UNIVERSIDADE FEDERAL DO ABC
CENTRO DE ENGENHARIA, MODELAGEM E CIÊNCIAS SOCIAIS APLICADAS
GRADUAÇÃO EM ENGENHARIA DE INFORMAÇÃO

Felipe Augusto Anon da Silva

Estudos do uso de rádios definidos por software para medidas na faixa do UHF

Santo André - SP
2021

Felipe Augusto Anon da Silva

Estudos do uso de rádios definidos por software para medidas na faixa do UHF

Monografia apresentada ao Curso de Graduação em Engenharia de Informação da Universidade Federal do ABC, como parte dos requisitos necessários para a obtenção do Título de Bacharel em Engenharia de Informação

Universidade Federal do ABC – UFABC

Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas

Programa de Graduação em Engenharia de Informação

Orientador: Prof. Dr. Marcelo Bender Perotoni

Santo André - SP

2021

RESUMO

Este trabalho tem como objetivo avaliar o uso de Rádios Definidos por Software para aquisição dos dados referentes à radiação de várias antenas, bem como a plotagem de seus diagramas de radiação. Apresenta resultados experimentais capturados por meio de programa desenvolvido utilizando-se a linguagem Python. Também faz um panorama dentre as diversas opções para desenvolvimento de aplicações baseadas nestes rádios definidos por softwares

Palavras-chaves: SDR. Diagrama de Radiação. Antenas.

ABSTRACT

This project aims to evaluate the usage of Software Defined Radios to study and plot Antennas radiation diagrams. It presents empirical results captured using a Python script in conjunction with specific SDR software. It also lists a few programs which can be used to develop Sofware Defined Radio applications.

Keywords: SDR. Radiation Diagram. Antennas.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPIO	***General Purpose Input-Output*** - Pinos dedicados à interface entre uma determinada máquina ou plataforma com outros dispositivos por meio de controles elétricos.
GPS	***Global Positioning System*** - Sistema Norte-Americano de satélites que permite a localização de um dispositivo no globo terrestre.
GRC	***Gnu Radio Companion*** - Software do pacote Gnuradio para criação de programas para processamento dos sinais do SDR.
LNA	***Low Noise Amplifier*** - Amplificador com a característica de um baixo ruído em sua saída.
RF	***RF***- Radiofrequência.
SDR	***Software Defined Radio*** - Sistema hardware-software que é utilizado para captura e/ou transmissão de sinais em radiofrequência, onde o software configura o hardware de acordo com os parâmetros desejados pelo usuário.
SSH	***Secure Shell*** - Padrão de comunicação seguro que permite acesso remoto ao console de um computador por meio de uma rede.

SUMÁRIO

1 INTRODUÇÃO

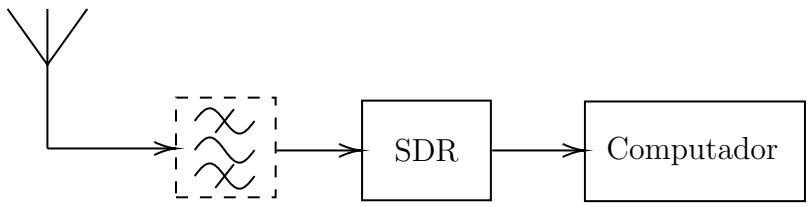
A idéia de rádios configuráveis data da década de 70, com o primeiro modelo sendo desenvolvido na década de 80 por membros da E-Systems, logo após vários desenvolvimentos na tecnologia de semicondutores (DI, 2018, p. 312), tendo como propósito o processamento digital de sinais e a aplicação de novas técnicas de demodulação para que fosse possível a captação de sinais que os sistemas tradicionais da época não conseguissem interpretar(JOHNSON, 1985).

Com o aumento do poder computacional, novos tipos de SDR's foram sendo lançados no mercado, de forma que a partir da década de 90 o universo dos rádios definidos por software foi se ampliando juntamente com o desenvolvimento de novas tecnologias, como por exemplo a utilização da tecnologia FPGA nos rádios (MACHADO; WYGLINSKI, 2015).

Nos dias atuais, os rádios definidos por software vem se popularizando graças ao seu baixo custo de implementação e sua versatilidade, sendo utilizado inclusive por militares (TRIBBLE, 2008). Existem hoje no mercado uma ampla gama de dispositivos SDR de baixíssimo custo disponíveis, aumentando de forma acelerada a adoção pelos mais diversos tipos de usuários.

A simplicidade de um sistema SDR pode ser vista na figura 1, que nos mostra um sistema de comunicação utilizando-se de um SDR e um filtro passa-baixas (que é inteiramente opcional, sendo utilizado em casos onde existam transmissões com muita energia que possam saturar a entrada do SDR ou de seu conversor analógico-digital), ou até mesmo um medidor de potência RF conforme desenvolvido pelo LARF (Laboratório de Antenas e Radio Frequência) da UFABC (SILVA, 2021).

Figura 1 – Exemplo da simplicidade na implementação de SDR



Fonte:Silva (2021)

A tabela 1 contém alguns SDRs comercialmente disponíveis, onde é possível ver que o RTL-SDR é a alternativa mais barata entre eles.

Tabela 1 – Comparação entre outros SDRs

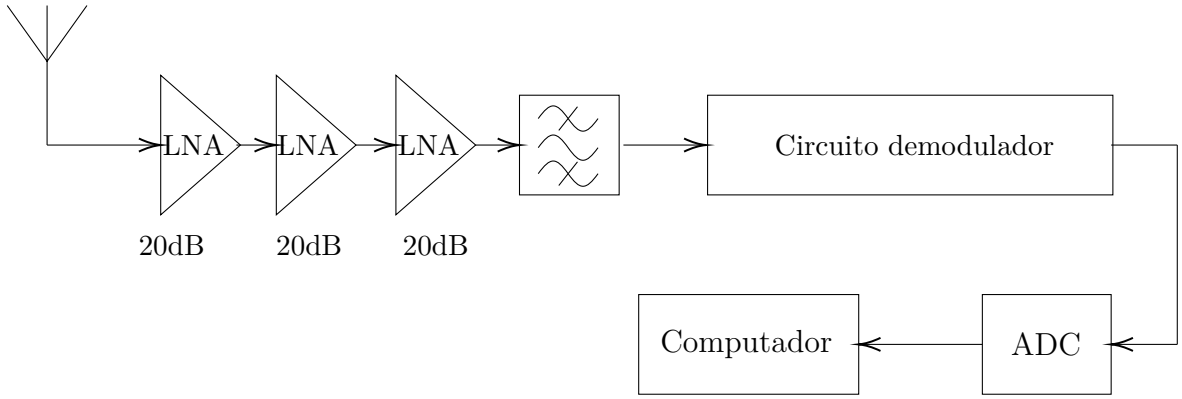
	Freq.(MHz)	**Banda(MHz)**	**ADC(Bits)**	**TX/RX**	**Preço(US$)**
Funcube	0.150 - 260 410 - 2050	0.192	16	RX.	200
HackRF	30 - 6000	20	8	RX/TX	299
BladeRF	300 - 3800	40	12	RX/TX	400, 650
USRP B200/B210	70 - 6000	56	16	RX/TX	675/1100
SockRock Ensemble II	0.180 - 3	0.192	[1]	RX/TX	67 (TX) 89 (RX/TX)
AirSpy	24 - 1750	10	12	RX	199
Mirics (Msi3101)	64 - 108 162 - 240 470 - 960	5	12	RX	85
SDR Play	100 – 380 430 – 2000	8	12	RX	149
RTL-SDRV3	24-1766	3.2	8	RX	25

Fonte: Silva (2021), adaptado de Laufer (2015)

Também é possível montar um sistema de comunicação baseado puramente em hardware, similar ao sistema da figura 1 (e que retornará ao computador o sinal transmitido demodulado e convertido em dados binários, a serem ou não processados posteriormente dependendo do *design* do demodulador). O diagrama em blocos é apresentado na figura 2. De qualquer maneira, o simples uso dos LNA's (*Low Noise Amplifier* - Amplificador de baixo ruído) já torna o sistema com maior custo do que o baseado em SDR, sem contar com os inúmeros outros componentes necessários para que esse sistema funcione de forma satisfatória, como por exemplo fontes DC ou baterias para alimentar os LNA's, o conversor analógico-digital, o demodulador (que dependendo do tipo de modulação utilizada pode ter um custo elevado). Lembrando que nem todos os componentes funcionam nas mesmas faixas de tensão, além disso, para que seja possível a ligação do conversor analógico-digital no computador seria necessária uma interface adicional, sendo esta omitida neste caso pois o Raspberry PI possui pinos em seu barramento que possibilitam que tal periférico seja conectado.

[1] Depende da placa de som

Figura 2 – Modelo de sistema com medidor de potência discreto

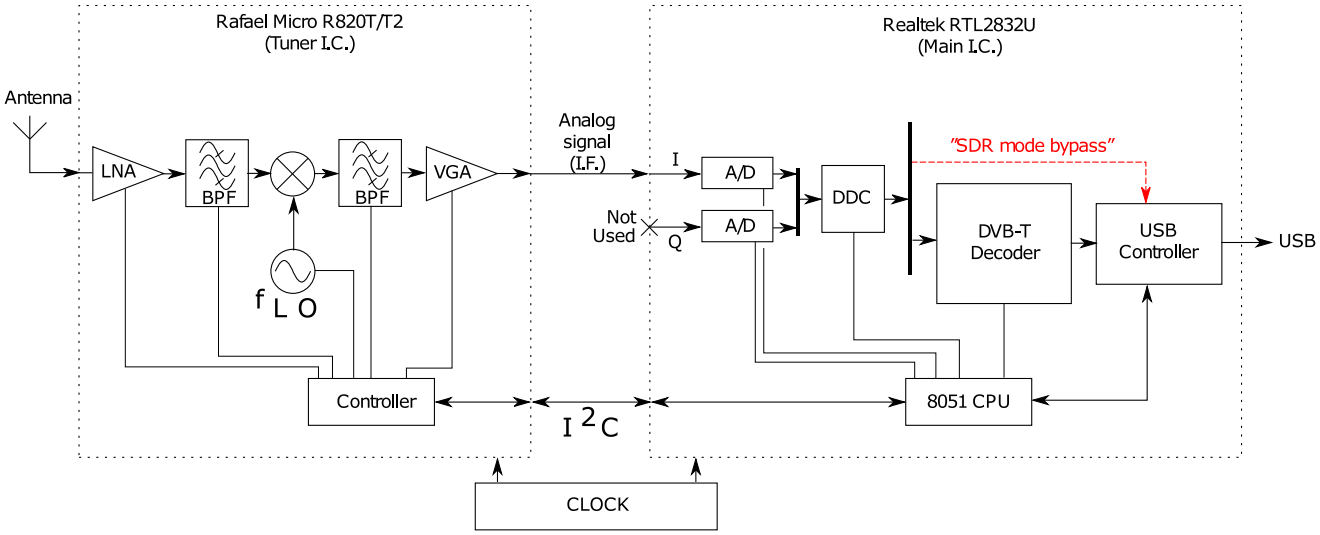


Fonte:O Autor(2020)

Tal sistema também demandaria um cuidado no layout e disposição dos componentes individuais visto que se tratam de frequências elevadas, em contraste com o SDR. Isso gera inclusive um custo adicional dos cabos coaxiais para interligação dos componentes. Conforme figura 3, o próprio SDR já conta internamente com filtro passa-faixas e amplificadores, o que representa uma economia significativa para essa montagem em contraste com a da figura 2.

Para efeito de comparação, o SDR utilizado nos estudos deste Trabalho de Graduação teve um custo de aproximadamente R$ 120, sendo que existem outros modelos ainda mais baratos no mercado, tornando essa solução muito atrativa do ponto de vista financeiro, além do SDR proporcionar uma flexibilidade, pois é possível reaproveitar o mesmo para outras funções variando apenas o programa rodando no computador, ao contrário de um hardware como o da figura 2 que seria única e exclusivamente um medidor de potência RF.

Figura 3 – Diagrama interno do RTL-SDR



Fonte: Muttoni & Veiga (2017)

O mesmo aconteceria para outras aplicações, como por exemplo um analisador de espectro, ou radiotelescópio. A utilização de softwares numa plataforma Linux por exemplo também permite a utilização de hardware adicional como uma antena GPS, câmera, sensores entre outros. No caso do Raspberry PI também é possível conforme dito anteriormente utilizar sua pinagem de GPIO para ligar componentes como por exemplo amplificadores, sendo que um programa *Python* poderia controlar todos esses aspectos de forma automatizada.

1.1 Trabalhos relacionados

Dentre os estudos de propagação, é possível citar o trabalho de Araujo (2019) que traz um panorama da propagação de rádio no ambiente indoor e outdoor em prédios e nas imediações da UFABC, contando com simulações e medidas empíricas, que podem ser aproveitadas nesse Trabalho de Graduação como fonte comparativa. Dentre os estudos de rádios definidos por software, convém citar a obra de Tuttlebee (2002) que mostra uma perspectiva da implementação e propostas futuras dos rádios definidos por software no início do milênio, de forma que se foca nas operadoras de telecom e a implementação de rádios definidos por software nesses sistemas.

Stewart *et al.* (2015) por outro lado aborda os rádios definidos por software mais populares,mais especificamente o RTL-SDR, de forma que em comparação com o trabalho de Tuttlebee (2002), é possível perceber que a descoberta do modo SDR nos *dongles* de TV digital ajudaram e muito a popularizar o tema. A obra de Stewart *et al.* (2015) contém uma série de exemplos da utilização do SDR em conjunto com a suite MATLAB/Simulink, que serão úteis na exploração do hardware do SDR.

Wright & Ball (2019) fazem uma medição da potência de RF utilizando-se de um programa próprio desenvolvido para tal, utilizando a API do Gnuradio. Em Wright & Ball (2019), os autores abordam o tema de medidas de potência em UHF usando o SDR, da mesma forma que Silva (2021) o faz, porém com um método diferente. Em ambos os trabalhos foram realizados estudos sobre a faixa dinâmica dos rádios, que foi estabelecido em torno de 80dB.

Com relação à (IEEE..., 2008), a entidade (IEEE) apresenta um padrão de procedimentos para testes de antenas, que foram demasiadamente úteis para a realização e interpretação dos resultados aqui contidos, em especial na parte de simulações das antenas. É conveniente citar que tal padrão de testes teve que ser adaptado para que correspondesse às limitações principalmente quanto à localidades de teste e quanto ao próprio equipamento disponível.

2 OBJETIVOS E METODOLOGIA PROPOSTA

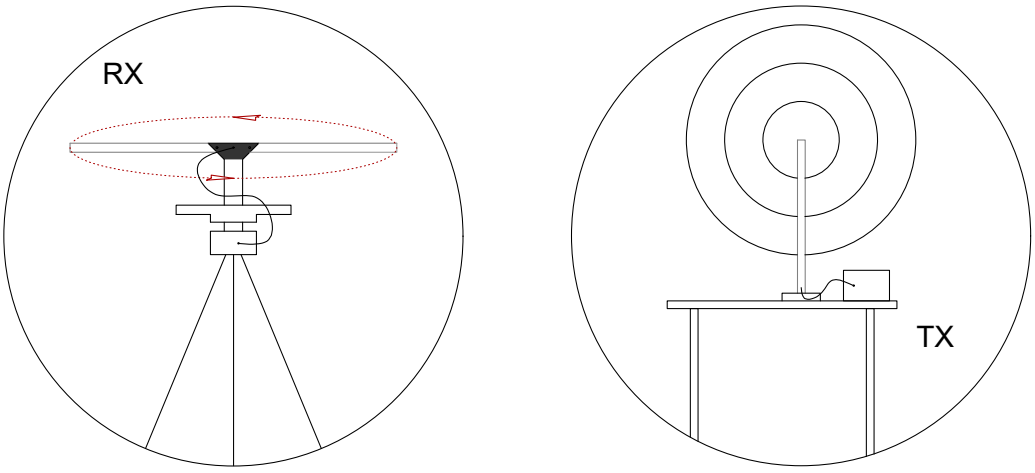
Este Trabalho de Graduação apresenta como objetivo principal o desenvolvimento de um software para a medição e plotagem de um diagrama de radiação 2D de uma antena, normalizado em seu pico (ou seja, tomando o ponto cuja amplitude de potência máxima é tomada como referência 0dB e comparando os outros pontos com relação à esta).

Para tanto, é proposto o sistema da figura 4, que consiste em um transmissor fixo irradiando à uma separação mínima do receptor para que ambos se encontrem no chamado campo distante (*far-field* ou região de Faunhofer), conforme fórmula da equação 2.1 (BALANIS, 2016):

$$\text{Distância de campo distante} = \frac{2D^2}{\lambda} \tag{2.1}$$

Além disso, segundo IEEE... (2008) o ideal para tal caracterização seria ainda a utilização de uma câmara anecóica ou um ambiente cuja separação horizontal e vertical sejam propícias (como o cume de montanhas ou entre duas colinas por exemplo) a fim de evitar reflexões e outros efeitos deletérios para as medidas das antenas.

Figura 4 – Trabalho proposto.



Fonte: O Autor (2021).

Por fim, este trabalho de Graduação ainda traz uma pequena compilação dos principais softwares disponíveis para trabalho com Rádios Definidos por Software, em especial o modelo utilizado na realização deste (RTL-SDRv3).

3 HARDWARE

3.1 RTL-SDR

O hardware escolhido para utilização nesse trabalho de graduação foi o RTL-SDR, rádio de baixo custo, que possui internamente um sintonizador modelo R820T2 em conjunto com um receptor RTL8232U que opera como um conversor analógico digital, em cujo modo de operação as amostras I e Q do sinal recebido são enviadas ao PC via conexão USB(LAUFER, 2015).

Figura 5 – RTL-SDR.

Fonte: Silva (2021).

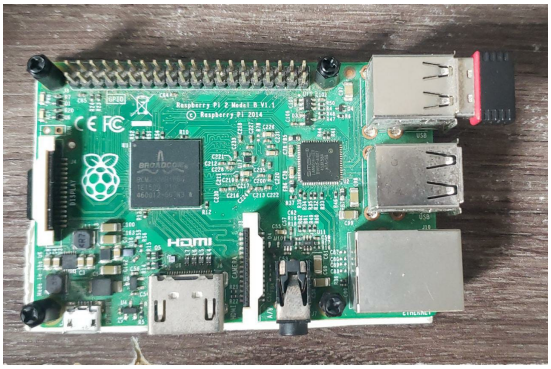
A conexão do mesmo com o PC é realizada por meio de um cabo USB, e a parte de RF por meio de um conector SMA 50 Ω. O mesmo possui amplificador interno com ajuste de ganho via software, permitindo assim realizar medidas em diferentes faixas de potência diferentes (com o fundo da escala em aproximadamente -110 dBm (SILVA, 2021)).

3.2 Raspberry PI

Outro elemento base para o desenvolvimento deste Trabalho de Graduação foi a plataforma microprocessada Raspberry PI, que consiste em um sistema completo (CPU + GPU + Memória) baseado na tecnologia ARM. Além disso, por rodar o linux e possuir diversos tipos de entradas e saídas, permite relativa flexibilidade para a utilização de periféricos diversos, como por exemplo a utilização de seus pinos de saída como um transmissor de RF por meio da utilização de um software específico.

Possui diversos formatos e versões, onde convém ressaltar o Raspberry Pi 2 e o Raspberry Pi Zero W, utilizados neste Trabalho de Graduação; ambos podem ser vistos na figura 6:

Figura 6 – Versões do Raspberry PI - Zero W e 2.



(a) Rasbperry Pi 2 Modelo B.

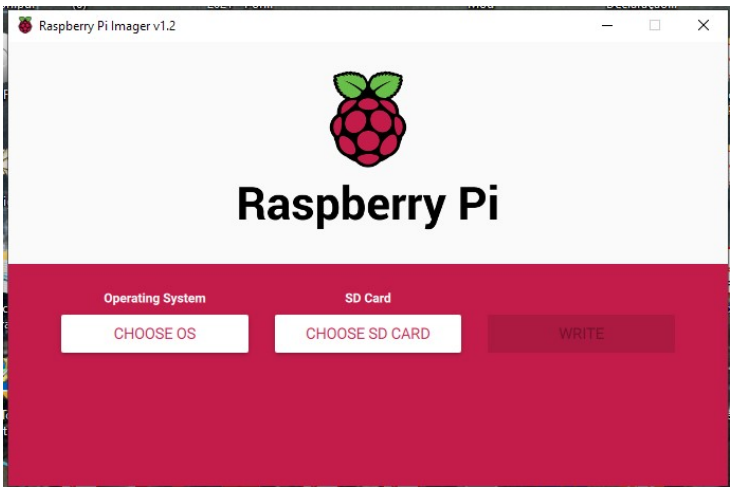
(b) Raspberry Pi Zero W.

Fonte: O Autor (2021).

3.2.1 Instalação do sistema Raspberry PI OS

De forma a ser possível a utilização do Raspberry PI, é necessária a instalação de um sistema operacional compatível. Para tanto, foi utilizado o software Raspberry PI Imager[1] , disponibilizado pela própria fabricante do Raspberry PI. O mesmo possui uma interface intuitiva, como pode ser visto na figura 7:

Figura 7 – Tela inicial do Raspberry PI Imager.



Fonte: O Autor (2021).

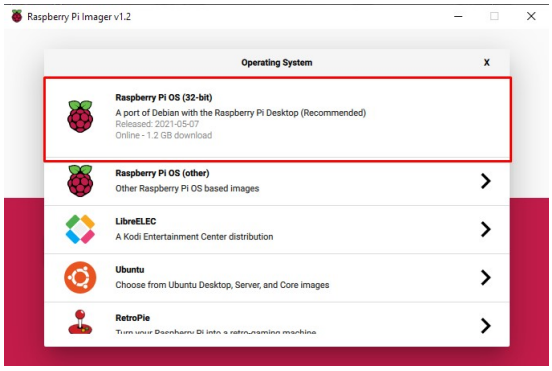
O processo de instalação passo a passo é descrito na figura 8. Como é possível perceber, trata-se de um processo intuitivo e fácil para que os usuários mesmo sem

[1] Raspberry PI Imager. Acesso em 17/06/2021

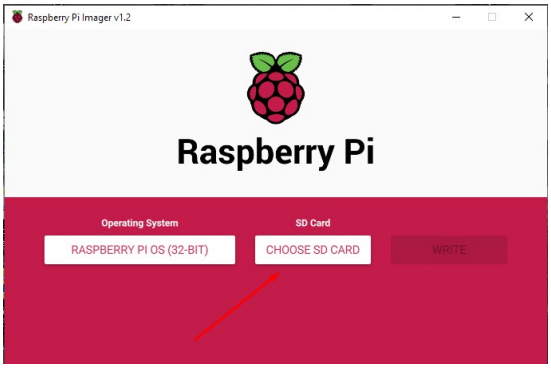
conhecimento técnico realizem a instalação, além disso o software ainda possibilita a instalação de outros sistemas operacionais (como por exemplo o Ubuntu), aplicativos para automação residencial (Home Assistant), entre outros, de forma que existe uma grande versatilidade nesse instalador.

O programa realiza automaticamente o download do sistema selecionado dentre suas opções, tendo ainda a possibilidade do usuário baixar sistema próprio e realizar a instalação utilizando o mesmo por meio de um arquivo do tipo .img fornecido pelo distribuidor do sistema operacional.

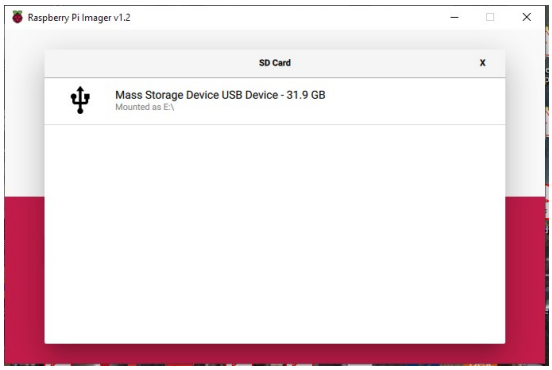
Figura 8 – Passo a passo da instalação do Raspberry PI OS.



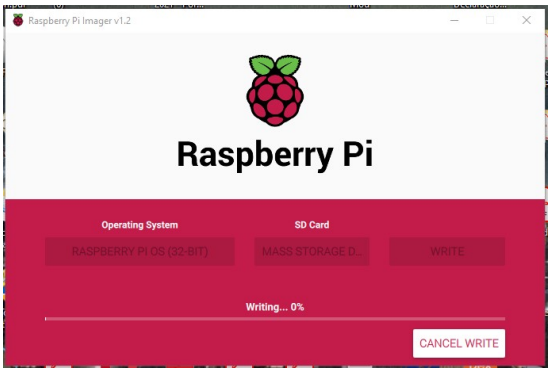
(a) Seleciona-se Sistema Operacional.



(b) Opção para selecionar o cartão de memória.



(c) Lista de cartões de memória.

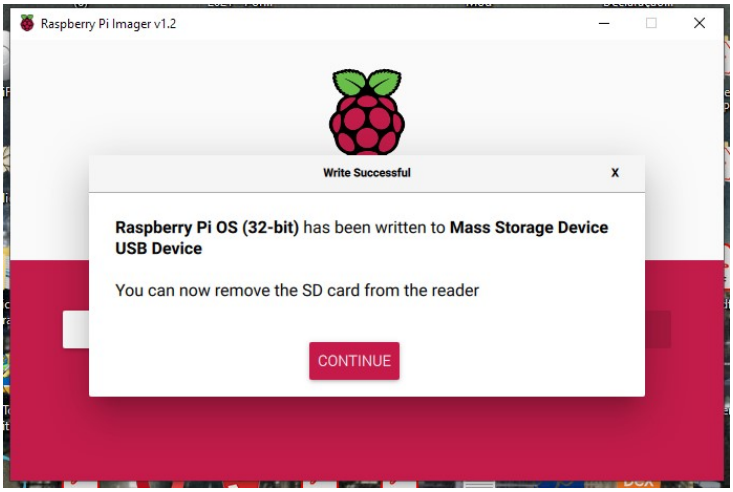


(d) Processo de instalação iniciado.

Fonte: O Autor (2021).

Após a conclusão do processo o programa mostra a mensagem da figura 9, onde o usuário pode retirar o cartão de memória e colocá-lo no Raspberry. É importante ressaltar que o procedimento é o mesmo para as diversas versões do Raspberry PI (no caso a Zero W e a 2 B+).

Figura 9 – Processo concluído - Instalação do Raspberry PI OS.



Fonte: O Autor (2020).

4 SOFTWARES HABITUALMENTE UTILIZADOS COM O RTL-SDR

Antes de iniciar a explicação sobre cada software utilizado para controlar e processar os dados do SDR, é importante lembrar que muitas dessas aplicações estão dependentes do sistema operacional utilizado, por exemplo, ao serem utilizados sistemas baseados em Linux, os mesmos dispõem de um terminal onde é possível executar *scripts*, rodar programas num determinado cronograma entre outros, que são difíceis de se replicar no sistema operacional Windows. Nesse trabalho de Graduação são apresentadas então algumas opções de softwares habitualmente utilizados em conjunto com o RTL-SDR, bem como outros rádios definidos por software.

Foram separadas em duas categorias: Ferramentas de desenvolvimento, que facilitam a criação de programas completos para a realização de diversas funções (seja o processamento do sinal ou também medições de campo como o tema desse Trabalho de Graduação); e os programas diversos, que contam com recursos próprios para o processamento das informações (desde a demodulação de um sinal FM até a identificação de um protocolo por exemplo).

4.1 Ferramentas de desenvolvimento

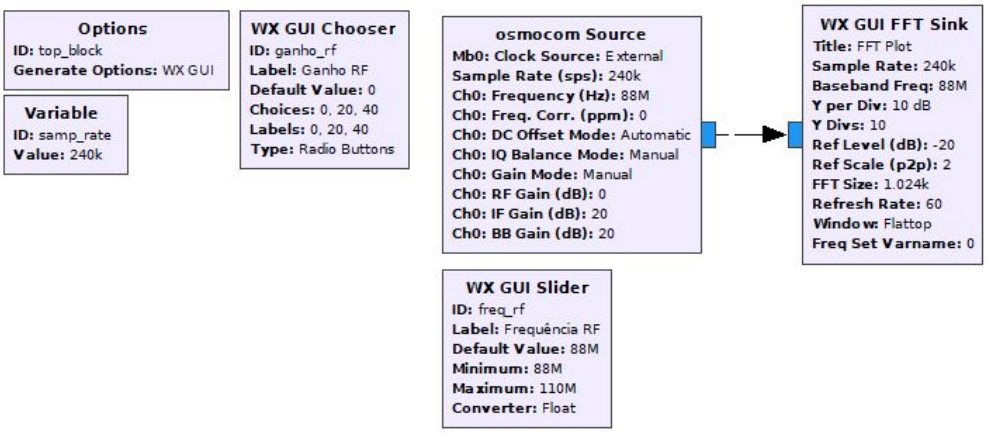
4.1.1 Gnuradio Companion

O programa Gnuradio companion é um dos mais populares softwares para desenvolvimento de aplicações utilizando-se SDR, traz uma interface de programação intuitiva baseada em blocos e de fácil aprendizado. Pode ser utilizado tanto no Windows quanto no Linux, e sua interface pode ser vista na figura 10, onde está rodando sob o Windows.

Possui blocos para realizar a interface entre o SDR e o programa, controlando assim diversos aspectos do rádio como por exemplo taxa de amostragem, banda e ganhos. Também é possível instalar outros blocos desenvolvidos usando a biblioteca própria sobre Python, a exemplo do osmocom Source mostrado na figura 10, alternativa aos blocos incluídos nativamente.

Além das fontes de sinal ainda existem blocos para as mais diversas funções, seja na parte do processamento do sinal quanto para sua transmissão, para o controle do rádio e para a decodificação. Por tratar-se de software livre, também é possível realizar modificações caso seja necessário.

Figura 10 – Exemplo da interface do Gnuradio Companion.

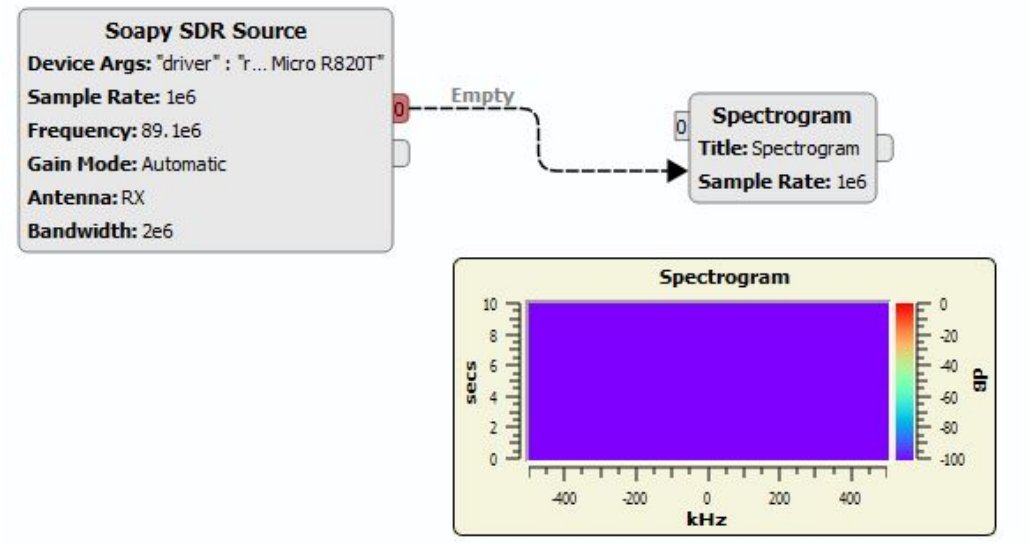


Fonte: O Autor (2020).

4.1.2 Pothos

Similarmente ao Gnuradio Companion, existe também o *software* Pothos Flow[1], cuja equipe de desenvolvimento também é composta por membros do projeto Gnuradio Companion, de forma que ambos apresentam interfaces e procedimentos de operação similares, como pode ser visto na figura 11.

Figura 11 – Exemplo da interface do Pothos Flow.



Fonte: O Autor (2020).

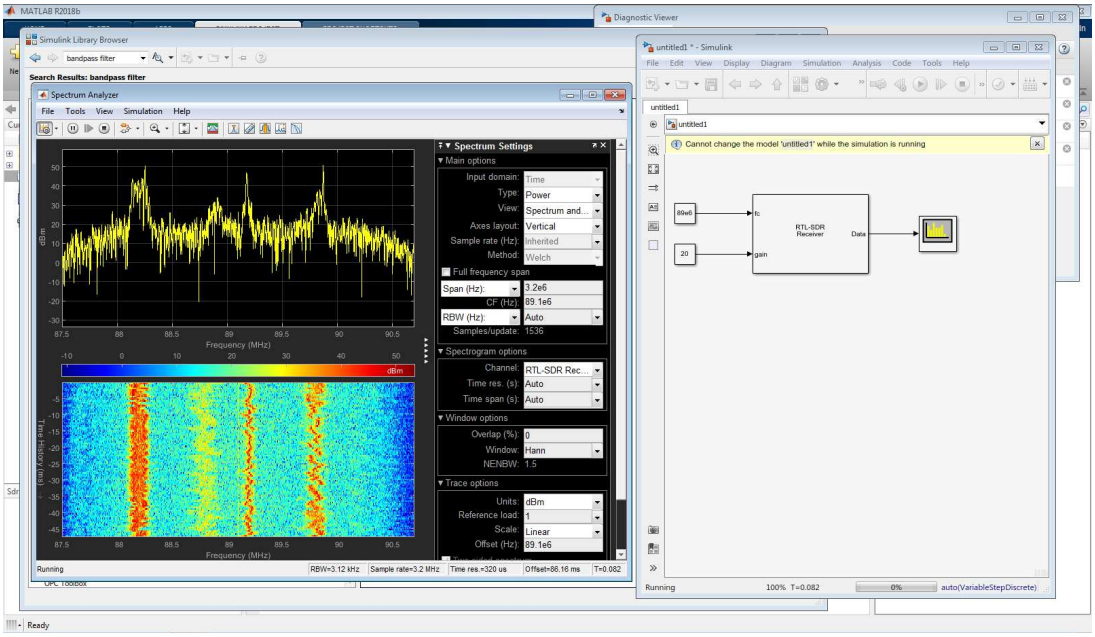
[1] Pothos Ware

A principal diferença entre o pothos flow e o gnuradio companion é a de que seus programas conseguem ser executados conforme estamos programando os mesmos (*"on the fly"*), ao contrário do Gnuradio onde é necessária uma compilação do mesmo para sua posterior execução.

4.1.3 Simulink

Dentre as alternativas pagas é possível citar o Simulink (figura 12), plataforma operando dentro do Matlab e de uso geral e com penetração ampla em várias áreas além de comunicações e RF.

Figura 12 – Exemplo de programa do Simulink.



Fonte: (SILVA, 2021).

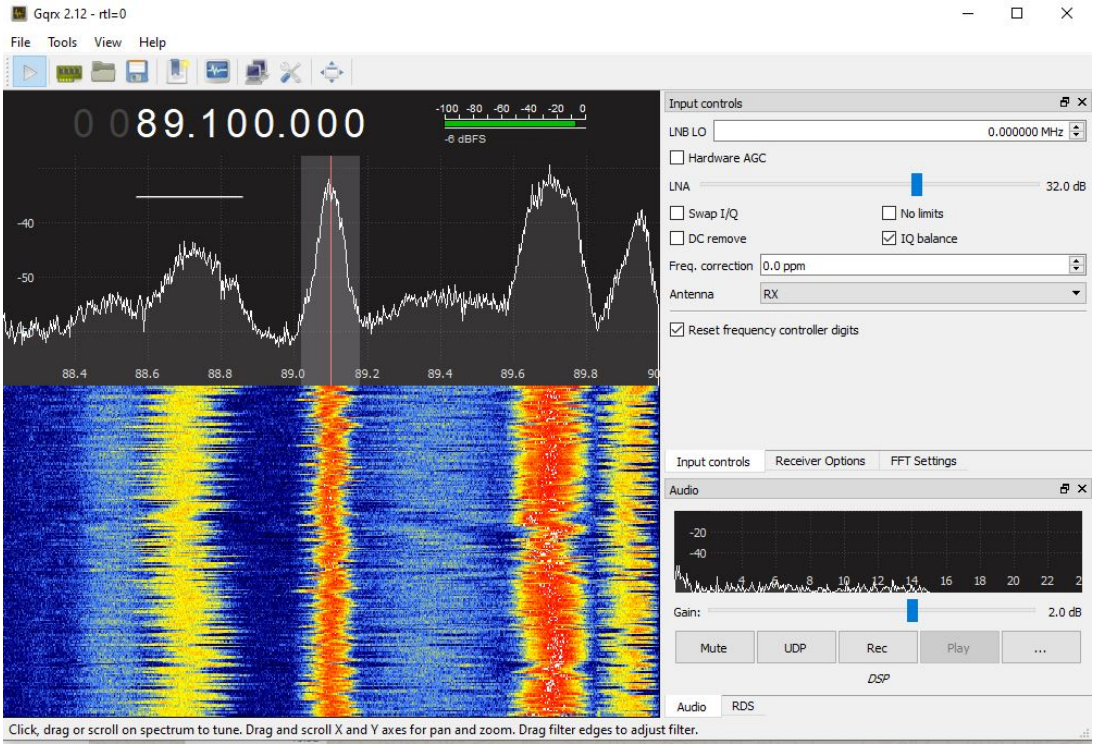
4.2 Programas para utilização com o RTL-SDR

4.2.1 GQRX

Ao contrário do Gnuradio Companion e do Pothos que são ferramentas de desenvolvimento, o GQRX[1] apresenta-se como um programa completo de controle do SDR. Sua interface pode ser vista na figura 13:

[1] GQRX

Figura 13 – Exemplo da interface do GQRX.



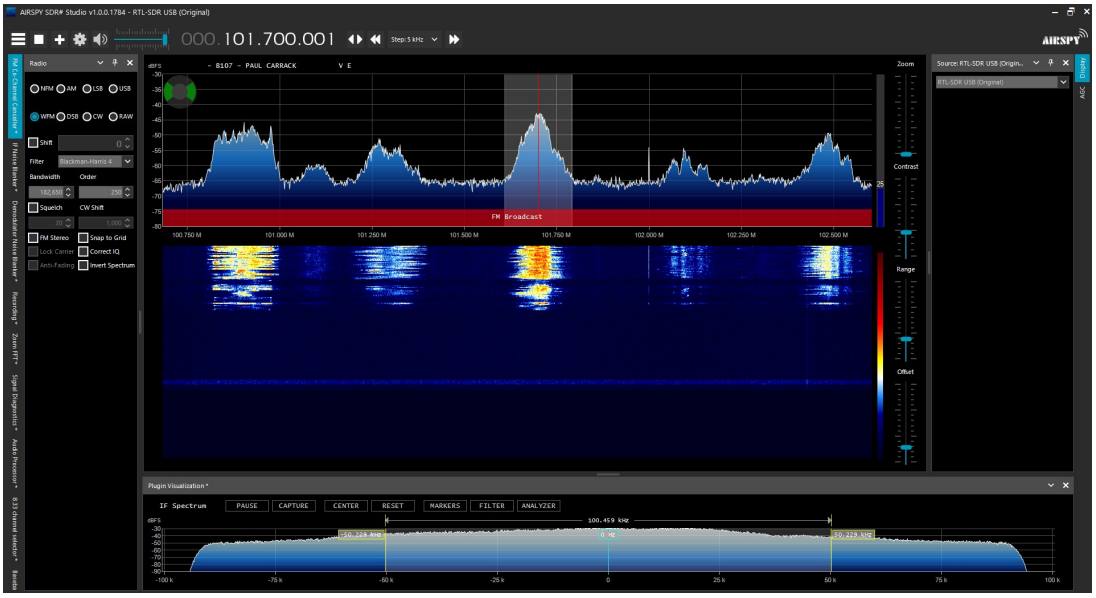
Fonte: O Autor (2021).

O uso do GQRX é interessante quando existe por exemplo uma necessidade de testar se o rádio está funcionando, bem como para os iniciantes em SDR poderem entender os conceitos por trás do mesmo. Esse software apresenta vários controles do rádio bem como demodulação de áudio nas modulações AM, FM, SSB e CW. Uma outra parte interessante do GQRX é que trata-se de software de código aberto, sendo possível ao usuário a modificação do mesmo para que supra suas necessidades.

4.2.2 SDR#

Também é interessante citar o software SDR# (*"SDR Sharp"*), programa desenvolvido pela AirSpy para uso com os rádios por ela desenvolvidos porém que também funciona com outros rádios, entre eles o RTL-SDR. Possui interface semelhante ao do GQRX, todavia ao contrário do anterior não é *open-source*, entretanto é disponibilizado para download gratuito por parte da AirSpy. Conta ainda com a possibilidade de instalação de *plugins* que ampliam suas capacidades originais como por exemplo scanner de frequência, decodificador de TV, entre outros. Permite também gravar o sinal captado, bem como o áudio decodificado de uma sintonização. Um exemplo de sua interface pode ser encontrado na figura 14, onde é possível perceber sua similaridade com o GQRX.

Figura 14 – Exemplo da interface do SDR Sharp.

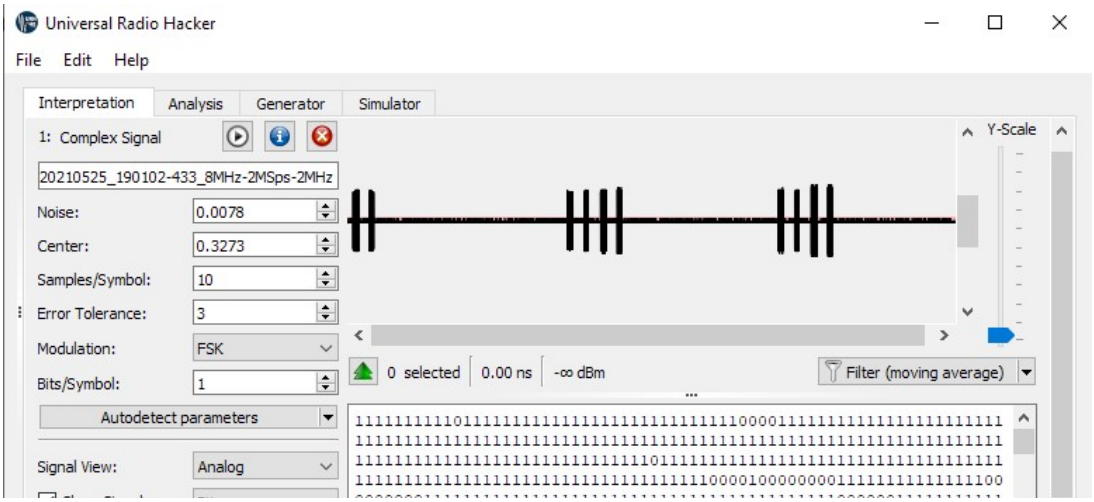


Fonte: O Autor (2021).

4.2.3 Universal Radio Hacker

Outra alternativa possível para caso de decodificação de sinais é conveniente citar o software Universal Radio Hacker, que conta com diversas funções para o processamento dos sinais recebidos, tais como demodulação e respectiva identificação do tipo de modulação, entre outras. Um exemplo de sua interface pode ser visto na figura 15, onde um sinal foi capturado e está sendo analisado.

Figura 15 – Exemplo da interface do Universal Radio Hacker.



Fonte: O Autor (2021).

Por fim, é importante ressaltar que a lista de softwares apresentados não é exaustiva, de modo que existem diversas outras opções de softwares para utilização com os rádios definidos por software, de forma que os softwares aqui apresentados são um conjunto introdutório de ferramentas para possibilitar o trabalho com os rádios definidos por software.

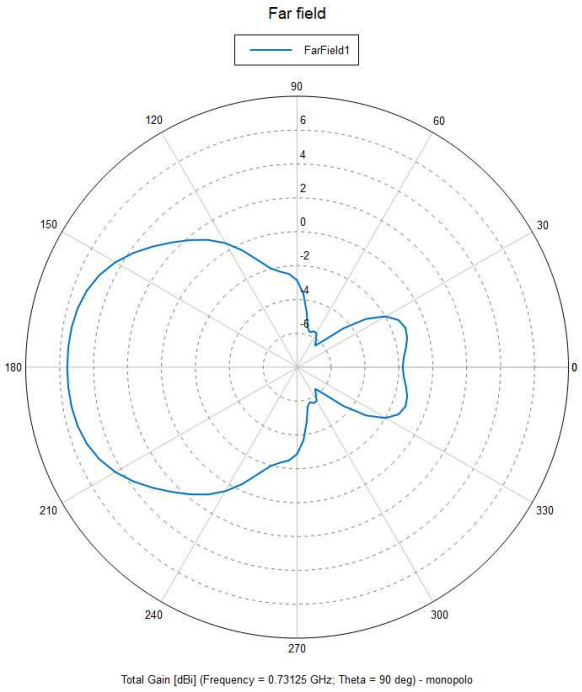
5 DESENVOLVIMENTO DO TRABALHO

A parte experimental desse Trabalho de Graduação consiste em um sistema para medição e apresentação do diagrama de radiação de uma antena. Para tanto é utilizado o módulo de medição de potência desenvolvido por Silva (2021), modificado para uma medição angular da potência recebida. Tal módulo possui uma interface com um programa *Python* desenvolvido com o intuito de capturar os ângulos, fazer a normalização e apresentar o diagrama normalizado de radiação.

O diagrama de radiação descreve em duas ou três dimensões a intensidade de potência que um observador receberá em determinada posição (BALANIS, 2016, p.25), de modo que é importante para determinar o posicionamento ideal de determinada antena ou instrumento para otimizar a eficiência em caso onde a antena exiba diretividade espacial, bem como a influência de outros elementos próximos à antena.

É apresentado um exemplo de diagrama de radiação 2D na figura 16, onde foi realizada a simulação de uma antena monopolo com dois refletores próximos à mesma. Tal simulação, bem como as outras realizadas ao longo desse Trabalho de Graduação foram realizadas utilizando-se da *suite* Altair FEKO, ferramenta disponibilizada de forma gratuita pela fabricante para uso educacional, com algumas limitações, não relevantes, contudo, para as simulações aqui realizadas.

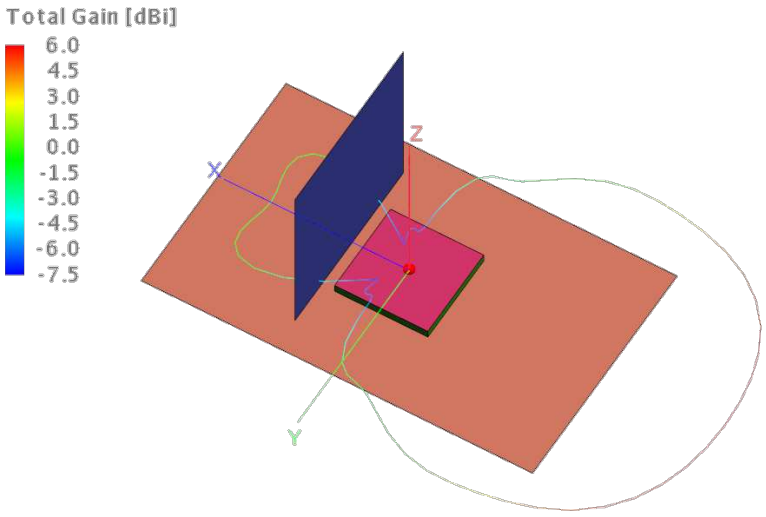
Figura 16 – Exemplo de diagrama de radiação 2D, formato polar.



Fonte: O Autor (2020).

É possível ver a correspondência do gráfico em corte com a simulação em 3D por meio da figura 17, onde são visíveis os elementos da antena (vista na cor verde e magenta, bem como seu elemento central) e os refletores (em azul e marrom).

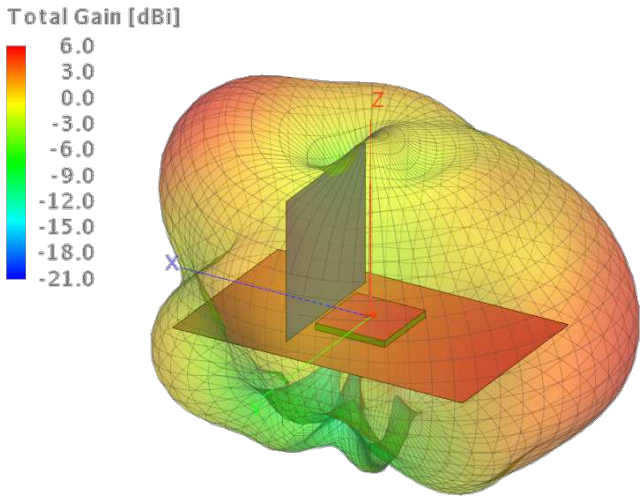
Figura 17 – Exemplo de diagrama de radiação em corte, 2D.



Fonte: O Autor (2020).

No contexto dos diagramas de radiação, existem ainda diagramas de radiação 3D, conforme pode ser visto na figura 18.

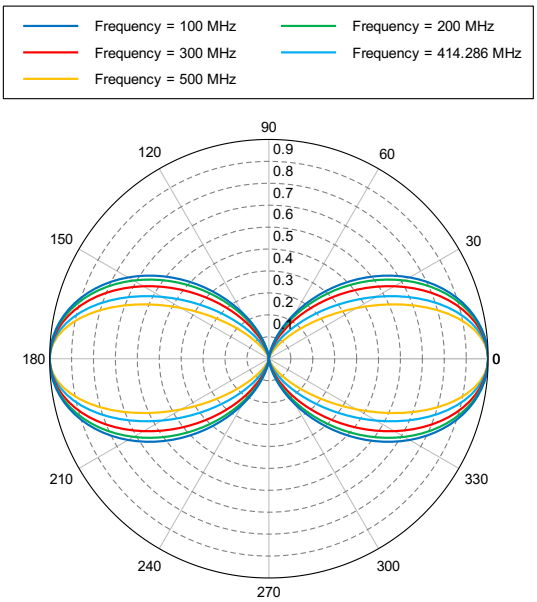
Figura 18 – Exemplo de diagrama de radiação 3D.



Fonte: O Autor (2020).

É importante lembrar também que é possível que exista uma variação do diagrama de radiação conforme a frequência analisada, como pode ser visto na simulação da figura 19, onde é apresentado o diagrama de radiação de um dipolo de acordo com a frequência.

Figura 19 – Variação do diagrama de radiação de acordo com a frequência.



Fonte: O Autor (2021).

No caso de uma antena de dipolo, o efeito observado para antenas cujo comprimento (l) são menores ou iguais ao comprimento de onda (λ), é que a largura de feixe de meia potência (*3dB Beamwidth*) (BALANIS, 2016) tende a diminuir conforme a frequência aumenta para um determinado tamanho da antena, conforme tabela 2:

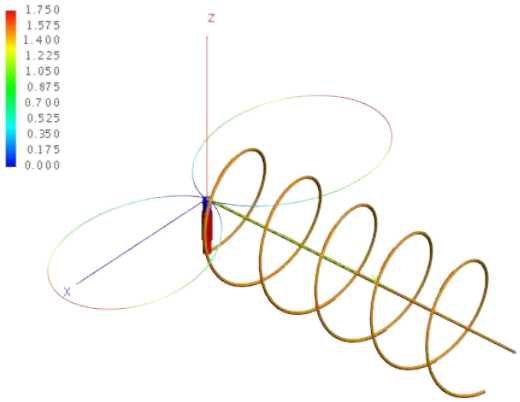
Tabela 2 – Larguras de feixe do dipolo.

Comprimento do dipolo (l)	Largura de feixe de meia potência
$l \ll \lambda$	90°
$l = \lambda/4$	87°
$l = \lambda/2$	78°
$l = 3\lambda/4$	64°
$l = \lambda$	47,8°

Fonte: (BALANIS, 2016).

É importante também também ressaltar que esse efeito existe para outros tipos de antena, sendo que com a variação o efeito pode se tornar drástico, como por exemplo na antena da figura 20, onde é apresentada uma antena helicoidal com dipolo central:

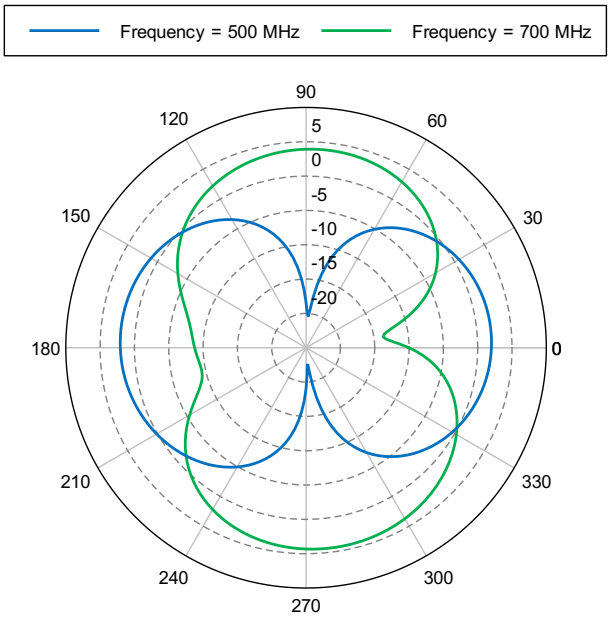
Figura 20 – Exemplo de diagrama de radiação - Antena helicoidal com dipolo central - f = 500 MHz.



Fonte: O Autor (2021).

É possível observar que na faixa dos 500 MHz, a mesma se comporta aproximadamente como um dipolo. Todavia, ao realizar a mesma simulação numa frequência de 700MHz, o diagrama de radiação muda radicalmente, conforme pode ser visto na figura 21, onde o diagrama de radiação para 500 MHz está destacado em cor azul e o de 700 MHz em verde.

Figura 21 – Variação do diagrama de radiação de acordo com a frequência.



Fonte: O Autor (2021).

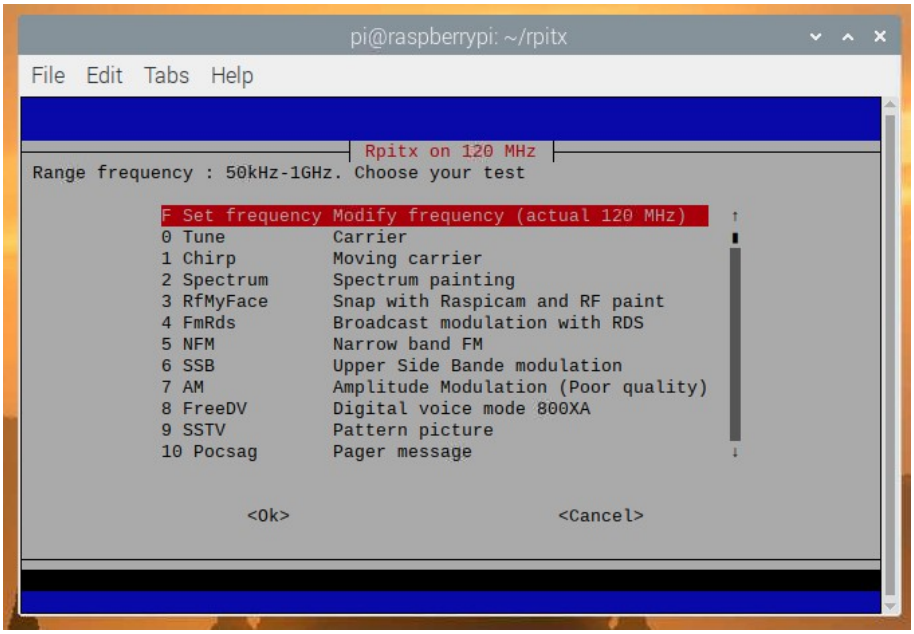
Como é possível observar, houve uma mudança na direcionalidade da antena em 90 graus; isso ajuda a demonstrar a importância dos diagramas de radiação, bem como a necessidade de uma flexibilidade com relação às frequências que podem ser utilizadas para testar determinada antena.

6 MONTAGEM EXPERIMENTAL

6.1 Transmissor de RF

Para que fosse possível a geração do diagrama de radiação da antena, foi necessário primeiramente a montagem de um sistema de transmissão de RF, o que foi feito empregando-se o programa RPITX rodando em um Raspberry PI modelo 2B, onde por meio do chaveamento de um de seus pinos GPIO (General Purpose Input-Output) é gerado um sinal de rádio de frequência ajustável. Uma tela desse programa pode ser visto na figura 22, onde é possível observar as diversas opções que o programa oferece, dentre elas a utilizada nesse Trabalho de Graduação, onde é transmitida uma portadora com o formato de uma onda quadrada, sendo a harmônica fundamental na frequência desejada.

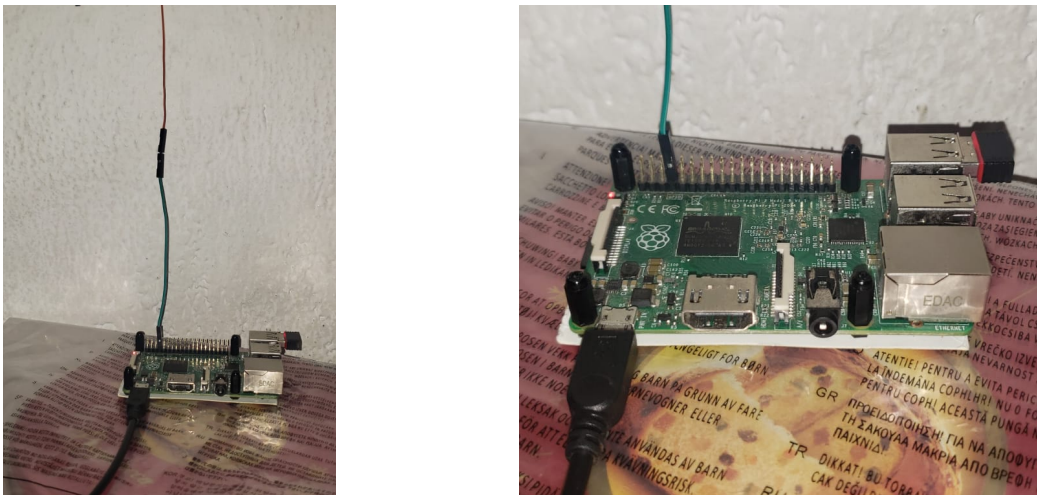
Figura 22 – *Screenshot* do aplicativo RPITX.



Fonte: O Autor (2021).

A frequência para a transmissão e subsequente medição da antena pode ser selecionada dentro de uma faixa que varia desde 5 kHz até 1,5 GHz, de forma que existe uma grande flexibilidade com relação à frequência transmitida. A primeira montagem experimental do transmissor (utilizada na medida da seção 7.1) pode ser vista na figura 23 (a), com o detalhe da conexão da antena na figura 23 (b).

Figura 23 – Transmissor RF montado - Raspberry PI 2.

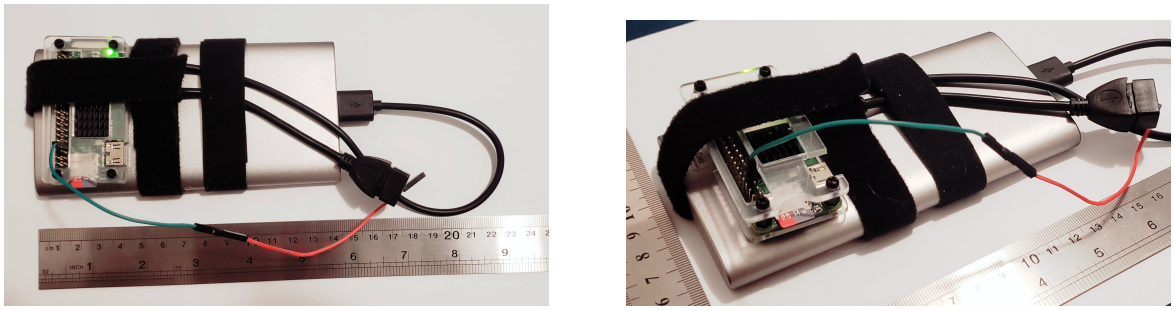


(a) Sistema de transmissão em funcionamento. (b) Detalhe da conexão.

Fonte: O Autor (2021).

Posteriormente outro sistema de transmissão foi montado, dessa utilizando para tanto o hardware do Raspberry PI Zero W, uma bateria portátil e os cabos para alimentação. Tal montagem pode ser vista na figura 24.

Figura 24 – Transmissor RF montado - Raspberry PI Zero W.



(a) Sistema de transmissão em funcionamento. (b) Detalhe da conexão.

Fonte: O Autor (2021).

O segundo sistema foi montado em substituição ao primeiro por permitir um melhor posicionamento do transmissor, dado que o primeiro não possuía nenhum tipo de proteção da placa tornando assim mais difícil a adequação do local. Em comparação, o segundo pode ser instalado em qualquer superfície plana, uma vez que encontra-se bem isolado do ambiente por meio das chapas de acrílico que compõem a carcaça do dispositivo.

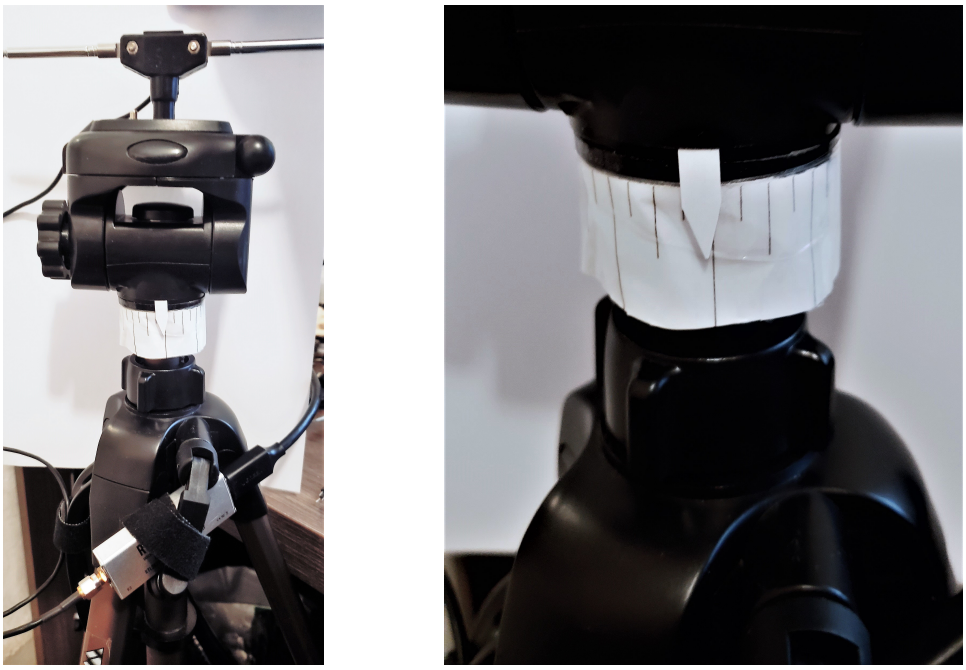
Outra vantagem da utilização do Raspberry Pi Zero W em relação ao Raspberry Pi 2 é o fato de contar com rede sem fio integrada à placa, dispensando a necessidade de utilizar um adaptador separado. Com isso é possível obter não só um menor custo da solução final como também uma maior estabilidade do sistema, uma vez que não depende de fios ou conectores. A utilização da conexão via rede sem fio é interessante pois permite que o controle seja feito de forma remota por meio do protocolo SSH (*Secure Shell*), o que permite que as medidas sejam tomadas de forma mais conveniente para o usuário.

6.2 Sistema de Medição

6.2.1 Montagem Mecânica

Na parte mecânica do sistema de medição é utilizada uma base para fixação da antena, que consiste num tripé de alumínio com a parte superior móvel. Foi fixado o RTL-SDR em um de seus componentes por meio de uma fita velcro conforme pode ser visto na figura 25 (a). Além disso, para que fosse possível medir o ângulo da medição foi instalada uma régua dividindo sua parte superior em 32 divisões, sendo assim 11,25 graus por divisão conforme figura 25 (b).

Figura 25 – Sistema de medição - montagem mecânica.



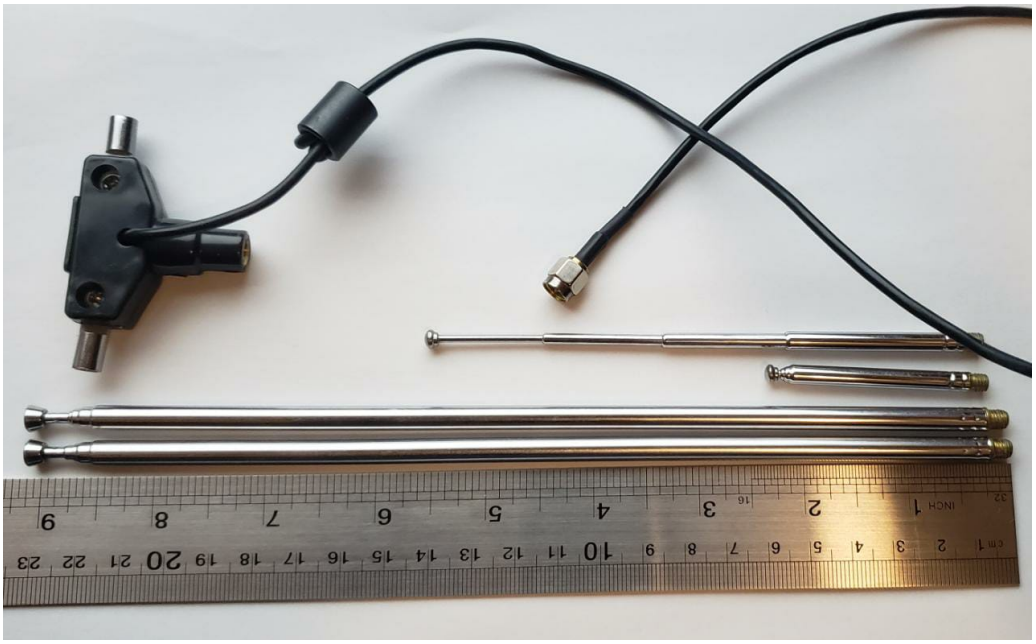
(a) Montagem mecânica do Sistema. (b) Detalhe do medidor de ângulo.

Fonte: O Autor (2021).

6.2.2 Antena

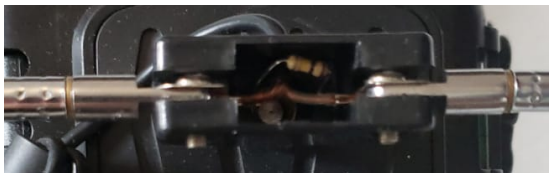
A antena utilizada para as medições desse Trabalho de Graduação é uma antena do tipo dipolo, com dois pares de elementos intercambiáveis e telescópicos, que podem ser vistos na figura 26. O menor dos elementos tem comprimento variável de 5,5 a 13,5cm, enquanto o maior pode ser configurado desde 23 até 100cm; possui em sua base também uma rosca para permitir a fixação da mesma num tripé, conta ainda com um conector SMA para conexão com o RTL-SDR. Um detalhe da parte interna da base pode ser visto na figura 27

Figura 26 – Antenas utilizadas nas medições - Base e elementos.



Fonte: O Autor (2021).

Figura 27 – Detalhe da parte interna da base.



Fonte: O Autor (2021).

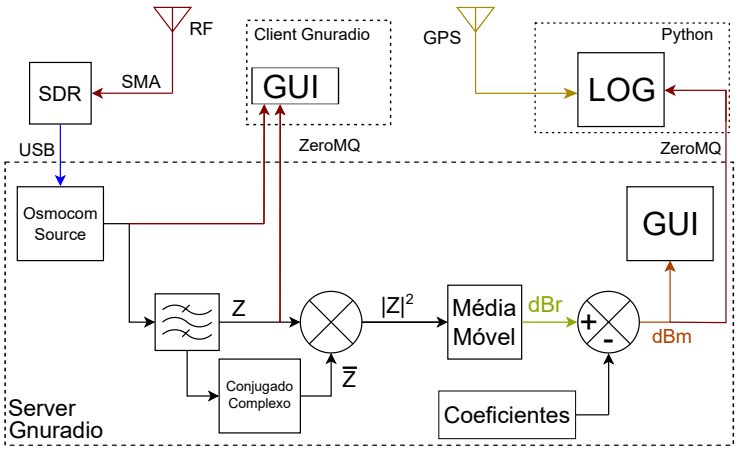
6.2.3 Software

Com relação ao software empregado nesse Trabalho de Graduação, é essencial ressaltar a divisão do mesmo em duas partes sendo elas o sistema de medição de potência e o sistema de captura, registro e plotagem do diagrama de radiação. Dada a modularidade do sistema, serão abordados separadamente a seguir.

6.2.3.1 Sistema de Medição de Potência

O sistema de medição de potência RF utilizado nesse Trabalho de Graduação é uma adaptação do sistema de medição desenvolvido por Silva (2021), de modo que o sistema de captura utiliza os dados enviados por esse módulo para a captura da potência. O sistema original pode ser visto na figura 28.

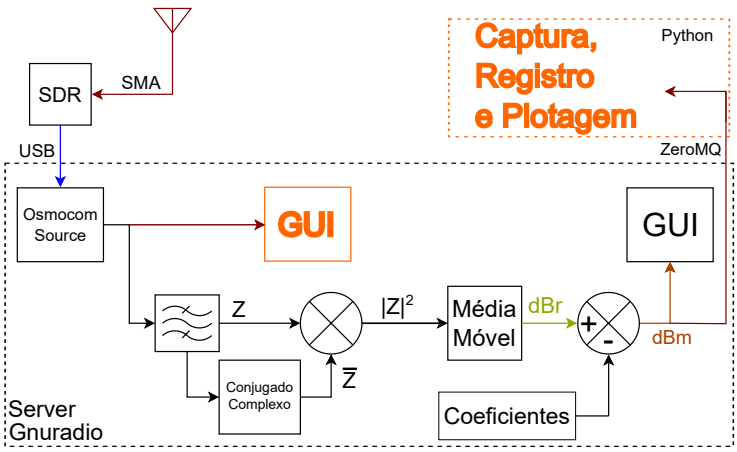
Figura 28 – Diagrama do sistema de medição de RF desenvolvido por Silva (2021).



Fonte: (SILVA, 2021).

Com as modificações realizadas destacadas em laranja na figura 29.

Figura 29 – Diagrama do sistema de medição de RF modificado.



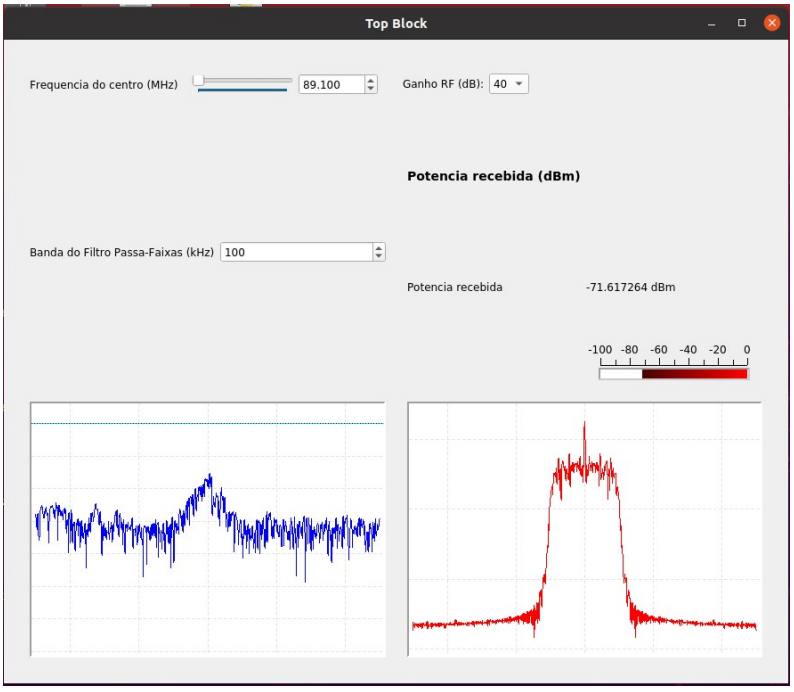
Fonte: O Autor (2021).

Basicamente o que foi realizado foi a troca do módulo original de georeferenciamento pelo módulo de Captura, Registro e Plotagem, bem como a inserção dos analisadores de espectro de forma interna ao programa, uma vez que não existia mais a necessidade de um baixo consumo de energia nem limitações computacionais, já que esse programa não mais rodaria dentro de um Raspberry PI e passaria a utilizar um computador convencional.

O sistema de medição funciona calculando a intensidade média do sinal multiplicado pelo seu conjugado complexo, obtendo-se assim o módulo do sinal ao quadrado (SILVA, 2021), que ao ser calculada por meio da média móvel, apresenta correlação direta com a energia do sinal. Aproveitando-se das propriedades do Teorema de Parseval(LATHI, 2019),combinado com a filtragem prévia do sinal (por meio de um filtro passa-faixas dentro do programa) obtém-se então a potência para determinada faixa de frequências. Após o cálculo da potência o sinal ainda necessita de uma correção para que o sistema entregue em sua saída corresponder à potência em dBm. Isso ocorre pois o sinal original capturado e entregue ao programa para processamento possui diversos níveis de ganho, bem como existem perdas internas ao dispositivo que necessitam de correção.

A interface do programa é representada na figura 30, onde é possível observar os controles da frequência central, ganho de RF, banda do filtro interno passa-faixas bem como os diagramas que mostram o espectro original em azul e após a filtragem (cuja potência é medida) em vermelho.

Figura 30 – Tela do servidor modificado.

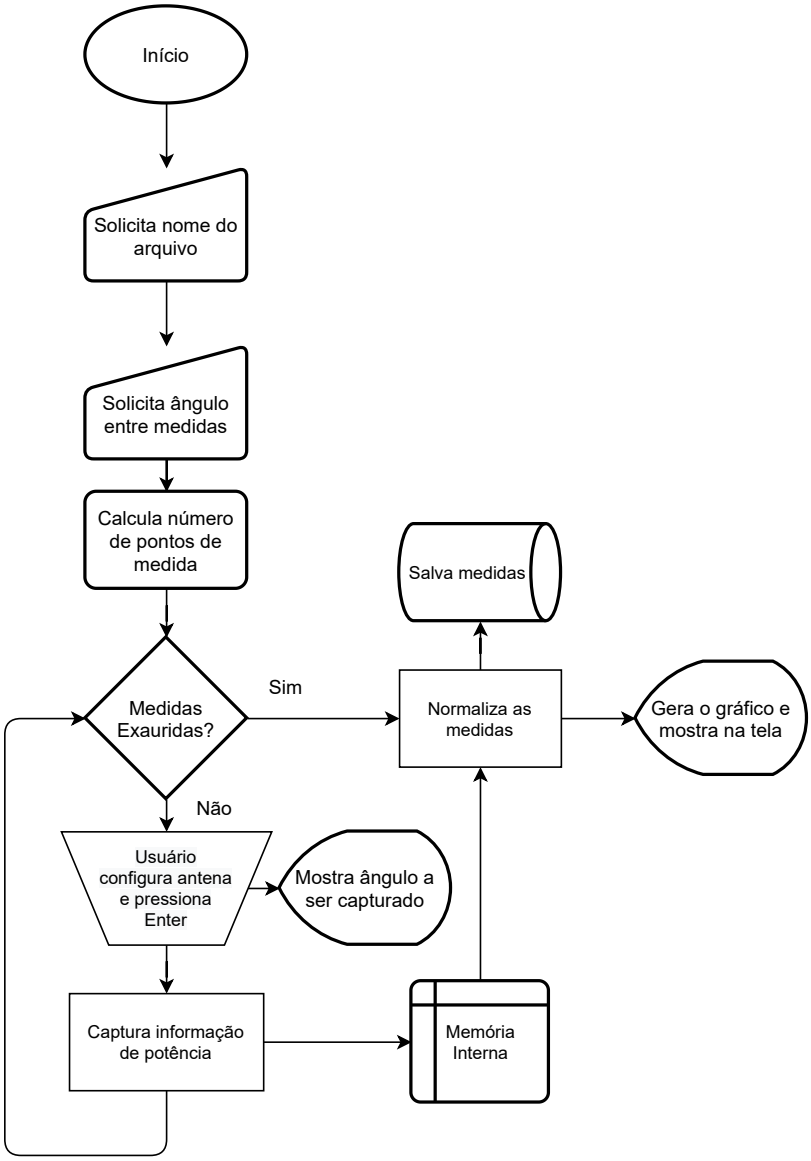


Fonte: O Autor (2021).

6.2.3.2 Sistema de Captura, Registro e Plotagem

Um fluxograma do sistema de captura, registro e plotagem desenvolvido nesse Trabalho de Graduação pode ser visualizado na figura 31.

Figura 31 – Fluxograma do Sistema de Captura, Registro e Plotagem.



Fonte: O Autor (2021).

Com relação ao acesso à informação de potência, o programa segue os seguintes passos:

- O programa se conecta via interface ZeroMQ ao sistema de medição de RF modificado (figura 29);
- ao receber os dados os mesmos são armazenados internamente junto com a informação de ângulo;

- a interface ZeroMQ é desconectada do servidor.

Ao fim da captura dos dados de potência, o programa ainda normaliza os dados, de forma a considerar o ponto cuja potência capturada foi a máxima, assim se tornando a referência de 0 dB de ganho. Isso é feito de forma a gerar um diagrama normalizado do ganho, uma vez que com a modificação do *setup* de transmissão a cada conjunto de medidas essa se torna uma forma eficiente de comparação, especialmente quanto à diretividade da antena e não necessariamente seu ganho comparado com outras antenas. Para computar o ganho propriamente dito seria necessário o uso de antenas de referência, de ganho conhecido, disponibilizadas comercialmente com alto custo, inviável para a maioria das explorações acadêmicas.

Apesar disso, com modificações no programa seria possível criar um programa para comparação entre a antena estudada e uma antena isotrópica por exemplo, bastando para isso capturar a potência recebida por tal antena e posteriormente utilizando tal potência como métrica para a normalização, ao invés do ponto de maior potência.

7 RESULTADOS

7.1 Medição do Dipolo - f = 200 MHz

7.1.1 Configuração experimental

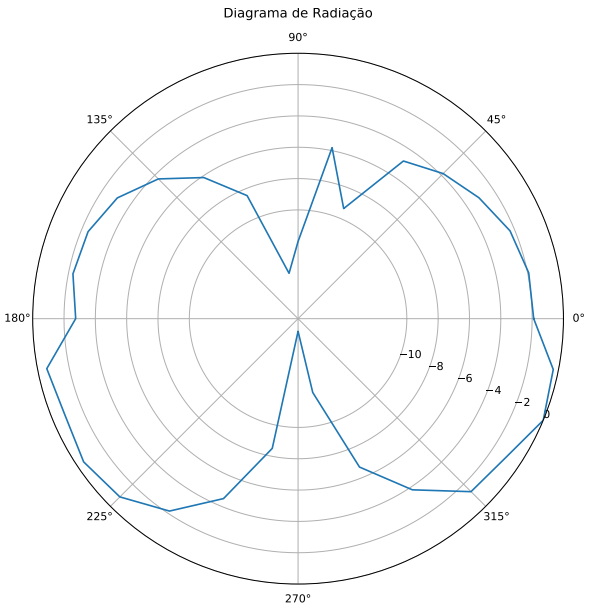
Para este experimento,a antena de recepção teve seu dipolo configurado para um comprimento de aproximadamente 24 cm por elemento, com a antena de transmissão localizada à aproximadamente 2 metros de distância; a antena foi montada no tripé metálico por meio de um parafuso já existente na base da mesma, de forma que os elementos ficaram alinhados perfeitamente com a base do tripé. A transmissão foi executada utilizando-se o programa RPITX rodando em um hardware Raspberry PI 2, rodando o sofware raspbian alimentado por uma bateria externa.

Os dados desse experimento foram capturados utilizando-se um RTL-SDR v3 conectado à uma máquina virtual (Oracle VM VirtualBox) rodando o sistema operacional Linux (Ubuntu 20.04.1).

7.1.2 Dados obtidos

Após capturados os dados, o programa gerou um gráfico, de forma que o resultado pode ser visto na figura 32.

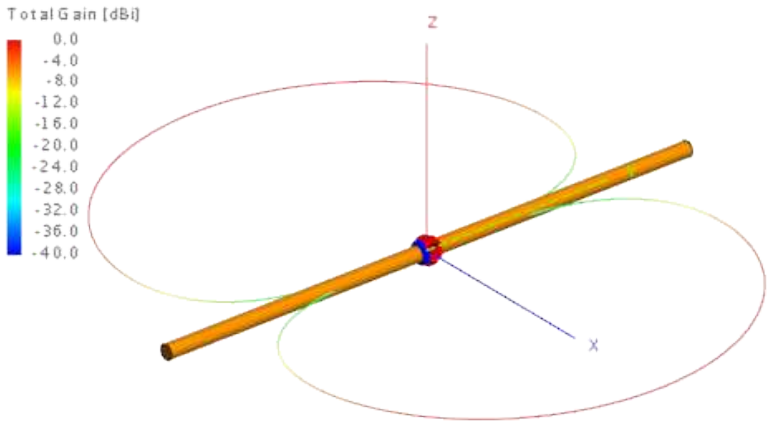
Figura 32 – Medição do Dipolo - Diagrama de Radiação normalizada - F = 200MHz.



Fonte: O Autor (2021).

Além disso, também foi realizada uma simulação no Altair Feko, com o resultado demonstrado na figura 33.

Figura 33 – Diagrama de radiação obtido - Simulação Altair Feko - 200 MHz.



Fonte: O Autor (2021).

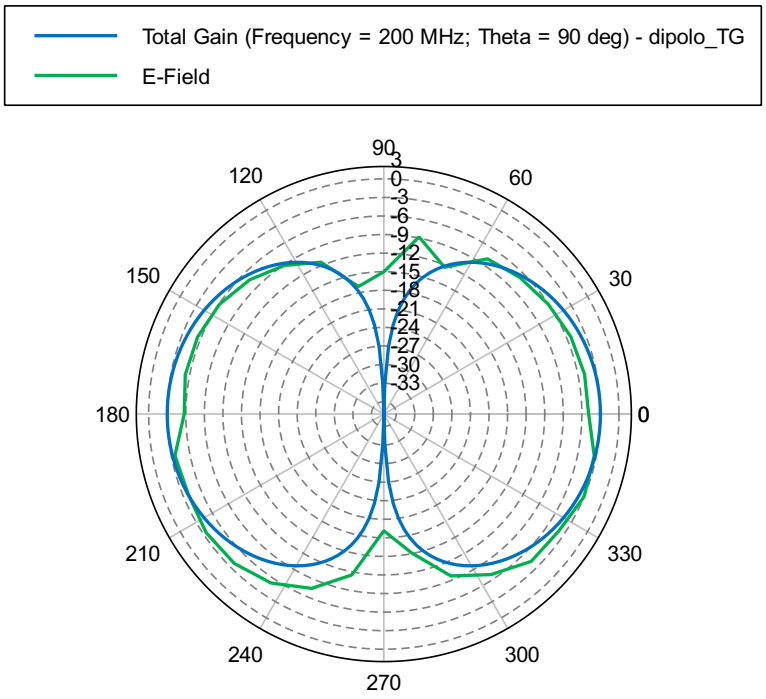
Para fins de comparação, a figura 34 mostra a antena que serviu de base para simulação, e que foi utilizada para a captura dos dados:

Figura 34 – Detalhe da antena - Dipolo 200MHz - 28cm.

Fonte: O Autor (2021).

Finalmente, a comparação entre a medida capturada e a simulação é mostrada na figura 35, onde o traço em azul representa a simulação e o verde a medida realizada:

Figura 35 – Comparação entre medida e simulação - 200 MHz.



Fonte: O Autor (2021).

7.2 Medição do Dipolo - f = 434 MHz

7.2.1 Configuração experimental

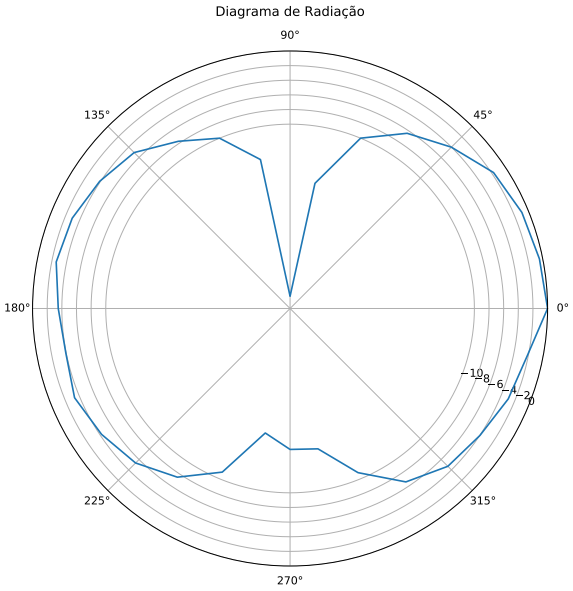
Neste experimento foi utilizado o sistema novo de transmissão, sendo composto pelo Raspberry PI Zero W, em conjunto com uma bateria. Na parte da recepção a antena teve seu elemento trocado para um menor, com um comprimento de aproximadamente 6cm por elemento. Também foi trocada a localização do transmissor, que encontra-se à aproximadamente 3,15 metros com uma parede de alvenaria entre o transmissor e o receptor (tal mudança foi realizada para mitigar parte dos efeitos de reflexões nas medidas).

Os dados desse experimento novamente foram capturados utilizando-se um RTL-SDR v3 conectado à mesma máquina virtual do experimento da seção 7.1(Oracle VM VirtualBox) rodando o sistema operacional Linux (Ubuntu 20.04.1).

7.2.2 Dados obtidos

Após capturados os dados, o programa gerou um gráfico, de forma que o resultado pode ser visto na figura 36.

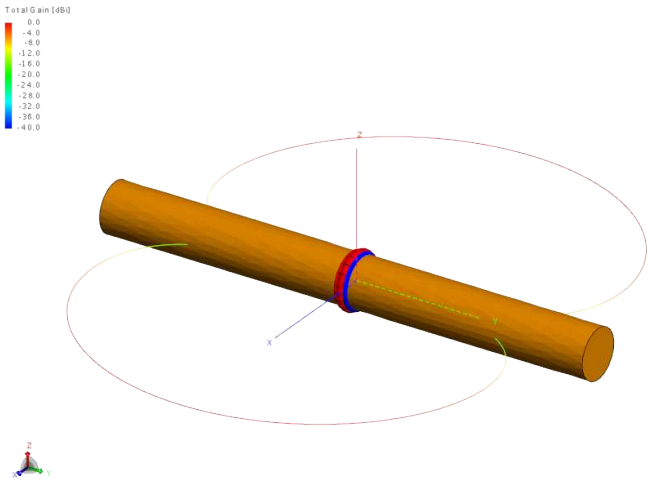
Figura 36 – Medição do Dipolo - Diagrama de Radiação normalizada - f = 434MHz.



Fonte: O Autor (2021).

Além disso, também foi realizada uma simulação no Altair Feko, com o resultado apresentado na figura 33.

Figura 37 – Diagrama de radiação obtido - Simulação Altair Feko - 434 MHz.



Fonte: O Autor (2021).

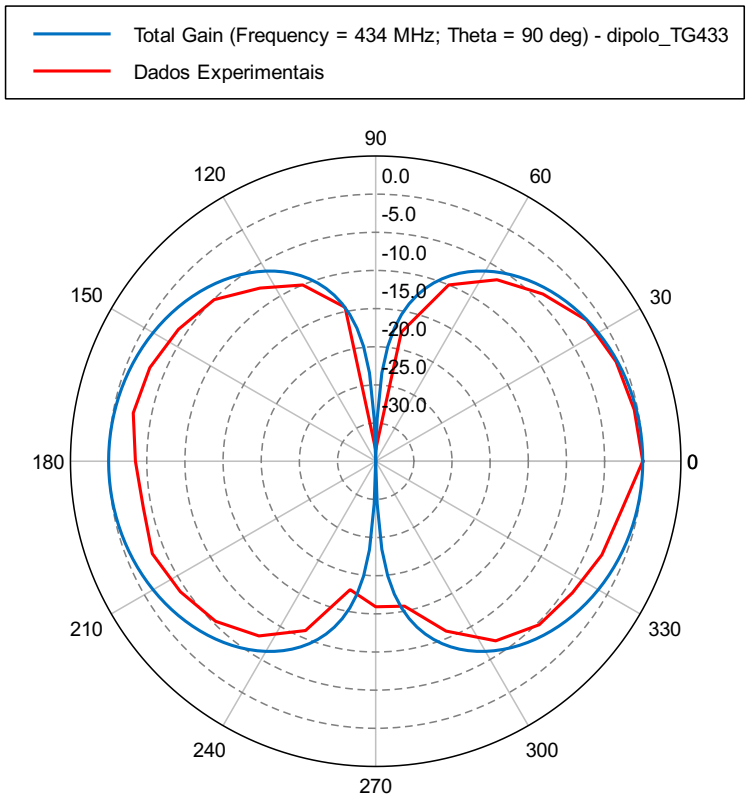
Novamente para fins de comparação, na figura 38 é apresentada uma imagem da antena que serviu como parâmetro para a simulação, e que foi utilizada para a captura dos dados:

Figura 38 – Detalhe da antena - Dipolo 434MHz - 6cm.

Fonte: O Autor (2021).

Finalmente, a comparação entre a medida capturada e a simulação é mostrada na figura 39, onde o traço azul representa a simulação e o vermelho os dados experimentais.

Figura 39 – Comparação entre medida e simulação - 434 MHz.



Fonte: O Autor (2021).

8 CONCLUSÃO

Conforme os resultados obtidos experimentalmente, é verificada primeiramente a grande versatilidade dos rádios definidos por software nos estudos referentes à propagação de RF, uma vez que é possível o desenvolvimento de diversas soluções com um mesmo hardware sem grandes modificações ou a necessidade de aquisição de equipamento adicional.

Parte essencial dessa versatilidade também é demonstrada na possibilidade de utilização dos rádios em diversas plataformas diferentes graças à utilização de software livre (Gnuradio e Linux), de forma que é possível a criação desde sistemas embarcados utilizando a plataforma Raspberry PI ou similares, quanto sistemas fixos utilizando-se computadores convencionais.

Outro ponto que é importante destacar é que com a utilização do sistema em ambientes cujas reflexões e outros efeitos sejam mais controlados, como por exemplo em uma câmara anecóica, é possível a coleta de dados mais robustos do que os que foram colhidos ao longo desse Trabalho de Graduação. Dessa forma, quando é realizada uma análise crítica dos resultados tendo em vista as limitações impostas tanto na parte mecânica quanto na parte do ambiente de testes, é possível inferir que os resultados obtidos possuem similaridade com aqueles simulados, de forma que foram satisfatórios.

8.1 Desenvolvimentos futuros

Como desenvolvimento futuro principal para o sistema de caracterização de antenas encontra-se a adaptação de uma plataforma de material não metálico (madeira ou plástico), bem como a motorização da base por meio de um motor de passo, de modo que o ângulo possa ser controlado com um ajuste fino, de forma completamente automática e com um passo menor do que o obtido atualmente (de 5.625 graus).

Além disso, seria interessante no aspecto da plataforma que em sua ponta dispusesse de uma base comum, onde por meio da utilização de uma impressora 3d diversos tipos de antena pudessem ser montados de forma eficiente para que fossem caracterizadas, independente de sua própria geometria.

Outro desenvolvimento possível seria a comparação do sistema com um outro hardware dedicado que realize medições de antenas, dentro de um ambiente controlado e que possibilite a comparação entre os mesmos, bem como a utilização de antenas com geometria mais exótica para fins de teste.

REFERÊNCIAS

ARAUJO, R. D. P. de. **A predição da propagação eletromagnética em LTE (4G) em um ambiente urbano densamente povoado.** Dissertação (Mestrado) — Universidade Federal do ABC, Santo André, 2019.

BALANIS, C. **Antenna theory : analysis and design**. Hoboken, New Jersey: John Wiley, 2016. ISBN 978-1-118-64206-1.

DI, T. F. C. **Software-Defined Radio for Engineers**. Artech House, 2018. ISBN 1630814571. Disponível em: https://www.xarg.org/ref/a/1630814571/.

IEEE Standard Test Procedures for Antennas. IEEE, 2008. Disponível em: https://doi.org/10.1109/ieeestd.1979.120310.

JOHNSON, P. New research lab leads to unique radio receiver. **E-Systems TEAM**, v. 5, n. 4, p. 6–7, May 1985. Disponível em: http://chordite.com/team.pdf.

LATHI, B. P. **Modern digital and analog communication systems**. New York: Oxford University Press, 2019. ISBN 9780190686840.

LAUFER, C. **The hobbyist's guide to the RTL-SDR : really cheap software defined radio : a guide to the RTL-SDR and cheap software defined radio by the authors of the RTL-SDR.com blog**. United States: Carl Laufer, 2015. ISBN 9781514716694.

MACHADO, R. G.; WYGLINSKI, A. M. Software-defined radio: Bridging the analog–digital divide. **Proceedings of the IEEE**, Institute of Electrical and Electronics Engineers (IEEE), v. 103, n. 3, p. 409–423, mar. 2015. Disponível em: https://doi.org/10.1109/jproc.2015.2399173.

MUTTONI, L.; VEIGA, A. Development of an 1-seg ISDB-t receiver using low-cost SDR hardware. In: **Anais de XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais**. Sociedade Brasileira de Telecomunicações, 2017. Disponível em: https://doi.org/10.14209/sbrt.2017.153.

SILVA, F. A. A. da. **Sistema de Medição de RF de Baixo Custo baseado em SDR**. Dissertação (Mestrado) — Universidade Federal do ABC, Santo André, 2021.

STEWART, R.; BARLEE, K.; ATKINSON, D.; CROCKETT, L. **Software Defined Radio using MATLAB & Simulink and the RTL-SDR**. [S.l.: s.n.], 2015. ISBN 978-0-9929787-2-3.

TRIBBLE, A. C. The software defined radio: Fact and fiction. In: **2008 IEEE Radio and Wireless Symposium**. IEEE, 2008. Disponível em: https://doi.org/10.1109/rws.2008.4463414.

TUTTLEBEE, W. **Software defined radio : origins, drivers, and international perspectives**. West Sussex, England New York: John Wiley, 2002. ISBN 978-0470844649.

WRIGHT, D. P.; BALL, E. A. Highly portable, low cost SDR instrument for RF propagation studies. **IEEE Transactions on Instrumentation and Measurement**, Institute of Electrical and Electronics Engineers (IEEE), p. 1–1, 2019. Disponível em: https://doi.org/10.1109/tim.2019.2959422.

Apêndices

APÊNDICE A – PROGRAMA PARA CAPTURA DOS DADOS

```
import zmq   #zeroMQ interface
import numpy #converter numeros
import matplotlib.pyplot as plt

# ZeroMQ Context
context = zmq.Context()
# Define the socket using the "Context"
sock = context.socket(zmq.SUB)
# Conecta ao programa server
sock.setsockopt(zmq.SUBSCRIBE, b"")

arquivo = input("Digite o nome do arquivo, e aperte enter para continuar:
↪  "+"\n")
print(arquivo)
x = 0
graus = float(input("Digite o intervalo em graus entre cada
↪  medida:"+"\n"))
pontos = (360/graus)
potencias = []
angulos = []
while x < pontos:
 angulo = str(x*graus)
 angulos.append(x*graus)
 input("Configure a antena num angulo de "+angulo+" graus, e aperte Enter
 ↪  para Continuar")
 sock.connect("tcp://127.0.0.1:120")
 message= sock.recv()     #potencia
 sock.disconnect("tcp://127.0.0.1:120")
 tem = numpy.frombuffer(message,dtype=numpy.float32) #potencia
 trem = numpy.array2string(tem.flat[0])   #potencia
 #f = open(arquivo, "a")
 potencias.append(float(trem))

 #print(trem+" "+angulo+"\n")
 #f.write(trem+" "+angulo+"\n")
 #f.close()
```

```
 x = x+1
#normalizar

maximo = max(potencias)
print(maximo)
print(str(potencias))
for i in range(len(potencias)):
  potencias[i] =potencias[i]- float(maximo)

print(angulos)
print(str(potencias))
f = open(arquivo, "a")
f.write(str(angulos)+"\n")
f.write(str(potencias))
f.close()

x = angulos
y = potencias
#fecha o grafico
x.append(x[0])
y.append(y[0])
#inicio do processamento para plotar

z = [element * 3.1415/180 for element in x]
ax = plt.subplot(111, projection='polar')
ax.plot(z, y)
ax.set_rmax(0)
ax.set_rticks([-10,-8,-6,-4,-2, 0])  # componentes da legenda
ax.set_rlabel_position(-22.5)  # posicao da legenda interna
ax.grid(True)
plt.gcf().set_size_inches(10,10)
ax.set_title("Diagrama de Radiação", va='bottom')
plt.show()
```

APÊNDICE B – SERVIDOR MODIFICADO

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#
# SPDX-License-Identifier: GPL-3.0
#
# GNU Radio Python Flow Graph
# Title: Top Block
# GNU Radio version: 3.8.1.0

from distutils.version import StrictVersion

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print("Warning: failed to XInitThreads()")

from PyQt5 import Qt
from PyQt5.QtCore import QObject, pyqtSlot
from gnuradio import qtgui
from gnuradio.filter import firdes
import sip
from gnuradio import analog
from gnuradio import blocks
from gnuradio import filter
from gnuradio import gr
import sys
import signal
from argparse import ArgumentParser
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
```

```
from gnuradio import zeromq
from gnuradio.qtgui import Range, RangeWidget
import osmosdr
import time
from gnuradio import qtgui

class top_block(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Top Block")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Top Block")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "top_block")

        try:
            if StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):

                ↪  self.restoreGeometry(self.settings.value("geometry").toByteArra
            else:
                self.restoreGeometry(self.settings.value("geometry"))
        except:
            pass
```

```
##################################################
# Variables
##################################################
self.variable_qtgui_range_0 = variable_qtgui_range_0 = 733
self.samp_rate = samp_rate = 1e6
self.ganho_rf = ganho_rf = 0
self.banda_filtro = banda_filtro = 100

##################################################
# Blocks
##################################################
self._variable_qtgui_range_0_range = Range(88, 1200, 0.1, 733,
↪  100)
self._variable_qtgui_range_0_win =
↪  RangeWidget(self._variable_qtgui_range_0_range,
↪  self.set_variable_qtgui_range_0, 'Frequencia do centro (MHz)',
↪  "counter_slider", float)
self.top_grid_layout.addWidget(self._variable_qtgui_range_0_win,
↪  0, 0, 1, 1)
for r in range(0, 1):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(0, 1):
    self.top_grid_layout.setColumnStretch(c, 1)
# Create the options list
self._ganho_rf_options = (0, 20, 40, )
# Create the labels list
self._ganho_rf_labels = ('0', '20', '40', )
# Create the combo box
self._ganho_rf_tool_bar = Qt.QToolBar(self)
self._ganho_rf_tool_bar.addWidget(Qt.QLabel('Ganho RF (dB)' + ":
↪  "))
self._ganho_rf_combo_box = Qt.QComboBox()
self._ganho_rf_tool_bar.addWidget(self._ganho_rf_combo_box)
for _label in self._ganho_rf_labels:
↪  self._ganho_rf_combo_box.addItem(_label)
```

```
self._ganho_rf_callback = lambda i:
↪  Qt.QMetaObject.invokeMethod(self._ganho_rf_combo_box,
↪  "setCurrentIndex", Qt.Q_ARG("int",
↪  self._ganho_rf_options.index(i)))
self._ganho_rf_callback(self.ganho_rf)
self._ganho_rf_combo_box.currentIndexChanged.connect(
    lambda i: self.set_ganho_rf(self._ganho_rf_options[i]))
# Create the radio buttons
self.top_grid_layout.addWidget(self._ganho_rf_tool_bar, 0, 1, 1,
↪  1)
for r in range(0, 1):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(1, 2):
    self.top_grid_layout.setColumnStretch(c, 1)
self._banda_filtro_range = Range(1, 1000, 1, 100, 100)
self._banda_filtro_win = RangeWidget(self._banda_filtro_range,
↪  self.set_banda_filtro, 'Banda do Filtro Passa-Faixa (kHz)',
↪  "counter", float)
self.top_grid_layout.addWidget(self._banda_filtro_win, 1, 0, 1,
↪  1)
for r in range(1, 2):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(0, 1):
    self.top_grid_layout.setColumnStretch(c, 1)
self.zeromq_pub_sink_0 = zeromq.pub_sink(gr.sizeof_float, 1,
↪  'tcp://127.0.0.1:120', 100, False, -1)
self.qtgui_number_sink_0 = qtgui.number_sink(
    gr.sizeof_float,
    1,
    qtgui.NUM_GRAPH_HORIZ,
    1
)
self.qtgui_number_sink_0.set_update_time(0.10)
self.qtgui_number_sink_0.set_title('Potencia recebida (dBm)')

labels = ['Potencia recebida', '', '', '', '',
    '', '', '', '', '']
units = ['dBm', '', '', '', '',
    '', '', '', '', '']
```

```
colors = [("black", "red"), ("black", "black"), ("black",
↪  "black"), ("black", "black"), ("black", "black"),
    ("black", "black"), ("black", "black"), ("black", "black"),
    ↪  ("black", "black"), ("black", "black")]
factor = [1, 1, 1, 1, 1,
    1, 1, 1, 1, 1]

for i in range(1):
    self.qtgui_number_sink_0.set_min(i, -100)
    self.qtgui_number_sink_0.set_max(i, 0)
    self.qtgui_number_sink_0.set_color(i, colors[i][0],
    ↪  colors[i][1])
    if len(labels[i]) == 0:
        self.qtgui_number_sink_0.set_label(i, "Data
        ↪  {0}".format(i))
    else:
        self.qtgui_number_sink_0.set_label(i, labels[i])
    self.qtgui_number_sink_0.set_unit(i, units[i])
    self.qtgui_number_sink_0.set_factor(i, factor[i])

self.qtgui_number_sink_0.enable_autoscale(False)
self._qtgui_number_sink_0_win =
↪  sip.wrapinstance(self.qtgui_number_sink_0.pyqwidget(),
↪  Qt.QWidget)
self.top_grid_layout.addWidget(self._qtgui_number_sink_0_win, 1,
↪  1, 1, 1)
for r in range(1, 2):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(1, 2):
    self.top_grid_layout.setColumnStretch(c, 1)
self.qtgui_freq_sink_x_1 = qtgui.freq_sink_c(
    1024, #size
    firdes.WIN_BLACKMAN_hARRIS, #wintype
    0, #fc
    samp_rate, #bw
    "", #name
    1
)
self.qtgui_freq_sink_x_1.set_update_time(0.10)
```

```
self.qtgui_freq_sink_x_1.set_y_axis(-140, 10)
self.qtgui_freq_sink_x_1.set_y_label('Relative Gain', 'dB')
self.qtgui_freq_sink_x_1.set_trigger_mode(qtgui.TRIG_MODE_FREE,
↪  0.0, 0, "")
self.qtgui_freq_sink_x_1.enable_autoscale(True)
self.qtgui_freq_sink_x_1.enable_grid(True)
self.qtgui_freq_sink_x_1.set_fft_average(1.0)
self.qtgui_freq_sink_x_1.enable_axis_labels(False)
self.qtgui_freq_sink_x_1.enable_control_panel(False)

self.qtgui_freq_sink_x_1.disable_legend()


labels = ['', '', '', '', '',
    '', '', '', '', '']
widths = [1, 1, 1, 1, 1,
    1, 1, 1, 1, 1]
colors = ["red", "red", "green", "black", "cyan",
    "magenta", "yellow", "dark red", "dark green", "dark blue"]
alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0]

for i in range(1):
    if len(labels[i]) == 0:
        self.qtgui_freq_sink_x_1.set_line_label(i, "Data
        ↪  {0}".format(i))
    else:
        self.qtgui_freq_sink_x_1.set_line_label(i, labels[i])
    self.qtgui_freq_sink_x_1.set_line_width(i, widths[i])
    self.qtgui_freq_sink_x_1.set_line_color(i, colors[i])
    self.qtgui_freq_sink_x_1.set_line_alpha(i, alphas[i])

self._qtgui_freq_sink_x_1_win =
↪  sip.wrapinstance(self.qtgui_freq_sink_x_1.pyqwidget(),
↪  Qt.QWidget)
self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_1_win, 2,
↪  1, 1, 1)
for r in range(2, 3):
    self.top_grid_layout.setRowStretch(r, 1)
```

```
for c in range(1, 2):
    self.top_grid_layout.setColumnStretch(c, 1)
self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
    1024, #size
    firdes.WIN_BLACKMAN_hARRIS, #wintype
    0, #fc
    samp_rate, #bw
    "", #name
    1
)
self.qtgui_freq_sink_x_0.set_update_time(0.10)
self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE,
↪  0.0, 0, "")
self.qtgui_freq_sink_x_0.enable_autoscale(False)
self.qtgui_freq_sink_x_0.enable_grid(True)
self.qtgui_freq_sink_x_0.set_fft_average(1.0)
self.qtgui_freq_sink_x_0.enable_axis_labels(False)
self.qtgui_freq_sink_x_0.enable_control_panel(False)

self.qtgui_freq_sink_x_0.disable_legend()


labels = ['', '', '', '', '',
    '', '', '', '', '']
widths = [1, 1, 1, 1, 1,
    1, 1, 1, 1, 1]
colors = ["blue", "red", "green", "black", "cyan",
    "magenta", "yellow", "dark red", "dark green", "dark blue"]
alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0]

for i in range(1):
    if len(labels[i]) == 0:
        self.qtgui_freq_sink_x_0.set_line_label(i, "Data
        ↪ {0}".format(i))
    else:
        self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
```

```
        self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
        self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
        self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

    self._qtgui_freq_sink_x_0_win =
    ↪  sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(),
    ↪  Qt.QWidget)
    self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win, 2,
    ↪  0, 1, 1)
    for r in range(2, 3):
        self.top_grid_layout.setRowStretch(r, 1)
    for c in range(0, 1):
        self.top_grid_layout.setColumnStretch(c, 1)
    self.osmosdr_source_0 = osmosdr.source(
        args="numchan=" + str(1) + " " + ''
    )
    self.osmosdr_source_0.set_clock_source('external', 0)
    self.osmosdr_source_0.set_time_unknown_pps(osmosdr.time_spec_t())
    self.osmosdr_source_0.set_sample_rate(samp_rate)
    self.osmosdr_source_0.set_center_freq(variable_qtgui_range_0*1e6,
    ↪  0)
    self.osmosdr_source_0.set_freq_corr(0, 0)
    self.osmosdr_source_0.set_gain(ganho_rf, 0)
    self.osmosdr_source_0.set_if_gain(0, 0)
    self.osmosdr_source_0.set_bb_gain(0, 0)
    self.osmosdr_source_0.set_antenna('', 0)
    self.osmosdr_source_0.set_bandwidth(100e3, 0)
    self.blocks_threshold_ff_1 = blocks.threshold_ff(21, 30, 0)
    self.blocks_threshold_ff_0 = blocks.threshold_ff(5, 15, 0)
    self.blocks_sub_xx_0 = blocks.sub_ff(1)
    self.blocks_nlog10_ff_0 = blocks.nlog10_ff(10, 1, 0)
    self.blocks_multiply_xx_2_0 = blocks.multiply_vff(1)
    self.blocks_multiply_xx_2 = blocks.multiply_vff(1)
    self.blocks_multiply_conjugate_cc_0 =
    ↪  blocks.multiply_conjugate_cc(1)
    self.blocks_moving_average_xx_0 = blocks.moving_average_ff(50000,
    ↪  20, 4000, 1)
    self.blocks_complex_to_real_0 = blocks.complex_to_real(1)
    self.blocks_add_xx_0 = blocks.add_vff(1)
```

```
self.band_pass_filter_0 = filter.fir_filter_ccc(
    1,
    firdes.complex_band_pass(
        1,
        samp_rate,
        -(banda_filtro*1e3/2),
        (banda_filtro*1e3/2),
        100e2,
        firdes.WIN_HANN,
        6.76))
self.analog_const_source_x_0 = analog.sig_source_f(0,
↪  analog.GR_CONST_WAVE, 0, 0, ganho_rf)
self.a40db = analog.sig_source_f(0, analog.GR_CONST_WAVE, 0, 0,
↪  18.1)
self.a20db = analog.sig_source_f(0, analog.GR_CONST_WAVE, 0, 0,
↪  25.2)
self.a0db = analog.sig_source_f(0, analog.GR_CONST_WAVE, 0, 0,
↪  73.8)



##################################################
# Connections
##################################################
self.connect((self.a0db, 0), (self.blocks_add_xx_0, 2))
self.connect((self.a20db, 0), (self.blocks_multiply_xx_2, 1))
self.connect((self.a40db, 0), (self.blocks_multiply_xx_2_0, 1))
self.connect((self.analog_const_source_x_0, 0),
↪  (self.blocks_threshold_ff_0, 0))
self.connect((self.analog_const_source_x_0, 0),
↪  (self.blocks_threshold_ff_1, 0))
self.connect((self.band_pass_filter_0, 0),
↪  (self.blocks_multiply_conjugate_cc_0, 0))
self.connect((self.band_pass_filter_0, 0),
↪  (self.blocks_multiply_conjugate_cc_0, 1))
self.connect((self.blocks_add_xx_0, 0), (self.blocks_sub_xx_0,
↪  1))
self.connect((self.blocks_complex_to_real_0, 0),
↪  (self.blocks_moving_average_xx_0, 0))
```

```
        self.connect((self.blocks_moving_average_xx_0, 0),
        ↪  (self.blocks_nlog10_ff_0, 0))
        self.connect((self.blocks_multiply_conjugate_cc_0, 0),
        ↪  (self.blocks_complex_to_real_0, 0))
        self.connect((self.blocks_multiply_conjugate_cc_0, 0),
        ↪  (self.qtgui_freq_sink_x_1, 0))
        self.connect((self.blocks_multiply_xx_2, 0),
        ↪  (self.blocks_add_xx_0, 0))
        self.connect((self.blocks_multiply_xx_2_0, 0),
        ↪  (self.blocks_add_xx_0, 1))
        self.connect((self.blocks_nlog10_ff_0, 0), (self.blocks_sub_xx_0,
        ↪  0))
        self.connect((self.blocks_sub_xx_0, 0), (self.qtgui_number_sink_0,
        ↪  0))
        self.connect((self.blocks_sub_xx_0, 0), (self.zeromq_pub_sink_0,
        ↪  0))
        self.connect((self.blocks_threshold_ff_0, 0),
        ↪  (self.blocks_multiply_xx_2, 0))
        self.connect((self.blocks_threshold_ff_1, 0),
        ↪  (self.blocks_multiply_xx_2_0, 0))
        self.connect((self.osmosdr_source_0, 0), (self.band_pass_filter_0,
        ↪  0))
        self.connect((self.osmosdr_source_0, 0),
        ↪  (self.qtgui_freq_sink_x_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "top_block")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_variable_qtgui_range_0(self):
        return self.variable_qtgui_range_0

    def set_variable_qtgui_range_0(self, variable_qtgui_range_0):
        self.variable_qtgui_range_0 = variable_qtgui_range_0

        ↪  self.osmosdr_source_0.set_center_freq(self.variable_qtgui_range_0*1e6,
        ↪  0)
```

```
    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.band_pass_filter_0.set_taps(firdes.complex_band_pass(1,
        ↪  self.samp_rate, -(self.banda_filtro*1e3/2),
        ↪  (self.banda_filtro*1e3/2), 100e2, firdes.WIN_HANN, 6.76))
        self.osmosdr_source_0.set_sample_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.qtgui_freq_sink_x_1.set_frequency_range(0, self.samp_rate)

    def get_ganho_rf(self):
        return self.ganho_rf

    def set_ganho_rf(self, ganho_rf):
        self.ganho_rf = ganho_rf
        self._ganho_rf_callback(self.ganho_rf)
        self.analog_const_source_x_0.set_offset(self.ganho_rf)
        self.osmosdr_source_0.set_gain(self.ganho_rf, 0)

    def get_banda_filtro(self):
        return self.banda_filtro

    def set_banda_filtro(self, banda_filtro):
        self.banda_filtro = banda_filtro
        self.band_pass_filter_0.set_taps(firdes.complex_band_pass(1,
        ↪  self.samp_rate, -(self.banda_filtro*1e3/2),
        ↪  (self.banda_filtro*1e3/2), 100e2, firdes.WIN_HANN, 6.76))



def main(top_block_cls=top_block, options=None):

    if StrictVersion("4.5.0") <= StrictVersion(Qt.qVersion()) <
    ↪  StrictVersion("5.0.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)
```

```
    tb = top_block_cls()
    tb.start()
    tb.show()

    def sig_handler(sig=None, frame=None):
        Qt.QApplication.quit()

    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)

    timer = Qt.QTimer()
    timer.start(500)
    timer.timeout.connect(lambda: None)

    def quitting():
        tb.stop()
        tb.wait()
    qapp.aboutToQuit.connect(quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```