Robot Operating System (ROS)

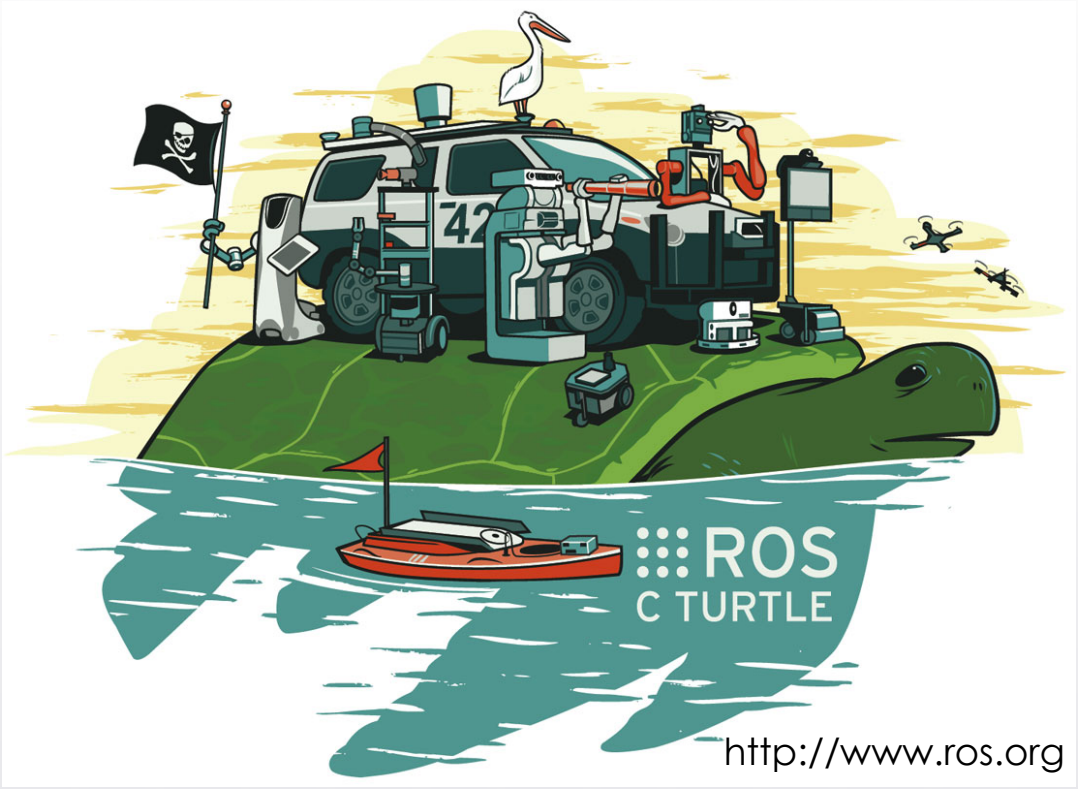
NICOLA BEZZO, MASOUD BASHIRI

Fall 2016

ROS is a flexible framework for writing robot software

Willow Garage

Open Source Robotics Foundation



- 100+ ROBOTS
- 100 SENSORS/ INTERFACES
- 2000+ LIBRARIES

CODE REUSABILITY

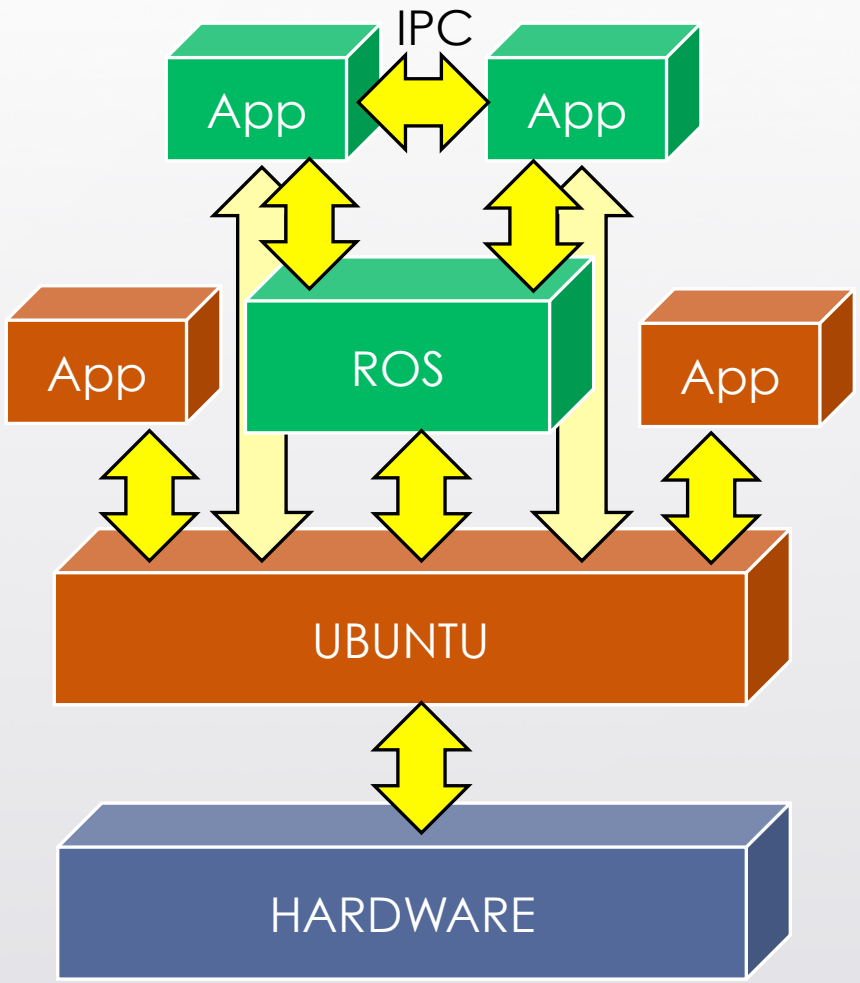

How Robotics Research Keeps...
Re-Inventing the Wheel
First, someone publishes...
...a paper with a proof-of-concept robot.
This prompts another lab to try to build on this result...
...but they can't get any details on the software used to make it work...
BARF!
...and all the code used by previous lab members is a mess.
So, a grandiose plan is formed to write a new software API...
...and countless sleepless nights are spent writing code from scratch.
But inevitably, time runs out...
...and they write code that barely works but lets them publish...

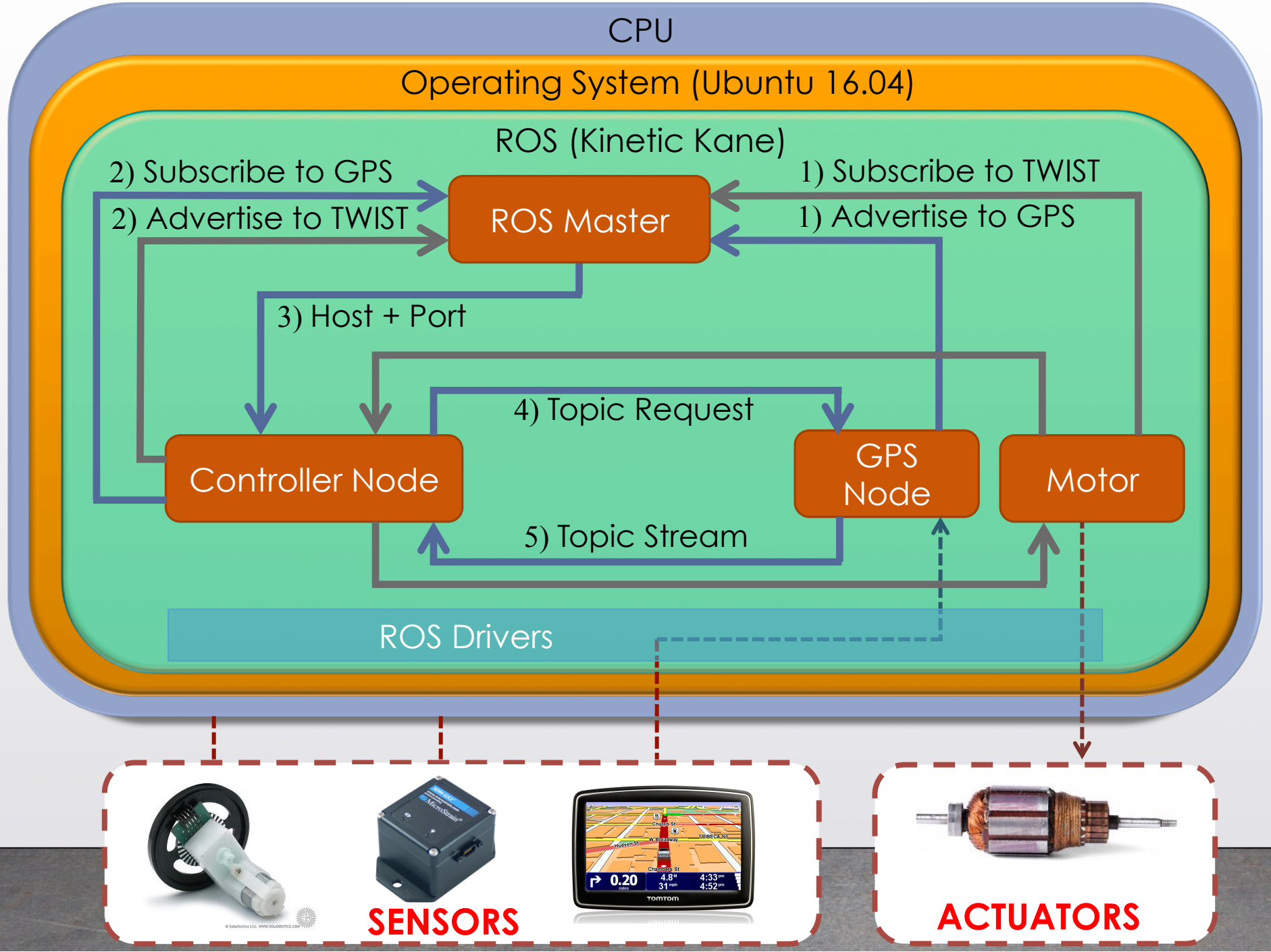
What is ROS?

- A meta Operating System for Robots. As an OS, provides:
  - low-level device control
  - implementation of commonly-used functionality
  - message-passing between processes (they're called nodes)
  - package management

- Tools and libraries for development and running code across machines
  - Supports C++, Python and Lisp
    - experimental libraries for Java

- ROS is not an actual OS! it is a middleware. It sits on top of an OS (Linux and Mac are officially supported).





- PUB/SUB ARCHITECTURE

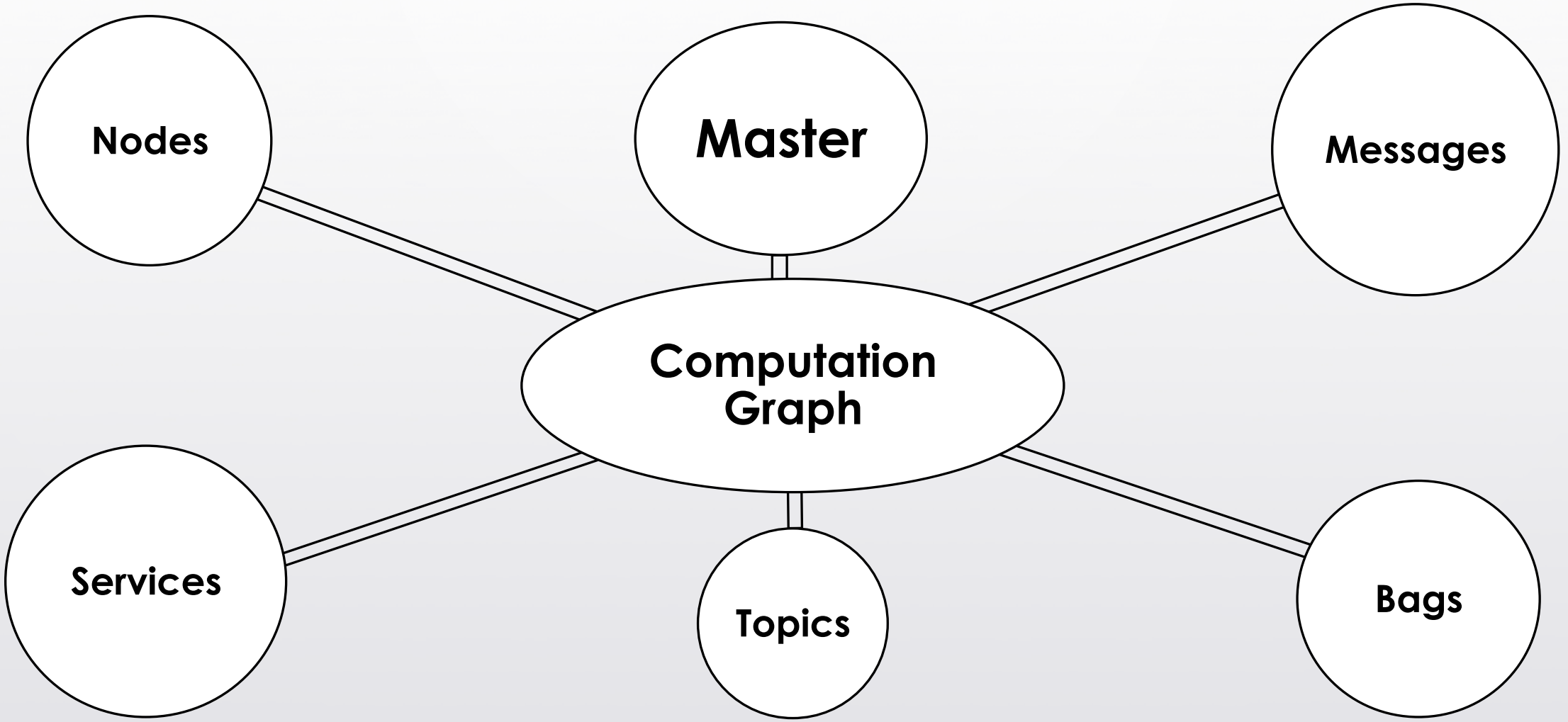
ROS Computation Graph

- **Master**
  - Provides name registration and look up to the nodes (Like a DNS Server).
- **Nodes**
  - Are executable that use ROS to communicate with other nodes.
  - A ROS node is written with the use of a ROS client library, such as roscpp or rospy.
- **Messages**
  - A ROS message is a data structure, comprising typed fields. Nodes communicate with each other by passing messages. (Structures defined in .msg files)
- **Topics**
  - Each topic is a message bus that carries a certain type of message. Multiple nodes can **subscribe** and **publish** to a single topic. Also one node can **subscribe** and **publish** to many topics.

ROS Computation Graph (Cont'd)

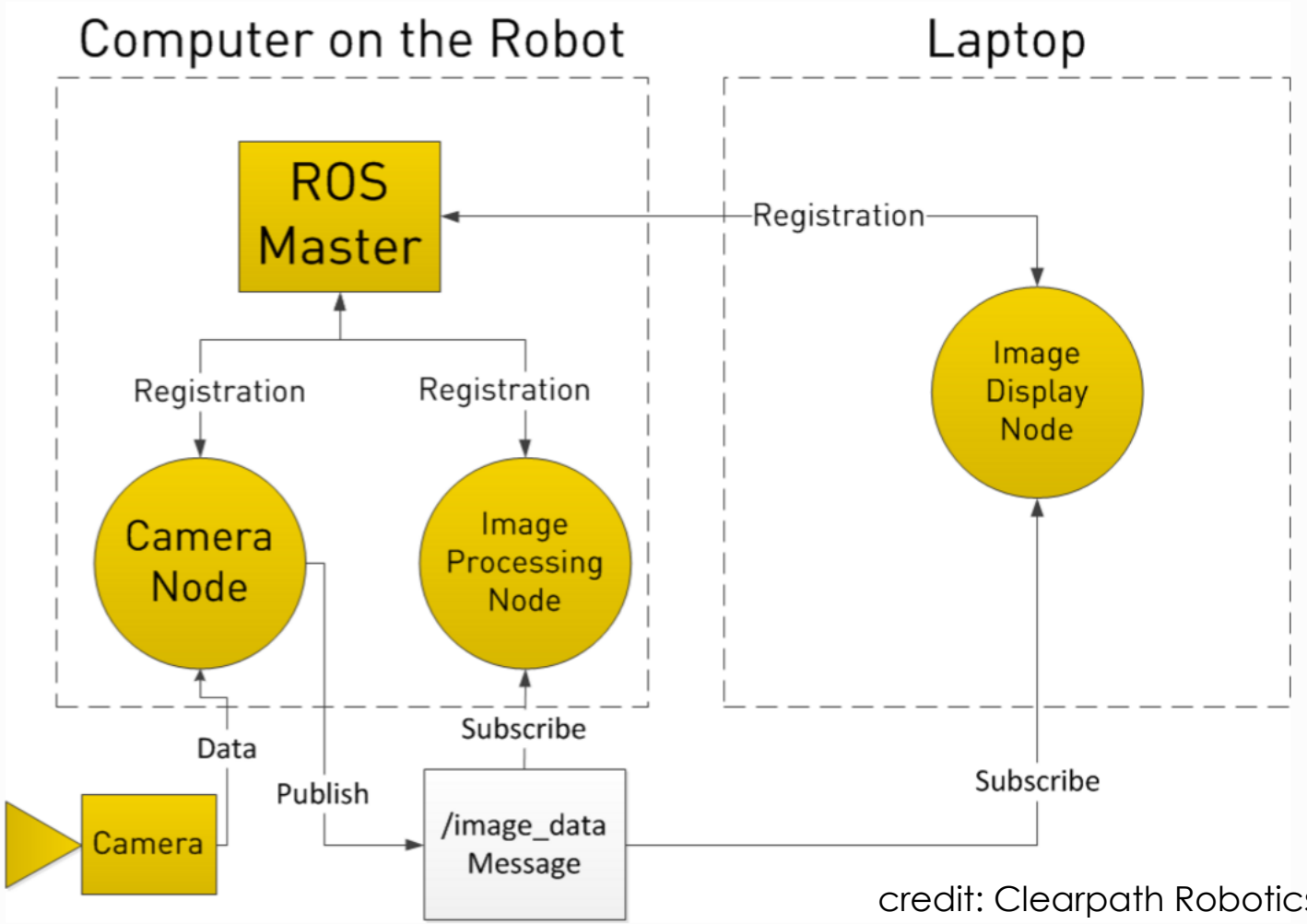
- **Services**
  - ROS Services complements the subscriber/publisher communication model (Topics), by adding a synchronous request/reply model.
  - A service provider node offers a service under a name and a client node uses the service by sending the request message and awaiting the reply.

- **Bags**
  - Bags are a format for saving and playing back ROS message data.
  - Example: Storing sensor data to be used later.


Nodes
Master
Messages
Computation Graph
Services
Topics
Bags

Example: Topics

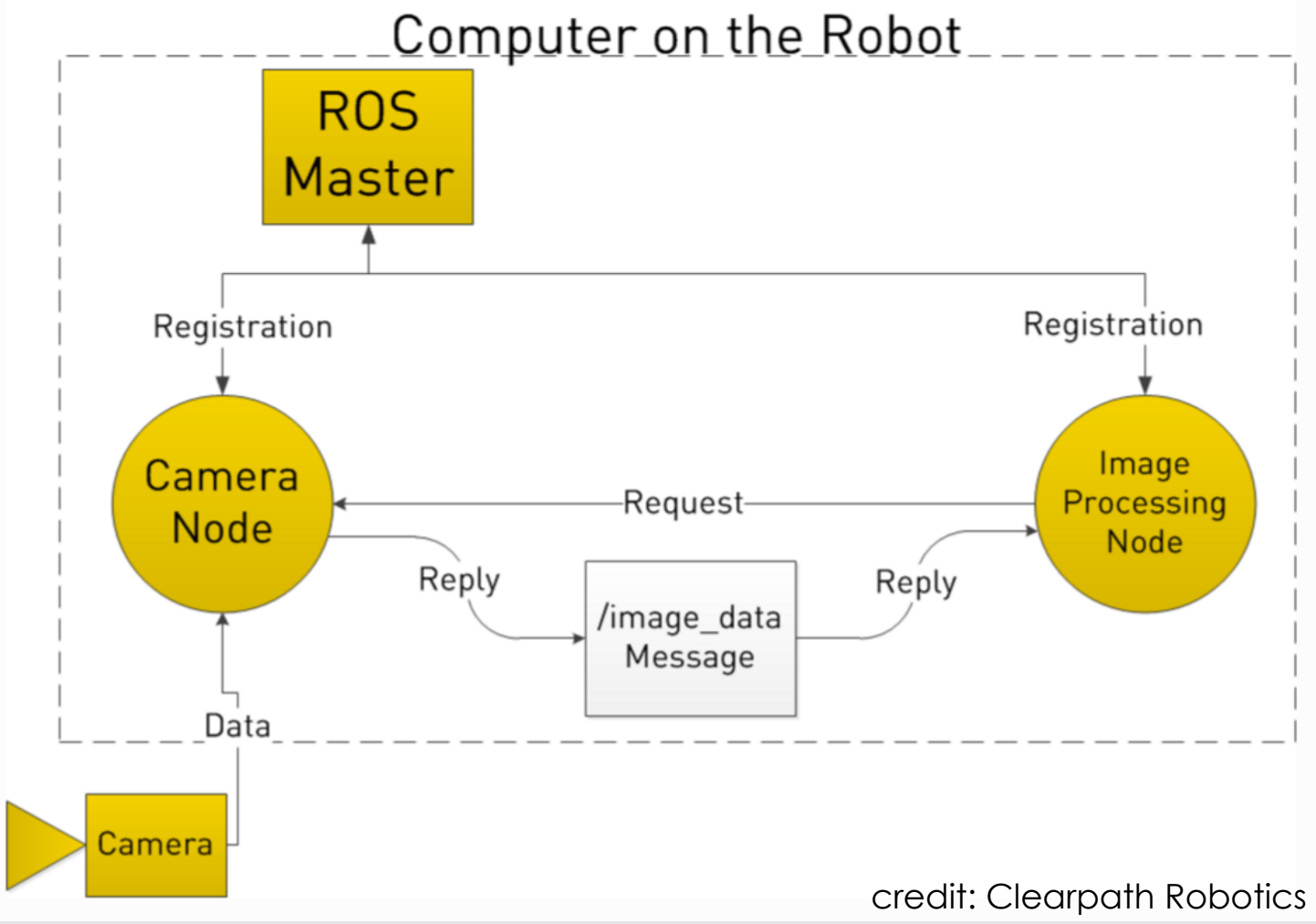
- Asynchronous "Stream-like" Communication
- TCP/IP or UDP Transport
- Strongly typed (.msg files), supports primitive datatypes and arrays
- Multiple Publishers
- Multiple Subscribers



credit: Clearpath Robotics

Example: Services

- Synchronous "RPC-like" Communication
- TCP/IP or UDP Transport
- Strongly typed (.srv spec)
- One Server
- One or many clients



credit: Clearpath Robotics

OPEN YOUR BOX

Rules

1. Your group number is the number that you find on the Turtlebot and computers that you have received.
2. Assign a name to your turtlebot: that's your group's name
3. Every time there is a Lab (usually on Thursdays) you (at least a member from your group) must pick up your robot from Rice 240 and bring it down to Rice 120. Viceversa, at the end of each class, you'll have to bring the equipment back to Rice 240.
4. Please remember to charge your turtlebot and laptops before each lab session to avoid delays during the lab.
5. You are allowed to use only your assigned robot and computers (i.e., you cannot use other groups equipment)
6. Three dedicated networks are available to connect laptops and turtlebots. The networks SSID are AMR 1, 2, and 3. These networks are up and running in Rice 240. Please don't remove the routers from the room.
7. You are allowed to use the equipment at any time during the semester.
8. The equipment can be used only inside Rice Hall, preferably inside/around Rice 240

Communication

1)

2)

Connect to your TurtleBot

1. Turn on your turtlebot laptop and connect to a network as follows
   - 1~3 connect to AMR1,
   - 4~6 connect to AMR2,
   - 7~10 connect to AMR3,

   } network password is **turtlebot**
2. Turn on your workstation laptop (password=username) and connect to **the same network**
3. On the **turtlebot laptop**, open a terminal and type in **ifconfig** to find your IP. We will refer to this as IP_OF_TURTLEBOT
4. On the **workstation laptop**, open a terminal and type in **ifconfig** to find your IP. We will refer to this as IP_OF_PC

Network Configuration

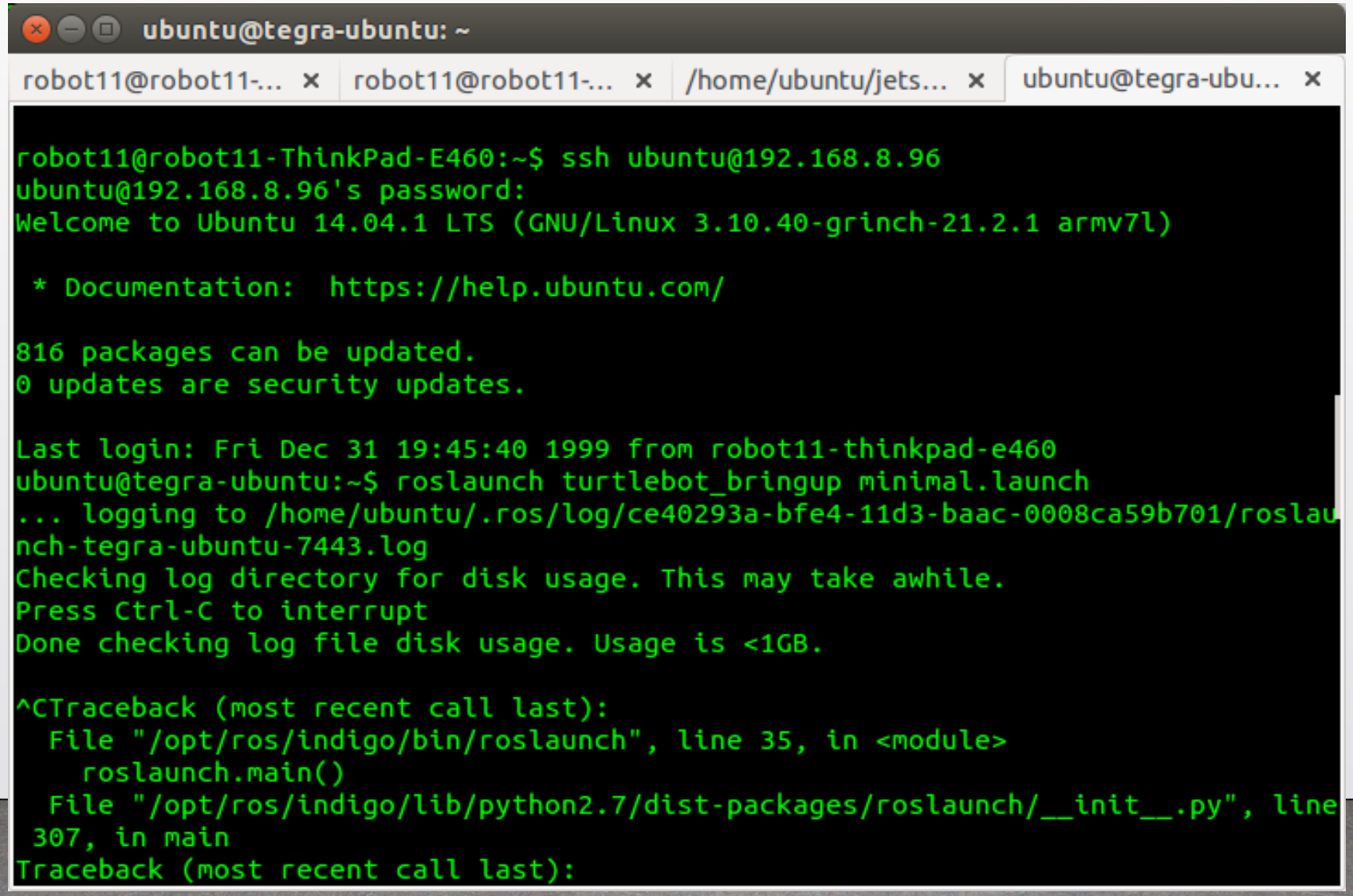
1. On the workstation Laptop (Lenovo)
   - `echo export ROS_MASTER_URI=http://IP_OF_TURTLEBOT:11311 >> ~/.bashrc`
   - `echo export ROS_HOSTNAME=IP_OF_PC >> ~/.bashrc`
2. On the Turtlebot Laptop (Acer)
   - `echo export ROS_MASTER_URI=http://localhost:11311 >> ~/.bashrc`
   - `echo export ROS_HOSTNAME=IP_OF_TURTLEBOT >> ~/.bashrc`

- Verify the connection
  - On the workstation Laptop, open a terminal and: **rostopic list**
  - Do you see a list of topics? If not, verify that ROS_HOSTNAME is set properly on the turtlebot laptop

How to SSH into your Turtlebot

- ***ssh turtlebot@[ip_of_turtlebot]***
- The password is 'turtlebot'
- Now you have a secure shell connection to your acer laptop.
- In the terminal type in: ***roslaunch turtlebot_bringup minimal.launch***
  - *This roslaunch command runs a series of nodes in your turtlebot that allow you to get data from IMU, encoders, cliff sensors, bumper switches and send commands to the actuators*
- If you need another terminal connected to turtlebot you need to repeat the ssh procedure in a new terminal

Please note that in the example below the username is 'ubuntu' whereas in your case it should be 'turtlebot'.



Some Command-Line Tools

- **Roscore**
  - Starts up ROS Master, ROS Parameter Server and RosOut logging node.
- **Rosbag**
  - Used for recording and playing data (from topics)
- **Rospack**
  - helps you find a package path. Usage: rospack find [package_name]
- **Roscd**
  - Allows you to cd directly to a package
  - Also rosls shows the files and folders inside a package

Some Command-Line Tools (Cont'd)

- **rosrun**
  - allows you to run an executable in an arbitrary package.
- **roslaunch**
  - launches a set of nodes from an XML configuration file (.launch file) and includes support for launching on remote machines.
- **rostopic**
  - displays run-time information about topics and also lets you print out messages being sent to a topic.
- **rosparam**
  - enables getting and setting parameter server values from the command-line.

Some Command-Line Tools - Graphical

- **rqt_plot**
  - plots numerical data on a ROS topic over time.
- **rqt_graph**
  - displays an interactive graph of ROS nodes and topics.
- **rqt_bag**
  - is a graphical tool for viewing data in ROS bag files.

Command tools: rostopic

rostopic echo [topic_name] *//see the messages being publish in a specific topic*
rostopic list *//see a list of available topics*
rostopic hz [topic_name] *//see the rate of a topic*
rostopic bw [topic_name] *//see the bandwidth usage of a topic*

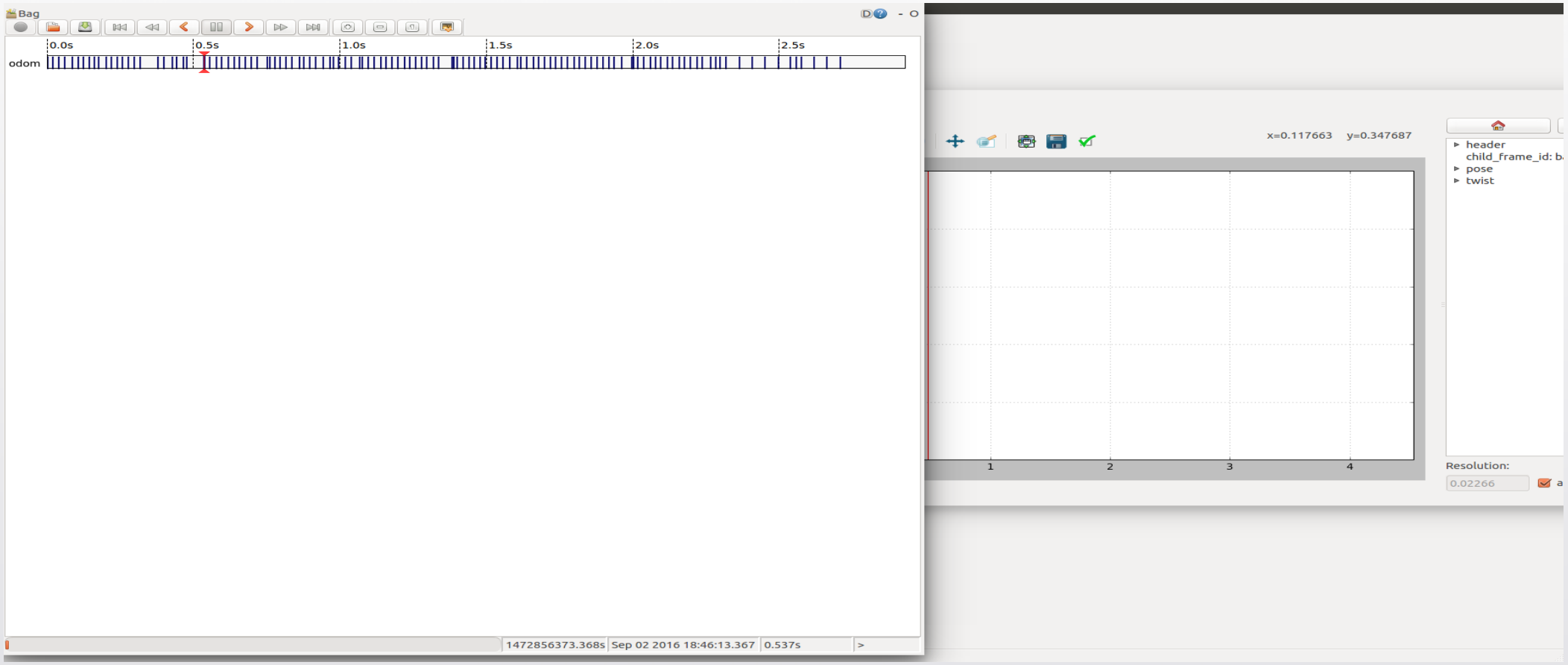
Publish to a topic in command line

rostopic pub -r [rate] [topic_name] [msg] //publish to a topic
Example: Let's make the robot move by publishing a Twist Message to the '/cmd_vel/input/navi' that specifies a linear Velocity of 0.1(m/s), play around with the parameters and observe the behavior of turtlebot.
rostopic pub -r 10 /cmd_vel_mux/input/navi geometry_msgs/Twist '{linear: {x: 0.1,y: 0.0,z: 0.0},angular: {x: 0.0,y: 0.0,z: 0.0}}'

Use rosbag and rqt_bag to record and playback/ plot topic data

First make a directory to save your bag files
mkdir ~/bagfiles
Now let's start recording a topic, for examle /odom
Rosbag record /odom
Hit ctrl+c to stop the recording process now use rqt_bag
To playback and plot the recorded data
rqt_bag [your_bag_file]

Use rosbag and rqt_bag to record and playback/plot topic data



Catkin Package Management Tool

- Catkin is a package management tool developed for ROS users that makes package management easier.
- Let's create a catkin workspace and our first package
- Open a terminal

```
bash-4.3$ mkdir -p ~/catkin_ws/src
bash-4.3$ cd catkin_ws/src/
bash-4.3$ catkin_init_workspace
```

Create a package and build a catkin workspace

- ***cd ~/catkin_ws/src***
- ***catkin_create_pkg [package_name] dep1 dep 2 …***
- ***cd ~/catkin_ws/***
- ***catkin_make***
- Next step is sourcing the setup file
- ***Source ~/catkin_ws/devel/setup.bash***

Example Code: turtlebot_turn.cpp

```
#include "ros/ros.h"
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/Imu.h>

void imuCallback(const sensor_msgs::Imu::ConstPtr& data)
{
  ROS_INFO("Angular Velocity = %f", data->angular_velocity.z);//print out the
angular velocity
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "turtlebot_turn");
  ros::NodeHandle nh;
  ros::Publisher vel_pub;
  vel_pub = nh.advertise<geometry_msgs::Twist>("/cmd_vel_mux/input/navi", 1,
true);
  ros::Subscriber sub = nh.subscribe("/mobile_base/sensors/imu_data", 10,
imuCallback);
  ros::Rate loop_rate(10);
  int count = 0;
  geometry_msgs::Twist vel;
  vel.linear.x = 0;//linear velocity(m/s)
  vel.angular.z = 1.0;//angular velocity(rad/s)
```

```
  while (ros::ok())
    {
      vel_pub.publish(vel);
      ros::spinOnce();
      loop_rate.sleep();
      ++count;
      if(count==20)
       vel.angular.z = 0;
    }
    return 0;
}
```

Useful Links

- Ros Wiki Turtlebot Page
- Catkin Tutorial
- The Turtlebot Stack on Github
- Turtlebot Demos
- UNIX Tutorial for Beginners
- Turtlebot Guide and Support