D2.14

Report on Platform Support

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D2.14
Deliverable name	Report on Platform Support
Date	June 2020
Dissemination level	public
Workpackage and task	WP2 - Task 2.5
Author	Pablo Garrido Sánchez (eProsima)
Contributors	
Keywords	Robots, hardware platforms, requirements, ROS2
Abstract	Initial report with detailed description for micro-ROS hardware and platform support.

Contents

1 Acronyms and keywords

Term	Definition
ROS	Robot Operating System
RTOS	Real-time Operating System
OS	Operating System
CPU	Central Processing Unit
MPU	Microprocessor Unit
MCU	Microcontroller Unit
OMG	Object Management Group
RAM	Random Access Memory
DDS	Data Distribution Service
XRCE-DDS	Extremely Resource Constrained Environment DDS
AI	Artificial Inteligence
TOF	Time of Flight
I2C	Inter-Integrated Circuit
RPi4	Raspberry Pi 4
CAN	Control Area Network
IoT	Internet of Things
UART	Universal Async Receiver-Transmitter
RCLC	ROS Client Library for C
IMU	Inertial Measurement Unit
RMW	ROS Middleware
HAL	Hardware Abstraction Layer

2 Introduction

In this initial version of the hardware platform support document we expose a complete overview of the supported platforms to which micro-ROS has been ported. The work package 2 (WP2), of which this deliverable is part, is concerned with the selection, definition and support of the hardware and lower level abstraction layers of the microROS project. The objective of this specific document is to review the current status of the support of different MCUs, development boards and RTOSes achieved during the development of the micro-ROS project.

micro-ROS' first purpose is to bring the possibility of running the ROS 2 core layers to extremely resource constrained devices. The support of these ROS 2 layers in such systems enables users to develop robotic applications using ROS 2 entities and concepts in embedded devices such as microcontrollers and hardware development boards.

The main difference between micro-ROS and ROS 2 precludes the possibility of directly porting it to a resource constrained device. In order to address this impediment, micro-ROS has been designed to use DDS-XRCE as a middleware: an OMG standard specification for interfacing extremely resource constrained devices with DDS. Specifically, the DDS-XRCE implementation selected for micro-ROS (Micro XRCE-DDS) matches the resources requirements as it is designed to heavily optimize mem-

ory usage and to have as few OS dependencies as possible. Please refer to deliverable D1.4 *Overall Architecture Definition* and following releases for a more exhaustive account on the subject.

In general terms, there are three key points determining the feasibility of using micro-ROS in a certain hardware platform and on a given RTOS: - The hardware platform and the RTOS must provide a communication mechanism such as a network stack (TCP or UDP) or a serial-like communication stream. This will be used by the micro-ROS client middleware to communicate with the micro-ROS Agent. The micro-ROS client middleware provides interfaces for creating custom transports based on network or serial paradigms. - The RTOS must have a minimum POSIX compatibility. In the worst case, the RTOS must at least manage the system clocks and provide timing functions (required by the middleware). Ideally, it should support all POSIX functions in order to port all ROS 2 layers features. - The RTOS build system must support a minimum set of C++ functionalities. Ideally, it should comply with the C++11 standard.

2.1 Supported RHDP evolution

As detailed in the deliverables contained in the work package 2, the default Reference Hardware Development Platforms (RHDP) is the Olimex STM32-E407 and the NuttX 7.26 RTOS. However, as will be explained later on in this document, additional work has been carried out to extend the support to other hardware platforms and RTOSes, in order to provide micro-ROS a wider spectrum of utilization. The selection of the main candidates boards and RTOSes to which extend the micro-ROS support to has followed a usage criterion that takes into account the diverse casuistry associated either to low-power, regular or safety-related use-cases. Please refer to deliverable D2.1 *Report on the reference hardware development platforms* for more details.

In Task 2.2, named as *platform lower level and firmware and libraries*, the test and adjustment of the default RTOS (NuttX) has been done. Within this task, a first outcome of the NuttX release has been provided. This outcome consists in a first iteration of a build system supporting micro-ROS Dashing Diademata release, a set of tutorials and examples, and a repository with Docker files that eases getting started with micro-ROS on an Olimex STM32-E407 for the user.

During the evolution process of the micro-ROS environment, specifically during the micro-ROS Foxy Fitzroy release planning, some other development MCU development boards have been tested to function with micro-ROS. This was possible thanks to the fact that some other MCU development boards based on the STM32 MCU family (the same family to which the MCU run by the Olimex belongs) can be ported to the existing micro-ROS cross-compilation architecture. As is going to be detailed in the section *Minimum requirements for micro-ROS*, some requirements regarding RAM, flash and peripherals constrain the set of capabilities that a MCU must have in order to run the micro-ROS layer stack properly.

The boards whose support was developed in parallel with the micro-ROS Foxy Fitzroy porting and during the use-cases deployment have been selected for application-specific purposes. The first of them is the STM32L4 Discovery kit IoT, chosen for its extensive integration of on-board sensors. This enables the use-cases and demos of micro-ROS to interact with environmental sensors such as humidity, temperature, IMU or ToF ranging sensors. The second platform whose support was added during the micro-ROS Foxy Fitzroy porting is the Crazyflie 2.1 drone development platform. This drone platform provides an open-source hardware and firmware product that enables developing real drone applications. The Crazyflie 2.1 build tools have been integrated into the micro-ROS build system in order to enable the use of micro-ROS APIs in the drone application code.

2.2 Supported RTOS evolution

In Task 2.4 we performed an investigation over the RTOSes' scheduling mechanisms. NuttX and FreeRTOS scheduling tools have been also analyzed to check whether they fit with the technical requirements of micro-ROS. As an outcome, this has generated the content of deliverables D2.10 and D2.11, *Report on RTOS scheduling - Initial* and *Report on RTOS scheduling - Revised*. This work has been useful to profile the RTOS characteristics required for the micro-ROS application development process. During the supporting period of micro-ROS Dashing Diademata and micro-ROS Foxy Fitzroy porting, the consortium has realized the growing importance of a new RTOS: Zephyr RTOS. This RTOS, belonging to the Linux Foundation Projects, provides a modern and highly configurable operative system API. It complies with the RTOS scheduling requirements of micro-ROS, so an initial port and prospective has been done. Another good reason for porting micro-ROS to Zephyr is that one of its key features is security, and having the micro-ROS environment running on top of a security-concerned RTOS has been considered valuable by the consortium.

As mentioned before, adding the support for a new RTOS is possible only if given requirements are met. Thanks to its wide POSIX compatibility, the networking/serial APIs, and its partial C++11 compatibility, Zephyr meets these requirements, thus allowing to achieve a micro-ROS Foxy Fitzroy release with Zephyr support. This, in turn, enables and eases the port of micro-ROS to a bigger selection of hardware development boards integrated into the Zephyr RTOS ecosystem that complies with the above-listed resources requirements.

2.3 Minimum hardware requirements for micro-ROS

micro-ROS' minimum hardware requirements are defined by three elements:

1. The instruction memory used by the cross-compiled micro-ROS code.
2. The program memory in stack, dynamic and static utilization.
3. The communication peripherals used by the DDS-XRCE middleware.

The choice of the hardware must be therefore driven and oriented by the need to be compliant with the constrains imposed by the above elements. A thorough and complete analysis to establish such constrains was perfomed by PIAP, yielding as a result the quantitative assessment of several parameters, such as memory usage, time performance and bandwidth consumption of a micro-ROS application under given prototypical circumstances. These results are summarized in deliverables D 5.2 'Micro-ROS benchmarks - Initial' and D 5.3 'Micro-ROS benchmarks - Revised', and the results that are relevant for selecting the adequate hardware are those related to memory profiling. The testing was carried out using for the hardware the Olimex STM32-E407 reference board, and for the software part:

- NuttX RTOS (reference RTOS of the micro-ROS project).
- The Publisher application from NuttX's applications set. • micro-ROS' middleware based on Dashing.

The communication benchmarking was done using both the Serial and the Ethernet connection, with the agent running on a Docker connected to UART/Ethernet.

Regarding the flash memory consumption of a cross-compiled micro-ROS binary, the benchmarking shows that around 512 KB of flash memory is needed. As for the RAM, it shows that micro-ROS uses at least 95 KB, corresponding to 30 KB of static memory usage (stack) + around 65 KB of dynamic memory allocated by the publisher application when running (heap).

Notice, however, that the static memory utilization of a micro-ROS app is highly dependent on the micro-ROS RMW configuration. Given that each micro-ROS entity is associated with a configurable static pool memory and that micro-ROS offers a fully configurable support for transports buffers and other middleware-related entities, it is not possible to provide a quantitative approximation. Please refer to micro-ROS RMW optimization article in micro-ROS website for detailed information.

Taking into account these experimental informations and leaving room for future improvements and optimization, it can be stated in a qualitative manner that a micro-ROS hardware platform should have at least:

- Around 1 MB of flash for instruction memory
- More than 100 kB of RAM memory
- One peripheral that allows packet or stream communication such as a network stack (IP/UDP or IP/TCP) or a serial port (UART or UART over USB).

As stated before, a full micro-ROS support requires, furthermore, compiler with C++ compatibility and a set of POSIX functions in order to be feasible.

2.4 The micro-ROS build system

micro-ROS, since its Dashing Diademata release, provides a build system tool that enables the final user to crosscompile the whole environment for a certain hardware platform and RTOS. This micro-ROS build system is provided as a ROS 2 package so it can be integrated in a ROS 2 workspace.

The micro-ROS build system's workflow is a four-step procedure:

- **Create step:** This step is in charge of downloading all the required code repositories and cross-compilation toolchains for the specific hardware platform. Among these repositories, it also downloads a collection of ready-to-use micro-ROS apps.
- **Configure step:** In this step, the user can select which app is going to be cross-compiled by the toolchain. Some other options, such as transport, micro-ROS agent address, port or serial descriptor can also be selected in this step.
- **Build step:** Cross-compilation takes place and the platform-specific binaries are generated.
- **Flash step:** The binaries generated in the previous step are flashed onto the hardware platform memory, in order to allow the execution of the micro-ROS app.

The micro-ROS build system also integrates convenience tools for installing a micro-ROS agent in a ROS 2 workspace. This way, a final user can integrate a set of ROS 2 applications with a micro-ROS application for a certain target using the default ROS 2 procedure.

3 Development board support

The following platforms are currently supported in the micro-ROS build system for the Foxy Fitzroy release:

3.1 Olimex STM32-E407

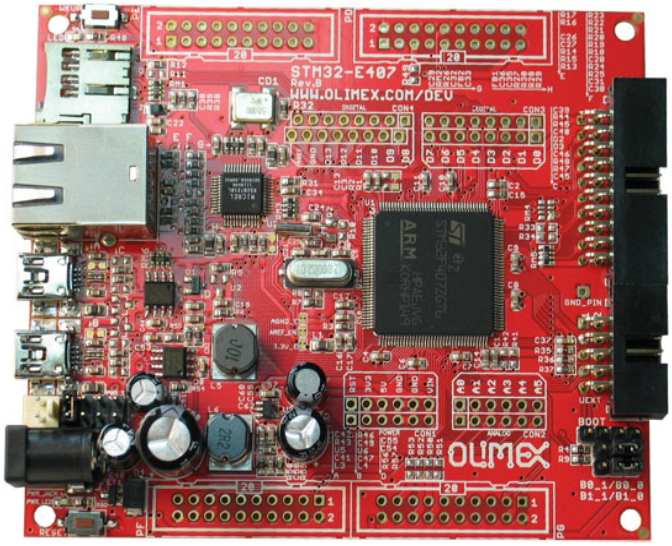

Figure 1: Olimex LTD STM32-E40 hardware development board

The Olimex LTD STM32-E407 is the default Reference Hardware Development Platforms for micro-ROS. It is an open-hardware low-cost entry board for developing custom applications with the STM32F407ZGT6 Cortex-M4F microcontrollers from STMicroelectronics.

As it is the default reference board for micro-ROS, it has suppport for the three RTOS supported by the build system. Please refer to section *Summary of micro-ROS support* for more detailed information.

It contains 196KB of RAM and 1MB of Flash. It is a very complete board thanks to the wide variety of communication interfaces it offers: USB OTG, Ethernet, SD Card slot, SPI, CAN or I2C buses are exposed. The board contains various expansion options available: Arduino-like headers for attaching daughter boards, many pins exposed, as well as a UEXT connector. This connector is a custom pin-out bus and is used to attach sensor breakouts sold by the manufacturer.

3.2 STM32L4 Discovery kit IoT

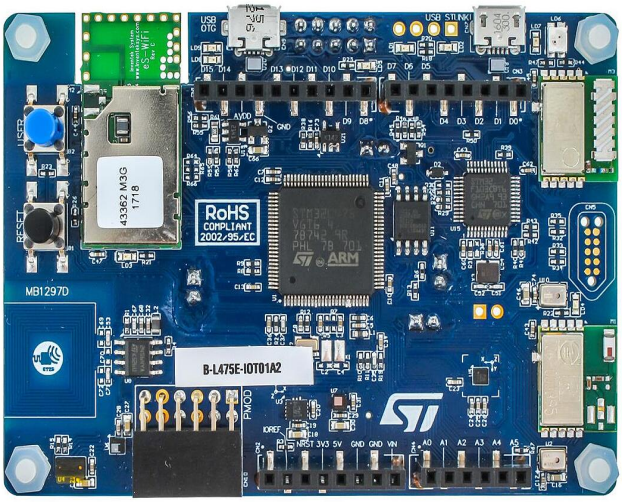

Figure 2: STM32L4 Discovery kit IoT

The STM32L4 Discovery kit IoT (B-L475E-IOT01A) evaluation board is a ready to use IoT kit provided by ST Microelectronics with on-board integrated sensors.

The STM32L4 Discovery kit IoT enables a wide diversity of applications by exploiting low-power communication, multiway sensing and ARM Cortex M4 core-based STM32L4 Series features. The support for Arduino and PMOD connectivity provides expansion capabilities with a large choice of specialized add-on boards making it suitable for micro-ROS developments targeting IoT applications.

This board features a STM32L475E MCU with 1 MB of Flash memory and 128 KB of RAM. In addition to the MCU peripherals, the board includes:

- 64 Mb external SPI Flash memory
- Bluetooth V4.1 module (SPBTLE-RF)
- 915 MHz low-power RF module (SPSGRF-915)
- 802.11 b/g/n module (ISM43362-M3G-L44)
- NFC tag based on M24SR with printed antenna
- 2 digital microphones (MP34DT01)
- relative humidity and temperature digital sensor (HTS221)
- 3-axis magnetometer (LIS3MDL)
- 3-axis accelerometer and gyroscope (LSM6DSL)
- digital barometer (LPS22HB)
- Time-of-Flight and gesture-detection sensor (VL53L0X)
- programmable push-buttons
- USB OTG FS with Micro-AB connector
- on-board ST-LINK/V2 debugger and programmer

3.3 Crazyflie 2.1 drone platform

Figure 3: Crazyflie 2.1 Drone

The Crazyflie 2.1 Drone is a versatile open-source flying development platform that only weighs 27 g and is less than 10 cm square. Crazyflie 2.1 is equipped with multiple inertial sensors and low-latency/long-range radio as well as Bluetooth LE.

This drone features a STM32F405 ARM Cortex-M4 MCU running up to 168 MHz with 1 MB of Flash and 192 KB of RAM. It also features the following sensors and coprocessors:

- nRF51822 radio and power management MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash)
- USB
- LiPo battery charger
- 8KB EEPROM
- 3-axis accelerometer and gyroscope (BMI088)
- pressure sensor (BMP388)
- headers with peripheral access: SPI, I2C, UART, 1-wire and GPIO

4 micro-ROS client RTOS support

The following RTOSes currently support the micro-ROS client in the Foxy Fitzroy release:

4.1 micro-ROS for Nuttx

Figure 4: NuttX logo

As stated in deliverable D2.3 *micro-ROS default RTOS release*, NuttX is a RTOS that emphasizes its compliance with standards (such as POSIX) and small footprint. It can be fit in 8 to 32 bit microcontrollers. The use of POSIX and ANSI standards, together with its mimicking UNIX APIs, makes it friendly to the developers that are used to Linux. The RTOS is licensed under BSD license and makes use of GNU toolchain. In order to obtain more information, please visit NuttX Github reposiroty.

micro-ROS port for Nuttx is based on NuttX 7.26 release and takes advantage of its C++ compliance and POSIX interface to bring support to the default hardware platform of micro-ROS.

micro-ROS sample applications for NuttX can be found in the Nuttx Apps repository

- Main website: https://nuttx.apache.org/
- Documentation: https://cwiki.apache.org/confluence/display/NUTTX/Nuttx

4.2 micro-ROS for Zephyr

Figure 5: Zephyr logo

According to its official web page:

> The Zephyr Project is a Linux Foundation hosted Collaboration Project. It's an open source collaborative effort uniting developers and users in building a best-in-class small, scalable, real-time operating system (RTOS) optimized for resource-constrained devices, across multiple architectures. The Zephyr Project is a neutral project where silicon vendors, OEMs, ODMs, ISVs, and OSVs can contribute technology to reduce costs and accelerate time to market for billions of connected embedded devices. The software is a perfect choice for simple connected sensors, LED wearables, modems, and small wireless gateways. Because Zephyr is modular and supports multiple architectures, developers can create a solution that meets their needs. As an open source project, the community evolves the project to support new hardware, developer tools, sensors, and device drivers. Improvements are frequently delivered to incorporate enhancements in security, device management capabilities, connectivity stacks, and file systems.

Zephyr RTOS is supported by the micro-ROS build system in its Foxy Fitzroy release and it has been used for some of the consortium demos as the preferred RTOS. The micro-ROS support for Zephyr uses its last stable version: Zephyr v2.3.0.

By default Zephyr provides micro-ROS with an abstraction layer for compatibility with all the supported boards that meets the minimum requirement criterion. Thanks to is wide sensors and drivers support it enables a high level of interaction between micro-ROS nodes and peripherals.

As a highlight, Zephyr provides an emulator for native POSIX platforms such as Linux. micro-ROS has been also ported to this emulator in order to provide a easy to use testing platform where micro-ROS applications can be ported directly from the emulator to the real hardware.

It is also remarcable that Zephyr strives for a functional safety certification, which would make it the first open-source RTOS with such a certification.

micro-ROS sample applications for Zephyr can be found in the Zephyr Apps repository

- Main website: https://www.zephyrproject.org/
- Documentation: https://docs.zephyrproject.org/2.3.0/

4.3 micro-ROS for FreeRTOS

Figure 6: FreeRTOS logo

FreeRTOS is a real time operative system distributed freely under the MIT open source license. It includes a kernel and a growing set of libraries suitable for use across all industry sectors. FreeRTOS comes with an emphasis on reliability and ease of use. The FreeRTOS kernel is composed by a basic set of RTOS components, while other modules such as the POSIX extensions or the network stack are provided separately.

micro-ROS support for FreeRTOS is platform-specific because there is no universal FreeRTOS build system that groups all the available platforms. Usually, FreeRTOS build is integrated into the application specific build system (as in the case of the Crazyflie 2.1 build system), but there are some vendor specific solutions. STM32CubeMX is a tool provided by ST Microelectronics that generates builds systems and HAL support for STM32 MCUs. micro-ROS for FreeRTOS port provides a fully customizable STM32CubeMX project integrated into the micro-ROS build system.

For micro-ROS, we make use of the POSIX extension.

micro-ROS sample applications for FreeRTOS can be found in the FreeRTOS Apps repository

- Main website: https://www.freertos.org/
- Documentation: https://www.freertos.org/Documentation/RTOS_book.html

4.4 micro-ROS for Linux

micro-ROS' build system provides a utility script for building micro-ROS on Linux machines. This port can be seen as a ROS 2 stack with the Micro XRCE-DDS middleware. Thanks to such tool, final users can develop micro-ROS applications under Linux before porting their code to a specific hardware platform.

micro-ROS sample applications for Linux can be found in the micro-ROS demos repository.

4.5 micro-ROS for Raspbian

micro-ROS' build system also provides a utility script for building micro-ROS for Raspbian operative systems and Raspberry Pi hardware. This port allows having micro-ROS applications running both in a single board computer and in specific MCU boards.

micro-ROS sample applications for Raspberry Pi and Raspbian can be found in the micro-ROS demos repository.

5 micro-ROS Agent support

As stated before, the micro-ROS build system provides utility scripts for building and installing the micro-ROS Agent and Micro XRCE-DDS Agent. This tool allows two kinds of building:

- A native Linux build and install where the final user can obtain a ready-to-use micro-ROS Agent on a computer.
- A crosscompilation tool that provides micro-ROS Agent binaries for Raspberry Pi and Raspbian. This utility enables the possibility of having micro-ROS nodes in MCU hardware and the micro-ROS Agent in a single board computer, easing the development of real use cases.

6 Summary of micro-ROS support

The following table shows the intercompatibilty between hardware platforms and different RTOSes. When support exists, the available transports are detailed. Note that USB stands for virtual serial ports over USB-CDC interfaces.

	NuttX	FreeRTOS	Zephyr
Olimex LTD STM32-E407	USB, UART, Network	UART, Network	USB, UART, Network
STM32L4 Discovery kit IoT	*Not supported*	*Not supported*	USB, UART, Network
Crazyflie 2.1 Drone	*Not supported*	Custom Radio Link	*Not supported*