Grant agreement no.	**780785**
Project acronym	**OFERA**
Project full title	Open Framework for Embedded Robotic Applications
Dissemination level	Public
Date of Delivery	30/06/18
Deliverable Number	1.7
Deliverable Name	**Reference Scenarios and Technical System Requirements Definition**
AL / Task related	1.5
Author	(PIAP) Adam Dąbrowski
Contributors	(Bosch) Ingo Luetkebohle, Ralph Lange (eProsima) Jaime Martin Losa, Borja Outerelo Gamarra (ALR) Inigo Muguruza Goenaga, Victor Mayoral Vilches
Keywords:	Requirements, IoT, Robotics, ROS, microcontrollers, middleware
Abstract	The document presents reference scenarios and requirements for the micro-ROS platform. Starting with introduction to conceptual architecture of middleware akin to ROS2 and following through with description of use-cases, the document shows how requirements are drawn and structured.

Table of Contents

1. Introduction

1.1 Summary

In the document, we start with the introduction of basic micro-ROS concepts and use-cases which will be used to validate and demonstrate characteristics of the framework. There are four use-cases that will serve the purpose, covering a range of micro-ROS applications in robotics. We describe scenarios, platforms and all subsystem for each use-case.

Following the introduction of use-cases, we describe reference scenarios, which are a more generic and abstract way of thinking about a set of functionalities which are commonly required together. We proceed to explain how these more abstract scenarios match our use-cases. These reference scenarios are then analyzed with the use of Morphological Box. Since micro-ROS platform is to be modular and configurable, the analysis shows which aspects and parts of micro-ROS are required for each reference scenario. This helps to categorize requirements and understand which of them are crucial for specific scenarios.

Finally, the requirements are divided into non-technical and technical, which are further separated into functional and performance requirements. While functional requirements are best validated through use-cases, performance of the micro-ROS system in selected configurations is to be measured through the process of benchmarking. Functional requirements are grouped into several categories such as communication, network topology, information model and messages, and others.

1.2 Purpose of document

The document is of major importance to the process of design and implementation of the micro-ROS architecture. The role extends throughout the project and the document is to be updated if requirements change, as they often do in large undertakings, especially since our work goes beyond the state of art and the domain dynamics is high. We intend to keep this document a living one, synchronized with the documentation of architecture and our understanding of micro-ROS applications.

1.3 Partners involved

Short name	Full name	Contribution
PIAP	Industrial Institute for Automation and Measurements	Leading author
Bosch	Robert Bosch GmbH	requirements, morphological box and coverage
ALR	Acutronic Link Robotics AG	requirements
eProsima	Proyectos y Sistemas de Mantenimiento SL	requirements

1.4 List of acronyms

Acronym	Meaning
API	Application Programming Interface
CMSIS	Cortex Microcontroller Software Interface Standard
DDS-XRCE	DDS for Extremely Resource Constrained Environments
DDS	Data Distribution Service
IMU	Inertial Measurement Unit
MCU	Microcontroller Unit
QoS	Quality of Service
ROS	Robot Operating System
RTOS	Real Time Operating System

RTPS	Real Time Publish Subscribe
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle

2. Conceptual architecture

As introduced in the project proposal, micro-ROS software modules contain all the layers to allow a developer to create software using ROS 2 and micro-controller specific resources.

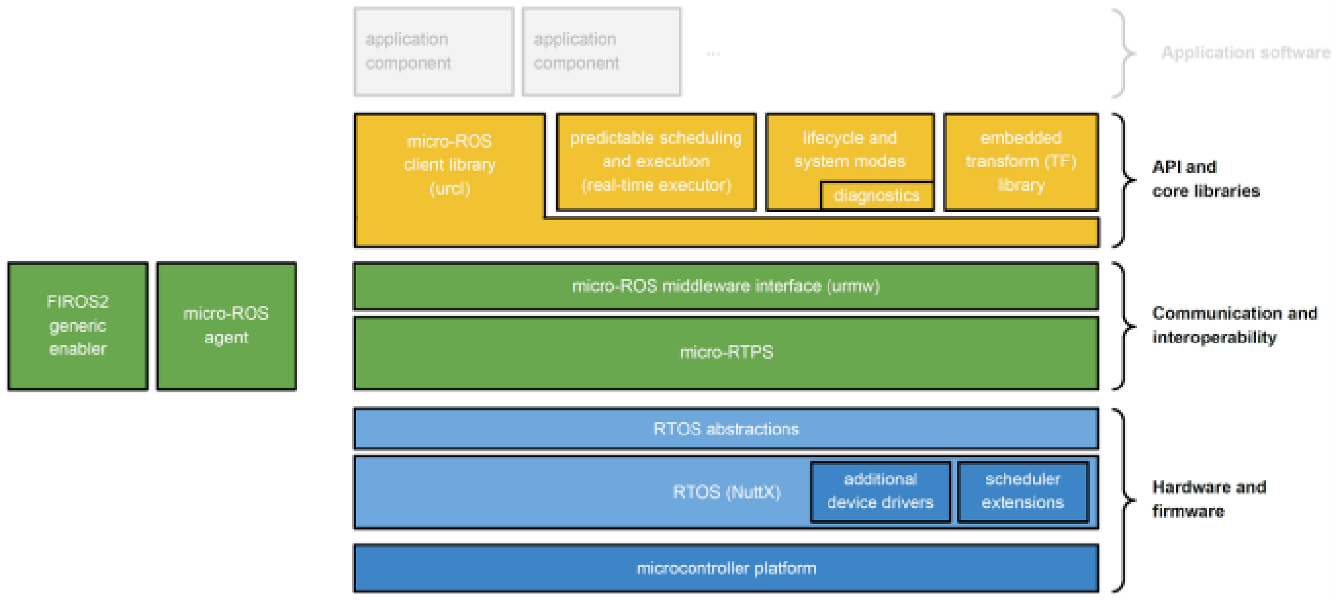

application component
application component
...
Application software
micro-ROS client library (urcl)
predictable scheduling and execution (real-time executor)
lifecycle and system modes
diagnostics
embedded transform (TF) library
API and core libraries
FIROS2 generic enabler
micro-ROS agent
micro-ROS middleware interface (urmw)
micro-RTPS
Communication and interoperability
RTOS abstractions
RTOS (NuttX)
additional device drivers
scheduler extensions
microcontroller platform
Hardware and firmware

Figure 1: An overview of conceptual architecture for micro-ROS

The lower layers comprise the microcontroller platform, the RTOS (where NuttX is the default choice for the scope of this project), and abstractions that will be defined (or used as--is, such as CMSIS) on top of the RTOS to ensure the compatibility of the upper layers with other RTOS.

The middle layers are organized corresponding to the ROS 2 architecture: Micro RTPS will be used as default DDS--XRCE implementation for micro--ROS, analogously to Fast RTPS being used as default DDS implementation for ROS 2. Just as ROS 2 defines a middleware interface (rmw) to hide subtle differences between the DDS implementation, we will define a micro--ROS middleware interface (urmw).

The next layer is the micro--ROS client library (urcl) analogously to the ROS client library (rcl) in ROS 2. The urcl constitutes the API for application software regarding communication, runtime configuration and execution management and will resemble the rcl's API for portability of application code and to ease the learning curve for developers. However, we will separate the APIs and mechanisms for component (node) lifecycle and system modes (with its diagnostics submodule) as well as for predictable scheduling and execution from the urcl into two core libraries as far as possible.

This shall ensure that parts of micro--ROS, in particular the communication stack, can be also integrated with other frameworks and that developers can save resources by picking only those features they really need. Moreover, the APIs for lifecycle, predictable scheduling and execution, and system modes will be much richer than the corresponding APIs of the ROS 2 rcl due to the advanced concepts specific to micro-ROS. The embedded transform (TF) library for the management of the robot transform graph forms the third core library.

3. Use-cases

Since the micro-ROS aims to create a generic framework for robotics in the spirit of ROS, possible applications are many and varied. We have selected use-cases to supply several good examples of how micro-ROS can be deployed in robotic systems, and to provide validation for the framework's components. Another consideration was the resources already available to each partner, especially the robotic platforms central to these scenarios.

The following use cases are described in this chapter:

- **Autopilot Drone**, concerned with data gathering from distributed sensors using micro-ROS.
- **Modular Arm**, where micro-ROS device is attached to extend functionality in a modular way.
- **Domestic Outdoor Robots**, in which an autonomous lawn-mower employs micro-ROS components in the context of commercial product.
- **Smart Warehouse**, where a micro-ROS bridge is used on a platform that travels through a spacious area, communicating with dynamically discovered micro-ROS devices.

3.1 Use-case: Autopilot Drone (AD)

Overview

Commercial and industrial UAVs use autopilots, which are typically powered by microcontrollers. When the drone application requires more on-board processing power, it is usually solved by adding a microprocessor-based companion computer. Good examples of such application are real-time computer vision or large-scale data gathering from external sources. It is important to integrate their capabilities without interfering with the UAV autopilot main task, the flight control.

A wide variety of sensing devices can be integrated into such setup, but the power efficiency and cost of the platform with all the components is a major factor. Adding new computing and sensing capabilities with minimal increase in battery consumption is especially important for UAVs. This is where micro-ROS is very useful, allowing to easily connect inexpensive and power-effective microcontroller-based external sensors to the autopilot.

Additional benefits come from seamless integration with existing ROS 2 packages and systems that can interact with drone thanks to micro-ROS.

This use-case is based on the data gathering and processing scenario commonly used in the commercial and industrial UAV markets. This scenario appears in different applications such as crop management (Precision agriculture), environment measure and observation, search and rescue and similar applications.

In this case, we use the environment measure and observation scenario, providing measures taken on the fly by the UAV and measures taken from an external sensor placed on a UAV reachable place. We will focus on environmental metrics like sound, humidity, pressure and/or temperature. External sensors measures will be used on the central computer, in this case, they will be incorporated into a historical of the measures in the gathering spot.

Apart from environmental measures, Autopilot will use data gathered from sensors placed on board for enhancing flight capabilities. These on-board sensors can improve flight control capabilities and precision. Adding external inertial measurement unit, IMU, or as in this case, using an external height/altitude sensor, e.g. barometer, rangefinder, can improve the flight control capabilities of the Autopilot.

Purpose

This use-case demonstrates how micro-ROS can be applied to a flying drone, providing value in the context of a larger robotic application. In particular, the Drone Autopilot use-case is focused on applying micro-ROS to efficiently extend UAV sensing capabilities and to integrate the autopilot system with ROS 2 components. Several different sensors will be used for better validation and added variety.

Scenario environment

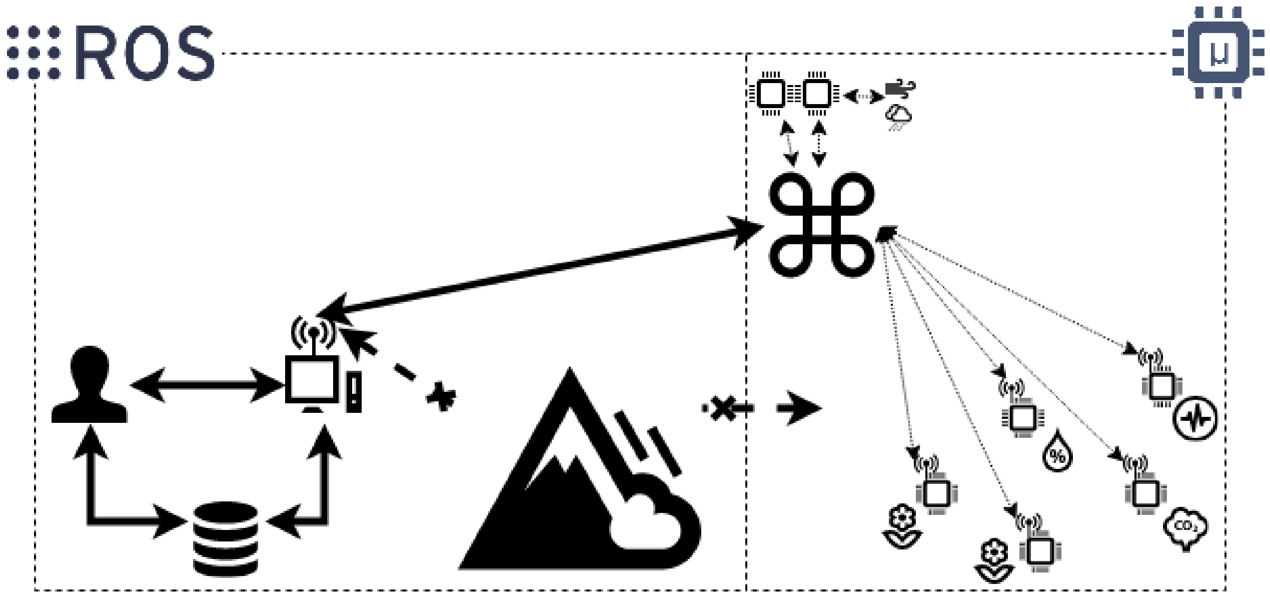


Figure 2: Overview of data gathering environment

The information gathering is done as part of a distributed robotic system where the UAV flight controller and the linked sensors are part of a bigger ROS 2 enabled system. The whole operation is driven by an external ROS 2 enabled computer. The computer is in charge of sending off-board control commands to the autopilot. Autopilot off-bard control is a completely different problem, so we will keep on the basic off-board control making good use of PX4 Autopilot off-board control use case. PX4 Autopilot off-board control is based on MavLink protocol.

Companion computer will pass the off-board control commands received from the control computer to the Autopilot. Apart off-board commands, Autopilot will be receiving altitude measures provided by on-board sensors using micro-ROS as communication enabler. We will use the manual control in those cases where off-board control would be impossible to use.

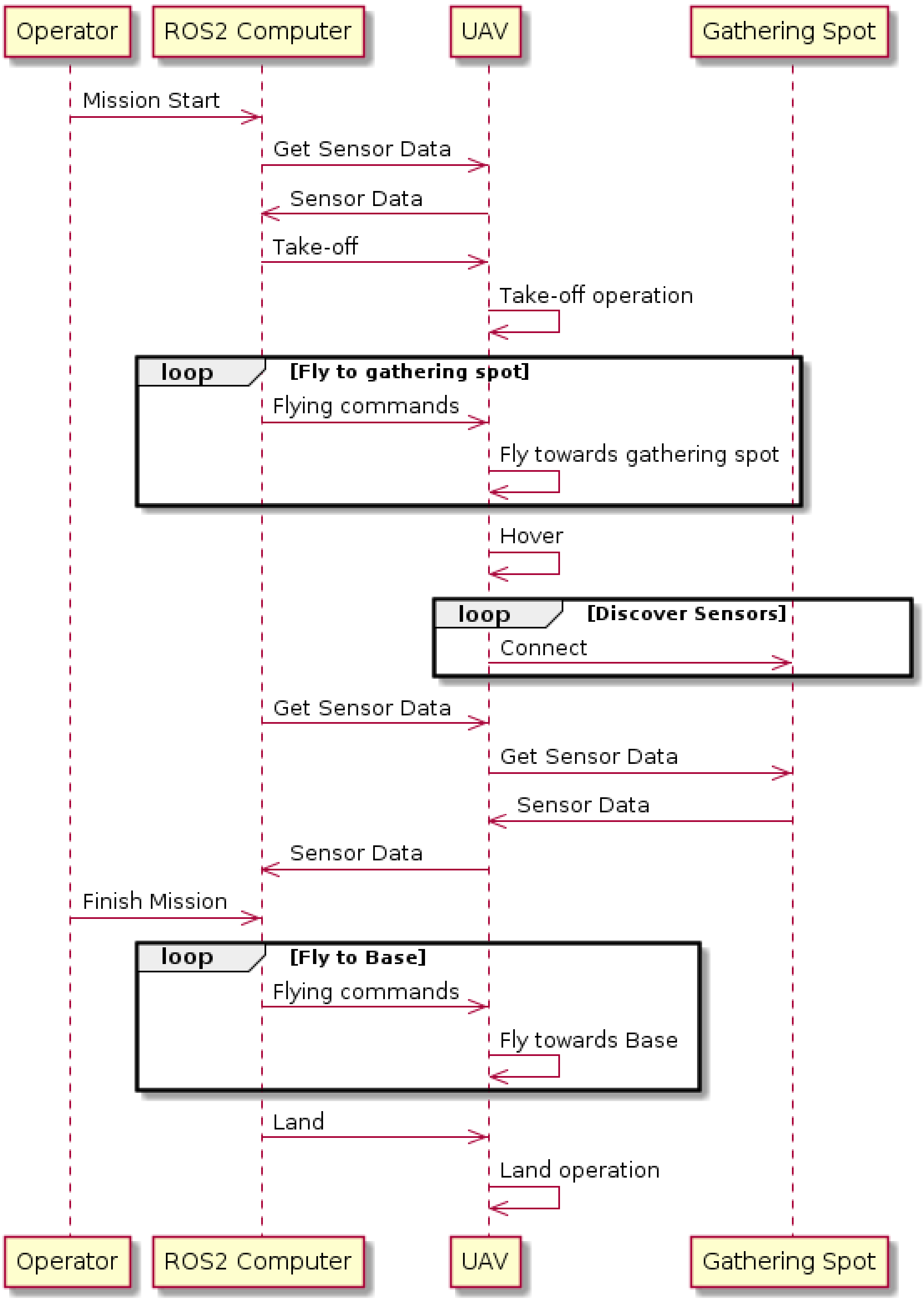
Autopilot commands describe how the UAV should approach a distant area where environmental sensors are deployed. In order to ease the showcase of the use-case, sensors deployment area and drone starting position will not require to be far away from each other. The UAV, once in the range of the wireless signal of the sensors, will notice the presence of new sensors and will make their data available to the rest of the system.

Having the UAV hovering on the gathering spot, the whole system has access to information from all the sensors, on-board and off-board, data of the autopilot and data coming from ROS2 modules on the base station computer. Autopilot hovering over the gathering spot is enabled thanks to altitude measures provided from an on-board sensor. Once all the data is gathered, the ground computer orders the UAV to go back to the base and land gently.

The scenario flow

A scenario utilizing off-board control proceeds as follows:

1. ROS 2 Computer operator gathers data from UAV on-board environmental sensors.
2. ROS 2 Computer operator issues commands for the autopilot requesting to take-off.
3. UAV receives command through micro-ROS and takes-off.
4. ROS 2 Computer operator issues commands to approach gathering spot.
5. UAV receives command and flies towards gathering spot.
6. UAV hovers on the gathering spot using altitude sensor information.
7. UAV switches companion computer on and dynamically detects the presence of off-board sensor and connects to it.
8. ROS 2 Computer operator gathers data from UAV on-board sensor.
9. ROS 2 Computer operator gathers data from UAV off-board sensor.
10. ROS 2 Computer operator issues command to return to the base.
11. UAV receives command and returns to the base.
12. ROS 2 Computer operator issues command to land.
13. UAV receives command and lands.


Operator
ROS2 Computer
UAV
Gathering Spot
Mission Start
Get Sensor Data
Sensor Data
Take-off
Take-off operation
loop
[Fly to gathering spot]
Flying commands
Fly towards gathering spot
Hover
loop
[Discover Sensors]
Connect
Get Sensor Data
Get Sensor Data
Sensor Data
Sensor Data
Finish Mission
loop
[Fly to Base]
Flying commands
Fly towards Base
Land
Land operation
Operator
ROS2 Computer
UAV
Gathering Spot

Platform and components

Platform

A custom UAV will be deployed in this use-case. It will utilize Pixhawk4, a px4 enabled autopilot, mounted on a custom quadcopter frame. The autopilot combined with a GPS module allows this basic setup to fly autonomously outdoors. Indoors, the UAV needs to be flown manually.

The platform will include a rangefinder sensor, a companion computer such as Raspberry Pi3 or Odroid UX4, and a set of environmental sensors to supply on-board measures.

Separately from the UAV, a regular computer and a set of environmental sensors will be used to complete the scenario.

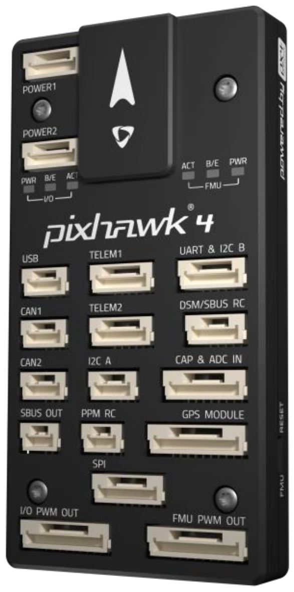


Figure 3: A Pixhawk4 autopilot for the Autopilot Drone use-case

Figure 4: Autopilot Drone quadrocopter

Components

Component	Role and description
ROS 2 computer	a general purpose computer. running a ROS2 node collecting all data gathered and another ROS2 node acting as the pilot of the UAV. The data gathered can be forwarded to other systems, cloud-based or specific data storage systems. It can be operated by the user (flight operator) to send commands or manually handle the UAV if required.
Companion computer	a reduced size computer which allows users to develop complex applications and mount it on relatively small devices. Common used computers examples: Nvidia TX series, Intel Aero, Intel Edison, Odroid XU4, Raspberry Pi3 or Qualcomm Snapdragon Flight. In this case, we use Odroid XU4 or Raspberry Pi3 due to its low costs. This computer provides the flying package with wireless communications as well as a way to attach new microcontroller-based systems, environmental sensors in our use-case as off-board sensors. In case of manual control, this companion computer could be off, saving battery, while drone approaches the gathering spot, where it can be switched on again to provide wireless access to ground sensors
Autopilot	a vital microcontroller component of the UAV, controlling the whole flying process, from processing external input to stabilizing the flying package. In this case, we use a Px4 autopilot running on top of the NuttX RTOS. This autopilot communicates with other micro-ROS/ROS2 parts using micro-ROS. Autopilot uses altitude measures from an on-board sensor as a flight enhancer. This on-board altitude sensor has a direct link with the autopilot for those cases where companion computer is off saving package battery. The base platform is Pixhawk 4 flashed with PX4 firmware

On-board sensors	a group of sensors including two distinct types: environmental and distance sensors. Environmental sensors take measures following a short sleep cycle. Measures include temperature, pressure, and humidity. Distance sensor measures the distance to the ground to maintain drone altitude allowing it to hover over gathering spot. The sensors take measures and expose them to the rest of the micro-ROS modules using a serial connection with the UAV companion computer and another serial link with the autopilot microcontroller. The proposed distance sensor is VL53L0x ToF, while environmental sensors are Bosch BME280 (humidity, pressure and temperature) and MOD-LTR501ALS (luminosity and proximity). All sensors use Olimex LTD STM32-E407
Off-board sensors	These environmental sensors take measures and communicate using micro-ROS wirelessly. The same hardware is used as for on-board sensors.

Architecture overview

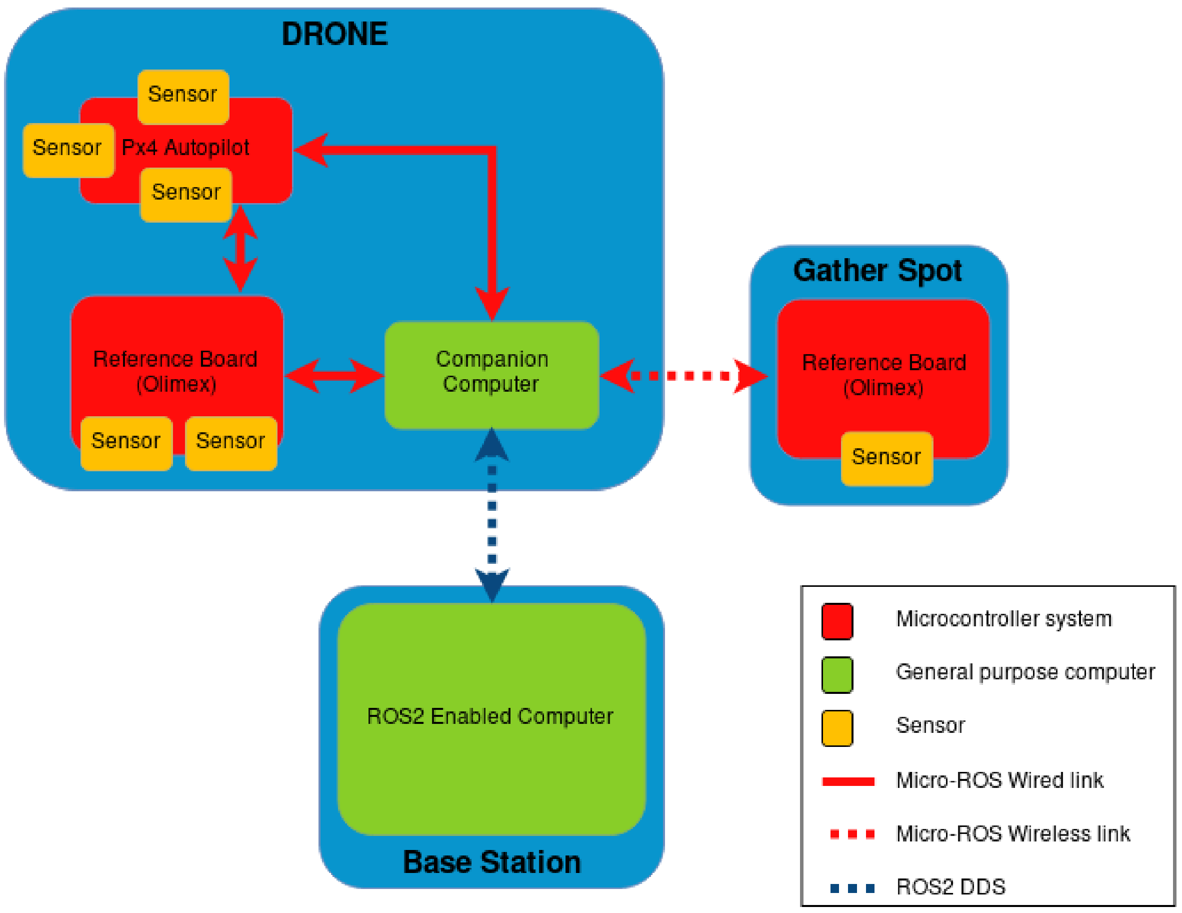


Figure 5: Architecture overview for Autopilot Drone use-case

Autopilot, companion computer and on-board sensor communicate with each other using a serial link (UART), while the off-board sensors use a wireless link (IEEE 802.15.4, ZigBee or

6LoWPAN) to communicate with the companion computer. The ROS 2 computer communicates with the drone through Wi-Fi.

Work Package dependencies

Work Package	Deliverable	Technology	Description
WP2	D2.1	Reference hardware	Use of supported hardware on the Micro-ROS enabled sensors
WP2	D2.2b	RTOS release	Use RTOS on sensors
WP2	D2.3b	Software Micro-ROS bridge	Provide communications
WP3	D3.1, D3.2, D3.3a	Micro-ROS middleware	Communications dependency
WP4	D4.1,4.2,4.3	Micro-ROS core libraries	Use on sensors

3.2 Use-case: Modular Arm (MA)

Overview

The use-case will be realized by Acutronic Link Robotics (ALR) and will demonstrate how one of their modular robot platforms (refer to the corresponding section below) can be easily extended with inexpensive microcontroller-based robot sensors that interoperate with the rest of the modules seamlessly. In particular, this use-case will demonstrate how using micro-ROS, traditional robot components can be replaced by microcontroller-based ones that display several advantages such as less power consumption, faster boot times and compact dimensions among others. All of this will be displayed in real-time in a portable user interface that will serve as a demonstrator for potential users of the technology and interested parties.

This use-case will focus on modular robots and how these adaptable machines can be extended with inexpensive and resource constrained plug-and-play sensors that interoperate through the use of micro-ROS. This work will bring together Acutronic Robotics' H--ROS® infrastructure and micro--ROS by integrating ROS-enabled sensors (using micro--ROS-) with existing modular robots (built out of H--ROS®). micro--ROS software and tools will be applied to *build inexpensive robot sensors* that will then be used to extend these modular robots

Purpose

The main objective of this use-case is to demonstrate that micro-ROS powered microcontrollers present a valuable opportunity for building interoperable robot sensors for modular robots. In particular, modular robot arms.

This use-case aims to test, demonstrate and validate:

- The integration of resource constrained sensors in a modular robot where all components speak ROS natively (in other words, all components are distributed and heterogeneous), specifically, the capability of micro-ROS devices interoperating through wired or wireless mediums.
- Plug-and-play capabilities of micro-ROS enabled sensors.
- How ROS-enabled sensor performs depending on whether it is implemented with 1) micro-ROS and 2) H-ROS® from the perspective of: a) power consumption, b) boot time, c) size and dimensions, d) real-time responses and e) communication latencies. This is an important aspect of the use-case in terms of driving the performance requirements.
- How modular robots can acquire the capability of extending their hardware configuration seamlessly with tiny and inexpensive sensors that yet, preserve relevant aspects for modularity such as interoperability, reusability and reconfigurability.

Scenario environment

The use-case will contain a modular robot arm built using H-ROS robot parts where new capability is desired. For that purpose, a micro-ROS based sensor will be attached, using a micro-ROS to H-ROS bridge for making compatible the incoming device with H-ROS technology. This sensor will provide robot access to new data that will allow gaining a better perception of the environment and allowing to enhance its behavior.

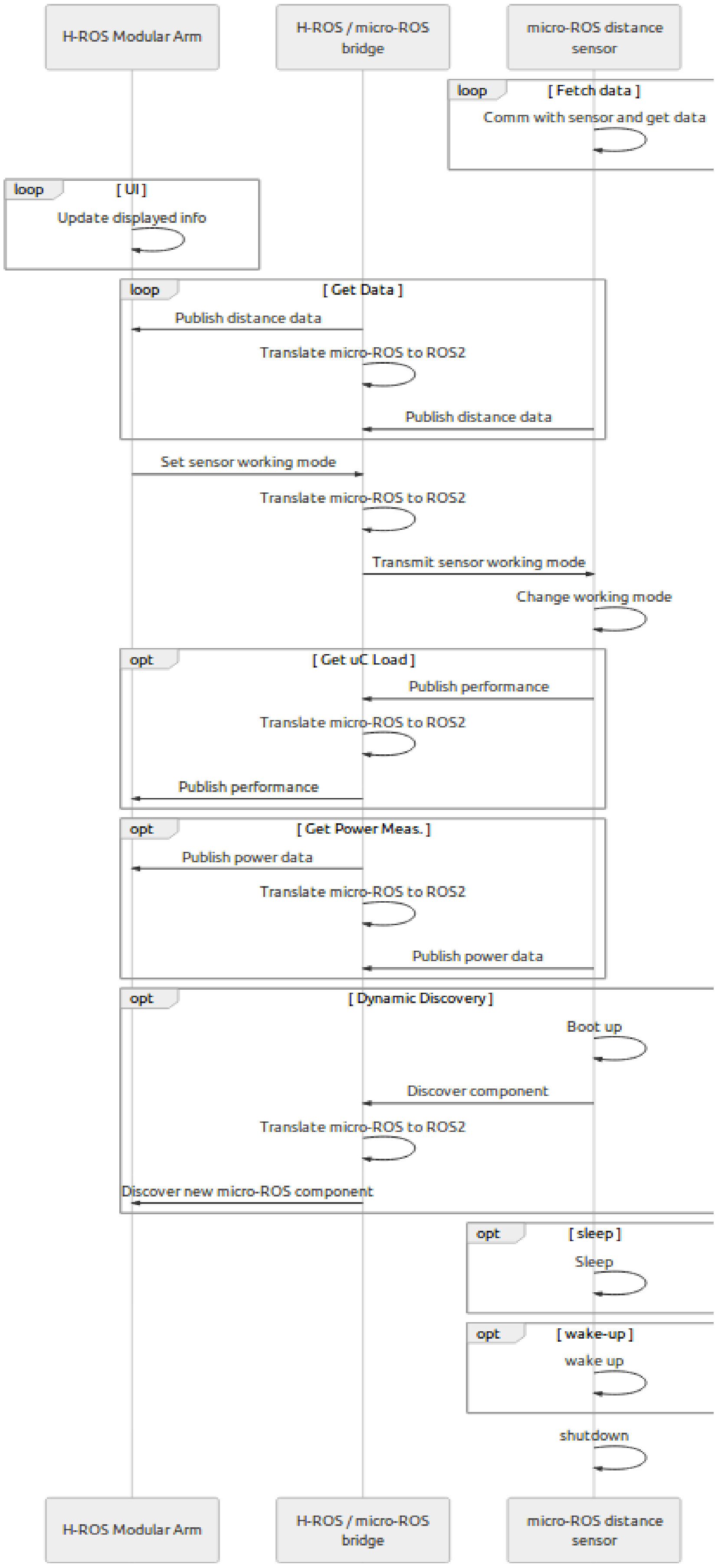
The discovery of the micro-ROS device could be static or dynamic, not having any working restrictions but the limitations mechanical attachment could lead to.

In order to make this use-case user friendly, the use of a visualization tool is expected, where several technical aspects of the robot will be displayed, such us CPU load, robot 3D model or robot topology.

Scenario flow

At this scenario, several test measurements will be performed, with the aim of comparing micro-ROS device to H-ROS based device in terms of power requirement, booting time, communication latency, real-time response or size/dimension.

The main flow focuses on the user display being updated with data gathered from connected sensors.



H-ROS Modular Arm
H-ROS / micro-ROS bridge
micro-ROS distance sensor
loop [Fetch data]
Comm with sensor and get data
loop [UI]
Update displayed info
loop [Get Data]
Publish distance data
Translate micro-ROS to ROS2
Publish distance data
Set sensor working mode
Translate micro-ROS to ROS2
Transmit sensor working mode
Change working mode
opt [Get uC Load]
Publish performance
Translate micro-ROS to ROS2
Publish performance
opt [Get Power Meas.]
Publish power data
Translate micro-ROS to ROS2
Publish power data
opt [Dynamic Discovery]
Boot up
Discover component
Translate micro-ROS to ROS2
Discover new micro-ROS component
opt [sleep]
Sleep
opt [wake-up]
wake up
shutdown
H-ROS Modular Arm
H-ROS / micro-ROS bridge
micro-ROS distance sensor

Platform and Components

Base platform

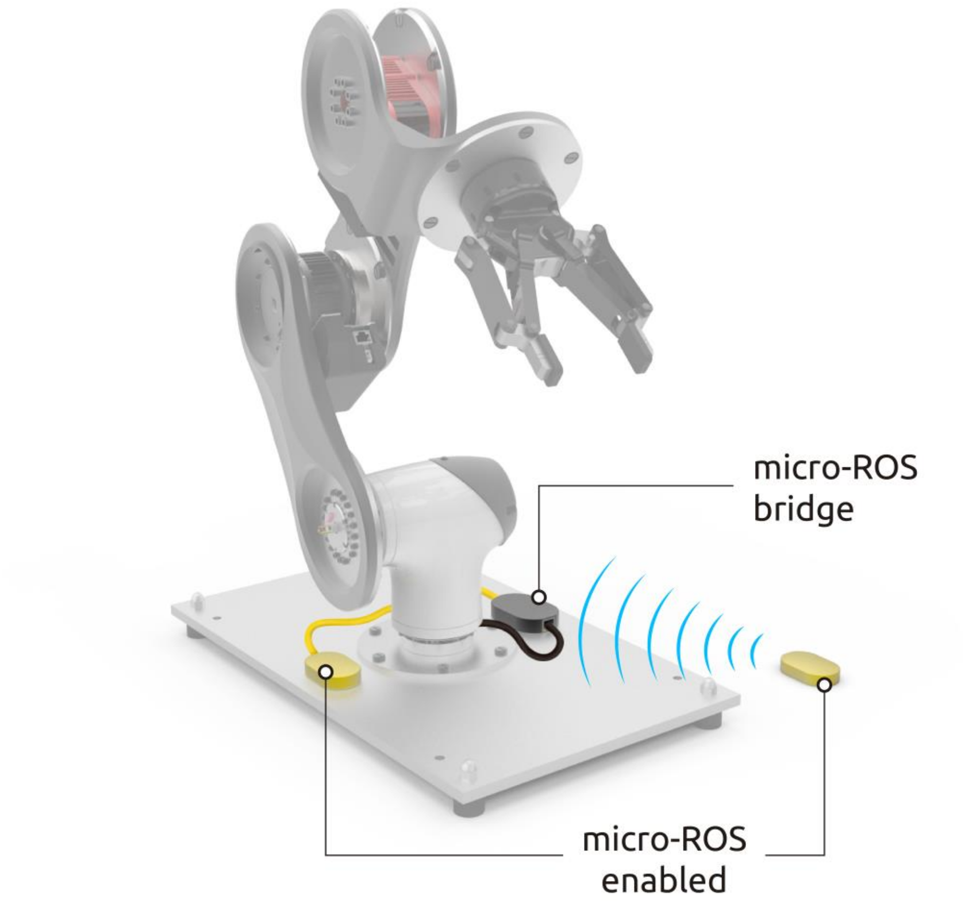


Figure 6: ALR modular robot arm

The base platform will be one of the Acutronic Link Robotic's modular robot that is built out of H-ROS technology. This robots supports by default ROS2 and thanks to the addition of micro-ROS sensor will enhance its functionality.

Additional to the modular robot arm, native H-ROS devices and micro-ROS devices will be used:

Hardware components

1. **micro-ROS enabled sensor**: Based on project's RHDPs, STMicroelectronics STM32F4 or STM32L1 microcontrollers, and the distance sensor. The firmware run at the MCU

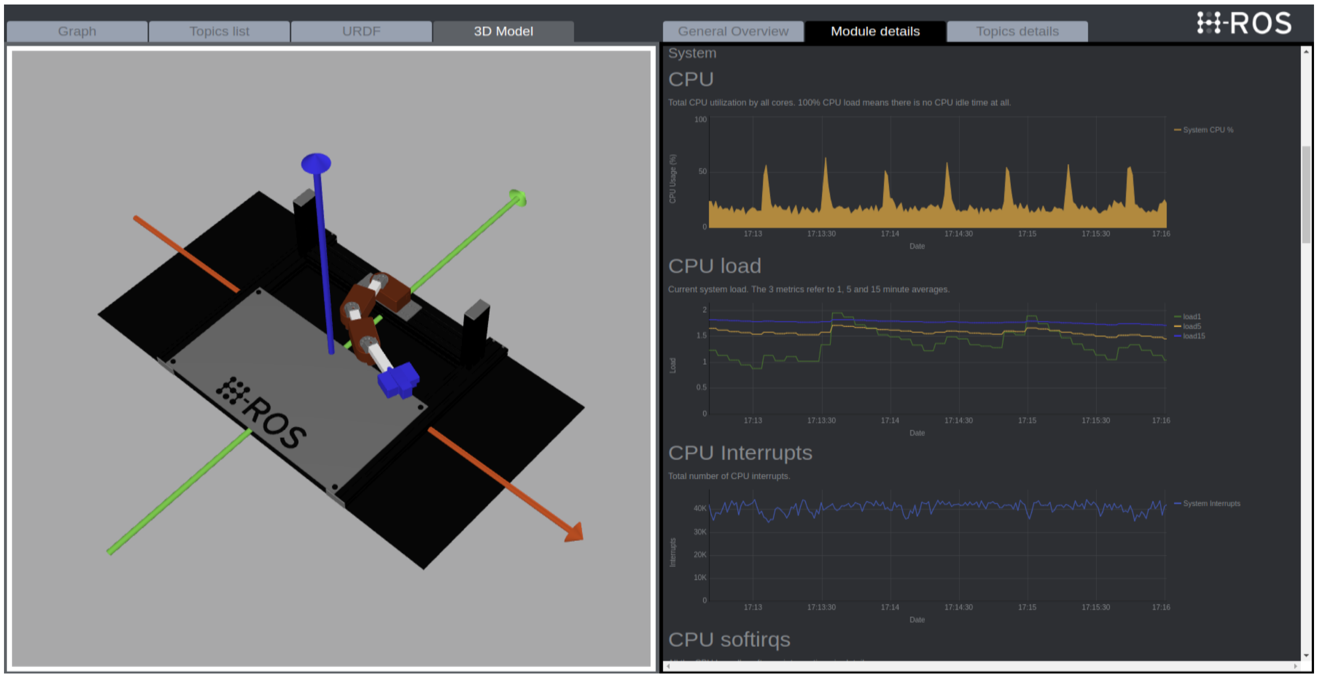
will contain micro-ROS code, which will fetch sensor data, process it and send it using to micro-ROS Agent.

2. **ROS enabled sensor**: ROS-enabled distance sensor created with H-ROS® technologies. This sensor represents regular H-ROS® sensor under which uses ROS2. It is going to be used to collect its performance data in order to compare it to micro-ROS powered sensor.

3. **Hardware micro-ROS bridge**: This bridge which will connect together the micro-ROS and ROS communication infrastructures, allowing micro-ROS devices to be H-ROS infrastructure participants.

Software components

1. **Hardware driver**: Hardware drivers for the sensor(s) selected (*proprietary*).
2. **micro-ROS framework**, including reference Real Time Operating System (RTOS), communication middleware, diagnostics and monitoring, real-time scheduling modules, lifecycle, etc.
3. **Visualization tool**: A visualization tool for modular robots (*proprietary software*).



Total CPU utilization by all cores. 100% CPU load means there is no CPU idle time at all.

Current system load. The 3 metrics refer to 1, 5 and 15 minute averages.

Total number of CPU interrupts.

Figure 7: A visualization tool for H-ROS modular robots

Work Package dependencies

Work Package	Deliverable	Technology	Description

WP2	D2.1 (month 6)	Reference hardware	use of supported hardware for micro-ROS enabled sensor to be built
WP2	D2.2b (month 12, 18)	RTOS release	use of RTOS for sensors
WP2	D2.3a (month 24)	Hardware bridge	Necessary to simplify the interoperability of micro-ROS enabled components, includes software bridge
WP2	D2.3b (month 18)	Software micro-ROS bridge	Used to make the first tests of interoperability with robots
WP4	D4.1,4.2,4.3 (variable, around month 24)	micro-ROS core libraries	Used to program the micro-ROS enabled sensor
WP4	D4.5a (month 26)	report on information model and interoperability	Used to facilitate the same language among H-ROS® and micro-ROS enabled devices

3.3 Use-case: Domestic Outdoor Robots (DOR)

Overview

Domestic outdoor robots currently in the market most notably include robotic lawn-mowers, with a number of other applications (e.g., weeding) being close to market.

These robots face a challenging sensing problem in a 3D environment as well as challenging control issues such as driving accurately on wet lawn. Moreover, they have to fulfill strict safety requirements, firstly, because of dangerous cutting implements, and secondly, because of the risks posed by driving outside of the designed operating area. In particular, entering public roadways could cause accidents. Last, but not least, cost is always a severely limiting factor for the design of such robots, since they are consumer products.

In this, the platform software is a large contributor to the cost of the overall product. Much of this software is basic functionality, e.g. serial drivers, message communication, state monitoring, etc. Such software is an ideal candidate for the use of open source, with a high potential for driving down costs over a wide range of products.

Purpose

The consumer robotics market is an important target market, and this use-case will demonstrate that micro-ROS is applicable for this market, both technically and economically. Technical feasibility goes beyond realizing the pure functionality, we must also demonstrate that all the edge cases of a product are handled. To be feasible economically, the hardware cost of the resulting solution must be comparable to existing products.

We can demonstrate both of these aspects by comparing to the existing domestic outdoor robot - a robotic lawn-mower - of partner Bosch.

Figure 8: Bosch Indego 400 Connect - Copyright Robert Bosch Power Tools GmbH

Specifically, this use-case will

- evaluate the effectiveness and functioning of the micro-ROS stack, including the core libraries for life cycle management and system modes, diagnostics, transform, and the real-time executor
- validate the interoperability with standard ROS 2.0
- evaluate the portability of software assets from ROS 2 to micro-ROS, to prove the reduction of efforts between advance development and series development
- validate the interoperability with standard ROS 2.0
- evaluate the portability of software assets from ROS 2 to micro-ROS, to prove the reduction of efforts between advance development and series development

We expect that we will be able to demonstrate a significant cost saving potential for future products based on the data from this use-case.

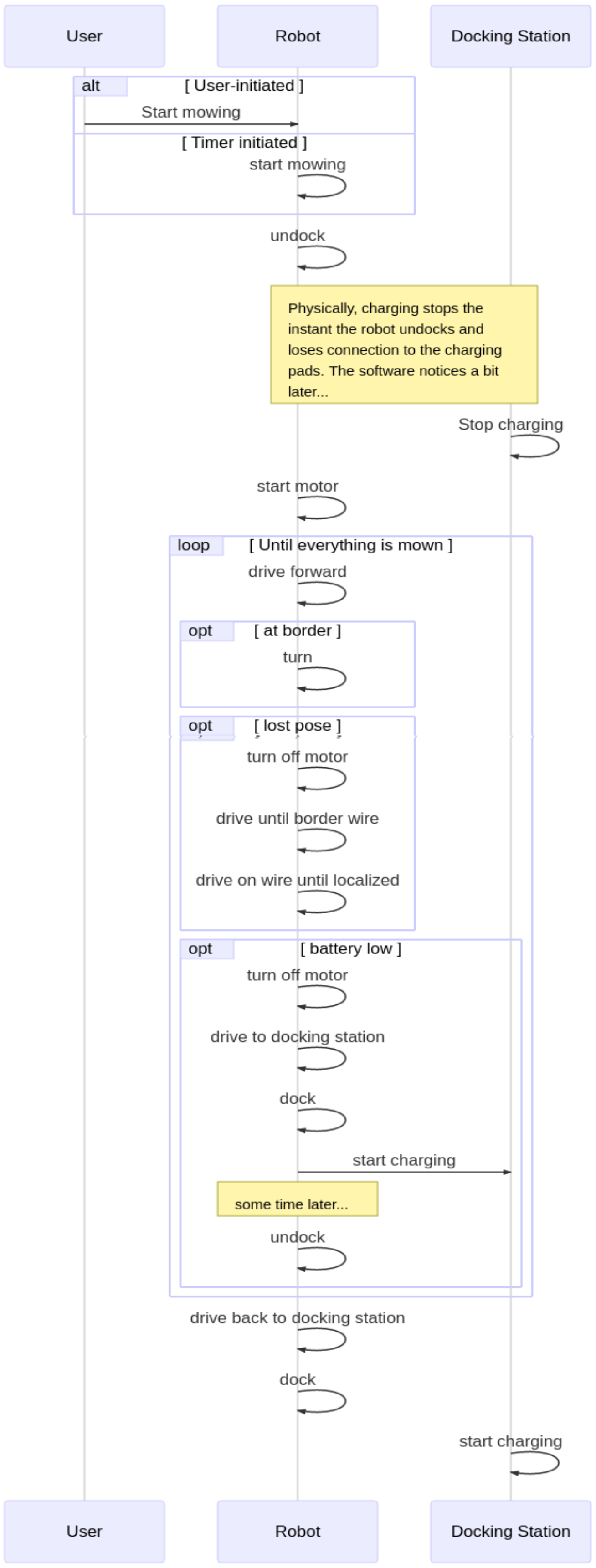
Scenario Environment

We use autonomous lawn-mowing as an example of domestic outdoor robots. The typical environment for this is a lawn outfitted with boundary and obstacle guidance. Current products recognize the border of their cutting area through a fixed boundary wire. It is placed directly on the earth and will be rapidly overgrown by grass, so it is usually invisible. To be fully autonomous, a docking station with charging capability is usually provided.

For testing purposes, and since cutting performance is not of concern in this project, an indoor arena can also be used. At Bosch Corporate Research in Renningen, a test area outfitted with optical tracking systems is commonly used to measure navigation performance. The floor of this area is a wooden platform with flat and slightly angled areas, overlaid with plastic grass, and surrounded by a perimeter wire.

Scenario flow

The mower starts at the docking station, then drives over the lawn to cut it, until either the grass is fully cut or the mower's battery is empty. In both cases, it will return to the docking station. When not fully done, it will recommence mowing after charging. Since current mowers leave the cut grass on the lawn, emptying the mower is not necessary.


User
Robot
Docking Station
alt
[User-initiated]
Start mowing
[Timer initiated]
start mowing
undock
Physically, charging stops the instant the robot undocks and loses connection to the charging pads. The software notices a bit later...
Stop charging
start motor
loop
[Until everything is mown]
drive forward
opt
[at border]
turn
opt
[lost pose]
turn off motor
drive until border wire
drive on wire until localized
opt
[battery low]
turn off motor
drive to docking station
dock
start charging
some time later...
undock
drive back to docking station
dock
start charging
User
Robot
Docking Station

Platform and Components

We will use the Bosch Indego as our test platform.
A unique feature of the Bosch Indego is the Logicut (tm) driving strategy, which builds a map and covers the lawn systematically by driving parallel lanes. Multi-sensors detect obstacles and maneuver around them before continuing cutting on the calculated route. The Bosch Indego lawn-mower is thus an example of not just current, but also of future computing architecture demands in the domestic outdoor robot market.

Components

As is typical, the Indego is a differential drive platform, caster wheels, perimeter sensors, tilt sensors, and a rotating cutting blade on the bottom (shown in the Figure).

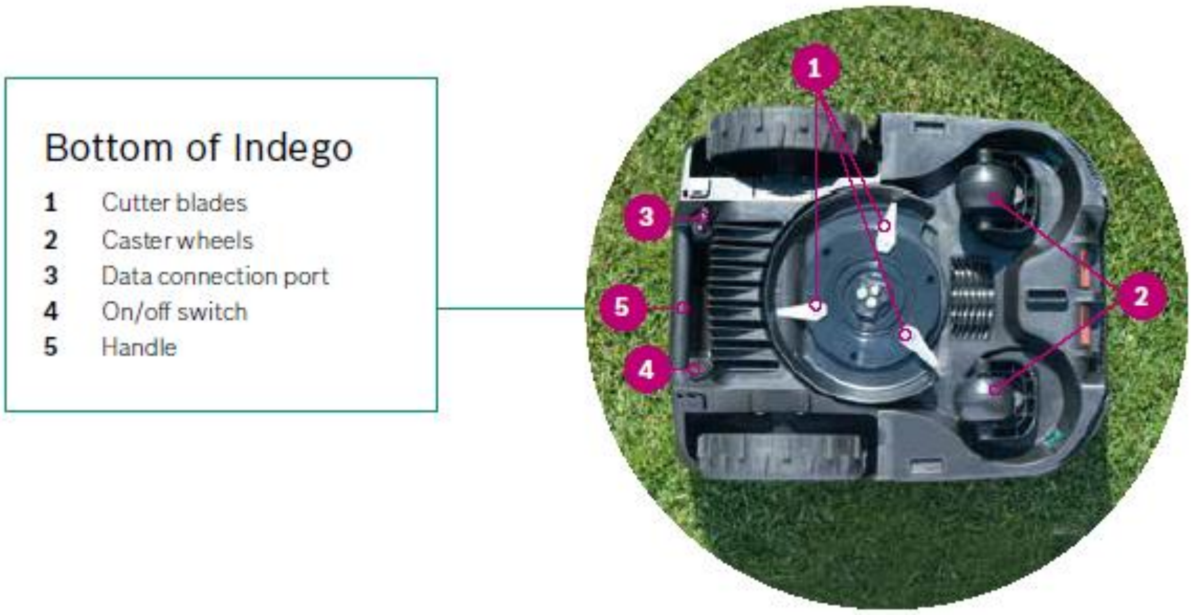


Figure 9: Bottom of Bosch Indego - Copyright Robert Bosch Power Tools GmbH

The most relevant part of the robot for the purposes of this project, and the only one to be modified during it, is the *compute hardware*.

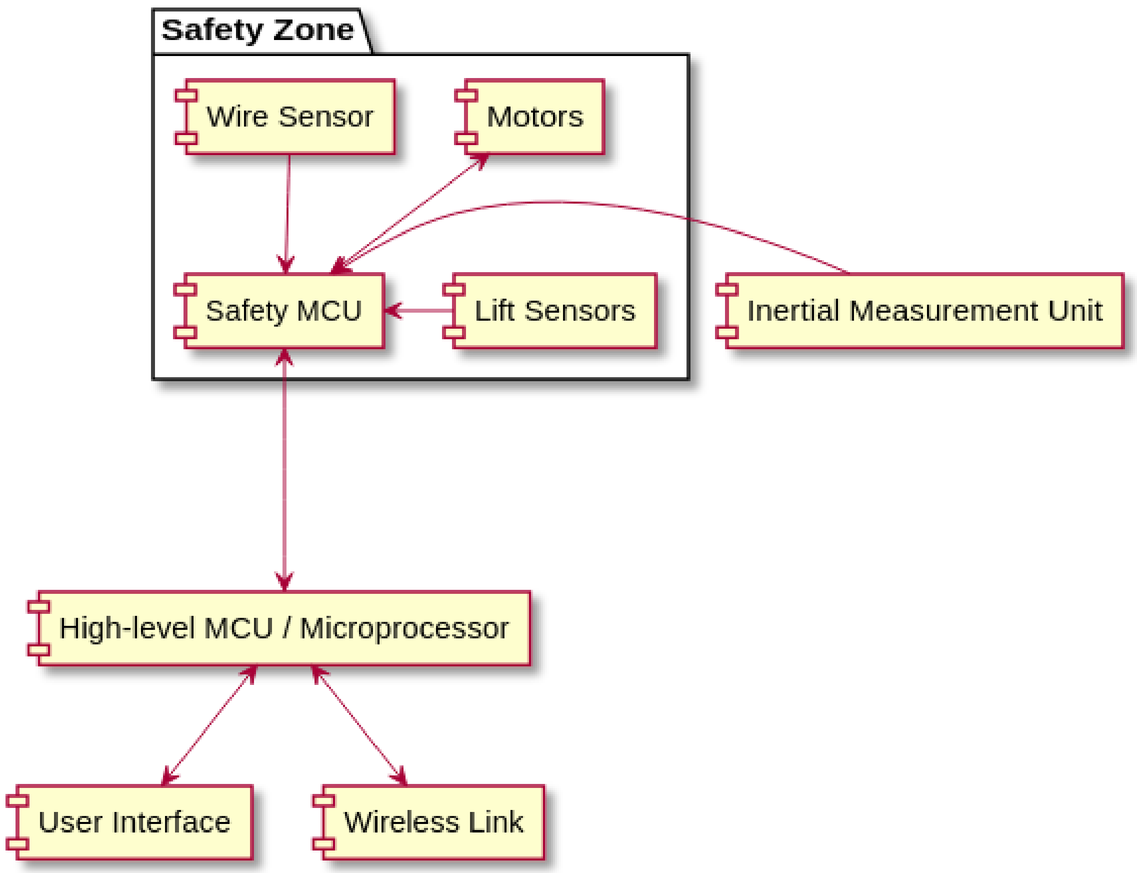
Domestic robots typically employ at least one safety-rated microcontroller connected to sensors and actuators, as well as non-safety-rated microcontrollers or microprocessors for higher-level functions, such as planning and communication. This setup is also used in the Indego lawn-mower.

To be a safe device, the lawn-mower must ensure that 1) the user cannot touch the cutting tool while it is running, 2) the robot does not drive into unsafe areas, such as public roads, and 3) the battery does not explode due to improper charging.

The robot has been designed such that, under normal circumstances, the blades can only be touched by lifting the robot. It is also assumed that the area guarded by the boundary wire is a typical lawn without big holes.

Therefore, the system requires a safety-rated sensor that detects when the robot is lifted, a safety-rated wire detection, a safety-rated motor shutdown procedure, a safety-rated charging module, and a safety-rated MCU to run the software on.

The following diagram shows the most important components and the safety zone.



Software

1. **Hardware driver**: For the wheel encoders, inertial measurement unit, wire sensors, motors, etc. (*proprietary*).
2. **micro-ROS framework**, including Real Time Operating System (RTOS), time synchronization, communication middleware, diagnostics and monitoring, real-time scheduling modules, lifecycle, etc.
3. **Functional software**: Coverage-based navigation, safety, docking, etc. (*proprietary*).

Note that safety-certifying micro-ROS is not part of this project. Therefore, on the safety MCU micro-ROS integration will be limited to basic communications (message serialization, serial comm, parameters). This is already a big help during integration, without costly re-certification.

Work Package dependencies

Work Package	Deliverable	Technology	Description
WP2	D2.2 - D2.6	Platform firmware	Use of firmware functionalities
WP2	D2.10 - D2.13	RTOS Scheduling	Use of RTOS scheduling concepts
WP3	D3.1 - D.39	Middleware functionalities	Communications dependency
WP4	D4.1 - D4.3	micro-ROS client library	Communication, node graph, parameters, …
WP4	D4.4 - D4.7	Real-time executor	Scheduling of micro-ROS nodes
WP4	D4.8 - D4.10	Lifecycle and system modes	Runtime system configuration and diagnosis
WP4	D4.11 - D4.13	Embedded transform (tf) subsystem	Use of TF data structures and algorithms
WP5	D5.5 - D5.6	Benchmark tooling for developers	Use of tools during development of use-case

3.4 Use-case: Smart warehouse (SW)

Overview

There is an increase in deployment of mobile robots in warehouses, where substantial area can be covered by fleets of logistic platforms, used for autonomous or semi-autonomous transportation of goods. Environment and software often need tailoring for the specifics of particular warehouse, but more adaptable platforms emerge. Robots can communicate with external systems to acquire tasks, trigger certain behaviors, notify of errors and to receive or store important data.

In realization of this scenario through the use-case, a robot is operating in a smart warehouse, where challenges are resolved by local communication with various devices. The robot moves through the area, interacting with other devices and gathering data. Devices include a central system that provides the context information as well as sensors and effectors, further detailed in this document.

Purpose

To test, demonstrate and validate:

- micro-ROS communication in a distributed system of heterogeneous devices,
- resilience to dynamic changes in the communication network,

- micro-ROS Bridge in a realistic environment, facilitating communication between low-power, microcontroller-based hardware and a regular robot,
- to showcase micro-ROS deployment with several different devices,
- and to show that micro-ROS enables enough diagnostics to empower safe and robust behavior in case of faults.

A degree of autonomy for the platform is required to successfully go through the scenario. Since the autonomy and reasoning systems are not in the focus of the project, but only serve as necessary means to prove a point about micro-ROS, the focus will not be on developing a sophisticated and robust solutions. Simplifications such as predetermined paths, controlled environments and human guidance will be used when necessary.

The scenario can be seen as a useful demonstration of micro-ROS capabilities, since one can point to real world use-cases in robotics that follow a similar pattern which starts with task and context acquisition and follows with execution through communication with heterogeneous distributed systems.

The use-case also places focus on safety in the context of communication, especially with the external laser scanner.

Scenario environment

To test whether the system is capable of meeting the challenges associated with navigating around such area, we will prepare a test area on the PIAP premises. In the past, PIAP had used DHS-NIST-ASTM International Standard Test Methods for Response Robots to validate and benchmark robots, which can be included as a part of the indoor scenario. However, in order to meet this use-case goals, a more spacious test area is required. The scenario area in PIAP will include an outdoor section to increase overall space, which is possible since the PIAP platform can traverse uneven terrain. The area will be available for on-site demonstration. It will include an indoor starting point, an intersection, an effector-opened door and an outdoor section with a final lights effector.

The scenario realized with MiR will not include an outdoor section, but there will be sufficient indoor area available.

Scenario flow

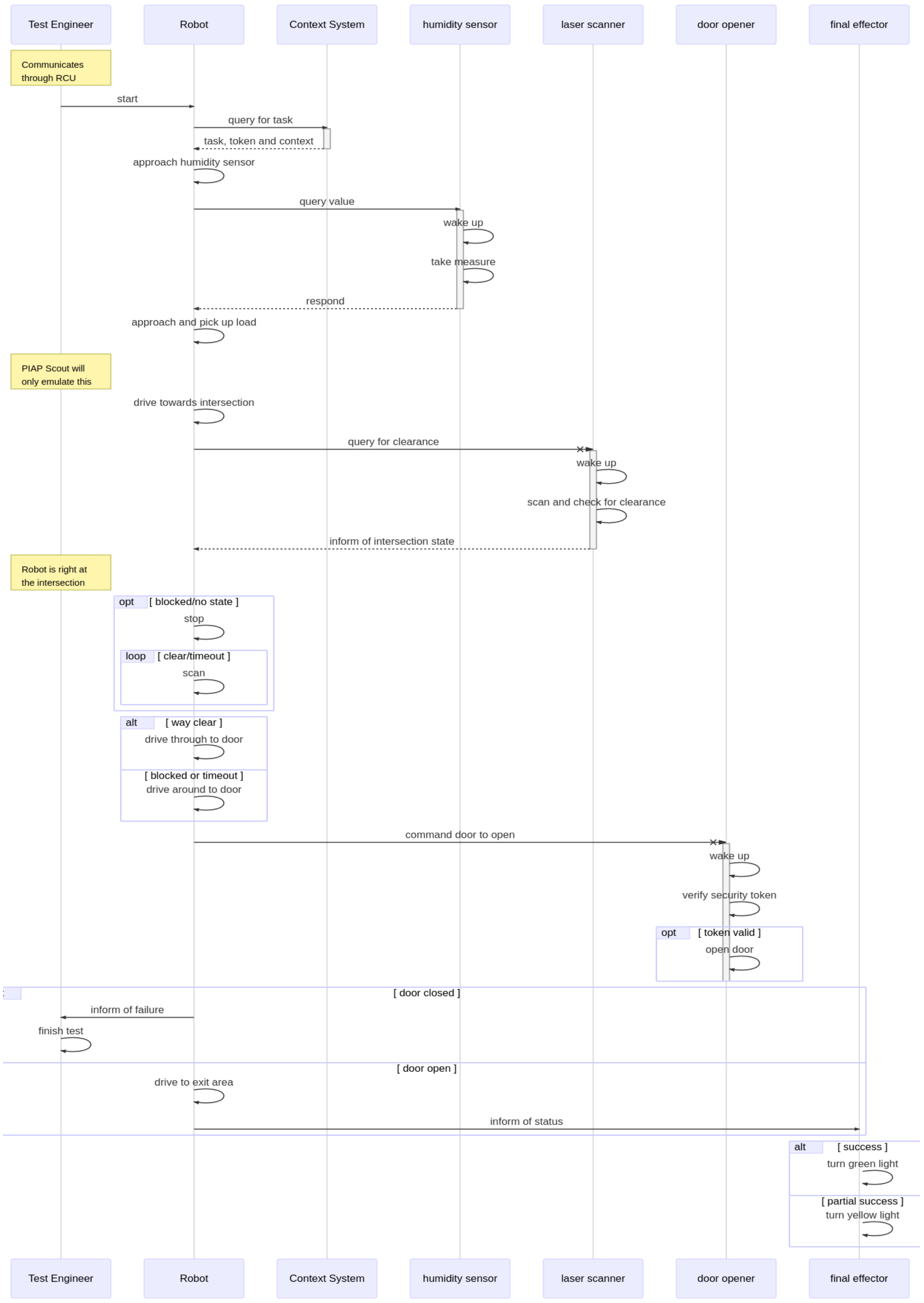
The scenario starts when robot is ordered to approach the context system and communicate to receive informations about the task. In the first implementation of the use-case, the system will only serve the security token necessary for opening the door. In the joint use-case, the system will provide all necessary information about the task and other actors, and additional data (such as whether the robot can go outdoor, by providing information from a weather station).

The platform will then acquire data by approaching and communicating with a humidity sensor. All sensors and effectors can function in a low-power mode at the start of the scenario and need to activate to send data. In the MIR scenario, the platform will additionally pick a load. PIAP Scout is unable to do that autonomously and without a gripper, but will emulate the action through communicating through ROS2 with a ROS2 node running on the Robot Control Unit. This way the validation process for both platforms will be more unified.

Subsequently, the platform continues through the indoor area towards the door. It will approach an intersection and communicate with a laser scanner system, commanding it to wake up and performs a scan to determine whether the crossing is clear. The robot pauses if it doesn't receive a "go" from the laser scanner, otherwise it passes the crossing without stopping.

The robot will approach the door and use the security token to activate the effector that opens it. The robot will proceed outdoors. In case of the MIR scenario, it will proceed to the next indoor area.

Finally, in the case of PIAP Scout, it will approach the final effector and communicate with it, which will result in a colored light appearing. MIR will instead deliver the cargo to a designated area and report task success.


Test Engineer
Robot
Context System
humidity sensor
laser scanner
door opener
final effector
Communicates through RCU
start
query for task
task, token and context
approach humidity sensor
query value
wake up
take measure
respond
approach and pick up load
PIAP Scout will only emulate this
drive towards intersection
query for clearance
wake up
scan and check for clearance
inform of intersection state
Robot is right at the intersection
opt [blocked/no state]
stop
loop [clear/timeout]
scan
alt [way clear]
drive through to door
[blocked or timeout]
drive around to door
command door to open
wake up
verify security token
opt [token valid]
open door
[door closed]
inform of failure
finish test
[door open]
drive to exit area
inform of status
alt [success]
turn green light
[partial success]
turn yellow light
Test Engineer
Robot
Context System
humidity sensor
laser scanner
door opener
final effector

Platforms and components

Robots

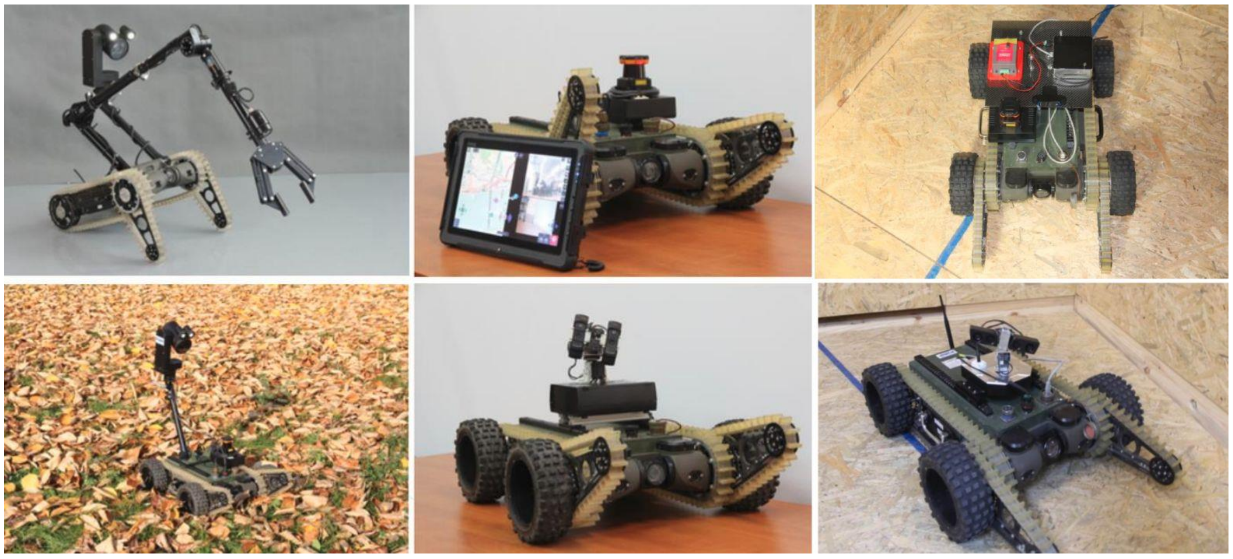


Figure 10: PIAP Scout in various setups, including modules for autonomy, telepresence, robotic arm, PTZ camera and task planning.

PIAP Scout® is a reconnaissance robot with a small frame suitable for operating in places that are not easily accessible for other, larger robots. It has excellent maneuverability and relatively high speed (7 km/h). The Scout has already been used by PIAP in multiple projects as a reliable, modular platform. For the use-case needs, the autonomy module and a laser scanner will be equipped, which are readily available. As soon as the micro-ROS hardware bridge module is ready, the platform will integrate it as well. PIAP Scout® is equipped with IMU and a GPS and supports Wi-Fi and Ethernet for communications. The platform runs native libraries and protocols, but supports ROS through the autonomy module and configurable interfaces.

- Dimensions: 67x54x57cm in standard configuration (L x W x H). Height is reduced with the use-case configuration, since the manipulator is removed.
- Weight: 20 kg and more, depending on the setup.

Modules include:

- ROS-based autonomy module. It will be adapted to work with the use-case. The adaptations will include porting it to a newest stable ROS 2 version and implementing necessary interfaces with the Bridge.
- Hardware micro-ROS bridge based on H-ROS®, to be provided by ALR. PIAP can support mounting necessary for the platform.

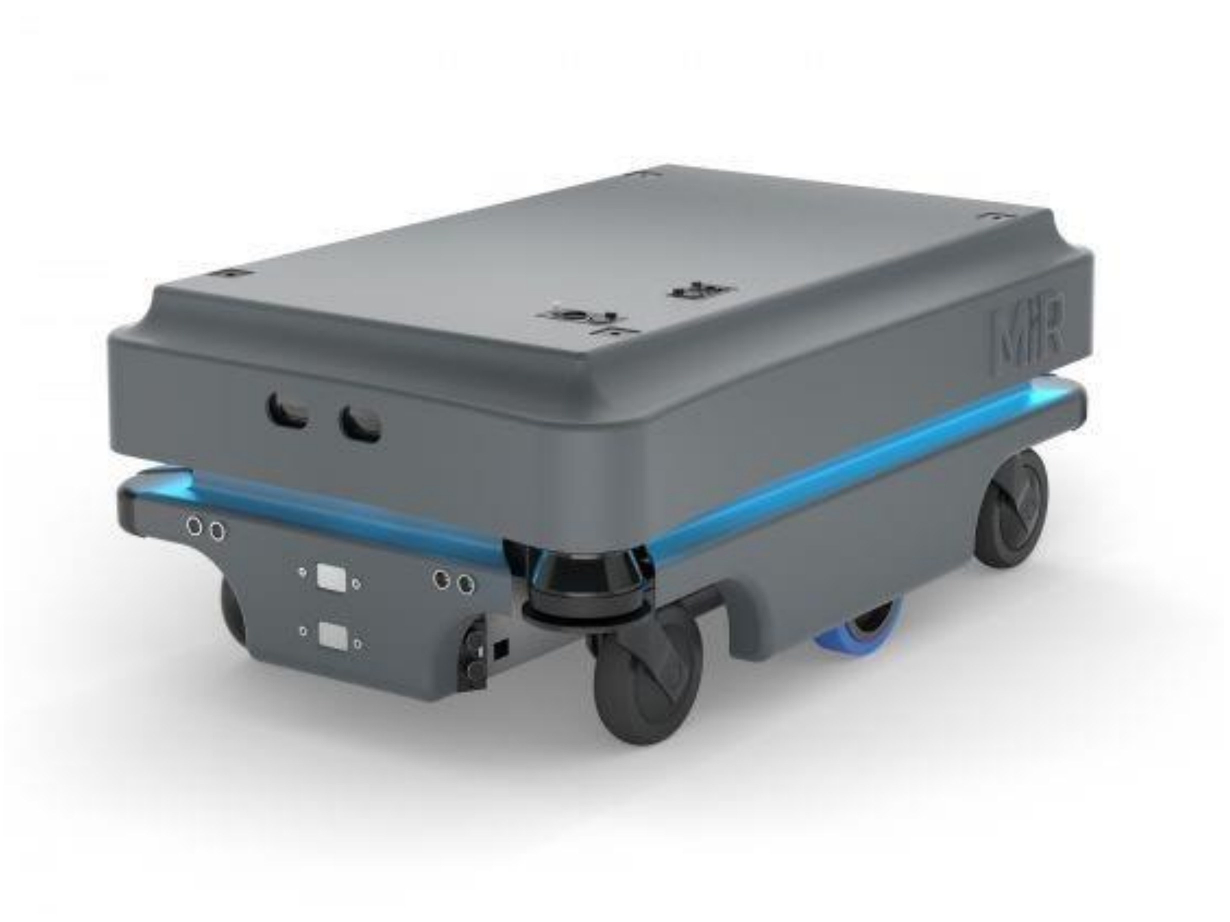


Figure 11: A MiR200 platform by Mobile Industrial Robots in a basic setup

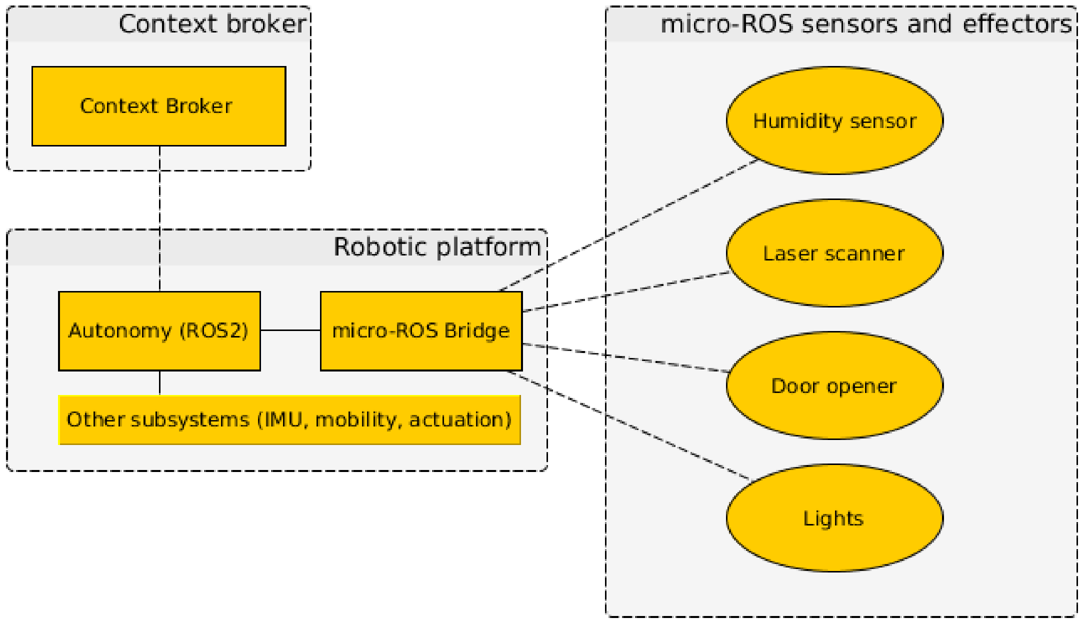
MiR200™ is a state of the art autonomous mobile logistics platform created by Mobile Industrial Robots. It can be used to transport payloads of up to 200 kg and safely maneuver around people and obstacles. MiR200™ can travel at speed of up to 4km/h in the forward motion and 1km/h backwards. In terms of communication, it supports WiFi and Bluetooth 4.0. The platform will run ROS 2.0 by the time it will be applied in the scenario and can be integrated with the micro-ROS bridge module through a scripting interface that provides autonomy primitives and high-level functions. If for some reason the platform won't be running ROS2 yet, we will use micro-ROS bridge together with the ROS2-ROS1 bridge to communicate with the ROS1 interface of the platform.
The robot will be equipped with a top-module suitable for pickup and delivery, which will be selected from one already produced and supported by Mobile Industrial Robots.

The bridge module is intended to be the only not yet supported extension to the existing commercial platform.

- Dimensions: 89x58x35.2cm (L x W x H). Height is increased if modules are mounted on top.
- Weight: 67 kg without any load

Components

Component	Description
micro-ROS bridge	facilitates connections between ROS2 system to micro-ROS sensors and effectors.
Laser scanner module	external, with a board running micro-ROS. Provides information on whether the intersection is free
Autonomy module	for both platforms, this is the component that interfaces with the micro-ROS Bridge to facilitate communication between robot internal modules and micro-ROS sensors and effectors. Runs ROS2.
Humidity sensor	mounted on a wall, running micro-ROS on the reference platform.
Door opener	an effector with a board running micro-ROS on the reference platform, engaging a motor to open a door in response to a valid token.
Context system	running on a desktop machine, it will provide the context information through Fiware Orion Context Broker for the joint use-case. The machine will communicate with the platform through ROS2.
Final effector system	showing green, yellow or red light, running micro-ROS on the reference platform.
Robot Control Unit	to be used to control and diagnose the robot, visualizes data and display the local map and robot position. Runs ROS2



Context Broker communicates with Robotic platform through ROS2, while all communication with external sensors and effectors goes through micro-ROS bridge.

Work Package dependencies

Work Package	Deliverable	Technology	Description
WP2	D2.1	Reference hardware	use of supported hardware for all sensors and effectors
WP2	D2.2b	RTOS release	use of RTOS for sensors and effectors
WP2	D2.3a	Hardware bridges	(upon completion) hardware bridge will replace the software version. Report will help to prepare for it
WP2	D2.3b	Software micro-ROS bridge	necessary to facilitate communication in the desired way
WP4	D4.1,4.2,4.3	micro-ROS core libraries	they are what is to be validated about micro-ROS through this scenario

4. Coverage Analysis

4.1. Approach

Coming both from our reference scenarios and envisioned applications of micro-ROS in general, there are multiple ways of using the modular platform of micro-ROS, putting focus on different aspects of the framework. These aspects include wired and wireless communication, various network topologies and discovery mechanisms, trade-offs between minimal power consumption and extended capabilities, different sensors and actuators. It is beneficial to analyze these aspects and attempt to predict or recognize typical configurations in which these aspects come together, since these configurations can produce significantly different requirements.

In this chapter, we setup a simple formal framework to derive such configurations systematically, but independent from our use-cases. Then, we analyze the coverage of these configurations by our use-cases to identify such aspects that are not covered by our reference scenarios.

In the project proposal, we already sketched typical communication configurations (cf. Section 1.3.3) and explained relevant configurations for safety-rated MCUs (Section 1.4.8) and low-power communication and computation (Sections 1.3.3.1 and 1.4.3). Furthermore, in the first months of the project phase, we analyzed typical applications such as (1.) a simple, wired actuator, (2.) a wireless sensor with low-power requirements, (3.) the use of micro-ROS on an existing safety-rated MCU and RTOS by means of all four layers of the conceptual architecture given in Section 2. Our analysis revealed that many of the aspects are independent of each other. Therefore, we decided to consider them as *configuration dimensions* of a (large) variant space, instead of defining them by means of complete descriptions from MCU hardware to application.

4.2. Morphological box of configuration dimensions

To specify the configuration dimensions in a systematic manner, we use a Morphological Box[1]. Each row represents one configuration dimension. The dimension's name is given in the first column, followed by two or more variability points - i.e. the *configuration options* or just *options* - for this dimension. All configuration dimensions are specified from the point of view of an MCU with software functions based on micro-ROS.

Dimension name	Option 1	Option 2	Option 3
Communication characteristics	mainly send	mainly receive	send and receive
Link type	wired (e.g., Ethernet, serial)	wireless	
Link topology	1 MCU only	1 MCU to 1 agent	n MCUs to 1 agent (*n MCUs to m agents in extended version*)
Topology dynamics	static	dynamic (including discovery)	
System dynamics	Static nodes and topics	Dynamics by means of node states and parameters only	Dynamic appearing/ disappearing of nodes and topics
Power characteristics	Always on	Periodic sleeping	"Aggressive" sleeping, wake-ups triggered by external signal
Integration with existing MCU software	none, use of full micro-ROS stack	Other RTOS than NuttX given	Other RTOS and framework (scheduling, communication, configuration) given
Application source code	From scratch	Ported from existing ROS application	Ported from existing non-ROS application
Real-time	few seconds delay tolerable	real-time computations only	real-time computation and communication
Safety	not safety critical	safety critical application/ components	

Note that by intention, we do not indicate an assignment to the conceptual layers given in Section 2 since many of the configuration dimensions describe cross-layer aspects. In the following, we further motivate and explain each dimension:

- *Communication characteristics* specifies the major communication direction of the micro-ROS-based application on the MCU. Sensors are a typical example for the first option *mainly send*. Most of the time, they send data only, but they may provide a configuration interface. In contrast, actuators *mainly receive* data only, but typically provide some feedback on the received commands. The third option *send and receive* covers all other applications. The *send and receive* option can also include the use of services.

- *Link type* coarsely classifies the link layer protocols and underlying physical layer into wired and wireless.
- *Link topology* characterizes import topologies with regard to the MCUs and the agents. Further stronger devices running DDS and ROS2 are not considered. The first option for this dimension describes the special case of a single MCU running micro-ROS only. The second option describes the most simple standard case consisting of one MCU and one agent. This also includes a special case of peer-to-peer link, where a minimal agent is used between two MCU nodes. The third option in contrast allows for communication between MCUs via one and the same agent. This includes the extended case that an MCU may be connected to multiple agents at the same time.
- *Link dynamics* specifies whether the topology is predefined statically or may change during runtime.
- *System dynamics:* This dimension describes the dynamics of the system/application on the component level with respect to components on the considered micro-ROS-based MCU.
- *Power characteristics:* If MCUs are embedded in larger robotic systems with powerful computational hardware and/or actuators, their power consumption is negligible compared to the other hardware. Often, they will therefore be used in an *always on* style. However, for smaller, battery-powered systems, an MCU's power consumption is relevant. Therefore, MCUs are often used with *periodic schedules* switching between sleep/stop phases and run phases. As the wake-up may take only about 10 to 200us (see *Application node on using STM32F4 MCU power modes*[2]), such periodic schedules are possible even for control or sensing tasks at 1kHz. For tiny, passive devices even lower power consumptions may be required. A prominent example for the third option *"Aggressive" sleeping, wake-ups triggered by external signal* are battery-powered sensors deployed in the infrastructure to supply mobile robots with additional information. To minimize the power consumption of the sensor's MCU, external mechanisms are used to wake up the MCU only when necessary, e.g., on incoming communication requests.
- *Integration with existing MCU software:* When developing micro-ROS-based application software for an MCU, different constraints by existing software might be present. We distinguish three options. The first option *none, use of full micro-ROS stack* describes the ideal case, where no existing software on the MCU has to be taken into account. The other extreme is the option *Other RTOS and framework (scheduling, communication, configuration) given*. In this case, the micro-ROS-based application software is only one component amongst many others in an existing software system for the MCU. Examples are the Intel Quark MCU stack based on the RTOS Zephyr or the Bosch XDK sensor box running FreeRTOS. These software stacks even come with specific IDEs. The option *other RTOS than NuttX given* describes a case in-between those two. For maintenance or safety reasons, an RTOS other than NuttX is already defined, but the micro-ROS-based application software can be developed without further restrictions.

- *Application source code:* The ideal case that all software is developed from *from scratch*. With regard to existing software to be included into a micro-ROS-based application, we distinguish between two options: *Ported from existing ROS application* versus *Ported from existing non-ROS application*. Here, the former case covers ROS1- and ROS2-based software.
- *Real-time:* MCUs are often used for real-time critical tasks due to their predictable computing times. One important differentiation with regard to micro-ROS is whether real-time constraints affect the MCU only or also the communication with other MCUs or processors, i.e. *real-time computations only* versus *real-time computation and communication*. An example for the former option is a safety-MCU that monitors a robotic system and stops the motors in case of danger. We deem the first case *few seconds delay tolerable* to be very rare. It might apply to MCUs used in sensor nodes as in the Smart Warehouse use-case.
- *Safety:* In the simple case *not safety critical*, none of the MCU software - micro-ROS-based or not - contains any safety-critical functions. In the *safety critical application/components* option, a safety-critical function is either implemented using micro-ROS or a micro-ROS-based software component is integrated with safety-critical software on the same MCU. Both require domain-specific measures during development.

4.3 Coverage by reference scenarios

From the Morphological box of configuration dimensions with their options, concrete configurations could be derived as points of the Cartesian product spanned by these dimensions. This yields 3x2x3x2x3x3x3x3x3x2 = 17496 concrete configurations, but some of them are *unimplementable* due to loose couplings between dimensions. For example, a dynamic topology with discovery implies a wireless link type and can thus not be combined with the wired link type option. Yet, thousands of implementable configurations remain and the use-cases are four examples of them.

To ensure that all options are considered in the requirements analysis, we classified our use-cases in terms of the configuration dimensions. Doing so, we identified those options that are not covered by our use-cases to pay particular attention to them.

Dimension name	Autopilot Drone (AD)	Modular Arm (MA)	Smart Warehouse (SW)	Domestic Outdoor Robots (DOR)	Uncovered configuration options
Communication characteristics	3	1	1	3	2
Link type	2	1	2	1	-
Link topology	3	3	3 (extended)	2	1
Topology dynamics	1	2	2	1	-
System dynamics	2	3	3	2	1

Power characteristics	2	2	3	1	-
Integration with existing MCU software	1	1	1	3	2
Application source code	1	2	1	3	-
Real-time	3	3	1	2	-
Safety	1	1	1	2	-

This classification revealed four uncovered configuration options. Obviously, the first two are very closely connected and represent the simple special case of robotics system consisting of a single MCU with a micro-ROS-based application without any further connection. A cheap toy robot could be an example of such a system.

The third and fourth uncovered configuration options are "Static nodes and topics" of the dimension "System dynamics" and "Other RTOS than NuttX given" of the "Integration with existing MCU software" dimension, respectively. Our use-cases cover more complex options of those two dimensions. However, support for these simpler options without unnecessary complexity should be ensured.

5. General requirements from reference scenarios

Requirements are divided into functional, performance and non-technical.

5.1 Functional requirements for micro-ROS and underlying platform

Functional requirements introduced in this section are grouped into following categories:

- **Communication**: including supported protocols and required capabilities.
- **Network topology**: what configurations of communicating nodes are to be supported.
- **Micro-ROS bridge**: functionalities that are specific to the bridge between ROS2 and micro-ROS.
- **Platform support**: which hardware and RTOS configurations micro-ROS is required to support.
- **Power management**: which power management functionalities of the underlaying platform should be supported by micro-ROS framework.
- **Diagnostics and tooling**: developer tools and diagnostic functionalities that are required from micro-ROS to assist developers.
- **System runtime configuration and tooling**: allowing developers to manage the lifecycle, parametrization and states of the running components (nodes).
- **Information model and messaging**: requirements for standardization of messages as well as providing a set of standard messages with support for developing custom ones.

To indicate the source of requirement, abbreviations for use-cases will be uses as follows:

Short	Use-case
AD	Autopilot Drone
MA	Modular Arm
DOR	Domestic Outdoor Robots
SW	Smart Warehouse

Communication

Micro-ROS shall support several types of protocols and capabilities of communication layer as detailed in the table:

ID	Short name	Description	Relevant use-cases
COM1	Wireless	Support for either 6LoWPAN or ZigBee (or both)	SW, AD
COM2	Ethernet	Support for Ethernet	MA
COM3	USB/UART/SPI	Support for serial protocols, SPI support needs to take DMA limits into account	DOR, MA, AD
COM4	Independent serialization	micro-ROS serialization has to be usable independent of the whole stack, integrating with existing application specific protocol over SPI	DOR
COM5	Data forwarding	Option to forward all data (i.e. subscribe to all)	DOR
COM6	Dynamic discovery	Dynamic discovery of participants without previous knowledge about their configuration	MA, SW
COM7	Reliable QoS	Support for reliable (TCP-like) QoS	SW

Network topology

The micro-ROS system should be useful for several common distributed topologies, as guided by application in the reference scenarios, through the dimension “Link topology”.

ID	Short name	Description	Relevant use-cases
NET1	Client-server	Static topology with client/server-like configuration (one MCU, one micro-processor)	DOR, MA
NET2	Peer to peer	Peer to peer communication between micro-ROS nodes, in case a micro-ROS Agent is not present and there is a link between the nodes. This requirement could be met through lightweight agents created dynamically	AD

NET3	One-to-many	One-to-many topology with micro-ROS bridge	MA, SW
NET4	Many-to-many	Many to many topology, involving multiple robots present, as in extended version of reference scenario 3	SW

micro-ROS bridge (specific)

The micro-ROS bridge should support the use in reference scenario and provide a seamless integration experience for developers accustomed to ROS ecosystem. A bridge includes a micro-ROS agent and corresponds to reference scenarios 2 and 3 in the link topology dimension. It is going to be deployed and validated through Modular Arm and Smart Warehouse use-cases. Requirements below have been drawn through analysis of the application backgrounds behind these use-cases.

ID	Short name	Description	Relevant use-cases
BRG1	Ethernet	Support for Ethernet in hardware bridge	MA, SW
BRG2	QoS configuration and translation	Configuration of QoS and translation from ROS2 into micro-ROS QoS and the other way around	SW
BRG3	Mechanical	Ease of attachment, portability and fitting dimensions of the hardware box for the platforms. Optional battery feeding connector	MA, SW
BRG4	Bridge tooling	An extension of tooling for debugging, diagnostics, logging and monitoring for micro-ROS bridge (including integration for ROS2 tools)	MA, SW
BRG5	Topic priorities	Support for topic priorities when bandwidth is limited (e.g. control messages over transient, frequent readings)	SW
BRG6	Seamlessness	The micro-ROS component should start, terminate and function seamlessly alongside the platform, and also work in a plug-and-play manner as long as the robot is ready for the topics provided	MA, SW

Platform support

The micro-ROS aims to abstract hardware and OS for the user, however, due to limited resources we aim to provide full support for focused setups.

ID	Short name	Description	Relevant use-cases

PLA1	Olimex	The reference/default development board is Olimex LTD STM32-E407 (cf. Deliverable 2.1) and full support for it is required from micro-ROS. The rationale is that Olimex LTD STM32-E407 offers a good communication means - contains Ethernet port and several wired communication bus interfaces - and enough computing resources for a typical micro-ROS scenario where Micro RTPS is used for sending and receiving data	All
PLA2	STM-L1	Support for STM-L1 board. For simpler micro-ROS profile devices, for example only-sender or only-receiver devices (section 4.2, Morphological box), the STM-L1 board is more suitable, as it should be enough for running Micro RTPS and computing simpler pieces of code, such as sensor data fetching and sending	All
PLA3	NuttX	Support for NuttX RTOS. NuttX is POSIX based works well with the reference board. This setup allows us to cover the specific requirements of reference scenarios	All
PLA4	Other RTOS	Support for other RTOS's as required by existing products to the extend needed (i.e. communications only)	All

Power Management

The micro-ROS library should make good use of power management available on the device through abstractions of the OS calls. This is a non-trivial feature, due to the fact that each supported RTOS could have each specific mechanism, making difficult the abstraction to upper layers. At the beginning, this power management control could be handled at the RTOS level implementing cyclic or interrupt based micro-controller wakes up.

This management is already contemplated and tested under the default RTOS. Deliverable 2.1 offers a first insight of how is implemented and measurements of the power savings under test examples. This requirement is mainly driven by reference scenarios 2 and 3

ID	**Short name**	**Description**	**Relevant use-cases**
PWR1	Low-power MCU state	Support for low-power stand-by state on the MCU	AD, MA, SW
PWR2	Wake-up through message	Quick platform wake-up from sleep mode through a message	AD, MA, SW
PWR3	Power states control and configuration	Sleep/idle/low power mode should be configurable	AD, MA, SW

Diagnostics and tooling

These requirements are generalized from reference scenarios and our experience with development frameworks (such as ROS). Application level support for diagnostic and specific tooling aims to support developers of micro-ROS applications in debugging, finding performance bottlenecks as well as understanding topology and behavior of participating nodes. Diagnostic tooling is relevant for all use-cases.

ID	Short name	Description
DIA1	Platform diagnostics, HW	Support for underlaying platform diagnostics, including hardware error messages and power consumption of MCU and connected devices and battery state.
DIA2	Platform diagnostics, OS	OS memory consumption, computational load, deadline misses.
DIA3	Communication diagnostics	Communication statistics: message latencies, delayed messages, message corruptions.
DIA4	Components diagnostics	Support for diagnostics of functional performance of application components/nodes, such as localisation losses.
DIA5	Monitoring	Diagnostic information should be provided in real-time for online monitoring, but summary reports should also be queryable.
DIA6	ROS2 compatible	The data formats should preferably be based on / extend existing ROS diagnostics data types for compatibility with existing tools for data aggregation and visualization. Similarly, tooling for data aggregation and visualisation of MCU and micro-ROS specific diagnostic information should be interoperable or extend common ROS tools (e.g., in form of plugins)

System runtime configuration and tooling

Runtime configuration and tooling is relevant to all use-cases.

ID	Short name	Description
RUN1	ROS2 lifecycle	micro-ROS nodes should support the node lifecycle defined for ROS2 nodes[3]
RUN2	Modeling concepts	micro-ROS has to provide modeling concepts for hierarchical composition of (sub-)systems and the definition of corresponding system modes based on application nodes and their lifecycles
RUN3	Start-up	Based on these models, micro-ROS shall provide mechanisms to start-up whole (sub-)systems as well as to switch between their modes deterministically. Corresponding mechanism shall be also provided for ROS, to allow for distributed system runtime configuration across multiple MCUs and micro-processors

RUN4	Dynamic component parameters	micro-ROS shall provide mechanisms for dynamic management of component parameters, compatible with ROS mechanisms (which are not yet defined for ROS 2.0)
RUN5	States and mode changes	Information about states and system mode changes, including failures (exception states / modes), has to be provided in real-time and has to be queryable by other nodes. This implies that micro-ROS-based components on an MCU have to be reconfigurable in terms of the lifecycle and their parameters from other MCUs or micro-processors via the agent. Relevant application components (e.g., from some executive/deliberation layer) shall be able to switch between system modes and node states) using a suitable ROS service interface.
RUN6	Sub-states	micro-ROS shall also allow to define application-specific sub-states of a node's lifecycle states (e.g., safety modes of device drivers) and allow their communication and runtime configuration

Information model and messaging

The micro-ROS information model aims to standardize types and information content of messages. Clear standards allow easy integration of conforming subsystems. A subset of messages should be supported from the start, ranging from traditional ROS messages for navigation, geometry and sensors, through more micro-ROS specific diagnostic and power messages. These requirements are driven by use-cases. The case for developing a proper information model also comes from plans of integrating with the FIWARE Orion Context Broker.

ID	**Short name**	**Description**	**Relevant use-cases**
INF1	Geometry and Nav	Support for standard messages of geometry and nav: odometry, velocity command, etc.	AD, DOR
INF2	Sensor	Support for standard sensor messages: bumper, IMU, laser scan, rangefinder, etc.	AD, MA, DOR
INF3	Diagnostic messages	Support for standard diagnostic messages: HW state, SW state, safety related, battery state, resource utilization etc	DOR, MA, SW
INF4	Custom messages	Support for custom messages, including security token exchange and state report for Smart Warehouse scenario, drone control messages for Autopilot Drone scenario	AD, SW
INF5 (Optional)	Authentication	A nice to have would be to introduce a standard way of authenticating as a part of request or discovery	SW

Details of specific message requirements per use-case:

Use-case	messages
Modular Arm	distance measurement, power measurement, resource utilization, power mode selection, range sensor data, range sensor rate configuration
Domestic Outdoor Robots	odometry, command velocity, bumper, IMU, battery states, diagnostics (HW, SW, safety states)
Smart Warehouse	power messages such as wake-up and sleep suggestion, humidity value reading, opening a door with security token (preferably, this should be a service), final state message, diagnostic messages throughout the scenario
Autopilot Drone	IMU, range sensor data, flying commands (off-board control), diagnostics, sensor messages, autopilot position, odometry

5.2 Performance requirements for micro-ROS

Challenges in drawing performance requirements

Defining performance requirements for the micro-ROS platform is not an easy task. Firstly, it is difficult to make a comparison of micro-ROS as a whole against a state of art solution that would offer similar functionality in a similar domain, simply because there are no such solutions. The best we can do is to find a system that realizes a minimal version of micro-ROS in a limited domain and another one, which is functionally more akin to a full-fledged micro-ROS configuration. In the light of this reasoning, we have chosen rosserial as the minimal system and ROS2 embedded in RIOT as the other end to compare against. It is still not easy to select axes of comparison or to determine relevant configurations to compare, but we can make measures pertaining to the entirety of the system that make sense, such as resource consumption or overall latencies.

Possible uses of micro-ROS are numerous and the target groups of users are diverse, ranging from academics to developers of demanding industrial robotic systems. Ultimately, the performance of the micro-ROS system should be to their satisfaction, removing the incentive to implement tailored solutions from most applications.

Another important issue with drawing precise performance requirements is the multitude of possible micro-ROS configurations. Since the platform is intended as modular and configurable, there are numerous relevant configurations that will differ in their performance profile, especially when deployed in different environments.

A major advantage of micro-ROS is that it will come with its performance comprehensively benchmarked. The benchmarking, realized as a continuous process with subsequent releases of micro-ROS, will provide help to identify and quantify performance issues during the development as well as providing micro-ROS users with relevant information for their micro-ROS applications.

Performance requirements

Temporal

ID	Short name	Description	Main use-cases
TEM1	Booting time	Booting time to a ready state should be short enough. It will be compared against H-ROS.	MA
TEM2	Total latency	Total latency measured as time from the sensor equipment timestamp on data acquisition to data availability in micro-ROS userspace (rcl layer). Other benchmarks for latency will also be tested.	AD, MA, DOR
TEM3	Time precision	Clock synchronization between main micro-processor and MCU should be precise, with precision not less than 1ms. For the DOR use-case, the frequencies of operation are up to 200Hz, but for small amounts of data and simple fusion algorithms only	DOR
TEM4	Discovery time	If possible, should provide guarantees for common scenarios (discovery in sleep mode, active mode), predictable is better. Discovery time shouldn't include hampering delays, especially with robot moving and demanding time-critical information about the intersection. 300 ms for the discovery of devices in sleep mode is deemed to not significantly interfere with the SW use-case	AD, SW
TEM5	No-copy	Communication between nodes on the same MCU should be effective (no-copy)	DOR

Power

Since part of the rationale for using MCU in robotic systems is due to lower power consumption, it is a requirement for the platform to minimize it.

ID	Short name	Description	Main use-case
PPR1	Power consumption	Power consumption of device working in different states has to be as low as possible without compromising required functionalities. This will be benchmarked. In case of MCU-based modules, increase in power consumption can be investigated by comparing to a simple minimal profile (a baseline implementation of the same functionality such as sending data)	SW

Resources: memory and space

Micro-ROS platform should be resource-efficient in terms of memory and space.

ID	Short name	Description	Main use-case
RES1	Memory usage: node	Relevant micro-ROS components (serialization, diagnostics, run-time configuration, RTOS abstractions, …) shall fit on MCU with 192kB SRAM, together with existing application software	DOR
RES2	Memory usage: agent	micro-ROS agent and standard ROS components for diagnostics and run-time configuration shall fit on micro-processor running an embedded Linux with 256MB RAM only and leave enough space for an navigation stack optimized for low memory consumption	DOR
RES3	Space usage: maximum	Embedded Linux with application software and micro-ROS agent and standard components for diagnostics and run-time configuration must not exceed 2GB on flash. First tests with Yoccto Linux show that an embedded Linux with minimal ROS installation and ROS nav stack takes about 380 MB, thus micro-ROS should not be larger	DOR
RES4	Space usage: efficient	In reference to the DOR use-case, the flash memory footprint of current software is several hundred kB. Any code that shares the base computing platform should not induce a disproportional space requirement. This implies that modularity of micro-ROS should allow to significantly cut on space requirements if only simple functionality is required	DOR

5.3 Non-technical requirements

Non-technical requirements are especially important to drive adoption of micro-ROS in the robotic community. Developers should find using micro-ROS intuitive and rewarding, having support for their issues and power to configure modules and packages to their needs. A natural group of adopters should be the community of current ROS users. This realization drives several non-technical requirements:

ID	Short name	Description
NTC1	ROS standards	Compliance with ROS standards
NTC2	Documentation	Good documentation for developers/users
NTC3	Learning curve	Intuitive use for experienced ROS users, smooth introduction for newcomers - creating your first node has to be easy.
NTC4	Community	Creating community around micro-ROS (adoption is the key!)
NTC5	Extensibility	Extensible through new modules and packages

NTC6	Evolving	Easily upgraded to new versions
NTC7	Transferable	Moving a standard ROS 2 node to micro-ROS or the other way around should be straightforward and documented

[1] Tom Ritchey, General Morphological Analysis – A general method for non-quantified modeling, 2002, http://www.swemorph.com/pdf/gma.pdf

[2] STMicroelectronics, Using STM32F4 MCU power modes with best dynamic efficiency, 2014

[3] ROS2 Design, Open Source Robotics Foundation Inc., http://design.ros2.org/