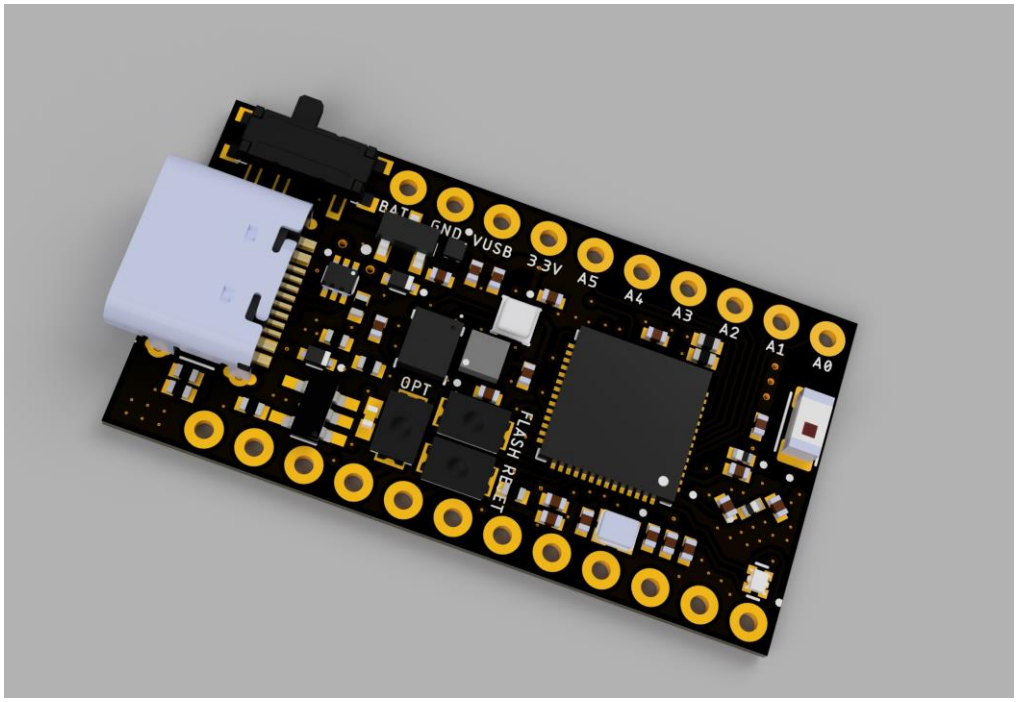

Responsive Internet of Things – Wi-Fii Wireless IMU Motion Sensor Platform

User Manual V1.0




**ATTENTION**

Please read this manual before using your IRCAM - EOWAVE R-IoT

The information contained in this manual has been carefully checked and we made every effort to ensure its quality. IRCAM & EOWAVE reserves the right to make changes and improvements to this manual and products referenced at any time without notice.

Please check your kit after receiving and before using it the first time to confirm if it contains all the ordered sensors, accessories and other components. Contact our support if there are any variations from your original order.

**EOWAVE**

email: contact@eowave.com

web: https://www.eowave.com/

Headquarters

6 Rue Marceau, 94200 Ivry-sur-Seine

France

tel.: +33 (0)1 45 15 41 95

**Institut de Recherche et Coordination Acoustique/Musique (IRCAM)**

Web: https://www.ircam.fr/

1 Place Igor Stravinsky 75004 Paris

France

tel: +33 1 44 78 48 43

# TABLE OF CONTENTS

1 Regulatory & Legal Information ..... 7
1.1 Disclaimer ..... 7
1.1 Guarantee of Quality & Warranty ..... 7
1.1.1 Warranty Voidance ..... 8
1.2 Contact & Support ..... 8
2 List of Acronyms ..... 9
3 Introduction ..... 10
3.1 Overview ..... 10
3.1.1 The ESP32-S3 MCU ..... 10
3.1.2 The Arduino IDE ..... 12
3.2 IRCAM/EOWAVE R-IoT (R-IoT v3) ..... 13
4 Hardware Description ..... 14
4.1 Port A ..... 15
4.2 Port B ..... 16
4.3 Battery Considerations ..... 16
4.4 Basic operations ..... 17
5 R-IoT Configuration & Setup ..... 18
5.1 Module default configuration mode ..... 18
5.2 Alternate configuration method ..... 19
5.3 Network Connection Details ..... 21
5.4 ID when using Multiple Devices ..... 21
6 Wi-Fi and Computer setup ..... 22
6.1 Change Computer IP ..... 22
6.2 Configure the Wi-Fi Access Point and Router ..... 25
6.2.1 How to find the router IP address. ..... 26
6.2.2 Access the Wi-Fi Access Point for the first time ..... 28
6.3 Local Network Conventions and Standards ..... 30
7 The OSC structure ..... 31
7.1 OSC Syntax ..... 31
7.1.1 Atomic Data Types ..... 31
7.1.2 OSC Packets ..... 31
7.1.3 OSC Messages ..... 32

7.1.4 OSC Bundles ..... 32
8 R-IoT Streaming, Sensor Fusion and Analysis ..... 33
8.1 Sensor Fusion ..... 33
8.2 R-IoT Code Repository ..... 33
8.3 Receive Sensors Data from the Module ..... 33
8.4 Max Abstractions & patches for the R-IoT v3 ..... 36
9 Programming the R-IoT ..... 37
9.1 Install the Arduino IDE ..... 37
9.2 Add the ESP32-S3 toolchain ..... 37
9.3 Customize the toolchain ..... 37
9.4 Install additional libraries ..... 38
9.5 Use the Arduino IDE ..... 38
9.6 Flashing the Firmware ..... 41
10 Advanced configuration ..... 43
10.1 Configuration file and parameters ..... 43
10.2 Serial commands ..... 49
10.3 IP management & local DNS ..... 51
10.3.1 Automatic IP handling & DHCP addresses reservation ..... 52
10.3.2 Multicast DNS & host naming ..... 55
11 Firmware update methods ..... 56
11.1 USB standard update ..... 56
11.2 Arduino IDE update ..... 57
11.3 Wireless update over Wi-Fi ..... 57
11.4 Checking the firmware version ..... 58
12. Calibration ..... 59
12.1 Accelerometers & Gyroscopes calibration ..... 59
12.2 Magnetometers calibration ..... 60
13. Advanced wiring ..... 65
13.1 Battery ..... 65
13.2 External tactile switch ..... 66
14. Glossary ..... 67
15. Bibliography ..... 69

# LIST OF FIGURES

Figure 3-1: Functional Block Diagram ..... 11
Figure 3-2: Arduino IDE interface ..... 12
Figure 4-1: Top View ..... 14
Figure 4-2: Bottom View ..... 14
Figure 4-3: Port A Pinout detail ..... 15
Figure 4-4: Port B Pinout detail ..... 16
Figure 5-1: Configuration file config.txt ..... 18
Figure 5-2: AP Mode details ..... 19
Figure 5-3: R-IoT Wi-Fi configuration access point example ..... 19
Figure 5-4: Configuration HTML page ..... 20
Figure 6-1: Change computer IP on MacOS ..... 22
Figure 6-2: Change IP – Windows (Step1) ..... 23
Figure 6-3: Change IP - Windows (Step 2) ..... 23
Figure 6-4: Change IP (Step 3) ..... 24
Figure 6-5: Change IP (Step 4) ..... 24
Figure 6-6: Finding IP (Step1) ..... 26
Figure 6-7: Finding IP (Step 2) ..... 26
Figure 6-8: Finding IP (Step 3) ..... 27
Figure 6-9: TP-link MR3020 3G/Wi-Fi router ..... 28
Figure 6-10: AP router Configuration ..... 28
Figure 6-11: AP router Configuration, SSID setup ..... 29
Figure 8-1: Max/MSP file preferences & paths ..... 34
Figure 8-1: riot-v3 abstraction and its parameters ..... 34
Figure 8-2: Inside the Max/MSP R-IoT v3 sensor receive abstraction ..... 35
Figure 8-2: Abstraction analysis-example.maxpat ..... 36
Figure 9-1: Arduino Blink Code example ..... 39
Figure 9-2: Selection of the Board type & compilation platform ..... 40
Figure 9-3: USB Communication Port configuration ..... 41
Figure 9-4: Flashing firmware ..... 42
Figure 10-1: Board orientations and axis ..... 47
Figure 10-2: Smartphone natural orientation as per W3C ..... 48
Figure 10-3: Serial Terminal / Console in the Arduino IDE ..... 49
Figure 10-4: Serial plotter in the Arduino IDE ..... 51
Figure 10-5: Successful network ping of a R-IoT module ..... 52
Figure 10-6: MAC address log of a R-IoT module ..... 53
Figure 10-7: DHCP address reservation in a TP-link Wi-Fi access point ..... 54
Figure 10-8: riot-v3 abstraction instantiated in a Max/MSP patch ..... 54
Figure 10-9: Max/MSP riot-v3 abstraction with bi-directional communication in place ..... 55
Figure 10-10: Accessing the R-IoT configuration webpage using mDNS ..... 56
Figure 11-1: Accessing the R-IoT update webpage using mDNS host naming ..... 57
Figure 12-1: Accelerometer & Gyroscope calibration in Max/MSP ..... 59
Figure 12-2: Saving Accelerometer & Gyroscopes calibration ..... 60
Figure 12-3: 3D raw magnetometers data feat. hard and soft iron effects ..... 61
Figure 12-4: Magnetometer calibration visualization window ..... 61
Figure 12-5: Magnetometer hard-iron calibration completed ..... 62
Figure 12-6: 8-figure motion pattern ..... 62
Figure 12-7: Magnetometer full calibration completed ..... 63
Figure 12-8: Magnetometer successful calibration with centered ellipsoids ..... 63

Figure 13-1: Protected Lithium-Polymer battery ..... 65
Figure 13-2: Battery wire management & direct soldering to the R-IoT ..... 65
Figure 13-3: Tactile switch attached to GPIO #41 ..... 66

# 1 Regulatory & Legal Information

## 1.1 Disclaimer

EOWAVE/IRCAM R-IoT devices are intended for use in life musical applications, science education and research applications only; they are not medical devices, nor medical software solutions, nor are they intended for medical diagnosis, cure, mitigation, treatment or prevention of disease and is provided to you "as is".

We expressly disclaim any liability whatsoever for any direct, indirect, consequential, incidental or special damages, including, without limitation, lost revenues, lost profits, losses resulting from business interruption or loss of data, regardless of the form of action or legal theory under which the liability may be asserted, even if advised of the possibility of such damages.

## 1.1 Guarantee of Quality & Warranty

The IRCAM/EOWAVE R-IoT acquisition system has a two-year warranty from the date of purchase. R-IoT sensors have three-month warranty from the date of purchase. PLUX guarantees that the system, sensors and accessories will be free from material or manufacturing defects for the mentioned time periods following date of purchase.

If PLUX receives a notification of any such defects within the warranty period, it will repair or replace with the same unit/model, any products with proven defects at no cost to the client. During the repair period EOWAVE promises to provide a temporary replacement under the same specification. Repairs will be carried out at EOWAVE's premises after the equipment has been received.

### 1.1.1 Warranty Voidance

Usage of the device that is not in accordance with the handling instructions indicated in the manual or use with accessories other than those manufactured by EOWAVE will invalidate the warranty of your devices.

Be careful when connecting your IRCAM/EOWAVE R-IoT devices, sensors and/or accessories to any third party device including the usage of the third party connection components that are available for IRCAM/EOWAVE R-IoT systems as the usage of these components will void the electrical warranty of your device and sensors and, if not indicated otherwise, the warranty of the third party system you're connecting to the device. Check the electrical specifications of both systems you want to connect to prevent any damage to the user(s) or the systems.

## 1.2 Contact & Support

Contact us if you're experiencing any problems that cannot be solved with the information given in the IRCAM/EOWAVE R-IoT or manual. We'll get back to you as soon as possible to find the best solution for your problem.

Please send us an e-mail with precise information about the error occurrence, device configuration, and, if possible, screenshots of the problem to contact@eowave.com. mailto:support@plux.info

# 2 List of Acronyms

**AP** – Access Point

**DHCP** - Dynamic Host Configuration Protocol

**I2C** - Inter-Integrated Circuit

**IC** - Integrated Circuit

**IDE** - Integrated Development Environment

**IP** – Internet Protocol

**IMU** - Inertial Measurement Unit

**IoT** – Internet of Things

**OSC** – OpenSound Control (protocol)

**SPI** – Serial Peripheral Interface

**SSID** - Service Set Identifier

**TCP/IP** - Internet Protocol Suite

**UART** - Universal Asynchronous Receiver/Transmitter

**UDP** - User Datagram Protocol

**UPnP** – Universal Plug and Play

**WAN** – Wide Area Network

**WISH** – Wireless Intelligent Stream Handling

**WPS** – Wi-Fi Protected Setup

# 3 Introduction

The IRCAM/EOWAVE R-IoT module is the 8th generation of IRCAM's wireless sensor digitizing unit, an essential tool linking motion sensing, gestural recognition and Live Performance Arts.

Since 2001, IRCAM aimed to provide a low latency, high data-rate & resolution and stage compatible devices, capable of streaming gestural information to the computer from multiple performers.

The IRCAM/EOWAVE R-IoT embeds a 9-axis digital IMU sensor (LSM6D+LIS3) featuring a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer, allowing for instance the onboard computation of the absolute orientation of the module in space. The sensor is attached to the SPI port to sample the 16-bit motion data at high speed. In addition, this new generation embeds a digital barometer sensor (BMP390) that allows for measuring absolute pressure and relative altitude.

The core of the IRCAM/EOWAVE R-IoT module is based upon the ESP32-S3 [1] chip from Espressif and it is compatible with the Arduino framework, API and IDE [2].

## 3.1 Overview

### 3.1.1 The ESP32-S3 MCU

Created for the Internet of Things (IoT), the ESP32-S3 device is a wireless MCU that integrates two high-performance RISC-V cores allowing developers to build an entire application with a single IC. With on-chip Wi-Fi, BLE, internet and robust security protocols, no prior Wi-Fi experience is needed for faster development.

The ESP32-S3 features native USB support as well as an Arduino core and toolchain provided and maintained by Espressif which allows for easy programming of custom firmware and applications with many existing open source libraries.

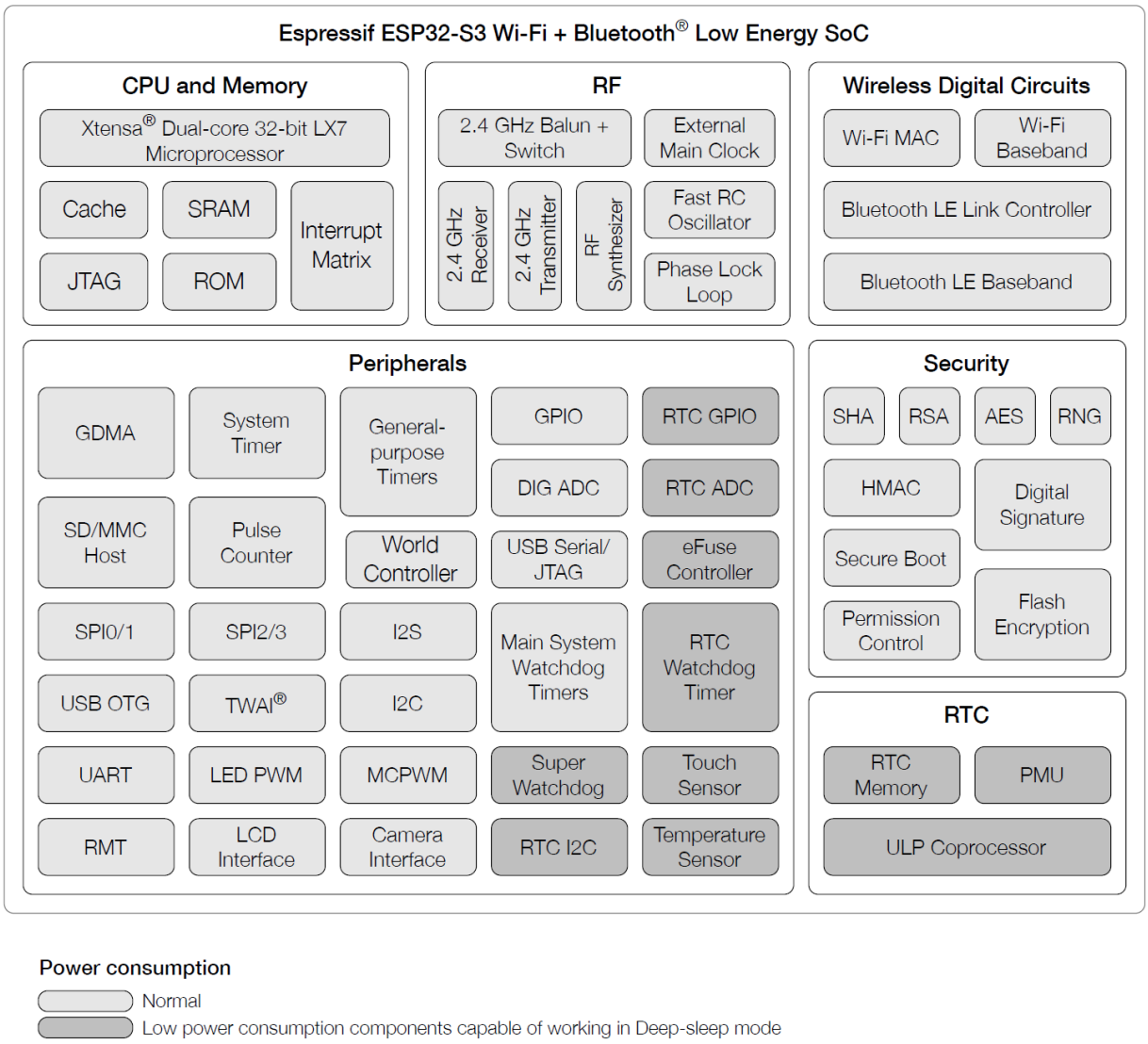



*Figure 3-1: Functional Block Diagram*

### 3.1.2 The Arduino IDE

Arduino is an open-source platform for electronics prototyping that bring the Wiring [3] to a very large number of boards said to be Arduino-compatible. Arduino IDE is cross platform and supported on Mac OS, Windows and Linux.

The foundation of Arduino is the Wiring framework that was developed by Hernando Barragan That was thoughtfully created with designers and artists in mind to encourage a community where both beginners and experts from around the world share ideas, knowledge and their collective experience. It rose as the Arduino ecosystem under the work of Massimo Banzi and Al [4].

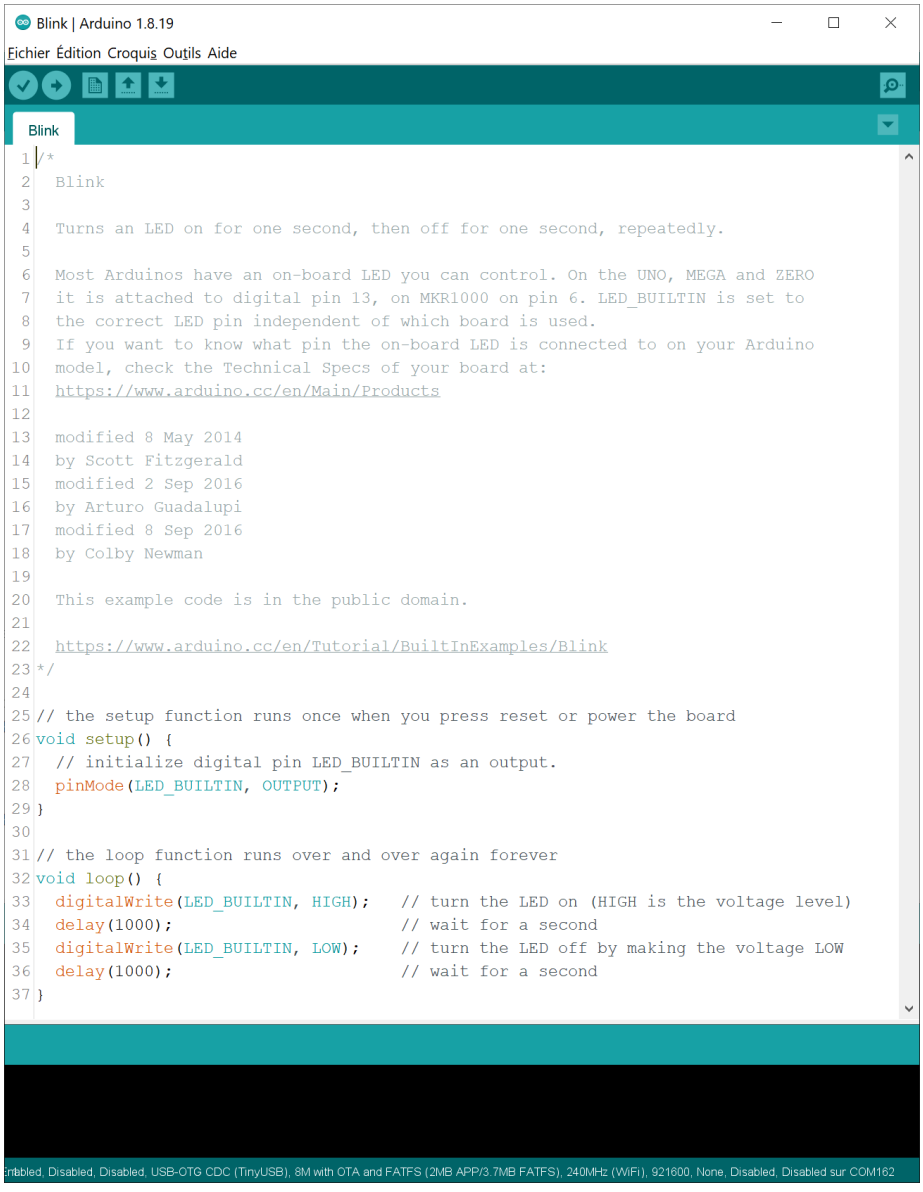



*Figure 3-2: Arduino IDE interface*

## 3.2 IRCAM/EOWAVE R-IoT (R-IoT v3)

The IRCAM/EOWAVE R-IoT is a complete redesign of the former R-IoT device from IRCAM's lab teams ISMM & PIP. It's aimed to refine a few aspects of the original design to match with other populations of users, and some new use cases too. Overall, it retains the original goal of being a modular Wi-Fi platform for gesture & inertial sensors acquisition transmitted as OSC message (see 14Glossary).

In addition, a new form factor of the module allows for soldering more additional sensors (analog or digital) or accessories (e.g. switch, LED, piezo) to expand its possibilities. Finally, an RGB Pixel (WS2812 compatible) was added for an enhanced visual interaction with the user, easing the boot & connection sequence (Wi-Fi connection, streaming, charging). To configure a network yourself or change the settings on the device, we advise you to read the following chapters.

The R-IoT v3 now features a sturdier, anchored USB-C connector that powers the onboard lithium-polymer battery changer. Thanks to its native USB support, the module enumerates itself as a USB serial port combined with a mass storage (flashdrive) of 3.7 MB that hosts file storage such as webserver HMTL files and configuration files that can be edited in place for quick configuration.

The R-IoT v3 exports a larger number of the available I/O's from the ESP32-S3 for matters of size form factor. It remains small enough to be installed anywhere: on the body, on an instrument or in an object and it can be adapted to various contexts.

An I2C bus is also exported allowing for a complete chain of accessories to be talked to, from LED controllers to additional sensors, I/O extenders or OLED displays and it also embeds all the features from the original design. While the R-IoT v3 is targeted to be a Wi-Fi / OSC sensor digitizer, it can also be used as a powerful BLE-MIDI device or as a USB powered (wired) MIDI device.

To ease the reading of this document, we will refer from now to the IRCAM/EOWAVE R-IoT as **R-IoT v3**.

# 4 Hardware Description

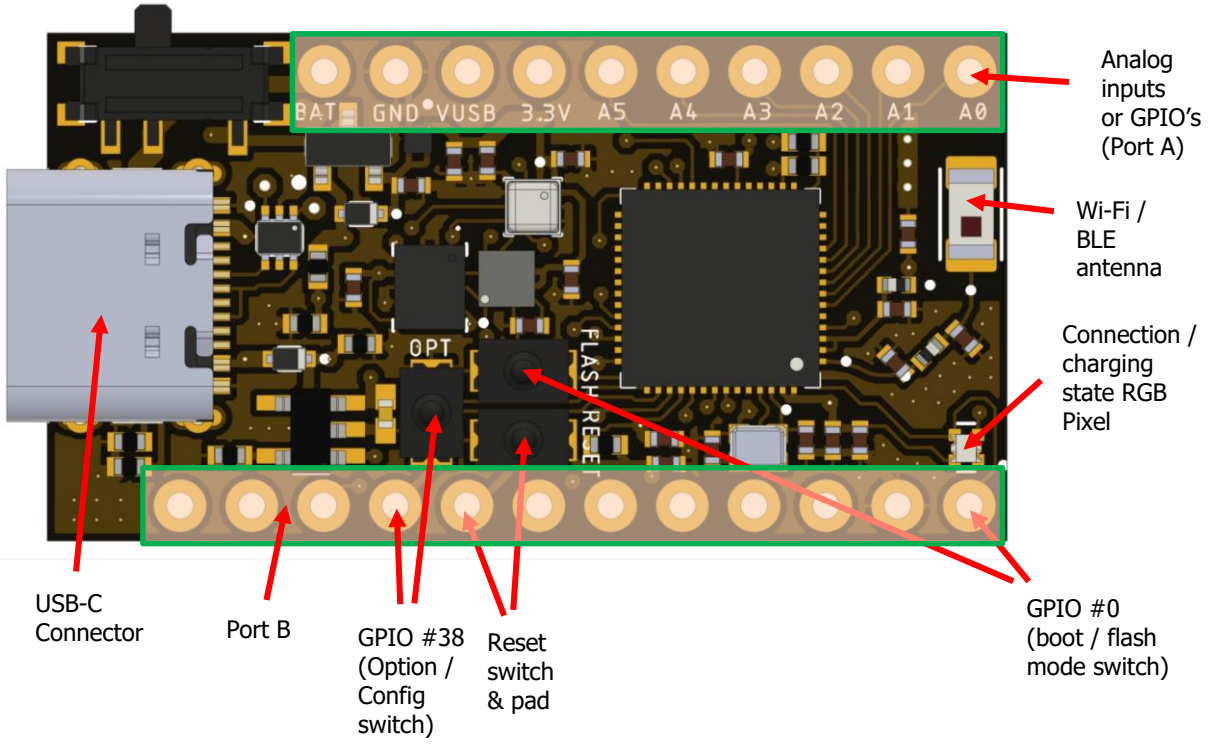



*Figure 4-1: Top View*

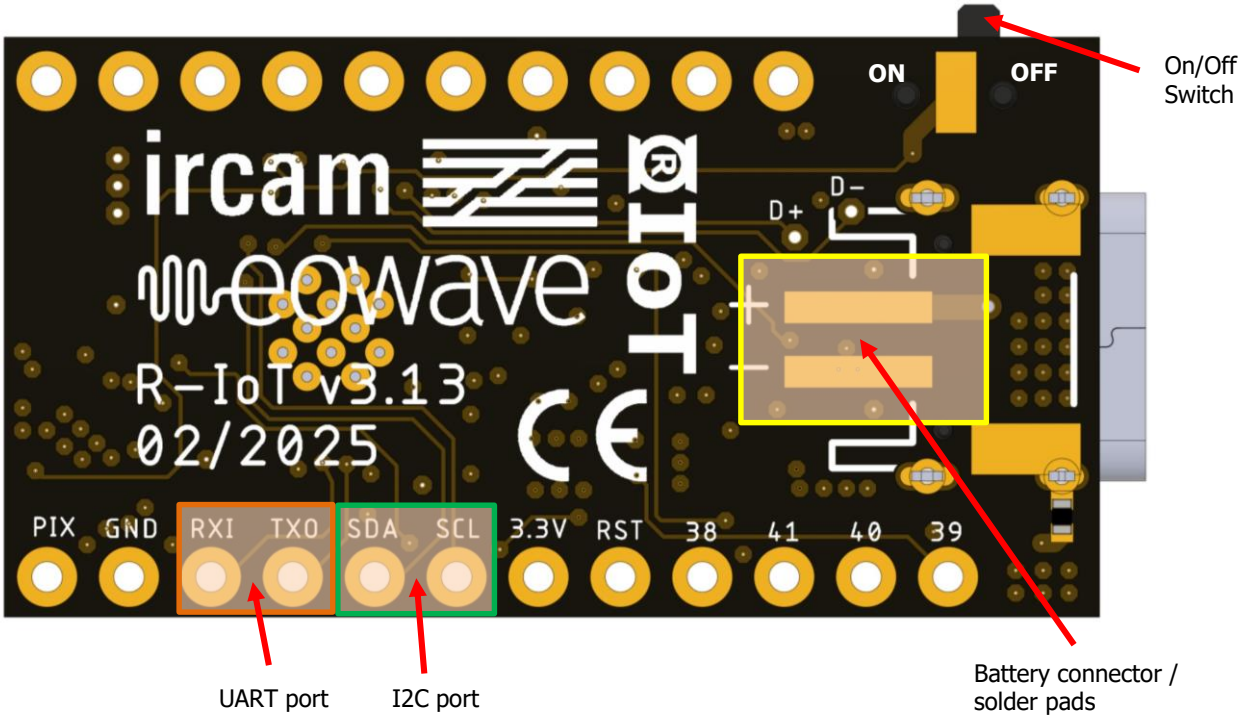



*Figure 4-2: Bottom View*

## 4.1 Port A


*Figure 4-3: Port A Pinout detail*

| Reference | Pin Alias | Signal Name (ESP32-S3) | Signal Description |
|---|---|---|---|
| **BAT** | BAT | - | Battery (+) |
| **GND** | GND | - | Common Ground & Battery (-) |
| **3.3V** | VDD | - | +3.3V / 700mA supply |
| **A5** | A5 | GPIO6 | 12 bit ADC input 5 or GPIO |
| **A4** | A4 | GPIO5 | 12 bit ADC input 4 or GPIO |
| **A3** | A3 | GPIO4 | 12 bit ADC input 3 or GPIO |
| **A2** | A2 | GPIO3 | 12 bit ADC input 2 or GPIO |
| **A1** | A1 | GPIO2 | 12 bit ADC input 1 or GPIO |
| **A0** | A0 | GPIO1 | 12 bit ADC input 0 or GPIO |

*Table 1 – Port A Pin Attributes*

## 4.2 Port B


*Figure 4-4: Port B Pinout detail*

| Reference | Pin Alias | Signal Name (ESP32-S3) | Signal Description |
|---|---|---|---|
| **PIX** | BOOT | GPIO0 | Pixel data & Booloader mode selection for flashing manually |
| **GND** | GND | - | Common Ground |
| **RXI** | RX | U0RXD | UART0 Rx pin |
| **TXO** | TX | U0TXD | UART0 Tx pin |
| **SDA** | SDA | GPIO8 | SDA (data) of I2C bus |
| **SCL** | SCL | GPIO9 | SDA (clock) of I2C bus |
| **3.3V** | VDD | - | +3.3V / 700mA supply |
| **RST** | RESET | CHIP_PU | Reset ESP32 |
| **38** | PIN_OPTION_SW | GPIO38 | Connected to the OPT switch |
| **41** | 41 | GPIO41 | Free GPIO |
| **40** | REMOTE_OUTPUT | GPIO40 | OSC controlled GPIO |
| **39** | 39 | GPIO39 | Free GPIO |

*Table 2 – Port B Pin Attributes*

## 4.3 Battery Considerations

- Polymer Lithium Ion Battery
- Weight: 6g
- Size: 60 x 17 x 3mm
- Output: 3.7V
- Capacity: 260mAh
- Connector (if used): 2-pin JST. Charge using R-IoT via Battery connector on the PCB, using the provided USB cable. Charges at 330 mA.

## 4.4 Basic operations

When battery powered, starting the R-IoT is achieved by simply sliding the tiny side on-off switch. The RGB pixel will light up from red to green and finally blue (or custom color) once connected to Wi-Fi and streaming data. Miniature tactile switches are usually not necessary, neither to flash the device or update its firmware. We'll see that in some case, the reset switch or the option switch have to be use. To press on the switch, use the end of your nail or a wooden tooth pick and avoid sharp objects.

The module features an onboard battery charger configured to charge at a current of 330 mA which would charge the provided standard battery (300 mAh) within one hour or less. The charger is always active as long as the USB connector is plugged to a +5V source, no matter the position of the on-off switch. Depending on the configuration, the R-IoT module will either stream data all the time while charging or only charge without streaming.

The RGB pixels reflects the charging (and streaming) state using the following color codes:

- Slow pulsing purple: charging
- Static green: battery fully charged

# 5 R-IoT Configuration & Setup

## 5.1 Module default configuration mode

This version simplifies configuration a lot compared to the previous version thanks to native USB support of the ESP32-S3 MCU. The firmware exposes some of the internal program memory a USB flash drive of 3.7 MB. The drive stores some of the files used for configuration and use of the module. The main configuration file config.txt can be edited in place on the computer after plugging the R-IoT USB connector (make sure to use a data cable and not just a charging cable). Once configuration is finished and file is saved, module can be restarted by pressing the reset switch or simply cycling power using the on-off switch.

The configuration file edition is the simplest and quickest way to setup your R-IoT module with only a few parameters to change to get you started.

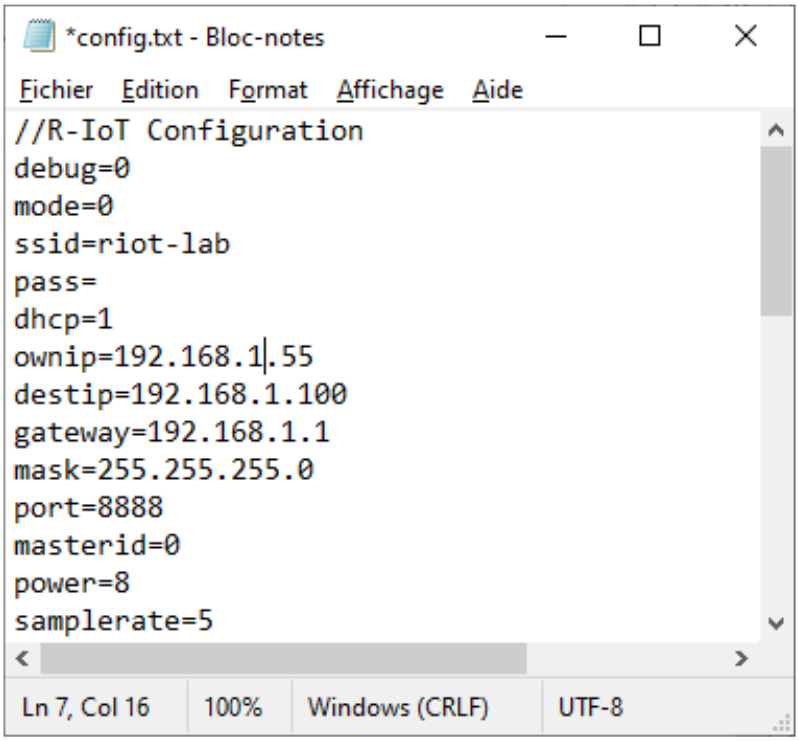



*Figure 5-1: Configuration file config.txt*

Basic configuration essentially consists in setting the name of the Wi-Fi network the module should join and the computer's destination address. The default SSID is "***riot-lab***" with no security / password (leave blank); for a quick install, the simplest is to configure your router and computer rather than edit the R-IoT configuration in first place.

By default, the module will be sending OSC data (*The OSC message structure is described in section 7*) to the **DEST IP** (destination IP address) **192.168.1.100** on port **8888**.

If configuration is correct, the module will connect to the Wi-Fi network with a red-green-blue color scheme and data will be streamed to the destination computer.

## 5.2 Alternate configuration method

The R-IoT v3 inherits from the previous configuration systems which doubles the file edition configuration method using an access point and a webserver. To enter this configuration mode, the option (OPT) switch must be pressed when applying power to the module. The RGB pixel will then blink red fast for 1.5 seconds before becoming static indicating the configuration access point has been created.

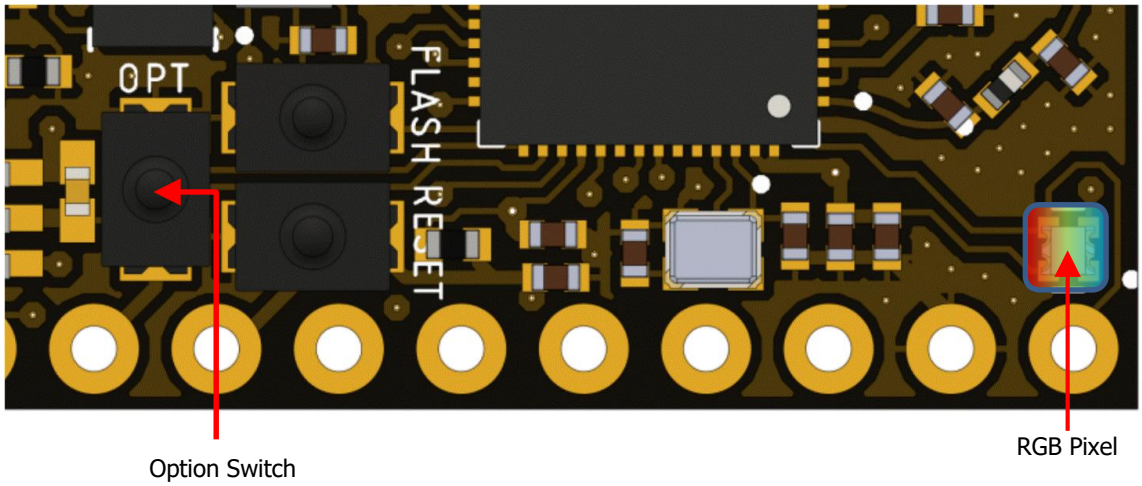



*Figure 5-2: AP Mode details*

Identify the created ssid in your list of Wi-Fi networks, which should be named **`riot-xx:yy`**, with the last 4 digits of the module's MAC address (Figure 5-3).

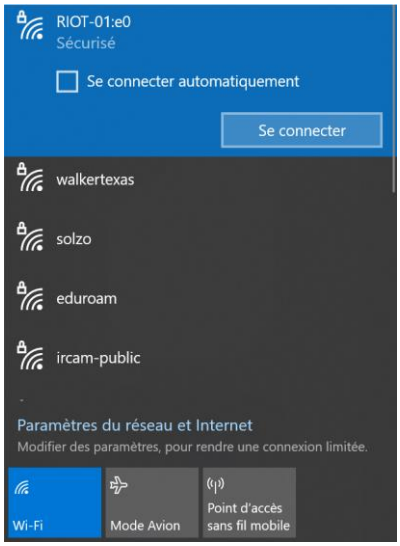



*Figure 5-3: R-IoT Wi-Fi configuration access point example*

To continue, please follow these steps.

1. Set your computer as host (Section 6) and lookup the Wi-Fi network proposed by the module;
2. Connect to the network (password is riot1234);
3. Open your web browser and open the URL **http://192.168.3.1** (which is the pre-configured IP);

After these steps you should be redirected to the web page hosted by the module, like the one on Figure 5-4, where you can configure its network behavior & parameters.

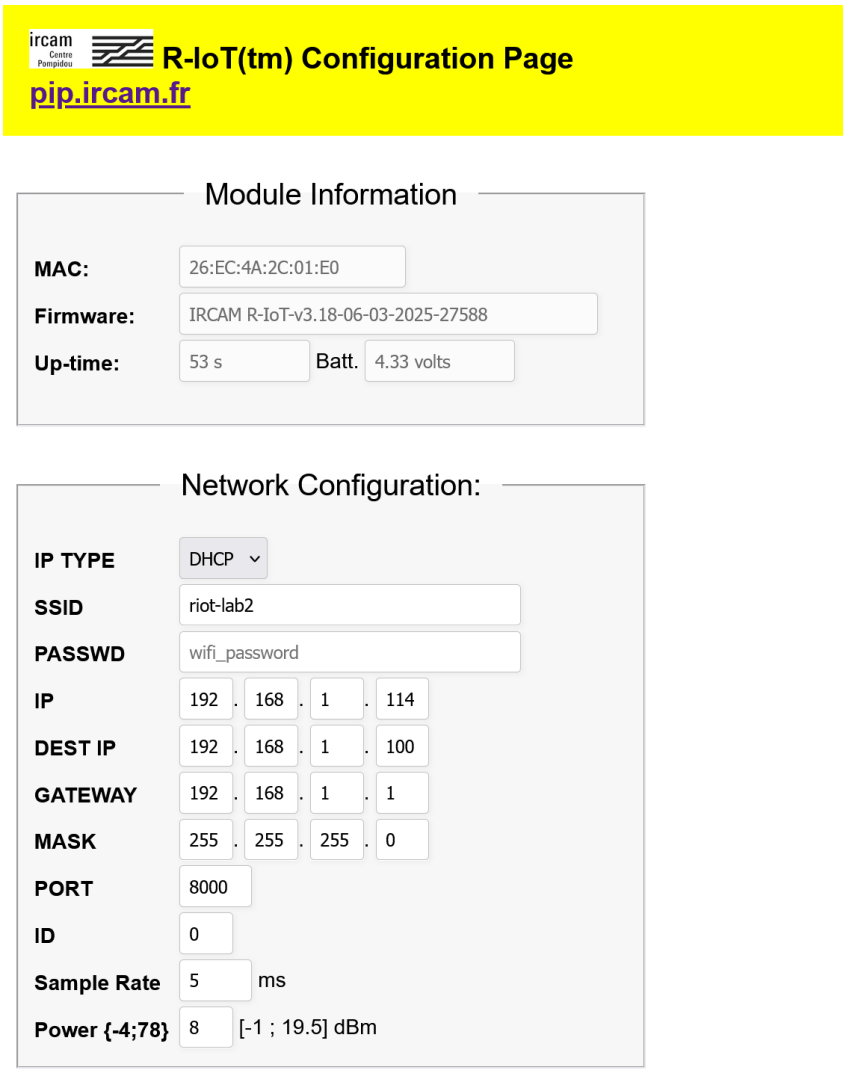



*Figure 5-4: Configuration HTML page*

Once configured, click the submit button at the bottom of the page to apply the new configuration. It's recommended to restart the module if network or IP settings have been modified. This configuration method comes very handy while live tuning a

module that is not next to the computer, when on stage for instance.

## 5.3 Network Connection Details

The R-IoT connects to the network name set in the SSID field above. It should match the network name set in your Wi-Fi access point (see Section 6).

Most of the default settings should work with your Wi-Fi infrastructure. We suggest keeping the R-IoT in station mode (AP is provided by the infrastructure router).

If some security is absolutely needed, WPA-2 can be enabled along with a password or passphrase, however security uses more bandwidth and can be delicate to configure for new users. If the goal is to prevent unwanted access to the local network during sensor. If some security is needed to prevent unwanted access to streaming and collection, we rather recommend to use the MAC filtering feature of your Wi-Fi access point.

In addition, the SSID beaconing can be disabled (also on the Access Point) to prevent finding the name of the network, therefore allowing connection only to parties who know the said SSID name.

In DHCP mode, there's no need to fill in the gateway and mask fields as those will be provided by their DHCP server.

The only other important parameters to be chosen are the recipient IP address (the IP of the computer receiving the data) and the UDP port.

## 5.4 ID when using Multiple Devices

The ID is used to ease the data routing when identifying the R-IoT in the software application. We suggest using both a port and ID based routing in order to ensure the proper identification of the data flow source. A simple numbering convention makes it easy to remember.

For example:

- R-IoT module #0 (ID0) uses port #8000
- R-IoT module #1 (ID1) uses port #8001
- R-IoT module #8 (ID8) uses port #8008

In the case the software application has limited flexibility for choosing the UDP port (or when there's a single UDP port available for all input flows), the ID is then used to separate and route the different modules.

Once the parameters are all set in the configuration page, click on submit button to save the changes in the non-volatile memory of the R-IoT module, which can be rebooted to apply the new settings (turn the device off and then on again or press the reset switch).

# 6 Wi-Fi and Computer setup

As explained above, the R-IoT can work as out-of-the-shelf product when it is acquired with a TP-Link Wi-Fi router which have been pre-configured to work together.

The R-IoT can operate on both a local network and over the internet if necessary. As already mentioned above, the R-IoT will be sending data to its DEST IP (destination IP address), identified in the field of the configuration web page (Figure 5-4) or the `destip` parameter of the configuration file, which is by default 192.168.1.100.

Therefore, a local network must be created using the AP router, and the proper destination address set on the computer.

## 6.1 Change Computer IP

On **MacOS**, go to the system preferences then network preferences. Select the Wi-Fi connection and connect to **Wi-Fi (riot-lab)** instead of your home Wi-Fi router; change the IP address method to "manually" and set it to 192.168.1.100. Set the network mask to 255.255.255.0. In order to not break your computer ISP configuration, additional network profiles can be created on MacOS, creating a dedicated one for using the R-IoT over a local network is a good practice.

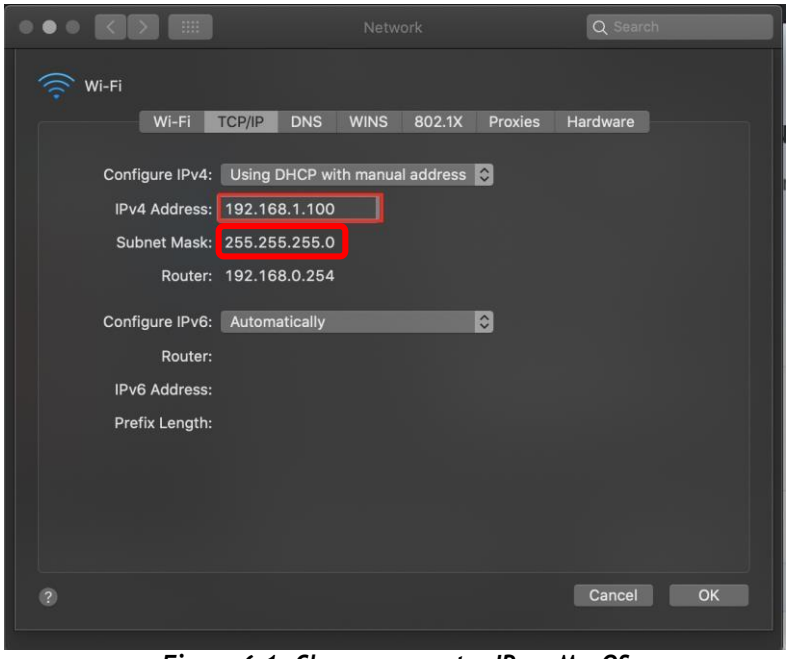



*Figure 6-1: Change computer IP on MacOS*

On **Windows**, the procedure is as follows:

1. Open the **Network and Sharing Center** and select **Wi-Fi**;

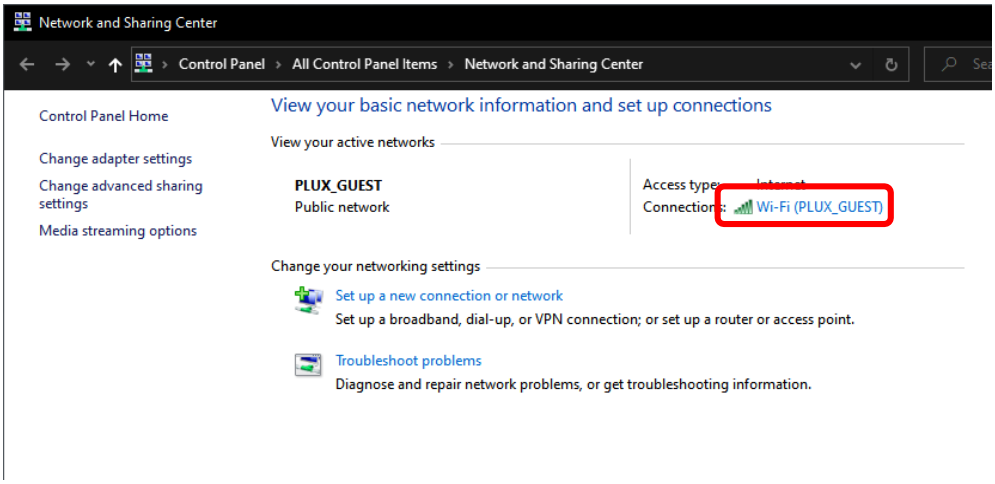



*Figure 6-2: Change IP - Windows (Step1)*

2. A window will open. Then you need to select **Properties;**

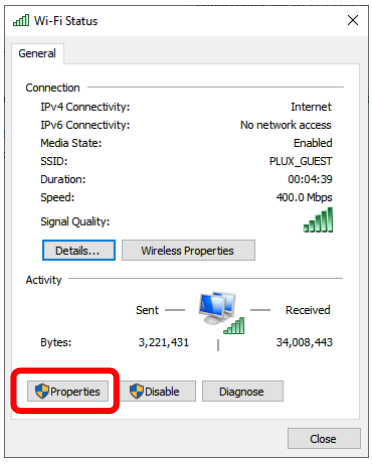



*Figure 6-3: Change IP - Windows (Step 2)*

3. Another window will show up. Now, select **Internet Protocol Version 4 (TCP/IPv4)** and go to **Properties**;

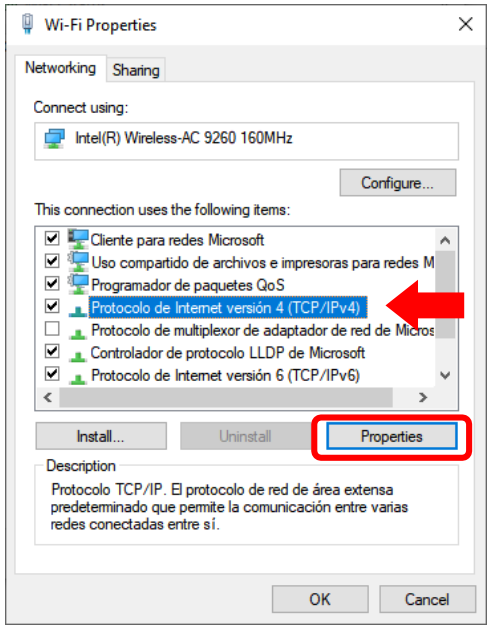



*Figure 6-4: Change IP (Step 3)*

4. The final step is to fill the final windows as shown in the next figure and click **OK** (the IP address should be the same as the DEST IP set in the configuration of the R-IoT);

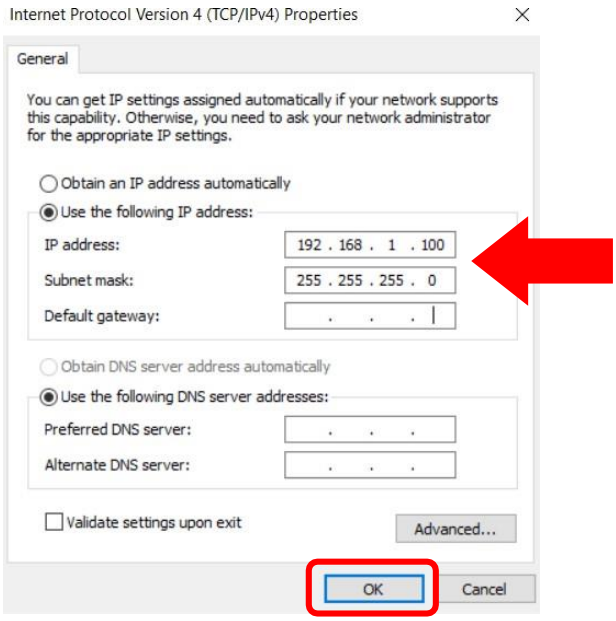



*Figure 6-5: Change IP (Step 4)*

5. After finishing using the R-IoT, you should change back this setting to **Obtain an IP address automatically to work with your regular Wi-Fi network and ISP or switch back to your regular ISP network profile**.

On both operating systems, you can also use network profiles that allows quick switching between your regular internet configuration (i.e. using DHCP for most use cases) and a dedicated sensor configuration to use the R-IoT over a local network (using a manually set IP address). MacOS has a profile system in its network preferences. On Windows you might have to install a free software to manage the profiles, like **NetsetMan**[1].

If you further need to change the destination IP, just make sure the R-IoT module is also configured to send to the right one. The selected IP address must match the network base address (192.168.1.XXX in our case) so that the OSC messages reach the recipient.

## 6.2 Configure the Wi-Fi Access Point and Router

Most Wi-Fi access points combine Wi-Fi network creation and routing features. They are usually called Wi-Fi routers. Some devices require an initial configuration to define their operating mode, select Access point in the base settings.

Take care of the following things before you configure your router to be an Access Point:

- Disable UPnP.
- Disable features that depend upon WAN: Virtual Server, SPI and Firewall, Application and Port Forward Rules, Access Control and Web Filters, Time, WISH and WPS (expert features can be further re-enabled to secure the Wi-Fi network if necessary).
- Change the IP address of the LAN on your network to an available address (the standard IP address scheme is 192.168.1.xxx with the access point IP being, most of the time either 192.168.1.1 or 192.168.1.254)

The essential information to know (provided by the user's manual of the device) is its IP address, so that you can connect to the device configuration page using your favorite browser.

[1] *For further information please visit https://www.netsetman.com/en/freeware.*

### 6.2.1 How to find the router IP address.

On Windows, the procedure is as follows:

1. *Open the Network and Sharing Center and select Wi-Fi;*

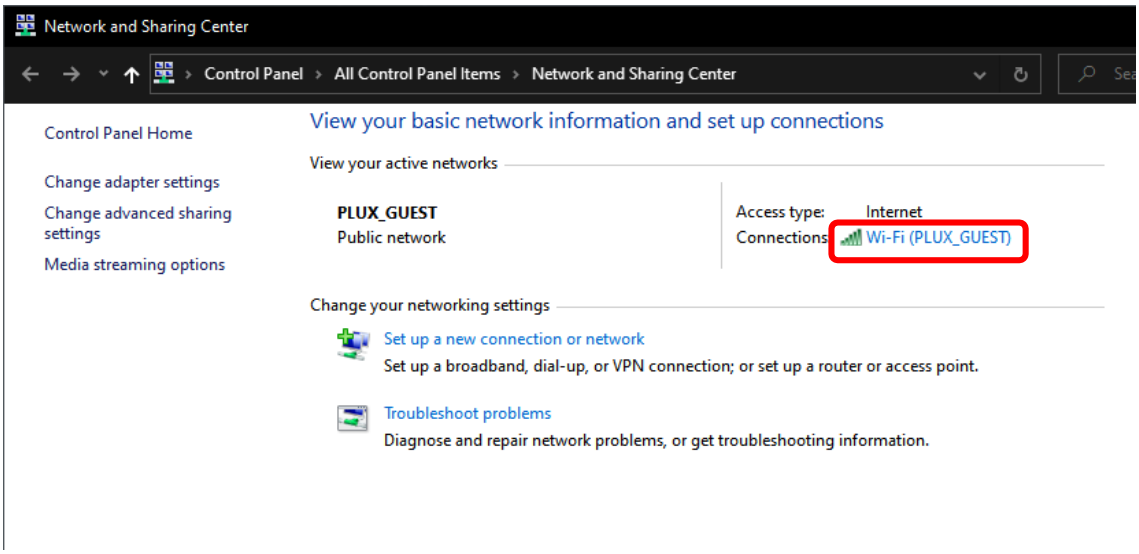



*Figure 6-6: Finding IP (Step1)*

2. *A window will open. Then you need to select **Details**;*

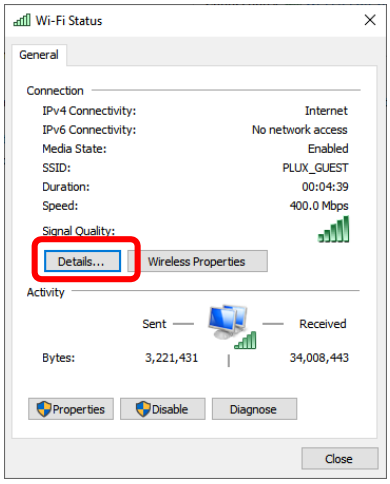



*Figure 6-7: Finding IP (Step 2)*

3. *The recently open window will show al the **Network Connection Details.** The IP address will be displayed.*

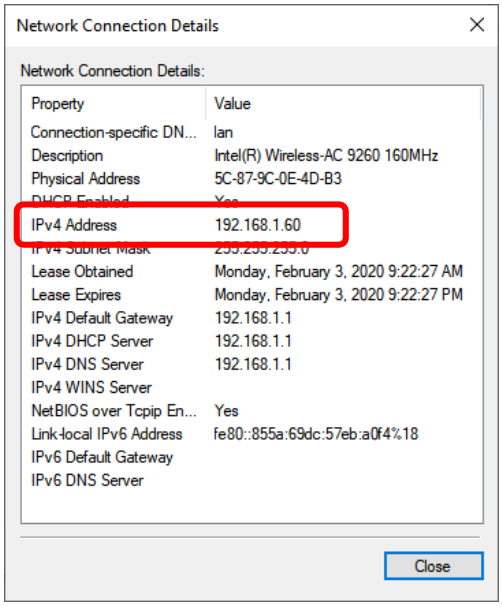



***Figure 6-8: Finding IP (Step 3)***

Each brand or manufacturer tend to have its own default configuration. Further configuration steps may be specific to each Access Point and, thus, you should refer to the user manual of the one you own. Nonetheless, we will explain the procedure for the one we commonly use.

### 6.2.2 Access the Wi-Fi Access Point for the first time


*Figure 6-9: TP-link MR3020 3G/Wi-Fi router*

The configuration of the access point can be reached by connecting the computer to it, and then go to tplinkWi-Fi.net. This will prompt you for a login page where you will need to insert the pre-configured password that is on the user's manual of the Access Point.

The default configuration of the R-IoT is to connect to a computer with the IP address 192.168.1.100 (and can be changed). The default DHCP settings of the router usually include this IP address, with a DHCP lease pit set from 192.168.1.100 up to 192.168.1.199, as shown on the next figure:

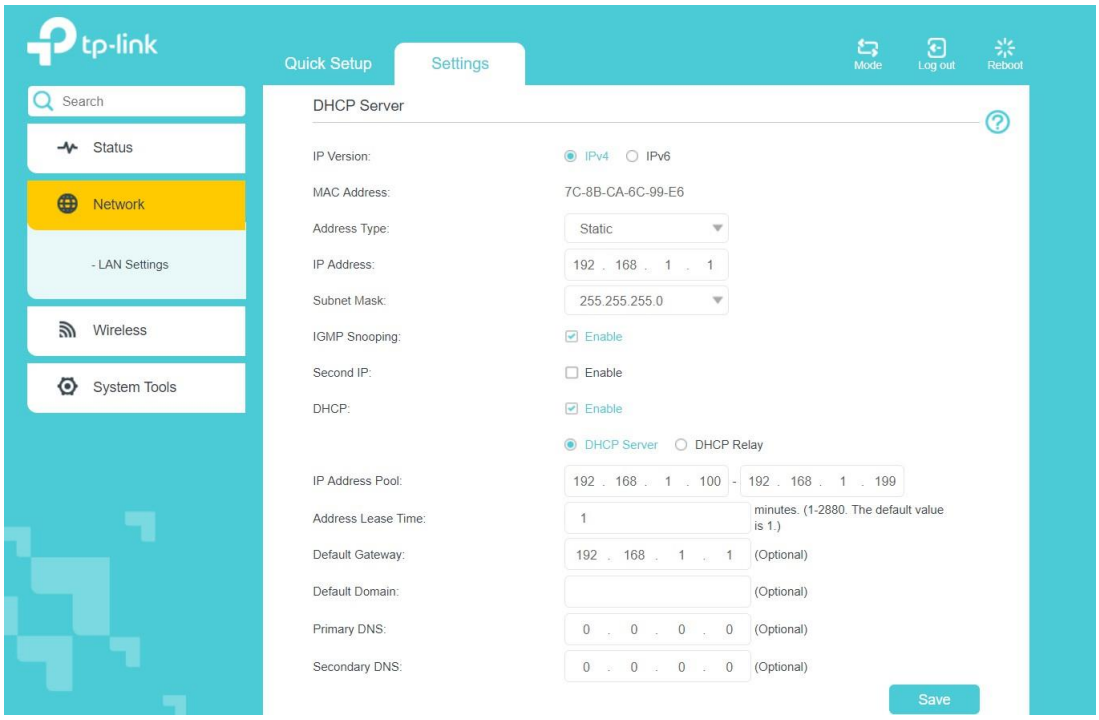



*Figure 6-10: AP router Configuration*

Tune the DHCP settings. Most of the time the DHCP will be active on addresses from 192.168.1.100 to 192.168.1.199. Change this from 192.168.1.**110** to 192.168.1.150, or anything to avoid having the default destination IP address (192.168.1.100) to be in the DHCP lease pit. Alternatively, you can leave it in the DHCP area and reserve the address based on the MAC address of your computer (see DHCP address reservation).

Finally, go to the Wireless settings and change the SSID of your network. If you are using the R-IoT defaults, use **riot-lab** as the network name.

Select a Wi-Fi channel that is possibly different than your neighbor. Finally, go to the security section and disable encryption (WPA-2 encryption is supported but not recommended on first use).

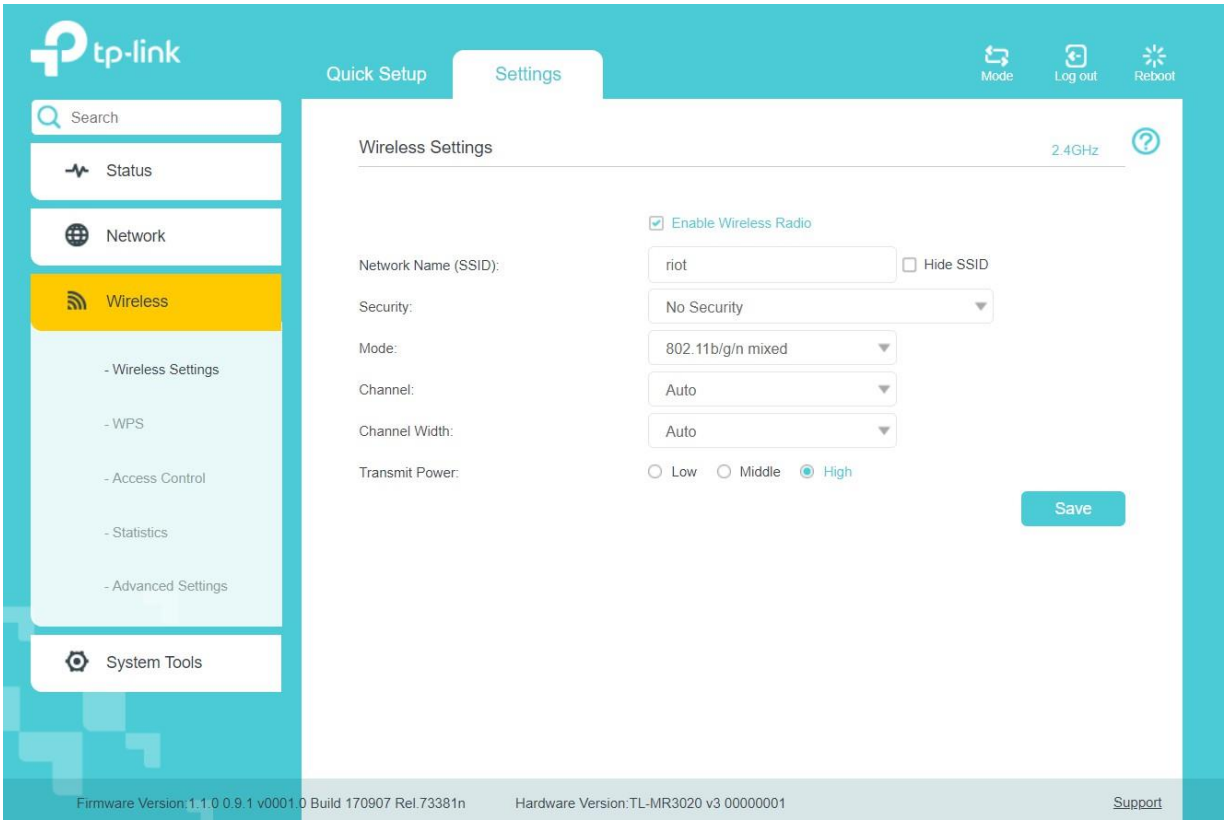



*Figure 6-11: AP router Configuration, SSID setup*

Once the router is configured with this 192.168.1.XXX address scheme and DHCP, the computer IP address can be set to 192.168.1.100 (as explained above), the default destination address the R-IoT module sends data to.

## 6.3 Local Network Conventions and Standards

This IP address configuration requires a few steps, but it is a onetime process. You've just created what is possibly your first local network (congratulations!) and switching between your Internet profile and your sensors profile will soon become some easy gym!

There's no absolute standard on what to use as your IP address and local network IP scheme. "Our" IRCAM standard using a 192.168.1.100 recipient IP address inherits from the hardware we designed back in 2004 using Linksys Wi-Fi routers which were on 192.168.1.XXX by default.

Moreover, another classic network numbering uses the scheme 10.0.0.XXX or similar. The important thing is to understand your 3 devices (the R-IoT, the Wi-Fi access point and the computer) must be on the same scheme to be able to "see each other". In addition, the R-IoT must know to which address it has to send data to (the computer IP).

Ideally you will label your AP with a stiff sticker to remember its IP and find an easy way to remember the ID and port configuration of your R-IoT module.

Once all set, the best practice is to keep the AP and the R-IoT modules paired and stored together to avoid any mismatch.

# 7 The OSC structure

For a better understanding of the next sections of this manual, it is necessary to dedicate a few lines to the structure of the OSC messages.

## 7.1 OSC Syntax

Open Sound Control (OSC) is an open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices [5] [6].

The syntax of OSC data its structured as follows:

### 7.1.1 Atomic Data Types

All OSC data is composed of the following fundamental data types:

- **int32:** A 32-bit big-endian two's complement integer.
- **OSC-timetag:** A 64-bit big-endian fixed-point time tag, semantics defined below
- **float32:** A 32-bit big-endian IEEE 754 floating point number.
- **OSC-string:** A sequence of non-null ASCII characters followed by a null, followed by 0-3 additional null characters to make the total number of bits a multiple of 32. (OSC-string examples) In this document, example OSC-strings will be written without the null characters, surrounded by double quotes.
- **OSC-blob:** An int32 size count, followed by that many 8-bit bytes of arbitrary binary data, followed by 0-3 additional zero bytes to make the total number of bits a multiple of 32.

  The size of every atomic data type in OSC is a multiple of 32 bits. This guarantees that if the beginning of a block of OSC data is 32-bit aligned, every number in the OSC data will be 32-bit aligned.

### 7.1.2 OSC Packets

The unit of transmission of OSC is an OSC Packet. Any application that sends OSC Packets is an *OSC Client*; any application that receives OSC Packets is an *OSC Server*.

An OSC packet consists of its contents, a contiguous block of binary data, and its size, the number of 8-bit bytes that comprise the contents. The size of an OSC packet is always a multiple of 4.

The underlying network that delivers an OSC packet is responsible for delivering both the contents and the size to the OSC application. An OSC packet can be naturally represented by a datagram by a network protocol such as UDP. In a stream-based protocol such as TCP, the stream should begin with an int32 giving the size of the first packet, followed by the contents of the first packet, followed by the size of the second packet, etc.

The contents of an OSC packet must be either an OSC Message or an OSC Bundle. The first byte of the packet's contents unambiguously distinguishes between these two

alternatives.

### 7.1.3 OSC Messages

An OSC message consists of an *OSC Address Pattern* followed by an *OSC Type Tag String* followed by zero or more OSC Arguments.

- **OSC Address Patterns:** An OSC Address Pattern is an OSC-string beginning with the character '/' (forward slash).
- **OSC Arguments:** A sequence of OSC Arguments is represented by a contiguous sequence of the binary representations of each argument.
- **OSC Type Tag String**[2]**:** An OSC Type Tag String is an OSC-string beginning with the character ',' (comma) followed by a sequence of characters corresponding exactly to the sequence of OSC Arguments in the given message. Each character after the comma is called an OSC Type Tag and represents the type of the corresponding OSC Argument. (The requirement for OSC Type Tag Strings to start with a comma makes it easier for the recipient of an OSC Message to determine whether that OSC Message is lacking an OSC Type Tag String.)

### 7.1.4 OSC Bundles

An OSC Bundle consists of the OSC-string "`#bundle`" followed by an OSC Time Tag, followed by zero or more OSC Bundle Elements. The OSC-timetag is a 64-bit fixed point time tag whose semantics are described below.

An *`OSC Bundle Element`* consists of its size and its contents. The size is an int32 representing the number of 8-bit bytes in the contents and will always be a multiple of 4. The contents are either an OSC Message or an OSC Bundle. To be more specific a bundle may contain bundles. The R-IoT v3 uses the OSC bundle feature to reduce the network load, yet proposing a clear, easily routable OSC naming structure. The OSC name space conforms with the W3C standard[3] for exporting IMU sensor data and will be described in section 8.

[2] For more detailed information about OSC Type String please visit: http://opensoundcontrol.org/spec-1_0

[3] https://www.w3.org/TR/accelerometer/

# 8 R-IoT Streaming, Sensor Fusion and Analysis

## 8.1 Sensor Fusion

Calibration of the sensors and the data fusion allowing for computing the quaternions and Euler angles are provided. The data fusion is implemented using the open-source code provided by Sebastian Madgwick [7] [8].

## 8.2 R-IoT Code Repository

Several code examples dedicated to the R-IoT platform are available on the IRCAM/EOWAVE GitHub repository https://github.com/ircam-ismm/riot-v3.

The repository contains the main firmware, which achieves several analyses on the sensor data and allows the configuration of the module (IP address, UDP port, module ID) via the configuration file, the web server or the USB serial port.

Other simpler examples show how to write dedicated code for the R-IoT platform for specific motion analysis for instance, using the motion bricks.

Being an Arduino compatible development platform, the R-IoT module role isn't bound to motion analysis or sensor data streaming. Using the additional I/Os and analog input and the Wi-Fi or BLE modem, the unit can be turned into an efficient Internet of Things (IoT) object, a miniature web server, a car alarm or a USB MIDI instrument.

## 8.3 Receive Sensors Data from the Module

Max/MSP or PureData software provide an easy way to acquire and visualize data from sensors thru OpenSound Control protocol.

The sensors data is exported in an OSC bundle which groups several OSC messages in a single UDP packet for enhanced parsing and time alignment. The messages are based upon the following syntax: **/riot/api-version/sensor-type <data list>**[4]. The data split follow the W3C sensor API as proposed here : https://www.w3.org/TR/orientation-event/

The R-IoT exports the following data sets within the bundle, in SI units:

- 3 axis accelerometer (1 float per axis) in m.s-2 (ie g x 9.81)
- 3 axis gyroscope {-35 ; +35} rad/s (ie ° / 180 x PI)
- 3 axis magnetometer {-400 ; +400} µT ( 100 µT = 1 Gauss)
- 3 different Board Temperature in °C
- Switch states (GPIO38 & GPIO41) {0 / 1}

[4] Current API version is v3

- Battery voltage + analog Inputs A0 and A1 {0 ; 3.3V}
- Normalized battery charge level {0;1} and charging state
- Quaternions {-1 ; 1}
- Euler Angles {-180 ; 180} °
- Magnetic Heading (compass) {0; 360} °
- Gravity Vector x, y, z {-1 ; 1}

Receiving sensors data in Max/MSP can be achieve by using the provided abstraction `riot-v3`. It contains all the data splitting and routing in a convenient way. It can be used as a data hub and entry point in your data collection patch or can be used as an OSC syntax parser base for your own application. The `riot-v3` abstraction and associated files are available from the https://github.com/ircam-ismm/riot-v3 github repository and should be installed in a folder of your choice which path must be added to the Max file Preference (Figure 8-1).

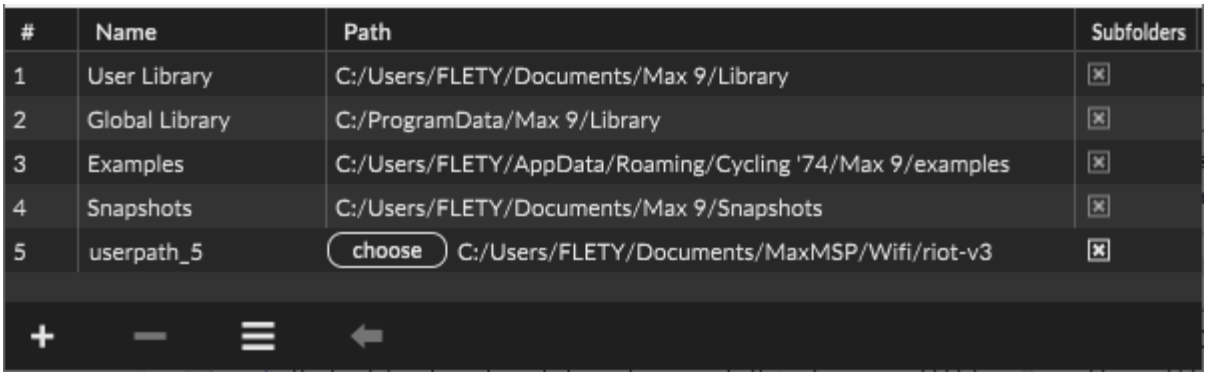



*Figure 8-1: Max/MSP file preferences & paths*

In a new patcher, instantiate the abstraction using 4 parameters, as per the module's configuration: ID, transmitting UDP/OSC port, receiving port and IP address (if known, see Section 10.3.1 for DHCP address reservation)


*Figure 8-2: riot-v3 abstraction and its parameters*

After instantiation in your Max/MSP patch, double click on the abstraction to reveal the data display and bargraphs. A successful connection of the R-IoT to the computer will be reflected by animated OSC data flowing. An alternative quick way to verify that the data is flowing correctly between the R-IoT and your computer is to use the OSC Data Monitor [9]:

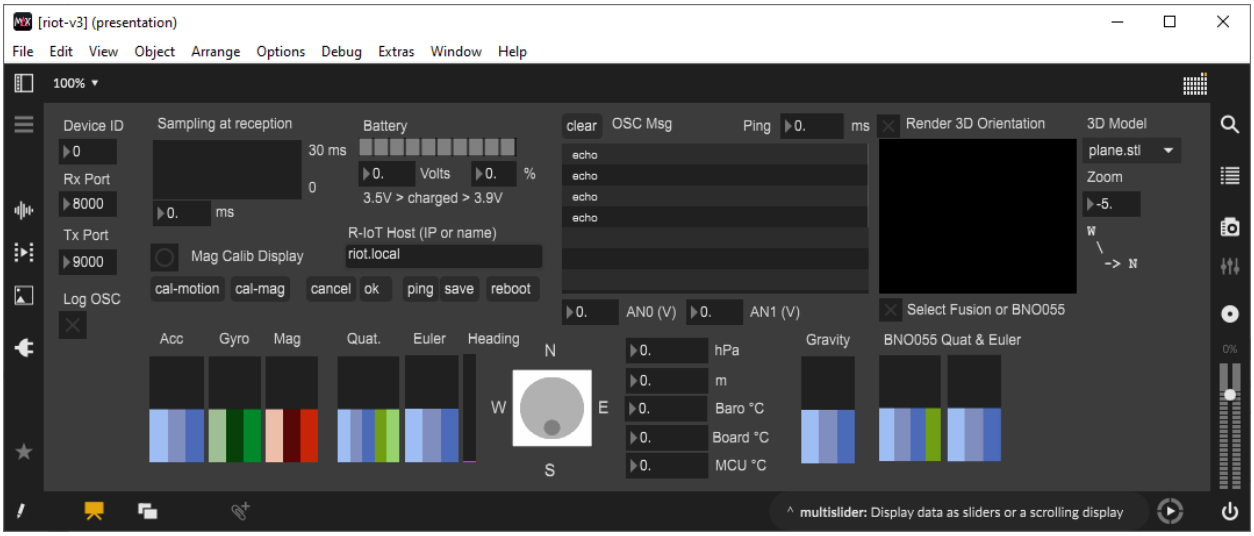



*Figure 8-3: Inside the Max/MSP R-IoT v3 sensor receive abstraction*

The first outlet of the abstraction contains the complete data structure of the module, minus the start of the OSC address routed from **`/riot/v3/<ID>`**.

## 8.4 Max Abstractions & `patches` for the R-IoT v3

Several abstractions and example patches for Max (Cycling'74) are provided in the GitHub https://github.com/ircam-ismm/riot-v3

For example, the following abstraction *analysis-example.maxpat* shows example of various small abstraction that can be used to transform the raw accelerometer and gyroscope in intensity parameters and detecting kicks. This abstraction uses the free Max library MuBu available here: http://forumnet.ircam.fr/product/mubu/

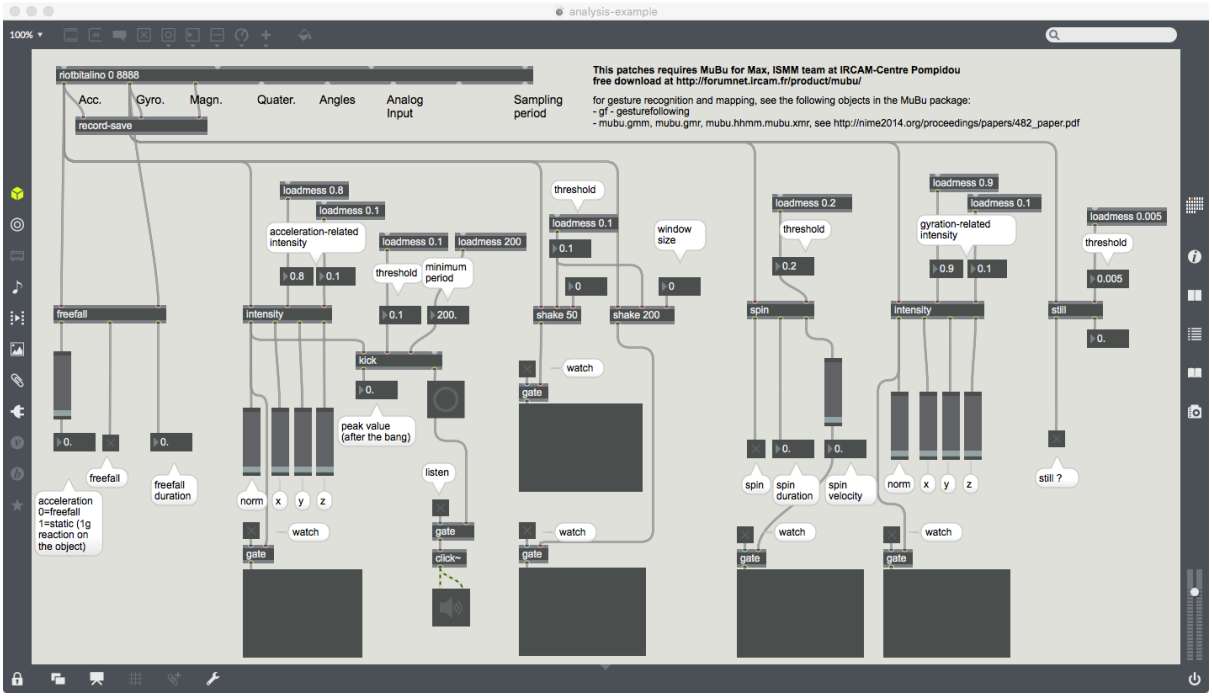



*Figure 8-4: Abstraction analysis-example.maxpat*

# 9 Programming the R-IoT

## 9.1 Install the Arduino IDE

1. Visit https://www.arduino.cc/ and look for the download section
2. Download and install the IDE (version 1.8.x or version 2.x)
3. On Windows, simply unzip the Arduino folder on your main hard drive (version 1.8.x) or use the EXE installer (version 2.x). On Mac, place the folder-program in the Application folder.

## 9.2 Add the ESP32-S3 toolchain

In the Arduino IDE, open the preferences and look for the "additional board manager" URL input. Add the following line:

https://espressif.github.io/arduino-esp32/package_esp32_index.json

Restart the IDE then go in Tools -> Board Type -> Board Manager and look for Espressif's ESP32 toolchain, then install version 3.1.1 (internet connection required)

## 9.3 Customize the toolchain

In order to compile the riot firmware, a small modification must be added to the tool chain for the flash partition.

On Windows:

- Open the location of /<User>/AppData/Local/Arduino15/packages and continue inside the esp32/hardware/3.1.1/tools/partition
- Add or replace **default_ffat_8MB.csv** (from the R-IoT code github repository)
- Edit boards.txt, locate **esp32s3.name=ESP32S3 Dev Module** then add the following partition item to have it in the board options menu of the IDE.
    - esp32s3.menu.PartitionScheme.8MB_ota_ffat=8M with OTA and FATFS (2MB APP/3.7MB FATFS)
    - esp32s3.menu.PartitionScheme.8MB_ota_ffat.build.partitions=default_ffat_8MB
    - esp32s3.menu.PartitionScheme.8MB_ota_ffat.upload.maximum_size=2097152

On Mac OS:

- Locate the ESP32 toolchain path which root path should be <user home>/Library/Arduino15
- Follow the same process as explained for Windows above

## 9.4 Install additional libraries

The R-IoT firmware uses a few external libraries to be compile. They can be installed from the Arduino IDE using the library manager, or visiting their respective repositories. Libraries get installed in the `/<user>/…/Arduino/Libraries` folder.

The firmware relies on the following third-party libraries:

- U8g2
- ESPmDNS
- MicroOscUdp

## 9.5 Use the Arduino IDE

Launch the Arduino IDE. A blank sketch appears. A sketch is composed of two essential functions, `setup()` and `loop()` which is equivalent to the main function in traditional C language. Arduino, uses C and C++, along with a basic API and set of classes to access the hardware in what became the Arduino standard.

https://docs.arduino.cc/language-reference/

The setup function, called prior the main loop, is used to configure the module hardware, default behaviors and everything that requires some initialization before executing the main program.

Once returning from the setup function, the loop function is called repetitively and is equivalent to any traditional endless program loop used on embedded platforms (a `while(1)` statement within the main function).

Below an example of blinking endlessly a GPIO:

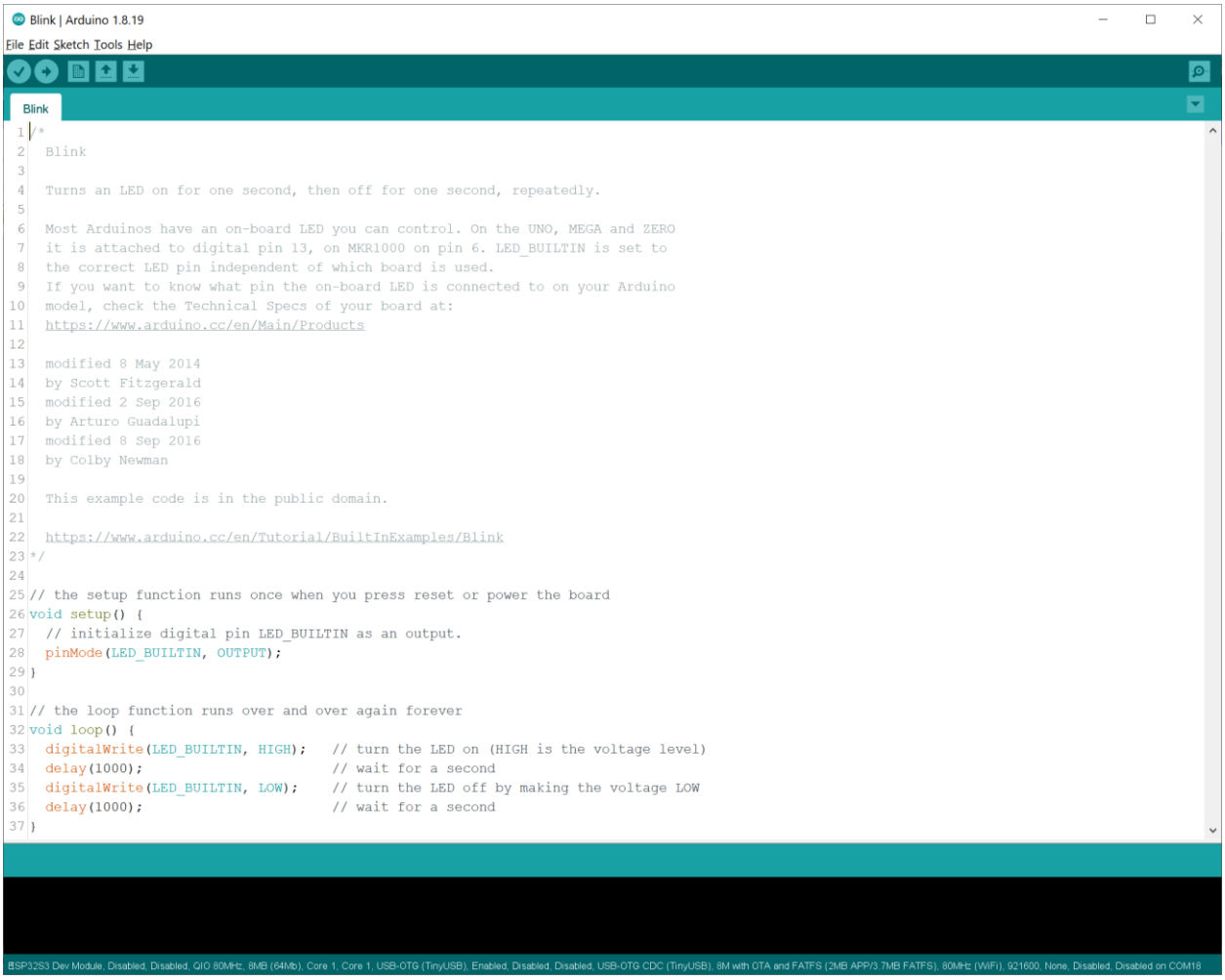



*Figure 9-1: Arduino Blink Code example*

Before compiling, select the proper target in the Tools -> Board menu and select the ESP32-S3 Dev Module.

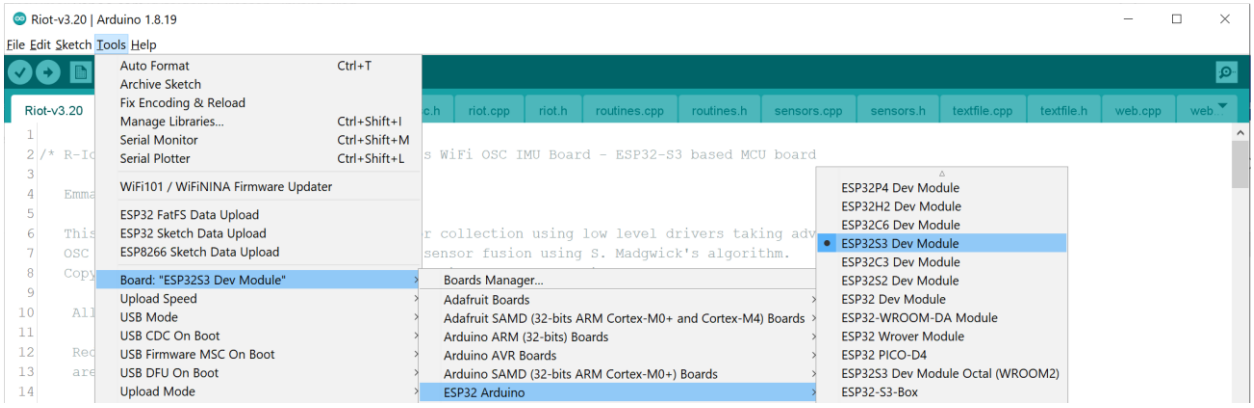



*Figure 9-2: Selection of the Board type & compilation platform*

Then in the Tools menu, select the following compilation options:

- USB Mode: USB-OTG / CDC on boot : enabled
- DFU on boot: disabled
- Upload Mode: USB-OTG (TinyUSB)
- CPU frequency: 80 MHz
- Flash Mode: QIO 80 MHz
- Flash Size: 8 MB / Partition : 8MB with FAT (2MB app / 3.7MB FFAT)

To compile the program, simply click on the left-most icon (tick).

## 9.6 Flashing the Firmware

Flashing the firmware is the uploading operation that transfers the result of the compilation of the code to the R-IoT ESP32 microcontroller. This is performed over USB. First plug the USB serial cable in a USB port then select the matching COM port in Arduino Tools->Serial port menu.

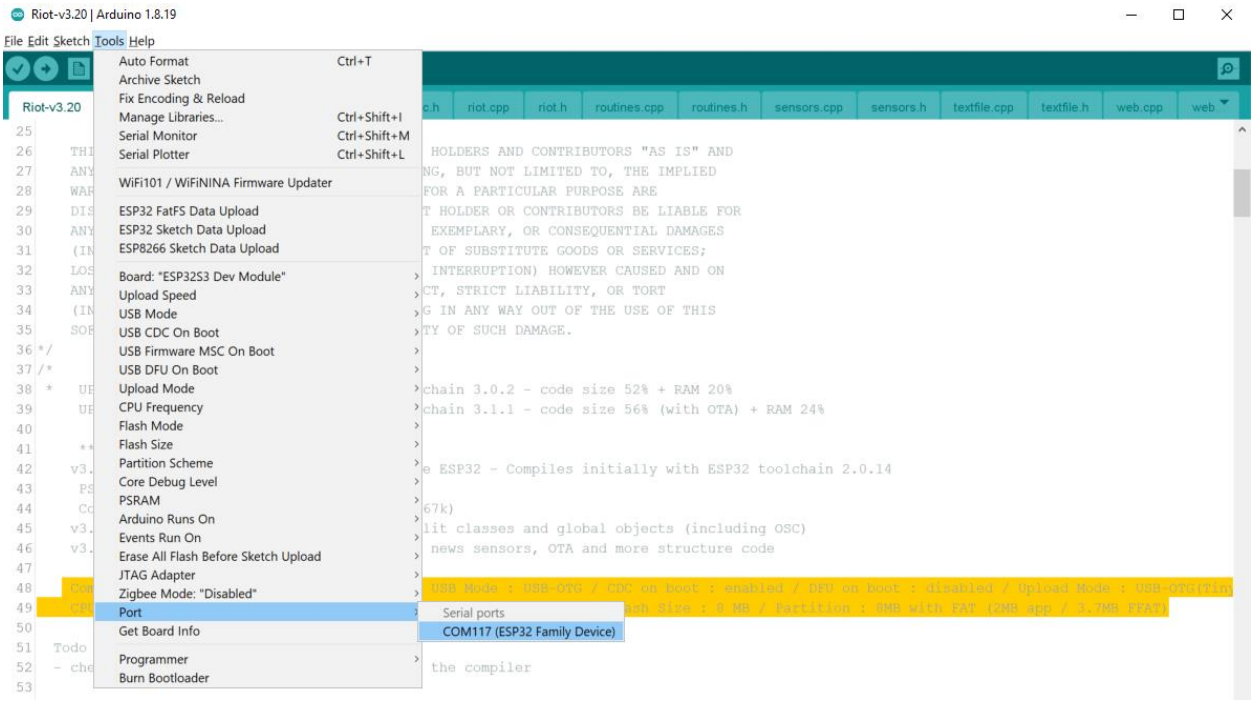



*Figure 9-3: USB Communication Port configuration*

After a successful compilation, click the upload icon on the top left corner (arrow icon) to flash the code on the board. The R-IoT should automatically go in bootloader mode and upload the new firmware.

If the module fails to enter bootloader mode, it can be forced to by pressing both reset and flash onboard switches, then releasing reset first, then flash. The module will then present another USB port for upload, be sure to select the adequate port in the Tools->Port menu before uploading.

After upload completion, the following log will be displayed in the bottom section of the IDE.


*Figure 9-4: Flashing firmware*

After upload, the module will automatically reboot with the updated firmware in action.

# 10 Advanced configuration

## 10.1 Configuration file and parameters

Most of the configuration of the R-IoT module can be achieved by manually editing the configuration text file `config.txt` present on the exposed flash drive after connecting the module to a computer over USB. The configuration file contains a diversity of parameters broken down below with their role and range. Parameters consist in a command followed by the '=' sign and one or more parameters.

- **debug <0/1>**: enables debug mode, makes the module more verbose over the USB serial console, especially at boot time.
- **mode <0/1>**: defines the operating mode of the module, 0 for station mode (the R-IoT connects to a Wi-Fi access point), 1 for Access Point mode (the R-IoT becomes a Wi-Fi access point to which the computer connects).
- **ssid <Wi-Fi name>**: the name of the Wi-Fi network the R-IoT should connect to. Must match the AP / router network name
- **pass <password>**: Wi-Fi password if security is enabled on the Wi-Fi Access Point (AP). Leave blank if no security is used. Otherwise, ensure the password is 8 to 32 characters long.
- **mdns <local DNS name>**: The R-IoT gets advertised on the local network by a name instead of its IP address by the mDNS system, also known as Apple's Bonjour protocol. Set the name of your choice and the module can be contacted over HTTP or OSC by the set name.local. For example, setting mdns=my-riot will have the module seen as my-riot.local. For an easy identification of your modules, we recommend to use a root name concatenated with the module's ID (see below). For instance, my-riot3 for the R-IoT that has the ID #3.
- **dhcp <0/1>**: enables the use of DHCP (1) vs. fixed IP address (0). When using DHCP, the access point will provide the IP address to the R-IoT, if the feature is enabled on the AP. Using DHCP is the preferred and most convenient mode. When disabled, a fixed IP address will be used, specified by the parameter ownip below.
- **ownip <IP address>**: specifies the fixed IP address of the module when DHCP isn't used.
- **destip <IP address>**: destination IP address to send OSC data to. It is the computer's IP address (default 192.168.1.100)
- **gateway <IP address>:** in a network, the gateway is in charge to route the messages and packets outside of the network, for instance for joining a destination computer over the Internet or another local network. Most of the

time, the gateway is the IP address of the Wi-Fi access point, with commonly used IP addresses 192.168.1.1 or 192.168.1.254 (check your AP configuration).

- **mask <network mask>**: the network mask is a filtering option that defines which IP address to take into consideration in the network. The default value of 255.255.255.0 requires all IP address to have their first 3 numbers to be the same as the network root class, for instance 192.168.1.xxx. This is the common setting for the network mask.
- **port {0;65000}**: the UDP port to which the module's OSC data is sent. It should match the `udpreceive` port number of the Max/MSP object on the computer. Standard values are in the range of 8000 to 10000 however, certain restrictions exist depending on the operating system used on the computer (reserved ports & services)[5]. For optimum use of the Max/MSP scheduler, we recommend to use a specific port per module so that the `udpreceive` object has its own OSC data reception thread. Our standard numbering scheme uses the modules ID as an offset to the root port number, for instance port 8000 for module #0, 8001 for module #1 and so forth.
- **rxport {0;65000}**: UDP port used to send OSC data from the computer to the module, during calibration for instance. The port can be the same for all modules, since they are individualized by their respective IP address (default: 9000)
- **masterid {0;100}**: ID number used to identify a source module in a received OSC sensor stream. The ID number is concatenated with the OSC address, for instance: `/riot/v3/0/xxxxx` for the module #0. This ensures the ability to know the origin of the data even if several modules are sharing the same UDP port.
- **power {-4;78}**: maximum Wi-Fi power during transmission. See the readme.txt file on the module's flash drive for the corresponding power in dBm (default value: 8 ⇔ 2dBm). To be increased if your Wi-Fi network experiences range issues.
- **samplerate {5;20000}**: sample period of the sensors and OSC transmission, in milliseconds. Increase if you experience lag or transmission gaps when a large number of modules are used on the same Wi-Fi channel or network (default value: 5ms ⇔ 200 Hz).
- **remote <0/1>**: enables the reception and parsing of OSC messages by the module. While enabling it doesn't create any noticeable computing load, it can be disabled during live performance to avoid triggering any calibration process or ping request.
- **forceconfig <0/1>**: enables the access to the web configuration and firmware

[5] https://support.apple.com/en-us/103229

update pages and associated web server. The R-IoT v3 inherits from its legacy web page based configuration system (see section 5.2). The web configuration mode is triggered by pressing the Option switch at boot time when powering up the device. Since the option switch is now miniature and can be hard to access if the module is encased, this option allows for enabling the webserver at all times, which can be useful in early configuration of the module while it's being operated on stage, or during development of the firmware. When the option is enabled, the configuration webpage is accessible in a web browser at the module's IP address http://192.168.1.xxx (or any IP address scheme you have configured) or its mDNS name (see parameter mdns above) http://my-riot.local. The firmware update page is available at the address **Erreur ! Référence de lien hypertexte non valide.**. To be noted that the webserver induces some load and it can make the OSC sampling period slightly less accurate with some jitter and should be disabled during live performance.

- **calibration {0;20000}**: duration in millisecond during which the calibration process can be triggered at boot time. Calibration will be available after a successful connection to the Wi-Fi network, for the configured time. In that timeframe, pressing the option switch will start the process that is also reflected in the USB serial console (available in the Arduino IDE) and visually by the onboard RGB pixel. Like the web configuration method, it's a legacy calibration system that has been replaced by the OSC remote system with much better performance and visual aid. It can be used as a fallback method for calibration but we highly recommend to use the OSC advanced calibration method achieved in Max/MSP using the `riot-v3` abstraction. Set to 0 to disable.
- **charger <0/1/2>**: module's battery charging modes. Mode 0 will always stream OSC whether plugged into USB or not. Mode 1 will never stream if plugged & charging over USB. Finally, mode 2 will stream if plugged but only if the onboard switch is pushed in ON position. We recommend to use setting 2 that facilitates the bench use of the module and makes it easy to switch between charging and streaming modes when USB plugged.
- **ledcolor <R,G,B>**: defines the color of the onboard RGB pixel when streaming OSC. Color is given by a comma separated list of 8-bit red, green and blue values, in the range of {0;255}. This allows for customizing the color and making the pixel extremely dim when needed.
- **cpu <80/160/240>**: specifies the CPU frequency (in MHz) when processing sensors data or computing fusion as well as when assembling the OSC bundle and sending it over Wi-Fi. 80 MHz works fine with the default firmware. More CPU intensive calculation in a custom firmware could use 160 or 240 MHz to expedite computations in a shorter time but will slightly reduce runtime (default: 80).
- **doze <80/160/240>**: defines the CPU frequency (in MHz) when the module

isn't processing sensors and just handling backburner tasks such as the webserver or OSC reception and parsing (default: 80).

- **declination {-40.0;40.0}**: specifies the magnetic declination at the geographic location of use (decimal value). The magnetic declination indicates the offset in decimal degrees between the magnetic north and the geographic north. This value can be obtained at the following URL: https://www.magnetic-declination.com/. This value compensates for the calculated yaw orientation based on the magnetometer sensors.
- **accrange <2/4/8>**: accelerometer sensitivity and range, in g (default: 8)
- **gyrorange <125/500/1000/2000>**: gyroscope sensitivity and range in °/second (default: 2000 dps)
- **magrange <4/8/12/16>**: magnetometer sensitivity and range, in Gauss (default: 4 Gauss)
- **gyrogate {0.0;500.0}**: parameter of the orientation filter and fusion algorithm to determine if the yaw, pitch and roll Euler angles are updated or locked. Acts as a floor noise gate on the gyroscope activity with a °/s angular speed threshold. Doesn't affect the orientation filter calculation that continues to run in background. A low value of 1.5 to 3dps can avoid jitter and produce a stable output. 0 disables the noise gate.
- **baromode {0;5}**: sampling mode of the barometer sensor. Depending on the selected mode, the pressure sensor can have a fast sampling with low precision or a lower sampling rate with higher resolution. The following modes are available:
    - 0: low power, ultra-low precision, sample rate 81 second (weather logging usage)
    - 1: low precision, 100 Hz sample rate, no filtering
    - 2: normal precision, 50 Hz sample rate, filter strength 3
    - 3: normal precision, 50 Hz sample rate, filter strength 1
    - 4: high precision, 12.5Hz sample rate, filter strength 1
    - 5: ultra-high precision, 25 Hz sample rate, filter strength 3

  (default value: mode #3, good update rate and smoothed, good bargain for live performance use)
- **baroref {0.0;1000.0}**: reference altitude of the sensor a boot time. This value is used either to define a relative altitude (0m for instance) where the sensor is used or to recalibrate the altimeter algorithm based on the known geographic altitude of the use location. In the context of live performance, 0.0 can be used so that the relative altitude referenced to the user's body can be measured. To be noted that the altitude measurement is highly dependent of the ambient

temperature and the modules temperature increases of 10 to 20 °C after boot time which will affect the reference. We suggest to use 0m regardless and deal with absolute or differential altitude in the Max/MSP patch, when the temperature has stabilized[6].

- **orientation <0/1/2/3>**: defines the reference orientation frame of the module and its relation to the pointed north (yaw = 0°). This affects the signs and axis of the raw sensors value of the physical sensor placed on the module's PCB, in particular what the X and Y axis are. The illustrations below show the axis affectation for each orientation.

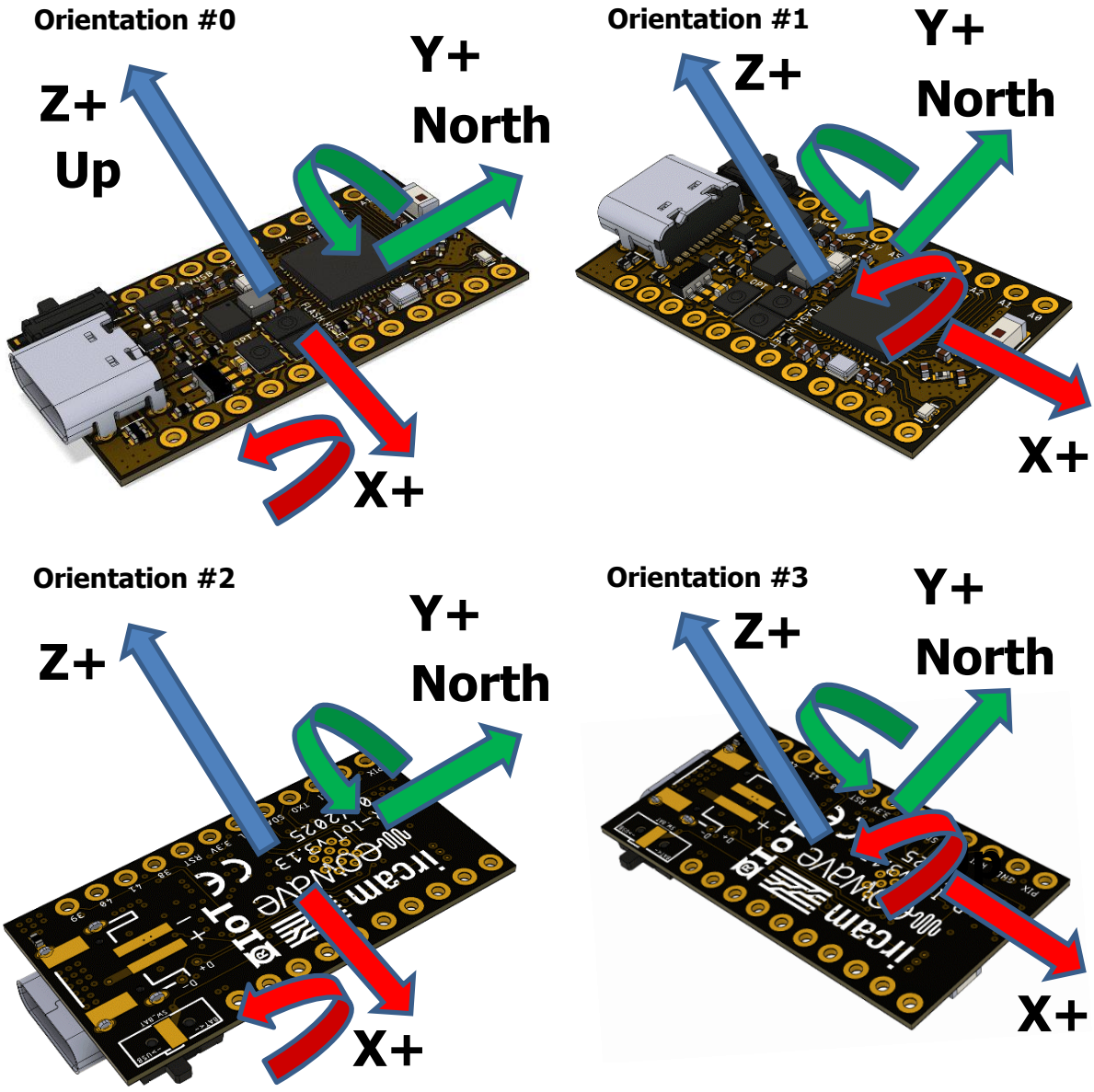



*Figure 10-1: Board orientations and axis*

[6] Future versions of the firmware will include an altitude reset remotely controlled by OSC.

Orientation #0 is quite common and matches the natural orientation of a smartphone as per the W3C standard.

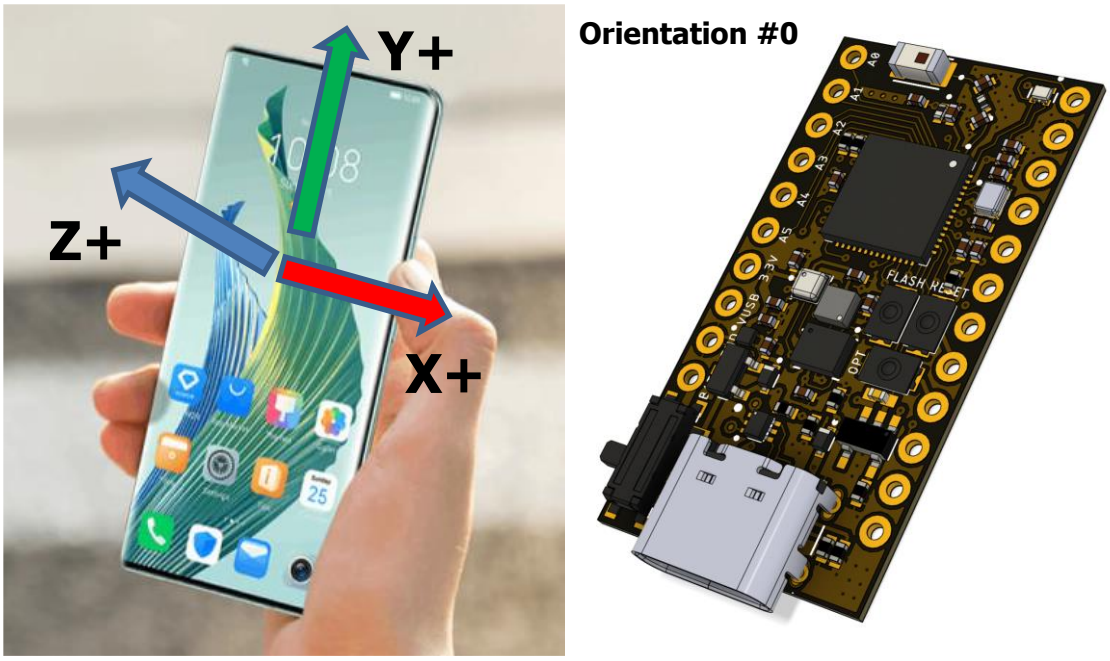



*Figure 10-2: Smartphone natural orientation as per W3C*

- **bno_orient {0;7}**: reference orientation and axis of an external BNO055 IMU sensor connected on the I2C bus of the module. This sensor computes internally an orientation filter producing Euler angles. Refer to the BNO055 datasheet to select the adequate reference orientation which will depends on the breakout the sensor is soldered on, and the relative placement of the breakout to the module.
- **acc_offsetx/y/z**: accelerometer calibration offsets defining the reference plane of the module. Those values are automatically populated by the calibration process
- **gyr_offsetx/y/z**: gyroscope calibration offsets. Automatically populated by the calibration process after noise averaging and stillness analysis.
- **mag_offsetx/y/z**: magnetometer calibration offsets or hard-iron offsets. Automatically populated by a dedicated calibration process (see Section 12) to center the magnetic sensor spheroids.
- **soft_matrix1/2/3**: magnetometer soft-iron correction matrix containing 3 lines of symmetrical coefficients compensating the non-linearity of the measured magnetic fields, usually caused by surrounding ferromagnetic objects (battery, casing, PCB traces, nearby loudspeakers). This matrix should be as close as an identity diagonal matrix.
- **beta {0.0;2.5}**: Madgwick orientation filter convergence algorithm parameter. A small value favors the use of the gyroscopes (0 uses gyroscopes only). A small

value leads to slow convergence of the Euler angles and is ideal with noisy magnetometers. A larger value leads to faster response on the Euler angles but might feature angle jittering (default value 0.4). To be noted that at boot time, the module will use a linear regression over 1s starting from a high beta value (2.5) and converging towards the beta specified in the configuration file. This allows for a fast initial convergence of the orientation of the module then using the desired filter stability.

## 10.2 Serial commands

In order to debug, diagnose or operate the module, serial commands can be sent to the R-IoT via the serial console or terminal, with the USB cable plugged to the computer. The Arduino IDE provides such a console in a convenient way. To access it, select the appropriate serial port from the Tools->Port menu then click on the top-right corner magnifier icon "serial monitor" (also available from the Tools menu). A new window will open. At the bottom, ensure the line ending settings are set to "new line".

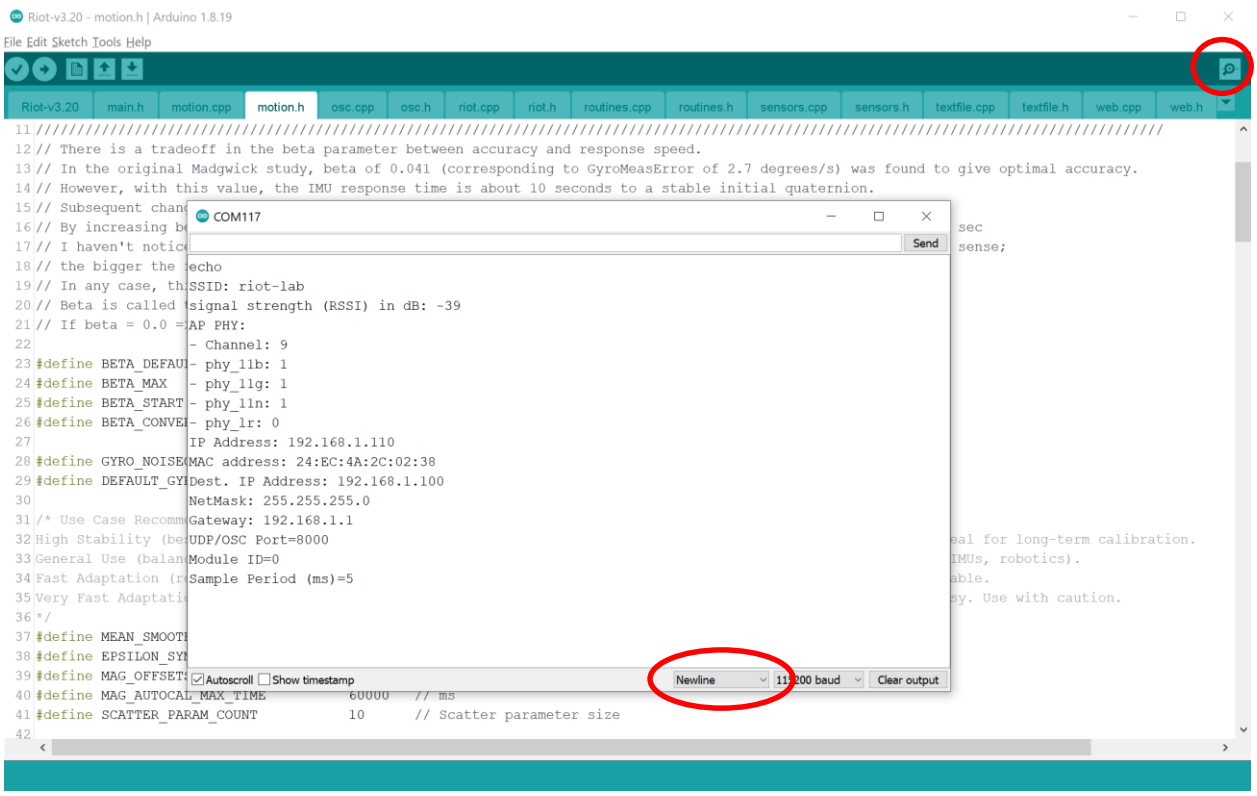



*Figure 10-3: Serial Terminal / Console in the Arduino IDE*

The serial terminal accepts textual commands with parameter(s) using the same syntax as the one from the configuration file. That allows for both controlling the module and changing some of its parameters live without the need of editing the configuration.

For instance, the user can type:

```
samplerate=10
```

followed by the return key. The module will instantly use a sample rate of 10ms. Other commands that aren't in the configuration file can be used to calibrate, diagnose or control the module's behavior. Commands should be terminated by a new line ASCII character (DEC 10) or carriage return (DEC 13). Those termination characters are automatically added in the Arduino IDE terminal when validating the command with the Enter or Return keys, if the appropriate setting has been selected at the bottom of the window. Alternatively, other terminal applications can be used, such as coolTerm, docklight or advanced serial port terminal. Somme commands have parameters following the same syntax as the configuration, using the '=' sign as separator between the command and its argument.

**List of (non-parameter) commands:**

- **ping**: questions the module for being "alive". The module should answer "echo" if receptive.
- **cfgrequest**: displays the current configuration of the module.
- **savecfg**: saves the current configuration of the module in config.txt.
- **reset**: reboots the module, for instance for applying recent changes made to config.txt or after dynamically modifying the parameters using serial commands.
- **defaults**: restore (some) default parameters, mostly concerning sensor and motion calibration data. Combine with `savecfg` command to update `config.txt`.
- **format**: formats the R-IoT flash drive and creates a default configuration file. Additional files such as webserver contents and default package should be copied manually.
- **calibrate**: starts the legacy calibration process. Passing thru the different calibration steps can be done by pressing the Option switch or typing the `GO` command.
- **autocalmotion**: starts the accelerometer and gyroscopes offset calibration. Module must be placed on a flat reference surface defining the zero plane and set still. After analysis, motion sensors offsets are produced. They can be stored in the config.txt file using the `savecfg` command.
- **autocalmag**: starts the magnetometer calibration process (see Section 12)
- **Wi-Fi**: lists the main information of the module's connection such as its MAC address, Wi-Fi network status and IP address as well as its main configuration parameters.
- **rssi**: displays the Wi-Fi connection signal strength.

- **version**: logs the current firmware version. Adding =1 as an argument to the command will log this information to version.txt on the module's mass storage flash drive.
- **slowboot {0;20000}**: permanently store the duration of a wait delay executed at boot time. It allows for having enough time to open the serial connection when debugging the module so that the boot log is complete. Do not include in the configuration file and only use as a serial terminal command
- **logmotion <0/1>**: enables the ASCII log of the accelerometers and gyroscopes values. They can be observed textually in the Arduino IDE serial console, or as a plotted graph using the serial plotter from the Tools menu.

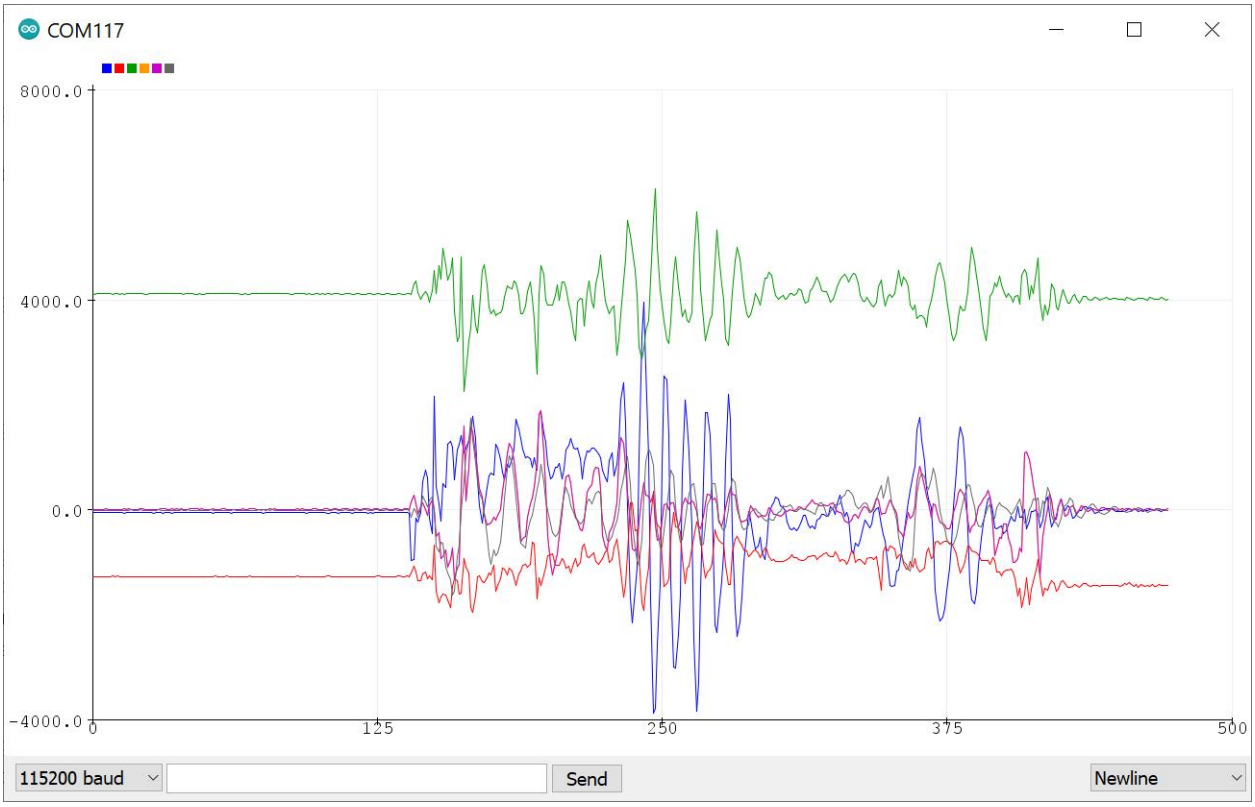



*Figure 10-4: Serial plotter in the Arduino IDE*

- **logmag <0/1>**: enables the ASCII log of the three-dimensional magnetometer in a similar way to the motion sensors above.
- **battery**: displays the battery voltage in volts (the main power switch must be ON)
- **usb**: displays the USB voltage if the module is plugged.

## 10.3 IP management & local DNS

### 10.3.1 Automatic IP handling & DHCP addresses reservation

This section indicates some good practice regarding the handling of R-IoT modules IP addresses and the associated IP map. Handing the IP addresses management to the Wi-Fi access point using DHCP is convenient as the user doesn't need specifically to decide on the IP map. The R-IoT module usage is mostly based on streaming OSC data to a specific computer identified by a fixed IP address, on a known port, so knowing the module's IP address isn't absolutely mandatory.

However, during the configuration or calibration phases, know the modules IP can be useful to ensure their proper connection to the Wi-Fi network or that they are correctly seen by the computer, for instance by using the network ping command in a terminal.

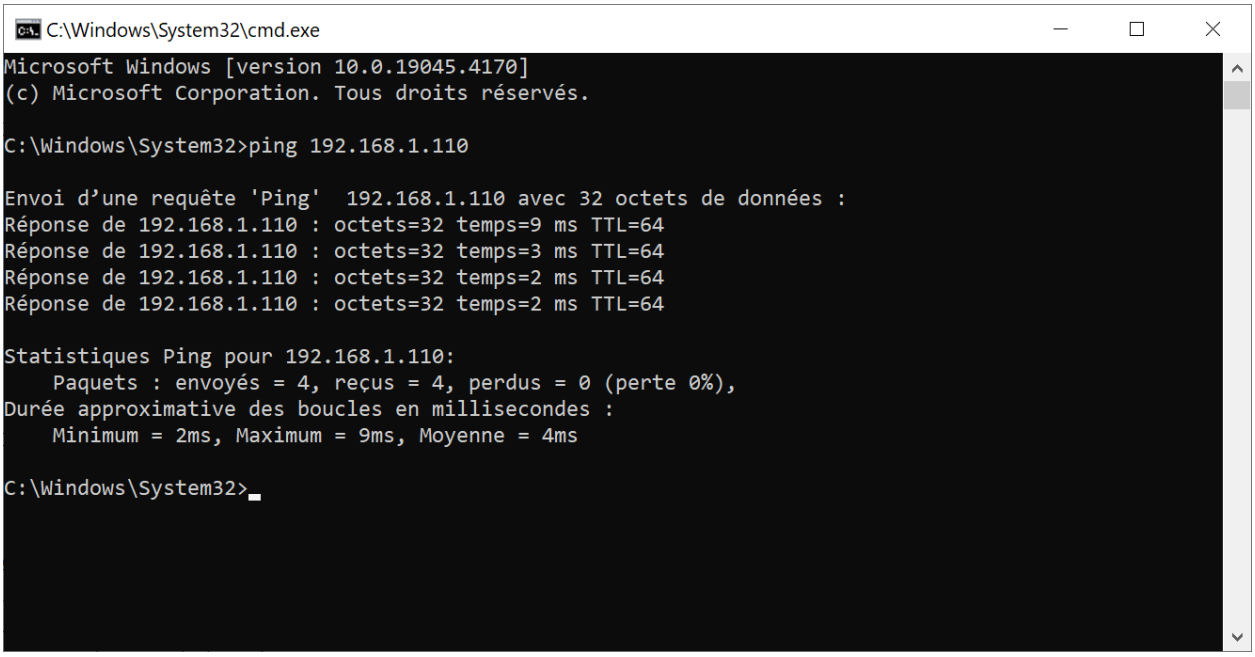



*Figure 10-5: Successful network ping of a R-IoT module*

The R-IoT can be configured with a fixed IP address, however this solution binds the configuration to a specific network addressing scheme and it can make the re-configuration of a set of module tedious when switching contexts or use cases. For that reason, we suggest the user to utilize the DHCP address reservation system offered by most Wi-Fi access points. The simple idea behind it is to make the DHCP server attribute always the same IP address to a specific R-IoT module using its unique network identifier, the MAC address. The MAC address is a 6 number identifier unique to each network peripheral. It can be conveniently obtained using the serial terminal and the command `Wi-Fi`.

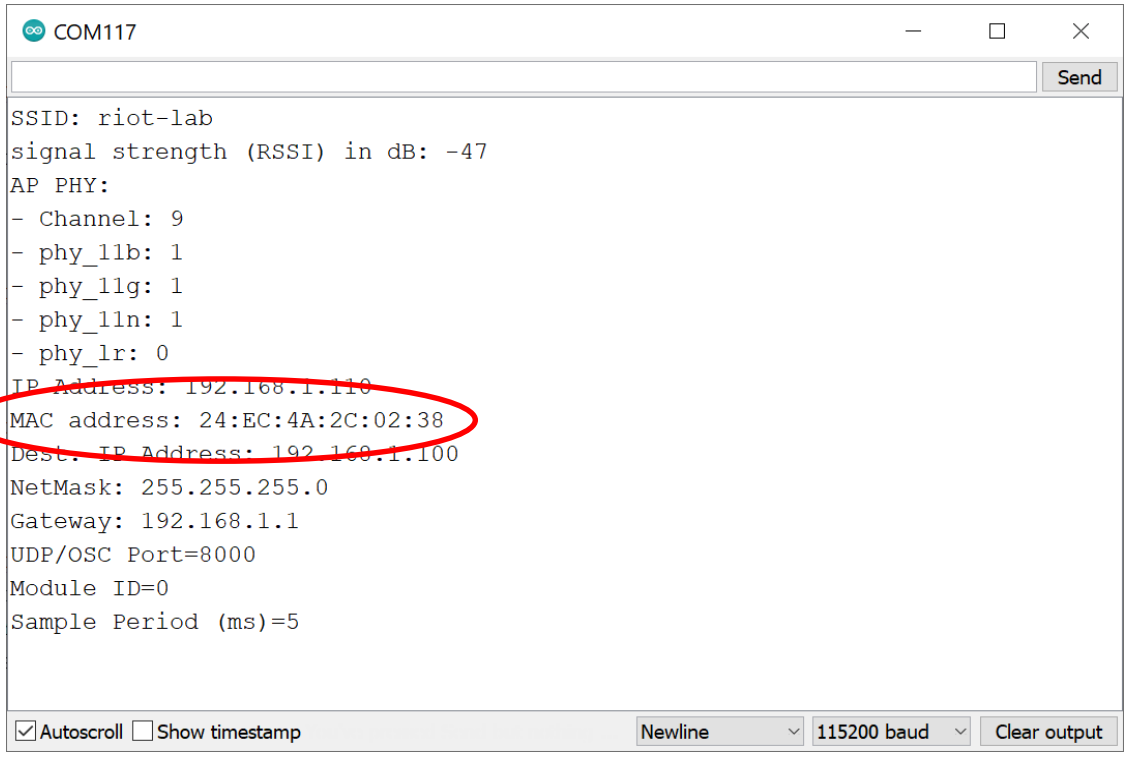



*Figure 10-6: MAC address log of a R-IoT module*

When handling a set of several R-IoT modules, each of them should be ideally numbered with a label figuring its ID as configured per the `masterid` in the `config.txt` file. In a spreadsheet of your choice, list each module with its ID and MAC address, in hexadecimal form and colon separated, as output by the serial terminal. From there, open the access point configuration page in a web browser and reach out the DHCP – address reservation section.

Add new reservations for each of your modules, specifying the MAC address and the desired IP address, matching the scheme of your network, for instance 192.168.1.xxx. Use an IP address range within the DHCP lease pit. For instance, if your DHCP is configured to provide addresses from 192.168.1.110 to 1.150 start from 1.130 for module ID #0 and increment to preserve a logic numbering. Note the associated IP address to each module along its ID and MAC address in the spreadsheet as well as the reference of the access point used, its Wi-Fi name, IP address and administration page login and password.

This topology and architecture implies that the IP addressing is fully handled by the access point and not by the modules. If those are used for another context or performance, they can simply be used with another router, with its own DHCP configuration, and the modules IP addresses will automatically adapt.

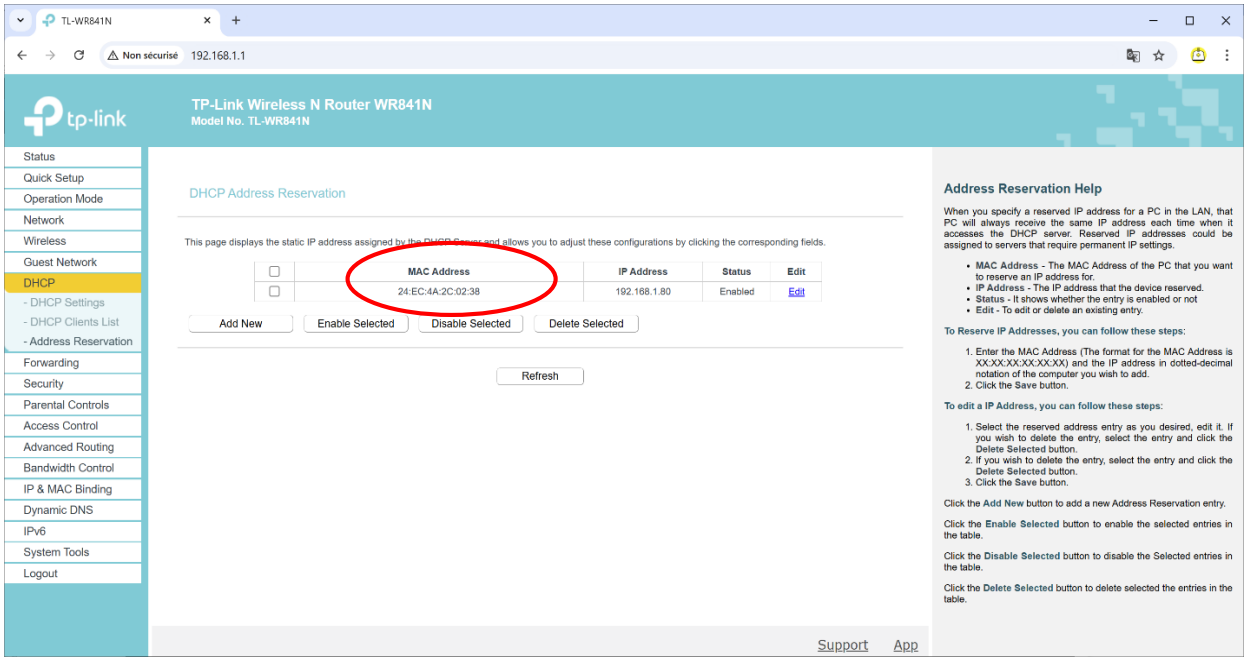



*Figure 10-7: DHCP address reservation in a TP-link Wi-Fi access point*

Finally, thanks to the R-IoT mass storage flash drive, the user can also keep track of this information in a dedicated text file, for instance network.txt, where all useful information is preserved (MAC address, IP address reserved, network name, access point login).

With that information known and reliable, each R-IoT IP address can be reference in the Max/MSP abstraction

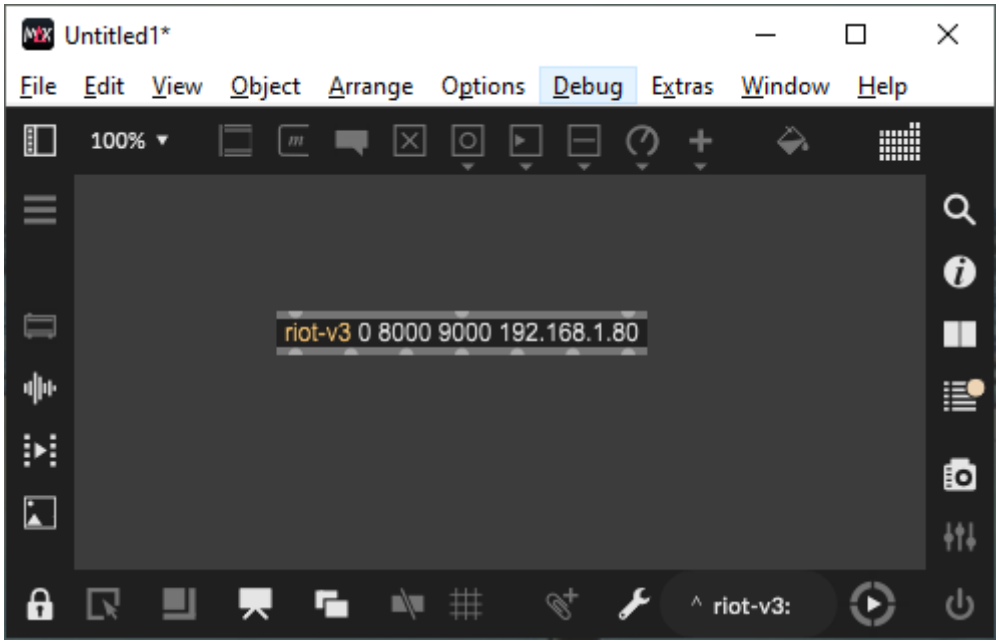



*Figure 10-8: riot-v3 abstraction instantiated in a Max/MSP patch*

If communication settings (IP, ports) are correct, clicking the **ping** button should produce an **echo** response in the log window. Bi-directional communication now allows for talking back to each R-IoT module, in particular for the sensors wireless calibration described in Section 12.

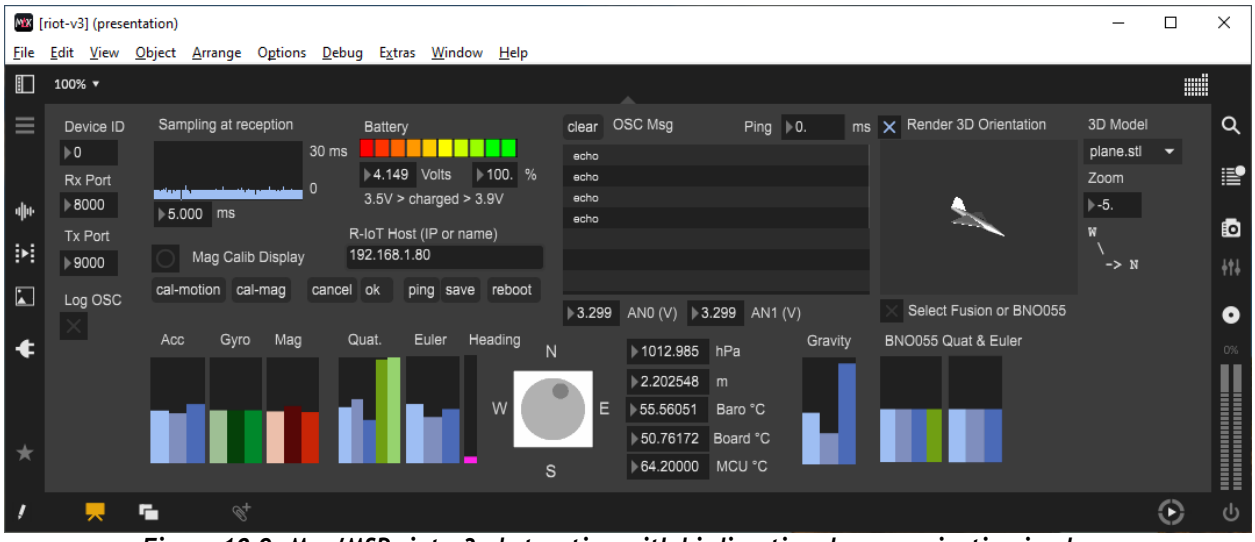



*Figure 10-9: Max/MSP riot-v3 abstraction with bi-directional communication in place*

### 10.3.2 Multicast DNS & host naming

The R-IoT features multicast DNS or mDNS, also referred as Bonjour (Apple's former implementation). This system allows for a network peripheral to advertise and be joined as a host name instead of its IP address. This principle allows for an easier access of the module's configuration web page without having to refer to the IP address spreadsheet suggested in the previous section. The mDNS name of the module is defined in the configuration file using the parameter `mdns`.

From there, according to the fact the module is powered and connected to the network, with its webserver active (either in standard configuration mode as explained in section 5.2 or in forced configuration mode by parameter `forceconfig`), it can be reached using the concatenated mDNS name and .local.

In a similar pattern, the mDNS host name can be used in the Max/MSP `riot-v3` abstraction or directly in a `udpsend` OSC object. However, this needs to be used with caution, in particular in a performance Max/MSP patch, as the `udpsend` object will try to resolve the host name before fully opening the patch: as a result, if the module isn't reachable (powered off) or misnamed, and this can lead a large opening delays. In performance situations, we recommend to rather use the known IP address (established automatically with DHCP address reservation) in the concert or experiment patch and restrict the use of mDNS host naming for configuration or diagnose purposes

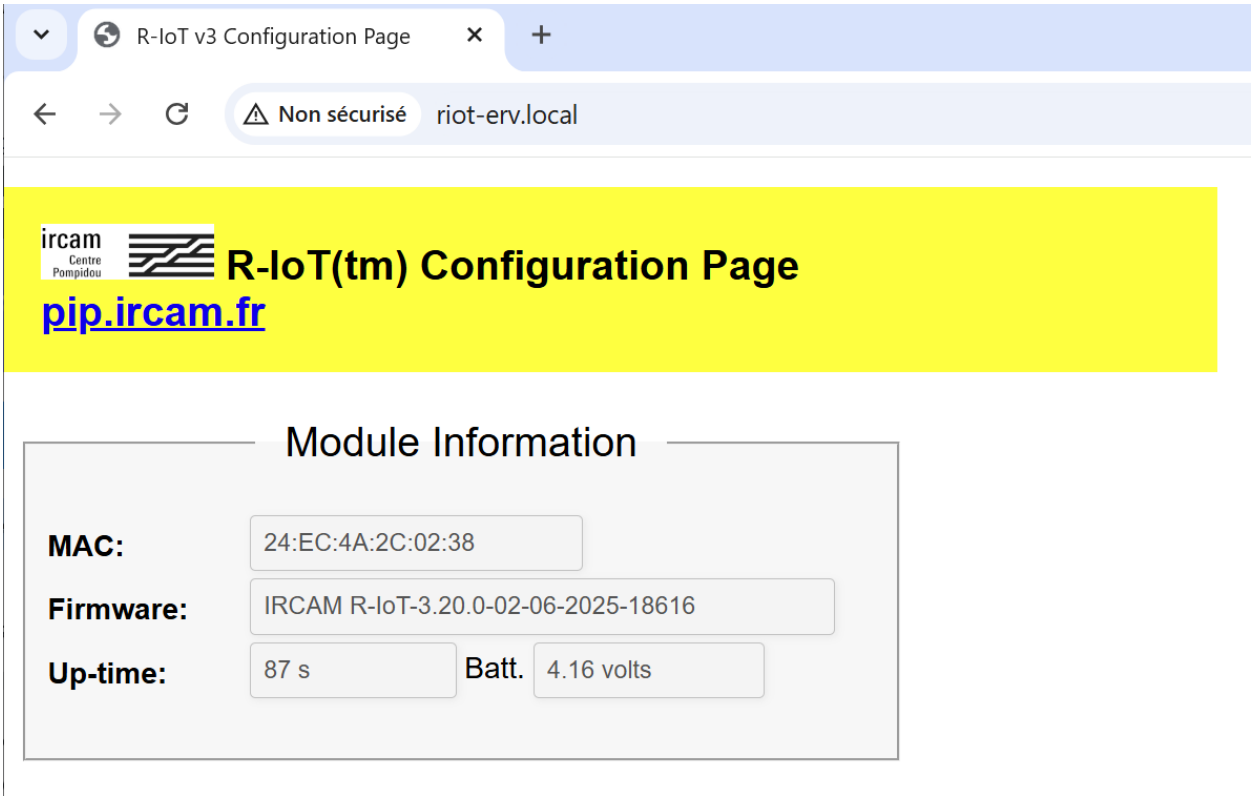



*Figure 10-10: Accessing the R-IoT configuration webpage using mDNS*

# 11 Firmware update methods

The R-IoT module can be updated using several methods, over USB or wireless.

## 11.1 USB standard update

This is the preferred method for regular users and it uses a precompiled binary image of the firmware, received by email or gathered from the github repository of the R-IoT project[7] within the release section. Plug the R-IoT to a computer using a USB cable. Open the flash drive and copy the firmware .bin file on it. Rename the file `update.bin`. Unplug the module and reset it by power cycling the ON-OFF switch or pressing the reset switch[8]. After rebooting the onboard RGB pixel will blink white, then move to static white while the update is performed. Upon completion, the pixel will turn green and the module will reboot again (the `update.bin` file will be automatically deleted).

The new firmware version can be verified by observing the contents of the file version.txt that is automatically created and populated after the update.

[7] https://github.com/ircam-ismm/riot-v3

[8] The serial command 'reset' can also be used from the serial terminal

## 11.2 Arduino IDE update

This method is described in section 9 and includes direct flashing using the Arduino IDE environment.

## 11.3 Wireless update over Wi-Fi

This update method is offered thru the R-IoT webserver and the OTA (Over-The-Air) feature proposed by the ESP32. To be used, the R-IoT must be in configuration mode (see section 5.2) or in forced configuration mode using the config.txt parameter `forceconfig=1`.

When in regular configuration mode, first connect to the Wi-Fi network created by the R-IoT itself (ideally, update only one module at a time) and access the update page using the URL http://192.168.3.1/update in your preferred web browser. When the R-IoT is in forced configuration mode, use the module's IP address or mDNS name (ending with .local) for instance http://192.168.1.50/update or http://my-own-riot.local/update.

The following firmware update webpage will be displayed:

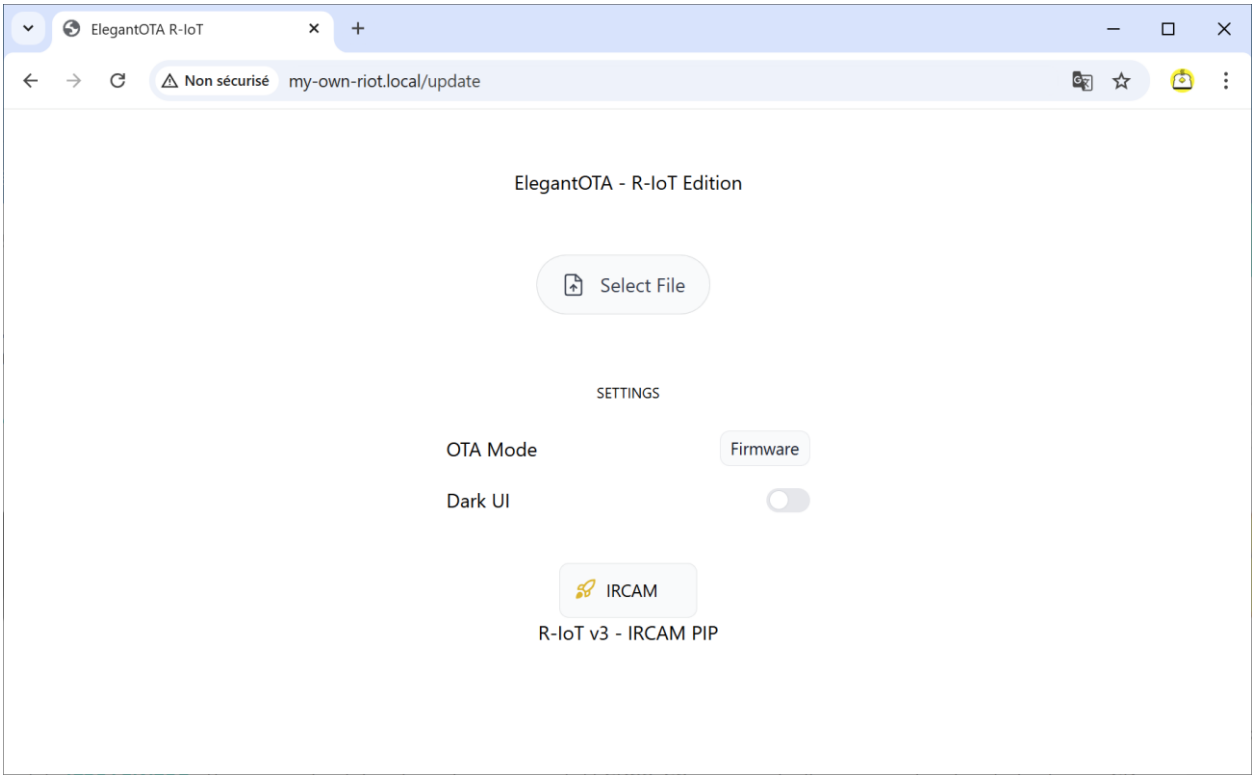



*Figure 11-1: Accessing the R-IoT update webpage using mDNS host naming*

To update the firmware, click on the Select File button. The user will be prompt to select the binary (.bin) firmware file and the R-IoT will automatically start the update process and display a progress bar. Upon completion, the module will reboot and the freshly updated firmware can be used.

## 11.4 Checking the firmware version

After a successful update, the R-IoT will log the firmware version to the file version.txt on the USB flashdrive of the module. This operation will also be executed at any boot time if the file version.txt is missing, providing the user an indication of the current firmware installed on the device. The MAC address is also conveniently logged in the very same file. This operation can be manually executed by typing the command `version=1` in the serial terminal which will both display the version in the console and write it in the file.

# 12. Calibration

In order to properly compute the module's orientation in the form of 3 Euler angles yaw, pitch and roll, the R-IoT firmware fuses sensors data from its accelerometer, gyroscopes and magnetometers using the Madgwick Orientation Filter [7]. This computation supposes that the sensors are properly calibrated. For the accelerometers, offsets should be removed so that the module has a proper reference plane and displaying a -1g gravity force on the Z axis when laying on a flat surface. For the gyroscopes, noise offset must be removed from raw data on all axis so that an average 0°/s rotation speed is obtained when the module isn't spinning. Finally, magnetometers must have their offsets removed in order to be centered in an orthonormal coordinate system, in a similar way the accelerometers are corrected.

## 12.1 Accelerometers & Gyroscopes calibration

Those 2 sensor classes can be calibrated at the same time using a simple procedure. Calibration can be triggered via the serial console (with the R-IoT plugged over USB) and the command `autocalmotion` after placing the module on a flat and stable surface. The same process can be triggered within the `riot-v3` Max/MSP abstraction. After verifying the good bi-directional communication with the module by clicking the **ping** button and the corresponding **echo** response shown in the OSC message log window (Figure 12-1), click the **cal-motion** button.

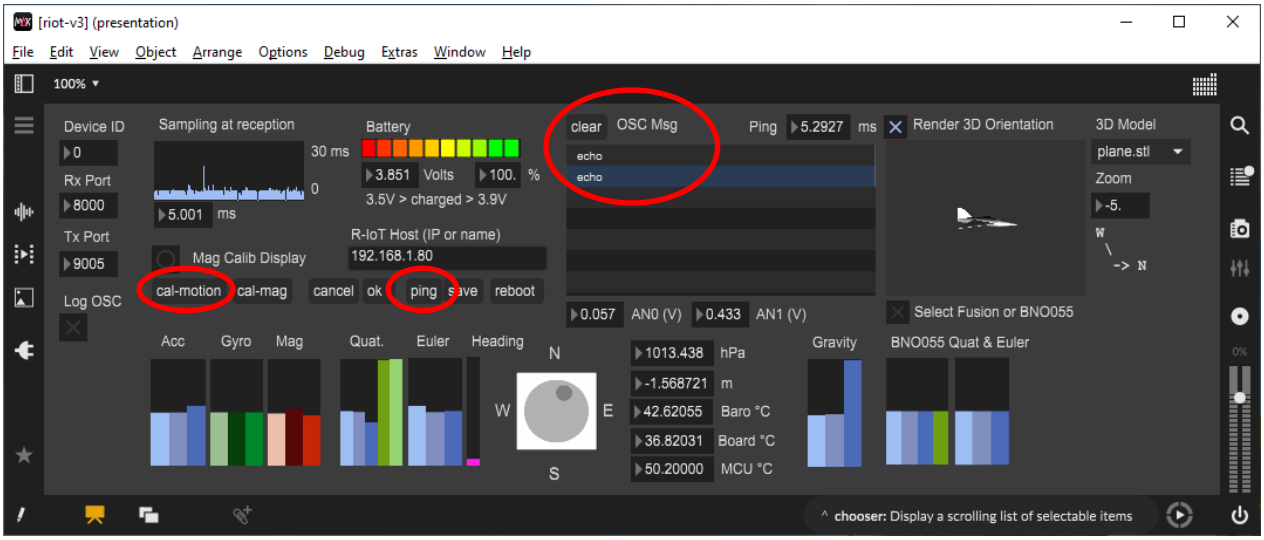



*Figure 12-1: Accelerometer & Gyroscope calibration in Max/MSP*

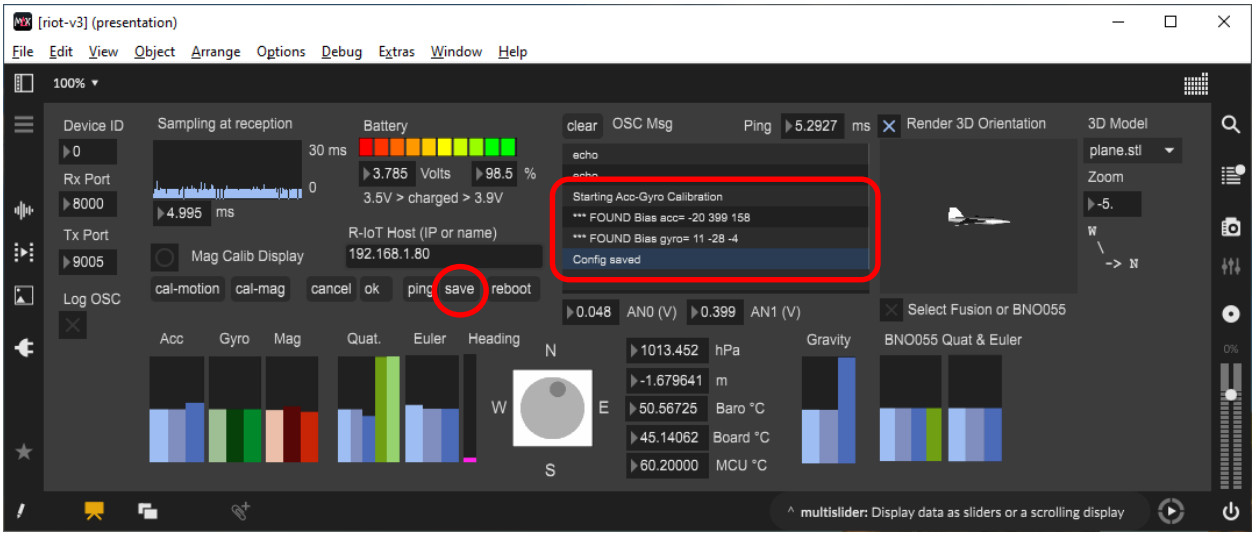



*Figure 12-2: Saving Accelerometer & Gyroscopes calibration*

Upon completion the module will log the found correction offsets (Figure 12-2). Process should be completed by saving the offsets, either by the save button in the Max/MSP abstraction or typing the serial command `savecfg`. This calibration process should be executed after the R-IoT module has warmed up a little, since the gyroscope noise and offsets tend to change a little with temperature. Yet, once executed, and unless the reference plane changes (moving the module to another casing for instance), it doesn't need to be executed too often. Finally, note that the calibration relies on the selected orientation of the module (`orientation` parameter in config.txt) which should be selected first place before launching any calibration.

## 12.2 Magnetometers calibration

The magnetometers sensors are slightly trickier to calibrate than the motion sensors. Magnetometers measure the surrounding magnetic fields: those include Earth's natural magnetic field combined with any additional magnetic sources such as nearby loudspeakers, magnets and some ferromagnetic materials.

The Magwick fusion algorithm needs the magnetometers data to be calibrated which requires a 2-stage process:

- Hard-iron calibration: the hard-iron effects are the constant magnetic fields in the environment that are detected by the sensors which results in a non-centered data set.
- Soft-iron calibration: the soft-iron are non-linearity distortion caused by ferro-magnetic materials immediately surrounding the sensor (sometimes also contributing to the hard-iron effect) and creating a non-linear response between the sensor reading and the actual magnetic field.

As a result, the hard-iron effect is observed by an offset sphere of data points, while the soft-iron effect distorts its spherical aspect and turns it into a crushed, potato-like shape (Figure 12-3).

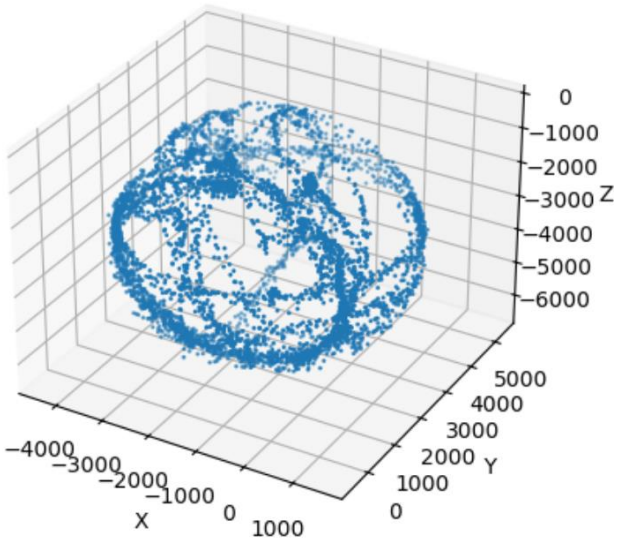



*Figure 12-3: 3D raw magnetometers data feat. hard and soft iron effects*

The calibration process aims to find the hard-iron offsets (to be permanently subtracted from the raw values) and apply a soft-iron correction matrix to reshape the response into a sphere, on all axis. This process is made available thru the Max/MSP riot-v3 abstraction just like the motion sensor calibration. After the communication with the module has been checked (**ping** button), first click the **Mag Calib Display** button to open the magnetometer ellipsoids window (Figure 12-4):

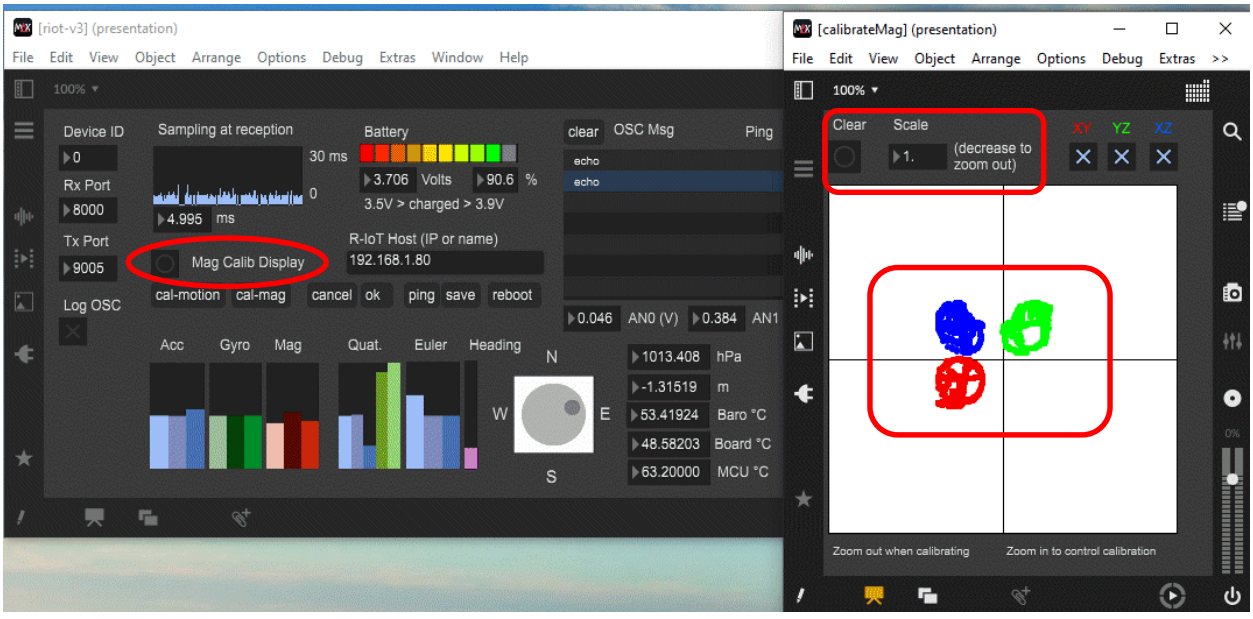



*Figure 12-4: Magnetometer calibration visualization window*

When the magnetometers aren't calibrated yet, the configuration file would contain zeroed offsets as well as an identity soft-iron correction matrix (3x3 matrix with 1.0 on the diagonal terms, 0.0 elsewhere). When rotating the module around its 3 axis, the general result observed is set of 3 colored circles offset from the center, non-concentric and eventually slightly distorted (not totally circular).

Start the magnetometers calibration by clearing the calibration display window then click the **cal-mag** button. Observe the calibration graphic window and spin the module around all axis so that 3 circles or discs are drawn. After a while, the message "Mag means stable for long enough – Hard iron completed" (Figure 12-5).

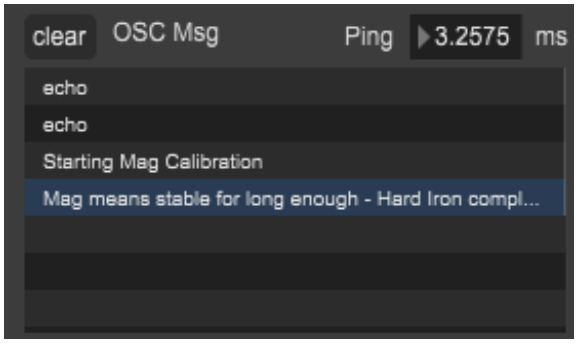



*Figure 12-5: Magnetometer hard-iron calibration completed*

This marks the beginning of the second analysis and calibration phase. Click the clear button of the calibration display window and continue spinning the module over the 3 axis to complete the 3 colored circles again, and end by "coloring" them by waving the R-IoT with a circular wave or 8-figure motion pattern[9] (Figure 12-6).

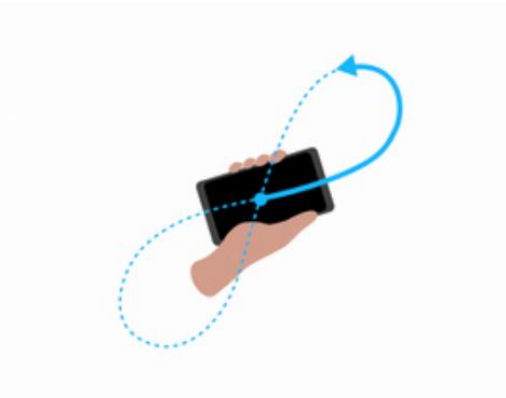


*Figure 12-6: 8-figure motion pattern*

Upon completion, after about 30 seconds of waving the module, click the **ok** button. This will finalize the calibration by computing the hard-iron offsets and the soft-iron correction matrix. This will be reflected in the log window (Figure 12-7)

[9] https://calibratecompass.com/images/figure-8-compass-calibration.gif


*Figure 12-7: Magnetometer full calibration completed*

Calibration can be further tested in the magnetometer ellipsoid tracing window, by zooming a little and observing 3 concentric circles indicating calibration success (Figure 12-8). Furthermore, the module should now provide a correct yaw orientation and proper pointing the direction of the North.

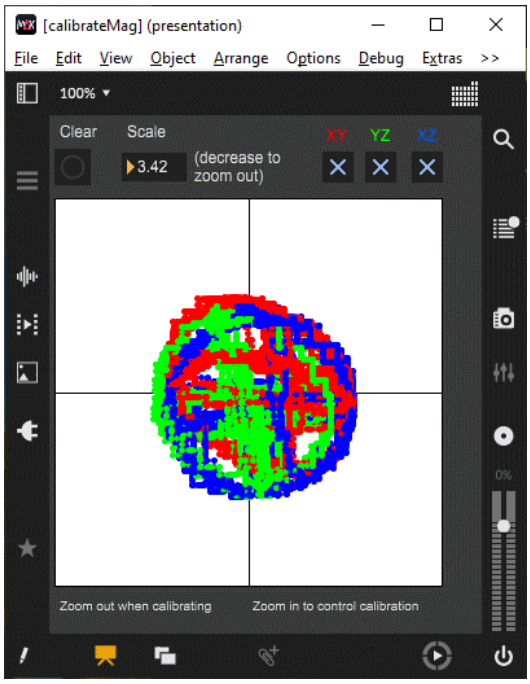



*Figure 12-8: Magnetometer successful calibration with centered ellipsoids*

If the calibration is satisfying, click the **save** button to store the offsets and correction matrix to `config.txt`. If one or more of the ellipsoids aren't too circular, the soft-matrix computation most likely failed. Observe the resulting matrix in the configuration file, diagonal terms should be 1.0 (as the matrix is normalized) and other terms should be very small and symmetrical around the diagonal. Those terms indicate the cross correlation between certain axis, values should not exceed 0.1 in absolute value. Higher cross-correlation terms usually indicate a failure of the calibration process (insufficient spinning over the different axis) or a strong local magnetic field disturbance. The soft-

iron correction can be disabled by resetting the matrix to an identity matrix[10] in the configuration file.

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

The magnetometer calibration must be triggered each time the magnetic environment changes, essentially for the hard-iron offset and ellipsoids centering, due to the local magnetic perturbations such as a metallic structure or loudspeakers.

[10] Future firmware revision will most likely have a parameter to decide whether to use the soft-iron correction or not.

# 13. Advanced wiring

This section covers the wiring of additional accessories or sensors to the R-IoT module.

## 13.1 Battery

Protected lithium-polymer batteries with capacity ranging from 250 to 700 mAh can be used with the module. The provided case 3D models use a 250mAh 50x20x3mm EEMB battery as shown below.


*Figure 13-1: Protected Lithium-Polymer battery*

While the battery can be connected to the R-IoT using the provided JST connected (to solder on the bottom side of the PCB), this takes a lot of space and isn't practical to use when the module is encased in a plastic housing, such as the 3D printed cases provided on the code repository. Instead, the battery can be directly soldered to the R-IoT and installed permanently in the 3D printed casing. To solder the battery, ensure the power switch is in off position first. Then the wires wires should be cut, stripped and soldered **one by one**. Shorting the battery must be avoided by all means[11].

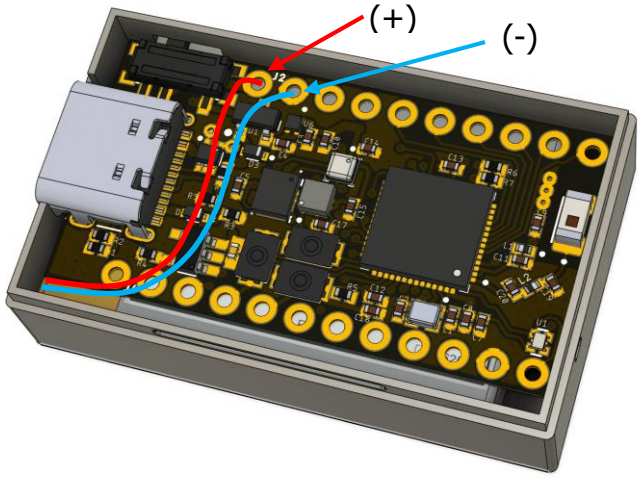



*Figure 13-2: Battery wire management & direct soldering to the R-IoT*

[11] Also, un-protected batteries must never be used

## 13.2 External tactile switch

Hooking switches to the module can be useful for modal control of the Max/MSP patch, for instance for triggering a gesture recording, enabling an effect or controlling a specific process (like resetting the relative altitude measurement or a relative angle position). The default firmware exports the state of 2 digital inputs of the module:

- GPIO #38: this pad is actually connected to the onboard Option switch, used by the firmware to control calibration or configuration mode at boot time, but can also be used during normal streaming as a standard switch input action
- GPIO #41

Both pads have internal pull-up resistors enabled and don't require any additional electronic part aside the switch itself. The state of the 2 switches are exported in the OSC message structure under the address ending with `/control`.

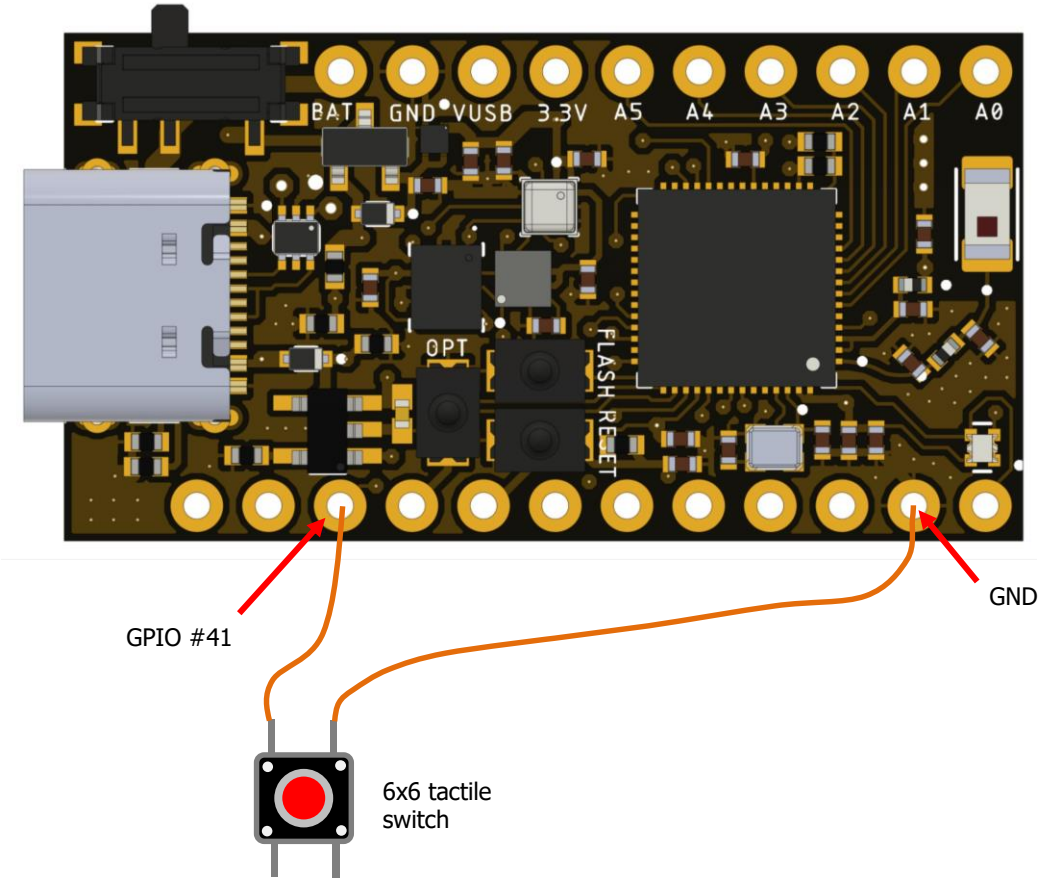



*Figure 13-3: Tactile switch attached to GPIO #41*

# 14. Glossary

### Access Point

An access point is a device that allows other wireless devices to connect to it by the means of antennas.

### IMU

An inertial measurement unit (**IMU**) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers.

For further information about the R-IoT sensors, please refer to the following datasheets: LSM6DSL, LIS3MDL, BMP390.

### Max

Max, also known as Max/MSP/Jitter, is a visual programming language for music and multimedia developed and maintained by San Francisco-based software company Cycling '74. Over its more than thirty-year history, it has been used by composers, performers, software designers, researchers, and artists to create recordings, performances, and installations [10].

The Max program is modular, with most routines existing as shared libraries. An application programming interface (API) allows third-party development of new routines (named external objects). Thus, Max has a large user base of programmers unaffiliated with Cycling '74 who enhance the software with commercial and non-commercial extensions to the program. Because of this extensible design, which simultaneously represents both the program's structure and its graphical user interface (GUI) [11].

For further information please visit https://cycling74.com/.

### OSC

Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. Bringing the benefits of modern networking technology to the world of electronic musical instruments, OSC's advantages include interoperability, accuracy, flexibility, and enhanced organization and documentation.

This simple yet powerful protocol provides everything needed for real-time control of sound and another media processing while remaining flexible and easy to implement.

Features:

- Open-ended, dynamic, URL-style symbolic naming scheme
- Symbolic and high-resolution numeric argument data
- Pattern matching language to specify multiple recipients of a single message
- High resolution time tags

- "Bundles" of messages whose effects must occur simultaneously
- Query system to dynamically find out the capabilities of an OSC server and get documentation

For further information please visit http://opensoundcontrol.org/.

# 15. Bibliography

[1] Espressif, "ESP32-S3 SoC." [Online]. Available: https://www.espressif.com/en/products/socs/esp32-s3
[2] Arduino, "Arduino IDE." [Online]. Available: https://www.arduino.cc/
[3] H. Barragán, "Wiring." [Online]. Available: http://wiring.org.co/
[4] C. Severance, "Massimo banzi : Building arduino," *Computer*, vol. 47, no. 1, pp. 11–12, 2014.
[5] A. Freed, "Open Sound Control : A New Protocol for Communicating with Sound Synthesizers"," in *International Computer Music Conference*, Thessaloniki, Greece, 1997.
[6] M. Wright, "Open Sound Control." [Online]. Available: https://ccrma.stanford.edu/groups/osc/index.html
[7] S. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," University of Bristol, Technical Report, 2010. [Online]. Available: https://x-io.co.uk/downloads/madgwick_internal_report.pdf
[8] S. Madgwick, "Open source IMU and AHRS algorithms." [Online]. Available: https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/
[9] K. Kamperman, "Open Sound Control (OSC) Monitor." [Online]. Available: https://www.kasperkamperman.com/blog/processing-code/osc-datamonitor/
[10] P. Mantione, "Max/MSP: The History and Utility of this Iconic Language from Cycling '74." [Online]. Available: https://waveinformer.com/2024/04/15/max-msp-the-history-and-utility/
[11] T. A. Place and T. Lossius, "Jamoma: A Modular Standard for Structuring Patches in Max," in *International Conference on Mathematics and Computing*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:37150607