An introduction to ROS

matteo.luperto at unimi.it

ROS: the Robot Operating System

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. [wiki.ros.org]

Robot software architecture

Low level functionalities as real-time motor controllers, sensors drivers, battery management

+

Core functionalities as mapping, localization, navigation, people detection

+

Reasoning mechanism for path planning, task allocation, self management

ROS

Robot software architecture

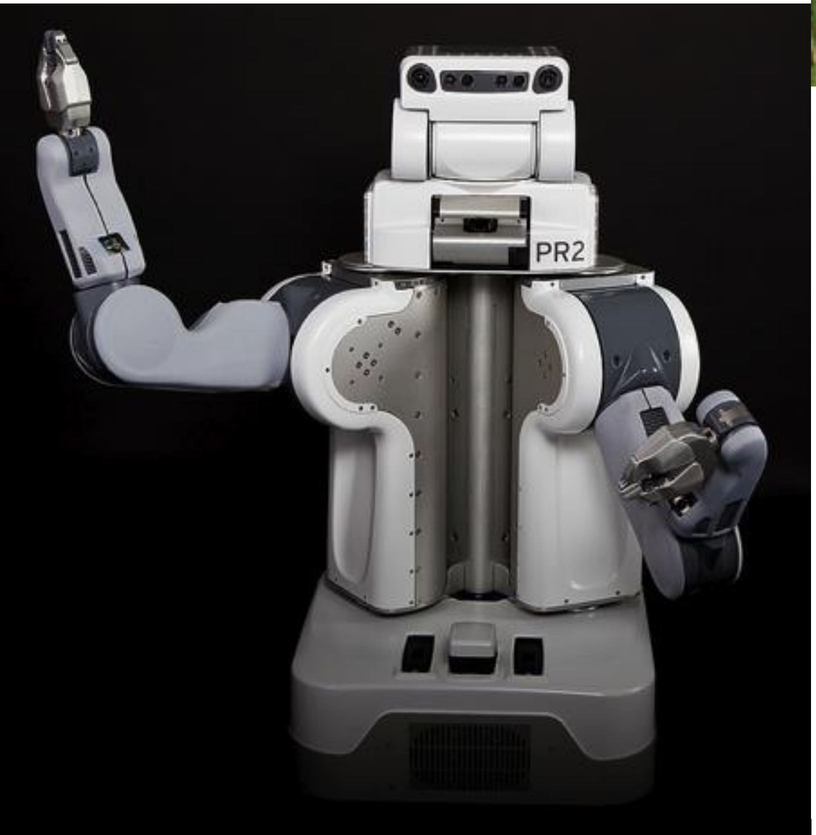
The development of (even a single) robots (functionality) requires both low-level hardware related and high-level AI-based mechanism

Modularity and scalability are consequently core features in a robot software architecture

ROS provide this

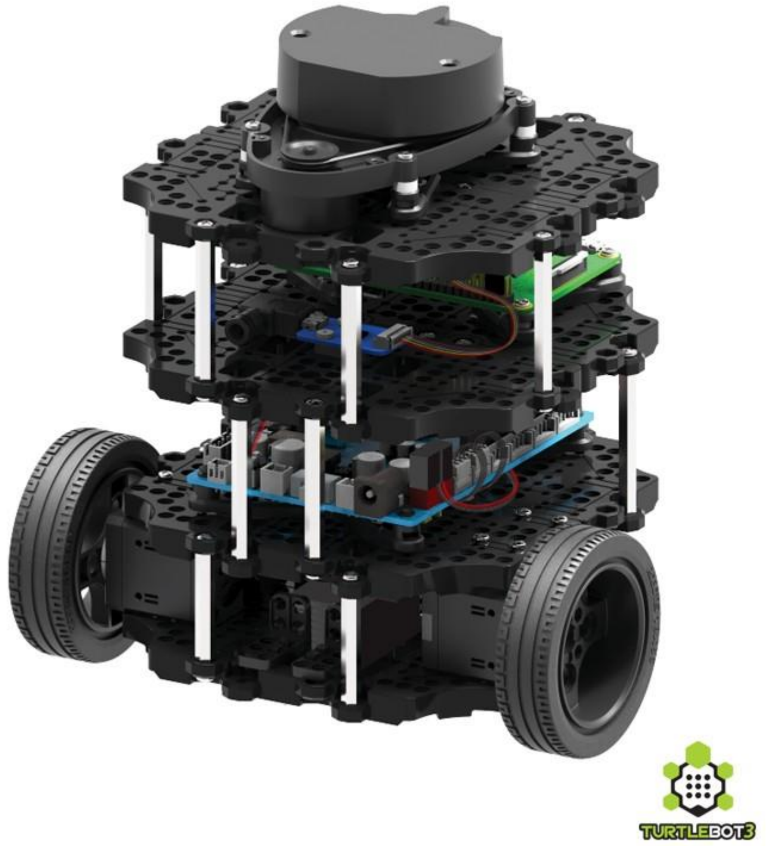
ROS has established itself as the de-facto standard for robot development

ROS



More at:
robots.ros.org

Our ROS robots



Sensors with ROS [wiki.ros.org]

What is ROS?

Is a Meta-Operating System

- Scheduling – loading – monitoring, and error handling
- virtualization layer between applications and distributing computing resources
- runs on top of (multiple) operating system(s)

- is a framework
- not a real-time framework but embed real-time code
- enforce supports a modular software architecture

ROS SW architecture

- distributed framework of processes (*Nodes*)
- enables executables to be individually designed and loosely coupled at runtime.
- processes can be easily shared and distributed.
- supports a federated system of code *Repositories* that enable collaboration to be distributed as well.

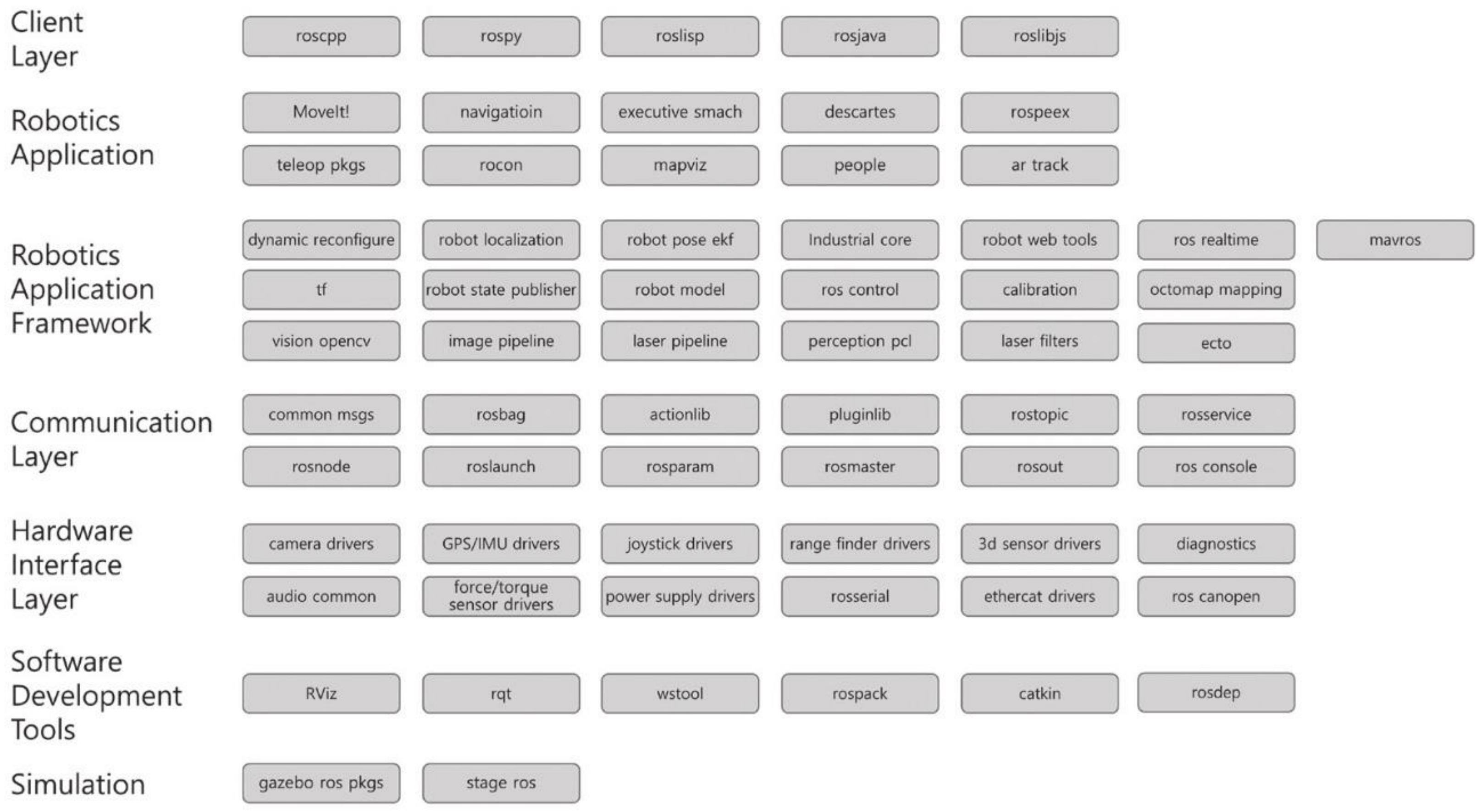
This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools.

More ROS features

- thin: ROS is designed to be as thin as possible
- easy to integrate with other frameworks and libraries
- language independence
  core languages are Python and C++ but you can use what you want
- easy testing: built in unit/integration test framework and debug tool
- scaling: ROS tools can be distributed across different machines and is appropriate for large development process

The core idea behind all of this is: <u>code reuse + modularity</u>

What ROS provides

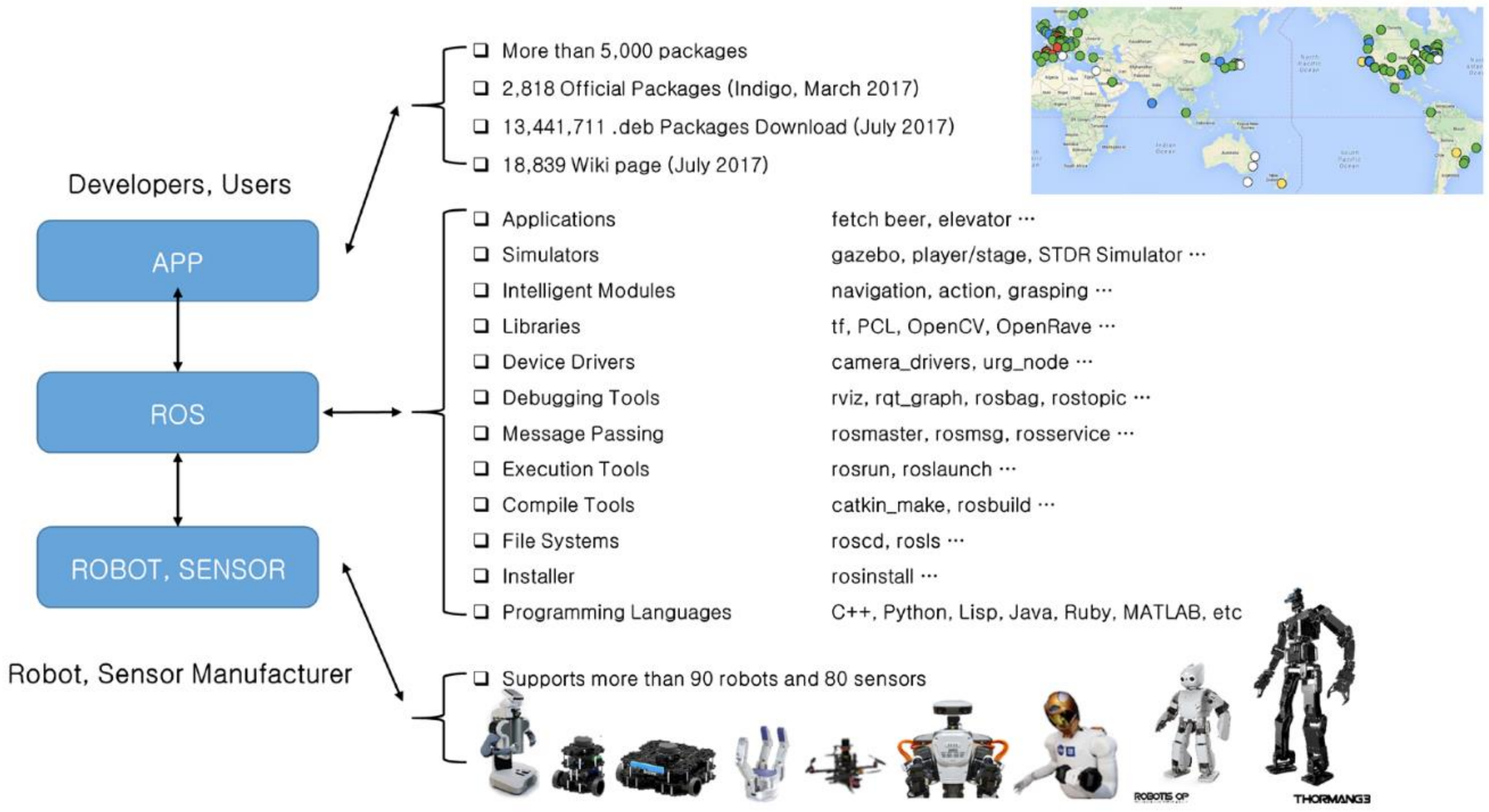


Client Layer	roscpp	rospy	roslisp	rosjava	roslibjs		
Robotics Application	MoveIt!	navigatioin	executive smach	descartes	rospeex		
	teleop pkgs	rocon	mapviz	people	ar track		
Robotics Application Framework	dynamic reconfigure	robot localization	robot pose ekf	Industrial core	robot web tools	ros realtime	mavros
	tf	robot state publisher	robot model	ros control	calibration	octomap mapping	
	vision opencv	image pipeline	laser pipeline	perception pcl	laser filters	ecto	
Communication Layer	common msgs	rosbag	actionlib	pluginlib	rostopic	rosservice	
	rosnode	roslaunch	rosparam	rosmaster	rosout	ros console	
Hardware Interface Layer	camera drivers	GPS/IMU drivers	joystick drivers	range finder drivers	3d sensor drivers	diagnostics	
	audio common	force/torque sensor drivers	power supply drivers	rosserial	ethercat drivers	ros canopen	
Software Development Tools	RViz	rqt	wstool	rospack	catkin	rosdep	
Simulation	gazebo ros pkgs	stage ros					

- core and advanced robot functionalities (mapping, localization, navigation, obstacle avoidance)
- drivers and integration with sensors
- integration with multiple robot architectures
  UAV – manipulators –wheeled robots
- integration with libraries (OpenPose, OpenCV, deep learning fw)
- simulation tools

All free and ready to use
Support from the community

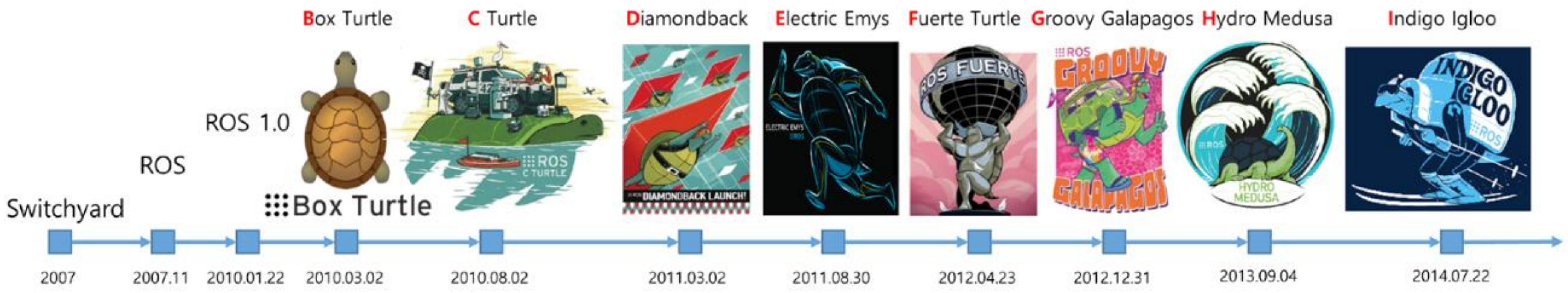
ROS-community

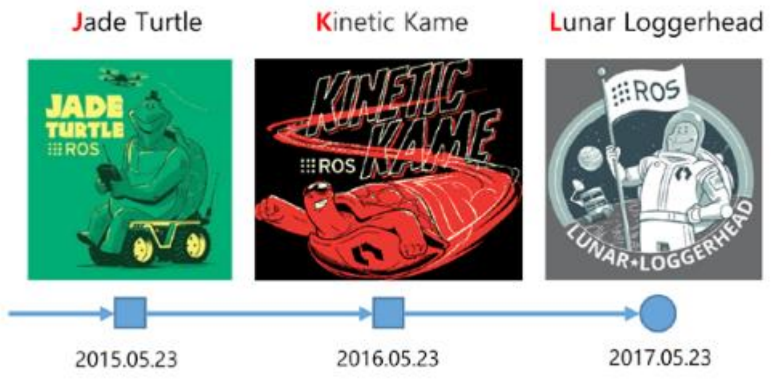


- ❑ More than 5,000 packages
- ❑ 2,818 Official Packages (Indigo, March 2017)
- ❑ 13,441,711 .deb Packages Download (July 2017)
- ❑ 18,839 Wiki page (July 2017)

❑ Applications	fetch beer, elevator ···
❑ Simulators	gazebo, player/stage, STDR Simulator ···
❑ Intelligent Modules	navigation, action, grasping ···
❑ Libraries	tf, PCL, OpenCV, OpenRave ···
❑ Device Drivers	camera_drivers, urg_node ···
❑ Debugging Tools	rviz, rqt_graph, rosbag, rostopic ···
❑ Message Passing	rosmaster, rosmsg, rosservice ···
❑ Execution Tools	rosrun, roslaunch ···
❑ Compile Tools	catkin_make, rosbuild ···
❑ File Systems	roscd, rosls ···
❑ Installer	rosinstall ···
❑ Programming Languages	C++, Python, Lisp, Java, Ruby, MATLAB, etc

- ❑ Supports more than 90 robots and 80 sensors





- 10y of ROS now
- last version: ROS Melodic (2019)
- next mayor release: ROS 2

Core aspects of

ROS

ROS aspects

- nodes
- topics
- messages

Building blocks of ROS

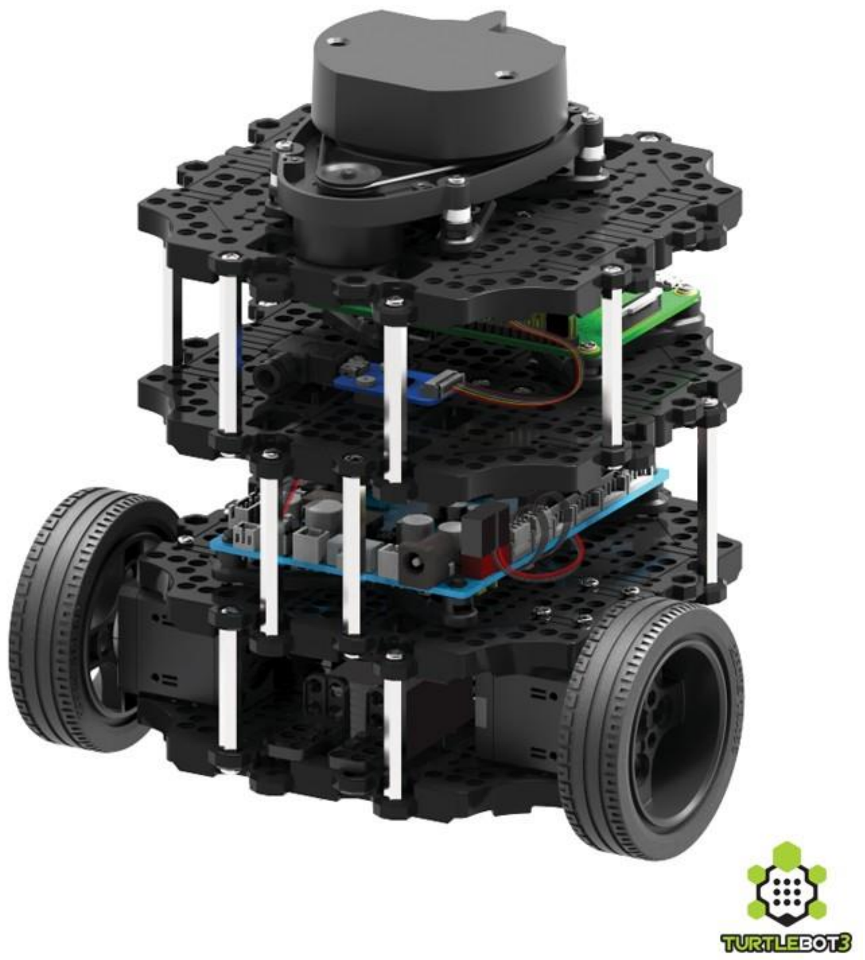
- services
- actions
- transforms

Communication / SW architecture

- debugging Tools
- simulations
- bags

Developers tools

ROS *nodes*

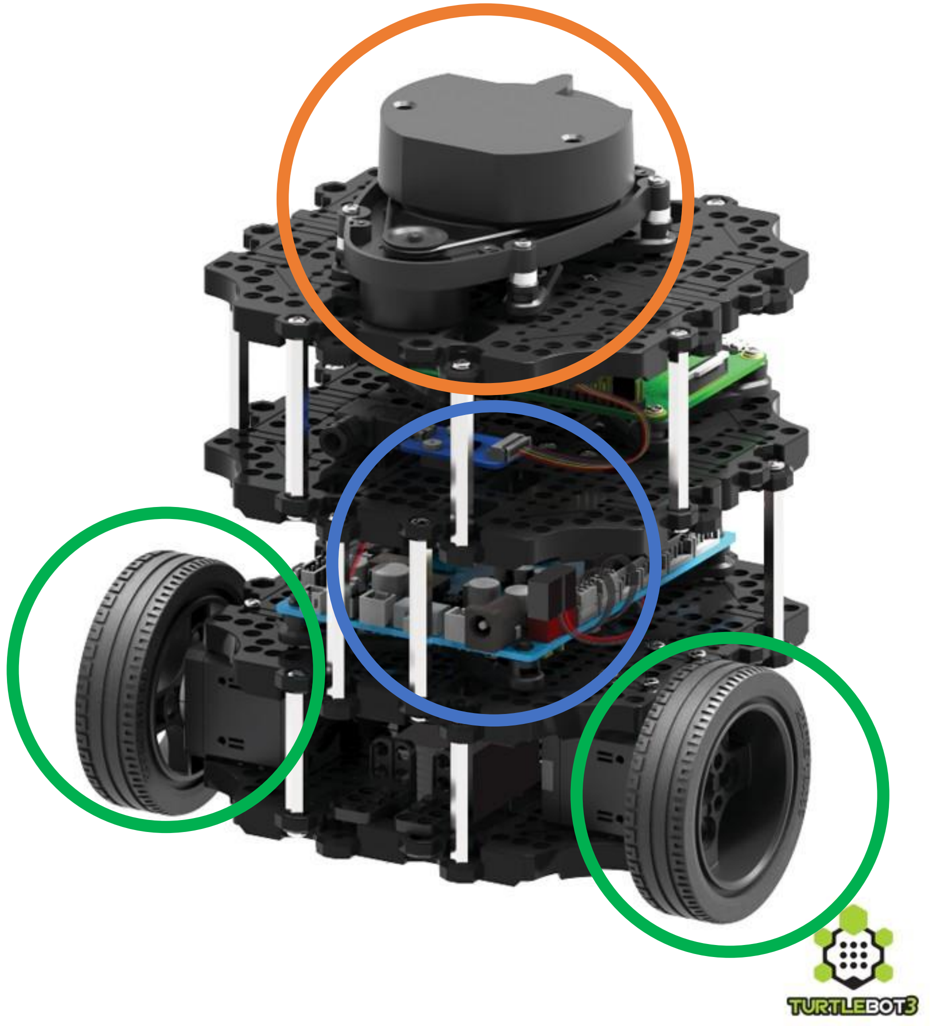


A *node* is a process that performs computation:

- nodes are combined together into a graph and communicate with one another using streaming topics, services, and parameters,
- are meant to operate at a fine-grained scale,
- a robot control system will usually comprise many nodes.

ROS

ROS *nodes*



For example, one node controls a laser range-finder, one Node controls the robot's wheels motors, one node performs mapping, one localization, one node performs path planning, one node gives velocity commands to the wheels, one node provides a graphical view of the system, and so on.

ROS *nodes*

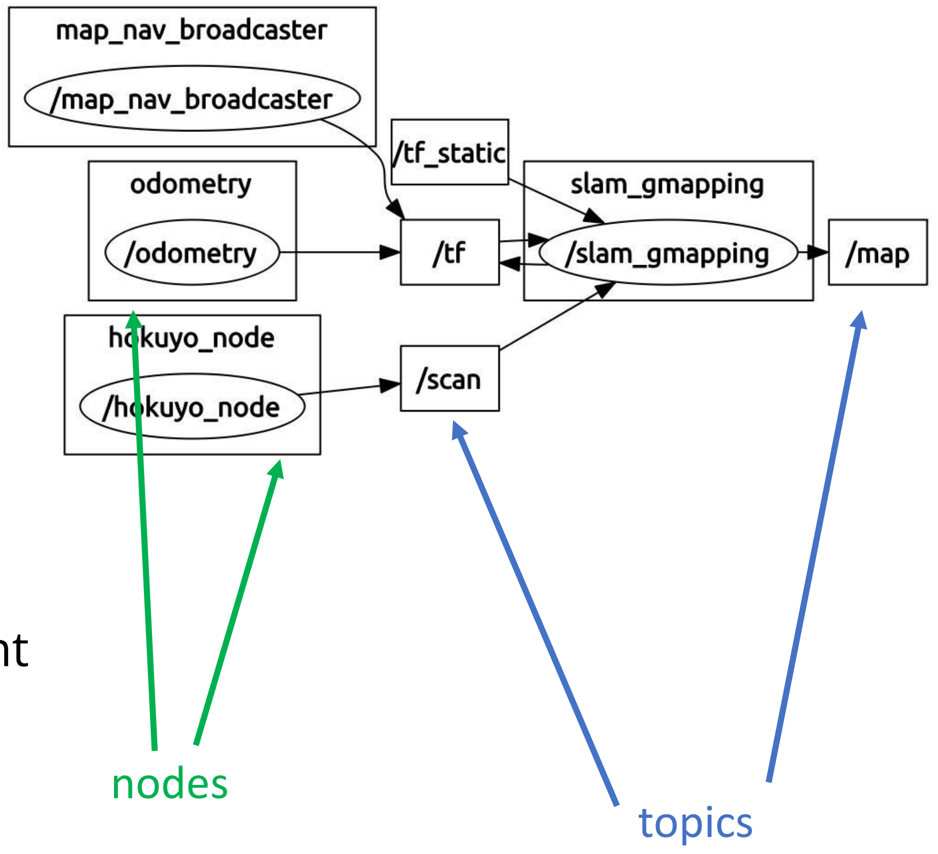
The use of nodes in ROS provides several benefits to the overall system.

- *fault tolerance* as crashes are isolated to individual nodes.
- *code complexity* is reduced in comparison to monolithic systems. Implementation details are also well hidden - nodes expose a minimal API –
- alternate implementations, even in other programming languages, can easily be substituted.

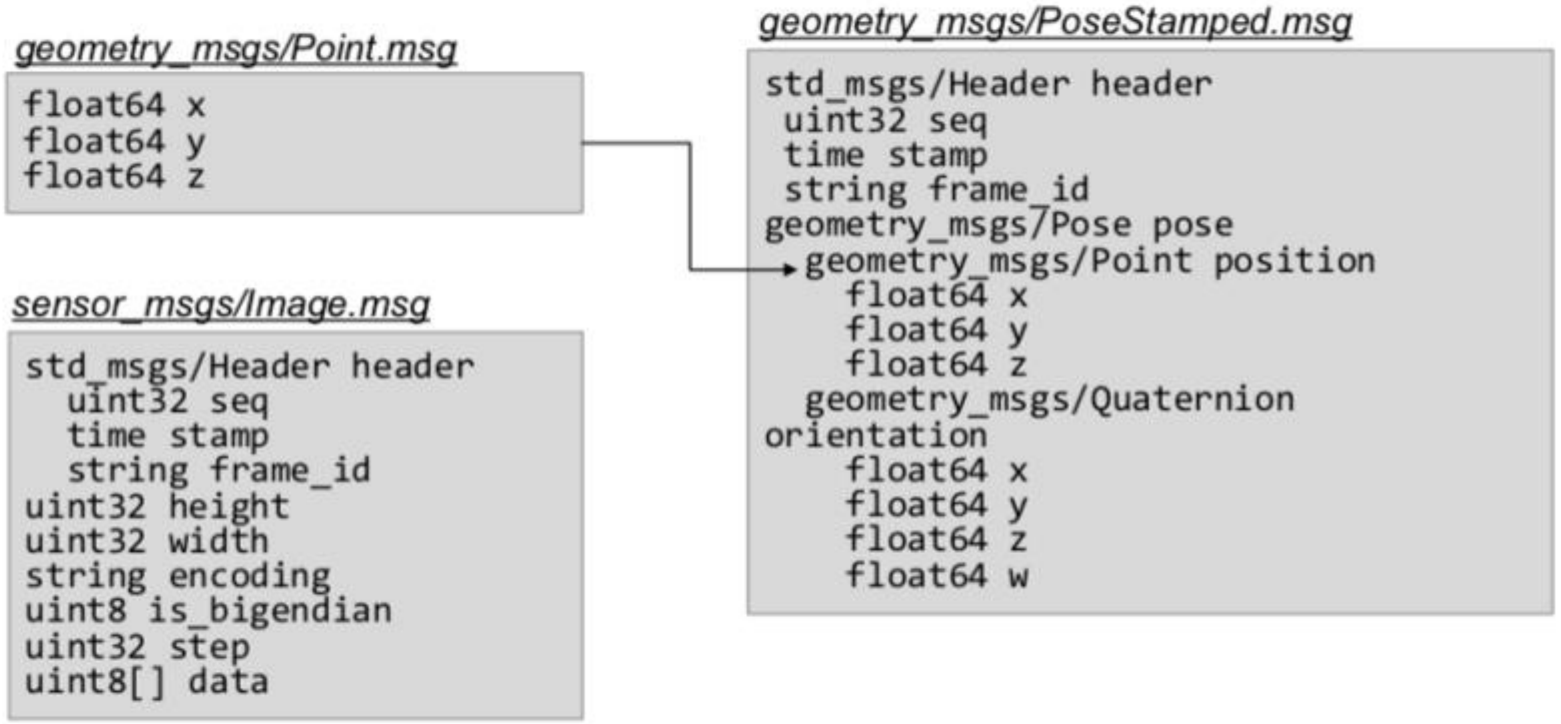
Nodes and *topics*

Topics are named buses over which nodes exchange messages.

- topics have **anonymous publish/subscribe semantics**, which decouples the production of information from its consumption.
- nodes are not aware of who they are communicating with.
- nodes that are interested in data *subscribe* to the relevant topic; nodes that generate data *publish* to the relevant topic.
- there can be multiple publishers and subscribers to a topic.



ROS *topics* and *messages*



geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

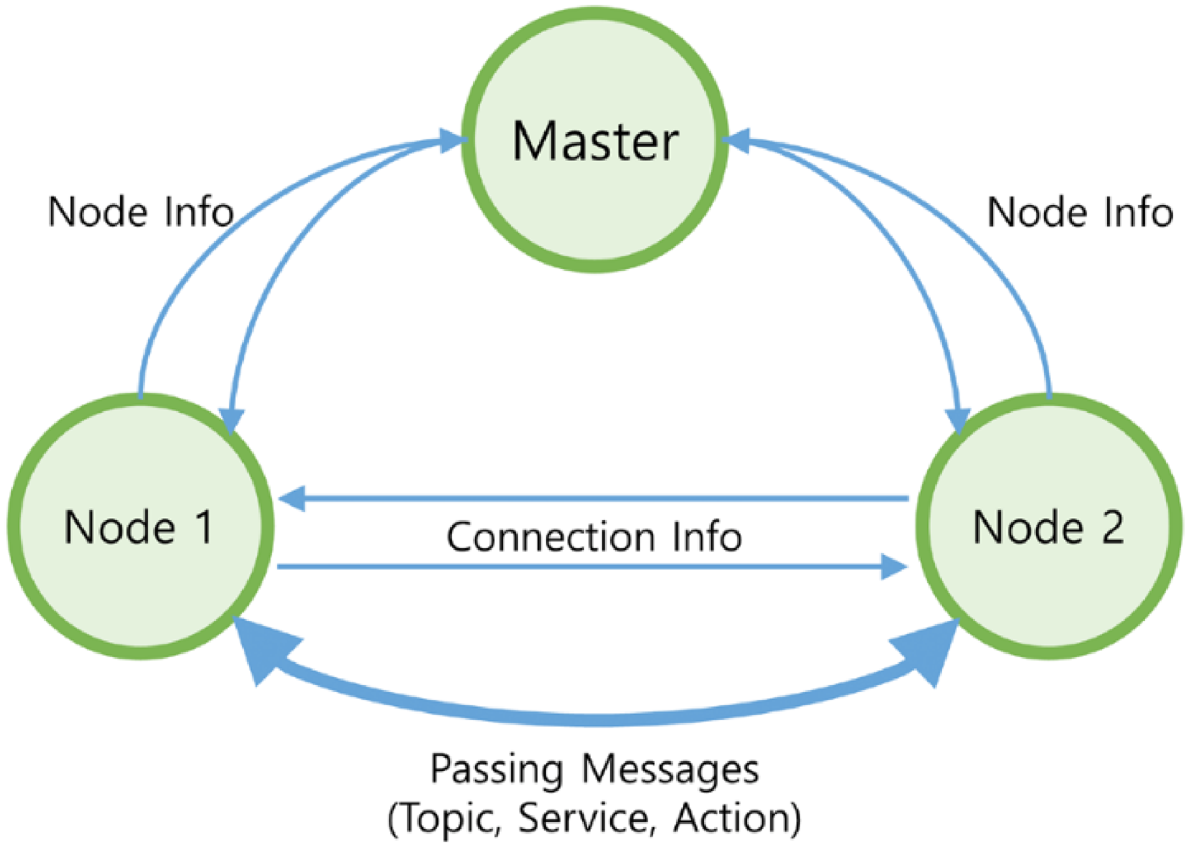
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion
orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

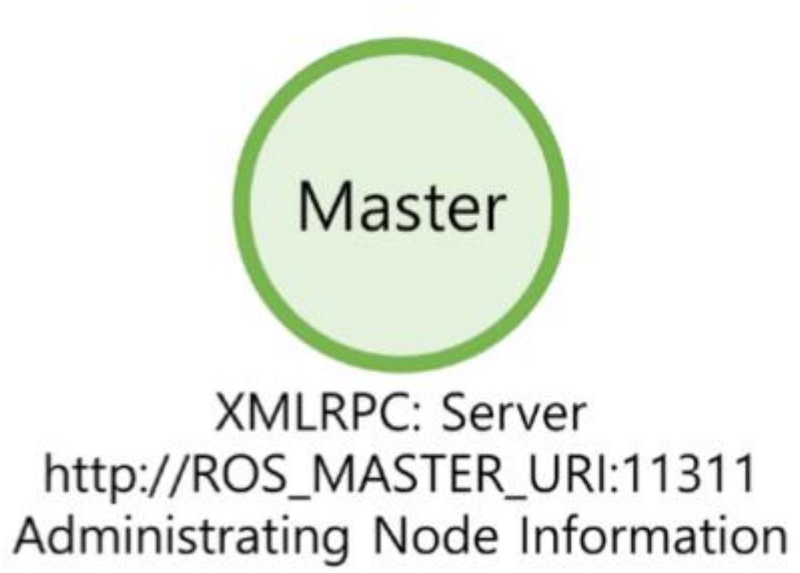
- each topic is strongly typed by the ROS message type used to publish to it
- nodes can only receive messages with a matching type.
- type consistency is not enforced among the publishers, but subscribers will not establish message transport unless the types match.
- all ROS clients check to make sure that an MD5 computed from the message format match.

ROS master



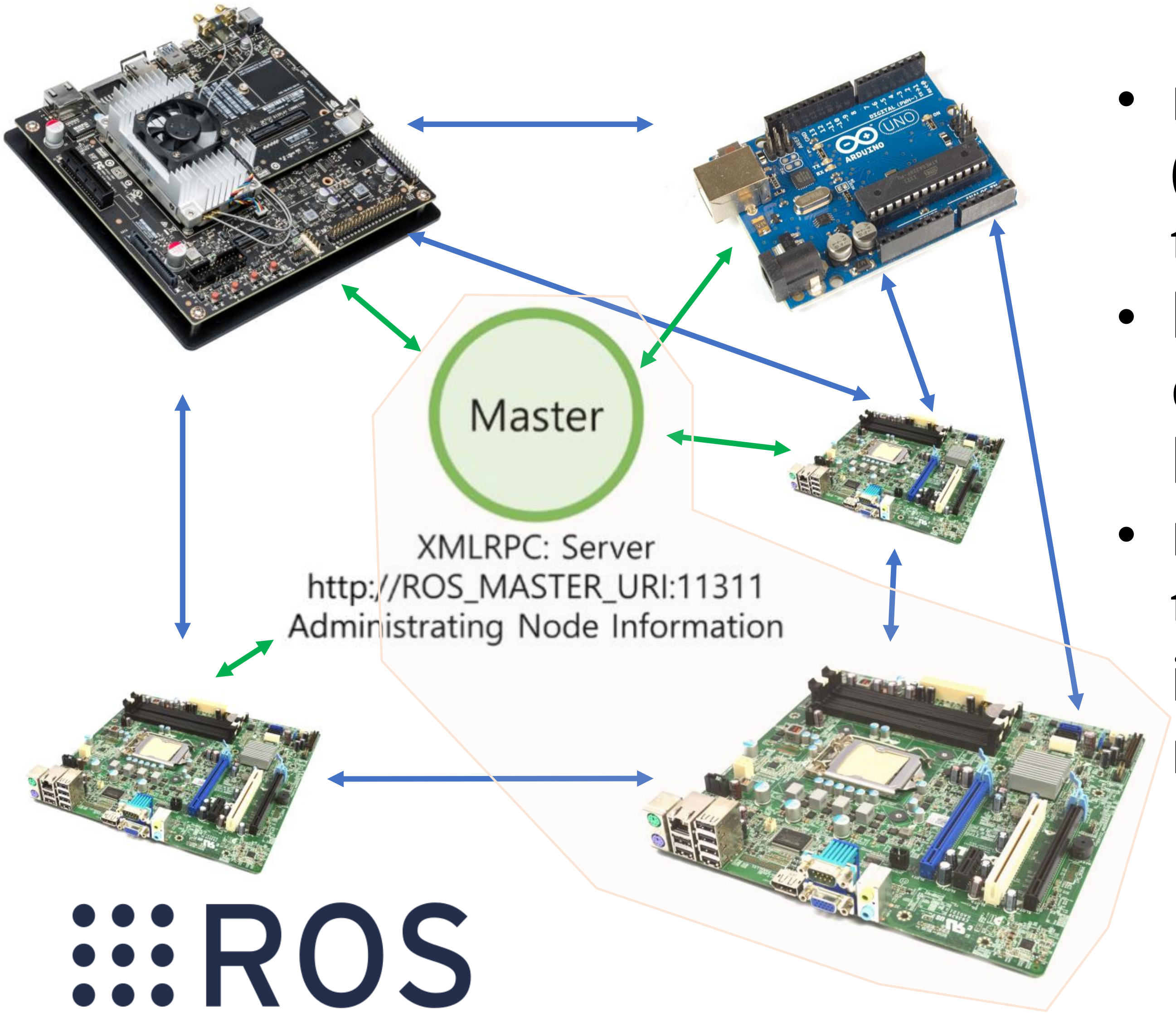
- the ROS Master provides naming and registration services to the rest of the nodes in the ROS system.
- it tracks publishers and subscribers to topics.
- it enables individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

ROS master and nodes



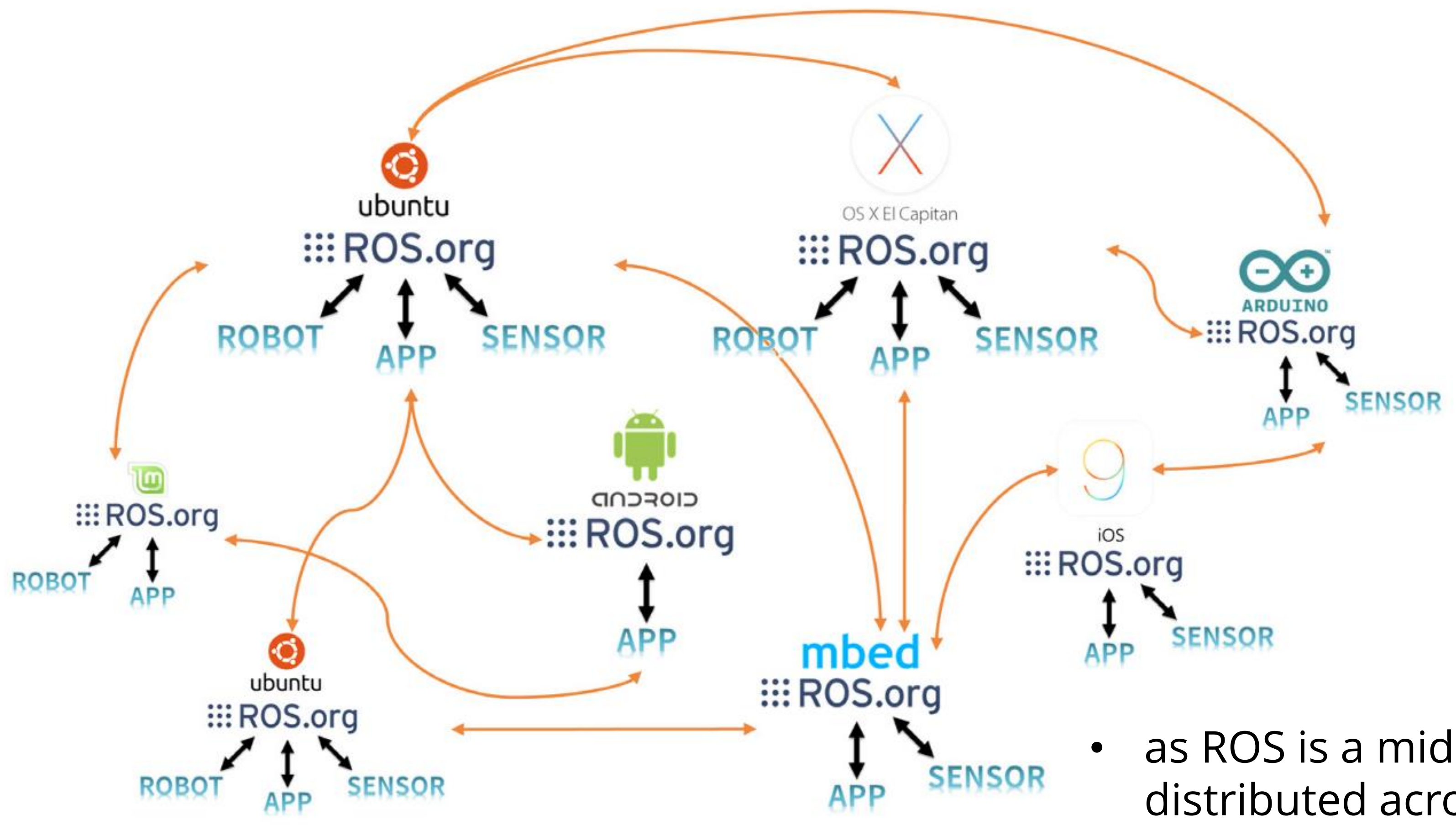
- the ROS master is a process and it is defined by its IP/port shared across all nodes
- acts as coordinator and manages the communication among nods
- this allows nodes to be distributed on different machines
  (in the same network)
- this mechanism allowing to decouple the execution of a process from the machine where the process id distributed

ROS master and nodes



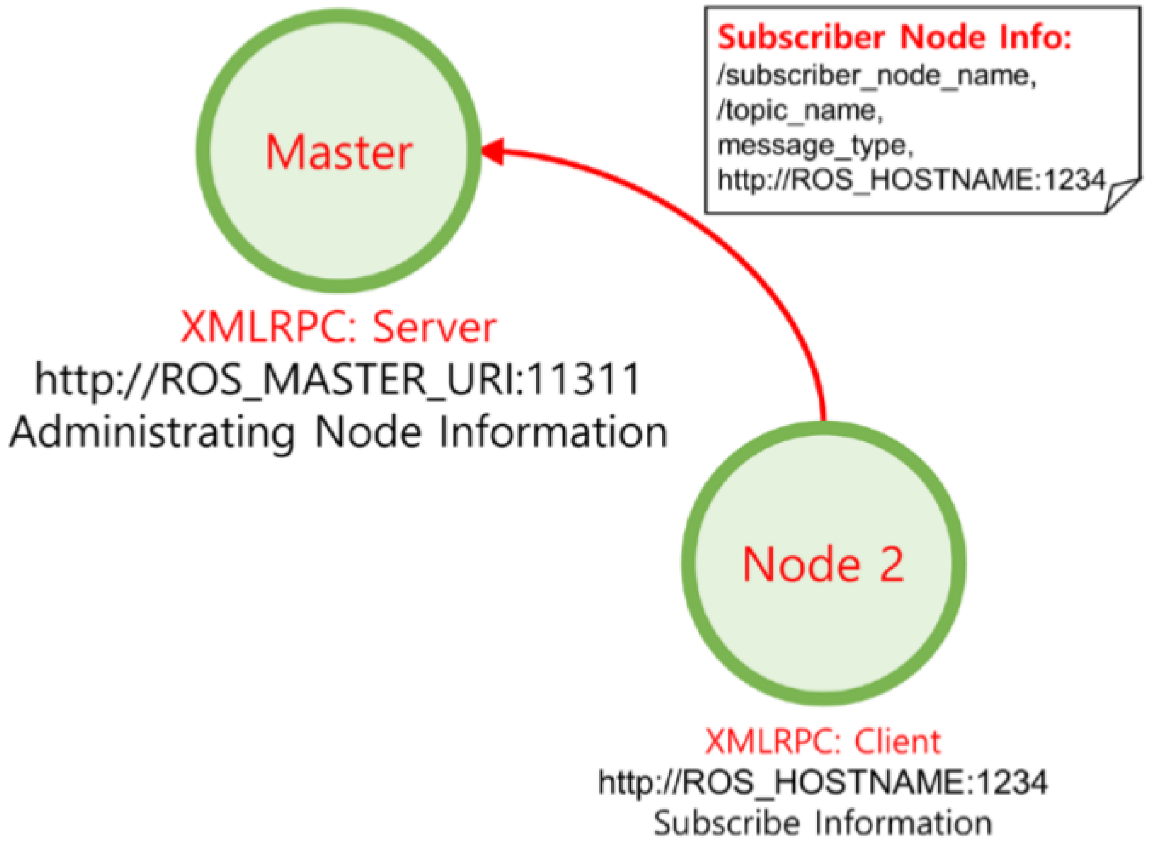
- robots may have to perform several (computationally intensive) tasks together
- hardware decoupling allows to distribute such tasks on dedicated hardware (e.g., Nvidia Jetson for GPUs)
- moreover, robots are <u>hardware</u> and this architecture allows to easily interface control boards for sensors, motors, etc.. (e.g., Arduino)

ROS on multiple platforms



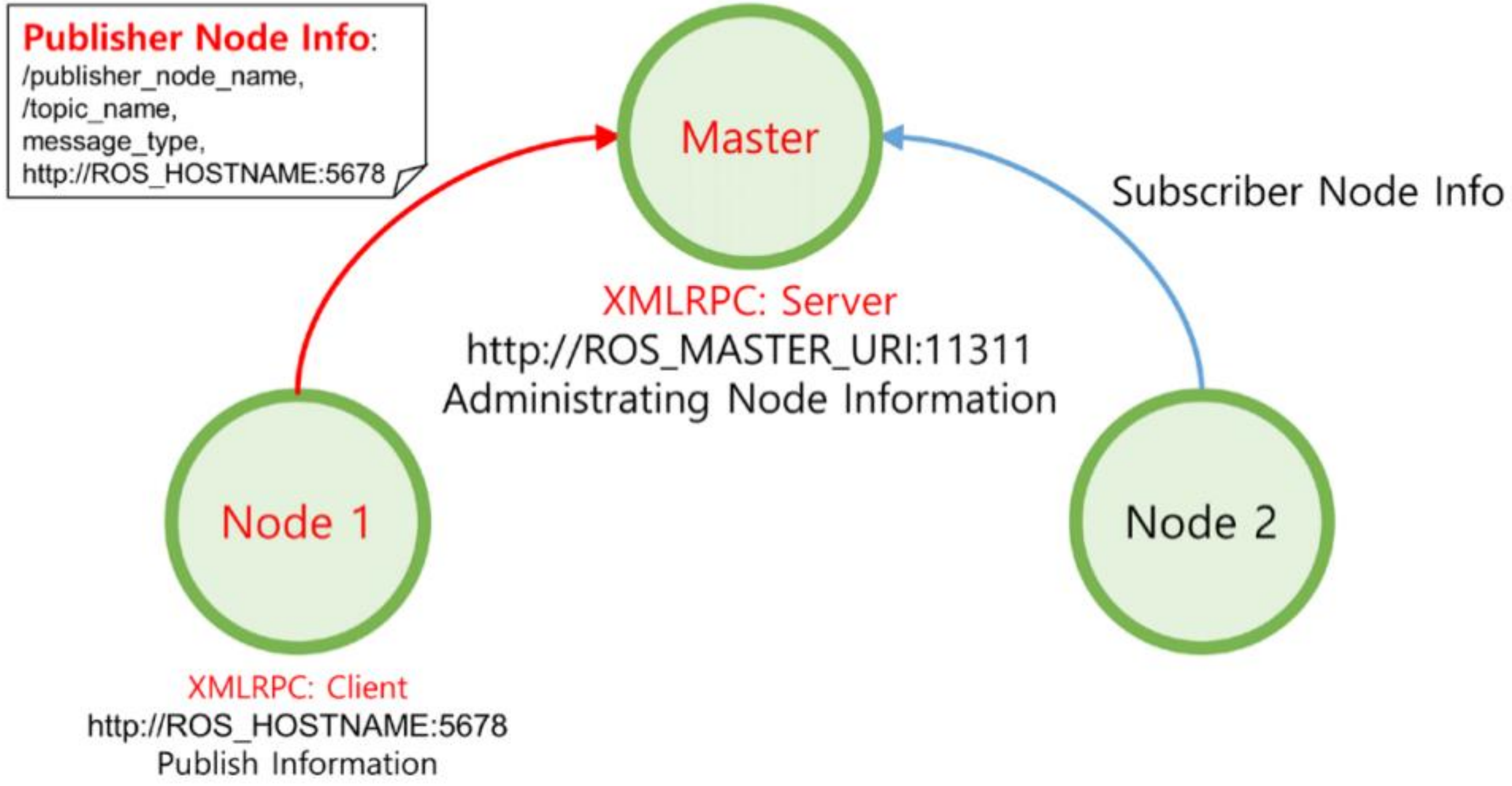
- as ROS is a middleware, computation can be distributed across different OS
- however, this *in practice* is far than ideal
- OS independence is de-facto provided for linux-based and embedded systems.
- rule-of-thumb: use Ubuntu (not all versions either!)

Set up a ROS topic publisher/subscriber



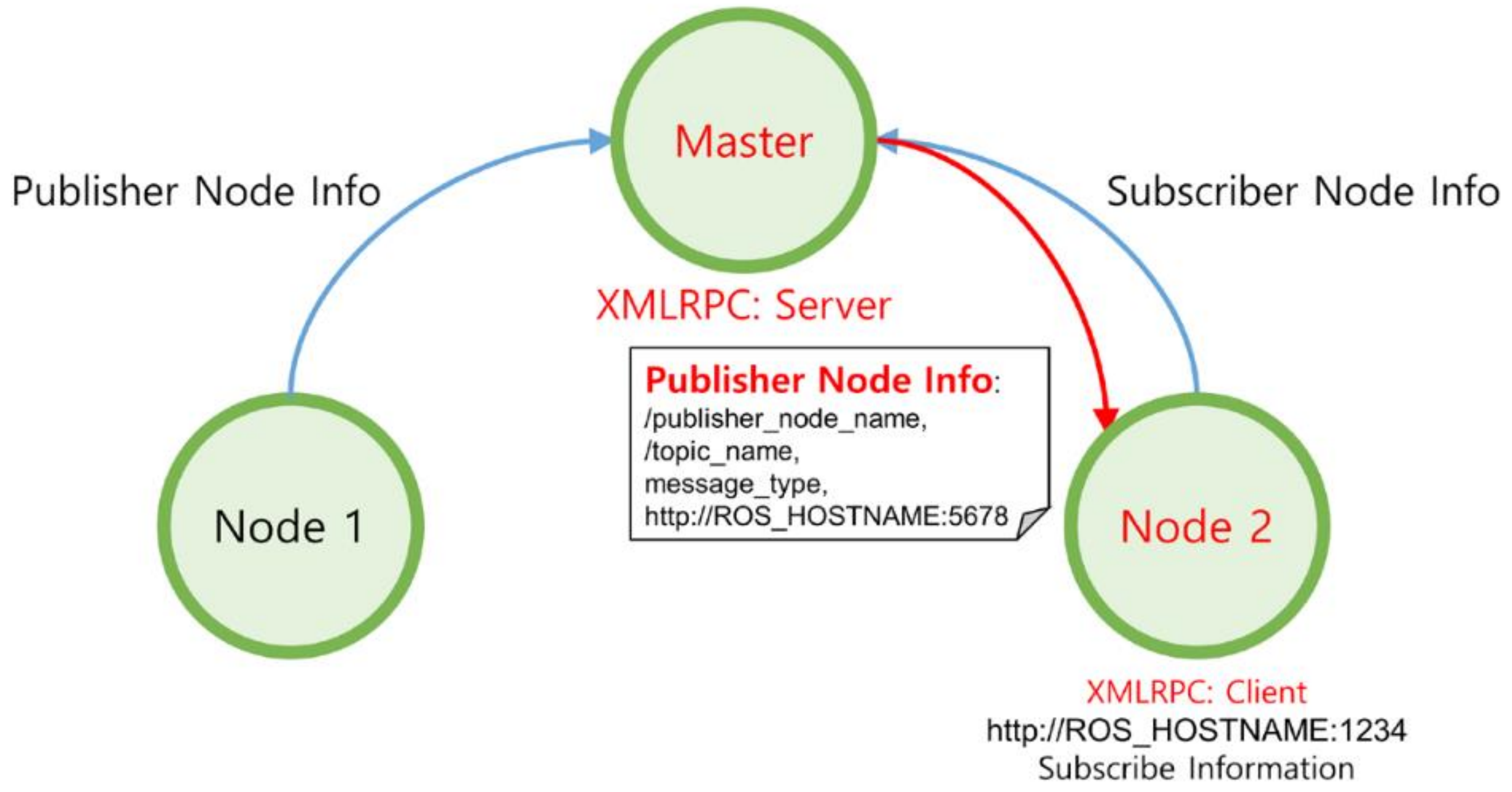
- a subscriber node registers to the ROS MASTER
- and announces its
  - Name
  - Topic name
  - Message Type
- communication is performed using XMLRPC

Set up a ROS topic publisher/subscriber



A publisher node now registers to the
ROS MASTER

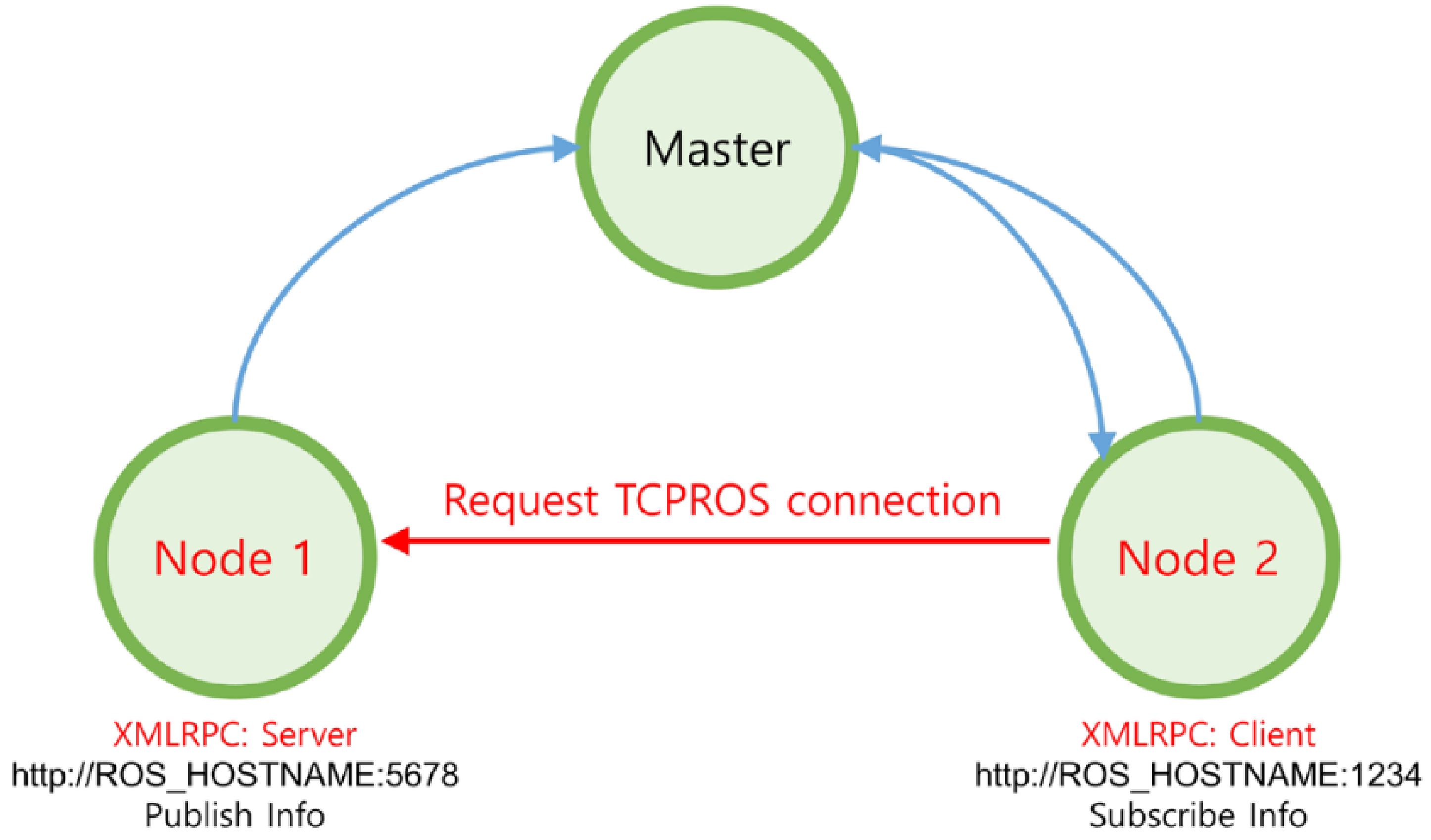
Set up a ROS topic publisher/subscriber



Publisher Node Info:
/publisher_node_name,
/topic_name,
message_type,
http://ROS_HOSTNAME:5678

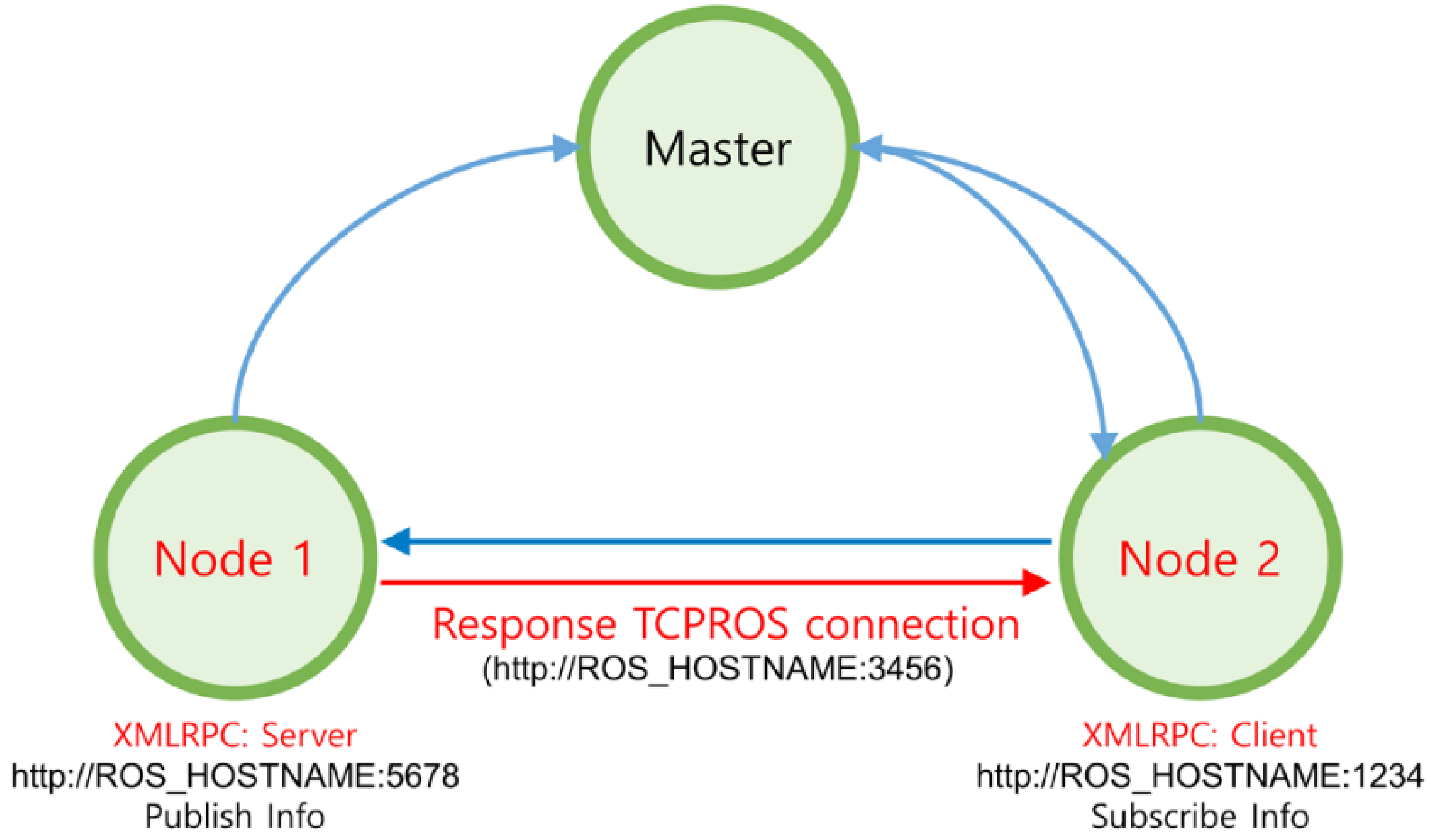
The ROS MASTER distributes info as all subscribers that want to connect to the topic and to the publisher node

Set up a ROS topic publisher/subscriber



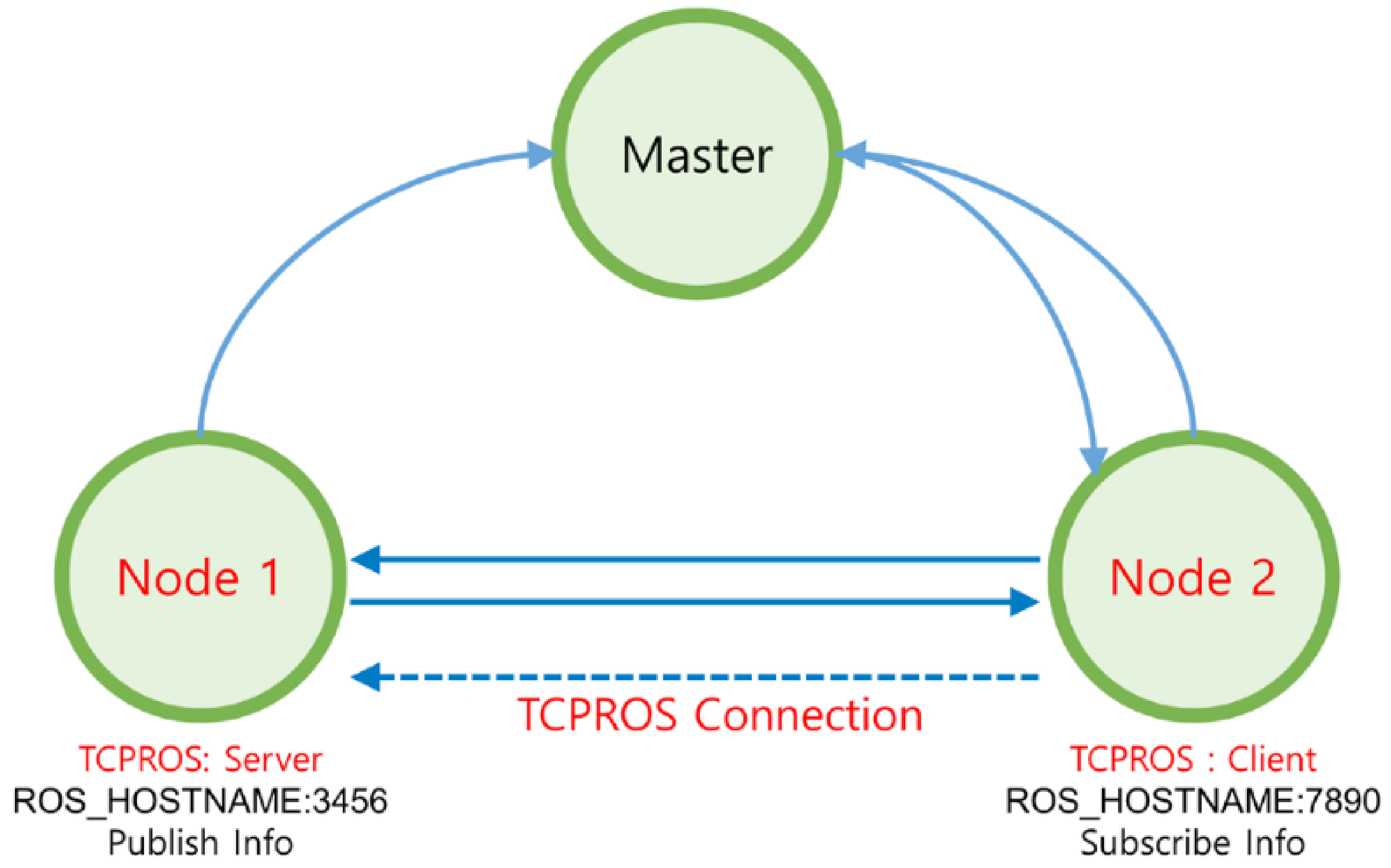
The subscriber node requests a direct connection to the published node and transmits its information to the publisher node

Set up a ROS topic publisher/subscriber



The publisher node sends the URI address and port number of its TCP server in response to the connection request.

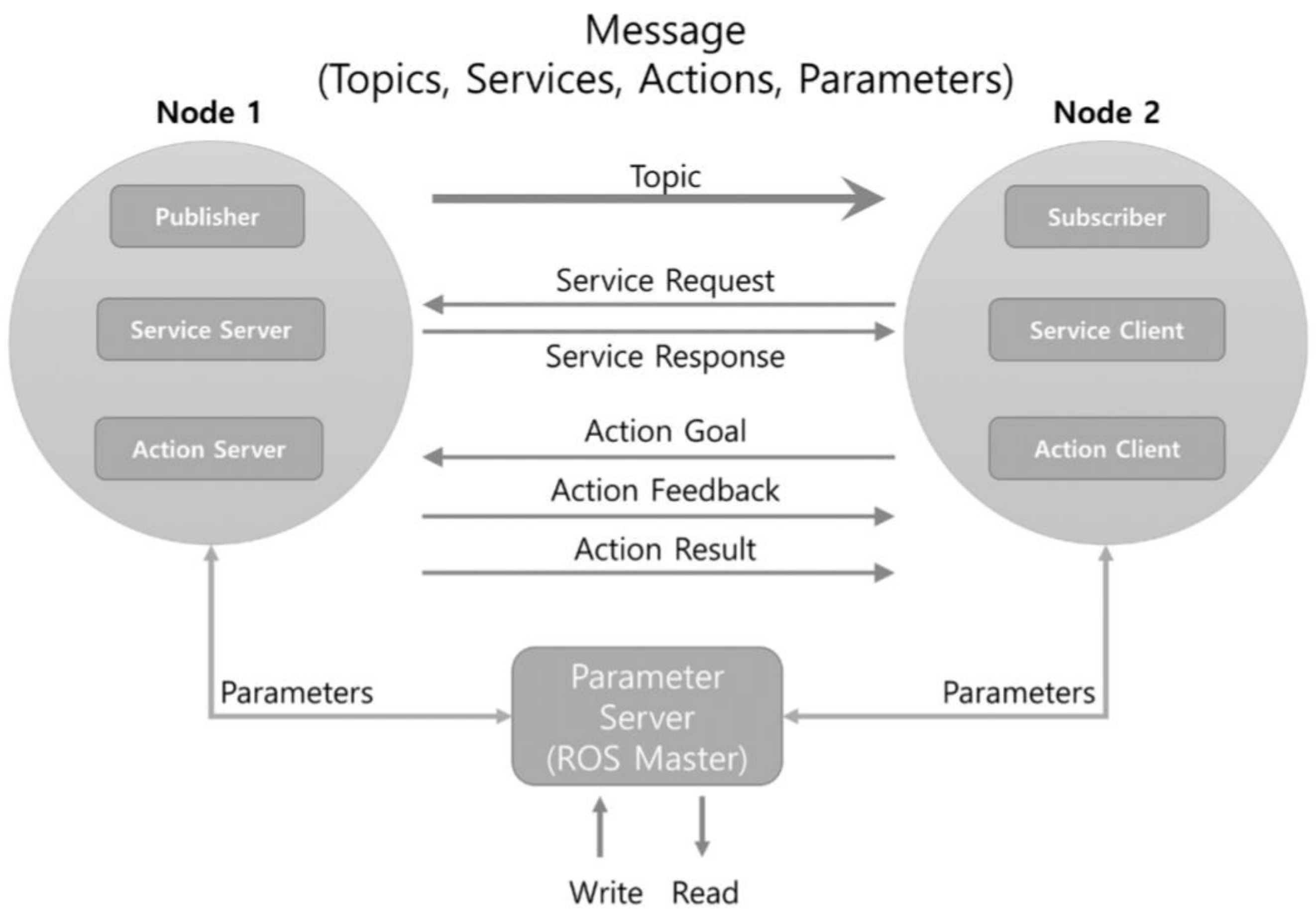
Set up a ROS topic publisher/subscriber



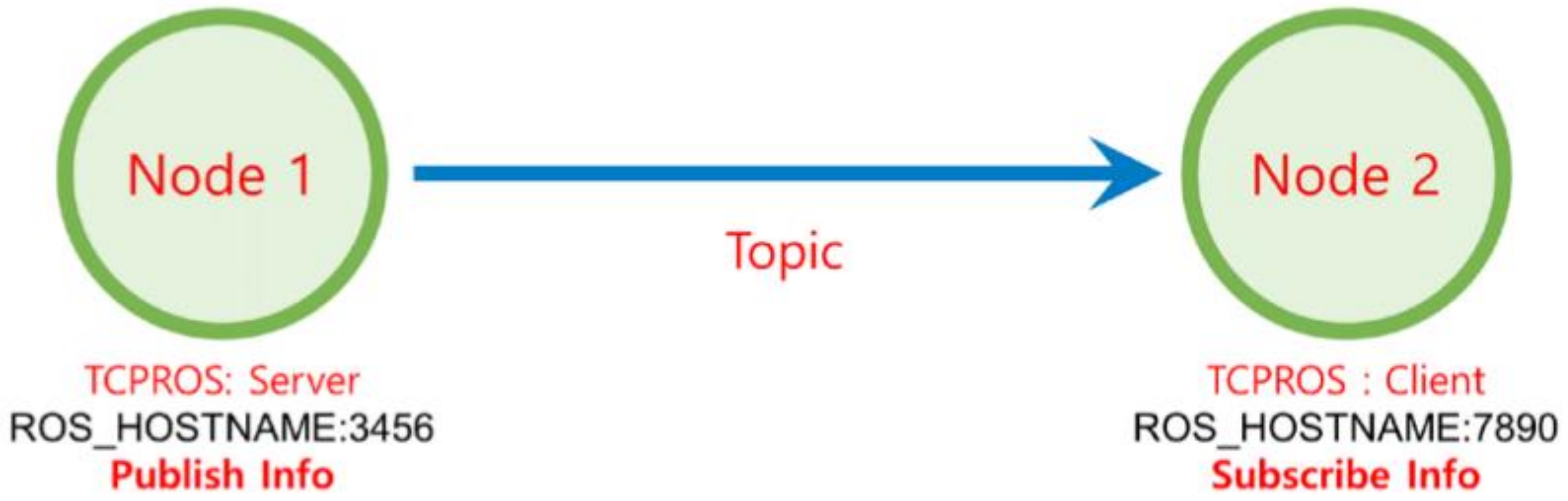
At this point a direct connection between publisher and subscriber node is established using TCPROS (TCP/IP based protocol)

Communication among nodes

After communication between nodes is established, ROS provides 3 types of interactions
- Topics
- Services
- Actions

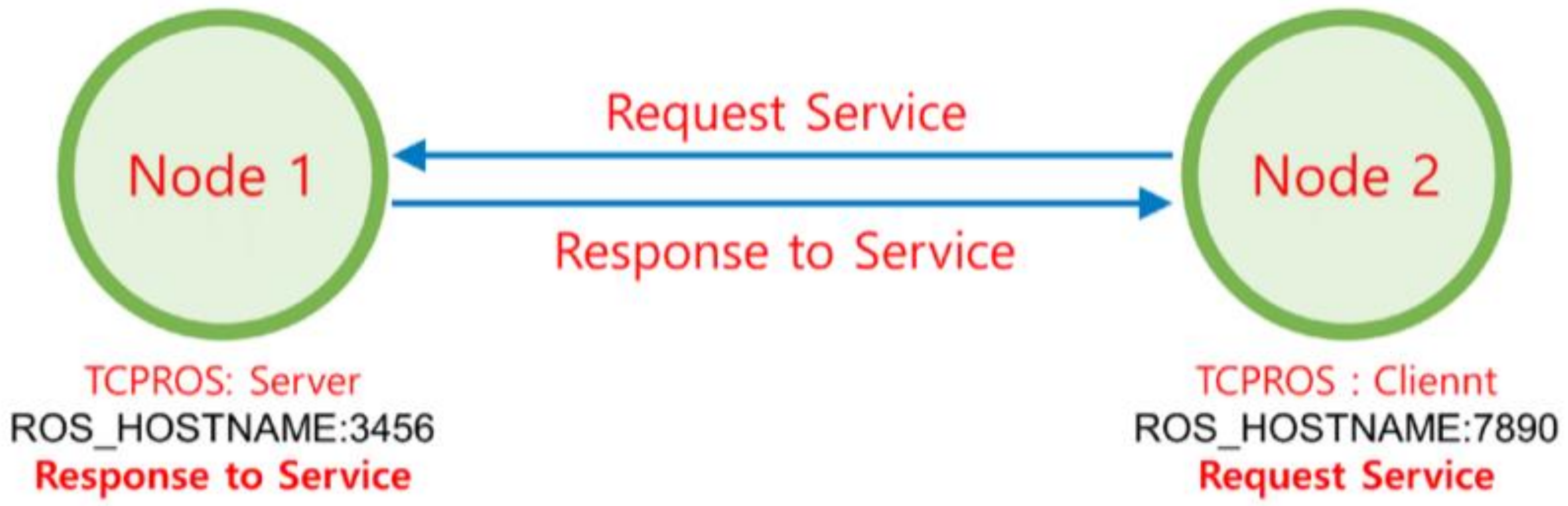


Communication among nodes



- The standard communication mechanism is using ROS topics.
- Nodes can have multiple topics
- Nodes can even use topics for internal communication
- Continuos -loop()- or one-shot (e.g. when data are ready)

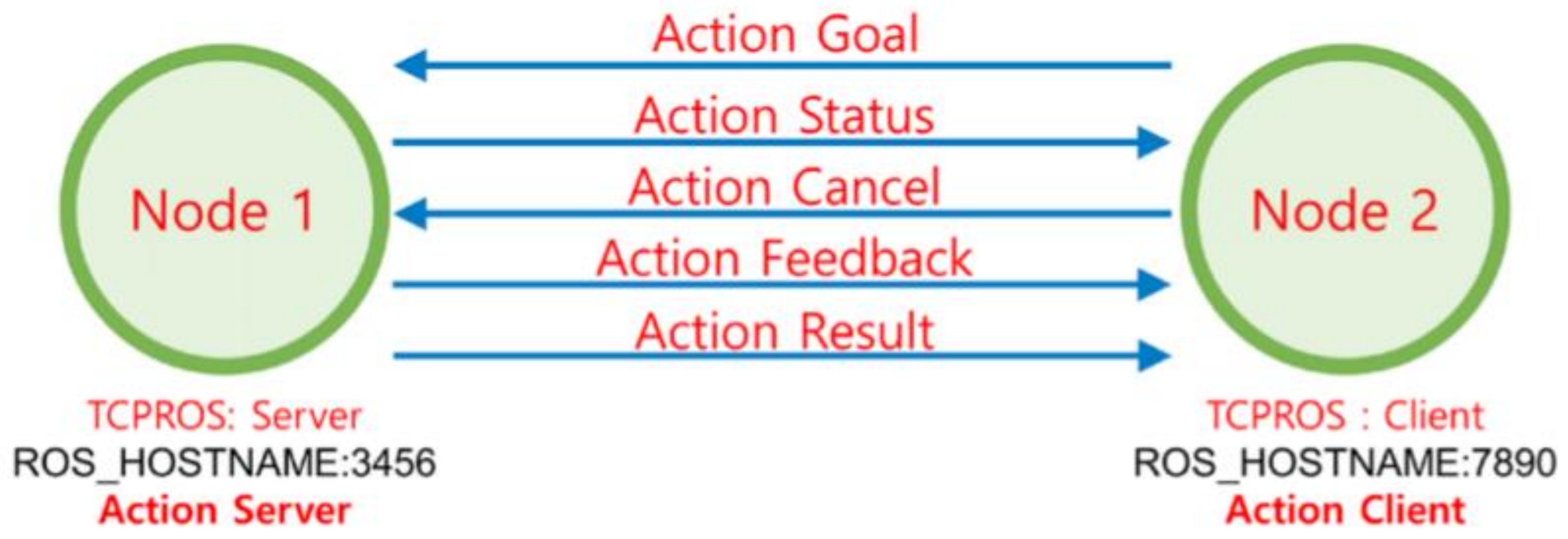
ROS *Services*



- ROS services are synchronous request from one node to another.
- Request/Reply mechanism.

A client can make a persistent connection to a service, which enables higher performance at the cost of less robustness to service provider changes.

ROS *Actions*



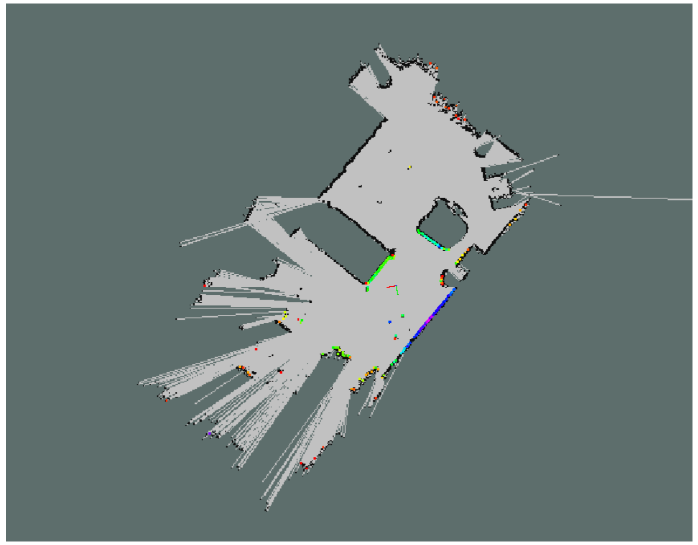
If the service takes a long time to execute, the user might want the ability to cancel the request during execution or get periodic feedback about how the request is progressing.
Action Services are for these tasks.

- ROS services are asynchronous request from one node to another.
- Request/Reply mechanism, with feedbacks and the possibility to cancel the request.

ROS *parameter server*

The parameter server is a shared, multi-variate dictionary that is accessible via network APIs.

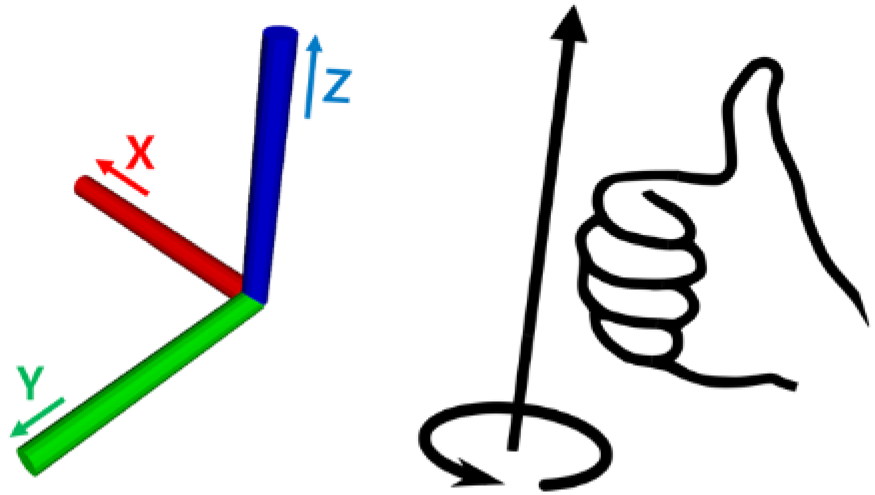
- nodes use this server to store and retrieve parameters at runtime.
- used for static, non-binary data such as configuration parameters.
- globally viewable so that tools can easily inspect the configuration state of the system and modify if necessary.



Example of params are map size/resolution and sensor configuration/settings.

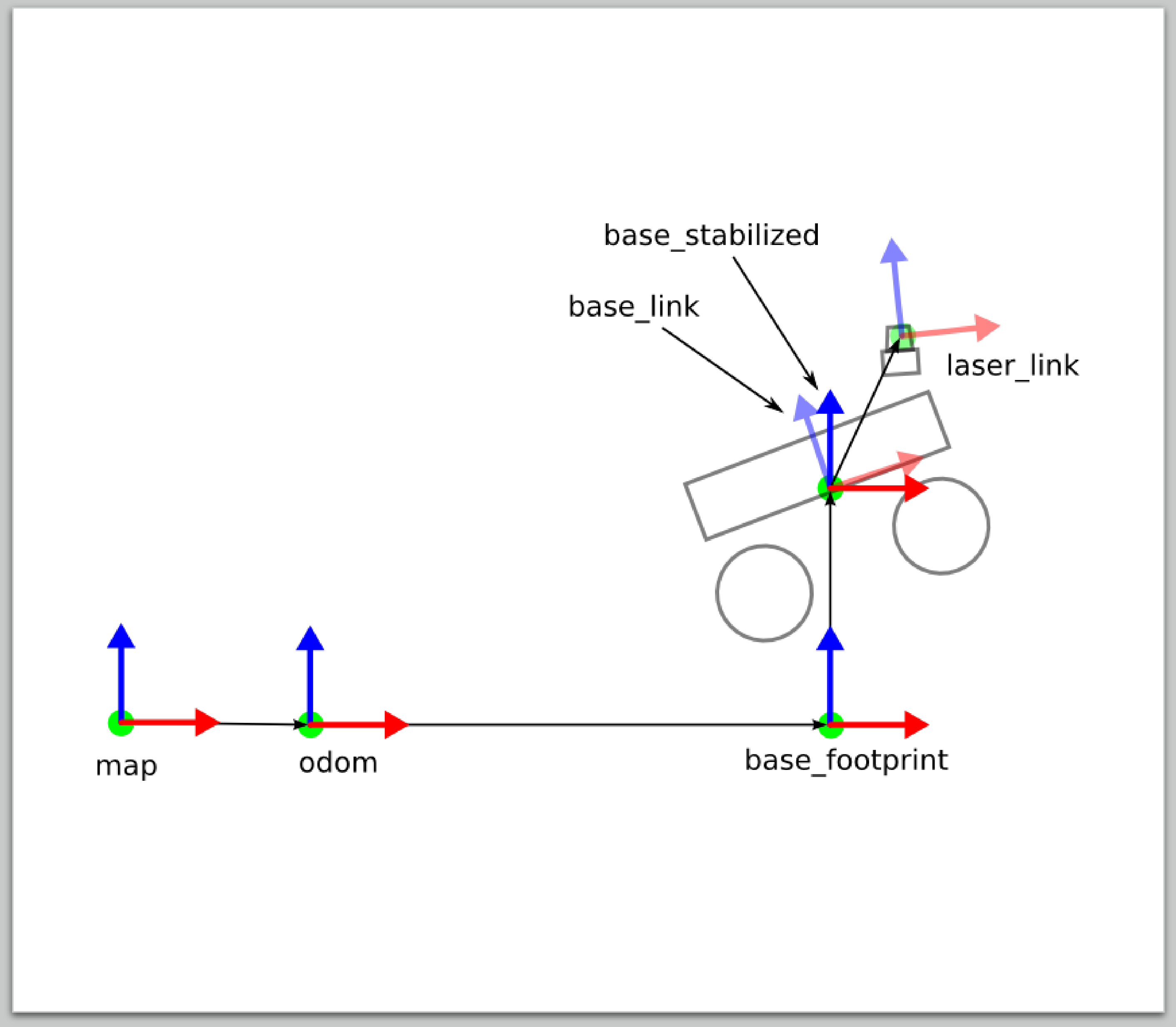
ROS *Transforms*

- in robotics programming, the robot's joints, or wheels with rotating axes, and the position of each robot through coordinate transformation are very important
- in ROS, this is represented by TF (transforms)
- TF are published with a mechanism similar to (and parallel) the one used for ROS Topics



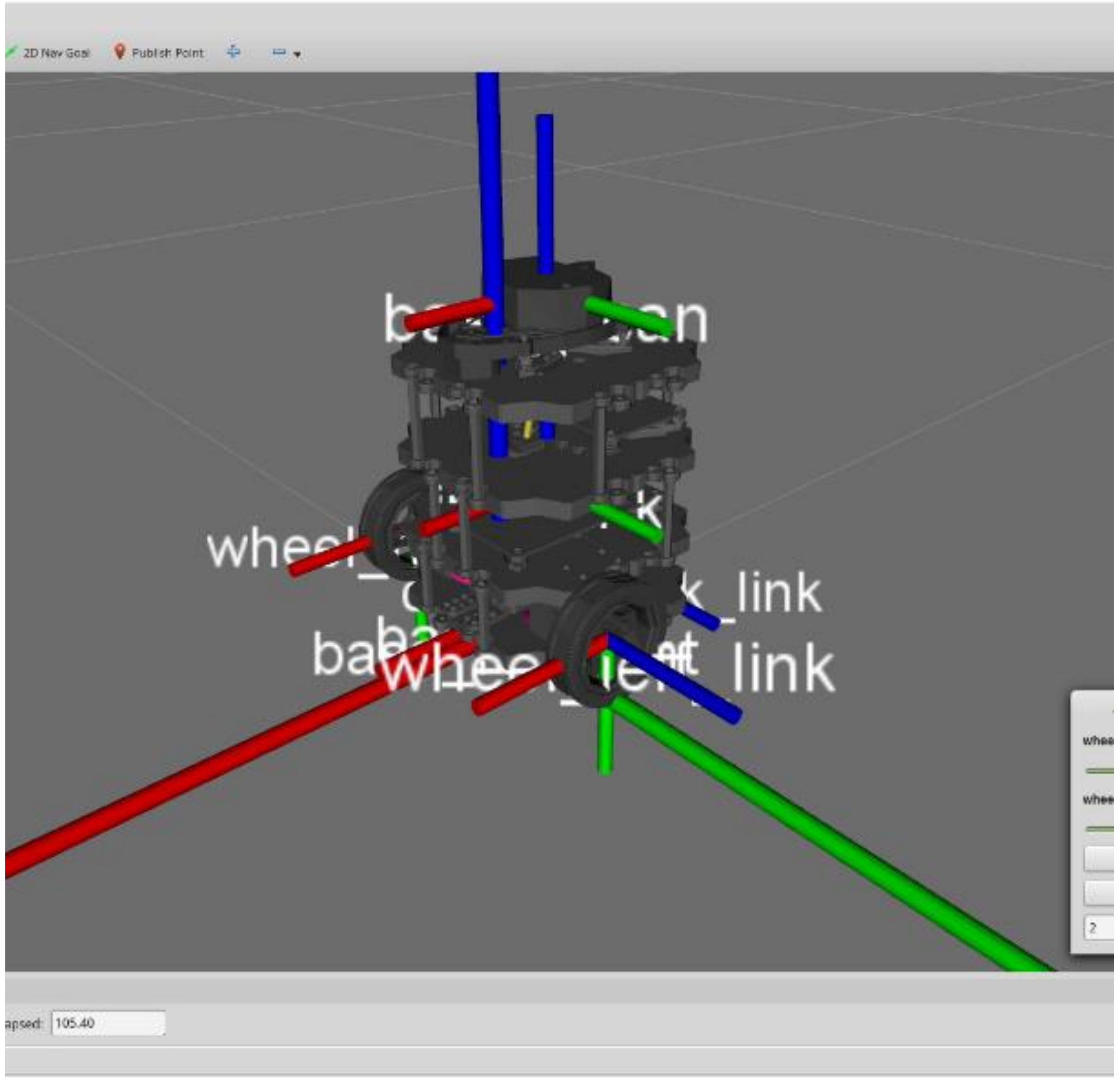
ROS *Transforms*

- all components of the robots should be connected through a chain of TF to a global reference frame (*world* or *map*)
- this is particularly important, as TFs allow the robot to project sensors onto a global reference frames



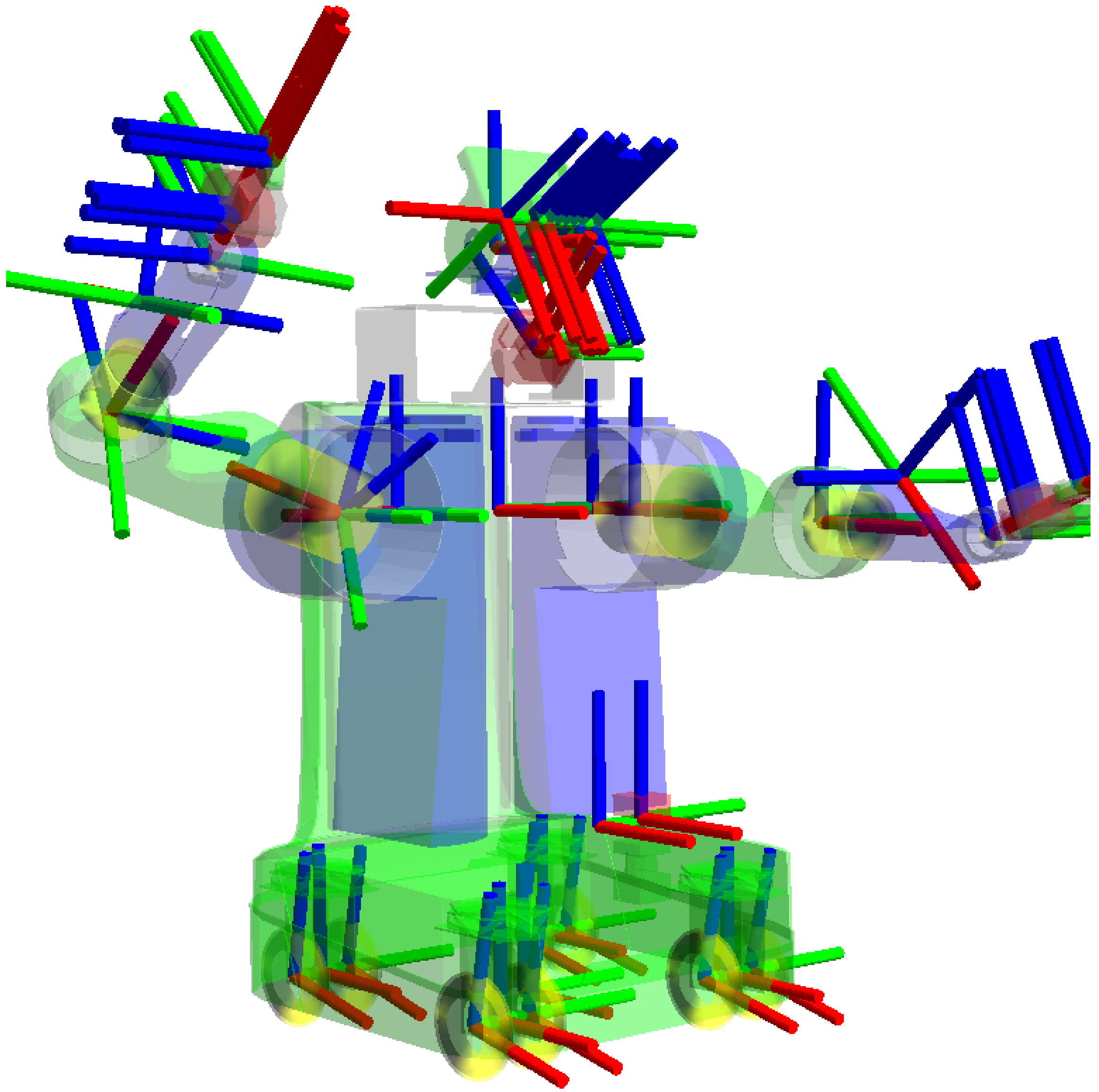
ROS *Transforms*

- some TF are static (e.g., the position of sensors w.r.t. The robot reference frame)
- some TF are dynamic and are computed real-time by nodes
  (e.g. the position of the robot in the map, the position of joints in a hand gripper)



ROS *Transforms*

- TF can become complex, especially for robot with a lot of Degrees Of Freedom (DOF) as grippers
- ROS provides visualization tools for controlling such aspects

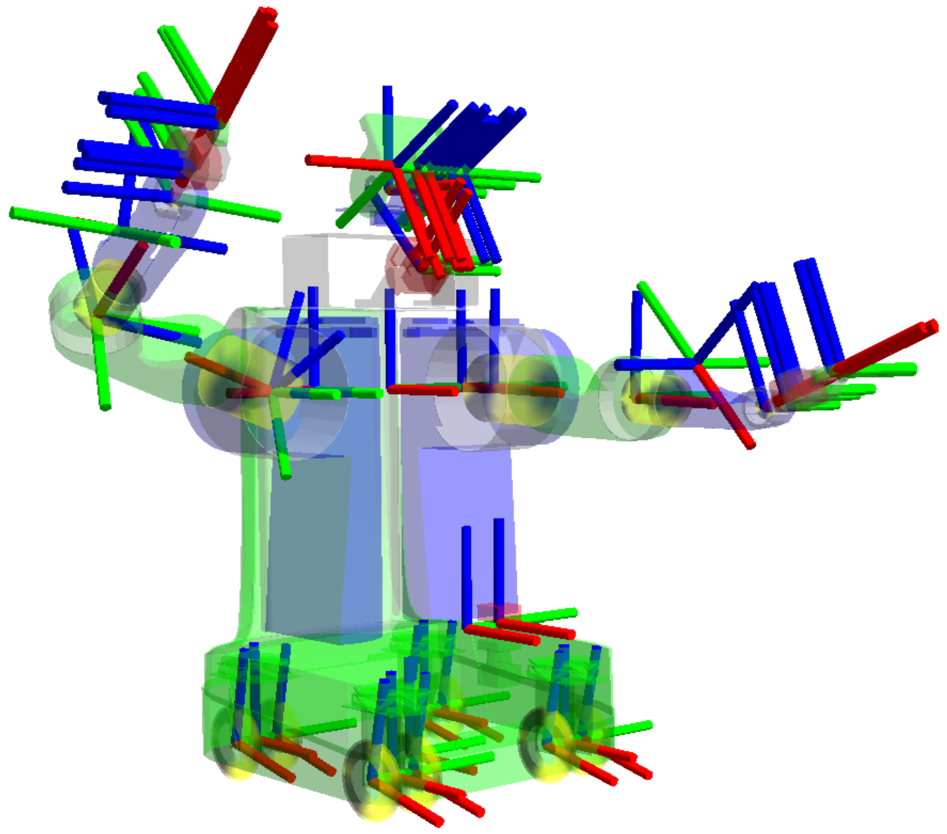


ROS

Developing toos

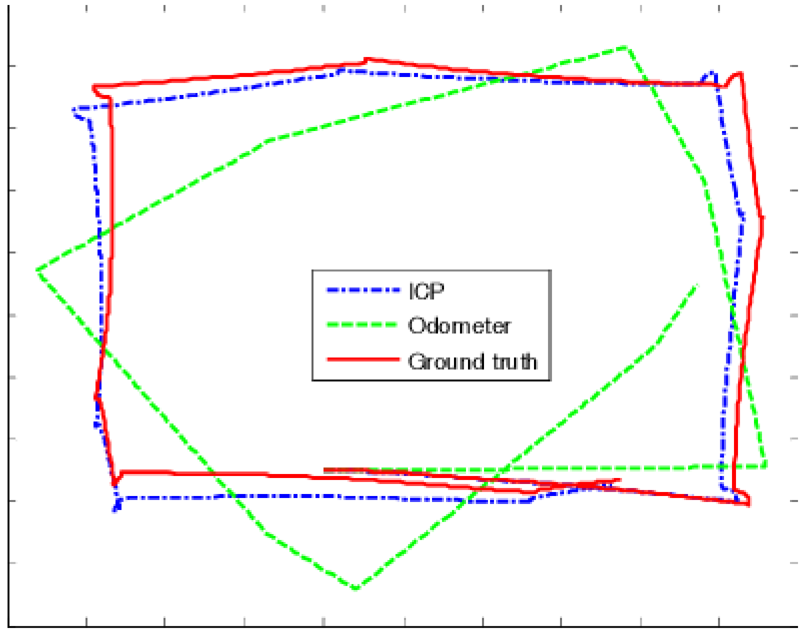
Developing a robot in ROS

- mobile robots easily became very complex objects
- issues can emerge with single components, hardware failures, integration, …
- impossible to control all possible sources of uncertainty



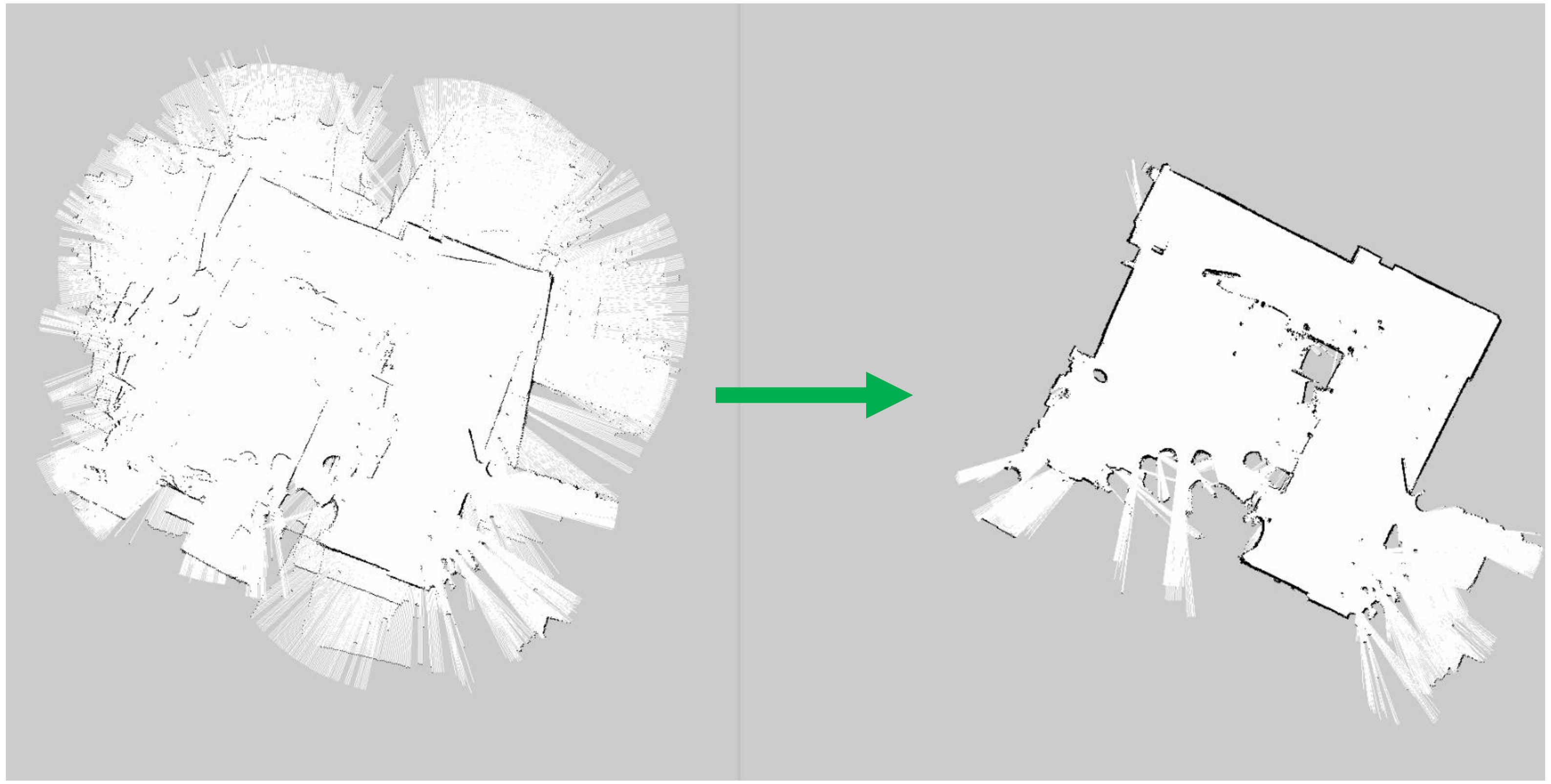
Environmental inaccuracies

- All of the robot available knowledge is based on sensors but…
- …sensors itself are (very) noisy
- odometry is the estimation of the robot motion from internal sensors (e.g. IMU or velocity)
- odometry itself is very noisy and unreliable



Reducing environmental inaccuracies

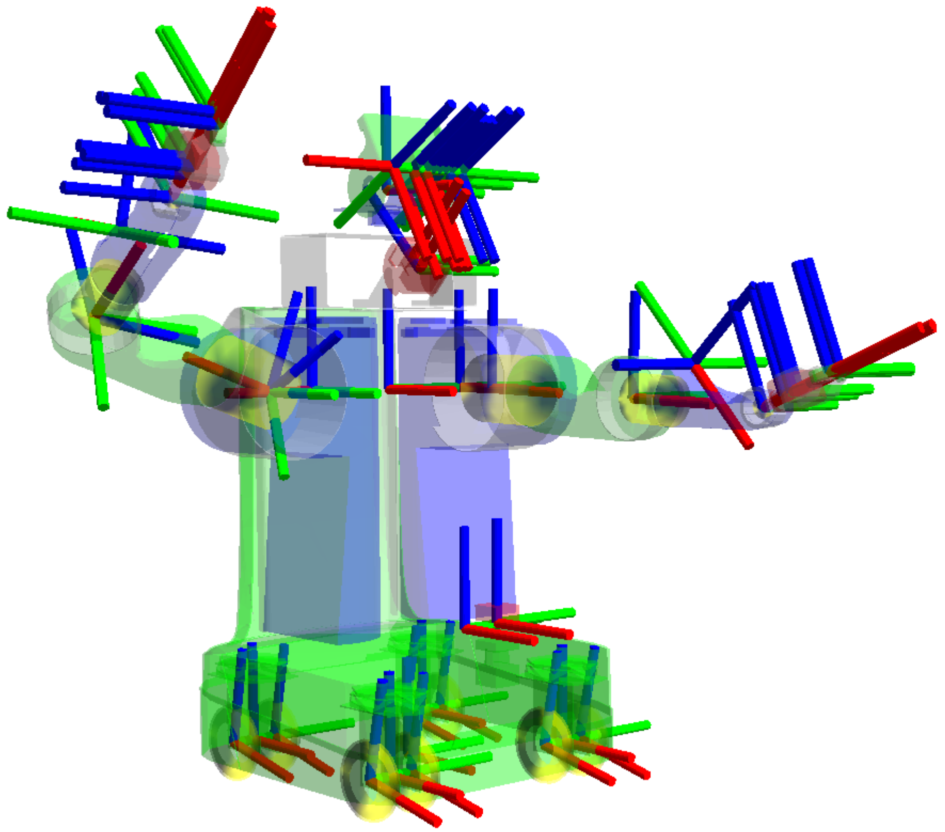
Even if assuming that there are no unexpected failures in the robot modules, some of the robot modules are designed to cope and reduces known sources of uncertainty and to integrate data together



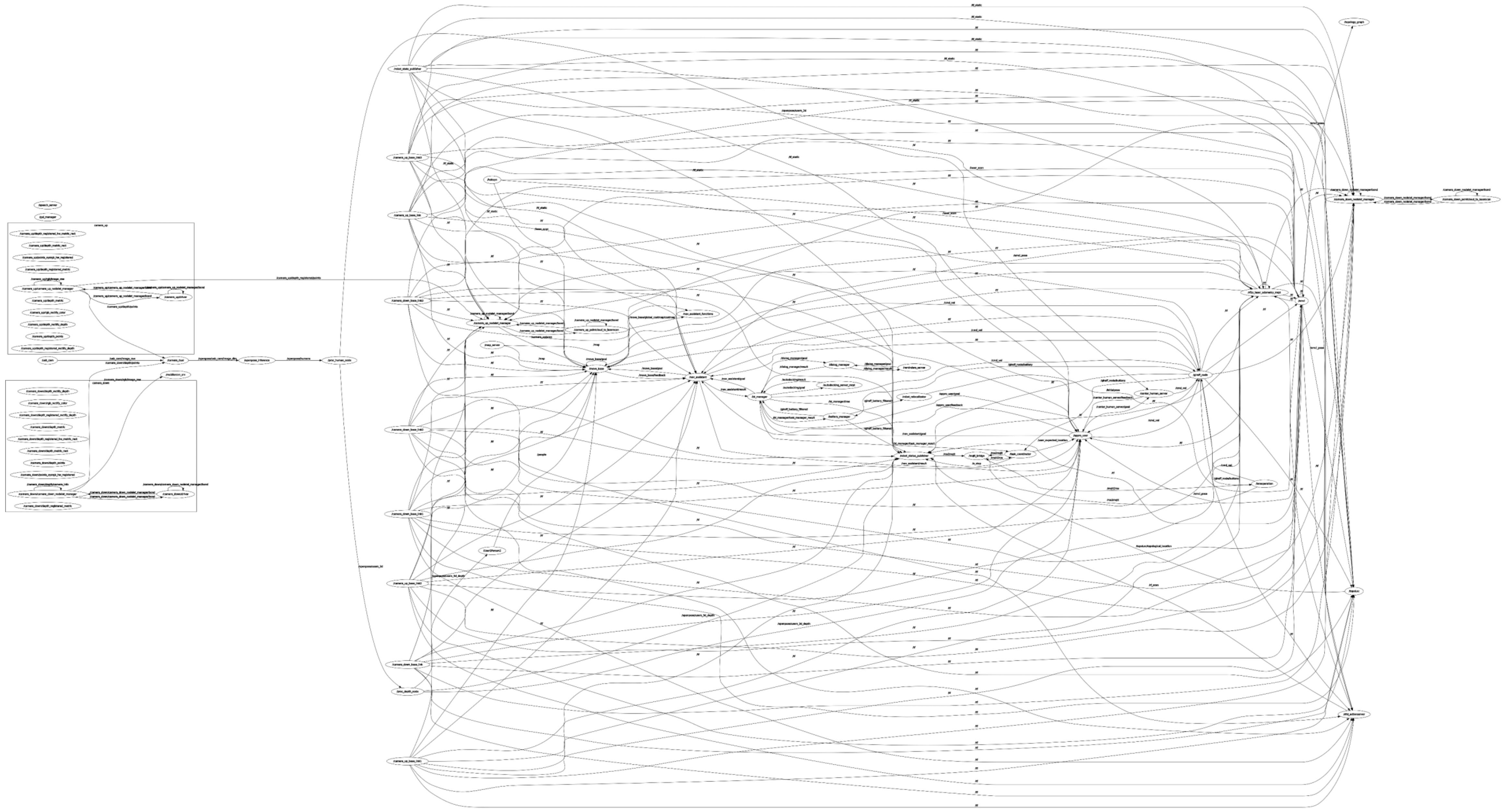
Mapping integrates sensor readings (e.g., laser range scanner) together reducing odometry error thus obtaining a valid map of the environment

Developing a robot in ROS

- Modularity and scalability of nodes and topics help in developing complex integrated system but…
- …still the resulting ROS computational graph is impossible to be analyzed at glance



ROS



The graph of ROS nodes and topics of a real robot

How to program robots then?

- A lot of components and modules integrated among them
- Sensors and robot hardware are noisy and can fail
- Impossible to control all possible sources of uncertainty

Making even a simple run with a robot can be very time consuming

How to program robots then?

- A lot of components and modules integrated among them
- Sensors and robot hardware are noisy and can fail
- Impossible to control all possible sources of uncertainty

Developing and integrating a new functionality into a pre-existing robot can be difficult too

Why ROS is useful

- A lot of components and modules integrated among them
- Sensors and robot hardware are noisy and can fail
- Impossible to control all possible sources of uncertainty

- Use packages provided by the community
- Split computation into nodes
- Test in advance in simulations
- Use pre-recorded sensor inputs
- Visual inspection tool for monitoring all of the robot aspects

Why ROS is useful

- A lot of components and modules integrated among them
- Sensors and robot hardware are noisy and can fail
- Impossible to control all possible sources of uncertainty

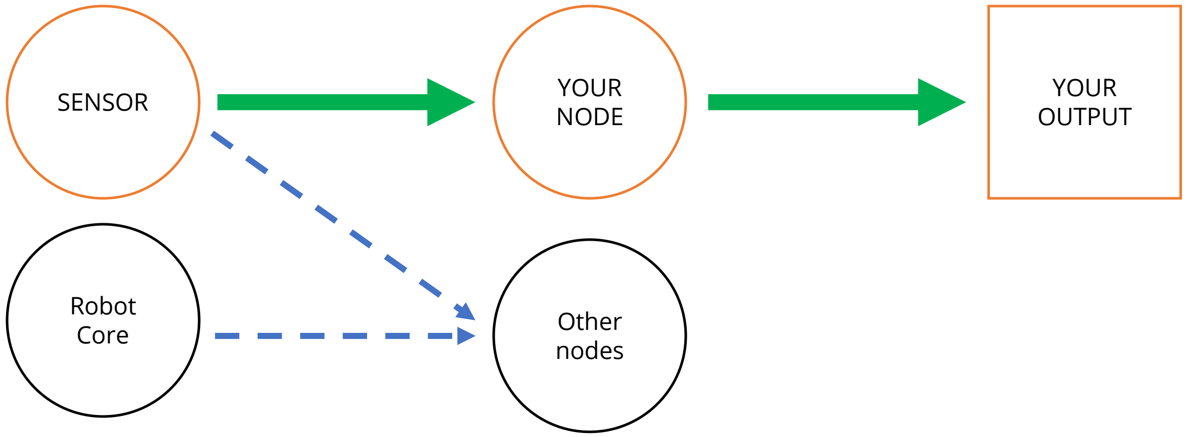
- Use packages provided by the community
- Split computation into nodes
- Test in advance in simulations
- Use pre-recorded sensor inputs
- Visual inspection tool for monitoring all of the robot aspects

Why ROS is useful

- A lot of components and module integrated among them
- Sensors and robot hardware are noisy and can fail
- Impossible to control all possible sources of uncertainty

- Use packages provided by the community
- Split computation into nodes
- Test in advance in simulations
- Use pre-recorded sensor inputs
- Visual inspection tool for monitoring all of the robot aspects

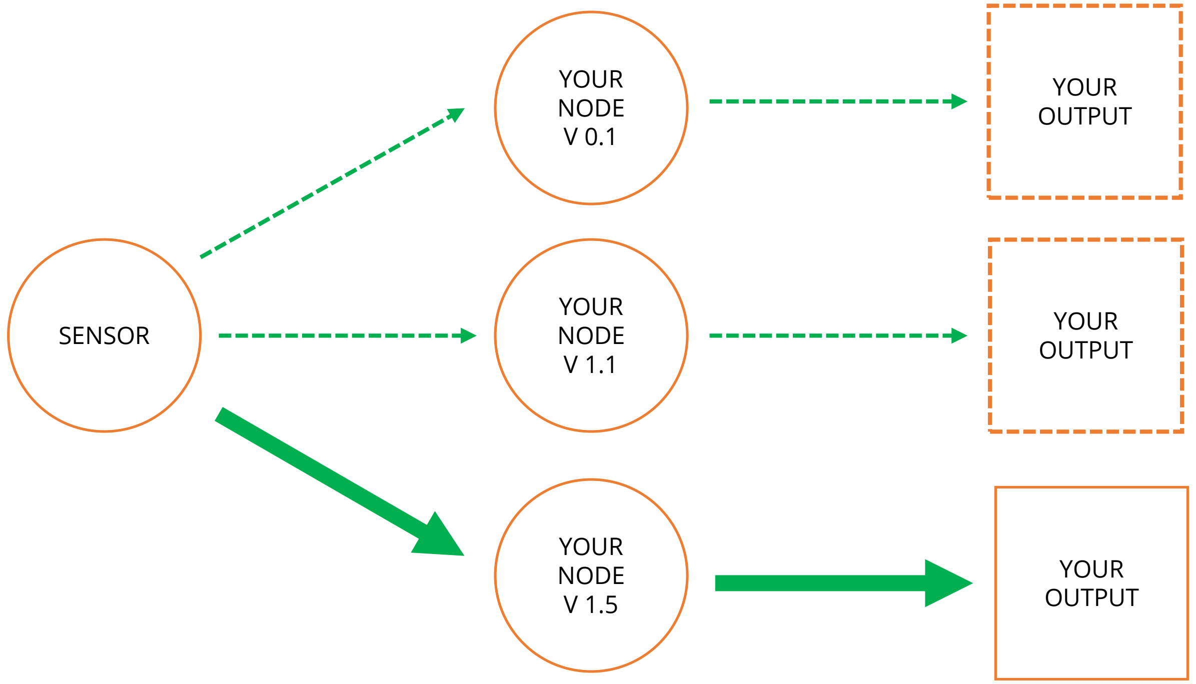
An example: writing your own *Node*



Assume that you have to implement an algorithm for a robot, e.g. a module that <u>detects narrow passages</u> that are challenging for the robot navigation.

ROS allows you to develop just your node while using a pre-built robot set up (from the community) and to use pre-existing robot functionalities (remote commands, mapping, odometry, sensors parsing, mapping, localization)

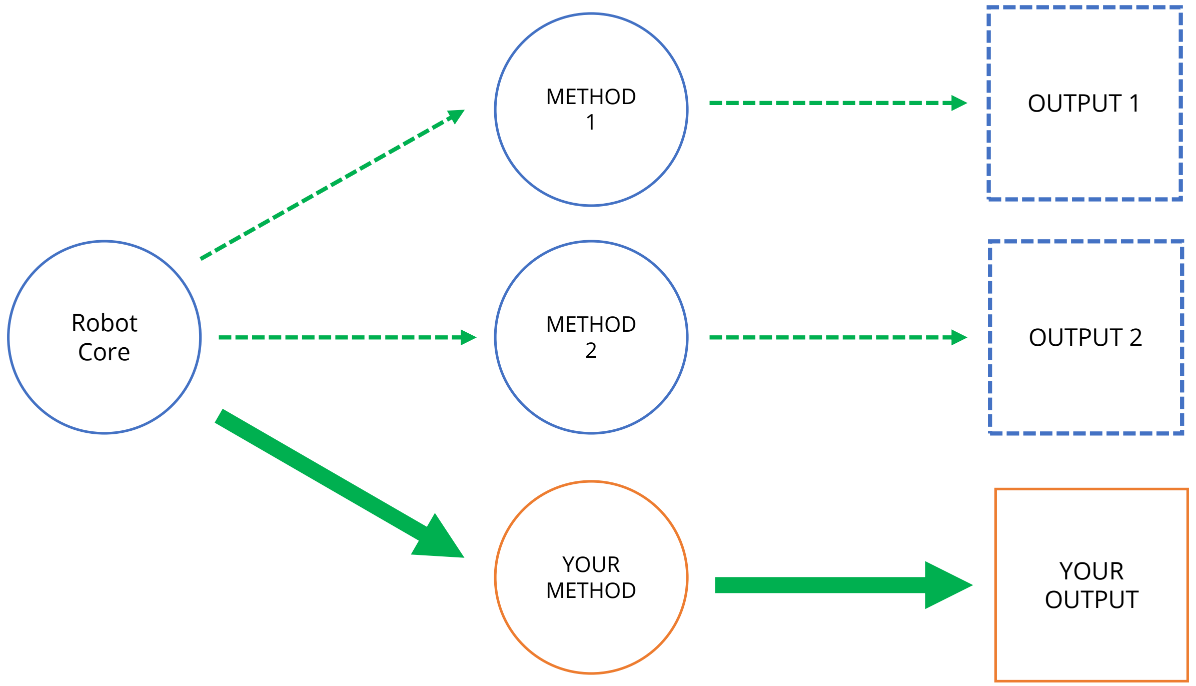
An example: writing your own *Node*



Probably you will develop several version of your node. The first one will have bugs and wont work, then a new version is produced with improvements, …

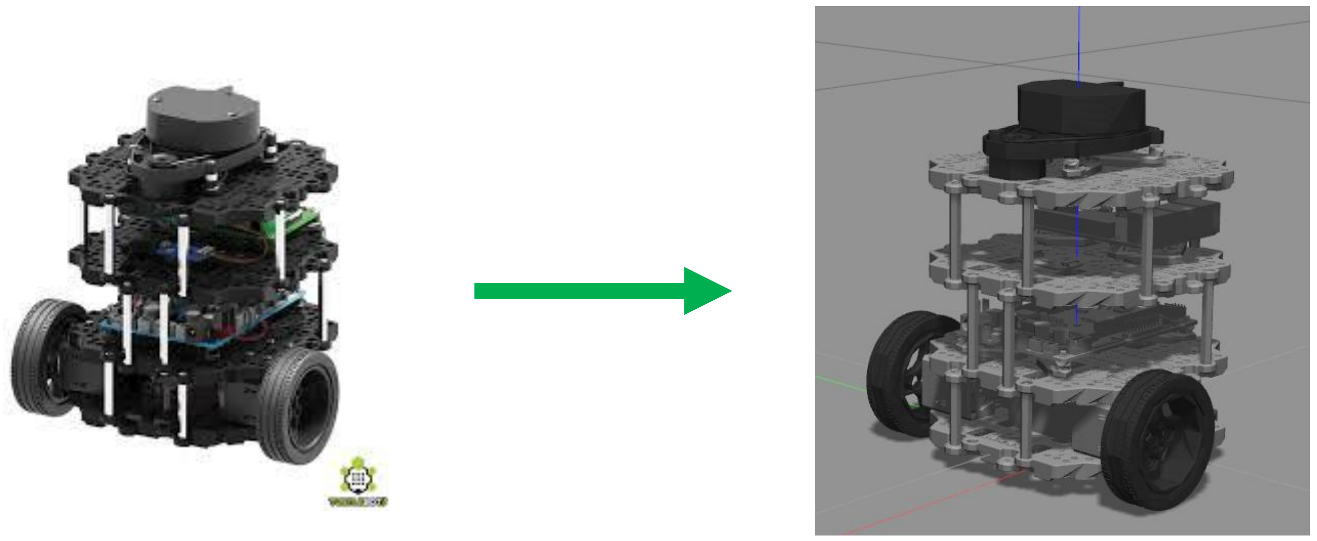
… and testing the result of different version together could be a good idea

An example: writing your own *Node*



Using *nodes* and *topics* it is also straightforward to test several methods (to see what is more useful for your robot) or to compare the results of your method with the one available to the community (and release it).

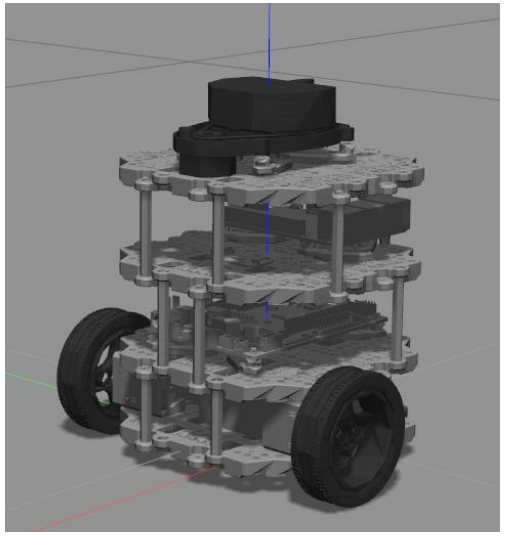
ROS and Simulations



One of the most powerful tool that ROS have is the possibility to use integrated 2D and 3D ROS simulations.

ROS simulation nodes replaces sensor drivers and allows to test the same algorithm with real robot and simulations

ROS and Simulations



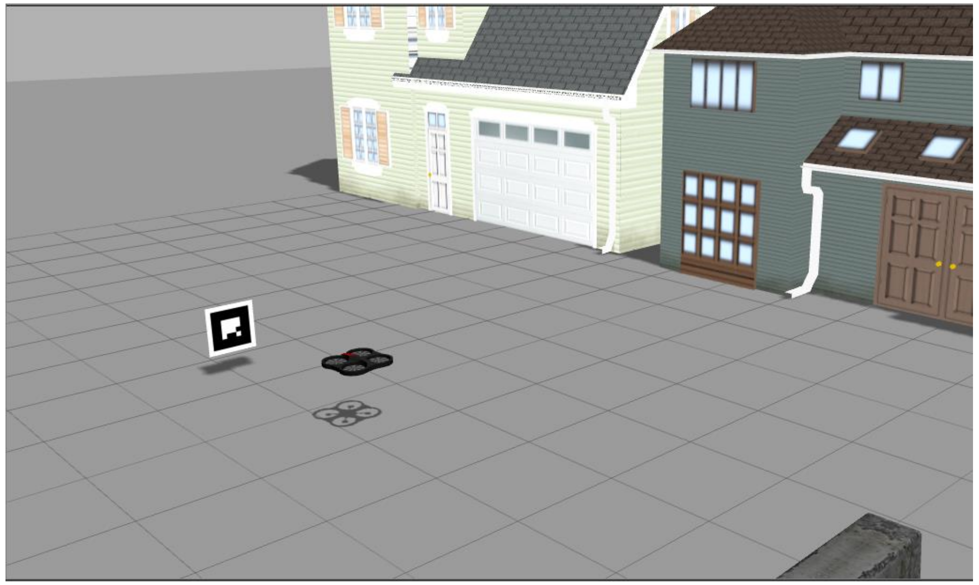
(iterative)
development of
a new method in simulations

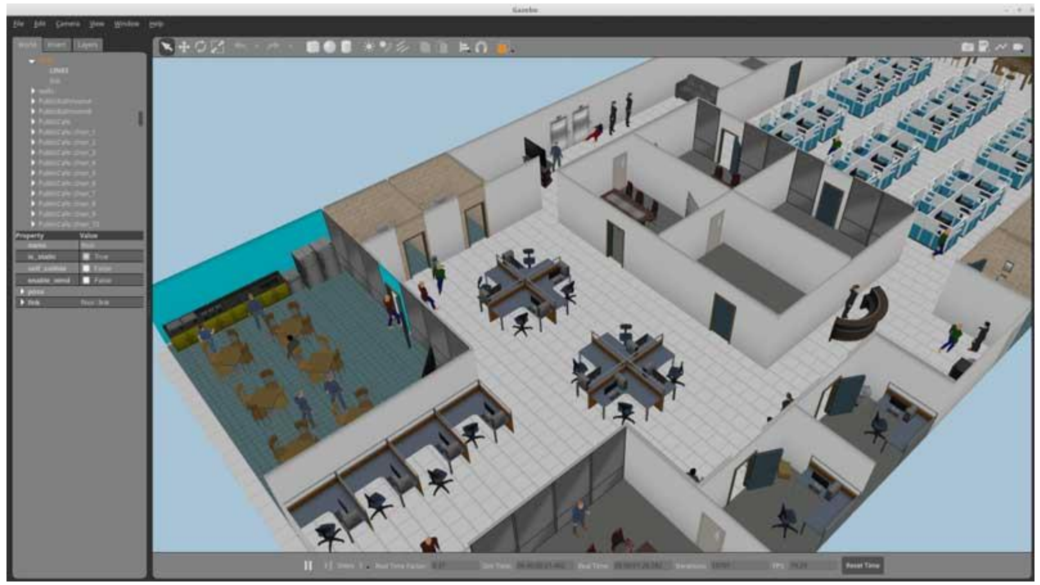
Test and validation with real robots

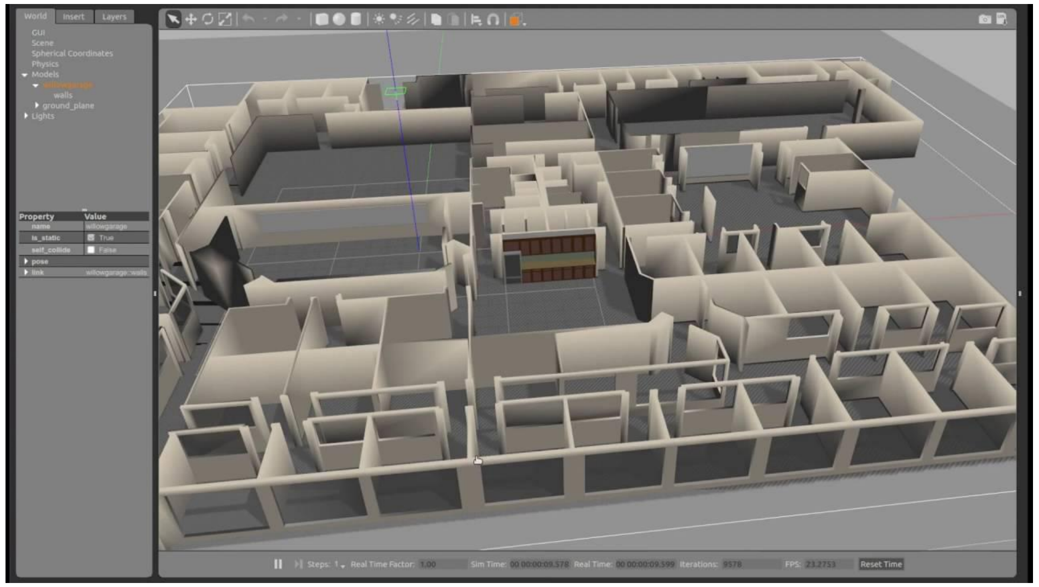
- Robots in ROS simulations are modeled starting from their real counterpart.
- This allow a fast transitioning from tests performed with simulation and with real robot just changing a few lines of code.

Simulations with GAZEBO

Gazebo (3D) is the most popular and used ROS simulation tool, and it allows to simulate mobile robots, UAVs, manipulators, indoor and outdoor environments, …



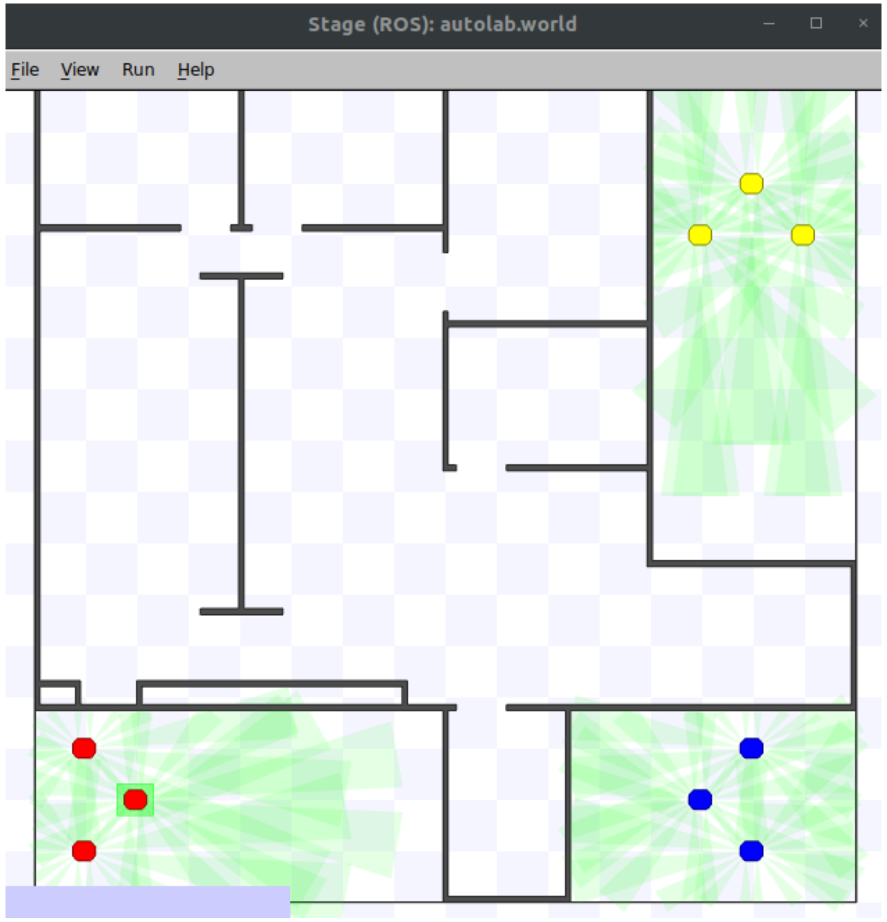




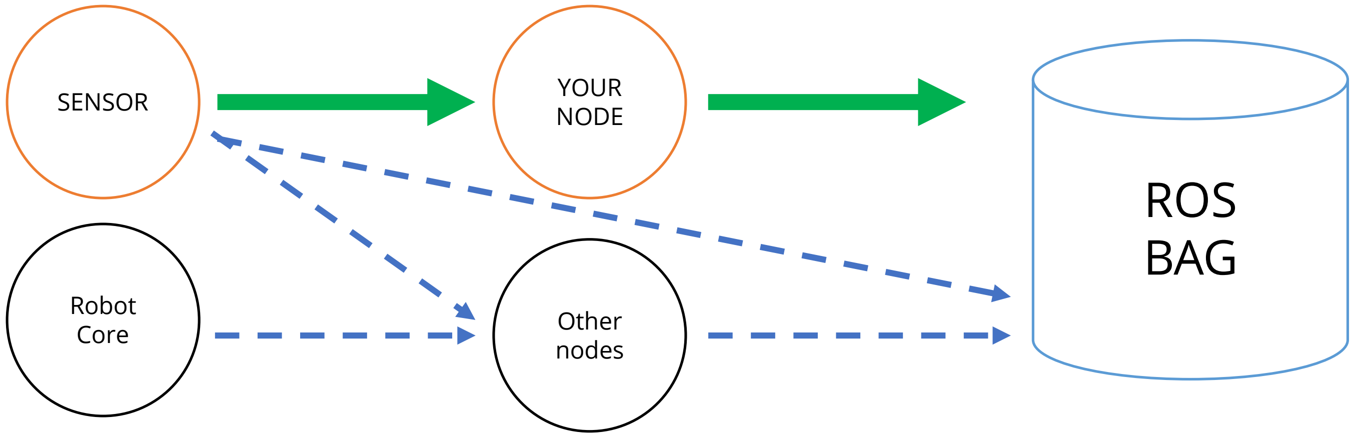
Simulations with STAGE

Stage is a simple 2D robot simulator.
It is useful for testing multi-robot systems, swarm robotics (even 10k robots) and for testing robotic tasks that require a higher level of abstraction.

Besides Gazebo and Stage, ROS can work with many commercial and open-source robotic simulation tools.

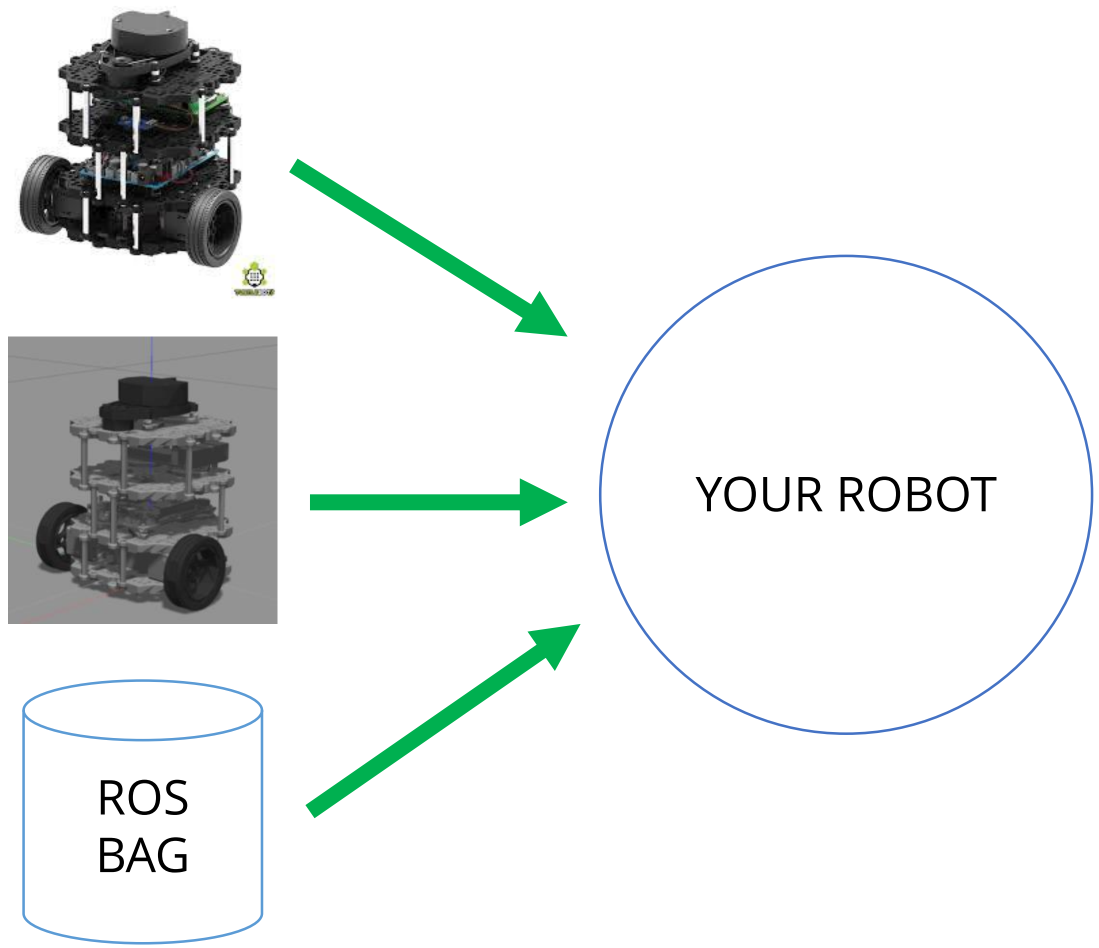


Another solution – use datasets



- Another important tool embedded in ROS is the possibility to record robot runs (in simulations or with real robots and to replay them).
- **ROS bags** store a time-stamped serialized version of all selected topics (sensors, nodes outputs, …)
- Different algorithms can be tested with the same input to test improvements
- Bags can be reproduced at 2x, 4x, 5x → speed up of development times
- Publicly available datasets are shared among the community

Wrapping up



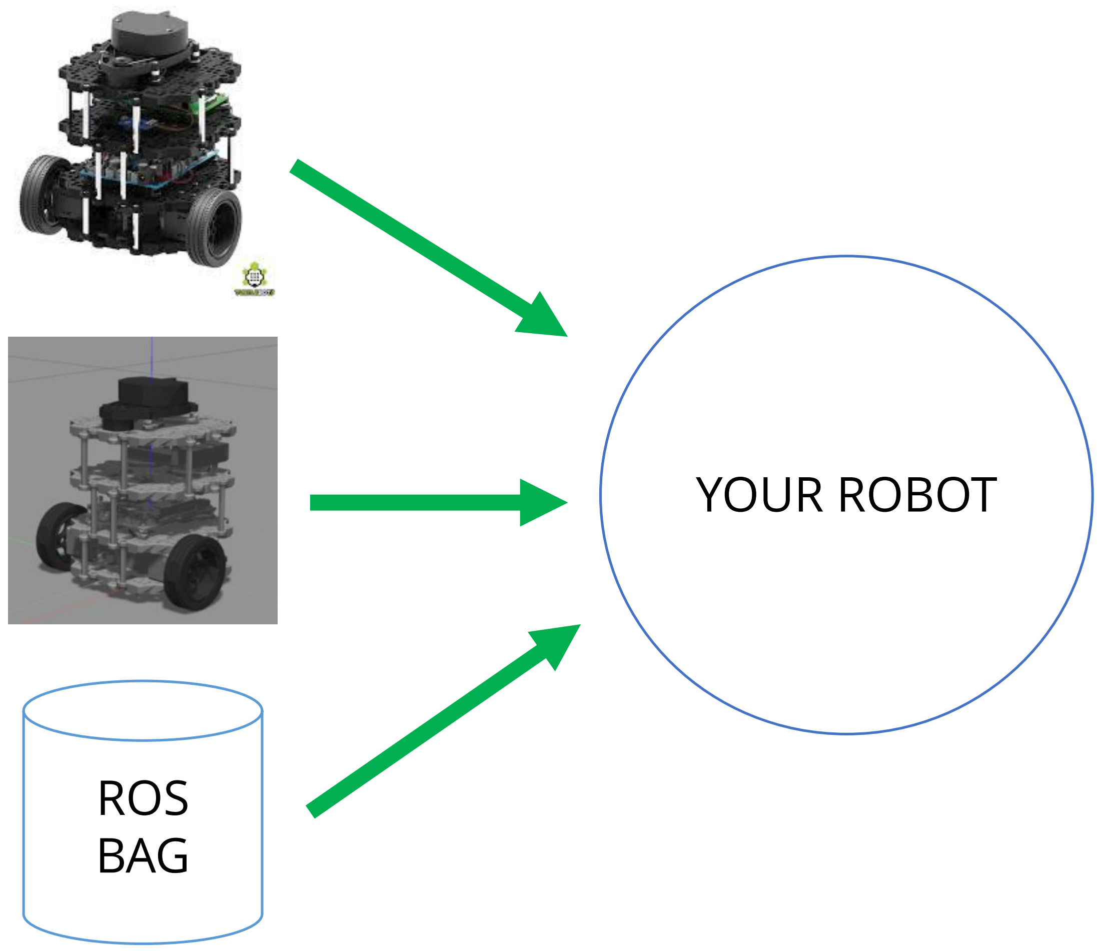
With the only constraint of using the same topics and the same msg format, you can switch between:

- real robots
- simulations
- pre-recorded data streams

without changing the other part of the robot code.

The rest of the robot code structure and of the nodes used remain the same.

Wrapping up



This has multiple (positive) side effects:

- You can focus on the specific task you want to develop
- You can develop a robot even without having a robot – use simulations
- You are not even forced to acquire no runs – just use datasets/rosbags

Debugging tool with ROS

Even using simulations or ROS bags, robots are still complex. ROS offers several visualization tools that are useful for debugging of a complex system.

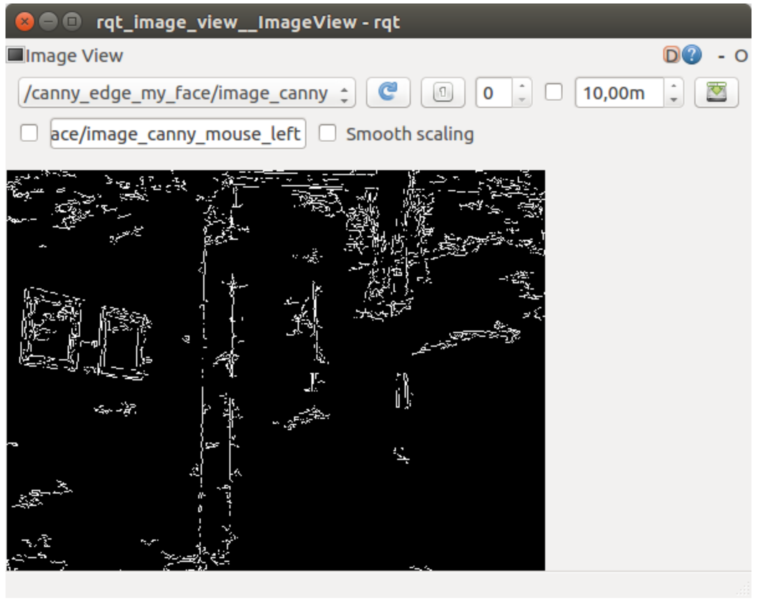
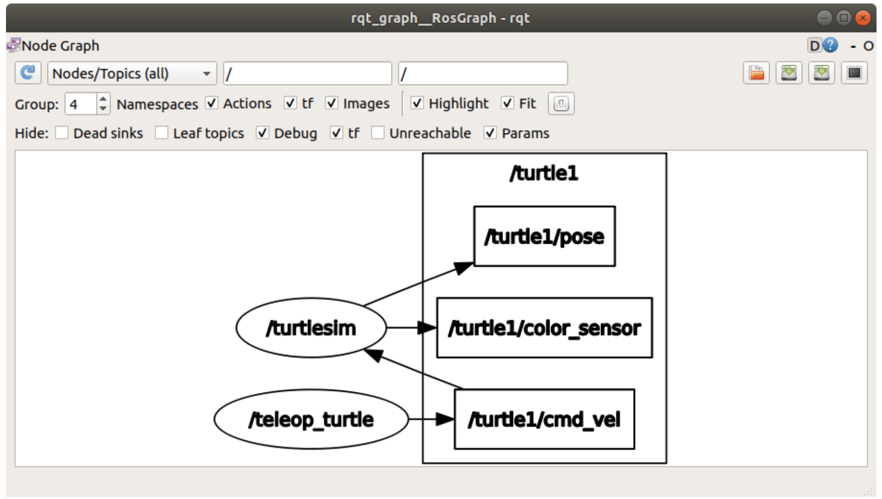

rqt_image_view__ImageView - rqt

Image View

/canny_edge_my_face/image_canny 0 10,00m

ace/image_canny_mouse_left Smooth scaling


rqt_graph__RosGraph - rqt

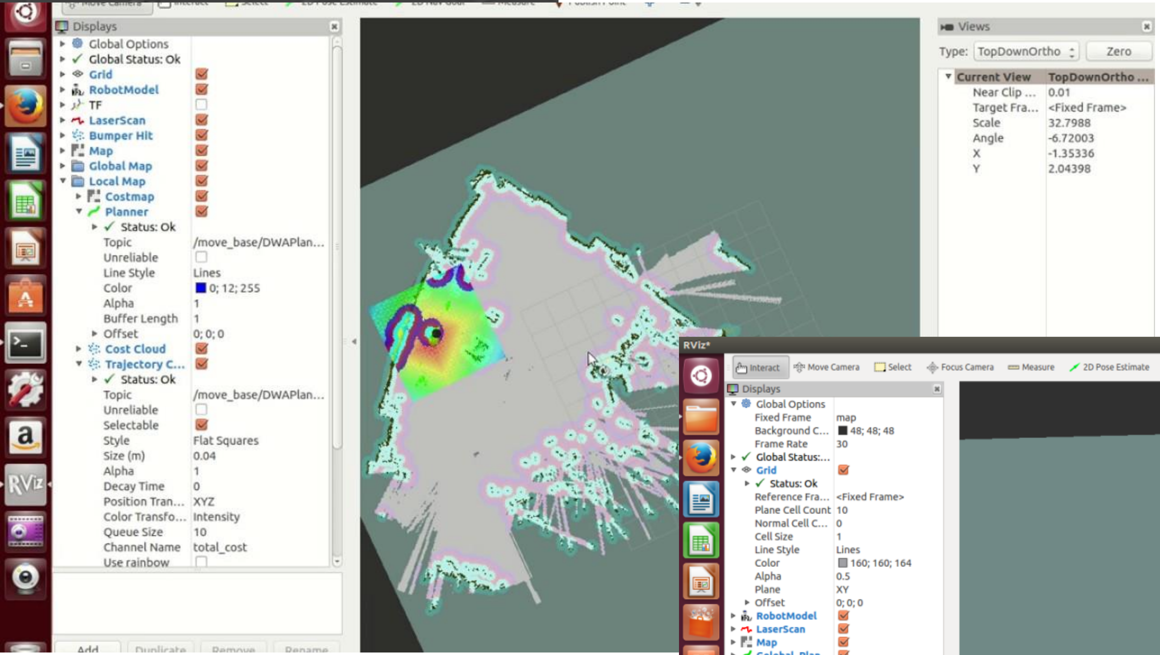
Node Graph

Nodes/Topics (all) / /

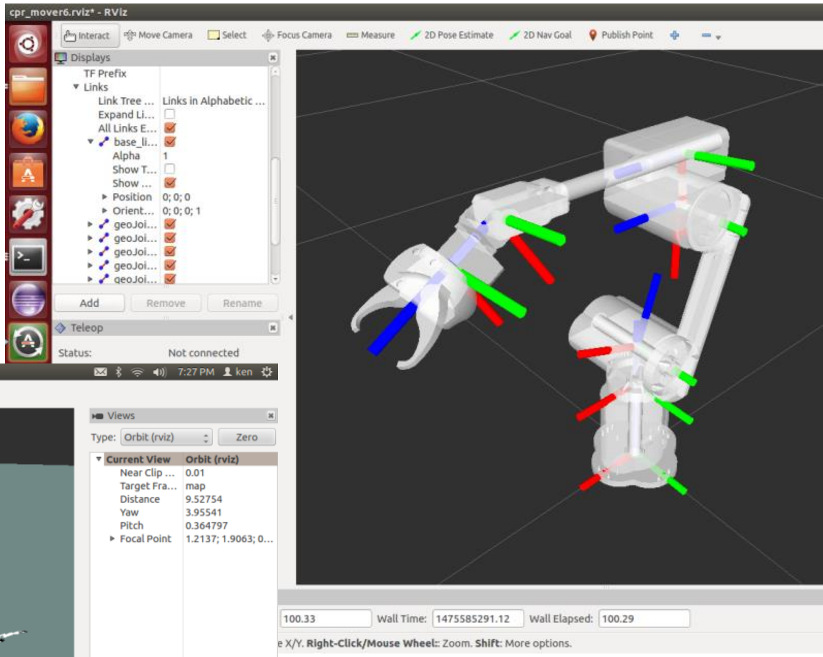
Group: 4 Namespaces ✓ Actions ✓ tf ✓ Images ✓ Highlight ✓ Fit

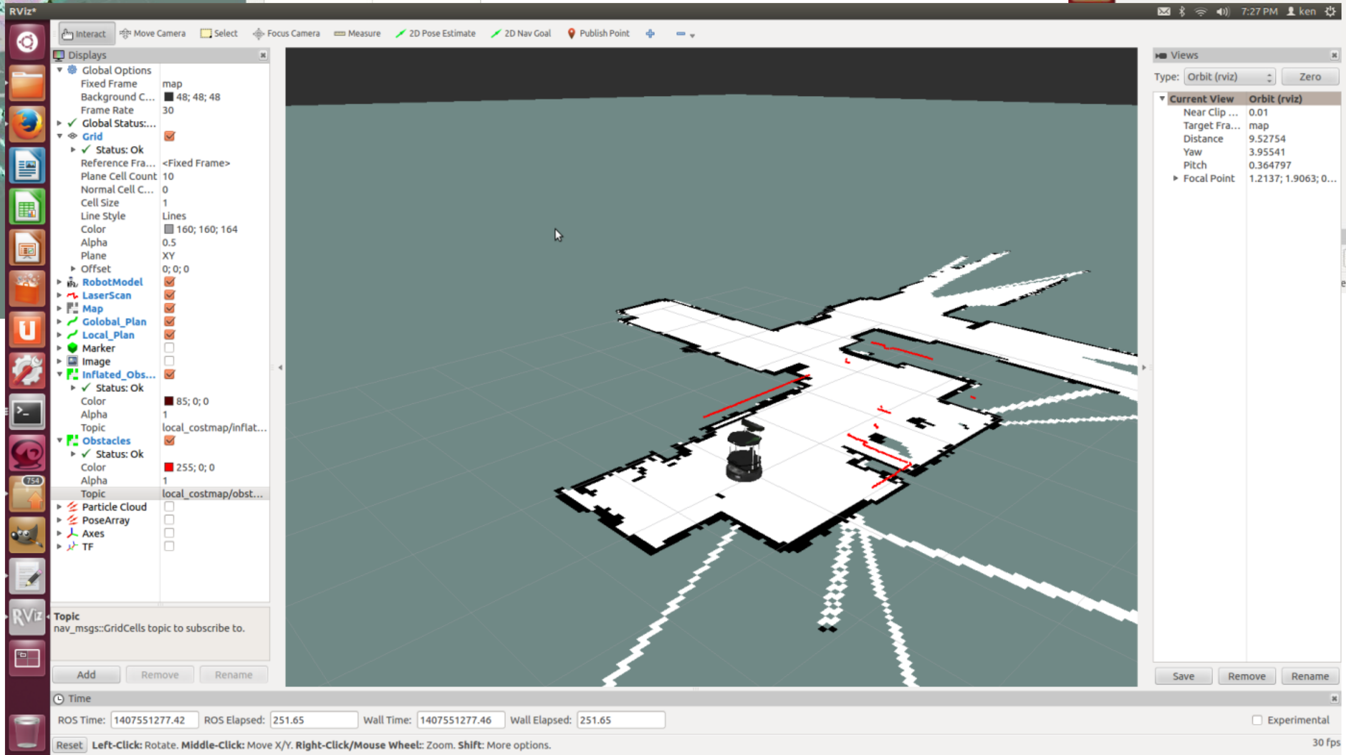
Hide: Dead sinks Leaf topics ✓ Debug ✓ tf Unreachable ✓ Params

RViz is the main visualization tool of ROS.
It is used to display sensor readings, maps, cost maps, joints, TF, and, in general, to have an overview of the internal status of the robot and of all its sensors and nodes.


Displays
Global Options
Global Status: Ok
Grid
RobotModel
TF
LaserScan
Bumper Hit
Map
Global Map
Local Map
Costmap
Planner
Status: Ok
Topic /move_base/DWAPlan...
Unreliable
Line Style Lines
Color 0; 12; 255
Alpha 1
Buffer Length 1
Offset 0; 0; 0
Cost Cloud
Trajectory C...
Status: Ok
Topic /move_base/DWAPlan...
Unreliable
Selectable
Style Flat Squares
Size (m) 0.04
Alpha 1
Decay Time 0
Position Tran... XYZ
Color Transfo... Intensity
Queue Size 10
Channel Name total_cost
Use rainbow

Views
Type: TopDownOrtho Zero
Current View TopDownOrtho ...
Near Clip ... 0.01
Target Fra... <Fixed Frame>
Scale 32.7988
Angle -6.72003
X -1.35336
Y 2.04398


cpr_mover6.rviz* - RViz
Displays
TF Prefix
Links
Link Tree ... Links in Alphabetic ...
Expand Li...
All Links E...
base_li...
Alpha 1
Show T...
Show ...
Position 0; 0; 0
Orient... 0; 0; 0; 1
geoJoi...
geoJoi...
geoJoi...
geoJoi...
geoJoi...
Teleop
Status: Not connected
100.33 Wall Time: 1475585291.12 Wall Elapsed: 100.29
e X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.


RViz*
Displays
Global Options
Fixed Frame map
Background C... 48; 48; 48
Frame Rate 30
Global Status:...
Grid
Status: Ok
Reference Fra... <Fixed Frame>
Plane Cell Count 10
Normal Cell C... 0
Cell Size 1
Line Style Lines
Color 160; 160; 164
Alpha 0.5
Plane XY
Offset 0; 0; 0
RobotModel
LaserScan
Map
Golobal_Plan
Local_Plan
Marker
Image
Inflated_Obs...
Status: Ok
Color 85; 0; 0
Alpha 1
Topic local_costmap/inflat...
Obstacles
Status: Ok
Color 255; 0; 0
Alpha 1
Topic local_costmap/obst...
Particle Cloud
PoseArray
Axes
TF

Topic
nav_msgs::GridCells topic to subscribe to.

Views
Type: Orbit (rviz) Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Target Fra... map
Distance 9.52754
Yaw 3.95541
Pitch 0.364797
Focal Point 1.2137; 1.9063; 0...

Time
ROS Time: 1407551277.42 ROS Elapsed: 251.65 Wall Time: 1407551277.46 Wall Elapsed: 251.65 Experimental
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options. 30 fps

ROS

ROS command line tools

- read and publish messages on topic
- have a list of all active services/nodes/topics/params
- find packages and folders
- compilation tools
- edit tools
- check tf
- check frequency of nodes
- …

ROS installation

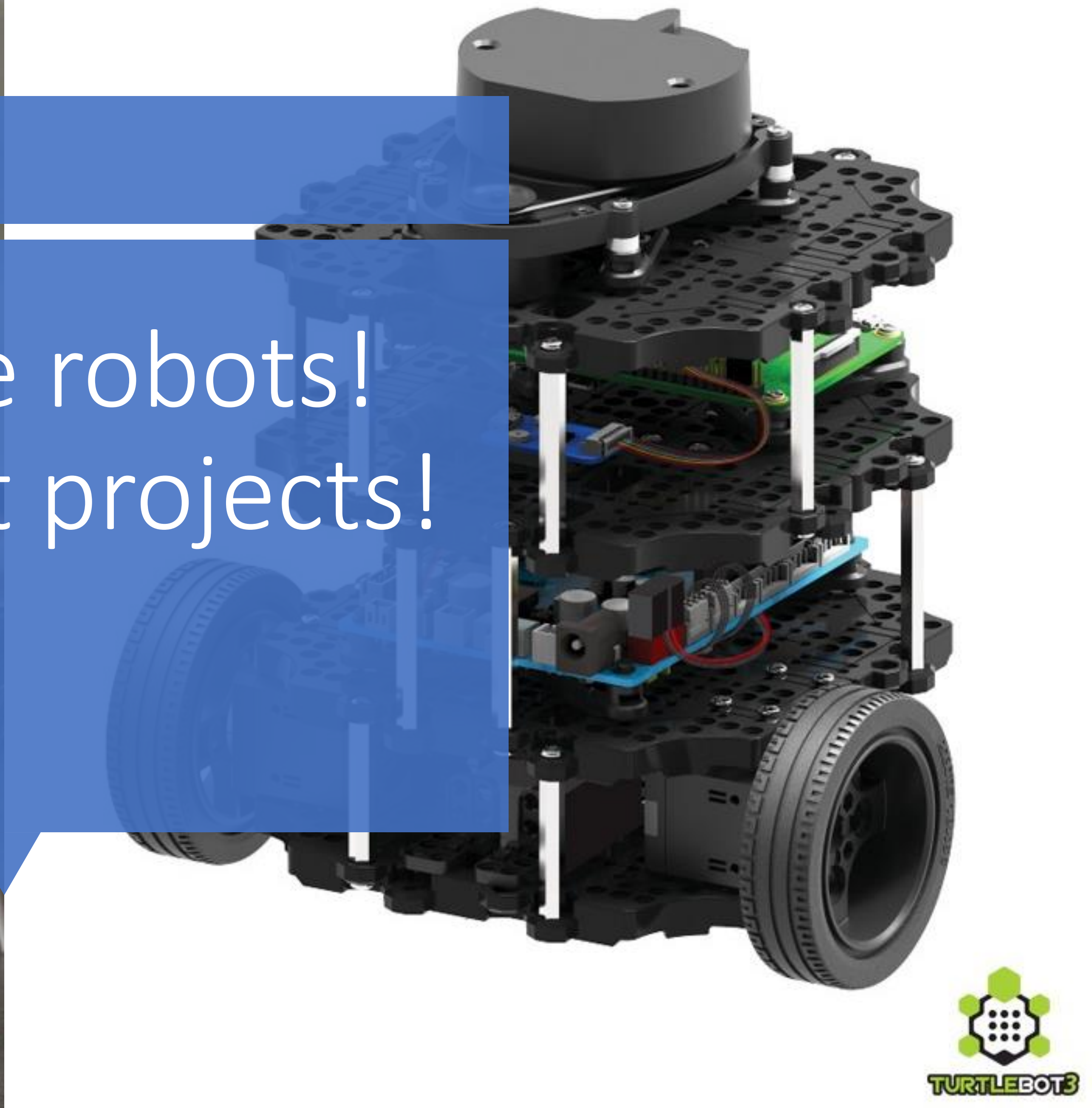
- ROS can be installed with multiple OS, but the simplest way for starting is using Ubuntu
- Each version of Ubuntu has its own ROS distro:
  - Ubuntu 16.04 → ROS Kinetic
  - Ubuntu 17.04 → ROS Lunar
  - Ubuntu 18.04 → ROS Melodic
- This year (2019) no new ROS version cause ROS 2.0 is on its way
- Stick with a LTS (ROS Kinetic or ROS Melodic)
- A VM is fine too for starting

ROS installation

- Follow guide at http://wiki.ros.org/ROS/Installation
- basically installation on Ubuntu is:
  sudo apt install ros-melodic-desktop-full
  and wait
- then follow the basic tutorials (2/3h tops)
  http://wiki.ros.org/ROS/StartGuide
  http://wiki.ros.org/ROS/Tutorials
- and you are ready to go, e.g. with simulated robots.

Sources - References

- Wiki.ros.org
- ROS Robot Programming
  A Handbook Written by Turtlebot3 Developers
  (available at http://www.robotis.com/service/download.php?no=719)
- Robotis Turtlebot3 documentation
  http://emanual.robotis.com/docs/en/platform/turtlebot3/getting_started/
- Jason O'Kane, A Gentle Introduction to ROS
  https://cse.sc.edu/~jokane/agitr/agitr-letter.pdf



We have robots!
Ask about projects!