Robot Middleware

EECS 367
Intro. to Autonomous Robotics

ROB 320
Robot Operating Systems

Winter 2022

autorob.org

Robot Middleware

EECS 367
o. to Autonomous Robotics

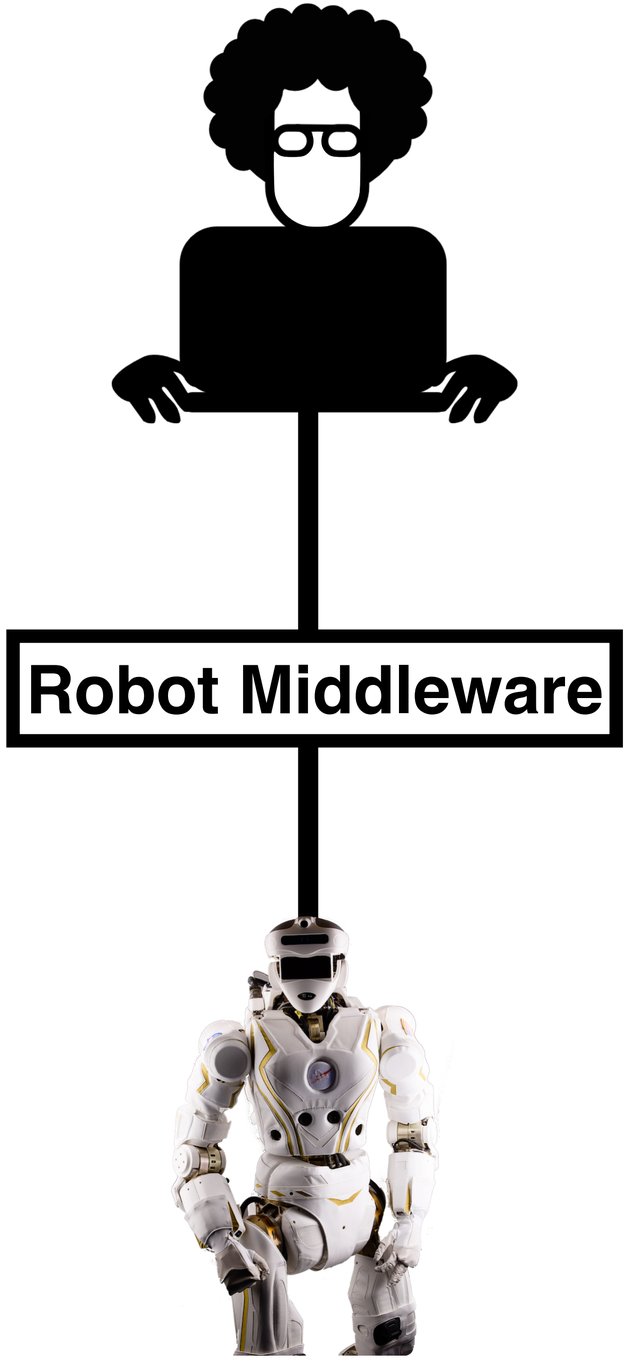
ROB 320
obot Operating Systems

Winter 2022

au


PR2

Robot Middleware



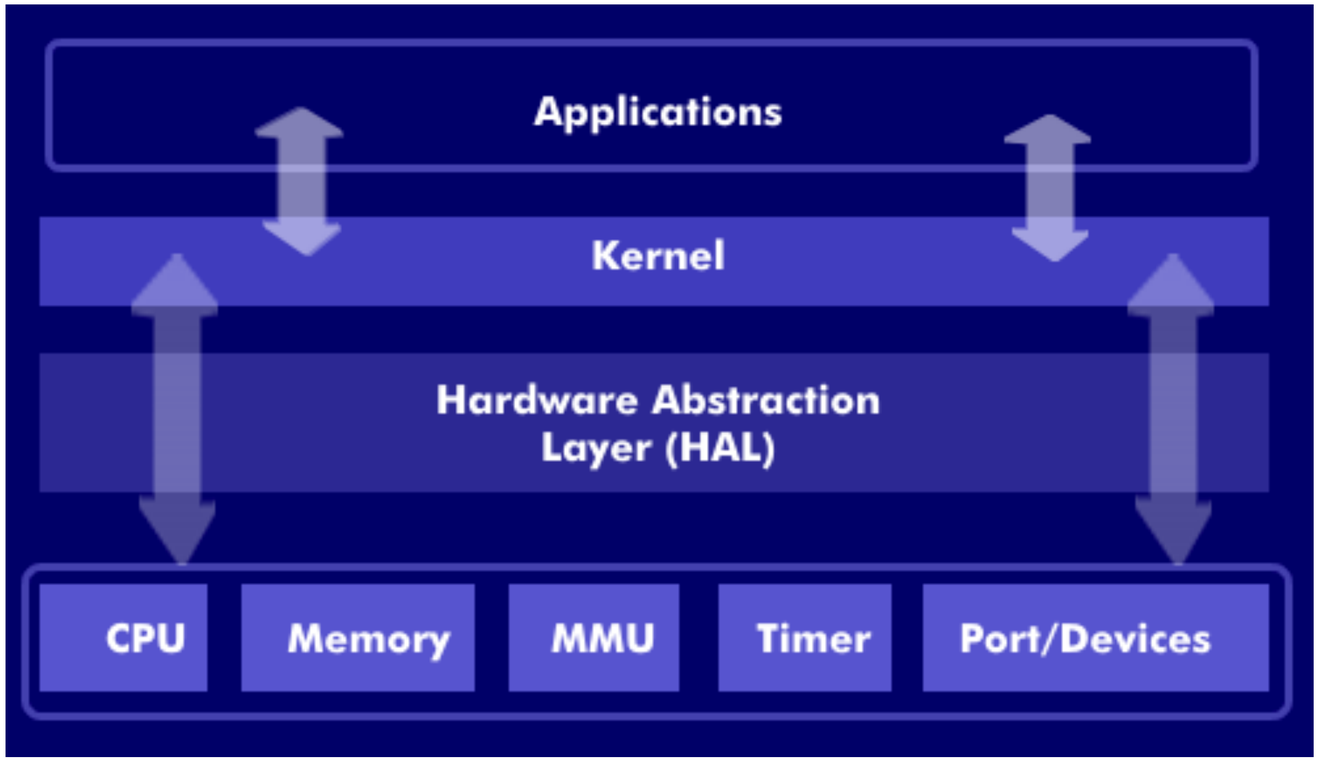
Hardware Abstraction

Robot middleware provides a hardware abstraction layer similar to a computer operating system

Programs

Abstraction

Devices



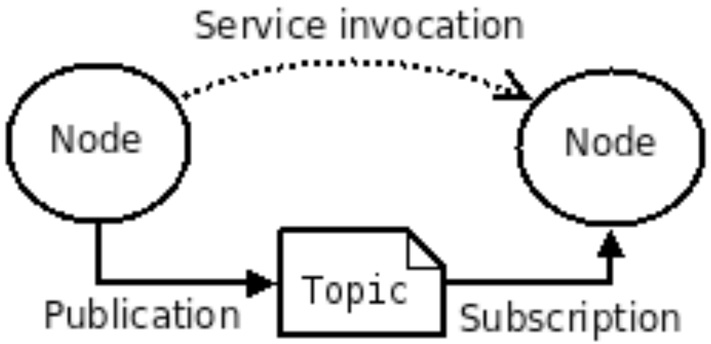
Abstraction enables “plug-and-play” interoperability and portability across compute platforms

Robot Middleware Choices

- ROS (Robot Operating System)
  - most widespread use
- LCM (Lightweight Communications and Marshalling)
  - ROB 550 and EECS 467, probably most efficient messaging
- Many other options
  - Yarp (Yet Another Robot Platform)
  - JAUS (Joint Architecture for Unmanned Systems)
  - MOOS
  - Player/Stage

ROS (ros.org)

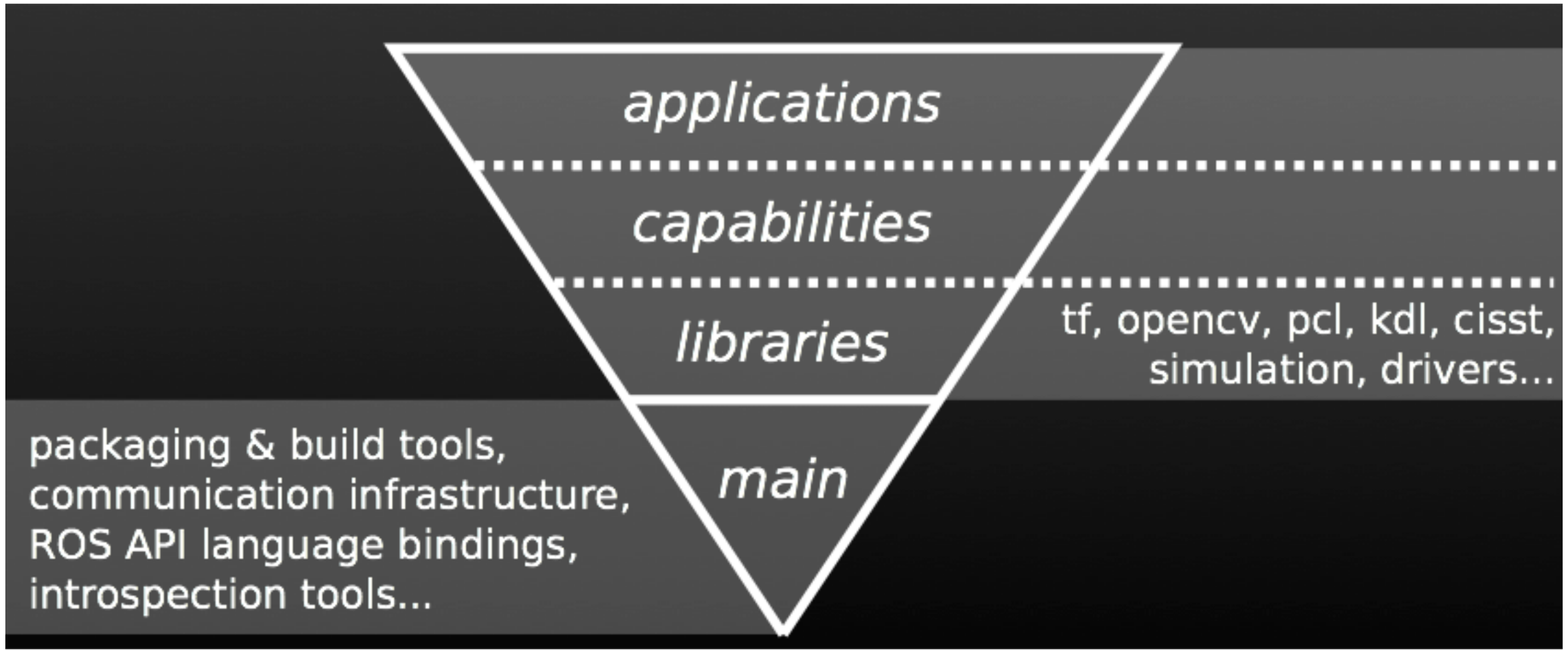
- “Robot Operating System”
  - Created by Morgan Quigley and Willow Garage
- Reduce “reinvention”, increase interoperability and reproducibility
- Peer-to-peer architecture over network
  - inter-process communication for robots
- Software functionality modularized as ROS nodes
  - Run-time system: nodes communicate over IP network
  - Packaging system: nodes organized into distributable packages



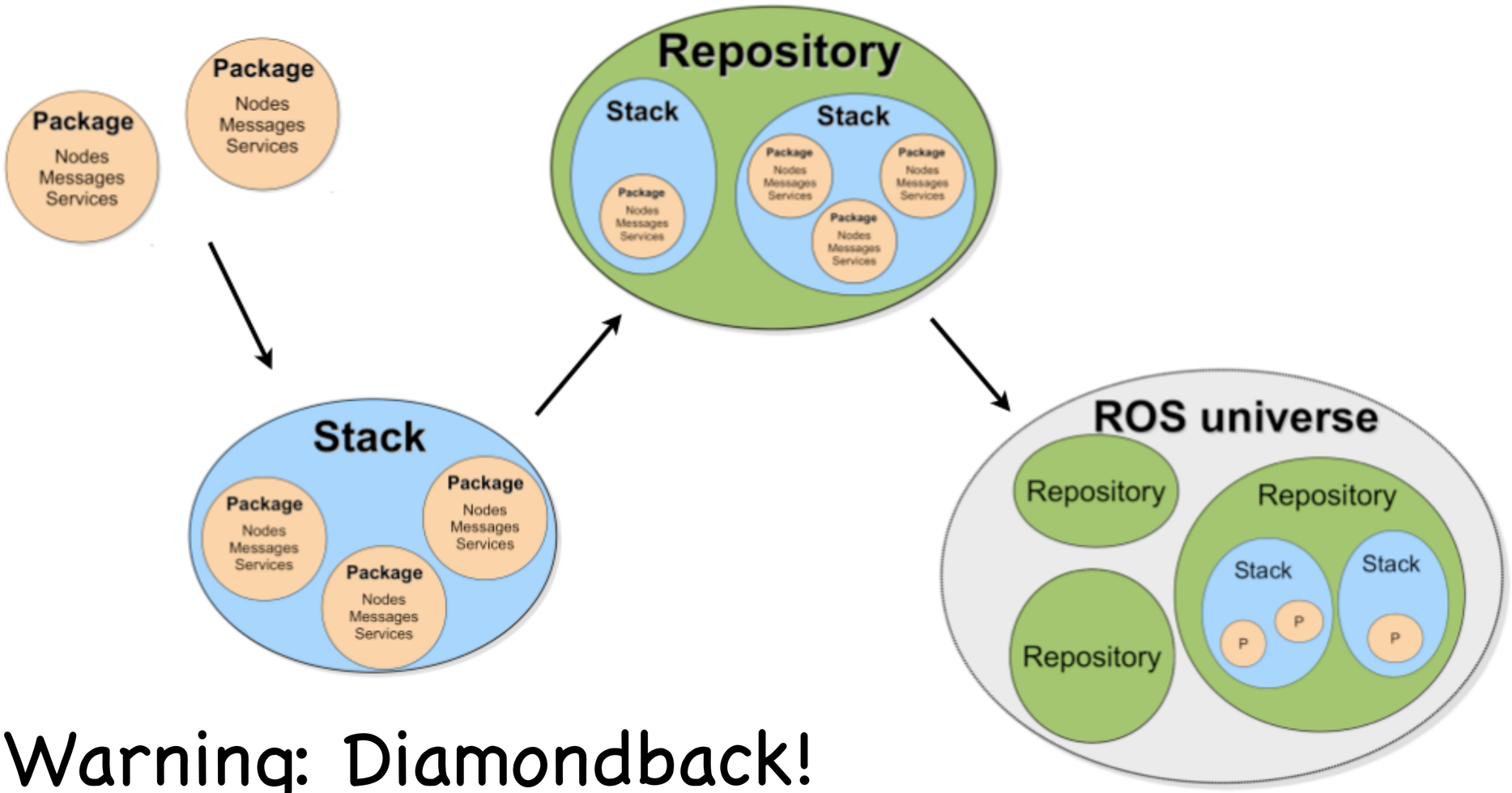
ROS (ros.org)

- WARNING! ROS is a moving target
  - ROS stability issues, adaptation is often necessary
    - These slides assume Diamondback distribution (2011)
    - Electric, Fuerte, …, Lunar, Melodic, Noetic released since
  - ad-hoc documentation (ros.org) supplemented by ROS answer board (answers.ros.org), mailing list (ros-users)

ROS as a development ecosystem

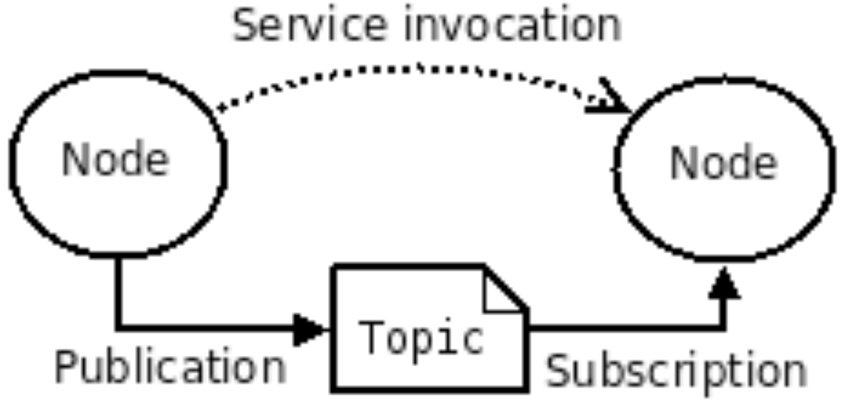


ROS Packaging System



Warning: Diamondback!

ROS Run-time

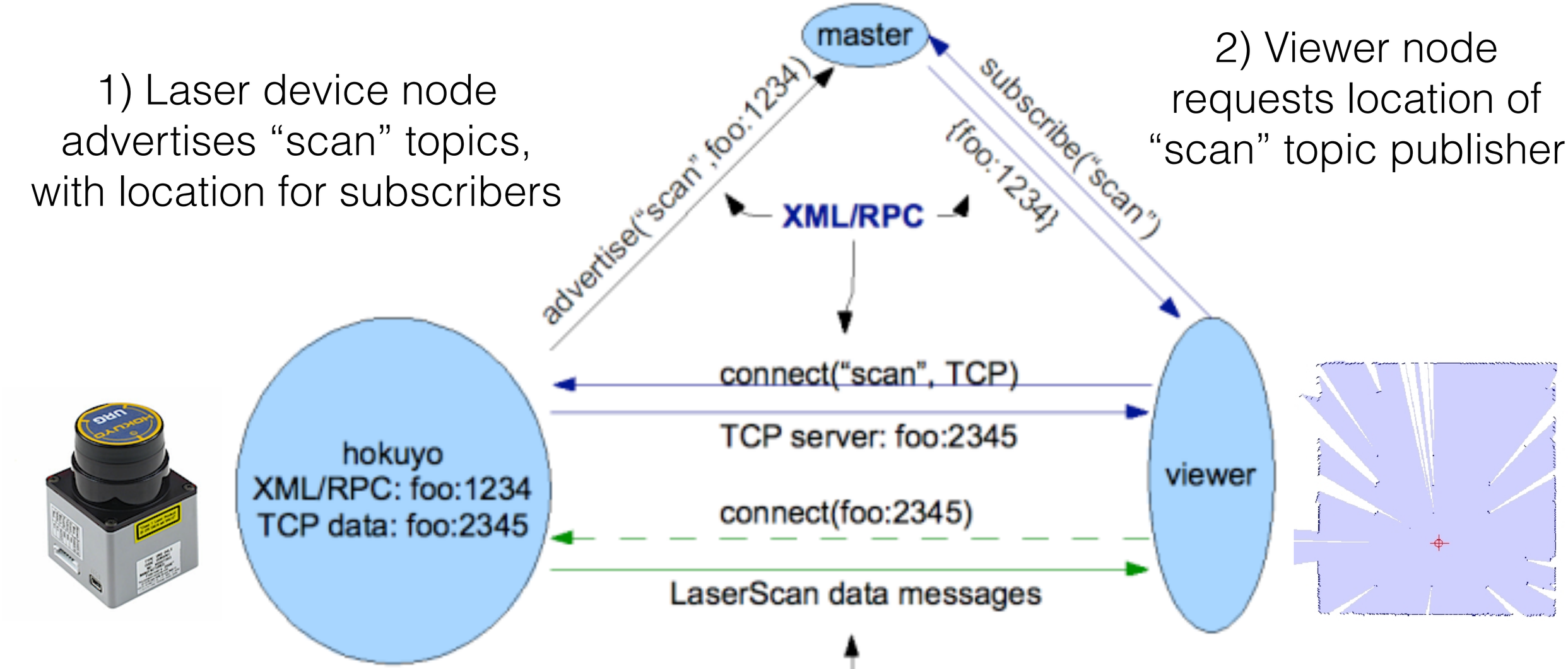


- ROS node: core unit of ROS run-time environment
  - Node is an executing process on an IP network
  - Node can publish or subscribe messages on a topic
- ROS Topic: basic message structure data exchange
  - Nodes subscribe to and publish topics as a stream
  - ROS Service: “function-like” request-reply
  - Transport: TCPROS (TCP/IP) or UDPROS (UDP/IP)
- ROS Master: topic name service for matching publisher and subscribers
  - Transport: XML-RPC (HTTP)

Laser range viewing example

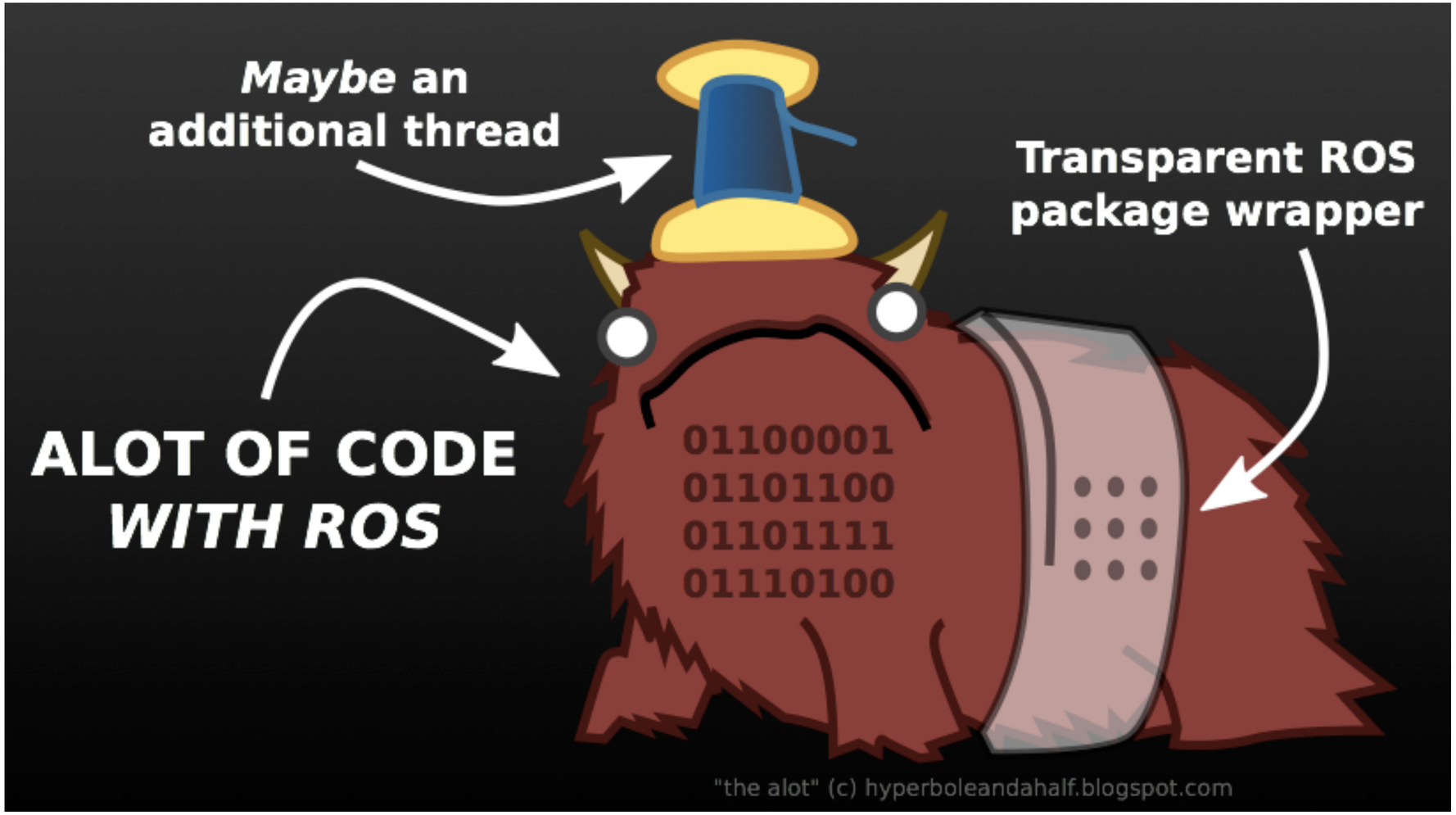
1) Laser device node advertises “scan” topics, with location for subscribers

2) Viewer node requests location of “scan” topic publisher



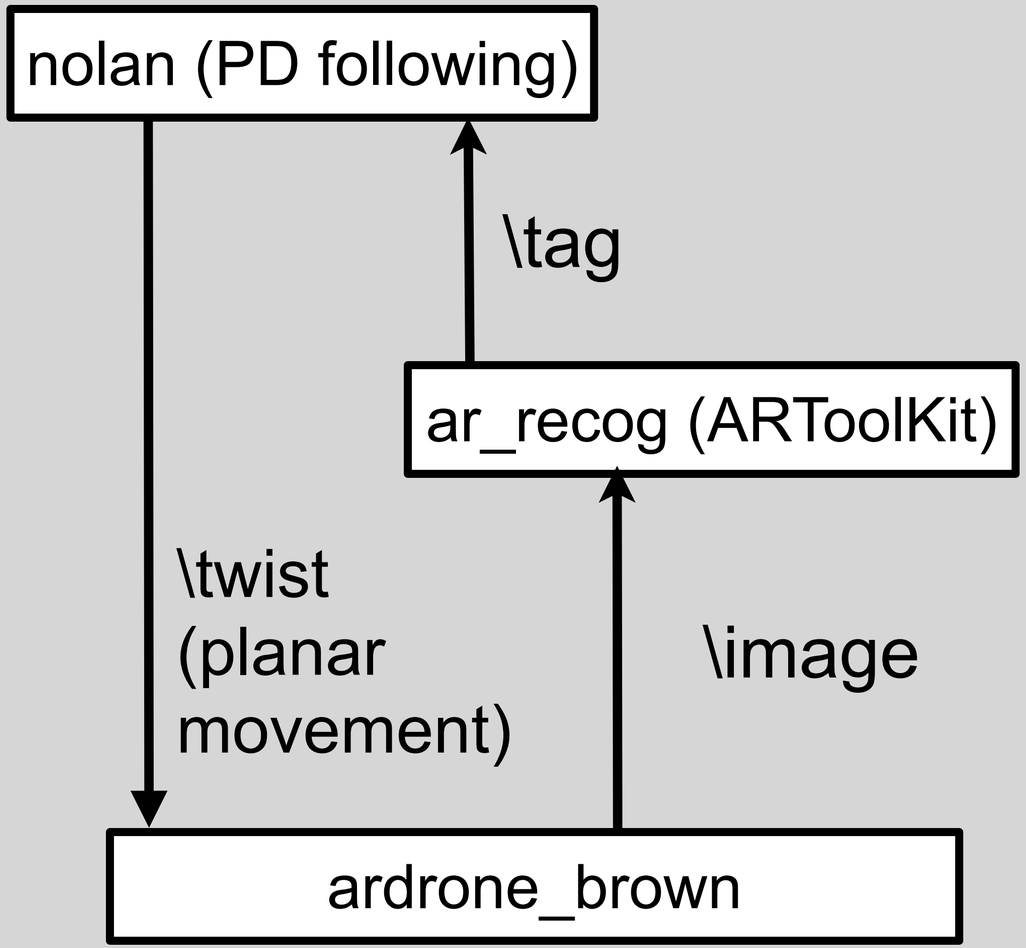
4) Viewer connects to laser node for transmission of topic messages

3) Viewer requests subscription of “scan” topic from laser



- ROS nodes are typically wrappers around a device, software library, or some “alot” of code

Software portability across robots



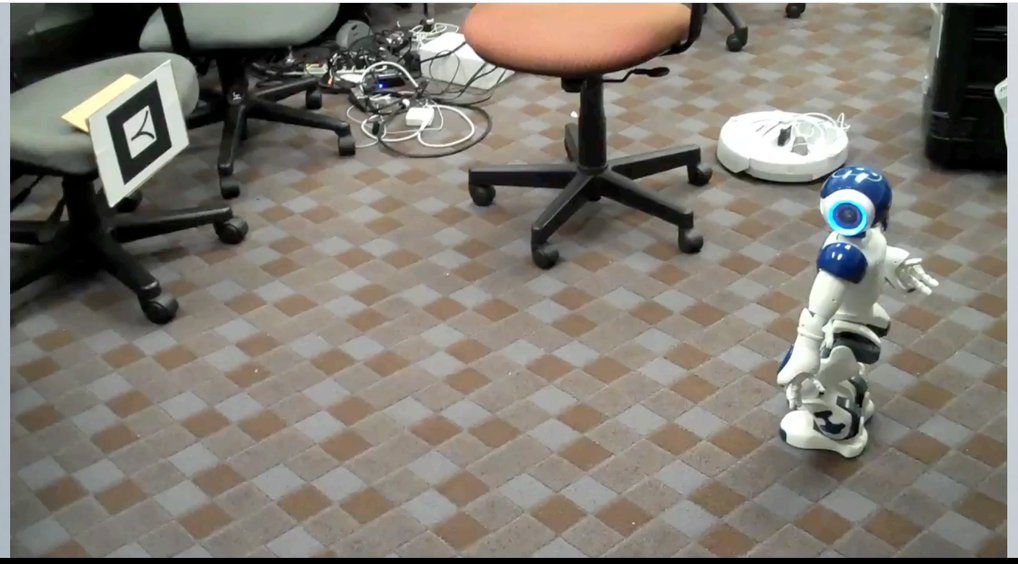
Search projects

ardrone_brown second flight - Nov 2010

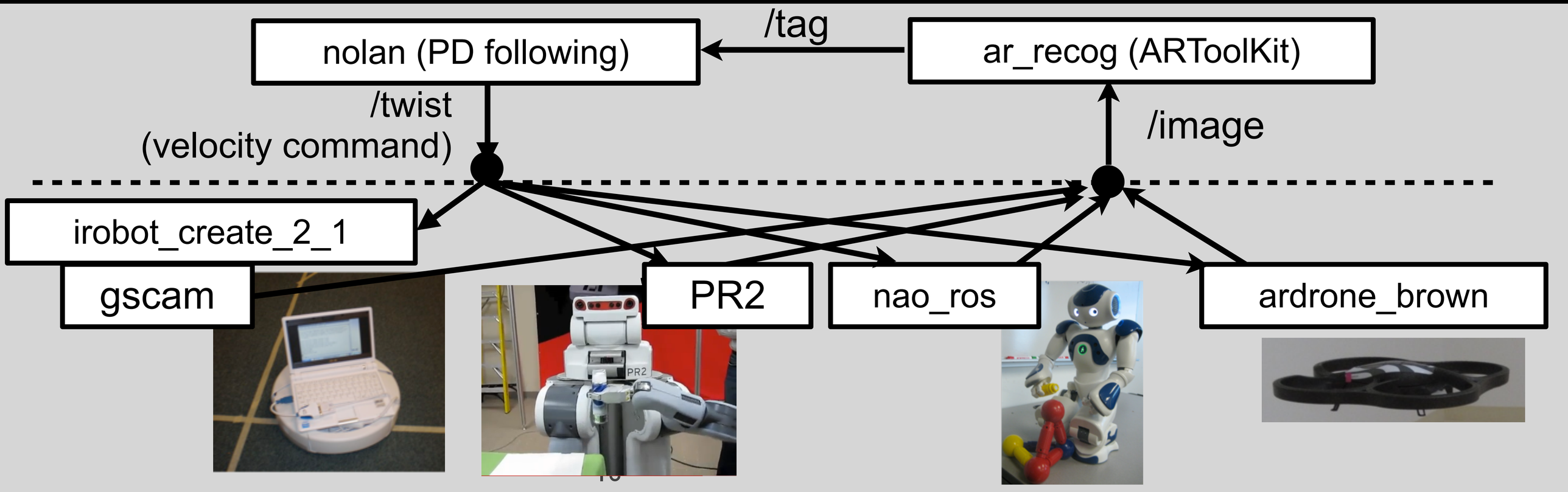
Project Hosting Help

ect Hosting

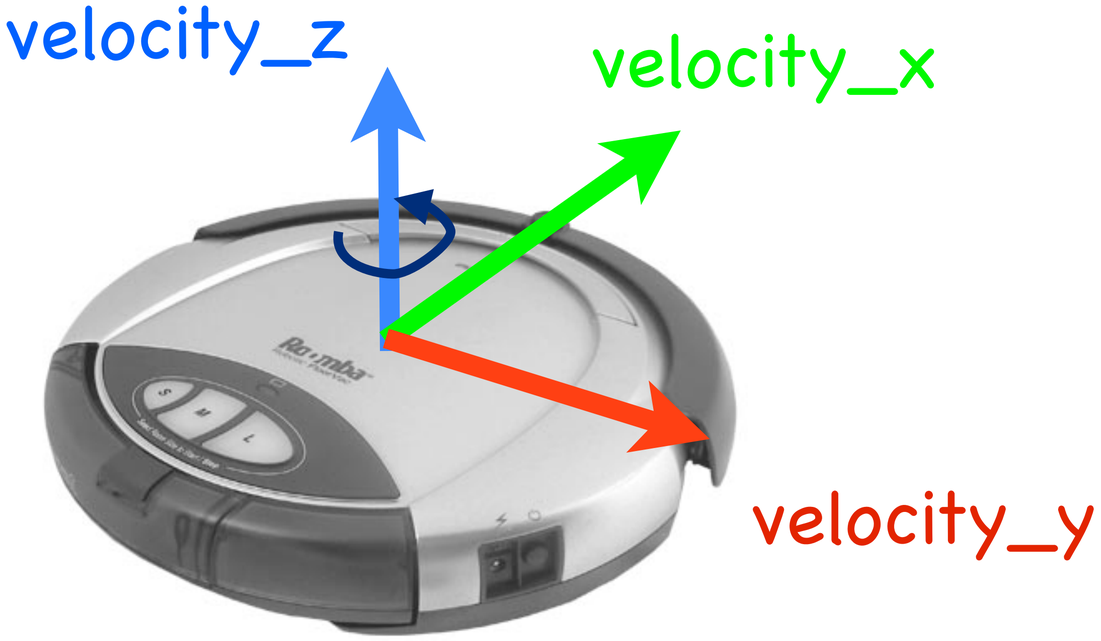
http://www.youtube.com/watch?v=mKmqgVUbQQM



http://www.youtube.com/watch?v=7eCgll-4hjk



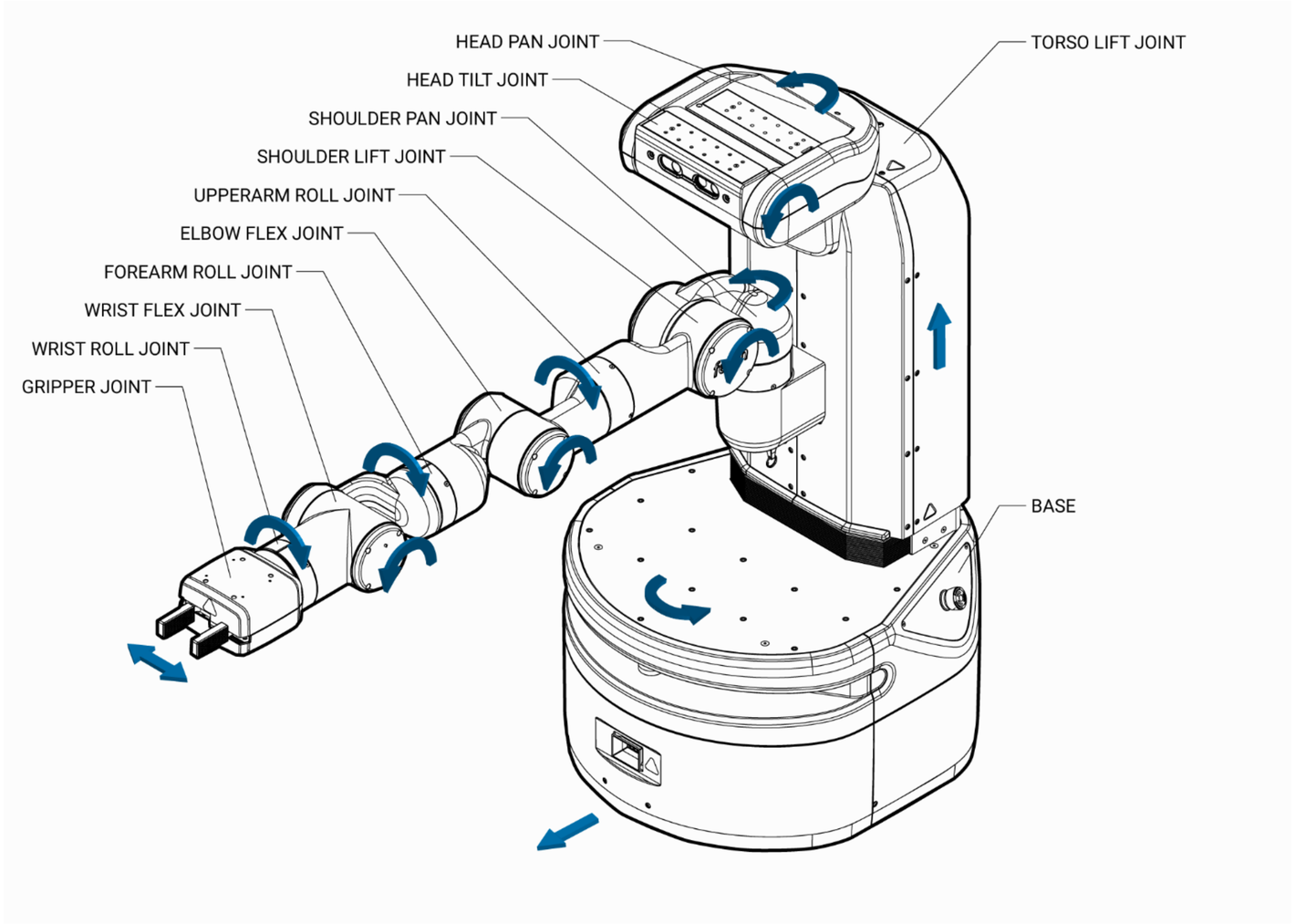
Topic descriptions



Twist message:

Vector3 linear
  float64 x
  float64 y
  float64 z
Vector3 angular
  float64 x
  float64 y
  float64 z

Topic descriptions

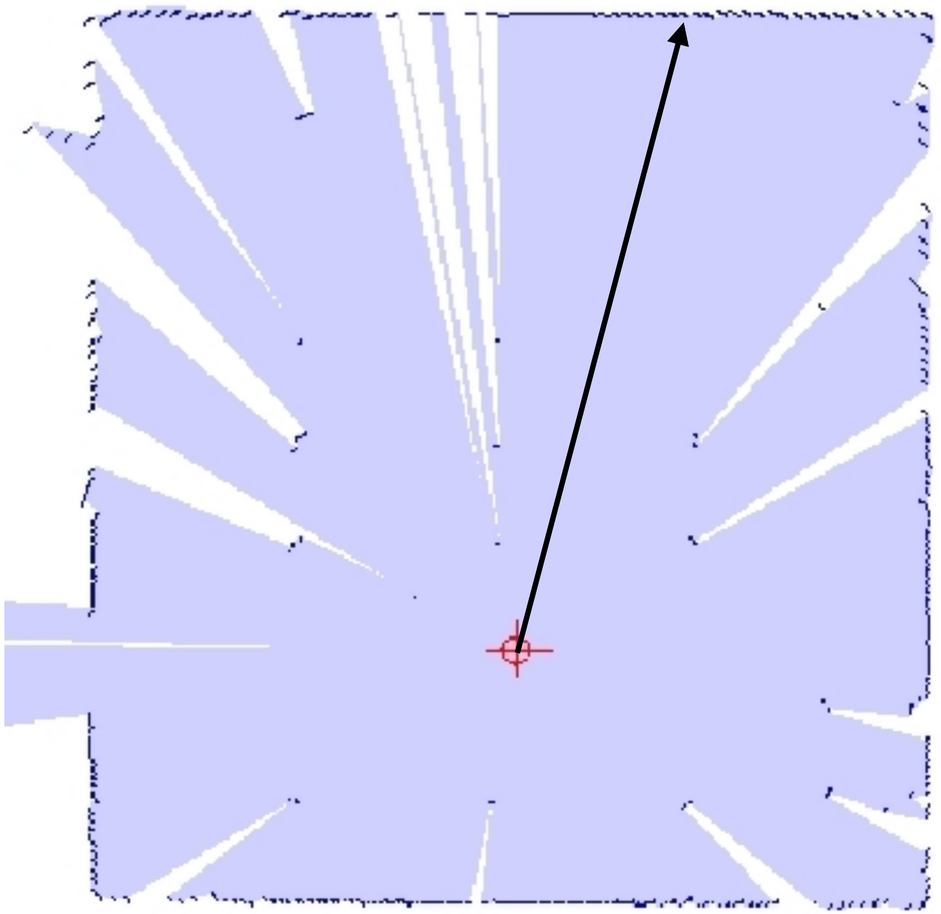


Joint State message:

std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort

an array entry for each joint

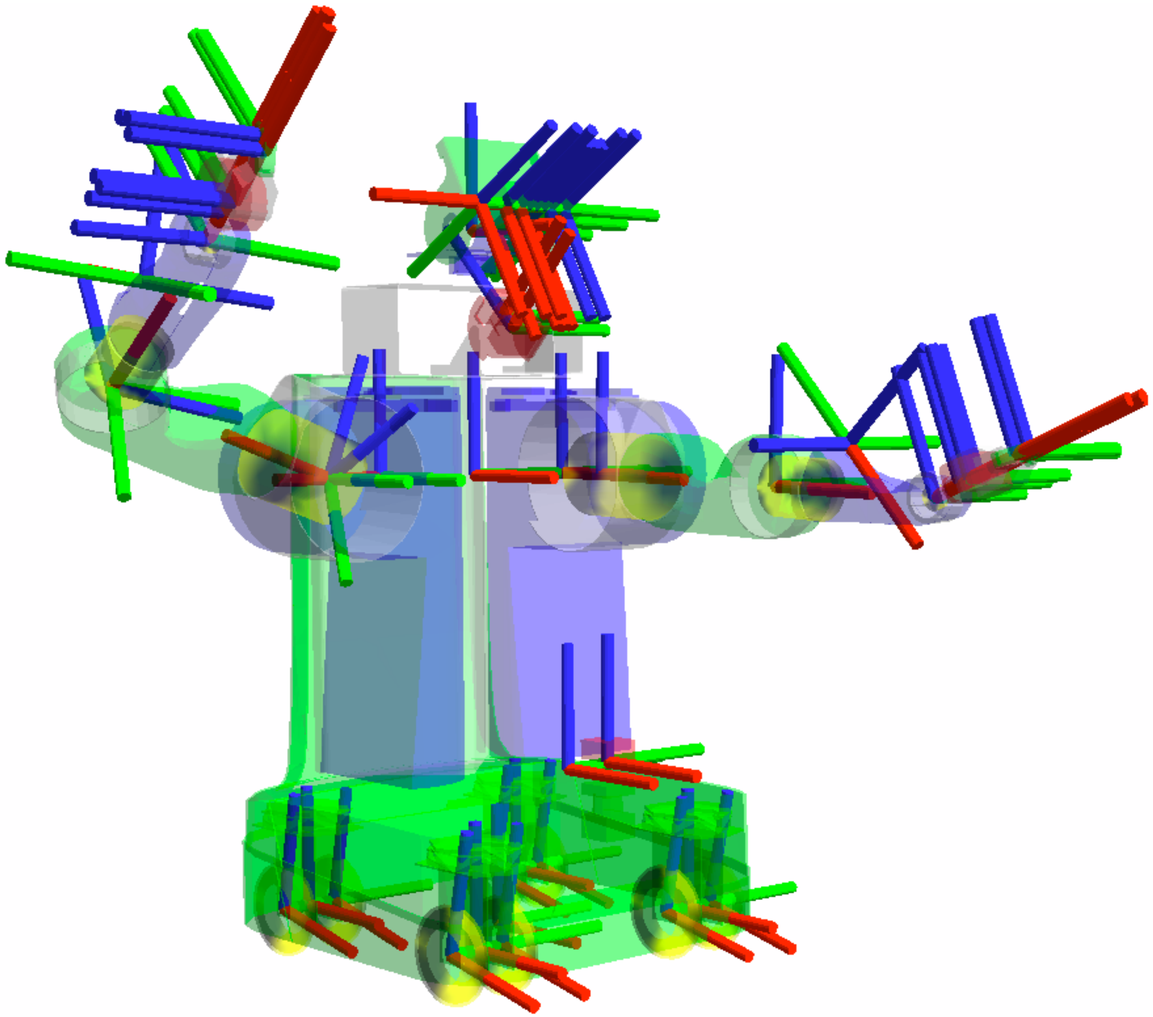
Topic descriptions



Laser Scan message:

std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities

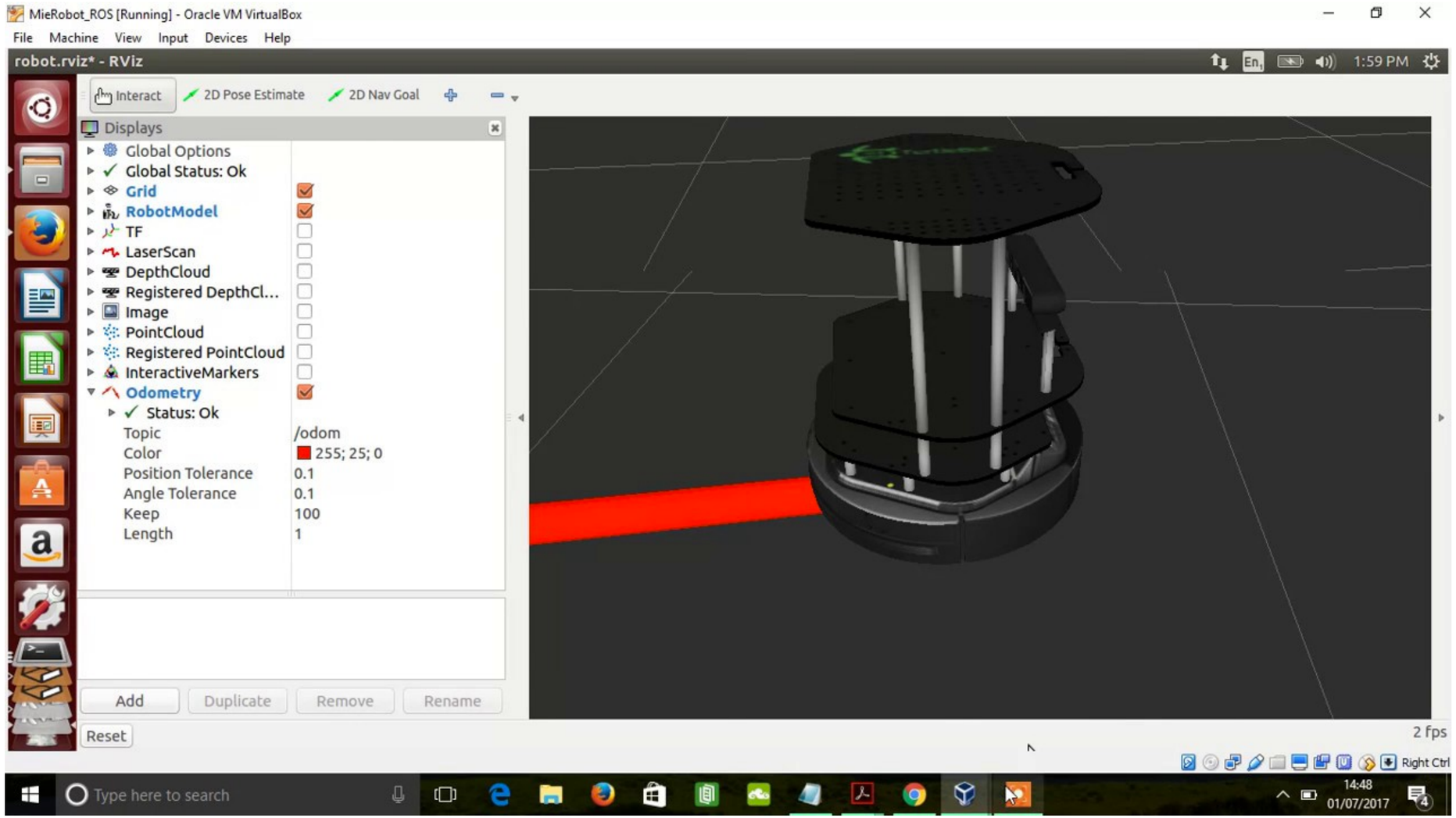
Topic descriptions



tf (transform) message:

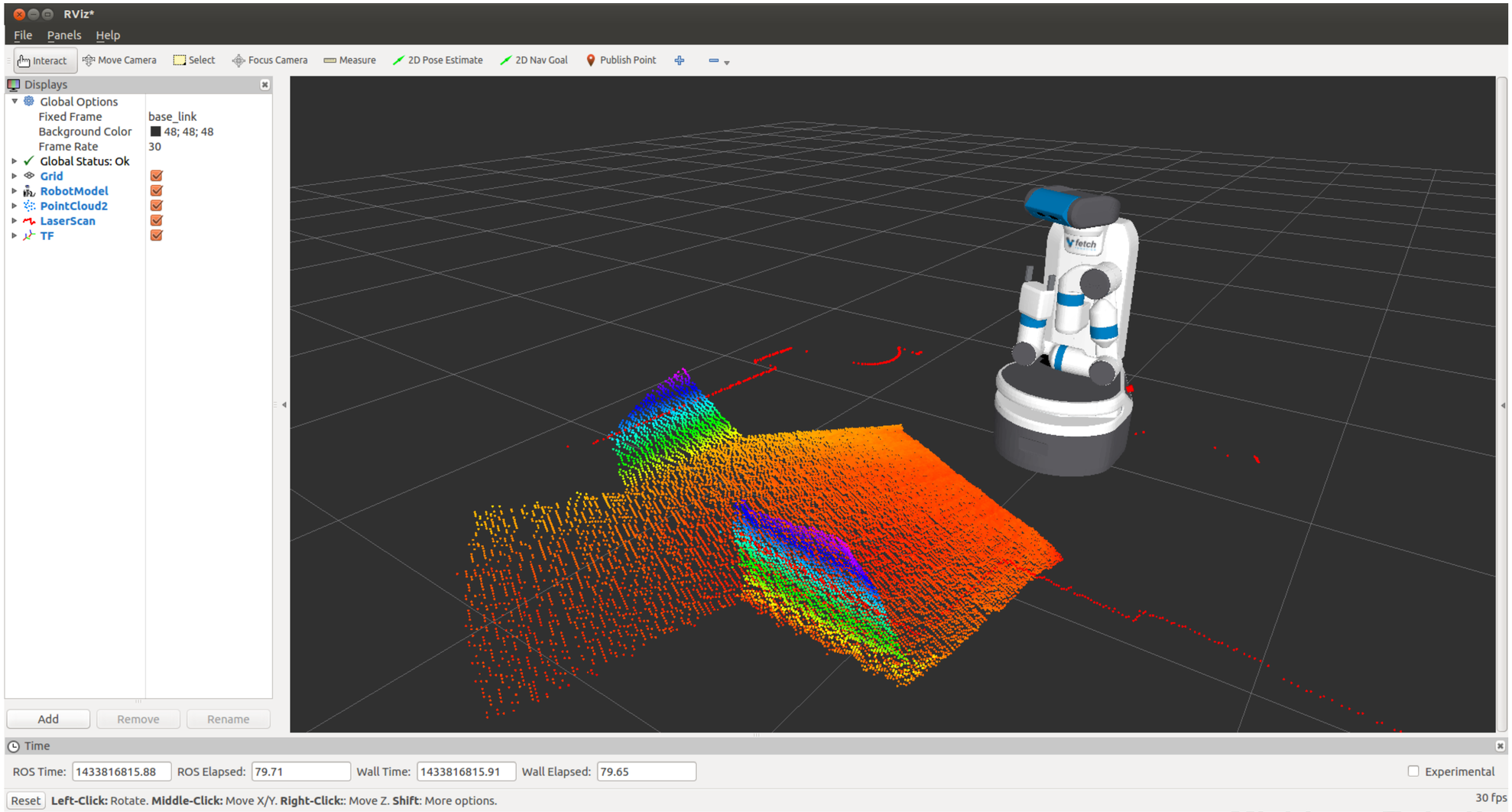
std_msgs/Header header
string child_frame_id
geometry_msgs/Transform transform
  geometry_msgs/Vector3 translation
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion rotation
    float64 x
    float64 y
    float64 z
    float64 w

Topic descriptions



Odometry

Topic descriptions



PointCloud2

Writing ROS nodes (warning: Diamondback!)

ROS Packaging System

- Each project will be a "package", ROS's basic dev unit
  - Packages are built essentially by CMake
- Client libraries (eg, roscpp, rospy, rosjs, rosjava)
- Package management: integrated build system
  - roscreate-pkg to create a package
  - rosmake to build, rosrun to execute nodes
- Integration of external packages and repositories
  - OpenCV, OpenRAVE, Player, etc.
- brown-ros-pkg contains drivers for Create, PS3 cam,...

Common ROS pkg structure

ROS packages tend to follow a common structure. Here are some of the directories and files you may notice.

- `bin/`: compiled binaries (**C++ nodes**)
- `include/package_name`: C++ include headers
- `msg/`: **Message** (msg) types
- `src/package_name/`: Source files
- `srv/`: **Service** (srv) types
- `scripts/`: executable scripts (**Python nodes**)
- `launch/`: launch files
- `CMakeLists.txt`: CMake build file (see **CMakeLists**)
- `manifest.xml`: Package **Manifest**
- `mainpage.dox`: Doxygen mainpage documentation

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

invoke python

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

python includes

My first python program "hello_create"

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

import topic definitions

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

start event loop for create_spin_and bump()

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True

                                        initialize node and topics
def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

initialize node and topics

publish twist messages for Create's "cmd_vel" topic

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

initialize node and topics

subscription to Create's sensorPacket topic, spawns message handler thread

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

initialize node and topics

register node with master

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

allocate and initialize variables

My first python program "hello_create"

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

handle event for subscribed topic

event loop

publish control

DO NOT USE large sleep value

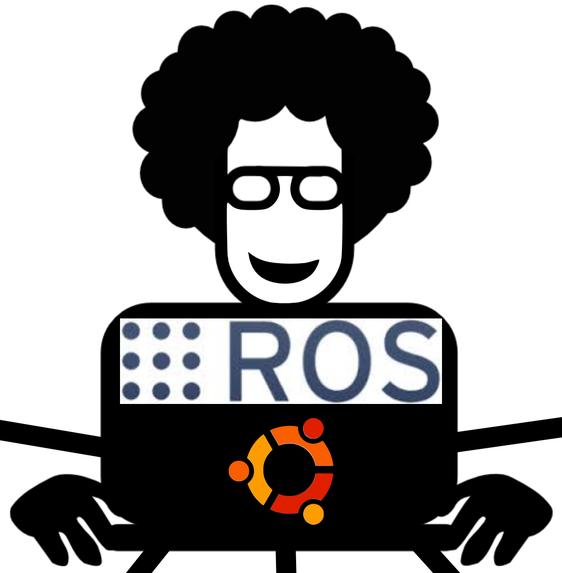
```
#!/usr/bin/env python
import roslib; roslib.load_manifest('hello_create')
import rospy
#from std_msgs.msg import String
from geometry_msgs.msg import Twist
# listen
from irobot_create_2_1.msg import SensorPacket
# global variables
bump = False

# listen (adapted from line_follower
def processSensing(sensorPacket):
    global bump
    bump = sensorPacket.bumpLeft or sensorPacket.bumpRight
    #newInfo = True


def create_spin_and_bump():
    pub = rospy.Publisher('cmd_vel', Twist)
    rospy.Subscriber('sensorPacket', SensorPacket, processSensing)
    rospy.init_node('create_spin_and_bump')
    #listen
    global bump
    twist = Twist()
    while not rospy.is_shutdown():
        if bump:
            str = "hello create, you have bumped into something  %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.0; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = -0.5
            bump = False
        else:
            str = "hello create, you can spin now %s"%rospy.get_time()
            rospy.loginfo(str)
            twist.linear.x = 0.1; twist.linear.y = 0; twist.linear.z = 0
            twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0.5
        pub.publish(twist)
        rospy.sleep(1.0)
if __name__ == '__main__':
    try:
        create_spin_and_bump()
    except rospy.ROSInterruptException: pass
```

My first python program “hello_create”

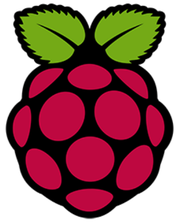
What behavior will this node produce?

Heterogeneity

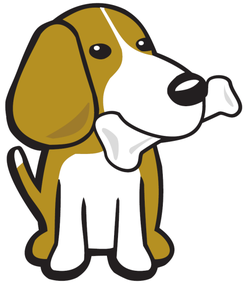

PR2

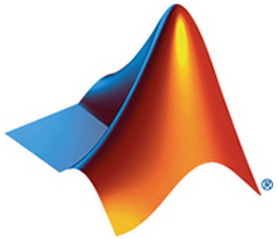

ROS

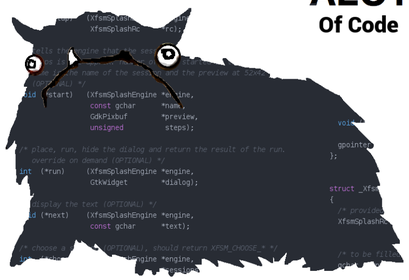

PR2

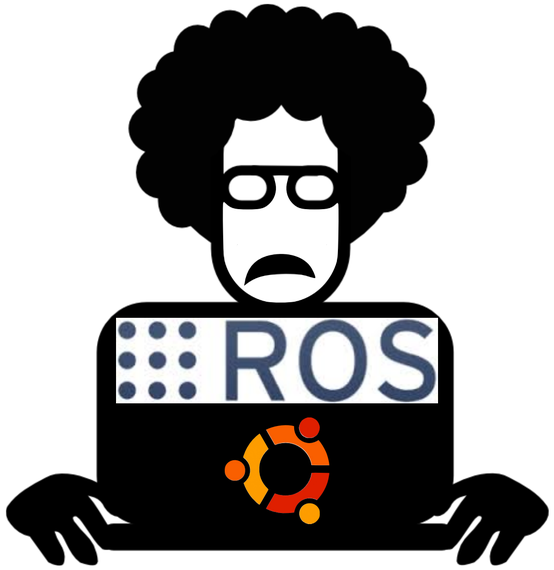



ARDUINO






ALOT
Of Code


ROS

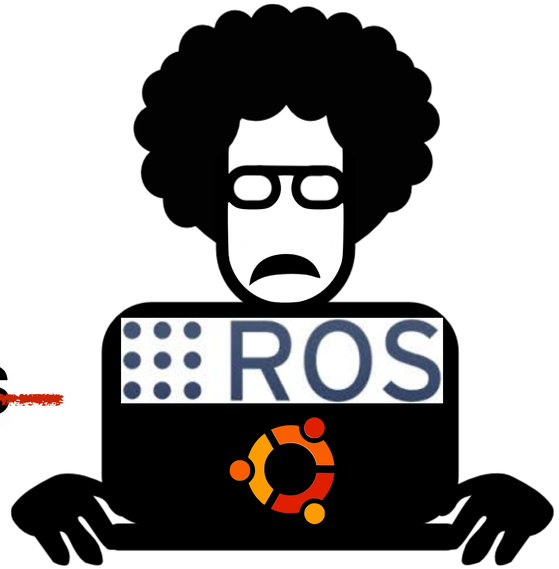
~~Different Compute Platforms~~

~~Different Programming Languages~~

~~Different Operating Systems~~

~~Custom non-ROS Codebases~~

ALOT
Of Code



~~Different Front-ends~~

ROS is great...

It affects the quality of science in robotics

- Provides interoperability within ROS environment
- Platform as a build and distribution system
- Easy to get started and working
- Peer-to-peer communications

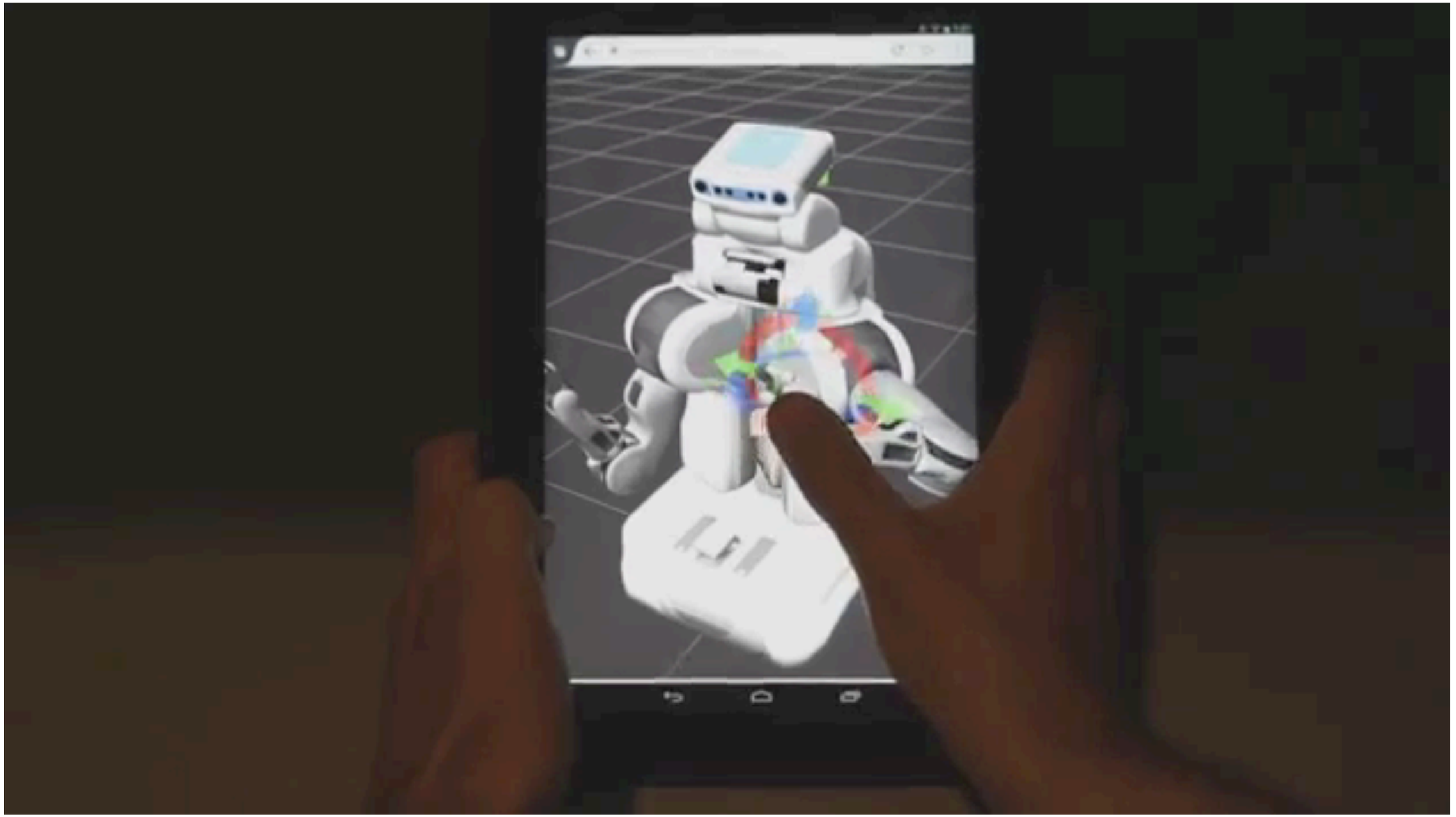
ROS is great... but...

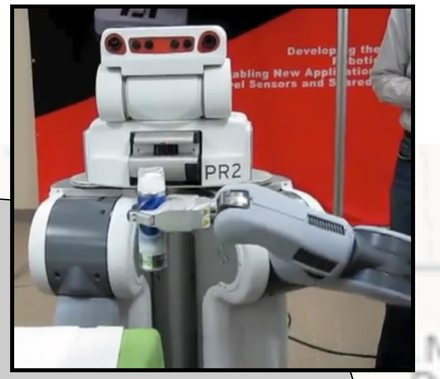
It affects the quality of science in robotics

- Provides interoperability within ROS environment
- Platform as a build and distribution system
- Easy to get started and working
- Peer-to-peer communications

- Limited interoperability beyond specific ROS/Ubuntu versions
- No defined protocol or standards, restricts to specific platform
- Difficult to maintain and handle dependencies
- Not suited to client/server comms

Robot Web Tools [Toris, Jenkins, et al., IEEE RAM 2012, IROS 2015, http://robotwebtools.org]





ROS

PR2

rosbridge_server

websocket

browser

teleop.html

roslibjs.js

```
{ "op": "publish",
  "topic": "/cmd_vel",
  "msg": {"linear":{"x":1.0,"y":0,"z":0},
          "angular":{"x":0,"y":0,"z":0}
}
```

teleop.html

```
<html><head>
<script>type="text/javascript" src="roslibjs.js"</script>
<script>type="text/javascript" src="jquery-1.2.6.min.js"</script>
...

var ros = new ROS("ws://10.100.0.100:9090");
...

ros.publish('/cmd_vel','geometry_msgs/Twist','{"linear":{"x":'+x+',"y":0,"z":0},"angular":{"x":0,"y":0,"z":'+z+'}}');
...

ros.callService('/rosjs/subscribe',json(['/gscam/image_raw',0,'jpeg',128,96,90]),nop);
```

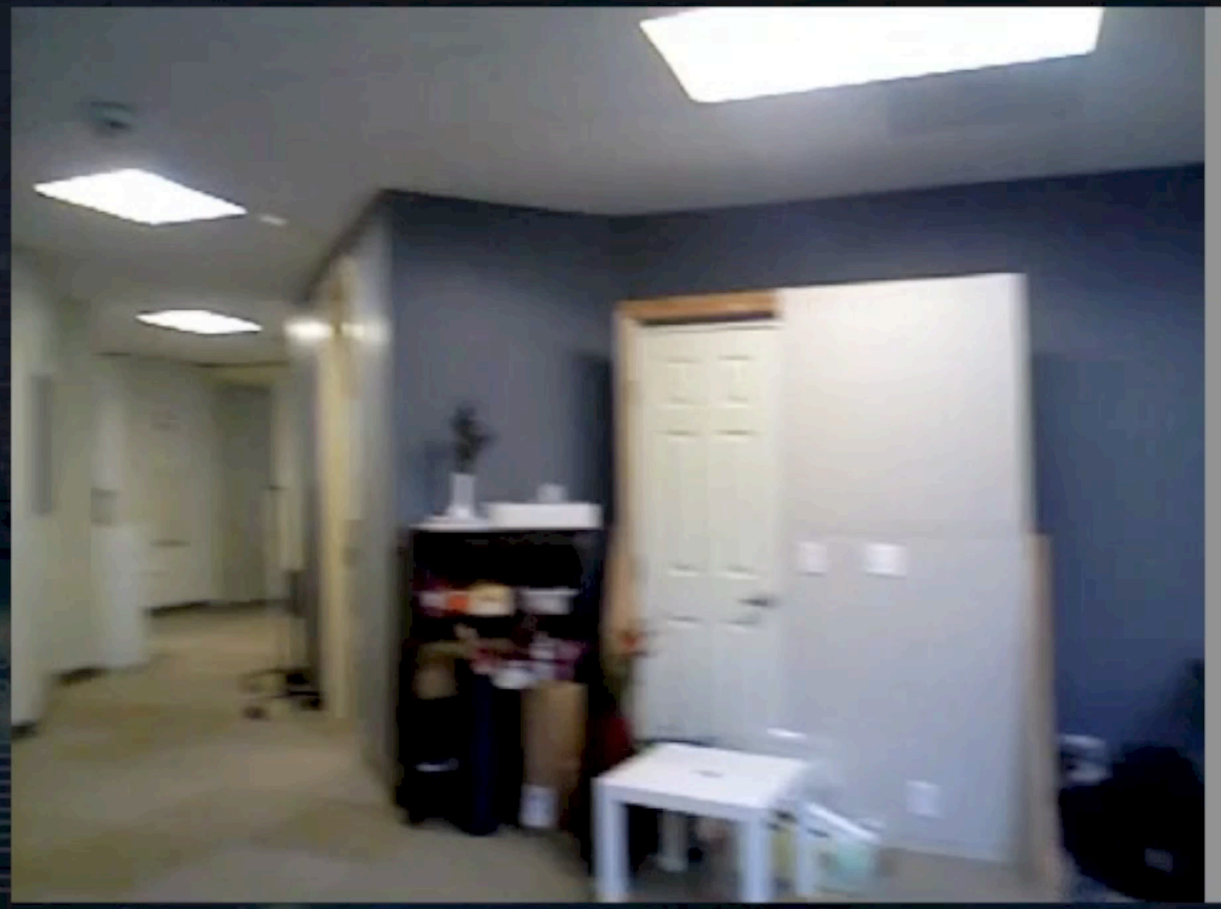
Open websocket connection

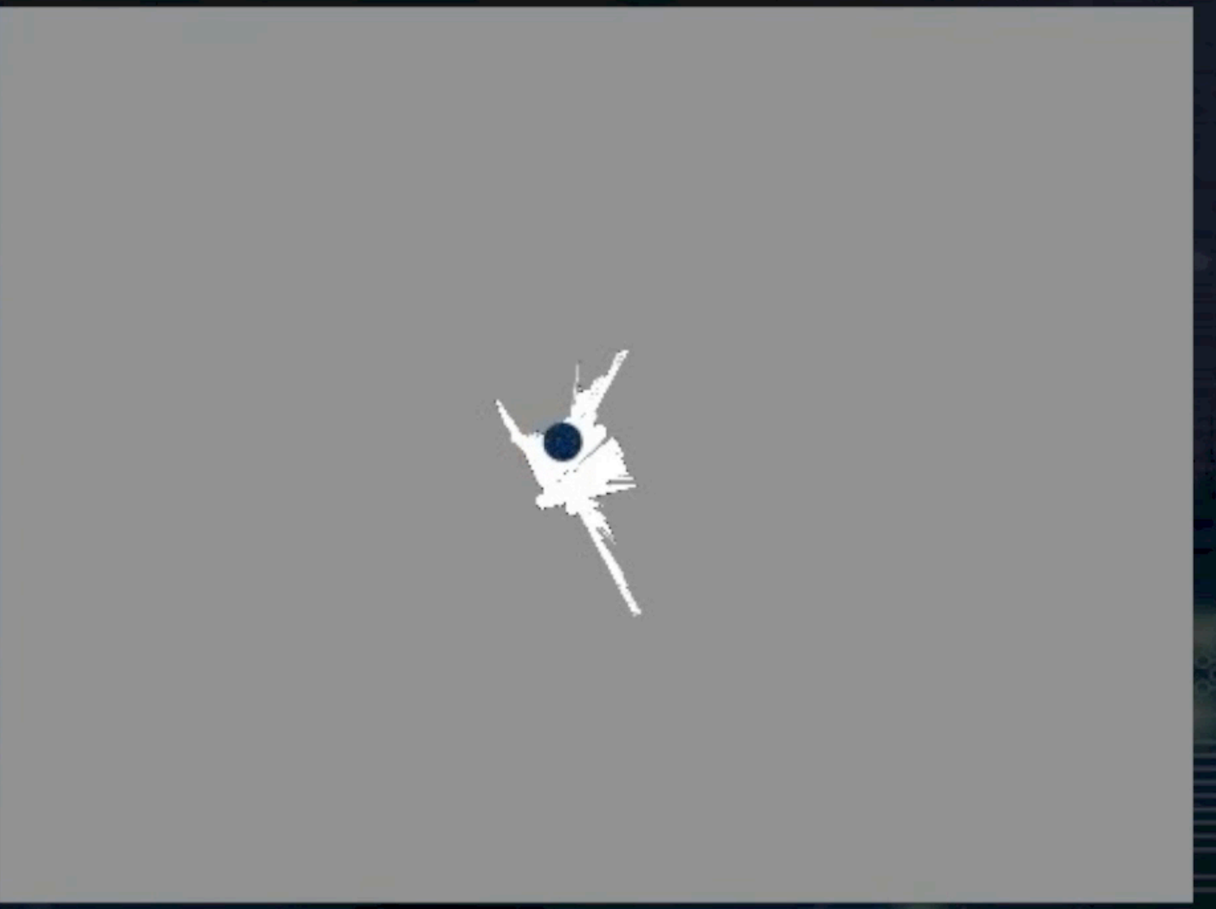
Send 6DOF velocity command (JSON-contained ROS topic)

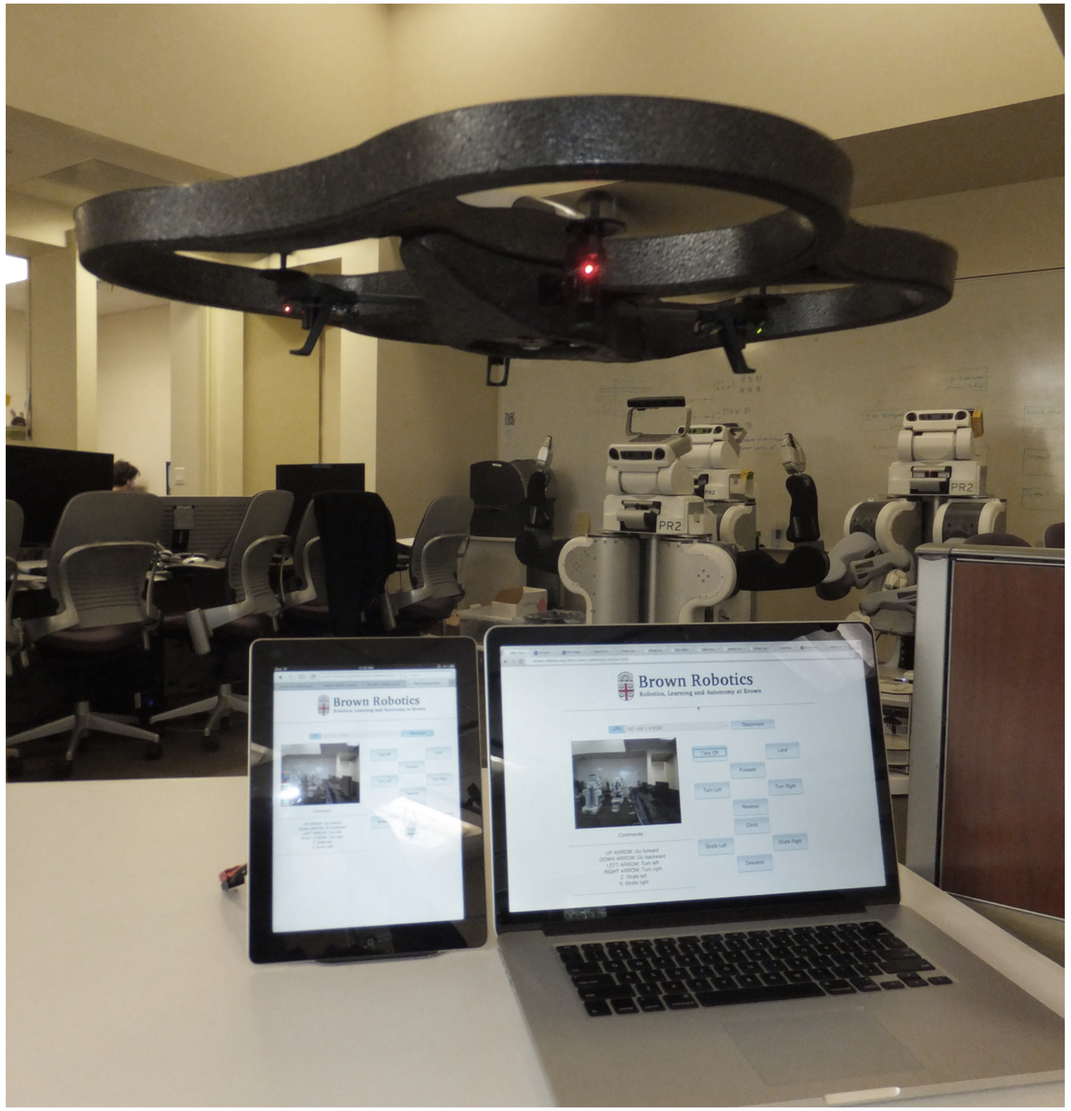
Return 128x96 jpeg array at 90% quality at ROS frame rate

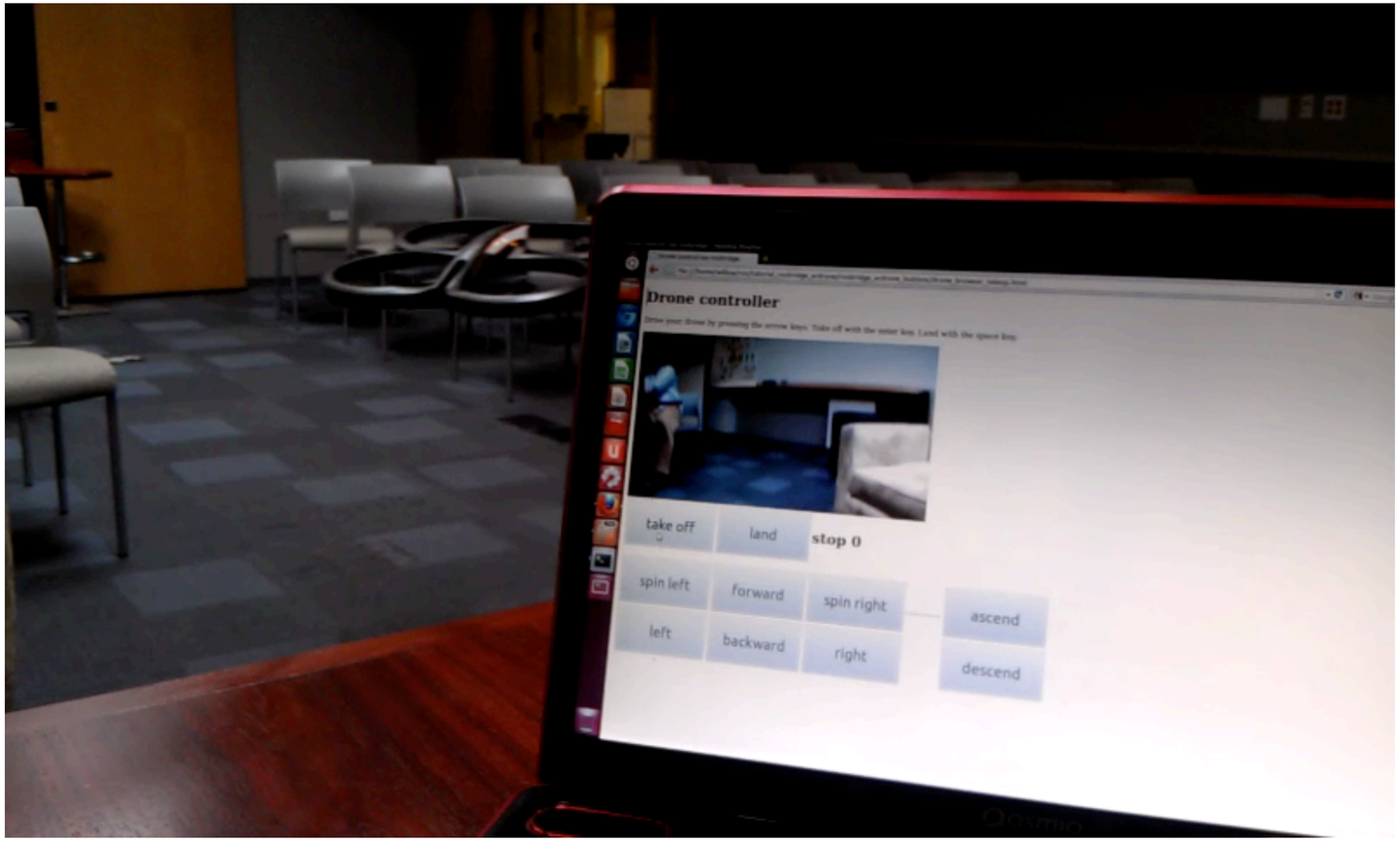
Map Builder

Speed: 90%






Brown Robotics
Robotics, Learning and Autonomy at Brown
Brown Robotics
Robotics, Learning and Autonomy at Brown
PR2
PR2


Drone controller
take off
land
stop 0
spin left
forward
spin right
ascend
left
backward
right
descend

[Beetz et al. 2015]

GitHub, Inc. (US) | https://github.com/RobotWebTools

robotwebtools

Search GitHub

Pull requests Issues Gist

Robot Web Tools

http://robotwebtools.org/

Repositories | People 7 | Teams 1 | Settings

Filters | Find a repository…

+ New repository

ros3djs

JavaScript ★ 36 ⑂ 35

3D Visualization Library for use with the ROS JavaScript Libraries

Updated 3 hours ago

roslibjs

JavaScript ★ 56 ⑂ 57

The Standard ROS JavaScript Library

Updated 19 hours ago

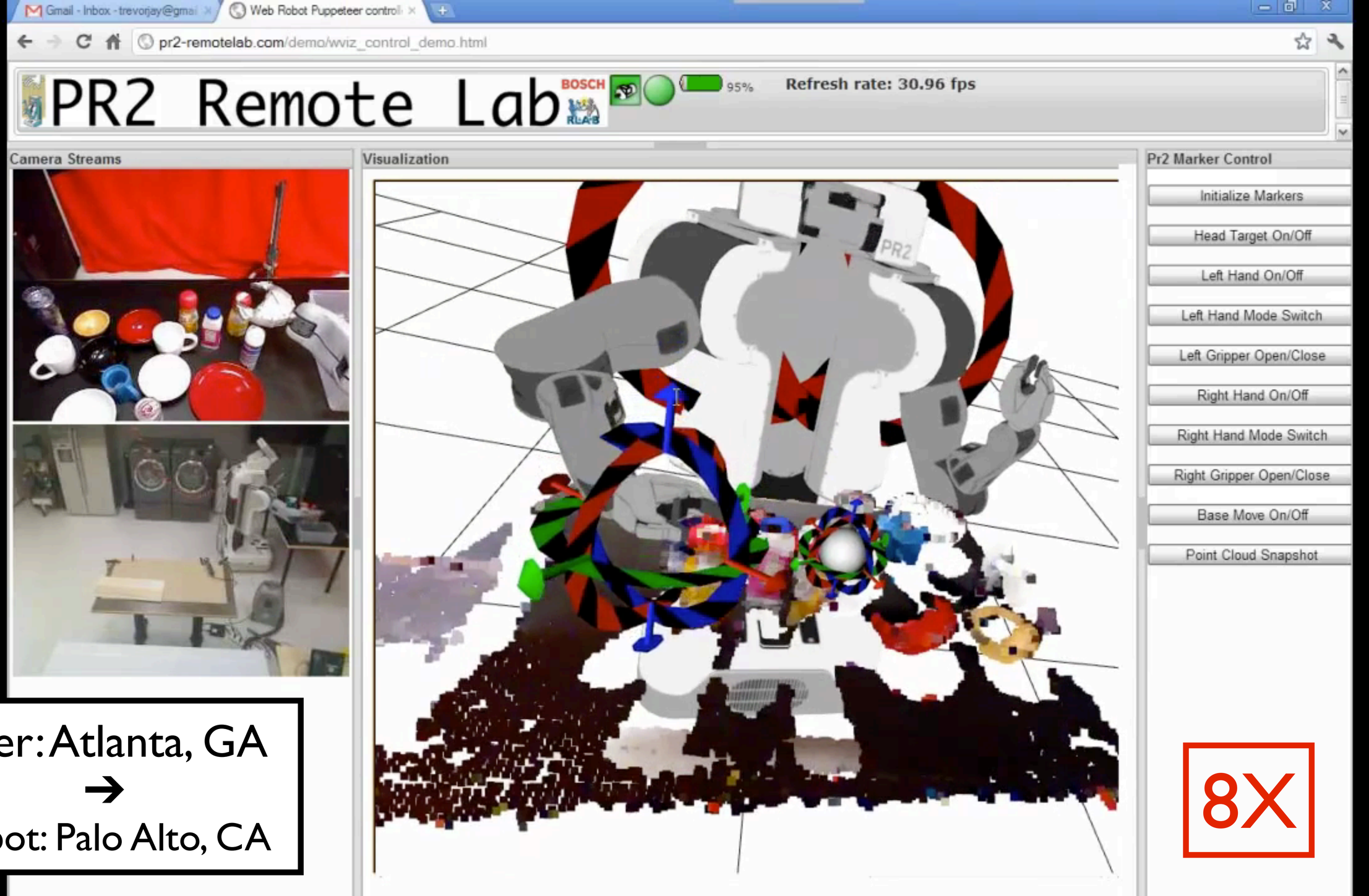
rosbridge_suite

Python ★ 37 ⑂ 67

Server Implementations of the rosbridge v2 Protocol

People 7 >

Invite someone



User: Atlanta, GA

→

Robot: Palo Alto, CA

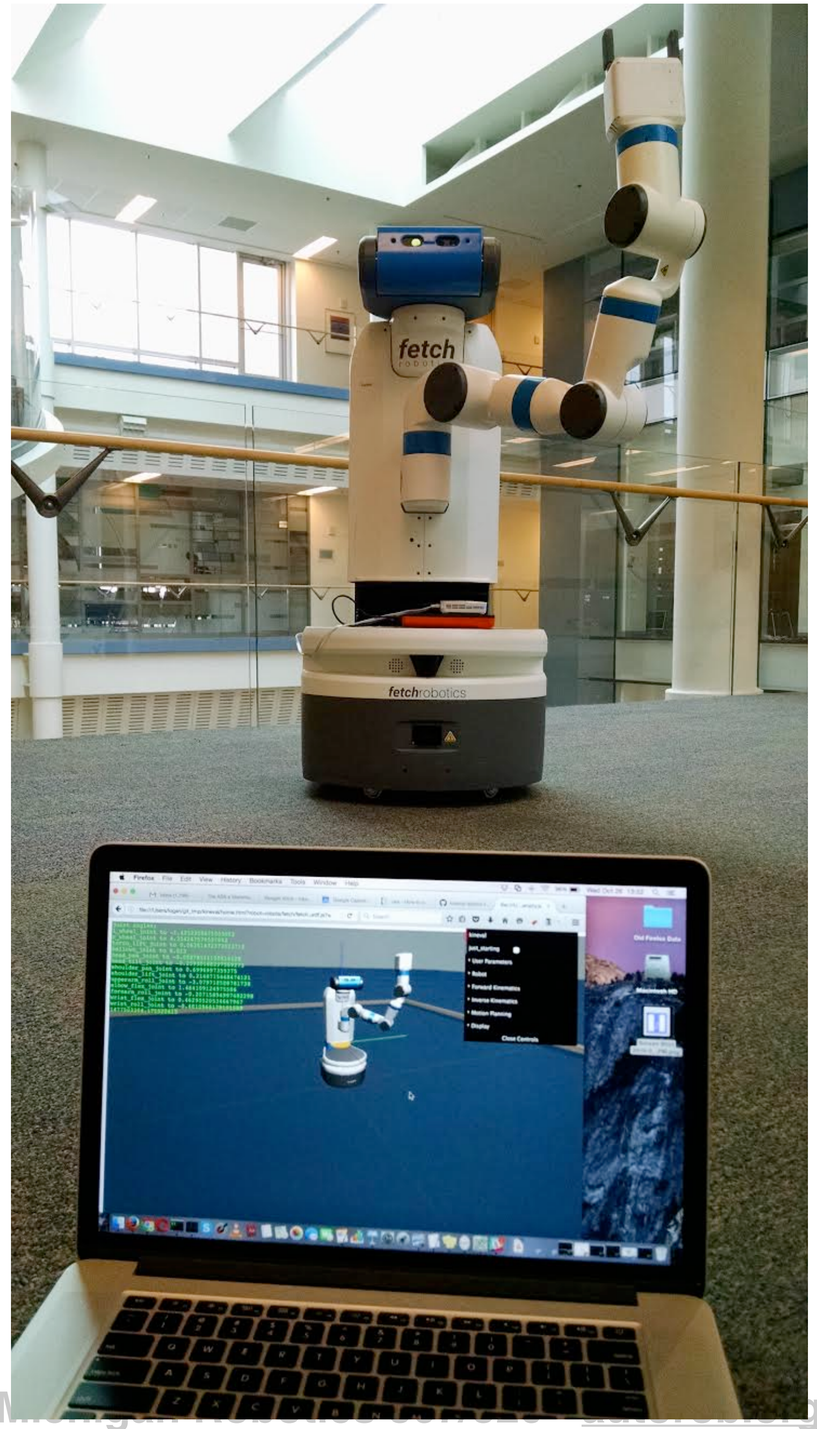
8X

How much bandwidth?

Problematic Topics for Streaming

- Kinematic transforms (tf)
- Images
- 3D point clouds from depth cameras
- General high-bandwidth topics (including 2D maps)

KinEval and *rosbridge*



KinEval and *rosbridge*

- Connect to mesh radio access point (PROGRESS 1)
- From terminal on robot **(DO NOT DO THIS ON YOUR OWN)**
  - run rosbridge_server to serve JSON encoded ROS topics:
    - rosrun rosbridge_server rosbridge_websocket
  - run topic throttler to publish joint state messages at 5Hz
    - rosrun topic_tools throttle messages joint_states 5

KinEval and *rosbridge*

- In kineval.js, uncomment call to kineval.initrosbridge()

```
55
56      // initialize rosbridge connection to robot running ROS, if available
57      // KE 2 : uncomment and add toggle
58      //kineval.initrosbridge();
59
```

- In kineval_rosbridge.js, update URL to websocket port on robot

```
1
2   kineval.initrosbridge = function init_rosbridge() {
3
4   // KE 2 : add this to kineval object
5   ros = new ROSLIB.Ros({
6       //url : 'ws://192.168.1.152:9090'
7       //url : 'ws://fetch7:9090'
8       //url : 'ws://fetch18.lan:9090'
9       url : 'ws://192.168.1.118:9090'
10  });
11
```

(ask instructor)

- Open home.html and verify your robot visualizes joint states


Next class:
Inverse Kinematics