XXXVII SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES - SBrT'19, 29 DE SETEMBRO à 02 DE OUTUBRO DE 2019, RIO DE JANEIRO, RJ

RaspBed: a low cost experimentation platform based on Raspberry Pi for wireless communication studies

João Francisco Nunes Pinheiro, Ezequias Márcio Silva de Santana Jr, Tarcisio Ferreira Maciel

Abstract— **In this paper we propose a low cost platform for the study and teaching of radio communications, making it possible to reduce the gap between practice and theory in this area. The platform is based on experimental testbeds that are already functional in real world, allowing experimenters to test their developed theory in real world hardware. But since those tend to use expensive hardware, we aimed at using simpler but effective hardware, making this platform more accessible for deployment. We show here all components of the platform and what are their roles and costs. Also we provide a few examples of usage of the platform, guiding the user through its basic capabilities.**

Keywords— **Testbed, Raspberry Pi, USRP, Mobile Communications, GNU Radio, RTL-SDR**

I. Introduction

Wireless communications have experienced huge advances since the introduction of the 1st Generation (1G) of cellular systems – mainly focused on analog voice services – until the commercial deployment Internet Protocol (IP)-based fully digital 4th Generation (4G) systems – centered around modern multimedia services [1]. Moreover, with the popularization of smartphones and the traffic growth forecasts for the upcoming 5th Generation (5G) systems in 2020 and beyond [2], wireless communications will solidify their already indisputable position in the daily life of our modern connected society.

Along this evolution path, radio communication rose to a new level of relevance in telecommunications careers, since it is the fundamental basis of such wireless systems and their newest technologies (such as massive MIMO and mmWave communication) that play as pillars for the 5G wireless systems [1].

Consequently, many innovative solutions populated the literature of wireless communications, such as polar codes [3], hybrid beamforming, network-assisted Device-to-Device (D2D) and Vehicle-to-Anything (V2X) communications [1]. Most of these solutions are conceived, evaluated, and optimized following a theoretical approach of studies, mainly concentrated in software simulations and analytical studies, and have their physical concretization only in rare research platforms [4] or later when they arrive at the industry as products.

While it is relatively accessible for telecommunication professionals to get in contact with the theory of communications, the contact with hardware and study platforms is still quite limited nowadays despite the efforts of many institutions that made platforms available to the scientific community over the Internet [5].

On the one hand, the importance of radio communications is crescent and presses for improving the formation of future telecommunications professionals in both theory and practice. On the other hand, the costs of deploying radio platforms for studies limits the opportunities of these professionals to have a hands-on contact with radio communications, creating a learning gap between theory and practice. In this context, Software-Defined Radio (SDR) appears as a promising solution to reduce this gap, by offering the possibility of implementing different radio systems in software (with close-enough hardware limitation concerns such as real time sampling and synchronization) which are coupled to a flexible radio front-end that allows for real physical communication [6].

Despite cost reductions, SDR platforms for radio communication studies are still relatively expensive in Brazilian terms. Inspired by this problematic, in this paper we propose an Open Source experimentation platform that combines low cost SDR devices and computational resources and offers an alternative to assist in the theoretical and practical studies of telecommunications professionals. The remainder of this paper is organized as follows: Section II presents the platform components and specifications, for both hardware and software components. Section III presents how the platform can be used by an experimenter and show basic examples. Section IV presents the achieved conclusions of this work.

II. Platform Design and Description

Based on experimental testbed models such as [7], [8], this platform uses a server to allocate resources linking virtual machines and hardware resources, namely Universal Software Radio Peripherals (USRPs) and Raspberry Pi (RPi) boards equipped with RTL-SDR dongles, which enables practical studies on radio communications, making it possible for students and professors to demonstrate and implement practical experiments. Thus, basic concepts of this area can be seen and verified in the real world physical communications. The proposed testbed architecture can be seen in Fig. 1.

The testbed is composed of a main server, responsible for allocating virtual machines with or without USRPs attached to them, according to the requirements of the experiments, and RPi boards with RTL-SDR radios attached to them. Together, these two main components of this system can implement a complete radio transmission system, from the data source

João F. N. Pinheiro, e-mail: joao@gtel.ufc.br, Ezequias M. S. de Santana Jr, e-mail: ezequias@gtel.ufc.br, Tarcisio F. Maciel, email: maciel@gtel.ufc.br, Wireless Telecommunications Research Group, Teleinformatics Engineering Department, Center of Technology, Federal University of Ceará, Fortaleza, Brazil. T. F. Maciel was supported by CNPq under the grants 426385/2016-0 and 308621/2018-2. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

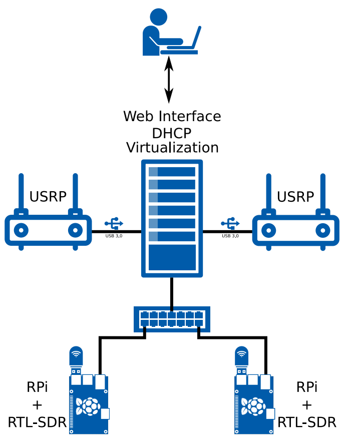


Fig. 1. System Architecture

up to the transmission in a first device through the radio channel and reception and data recovery in a second device. The components of the system will be described in detail in the sections in the sequel.

A. Hardware Components

1) Server: The server works as a central controller of the testbed, being the main component of the testbed. It runs the rolling release Arch Linux [9] distribution in order to ease updates and maintenance.

Due to the high computational demand, the following requirements are the minimum in order to accommodate at least 2 experimenters using the platform at the same time, maintaining good performance:

- Motherboard: Intel Desktop Board DH61BE
- CPU: Intel(R) Core(TM) i5-2310 CPU @ 2.90GHz
- SSD: Kingston A400 240GB
- Memory: 2x 4GB DDR3 1600MHz
- Power Supply: According to your hardware specifications
- USB 3.0 support is a must

Since it will be responsible for managing all allocation and managing software and some of the hardware components of the testbed, it can be said that this system is the main system of our platform. The main softwares will be described over the next sections, specially Section II-B

2) Universal Software Radio Peripherals: USRPs (see Fig. 2 right image) are SDRs created for usage in low cost experimentation, with support to a large number of open source softwares, such as GNU Radio. For this platform, the model used was the USRP B210 from Ettus Research, that operates on frequencies from 70 MHz up to 5 GHz [10]. They are equipped with USB 3.0, a Xilinx Spartan 6 FPGA and AD9361 Analog transceptors, capable of handling full duplex Multiple Input Multiple Output (MIMO) communications with 2 transmit or 2 receive antennas.

To manage these radios, we used the UHD drivers from Ettus Research git repository available in [11].

An important aspect for the best operation of these radios is the need for USB 3.0 controllers in the machines that will use the radios. This is required due to the high sample rate that they are capable of achieving, which goes up to 56 MHz for Single Input Single Output (SISO) systems and 30.72 MHz for 2×2 MIMO systems [12]. Because of that, some further configuration is required when allocating virtual machines with attached USRPs, so that these machines have compatible USB 3.0 controllers.

3) RTL-SDR: RTL-SDR are low cost SDRs that aim at the reception of AM, FM and digital television signals (see Fig. 2 left image).

The model used in our testbed is the RTL2838, which supports up to 3.2 MHz sample rate. This radio can be used only as receptor. It has a Elonics E4000 tuner, which operates in frequencies from 52 MHz to 1100 MHz and from 1250 MHz to 2200 MHz [13].

To manage it, we need the `librtlsdr`, which is a library with the necessary tools to manage RTL-SDR radios.

Fig. 2. RTL2838 device (left) and USRP B210 device (right).

4) Raspberry Pi: Raspberry Pi (RPi) (see Fig. 3) is a low cost, credit card sized computer board intended to accessible education in programming and computer skills. The main operating system used in this platform is Raspbian, a Debian-based Linux distribution. Nevertheless, other Ubuntu flavors and Windows IoT core are also supported.

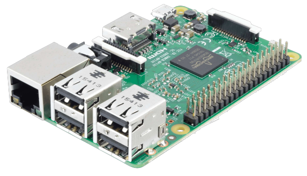


Fig. 3. RPi 3 board.

Among its specifications [14], it is worth mentioning its ARM Cortex A53 processor, 1 GB of RAM and its 10/100 Ethernet, 802.11n, Bluetooth 4.1, 40-pin GPIO interfaces, making it connectable and expansible to other technologies.

With these specifications, the RPi board has been chosen to host our RTL-SDR dongles, since it has enough capacity to handle the SDR resources, providing the users with the capacity to collect real-world data or, for simple experiments, processing it in real time. For more complex experiments, we recommend that the data collection be done at the RPi devices and that processing be done afterwards in a more powerful computer, so that the experiment's performance are not affected.

Due to compatibility issues with some RTL-SDR libraries, we opted to use the Ubuntu Mate linux distribution, in which the graphical interface was disabled for performance improvement, i.e., so we dedicate all available resources to the tasks associated with the connected RTL-SDR device.

The experimenters can access the RPi computers using a SSH terminal. In order allow them to use some graphical tools, such as the GNU Radio Companion [15], we have enabled X forwarding. Note that the use of the graphical interface is only recommended for simple experiments or during the design phase of the studies due to the limited processing power of the RPi boards.

B. Software Components

1) GNU Radio: GNU Radio is an Open Source tool aimed at signal processing. This software allows the usage of Software defined radios, like USRP and RTL-SDR, so that users can have the experience of implementing their own simulated or real radio systems.

It works on a block based IDE. These blocks implement most of signal processing functionalities and are organized in a "development" canvas that interconnects different blocks. The connections will form a complete signal processing chain, from the source of the data, like audio sinks or SDRs, passing by filtering, data processing, data recovery or any other steps required in a signal processing or radio transmission chain.

Due to its ease of use and completeness, we chose this software as main development environment to be supported by our platform. Its integration with available SDRs is supported by different blocks that are pre installed in all resources in the testbed. Finally, since this software is open source, it is open to developers to create and implement new blocks, promoting a larger variety of experiments to be executed.

2) Virtual Machines: In the testbed, all the computer resources are allocated to the experimenters in a virtualized way. In order to the users to have access to the USRPs, which require more computing power that the RPi can offer, we have created a Virtual Machine (VM) server. This server can allocate virtual machines with predefined software installed, that meet the requirements for usage with SDRs.

For better performance, we have used Kernel Based virtualization (KVM) with the `libvirt` interface, so we could control the hypervisor with Python scripts [16].

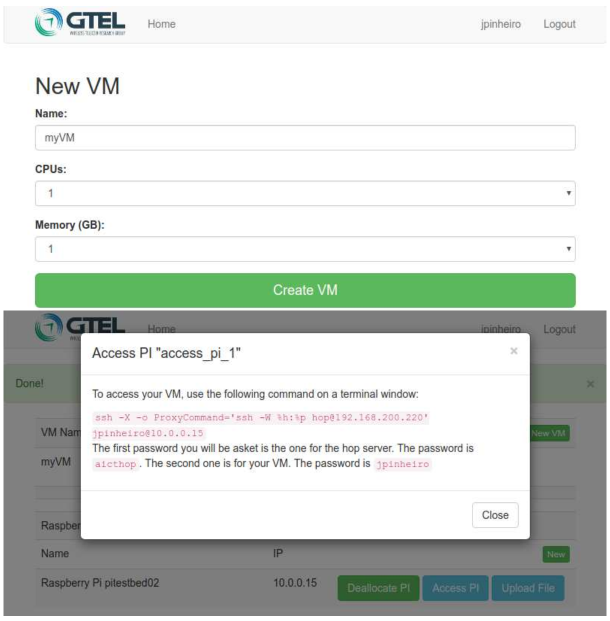
In our setup, the virtual machines come with Ubuntu 18.04 with all USRPs and RTL-SDR drivers installed and up to date, as well as with GNU Radio Companion [15].

These VMs are based on a previously created template, that is cloned every time a user requests a new VM. Each VM has a total storage capacity of 10 GB and can have 1 or 2 virtual CPUs, and 1 to 4 GB of RAM. The testbed users and administrators need to reserve the resources they need carefully: on the one hand, if they allocates a VM with too many resources, later they might not be able to allocate more VMs; on the other hand, with too few resources they might have a poorer performance than expected.

3) Web Interface: So the users could have an easy access to this platform, a Web Interface was created, making it possible for them to perform allocation and management of resources independently. This interface was developed using Flask, a Python micro-framework for web development. This provided a clean and easy to use front-end for the users, while also controlling the back-end and resource management. Illustrative screenshots can be seen in Figs. 4 to 6

This interface is freely available on our GitHub repository [17].

This interface has a user control, so new users can be added and removed as needed, and a resources allocation control.



New VM

Name: myVM

CPUs: 1

Memory (GB): 1

Access PI "access_pi_1"

To access your VM, use the following command on a terminal window:

```
ssh -X -o ProxyCommand='ssh -W %h:%p hop@192.168.200.220'
jpinheiro@10.0.0.15
```

The first password you will be asket is the one for the hop server. The password is `aicthop`. The second one is for your VM. The password is `jpinheiro`

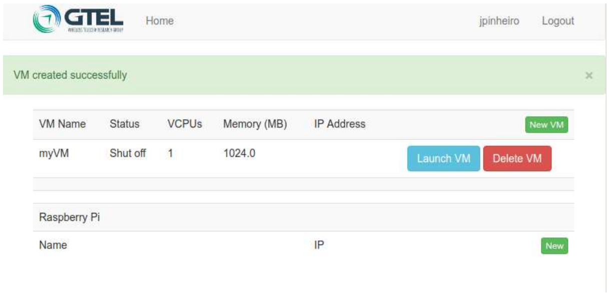
Name	IP
Raspberry Pi pitestbed02	10.0.0.15

Fig. 4. Virtual Machine allocation and access screens

These resources are basically virtual machines (with or without USRPs) and RPis.

Each hardware resource (USRP or RPi) needs to be preconfigured in two configuration files: one for the USRPs and other for the RPis. After that, the web interface needs to be restarted.

Finally, after the resources are allocated, the interface provides the user with commands to access the allocated resources via a remote SSH terminal, as shown in Fig. 6.



VM created successfully

VM Name	Status	VCPUs	Memory (MB)	IP Address
myVM	Shut off	1	1024.0	

Raspberry Pi

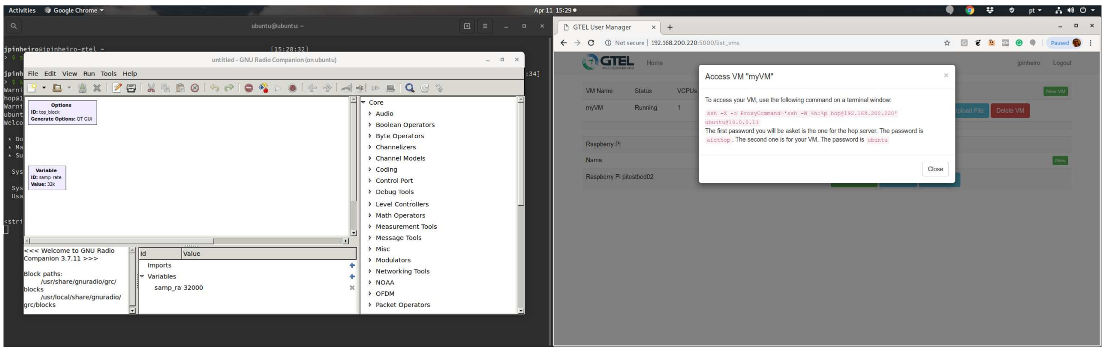
Name	IP

Fig. 5. Main interface

III. Usage Examples

Over the following sections, we are going to present three basic experiments used to test our platform. It is worth to mention that this platform is able to perform not only these, but many more experiments in regard to radio communications. The ones shown here are to be used as a starting point for the users to develop and test their own experiments.

The basic usage of this platform can be done in the following way: First the user logs in the platform. Following, it decides which resources he needs and allocates them, using the main interface seen in Fig. 5. Finally, the experimenter can access the resources with the instructions provided by the platform, like shown in Fig. 4, and perform the experiments that shall be introduced over the next sections. To finalize, the experimenter needs to release the used resources so the next


untitled - GNU Radio Companion (on ubuntu)

Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

```
<<< Welcome to GNU Radio
Companion 3.7.11 >>>

Block paths:
        /usr/share/gnuradio/grc/
blocks
        /usr/local/share/gnuradio/
grc/blocks
```

Id	Value
Imports	
Variables	
samp_ra	32000

GTEL User Manager — 192.168.200.220:5000/list_vms

Access VM "myVM"

To access your VM, use the following command on a terminal window:

```
ssh -X -o ProxyCommand='ssh -W %h:%p hop@192.168.200.220' ubuntu@10.0.0.13
```

The first password you will be asket is the one for the hop server. The password is `aicthop`. The second one is for your VM. The password is `ubuntu`

VM Name	Status	VCPUs
myVM	Running	1

Raspberry Pi

Name

Raspberry Pi pitestbed02

Fig. 6. Virtual machine with GNU Radio being accessed using the available instructions on the web interface

one can use them. A flowgraph of a basic experiment is shown in Fig. 7

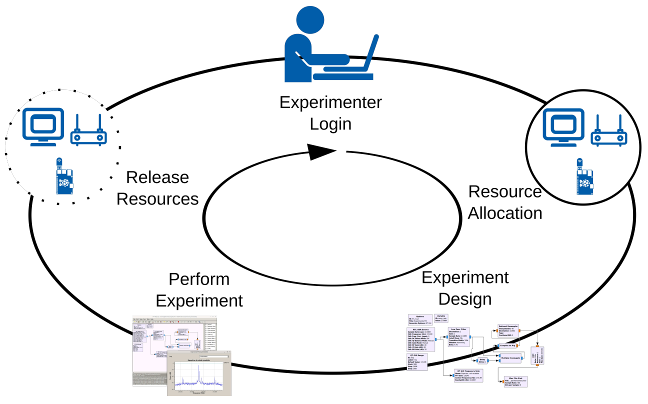


Fig. 7. Basic Experiment flowgraph.

The following experiments were based on [6] and their implementations are available on the platform's GitHub repository [17].

A. *Example* 1 *- Amplitude Modulation (AM)*

This experiment means to illustrate the performance of a communications system that makes use of AM. The user can alter different parameters from different blocks that are used to compose the transmitter and receiver in order to observe the effects of those parameters on the recovered data.

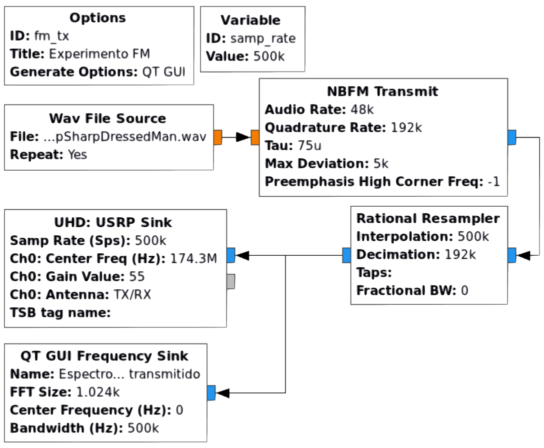
The transmitter is executed on a VM. Here we implement a dual tone signal using conventional amplitude modulation, where the user can tweak parameters such as the modulation coefficient and, after all operations, the AM signal is transmitted using the USRP using the *UHD:USRP Sink* block.

The receiver is implemented on the RPi and its where the AM signal is receiver on the RTL-SDR module, using the *RTL-SDR source* block. After, the recovery of the message is done, resulting in a *wav* file with the dual tone signal that was transmitted. On the spectrograms, we can see the received signal spectrum and the demodulated signal's envelope. This allows us to observer the effects of the tweaked parameters on the transmitter side.

B. *Example* 2 *- Frequency Modulation (FM)*

This experiment aims at illustrating a Frequency modulated communications system, by implementing Narrow Band FM transmission and reception of an audio file. Here the user is capable of not only observing the spectra of the transmitted signal and the received one, but also recovering the audio file on the receiver side.

The *flowgraph* (Fig. 8) is executed in a VM to perform the transmission. It contains a file selector block, to choose the audio file (*wav* format) to be transmitted, in conjunction of a Narrow Band FM transmit block for modulation and a *UHD:USRP Sink* to send the modulated signal over the radio channel.


Options
ID: fm_tx
Title: Experimento FM
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 500k

Wav File Source
File: ...pSharpDressedMan.wav
Repeat: Yes

NBFM Transmit
Audio Rate: 48k
Quadrature Rate: 192k
Tau: 75u
Max Deviation: 5k
Preemphasis High Corner Freq: -1

Rational Resampler
Interpolation: 500k
Decimation: 192k
Taps:
Fractional BW: 0

UHD: USRP Sink
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 174.3M
Ch0: Gain Value: 55
Ch0: Antenna: TX/RX
TSB tag name:

QT GUI Frequency Sink
Name: Espectro... transmitido
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 500k

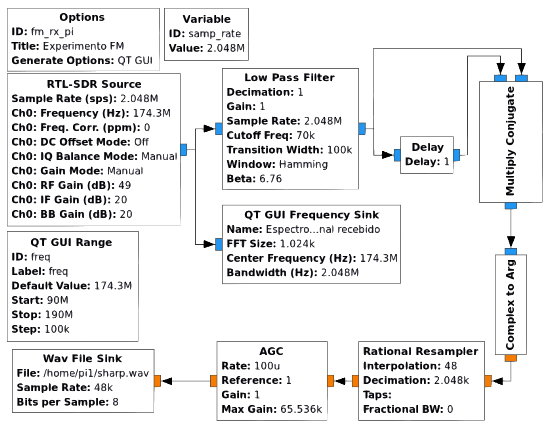
Fig. 8. FM transmission flowgraph.

In the receiver (see Fig. 9), which is executed on a RPi, the FM signal is received by the RTL-SDR module, like in the previous example. It is then treated on the receiver in a way where the user can recover all information contained in the signal's phase and can also listen to the signal, when saved in an audio file.

C. *Example* 3 *- Digital Transmission*

Here we will illustrate the performance of a Digital communication system. Since the platform is limited in terms of performance, the user uses it only to transmit the signal an receive the raw data, leaving the post-processing to be done offline in a more powerful machine. This is another way of using our platform, for data collection only, leaving the processing to a secondary location.

The experiment consists in a transmitter (see Fig. 10), executed in a VM, that is responsible for sending a Quadrature


Options
ID: fm_rx_pi
Title: Experimento FM
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.048M

RTL-SDR Source
Sample Rate (sps): 2.048M
Ch0: Frequency (Hz): 174.3M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Manual
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 49
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 2.048M
Cutoff Freq: 70k
Transition Width: 100k
Window: Hamming
Beta: 6.76

Delay
Delay: 1

QT GUI Frequency Sink
Name: Espectro...nal recebido
FFT Size: 1.024k
Center Frequency (Hz): 174.3M
Bandwidth (Hz): 2.048M

QT GUI Range
ID: freq
Label: freq
Default Value: 174.3M
Start: 90M
Stop: 190M
Step: 100k

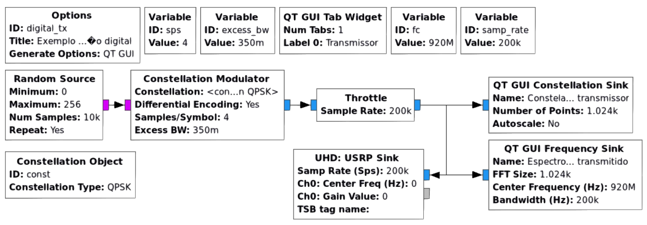
Wav File Sink
File: /home/pi1/sharp.wav
Sample Rate: 48k
Bits per Sample: 8

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 65.536k

Rational Resampler
Interpolation: 48
Decimation: 2.048k
Taps:
Fractional BW: 0

Fig. 9. FM reception flowgraph.

Phase Shift Keying (QPSK) signal using a USRP, and a receiver (see Fig. 11) in a RPi, that captures the raw data samples and saves it into a file, using the RTL-SDR. Here the user can tweak the kind of modulation that he/she wants for the transmission, by altering the type of constellation used and the *roll-off* factor of the transmitted pulse.


Options
ID: digital_tx
Title: Exemplo ...�o digital
Generate Options: QT GUI

Variable
ID: sps
Value: 4

Variable
ID: excess_bw
Value: 350m

QT GUI Tab Widget
Num Tabs: 1
Label 0: Transmissor

Variable
ID: fc
Value: 920M

Variable
ID: samp_rate
Value: 200k

Random Source
Minimum: 0
Maximum: 256
Num Samples: 10k
Repeat: Yes

Constellation Modulator
Constellation: <con...n QPSK>
Differential Encoding: Yes
Samples/Symbol: 4
Excess BW: 350m

Throttle
Sample Rate: 200k

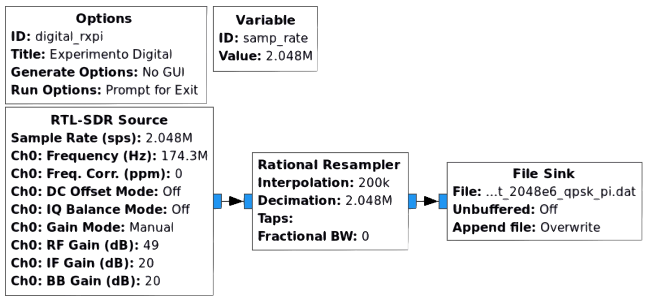
QT GUI Constellation Sink
Name: Constela... transmissor
Number of Points: 1.024k
Autoscale: No

Constellation Object
ID: const
Constellation Type: QPSK

UHD: USRP Sink
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 0
Ch0: Gain Value: 0
TSB tag name:

QT GUI Frequency Sink
Name: Espectro... transmitido
FFT Size: 1.024k
Center Frequency (Hz): 920M
Bandwidth (Hz): 200k

Fig. 10. Digital transmission flowgraph.


Options
ID: digital_rxpi
Title: Experimento Digital
Generate Options: No GUI
Run Options: Prompt for Exit

Variable
ID: samp_rate
Value: 2.048M

RTL-SDR Source
Sample Rate (sps): 2.048M
Ch0: Frequency (Hz): 174.3M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 49
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Rational Resampler
Interpolation: 200k
Decimation: 2.048M
Taps:
Fractional BW: 0

File Sink
File: ...t_2048e6_qpsk_pi.dat
Unbuffered: Off
Append file: Overwrite

Fig. 11. Digital reception and demodulation flowgraphs.

IV. Conclusions

As detailed in previous sections, with this platform we make it possible to perform experiments regarding radio communications, from data sources to transmission, reception and decoding, thus reducing the gap between theory and practice and enriching the teaching of this area.

The main point of this platform was its low cost when compared to more robust and complex teaching platforms. Also, with the usage of Open Source Software, we allow the users not only to perform simple experiments, but also to broaden the possibilities of experiments in the radio communications area by creating their own implementation of systems, considering bigger scenarios and new techniques.

Appendix

Platform cost estimates

In this section, we present an estimate of the cost, in BRL, of the elements that compose the platform, shown in Table I.

TABELA I

Example of cost table for the platform.

Material	**Quantity**	**Unit Price**	**Total Price**
Network Cables	3	R$ 3,90	R$ 11,70
Raspberry Pi	2	R$ 239,90	R$ 479,80
8GB SD Card	2	R$ 44,90	R$ 89,80
RPi Power supply	2	R$ 34,90	R$ 69.80
Virtualization server	1	R$ 2.500,00	R$ 2.500,00
USRP	2	R$ 5.420,00	R$ 10.840,00
RTL-SDR	2	R$ 150,00	R$ 300,00
Total	-	-	R$14.291,10

References

[1] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The Next Generation Wireless Access Technology*. Elsevier Academic Press, 1 ed., 8 2018.
[2] Ericsson, "Ericsson mobility report - q4 2018," tech. rep., Ericsson, 2019.
[3] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, pp. 3051–3073, 7 2009.
[4] mmMagic Project, "mm-Wave based mobile radio access network for 5G integrated communications." Online at https://5g-mmmagic.eu/.
[5] CorteXlab, "FIT/CorteXLab cognitive radio platform." Online at https://5g-mmmagic.eu/.
[6] R. W. Stewart, K. W. Barlee, D. S. W. Atkinson, and L. H. Crockett, *Software Defined Radio using MATLAB & Simulink and the RTL-SDR*. Strathclyde Academic Media, 2015.
[7] Universidade Federal de Minas Gerais - UFMG, "UFMG FUTEBOL Testbed." Online at http://futebol.dcc.ufmg.br/index.html.
[8] Trinity College Dublin - Connect Centre, "Iris: The reconfigurable radio testbed." Online at https://iris-testbed.connectcentre.ie/.
[9] "Arch linux official website." Online at https://archlinux.org.
[10] Ettus Research, "B200/b210/b200mini/b205mini." Online at https://kb.ettus.com/B200/B210/B200mini/B205mini.
[11] Ettus Research, "The usrp^{TM} hardware driver repository." Online at https://github.com/EttusResearch/uhd.
[12] Ettus Research, *USRP B200/B210 Bus series*. Online at https://www.ettus.com/wp-content/uploads/2019/01/b200-b210_spec_sheet.pdf.
[13] S. Markgraf, D. Stolnikov, Hoernchen, K. Keen, C. Vogel, and H. Welte, "rtl-sdr." Online at https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr.
[14] The MagPi Magazine: The official Raspberry Pi magazine, "Raspberry pi 3: Specs, benchmarks & testing." Online at https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/.
[15] "Gnuradio official website." Online at https://www.gnuradio.org/.
[16] "Libvirt python api bindings." Online at https://libvirt.org/python.html.
[17] J. F. N. Pinheiro, E. M. S. de Santana Junior, and T. F. Maciel, "The Raspberry Pi SDR testbed." Online at https://github.com/jpinheiro228/pi_testbed.