

ROS2

for
ROS Developers

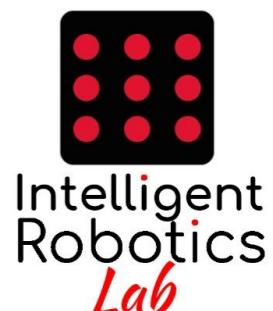
2

ROS



Prof. Dr. Francisco Martín Rico
Madrid 2-3-2022

francisco.rico@urjc.es
@fmrico



Introduction



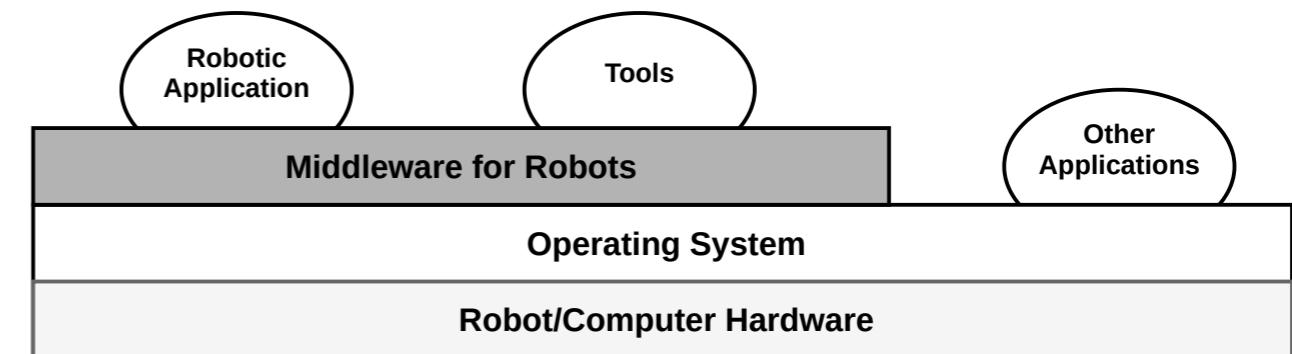
ROS (Robot Operating System)

“The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.”

<http://www.ros.org/>



Open Source Robotics Foundation



History

- ROS was born in 2006 at Stanford, within STAIR project
- Willow Garage in 2007
- Open Source Robotics Foundation in 2013
- Since 2013 is considered the standard in Robotics



ROS Distributions



Mar 2, 2010

ROS Box Turtle



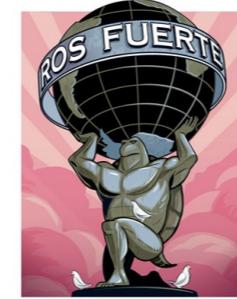
Aug 2, 2010



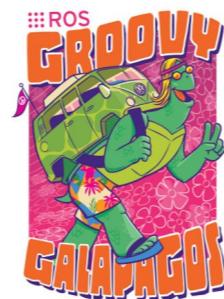
Mar 2, 2011



Aug 30, 2011



Apr 23, 2012



Dec 31, 2012



Sept 4th, 2013



Jul 22nd, 2014



May 23rd, 2015



May 23rd, 2016



May 23rd, 2017



May 23rd, 2018



May 23rd, 2020

Resources



- [ROS.org \[link\]](#)
- [Distributions \[link\]](#)
- [Package documentation \[link\] \[link\]](#)
- [ROS Wiki \[link\]](#)
- [ROS Answers \[link\]](#)
- [ROS Discourse \[link\]](#)
- [Blog \[link\]](#)



[What is ROS?](#)
The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

[Read More](#)

[ROS Melodic Morenia](#)
Melodic Morenia is the 12th official ROS release. It is supported on Ubuntu Artful and Bionic, along with Debian Stretch. Get Melodic Morenia now!

[Download](#)

[ROS Kinetic Kame](#)
Kinetic Kame is the 10th official ROS release. It is supported on Ubuntu Wily and Xenial. Get Kinetic Kame now!

[Download](#)

[Wiki](#) Find tutorials and learn more

[ROS Answers](#) Ask questions. Get answers

[Blog](#) Get the latest news

ROS Resources: Documentation | Support | Discussion Forum | Service Status | Q&A answers.ros.org

ROS Discourse

all categories > all tags > Categories Latest New (1) Unread (5) Top + New Topic

Category Topics Latest

General 896 1 unread

Welcome to discourse.ros.org 0 Feb '19

Webviz now supports live robots! 20 1h

Autoware.ai 1.13 Released! 3 5h

Proposed Edge AI WG 13 6h

OpenSplice move to Eclipse Cyclone DDS in ROS 2 2 8h

ROS@CES2020 Meetup 11 8h

Bosch ROSCon Talk Slides 0 11h

Navigation2 WG changes and HELP 25 1d

Autoware 171

Welcome to Autoware forum for project announcements, future development roadmaps, releases, and community discussion.

Next Generation ROS 469 3 unread

This is a forum to talk about the next generation of ROS.

Quality Assurance 62 1 new

A dedicated place for quality assurance discussion, where we can explain, promote, develop and discuss QA practices, techniques and tools. This is for everybody, from the quality enthusiast, who cares about quality of ROS, to a community member who has a question or feedback. Here, you can get help ...

ROS ANSWERS

ALL UNANSWERED Search or ask your question ASK YOUR QUESTION

Hi there! Please sign in help

TF Lookup would require extrapolation into the future

4 t robot_localization gps

I would appreciate some guidance to understand what may be going on here.

I am trying to use the very nice robot_localization package to integrate GPS, IMU, Optical Flow and a few other sensors to localize a drone.

When running my configuration with sensor data coming from a recorded bag file, I get the following error from the tf_localization_node node:

[WARN] [1406560584.464395417, 1406223582.586891248]: Could not obtain transform from utm to nav. Error was Lookup would require extrapolation into the future. Requested time 1406223582.123000000 but the latest data is at time 1406223582.576822666, when looking up transform from frame [utm] to frame [nav]

I've been trying to understand why this is happening and I've gathered the following information:

- The CPU usage is pretty low. I don't think this could be happening because of lack of processing capacity
- What the code seems to be trying to do is using lookupTransform to calculate TF using the timestamp of the GPS message received
- The transformation between [utm] and [nav] is being published at 50Hz. GPS messages are being published at 1Hz.
- I can see GPS messages being published with timestamps matching the warning messages. I can also see TF publishing transformations matching the timestamp of "the latest data is at time ..." but also many more later messages:

```
stamp:  
secs: 1406223582  
nsecs: 576822666  
frame_id: nav  
child_frame_id: utm  
transform:  
translation:
```

ROS.org

About | Support | Discussion Forum | Service Status | Q&A answers.ros.org

Search: Submit

Documentation Browse Software News Download

navigation

Indigo | Kinetix | Junar | melodic Show EOL distros:

Documentation Status

navigation: amcl | base_local_planner | carot_planner | clear_costmap_recovery | costmap_2d | dwa_local_planner | fake_localization | global_planner | map_server | move_base | move_base_msgs | move_slow_and_clear | nav_core | navfn | rotate_recovery | voxel_grid

Package Summary

✓ Released ✓ Continuous Integration: 81 / 81 - ✓ Documented

A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.

- Maintainer status: maintained
- Maintainer: Michael Ferguson <mfergus7 AT gmail DOT com>, David V. Lutt <davidvlut AT gmail DOT com>, Aaron Hoy <aaron.hoy AT fetchrobotics DOT com>
- Author: contradic8@gmail.com, Eitan Mander-Eppstein
- License: BSD, LGPLv2.1, (amcl)
- Source: git <https://github.com/ros-planning/navigation.git> (branch: melodic-devel)

Table of Contents

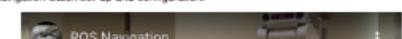
1. Overview
2. Hardware Requirements
3. Documentation
4. Report a Bug
5. Examples
6. Tutorials
 1. Basic ROS Navigation Tutorials
 2. Navigation Tutorials for the Care-O-bot
 3. Navigation Tutorials for the TurtleBot
 4. Navigation Tutorials for Husky
 5. Navigation Tutorials for the MRP2
 6. Navigation Tutorials for evanbot
 7. MPR2 Navigation Tutorials
 8. Related Applications
 9. Related Publications

Available Translations: SimpleChinese

R.O.B.O.T. Comics

1. Overview

The Navigation Stack is fairly simple on a conceptual level. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Use of the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a pre-requisite for navigation stack use, the robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level. To help with this process, this manual is meant to serve as a guide to typical Navigation Stack set-up and configuration.



Resources



ROS COMMANDS CHEAT SHEET v1.0



ROS.org

MASTER

Run a Master
\$ roscore

NODES

Get list of nodes
\$ rosnodes list

Get info of a node

Given a node name, it return its subscriptions, publications and services
\$ rosnode info [node]

Terminate a node

\$ rosnode kill [node]

Test if a node is alive

Get latency and if node is alive given its name
\$ rosnode ping [node]

TOPICS

Get list of topics
\$ rostopic list

Get info of topics

Get publishers and subscribers, as well as the message type
\$ rostopic info [topic]

Manual publish

Manually publication to a topic
\$ rostopic pub [-r freq] [topic] [msg_type] [msg]

Visualize topic publications

\$ rostopic echo [topic]

MESSAGES

Get list of available messages
\$ rosmsg list

Show message structure

\$ rosmsg show [msg_type]

SERVICES

Get list of available service types
\$ rossrv list

Show service type structure

\$ rossrv show [srv_type]

Show available running services

\$ rosservice list

Show service type of a running service

\$ rosservice info [srv]

Call a running service

\$ rosservice call [srv] [srv_type] [request]

FILESYSTEM

change working package dir

Change to active workspace dir

\$ roscd

Change to a dir starting from a package

\$ roscd [package]

List package content

\$ rosls [package]

Running a node

Requires a running Master

\$ rosrun [package] [node]

Launch an application

\$ roslaunch [package] [launcher]

CONFIGURE DISTRIBUTED SYSTEM

Host: machine that contains the master

Remote: machine that connects to Host

Both machines have to ping each other by name

In Remote:

\$ export ROS_IP=[REMOTE_IP]
\$ export ROS_HOSTNAME=[REMOTE_HOSTNAME]
\$ export ROS_MASTER_URI=http://[HOST_HOSTNAME]:11311

Check for problems (even in package):

\$ rosrun tf

DEBUGGING

Show nodes and topics connections
\$ rosrun rqt_graph rqt_graph

Show TF transforms tree

\$ rosrun rqt_tf_tree rqt_tf_tree

Show TF frames info

\$ rosrun tf tf_monitor

Show TF transformation

\$ rosrun tf tf_echo [frame_src] [frame_dst]

COMPILE

Compile a workspace

Compile all the packages in a workspace with num_threads threads

\$ catkin_make [-j num_threads]

Compile a specified pkg

\$ catkin_make [-j num_threads] [--pkg package]



Intelligent Robotics *Lab*

www.inrobots.es

www.intelligentroboticslab.robotica.gsync.es



Limitations

- Centralized in Master
- Own networking implementation
 - No real multicast
 - No QoS
- No multirobot
- No Real Time
- No multiplatform
- Important components are patches
- Only C++ and Python independent implementations
- Security



ROS2 Distributions



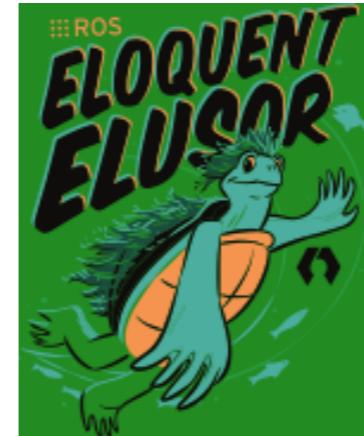
ROS2 Distributions



Dec 8th, 2017



Dec 14th, 2018



Nov 22nd, 2019



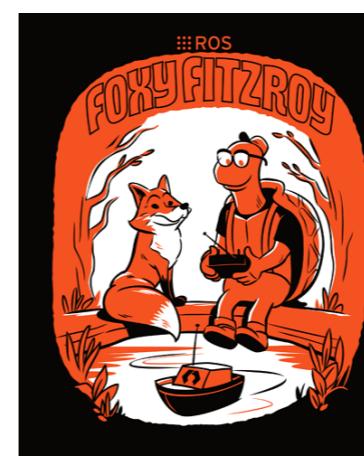
May 23rd, 2022



July 2nd, 2018



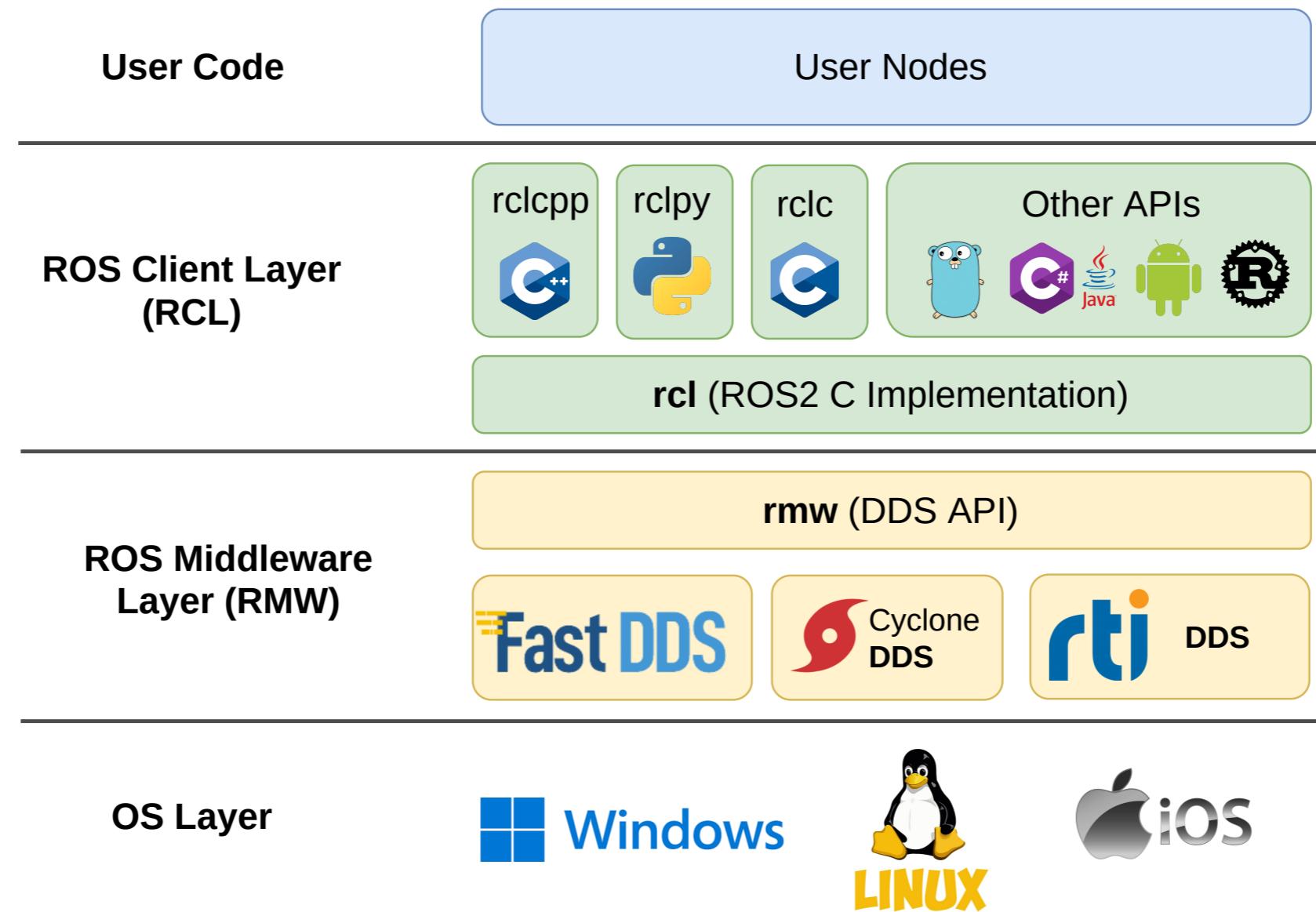
May 31st, 2019



Jun 5th, 2020

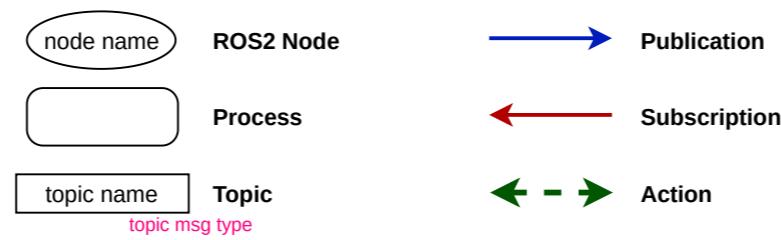
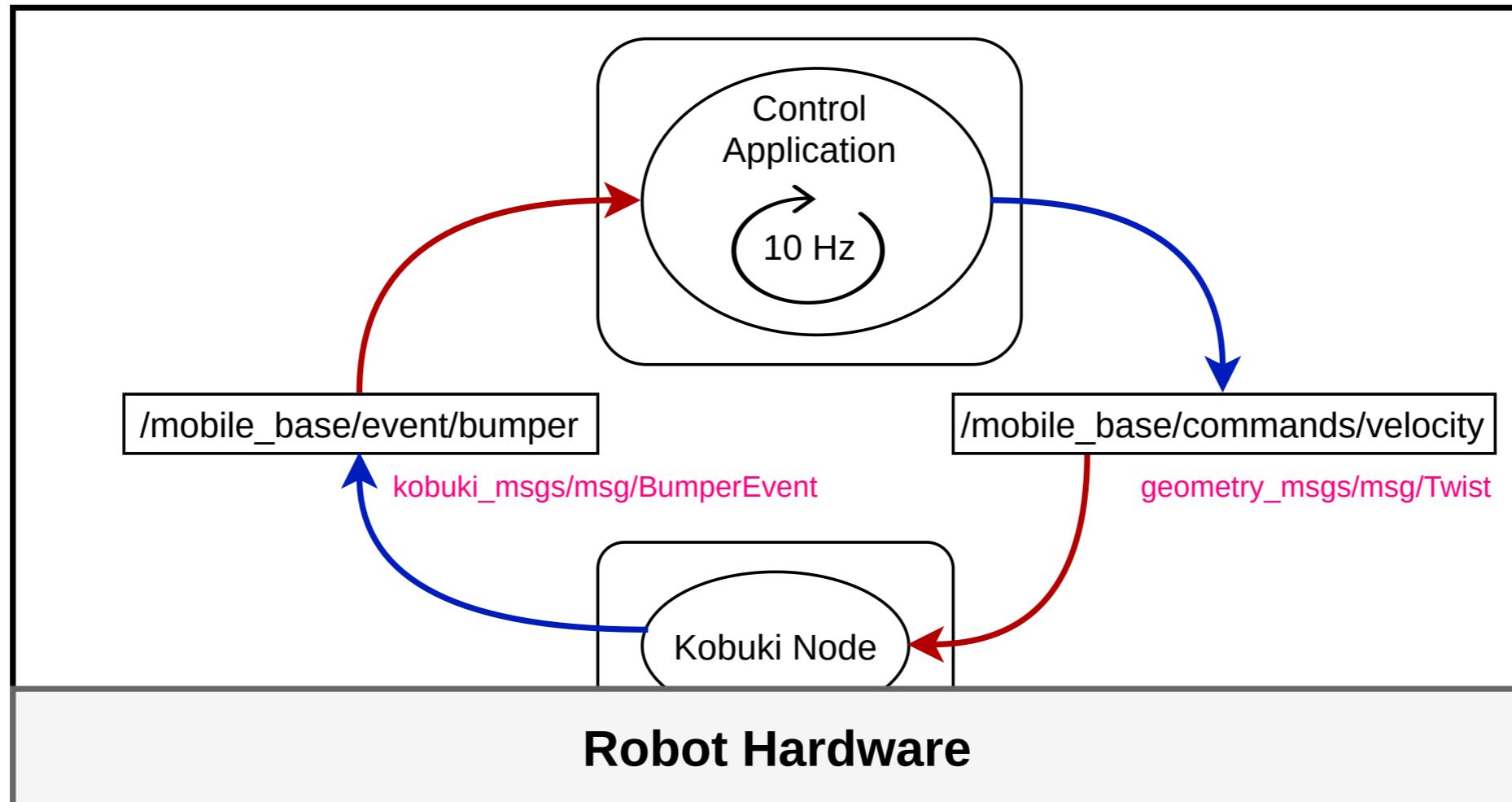


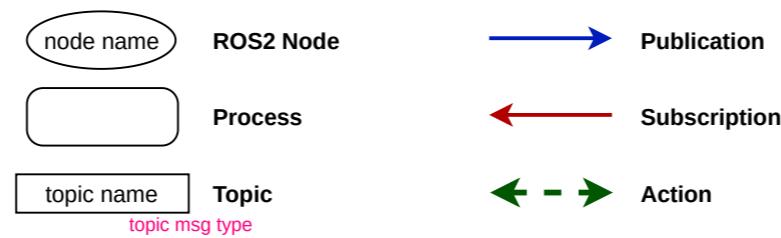
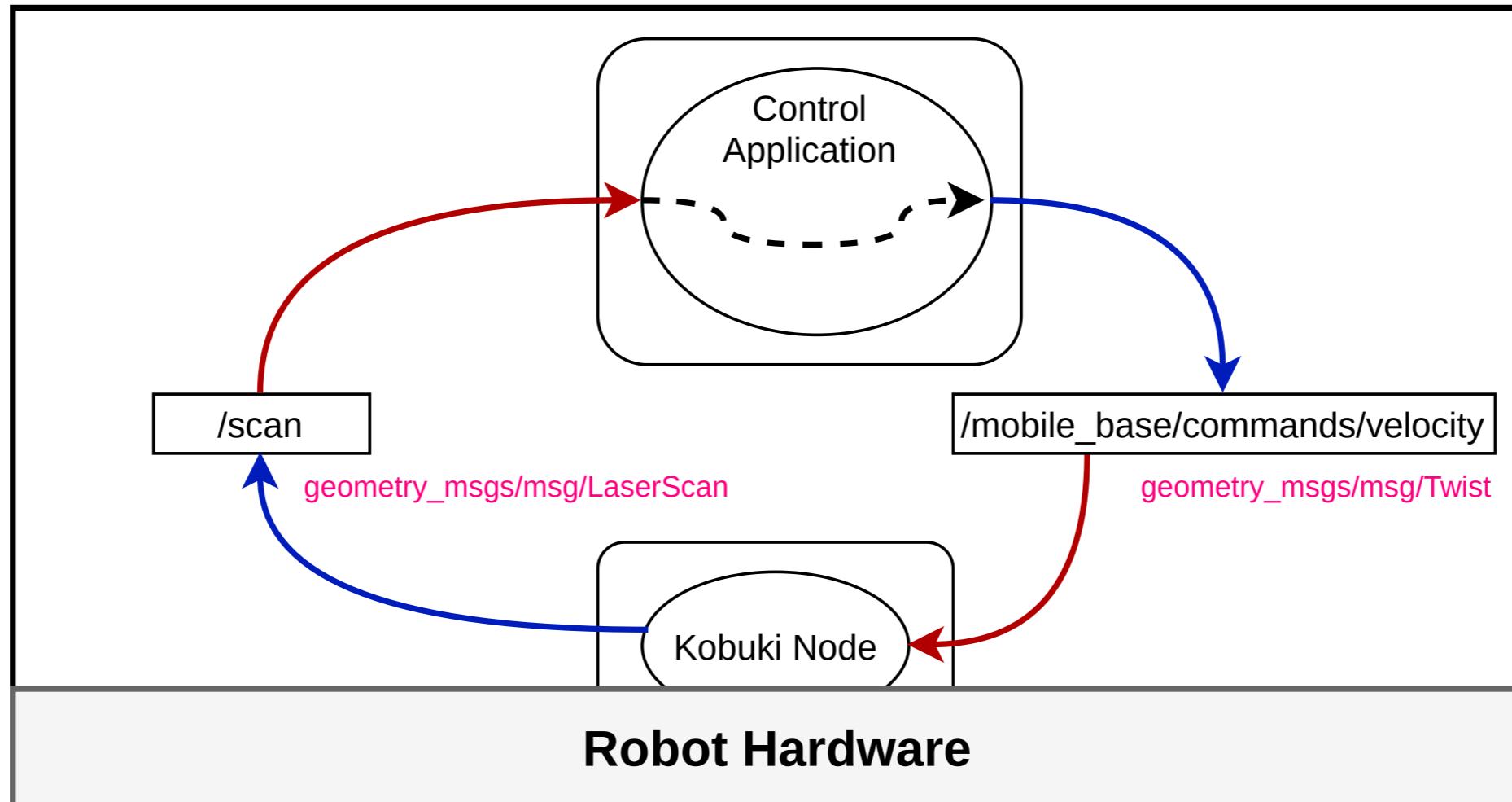
Design

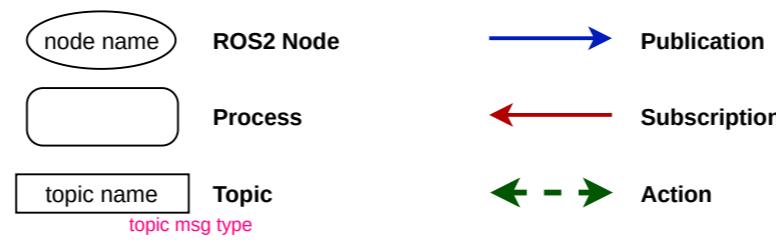
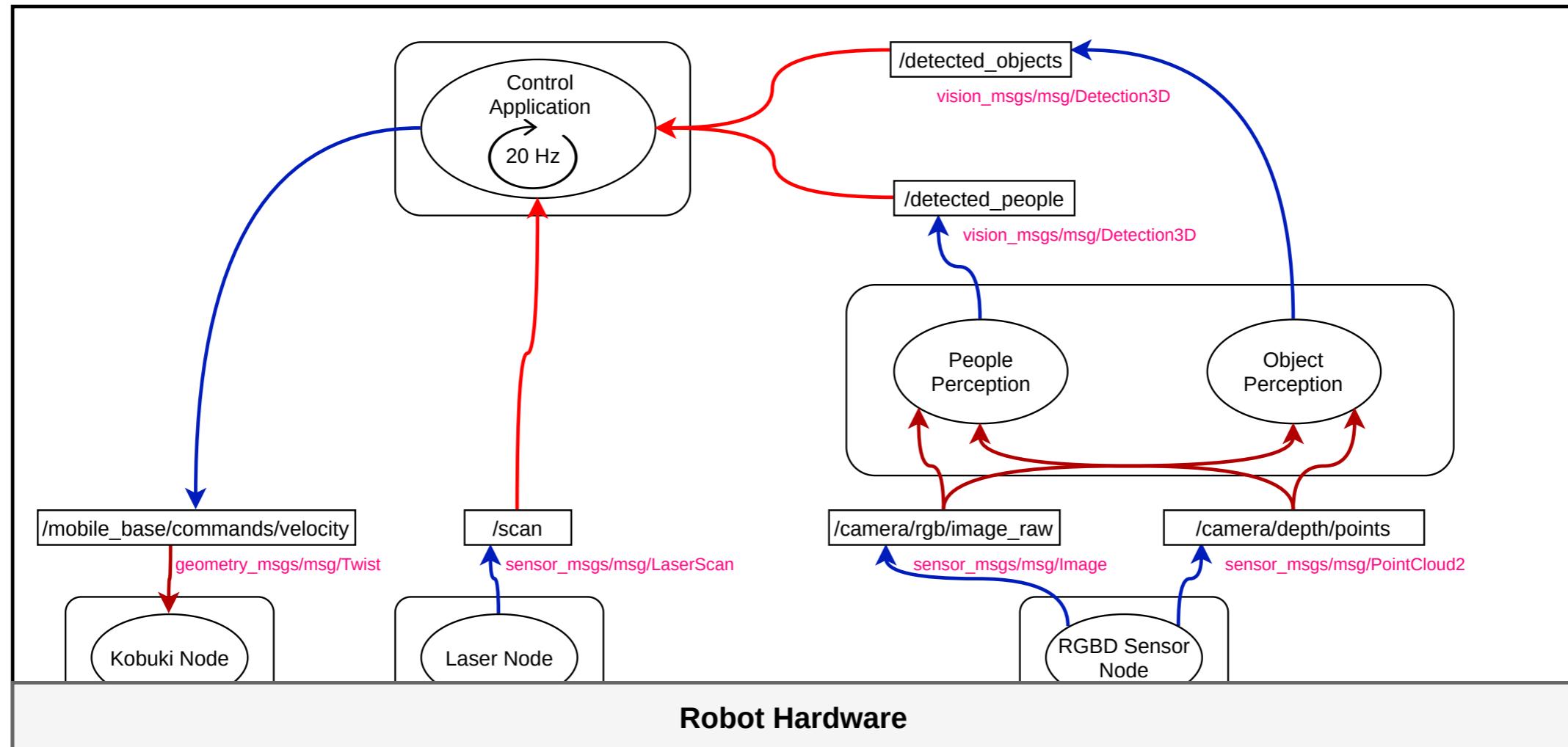


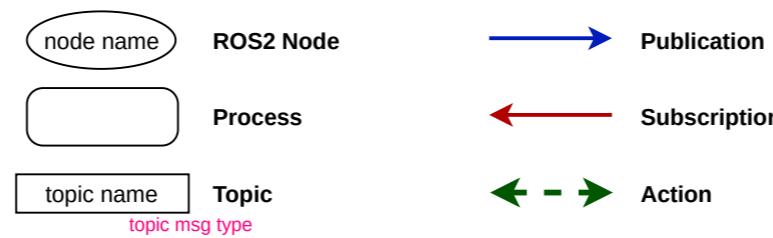
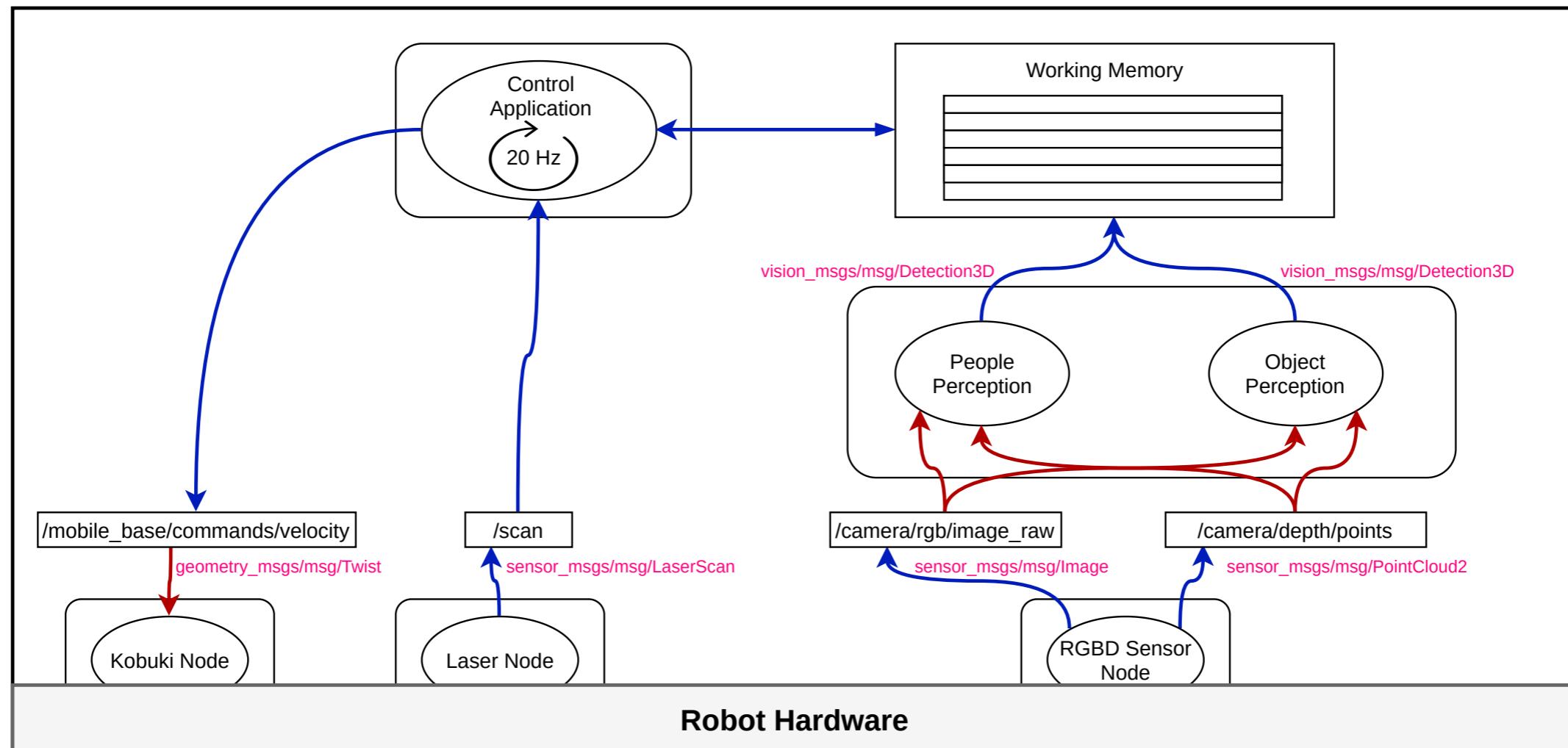
Applications in ROS2

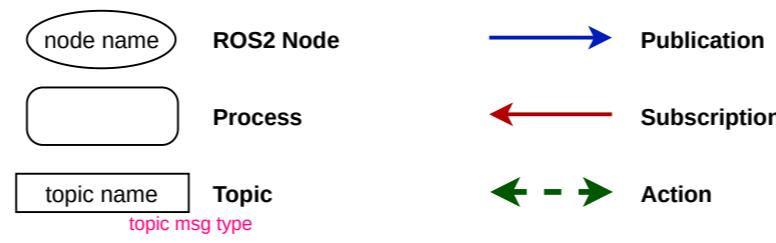
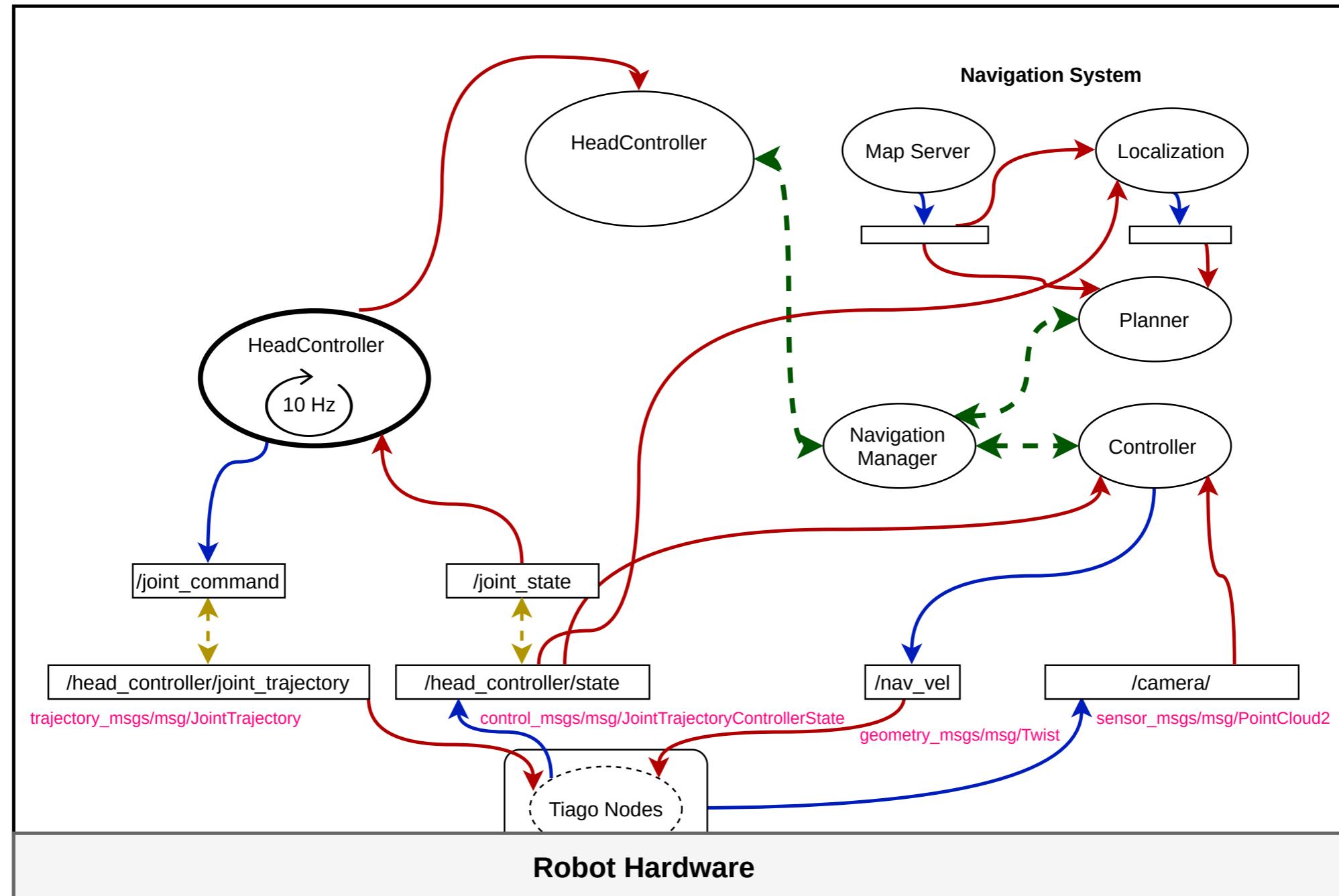






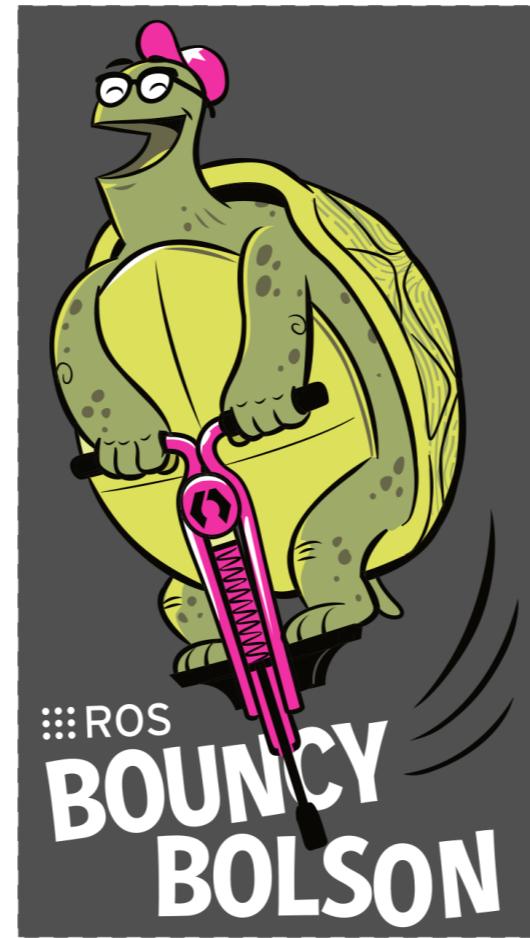






Hands on!!

ROS2



ROS2 Setup



- Install ROS2

<https://index.ros.org/doc/ros2/Installation/>

- Setup ROS2

```
source /opt/ros/eloquent/setup.bash  
source /home/fmrico/ros/ros2/ros2talk_ws/install/setup.bash
```

- DDS Selection



Fast-RTPS: Official ROS2 DDS vendor (by default).

OpenSplice:

```
export RMW_IMPLEMENTATION=rmw_opensplice_cpp
```



RTI Connext:

```
export RMW_IMPLEMENTATION=rmw_connex_cpp
```



Eclipse Cyclone:

```
export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
```

Firsts steps ROS2



- Your ROS2 workspace
- colcon as build tool
- build, install, log, devel

rclcpp API

[intro_ros2/src/simple_node.cpp](#)

- Extensive use of advanced C++ (C++14 mostly)
- rclcpp::
- Nodes are no longer processes; they are objects
- No need for NodeHandle
- ros2 as super command ([ros2cli](#))

```
$ colcon build --symlink-install  
$ ros2 run intro_ros2 nodo_simple  
$ ros2 node list  
$ ros2 node info /simple_node
```

[ROS2 Cheat Sheet](#)

intro_ros2/src/simple_node_pub.cpp

- Messages includes change: lowercase and _
- Differentiation between msg, srv and action

```
$ ros2 msg list  
$ ros2 msg std_msgs/msg/String
```

← Deprecated

```
$ ros2 interface list -m  
$ ros2 interface show std_msgs/msg/String
```

- publishers and subscriptions are requested to nodes
- `ROS_INFO` → `RCLCPP_INFO`
- Logger associated to a node id
- `spin_some` and `spin_once` (and more spinners)

```
$ ros2 run intro_ros2 simple_node_pub  
$ ros2 topic echo /chatter  
$ ros2 topic info /chatter  
$ ros2 topic hz /chatter  
$ ros2 topic bw /chatter
```

intro_ros2/src/simple_node_sub.cpp

[intro_ros2/src/simple_node_pub_qos.cpp](#)
[intro_ros2/src/simple_node_sub_qos.cpp](#)

```
auto subscription = node->create_subscription<std_msgs::msg::String>(
    "chatter", rclcpp::QoS(100).transient_local(), callback);
```

```
auto publisher = node->create_publisher<std_msgs::msg::String>(
    "chatter", rclcpp::QoS(100).transient_local());
```

[rclcpp QoS API](#)

\$ ros2 topic info /chatter –verbose

QoS policies

History	Keep last	Keep all
Depth	Size	
Reliability	Best effort	Reliable
Durability	Transient local	Volatile
Deadline	Duration	
Lifespan	Duration	
liveliness	Automatic	Manual by Topic
Lease Duration	Duration	

QoS profiles

Default	Reliable	Volatile	Keep last
Services	Reliable	Volatile	normal queue
Sensor	Best effort	small queue	
Parameters	Reliable	Volatile	Larger queue

QoS in ROS2



[intro_ros2/src/simple_node_pub_qos.cpp](#)
[intro_ros2/src/simple_node_sub_qos.cpp](#)

```
auto subscription = node->create_subscription<std_msgs::msg::String>(  
    "chatter", rclcpp::QoS(100).transient_local(), callback);
```

```
auto publisher = node->create_publisher<std_msgs::msg::String>(  
    "chatter", rclcpp::QoS(100).transient_local());
```

[rclcpp QoS API](#)

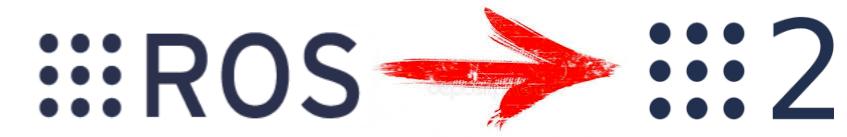
Compatibility of
QoS **durability**
profiles:

		Subscriber	
		Volatile	Transient Local
Publisher	Volatile	Volatile	No Connection
	Transient Local	Volatile	Transient Local

Compatibility of
QoS **reliability**
profiles:

		Subscriber	
		Best effort	Reliable
Publisher	Best effort	Best effort	No Connection
	Reliable	Best effort	Reliable

Better design for nodes



[intro_ros2/src/composable_node.cpp](#)

- Inherit from `rclcpp::Node`

[intro_ros2/src/composable_node_pub.cpp](#)

- Publishers/subscribers inside nodes

[intro_ros2/src/composable_node_sub.cpp](#)

- Multiples nodes per process

- **Executors**

- `rclcpp::executors::SingleThreadedExecutor` [API](#)
- `rclcpp::executors::MultiThreadedExecutor` [API](#)

[intro_ros2/src/composable_node_pub_sub.cpp](#)

- `spin_until_future_complete`

[intro_ros2/src/composable_node_pub_timer.cpp](#)

- Timers for controlling rhythm
 - No extra threads - Graceful degradation

Launchers



- In Python
- More control

[intro_ros2/launch/composable_pub_timer_launch.py](#)

- `generate_launch_description` function that returns a `LaunchDescription` object

```
$ ros2 launch intro_ros2 composable_pub_timer_launch.py
```

- It contains actions: execute node, set env var, include other launcher and launcher args

[launchers_ros2/launch/first_launch.py](#)

[launchers_ros2/launch/second_launch.py](#)

- Set parameters from files

[parameters_ros2/launch/param_file.launch.py](#)

[parameters_ros2/config/config_2.yaml](#)

Parameters

- Parameters are stored in nodes
- They must be declared

parameters_ros2/src/simple_param_node.cpp

```
$ ros2 run parameters_ros2 simple_param_node  
$ ros2 param get /simple_param_node message
```

- You can subscribe to changes

parameters_ros2/src/reconf_param_node.cpp

```
$ ros2 run parameters_ros2 reconf_param_node  
$ ros2 param set /ReconfigurableNode speed 5.6
```

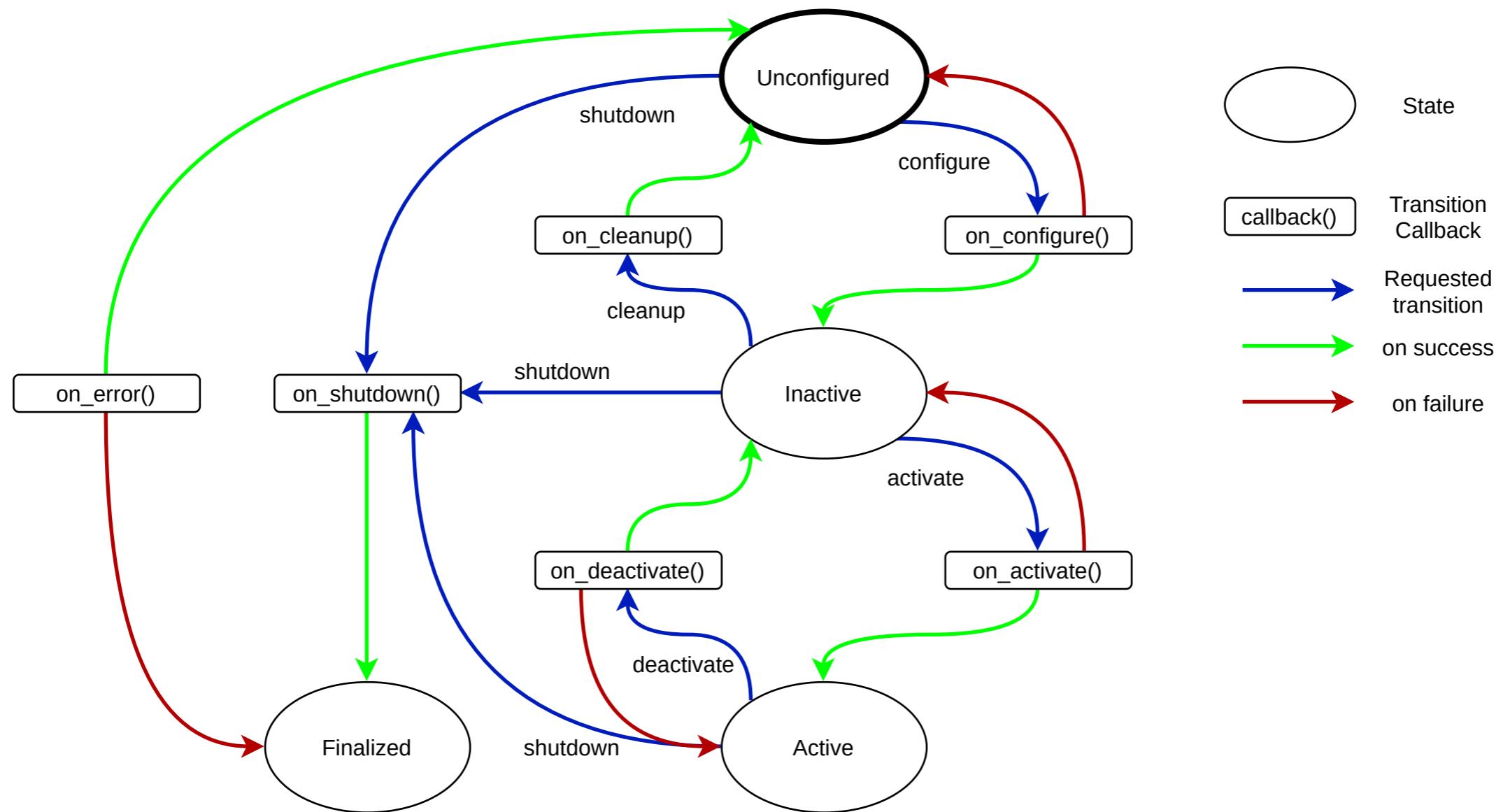
Lifecycle nodes



- In ROS, nodes are executing from initial
- Some nodes open devices
- Some nodes read params and configurations
- Specially critical for drivers
- In a launcher, there is not a way to control the boot order
- No way to pause, reconfigure, cleanup and shutdown nodes
- **ROS2: Lifecycle Nodes**
- LF nodes are in a state
- In each state, there are some available transitions
- Observed and commanded from outside (from launchers!!!)

States and Transitions

ROS → 2



lifecycle_nodes_ros2/src/simple_lifecycle_node.cpp

- Inheritance from `rclcpp_lifecycle::LifecycleNode`
- Control of code in transitions
 - `on_configure`
 - `on_activate`
 - ...
- You can accept or reject a transition request

```
$ ros2 launch lifecycle_nodes_ros2 simple.lifecycle_node
$ ros2 lifecycle nodes
$ ros2 lifecycle get /lifecycle_node_example
$ ros2 lifecycle list /lifecycle_node_example
$ ros2 lifecycle set /lifecycle_node_example configure
$ ros2 lifecycle set /lifecycle_node_example activate
$ ros2 lifecycle set /lifecycle_node_example deactivate
```

- Lifecycle publishers

lifecycle_nodes_ros2/src/pub_lifecycle_node.cpp

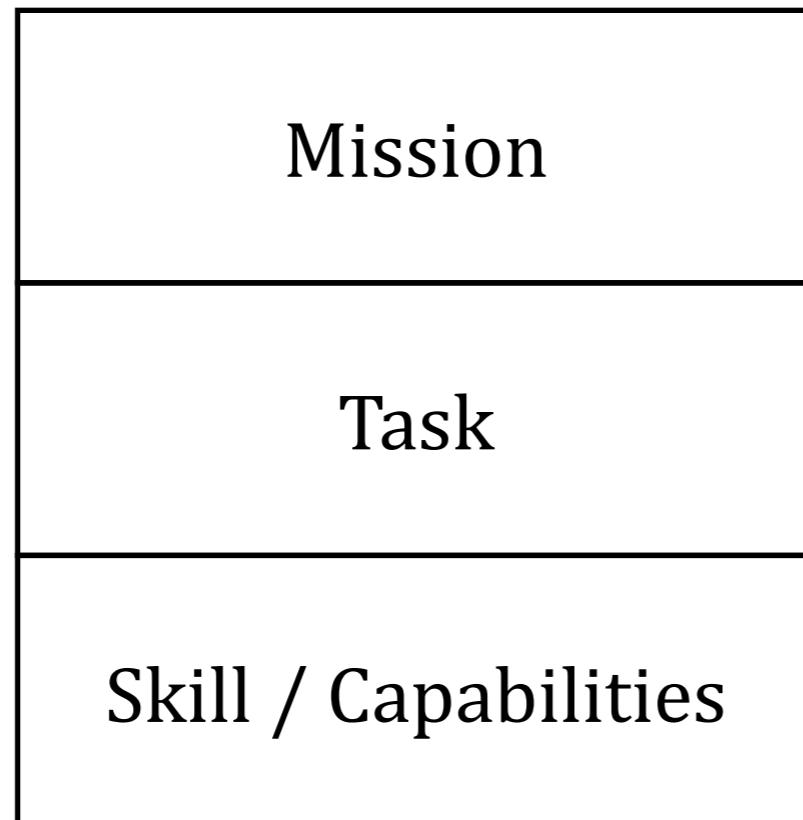
- Beyond Lifecycle... System Modes

Examples of Software in ROS2

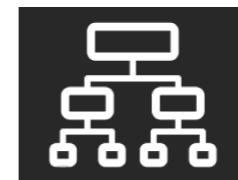


Software Architectures

ROS → 2



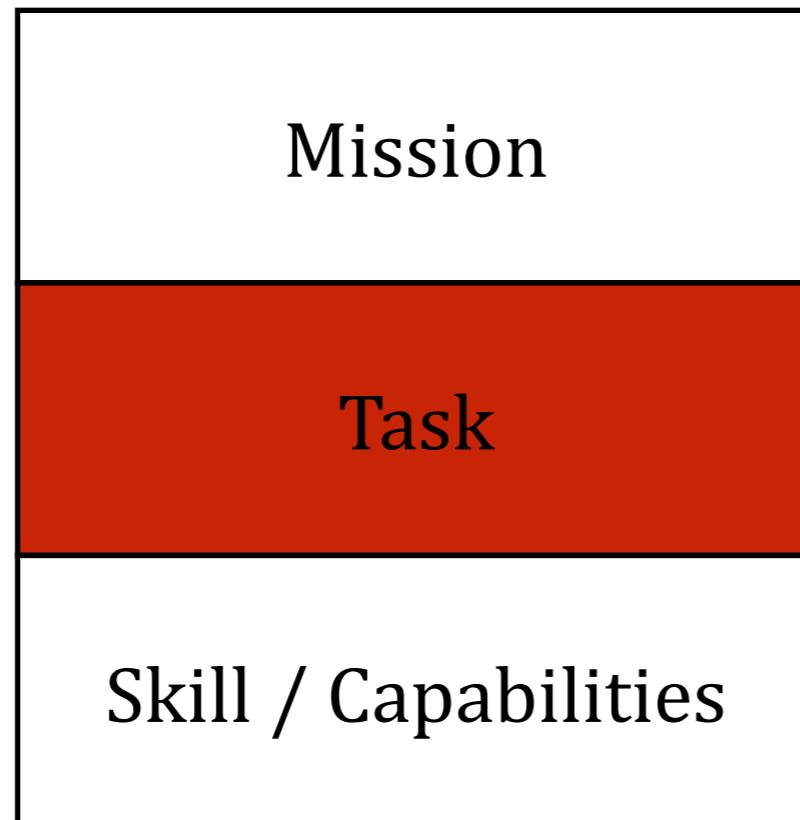
PlanSys 2



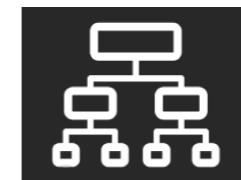
N A V 2

Software Architectures

ROS → 2



PlanSys 2



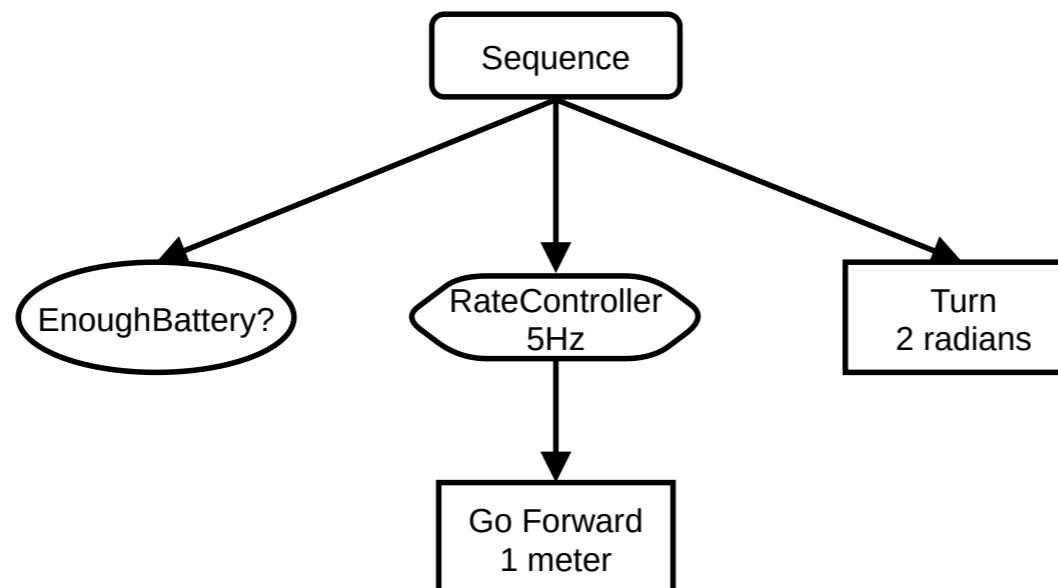
N A V 2

Behavior Trees

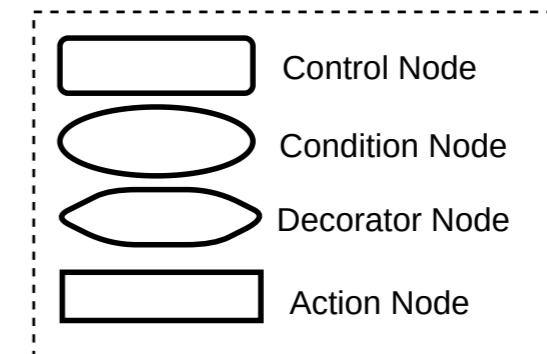
- Very popular in recent years
- Compared with FSM
- A Behavior Tree (BT) is a mathematical model to encode the control of a system
- It is a hierarchical data structure defined recursively from a root node with several child nodes.

- The basic operation is the **tick**:

- SUCCESS
- FAILURE
- RUNNING

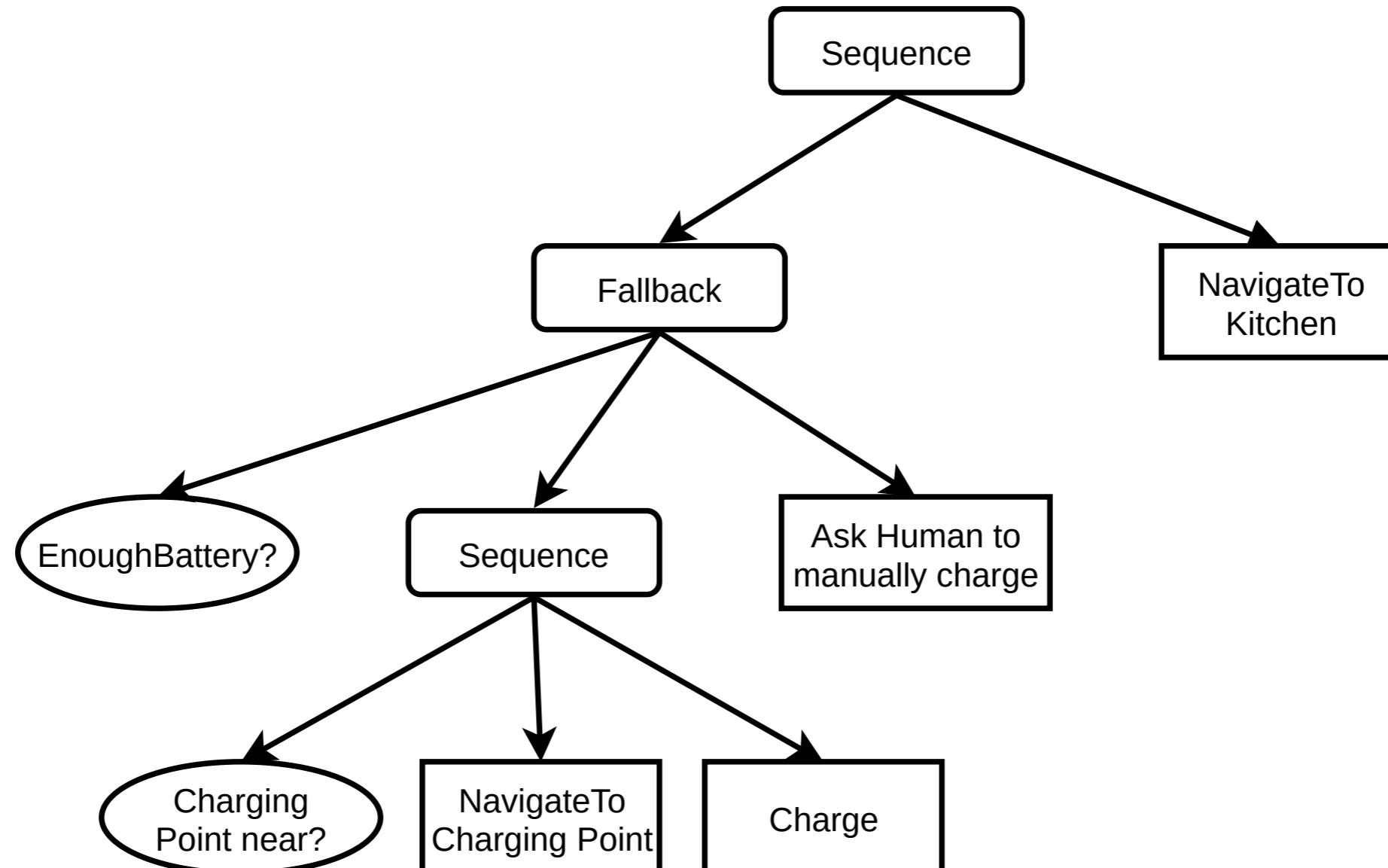


- BT node types:



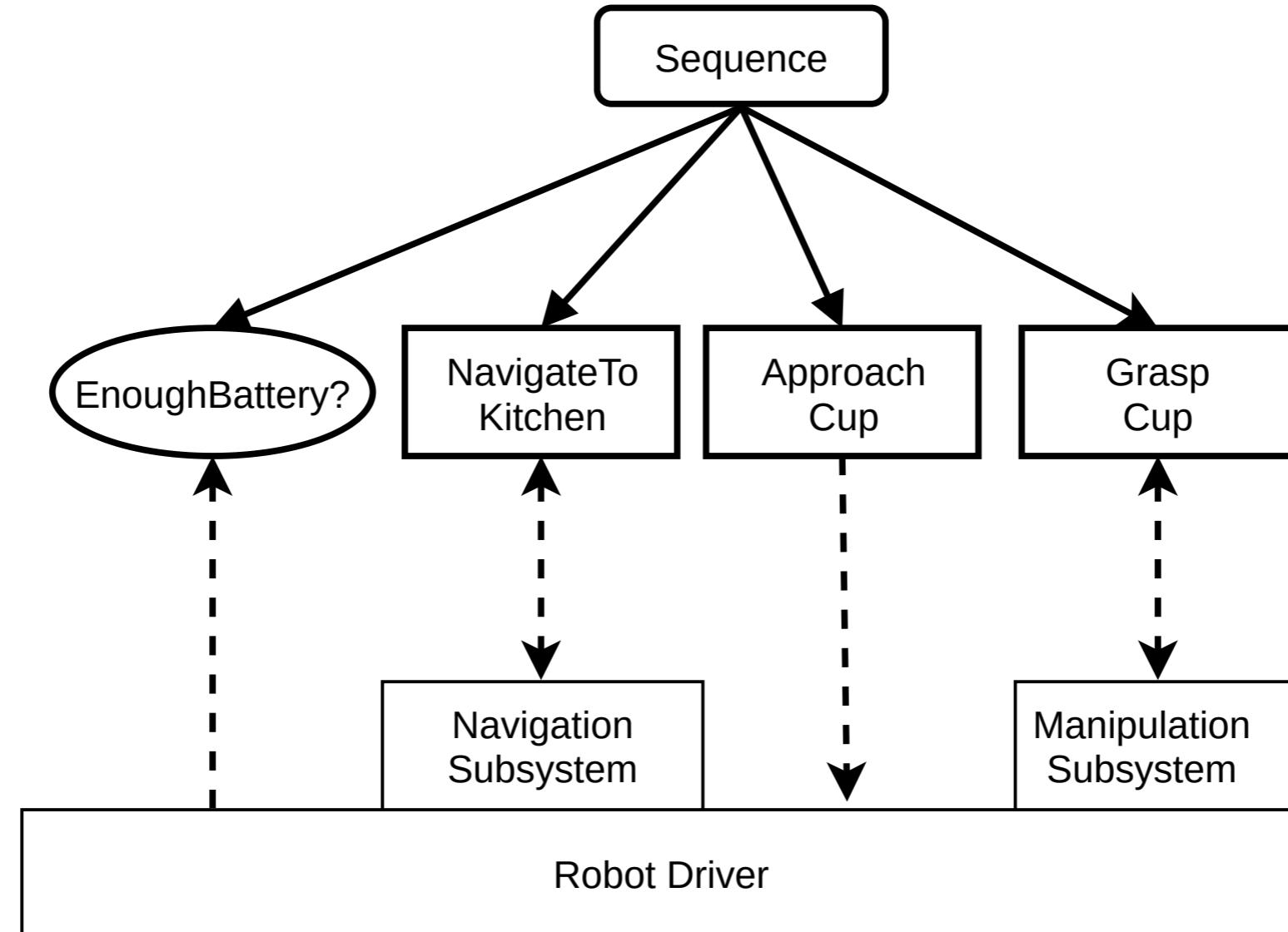
Behavior Trees

ROS → 2



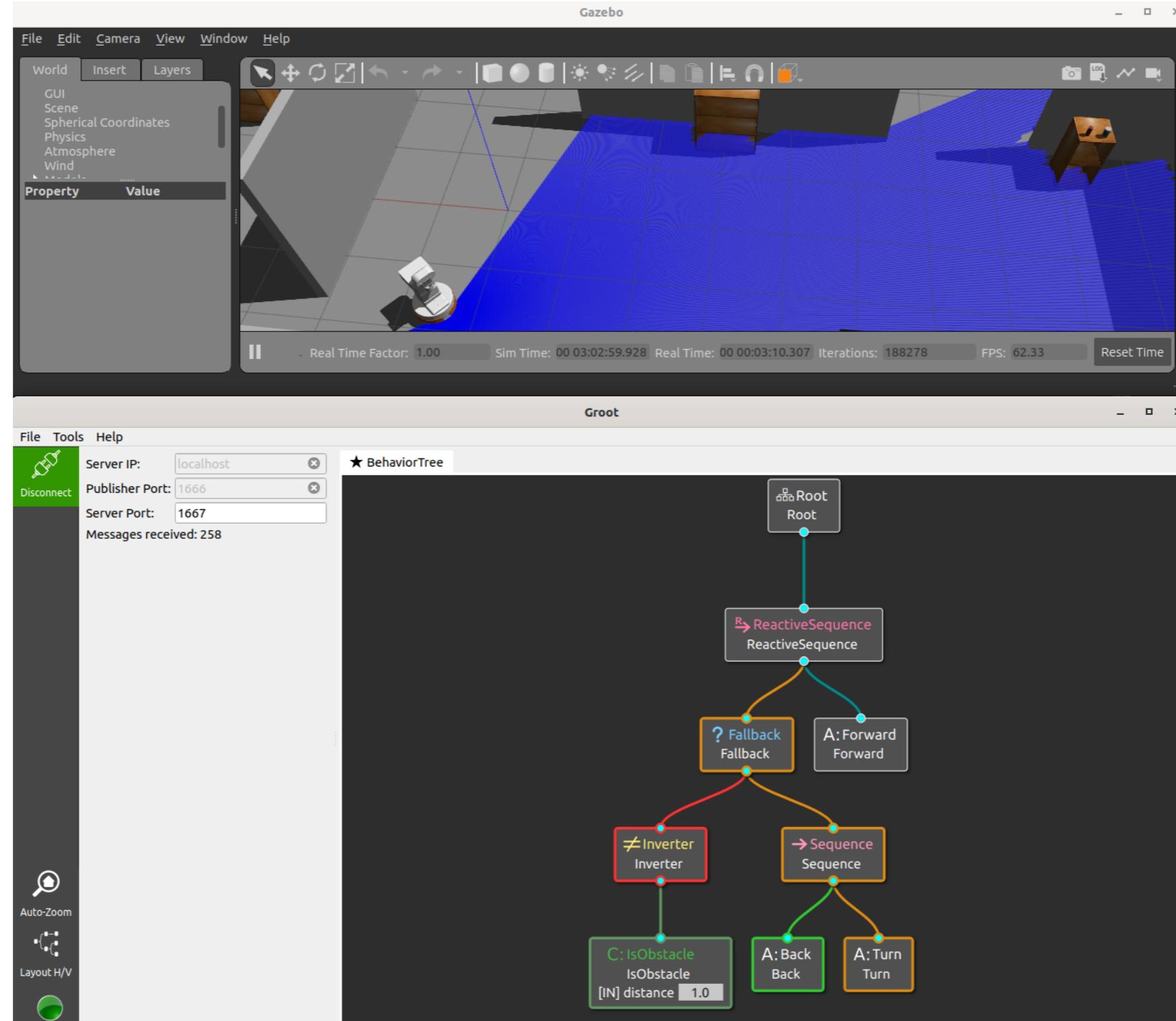
Behavior Trees

ROS → 2



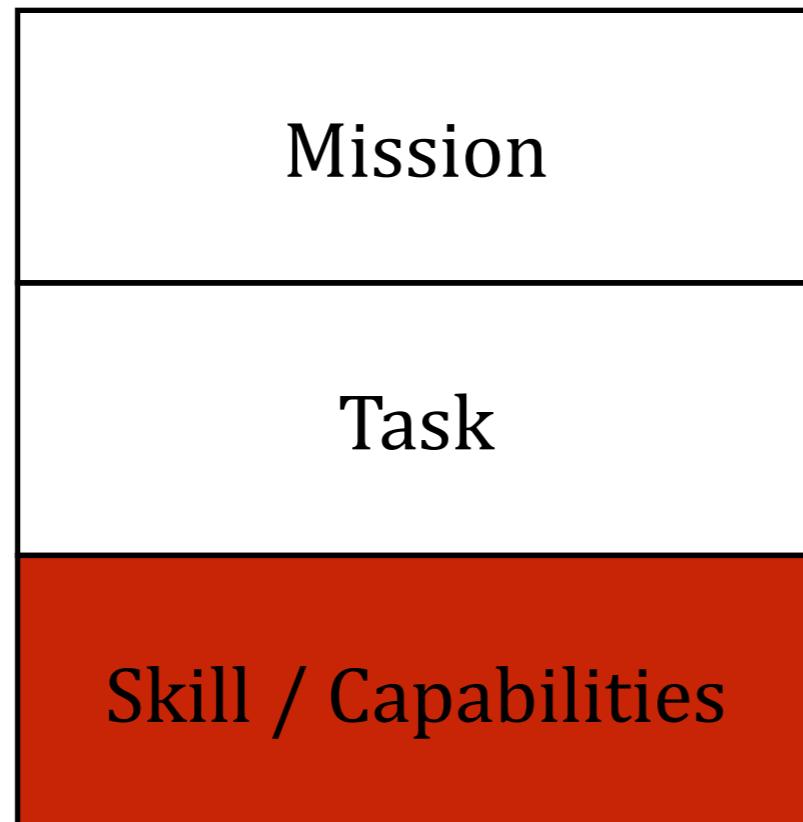
Behavior Trees

ROS → 2

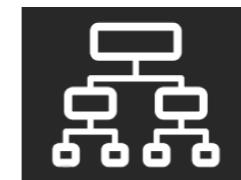


Software Architectures

ROS → 2



PlanSys 2



N A V 2

Overview



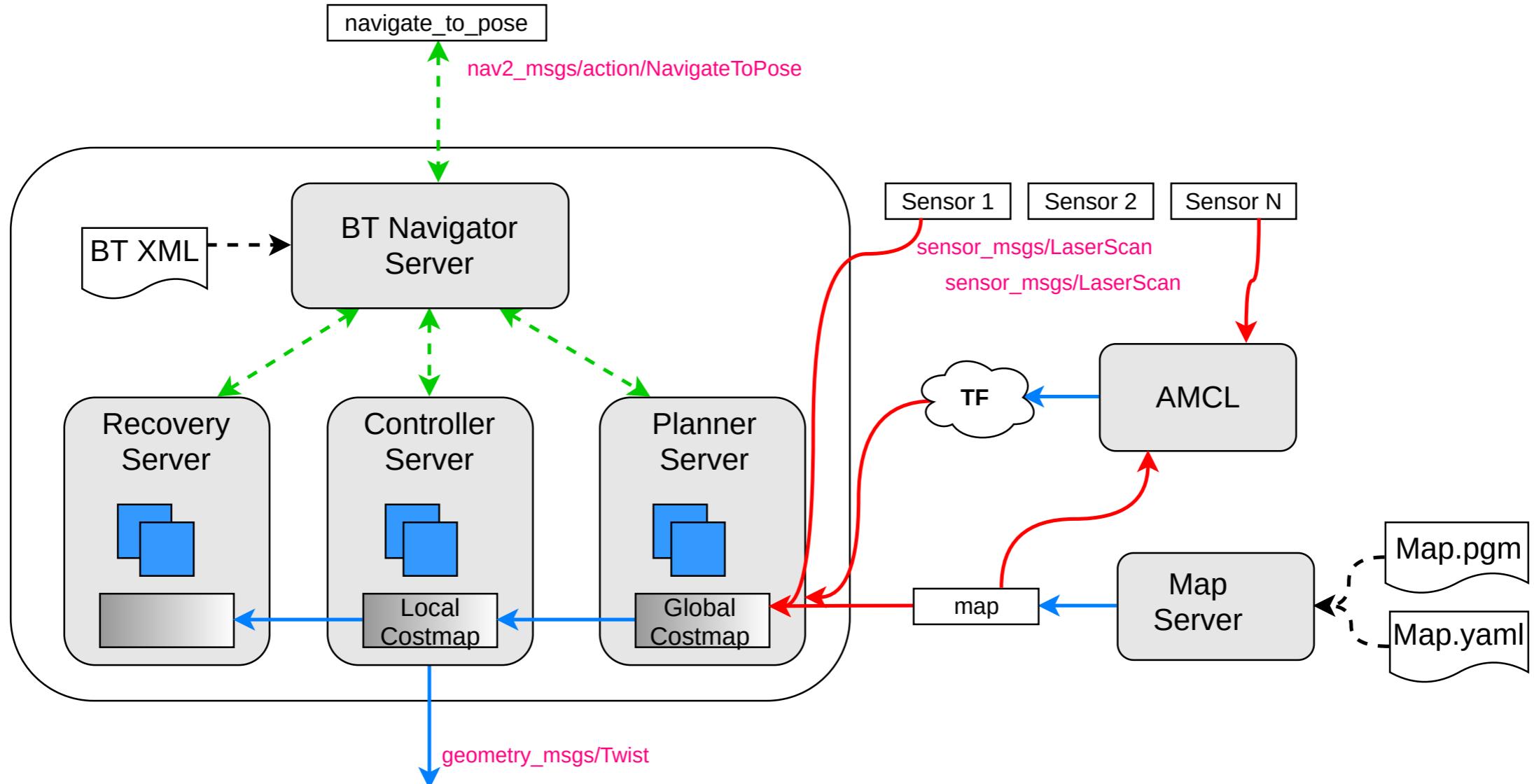
- Main author: **Steven Macenski**
- Reimplementation of Navigation stack
- Example of good practices in ROS2
- More algorithms, plugins, components...
- Contributions: 59 contributors, Intel, Samsung

The screenshot shows the ROS Navigation 2 documentation page. The left sidebar contains links for Getting Started, Build and Install, Navigation Concepts, Tutorials, Configuration Guide, Navigation Plugins, Migration Guides, Getting Involved, About and Contact, and Projects for 2020 Summer Student Program. The main content area features a title "ROS 2 Navigation" and two side-by-side images showing a mobile robot navigating through a hallway. Below the images is a section titled "Overview" which describes the project's purpose and structure.

The screenshot shows the GitHub repository page for ros-planning/navigation2. The top navigation bar includes links for Pull requests, Issues, Marketplace, and Explore. The repository name "ros-planning / navigation2" is displayed, along with statistics: 51 issues, 327 forks, and 215 contributors. The main content area shows a list of recent commits from the master branch, with details like author, commit message, timestamp, and commit count. On the right side, there are sections for "About", "Releases" (with a latest release at version 0.4.1), and "Contributors" (listing 59 contributors). A footer for "Intelligent Robotics Lab" is visible.

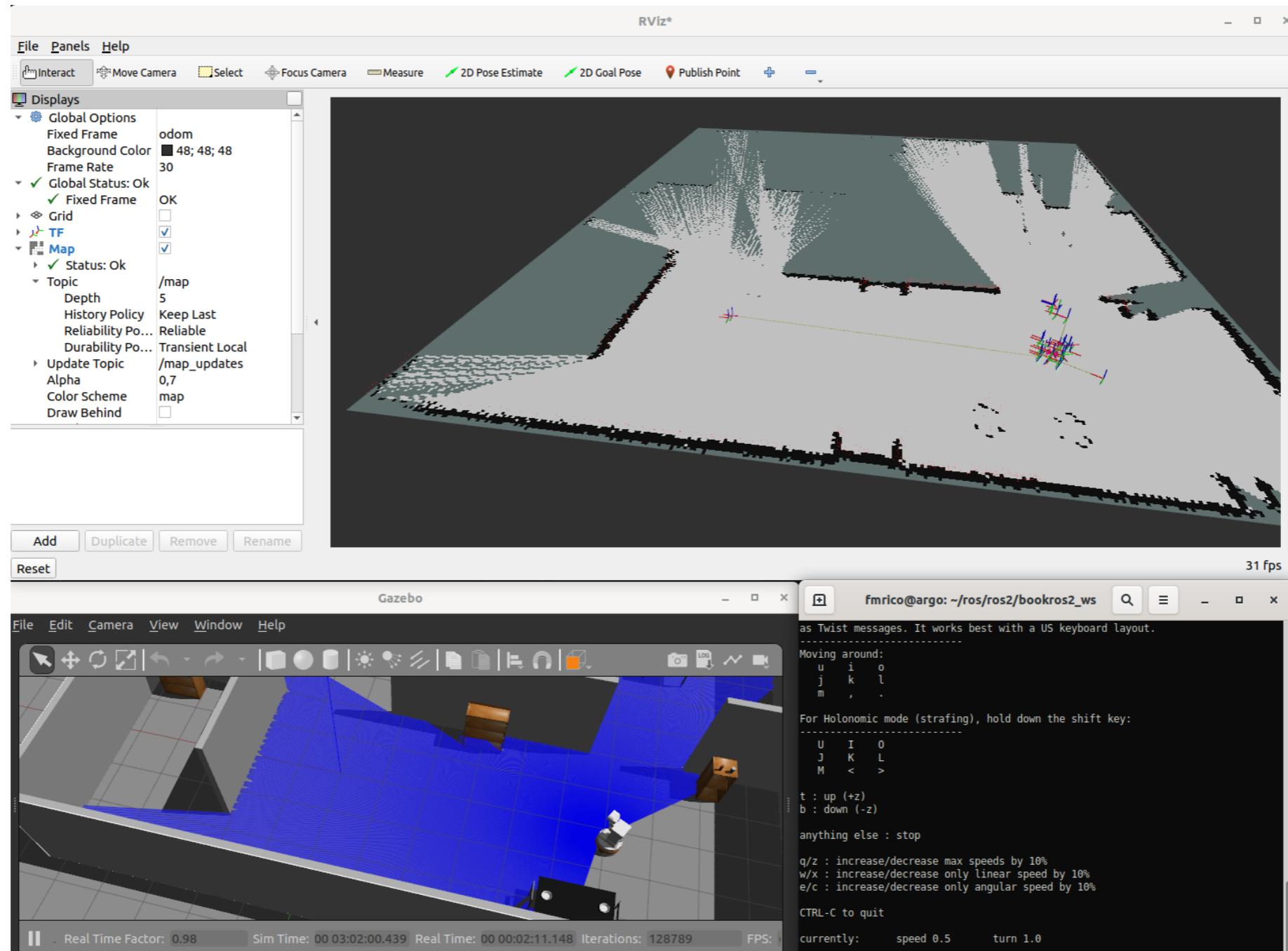
Architecture

ROS → 2



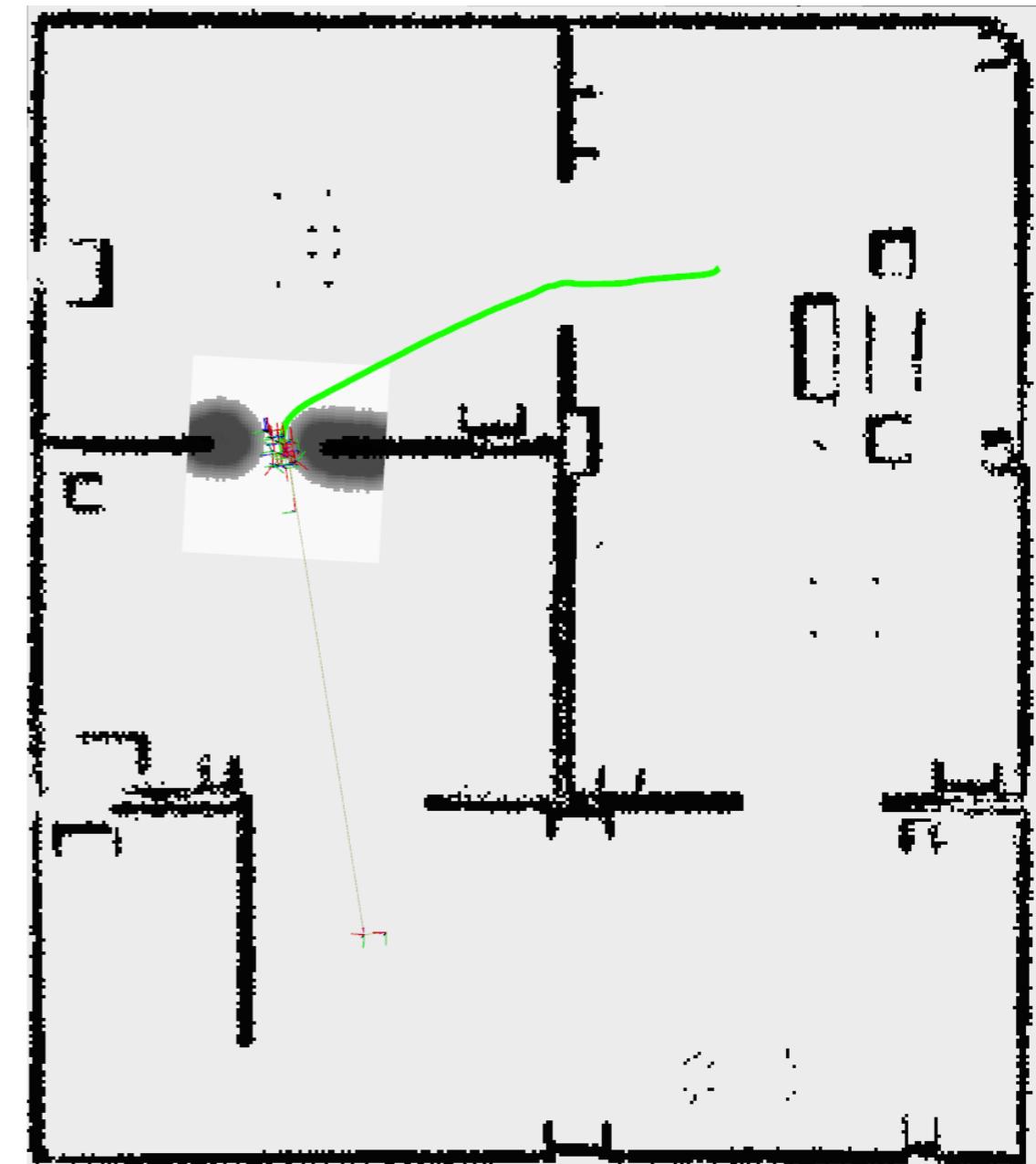
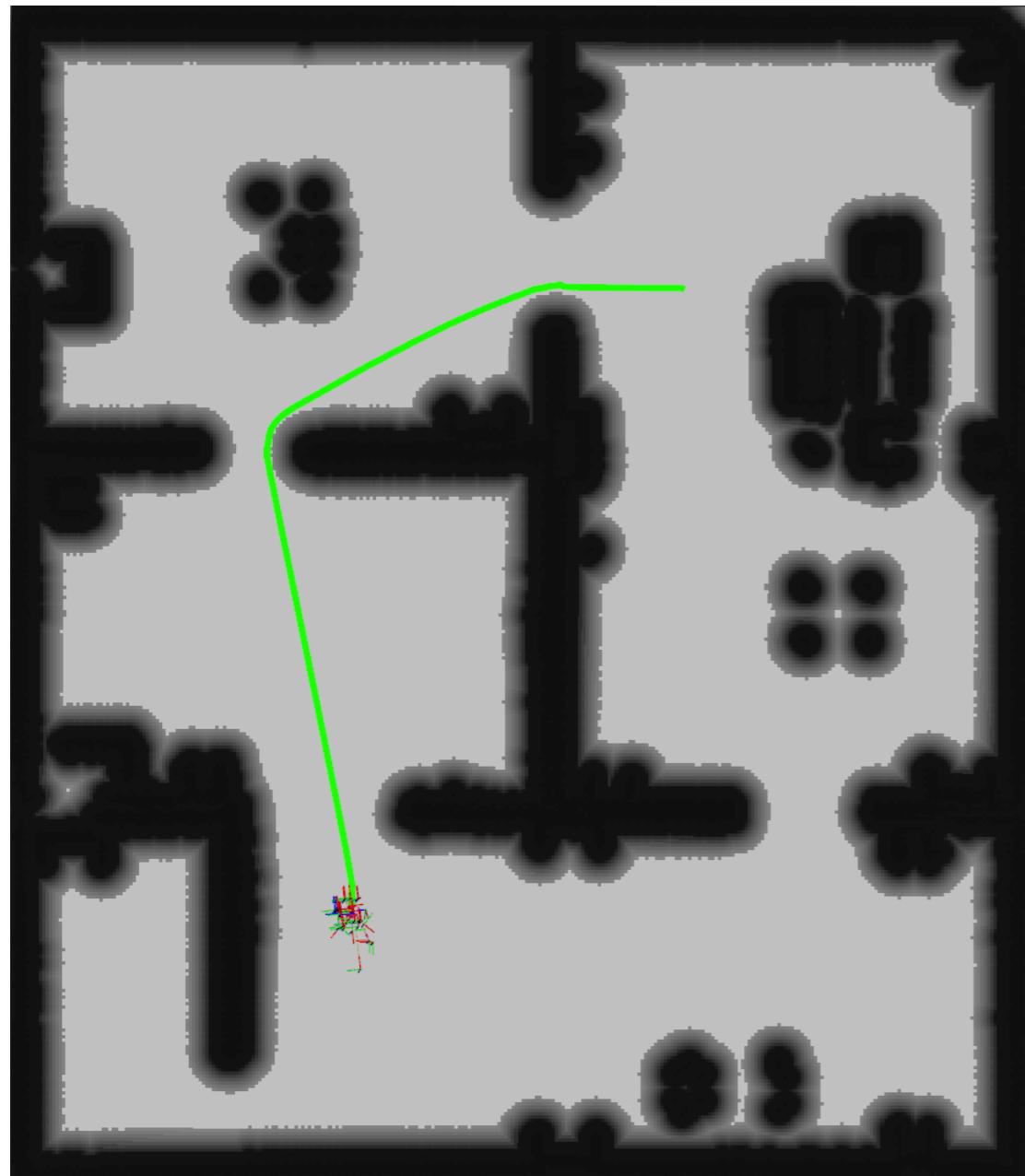
SLAM

ROS  2



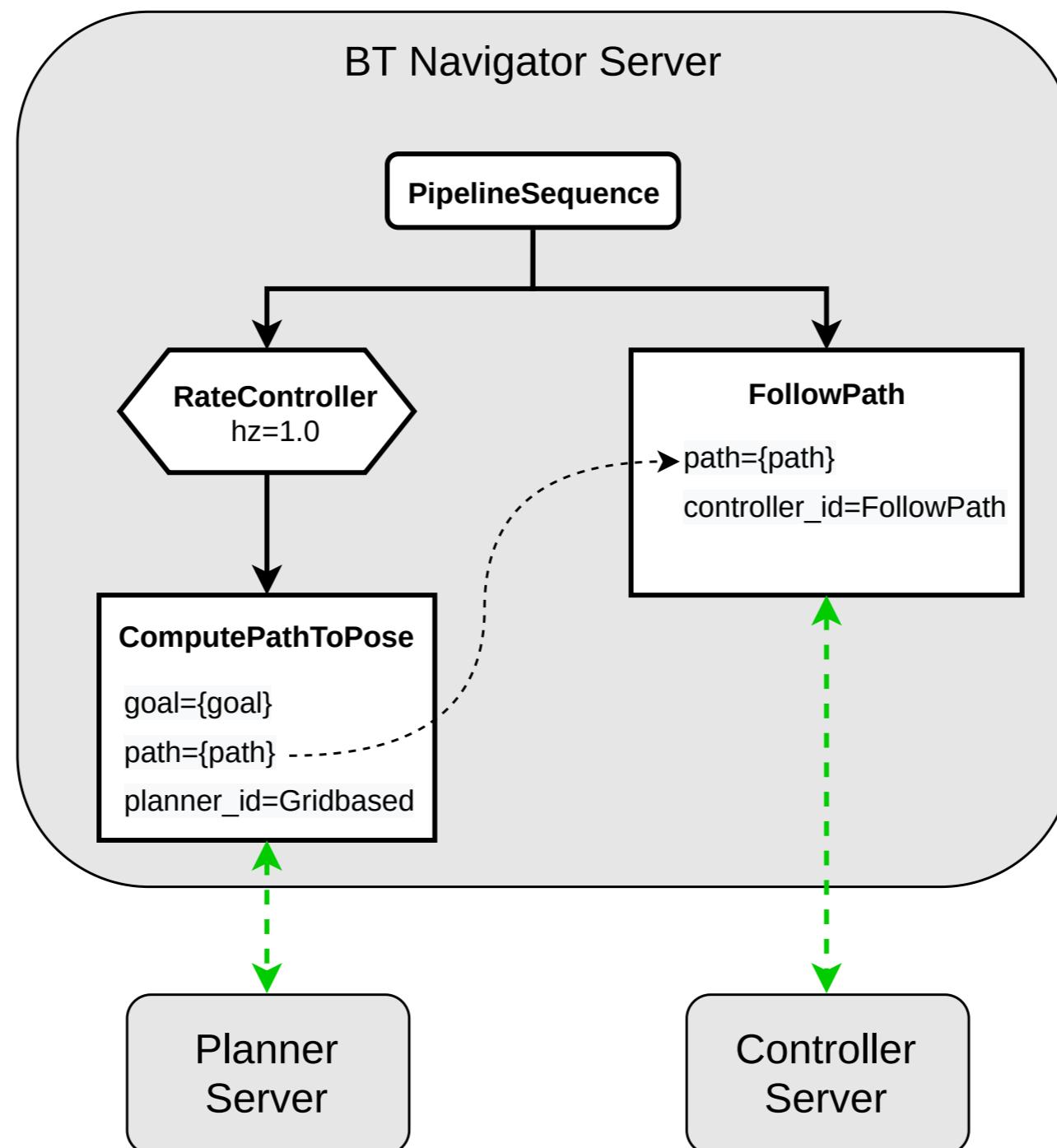
Costmaps

ROS → 2



BT Navigator

ROS → 2



THE MARATHON 2

A NAVIGATION SYSTEM

STEVE MACENSKI, FRANCISCO MARTIN, RUFFIN WHITE,
JONATHAN GINES

IROS 2020

Results

ROS → 2

- Ultra Marathon: 37 miles
- No collisions
- No manual restart
- No navigations aborted
- No students were damaged!!!

	RB1	TIAGo	Total
Time (hrs)	9.4	13.4	22.8
Distance (miles)	15.6	21.8	37.4
Recoveries	52	116	168
Recoveries per mile	3.3	5.3	4.3
Avg. speed (m/s)	0.39	0.35	0.37
Max speed (m/s)	0.45	0.45	0.45
Num. of Collisions	0	0	0
Num. of Emergency stops	0	0	0



Tiago

Pal Robotics (Barcelona, Spain)



- Base + Torso(0.54 m x 1.45 m)
- Max speed: 1.0 ms/s
- Battery: 2 x 36V 20Ah
- Intel i7 - 7700, 8GB RAM, 250 GB SSD
- Ubuntu 16.06 LTS (Kinetic)
- Camera RGBD Orbbec Astra S
- Laser Sick TIM561
- Navigation2 executed externally

RB1 base

Robotnik (Valencia, Spain)

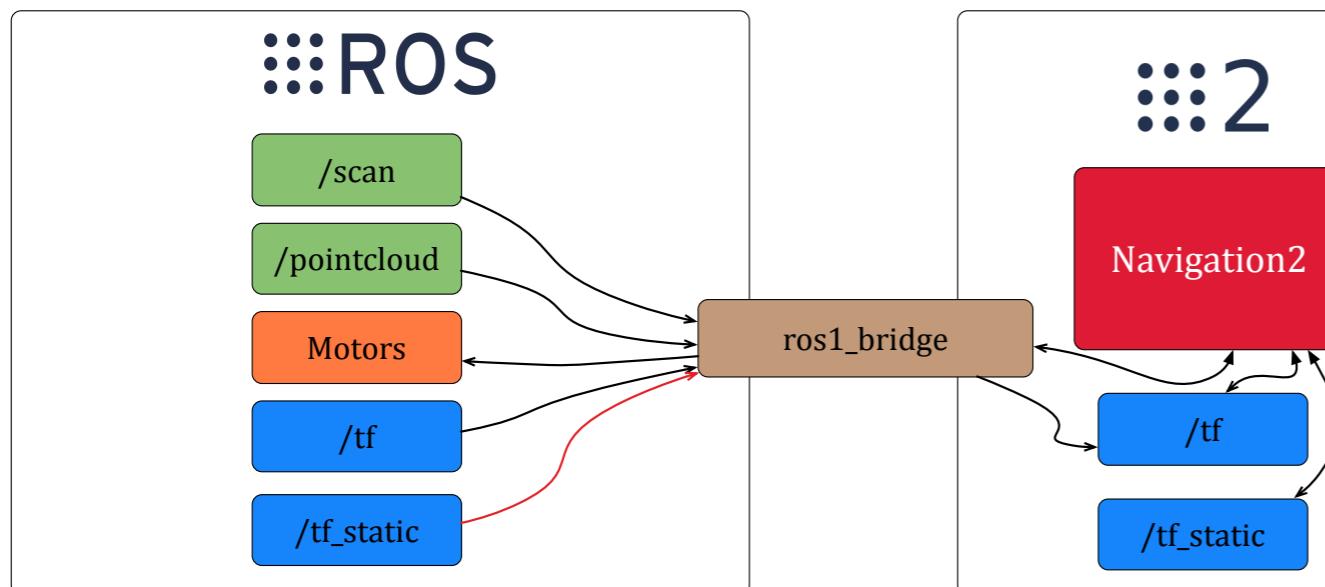


- Base (0.50 m x 0.251 m)
- Max speed: 1.5 ms/s
- Battery: 24V 30Ah
- Intel i7 - 7567, 8GB RAM, 120 GB SSD
- Ubuntu 18.06 LTS (Melodic)
- Camera RGBD Orbbec Astra
- Laser Sick TIM571
- Navigation2 executed onboard

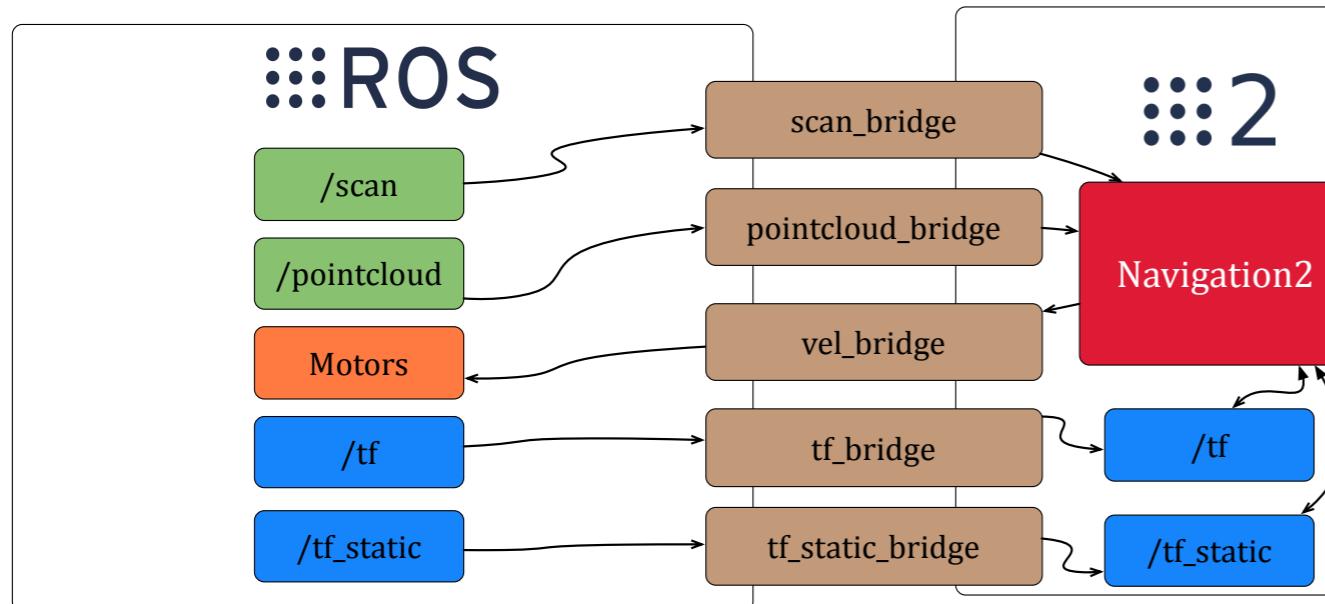
The problems



- The drivers of the robot was for ROS1
- We used ros1_bridge
- Continuous flow problem



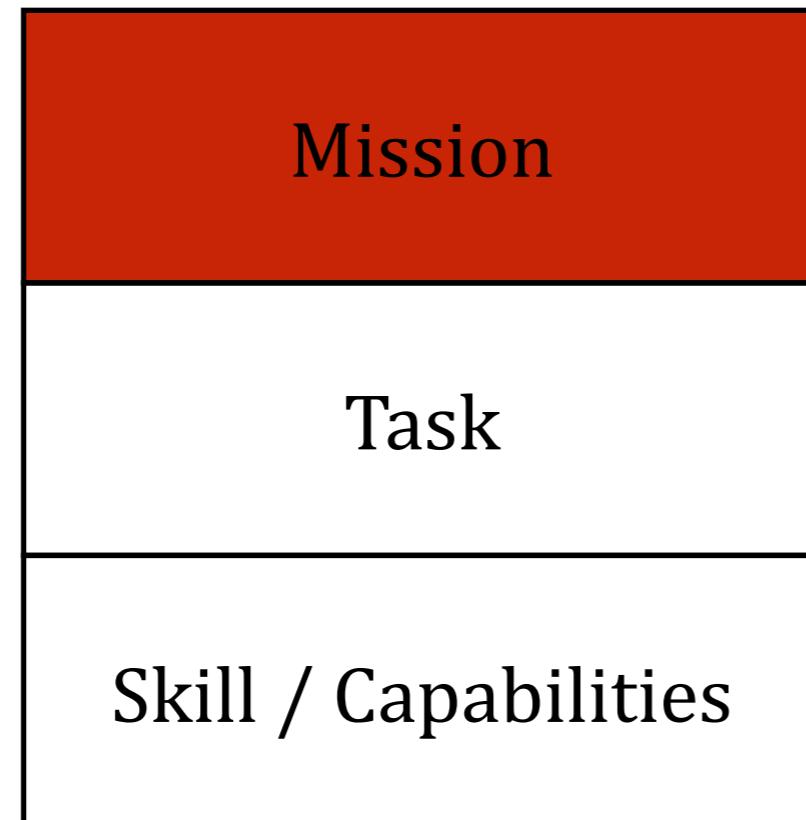
A screenshot of a GitHub repository page for 'ros2/ros1_bridge'. The repository has 9 branches, 32 tags, and 276 commits. The codebase includes .github, bin, cmake, doc, include/ros1_bridge, resource, ros1_bridge, src, test, .gitignore, CHANGELOG.rst, CMakeLists.txt, CONTRIBUTING.md, LICENSE, README.md, and package.xml files. The repository was last updated 14 days ago. A note in the README.md file states: "Bridge communication between ROS 1 and ROS 2".



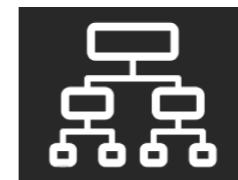
- QoS per topic

Software Architectures

ROS → 2



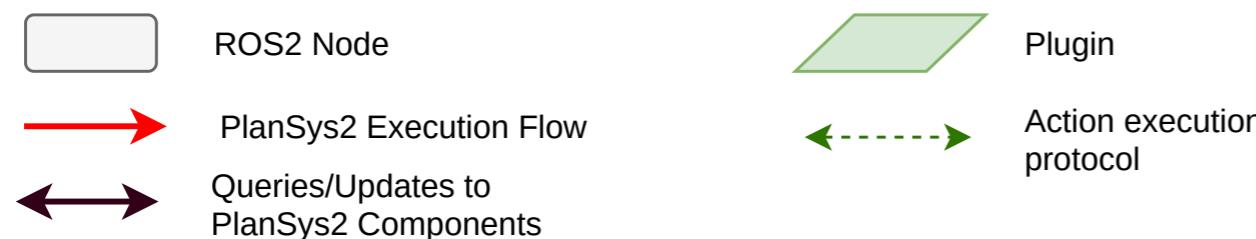
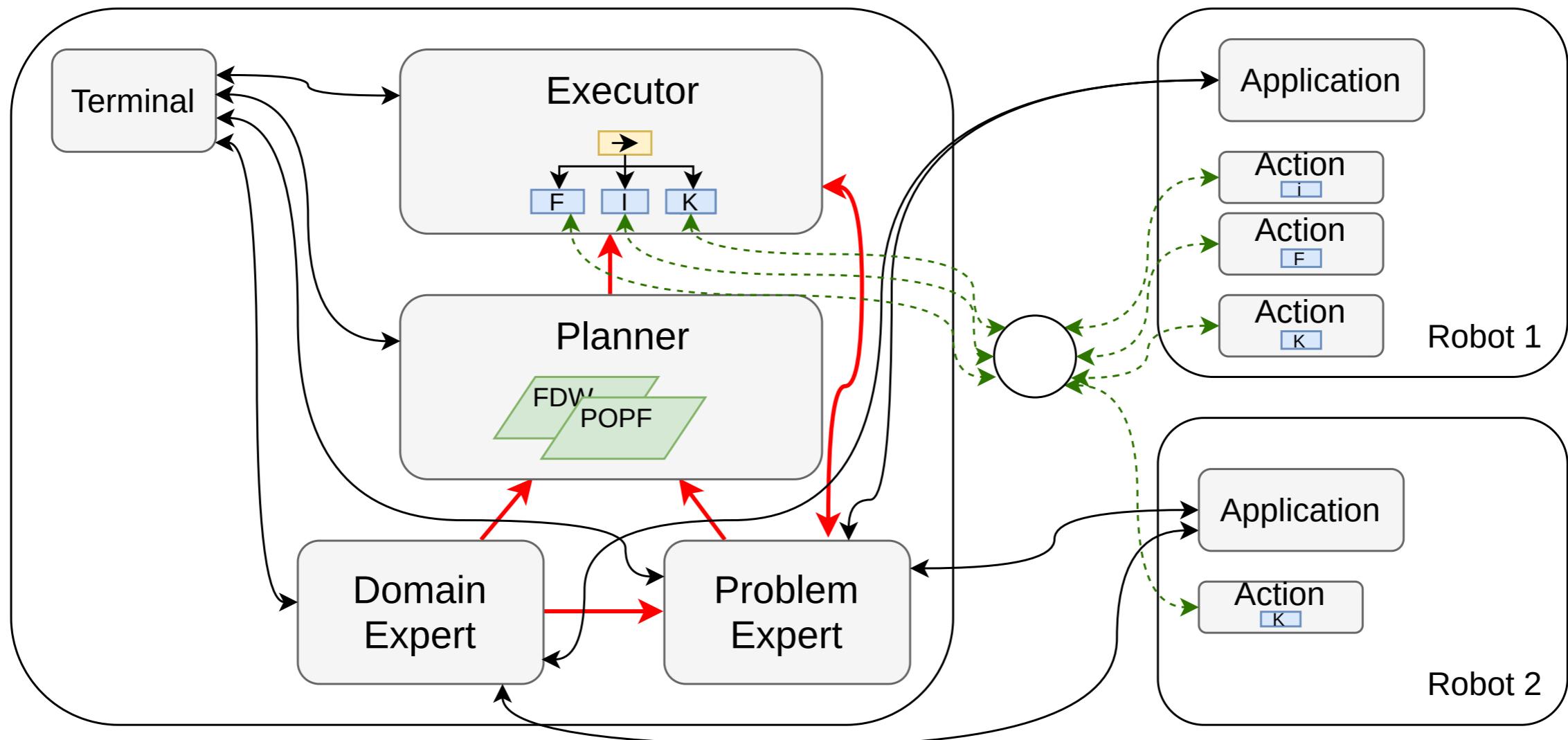
PlanSys 2



N A V 2

:::2 *Planning System*

- ROS2 Planning System (PlanSys2 in short) is a framework for symbolic planning that incorporates novel approaches for execution on robots working in demanding environments
- PlanSys2 aims to be the reference task planning framework in ROS2
- Among its main features, it can be highlighted the optimized execution, based on Behavior Trees, of plans through a new actions auction protocol and its multi-robot planning capabilities



⌘2 **Planning System**

PlanSys2 aims to be the reference Planning System in ROS2

- *Reliable, Secure and Predictable*
- *Modular and Extensible*
- *Open Source*
- *Multi-robot and Specialized Action Performers*
- *Efficient and Explainable*

Software growing up!!

ROS → 2

- Growing last months
- Principal packages ported to ROS2
 - ImageTransport
 - TF2
 - Robots: Kobuki, TB3....
 - Drivers: Cameras, Lasers..
 - Behavior Trees
 - Navigation2
 - PlanSys2
 - ...

Migrate!!

ROS → ROS2

1. package.xml and CMakeLists.txt

- a. Look for dependencies in packages
- b. Look for dependencies in sources
- c. Migrate dependencies, if possible

2. Direct migration:

- a. New message and rclcpp headers
- b. Create a rclcpp::Node. Direct for Nodelets.
- c. Change every publisher / subscriber
- d. ROS_INFO to RCLCPP_INFO,...
- e. Services and actions are a little harder
- f. Launchers

3. Improved migration:

- a. Nodes to classes and instantiate in one executable
- b. Consider usage lifecycle nodes
- c. Rethink QoS
- d. Split in Nodes
- e. Multiple nodes with executors



MIGRATE TO ROS2 !!!!!

ROS2

for
ROS Developers



Muchas gracias!!!



Prof. Dr. Francisco Martín Rico
Madrid 2-03-2022

