UNIVERSITY OF BIRMINGHAM

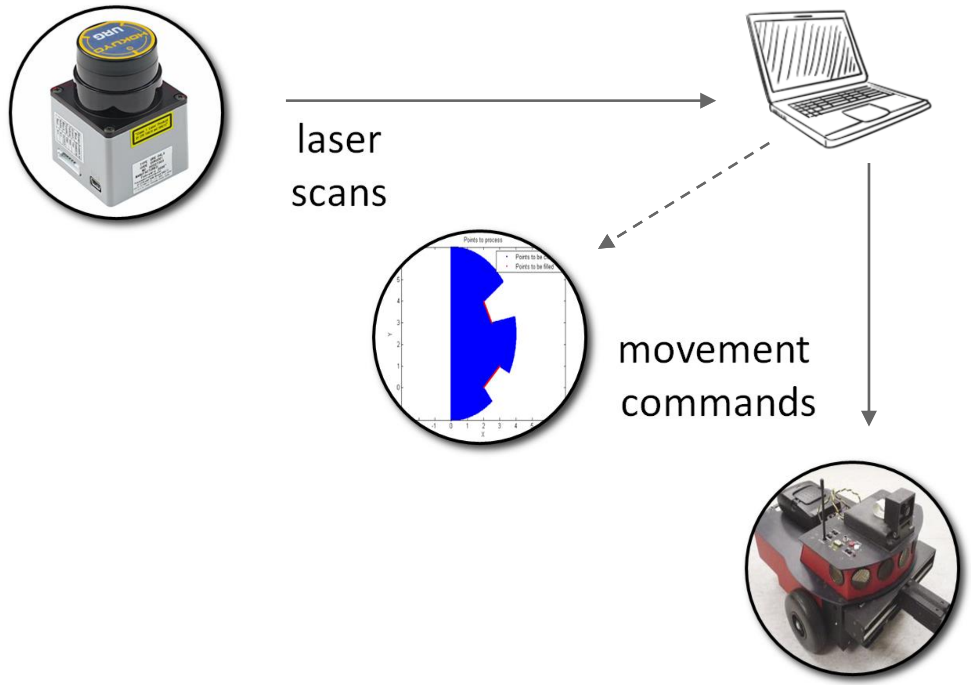
Introduction to ROS

Intelligent Robotics 2019
University of Birmingham

Module Lead:
Mohan Sridharan

Teaching Assistants:
Laura Ferrante (LXF656@bham.ac.uk)
Saif Sidhik (SXS1412@bham.ac.uk)

Robot: Sensors and Actuators



Sensors: information about environment/robot

Actuators: Make changes to robot state and/or environment

What is ROS?

ROS = Robot Operating System

- **Middleware:**
  Provides unified API for getting sensor info and commanding actuators

- **Peer to peer:**
  Individual programs communicate over defined API (ROS messages, services, ...)

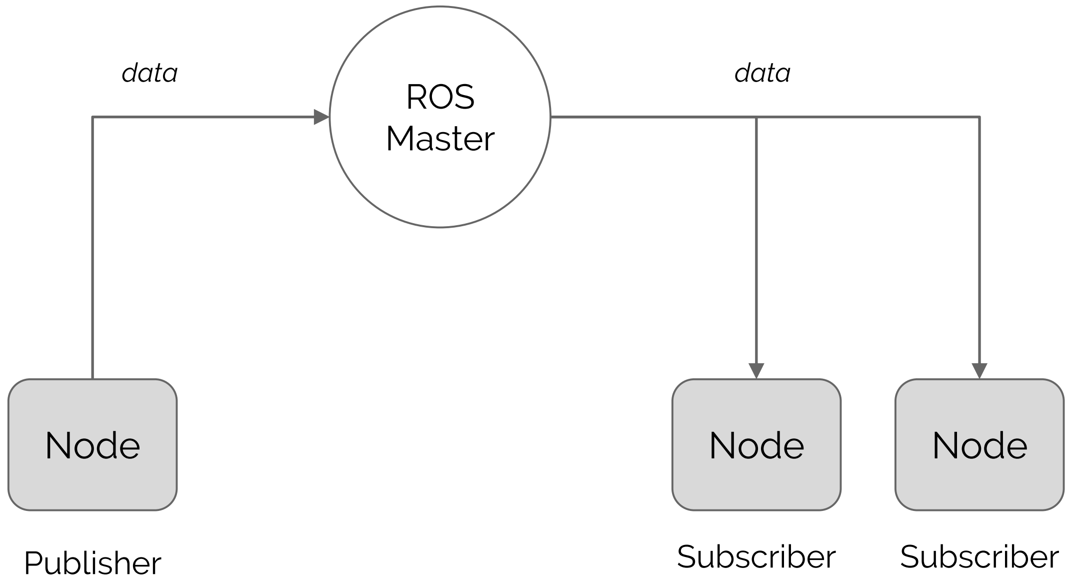
- **Interface** between low-level robot and high-level programming languages.
  Languages: C++, Python, Java...(any language for which a client library exists!)
  Robots: all sorts!

- **Distributed:**
  Programs can be run on multiple computers and communicate over the network

- **Open-source** community-driven platform, supported by Willow Garage and Google

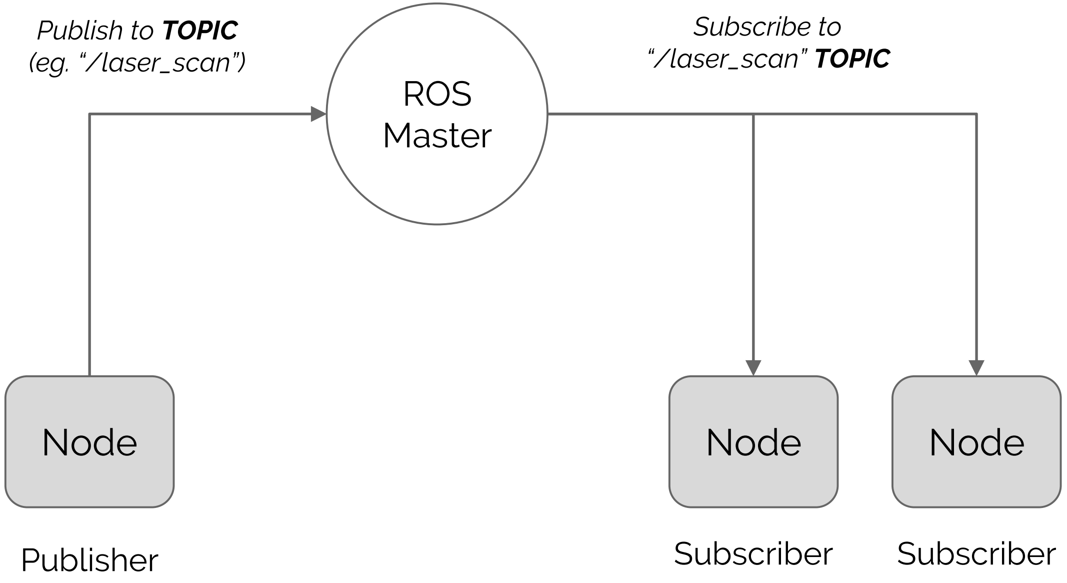
- Fast becoming the industry **standard** in robotics software

ROS: Basic Communication between Nodes



- All **nodes** connected to the same **ROS Master** can communicate with each other
- **Publisher** Nodes can **publish** information to the master
- Multiple nodes can publish/subscribe at the same time

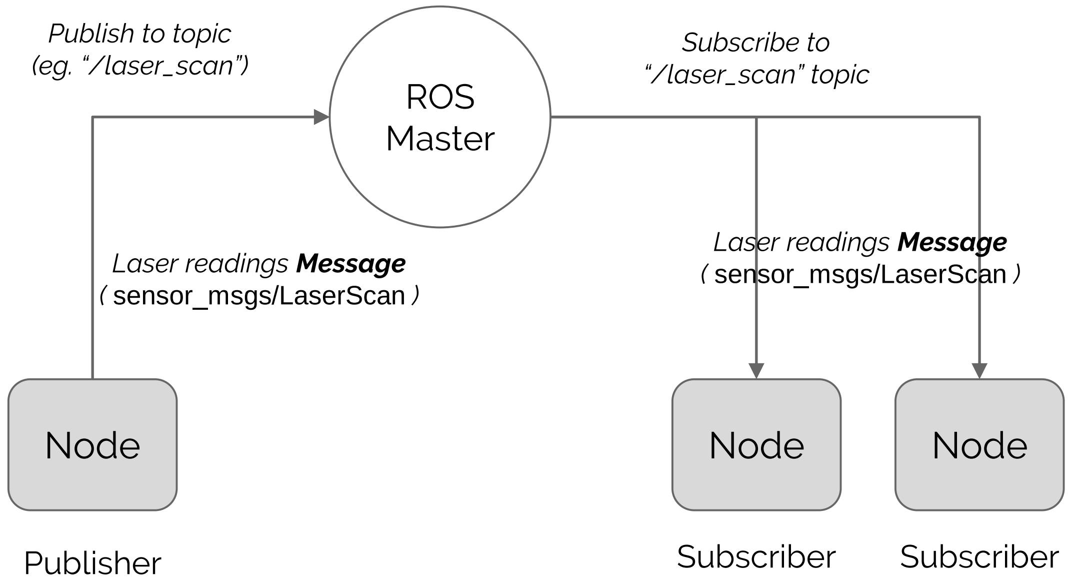
ROS: Basic Communication between Nodes



- All communications happen via **ROS topics**
- Publishers "publish" appropriate data to a "topic".
- Subscribers "subscribe" to the correct "topic" to get the data.

Topic *name* is any (unique) string value that you define. By convention, they start with "/"

ROS: Basic Communication between Nodes



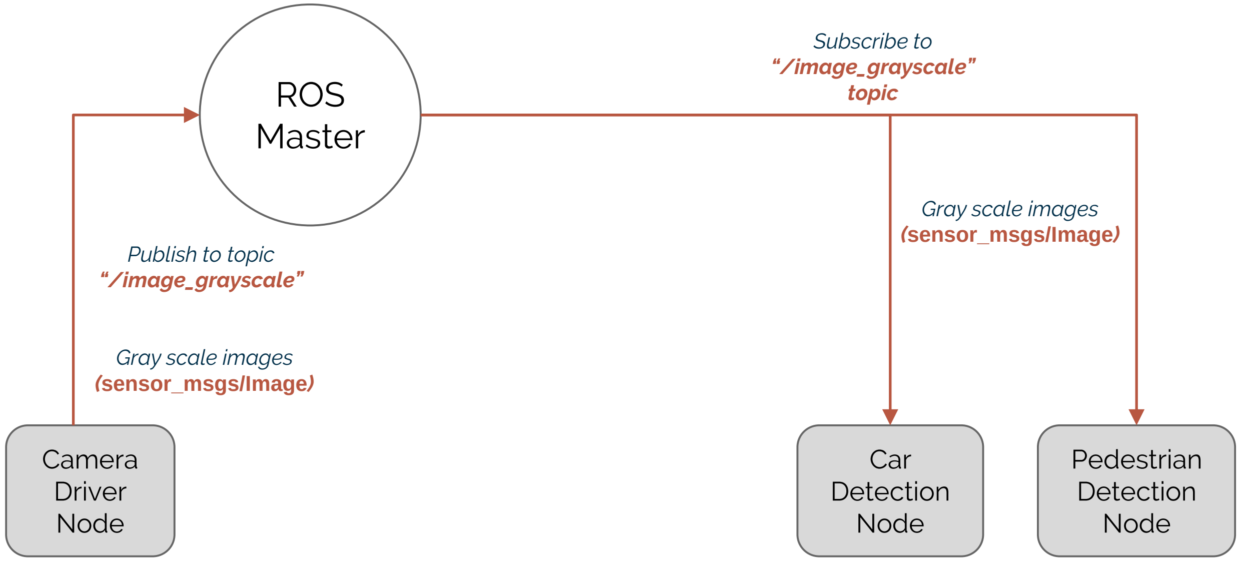
Publish to topic (eg. "/laser_scan")
ROS Master
Subscribe to "/laser_scan" topic
Laser readings ***Message*** (sensor_msgs/LaserScan)
Laser readings ***Message*** (sensor_msgs/LaserScan)
Node
Node
Node
Publisher
Subscriber
Subscriber

- Data is transmitted through *topics* as **ROS Messages**

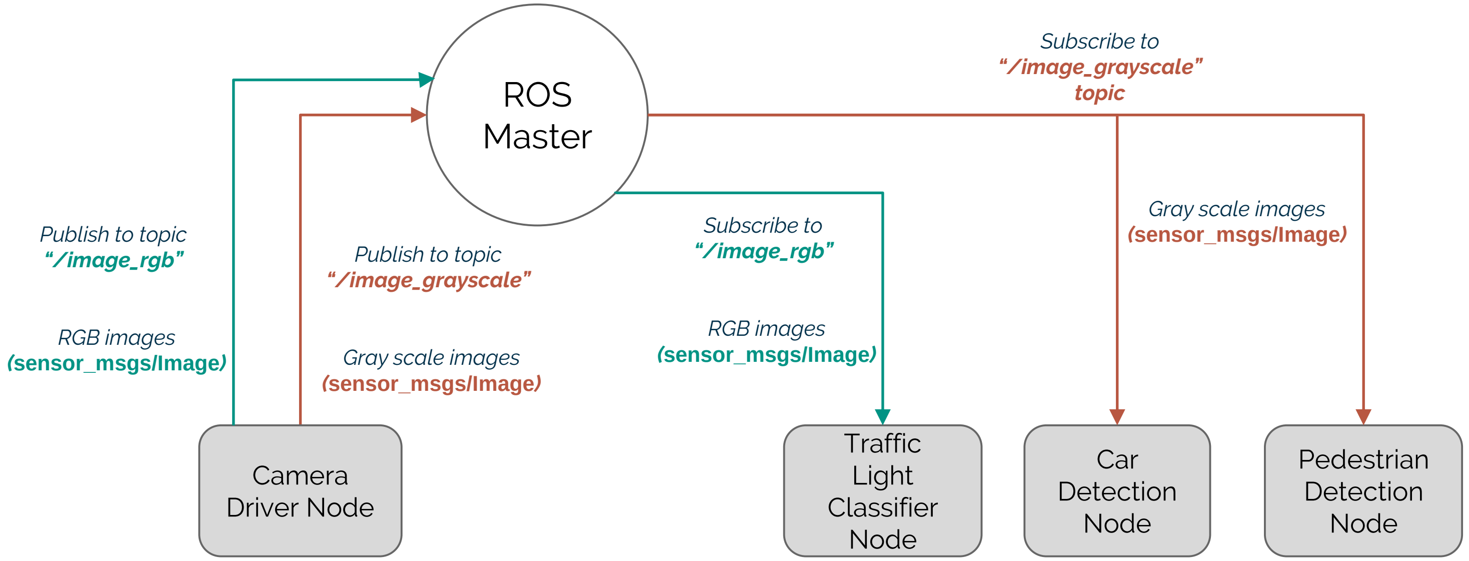
- ***Only one type of ROS Message can be transmitted through each topic***

When defining/initialising a publisher or subscriber in a node, the correct *topic_name* and *msg_type* is specified

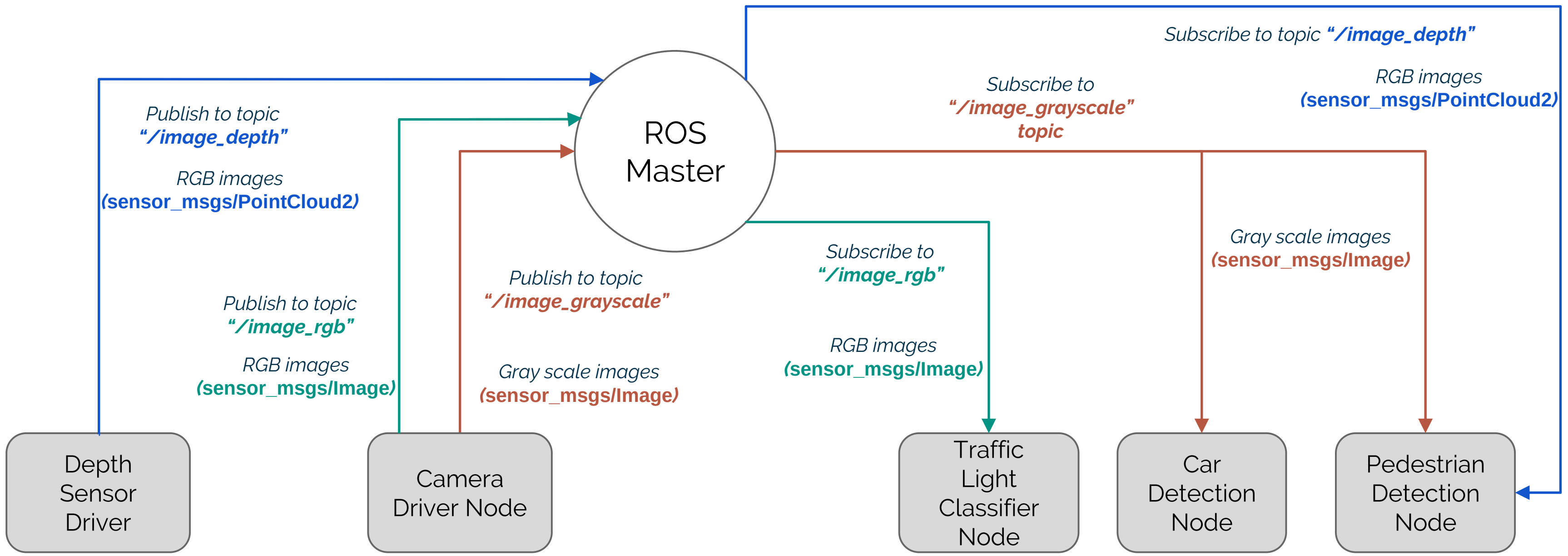
ROS Nodes Example:


ROS Master
Subscribe to "/image_grayscale" topic
Gray scale images (sensor_msgs/Image)
Publish to topic "/image_grayscale"
Gray scale images (sensor_msgs/Image)
Camera Driver Node
Car Detection Node
Pedestrian Detection Node

ROS Nodes Example:


ROS Master
Publish to topic "/image_rgb"
RGB images (sensor_msgs/Image)
Publish to topic "/image_grayscale"
Gray scale images (sensor_msgs/Image)
Subscribe to "/image_rgb"
RGB images (sensor_msgs/Image)
Subscribe to "/image_grayscale" topic
Gray scale images (sensor_msgs/Image)
Camera Driver Node
Traffic Light Classifier Node
Car Detection Node
Pedestrian Detection Node

ROS Nodes Example:



ROS:
File Structure

Catkin workspace

Packages:
contain code for nodes, configuration files, build scripts,
launch files,...

Nodes:
executable or library

ROS:
File Structure

Catkin workspace

Meta Packages:
References to or contains collection of related packages

Packages:
contain code for nodes, configuration files, build scripts, launch files,...

Nodes:
executable or library

Building your workspace

- Create and build your catkin workspace:

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

- Create your package in the workspace:

http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage

- Writing your first node (publisher and subscriber):

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

★ *Remember to (re)build your catkin workspace each time you modify your package.*

ROS Nodes

- single-purpose, executable program
  - To make a python file executable, run `$ chmod +x my_file.py`
- individually compiled, executed, managed
- organized in packages
- Basic console commands:

Run a node	`$ rosrun package_name node_name`
See active nodes	`$ rosnode list`
Retrieve info about a node	`$ rosnode info node_name`

ROS Topic Console Commands

Some of the main commonly-used commands are:

- List of active topics `$`
  `rostopic list`
- Subscribe and print the content of a topic with `$ rostopic echo`
  `/topic_name`
- Show info about a topic `$`
  `rostopic info /topic_name`

More commands and documentation: http://wiki.ros.org/rostopic

Tutorial: http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

ROS Publishers/Subscribers: Additional Notes

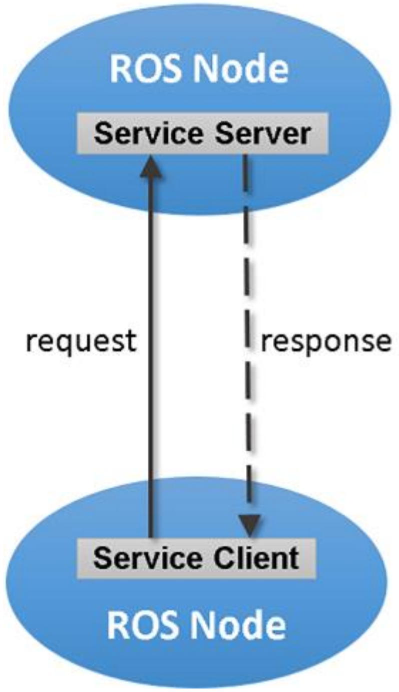
- Each node may have multiple publishers and/or subscribers
- Communication is non-blocking, updated continuously
- Multiple nodes may publish/subscribe to the *same topic (although publishing to same topic is not recommended)*
- *However, each topic can have only one message type*
- Most of the *Message Types* that you would require are available in default ROS installation (in packages such as nav_msgs, geometry_msgs, sensor_msgs, etc.). Many others are available from external ROS packages.
    - You can also create custom messages as you need (http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv)

ROS: Some other concepts

Services

- Request-response architecture
- Blocking
- On-demand data

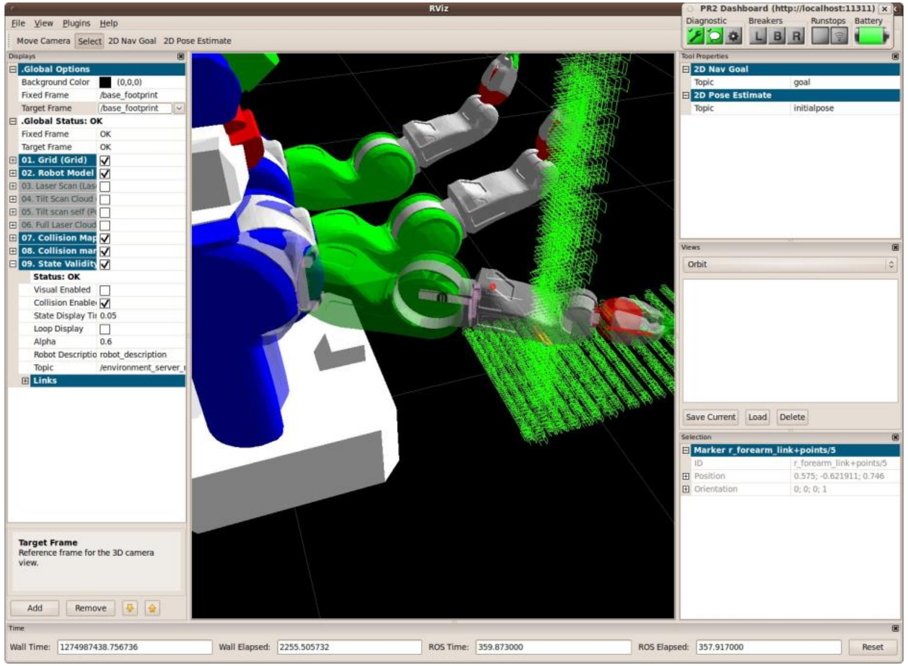
Tutorial:
http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29



ROS: Some other concepts

- **Parameter Server:** Stores master-wide parameters (eg: robot model)
- **ROS Bags:** used for recording and playing ROS messages
    - Reference: http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data
- **Launch Files:** Utility file for running multiple nodes at the same time, set parameters, remap topics, etc.
    - Reference: http://wiki.ros.org/roslaunch/Tutorials
    - Usage: `$ roslaunch package_name launch_file <optional_arguments>`

ROS Visualisation Tool: RViz


RViz
File View Plugins Help
Move Camera Select 2D Nav Goal 2D Pose Estimate
PR2 Dashboard (http://localhost:11311)
Diagnostic Breakers Runstops Battery
L B R
Displays
Tool Properties
Views
Selection
Time
Add Remove
Save Current Load Delete
Reset

.Global Options	
Background Color	■ (0,0,0)
Fixed Frame	/base_footprint
Target Frame	/base_footprint
.Global Status: OK	
Fixed Frame	OK
Target Frame	OK
01. Grid (Grid)	✓
02. Robot Model	✓
03. Laser Scan (Las	☐
04. Tilt Scan Cloud	☐
05. Tilt scan self (P	☐
06. Full Laser Cloud	☐
07. Collision Map	✓
08. Collision mar	✓
09. State Validity	✓
Status: OK	
Visual Enabled	☐
Collision Enable	✓
State Display Ti	0.05
Loop Display	☐
Alpha	0.6
Robot Descriptio	robot_description
Topic	/environment_server_
Links	

Target Frame
Reference frame for the 3D camera view.

2D Nav Goal	
Topic	goal
2D Pose Estimate	
Topic	initialpose

Orbit

Marker r_forearm_link+points/5	
ID	r_forearm_link+points/5
Position	0.575; -0.621911; 0.746
Orientation	0; 0; 0; 1

Wall Time: 1274987438.756736 Wall Elapsed: 2255.505732 ROS Time: 359.873000 ROS Elapsed: 357.917000

```
$ rosrun rviz rviz
```

- Visualise messages in chosen topic(s)
- Debug messages from/to nodes
- Simulate before testing on real robot

Note: A ‘global frame’ should be defined for all topics you want to visualise

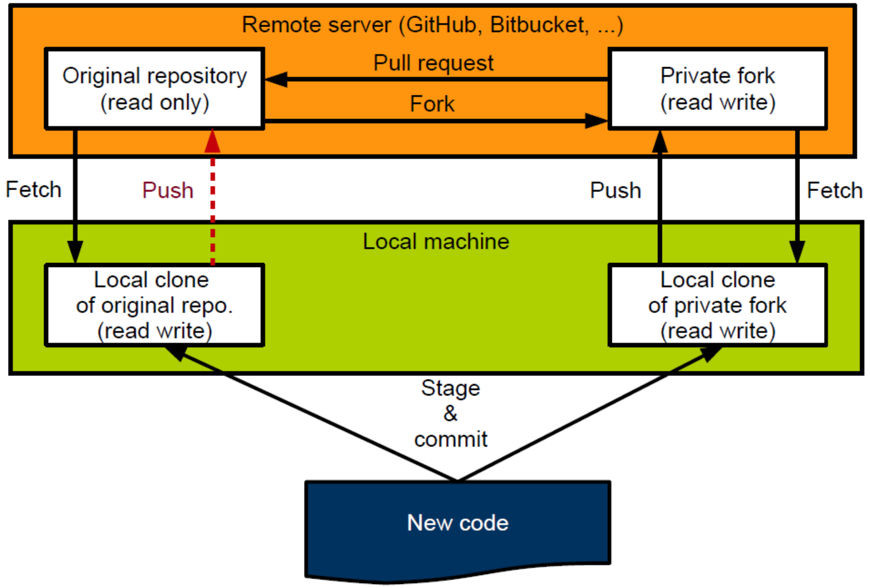
  - All topics must be defined in or have a transformation to this ‘global frame’

Lab Sessions

- Each team will be provided with a laptop (with **Ubuntu 18.04** and **ROS Melodic** already installed), a Pioneer P3DX robot, and an equipment kit
    - If you want to use own machine, it is highly recommended to install Ubuntu 18.04 and ROS Melodic (**we will not be able to help you otherwise**)

Version Control is your friend

- Use Git (Github/Gitlab)
- Always version control your code with proper commit messages
- Comment and document your codes


Remote server (GitHub, Bitbucket, ...)
Original repository (read only)
Pull request
Fork
Private fork (read write)
Fetch
Push
Push
Fetch
Local machine
Local clone of original repo. (read write)
Local clone of private fork (read write)
Stage & commit
New code