.Publisher("/bebop/cmd_vel", Twist, queue_size=1)
```

Communication: Subscribers

```
camera_sub = rospy.Subscriber("/bebop/image_raw", Image, camera_callback)


def camera_callback(image_msg):
        pass
```

Initialization: Libraries and Globals

```python
import rospy
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
from std_msgs.msg import Empty
import cv2
import cv_bridge
import dlib
import time

FLIGHT_TIME = 20 # seconds
FACE_REC_INTERVAL = .1 # seconds
FRAME_WIDTH = 428
FRAME_HEIGHT = 240

bridge = None
face_detector = dlib.get_frontal_face_detector()
win = dlib.image_window()
last_image = None

drone_pub = None
```

Main Program and Pub/Sub Initialization

```python
def main():
    global last_image, drone_pub
    rospy.init_node("FaceTracker")
    camera_sub = rospy.Subscriber("/bebop/image_raw", Image, img_callback)
    drone_pub = rospy.Publisher("/bebop/cmd_vel", Twist, queue_size=1)
    takeoff_pub = rospy.Publisher("/bebop/takeoff", Empty, queue_size=1)
    landing_pub = rospy.Publisher("/bebop/land", Empty, queue_size=1)

    while last_image is None:
        time.sleep(0.5)

    takeoff_pub.publish(Empty())
    time.sleep(2.)

    last_call = 0
    start_time = time.time()
    while not rospy.is_shutdown() and time.time() - start_time < FLIGHT_TIME:
        if time.time() - last_call > FACE_REC_INTERVAL:
            last_call = time.time()
            face_position = find_faces(last_image)
            print face_position

            adjust_drone_pos(face_position)

    landing_pub.publish(Empty())

    #landing_pub.publish(Empty())
    print("Shutdown.")

    cv2.destroyAllWindows()


if __name__ == '__main__':
    bridge = cv_bridge.CvBridge()
    main()
```

```
def img_callback(img_msg):
    global bridge, last_image
    unscaled_cv_image = bridge.imgmsg_to_cv2(img_msg, "mono8")
    cv_image = cv2.resize(unscaled_cv_image, None, fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)
    last_image = cv_image
    # print last_image.shape
```

Image Callback

```
def find_faces(cv_image):
    global face_detector, win, last_call

    face_position = [None, None]
    # frame = skimage.io.imread('best_group.jpg') # Offline mode

    faces = face_detector(cv_image, 1)

    print("Detections: {}".format(len(faces)))
    for i, d in enumerate(faces):
        print("Face {}: Left: {}, Top: {}, Right: {}, Bottom: {}".format(i, d.left(), d.top(), d.right(), d.bottom()))
        if i == 0: face_position = (.5 * (d.right() + d.left()) / FRAME_WIDTH, .5 * (d.bottom() + d.top()) / FRAME_HEIGHT)

    rects = dlib.rectangles()
    rects.extend([d for d in faces])
    win.clear_overlay()
    win.set_image(cv_image)
    win.add_overlay(rects)

    #cv2.imshow("camera_raw", cv_image)
    #cv2.waitKey(3)

    return face_position
```

Face Detector

```python
def adjust_drone_pos(face_pos):
    global drone_pub

    if face_pos[0] is None: return

    pos_update = Twist()

    if face_pos[0] < 0.3:
        # Turn CCW
        print "CCW"
        pass
    elif face_pos[0] > 0.7:
        # Turn CW
        print "CW"
        pass

    if face_pos[1] < 0.3:
        # Increase altitude
        print "Ascend"
        pass
    elif face_pos[1] > 0.7:
        # Reduce altitude
        print "Descend"
        pass

    drone_pub.publish(pos_update)
```

Drone Control

Remaining Time: Project Feedback