IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 14, 2021, accepted July 28, 2021, date of publication August 16, 2021, date of current version August 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3104903

A Wireless Instrumentation Control System Based on Low-Cost Single Board Computer Gateways

DANIEL ENÉRIZ ORTA, (Graduate Student Member, IEEE),
NICOLÁS MEDRANO, (Senior Member, IEEE),
AND BELÉN CALVO, (Senior Member, IEEE)
Faculty of Science, University of Zaragoza, 50009 Zaragoza, Spain

Corresponding author: Daniel Enériz Orta (eneriz@unizar.es)

This work was supported in part by the Ministero de Ciencia e Innovación, through the Research Project under Grant PID2019-106570RB-I00/AEI/10.13039/501100011033, and in part by the Government of Aragon Ph.D. Grant BOA20201210014.

ABSTRACT Nowadays, most of the automatized measurement processes are carried out by VISA (Virtual Instrument Software Architecture) compatible instruments, that execute the instructions provided by a host computer connected through wired standard buses, as USB (Universal Serial Bus), GPIB (General-Purpose Instrumentation Bus), PXI (PCI eXtensions for Instrumentation) or Ethernet. To overcome the intrinsic limitations associated to these wired systems, this work presents an instrumentation control system based on the IEEE 802.11 wireless communications standard. Intended for instruments having a USB control port, this port is connected to a gateway based on a compact Raspberry Single Board Computer (SBC) and thus the instrument can be connected to the host computer via Wireless Fidelity (WiFi), easily allowing the deployment of an ad-hoc instruments communication network in the working area or its connection to a previously deployed general purpose WiFi network. Developed under Python, the operation commands, wireless link protocol, and USB connection allow two modes of operation to provide system flexibility: a live mode, where commands are sent individually from the host computer to the selected instrument; and a standalone mode, where a full measurement process can be entirely downloaded in the gateway to be autonomously executed on the instrumentation. The system performance in both operation modes, distance of operation, time latencies, and operating lifetime in battery operation have been characterized.

INDEX TERMS Wireless instrumentation control, single-board computer, VISA, remote measurement.

I. INTRODUCTION

Automatic measurement processes are mainly performed using wired-controlled instruments, via either specific instrumentation buses such as GPIB (General-Purpose Instrumentation Bus) or PXI (PCI eXtensions for Instrumentation), or general-purpose communication buses such as USB (Universal Serial Bus) or Ethernet. These buses allow straightforward communication between the corresponding instruments and the host computer that manages the measurement process, sending the suitable commands and queries to the instruments and receiving their answers to be processed.

The diversity of available buses enables to control instrumentation in multiple scenarios: USB is nowadays a standard in low-range communications, covering distances of few meters, but its use is dismissed in applications covering areas of several tens of meters or more. As an alternative, GPIB offers good noise isolation and distance coverage, but it highly increases the infrastructure cost due to specific hardware adaptors, cables and connectors are required. However, in general, for all these approaches, the lack of flexibility of measurement systems based on wired-communications hinders the rearrangement of the instrumentation, preventing or even making impossible the implementation of wide-area flexible measurement systems.

Therefore, it is of interest to have a wireless instrumentation control system that improves the flexibility of the measurement system configuration, facilitating the relocation of the instrumentation deployed in large areas, while being able to manage the measurement system in the same way as using traditional wire-based instrumentation control, as it is done in other RF applications: for instance, the

The associate editor coordinating the review of this manuscript and approving it for publication was Paulo Mendes.

IEEE 802.15.1 (Bluetooth) or other proprietary wireless communication protocols let the development of wireless computer peripherals, like keyboards and mice, microphones, printers, etc. The computer connection to the internet has also become wireless, replacing the Ethernet wired connection with the IEEE 802.11 (WiFi, Wireless Fidelity) RF standard. All of these changes enhance the mobility of the systems while keeping the same operating mode for the end-user.

Focusing on the area of instrumentation systems, different solutions realize the wireless acquisition of physical magnitudes, such as data loggers [1]–[3], Wireless Sensor Networks [4]–[6], or SCADAlink [7], [8], being all of them sensor-oriented solutions. Considering instrument-oriented solutions, some commercial wireless solutions are available, as the IkaLogic wireless oscilloscope probe [9] or the Fluke 3000 FC wireless multimeter [10], but both of them are proprietary instruments not compatible with other mainstream systems. Following the goal of a wireless control system for general laboratory instrumentation, some instrument manufacturers have introduced their own implementation, as the Agilent E5810B gateway [11], that connected to a WiFi Access-Point (AP) enables the wireless control of GPIB, RS-232, and USB instrumentation; or the Tektronix TekCloud [12], a cloud-based solution that allows a real-time remote oscilloscope control. These proprietary solutions attempt to improve the instrument mobility enabling their remote control, though the wireless use of the instrumentation is not natively supported, requiring the use of additional wireless adapters.

To the best of the authors' knowledge, there are few RF instrumentation control systems based on standard wireless communications in the technical literature [13]–[15]. A Bluetooth-based interface is presented in [13], in which a GPIB interface is emulated using an FPGA (Field-Programmable Gate Array), thus eliminating the need for instrument wiring. In [14], [15], a more general interface is described, in which the RS232, GPIB, and USB buses are emulated using the IEEE 802.15.1 standard. This solution supports a variety of buses, allowing to control any type of instrument in practice at a reduced cost. However, this Bluetooth-based system presents three main limitations: (i) the maximum number of instruments that can be actively controlled is limited to 7, due to the IEEE 802.15.1 characteristics; (ii) the limited coverage range, between 10 m and 50 m under controlled indoor environments with two obstacles; and (iii) the need of a custom board per each of the instruments to be connected.

To get a general system as affordable and low-cost as possible, overcoming the aforementioned inherent lack of scalability, coverage range, and accessibility, this work presents a WiFi-based solution [16]. Compared to the Bluetooth protocol, the WiFi standard highly increases the limit of active devices connected in a network, as well as allowing a larger distance range for the system, thus extending the available measurement area. Besides, while the Bluetooth-based solution requires a wireless module connected to each instrument in the system, the proposed WiFi solution uses a gateway to interface the wired connection and the wireless network, so that a single wireless gateway can control several instruments if they are close enough. Instead of using a custom board, the gateways used in this work are based on a commercial low-cost Raspberry Pi Zero W SBC (Single Board Computer), getting a more accessible solution.

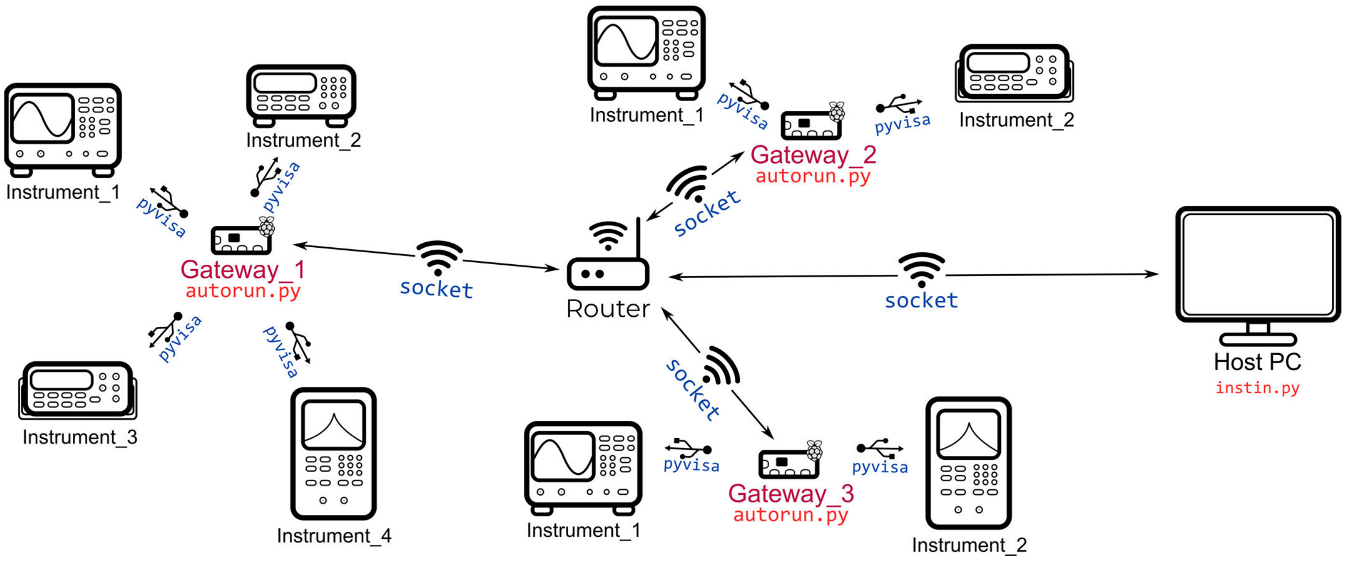


FIGURE 1. An illustrated scheme of the proposed wireless instrumentation control system. The host PC is connected to the same WiFi network than the Raspberry-based gateways. Each gateway can be connected up to four instruments using their USB control ports. The python packages used to manage the communication protocols are shown in blue, and the developed control libraries (instin.py for the host PC and autorun.py for the gateways) are shown in red.

This paper is organized as follows: Section II describes the communication protocols and the gateway hardware and software used to develop the system. Section III shows the system operation, including the two different operation modes: the so-called live mode, where commands are sent individually to the corresponding instrument; and a standalone mode, where a full measurement process can be entirely downloaded in the gateway to be autonomously executed. Section IV presents the test results obtained in two realistic measurement situations, one for each operation mode. In Section V, a battery-powered gateway scenario is evaluated. Finally, in Section VI some conclusions are drawn. Additionally, there is a release of the open-source code developed in this work.[1]

II. PROPOSED SYSTEM OVERVIEW

Figure 1 shows the general scheme of the proposed wireless control system for VISA compatible and SCPI compliant instruments with a USB control port. It relies on the use of gateways based on a low-cost SBC, which are connected to the USB port of the instruments, while its communication to the host computer is carried out via WiFi. The WiFi protocol linking the host and gateways, and the VISA link between the gateway and the instruments are both developed in Python to be user-friendly, only requiring the use of a custom Python package to control the instrumentation through the wireless system. In this section, the main resources of the system are discussed.

A. WIRELESS COMMUNICATIONS

Because of its point-centered architecture and extended coverage range, the widely used IEEE 802.11 is the standard selected to link the instrumentation to the computer in charge of driving the measurement process. The maximum number of gateways that can be connected to a host computer is limited by the maximum number of RF links allowed by the WiFi router, typically up to 256 devices for a household router, which is enough, in general, for large measurement systems. On the other hand, the signal range depends on the Access Point (AP) hardware, with a typical value of 30 m indoor. This range can be enhanced by increasing the number of APs, thus extending the areas where the instrumentation can be located. Additionally, using WiFi as communication protocol allows the use of already deployed networks, avoiding in this case the need for a dedicated network. Another benefit of the WiFi protocol is the remote control since the management of the instrumentation can be accomplished from anywhere if internet is available.

The protocol selected to manage the WiFi connection, to send the information from one point to another, is the Transmission Control Protocol (TCP) [17]. TCP is simple and robust, besides that is one of the main transmission protocols on the internet, from which other popular protocols are based, such as the HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), or FTP (File Transfer Protocol).

[1] https://github.com/eneriz-daniel/instin

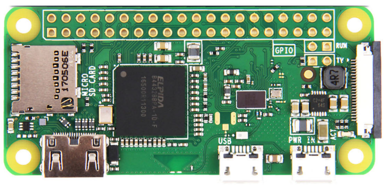
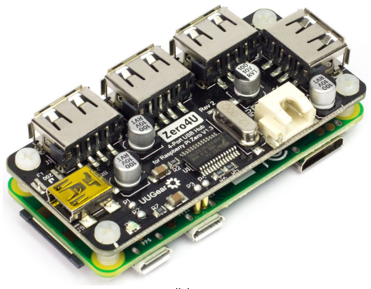
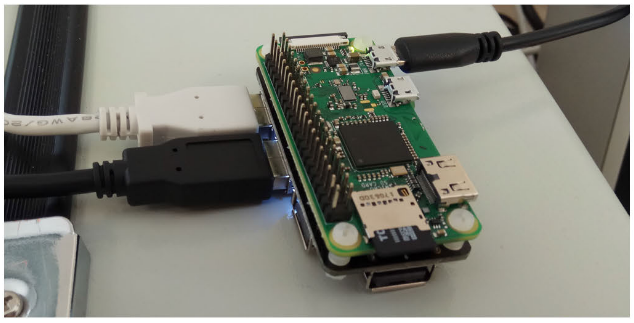

(a)

(b)

(c)

FIGURE 2. (a) Raspberry Pi Zero W top view, (b) Zero4U USB hub coupled to the Raspberry Pi Zero W and (c) Raspberry-based gateway with a microUSB feeding the gateway, connected to the control port of two instruments through two USB type A connectors.

B. INSTRUMENTATION INTERFACE

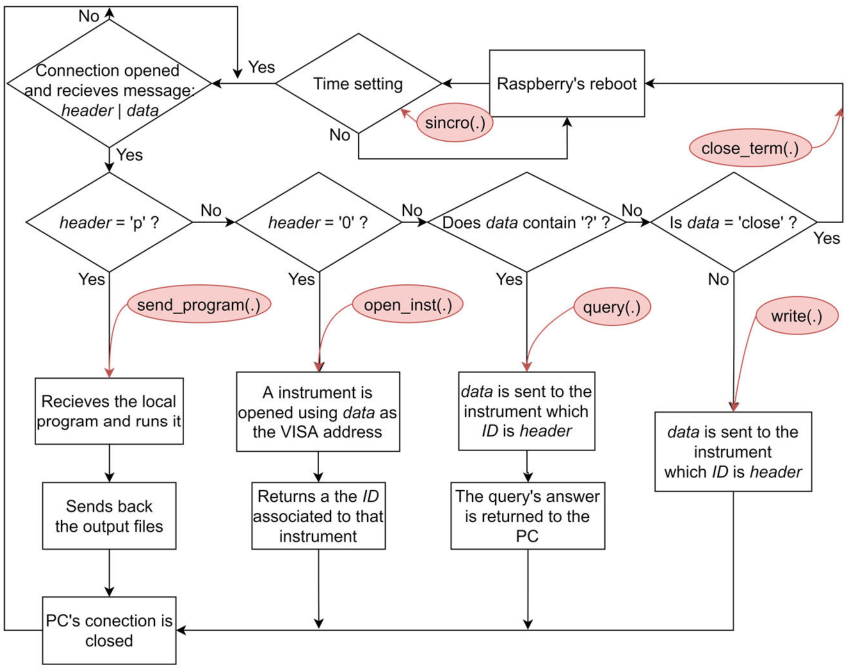
The gateway is connected to the instrument through its available USB control port, so that it can be controlled using the Virtual Instrument Software Architecture (VISA) protocol, an industry-standard Application Programing Interface (API) for instrumentation. It allows to send commands and queries to the instruments and to receive the corresponding answers. Depending on the manufacturer and instrument type, commands and queries present different formats, being the most common ones C-like functions and, mostly, the Standard Commands for Programmable Instruments (SCPI).

C. HARDWARE

The gateway interfaces the host PC and the instrumentation, so that the device must be compatible with the hardware and software of the different communication protocols used in both directions. The hardware selected as the gateway is a Raspberry Pi Zero W (Fig. 2a), an SBC with a small form factor (65 mm × 30 mm × 5 mm) and low price

```
def sincro(IP, PORT):
    Sends the PC time to the gateway


def open_inst(IP, PORT, VISA):
    Sends '0'+VISA to the gateway
    Returns the instrument ID


def write(IP, PORT, InstID, command):
    Sends InstID+command to the gateway


def query(IP, PORT, InstID, command):
    Sends InstID+command to the gateway
    Waits for a gateway's answer
    Returns de answer


def close_term(IP, PORT):
    Sends '0'+'close' to the gateway


def send_prog(IP, PORT, NAME_IN, *NAMEs_OUT):
    Sends the input program to the gw
    Waits for a gateway's answer
    Receives and saves the output files
```

(a)


No
Connection opened and recieves message: header | data
Yes
Time setting
Raspberry's reboot
sincro(.)
No
close_term(.)
header = 'p' ?
No
header = '0' ?
No
Does data contain '?' ?
No
Is data = 'close' ?
Yes
Yes
Yes
Yes
No
send_program(.)
open_inst(.)
query(.)
write(.)
Recieves the local program and runs it
A instrument is opened using data as the VISA address
data is sent to the instrument which ID is header
data is sent to the instrument which ID is header
Sends back the output files
Returns a the ID associated to that instrument
The query's answer is returned to the PC
PC's conection is closed

(b)

FIGURE 3. **(a) List of instin.py main functions with their arguments and brief description. (b) Flowchart of autorun.py. Functions developed in instin.py package (in red) give access to the different functionalities in autorun.py, according to the header included in the outgoing message from the host PC.**

(about 10 €), that includes a Broadcom BCM2835 system-on-chip (SoC) including an ARM11 CPU running at 1 GHz and 512 MB of RAM, a 2.4 GHz WiFi antenna and a microUSB port. To make feasible the connection of several instruments to a gateway, a Zero4U USB hub (Fig. 2b) is connected to the microUSB port of the main Raspberry board, providing four additional USB type-A connectors at a cost below 10 €.

D. SOFTWARE

The gateway runs on Raspbian Operating System (OS), a Debian version specifically designed for this SBC. On this OS, the software applications have been developed on Python, specifically CPython, the most used Python distribution. Moreover, available Python packages to manage both the TCP protocol and the VISA API are used.

Figure 1 shows the corresponding Python packages required for the hardware connections (*socket* and *pyvisa*, in blue). In particular, the *socket* package [18] allows the control of low-level networking interfaces, covering the TCP protocol, among others. Once the socket objects are initialized using the corresponding package functions, the client device can open a connection to a host whose Internet Protocol (IP) address and connection port are known, therefore starting a communication between them. By configuring the gateways as communications hosts and the measurement PC host as communications client, it is possible to send commands to any instrument whenever required through the functions included in the *socket* package.

On the other hand, *pyvisa* [19] is a community-developed package that consists in a full VISA front-end. In addition, to overcome the limitations derived from the incompatibility of VISA proprietary drivers (National Instruments, Keysight) with some OS (as Debian), in this work we will use the *pyvisa-py* package, a *pyvisa* project extension consisting of a VISA backend fully compatible with the proposed Raspberry gateway OS.

Finally, the Python files *instin.py* and *autorun.py* (in red in Figure 1) contain the code developed in this work to fully manage the system operation over the measurement processes.

III. SYSTEM OPERATION

After presenting the main components of the wireless instrumentation control system, this Section describes its operation. Software is organized in two Python files, one running on the measurement host, while the other one is running on the gateways physically connected to the instruments.

A. HOST PACKAGE

The file *instin.py* includes a custom package developed as a front end of the system that is executed on the host PC. It consists of a set of functions emulating a VISA library, including opening an instrument connection, sending a command, or querying for information (Figure 3a). All these functions have been adapted to the wireless platform by incorporating two additional arguments: the gateway IP in which the instrument is connected, and the opened port for the wireless communication. They outcome ASCII-based arbitrary-length data packets, which are sent through TCP to the selected gateway. Packets contain a byte header whose value identifies the function class (*open*, *write*, *query*, etc.). The rest of message corresponds to the data required by the gateway to perform the chosen function; for instance, the SCPI command to perform a write operation, or the VISA address required to start an instrument connection.

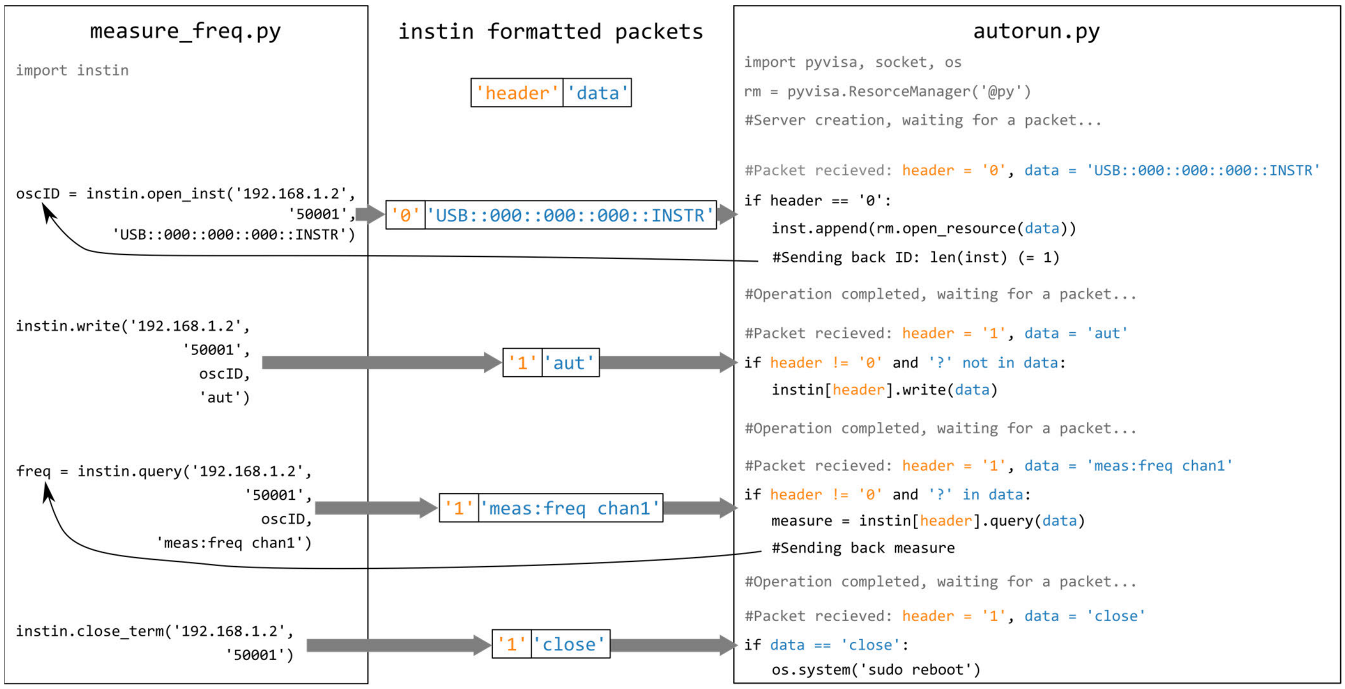
B. GATEWAY PROGRAM

The file *autorun.py* contains the instructions required for the system backend operation. This file is automatically launched on the SBC every time the Raspberry-based gateway boots, and it keeps scanning for connections from the host PC. As the packets sent from the host through *instin.py* have the format described above (1-byte header and variable-length data), *autorun.py* is able to properly identify the functions coming from the host to be executed. A detailed flowchart of *autorun.py* is shown in Figure 3b, where the different header options are shown, including two different execution modes, live (*header* $\neq$ ‘p’) and standalone (*header* = ‘p’).

C. LIVE MODE

In live mode operation, the host sends sequentially the instructions to the instruments involved in the measurement, so that the instrumentation placed in different gateways are controlled using a common single program, similar to how a measurement process is executed using wired protocols. This mode can be then suitable when the operation of instruments in different gateways have to be coordinated by a central host.

Figure 4 shows an example of a measurement process using this live mode. The first time the host connects to an instrument, the function *open_inst* in the package *instin.py* is called, including as function arguments the instrument VISA address as well as the IP and the port of the gateway where the instrument is connected. A data packet is sent to the gateway (Figure 4); the header (in orange) is the character ‘0’, and the rest of message is the instrument VISA address (in blue). Once executed, *open_inst* will return the number or ID identifying the instrument that is provided by the gateway. After the instrument communication is opened, the user can control its operation using the *write* and *query* functions implemented in *instin.py*. Both functions will require as arguments the gateway IP and port, together with the instrument ID and the SCPI command. The function will link to the gateway identified by its two first arguments, combining the ID and instrument command (header and data, respectively) in the packet sent. In the case of the function *query*, the host computer waits for an answer until it is received, or a timeout occurs. Concerning the gateway operation, *autorun.py* recognizes a command as a query due to the presence of a question mark (‘?’ character). Finally, once the measurement process is completed, the function *close_term* included in *instin.py* in the host PC sends a reboot instruction to the gateway, closing the connection.



```
import instin

oscID = instin.open_inst('192.168.1.2',
                         '50001',
              'USB::000::000::000::INSTR')

instin.write('192.168.1.2',
             '50001',
             oscID,
             'aut')

freq = instin.query('192.168.1.2',
                    '50001',
                    oscID,
                    'meas:freq chan1')

instin.close_term('192.168.1.2',
                  '50001')
```

```
import pyvisa, socket, os
rm = pyvisa.ResourceManager('@py')
#Server creation, waiting for a packet...

#Packet recieved: header = '0', data = 'USB::000::000::000::INSTR'
if header == '0':
   inst.append(rm.open_resource(data))
   #Sending back ID: len(inst) (= 1)

#Operation completed, waiting for a packet...
#Packet recieved: header = '1', data = 'aut'
if header != '0' and '?' not in data:
   instin[header].write(data)

#Operation completed, waiting for a packet...
#Packet recieved: header = '1', data = 'meas:freq chan1'
if header != '0' and '?' in data:
   measure = instin[header].query(data)
   #Sending back measure

#Operation completed, waiting for a packet...
#Packet recieved: header = '1', data = 'close'
if data == 'close':
   os.system('sudo reboot')
```

FIGURE 4. Example of a basic measurement program, measure_freq.py (left), the instin formatted packets sent for each function (middle) and a simplified version of autorun.py (right) executed in the gateway to perform those operations.

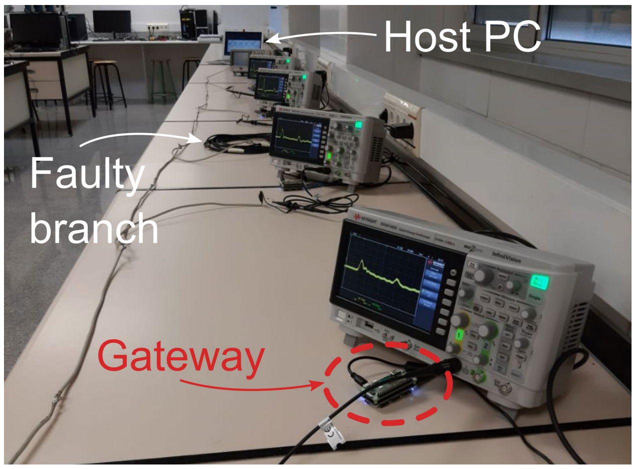


FIGURE 5. Deployed coaxial network and the distributed measurement system.

D. STANDALONE MODE

This mode fully executes a measurement process in each gateway autonomously, thus making possible through local operation to complete independent measurement processes, the parallel characterization of devices in batch, or measurements where the transmission delay due to the wireless communication can affect the required frequency of acquisitions.

Standalone mode is activated by calling the host function *send_program* included in the *instin.py* package (Figure 3a), and whose arguments are the name of the Python file that will be sent from the host for its execution in local mode, the IP address and port of the gateways in which the code must be downloaded and executed, and the names of the files where measurements and results will be saved at the end. As it is shown in Figure 3b, the header of the message sent to the selected gateways is the character 'p', while the data of this first packet includes the name of the Python file to be downloaded. After the local program is executed in the corresponding gateway, the result files are sent back to the host. Unlike the case of live mode, where additional information related to the RF host-gateway connection must be included in the measurement commands sent from the host, the functions used to control the instrumentation in standalone mode are those included in the *pyvisa* package, all included in the previously downloaded local program.

IV. SYSTEM PERFORMANCE

To verify the functionality of the proposed system, two measurement tests, corresponding to the two operation modes described above, have been carried out. Additionally, the latency introduced by the wireless system has been also characterized.

A. DISTRIBUTED SYSTEM

An example of a distributed (e.g. covering few tens of meters) monitorization and measurement system using the live mode is fault detection in a data transmission system using coaxial cable (either by short or open circuit). To emulate this situation, a coaxial system with access to N branches (Figure 5) is deployed placing 50 Ω resistors at the endings of the network signal sockets. An open-circuit fault is introduced in the branch labeled A in Figure 6, at 10.4 m from its connection to the mainline. To determine the position of the fault in the transmission system, a Tektronix AFG3252 arbitrary function generator is connected to the main coaxial line to generate pulse signals entering the network, while the signal behavior is monitored by means of four Agilent DSOX1102G oscilloscopes.

Given that the reflection coefficient Γ in a coaxial cable can be defined as [20]:

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{1}$$

where Z_0 is the characteristic impedance ($Z_0 = 50\ \Omega$) of the wire and Z_L is the load impedance, when the load impedance is extremely high ($Z_L \rightarrow \infty$, open circuit) coefficient $\Gamma = +1$ (Fig. 7), while $\Gamma = 0$ when a $Z_L = 50\ \Omega$ socket is used.

Since the pulse propagation velocity v is known [21], the faulty open-circuit position can be determined by measuring the delay between the forward and backward signals (transmitted and reflected signal at the open

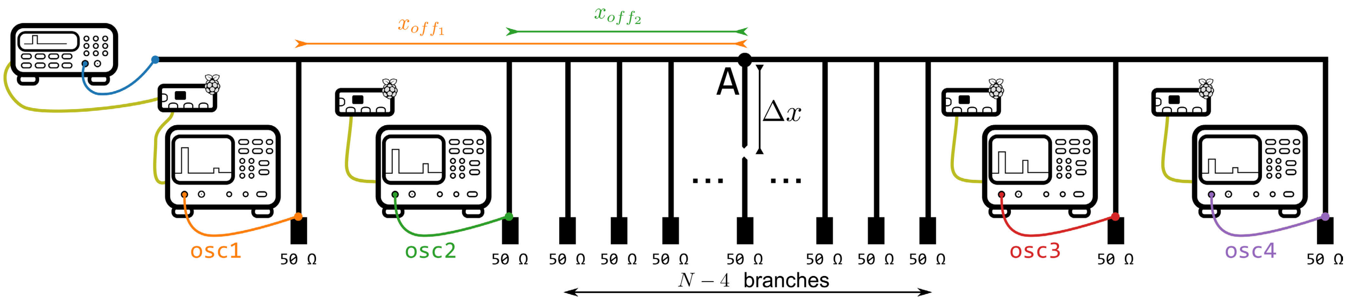


FIGURE 6. Scheme of the coaxial network and its measurement system. An arbitrary function generator (AFG) is sending pulses along the main line, where are split into the N branches (N = 5 in our test case). The first two and the last two have an oscilloscope probing the signal at their end, where two pulses are expected to display, the pulse sent by the AFG and the reflected on the open circuit. Additionally, the union of the faulty branch with the main line is marked with the letter A. The distance between A and the open circuit (Δ*x*) and the distance between A and the union of the first two branches to the main line (*x* of f_1 and *x* of f_1) are also drawn.

circuit, respectively) at the end-of-branch sockets, given by:

$$\Delta t_i = \begin{cases} 2\dfrac{\Delta x + x_{off_i}}{v} & \text{before the fault} \\ 2\dfrac{\Delta x}{v} & \text{after the fault} \end{cases} \quad (2)$$

where Δx is the distance between point A and the open circuit and x_{off_i} is the distance between A and the i-th oscilloscope branch intersection with the mainline. According to eq. (2), to determine the location of the fault two oscilloscopes are connected to the last two sockets so that their delays are used to estimate Δx. The remaining two oscilloscopes are in the first two sockets, and their delays allow the calculation of each offset distance, x_{off_1} and x_{off_2} once estimated Δx (Fig. 6). Note that only two measurement instruments are required to determine the position of the circuit defect (one before and one after the defective branch). However, though the use of four devices is redundant, we selected this setup configuration in order to evaluate the differences in the estimation of the open-circuit position in four different instrument position combinations.

A Python program is coded to use *instin* to control the AFG and the oscilloscopes to estimate the fault position using the measured delays in each oscilloscope. This program sets the AFG to send pulses of 1 V amplitude, 1 MHz frequency, and 10 ns of duty cycle. Oscilloscopes are configured to fit the screen to display the signal shown in Fig. 7 and measure the time delays, Δt_i, which are used to calculate Δx, x_{off_1} and x_{off_2}. The obtained and actual values are available in Table 1.

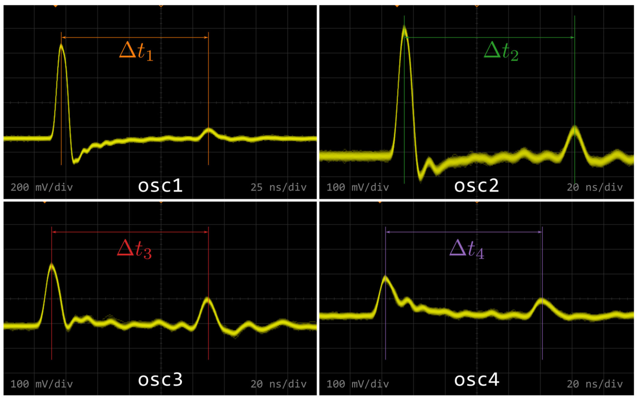
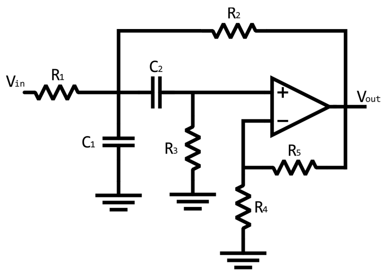


FIGURE 7. Oscilloscopes screenshots showing the measured signals. Marked the time delay between the original pulse and the reflected on the open circuit.

TABLE 1. Open circuit position.

i	1	2	3	4
Δt_i (ns)	116.9(2)	107.5(2)	99.1(2)	98.7(2)

Distance parameter	Δx	x_{off_1}	x_{off_2}
Actual value (m)	10.4	1.87	0.91
Measured value (m)	10.38(1)	1.87(4)	0.92(4)



	BPF1	BPF2
R_1	36 kΩ	
R_2	10 kΩ	
R_3	8.2 kΩ	
R_4	390 Ω	
R_5	1 kΩ	
C_1 C_2	10 nF	1 nF

FIGURE 8. Sallen-Key butterworth bandpass filter topology.

B. PARALLEL CHARACTERIZATION

A realistic measurement scenario in which the standalone mode can be assessed is the batch characterization of circuit samples. For this, four gateways are used to characterize eight Sallen-Key Butterworth bandpass filters [22] (Figure 8), one half designed with a central frequency $f_0 = 2$ kHz, a quality factor Q = 8 and a gain $A_0 = 4$ (BPF1, nominal values), and the rest presenting $f_0 = 20$ kHz, Q = 5 and $A_0 = 4$ (BPF2, nominal values). The transfer function is given by [22]:

$$H(s) = \frac{A_0 \frac{\omega_0}{Q} s}{s^2 + \frac{\omega_0}{Q} s + \omega_0^2} \quad (3)$$

Each filter is connected to an Agilent DSOX2002A oscilloscope (Figure 9), that provides the input signal to the filter and reads the output, thus measuring the circuit gain and phase delay in a specific frequency range. Oscilloscopes are connected in pairs to a gateway, that provides the WiFi link to the host PC. To start, the standalone mode is called in the four gateways utilizing the *send_program* function of *instin*, that sends the characterization program to the gateways. Once downloaded from the host, the program independently runs on every gateway, managing the operation of each couple of oscilloscopes. The corresponding operations on the circuits under test are done, acquiring the gain and the phase delay of the filters at 100 frequency values. Once each measurement is finished, the files generated with the characterization data are sent to the host PC. The time required for each measurement process, from the call of the function *send_program* to the end of the characterization process, is similar for every gateway, with a mean run time of 4 minutes and 19 seconds. Once all the measurements have been received by the host PC, the measured filter parameters (A_0, f_0, Q) are extracted (Table 2). Figure 10 shows the measured responses *vs.* the corresponding modeled transfer functions (equation (3), with parameters in Table 2). Dispersion in the measured values is due to tolerances in the R (5 %) and C (10 %) passive components conforming each filter.

C. LATENCY DUE TO RF OPERATION

Compared to the control of instruments using USB communications, wireless control is expected to add an extra time delay due to the process of sending-to and receiving-from the instrumentation. Besides, this time delay is expected to be instrument-to-router distance dependent [23]. To analyze

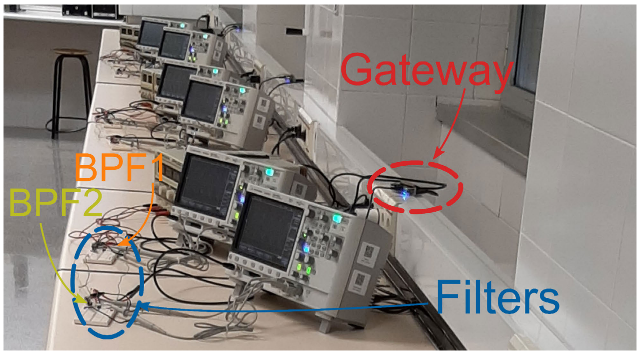


FIGURE 9. Parallel characterization of eight bandpass filters using eight different oscilloscopes controlled using four gateways working in standalone mode.

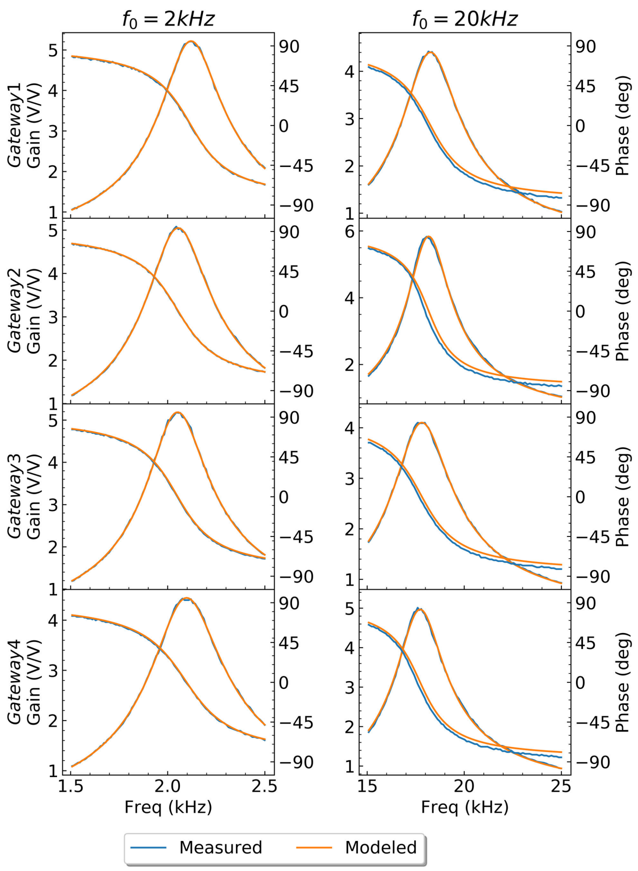


FIGURE 10. Results of the eight BPF1 and BPF2 sallen-key butterworth bandpass filters' characterization (in blue) and their fitted models (in orange).

this effect, the average times of the *instin write* and *query* functions sent from a host PC to control an Agilent DSOX2002A oscilloscope are estimated from a set of SCPI *query* and *write* commands. An oscilloscope is located at four different distances from the AP (an Alpha ASL-26555 router mounting a Ralink RT3052F SoC at 384 MHz, that enables a 2.4 GHz IEEE802.11n WLAN [24]) in a real laboratory facility, comparing the measured times to the time delay obtained by directly using the USB port.

TABLE 2. Bandpass filters measured parameters.

f_0^* (kHz)		Gateway				Avg.
		1	2	3	4	
2	A_0	5.22	5.05	5.19	4.44	4.98(16)
	f_0^* (Hz)	2120	2054	2052	2098	2081(15)
	Q	6.97	6.53	6.75	5.94	6.54(19)
20	A_0	4.40	5.84	4.10	4.97	4.83(33)
	f_0^* (Hz)	18260	18167	17817	17728	17993(113)
	Q	6.57	8.67	6.28	7.41	7.23(46)

*The f_0 in the first column is the target design central frequency, while the rest are the extracted parameter values from the data acquired in the characterization processes.

TABLE 3. Time delay at different distances.

Distance (m)	USB	6	15.5	18.2	26.4
Query time delay (ms)	224(20)	287(17)	289(17)	296(16)	564(30)
Write time delay (ms)	0.76(1)	61.3(7)	62.2(7)	68.3(8)	194(6)

As it is shown in Table 3, there is a significant difference in the time delay between a query and a write statement, as expected since the first one involves an answer sent from the instrument. Also, it is remarkable the similarity in the time delay at distances of 6, 15.5, and 18.2 m from router to instrument, far away from the time interval measured at a distance of 26.4 m, close to the AP coverage limit range.

V. BATTERY-POWERED GATEWAY

The ubiquity that can be achieved by using a battery powered measurement system will allow the employment of automatic controlled instrumentation in environments where no mains infrastructure is available. In this case, a battery-powered gateway may be useful, so that a truly autonomous wireless controlled instrument can be achieved, becoming a useful tool in fieldwork applications. For an adequate knowledge of the reliability of such measurement system, a suitable estimation of the lifetime of the battery

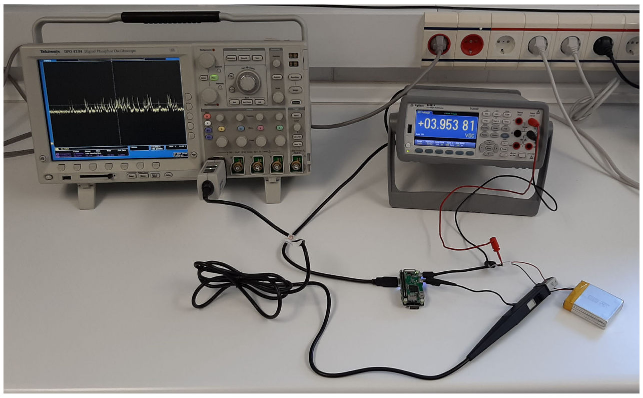


FIGURE 11. Gateway battery consumption measurement setup. The Tektronix DPO4104 oscilloscope monitors the dynamic current feeding the gateway using a Tektronix TCP0030 Hall-effect probe. The Agilent 34461A DMM is measuring the voltage in battery.

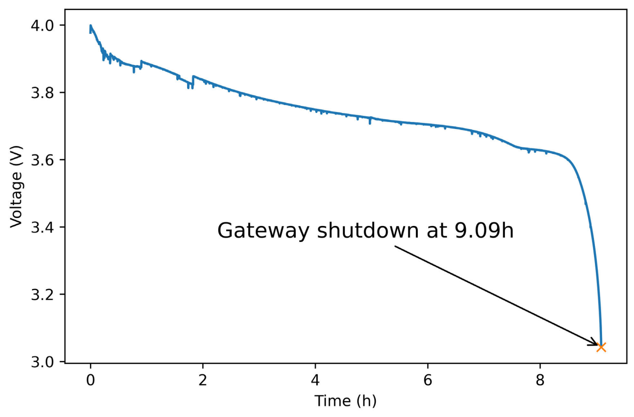


FIGURE 12. Experimental battery discharge curve.

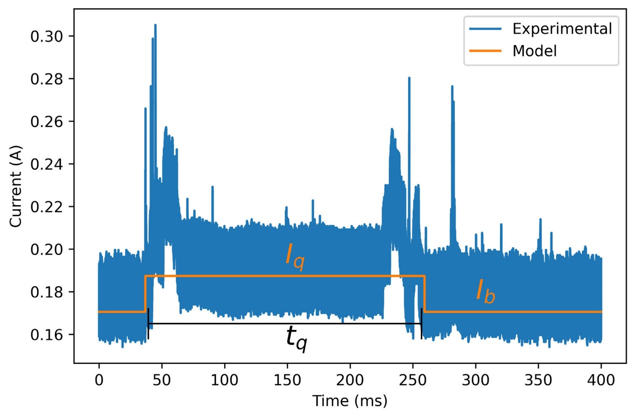


FIGURE 13. In blue, the dynamic current consumption in a query statement. In orange, the modeled average current levels in passive and active states.

feeding the proposed Raspberry-based gateway is essential. So, inspired by the methodology shown in [25], [26], we have applied the proposed wireless control system to characterize a 3.7 V-2000 mAh battery (nominal values) biasing the gateway itself while it controls an Agilent 34461A digital multimeter (DMM) (Figure 11). This DMM in turn is measuring the remaining voltage in the gateway battery. Figure 12 shows the voltage discharge curve obtained as a function of time at a sampling frequency of $f_q = 0.1$ Hz. The battery voltage remains close to the nominal value (3.7 V) until its complete drain, when the voltage dramatically decays after 9.09 hours of operation.

On the other hand, the current consumption profile of the gateway receiving, executing, and answering a query statement from the host is also acquired using a Tektronix DPO4104 oscilloscope connected to the gateway power line through a Tektronix TCP0030 current probe (Figure 11). This measure allows modelling the current consumption profile of a *query (.)* statement as a squared pulse (Figure 13), obtaining the average current value when the gateway is waiting for a host request (in passive mode, the low level, I_b) and when it is executing a query operation (in active mode, the high level, I_q).

Therefore, using both the voltage discharge curve and the current consumption model shown respectively in Figures 12 and 13, the remaining battery energy can be dynamically estimated. First, using the current model, the charge consumed in a query statement can be defined as:

$$Q_i = I_b(f_q^{-1} - t_q) + I_q t_q \tag{4}$$

where I_b, is the current consumption in passive mode and I_q, in active mode (Fig. 13) and t_q the time between the reception of a command from the host and the transmission of the measured values (i.e. the duration of the modeled current pulse). Then, the total consumed energy can be estimated using:

$$E_0 = \sum_i Q_i V_i \tag{5}$$

where V_i are the voltage values measured by the DMM (Fig. 12). Assuming that the maximum storable energy of the battery is the total consumed energy, it can be estimated in $E_0 = 6.735$ Wh, close to its nominal value, namely 7.4 Wh.

Finally, the lifetime of the battery-powered gateway can be modeled using the number of queries per unit of time (f_q) and the *query* current consumption model. Thus, the average energy consumed by a *query* statement can be modeled (Fig. 13) as:

$$\bar{E} = (I_b(f_q^{-1} - t_q) + I_q t_q)\bar{V} \tag{6}$$

where $\bar{V}$ is the nominal voltage of the battery. Thus, the lifetime of the gateway powered by a battery is given by:

$$t_{\max} = \frac{E_0}{\bar{E} \cdot f_q} \tag{7}$$

By applying the experimental models of current consumption profile (Fig. 13) and battery voltage decay curve (Fig. 12), the estimated lifetime of the gateway in the measurement process is $t_{\max} = 9.17$ h, remarkably close to the 9.09 h measured. It is also noticeable that equation (6) can be easily generalized to any number of queries and commands, extrapolating this model to any measurement process once the maximum storable energy of the battery, E_0, has been characterized.

VI. CONCLUSION

In this paper a wireless instrumentation control system has been proposed. It is based on a Raspberry Pi Zero W SBC acting as RF gateway connected to the USB port of the instruments to be controlled. The use of WiFi as the wireless communication standard allows the control of a large number of instruments at the same time, as well as the possibility of deploying the instrumentation in large measurement areas when required, without the need of additional infrastructure. Besides, it is compatible with most of the modern instrumentation, due to the popularity of USB control ports and the extended use of VISA-friendly and SCPI compliant instrumentation, making it easy to implement in any nowadays laboratory.

The software layer under the system is based on Python. The use of its standard package *socket* to manage the TCP connection ensures a robust communication channel between the host and the gateways. Instruments control is developed using the free community-developed *pyvisa* Python package and its VISA backend *pyvisa-py*. Combining the use of both packages provides adequate reliability for their use in research and industrial measurement processes.

The proposed control system allows the selection of two different operation modes: the live mode, in which instructions are sent individually to the instruments through their gateways via WiFi, coordinating its operation similarly to when using wired control standards. It presents advantages compared to standard wired systems when the instruments are far from the host computer and the wired solution is too expensive or unaffordable. The second mode, the standalone, is specifically designed to autonomously run a complete measurement process over each Raspberry-based gateway, thus avoiding the delays the wireless transmission can introduce in the process. It allows performing measurement sets in parallel in several distributed instrument modules, in which the control takes place locally, avoiding the delays associated to periodic connections to the host.

In order to test its characteristics, the proposed system has been applied in two different realistic scenarios, corresponding to the two implemented measurement modes. Additionally, to characterize the time delay introduced by WiFi communications, another test has been accomplished, in which the time duration of *query* and *write* commands at different distances between the instrument and the router have been determined. Finally, the feasibility of powering the gateways using batteries has been demonstrated, widely extending its application possibilities, even under conditions where no mains infrastructure is available.

Although the advantages listed above, our system currently only supports VISA compatible and SCPI compliant instruments having USB control port. The compatibility to non-SCPI instrumentation can be addressed by adapting the code in *instin.py* and *autorun.py* to its specific requirements. On the other hand, to deal with instruments using buses other than USB, as GPIB, Ethernet, RS232, etc., some slight hardware modifications on the gateway could be introduced, thus easily extending the range of its possible applications.

REFERENCES

[1] M. F. Hunar, E. A. Azrulhisham, K. Hamzani, W. M. W. Sulong, S. Abdullah, M. J. M. Basri, and M. A. Dandu, "GSM wireless datalogger of small hydro power generation system," in *Proc. 4th Int. Conf. Eng. Technol. Technopreneuship (ICE T)*, Aug. 2014, pp. 246–251, doi: 10.1109/ICE2T.2014.7006256.

[2] C. M. Vancea and L. Viman, "Wireless data logger for thermal validation systems," in *Proc. IEEE 17th Int. Symp. Design Technol. Electron. Packag. (SIITME)*, Oct. 2011, pp. 295–298, doi: 10.1109/SIITME.2011.6102739.

[3] M. Brandl, J. Grabner, K. Kellner, F. Seifert, J. Nicolics, S. Grabner, and G. Grabner, "A low-cost wireless sensor system and its application in dental retainers," *IEEE Sensors J.*, vol. 9, no. 3, pp. 255–262, Mar. 2009, doi: 10.1109/JSEN.2008.2012205.

[4] D. Antolín, N. Medráno, B. Calvo, and F. Pérez, "A wearable wireless sensor network for indoor smart environment monitoring in safety applications," *Sensors*, vol. 17, no. 2, p. 365, 2017, doi: 10.3390/s17020365.

[5] M. Sidorov, P. V. Nhut, Y. Matsumoto, and R. Ohmura, "LoRa-based precision wireless structural health monitoring system for bolted joints in a smart city environment," *IEEE Access*, vol. 7, pp. 179235–179251, 2019, doi: 10.1109/ACCESS.2019.2958835.

[6] S.-J. Hsiao and W.-T. Sung, "Building a fish–vegetable coexistence system based on a wireless sensor network," *IEEE Access*, vol. 8, pp. 192119–192131, 2020, doi: 10.1109/ACCESS.2020.3032795.

[7] R. Muthunagai and S. Rajkumar, "Remote monitoring of distribution transformer with power theft detection using PLC & SCADA," in *Proc. Int. Conf. Syst., Comput., Autom. Netw. (ICSCAN)*, Jul. 2020, pp. 1–4, doi: 10.1109/ICSCAN49426.2020.9262306.

[8] T. Turc, A. Gligor, and C.-D. Dumitru, "Web-based wireless sensor system for SCADA environment," *Procedia Eng.*, vol. 181, pp. 546–551, Jan. 2017, doi: 10.1016/j.proeng.2017.02.432.

[9] *IkaScope WS200*. Ikalogic. Accessed: Dec. 28, 2020. [Online]. Available: https://cdn.ikalogic.com/docs/ds/ws200-series-manual-EN.pdf

[10] *Fluke 3000 FC Wireless Multimeter User's Guide*. Fluke Corporation. Accessed: May 2014. [Online]. Available: https://dam-assets.fluke.com/s3fs-public/3000fc__umeng0100.pdf

[11] *Keysight E5810B LAN/GPIB/ USB Gateway User's Guide*. Keysight Technologies. Accessed: Jun. 6, 2018. [Online]. Available: https://literature.cdn.keysight.com/litweb/pdf/E5810-90004.pdf?id=2302065

[12] *TekScope Analysis Datasheet*. Tektronix. Accessed: Feb. 10, 2021. [Online]. Available: https://www.tek.com/datasheet/tekscope-analysis-datasheet-analyze-anywhere-anytime

[13] P. Fu, W. J. Ma, and C. J. Huang, "Design of wireless GPIB interface module based on bluetooth," *J. Phys., Conf. Ser.*, vol. 48, pp. 1279–1283, Oct. 2006, doi: 10.1088/1742-6596/48/1/238.

[14] L. Ferrigno, V. Paciello, and A. Pietrosanto, "A Bluetooth-based proposal of instrument wireless interface," *IEEE Trans. Instrum. Meas.*, vol. 54, no. 1, pp. 163–170, Feb. 2005, doi: 10.1109/TIM.2004.840245.

[15] L. Ferrigno, V. Paciello, and A. Pietrosanto, "Performance characterization of a wireless instrumentation bus," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 12, pp. 3253–3261, Dec. 2010, doi: 10.1109/TIM.2010.2047307.

[16] D. Eneriz, N. Medrano, B. Calvo, and J. Perez-Bailon, "A wireless instrumentation control system based on low-cost single board computers," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I MTC)*, Dubrovnik, Croatia, May 2020, pp. 1–5, doi: 10.1109/I2MTC43012.2020.9129142.

[17] W. R. Stevens and G. R. Wright, *TCP/IP Illustrated*. Reading, MA, USA: Addison-Wesley, 1994.

[18] Python Software Foundation. (Accessed: Apr. 28, 2021). *Socket—Low-Level Networking Interface*. Python 3.9.4 Documentation. [Online]. Available: https://docs.python.org/3/library/socket.html

[19] PyVISA Authors. (Accessed: Apr. 28, 2021). *PyVISA: Control Your Instruments With Python*. PyVISA 1.11.3 Documentation. [Online]. Available: https://pyvisa.readthedocs.io/en/1.11.3/

[20] S. Rosenstark, "Reflections on transmission lines," in *Transmission Lines in Computer Engineering*. New York, NY, USA: McGraw-Hill, 1994.

[21] D. Eneriz, N. Medrano, and B. Calvo, "Live demonstration: A low-cost wireless instrumentation control system," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seville, Spain, Oct. 2020, p. 1, doi: 10.1109/ISCAS45731.2020.9180681.

[22] A. S. Sedra and K. C. Smith, "First-order and second-order filter functions," in *Microelectronic Circuits*, 6th ed. New York, NY, USA: Oxford Univ. Press, 2010.

[23] F. Heereman, W. Joseph, E. Tanghe, D. Plets, L. Verloock, and L. Martens, "Path loss model and prediction of range, power and throughput for 802.11n in large conference rooms," *AEU-Int. J. Electron. Commun.*, vol. 66, no. 7, pp. 561–568, Jul. 2012, doi: 10.1016/j.aeue.2011.11.008.

[24] *RT3050/52 Datasheet*. Ralink Technology Corporation. Accessed: Aug. 14, 2008. [Online]. Available: https://datasheetspdf.com/pdf-file/785622/Ralink/RT3050/1

[25] D. Antolin, N. Medrano, and B. Calvo, "Analysis of the operating life for battery-operated wireless sensor nodes," in *Proc. 39th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Nov. 2013, pp. 3883–3886, doi: 10.1109/IECON.2013.6699755.

[26] D. Antolín, N. Medrano, and B. Calvo, "Reliable lifespan evaluation of a remote environment monitoring system based on wireless sensor networks and global system for mobile communications," *J. Sensors*, vol. 2016, pp. 1–12, Jan. 2016, doi: 10.1155/2016/4248230.

DANIEL ENÉRIZ ORTA (Graduate Student Member, IEEE) received the B.S. degree in physics and the M.S. degree in physics and physical technologies from the University of Zaragoza, Spain, in 2019 and 2020, respectively, where he is currently pursuing the Ph.D. degree. He is currently with the Group of Electronic Design, Aragon Institute for Engineering Research (GDE-I3A), where his research interests include the design of electronic systems, intelligent instrumentation, and the edge computing of neural networks.

NICOLÁS MEDRANO (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees in physics from the University of Zaragoza, Spain, in 1989 and 1998, respectively. He is currently a Full Professor of electronics with the Faculty of Physics, University of Zaragoza, where he is also a member of the Group of Electronic Design, Aragon Institute of Engineering Research. His current research interests include hardware implementation of neural network models for signal processing, smart sensor interfaces, wireless sensor networks, and intelligent instrumentation.

BELÉN CALVO (Senior Member, IEEE) received the B.Sc. degree in physics and the Ph.D. degree in electronic engineering from the University of Zaragoza, Spain, in 1999 and 2004, respectively. She is currently a member of the Group of Electronic Design, Aragon Institute of Engineering Research (GDE-I3A), University of Zaragoza. Her research interests include analog and mixed-mode CMOS IC design, on-chip programmable circuits, integrated optical receivers, low-voltage low-power monolithic sensor interfaces, and wireless sensors networks.