Robot Middleware

EECS 398
Intro. to Autonomous Robotics

ME/EECS 567 ROB 510
Robot Modeling and Control

autorob.org

Fall 2018

Robot Middleware

EECS 398
o. to Autonomous Robotics

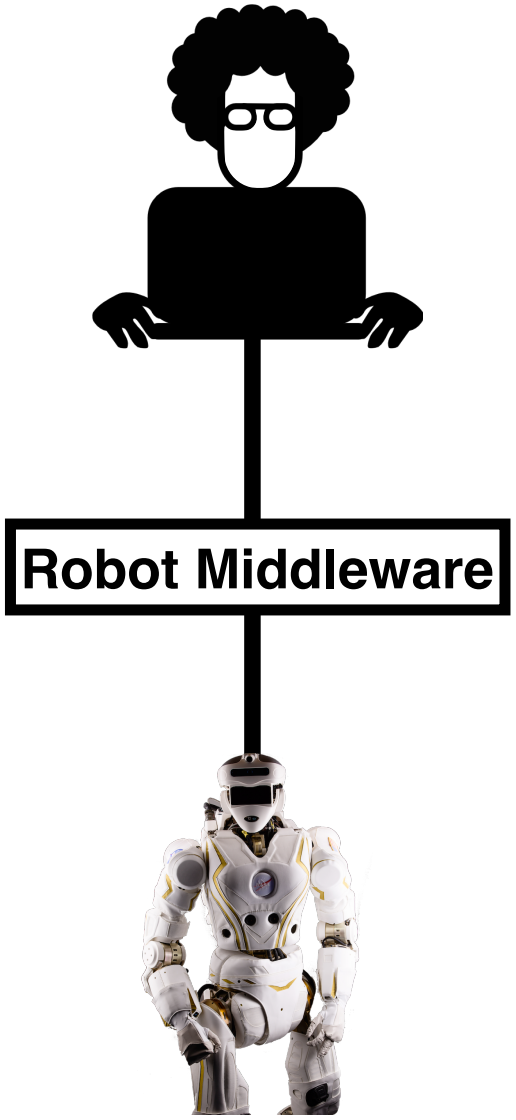
E/EECS 567 ROB 510
bot Modeling and Control

Fall 2018

au


PR2

Robot Middleware



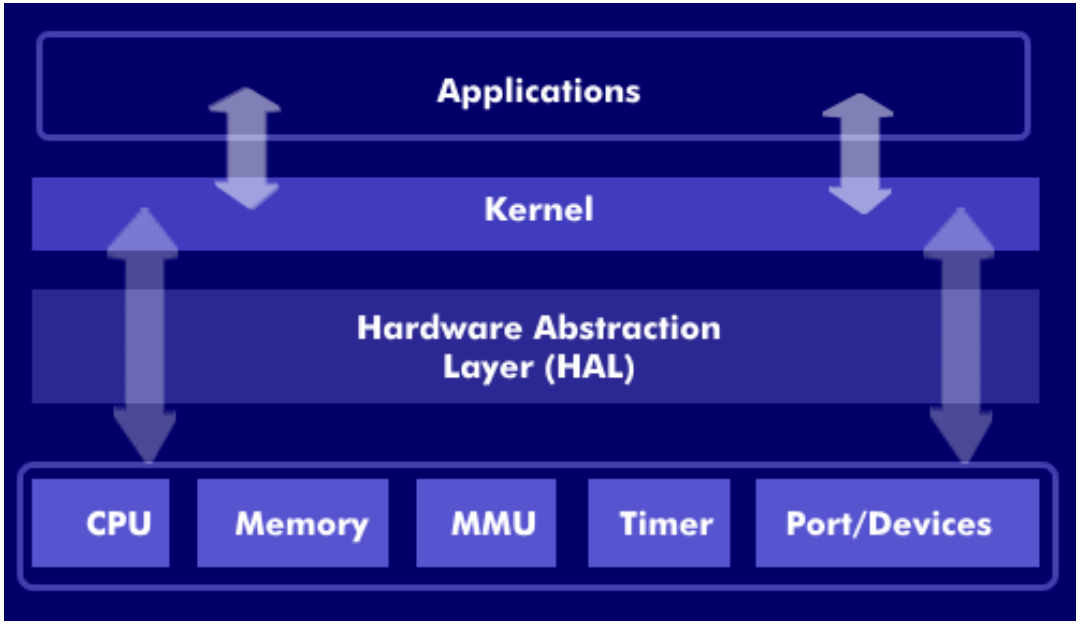
Hardware Abstraction

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

Programs

Abstraction

Devices



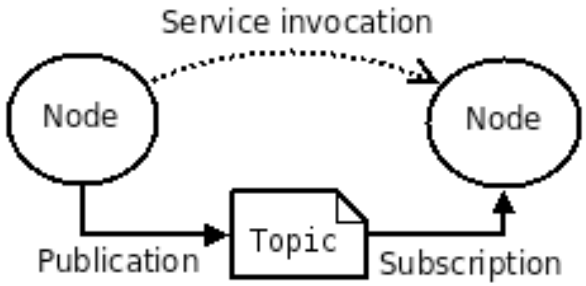
Abstraction enables “plug-and-play” interoperability and portability across compute platforms

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)

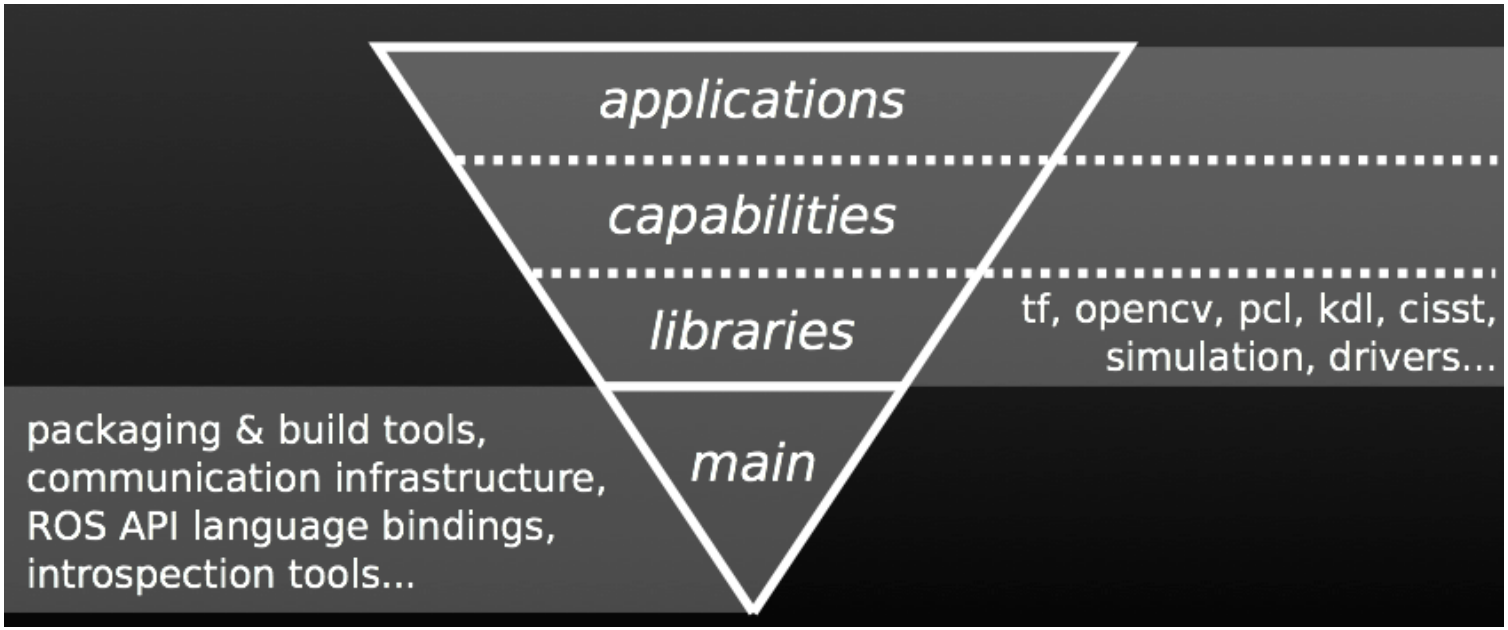
- “Robot Operating System”
  - Created by Morgan Quigley and Willow Garage
- Reduce “reinvention”, increase interoperability and reproducibility
- Peer-to-peer architecture over network
  - inter-process communication for robots
- Software functionality modularized as ROS nodes
  - Run-time system: nodes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



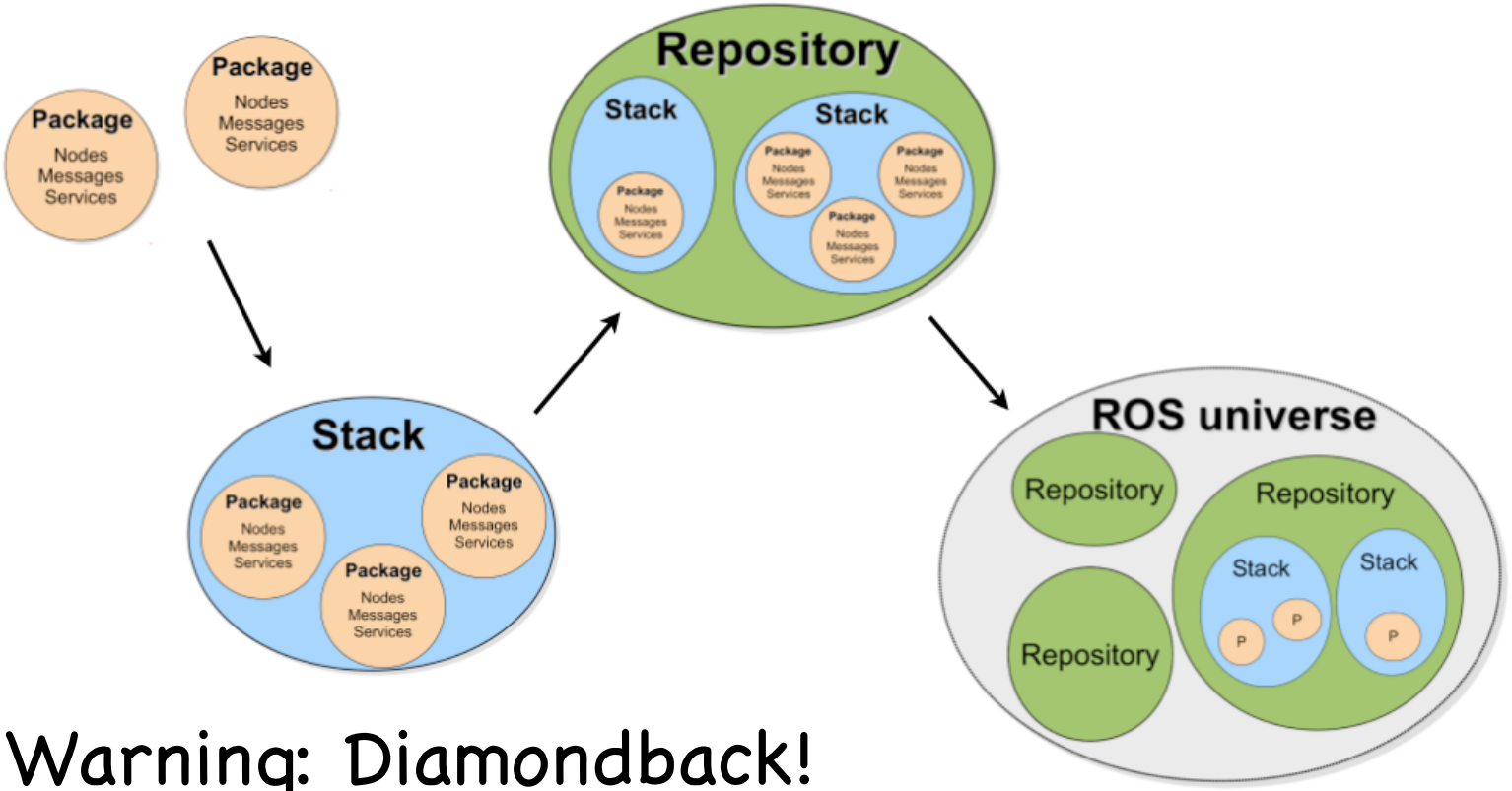
ROS (ros.org)

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

ROS as a development ecosystem

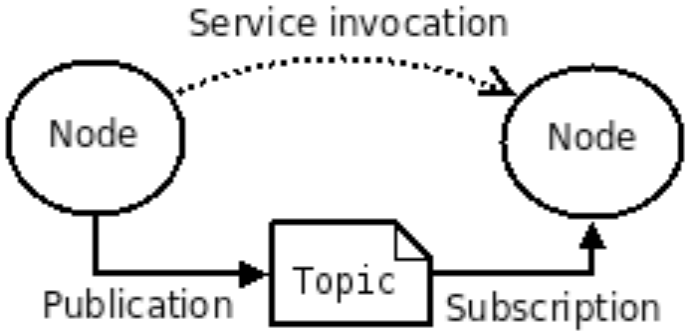


ROS Packaging System



Warning: Diamondback!

ROS Run-time

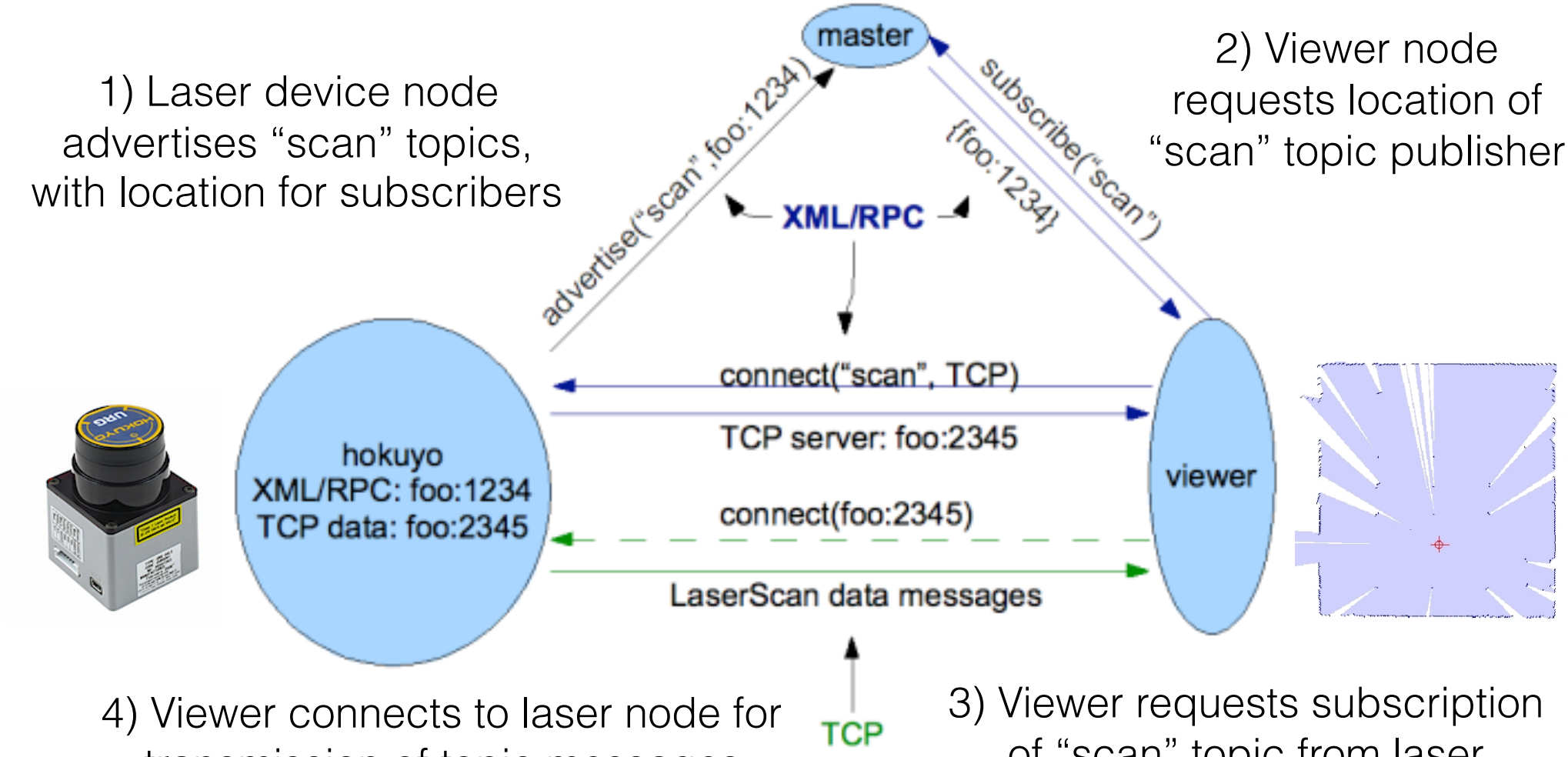


- ROS node: core unit of ROS run-time environment
  - Node is an executing process on an IP network
  - Node can publish or subscribe messages on a topic
- ROS Topic: basic message structure data exchange
  - Nodes subscribe to and publish topics as a stream
  - ROS Service: “function-like” request-reply
  - 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

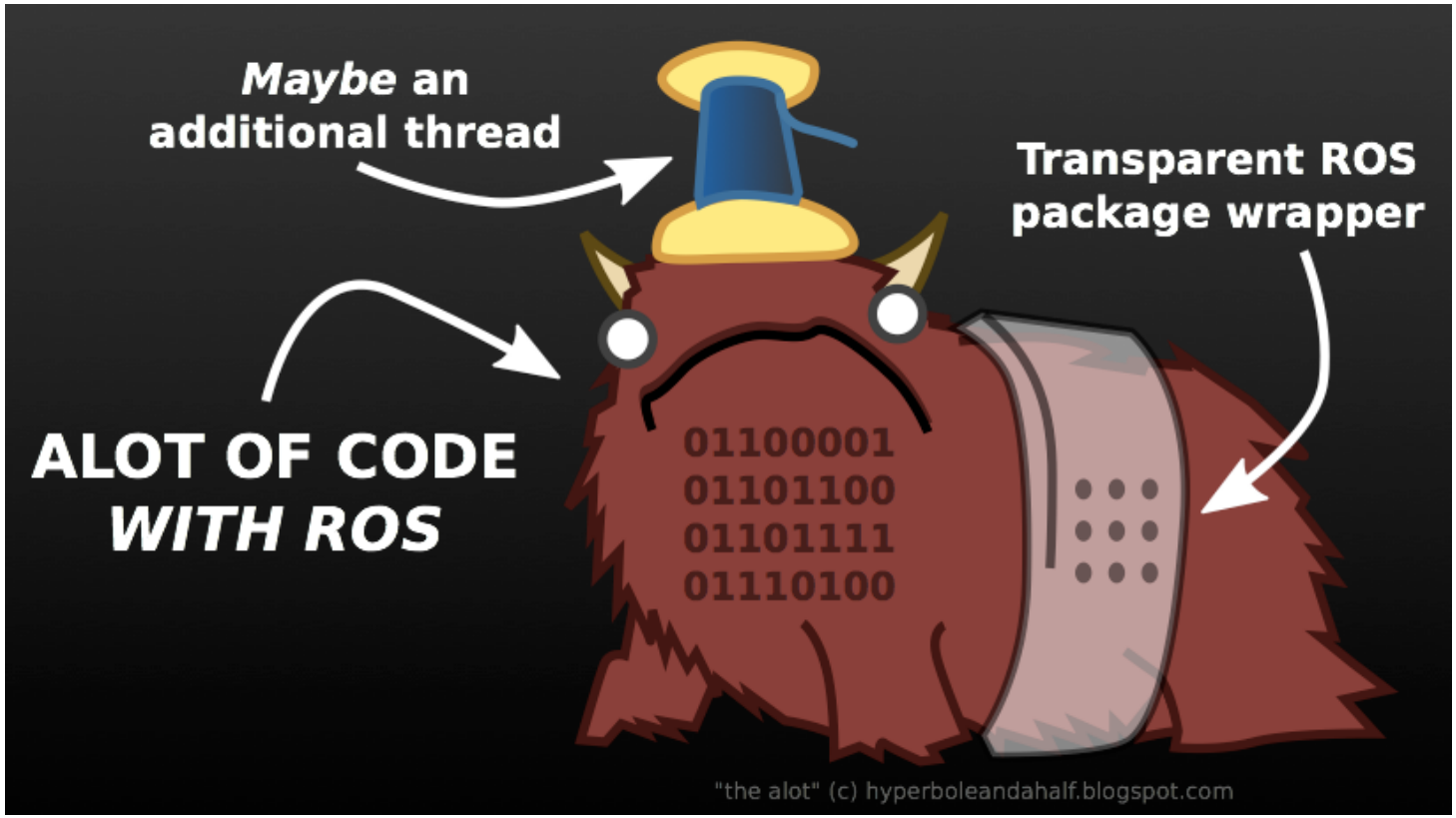
1) Laser device node advertises “scan” topics, with location for subscribers

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

Washington National Airport

Brookstone

Enter Keyword or Item # Search

Find a Store | Track Your Order | Email Signup

Customer Care 866-576-7337 | Shopping Cart

All Categories | Outdoor Living | Electronics | Bedroom | Massage | iPad & iPhone | Wine & Bar | Games & Toys | Travel | Bath & Spa | Gifts | Sale

Back Yard | Outdoor Seating | Pool Accessories | Hammocks | Grilling | Furniture Covers | All Outdoor Furniture

Summer Cooling shop all

Home > Games & Toys > Electronic Toys > AR.Drone & Accessories > Parrot® AR.Drone® 2.0 App-Controlled Quadricopter

Parrot® AR.Drone® 2.0 App-Controlled Quadricopter

Captures HD video and stills of your flights–lets you easily share them online.

4.4 (85 reviews)

Images (18) | Videos (5)

More Colors!

Play

ENLARGE

For ages 14 and up. WARNING: Choking hazard due to small parts.

Easier to fly than a traditional remote control helicopter and so much more fun!

- Fly your AR.Drone 2.0 with the free downloadable app* for your iOS or Android Device
- Perform advanced stunts ordinary RC helicopters can't
- Fly up to 165 feet from your Wi-Fi device
- See what the pilot sees with new front facing camera
- Capture 720p HD videos and stills to share online
- Enjoy controlled, level flying with automatic stabilization system
- Play augmented reality games that turn the world around you into a video game
- Fly indoors or out

By clicking the "Buy Now" button, you acknowledge that you have read, understand and accept the terms of the Customer Purchase Agreement. Please click here to view the full text of this agreement as well as the Limited Warranty from Parrot.

More...

Email 304 | Pinterest 208 | Tweet 119 | +1 131

Order Now

$299.99

In Stock

Color Select

Qty 1

Buy Now

Add Protection Plus

Service Plan - 2 Year $39.99

Service Plan - 3 Year $49.99

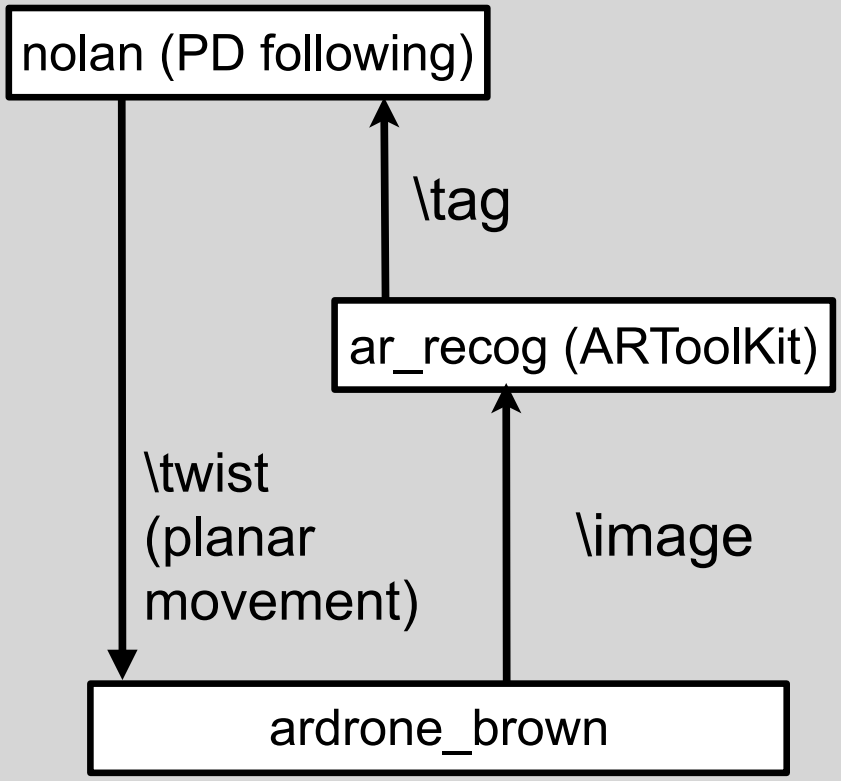
For Your Convenience

Gift Wrap $6.95

AR.Drone Tool Kit $19.99

AR.Drone 2.0 Battery $39.99

Buy Now



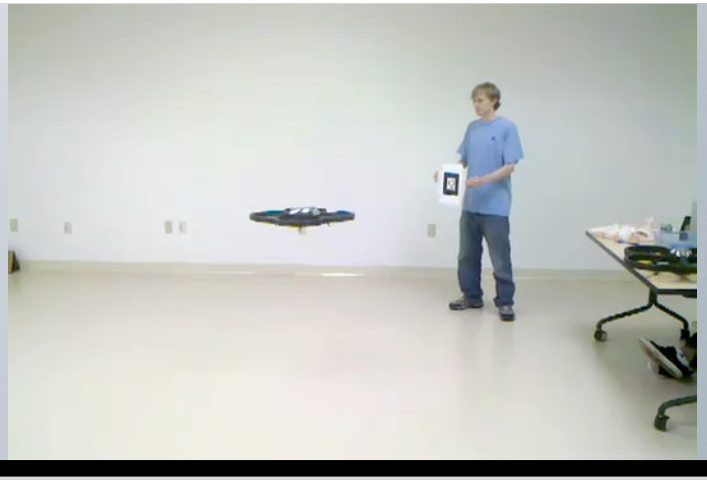
Search projects

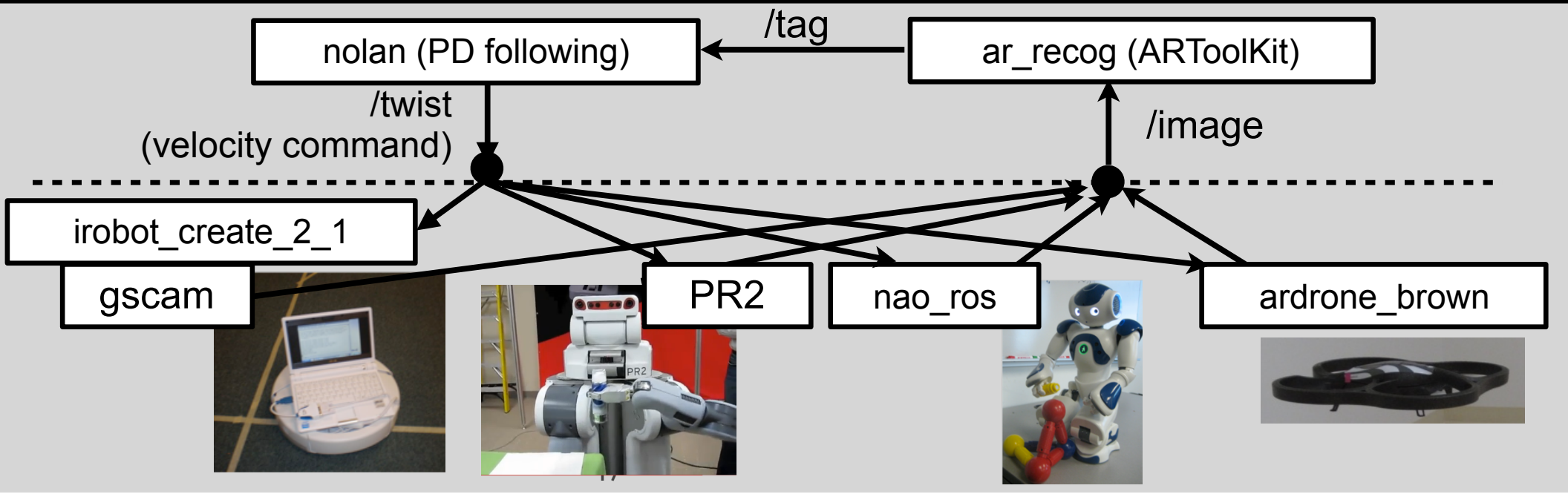
Project Hosting Help

ect Hosting

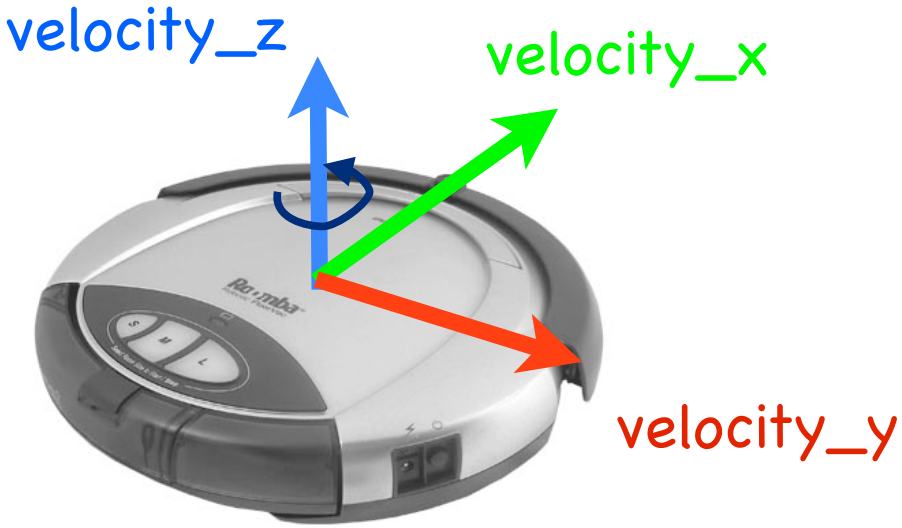
http://www.youtube.com/watch?v=mKmqgVUbQQM

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





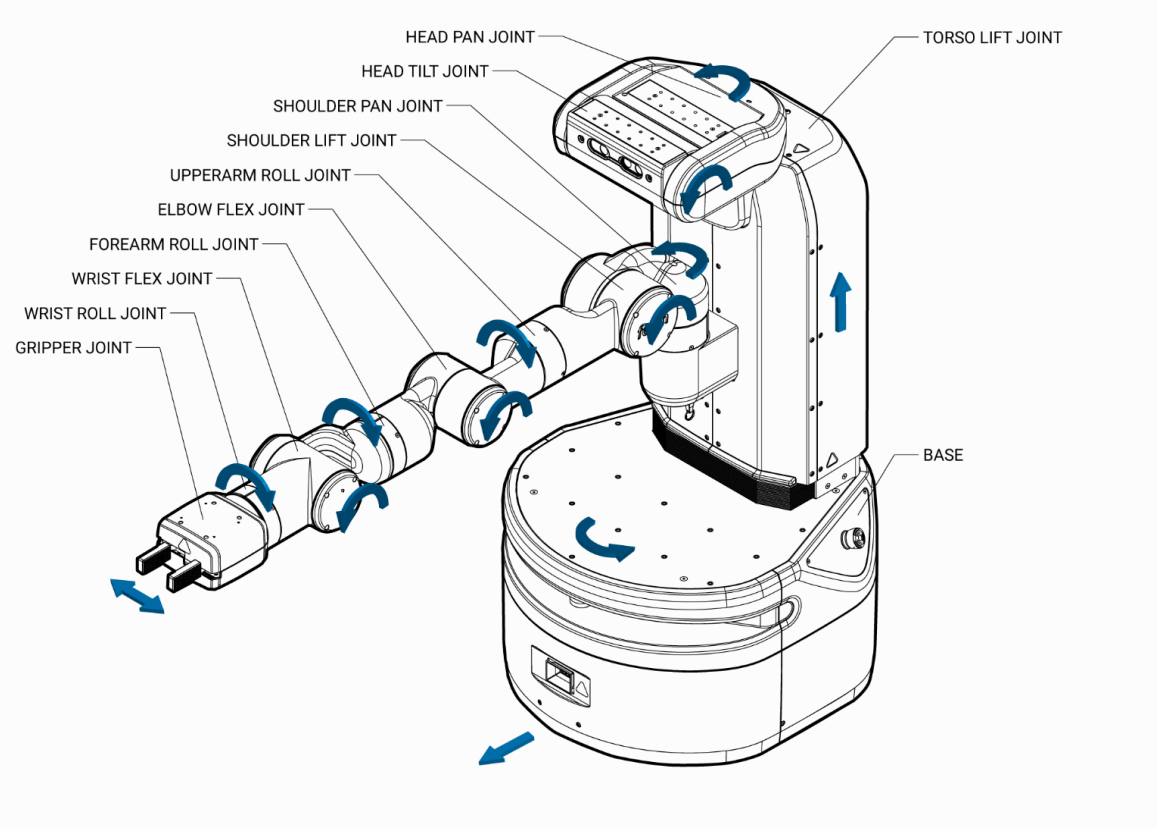
Topic descriptions



Twist message:

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

Topic descriptions

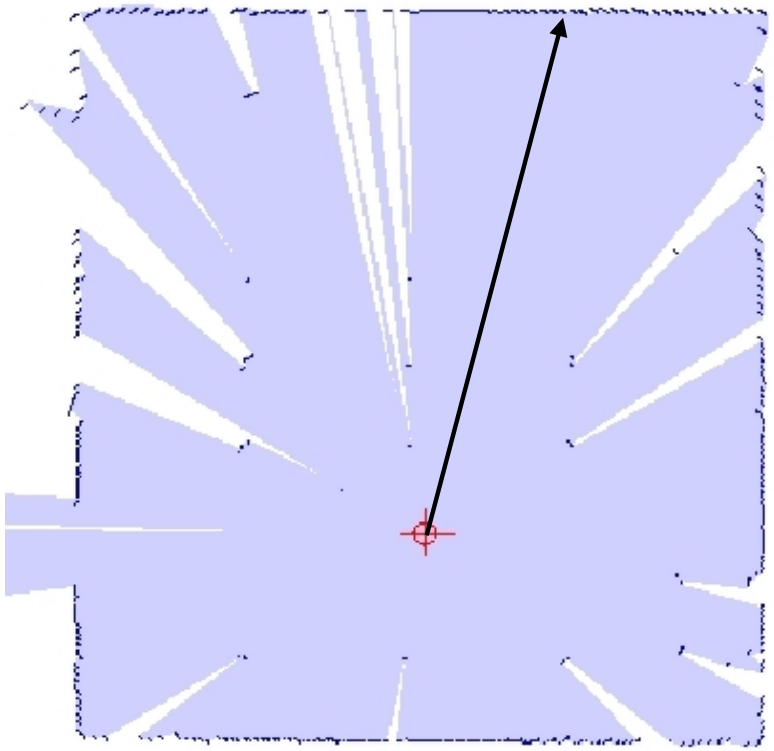


Joint State message:

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

an array entry for each joint

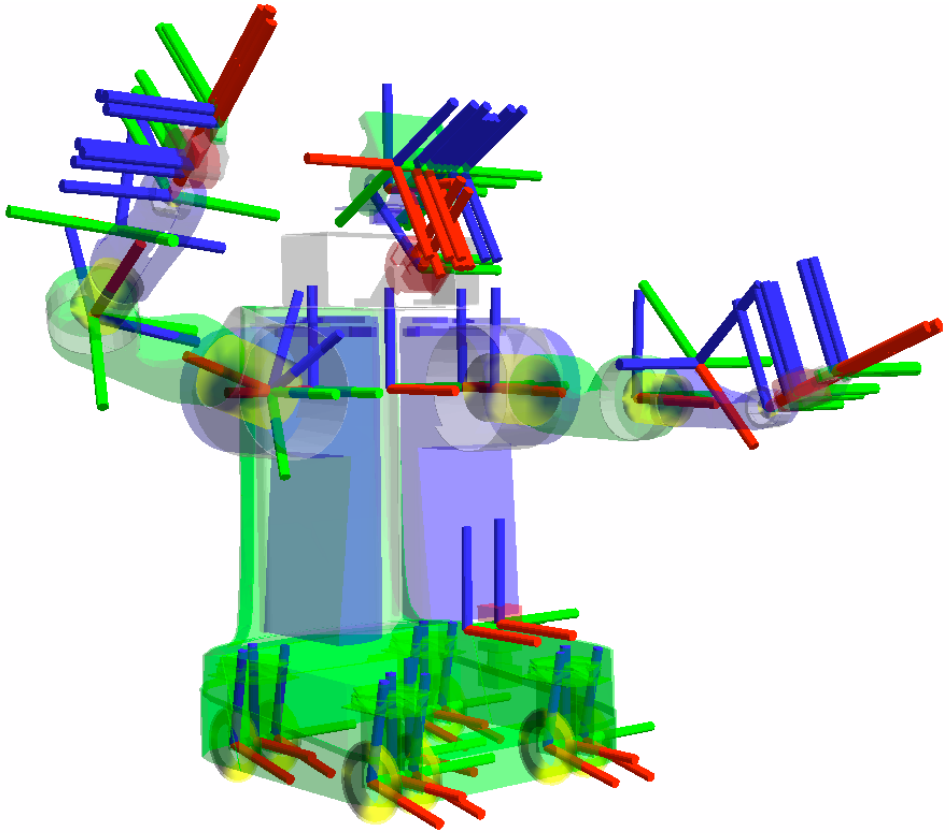
Topic descriptions



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

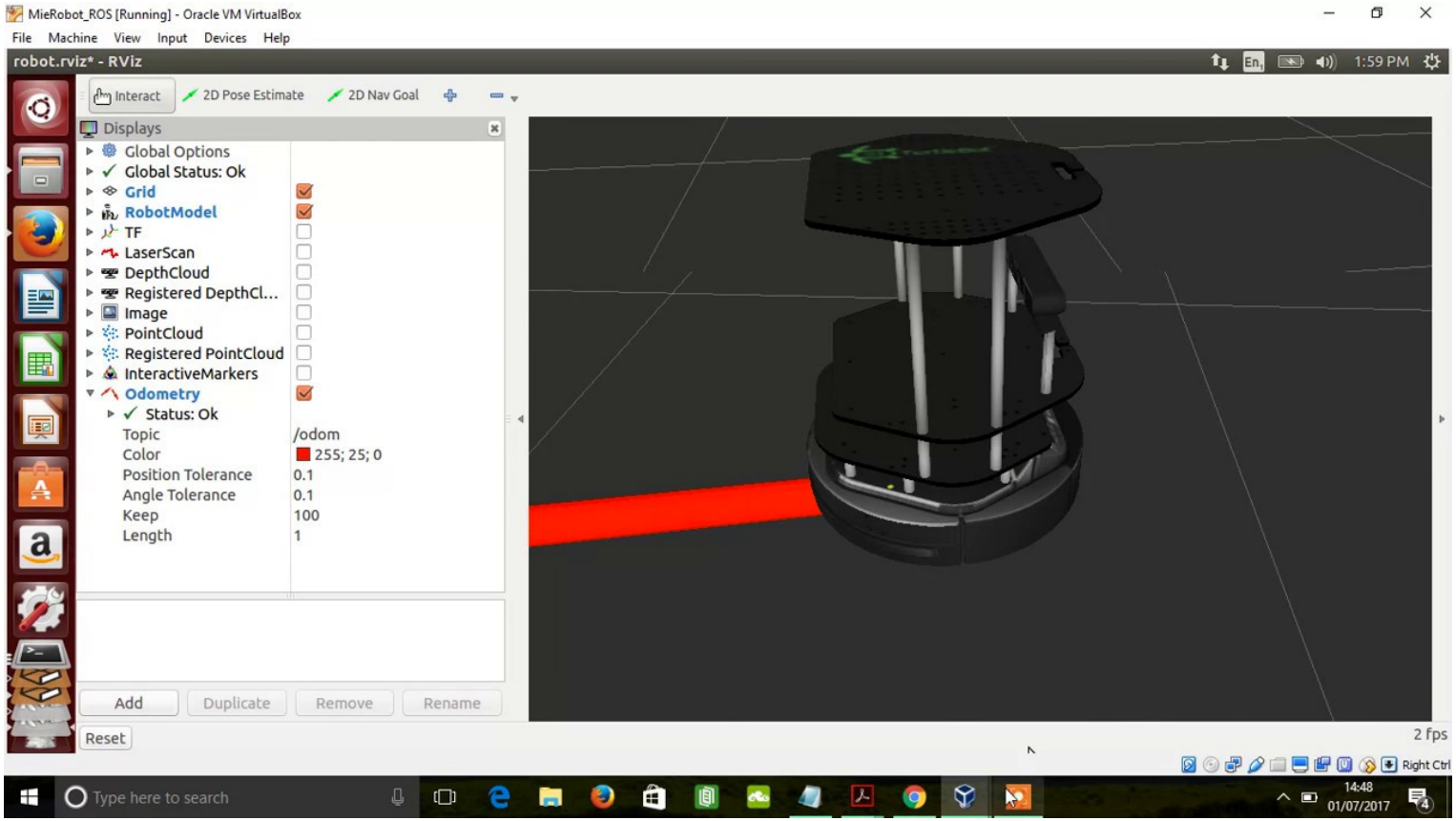
Topic descriptions



tf (transform) message:

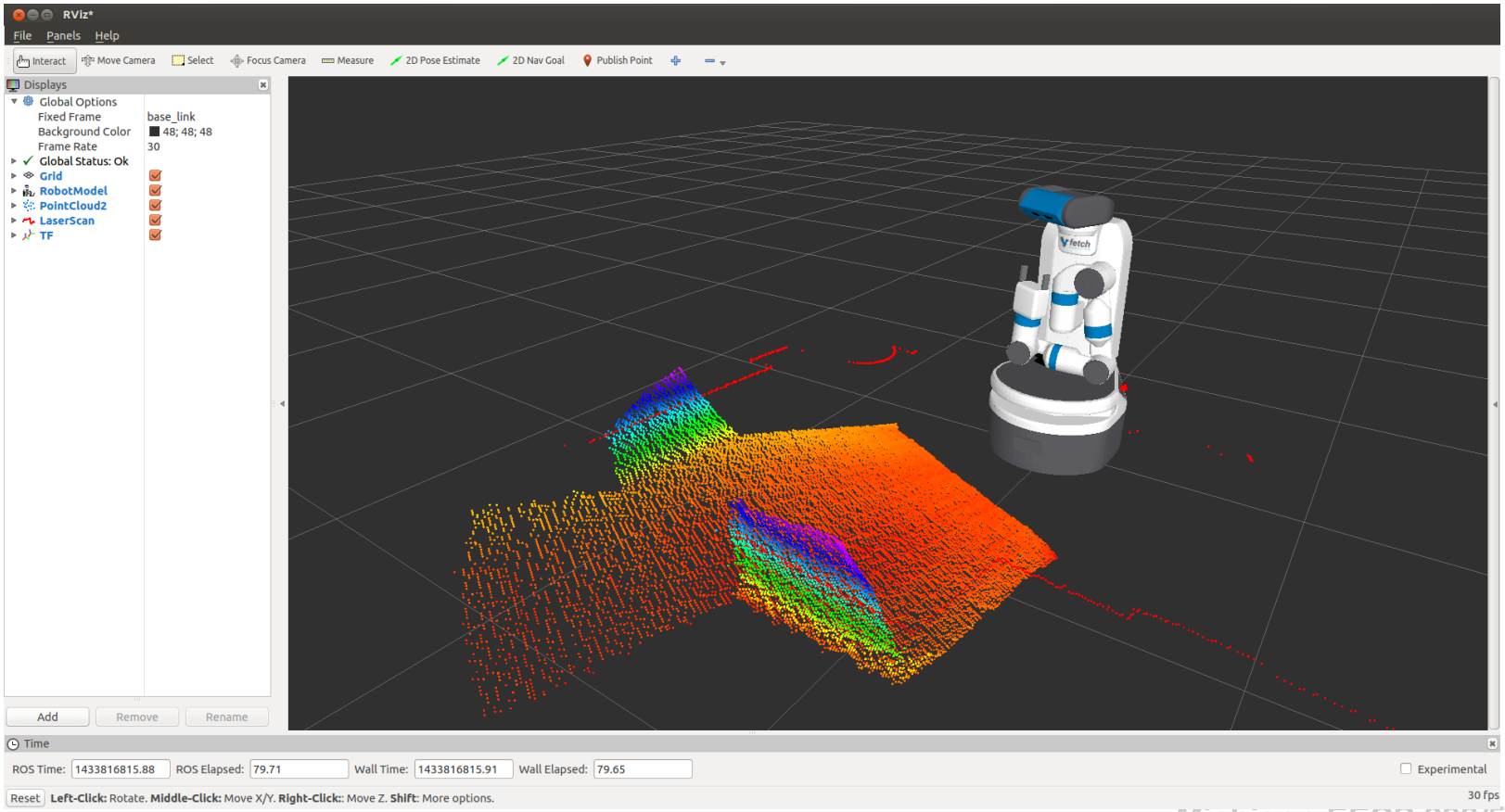
```
std_msgs/Header header
string child_frame_id
geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
        float64 x
        float64 y
        float64 z
    geometry_msgs/Quaternion rotation
        float64 x
        float64 y
        float64 z
        float64 w
```

Topic descriptions



Odometry

Topic descriptions

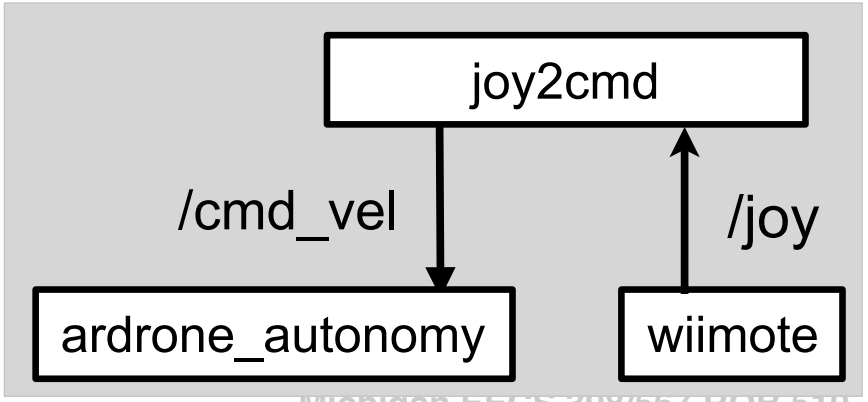


PointCloud2

Writing ROS nodes (warning: Diamondback!)

joy2cmd node

```python
#!/usr/bin/python
import rospy
from std_msgs.msg import String
from sensor_msgs.msg import Joy
from geometry_msgs.msg import Twist

twist = Twist()

def callback(data):
    print data.axes[0]
    twist.angular.z = data.axes[0]/5


def joy2cmd():
    rospy.init_node('joy2cmd')
    rospy.Subscriber("/joy", Joy, callback)
    pub = rospy.Publisher("/cmd_vel", Twist)
    twist.linear.x = 0;
    twist.linear.y = 0;
    twist.linear.z = 0;
    twist.angular.x = 0;
    twist.angular.y = 0;
    twist.angular.z = 0;

    #rospy.spin()
    while not rospy.is_shutdown():
        pub.publish(twist)
        rospy.sleep(0.001)

if __name__ == '__main__':
    joy2cmd();
```



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

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
```

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"

```
#!/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

publish control

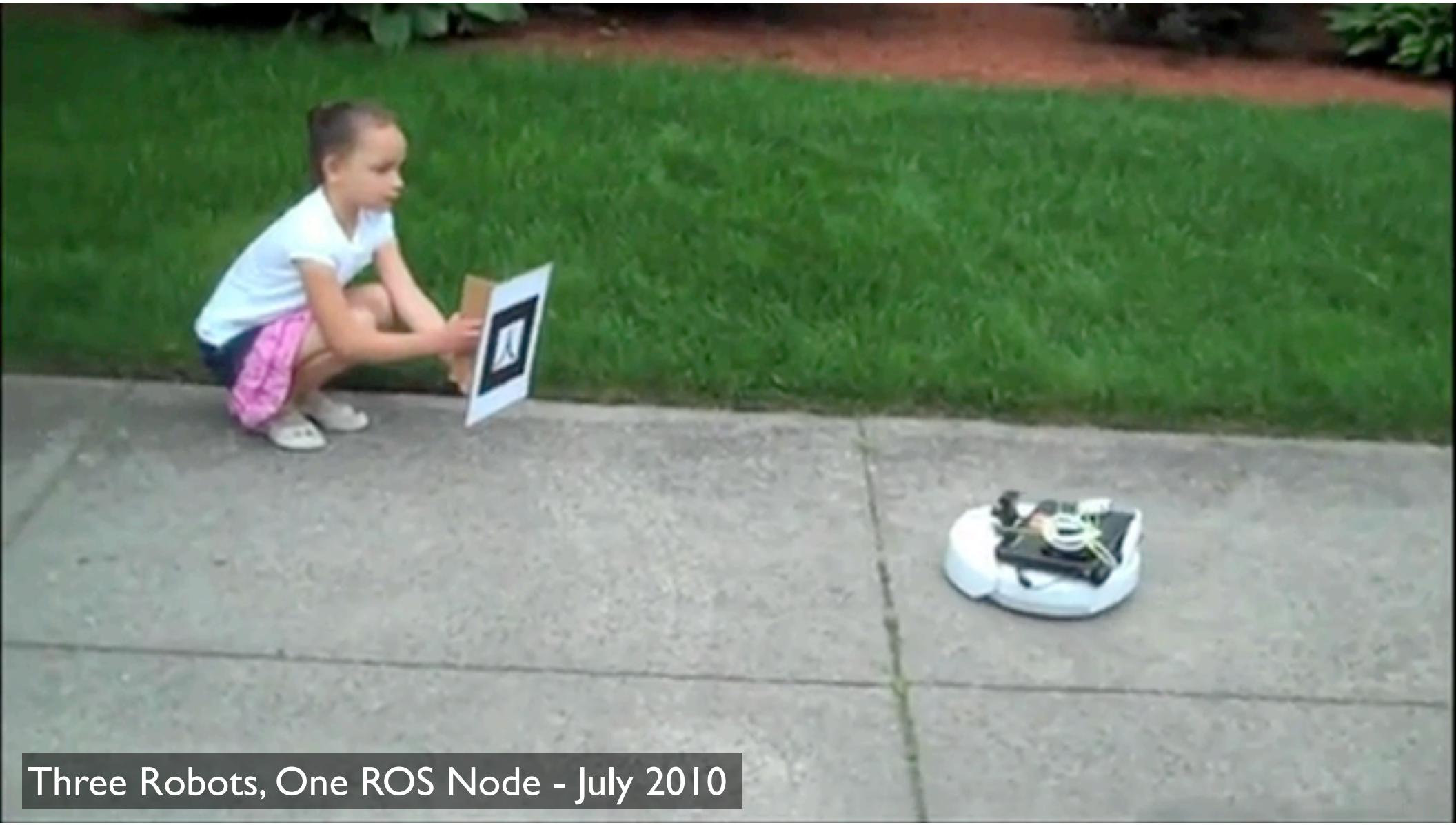
DO NOT USE large sleep value

event loop

```
#!/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"

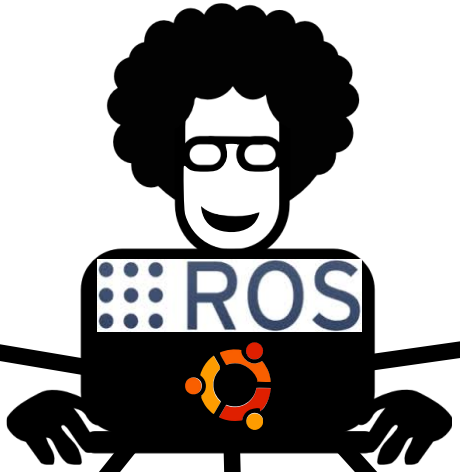
What behavior will this node produce?


Three Robots, One ROS Node - July 2010

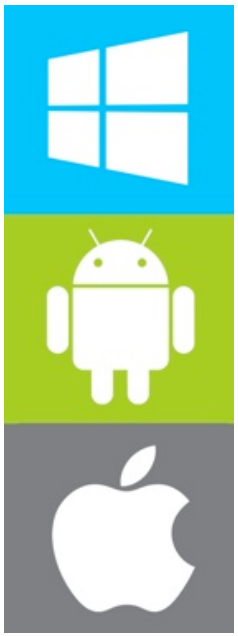
ardrone_brown first flight - Nov 2010

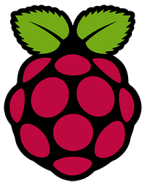
Heterogeneity


PR2

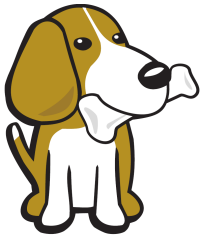

ROS

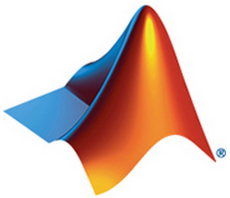

PR2

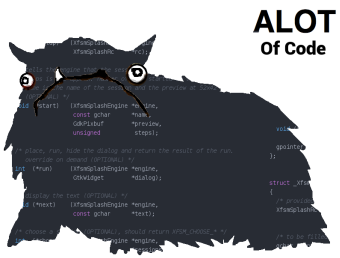


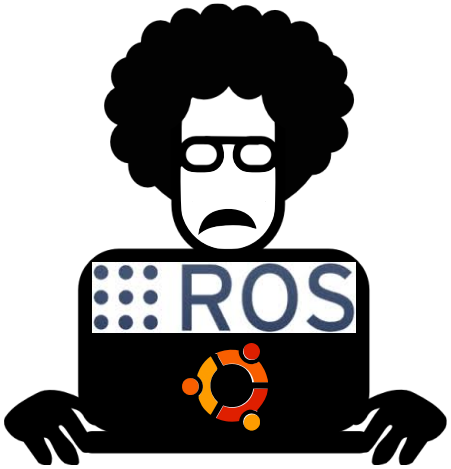



ARDUINO






ALOT
Of Code

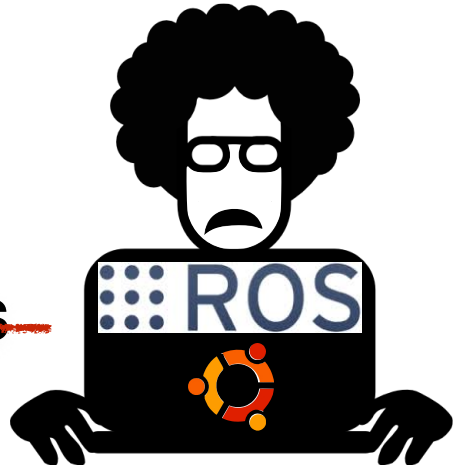

ROS

~~Different Compute Platforms~~

~~Different Programming Languages~~

~~Different Operating Systems~~

~~Custom non-ROS Codebases~~



~~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

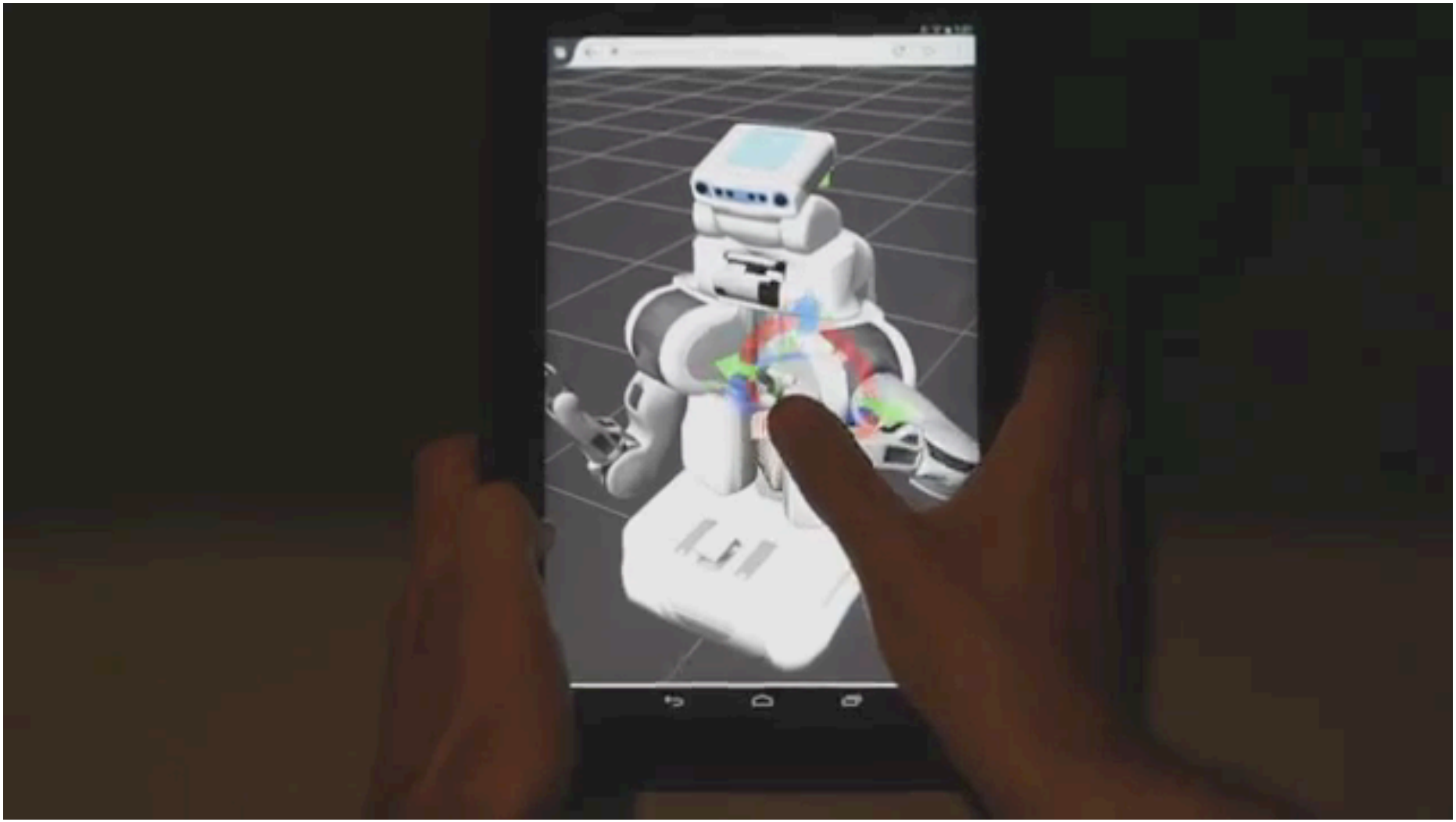
ROS is great... but...

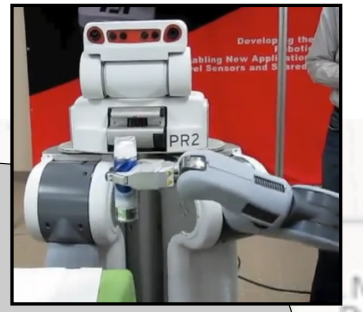
It affects the quality of science in robotics

- 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>
...
```

Open websocket connection

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

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

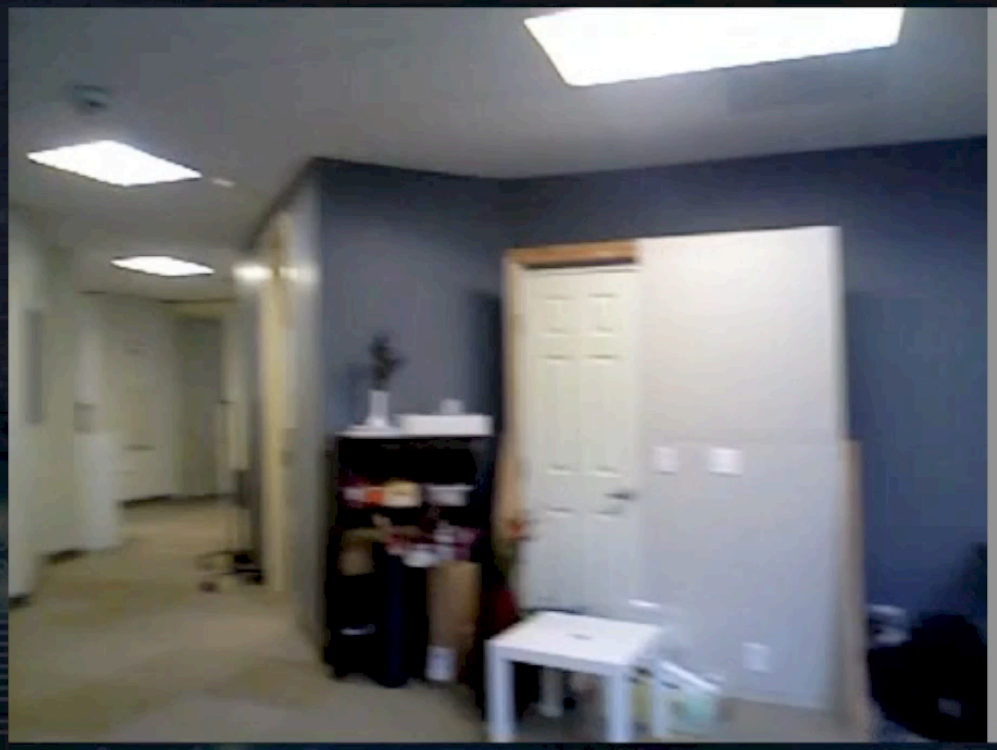
```
ros.publish('/cmd_vel','geometry_msgs/Twist','{"linear":{"x":'+x+',"y":0,"z":0},"angular":{"x":0,"y":0,"z":'+z+'}}');
...
```

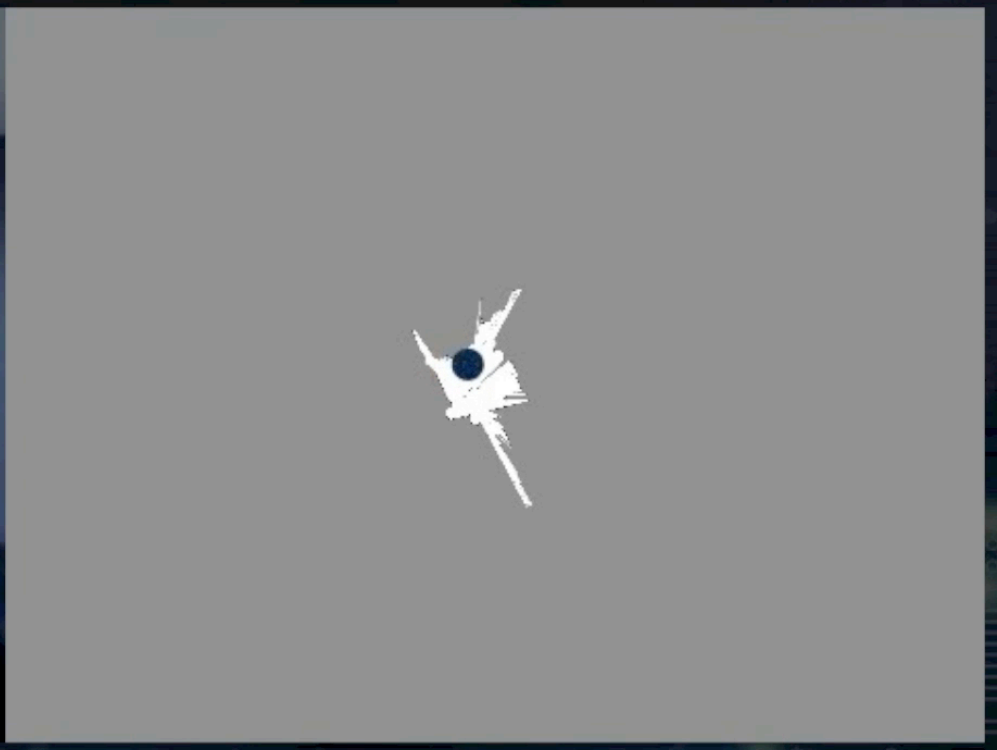
Return 128x96 jpeg array at 90% quality at ROS frame rate

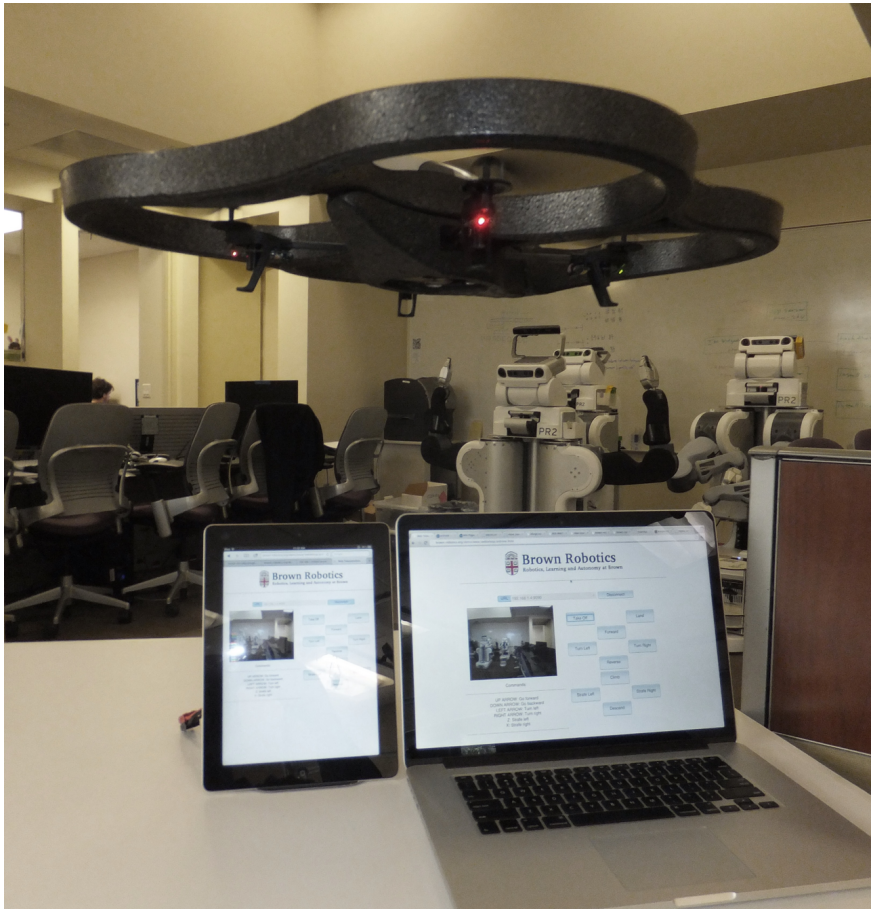
```
ros.callService('/rosjs/subscribe',json(['/gscam/image_raw',0,'jpeg',128,96,90]),nop);
```

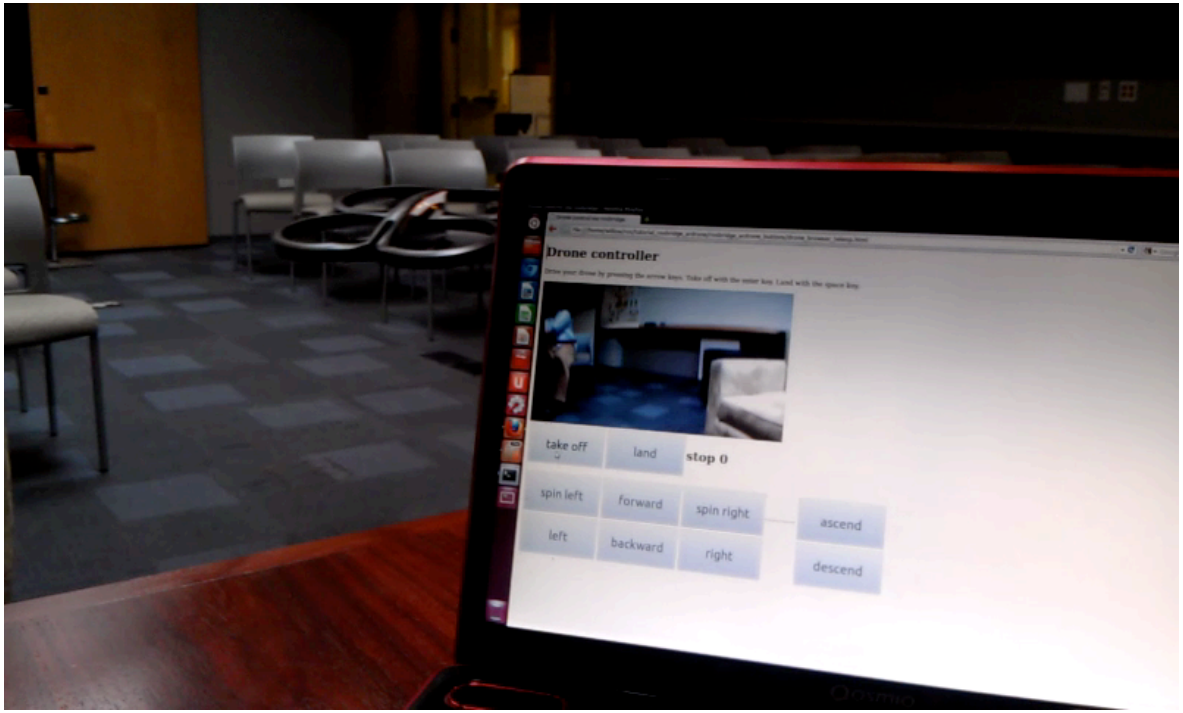
Map Builder

Speed: 90%

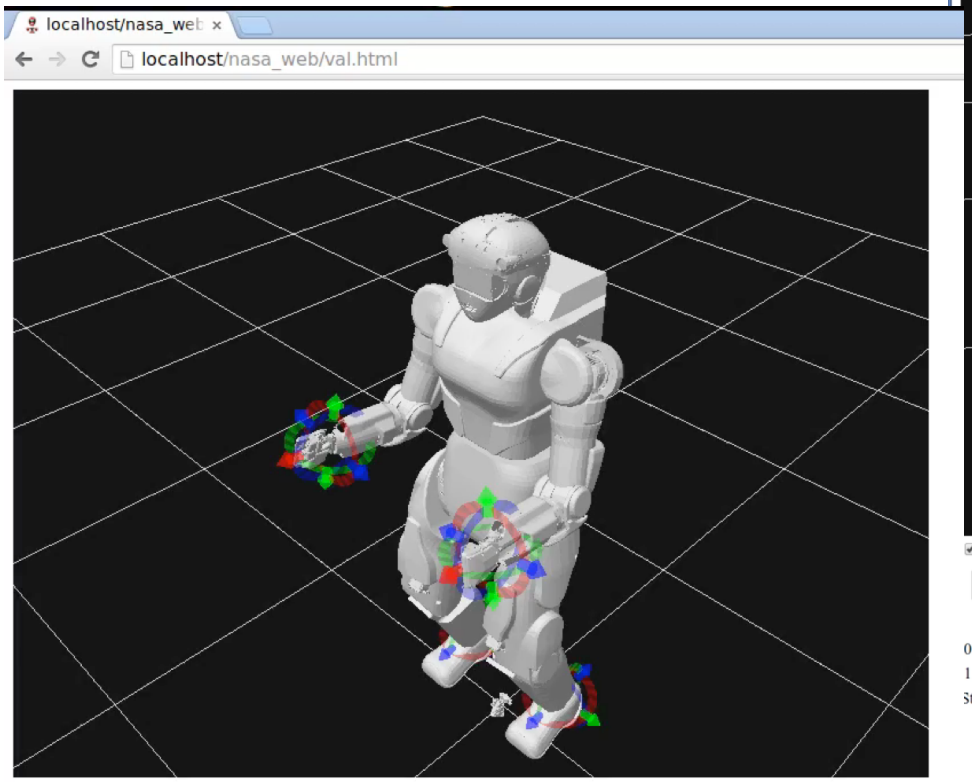


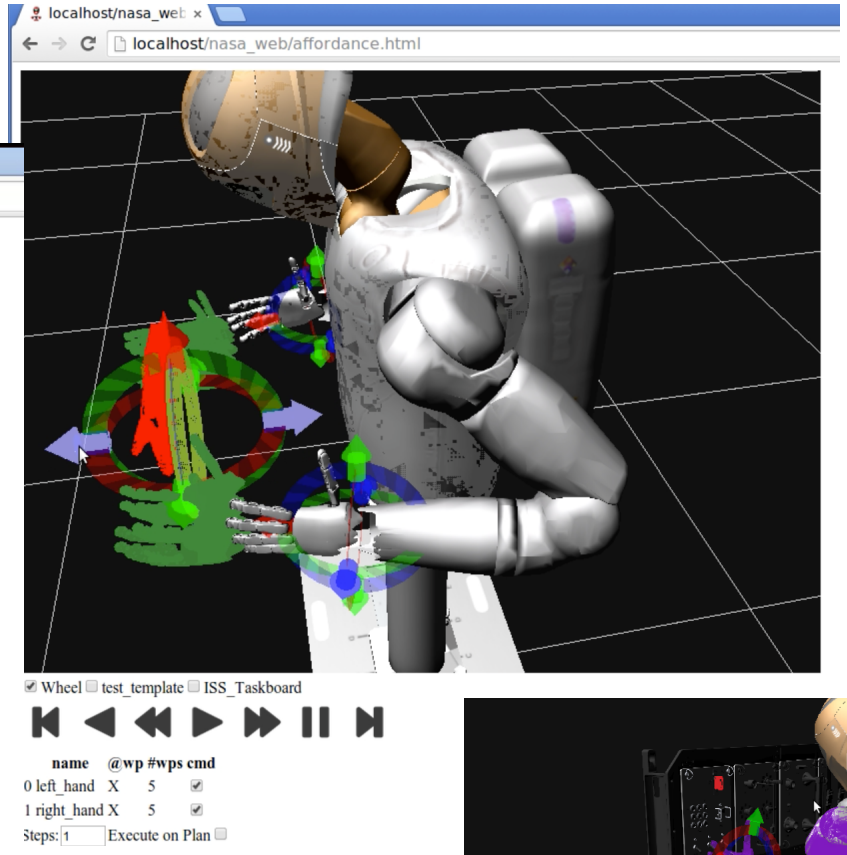


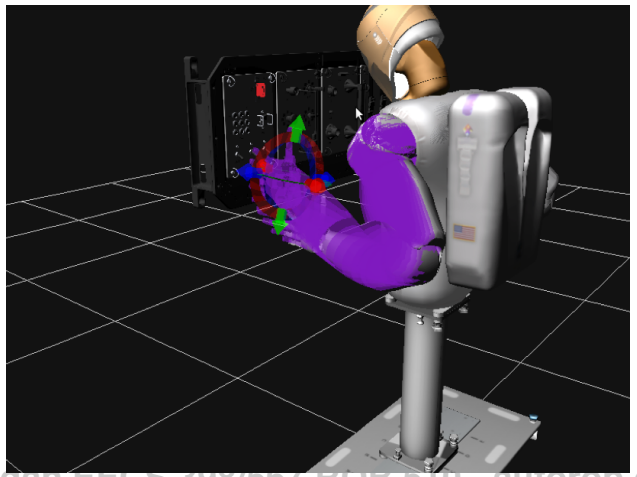

Brown Robotics
Brown Robotics


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

[Kemp et al. 2012]







NASA Valkyrie and NASA Robonaut 2

[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

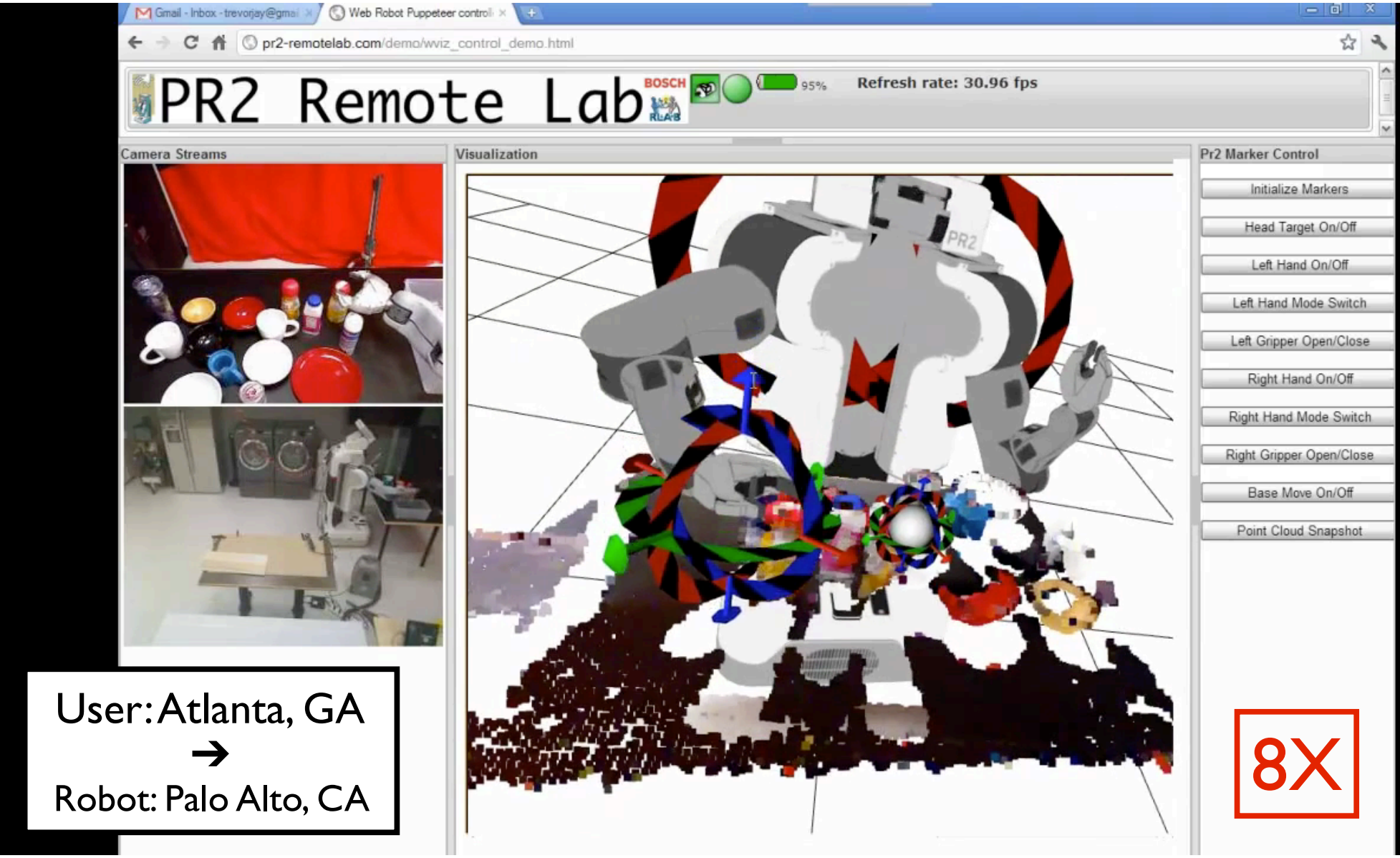
rosbridge_suite

Python ★ 37 ⑂ 67

Server Implementations of the rosbridge v2 Protocol

People 7 >

Invite someone



User: Atlanta, GA

→

Robot: Palo Alto, CA

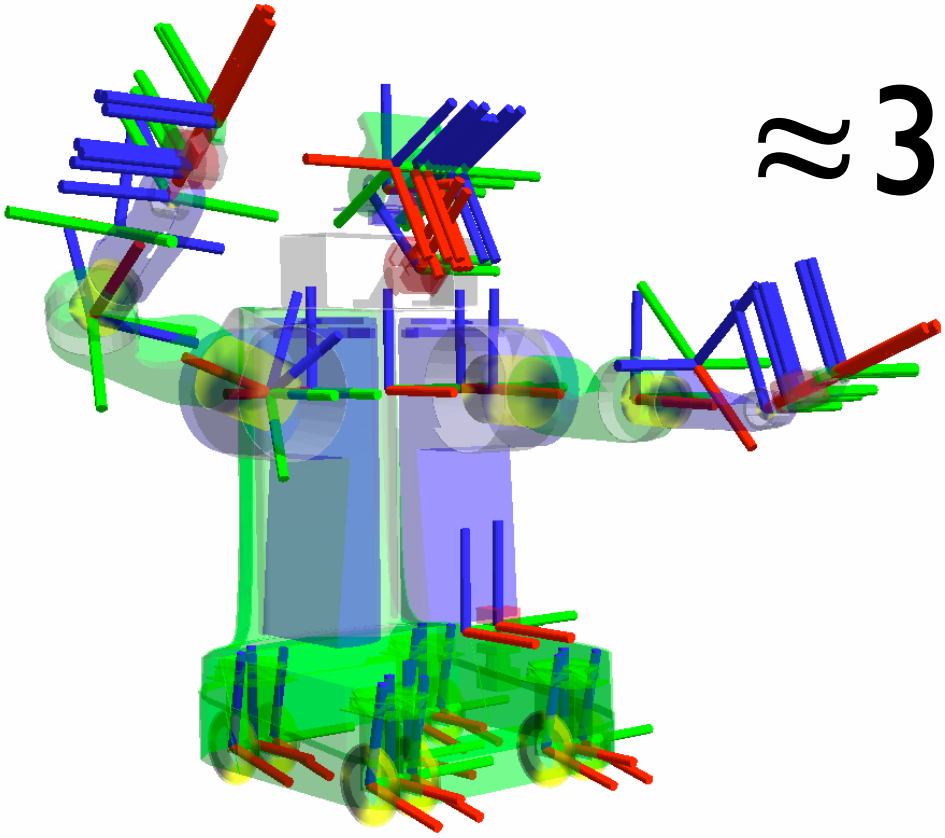
8X

How much bandwidth?

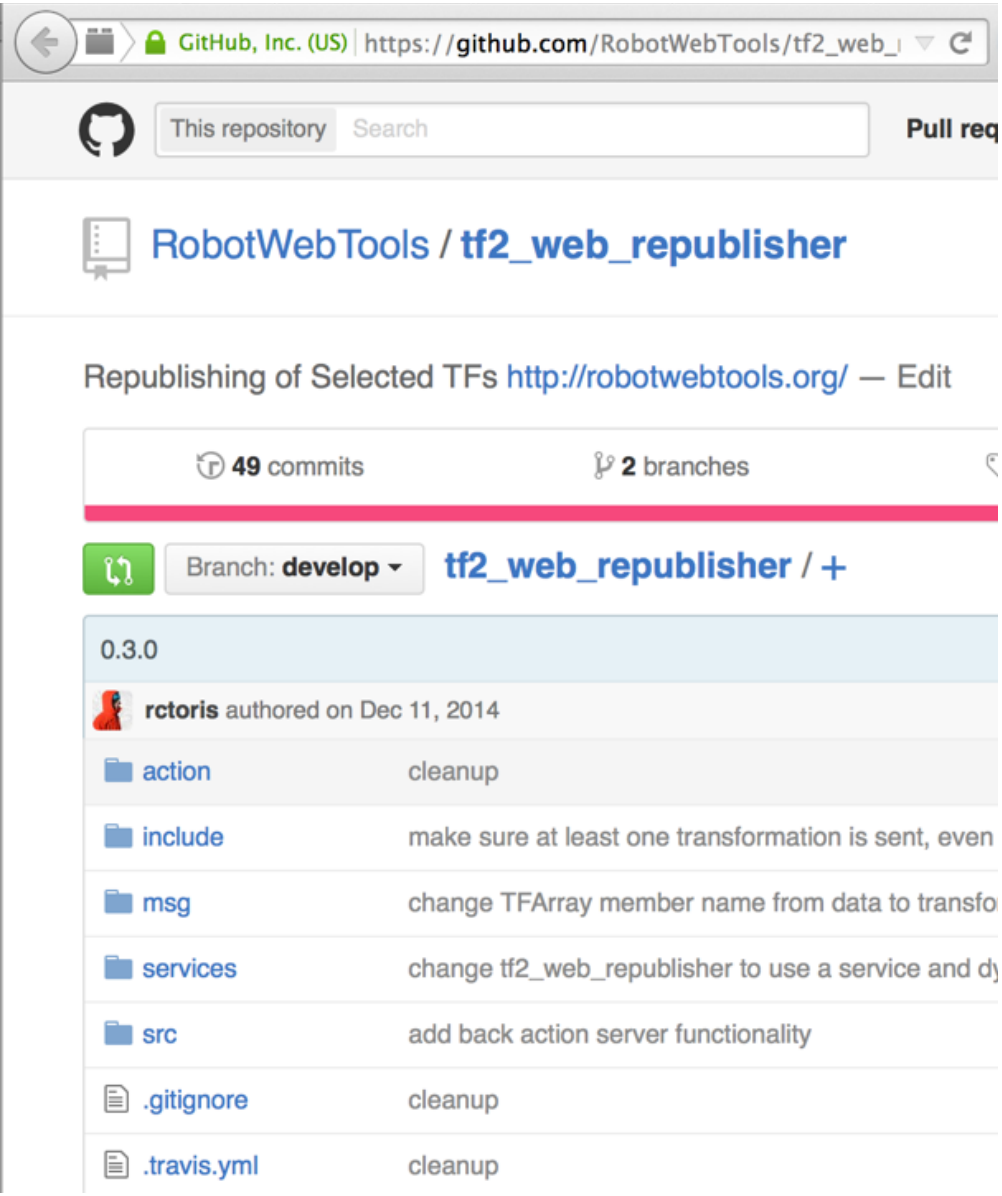
Problematic Topics for Streaming

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

Messaging tf trees



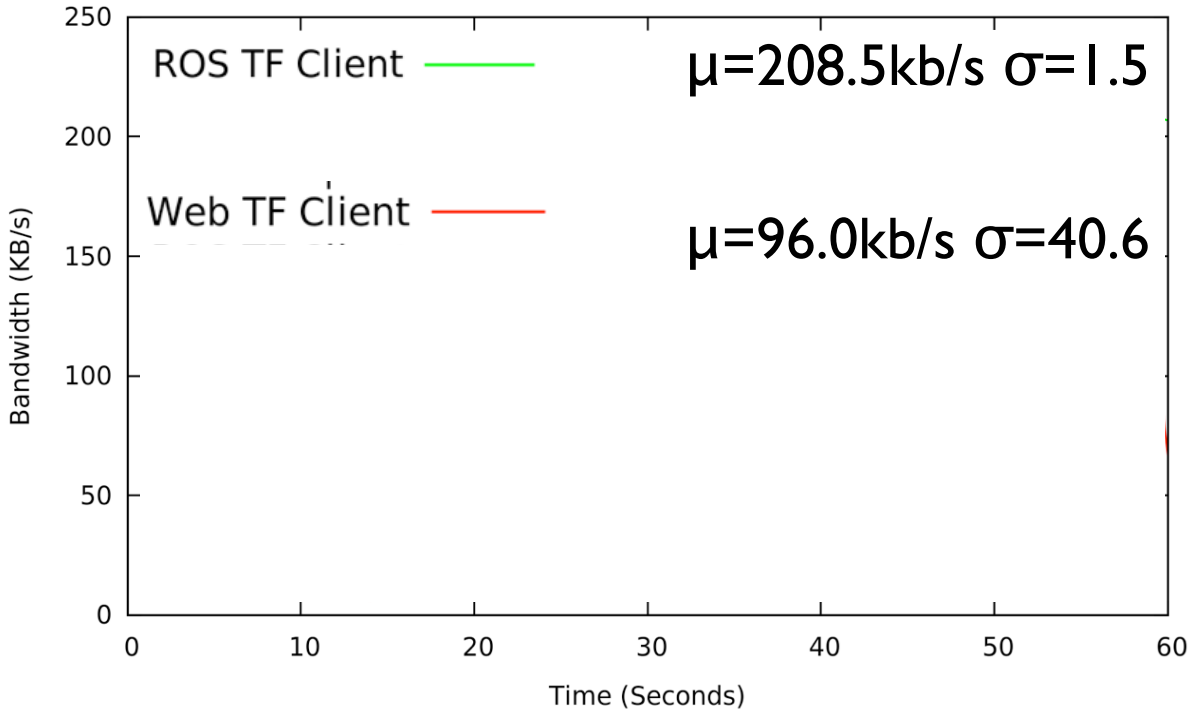
≈3MB/s



tf2_web_republisher

Only send tf message when state changes

WPI CARL mobile manipulator
52 tf frames



rosbridge client subscription
10Hz throttle
Delta thresholds:
- 0.01 meters
- 0.01 radians

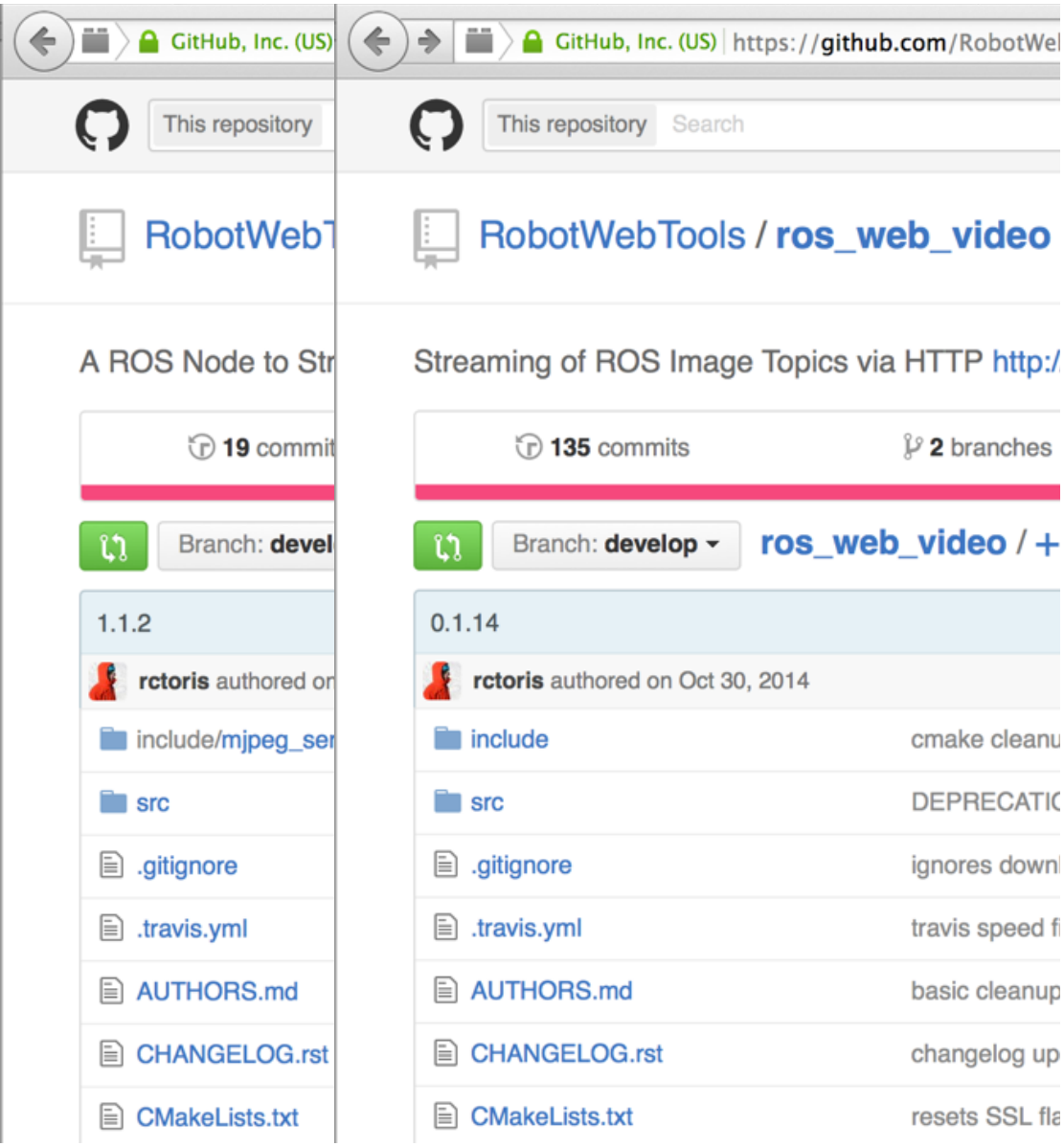
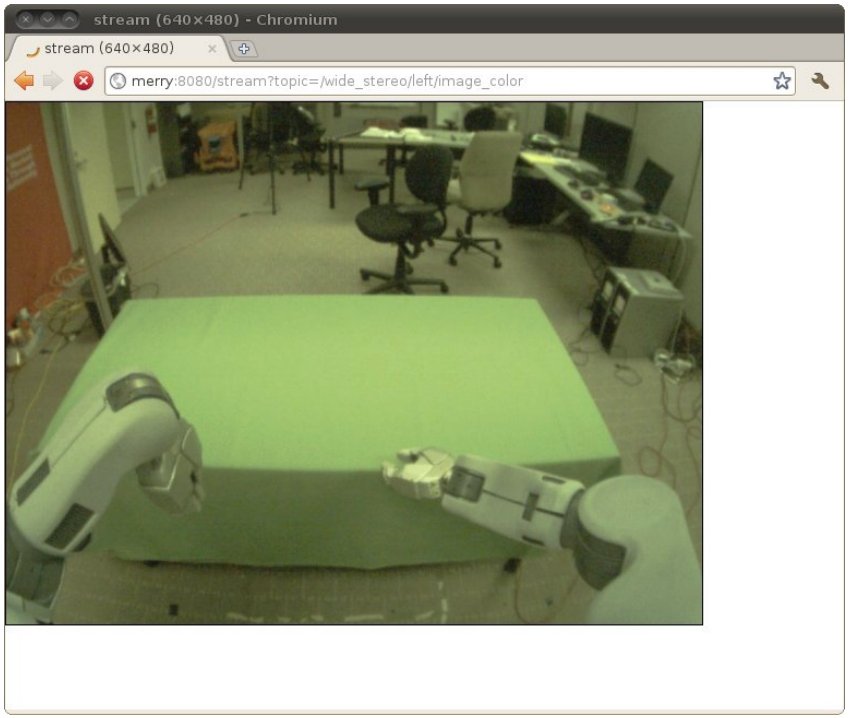


Image Streaming

- ROS Internal
  - Raw/uncompressed
  - JPEG|PNG compression
  - Theora codec
- mjpeg_server (Ben Pitzer)
  - include with <img> HTML5 tag
- VP8 codec (WebM)
  - include with <video> HTML5 tag



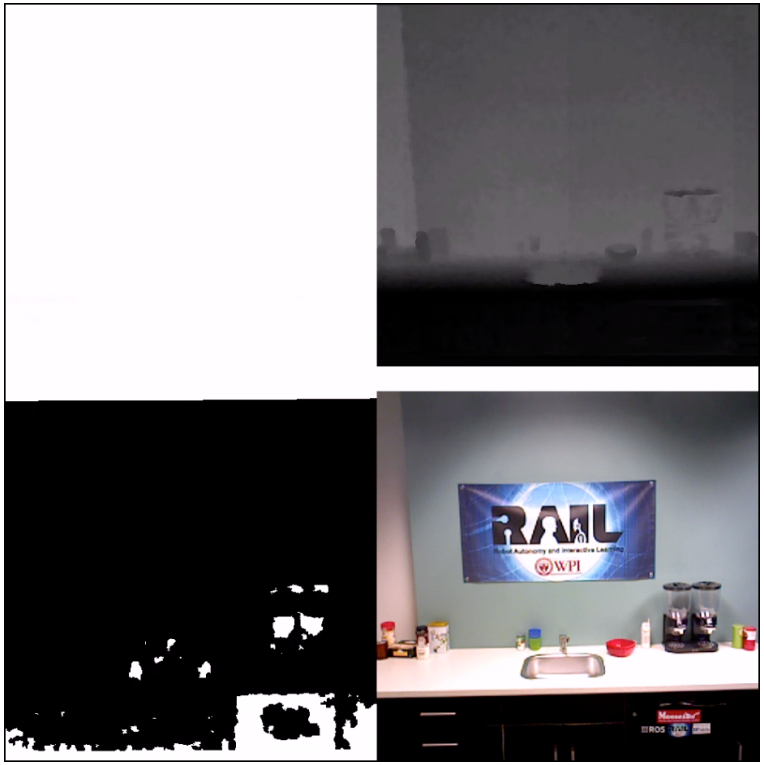
	Web		ROS Internal		
	MJPEG	VP8	RAW	Compressed	Theora
μ	1317.0	342.2	12052.5	1309.3	19.4
σ	11.1	18.3	1.7	11.6	8.9

2 minute subscription of
640x480 images
30Hz streaming

MJPEG at 90% quality
VP8 100K default bitrate

VP8 point cloud compression

Depth
0-3m



Depth
3-6m

Validity
mask

RGB
image

Point cloud reconstructed
in browser

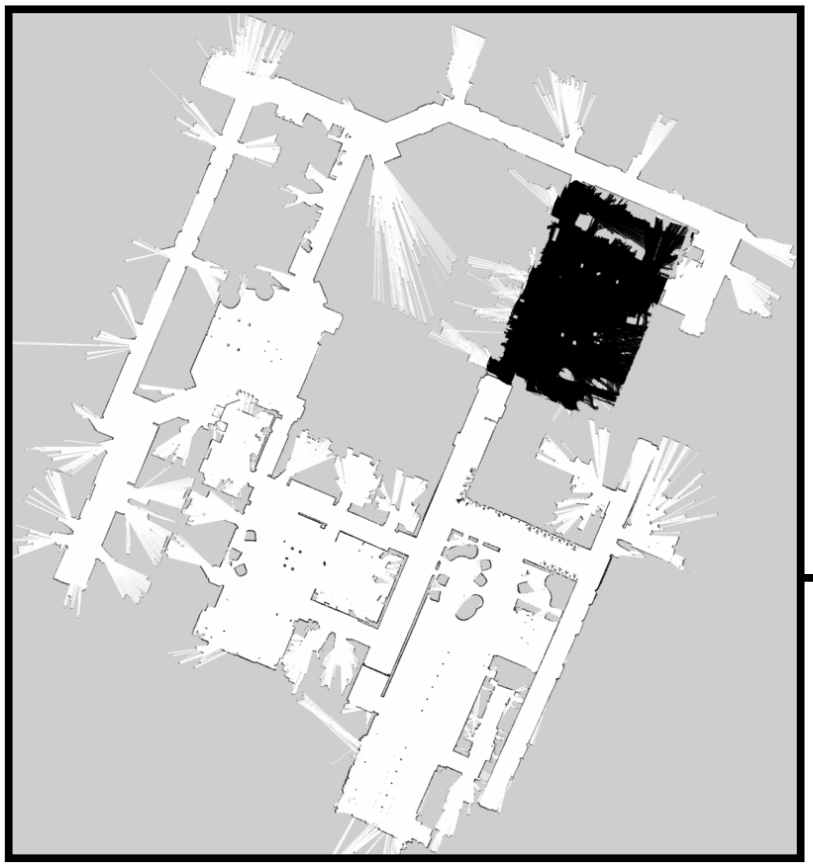
Create single image on robot, compress with VP8

2 minute subscription from Asus Xtion Pro

	ROS Internal	Web Streaming
μ	5591.6	568.4
σ	70.5	133.5

Typically about 10% in size

Generic message PNG compression



Cast raw bytes of JSON message as square RGB image

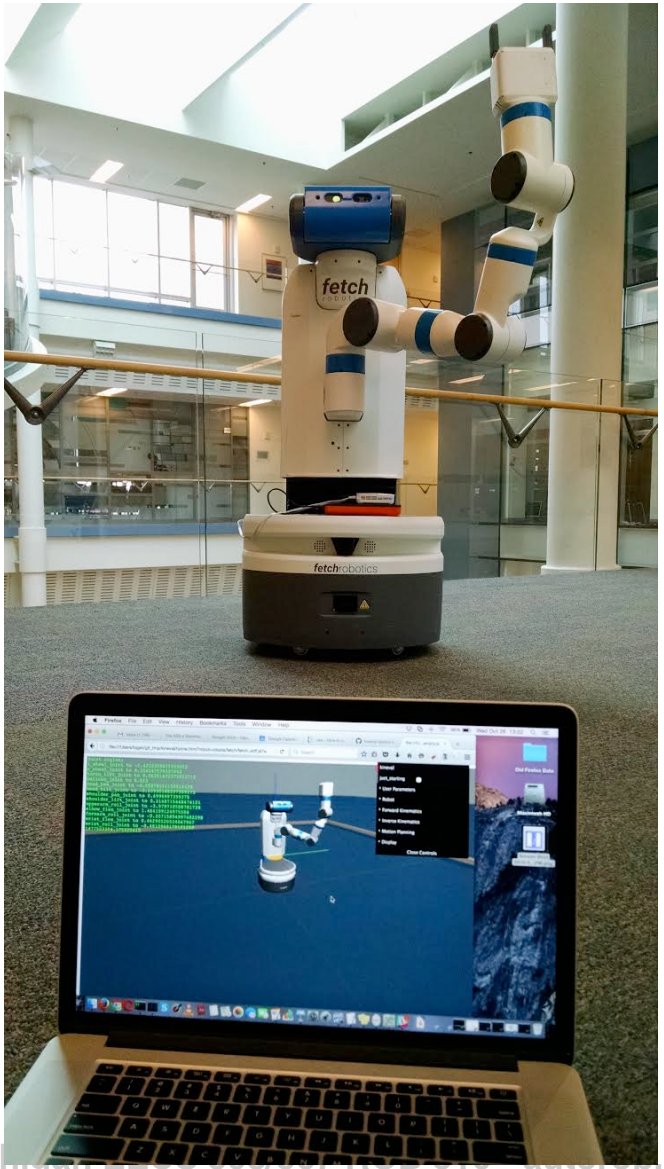
Fill empty pixels with null value

Compress and decompress as PNG image

	Original	Compressed	Time
Map (732x413)	2725	39	40.0
3D Point Array Marker	42.1	5.1	0.8
3D Line Strip Marker	42.1	4.8	0.9
3D Line List Marker	78.1	7.7	1.0
	(kb)	(kb)	(s)

Typically less than 30% in size

KinEval and *rosbridge*



KinEval and *rosbridge*

- Connect to mesh radio access point (FETCH 2)
- 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