ROS 2 – Overview 1

Robotika a počítačové vidění
BPC-PRP

Ing. Miloš Cihlář

Ing. Jakub Minařík

Brno University of Technology
2023

- ROS – what is it and why
- Nodes, messages
- Topics, services, actions
- Tools - CLI, RQT
- Basic Publisher and Subscriber
- Logging

Autonomous robot

[1]

[2]

[3]

HW – pc, motors, chassis, wheels, propellers

Sensors – encoders, imu, lidars, range sensors

RGB cams, IR cams, GNSS

Localization Algorithms

Navigation Algorithms

Mission Planner

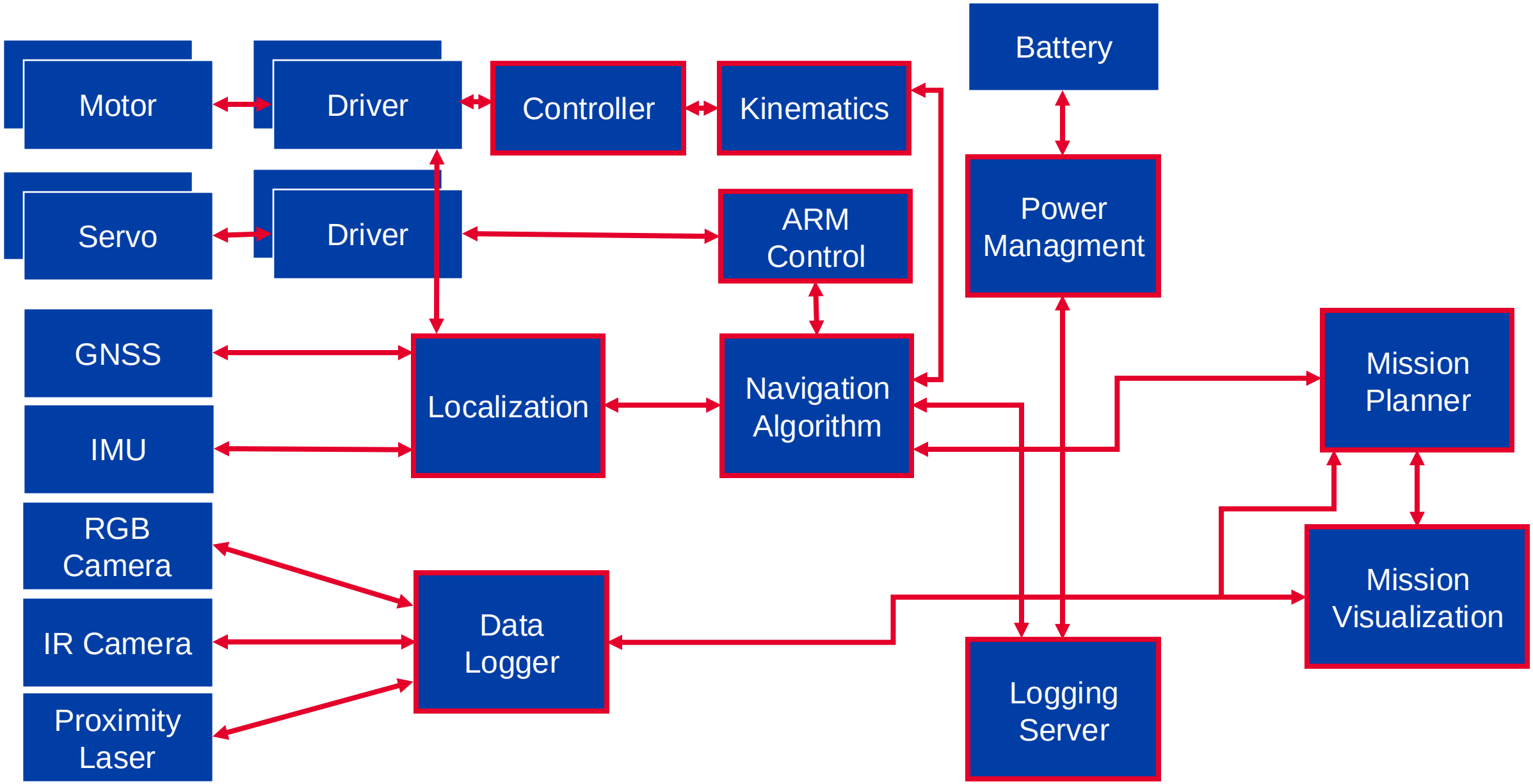
Chassis Motors Control

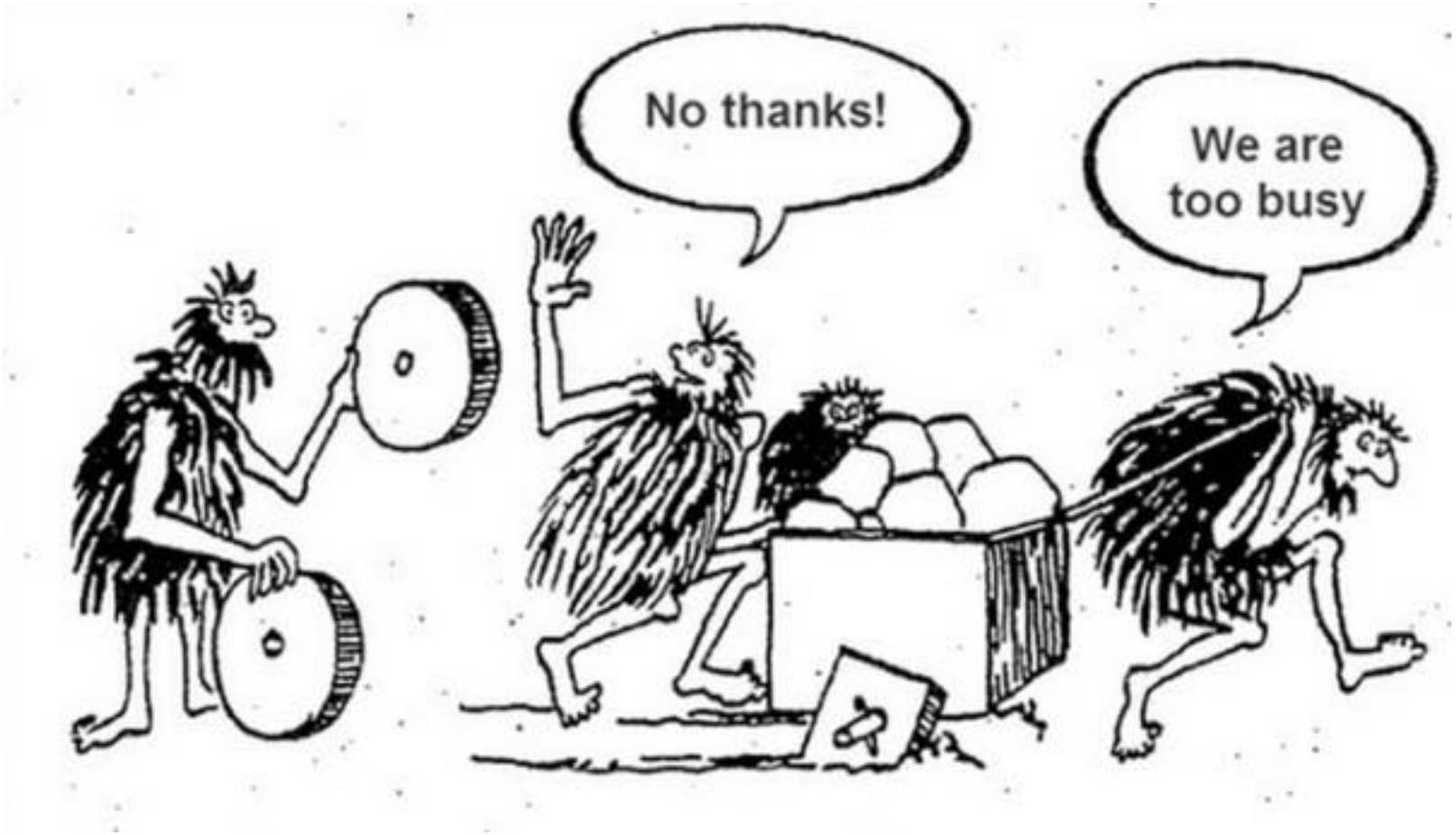
Power Management

Networking (QoS management)

[1] https://www.engineerlive.com/content/autonomous-robot-relies-3d-printing
[2] https://highways.today/2020/07/31/velodyne-lidar-emesent-hovermap/
[3] https://velodynelidar.com/blog/team-costar-velodyne-lidar-darpa-subterranean-challenge-urban-circuit/

DDS – Data Distribution Service





1. https://roboticsbackend.com/what-is-ros/

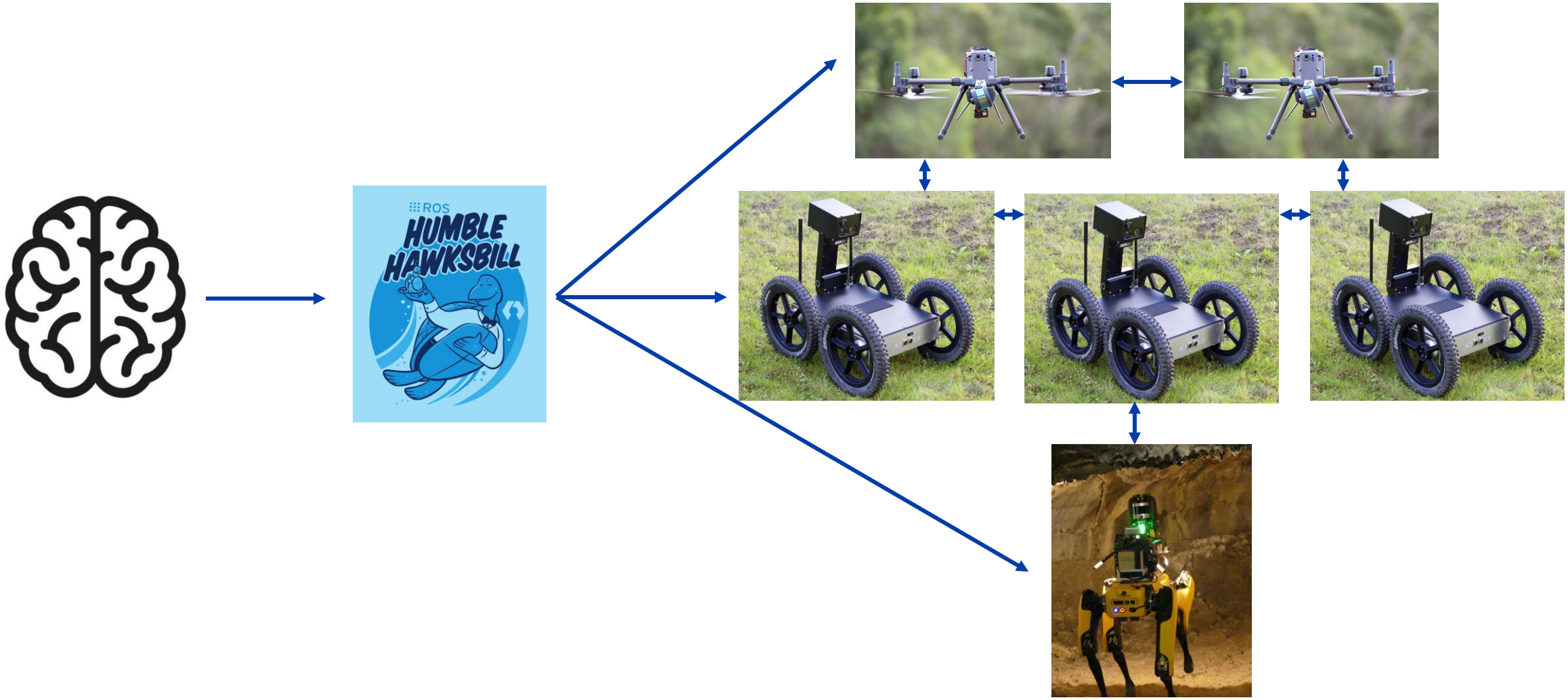
ROS Overview

- **”The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications.** From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project”.
- “Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn’t”.

ROS2 Documentation Website

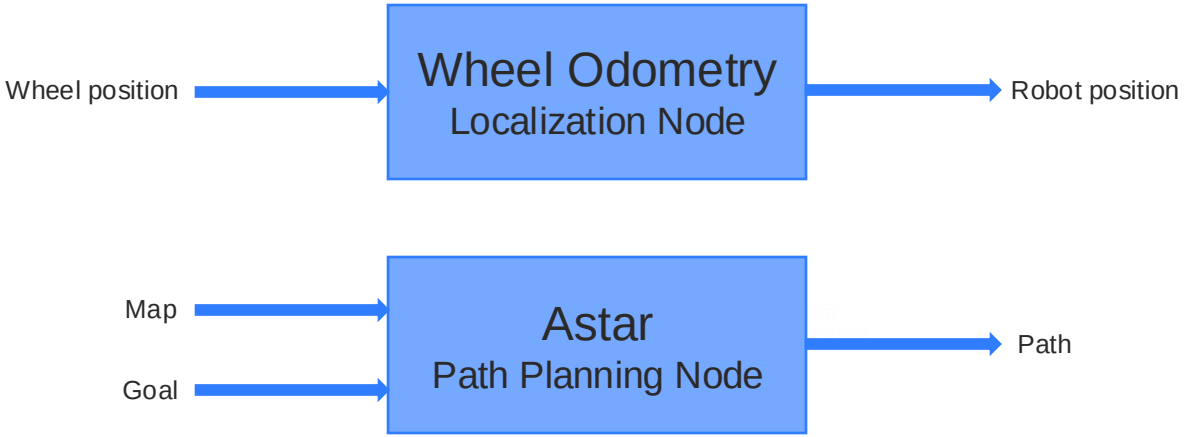
ROS architecture allows to build up a scalable and modular software solution with high level of abstraction between the encapsulated functional blocks

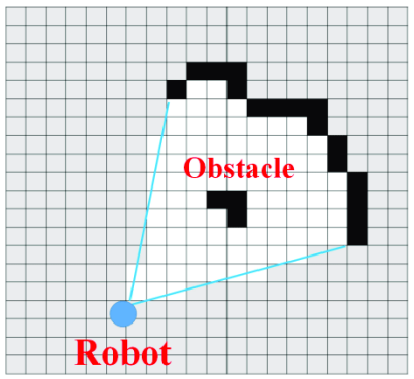
- The most important purpose of the ROS is the abstraction above the communication layer.



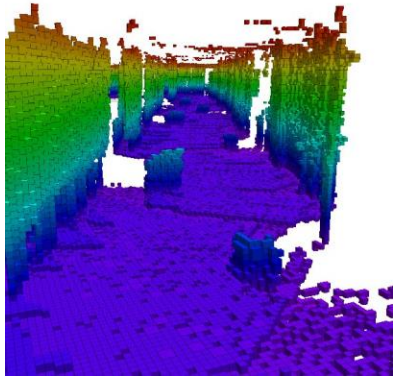
Nodes

- Node is a single entity in ROS2 communication grid
- Node should represent an encapsulated functionality
  - Localization
    - wheel and visual odometry, GPS, …
  - Mapping
    - occupancy grid, point cloud map, octomap, heightmap, …
  - Simultaneous localization and mapping (SLAM)
    - EKF SLAM, Fast SLAM, GraphSLAM, …
  - Motion control
    - PSD, Pure Pursuit, Feedback linearization, Model predictive control
  - Path planning
    - Astar, Dijkstra, Voronoi graph, Rapidly-exploring random tree
- Usually as a node we understand a single instance of program that includes ROS2 functionality
  - Warning: single program can instance multiple nodes
  - Local, global map and planning
- ROS2 is able to discovery other nodes, discovere and propagate topics and send and receive messages
  - Publisher - Subscriber
- Nodes create easy to scale and modular architecture

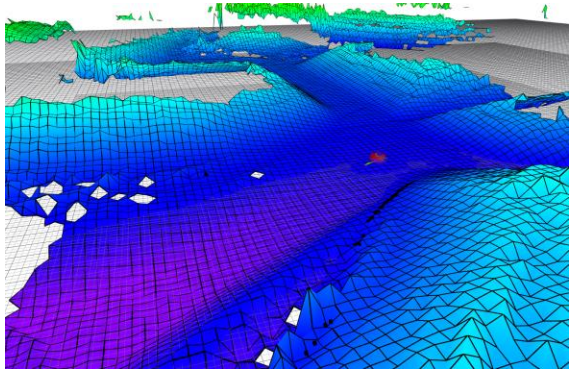




Grid map [1]



Octomap [2]



Elevation map

[1] https://www.researchgate.net/figure/Occupancy-grid-map_fig1_321326800
[2] https://www.researchgate.net/figure/Octomap-of-the-point-cloud-map-shown-in-Figure-14b_fig6_319133173

Message – data types

- Message is a single instance of an information transmitted and received by node(s)
- Each message has a predefined format and is strong typed
- Defined in .msg files – custom messages
- `common_msgs` are messages widely used in ROS packages
  - `geometry_msgs`
    - `Point, PointStamped, Pose, PoseStamped, Vector3`
    - `Twist, TwistStamped`
  - `nav_msgs`
    - `Path, Odometry, OccupancyGrid`
  - `sensor_msgs`
    - `Imu, PointCloud, Range, Image, BatteryState`
- `std_msgs`

https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html
http://wiki.ros.org/common_msgs

Message – data types

File: nav_msgs/Odometry.msg

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Compact Message Definition

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

File: std_msgs/Header.msg

Raw Message Definition

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

Compact Message Definition

```
uint32 seq
time stamp
string frame_id
```

File: geometry_msgs/PoseWithCovariance.msg

Raw Message Definition

```
# This represents a pose in free space with uncertainty.

Pose pose

# Row-major representation of the 6x6 covariance matrix
# The orientation parameters use a fixed-axis representation.
# In order, the parameters are:
# (x, y, z, rotation about X axis, rotation about Y axis, rotation about Z axis)
float64[36] covariance
```

Compact Message Definition

```
geometry_msgs/Pose pose
float64[36] covariance
```

File: geometry_msgs/TwistWithCovariance.msg

Raw Message Definition

```
# This expresses velocity in free space with uncertainty.

Twist twist

# Row-major representation of the 6x6 covariance matrix
# The orientation parameters use a fixed-axis representation.
# In order, the parameters are:
# (x, y, z, rotation about X axis, rotation about Y axis, rotation about Z axis)
float64[36] covariance
```

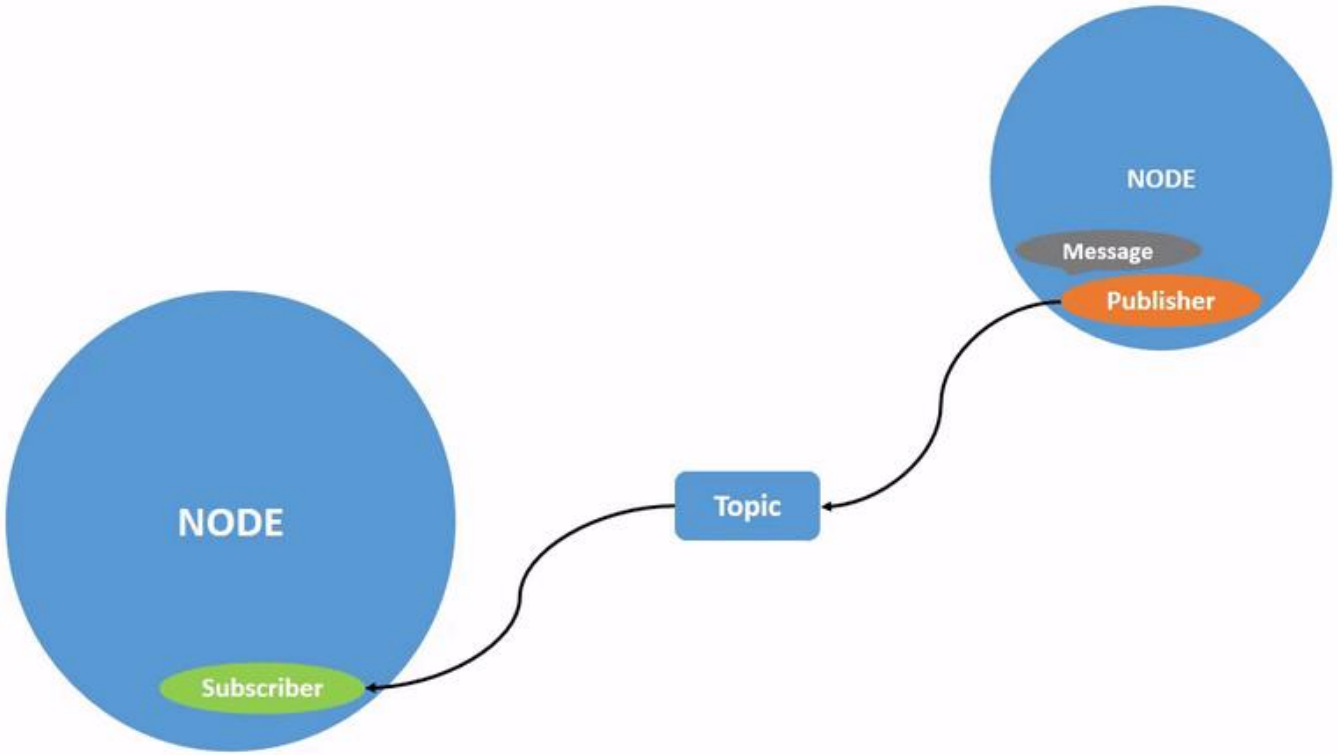
Compact Message Definition

```
geometry_msgs/Twist twist
float64[36] covariance
```

1. http://wiki.ros.org/common_msgs

Topic

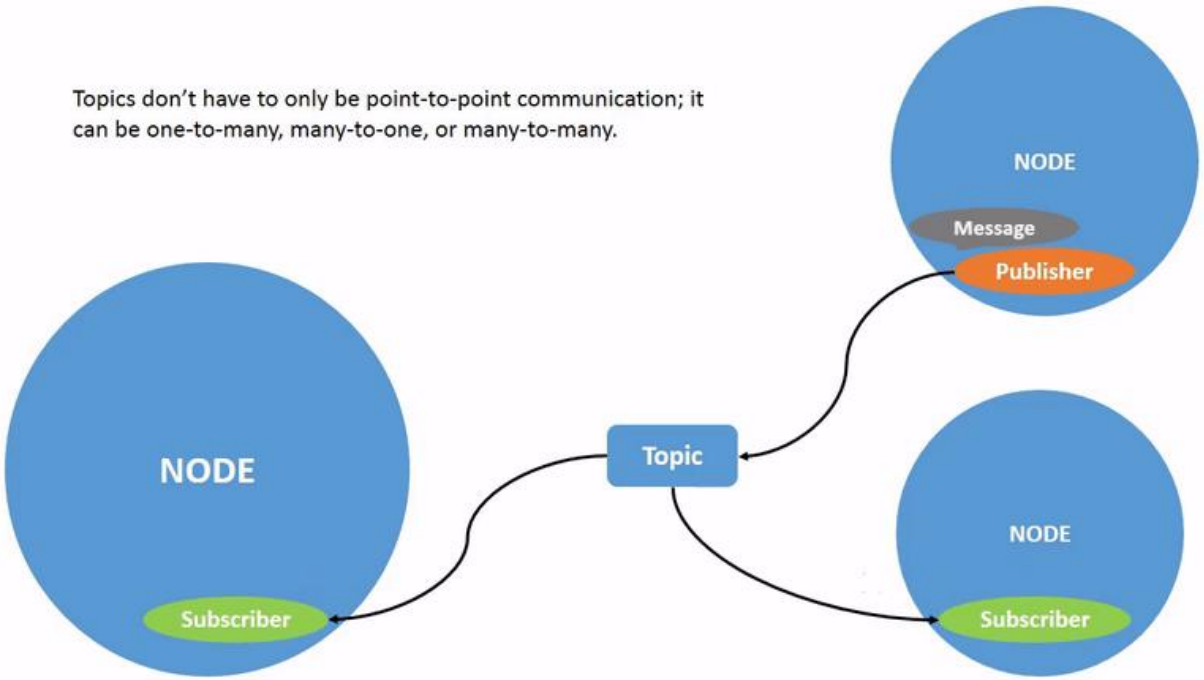
- Topic are an abstract layer above the internal ROS2 communication.
- It allows nodes to discover and exchange information (messages) between each other.
- The topics are defined by human readable string



1. https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

Publisher - Subscriber

- The simplest model of communication in ROS2
- Publisher creates the instance of message and transmits it via the topic. All subscribers, that subscribes this topic are going to recive the copy of the published mesasage
- Example of usage: continual flow of data from sensor to data processing algorithm



Service

- Service is the “On demand” communication model
- **Client node** requests service from **server node** by sending request message.
- **Server node** responses to a specifil **client node** by the response message
- Example: Robot found an obstacle in planned path and requests path panning node toupdate path plan

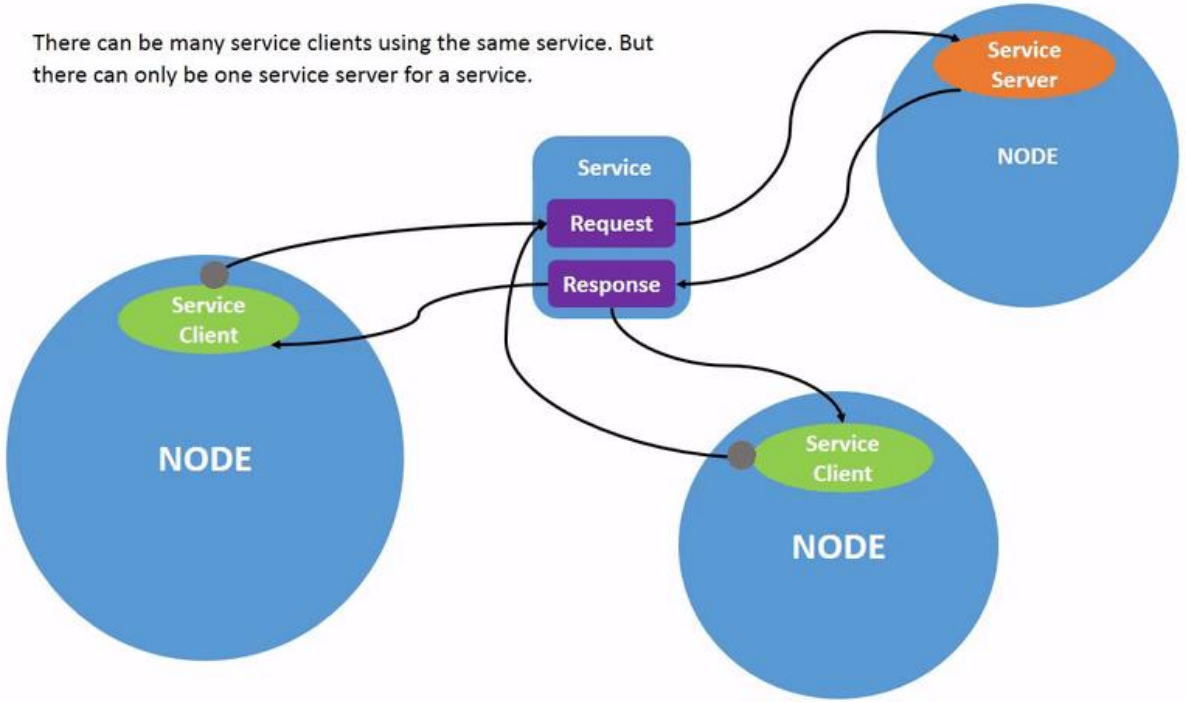
File: nav_msgs/GetPlan.srv

Raw Message Definition

```
# Get a plan from the current position to the goal Pose

# The start pose for the plan
geometry_msgs/PoseStamped start

# The final pose of the goal position
geometry_msgs/PoseStamped goal

# If the goal is obstructed, how many meters the planner can
# relax the constraint in x and y before failing.
float32 tolerance
---
nav_msgs/Path plan
```

Compact Message Definition

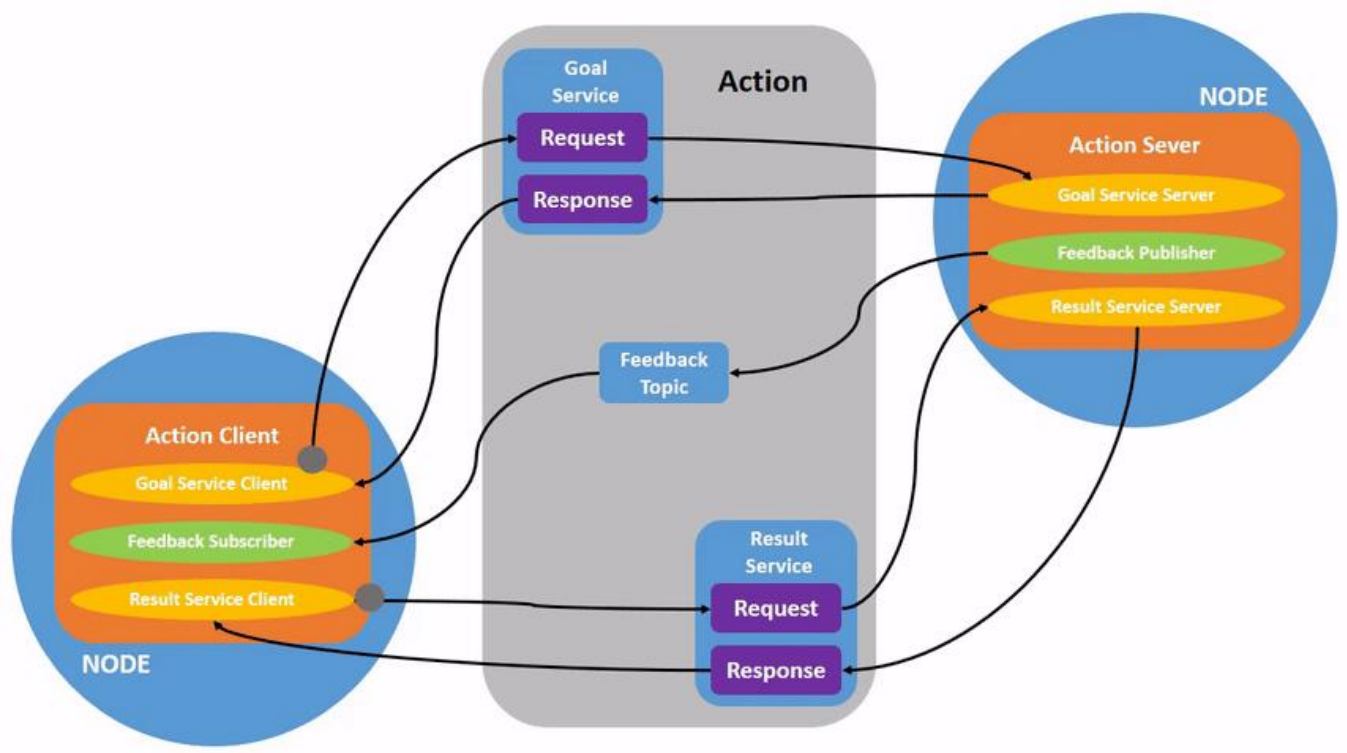
```
geometry_msgs/PoseStamped start
geometry_msgs/PoseStamped goal
float32 tolerance
---
nav_msgs/Path plan
```



1. https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

Action

- Action is a communication pattern used for a long term client-service interaction
- **Client** requests an action from a **server**. **Server** informs **client** about the state of the action via feedback.
- When the action is finished, server terminates the interaction by sending the response.
- Example of usage: long-term actions, like:
  - Navigate robot from point A to B
  - Cumulated long term mesaurements
  - Operations with unpredictable time of duration, like: robot, use the arm to put object into the backet



https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html

Comand Line Interface – Basic commands

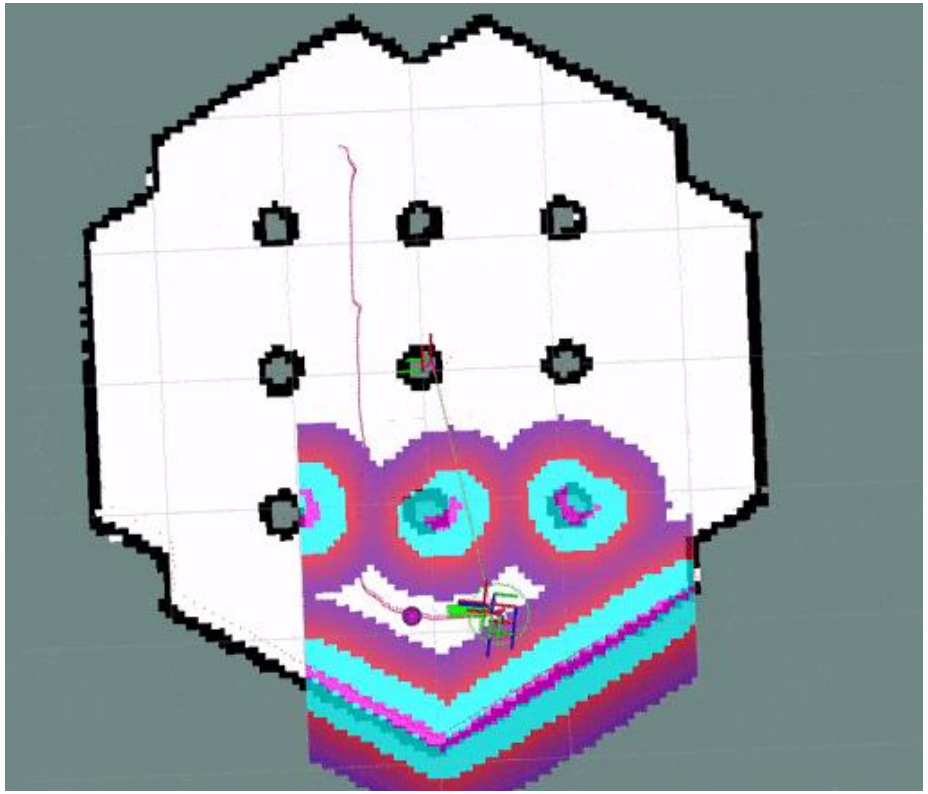
- `ros2 run <package_name> <executable_name>`
  - Run executable, single node from package
- `ros2 launch <package_name> <launch_file_name>`
  - Start complex application with multiple nodes and settings form launchfiles
- `ros2 node list`
  - List all nodes
- `ros2 node info <node_name>`
  - Output information about node
- `ros2 topic/service/action list`
  - List all topics or services or actions

CLI - Topic

- `ros2 topic list`
- `ros2 topic echo <topic_name> [message_type]`
  - Subscribe to topic and display incoming messages
- `ros2 topic pub <topic_name> <message_type> [values]`
  - Publish topic by described name and type, with specific value in YAML format
- `ros2 topic type <topic_name>`
  - Return message type
- `ros2 topic hz <topic_name>`
- `ros2 topic find <message_type>`

Packages and Code Distribution

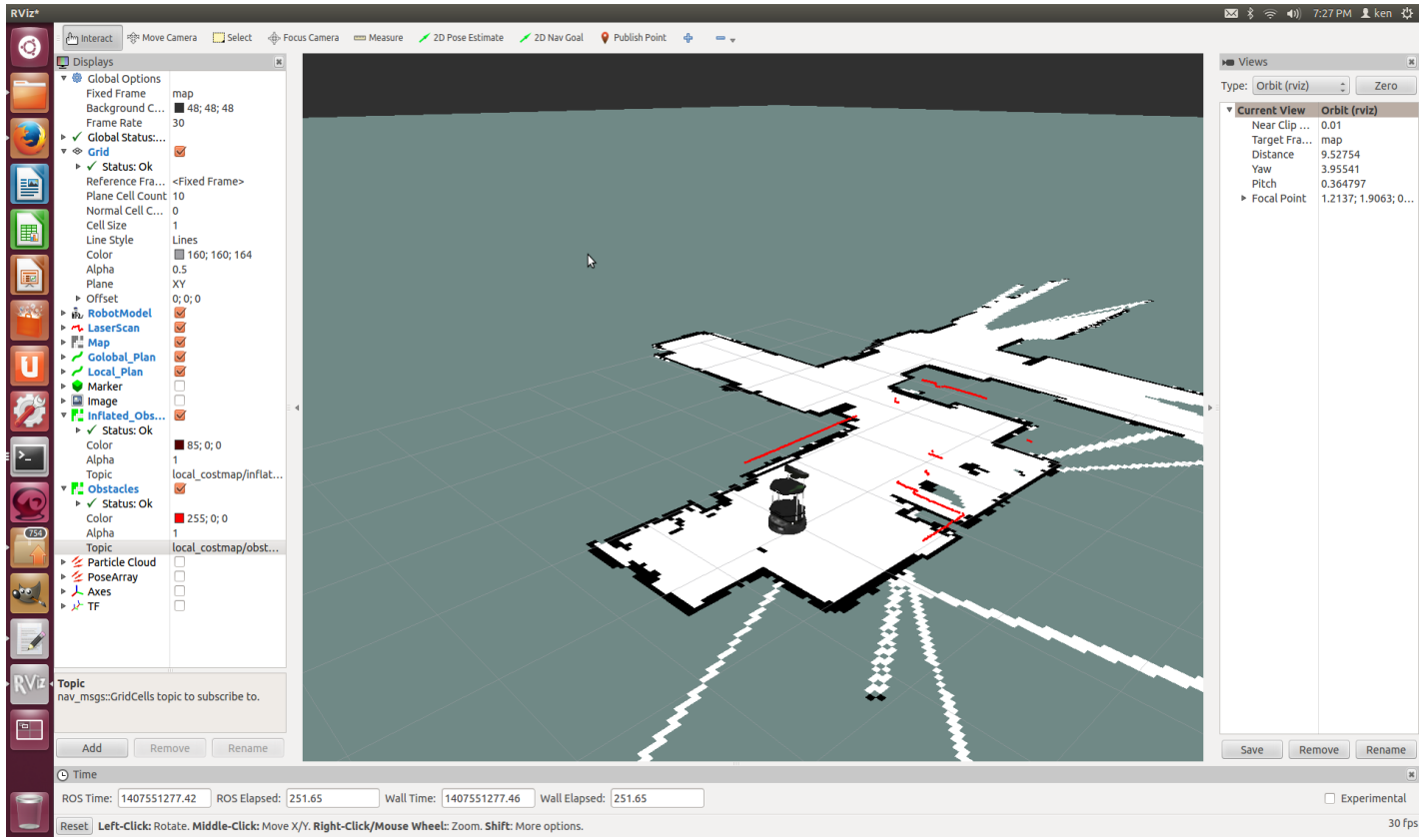
- Packages are the way to organize the ROS code into the encapuslated baches.
- Package contains code to compile/run single or multiple executables
- Each of these executables contains one or more nodes
- Packages can be uploaded to web and provided to community
- Examples:
  - **Navigation2** package contains all code, algorithms and resources to run nodes that will handle robot navigations
  - **MoveIt2** - MoveIt 2 is the robotic manipulation platform for ROS 2
  - **Ros2_control** is a framework for (real-time) control of robots
- Online package repo: https://index.ros.org/packages/



https://github.com/ros-planning/navigation2/blob/main/nav2_regulated_pure_pursuit_controller/README.md

Visualization - RVIZ

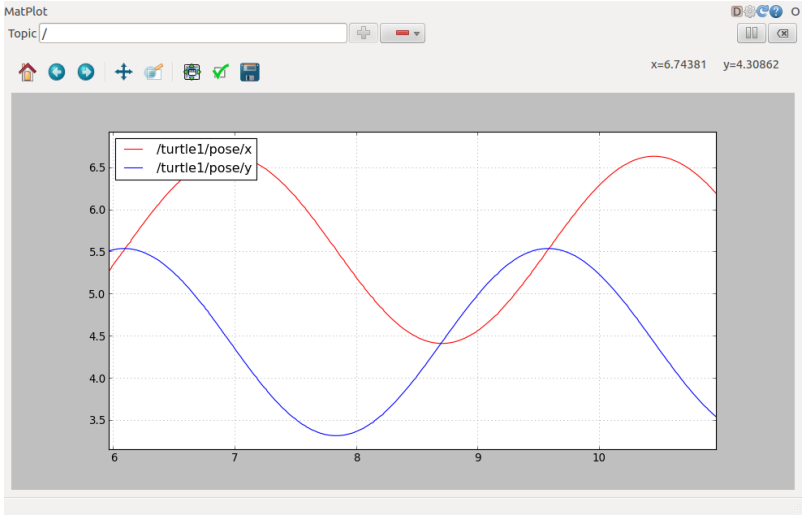
- ROS2 includes the visualization tool called RVIZ
- RVIZ is a GUI program that allows to render objects in 3D
- Useful for visualizing robot‘s position, telemetry, states, map, detections, surrounding environment, etc.



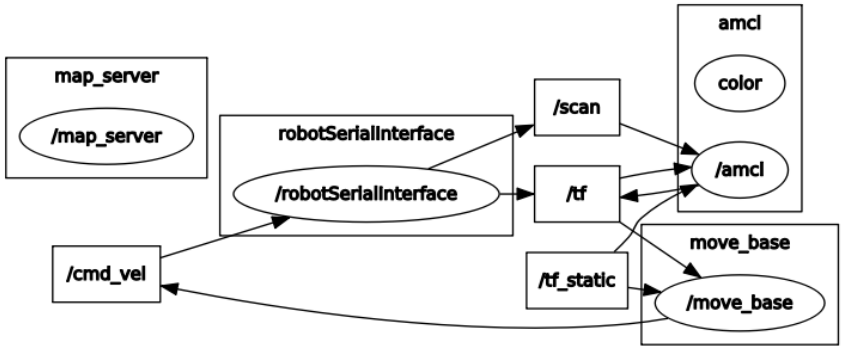
https://answers.ros.org/question/189547/obstacle-doesnt-show-on-rviz/

Visualization - RQT

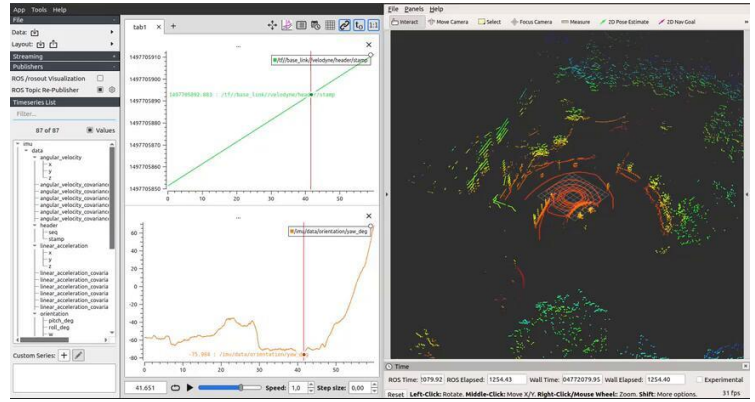
- RQT_GRAPH – visualize nodes-topics interconnection grid
- RQT_PLOT – visualize time series of scalar published on some topic
- RQT_BAG – tool to handle ROS2 bag content and playback
- PlotJuggler – Third party visualization tool (https://plotjuggler.io/)



https://github.com/ros-visualization/rqt_plot/issues/39



https://answers.ros.org/upfiles/15026035891055113.png



https://plotjuggler.io/

ROS2 Examples – C++ and Python Node Examples

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class MinimalPublisher(Node):
  def __init__(self):
    super().__init__('minimal_publisher')
    self.publisher_ = self.create_publisher(String, 'topic')
    timer_period = 0.5 # seconds
    self.timer = self.create_timer(timer_period, self.timer_callback)
    self.i = 0

  def timer_callback(self):
    msg = String()
    msg.data = 'Hello World: %d' % self.i
    self.publisher_.publish(msg)
    self.i += 1

def main(args=None):
  rclpy.init(args=args)
  minimal_publisher = MinimalPublisher()
  rclpy.spin(minimal_publisher)

if __name__ == '__main__':
  main()
```

ROS2 Examples – C++ and Python Node Examples

```cpp
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
using std::placeholders::_1;

class MinimalSubscriber : public rclcpp::Node
{
public:
  MinimalSubscriber(): Node("minimal_subscriber") {
    subscription_ = this->create_subscription<std_msgs::msg::String>(
      "topic", std::bind(&MinimalSubscriber::topic_callback, this, _1));
  }

private:
  void topic_callback(const std_msgs::msg::String::SharedPtr msg) {
    RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg->data.c_str())
  }
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalSubscriber>());
  rclcpp::shutdown();
  return 0;
}
```

- RCLCPP_DEBUG
- RCLCPP_INFO
- RCLCPP_WARN
- RCLCPP_ERROR
- RCLCPP_FATAL

- ros2 run <package> <exec> --ros-args --log-level debug

```
[DEBUG] [1740933157.726833473] [rcl]: Subscription take
succeeded: true

[DEBUG] [1740933158.226544559] [rcl]: Calling timer

[INFO] [1740933158.226629828] [logger_usage_demo]:
Timer callback called (this will only log once)

[INFO] [1740933158.226827642] [logger_usage_demo]:
Publishing: 'Current count: 0'

[ERROR] [1740933158.226934273] [logger_usage_demo]:
Modulo divisor cannot be 0

[DEBUG] [1740933158.226994624] [logger_usage_demo]:
Count is even (expression evaluated to true)

[DEBUG] [1740933158.726437638] [rcl]: Calling timer

[INFO] [1740933158.726468896] [logger_usage_demo]:
Publishing: 'Current count: 1'

[DEBUG] [1740933158.726502748] [logger_usage_demo]:
Count divides into 12 (function evaluated to true)

[INFO] [1740933159.056950542] [rclcpp]:
signal_handler(signum=2)

[DEBUG] [1740933159.057098489] [rclcpp]:
signal_handler(): notifying deferred signal handler
```

Ing. Jakub Minařík
203294@vut.cz

Brno University of Technology
Faculty of Electrical Engineering and Communication
Department of Control and Instrumentation

Robotics and AI Research Group