ROS2 LECTURE

ROBOT GUIDANCE AND SOFTWARE

BASED ON NOTES BY KARL KRUUSAMÄE

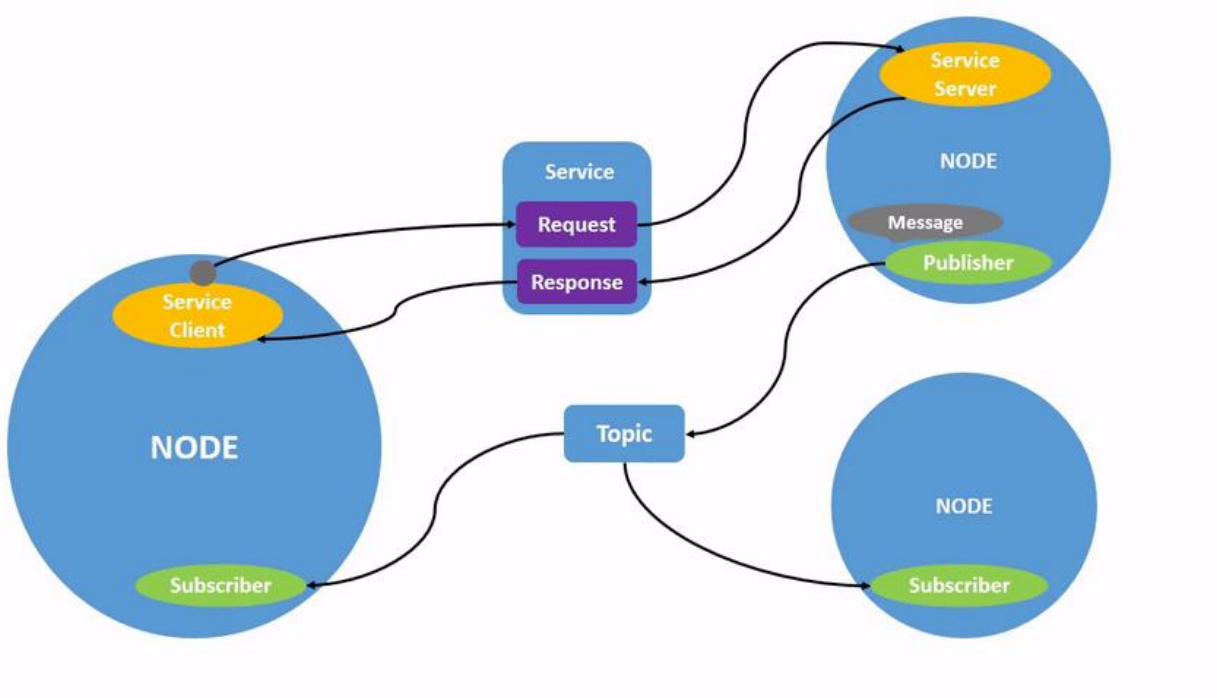
Center for Biorobitics
Tallinn University of Technology

ROS

FUNDAMENTALS

BASIC ROS2 CONCEPTS

- ROS graph - a network of ROS 2 elements processing data together at the same time.
- Node - software module that is sending and/or receiving data from other nodes. Each can send and receive data from other nodes via topics (also services and actions).
- Message - a way for a ROS 2 node to send data on the network to other ROS nodes.
- Topic - one of the of styles interfaces provided by ROS 2, used in (continuous) stream of messages of defined type.
- Publisher - Producer of data
- Subscriber - User of data

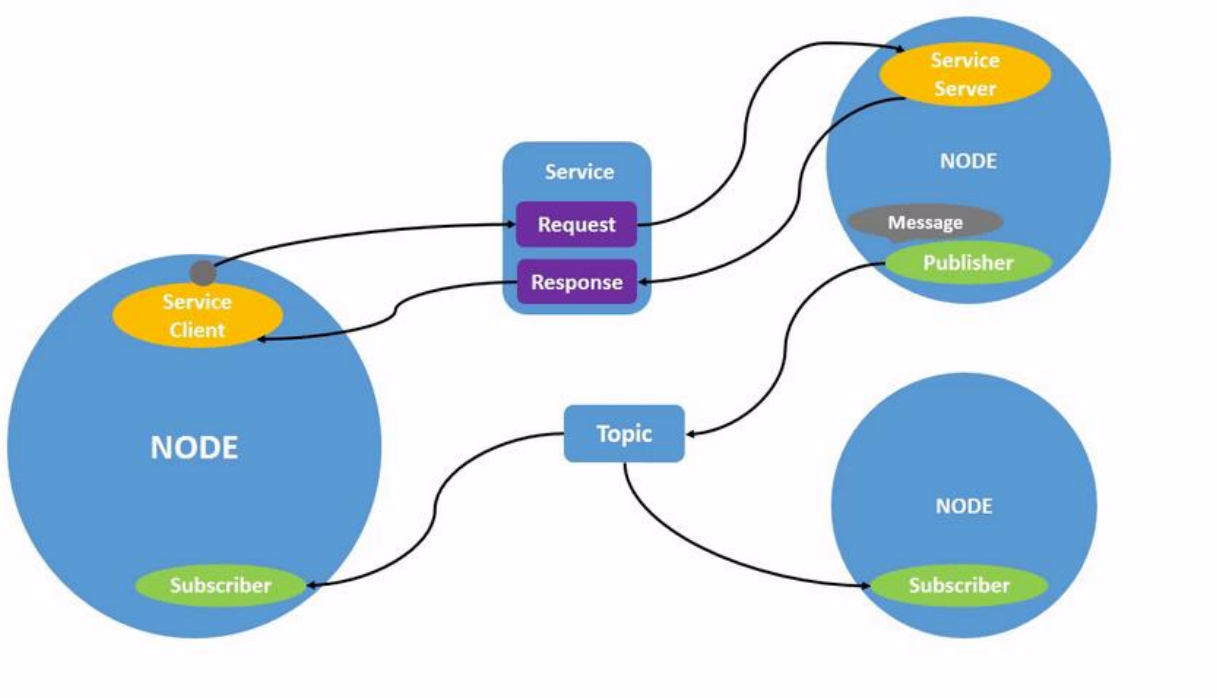


MORE ROS2 CONCEPTS

- Service - a service refers to a remote procedure call.
- Action - an action refers to a long-running remote procedure call with feedback and the ability to cancel or preempt the goal.
- Parameter - a configuration value of a node.

CONFIGURATION FILES

- URDF – Unified Robot Description Format
- Launch - automate the running of many nodes with a single command.
- Package - A single unit of software, including source code, build system files, documentation, tests, and other associated resources.

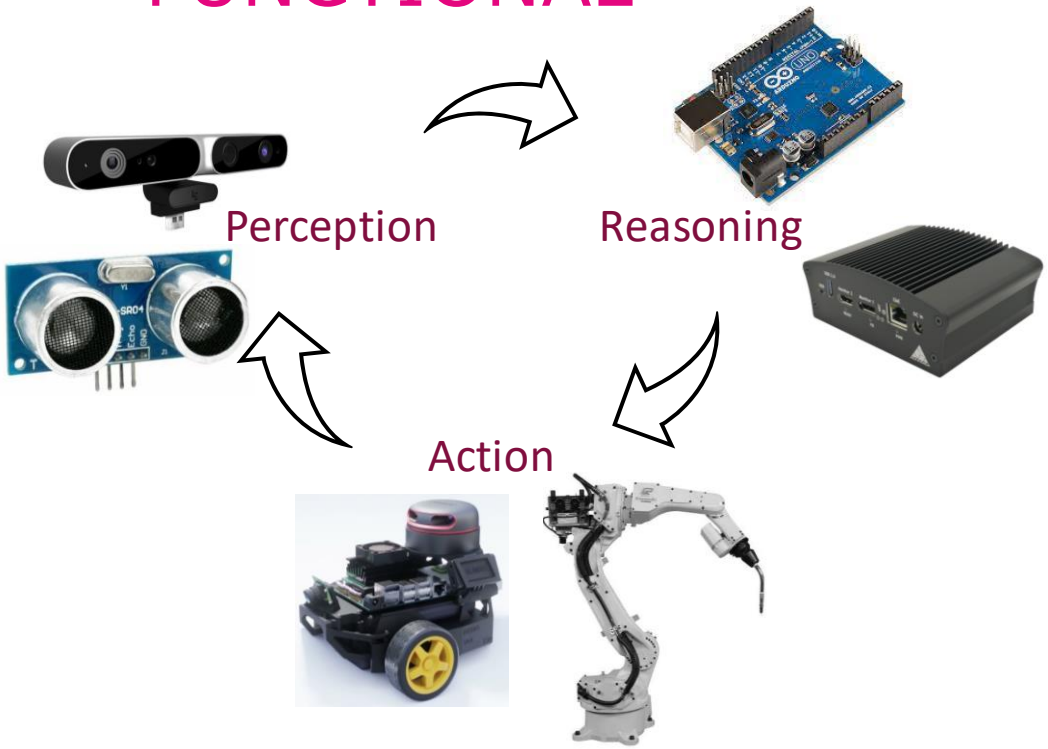


ROS

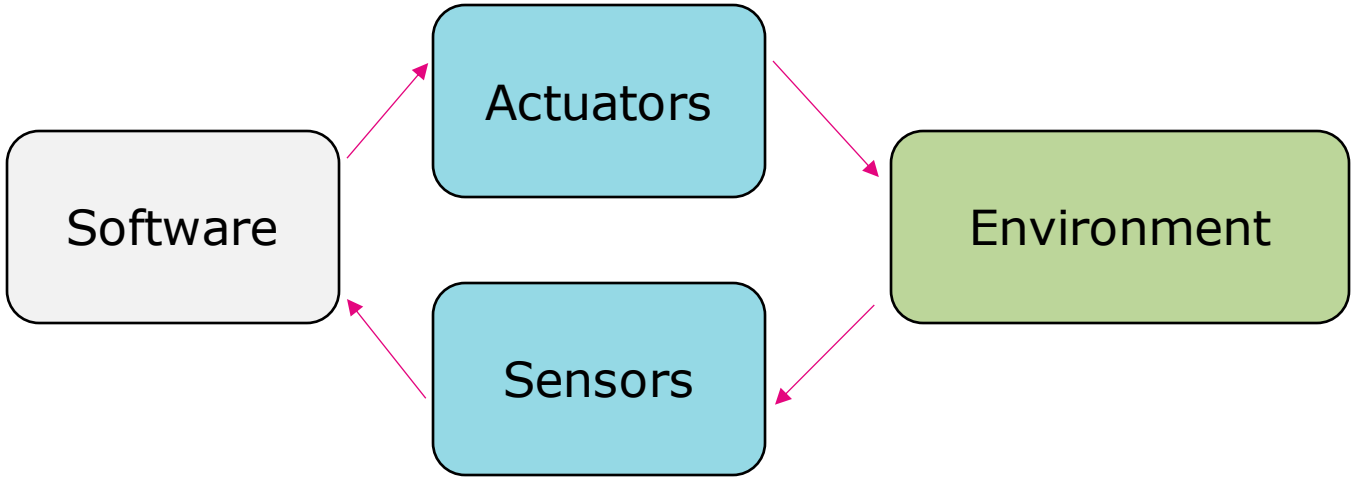
CONCEPTS AND CONVENTIONS

ROBOTS DEFINITION

FUNCTIONAL



ARCHITECTUAL

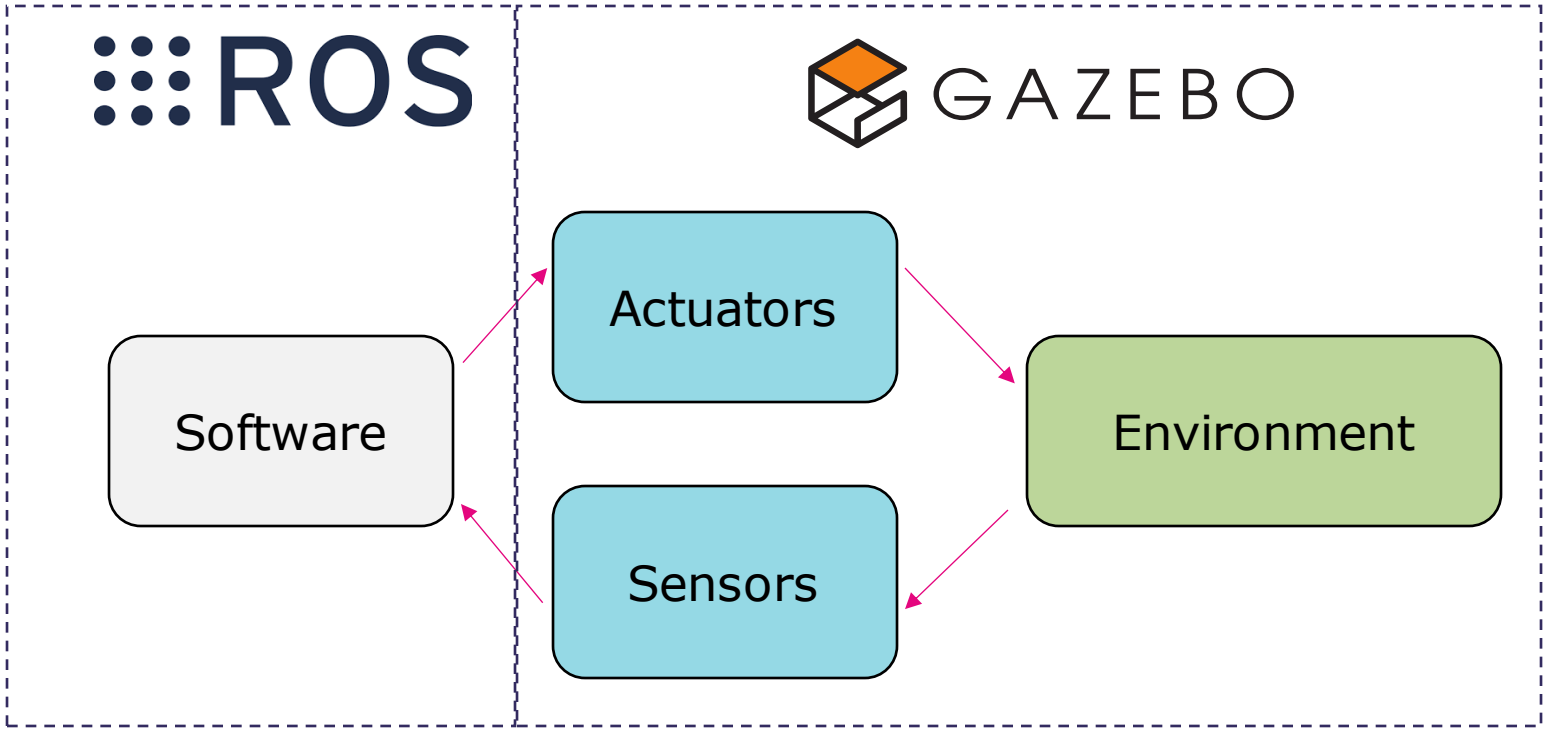


Robots are software connecting sensors to actuators to interact with the environment

TAL TECH

WHY ROS?

- Break complex software into smaller pieces
- Provide a framework, tools and interfaces for distributed development
- Encourage re-use of software
- Easy transition between simulation and hardware



TAL TECH

ROS2 PYTHON PACKAGES

Package - an organizational unit for your ROS 2 code.

It might contain ROS nodes, ROS-independent libraries, datasets, configuration files, 3rd party software, etc.

EXAMPLE PACKAGE:

```
my_package/
      package.xml
      resource/my_package
      setup.cfg
      setup.py
      my_package/
```

Naming convention for packages: lower case, underscore separators

MINIMUM REQUIRED CONTENTS

- **package.xml** - file containing meta information about the package
- **resource/<package_name>** - marker file for the package
- **setup.cfg** - required when a package has executables, so ros2 run can find them
- **setup.py** - containing instructions for how to install the package
- **<package_name>** - a directory with the same name as your package, used by ROS 2 tools to find your package, contains __init__.py

TAL TECH

PACKAGE.XML - INFORMATION AND REQUIRED DEPENDENCIES FOR THE PACKAGE.

- <name> - package name.
- <version> – the version of your package.
- <description> – short description of what the package does.
- <maintainer> - name and email of current maintainer. Can have multiple maintainer tags and author tags (with name and email) if distinction between authors and maintainers is needed.
- <license> - if you ever want to publish your package you'll need a license (for example BSD, MIT, GPLv3).
- <*_depend> - tag names ending with _depend. This is where your package.xml would list its dependencies on other packages, for colcon to search for.

```
<?xml version="1.0"?>
<?xml-model
   href="http://download.ros.org/schema/package_format3.xsd"
   schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
 <name>my_package</name>
 <version>0.0.0</version>
 <description>TODO: Package description</description>
 <maintainer email="user@todo.todo">user</maintainer>
 <license>TODO: License declaration</license>

 <test_depend>ament_copyright</test_depend>
 <test_depend>ament_flake8</test_depend>
 <test_depend>ament_pep257</test_depend>
 <test_depend>python3-pytest</test_depend>

 <export>
   <build_type>ament_python</build_type>
 </export>
</package>
```

TAL TECH

SETUP.PY

The **setup.py** file contains the same description, maintainer and license fields as package.xml, so you need to set those as well.

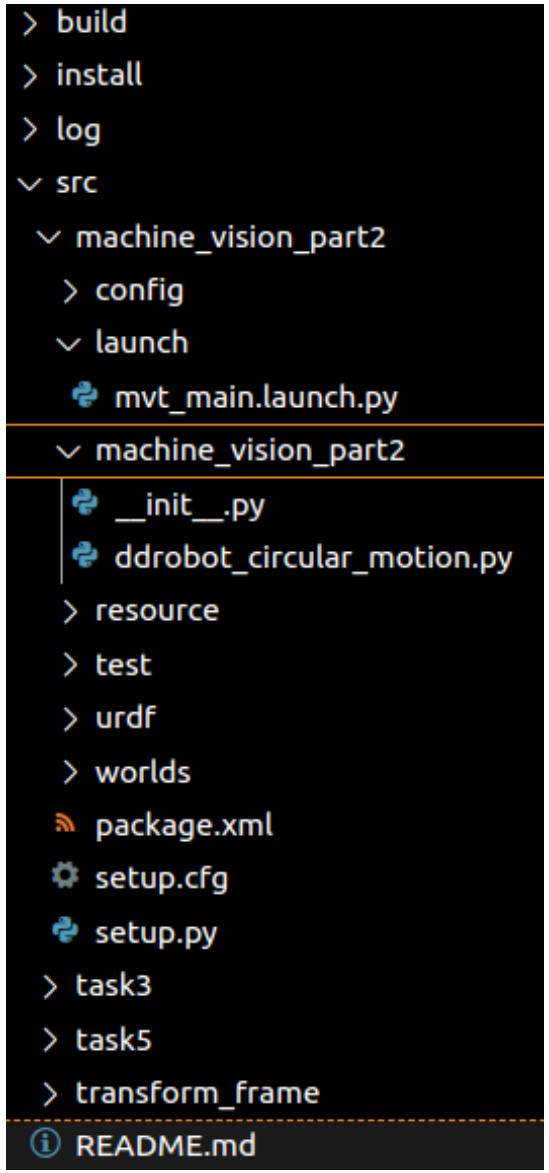
They need to match exactly in both files. The version and name (package_name) need to match exactly as well and should be automatically populated in both files.

```
from setuptools import setup

package_name = 'my_py_pkg'

setup(
 name=package_name,
 version='0.0.0',
 packages=[package_name],
 data_files=[
     ('share/ament_index/resource_index/packages',
             ['resource/' + package_name]),
     ('share/' + package_name, ['package.xml']),
   ],
 install_requires=['setuptools'],
 zip_safe=True,
 maintainer='TODO',
 maintainer_email='TODO',
 description='TODO: Package description',
 license='TODO: License declaration',
 tests_require=['pytest'],
 entry_points={
     'console_scripts': [
             'my_node = my_py_pkg.my_node:main'
     ],
   },
)
```

TAL TECH

OTHER FILES AND FOLDERS

- launch/ – folder containing launch files
- msg/ - folder containing custom messages
- srv/ - folder containing Service (srv) types
- config/ - folder containing configuration files
- data/ - folder containing different data files (images, .db3, etc.)
- urdf/ - folder for your robot urdf



TAL TECH

ROS CONVENTIONS: UNITS AND COORDINATES

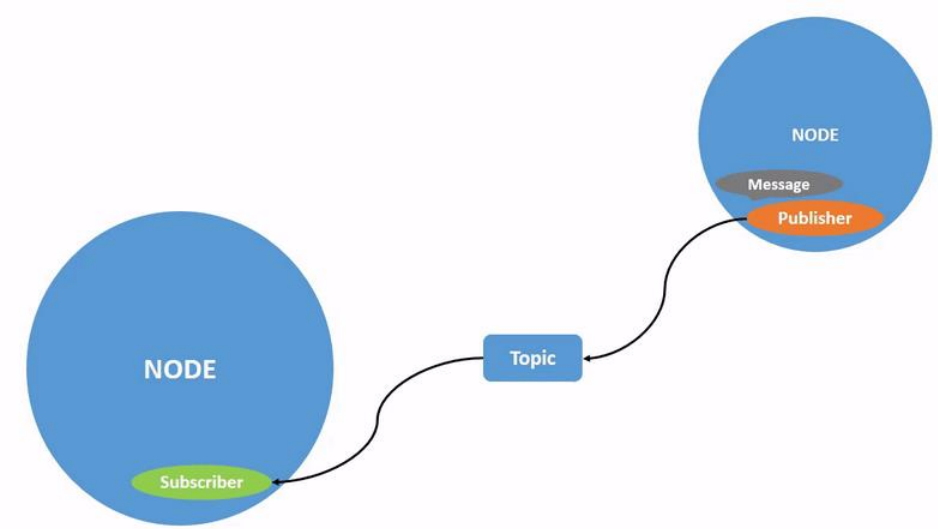
- SI-units and SI-derived units
- Right-handed coordinates. In relation to a body the standard is:
  - x forward
  - y left
  - z up
- Rotation – preferred Quaternions

Quantity	Units
Length	**Meter**
Mass	**Kilogram**
Time	**Second**
Current	Amper
Angle	Radian
Frequency	Herz
Force	Newton
Power	Watt
Voltage	Volt
Temperature	Celcius
Magnetism	Tesla

TAL TECH

PACKAGES AND NODES

- The command syntax for creating a new package in ROS 2 is: **ros2 pkg create --build-type ament_python <package_name>**
- Every ROS nodes belongs to a ROS package
- One package can contain multiple nodes
- Nodes are executables
- Command **ros2 run** launches executables from a package **ros2 run <package_name> <executable_name>**



TAL TECH

ROS NODE - A PROCESS THAT PERFORMS COMPUTATION

A robot system will usually comprise of many nodes

Benefits:

- Additional *fault tolerance* as crashes are isolated to individual nodes
- *Reduced code complexity* when compared to monolithic systems
- Nodes are *language agnostic*. You can write one node in Python, another node in C++, and both can communicate without any problem. Python and C++ are the 2 most common languages for ROS. Through some libraries you can use other languages to create new nodes.

```python
import rclpy
from rclpy.node import Node

class MyNode(Node):
    def __init__(self):
        super().__init__('my_node_name')
        self.create_timer(0.2, self.timer_callback)

    def timer_callback(self):
        self.get_logger().info("Hello ROS2")

def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
    rclpy.shutdown()

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

TAL TECH

SIMPLE ROS NODE

Simple ROS Node

- Node is created, and the timer starts.
- Then the node spins. This allows the timer to continue working.
- Press CTRL+C in the terminal, ends the spinning (and the timer ends).
- The program exits and the node is destroyed.

```python
import rclpy
from rclpy.node import Node

class MyNode(Node):
    def __init__(self):
        super().__init__('my_node_name')
        self.create_timer(0.2, self.timer_callback)

    def timer_callback(self):
        self.get_logger().info("Hello ROS2")

def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
    rclpy.shutdown()

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

```
$ ros2 run my_robot_tutorials minimal_python_node
[INFO] [my_node_name]: Hello ROS2
[INFO] [my_node_name]: Hello ROS2
[INFO] [my_node_name]: Hello ROS2
[INFO] [my_node_name]: Hello ROS2
[INFO] [my_node_name]: Hello ROS2
```

TAL TECH

BASIC ROS NODE EXPLAINED

Ros node in python: print “Hello ROS2” at 5Hz.

```python
import rclpy
from rclpy.node import Node

class MyNode(Node):
    def __init__(self):
        super().__init__('my_node_name')
        self.create_timer(0.2, self.timer_callback)

    def timer_callback(self):
        self.get_logger().info("Hello ROS2")

def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
    rclpy.shutdown()

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

Each ROS2 Python node is a superset of "rclpy.node.Node”.

1. Create Class which inherits from the rclpy Node class, and call super() in the constructor, in order to initialize the Node object.
2. Create a timer with "create_timer()" method from the inherited node class. Two arguments passed are time in seconds and timer function.
3. The timer_callback() method is part of the class created.
4. get_logger().info() will print a ROS2 log on the terminal.

The main function:

1. initialize ROS communications *rclpy.init(args)*
2. Instantiate the custom node *node = MyNode()*
3. Pause program execution *rclpy.spin(node).*
4. The node is continuously executed until node is killed (e.g., CTRL+C in terminal)
5. *rclpy.shutdown()* will shutdown what you started when you executed *rclpy.init()*

TAL TECH

LAUNCH FILES

Launch files enable running **multiple nodes** with a single command and **add logic** to your startup sequence.

- **Launch actions** – specific things launch file does. E.g., **Node action**
  - Important Node options are:
    - **namespace**: Launch the Node in a namespace, e.g., my_node
    - **parameters**: List of parameter files that are loaded for the Node (e.g., parameters="my_params.yaml")
    - **remappings**: Remapping of node names, e.g., remappings=[('/input/pose', '/turtlesim1/turtle1/pose')]
    - **arguments**: List of ROS arguments to hand over to the Node, e.g., arguments=["-topic", "/ddrobot/robot_description", "-entity", "ddrobot", "-z", "0.2"]
    - **arguments**: List of arguments to hand over to the Node, e.g., arguments=['-name', 'my_robot']
- **Event handlers** – The processes started from the launch files have different states (e.g., starting, running, stopping, etc.). When the state of a process changes we call it an event and event handlers allow us to react to those events.
- **Substitutions** - make your launch files more flexible. There is no need to hard coding all values.
- **Conditions** - help to add more logic to the start-up procedure. Conditions allow to make the execution of the actions dependent on something.

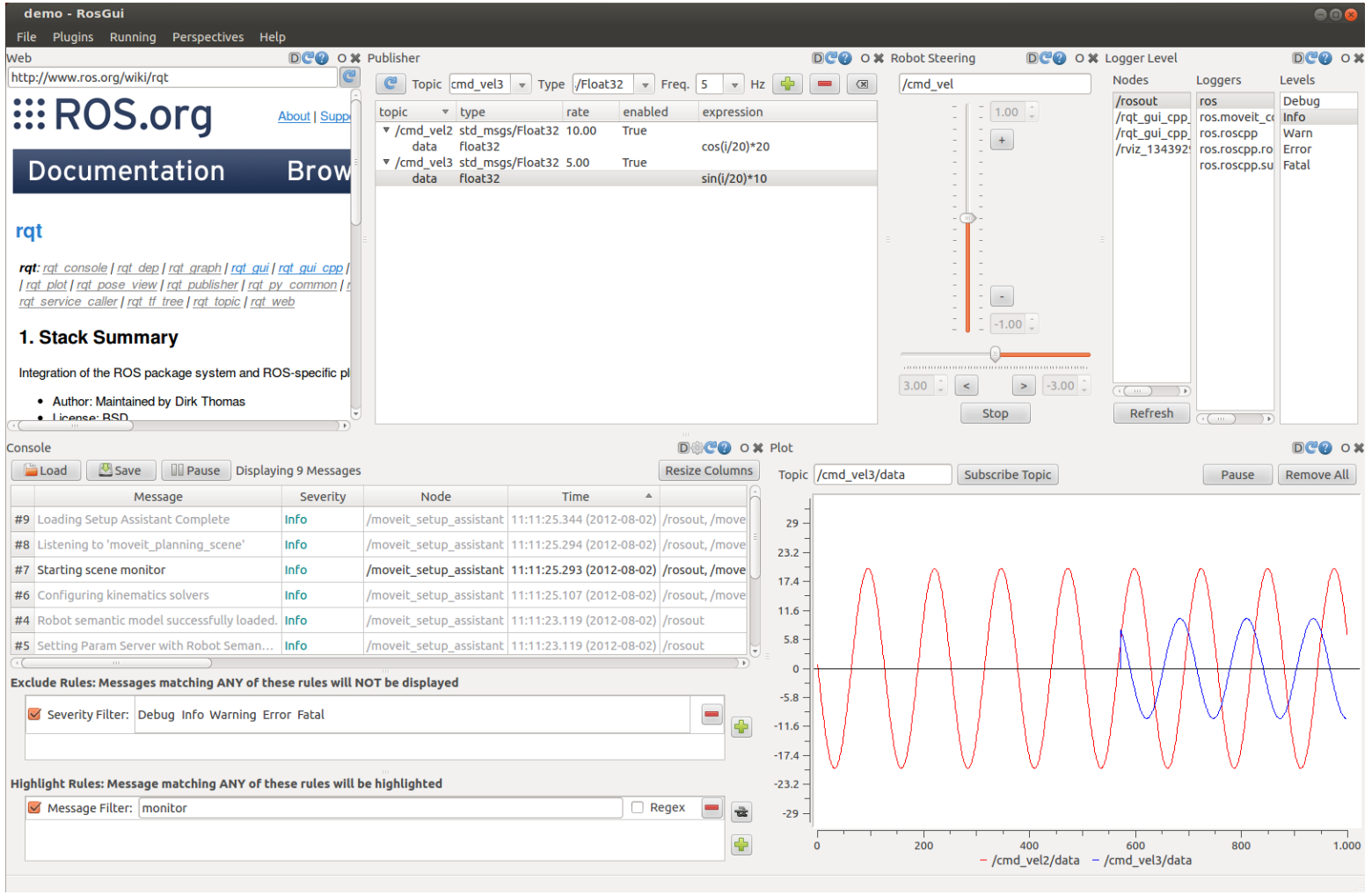
```
$ ros2 launch <package> <launch_file>
```

TAL TECH

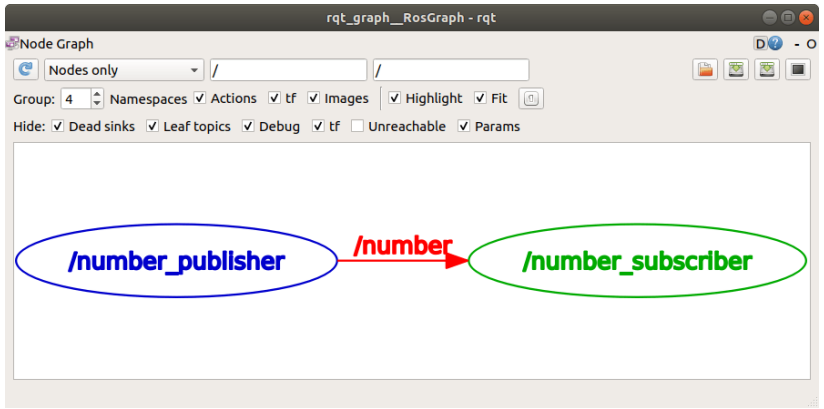
ROS

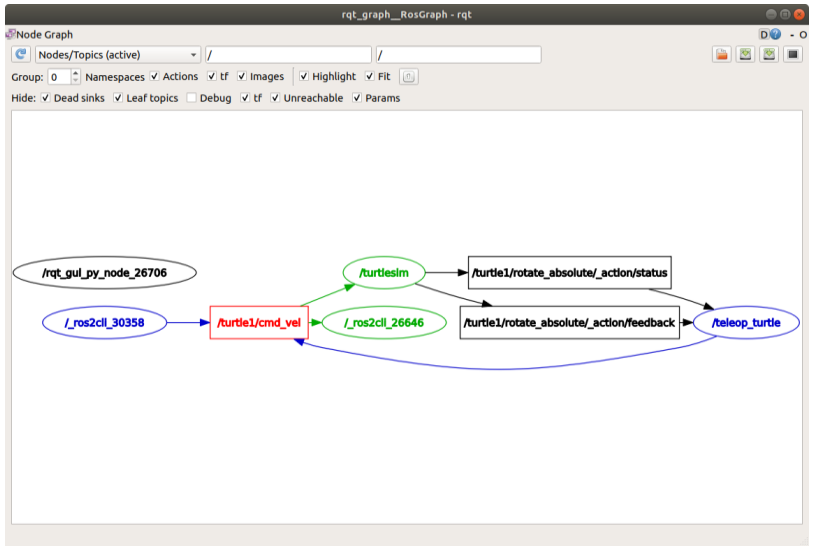
TOOLS

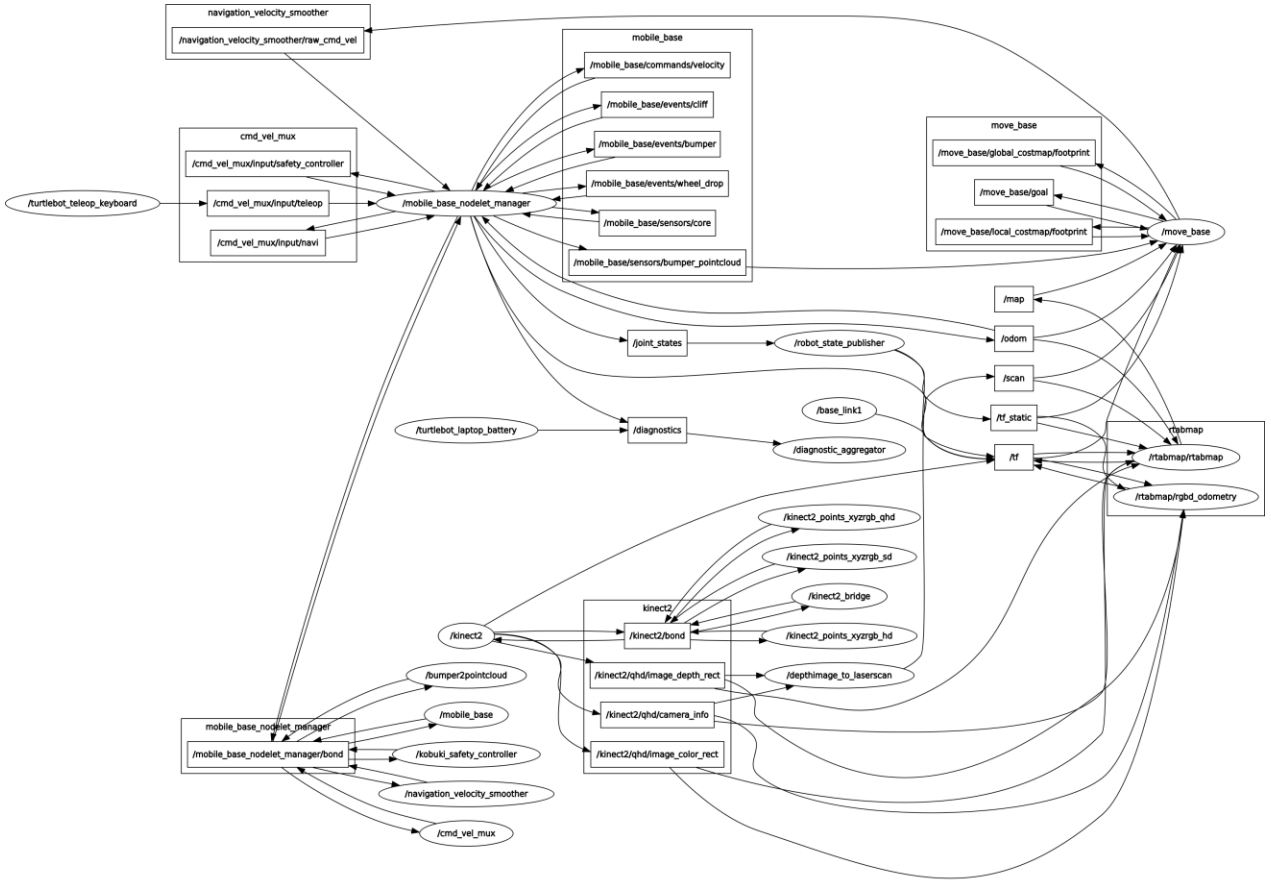
RQT - IS A GRAPHICAL USER INTERFACE FRAMEWORK



RQT-GRAPH







TAL TECH

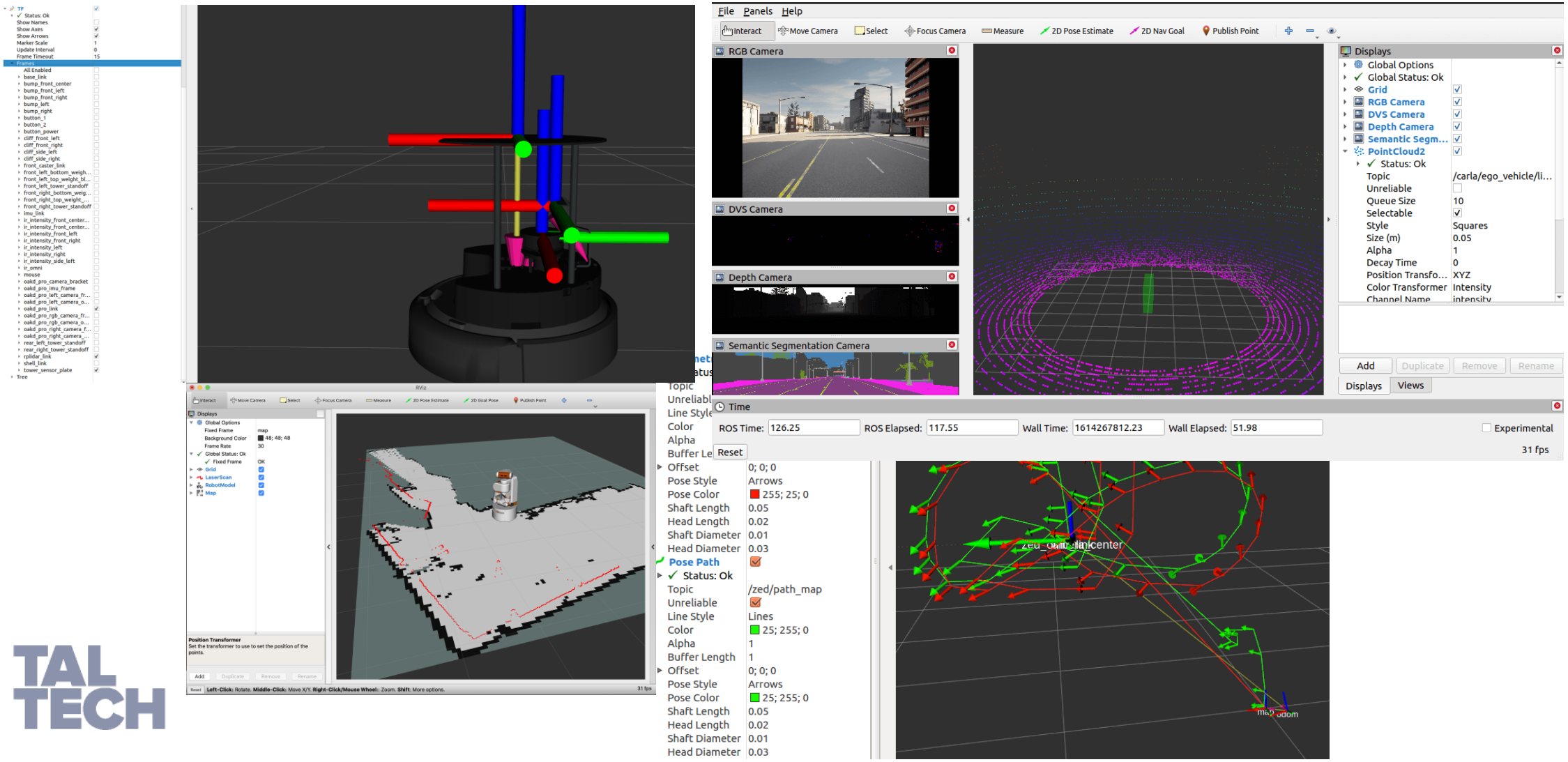
ROS2 BAG – COMMAND LINE TOOL FOR RECORDING DATA PUBLISHED IN YOUR TOPICS

Record data published on a topic so you can replay and examine it any time.

To record a bag:
ros2 bag record <topic_name>
The data will be accumulated in a new bag directory with a default name in the pattern of *rosbag2_year_month_day-hour_minute_second*. This directory will contain a *metadata.yaml* along with the bag file in the recorded format. *SQLite3* is the current *ROS 2* recording format default.
Following command:
ros2 bag record -o <bag_file_name> <topic1> <topic2>
Records a bag file with name *<bag_file_name> containing two topics*
e.g.,
ros2 bag record -o my_file_name /turtle1/cmd_vel /turtle1/pose
To get details of the recording :
ros2 bag info <bag_file_name>
To play the file back:
ros2 bag play subset

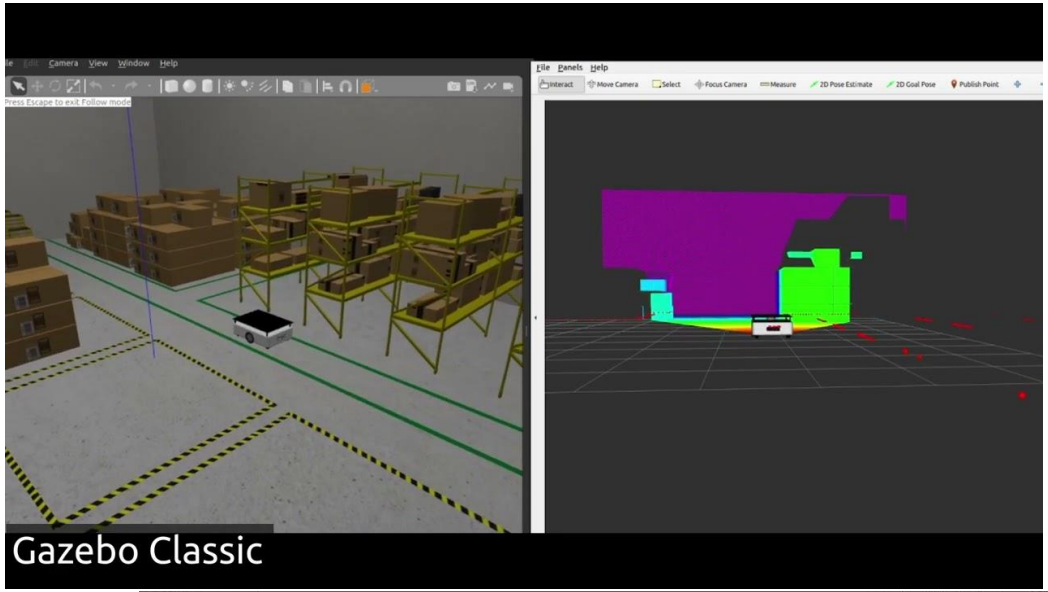
TAL TECH

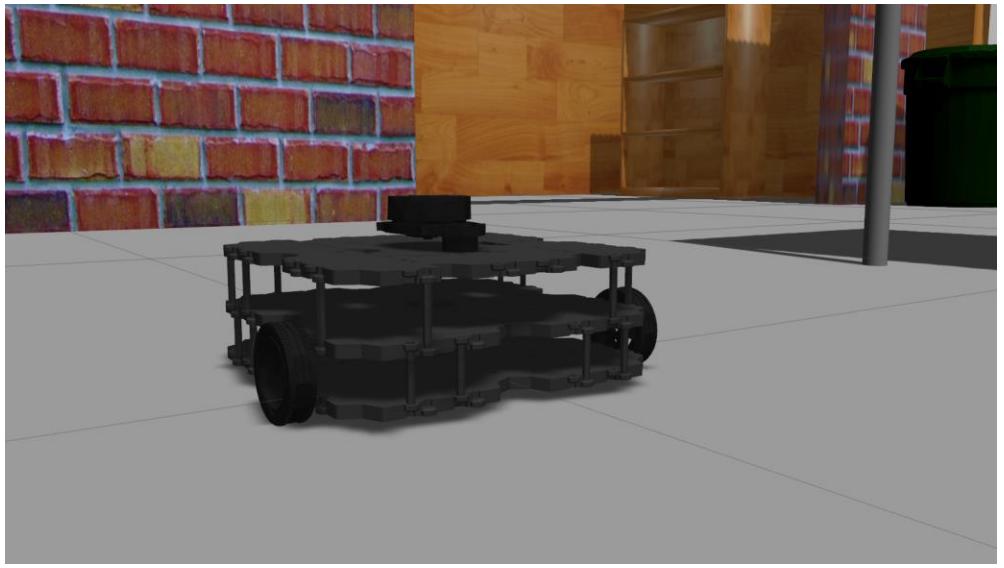
RVIZ - GRAPHICAL INTERFACE FOR USERS TO VIEW THEIR ROBOT, SENSOR DATA, MAPS, AND MORE

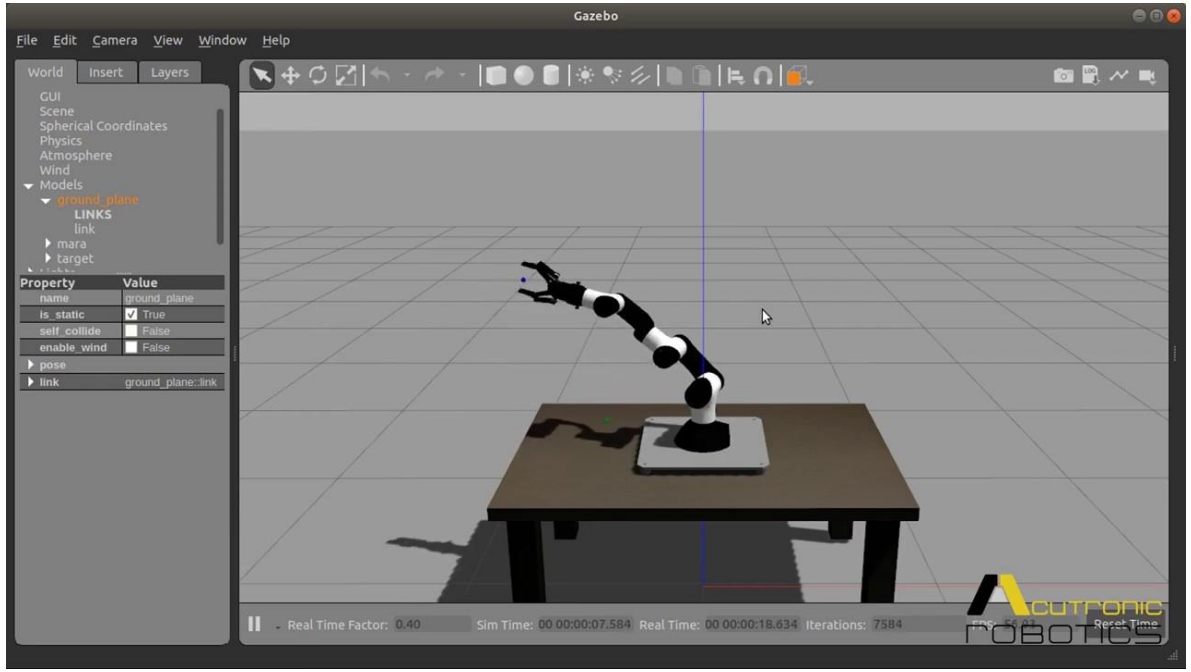


TAL TECH

GAZEBO – ROBOT SIMULATION

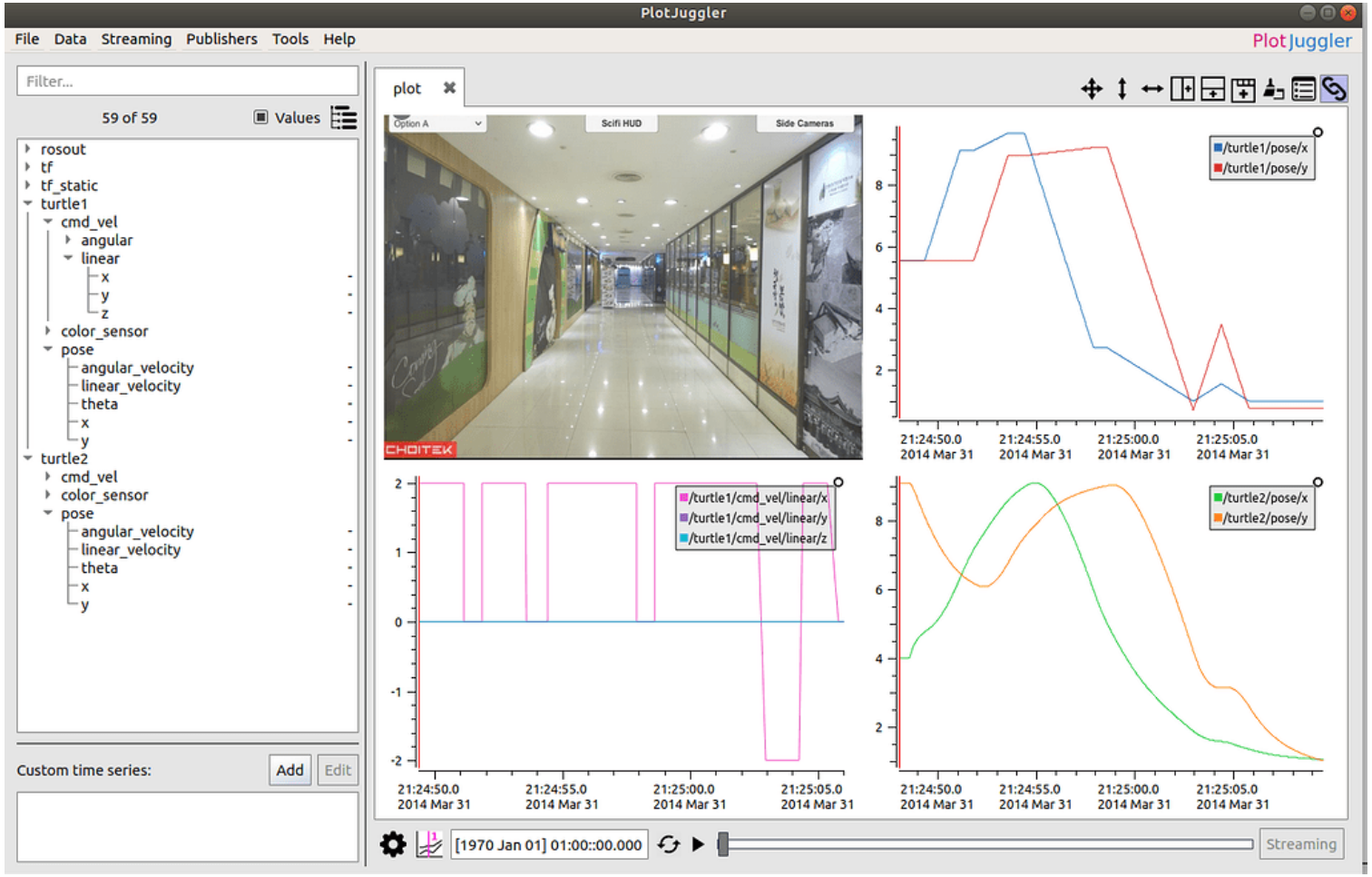


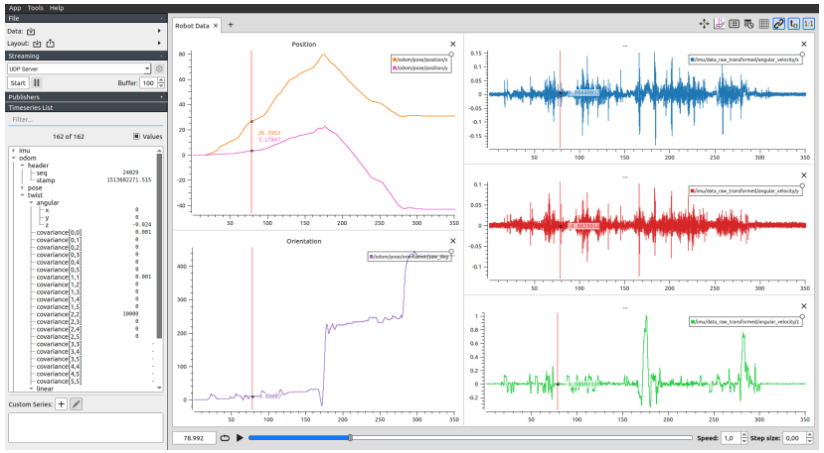




TAL TECH

PLOTJUGGLER





TAL TECH

ROS

WORKFLOW

PACKAGES

- ROS packages - ROS is widely used, and often useful capabilities exist somewhere in its community. Many stable resources exist as easily downloadable debian packages.
  sudo apt install ros-humble-<package_name_you_want_to_install>
  - https://index.ros.org/packages/#humble - list of packages
    - Note: If you need to use a package in the course, we will tell you!
- 3rd-party-packages - The fastest way to include 3rd-party python packages is to use their corresponding *rosdep* keys. These can be added to your *package.xml* file, which indicates to the build system that your package (and dependent packages) depend on those keys. In a new workspace, you can install all *rosdep* keys with:
  rosdep install -yr ./path/to/your/workspace
- Install from source – Packages can be installed from source code.
  - Find a git repository
  - Clone the repository
    cd ~/<your_ros2_workspace>/src
    git clone http://github.com/user/repository_name.git
  - Build your workspace
    cd ~/<your_ros2_workspace>
    colcon build
  - Source the workspace
    source install/setup.bash

TAL TECH

CREATING A PACKAGE

- Needa a ros2 workspace *<ros2_workspace>* with folder *src*. Make sure you are in the *src* folder before running the package creation command.
- Create an empty python package with required minimum contents
  ros2 pkg create --build-type ament_python <package_name>
- Return to the root of your workspace
- Build packages
  colcon build
- To build only the new package *<package_name>*
  colcon build --packages-select <package_name>
- Source the setup file
  source install/local_setup.bash

TAL TECH

COMBINE YOUR WORK WITH OTHER PACKAGES

- Modify package.xml
  - If the dependency is used during build time (headers, libraries).
    <build_depend>dep_1</build_depend>
    <build_depend>dep_2</build_depend>
    <build_depend>dep_n</build_depend>
  - If the dependency is used during run time (nodes in launch files).
    <exec_depend>dep_1</exec_depend>
    <exec_depend>dep_2</exec_depend>
    <exec_depend>dep_n</exec_depend>

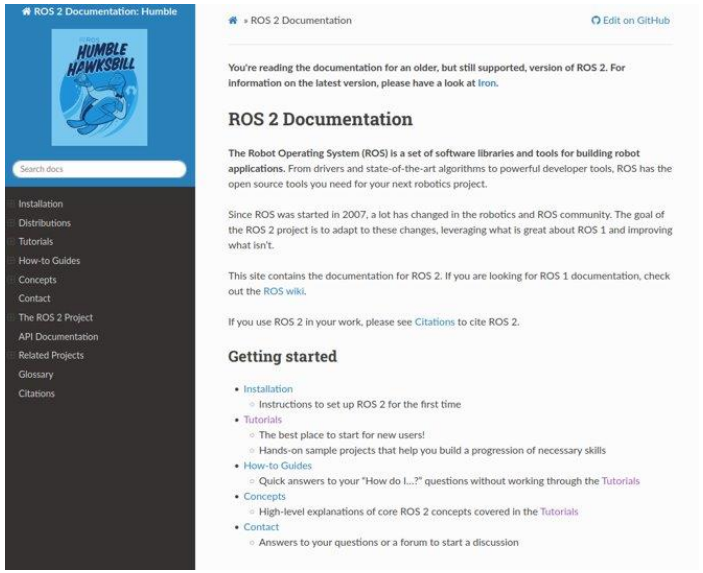
TAL TECH

ROS

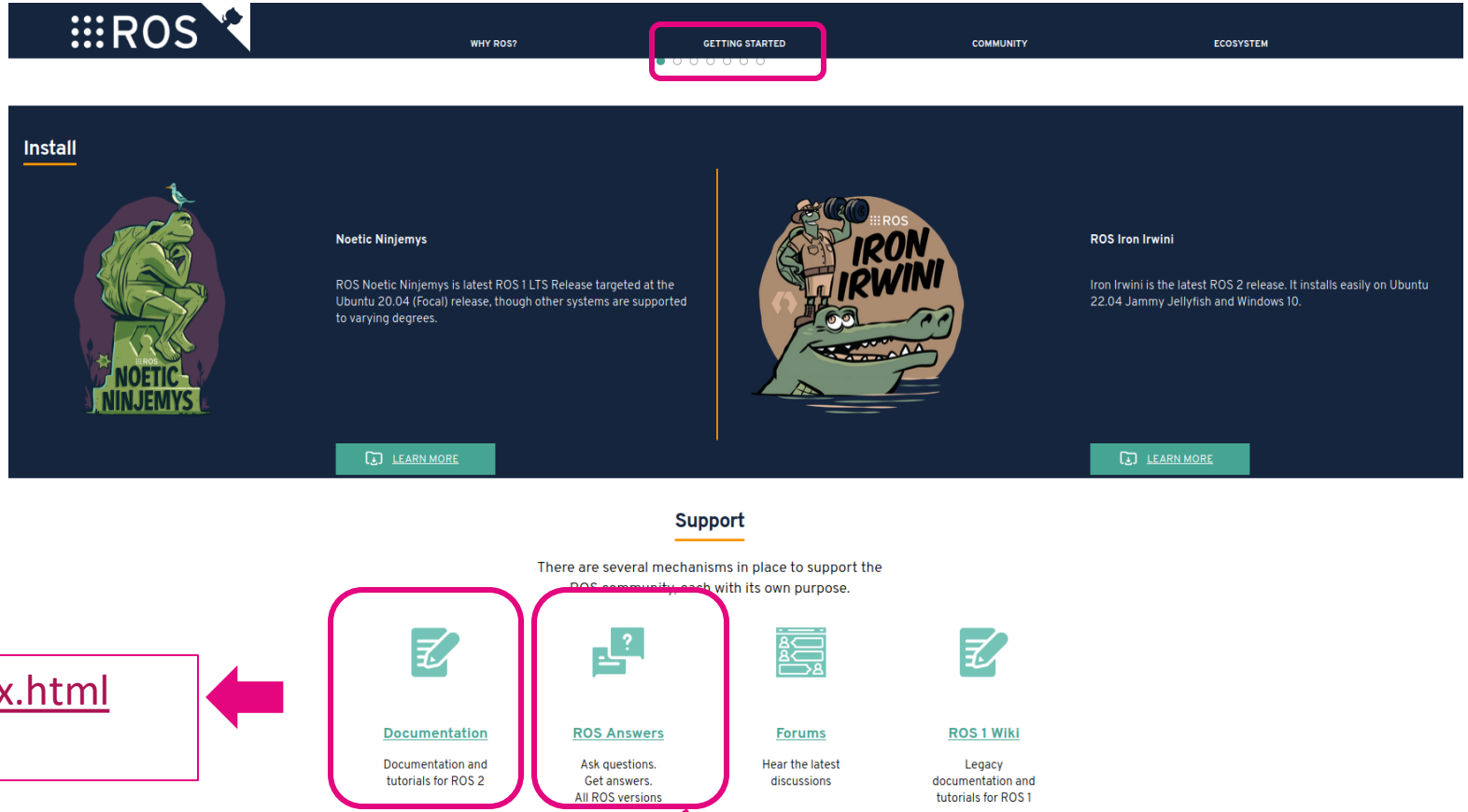
EVERYDAY

ROS.ORG

ROS Humble EOL: May 2027



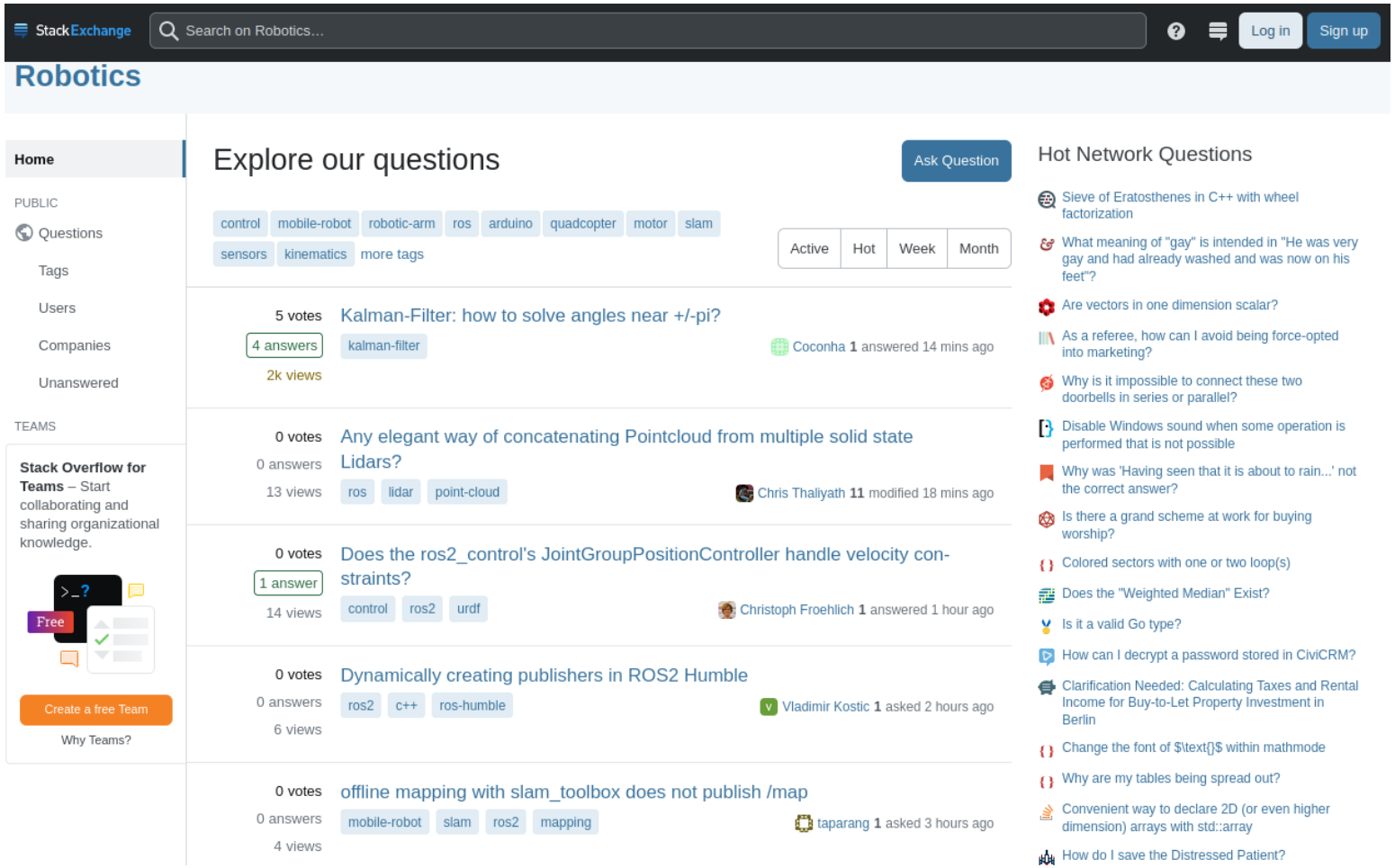
http://docs.ros.org/en/humble/index.html



answers.ros.org is deprecated as of August the 11th, 2023, but read only version is still available

TAL TECH

HTTPS://ROBOTICS.STACKEXCHANGE.COM/



TAL TECH