D2.8

Bridge logic software release Initial Y1

Grant agreement no.	780785
Project acronym	OFERA
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D2.8
Deliverable name	Bridge logic software release
Date	June 2019
Dissemination level	public
Workpackage and task	2.3
Author	Juan Flores Muñoz and Iñigo Muguruza Goenaga (Acutronic Robotics)
Contributors	
Keywords	micro-ROS, robotics, ROS, microcontrollers, simulator, simulation
Abstract	micro-ROS hardware bridge initial release

Contents

1 Summary

The objective of this task is to provide a device which can work as hardware bridge between ROS 2 and micro-ROS. This device must have all the communications interfaces required, such as Ethernet, serial or 6LOWPAN. This hardware bridge will allow to publish the micro-ROS topics created by the MCU to the ROS2 network. For this purpose, we have selected the Raspberry Pi embedded board, as it is a popular platform extended among developers, and, additionally, as it meets the requirements and other wishes we consider important. In order to facilitate the usage of the bridge, we have written tutorials and developed a demonstration.

2 Acronyms and keywords

Acronym	Explanation
6LOWPAN	IPv6 over Low power Wireless Personal Area Networks
YAML	YAML Ain't Markup Language
SPI	Serial Peripheral Interface
ROS	Robot Operating System

3 Overview to results

4 Link to software repositories

Bridge set-up tools and execution instructions at deliverable handling:

- Git repository: https://github.com/micro-ROS/micro-ROS-bridge_RPI
  - Branch name: master
  - Commit Hash: 09d4f3c

Polly fork for micro-ROS cross-compilation:

- Git repository: https://github.com/micro-ROS/polly
  - Branch name: master
  - Commit Hash: 49d79c1

ROS2_RASPBIAN_TOOLS fork for micro-ROS and ROS2 cross-compilation:

- Git repository: https://github.com/micro-ROS/ros2_raspbian_tools
  - Branch name: master
  - Commit Hash: ca1910f

micro-ROS bridge example and validation resource list:

- Git repository: https://github.com/micro-ROS/micro-ROS_temperature_publisher_demo
  - Branch name: master
  - Commit Hash: f7446e4
- Git repository: https://github.com/micro-ROS/NuttX
  - Branch name: uros_demo_acutronics
  - Commit Hash: a54d2a6
- Git repository: https://github.com/micro-ROS/apps
  - Branch name: apps_demo_hih6130
  - Commit Hash: 3313a79
- Git repository: https://github.com/micro-ROS/docker
  - Branch name: uros_demo_acutronics
  - Commit Hash: 564f6a3

5 Requirements

The requirements that are derived into the hardware bridge are defined by micro-ROS features. Firstly, regarding communication means, as the micro-ROS use-cases contemplates the use of 6LOWPAN, serial and Ethernet-based communications, the hardware bridge requires to support them. Secondly, the software support also needs to be taken into consideration. The main piece of software to execute is the micro-ROS Agent. This element executes both, micro-ROS stack and a fully fledged ROS 2 distribution. Thus, it requires to have a good dependency support for those packages. This translates into having a good C++ support as well as other software packages, such as Python or YAML.

In addition to the aforementioned requirements, we desire to have a good adoption of this bridge. We have considered additional features community are susceptible of. This is a wish-list we have considered:

- Inexpensive: the hardware bridge should be inexpensive, so any interested party can acquire it.
- Popular: a platform that many people already owns and is widely accepted and supported.
- Portable: regular computers are not a good candidate for use-cases where the space is a constrain.
- Open-source: the project is mainly based on open-source software, so the hardware bridge it should also be open-source software based.
- Low-power consuming: in order to favor its use in various scenarios.

Taking into account the aforementioned requirements and wishes, we have decided to pick the Raspberry Pi 3 as the hardware bridge, as it has Linux support, is able to execute ROS and is a really popular platform in the embedded and robotics world already. Regarding communication means, it supports serial, Ethernet and 6LOWPAN. This last one is achievable turning on a kernel module and attaching a wireless breakout through expansion pins, using SPI interface.

6 Support and tooling development

Once we had chosen the hardware platform we wanted to use, we thought about how to support the software we need to execute. As the Raspberry Pi uses an ARM architecture, we need to compile the source-code for that specific architecture. For making it easy to maintain and to use, we decided to release a cross-compilation tool that compiles the source-code, obtaining self-contained workspaces.

6.1 Cross-compilation process

As the micro-ROS Agent and Client can take several hours to compile in a Raspberry Pi, we decided to find a solution that improves the compilation time and the user experience. Seeking for a solution, we found a cross-compilation tool, and used it as a starting point. The repository is called ros2_raspbian_tools and is a solution developed by Esteve Fernandez. This approach builds a Raspbian image inside a Docker container. The problem of this solution is that requires a high quantity of dependencies, that needs to be installed in the host computer. This could be a problem, because the dependency installation is challenging, time to time.

The ROS community already offers a tutorial to resolve cross-compilation dependency issues, that is placed here. In order to avoid host computer pollution in the cross-compilation process, we decided to use three nested Docker files. The working flow is the following:

- First, we run a Docker(1) with Ubuntu 18.04 and do the next procedures:
  - Download all the dependencies.
  - Create an additional Docker(2) which can be run by the host PC.
  - Add a converted Raspbian image to the previously created Docker(2).
- Docker(1) shares the files with the host PC, so the host PC will start Docker(2) adding all the dependencies that we obtained on Docker(1).
- Docker(2) will install the dependencies needed on the Raspbian file system, and will create the final Docker(3) which will build micro-ROS Agent, Client or ROS 2 workspaces.
- Docker(2) share the files with the host PC and starts Docker(3).
- Docker(3) will build micro-ROS Agent/Client or ROS 2 using an ARM tool-chain and will return self-contained compilation folders, ready to be deployed in a Raspberry Pi.

To make it easier, we developed a script which automatizes all the process. The user can choose what does she/he want to cross-compile by just adding the argument: `Agent`, `Client` or `ROS2` when running the script. This process can take around 30 minutes the first time that you execute it and, around 10 minutes, each of the times you want to cross-compile each resource. So we can see that we managed to cut down dramatically the compilation time.

All the aforementioned utilities are placed in the next repository:

micro-ROS-bridge_RPI, Commit 09d4f3c

6.2 Bridge set-up

In addition to the self-contained cross-compiled folders containing micro-ROS Agent and Client and ROS 2, it is required to install dependencies and activate 6LOWPAN kernel module in the Raspberry Pi Linux image. In order to install the dependencies, we have prepared `README` guide to follow, located here. For enabling the kernel module, we have created a script that activates the kernel module, located here

7 Tutorials and demonstrations

In order to validate the micro-ROS bridge, we have developed two demonstrations.

The first one, verifies the interoperability among Raspberry Pi and the Olimex STM32-E407 board, using 6LOWPAN communication means. These demos uses a breakout board that contains a MRF24J40 chip, manufactured by Microchip. Thanks to this chip, you can send and receive 6LOWPAN packages in both boards. He have placed the next files in the repository:

- A `README` file containing MRF24J40 breakout connection table, configuration commands for both, the Raspberry Pi and the Olimex board. Also step-by-step guide for sending and receiving the data packages.
- Raspberry Pi 6LOWPAN receiver example source-code.
- Raspberry Pi 6LOWPAN sender example source-code.

Note that the micro-ROS Agent still does not fully support 6LOWPAN communications, that is why we haven't test it out.

The second demonstration consists of a micro-ROS temperature publisher that is linked through serial communication to a Raspberry Pi that acts as micro-ROS to ROS 2 bridge. In this example, we make use of the Agent to make available the data of the temperature sensor in ROS 2. Using a regular ROS 2 running computer that is located in the same local network, the computer is able to fetch the sensor data.

We have uploaded a video showing how it works, you can see it here. All the information about how to launch the tutorial is placed in a separate GitHub repository. We have also placed a new web-page entry, so new users can see it.