



**Elizeu Calegari**

**Implementação do detector de energia e  
esquemas de análise de desempenho  
no GNU Radio – simulações e testes**

**Dissertação de Mestrado**

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da PUC-Rio como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: Prof. Luiz Alencar Reis da Silva Mello

Rio de Janeiro

Abril de 2018



**Elizeu Calegari**

## **Implementação do detector de energia e esquemas de análise de desempenho no GNU Radio – simulações e testes**

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Engenharia Elétrica da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

**Prof. Luiz Alencar Reis da Silva Mello**

Orientador

Centro de Estudos em Telecomunicações – PUC-Rio

**Prof. Marco Antonio Grivet Mattoso Maia**

Centro de Estudos em Telecomunicações – PUC-Rio

**Prof. Carlos Vinicio Rodríguez Ron**

INMETRO

**Prof. Pedro Vladimir Gonzalez Castellanos**

UFF

**Prof. Márcio da Silveira Carvalho**

Coordenador Setorial do Centro

Técnico Científico – PUC-Rio

Rio de Janeiro, 12 de abril de 2018

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

## **Elizeu Calegari**

Engenheiro Eletrônico e de Telecomunicações graduado pela Universidade de Aveiro, Portugal em 2001 e com diploma revalidado pela Universidade Federal Fluminense em 2009. É pesquisador do Instituto Nacional de Metrologia, Qualidade e Tecnologia (INMETRO) e atua na Divisão de Metrologia em Tecnologias da Informação e Telecomunicações (DMTIC).

### Ficha Catalográfica

Calegari, Elizeu

Implementação do detector de energia e esquemas de análise de desempenho no GNU Radio – simulações e testes / Calegari, Elizeu; orientador: Luiz Alencar Reis da Silva Mello. – 2018.

156 f.; 30 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica.

Inclui bibliografia.

1. Engenharia Elétrica-Tese. 2. Rádio Cognitivo. 3. Detector de Energia 4. Sensoriamento de Espectro. 5. GNU Radio. 6. USRP. I Da Silva Mello, Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título

CDD: 621.3

A meus pais, minha esposa e meus filhos



## Agradecimentos

A Deus por todas as vitórias alcançadas em todos os dias da minha vida pois até aqui me sustentou e sempre esteve comigo, inclusive nas horas mais difíceis.

Ao meu orientador Professor Luiz da Silva Mello pelas dicas, ensinamentos, conselhos e orientações valiosas durante todo o desenvolvimento do trabalho

À CAPES, à PUC-Rio e ao Inmetro, pelos apoios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Aos meus pais, pela educação, apoio, atenção, compreensão e carinho de todas as horas.

A minha esposa, Synara, e a meus filhos Carol, Alexandra e Arthur, por todo o apoio, paciência e compreensão durante toda a realização deste trabalho.

Aos meus amigos e colegas do Inmetro e da PUC-Rio, Jussif, Marcelo, Marta, João Braz, Mauro, Fernando, Rodrigo, Charles, Ewerton e Rafael por todo o seu apoio.

Ao meu chefe no Inmetro, Rodolfo Souza, pois sem seu apoio este trabalho não teria sido possível.

A todos os demais colegas do CETUC e do INMETRO pelas importantes contribuições neste trabalho e pelas palavras de apoio.

Aos meus professores da Graduação em Engenharia Eletrônica e de Telecomunicações da Universidade de Aveiro, nomeadamente ao José Carlos Pedro e Nuno Borges, e a todos os demais da minha graduação.

Aos professores que participaram da Comissão examinadora.

A todos os professores do Departamento de Elétrica pelos ensinamentos e pela ajuda.

A todos os amigos e familiares que de uma forma ou de outra me estimularam ou me ajudaram.

## Resumo

Calegari, Elizeu; da Silva Mello, Luiz Alencar Reis (Orientador). **Implementação do detector de energia e esquemas de análise de desempenho no GNU Radio – simulações e testes**. Rio de Janeiro, 2018. 156p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

O rádio cognitivo é uma tecnologia que visa o compartilhamento do espectro radioelétrico entre usuários primários licenciados e os demais usuários secundários de maneira harmoniosa, sem provocar interferências que prejudiquem a prestação dos serviços e visando uma melhoria na eficiência do uso do espectro radioelétrico, que é um recurso cada vez mais escasso. No âmbito da pesquisa referente a esta dissertação, é construído e implementado no GNU Radio um esquema de avaliação de desempenho de detectores para rádio cognitivo, é construído o detector de energia, e são implementadas simulações computacionais e ensaios por meio de duas USRP para avaliar o desempenho do detector criado, visando os requisitos do padrão IEEE 802.22.

## Palavras-chave

Rádio Cognitivo; Detector de Energia; Sensoriamento de Espectro; GNU Radio; USRP.

## Abstract

Calegari, Elizeu; Da Silva Mello, Luiz (Advisor). **Implementation of the energy detector and performance analysis schemes in GNU Radio – simulations and tests**. Rio de Janeiro, 2018. 156p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Cognitive radio is a technology that aims to share the radio spectrum between licensed primary users and other secondary users in a harmonious way without causing interference that prevent the services provision and aiming at an improvement in the efficiency in the use of the radioelectric spectrum, which is a resource increasingly scarce. In the scope of the research related to this dissertation, a performance evaluation scheme of cognitive radio detectors is constructed and implemented in GNU Radio, the energy detector is constructed, and computational simulations and tests are implemented through two USRPs to evaluate the performance of the detector created, targeting the requirements of the IEEE 802.22 standard.

## Keywords

Cognitive Radio; Energy Detector; Spectrum Sensing; GNU Radio; USRP.

# Sumário

|  |    |
|--|----|
| 1 Introdução   | 16 |
| 2 Sensoriamento de Espectro                                  | 19 |
| 2.1. Surgimento do Rádio Cognitivo                           | 19 |
| 2.2. Buracos do espectro e Acesso Dinâmico do Espectro (DSA) | 19 |
| 2.2.1. Escassez do espectro                                  | 20 |
| 2.2.2. Acesso dinâmico do espectro                           | 20 |
| 2.3. Definições para o Rádio Cognitivo                       | 21 |
| 2.4. Padronização do Sensoriamento de Espectro               | 24 |
| 2.5. Padrão IEEE 802.22                                      | 25 |
| 3 Rádios Definidos por <i>Software</i> e o GNU Radio         | 27 |
| 3.1. Rádio definido por <i>Software</i> - SDR                | 27 |
| 3.1.1. Relacionamento entre SDR e o rádio cognitivo          | 28 |
| 3.1.2. A flexibilidade do SDR                                | 29 |
| 3.1.3. A arquitetura de um SDR real                          | 29 |
| 3.2. GNU Radio e USRP  | 30 |
| 3.2.1. O GNU Radio Companion - GRC                           | 31 |
| 3.2.2. Universal Software Radio Peripheral – USRP            | 33 |
| 4 Fundamentos Teóricos para a Detecção de Energia            | 39 |
| 4.1. A detecção de energia                                   | 39 |
| 4.2. Teste de Hipóteses                                      | 39 |
| 4.2.1. Teste de Bayes  | 42 |
| 4.2.2. Teste de Neyman-Pearson                               | 43 |
| 4.3. O detector de energia de Urkowitz                       | 46 |
| 4.4. Detector de energia – versão analógica vs. digital      | 49 |
| 4.5. A estatística da detecção de energia                    | 50 |
| 5 Implementação da Detecção de Energia                       | 56 |
| 5.1. Construção do detector de energia no GNU Radio          | 56 |

|  |     |
|--|-----|
| 5.2. Estrutura de avaliação de desempenho                          | 58  |
| 6 Simulações   | 62  |
| 6.1. Criando os <i>scripts</i> de Simulação                        | 62  |
| 6.2. Eclipse e PyDev   | 62  |
| 6.3. Simulações realizadas   | 64  |
| 7 Ensaios com as USRP  | 65  |
| 7.1. Montagem da configuração de Ensaio                            | 65  |
| 7.2. Geração e transmissão do sinal do usuário primário - PU       | 66  |
| 7.3. Recepção, detecção e análise de desempenho                    | 68  |
| 7.4. Ensaios realizados  | 71  |
| 8 Resultados   | 72  |
| 8.1. Simulações  | 72  |
| 8.1.1. $P_d$ x SNR   | 73  |
| 8.1.2. $P_d$ x $P_{fa}$  | 74  |
| 8.1.3. $P_d$ x N   | 75  |
| 8.2. Ensaios com as USRP   | 77  |
| 8.2.1. Ensaio via cabo coaxial e atenuador de 30 dB                | 78  |
| 8.2.2. Ensaio via antenas de 5 dBi e blindagem (gaiola de Faraday) | 82  |
| 8.2.3. Ensaio via antenas de 8 dBi                                 | 86  |
| 8.3. Análise dos resultados das simulações e ensaios               | 89  |
| 8.4. Incerteza de ruído e a barreira de SNR                        | 96  |
| 8.4.1. Simulações com a incerteza de ruído                         | 98  |
| 8.5. Parâmetros de projeto   | 100 |
| 8.5.1. Limiar  | 101 |
| 8.5.2. Número de amostras  | 101 |
| 9 Conclusões e Trabalhos Futuros                                   | 103 |
| 9.1. Conclusões  | 103 |
| 9.2. Trabalhos futuros   | 104 |
| 10 Glossário   | 105 |

|   |     |
|---|-----|
| 11 Referências bibliográficas                                   | 106 |
| A Instalação do GNU Radio e GNU Radio Companion                 | 110 |
| B Criação do bloco do detector de energia no GNU Radio          | 111 |
| B.1 Os módulos fora da árvore do GNU Radio – <i>OOT modules</i> | 111 |
| B.2 <i>gr_modtool</i> – A ferramenta de criação de um módulo    | 112 |
| B.3 CMake, make, etc.   | 115 |
| B.4 Criando o módulo OOT <i>myblocks</i>                        | 115 |
| B.5 Escrevendo o bloco <i>energy_detector_ff</i> em C ++        | 117 |
| B.6 Tipos de blocos quanto ao fluxo de dados                    | 122 |
| B.7 O <i>gr::sync_block</i>                                     | 123 |
| B.8 O código C++ do bloco do detector de energia                | 125 |
| B.9 A função <i>work()</i>                                      | 129 |
| B.10 O método <i>set_history()</i>                              | 131 |
| B.11 Implementando a detecção de energia                        | 131 |
| B.12 Compilação do bloco: usando <i>CMake</i>                   | 133 |
| B.13 Controle de qualidade: executando o <i>make test</i>       | 134 |
| B.14 Flexibilidade do <i>gr::block</i>                          | 135 |
| B.15 Tornando o bloco disponível no GRC                         | 136 |
| B.16 Métodos adicionais do <i>gr::block</i>                     | 139 |
| B.17 Tudo de uma só vez – referência de comandos                | 140 |
| C Criação da estrutura de análise de desempenho                 | 141 |
| D Automatização dos Fluxogramas de Simulação                    | 145 |
| E Automatização dos Fluxogramas de Ensaio                       | 153 |

## Lista de Figuras

|   |    |
|---|----|
| Figura 1 - Acesso dinâmico do espectro.....   | 21 |
| Figura 2 - Classificação das técnicas de sensoriamento de espectro .....                  | 23 |
| Figura 3 - Relação SDR vs. Rádio Cognitivo [19] .....                                     | 28 |
| Figura 4 – A clássica arquitetura básica de um SDR real .....                             | 30 |
| Figura 5 - Arquitetura típica de uma USRP [28] .....                                      | 33 |
| Figura 6 - Arquitetura típica da Placa-mãe de uma USRP [23] .....                         | 34 |
| Figura 7 - USRP N210 (Fonte: Ettus Research [25]) .....                                   | 36 |
| Figura 8 - Placa-mãe e placa-filha da USRP N210 Fonte: Ettus Research [25]) ..            | 37 |
| Figura 9 - Diagrama de interface da USRP N210 .....                                       | 37 |
| Figura 10 - Blocos do GNU Radio para uso da USRP .....                                    | 38 |
| Figura 11 - Regiões de decisão com mudança na variância [8] .....                         | 45 |
| Figura 12 - Detector de energia convencional: (I) analógico; (II) digital. ....           | 46 |
| Figura 13 - A detecção de energia no tempo e na frequência.....                           | 49 |
| Figura 14 - Cálculo da energia de N amostras no GNU Radio (N=1000).....                   | 57 |
| Figura 15 - Decisão comparando a energia calculada com um dado limiar.....                | 57 |
| Figura 16 - Bloco do detector de energia .....  | 57 |
| Figura 17 - Estrutura de análise de desempenho com estimativas de Pfa e Pd .....          | 59 |
| Figura 18 - Resultados para Pd e Pfa obtidos com a estrutura de análise .....             | 60 |
| Figura 19 - Visualização compacta de Pd e Pfa pela estrutura de análise.....              | 61 |
| Figura 20 - A plataforma <i>Eclipse IDE for C/C++ developers</i> .....                    | 63 |
| Figura 21 - Apresentação do ambiente de desenvolvimento do Eclipse .....                  | 63 |
| Figura 22 - Instalação do <i>plugin</i> Pydev no Eclipse para o python .....              | 64 |
| Figura 23 - Configuração típica de ensaio com USRP Tx e USRP Rx .....                     | 65 |
| Figura 24 - Esquema de conexão entre as USRP e a comunicação via LAN.....                 | 66 |
| Figura 25 - Fluxograma para gerar e transmitir o sinal primário .....                     | 67 |
| Figura 26 - Fluxograma em RX que envia a SNR requerida para o TX .....                    | 67 |
| Figura 27 - <i>Multiply Const</i> com fator de ajuste da energia do sinal no detector ... | 68 |
| Figura 28 - Fluxograma em RX com a estimativa da energia do sinal recebido...             | 69 |
| Figura 29 - Fluxograma em RX para análise de desempenho .....                             | 70 |
| Figura 30 - Exemplo de simulação no Eclipse para traçar Pd x Pfa.....                     | 72 |
| Figura 31 - Pd x SNR para N=100, 1000, 2500, 5000 e 10000.....                            | 73 |

|  |    |
|--|----|
| Figura 32 - Pfa x SNR para N=100, 1000, 2500, 5000 e 10000 .....               | 74 |
| Figura 33 - Pd x Pfa para N=100, 1000, 2500, 5000 e 10000 .....                | 74 |
| Figura 34 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000.....         | 75 |
| Figura 35 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 76 |
| Figura 36 - Pfa x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB.....    | 76 |
| Figura 37 - Montagens de ensaio com conexões entre as USRP.....                | 77 |
| Figura 38 - Pd x SNR com N=100, 1000, 2500, 5000 e 10000.....                  | 78 |
| Figura 39 - Pfa x SNR com N=100, 1000, 2500, 5000 e 10000 .....                | 78 |
| Figura 40 - Pd x Pfa com N=100, 1000, 2500, 5000 e 10000 .....                 | 79 |
| Figura 41 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000.....         | 80 |
| Figura 42 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 81 |
| Figura 43 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 81 |
| Figura 44 - Pd x SNR com N=100, 1000, 2500, 5000 e 10000.....                  | 82 |
| Figura 45 - Pfa x SNR com N=100, 1000, 2500, 5000 e 10000 .....                | 82 |
| Figura 46 - Pd x Pfa para N=100, 1000, 2500, 5000 e 10000 .....                | 83 |
| Figura 47 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000.....         | 84 |
| Figura 48 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 85 |
| Figura 49 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 85 |
| Figura 50 - Pd x SNR com N=100, 1000, 2500, 5000 e 10000.....                  | 86 |
| Figura 51 - Pfa x SNR com N=100, 1000, 2500, 5000 e 10000 .....                | 87 |
| Figura 52 - Pd x Pfa com N=100, 1000, 2500, 5000 e 10000 .....                 | 87 |
| Figura 53 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000.....         | 88 |
| Figura 54 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 89 |
| Figura 55 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB .....    | 89 |
| Figura 56 - Simulação Pd x Pfa com SNR=-15 dB e N=500, 1000 e 1500 .....       | 90 |
| Figura 57 - Pfa x Pfa nominal com SNR=-15 dB e N=500, 1000 e 1500 .....        | 91 |
| Figura 58 - Simulação Pd x Pfa para N=500 e SNR=-15, -12 e -10 dB.....         | 91 |
| Figura 59 - Pfa x Pfa nominal para N=500 e SNR=-15, -12 e -10 dB .....         | 92 |
| Figura 60 - Pd x SNR para N=20000, 50000, 100000 e 200000 (Pfa=0.1) .....      | 92 |
| Figura 61 - Pd x SNR com N=250000, 500000 com Pfa=0.01 e 0.1 .....             | 93 |
| Figura 62 - Pd x N para SNR=-21 dB e Pfa=0.1 e 0.01 .....                      | 94 |
| Figura 63 - Pd x Pfa para SNR = -21.0 dB e N= 200000 e 400000 .....            | 95 |
| Figura 64 - Pd x Pfa com N=500, SNR=-15 dB e fator de incerteza de ruído ..... | 99 |
| Figura 65 – Pfa x Pfa com N=500, SNR=-15 dB e fator de incerteza de ruído .... | 99 |



|  |     |
|--|-----|
| Figura 66 - gr_modtool: a ferramenta de criação de módulos e blocos .....                          | 113 |
| Figura 67 - Lista rápida de comandos para criar módulos e blocos .....                             | 114 |
| Figura 68 - Novo módulo e novo bloco na árvore do GNU Radio .....                                  | 114 |
| Figura 69 - SWIG cuida das interfaces (parâmetros) entre python e C++ .....                        | 116 |
| Figura 70 - <i>Script</i> de teste do bloco ( <i>qa_energy_detector_ff.py</i> ) .....              | 119 |
| Figura 71 - 1ª função de teste do bloco (test_001_t) .....   | 120 |
| Figura 72 - 2ª função de teste do bloco (test_002_t) .....   | 121 |
| Figura 73 - O bloco energy_detector_ff como um <i>sync_block</i> com histórico.....                | 125 |
| Figura 74 - Mudanças no arquivo de cabeçalho energy_detector_ff_impl.h .....                       | 127 |
| Figura 75 - energy_detector_ff_impl.cc - versão original do gr_modtool .....                       | 128 |
| Figura 76 - energy_detector_ff_impl.cc - versão modificada .....                                   | 128 |
| Figura 77 - A função <i>work()</i> - versão original criada pelo gr_modtool .....                  | 129 |
| Figura 78 - A função <i>work()</i> - versão modificada (1ª parte) .....                            | 129 |
| Figura 79 - A função <i>work()</i> - versão modificada (2ª parte) .....                            | 130 |
| Figura 80 - A função <i>work()</i> - versão modificada (3ª parte) .....                            | 132 |
| Figura 81 - Comandos típicos para compilação de um módulo .....                                    | 133 |
| Figura 82 - Testando o código do bloco com o <i>make test</i> .....                                | 134 |
| Figura 83 - Exemplo de erro ao testar o código de um bloco.....                                    | 135 |
| Figura 84 - Opção <i>makexml</i> para construir código xml de um bloco.....                        | 136 |
| Figura 85 - Arquivo XML gerado pelo gr_modtool - versão original.....                              | 137 |
| Figura 86 - Arquivo XML gerado pelo gr_modtool - versão modificada .....                           | 138 |
| Figura 87 - A função <i>forecast()</i> do <i>gr::block()</i> .....                                 | 139 |
| Figura 88 - Gerando amostras de sinal gaussiano com o <i>Noise Source</i> .....                    | 141 |
| Figura 89 - Detecção de energia com os blocos nativos do GNU Radio .....                           | 141 |
| Figura 90 - Energia do sinal primário com o ruído, limiar e decisões .....                         | 142 |
| Figura 91 - Estrutura para visualizar apenas o ruído e o sinal+ruído .....                         | 142 |
| Figura 92 - Esquema para contagem de ocorrências visando Pfa .....                                 | 143 |
| Figura 93 - Esquema para contagem de ocorrências visando Pd .....                                  | 144 |
| Figura 94 - O botão "Generate the Flowgraph" no GRC.....   | 145 |
| Figura 95 - Fluxograma do GRC gera um arquivo .grc em linguagem python...                          | 146 |
| Figura 96 - <i>Script</i> .py gerado pelo GRC a partir do fluxograma .grc .....                    | 147 |
| Figura 97 - A função <i>main()</i> no <i>script</i> .py gerado pelo fluxograma.....                | 147 |
| Figura 98 - Inclusão dos parâmetros Pfa, SNR_dB e N na inicialização .....                         | 148 |
| Figura 99 - Funções <i>Pd_teorica()</i> e <i>simular_Pd()</i> e a função <i>MilliSleep()</i> ..... | 149 |

|   |     |
|---|-----|
| Figura 100 - <i>Script</i> que automatiza o varrimento dos cenários (Pfa, SNR, N) ... | 150 |
| Figura 101 - Janela console exibindo o Log com os resultados da simulação ....        | 151 |
| Figura 102 - <i>Script</i> de ensaio em RX para automatizar a coleta de Pd e Pf ..... | 153 |
| Figura 103 - Fluxograma envia SNR requerida a TX com correção de ganho ...            | 154 |

## Lista de Tabelas

|   |    |
|---|----|
| Tabela 1 - Requisitos de sensoriamento do Padrão IEEE 802.22 [9]..... | 26 |
| Tabela 2 - Largura de banda da USRP [28].....                         | 35 |
| Tabela 3 - Características da USRP N210.....                          | 36 |
| Tabela 4 - Fator de ajuste e Ganho das USRPs para equivaler SNR.....  | 71 |

# 1

## Introdução

O objetivo principal deste trabalho é o estudo do algoritmo de detecção de energia e a implementação de um detector de energia realizável no GNU Radio, assim como um esquema de análise de desempenho. Essa proposta envolve, desde uma análise teórica do modelo e algoritmo que vêm sendo estudados e apresentados por diversos autores, como também a implementação desse algoritmo e realização de simulações e ensaios em plataformas de estudo e desenvolvimento, como é o caso do GNU Radio e a USRP.

Sendo assim, inicialmente no capítulo 2 será feita uma abordagem dos conceitos básicos de sensoriamento de espectro em função das necessidades de compartilhamento do espectro radioelétrico, uma vez que é um recurso escasso e predominante para o desenvolvimento de um país. Será abordado também o conceito dos *white spaces*, ou buracos de espectro, e a alocação dinâmica do espectro como forma de possibilitar seu compartilhamento. Os conceitos e as definições de rádio cognitivo também serão abordados, desde o conceito inicial de Mitola, o criador da ideia de rádio cognitivo até às contribuições de Haykin visando o aspecto prático do sensoriamento espectral como abordagem prática primordial na implementação da tecnologia. Além disso, serão citados também os esforços de padronização e a definição pelo IEEE do padrão 802.22, assim como seus requisitos de desempenho que norteiam as pesquisas sobre desenvolvimento de técnicas e algoritmos de detecção em estudo atualmente.

No capítulo 3, o conceito de rádio definido por *software* ou SDR, uma espécie de precursor do rádio cognitivo, é abordado apresentando o entendimento de diversos autores no sentido das mudanças necessárias para seu aprimoramento visando a implementação do rádio cognitivo. Serão apresentados também a plataforma GNU Radio e o *kit* de desenvolvimento SDR criado por Matt Ettus, a USRP, utilizados no âmbito deste trabalho para criação do detector de energia, implementação de simulações e ensaios em ambiente real através da USRP.

A fundamentação teórica utilizada na implementação do algoritmo para a detecção de energia é abordada no capítulo 4, começando por apresentar o trabalho de Urkowitz [4] que foi o primeiro a apresentar um trabalho de referência na área em 1967 e seguindo com os conceitos de detecção de energia, tanto na forma analógica como digital. Em seguida é apresentada a teoria estatística de decisão subjacente, desde o teste de hipóteses, passando pelas regras de decisão de Bayes e Neyman-Pearson até se chegar às expressões teóricas de desempenho que servem de referência para este trabalho, como as probabilidades de detecção e falso alarme, a definição do limiar e o número de amostras necessário para atingir o desempenho pretendido.

No capítulo 5 temos a descrição das tecnologias e todos os passos necessários para a criação do detector de energia e a implementação das estruturas necessárias para obter as métricas de desempenho. O GNU Radio é apresentado como plataforma de desenvolvimento para a criação do detector de energia e das estruturas de avaliação do detector criado, sendo descrito passo a passo todos os procedimentos necessários para a criação do bloco do detector de energia no GNU Radio e também para a criação da estrutura de análise.

A construção e implementação dos *scripts* de simulação para comparar os resultados simulados com os modelos teóricos são detalhadas no capítulo 6, passando também por um detalhamento maior das funcionalidades das plataformas GNU Radio e Eclipse para a criação dos fluxogramas necessários e os *scripts* de automatização para coleta dos resultados das simulações.

O GNU Radio é uma plataforma que já vem ganhando espaço e sendo bastante utilizada como ferramenta de desenvolvimento de soluções e aplicações no estudo de protótipos de produtos de telecomunicações por uma grande parte da comunidade acadêmica e científica. Apresenta não apenas versatilidade no uso dos diversos componentes pré-desenvolvidos (blocos) que permitem encapsular a complexidade de determinadas aplicações permitindo um foco maior na aplicação em estudo, como também permite testar o comportamento das soluções desenvolvidas em situações reais de meio físico através de equipamentos de desenvolvimento, como por exemplo a USRP. A USRP é um rádio flexível de custo acessível que transforma um PC comum em uma poderosa plataforma de prototipagem de sistemas sem fio.

No capítulo 7 são feitas as adaptações necessárias nos *scripts* de simulações para convertê-los em *scripts* de ensaio, e dessa forma, utilizar as estruturas de avaliação de desempenho em ensaios com duas unidades USRP, sendo uma para transmitir o sinal de usuário primário e a outra para receber e passar as amostras para o processo de detecção que é implementado em um computador pessoal. Em seguida os ensaios com as USRP são realizados.

O resultados obtidos tanto das simulações como dos ensaios são apresentados no capítulo 8, permitindo a sua comparação com as curvas teóricas baseadas nos modelos apresentados na fundamentação teórica para a aproximação gaussiana.

Ainda no âmbito do capítulo 8 é apresentado o conceito de incerteza na determinação do ruído com um fator crítico para o desempenho do detector de energia, é apresentado um modelo para estudar esse impacto e são implementadas simulações comparando os resultados obtidos com as previsões dos modelo.

No capítulo 9 são apresentadas as conclusões e uma previsão de trabalhos futuros visando o aproveitamento do trabalho desenvolvido e sua ampliação no estudo de outros algoritmos de detecção.

## 2

### Sensoriamento de Espectro

#### 2.1. Surgimento do Rádio Cognitivo

A ideia quanto ao desenvolvimento da tecnologia do Rádio Cognitivo surgiu a partir do trabalho do Dr. Joseph Mitola, que apresentou esse conceito no *Royal Institute of Technology* em 1998, e posteriormente desenvolveu a base da teoria do rádio cognitivo na sua tese de Doutorado.

Na visão de Mitola [1], um fator motivador para o desenvolvimento dessa tecnologia era a busca de uma solução para o problema da escassez cada vez maior do espectro de rádio disponível, e de acordo com sua tese e através do Rádio Cognitivo, as redes computacionais teriam inteligência suficiente para avaliar os recursos de rádio para efetuarem suas comunicações sem fio em função do contexto do uso do espectro. Mitola afirma também em sua tese que o termo “rádio cognitivo” surgiu de suas observações sobre abordagens inovadoras para a gestão do espectro de rádio para o FCC em abril de 1999, nas quais ele introduziu o termo de rádio cognitivo para a mídia. Posteriormente, o termo “rádio cognitivo” tem sido utilizado na comunidade de gerenciamento de espectro como uma espécie de “rádio inteligente”.

#### 2.2. Buracos do espectro e Acesso Dinâmico do Espectro (DSA)

As definições para os espaços brancos (*white spaces*) ou buracos de espectro e as técnicas de compartilhamento do espectro visando melhorar a eficiência do seu uso, e em particular o acesso dinâmico do espectro se originaram em função dos problemas advindos da escassez do espectro.

### 2.2.1. Escassez do espectro

De fato, em Novembro de 2002, o FCC publicou um relatório elaborado pela *Força-tarefa para a Política do Espectro* [24], composta de economistas, engenheiros e juristas, no qual é revelado o contexto de escassez do espectro:

*"Em muitas bandas, o acesso ao espectro é um problema mais significativo do que a escassez física de espectro, em grande parte devido ao legado da regulamentação comando-e-controle que limita a possibilidade de os utilizadores do espectro potenciais para obter tal acesso."*

(FCC, 2002)

Além disso, de acordo com Haykin [2], se analisarmos porções do espectro de rádio, incluindo o cenário típico em áreas urbanas, veríamos que:

1. *Algumas bandas de frequência do espectro são em grande parte desocupadas a maior parte do tempo;*
2. *algumas outras faixas de frequência são apenas parcialmente ocupadas;*
3. *as demais bandas de frequência são muito utilizadas.*

Ainda segundo Haykin, a subutilização do espectro eletromagnético nos leva a pensar em termos de “buracos de espectro”, termo para o qual foi oferecida a seguinte definição:

*“Um buraco de espectro é uma banda de frequências atribuídas a um usuário primário, mas, em um determinado tempo e localização geográfica específica, a banda não está sendo utilizado por aquele usuário.”*

(HAYKIN, 2005)

### 2.2.2. Acesso dinâmico do espectro

Sendo assim, a ideia defendida por MITOLA e posteriormente por Haykin, era de que a eficiência na utilização do espectro pode ser melhorada significativamente tornando possível a um utilizador secundário (que não está podendo ser servido por problemas de escassez de espectro disponível no momento)



acessar um buraco de espectro desocupado pelo usuário principal no local certo e no momento em questão.

Para permitir isso, o rádio cognitivo, inclusive o rádio definido por *software*, tem sido proposto como um meio para promover a utilização eficiente do espectro através da exploração da existência de buracos de espectro, como no exemplo do esquema de acesso dinâmico do espectro da Figura 1.

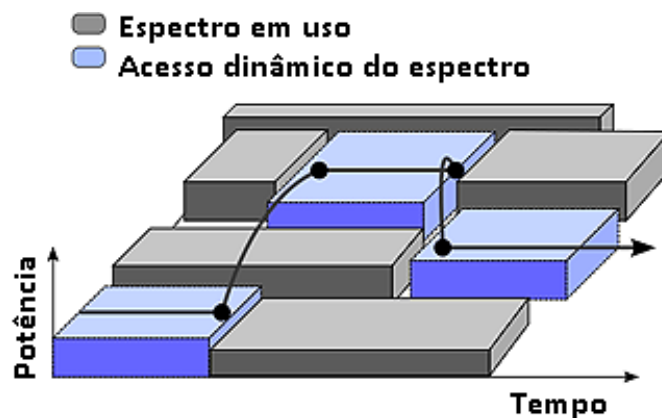


Figura 1 - Acesso dinâmico do espectro

### 2.3. Definições para o Rádio Cognitivo

Em seu livro, Fette [20] conduz o leitor através das tecnologias e considerações regulatórias que suportam as três principais aplicações que aumentam os recursos de um SDR (rádio definido por *software*) de forma a torná-lo em um rádio cognitivo:

- *gestão do espectro e otimizações;*
- *interface com uma ampla variedade de redes e otimização dos recursos da rede; e*
- *fazer a interface e prover recursos eletromagnéticos para auxiliar o ser humano em suas atividades.*

Também em seu artigo para o *IEEE Journal* de 2005, Haykin [2], propõe uma definição interessante para o uso do termo “rádio cognitivo”. Ele diz que para compreender esse termo é necessário atentar para o significado do termo

relacionado "cognição", e que, computacionalmente falando, temos três pontos de vista para o termo cognição:

- *Os estados mentais e processos que intervêm entre estímulos à entrada e respostas à saída.*
- *Os estados mentais e processos são descritos por algoritmos.*
- *Os estados mentais e processos prestam-se em si a investigações científicas.*

Com base nessas definições e na inferência que fez dos estudos sobre a cognição de Pfeifer e Scheier [3], os quais exploram os princípios gerais de inteligência através de uma metodologia sintética de *aprendizagem pelo entendimento*, Haykin [2] fornece a seguinte definição para o termo “rádio cognitivo”:

*“Rádio cognitivo é um sistema de comunicação sem fio inteligente que tem conhecimento do seu ambiente circundante (isto é, mundo exterior), e utiliza o método de compreensão-por-construção para aprender com o ambiente e adaptar seus estados internos às variações estatísticas dos estímulos de RF à entrada, fazendo modificações correspondentes nos seus parâmetros de funcionamento (por exemplo, potência de transmissão, frequência da portadora e estratégia de modulação) em tempo real, com dois objetivos principais em mente:*

- *comunicação altamente confiável quando e onde necessário;*
- *utilização eficiente do espectro de rádio.”*

(HAYKIN, 2005)

Ainda sobre a definição de rádio cognitivo, ARSLAN e CELEBI afirmam em seu livro [19] que uma das definições mais populares do rádio cognitivo é a de que:

*“O rádio cognitivo é um SDR que está consciente do seu ambiente, estado interno e localização, e autonomamente ajusta suas operações para alcançar objetivos designados.”*

(ARSLAN e CELEBI, 2007)

Ainda segundo Haykin [2], seis palavras-chave se destacam nessa definição: **consciência, inteligência, aprendizagem, adaptabilidade, confiabilidade e eficiência**. Ainda segundo ele, a realização desta combinação de capacidades de alcance tão grande só é de fato possível hoje, graças aos avanços espetaculares em processamento digital de sinal, o aprendizado cooperativo em redes (*networking*), aprendizagem das máquinas, avanços no desenvolvimento do *software* e do *hardware* de computador.

Um grande desafio na rádio cognitivo é que os usuários secundários precisam detectar a presença de usuários primários em um espectro licenciado e deixar a banda de frequência o mais rapidamente possível se o usuário primário correspondente surgir a fim de evitar a interferência com os usuários principais licenciados.

Esta técnica é chamada de sensoriamento de espectro. O sensoriamento de espectro e a estimação é o primeiro passo para desenvolver sistemas de rádio cognitivo. O esquema na Figura 2 mostra a classificação pormenorizada das técnicas de sensoriamento de espectro geralmente apresentadas por diversos autores do tema. Elas são amplamente classificadas em três tipos principais, detecção primária pelo transmissor ou não-cooperativa, detecção cooperativa e detecção baseada em sensoriamento de interferência.



Figura 2 - Classificação das técnicas de sensoriamento de espectro

Portanto, um dos principais interesses de pesquisa em rádio cognitivo é a técnica de detecção de espectro, e existem três técnicas principais para detecção de transmissão única. O mais simples deles é o detector de energia [5], que permite a

determinação da presença de um sinal em uma dada largura de banda ou janela de tempo delimitada. Uma segunda técnica usa filtragem casada e se baseia no conhecimento do sinal original transmitido. Finalmente, a detecção ciclo estacionária tem a capacidade de detectar a presença do sinal a partir do conhecimento da periodicidade dos sinais transmitidos.

Essas técnicas podem ser usadas em um ambiente colaborativo, expandindo o rádio cognitivo único em uma rede de rádio cognitiva (CRN). Por exemplo, algumas arquiteturas de redes de rádio cognitivas podem usar o detector de energia em um cenário colaborativo com uma abordagem centralizada. Os usuários móveis podem relatar informações relevantes ao sistema, como propriedades de ruído e áreas em que o usuário principal não tem cobertura, para que o núcleo CRN possa manter um banco de dados da cobertura efetiva do usuário principal sobre sua área de serviço e pontos onde o uso oportunista do espectro é possível.

Portanto, dependendo do conjunto de critérios tidos em conta ao decidir sobre as mudanças na transmissão e recepção, existem dois tipos principais de rádio cognitivo [12]:

- Rádio cognitivo total (de Mitola) – em que todos os parâmetros possíveis observáveis por um usuário secundário são levados em conta;
- Rádio cognitivo de sensoriamento de espectro (de Haykin) – em que apenas a forma de lidar com o espectro de radiofrequência é considerado.

Não se espera que o rádio de Mitola seja completamente implementado senão em um prazo de alguns anos, até que todo o *hardware* do SDR tenha se tornado disponível em um tamanho adequado.

O trabalho aqui apresentado é de configuração do SDR como um rádio cognitivo de sensoriamento de espectro, portanto, mais próximo do conceito de rádio cognitivo de HAYKIN.

## 2.4. Padronização do Sensoriamento de Espectro

No que diz respeito à padronização e aos requisitos do padrão IEEE 802.22 [6], diferentes emissoras de TV usam bandas do espectro radioelétrico para

transmissão de TV (por exemplo, 54-806 MHz nos EUA). Os *white spaces*, ou buracos de espectro (ou seja, slots de frequência não utilizados por transmissores de TV), podem incluir bandas de proteção, frequências gratuitas para TV analógica para conversão de TV digital (por exemplo, 698-806 MHz nos EUA) e bandas de TV gratuitas criadas quando o tráfego na TV digital é baixo e pode ser comprimido em bandas menores de TV.

A agência FCC nos EUA permitiu o uso *white spaces* por usuários sem licença. Posteriormente, seguindo os esforços de padronização, as seguintes especificações se materializaram [6]:

- O padrão IEEE 802.22 para espaços brancos de TV é lançado com especificações de camada física para WRAN.
- O ECMA 392 inclui especificações para dispositivos sem fio e portáteis operando em bandas de TV.
- O IEEE SCC41 desenvolve padrões de suporte para rádio e gestão dinâmica do espectro.
- O IEEE 802.11af é para implementar a tecnologia Wi-Fi nos *white spaces* de TV usando a tecnologia de rádio cognitivo.

## 2.5. Padrão IEEE 802.22

Entre os esforços dos padrões [6] acima, o IEEE 802.22 traz acesso de banda larga não apenas para dispositivos Wi-Fi, mas também para redes móveis gerais (por exemplo, micro, pico ou femto-células), permitindo o uso da técnica de rádio cognitivo sobre uma base não-interferente.

Como o IEEE 802.22 WRAN não prescreve uma técnica de detecção de espectro específica, os projetistas são livres para selecionar qualquer técnica de detecção, entretanto, a detecção de energia tem sido uma das escolhas mais óbvias em função da facilidade de implementação e também pelo fato de ser uma detecção cega, ou seja, que não requer um conhecimento prévio das características do sinal do usuário primário licenciado.

Na implementação da detecção de energia, as especificações que devem ser consideradas cuidadosamente são dadas abaixo (Tabela 1):

- O WRAN IEEE 802.22 limita as probabilidades de falso alarme (o que indica o nível de buracos de espectro não detectados) e de perda de detecção (o que indica a nível de interferência inesperada para usuários primários) em 10%.
- Enquanto as probabilidades de falso alarme e de perda de detecção refletem a eficiência geral e confiabilidade da rede cognitiva, o requisito de 10% deve ser cumprido mesmo em condições de SNR muito baixas, como -20 dB, e com potência de sinal de -116 dBm e um patamar de ruído de -96 dBm.
- Enquanto o detector de energia funciona bem em SNRs elevadas e moderadas, ele tem um desempenho ruim em uma SNR baixa. Embora o aumento do tempo de detecção seja uma opção válida para melhorar o desempenho de detecção, o IEEE 802.22 limita a detecção em 2 segundos, que inclui tempo de detecção e posterior tempo de processamento. Este limite de tempo máximo é crítico no sensoriamento de espectro para baixas relações sinal-ruído.

Tabela 1 - Requisitos de sensoriamento do Padrão IEEE 802.22 [9]

| <i>Parâmetro</i>                 | <b>TV Digital</b>         | <b>Microfone sem fio<br/>(Parte 74)</b> |
|----------------------------------|---------------------------|---|
| Tempo de detecção                | ≤ 2 seg.                  | ≤ 2 seg.                                |
| Tempo para abandonar o canal     | 2 seg.                    | 2 seg.                                  |
| Limiar (sensibilidade requerida) | -116 dBm<br>(sobre 6 MHz) | -107 dBm<br>(sobre 200 KHz)             |
| Probabilidade de detecção        | 0.9                       | 0.9                                     |
| Probabilidade de falso alarme    | 0.1                       | 0.1                                     |
| SNR                              | -21 dB                    | -12 dB                                  |

Sendo assim, os parâmetros do detector de energia devem ser projetados cuidadosamente com base nas especificações do sensoriamento de espectro.

### 3

## Rádios Definidos por *Software* e o GNU Radio

### 3.1.

#### Rádio definido por *Software* - SDR

Até o desenvolvimento da tecnologia do rádio cognitivo, o SDR foi proposto principalmente para realizar dispositivos sem fio multi-modo e multi-padrão, entretanto em rádios cognitivos, o SDR exerce uma função essencial, que é a **reconfigurabilidade**.

A reconfigurabilidade [2] consiste na capacidade de um rádio de ajustar seus parâmetros de operação (frequência da portadora, potência transmitida, estratégia de modulação, largura de banda, etc), portanto, sempre que um rádio cognitivo necessita de reconfigurar seus parâmetros de operação, ele naturalmente utiliza um SDR para executar esta tarefa. Para outras tarefas do tipo cognitivo, o rádio cognitivo olha para o processamento de sinal e para procedimentos de aprendizagem.

Portanto, aliando a reconfigurabilidade do SDR com o processo de cognição, o rádio cognitivo pode obter funcionalidades consideradas praticamente impossíveis a apenas alguns anos atrás [9].

Em resumo, para compreender os princípios da tecnologia do rádio cognitivo é necessário ter uma compreensão da capacidade e aplicabilidade da tecnologia SDR.

De acordo com a definição de FETTE [20],

*“Um SDR é um rádio no qual as propriedades de frequência de portadora, largura de banda do sinal, modulação, e o acesso à rede são definidas por software. “*

(FETTE, 2006)

Além disso, o SDR moderno de hoje também implementa qualquer criptografia necessária; codificação de correção de erro (FEC); e codificação de fonte de voz, vídeo ou dados em *software* também.

### 3.1.1. Relacionamento entre SDR e o rádio cognitivo

O SDR pode proporcionar uma funcionalidade de rádio muito flexível [19], evitando o uso de circuitos e componentes analógicos fixos específicos das aplicações. Em resumo, o rádio cognitivo precisa ser projetado em torno de SDR, ou seja, o SDR é a tecnologia núcleo para permitir o rádio cognitivo.

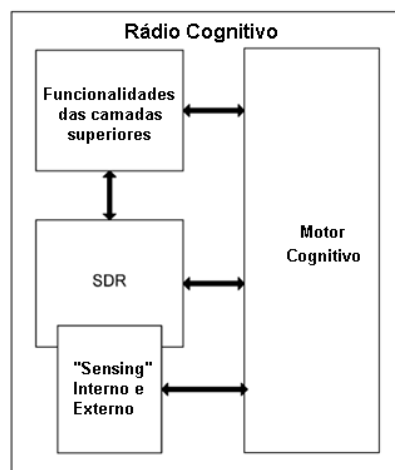


Figura 3 - Relação SDR vs. Rádio Cognitivo [19]

Neste simples modelo [19] apresentado na Figura 3, o rádio cognitivo é montado em torno SDR. Este modelo se adapta bem à definição acima mencionada de rádio cognitivo, onde a combinação de motor cognitivo, SDR e as outras funcionalidades de apoio (por exemplo, detecção) resultam no rádio cognitivo.

O motor cognitivo é responsável por otimizar e controlar o funcionamento do SDR baseado em alguns parâmetros de entrada, como base no que se sentiu ou aprendeu sobre o ambiente de rádio, contexto de utilizador, e condições da rede. O motor cognitivo está ciente das capacidades e dos recursos de *hardware* do rádio, bem como dos outros parâmetros de entrada, portanto, tenta satisfazer os requisitos de uma aplicação de camada superior (de enlace ou ligação) referente ao *link* de rádio com os recursos disponíveis como espectro e potência, em contraste com o



clássico rádio simples de *hardware*, no qual o rádio realiza apenas uma única ou muito limitada funcionalidade de rádio.

### 3.1.2.

#### A flexibilidade do SDR

A tecnologia SDR representa uma plataforma de rádio genérico muito flexível, e que é capaz de operar com muitas larguras de banda diferentes numa ampla gama de frequências e utilizando diversas modulações e formatos de onda, e como resultado, o SDR pode suportar múltiplos padrões (isto é, GSM, EDGE, WCDMA, CDMA2000, Wi-Fi, WiMAX) e tecnologias de acesso múltiplo, tais como Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA), Orthogonal Frequency Division Multiple Access (OFDMA) e Space Division Multiple Access (SDMA).

Basicamente, um rádio definido por *software* (*Software Defined Radio* – *SDR*) consiste em um rádio em que parte de suas funções de processamento do sinal (que historicamente são feitas em formato analógico, via *hardware* específico) são implementadas na forma digital por meio de um *software* em um computador, em vez de utilizar os dispositivos de *hardware* clássicos (*mixer* ou multiplicador, filtros, modulador/demodulador, etc).

### 3.1.3.

#### A arquitetura de um SDR real

A arquitetura de um rádio SDR real, conforme apresentada na Figura 4, pode ser mais facilmente compreendida se a dividirmos em duas partes (ou componentes) principais: o *front-end* e o *back-end*.

O *front-end* é normalmente responsável por cuidar do recebimento e transmissão das frequências de rádio (antena, filtragem e amplificação do sinal), enquanto o *back-end* cuida do processamento do sinal de rádio de forma adequada (conversão de frequência, modulação/demodulação da portadora em função do sinal em banda base, etc).

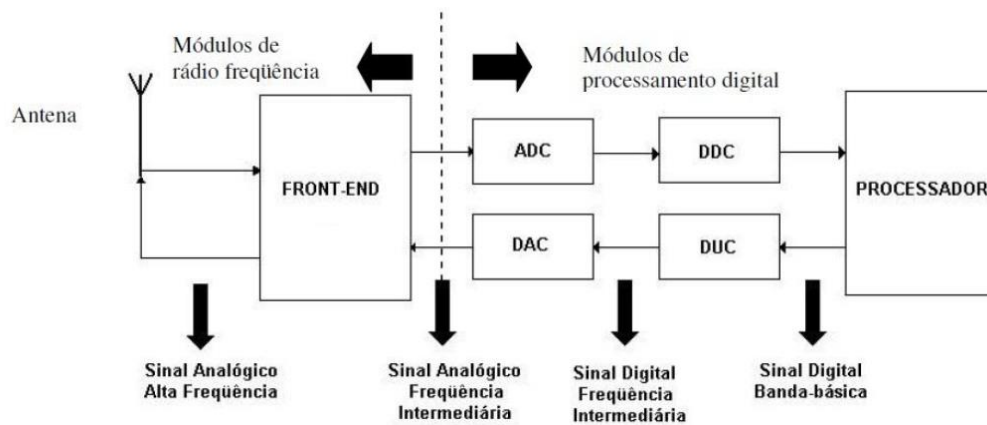


Figura 4 – A clássica arquitetura básica de um SDR real

O grande diferencial do SDR em relação ao rádio convencional é que os componentes dos módulos de processamento digital são implementados via *software*, ou seja, na tecnologia SDR o *hardware* compõe o *front end* e o *software* domina o processamento no *back end*.

Nesta concepção de SDR real, o *front end* é representado por um módulo no qual se prepara o sinal para a conversão AD (analógico/digital) e para o processamento do sinal e/ou prepara o sinal para transmissão após a conversão DA (digital/analógico). Esta preparação é feita através de uma amplificação do sinal, controle de ganho, deslocamento para uma frequência intermédia (no caso de recepção) ou deslocamento para frequência original do sinal (no caso de transmissão), e uma filtragem anti-*aliasing*.

Além de resolver o problema da tecnologia empregada nos conversores A/D e D/A, o *front end* pode ser projetado para otimizar o custo do projeto de um SDR, sendo possível ajustar o SDR para o uso de DAC's e ADC's de baixo custo.

### 3.2. GNU Radio e USRP

A plataforma SDR utilizada nesta pesquisa foi o GNU Radio. É um *software* gratuito e de código aberto com um *kit* de ferramentas de desenvolvimento para implementar rádios de *software* e pode ser usado para dar suporte tanto a pesquisas de comunicações sem fio como a sistemas de rádio no mundo real.

O GNU Radio pode ser usado para escrever aplicativos para processar, receber e transmitir dados em formato digital e dispõe de uma vasta coleção de

bibliotecas contendo blocos que implementam tarefas típicas de processamento de sinal, tais como filtros, codificadores e decodificadores de canal, elementos de sincronização, equalizadores, demoduladores, vocoders, decodificadores e muitos outros.

Além dos blocos disponibilizados em uma árvore nativa (estrutura de seleção de blocos que é disponibilizada no seu *software* de edição, o GNU Radio Companion - GRC) o GNU Radio fornece também recursos para criação de blocos definidos pelo usuário, caso seja necessário um bloco para uma aplicação específica. Neste trabalho mesmo, é criado um bloco para o detector de energia e todo o processo de criação será abordado passo a passo.

### 3.2.1. O GNU Radio Companion - GRC

O GNU Radio é um *kit* de ferramentas [27] de desenvolvimento de software gratuito e de código aberto que fornece blocos de processamento de sinal para implementar rádios de software. Ele pode ser usado com conjunto com hardware de RF externo de baixo custo para criar rádios definidos por software (SDRs) ou apenas em um ambiente de simulação, sem hardware.

Além de sua estrutura nativa de processamento para executar fluxogramas, criação da estrutura de conexão entre os blocos e o respectivo processamento *runtime* por meio de *threads* disparadas a partir de seu scheduler, possui uma vasta coleção de bibliotecas que pode ser acessada através seu software de interface para criação e edição de fluxogramas, o GNU Radio Companion (ou GRC).

Fluxogramas são scripts em python baseados na importação das bibliotecas e blocos necessários. *Threads* são processos que cumprem tarefas específicas sendo executados em paralelo, e python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, e orientada a objetos.

O GRC é uma interface gráfica para ambiente de desenvolvimento (IDE) que facilita muito a criação de fluxogramas (arquivos em python com extensão .grc) com a estrutura de processamento que se deseja implementar. O fluxograma é criado conectando os blocos já presentes nas bibliotecas do GNU Radio ou blocos criados pelo usuário. Após a sua criação, um fluxograma pode ser executado, ou podemos apenas utilizar a funcionalidade de gerar o arquivo python .grc

correspondente, e inclusive visualizá-lo ou editá-lo por meio de um editor de texto convencional.

A geração do *script* em *python* referente ao fluxograma e sua possível customização constitui um recurso valioso para expandir as funcionalidades do GNU Radio, por exemplo, podemos otimizar o processamento do código *python* gerado e modificar o *script* para criar diversas instâncias de um fluxograma sendo executadas em paralelo, que é o recurso que foi empregado neste trabalho para implementar as simulações e ensaios e que será abordado com mais detalhes na geração dos scripts de simulação e de ensaio.

A linguagem de programação *python* é usada principalmente para escrever os fluxogramas do GNU Radio, enquanto os blocos de processamento em geral são desenvolvidos em C ++, por questões de desempenho. Os blocos também podem ser desenvolvidos em *python*, mas o código terá sua rapidez de processamento prejudicada, uma vez que o *python* é uma linguagem interpretada, ao passo que o C++ é uma linguagem em que o código é compilado.

Na seção de *Frequently Asked Questions - FAQ* da página Wiki do GNU Radio [27], é citado que o GNU Radio usa o Python como uma linguagem de *script*, não (em geral) para processamento de sinal em tempo de execução. Todos os blocos do GNU Radio e o *scheduler* são escritos em C ++. Nós exportamos a interface para o Python para nos permitir usar isso como uma linguagem de *script* para facilitar a utilização de *flowgraphs* juntos. Em tempo de execução, o Python em geral deve “sair de caminho”, a menos que se tenha programado algo específico em Python, o que é comum para interfaces de usuário, controles e interação, que não estão no caminho principal de *runtime*, de qualquer maneira.

Uma questão que fica no ar é que, se os fluxogramas são gerados em *python* e os blocos geralmente são gerados em C++, como é que os códigos dessas duas linguagem “conversam” (ou seja, passam mensagens com dados e parâmetros) entre si? Para isso, é utilizado uma ferramenta denominada de SWIG, que é quem cuida de fazer interface com os aplicativos (fluxogramas) *front-end* do *Python* e as rotinas dos blocos escritas em C ++.

### 3.2.2. Universal Software Radio Peripheral – USRP

A USRP é um equipamento usado pela plataforma GNU Radio para acessar o espectro radioelétrico. É uma plataforma SDR projetada e vendida atualmente pela National Instruments e pela Ettus Research, e destina-se a servir como plataforma de *hardware* relativamente barata para utilizar como SDR.

Embora algumas características e especificações variem de modelo para modelo [28], todos os dispositivos USRP usam a mesma arquitetura geral. Em muitos casos, o *front-end* RF, os *mixers*, filtros, osciladores e amplificadores são necessários para traduzir um sinal do domínio RF e os sinais complexos de banda base ou IF.

A banda de base dos sinais IF é amostrada por ADCs e as amostras digitais são sincronizadas em um FPGA. A imagem da FPGA em operação fornece funcionalidade de *down-conversion* digital, que inclui ajuste de frequência fina e vários filtros para decimação. Após a decimação, amostras brutas ou outros dados são transmitidos para um computador (*host*) por meio da interface respectiva. O processo inverso aplica-se à cadeia de transmissão.

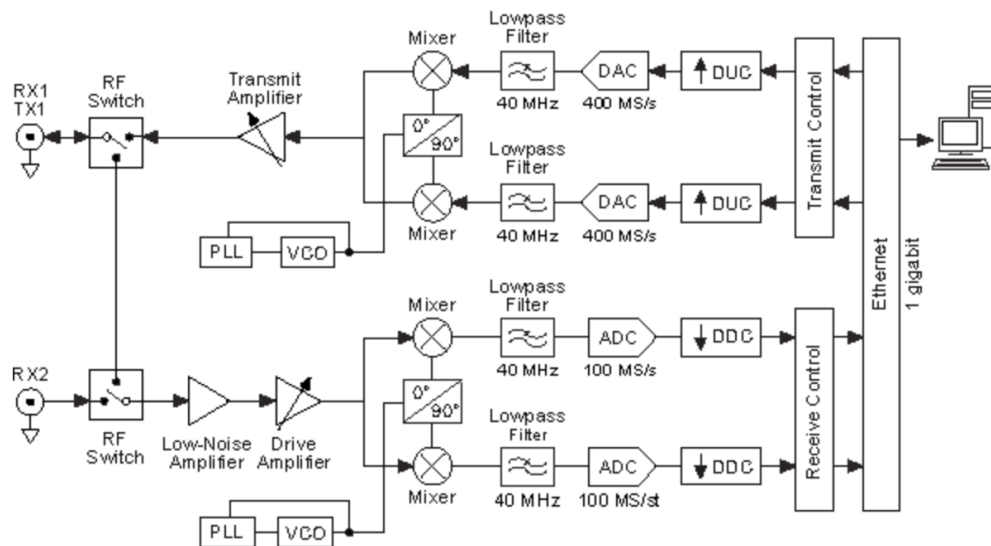


Figura 5 - Arquitetura típica de uma USRP [28]

A placa-mãe (Figura 6) tem quatro conversores analógicos-digitais (ADC) de 14 bits a 100 Ms/s e quatro conversores digitais-analógicos de 16 bits a 400 Ms/s.

Além disso, possui também quatro conversores digitais descendentes (DDC) e dois conversores digitais ascendentes (DUC) com taxas de interpolação programáveis.

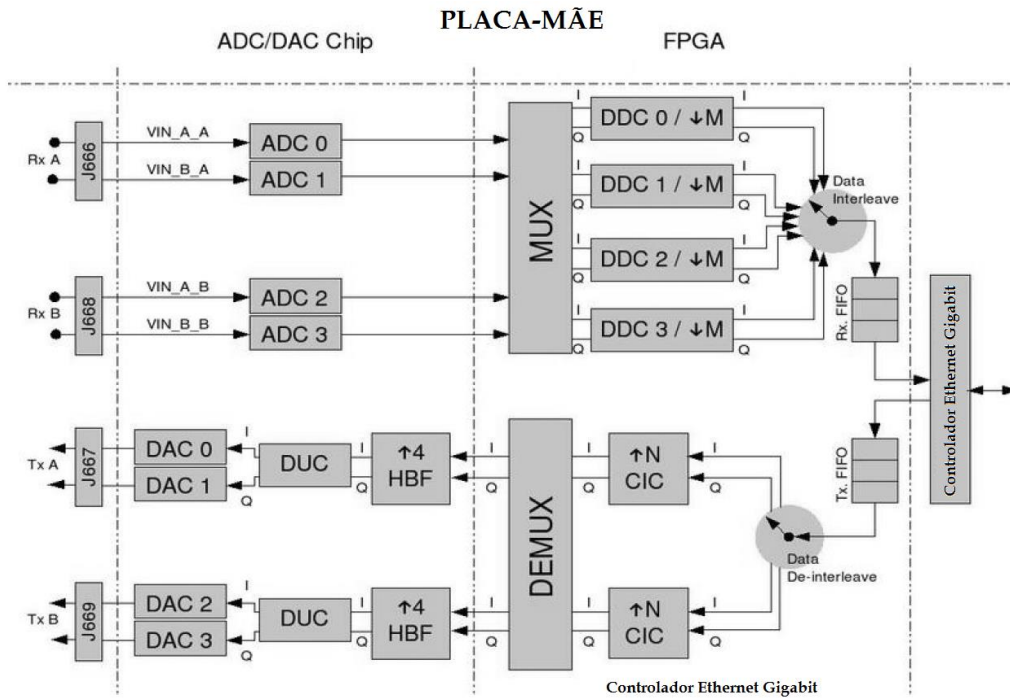


Figura 6 - Arquitetura típica da Placa-mãe de uma USRP [23]

Na USRP, o processamento de alta taxa de amostragem ocorre na FPGA, enquanto o processamento da taxa de amostragem mais baixa ocorre no computador. Os dois DDCs (*digital down-converters*) misturam, filtram e decimam os sinais de entrada (a até  $400 \times 10^6$  amostras/s) para a FPGA. Já os dois DUCs interpolam os sinais de banda base a até  $100 \times 10^6$  amostras/s antes de convertê-los na frequência de saída selecionada.

A largura de banda do dispositivo USRP [28] varia em cada ponto da cadeia de sinal. Três tipos gerais de especificações de largura de banda são a largura de banda analógica, a largura de banda de processamento do FPGA e a largura de banda do host. A largura de banda do sistema é geralmente o mínimo da placa-filha RF, processamento de FPGA e largura de banda do *host*. Também deve ser tomado cuidado para evitar uma largura de banda analógica maior que a taxa de amostragem ADC/DAC de qualquer dispositivo.

A largura de banda analógica é a quantidade de largura de banda útil (3 dB) entre a porta RF e a interface IF/banda base de um canal RF. Normalmente, essa largura de banda é definida por filtros IF ou de banda base na placa filha, que são projetados para evitar o *aliasing* quando emparelhados com uma placa-mãe USRP com dadas taxas de amostragem ADC/DAC.

Tabela 2 - Largura de banda da USRP [28]

| Placa Filha     | Faixa de frequência | Largura de banda analógica   |
|-----------------|---------------------|--|
| WBX-120         | 50 MHz - 2.2 GHz    | 120 MHz  |
| SBX-120         | 400 MHz - 4.4 GHz   | 120 MHz  |
| CBX-120         | 1.2 GHz - 6 GHz     | 120 MHz  |
| UBX-160         | 10 MHz - 6 GHz      | 160 MHz  |
| WBX             | 50 MHz - 2.2 GHz    | 40 MHz   |
| SBX             | 400 MHz - 4.4 GHz   | 40 MHz   |
| CBX             | 1.2 GHz - 6 GHz     | 40 MHz   |
| UBX-40          | 10 MHz - 6 GHz      | 40 MHz   |
| TVRX2           | 50 MHz - 860 MHz    | Configurável – 1.7 to 10 MHz   |
| DBSRX2          | 800 MHz – 2.3 GHz   | Configurável – 8 to 80 MHz   |
| BasicRX/BasicTX | 1 – 250 MHz         | *Determinado pelas taxas de amostragem da ADC/DAC. Filtro externo requerido. |
| LFRX/LFTX       | DC-30 MHz           | 30 MHz   |

Neste trabalho foram utilizadas placas-filhas WBX, e o modelo de USRP utilizado foi a USRP N210, que é considerado um dos modelos mais avançados dentro da família de produtos USRP, e cujas características são mostradas na Tabela 2.

A *Universal Software Radio Peripheral* (USRP), mostrada na Figura 7, é plataforma SDR flexível de baixo custo que pode ser utilizada com o GNU Radio por meio do driver UHD. Ela consiste em duas placas principais: a placa-mãe e a placa-filha (Figura 8).



Figura 7 - USRP N210 (Fonte: Ettus Research [25])

Tabela 3 - Características da USRP N210

| <b>USRP N210</b>   |
|--|
| ADC (conversor analógico-digital) de 14 bits dual de 100 Ms/s (amostras/seg)       |
| DAC (conversor analógico-digital) de 16 bits dual de 400 Ms/s (amostras/seg)       |
| DDC/DUC com resolução de 25 MHz  |
| Interface de rede Gigabit Ethernet com o computador                                |
| Streaming via Gigabit Ethernet de até 50 Ms/s                                      |
| FPGA Xilinx Spartan-3A DSP 3400  |
| Flexibilidade para relógio e sincronização   |
| Interfaces de RF do tipo SMA   |
| 1 MB de Memória High-speed Static Random Access Memory (SRAM)                      |
| GPS Disciplined Oscillator (OPCIONAL)  |
| Cabo MIMO da Ettus Research (pode ser utilizado para sincronismo entre duas USRPs) |
| 1 Slot para placa filha TX e 1 slot para placa-filha RX                            |



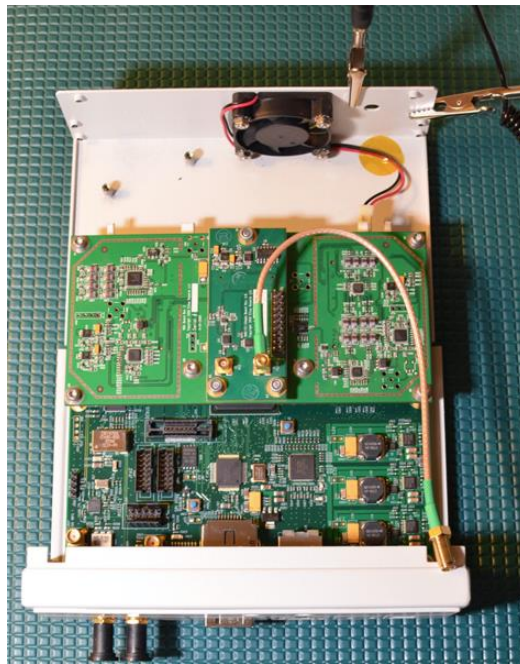


Figura 8 - Placa-mãe e placa-filha da USRP N210 Fonte: Ettus Research [25])

Conforme mostrado na Figura 9, os quatro canais de entrada e saída das ADCs e DACs são conectados a uma FPGA Xilinx Spartan-3A DSP 3400. A FPGA, por sua vez, conecta-se a um controlador Ethernet Gigabit, e por sua vez ao computador.

A utilização da USRP pelo GNU Radio é possível devido ao driver UHD que pode ser acionado tanto para transmissão como para recepção através dos blocos UHD:USRP Sink e UHD:USRP Source, respectivamente, conforme Figura 10.

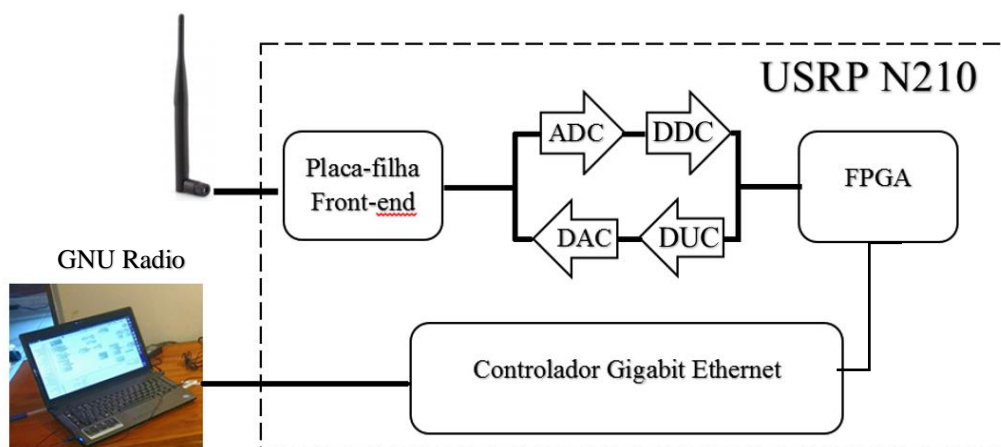


Figura 9 - Diagrama de interface da USRP N210

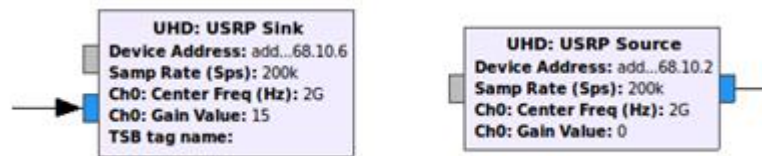


Figura 10 - Blocos do GNU Radio para uso da USRP

Conforme citado, a USRP N210 suporta diversos modelos de placas-filhas conforme tipicamente a faixa de frequência e o tipo de aplicação. No caso deste trabalho foi utilizada uma placa WBX.

A WBX [26] é um transceptor de ampla largura de banda que fornece até 100 mW de potência de saída e uma figura de ruído de 5 dB. Os osciladores locais para as cadeias de transmissão e recepção operam independentemente, mas podem ser sincronizados para operação MIMO. Ela fornece 40 MHz de capacidade de largura de banda e é ideal para aplicações que requerem acesso a diversas bandas diferentes dentro de sua faixa de operação, que vai de 50 MHz a 2,2 GHz.

Exemplos de áreas de aplicação incluem comunicações terrestres-móveis, rádios de banda marítima e de aviação; estações base de telefone celular, PCS e rádios multi-banda GSM; radares multi-estáticos coerentes; redes de sensores sem fio; transceptor amador cobrindo até 6 bandas; transmissão de TV; rádio cognitivo; segurança Pública; ISM.

## 4

## Fundamentos Teóricos para a Detecção de Energia

### 4.1.

#### A detecção de energia

A detecção de energia, como o próprio nome indica, é uma técnica de detecção não coerente (não é necessário sincronismo entre portadoras), que detecta o sinal primário comparando a sua energia medida ao longo do tempo com um determinado limiar para decidir sobre a presença ou não do sinal do usuário primário licenciado.

Além da sua relativa simplicidade teórica e de implementação em relação às demais técnicas como filtragem casada ou por cicloestacionariedade, é também considerada uma técnica de detecção cega, pelo fato de não necessitar ou utilizar qualquer conhecimento prévio das características do sinal do usuário principal. Este fato faz dela a mais adequada para uma busca rápida e pouco meticulosa do espectro, além disso, é considerada a técnica mais empregada de detecção de espectro atualmente.

O esquema de detecção é um dos mais elementares e pode ser considerado praticamente ideal, se tanto o sinal como o ruído são gaussianos e a variância de ruído é perfeitamente conhecida. Apesar disso, conforme veremos mais adiante, seu desempenho degrada-se rapidamente quando há incerteza no valor da potência de ruído e também é incapaz de diferenciar entre os sinais a partir de diferentes sistemas e entre estes sinais e o ruído.

O detector de energia às vezes também é conhecido como radiômetro [7] devido à sua aplicação muito usual em radiometria.

### 4.2.

#### Teste de Hipóteses

A teoria da decisão estatística é um pilar fundamental para a teoria da detecção, e nela, a tomada de decisão depende do conceito do teste de hipóteses. No caso da detecção de espectro, trabalhamos sempre com a questão de decidir se

o usuário primário está presente ou não, portando é uma condição binária e por isso a teoria da decisão se baseia no teste de hipóteses binárias.

O teste de hipóteses binárias considera portanto, o problema de definir uma regra de decisão que indica qual das duas hipóteses deve ser escolhida: a hipótese nula ( $H_0$ ) ou a hipótese alternativa ( $H_1$ ). Se as hipóteses nula e alternativa são definidas em termos da presença do sinal, temos a hipótese nula para sinal ausente e a hipótese alternativa para sinal presente.

Sendo assim, o problema de detecção da presença do usuário primário pode ser resolvido pelo teste estatístico de decisão entre as duas hipóteses:

$$\begin{cases} H_0: \text{usuário primário ausente} \\ H_1: \text{usuário primário presente} \end{cases} \quad (4.1)$$

Neste caso, o sinal no detector em cada hipótese assume a forma:

$$\begin{cases} H_0: x(n) = w(n) \\ H_1: x(n) = h(n) s(n) + w(n) \end{cases} \quad (4.2)$$

com  $n = 1, 2, \dots, N$  e  $w(n) \sim N(0, \sigma_w^2)$

onde  $x(n)$  é um vector bidimensional com as componentes I e Q do sinal recebido,  $w(n)$  é um ruído aditivo gaussiano branco (AWGN), com média nula e variância  $\sigma_w^2$  e  $s(n)$  o sinal enviado pelo usuário principal após a atenuação e distorção provocadas pelo canal.  $N$  é o número de amostras do sinal recebido usado no espectro processo de detecção e  $h(n)$  é a resposta impulsional do canal de transmissão.

Portanto, a regra de decisão entre as duas hipóteses é feita comparando uma variável estatística de teste  $T$  com um dado limiar  $\tau$

$$\begin{matrix} H_1 \\ T(x) \geq \tau \\ H_0 \end{matrix} \rightarrow \begin{cases} \text{Se } T > \tau \text{ decidimos pela hipótese } H_1 \\ \quad (\text{usuário primário presente}) \\ \text{Se } T < \tau \text{ decidimos pela hipótese } H_0 \\ \quad (\text{usuário primário ausente}) \end{cases} \quad (4.3)$$

O desempenho do detector é caracterizado principalmente por duas métricas: probabilidade de detecção e probabilidade de falso alarme. As probabilidades de detecção e de falso alarme são definidas como

$$\begin{aligned} P_d &= P(T > \tau \mid H_1) \\ P_{fa} &= P(T > \tau \mid H_0) \end{aligned} \quad (4.4)$$

É importante salientar que uma probabilidade de detecção baixa aumenta o risco de interferência provocada pelos usuários secundários aos usuários primários, ao passo que uma probabilidade de falso alarme elevada aumenta a ineficiência no aproveitamento das oportunidades espectrais pela rede secundária.

Se definirmos  $\mathbf{x} = [x_1, x_2, \dots, x_N]$  como o vetor de observação e as funções densidade de probabilidade (pdf) conjuntas destas  $N$  amostras ao observar  $\mathbf{x}$  dado que  $H_0$  (ou  $H_1$ ) seja verdadeira, a razão entre essas densidades é muitas vezes referida como função de verossimilhança do vetor de observação  $\mathbf{x}$ , e assim podemos definir o teste da razão de verossimilhança (*Likelihood Ratio Test* - LRT) como

$$\Lambda(\mathbf{x}) = \frac{p_{\mathbf{x}|H_1}(\mathbf{X}|H_1)}{p_{\mathbf{x}|H_0}(\mathbf{X}|H_0)} \underset{H_0}{\overset{H_1}{\geq}} \tau \quad (4.5)$$

Suponhamos que para cada hipótese tenhamos uma observação [14], neste caso, uma variável aleatória que é gerada de acordo com alguma lei de probabilidade. O teste de hipóteses consiste então em decidir qual é a hipótese correta baseado em uma medição dessa variável aleatória, neste caso, a variável  $\mathbf{x}$ .

Outro aspecto fundamental para o teste de hipóteses é o critério de decisão [14]. De acordo com esse critério, podemos, por exemplo, associar custos distintos para cada uma das hipóteses e determinar uma regra de decisão de forma que o custo médio (ou risco dessa decisão) seja minimizado. Essa é a abordagem baseada na *Regra de Bayes* e portanto é conhecida como *Baesian*.

Na verdade são definidas duas abordagens clássicas para o teste de hipóteses no aspecto da regra de decisão [8, 14]: *Neyman-Pearson* (NP) e a *Baesian*.

### 4.2.1. Teste de Bayes

No método Baesiano [11] o objetivo do teste de Bayes é minimizar um custo médio ou função "risco", cuja expressão pode ser avaliada como

$$C = C_{10} P(C_{10}) + C_{01} P(C_{01}) + C_{11} P(C_{11}) + C_{00} P(C_{00}) \quad (4.6)$$

onde  $C_{ij}$  é o custo assumido quando se aceita a hipótese  $H_i$ , sendo que  $H_j$  é a hipótese verdadeira (custo de, assumir-se  $H_i$ , dado  $H_j$ ). De forma análoga, indica a probabilidade de nós aceitarmos  $H_j$  quando, na verdade,  $H_i$  é a verdadeira. Partindo dessa expressão do custo é possível derivar a regra de decisão como

$$\Lambda(x) \underset{H_0}{\overset{H_1}{\geq}} \tau = \frac{P(H_0)(C_{10} - C_{00})}{P(H_1)(C_{01} - C_{11})} \quad (4.7)$$

onde as probabilidades  $P(H_0)$  e  $P(H_1)$  são chamadas de probabilidades de  $H_0$  e  $H_1$  respectivamente. Quando  $C_{10} - C_{00} = C_{01} - C_{11}$  o teste de Bayes torna-se o teste de máxima probabilidade *a posteriori* (MAP)

$$\Lambda(x) \underset{H_0}{\overset{H_1}{\geq}} \tau = \frac{P(H_0)}{P(H_1)} \Rightarrow p(x|H_1) P(H_1) \underset{H_0}{\overset{H_1}{\geq}} p(x|H_0) P(H_0) \quad (4.8)$$

e se  $H_0$  e  $H_1$  forem equiprováveis, ou seja,  $P(H_0) = P(H_1)$ , então o teste MAP torna-se o teste de máxima verossimilhança (ML)

$$\Lambda(x) \underset{H_0}{\overset{H_1}{\geq}} \tau = 1 \Rightarrow p(x|H_1) \underset{H_0}{\overset{H_1}{\geq}} p(x|H_0) \quad (4.9)$$

#### 4.2.2. Teste de Neyman-Pearson

De acordo com o teorema de *Neymann-Pearson* [11], para uma dada probabilidade de falso alarme fixa desejada ( $P_{fa} = \alpha$ ), o teste estatístico que maximiza a probabilidade de detecção é o *Likelihood Ratio Test* (LRT) que pode ser expresso como:

$$T_{LRT} = \Lambda(x) = \frac{p_{x|H_1}(x)}{p_{x|H_0}(x)} \underset{H_0}{\overset{H_1}{\geq}} \tau = \lambda \quad (4.10)$$

onde  $\lambda$  é o multiplicador de Lagrange e é numericamente equivalente ao limiar  $\tau$  do detector de energia.

O multiplicador  $\lambda$  é escolhido de forma a satisfazer a restrição

$$P_{fa_{desejada}} = P(T > \tau | H_0) = \int_{x|\Lambda(x) > \lambda} p_{x|H_0}(x) dx = \alpha \quad (4.11)$$

onde  $P_{fa}$  é a probabilidade de que o LRT seja maior que o limiar  $\tau$  quando a observação é composta apenas pelo ruído. Em geral, a detecção é realizada na base de um critério de taxa de falso alarme constante (*Constant False Alarm Rate* – CFAR), ou seja, a técnica NP proporciona um limiar para se obter uma detecção sujeita a uma dada  $P_{fa}$  constante.

Analogamente, a probabilidade de detecção pode ser obtida por

$$P_d = P(T > \tau | H_1) = \int_{x|\Lambda(x) > \lambda} p_{x|H_1}(x) dx \quad (4.12)$$

onde  $P_d$  é a probabilidade de que o LRT seja maior que o limiar quando a observação for composta do sinal de interesse somado ao ruído.

Como fica claro pelas expressões apresentadas, para utilizar o LRT, é necessário um perfeito conhecimento de parâmetros tais como o ruído e as distribuições de sinal primário, bem como as características de canais, no entanto, em cenários de rádio cognitivos, estas informações por vezes não estão disponíveis. Em tais casos, outras abordagens, como o método Bayesiano e o teste LRT generalizado (GLRT) são mais adequadas.

Em muitos casos, o esquema de sinalização (modulação) do usuário primário pode ser desconhecido para o usuário secundário; isso pode corresponder ao caso em que um usuário primário ágil (mudanças rápidas de modulação) tenha uma flexibilidade considerável e agilidade na escolha de sua modulação e modelagem de pulso. Nesse caso, o sinal pode ser modelado como um processo Gaussiano branco estacionário de média nula, independente do ruído de observação, que também é modelado como um processo gaussiano branco.

Quando não se tem conhecimento a priori das características de distribuição do sinal a ser detectado, podemos por exemplo assumir que siga uma distribuição gaussiana [6]. Nesse caso, Kay [8] afirma que a mudança da variância de uma estatística gaussiana pode ser usada para distinguir entre duas hipóteses.

Por exemplo, considere-se que estamos observando  $N$  amostras de um sinal  $x$  onde as essas amostras sejam consideradas estatisticamente independentes e identicamente distribuídas. Vamos então assumir que, no caso da hipótese  $H_0$ , a amostra  $x(n)$  segue uma distribuição normal com média nula e variância  $\sigma_0$ , e no caso da hipótese  $H_1$ ,  $x(n)$  segue uma distribuição normal com média nula e variância  $\sigma_1$ , ou seja

$$\begin{cases} H_0: x(n) \sim N(0, \sigma_0^2) \\ H_1: x(n) \sim N(0, \sigma_1^2) \end{cases} \quad (4.13)$$

Sob a hipótese  $H_0 \rightarrow p(x|H_0) \sim N(0, \sigma_0^2)$ , e sob a hipótese  $H_1 \rightarrow p(x|H_1) \sim N(0, \sigma_1^2)$ . Nesse caso o teste de Neyman-Pearson decide por  $H_1$  se

$$\Lambda(x) = \frac{p(x|H_1)}{p(x|H_0)} > \tau \quad (4.14)$$



Foi demonstrado por Kay [8] que o teste estatístico consiste apenas em uma estimação da variância, e se considerarmos  $\sigma_0^2 = \sigma_w^2$  e  $\sigma_1^2 = \sigma_x^2 + \sigma_w^2$  podemos perceber que, o detector de Neyman-Pearson calcula de fato é a energia do sinal recebido e a compara com um determinado limiar, e por isso, o detector de energia também é conhecido como o detector de Neyman-Pearson.

O teste de Neyman-Pearson [8] nos permite obter o LRT como

$$\Lambda(x) = \frac{p(x|H_1)}{p(x|H_0)} = \frac{1}{N} \sum_{i=1}^N x_i^2 \quad (4.15)$$

Segundo KAY (1998), na verdade, o teste estatístico baseado na soma dos quadrados das amostras pode ser visto como um estimador de variância, e portanto, o detector ótimo no sentido Neyman-Pearson para este caso é um simples detector de energia das N amostras consideradas.

$$T(x) = \frac{1}{N} \sum_{n=1}^N x^2(n) \quad (4.16)$$

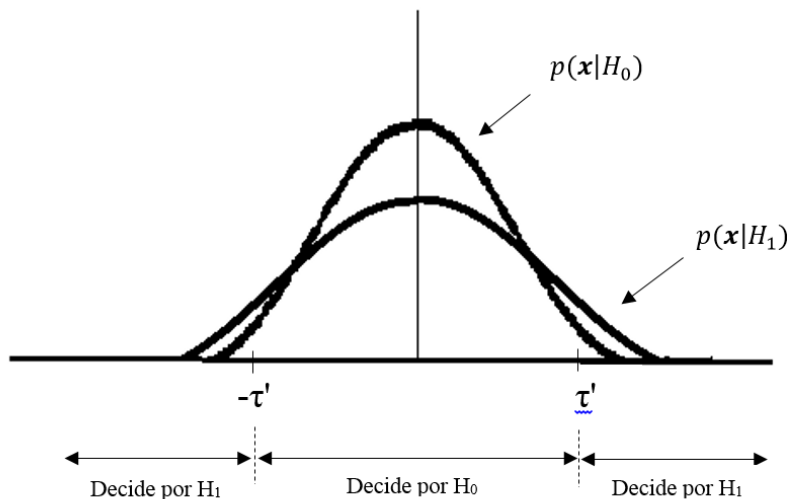


Figura 11 - Regiões de decisão com mudança na variância [8]

### 4.3.

#### O detector de energia de Urkowitz

O detector de energia convencional [6] mede a energia associada ao recebido sinal durante um determinado período de tempo e uma determinada largura de banda. O valor medido é então em comparação com um limiar apropriadamente selecionado para determinar a presença ou não do sinal primário.

O problema de detectar um sinal determinismo desconhecido sobre um canal de ruído gaussiano plano e limitado em banda foi abordado primeiramente por Urkowitz [4]. Em seu clássico trabalho, ao fazer-se uma análise teórica do detector de energia convencional no domínio do tempo, podemos considerar tanto as versões analógica como digital.

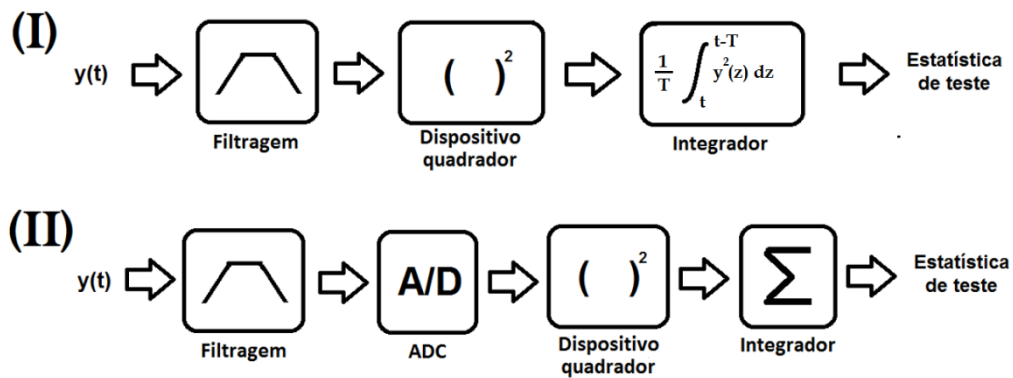


Figura 12 - Detector de energia convencional: (I) analógico; (II) digital.

O sinal de entrada  $y(t)$  passa primeiro por um filtro passa-banda com frequência central  $f_0$  e largura de banda  $W$ , o qual apresenta como função de transferência

$$H(f) = \begin{cases} \frac{2}{\sqrt{N_0}}, & |f - f_0| \leq W \\ 0, & |f - f_0| > W \end{cases} \quad (4.17)$$

onde  $N_0$  é a densidade espectral de potência de ruído unilateral. Esta definição foi considerada conveniente para se obter as probabilidades de detecção e de falso alarme usando a função de transferência relacionada.

Depois disso, o sinal é elevado ao quadrado, e integrado no intervalo de observação  $T$  para produzir uma estatística de teste  $V$ , a qual é comparada com um limiar  $\lambda$ . O receptor toma a decisão de que o sinal alvo foi detectado se, e somente se, o limiar for excedido.

Na versão digital, o sinal recebido  $y(t)$  é filtrado e digitalizado por um conversor analógico-digital (ADC) e em seguida as amostras são elevadas ao quadrado e passam por um acumulador que nos dá o conteúdo da energia nas  $N$  amostras consideradas. Essa energia então serve como uma estatística de teste para o detector de energia.

Portanto, o detector de energia mede a energia do sinal recebido em uma banda limitada em  $W$  Hz de uma forma de onda recebida  $y(t)$  com uma duração de sensoriamento de  $T$  segundos e aproxima esta medida pela soma dos quadrados de um número limitado de  $N$  amostras.

O sinal recebido  $y(t)$  do usuário secundário é submetido ao teste de hipóteses binárias e neste caso, o sinal no detector em cada hipótese assume a forma:

$$\begin{cases} H_0: y(t) = n(t) \\ H_1: y(t) = x(t) + n(t) \end{cases} \quad (4.18)$$

onde  $x(t)$  é o sinal determinístico desconhecido transmitido e  $n(t)$  o ruído gaussiano, assumido como um sinal AWGN com média nula e variância  $\sigma_n^2 = WN_0$  e é conhecido a priori. A relação sinal-ruído SNR é representada por  $\gamma = \frac{\sigma_s^2}{\sigma_n^2}$  onde  $\sigma_s^2$  é a variância do sinal e  $\sigma_n^2$  é a variância do ruído.

Segundo Urkowitz [4], as distribuições do teste estatístico  $V'$  para ambos os casos podem ser expressas por

$$V' \sim \begin{cases} H_0: \chi^2_{2TW} \\ H_1: \chi^2_{2TW}(\lambda) \end{cases} \quad (4.19)$$

Nesse caso, considerando um dado limiar  $V'_T$ , as probabilidades de detecção e falso alarme podem ser definidas como

$$P_d = P(V' > V'_T | H_1) = P(\chi^2_{2TW}(\lambda) > V'_T | H_1) \quad (4.20)$$

$$P_{fa} = P(V' > V'_T | H_0) = P(\chi^2_{2TW} > V'_T | H_0) \quad (4.21)$$

Quando o produto  $2TW > 250$ , podemos utilizar aproximações gaussianas para as funções de densidade de probabilidade de  $\chi^2_{2TW}$  e  $\chi^2_{2TW}(\lambda)$  referentes ao teste estatístico  $V'$ . Nesse caso, as expressões apropriadas podem ser determinadas de forma bem mais simples utilizando apenas distribuições gaussianas com a média e variância adequadas, para encontrar a probabilidade de  $V'$  exceder o limiar  $V'_T$ .

No caso da probabilidade de falso alarme, temos uma distribuição gaussiana de média  $2TW$  e variância  $4TW$ , logo

$$P_{fa} = P(V' > V'_T | H_0) = \frac{1}{\sqrt{2\pi} \sqrt{4TW}} \int_{V'_T}^{\infty} e^{-\frac{(x-2TW)^2}{2(4TW)}} dx \quad (4.22)$$

$$\Leftrightarrow P_{fa} = \frac{1}{2} \operatorname{erfc} \left[ \frac{V'_T - 2TW}{2\sqrt{2TW}} \right] \quad (4.23)$$

Já no caso da probabilidade de detecção, temos uma distribuição gaussiana de média  $(2TW+\lambda)$  e variância  $4(TW+\lambda)$ , logo

$$P_d = P(V' > V'_T | H_1) = \frac{1}{\sqrt{2\pi} \sqrt{4(TW + \lambda)}} \int_{V'_T}^{\infty} e^{-\frac{(x-(2TW+\lambda))^2}{2(4(TW+\lambda))}} dx \quad (4.24)$$

$$\Leftrightarrow P_d = \frac{1}{2} \operatorname{erfc} \left[ \frac{V'_T - (2TW + \lambda)}{2\sqrt{2(TW + \lambda)}} \right] \quad (4.25)$$

As expressões apresentadas por Urkowitz têm o mérito de demonstrar de uma forma direta a importância que a largura de banda  $W$  observada e também a duração do tempo de observação  $T$  possuem no desempenho do detector. É interessante observar que o desempenho depende do produto  $WT$ , e se o sinal  $y(t)$  for amostrado à taxa de Nyquist, isto é, se a frequência de amostragem for  $f_s=2W$  amostras/seg, isso leva a um número de amostras necessário de  $N=2WT$  amostras a serem observadas no intervalo de tempo de  $T$  segundos. Além disso, ao invés de reconstruirmos a forma de onda de  $y(t)$  para obter sua energia, podemos simplesmente usar a energia dessas  $N$  amostras observadas.

#### 4.4.

#### Detector de energia – versão analógica vs. digital

O detector de energia apresenta uma equivalência [6] entre suas versões analógica e digital. A versão analógica consiste em um pré-filtro seguido por um dispositivo de lei quadrática que implementa o quadrado do sinal (um diodo na região em que sua característica é quadrática, por exemplo) e um filtro integrador de tempo finito. O pré-filtro serve para limitar a largura de banda e normalizar a variância do ruído. A saída do integrador é então proporcional à energia do sinal recebido. O modelo digital consiste em um pré-filtro de ruído passa baixo que limita o ruído e os sinais de largura de banda adjacente, um conversor analógico-digital (ADC) que converte os sinais contínuos em amostras de sinais digitais discretos e um dispositivo de lei quadrática seguido por um integrador.

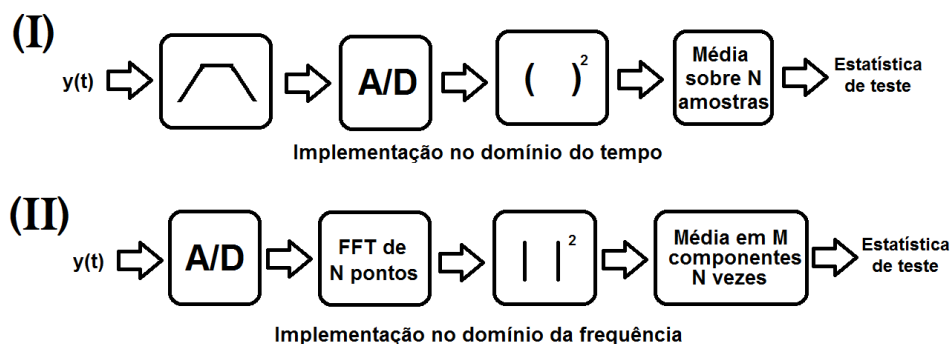


Figura 13 - A detecção de energia no tempo e na frequência

Essa mesma estrutura no domínio do tempo também pode ser realizada no domínio da frequência, e nesse caso, para se obter a energia do sinal em uma determinada largura de banda, o sinal no domínio do tempo é transformado para domínio de frequência usando FFT e podemos calcular a energia do sinal somando as componentes (tons) da FFT sobre toda a banda do sinal. A vantagem da implementação no domínio de frequência reside na flexibilidade que a conversão FFT pode proporcionar permitindo a troca de resolução temporal por resolução de frequência. Isto significa que um sinal de banda estreita e frequência central pode ser estimado sem a necessidade de um pré-filtro muito flexível (grande faixa de sintonia).

A saída do integrador (analógico ou digital) é chamada de estatística de decisão (ou de teste). Essa estatística é comparada com um limiar para se tomar a decisão final pela presença ou não do sinal primário. No entanto, vale observar que nem sempre a estatística de teste terá obrigatoriamente de se a saída do integrador, pois pode ser empregado também (geralmente por razões de simplificação ou implementação prática) uma função que seja monótona com a saída do integrador [4]. Neste trabalho foi implementada a detecção de energia apenas no domínio do tempo.

A estatística de teste nos submete à abordagem clássica para se chegar à conclusão pela presença ou não desse usuário primário que é a teoria do teste de hipóteses.

#### 4.5.

#### A estatística da detecção de energia

Conforme referido por Urkowitz [4] em seu estudo, a estatística do teste para o detector de energia pode ser definida tanto no domínio do tempo como no domínio da frequência, e, olhando para estatística do detector no domínio do tempo temos a seguinte métrica  $T(x)$  a ser utilizada:

$$\text{Domínio do Tempo} \rightarrow T(x) = \sum_{n=1}^N x_n^2 \underset{H_0}{\overset{H_1}{\geq}} \tau \quad (4.26)$$

e aplicando-a do teste de hipóteses definido e considerando o sistema puramente AWGN, ou seja,  $h(n)=1$  para qualquer  $n$ , então temos

$$\begin{cases} H_0: x(n) = w(n) \\ H_1: x(n) = s(n) + w(n) \end{cases} \quad (4.27)$$

onde o detector de energia irá contabilizar

$$T(x) = \begin{cases} \sum_{n=1}^N w(n)^2 & \text{dado } H_0 \\ \sum_{n=1}^N [s(n) + w(n)]^2 & \text{dado } H_1 \end{cases} \quad (4.28)$$

Nesse caso as probabilidades de detecção e de falso alarme serão dadas por

$$P_d = P(T > \tau | H_1) = P\left(\sum_{n=1}^N [s(n) + w(n)]^2 \geq \tau\right) \quad (4.29)$$

$$P_{fa} = P(T > \tau | H_0) = P\left(\sum_{n=1}^N w(n)^2 \geq \tau\right) \quad (4.30)$$

Assume-se aqui que o ruído tem uma densidade espectral de potência plana e limitada em largura de banda (W). Nesse caso, por meio de amostragem, a energia em uma amostra de tempo finito do ruído pode ser aproximada pela soma dos quadrados de variáveis aleatórias estatisticamente independentes com médias nulas e variâncias iguais. Essa soma tem uma distribuição qui-quadrado.

De acordo com o Teorema do Limite Central, considerando um número de amostras  $N$  suficiente (para Urkowitz,  $N > 250$ ), o somatório apresentado em  $T(x)$  para as hipóteses  $H_0$  e  $H_1$  faz com que  $T(x)$  possa ser considerada uma variável aleatória gaussiana, e nesse caso, o teste estatístico  $T(x)$  passa a poder ser expresso da seguinte forma [6]:

$$T(x) \sim N \left( \sum_{n=1}^N E [|x(n)|^2], \sum_{n=1}^N Var [|x(n)|^2] \right) \quad (4.31)$$

onde

$$E [|x(n)|^2] = \begin{cases} H_0: \sigma_w^2 \\ H_1: (\sigma_w^2 + \sigma_s^2) \end{cases} \quad (4.32)$$

$$Var [|x(n)|^2] = \begin{cases} H_0: 2(\sigma_w^2)^2 \\ H_1: 2(\sigma_w^2 + \sigma_s^2)^2 \end{cases} \quad (4.33)$$

e como os somatórios tem N termos iguais,

$$\sum_{n=1}^N E [|x(n)|^2] = \begin{cases} H_0: N\sigma_w^2 \\ H_1: N(\sigma_w^2 + \sigma_s^2) \end{cases} \quad (4.34)$$

$$\sum_{n=1}^N Var [|x(n)|^2] = \begin{cases} H_0: 2N(\sigma_w^2)^2 \\ H_1: 2N(\sigma_w^2 + \sigma_s^2)^2 \end{cases} \quad (4.35)$$

então,

$$T(x) \Rightarrow \begin{cases} \mathcal{N}(N\sigma_w^2, 2N\sigma_w^4) & \text{dado } H_0 \\ \mathcal{N}(N(\sigma_w^2 + \sigma_s^2), 2N(\sigma_w^2 + \sigma_s^2)^2) & \text{dado } H_1 \end{cases} \quad (4.36)$$

e podemos representar  $(\sigma_w^2 + \sigma_s^2)^2$  levando em consideração a SNR

$$(\sigma_w^2 + \sigma_s^2)^2 = \sigma_w^4(1 + \gamma)^2 \quad (4.37)$$

nesse caso, temos

$$T(x) \Rightarrow \begin{cases} \mathcal{N}(N\sigma_w^2, 2N\sigma_w^4) & \text{dado } H_0 \\ \mathcal{N}(N\sigma_w^2(1 + \gamma), 2N\sigma_w^4(1 + \gamma)^2) & \text{dado } H_1 \end{cases} \quad (4.38)$$



Como  $T(x)$  segue uma distribuição gaussiana com média  $\mu$  e variância  $\sigma^2$ , ou seja,  $T \sim \mathcal{N}(\mu, \sigma)$ , então a probabilidade de  $T(x)$  exceder o limiar  $\tau$  é dada por

$$P(T > \tau) = Q\left(\frac{\tau - \mu}{\sigma}\right) \quad (4.39)$$

onde a função  $Q$  é definida da seguinte forma

$$Q(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_x^{\infty} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \quad (4.40)$$

#### ***Uso da função de erro complementar no GNU Radio***

*Para o cálculo das probabilidades  $P_d$  e  $P_{fa}$  por meio da função  $Q(x)$  na expressões utilizadas no GNU Radio, é utilizada a função de erro complementar  $\text{erfc}(x)$  através da equivalência*

$$Q(x) = \frac{1}{2} \text{erfc}\left(\frac{x}{\sqrt{2}}\right) \quad (4.41)$$

Dessa forma, podemos utilizar a função  $Q(x)$  para obter as probabilidades de detecção e falso alarme da seguinte forma

$$P_d = P(T > \tau | H_1) = \int_{\tau}^{\infty} p_{t|H_1}(T) dT = \frac{1}{\sqrt{2\pi}\sigma} \int_{\tau}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (4.42)$$

$$P_d = Q\left(\frac{\tau - N(\sigma_w^2 + \sigma_s^2)}{\sqrt{2N(\sigma_w^2 + \sigma_s^2)^2}}\right) \quad (4.43)$$

$$P_{fa} = P(T > \tau | H_0) = \int_{\tau}^{\infty} p_{t|H_0}(T) dT = \frac{1}{\sqrt{2\pi}\sigma} \int_{\tau}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (4.44)$$

$$P_{fa} = Q\left(\frac{\tau - N\sigma_w^2}{\sqrt{2N\sigma_w^4}}\right) \quad (4.45)$$

ou podemos usar a representação com a SNR e então

$$P_d = Q\left(\frac{\tau - N\sigma_w^2(1 + \gamma)}{\sqrt{2N\sigma_w^4(1 + \gamma)^2}}\right) \quad (4.46)$$

$$P_{fa} = Q\left(\frac{\tau - N\sigma_w^2}{\sqrt{2N\sigma_w^4}}\right) \quad (4.47)$$

e temos ainda a representação com a aproximação para SNR baixa

$$P_d = Q\left(\frac{\tau - N\sigma_w^2(1 + \gamma)}{\sqrt{2N\sigma_w^4(1 + 2\gamma)}}\right) \quad (4.48)$$

$$P_{fa} = Q\left(\frac{\tau - N\sigma_w^2}{\sqrt{2N\sigma_w^4}}\right) \quad (4.49)$$

Colocando essas expressões da aproximação gaussiana em ordem a  $\tau$  e assumindo  $P_{fa}$  como constante, podemos obter a expressão onde, de acordo com o método de Neyman-Pearson, o limiar pode ser projetado para satisfazer o critério de uma taxa de falso alarme constante (CFAR).

$$\begin{aligned} \tau_{CFAR} &= N\sigma_w^2 \left( \sqrt{\frac{2}{N}} Q^{-1}(P_{fa}) + 1 \right) \\ \Leftrightarrow \tau_{CFAR} &= \sigma_w^2 (\sqrt{2N} Q^{-1}(P_{fa}) + N) \end{aligned} \quad (4.50)$$

Nesse caso, partindo das equações de  $P_d$  e  $P_{fa}$  o número mínimo de amostras do sinal recebido para atingir o desempenho definido em  $P_d$  e  $P_{fa}$  (para uma dada relação sinal-ruído  $\gamma$ ) é dado aproximadamente por:

$$N_{min} = 2 \left[ \frac{Q^{-1}(P_{fa}) - (1 + \gamma)Q^{-1}(P_d)}{\gamma} \right]^2 \quad (4.51)$$

## 5

## Implementação da Detecção de Energia

### 5.1.

#### Construção do detector de energia no GNU Radio

O trabalho de construção e implementação do detector de energia, assim como das simulações e ensaios, foi todo desenvolvido usando a plataforma GNU Radio versão 3.7.11 em dois computadores com o sistema operacional Ubuntu versão 16.04 LTS. Os detalhes do procedimento utilizado para instalação do GNU Radio são apresentados no Anexo A.

A primeira abordagem consiste na construção do bloco do detector de energia, que implementa o algoritmo de detecção de energia no domínio do tempo no GNU Radio. Para isso, foi utilizada a ferramenta gráfica de desenvolvimento do GNU Radio, que é o GNU Radio Companion, ou GRC.

Conforme citado no item 4.5, a métrica  $T(x)$  empregada para decidir sobre a presença ou não do usuário primário consiste no cálculo da energia das amostras do sinal recebido e a comparação com um limiar  $\tau$  projetado para satisfazer o critério de uma taxa de falso alarme constante (CFAR).

Portanto, para cada  $N$  amostras consideradas no processo de se chegar a uma decisão sobre a presença ou não do usuário primário, interessa-nos efetuar o processamento de acordo com a expressão (4.26).

Esse processamento no GNU Radio pode ser obtido conjugando o bloco *Multiply*, que obtém o quadrado das amostras, com o bloco *Moving Average*, que implementa um filtro de média móvel básico. Para se obter a média móvel das  $N$  amostras consideradas, o parâmetro *Length* deve assumir o valor de  $N$  e o parâmetro *Scale* deve ser definido como  $1/N$ . *Max Iter* é um parâmetro relativo ao número máximo de somas permitidas para se obter a média móvel, permitindo assim limitar a sobrecarga de operações no fluxo de amostras.

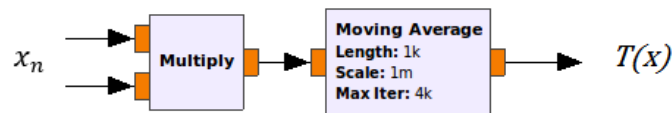


Figura 14 - Cálculo da energia de N amostras no GNU Radio (N=1000)

Para concluir o processo de detecção, só falta comparar o valor da estimação da energia do sinal obtido com um dado limiar por meio do bloco *Threshold*, que compara a métrica  $T(x)$  obtida com o limiar calculado de acordo com o critério CFAR já definido anteriormente e que resulta no valor da decisão do detector (usuário presente = 1, ausente = 0).

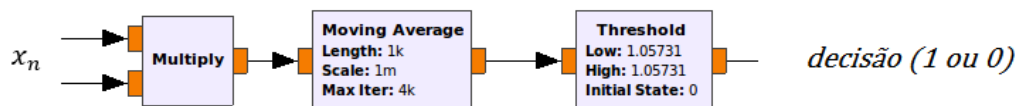


Figura 15 - Decisão comparando a energia calculada com um dado limiar

Portanto, conjugando o processamento desses três blocos em sequência, o GNU Radio nos permite criar um único bloco para essa tarefa que é o detector de energia, cujos passos da implementação por meio do novo bloco *Energy detector ff* em concreto são apresentados em anexo no item .

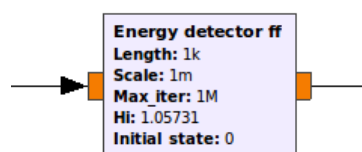


Figura 16 - Bloco do detector de energia

Vale destacar neste ponto, porém, as etapas essenciais na construção do bloco no GNU Radio, focando no exemplo de bloco *energy\_detector\_ff* em particular.

Os blocos no GNU Radio são construídos na forma de arquivos de código contendo os *scripts* de definição e processamento do bloco e que são inseridos

dentro de um módulo. Mas qual a relação entre um bloco e um módulo? O bloco na verdade é o objeto a ser instanciado dentro de um fluxograma enquanto o módulo é estrutura construída dentro do GNU Radio hospedar o código de definição do bloco, o qual quando instalado já vem com diversas bibliotecas e módulos nativos disponíveis. Normalmente, quando se deseja incluir um bloco em particular, é necessário importar no código *python* do fluxograma .grc o módulo que contém esse bloco da seguinte forma:

```
import "nome do módulo"
```

Caso esse módulo pertença a alguma biblioteca específica, como a biblioteca *gnuradio*, por exemplo, devemos declarar essa origem na importação do módulo da seguinte forma:

```
from gnuradio import "nome do módulo"
```

Este tipo de bloco específico concebido e implementado por um usuário do GNU Radio, obviamente não vem disponível na “árvore” nativa do GRC. Por esse motivo, ele é denominado de um módulo fora da árvore, ou módulo OOT.

Os detalhes de criação do bloco do detector de energia empregado neste trabalho são apresentados no Anexo B.

## 5.2. Estrutura de avaliação de desempenho

Para podermos fazer uma análise de desempenho do detector criado, foi criada uma estrutura em dois ramos com a presença apenas do ruído no ramo de cima e a soma do ruído com o sinal do usuário primário no ramo de baixo, juntamente com o detector de energia e uma estrutura que, através do bloco *Error Rate* permite obter a taxa de falso alarme no ramo de cima e a taxa de detecção no ramo de baixo, temos portanto uma estrutura que nos permite estimar o desempenho de um dado detector (neste caso o detector de energia, mas poderia ser de outro tipo) em termos de uma estimativa das suas probabilidades de detecção e falso

alarme, podendo inclusive obter uma estimativa das curvas ROC (*Receiver Operating Characteristic*) do detector em estudo.

Os passos para criação da estrutura de análise de desempenho estão no Anexo C.

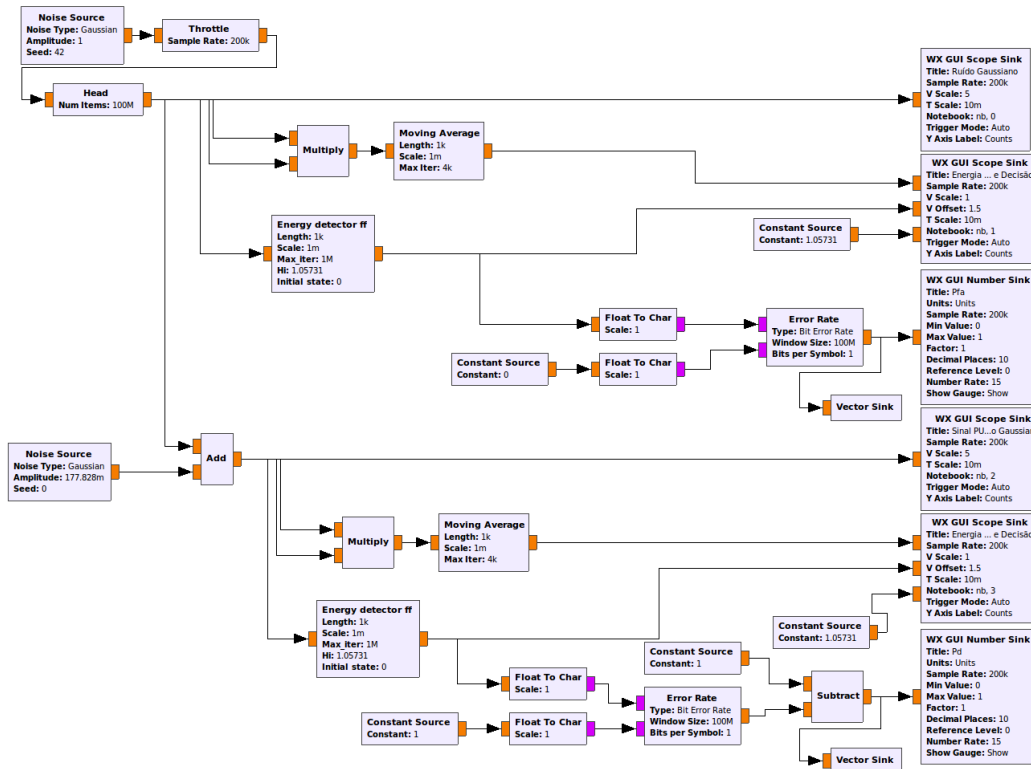
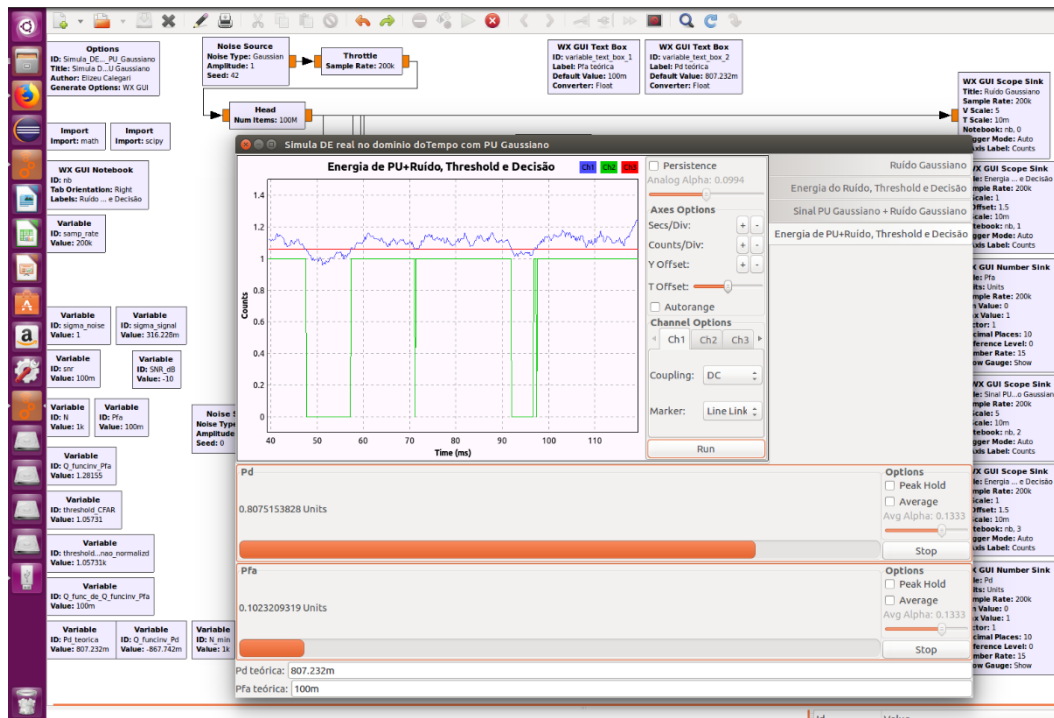


Figura 17 - Estrutura de análise de desempenho com estimativas de Pfa e Pd

Com essa estrutura implementada, podemos, por exemplo, obter uma estimativa da probabilidade de detecção especificando uma probabilidade de falso alarme nominal de  $P_{fa} = 0.1$ , considerando  $N=1000$  amostras e uma relação sinal-ruído  $SNR = -10$  dB.

Figura 18 - Resultados para  $P_d$  e  $P_{fa}$  obtidos com a estrutura de análise

Podemos observar por meio do bloco *WX GUI Scope Sink* a visualização em azul da energia do sinal do usuário primário contaminado pelo ruído, em vermelho o limiar obtido em função da probabilidade de falso alarme especificada e em verde as decisões determinadas pelo detector de energia em análise. Embaixo, o bloco *WX GUI Number Sink* exibe o valor numérico da estimativa para as probabilidades de detecção e falso alarme medidas e com uma animação estilo *runbar*. Esses valores são atualizados segundo uma taxa especificada no parâmetro *Number Rate* do bloco, neste caso, 15 vezes por segundo.

É importante observar que os blocos de visualização eventualmente sobrecarregam a execução do fluxo de dados e portanto caso ocorram travamentos ou se pretenda melhorar o desempenho do processamento podemos desabilitar alguns blocos, como por exemplo os blocos *WX GUI Scope Sink*, e dessa forma obtemos apenas os resultados das estimativas das probabilidades  $P_d$  e  $P_{fa}$  pretendidas sem sobrecarregar o fluxo de dados desnecessariamente.



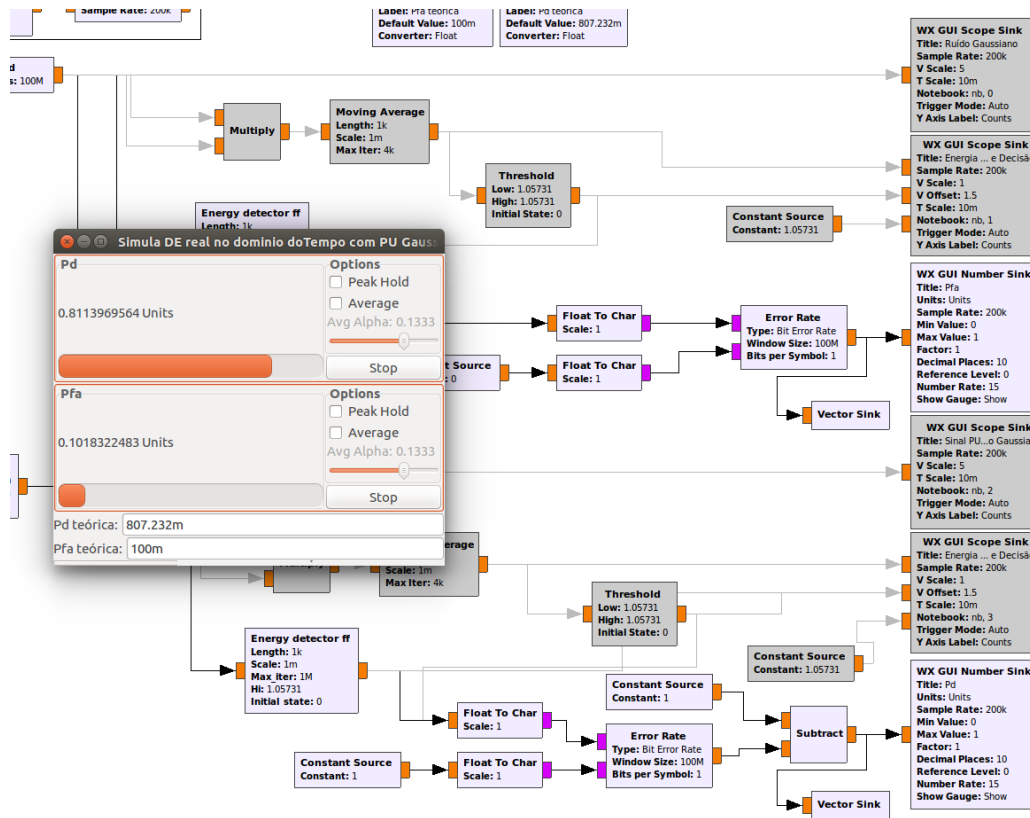


Figura 19 - Visualização compacta de Pd e Pfa pela estrutura de análise

Esse aspecto ganha ainda mais importância ao executarmos os *scripts* que irão gerar diversos fluxogramas desse tipo em sequência e variando os parâmetros de interesse, como por exemplo o número de amostras  $N$  ou a relação sinal ruído SNR<sub>dB</sub> de forma a podermos obter as curvas ROC do detector em análise.

## 6 Simulações

### 6.1. Criando os *scripts* de Simulação

Agora que temos uma estrutura de avaliação de desempenho do detector na forma de um fluxograma GRC capaz de obter as estimativas para as probabilidades de detecção  $P_d$  e falso alarme  $P_{fa}$  ao mesmo tempo, podemos otimizar o código python gerado pelo GRC para o fluxograma implementado e automatizar a obtenção dessas métricas variando a SNR, o número de amostras  $N$  ou até a  $P_{fa}$  desejada, para obter assim as curvas de desempenho ROC simuladas que nos permitirão avaliar o desempenho do detector.

A descrição de como foram gerados os scripts de automatização dos fluxogramas para efetuar as simulações está no Anexo D.

### 6.2. Eclipse e PyDev

Para automatizar o *script* em python foi utilizada a plataforma *Eclipse IDE for C/C++ developers*.

A apresentação do Eclipse e o link do instalador necessário para instalação do Eclipse encontram-se na internet no sítio <https://www.eclipse.org>, porém podemos adiantar que a apresentação principal consiste em uma IDE (Interface Development Environment), ou seja, um ambiente de desenvolvimento, que contempla principalmente uma guia “Project Explorer” (em vermelho), onde podem constar as pastas, módulos e *scripts* em desenvolvimento, uma janela de edição para o *script* de trabalho (verde) e um janela de interação do ambiente (azul), inclusive com um console de diálogo com o código em execução.

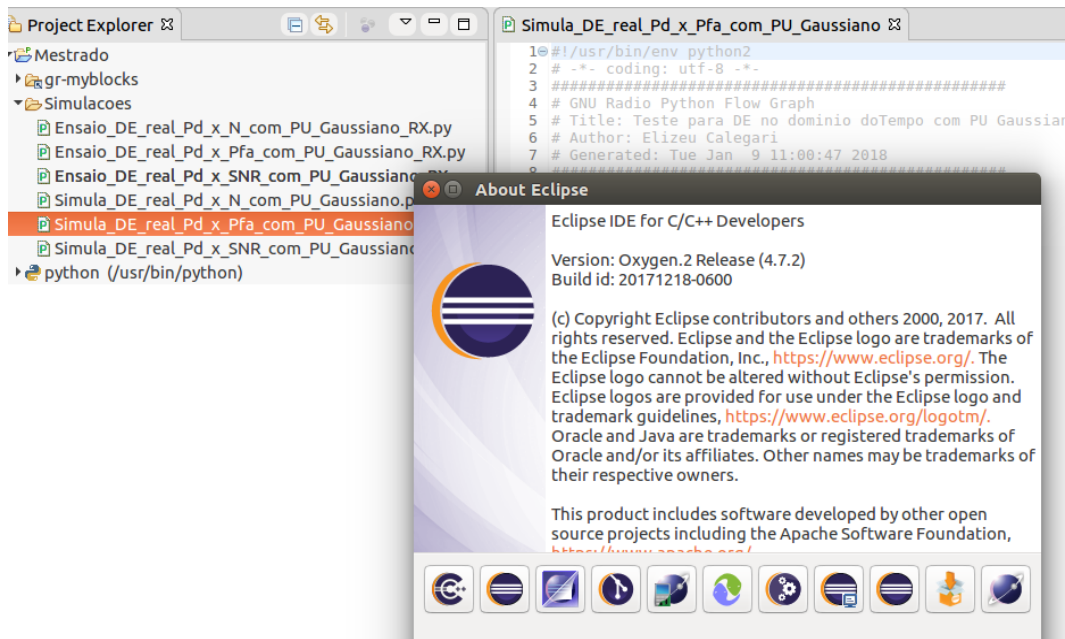


Figura 20 - A plataforma Eclipse IDE for C/C++ developers

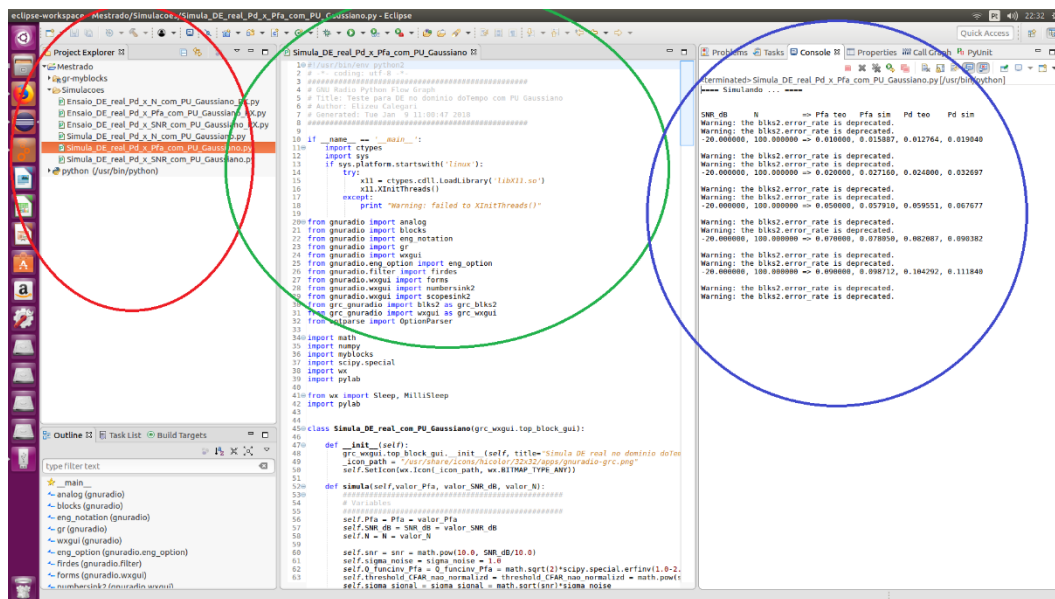


Figura 21 - Apresentação do ambiente de desenvolvimento do Eclipse

A versão do Eclipse instalada é própria para o desenvolvimento em C/C++, o que é bastante conveniente, visto que a grande maioria dos blocos no GNU Radio são implementados em C++. Por outro lado, o código gerado pelo GRC referente aos fluxogramas implementados para análise do detector é feito na linguagem *python* mas o Eclipse dispõe de recursos para atender essa necessidade. Outro aspecto que destaca a importância do Eclipse poder trabalhar (criar, editar, compilar

e executar) código em *python*, é que o GNU Radio também admite a implementação de blocos em *python*, e, mais do que isso, os *scripts* de controle de qualidade dos blocos no GNU Radio usualmente também são gerados em *python*.

Em função da relevância do manuseio do *python* ao implementar soluções personalizadas no GNU Radio, podemos aproveitar no Eclipse a disponibilidade de um *plugin* denominado *PyDev*, o qual permite de forma amigável a implementação e customização dos *scripts* gerados pelo GRC para seus fluxogramas, os quais serão bastante úteis na automatização necessária para implementar as simulações e ensaios necessários para avaliar o desempenho do detector.

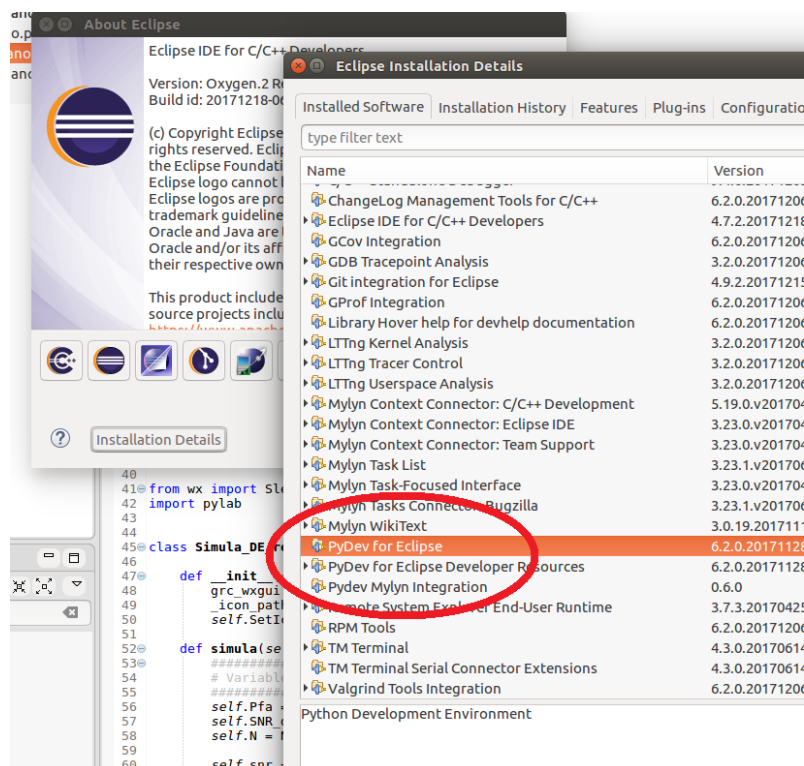


Figura 22 - Instalação do *plugin* Pydev no Eclipse para o python

### 6.3. Simulações realizadas

Tendo definido os *scripts* de simulação, foram realizadas três simulações para obter as curvas de probabilidade de detecção ( $P_d$ ) em função da relação sinal-ruído (SNR), probabilidade de falso alarme ( $P_{fa}$ ) e número de amostras ( $N$ ), conforme apresentado no capítulo referente aos resultados.

## 7

## Ensaio com as USRP

### 7.1.

#### Montagem da configuração de Ensaio

A criação do detector de energia no GRC e de um fluxograma com a estrutura de avaliação de desempenho foi necessária para permitir efetuar essa análise para cada cenário (Pfa, N e SNR) considerado. Assim como no caso da simulação, foram desenvolvidos *scripts* de automatização desse processo para varrimento em termos de Pfa, N ou SNR, porém agora, para podermos implementar os ensaios de avaliação de desempenho do detector de energia, precisaremos de dois fluxogramas GRC.

Um fluxograma será executado no computador que irá gerar o sinal de usuário primário e transmiti-lo via uma USRP (Tx) e outro fluxograma será executado no computador que irá receber esse sinal via outra USRP (Rx) que irá então realizar a detecção e análise de desempenho, conforme visto anteriormente na implementação da estrutura de avaliação de desempenho do receptor.



Figura 23 - Configuração típica de ensaio com USRP Tx e USRP Rx

Dessa forma, foi montada uma configuração de ensaio com um notebook para gerar o sinal de usuário primário e transmitir suas amostras pela USRP Tx para a

USRP Rx através do meio escolhido (por exemplo, um cabo coaxial com atenuador de 30dB). As amostras do sinal recebido foram processadas implementando a detecção de energia por meio do detector construído, e em seguida, foi realizada a análise de desempenho do sinal recebido (obtenção das estimativas para as probabilidades de detecção e falso alarme).

### *Fluxograma em Rx*

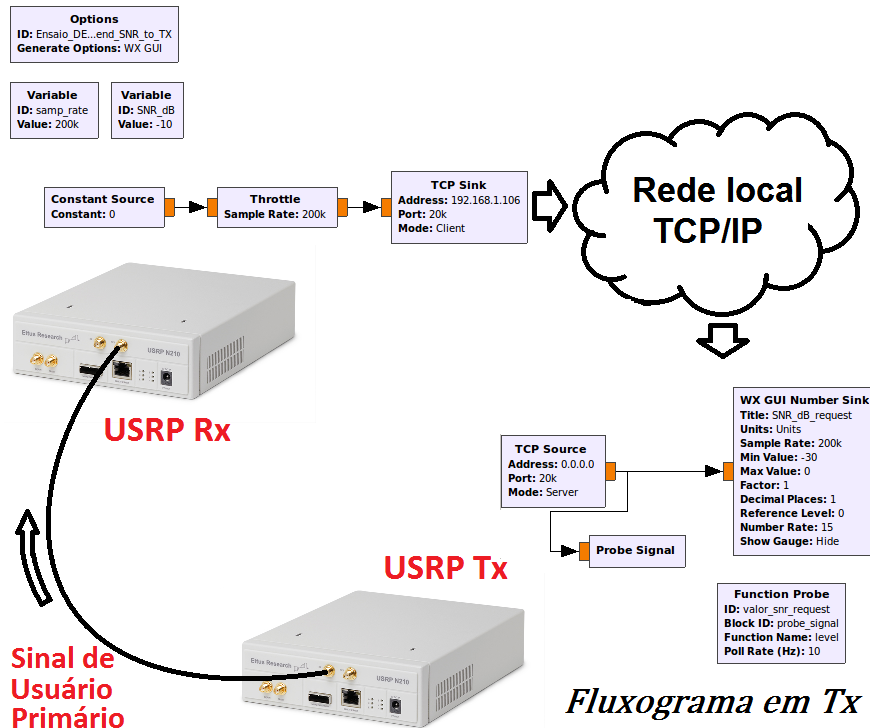


Figura 24 - Esquema de conexão entre as USRP e a comunicação via LAN

## 7.2.

### Geração e transmissão do sinal do usuário primário - PU

No caso do fluxograma que irá gerar o sinal de usuário primário, temos, tal como anteriormente, uma fonte geradora do sinal gaussiano (bloco *Noise Source*), cuja amplitude irá depender da SNR especificada, uma estrutura de estimação da energia do sinal a ser transmitido (blocos *Multiply*, *Moving Average* e os blocos visualizadores *WX GUI Scope* e *WX GUI Number Sink*) e o bloco *UHD: USRP Sink* que permite enviar dados para a USRP para transmissão do sinal de usuário primário gerado.

O fluxograma em Tx dispõe também de um mecanismo de comunicação que permite receber do *script* que implementa o ensaio no receptor uma solicitação de qual a relação sinal-ruído demandada para a determinação das probabilidades requeridas em uma dada configuração de número de amostras e probabilidade de falso alarme especificadas.

Essa comunicação foi realizada a partir do transmissor através de uma conexão TCP/IP de uma rede Wi-fi local utilizando o bloco *TCP Source*, o qual irá estabelecer a conexão com um bloco *TCP Sink* no receptor.

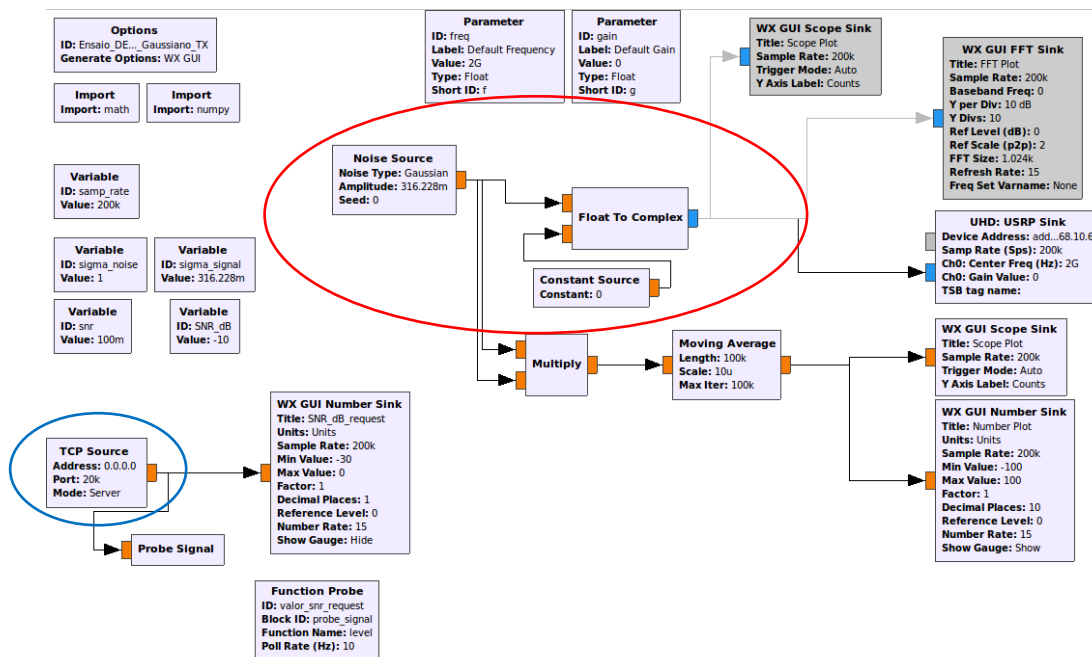


Figura 25 - Fluxograma para gerar e transmitir o sinal primário

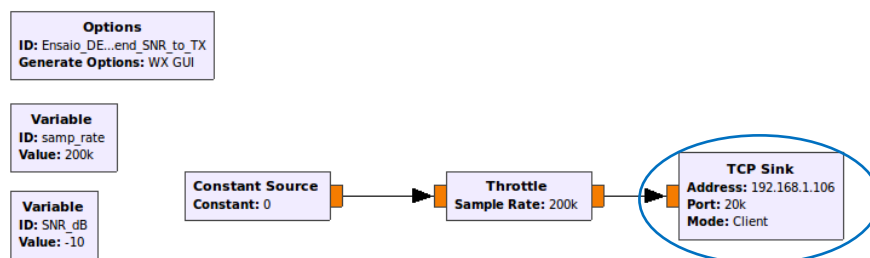


Figura 26 - Fluxograma em RX que envia a SNR requerida para o TX



Como a variância do ruído é unitária, essa demanda de SNR recebida acaba por determinar qual a amplitude do sinal gaussiano a ser gerado e transmitido via USRP Tx para o receptor.

No componente *TCP Source* do transmissor, podemos especificar o endereço IP 0.0.0.0 que fará com que a interface receba a comunicação de qualquer máquina da rede que tente comunicar por meio da porta especificada, neste caso a porta 20000. Além disso, no componente *TCP Sink* do receptor, especificamos o endereço IP destinatário e a porta a ser utilizada para enviar o valor de SNR pretendido à USRP que está transmitindo o sinal do usuário primário.

### 7.3. Recepção, detecção e análise de desempenho

No caso do fluxograma que irá receber o sinal de usuário primário, temos, tal como realizado anteriormente no fluxograma para a simulação, a geração, por meio de um bloco *Noise Source* de um ruído gaussiano local com média nula e variância unitária, que será somado ao sinal recebido via USRP (Rx), e a mesma estrutura de análise de desempenho (estimação das probabilidades de detecção e falso alarme) em função do cenário de Pfa, N e SNR especificado.

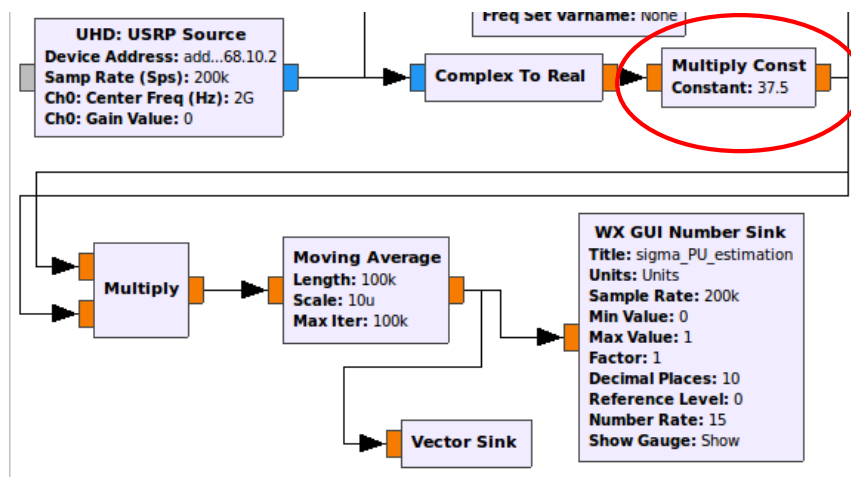


Figura 27 - *Multiply Const* com fator de ajuste da energia do sinal no detector

Neste caso, dispomos agora de um bloco *UHD: USRP Source* que permite receber as amostras de sinal transmitidas pela USRP (Tx), e novamente uma estrutura de estimação da energia do sinal recebido (blocos *Multiply*, *Moving*



*Average* e os blocos visualizadores *WX GUI Scope* e *WX GUI Number Sink*), que permitirá calibrar a energia do sinal que chega no detector, facilitando dessa forma a comparação com os resultados (desempenho) previstos no modelo teórico e os resultados obtidos nas simulações.

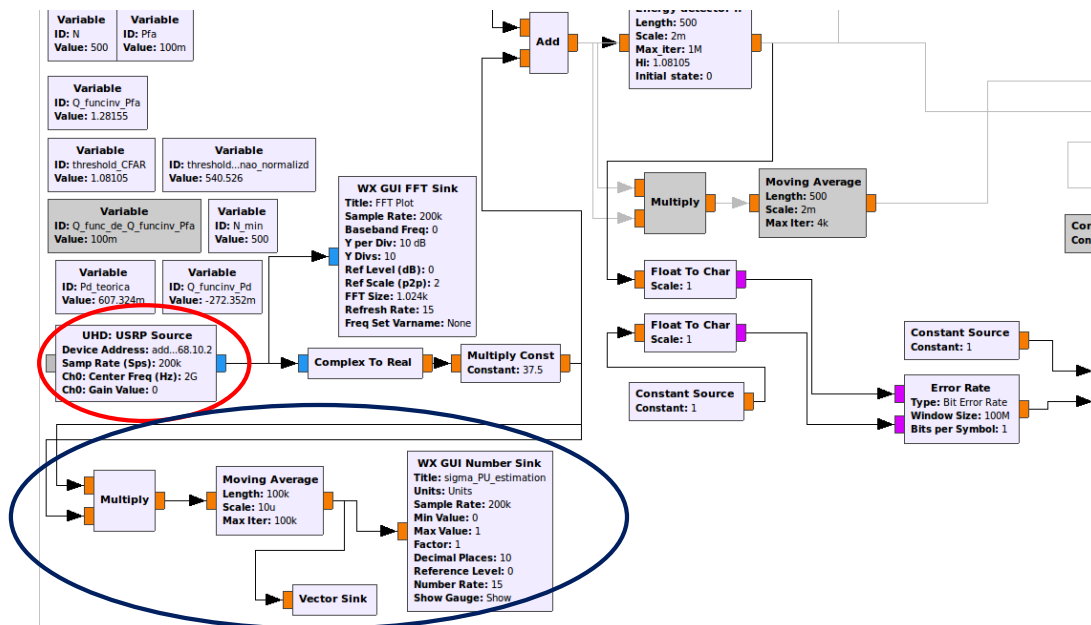


Figura 28 - Fluxograma em RX com a estimativa da energia do sinal recebido

Portanto, um aspecto muito importante na confiabilidade da análise de desempenho do detector é a garantia de que a energia de sinal do transmissor recebida seja muito próxima da energia do sinal gerada localmente nas simulações de forma a permitir uma comparação direta dos resultados das simulações e dos ensaios com os valores teóricos esperados. Essa estimativa da energia do sinal recebido em função de uma dada SNR especificada para o transmissor (por exemplo, -10 dB) é obtida por meio dos blocos *Multiply* e *Moving Average*, que fazem o cálculo da energia do sinal recebido e *Vector Sink* que armazena esses valores de forma a poderem ser utilizados para calibrar o fator de ajuste no bloco *Multiply Const* e garantir que a SNR recebida é muito próxima da SNR especificada. Esse bloco *Multiply Const* encontra-se logo depois do *USRP Source* de forma a obtermos a SNR desejada na detecção.

Sendo assim, no caso do ensaio em uma determinada configuração (por exemplo,  $N=1000$  amostras e  $Pfa=0.1$ ), e com uma SNR especificada em -10 dB, o

componente *TCP Sink* no receptor irá enviar constantemente amostras com o valor -10 (do tipo *float*) para o componente *TCP Source* do transmissor.

O valor dessas amostras é capturado por meio dos blocos *Probe Signal* e *Function Probe*, neste caso com uma taxa de atualização de 10 consultas por segundo (parâmetro *Pool Rate*).

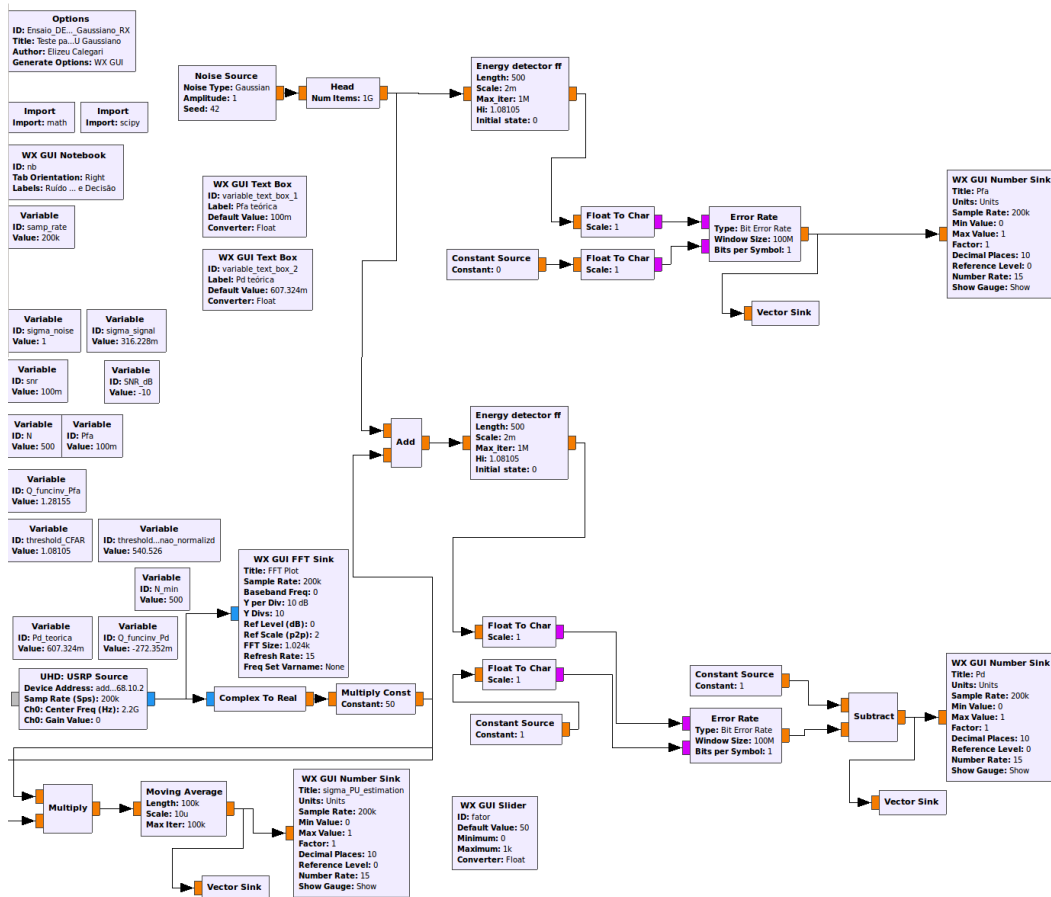


Figura 29 - Fluxograma em RX para análise de desempenho

Agora que foram criados os fluxogramas de geração e transmissão do sinal de usuário primário e de recepção, detecção e análise de desempenho, a descrição de como foram gerados os scripts de automatização desses fluxogramas para efetuar os ensaios está no Anexo E.

#### 7.4. Ensaios realizados

Tendo definido os *scripts* de ensaio, foram realizadas três ensaios para obter as curvas de probabilidade de detecção ( $P_d$ ) em função da relação sinal-ruído (SNR), probabilidade de falso alarme ( $P_{fa}$ ) e número de amostras ( $N$ ), conforme apresentado adiante.

Os *scripts* empregados nos ensaios para os diversos tipos de conexão considerados (cabo coaxial, antenas, etc) são praticamente idênticos, porém com distinção principalmente em fatores como o ganho das USRPs que é maior no caso do uso das antenas, e também o fator de ajuste no bloco *Multiply\_Const* logo depois do bloco *UHD:USRP\_Source*. Os valores empregados tiveram por objetivo controlar a energia do sinal à entrada do detector de forma a que os resultados entre as simulações e ensaios possam ter uma comparação praticamente direta.

Foram consideradas três tipos de condições de conexão entre a USRP transmissora e a USRP receptora:

1. conexão com cabo coaxial e atenuador de 30 dB;
2. conexão com antenas de 5 dBi com blindagem (gaiola de Faraday); e
3. conexão com antenas de 8 dBi.

Tabela 4 - Fator de ajuste e Ganho das USRPs para equivaler SNR

| <i>Conexão</i>                    | <i>Fator de ajuste<br/>(Multiply Const)</i> | <i>USRP Tx<br/>Ganho [dB]</i> | <i>USRP Rx<br/>Ganho [dB]</i> |
|-----------------------------------|---|-------------------------------|-------------------------------|
| Cabo coaxial e atenuador de 30 dB | 565   | 0                             | 0                             |
| Antenas de 5 dBi com blindagem    | 400   | 30                            | 10                            |
| Antenas de 8 dBi                  | 500   | 30                            | 30                            |

## 8 Resultados

### 8.1. Simulações

Apresentamos a seguir os resultados das simulações da probabilidade de detecção em função da relação sinal-ruído, probabilidade de falso alarme e número de amostras, assim como as curvas teóricas obtidas por meio da aproximação gaussiana citada no capítulo 4.

```
eclipse-workspace - Mestrado/Simulacoes/Simula_DE_real_Pd_x_Pfa_com_PU_Gaussiano.py - Eclipse

Project Explorer
Mestrado
├── gr-myblocks
└── Simulacoes
    ├── Ensaio_DE_real_Pd_x_N_com_PU_Gaussiano_RX.py
    ├── Ensaio_DE_real_Pd_x_Pfa_com_PU_Gaussiano_RX.py
    ├── Ensaio_DE_real_Pd_x_SNR_com_PU_Gaussiano_RX.py
    ├── Simula_DE_real_Pd_x_N_com_PU_Gaussiano.py
    └── Simula_DE_real_Pd_x_Pfa_com_PU_Gaussiano.py
python (/usr/bin/python)

Simula_DE_real_Pd_x_Pfa_com_PU_Gaussiano
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 #####
4 # GNU Radio Python Flow Graph
5 # Title: Teste para DE no dominio doTempo
6 # Author: Elizeu Calegari
7 # Generated: Tue Jan 9 11:00:47 2018
8 #####
9
10 if __name__ == '__main__':
11     import ctypes
12     import sys
13     if sys.platform.startswith('linux'):
14         try:
15             x11 = ctypes.cdll.LoadLibrary(
16
<terminated> Simula_DE_real_Pd_x_Pfa_com_PU_Gaussiano.py [/usr/bin/python]
===== Simulando ... =====

SNR_dB      N      => Pfa teo  Pfa sim  Pd teo    Pd sim
Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.010000, 0.015887, 0.012764, 0.019040

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.020000, 0.027160, 0.024800, 0.032697

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.050000, 0.057910, 0.059551, 0.067677
```

Figura 30 - Exemplo de simulação no Eclipse para traçar Pd x Pfa

Além da curva da probabilidade de detecção em função da relação sinal-ruído (SNR) e utilizando a estrutura criada para se fazer análise de desempenho, também foram realizadas estimativas para a probabilidade de falso alarme, de forma a averiguar se o comportamento está conforme o previsto no modelo teórico onde o

limiar é definido em função de um critério CFAR, ou seja, de se atingir uma dada taxa de falso alarme constante.

Podemos observar que as estimativas para a probabilidade de falso alarme obtidas pelas simulações encontram-se muito próximas do valor nominal esperado ( $P_{fa}=0.1$ ).

### 8.1.1. Pd x SNR

Nesta simulação foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa}=0.1$  e obtidas cinco curvas da probabilidade de detecção em função da relação sinal-ruído para  $N=100, 1000, 2500, 5000$  e  $10000$  amostras.

```

Pfa    = [0.1]
N      = [100, 1000, 2500, 5000, 10000]
SNR_dB = [-25.0, -22.5, -20.0, -19.0, -18.0, -17.0, -16.0, -15.0, -14.0,
          -13.0, -12.0, -11.0, -10.0, -9.0, -7.0, -5.0]

```

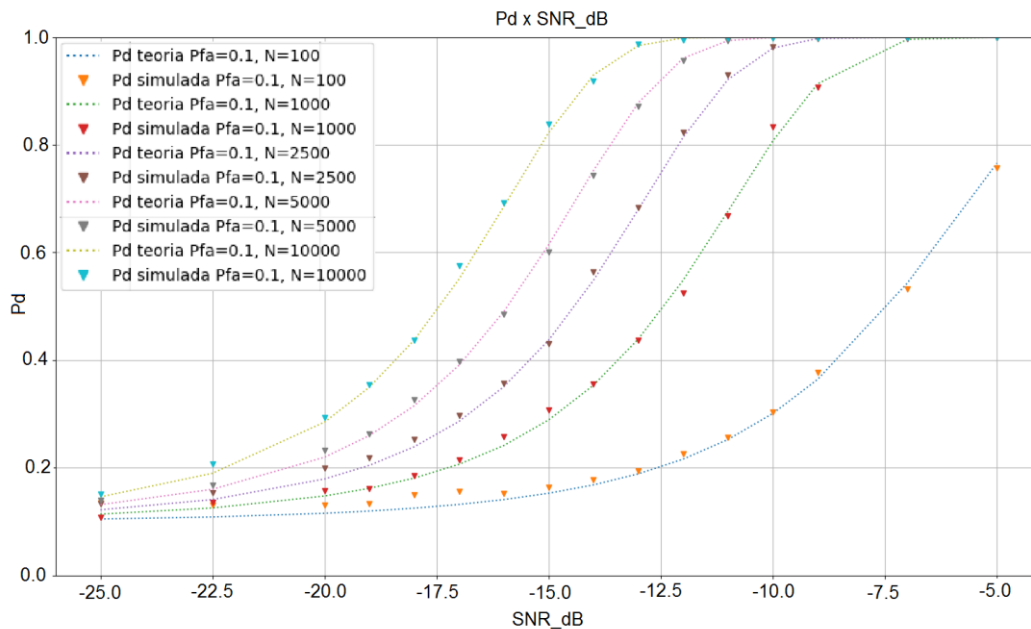


Figura 31 - Pd x SNR para  $N=100, 1000, 2500, 5000$  e  $10000$

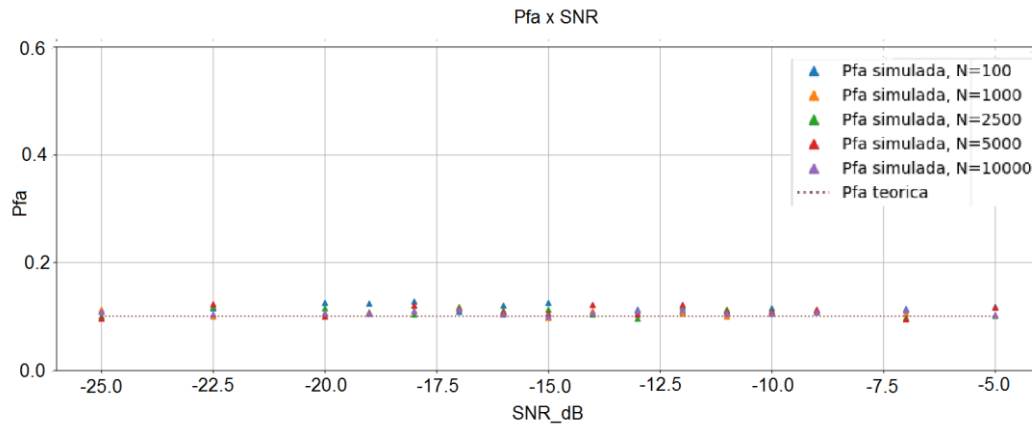


Figura 32 - Pfa x SNR para N=100, 1000, 2500, 5000 e 10000

### 8.1.2. Pd x Pfa

Nesta simulação foram considerados cenários apenas para uma dada relação sinal-ruído (-10 dB) e obtidas quatro curvas da probabilidade de detecção em função da probabilidade de falso alarme para N=100, 1000, 2500, 5000 e 10000 amostras.

```
SNR_dB = [-10.0]
N       = [100, 1000, 2500, 5000, 10000]
Pfa     = [0.01, 0.02, 0.03, 0.04, 0.05, 0.07, 0.1, 0.15, 0.2, 0.3, 0.4,
           0.5, 0.6, 0.7, 0.8, 0.9, 0.9999]
```

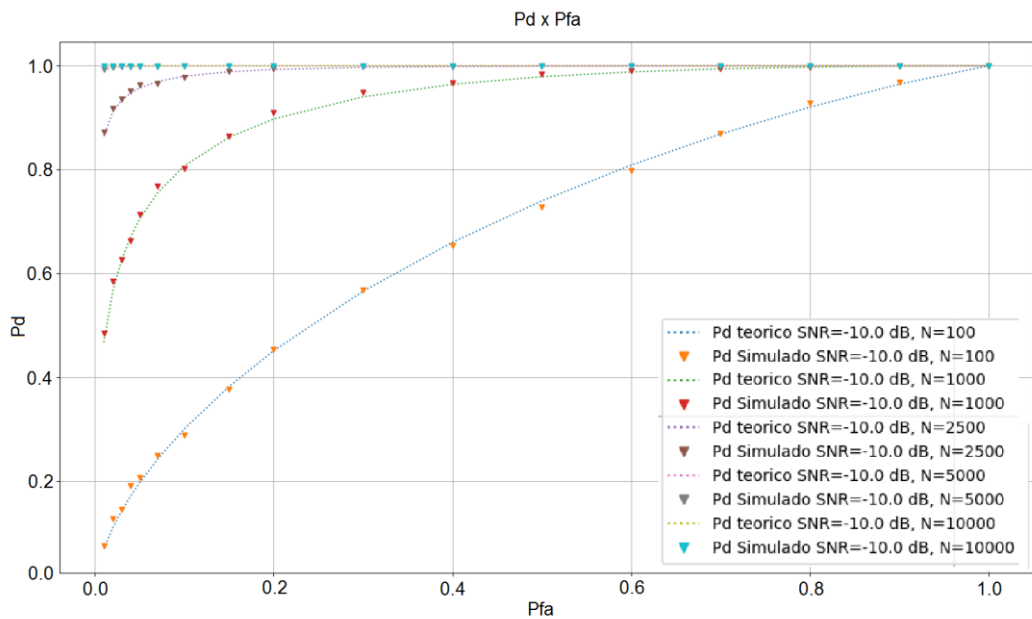


Figura 33 - Pd x Pfa para N=100, 1000, 2500, 5000 e 10000

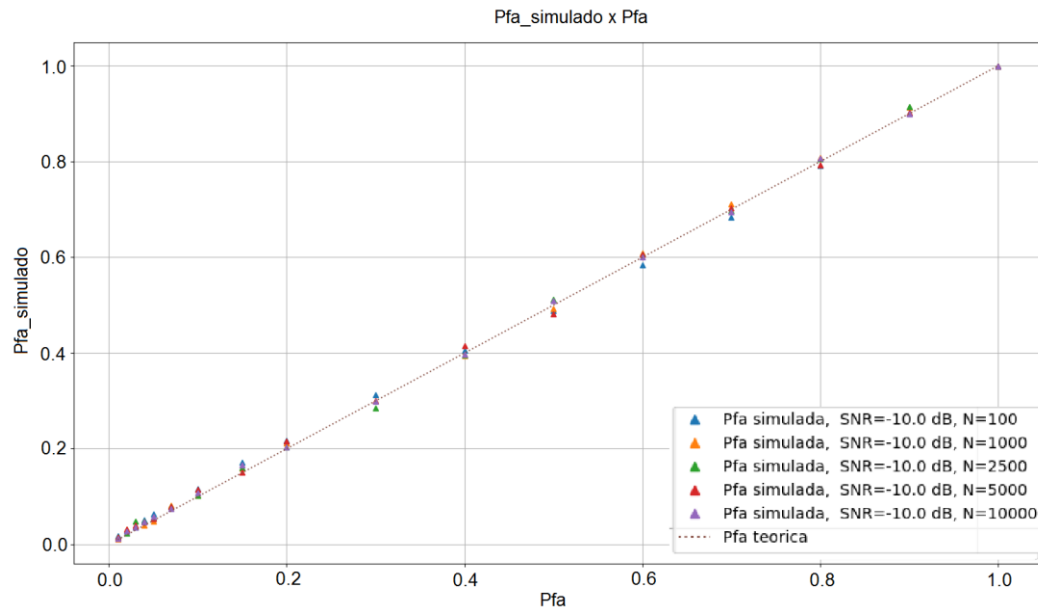


Figura 34 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000

### 8.1.3. Pd x N

Nesta simulação foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa}=0.1$  e obtidas seis curvas da probabilidade de detecção em função do número de amostras para SNR [dB] = -20.0, -15.0, -12.5, -10.0, -7.5 e -5.0.

```
Pfa      = [0.1]
SNR_dB   = [-20.0, -15.0, -12.5, -10.0, -7.5, -5.0]
N        = [100, 250, 500, 750, 1000, 1500, 2000, 3000, 5000, 10000]
```

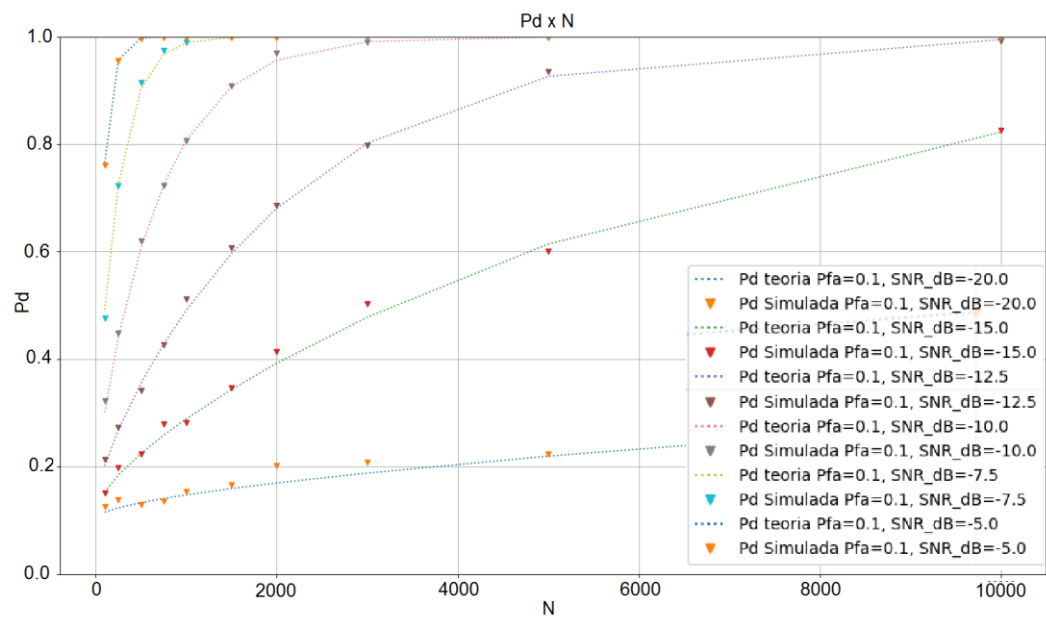


Figura 35 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB

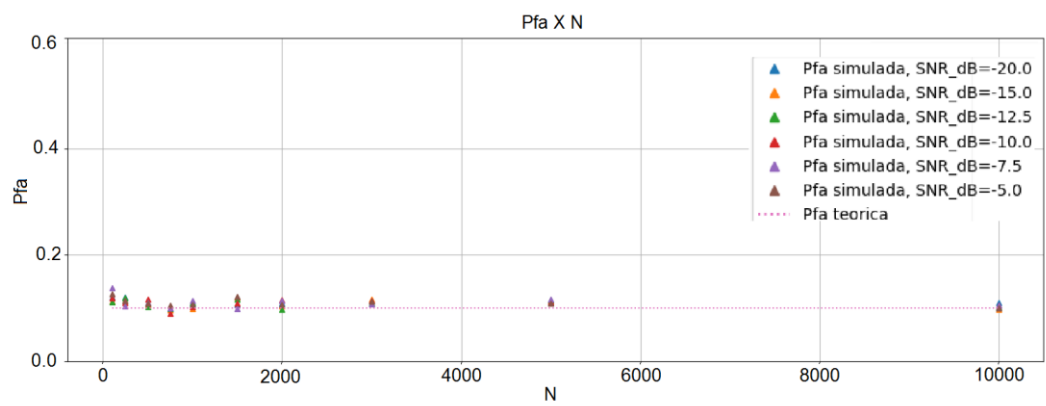


Figura 36 - Pfa x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB



## 8.2. Ensaio com as USRP

Assim como no caso das simulações, apresentamos a seguir os resultados dos ensaios da probabilidade de detecção em função da relação sinal-ruído, probabilidade de falso alarme e número de amostras, assim como as curvas teóricas obtidas.

Conforme já citado, foram realizadas configurações de ensaio considerando os seguintes tipos de conexão entre as USRPs:

1. conexão com cabo coaxial e atenuador de 30 dB;
2. conexão com antenas de 5 dBi com blindagem (gaiola de Farady); e
3. conexão com antenas de 8 dBi.



Figura 37 - Montagens de ensaio com conexões entre as USRP

8.2.1.  
Ensaio via cabo coaxial e atenuador de 30 dB

8.2.1.1.  
Pd x SNR

Neste ensaio foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa}=0.1$  e obtidas cinco curvas da probabilidade de detecção em função da relação sinal-ruído para  $N=100, 1000, 2500, 5000$  e  $10000$ .

|        |   |   |
|--------|---|---|
| Pfa    | = | [0.1]   |
| N      | = | [100, 1000, 2500, 5000, 10000]  |
| SNR_dB | = | [-25.0, -22.5, -20.0, -19.0, -18.0, -17.0, -16.0, -15.0, -14.0, -13.0, -12.0, -11.0, -10.0, -9.0, -7.0, -5.0] |

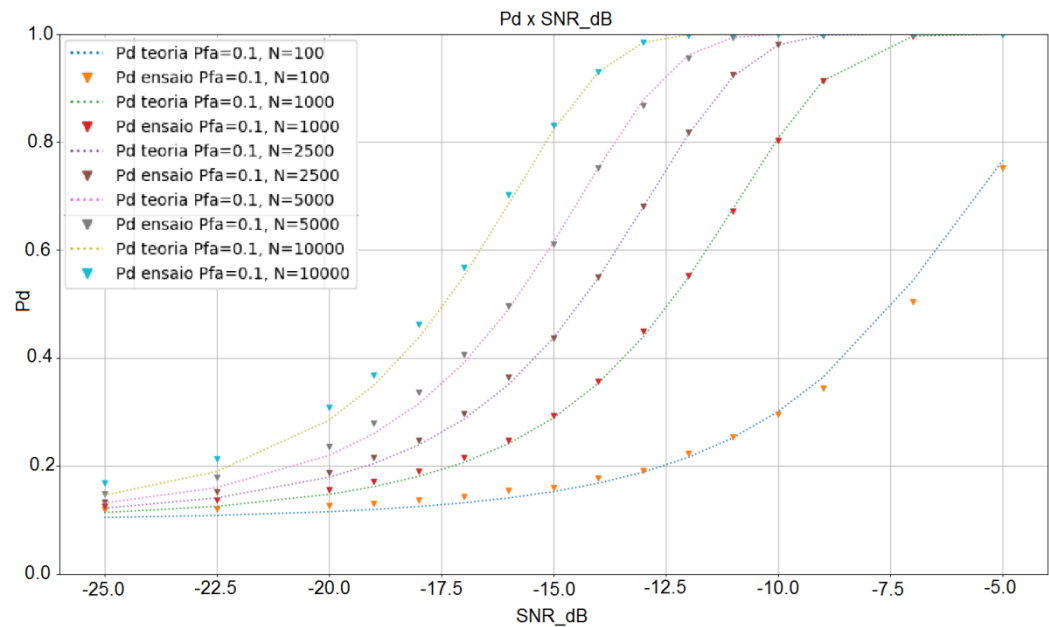


Figura 38 - Pd x SNR com N=100, 1000, 2500, 5000 e 10000

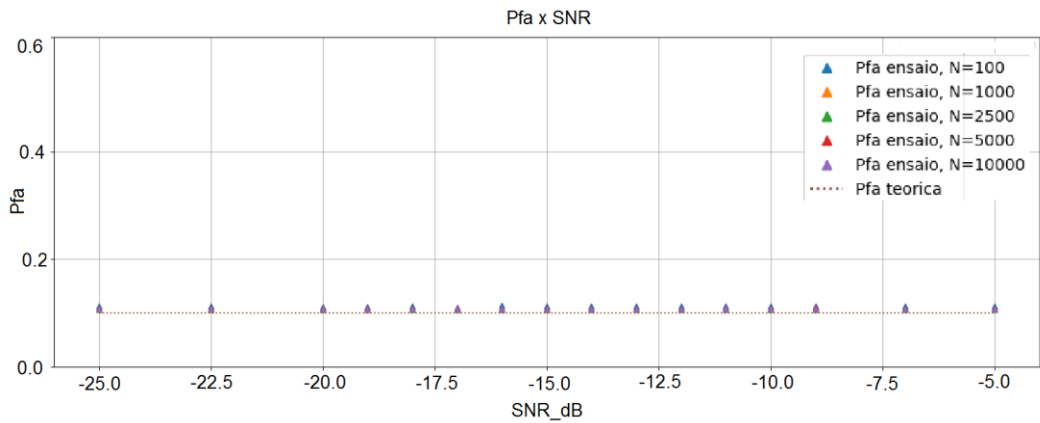


Figura 39 - Pfa x SNR com N=100, 1000, 2500, 5000 e 10000

8.2.1.2.  
Pd x Pfa

Neste ensaio foram considerados cenários apenas para uma dada relação sinal-ruído (-10 dB) e obtidas quatro curvas da probabilidade de detecção em função da probabilidade de falso alarme para N=100, 1000, 2500, 5000 e 10000 amostras.

```
SNR_dB = [-10.0]
N       = [100, 1000, 2500, 5000, 10000]
Pfa     = [0.01, 0.02, 0.03, 0.04, 0.05, 0.07, 0.1, 0.2, 0.3, 0.4,
           0.5, 0.6, 0.7, 0.8, 0.9, 0.9999]
```

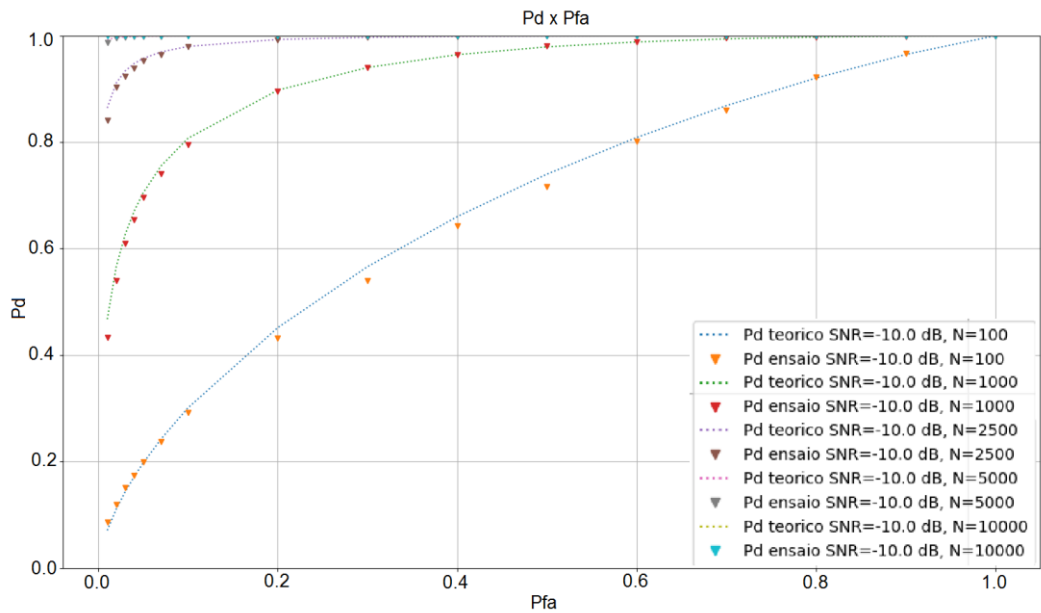


Figura 40 - Pd x Pfa com N=100, 1000, 2500, 5000 e 10000

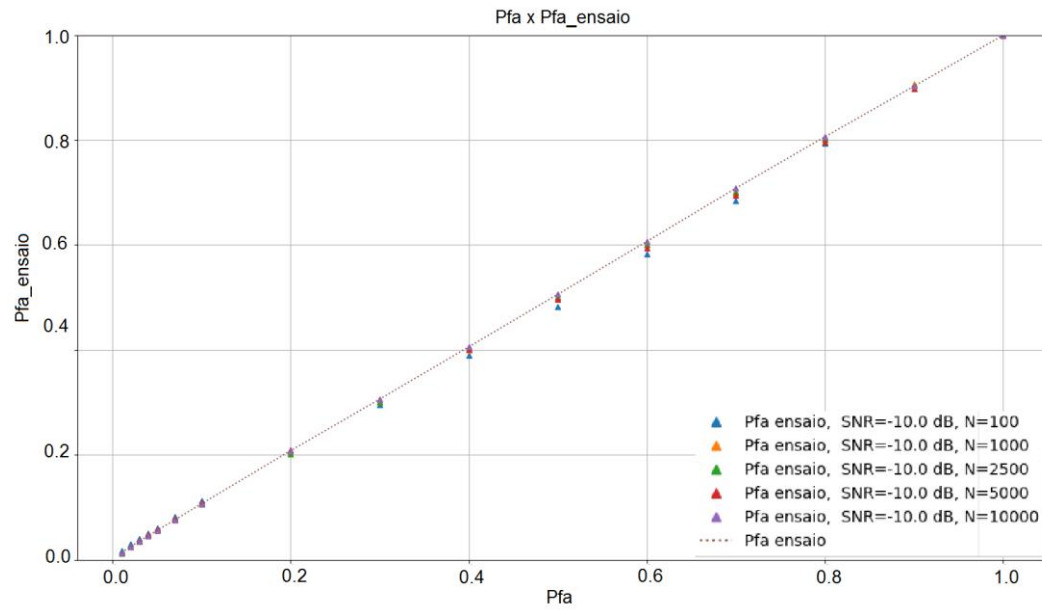


Figura 41 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000

### 8.2.1.3. Pd x N

Neste ensaio foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa} = 0.1$  e obtidas seis curvas da probabilidade de detecção em função do número de amostras para  $SNR [dB] = -20.0, -15.0, -12.5, -10.0, -7.5$  e  $-5.0$ .

|            |  |
|------------|--|
| $P_{fa}$   | = [0.1]  |
| $SNR_{dB}$ | = [-20.0, -15.0, -12.5, -10.0, -7.5, -5.0]       |
| $N$        | = [100, 500, 750, 1000, 2000, 3000, 5000, 10000] |

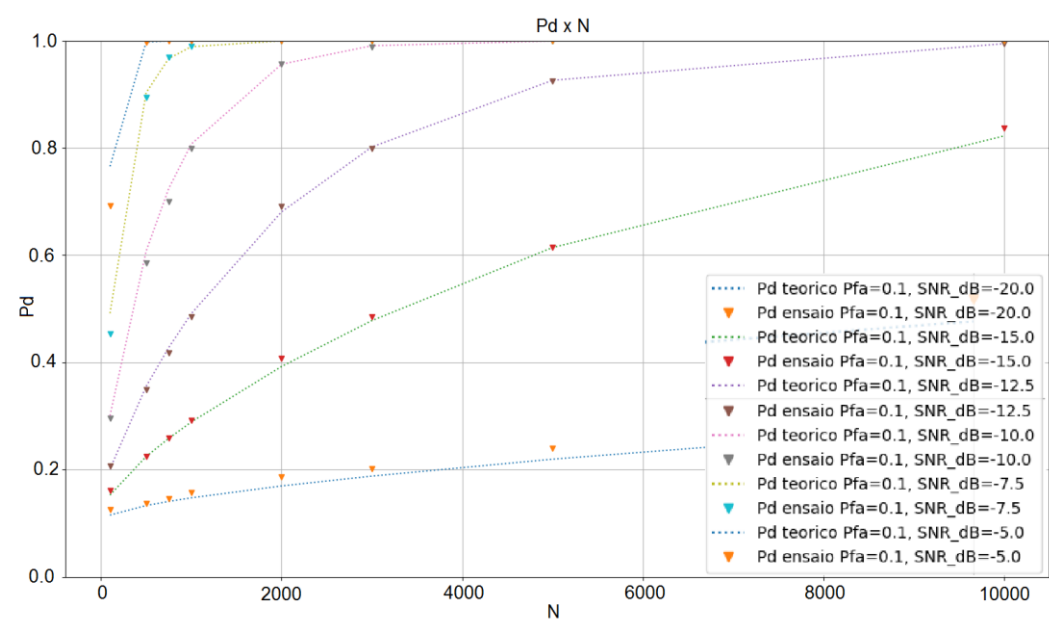


Figura 42 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB

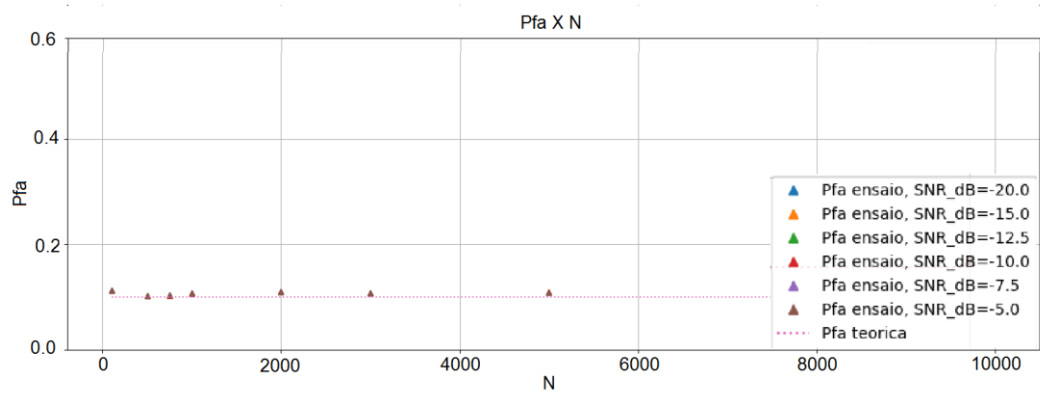


Figura 43 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB

## 8.2.2.

## Ensaio via antenas de 5 dBi e blindagem (gaiola de Faraday)

## 8.2.2.1.

## Pd x SNR

Neste ensaio foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa}=0.1$  e obtidas cinco curvas da probabilidade de detecção em função da relação sinal-ruído para  $N=100, 1000, 2500, 5000$  e  $10000$ .

```

Pfa    = [0.1]
N       = [100, 1000, 2500, 5000, 10000]
SNR_dB = [-25.0, -22.5, -20.0, -19.0, -18.0, -17.0, -16.0, -15.0, -14.0,
          -13.0, -12.0, -11.0, -10.0, -9.0, -7.0, -5.0]

```

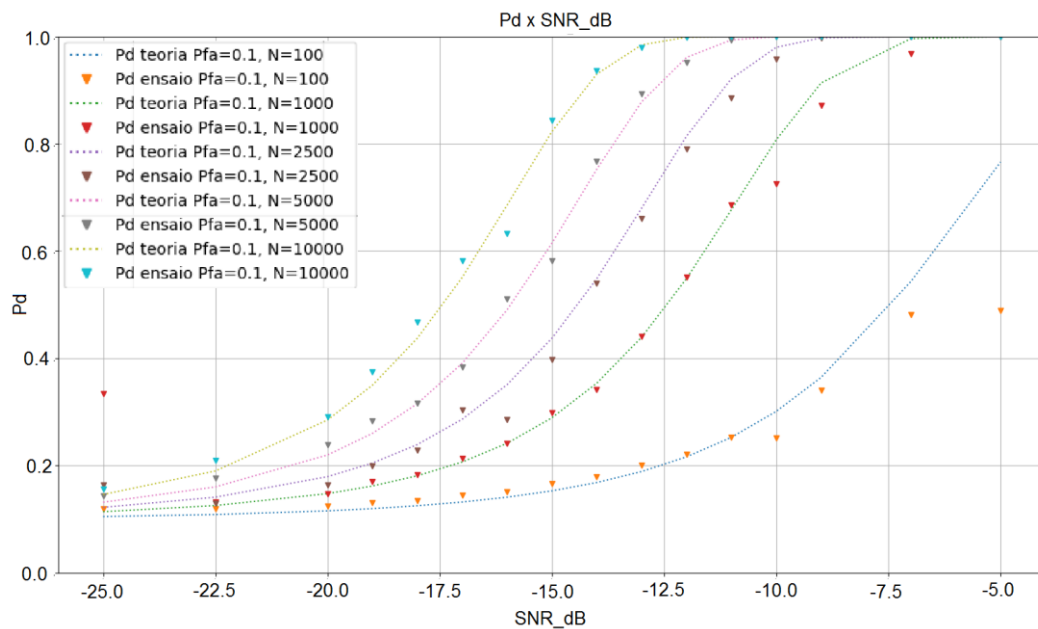


Figura 44 - Pd x SNR com N=100, 1000, 2500, 5000 e 10000

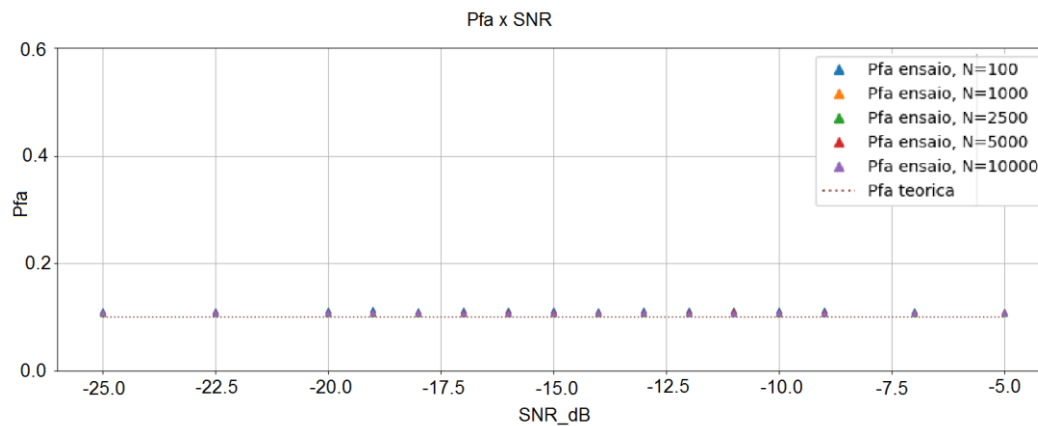


Figura 45 - Pfa x SNR com N=100, 1000, 2500, 5000 e 10000

8.2.2.2.  
Pd x Pfa

Neste ensaio foram considerados cenários apenas para uma dada relação sinal-ruído (-10 dB) e obtidas quatro curvas da probabilidade de detecção em função da probabilidade de falso alarme para N=100, 1000, 2500, 5000 e 10000 amostras.

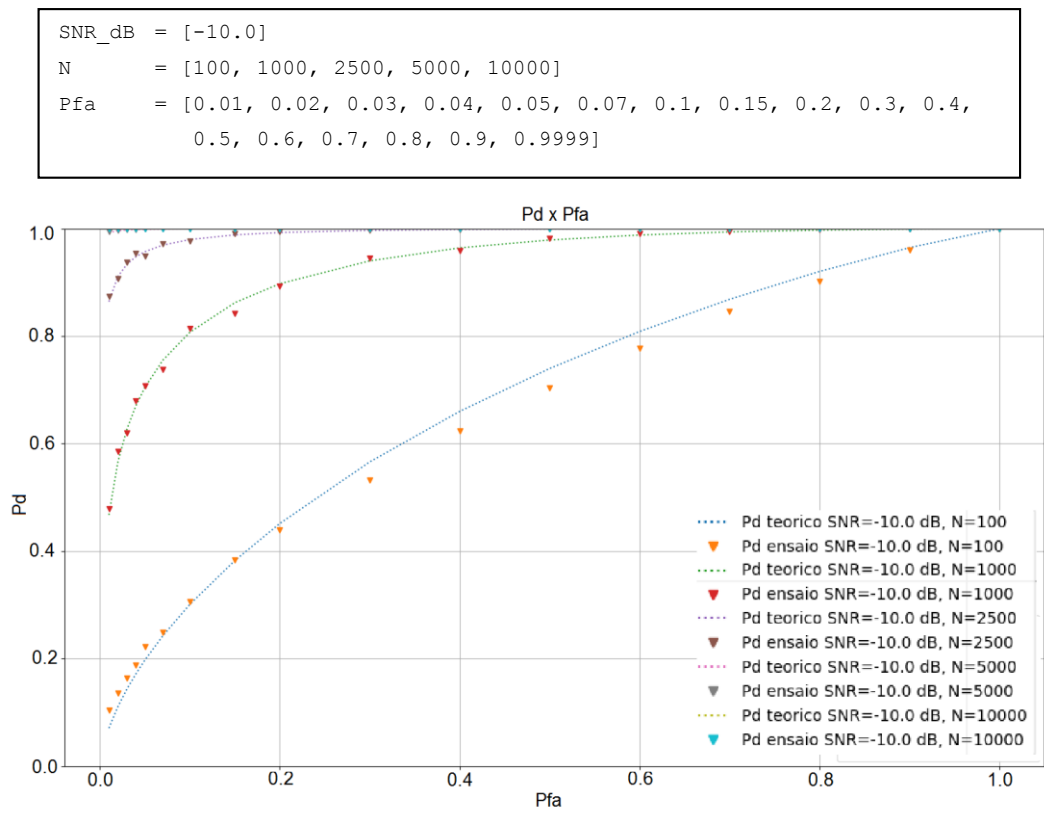


Figura 46 - Pd x Pfa para N=100, 1000, 2500, 5000 e 10000

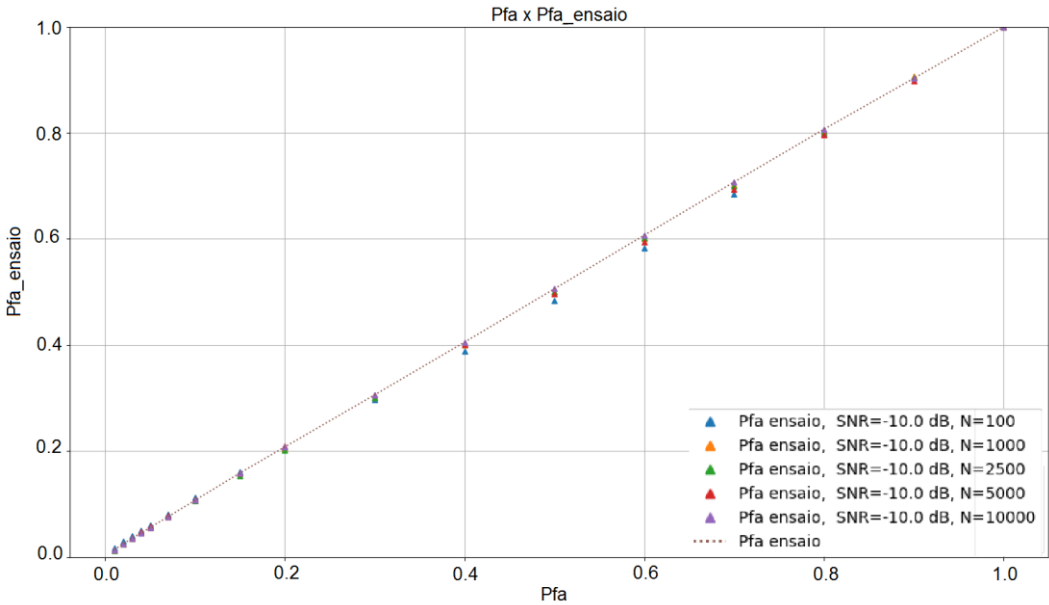


Figura 47 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000

8.2.2.3.  
Pd x N

Neste ensaio foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa} = 0.1$  e obtidas seis curvas da probabilidade de detecção em função do número de amostras para  $SNR [dB] = -20.0, -15.0, -12.5, -10.0, -7.5$  e  $-5.0$ .

|        |   |   |
|--------|---|---|
| Pfa    | = | [0.1]   |
| SNR_dB | = | [-20.0, -15.0, -12.5, -10.0, -7.5, -5.0]                  |
| N      | = | [100, 250, 500, 750, 1000, 1500, 2000, 3000, 5000, 10000] |



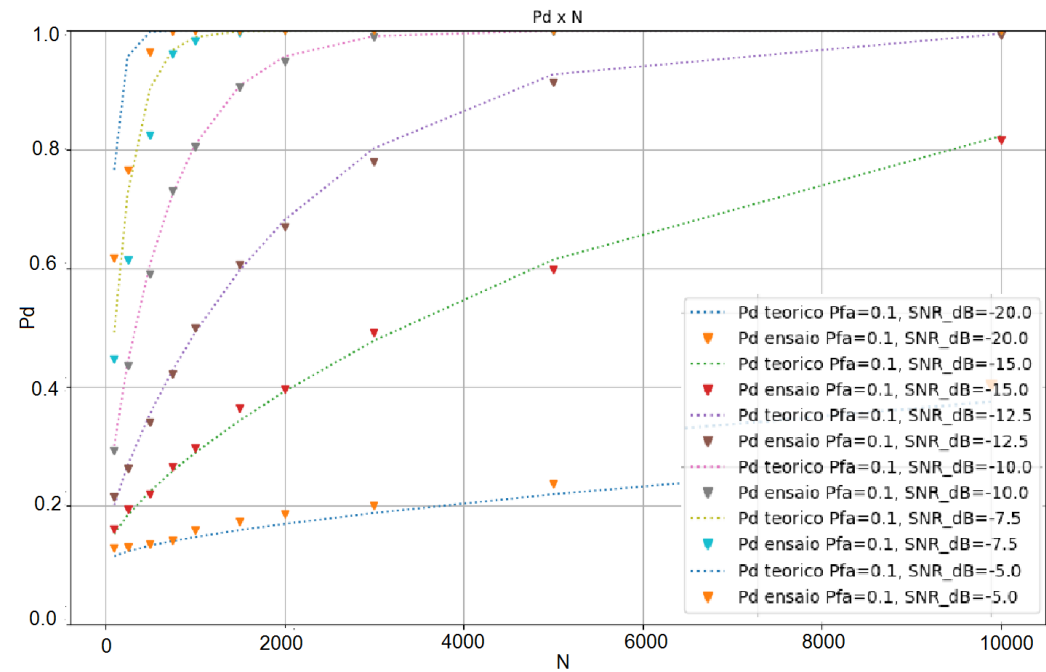


Figura 48 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB

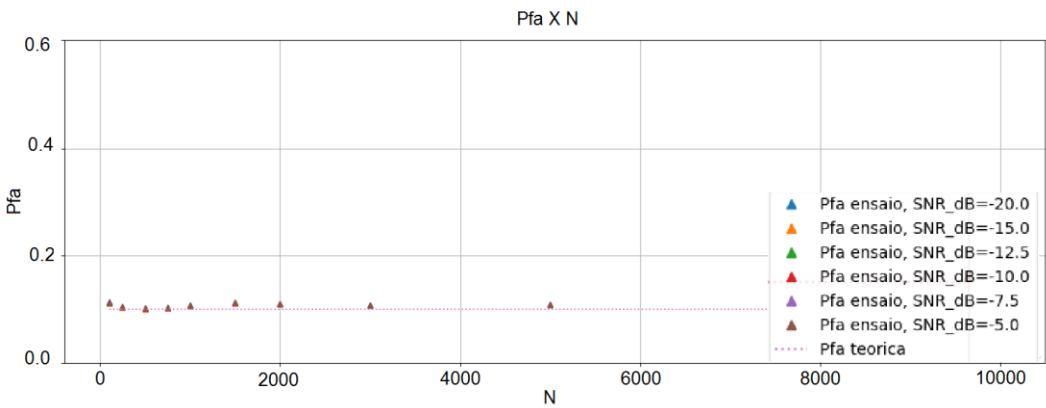


Figura 49 - Pd x N com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB

8.2.3.  
Ensaio via antenas de 8 dBi

8.2.3.1.  
Pd x SNR

Neste ensaio foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa}=0.1$  e obtidas cinco curvas da probabilidade de detecção em função da relação sinal-ruído para  $N=100, 1000, 2500, 5000$  e  $10000$ .

```
Pfa      = [0.1]
N        = [100, 1000, 2500, 5000, 10000]
SNR_dB   = [-25.0, -22.5, -20.0, -19.0, -18.0, -17.0, -16.0, -15.0, -14.0,
            -13.0, -12.0, -11.0, -10.0, -9.0, -7.0, -5.0]
```

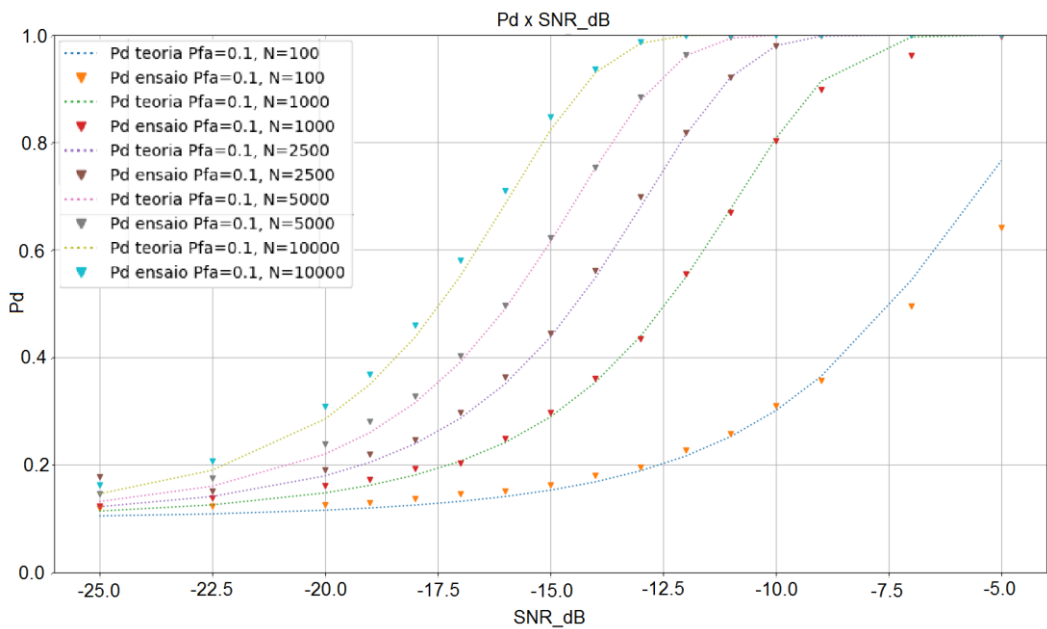


Figura 50 - Pd x SNR com  $N=100, 1000, 2500, 5000$  e  $10000$

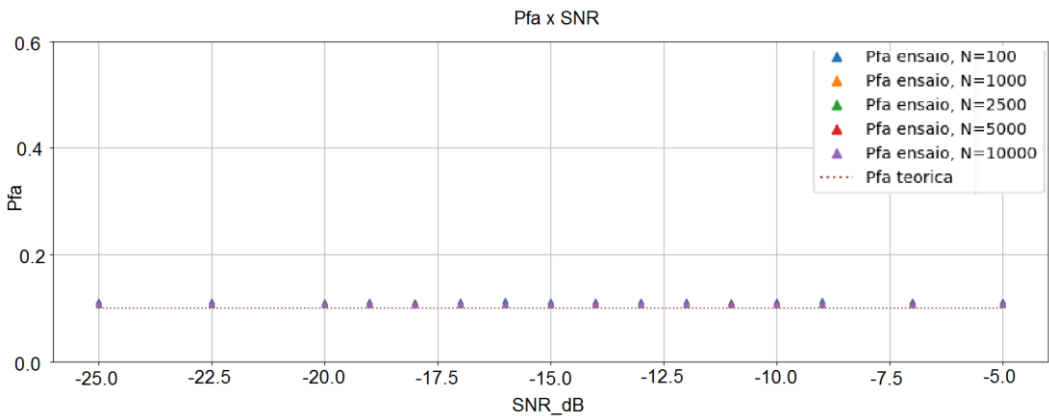


Figura 51 - Pfa x SNR com N=100, 1000, 2500, 5000 e 10000

8.2.3.2.  
Pd x Pfa

Neste ensaio foram considerados cenários apenas para uma dada relação sinal-ruído (-10 dB) e obtidas quatro curvas da probabilidade de detecção em função da probabilidade de falso alarme para N=100, 1000, 2500, 5000 e 10000 amostras.

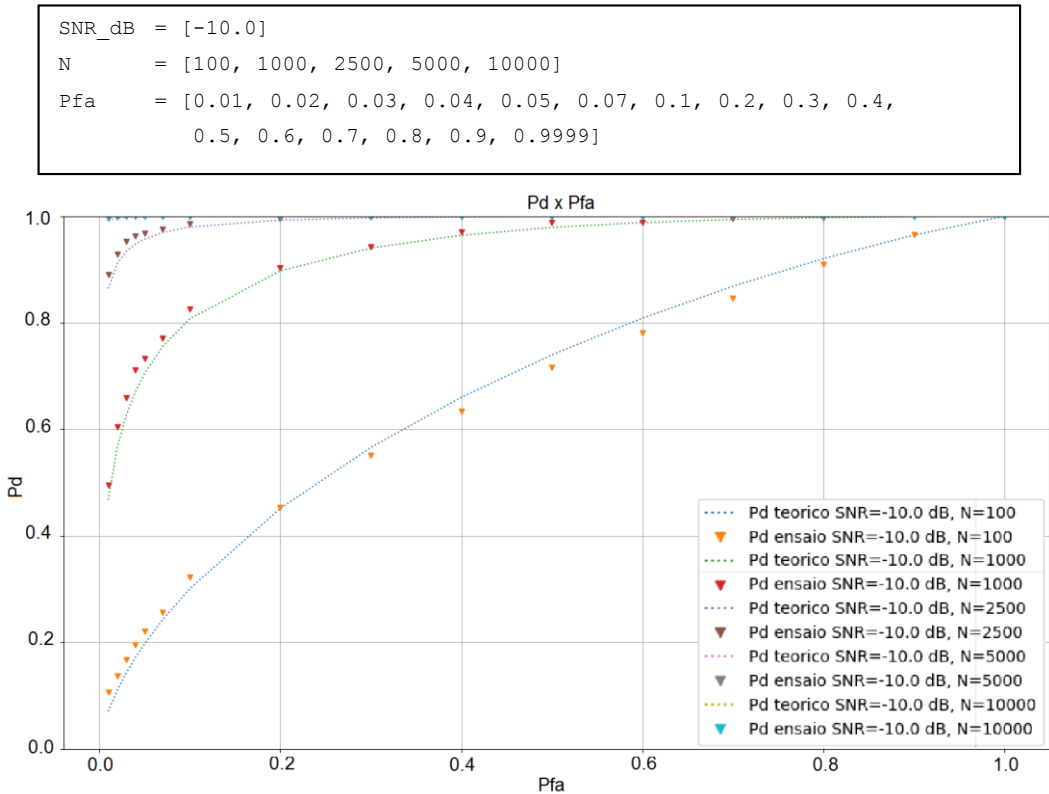


Figura 52 - Pd x Pfa com N=100, 1000, 2500, 5000 e 10000

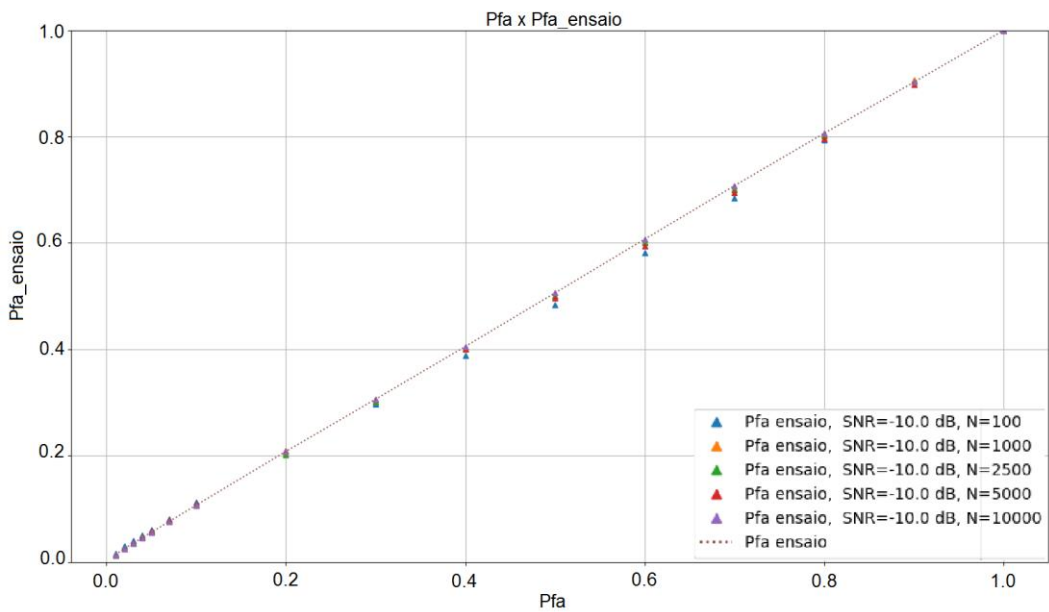
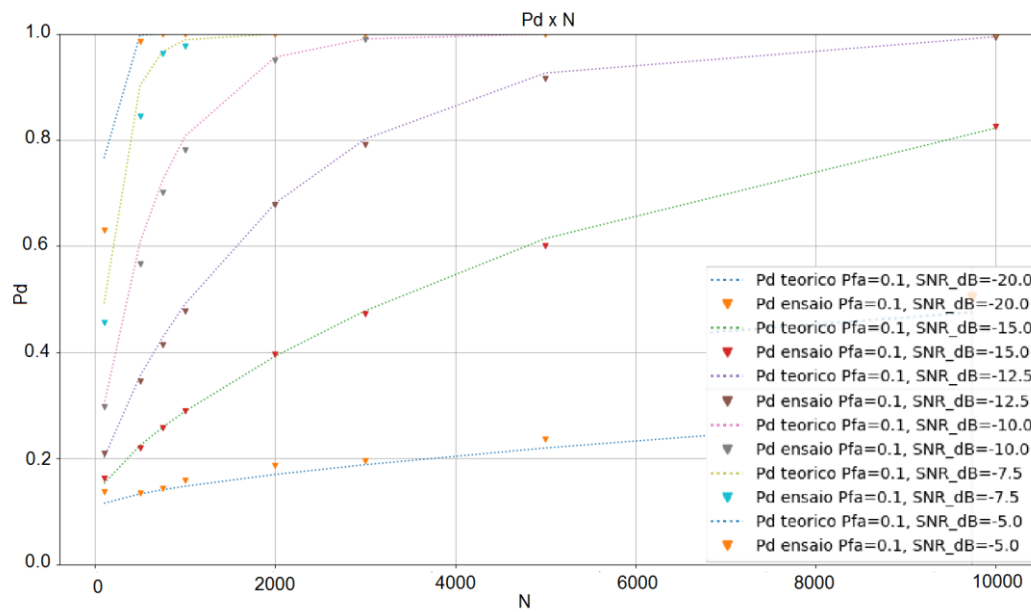
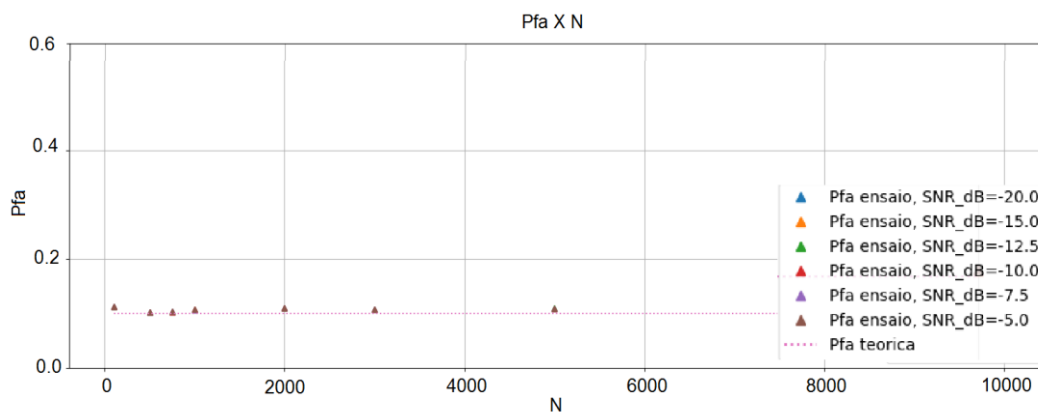


Figura 53 - Pfa x Pfa nominal com N=100, 1000, 2500, 5000 e 10000

8.2.3.3.  
Pd x N

Neste ensaio foram considerados cenários apenas para uma dada probabilidade de falso alarme  $P_{fa}=0.1$  e obtidas seis curvas da probabilidade de detecção em função do número de amostras para  $SNR [dB] = -20.0, -15.0, -12.5, -10.0, -7.5$  e  $-5.0$ .

```
Pfa      = [0.1]
SNR_dB   = [-20.0, -15.0, -12.5, -10.0, -7.5, -5.0]
N        = [100, 500, 750, 1000, 2000, 3000, 5000, 10000]
```

Figura 54 -  $P_d$  x  $N$  com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dBFigura 55 -  $P_d$  x  $N$  com SNR=-20.0, -15.0, -12.5, -10.0, -7.5 e -5.0 dB

### 8.3.

#### Análise dos resultados das simulações e ensaios

Os resultados apresentados até ao momento vieram demonstrar de certa forma a concordância esperada entre as curvas teóricas e todos os resultados obtidos, tanto nas simulações quanto nos ensaios apresentados, validando de certa forma os resultados obtidos.

Por exemplo, em todos os resultados obtidos pudemos observar comportamentos como o da simulação do detector de energia com  $\text{SNR}=-15\text{dB}$  e  $N=500$ , 1000 e 1500 amostras exibido na Figura 56, onde podemos perceber claramente que o desempenho ( $P_d$ ) cresce de forma monótona com a tolerância ao alarme falso, e também uma melhora do desempenho ( $P_d$ ) conforme aumentamos o número de amostras.

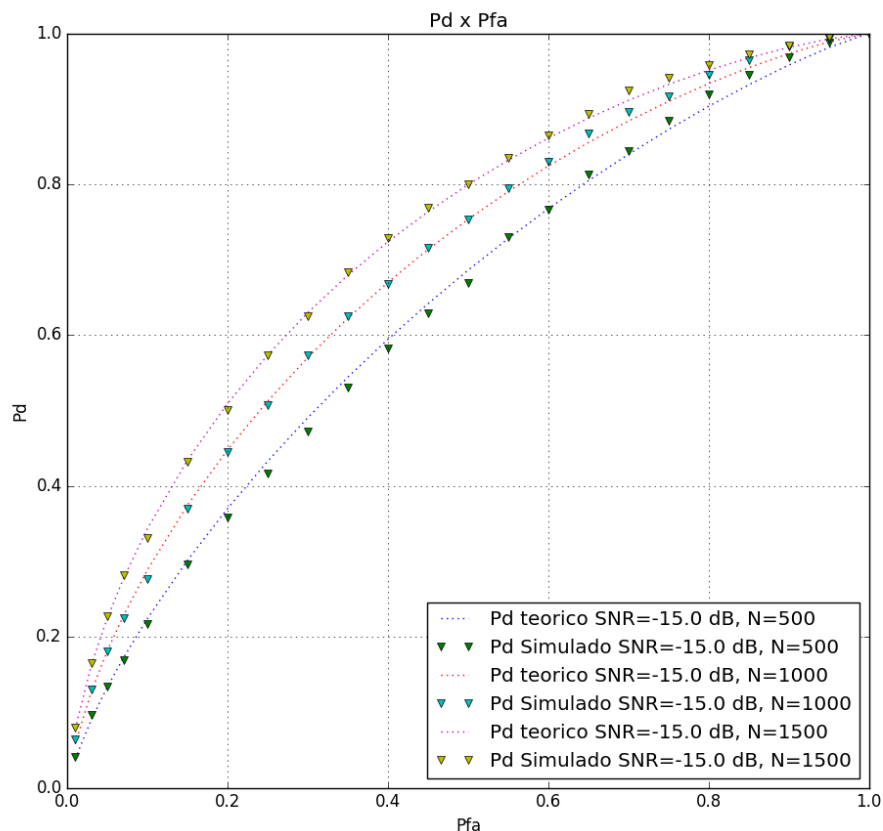


Figura 56 - Simulação  $P_d$  x  $P_{fa}$  com  $\text{SNR}=-15\text{ dB}$  e  $N=500$ , 1000 e 1500

Por outro lado, na Figura 57 podemos perceber que a probabilidade de falso alarme se mantém sob controle dentro dos valores esperados durante o varrimento na faixa de 0.01 até 1.0, inclusive independentemente do número de amostras especificado, conforme era esperado.

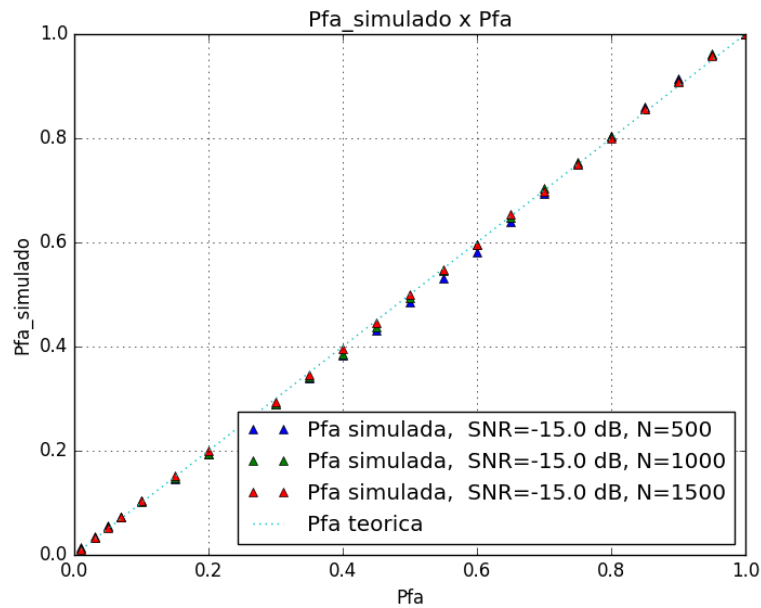


Figura 57 - Pfa x Pfa nominal com SNR=-15 dB e N=500, 1000 e 1500

Na Figura 58 podemos ver também que a detecção melhora com o aumento da relação sinal ruído, e na Figura 59 vemos que a Pfa permanece independente da SNR especificada, conforme previsto no critério CFAR para a determinação do limiar.

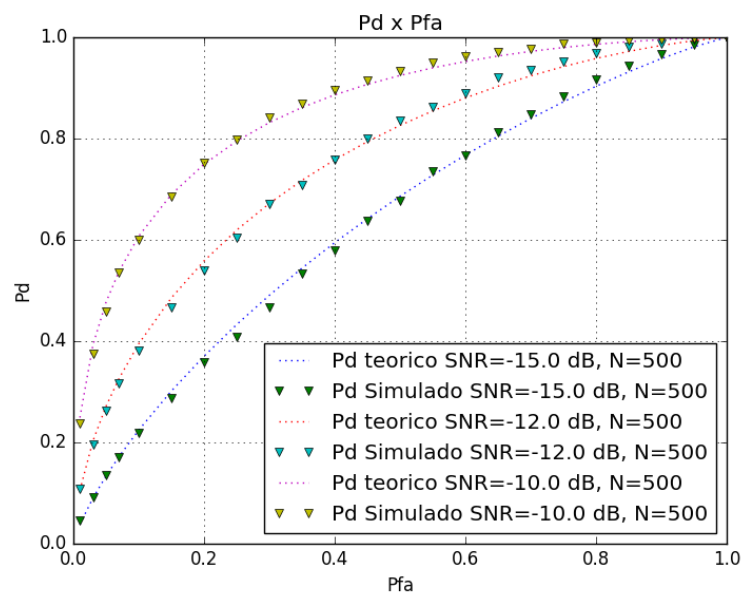


Figura 58 - Simulação Pd x Pfa para N=500 e SNR=-15, -12 e -10 dB

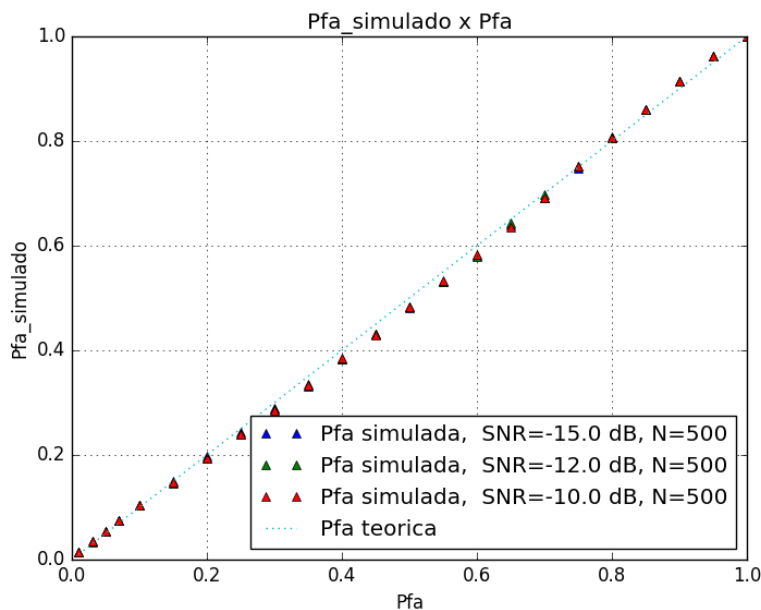


Figura 59 - Pfa x Pfa nominal para N=500 e SNR=-15, -12 e -10 dB

O padrão 802.22 por outro lado, conforme já citado no Capítulo 2 e detalhado na Tabela 1, especifica uma probabilidade de detecção mínima de 0.9 e uma tolerância para o falso alarme de até 10% a uma SNR de até -21 dB para TV digital.

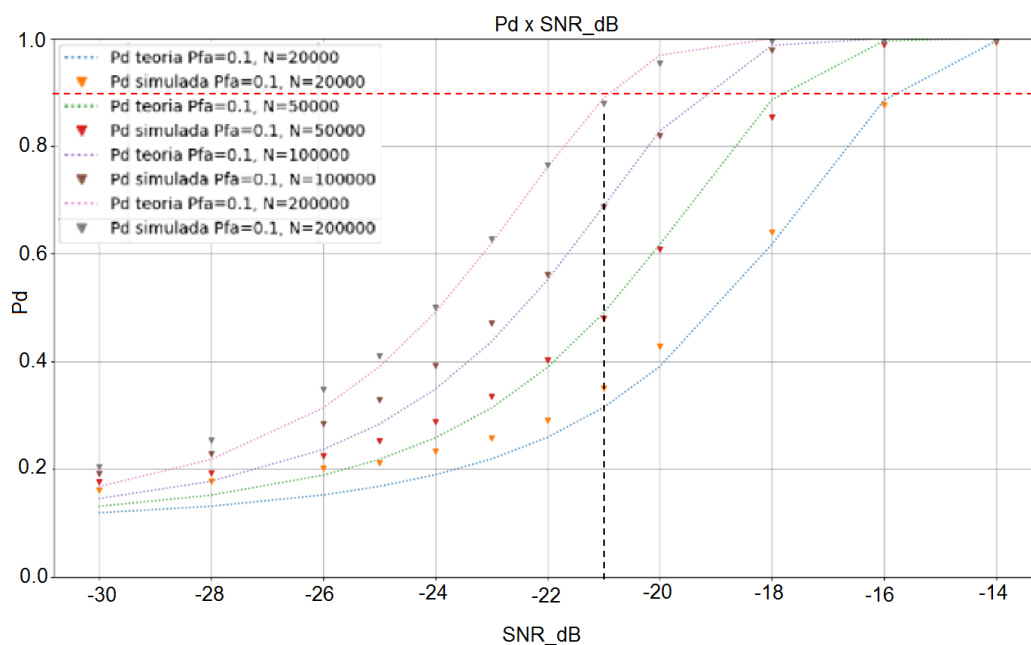


Figura 60 - Pd x SNR para N=20000, 50000, 100000 e 200000 (Pfa=0.1)



Na Figura 60 podemos observar que nenhuma das curvas consegue atingir esses requisitos especificados na IEEE 802.22, e a única que se aproxima de  $P_d=0.9$  para uma SNR de -21dB é a curva referente a 200 mil amostras, mas não chega a atingir o desempenho especificado.

Podemos também observar pelo gráfico de  $P_d$  x SNR na Figura 61 que, se considerarmos uma Pfa alvo de 0.1, esse resultado só é obtido para valores de N a partir de 250 mil amostras, aproximadamente. Por outro lado, se formos ainda mais restritivos para a tolerância ao alarme falso especificando, por exemplo, uma probabilidade de falso alarme de no máximo 1%, precisaremos praticamente dobrar o número de amostras para valores da ordem de 500 mil amostras para atingir tal desempenho.

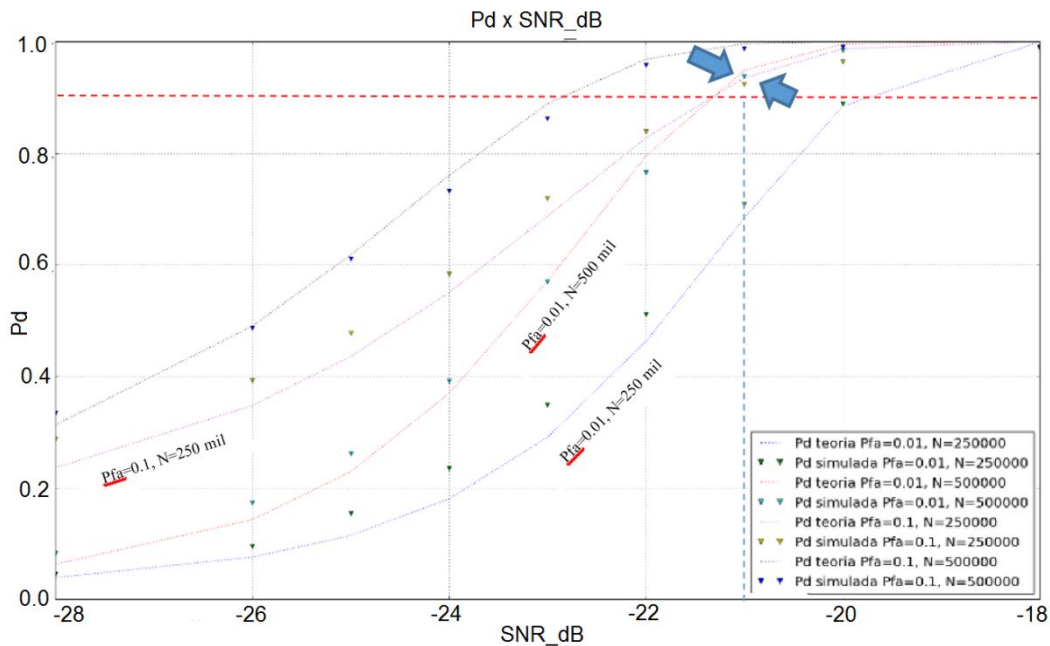


Figura 61 -  $P_d$  x SNR com  $N=250000$ ,  $500000$  com  $P_{fa}=0.01$  e  $0.1$

Esses resultados também podem ser observados nos gráfico de  $P_d$  x N (na curva para uma SNR de -21 dB) apresentado na Figura 62, na qual podemos ver que, tendo requerido uma  $P_{fa}=0.1$ , conforme requisito da IEEE 802,22, a curva da probabilidade de detecção só ultrapassa 0.9 para valores de N acima de 250 mil amostras, aproximadamente. Caso a Pfa seja ainda mais restritiva, com  $P_{fa}=0.01$ , podemos ver que o número de amostras necessário para atingir esse cenário praticamente dobra para valores da ordem de 500 mil amostras.

Tendo atingido o critério de desempenho da IEE802.22 (probabilidade de detecção de 90% com Pfa de 10%) com N da ordem de 250 mil amostras, o tempo de aquisição vai depender claramente da capacidade de processamento do detector no âmbito da amostragem e processamento da detecção (cálculo da média móvel com 250 mil amostras e decisão), o que para o tempo máximo de detecção de 2 segundos estipulado pelo padrão 802.22 requer uma capacidade de amostragem bem superior a 125 mil amostras por segundo.

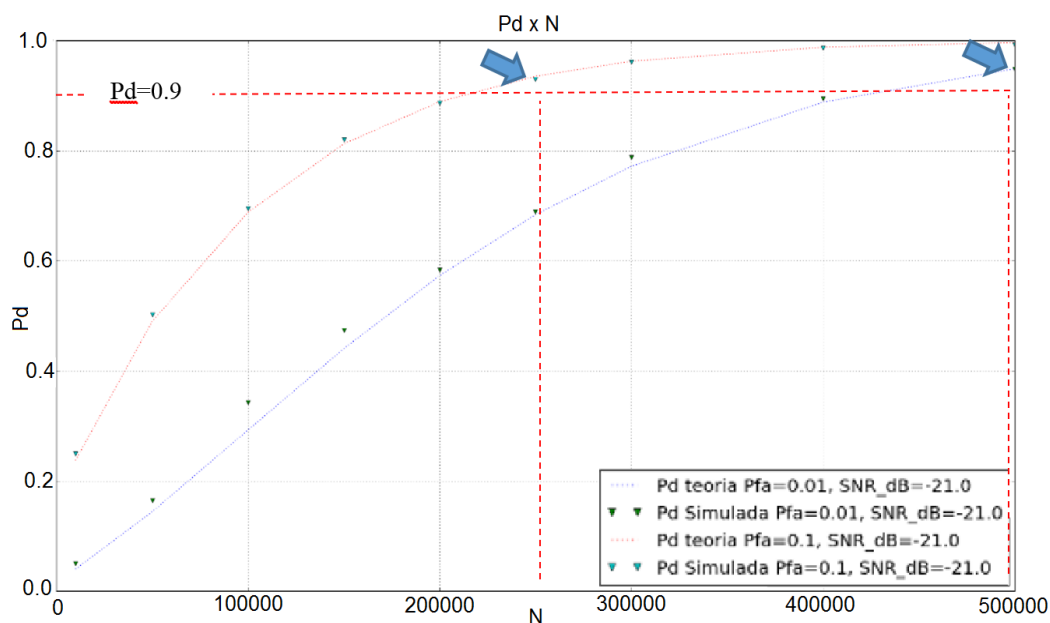


Figura 62 -  $P_d$  x  $N$  para  $SNR = -21$  dB e  $P_{fa} = 0.1$  e  $0.01$

Podemos também observar no Gráfico de  $P_d$  x  $N$  da Figura 62 que a probabilidade de detecção aumenta de forma monótona com o necessário de amostras para atingir esse desempenho ( $P_d$ ), e que, quanto mais restritiva é a exigência de falso alarme, ou seja, se requeremos uma probabilidade de falso alarme cada vez menor, essa curva cresce com uma inclinação cada vez menor, ou seja, se quisermos uma probabilidade de falso alarme mais restritiva (menor), precisaremos de um número ainda maior de amostras para atingir a mesma probabilidade de detecção que tínhamos antes.

Como já citado, é possível observar, no caso concreto, que para uma  $SNR$  de  $-21$  dB (requisito da IEEE 802.22), no caso de especificarmos um limiar CFAR que nos permite controlar a probabilidade de falso alarme em torno de 10%, uma

probabilidade de detecção de 90% (0.9) só é atingida a partir de 250 mil amostras, aproximadamente.

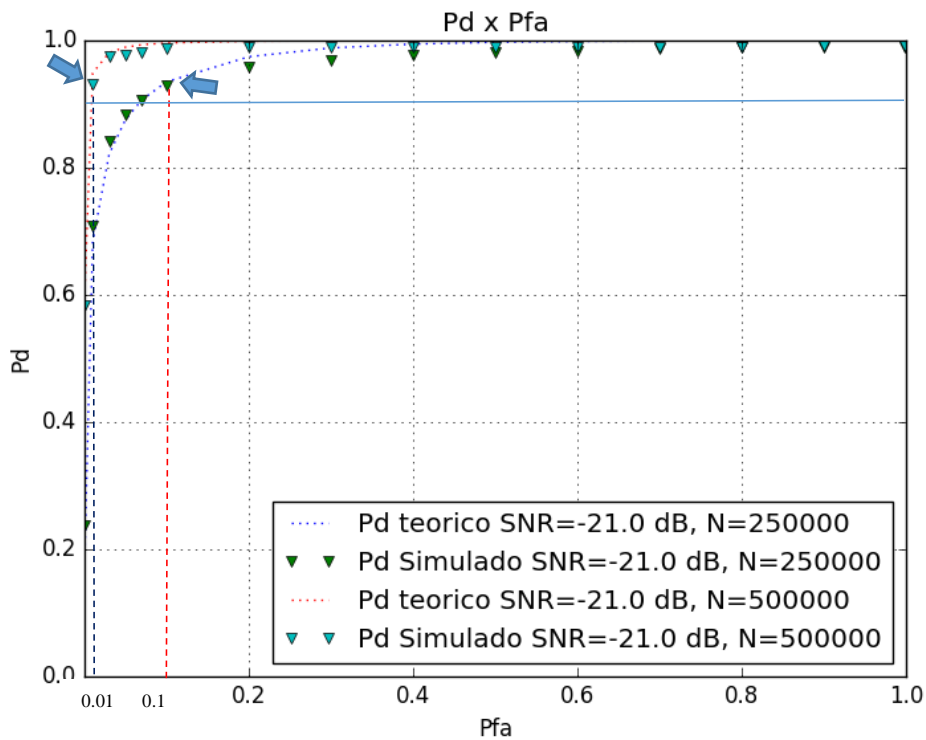


Figura 63 - Pd x Pfa para SNR = -21.0 dB e N= 200000 e 400000

Na Figura 63 podemos também observar que, se o requisito de falso alarme for ainda mais restritivo (uma Pfa de 1%, por exemplo) será necessário praticamente dobrar o número de amostras para cerca de 500 mil amostras para atingir o mesmo desempenho (Pd de 90%). Isso exigiria um aumento na capacidade de processamento do detector, tanto em termos de rapidez (taxa de amostragem mais elevada) como em armazenamento (memória), o que, em geral, leva a um aumento no custo de produção.

#### 8.4. Incerteza de ruído e a barreira de SNR

As simulações realizadas até aqui mostraram uma concordância razoável entre o modelo teórico e os resultados das simulações, porém a energia do ruído era perfeitamente conhecida na determinação do limiar CFAR.

Relembrando a expressão do limiar CFAR em (4.50), podemos observar que a partir do momento em que especificamos o número de amostras  $N$  e a probabilidade alvo de falso alarme (por exemplo,  $P_{fa}=0.1$ ), o limiar tem uma relação de dependência direta com a potência do ruído.

O que se pretende agora é averiguar até que ponto vai essa dependência, ou seja, qual o impacto que uma dada incerteza na potência do ruído (por exemplo, um erro de 1%) acarreta no desempenho do detector.

Podemos considerar o caso da incerteza inserida no modelo de ruído [21] por meio da seguinte representação. Temos um fator de incerteza  $\rho$  que define um intervalo para a incerteza de ruído da seguinte forma

$$\sigma^2 \in \left[ \frac{\sigma_w^2}{\rho}, \rho \sigma_w^2 \right]$$

onde  $\rho$  é o coeficiente da incerteza de ruído, sendo que  $\rho > 1$ .

Agora, podemos introduzir esse fator de incerteza  $\rho$  nas expressões de desempenho, e obter

$$P_d = Q\left(\frac{\tau - N(\sigma_w^2 + \sigma_s^2)}{\sqrt{2N(\sigma_w^2 + \sigma_s^2)^2}}\right) \Rightarrow P_d = Q\left(\frac{\tau - N(\sigma_w^2 + \sigma_s^2)}{\sqrt{2N(\sigma_s^2 + \frac{\sigma_w^2}{\rho})^2}}\right) \quad (8.1)$$

$$P_{fa} = Q\left(\frac{\tau - N\sigma_w^2}{\sqrt{2N\sigma_w^4}}\right) \Rightarrow P_{fa} = Q\left(\frac{\tau - N\sigma_w^2}{\sqrt{2N(\rho\sigma_w^2)^2}}\right) \quad (8.2)$$

$$P_{d \text{ incerteza ruído}} = Q \left( \frac{\tau - N(\sigma_s^2 + \sigma_w^2)}{\sqrt{2N(\sigma_s^2 + \frac{\sigma_w^2}{\rho})^2}} \right) \quad (8.3)$$

$$P_{fa \text{ incerteza ruído}} = Q \left( \frac{\tau - N\sigma_w^2}{\sqrt{2N(\rho\sigma_w^2)^2}} \right) \quad (8.4)$$

Conjugando as equações e eliminando o limiar  $\tau$  obtemos [21]

$$N = 2 \left[ \frac{\rho Q^{-1}(P_{fa}) - (1 + \gamma)Q^{-1}(P_d)}{\gamma - (\rho - \frac{1}{\rho})} \right]^2 \quad (8.5)$$

Lembrando que em (4.51), vimos que  $N$  sem a incerteza de ruído é dado por

$$N = 2 \left[ \frac{Q^{-1}(P_{fa}) - (1 + \gamma)Q^{-1}(P_d)}{\gamma} \right]^2$$

Portanto, devido ao impacto da incerteza de ruído, o número de amostras necessário para satisfazer os critérios de desempenho passa de (4.51) para (8.5), ou seja,

$$N = 2 \left[ \frac{Q^{-1}(P_{fa}) - (1 + \gamma)Q^{-1}(P_d)}{\gamma} \right]^2 \xrightarrow{\text{blue arrow}} 2 \left[ \frac{\rho Q^{-1}(P_{fa}) - (1 + \gamma)Q^{-1}(P_d)}{\gamma - (\rho - \frac{1}{\rho})} \right]^2$$

(4.51) → (8.5)

Comparando (8.5) com (4.51) podemos verificar que se  $\rho=1$ , as expressões tornam-se iguais, e, se  $\rho$  tiver apenas uma pequena variação, isso praticamente não tem impacto significativo nos resultados da expressão, mas isso depende sobretudo da magnitude da relação sinal ruído (SNR), principalmente porque ela aparece elevada ao quadrado no denominador.

Por exemplo, suponhamos que  $\rho \approx 1$ , nesse caso  $\gamma \approx \gamma - \left(\rho - \frac{1}{\rho}\right)$  e portanto (8.5)  $\approx (4.51)$ .

Agora, suponhamos um erro de 5% na estimativa do ruído, ou seja  $\rho = 1.05$  e considerando a relação sinal-ruído  $\gamma=0.1$ , nesse caso  $(\rho - 1/\rho) = 0.0976 \approx 0.1$  e portanto teríamos  $[\gamma - (\rho - 1/\rho)]^2 \approx [0.1 - 0.1]^2 \approx 0$ , o que, substituído na equação (8.5) faria  $N$  tender para infinito. Em outras palavras, precisaríamos de um tempo infinito de detecção, o que é impraticável.

Isso demonstra que um número infinitamente grande de amostras é necessário para alcançar as probabilidades de detecção e falso alarme pretendidas quando  $\gamma \rightarrow \left(\rho - \frac{1}{\rho}\right)$ .

Um detector de energia realizável jamais poderá ser implementado com uma relação sinal-ruído desse nível  $(\rho - 1/\rho)$ , referida como *barreira de SNR*.

#### 8.4.1. Simulações com a incerteza de ruído

Vimos portanto, que uma pequena variação na estimativa do ruído pode provocar uma severa degradação no desempenho, especialmente em casos em que a SNR é baixa.

Podemos observar essa situação em implementando uma simulação do detector de energia com  $N=500$  amostras e  $SNR = -15$  dB, onde o fator de incerteza acarreta uma estimativa errada para a potência do ruído com acréscimos de 1%, 3% e 5% acima da potência verdadeira (caso em que  $\rho=1.0$ ).

No caso das simulações, continuou sendo gerado um ruído gaussiano de média nula e variância unitária, porém agora o fator de incerteza ( $\rho$ ) foi aplicado no valor de  $\sigma_w^2$  empregado no cálculo do limiar.

$$\tau_{CFAR} = N\sigma_w^2 \left( \sqrt{\frac{2}{N}} Q^{-1}(P_{fa}) + 1 \right) \Rightarrow \tau_{CFAR} = N\rho\sigma_w^2 \left( \sqrt{\frac{2}{N}} Q^{-1}(P_{fa}) + 1 \right)$$

Na Figura 64, podemos claramente observar que o desempenho gradualmente cai como o fator de ruído aumentando. Isso indica que o detector de energia é muito

sensível à incerteza de ruído. A consequência disso é que com esse acréscimo na perda de detecção aumenta o número de usuários secundários que acabam prevendo que o espectro seja ocioso, o que aumenta muito além do esperado o nível de interferência aos usuários primários.

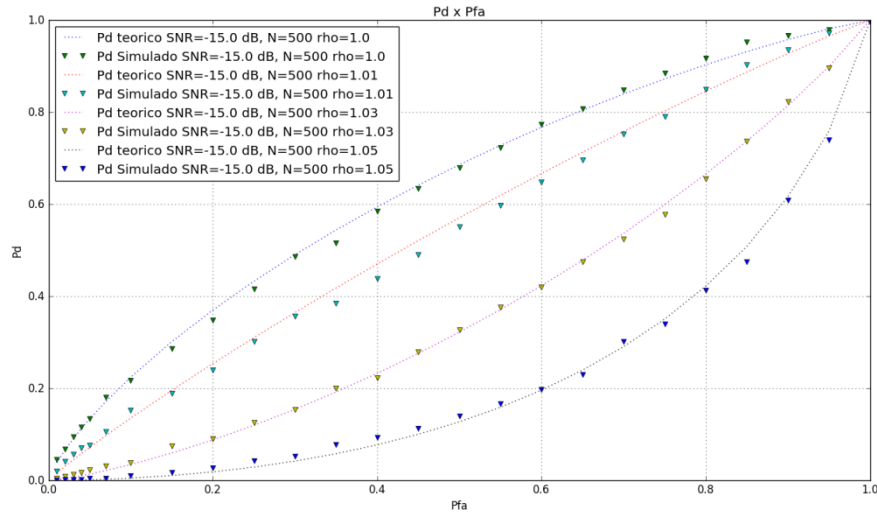


Figura 64 - Pd x Pfa com  $N=500$ ,  $SNR=-15$  dB e fator de incerteza de ruído

Por outro lado, podemos observar na Figura 65 que a probabilidade de falso alarme sofreu uma redução.

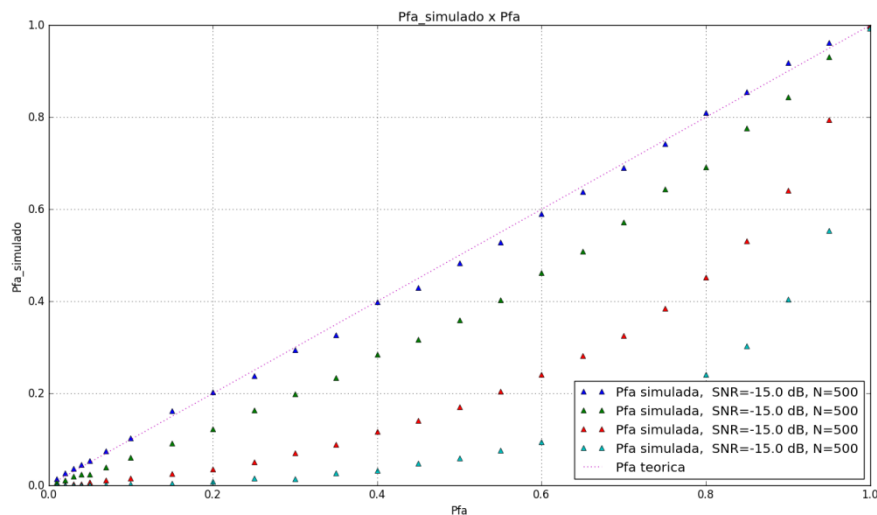


Figura 65 – Pfa x Pfa com  $N=500$ ,  $SNR=-15$  dB e fator de incerteza de ruído

Esta situação ocorre frequentemente em sistemas de rádio cognitivo, particularmente em ambientes de baixa relação sinal-ruído. Com esses resultados, fica clara a importância de escolher cuidadosamente o limiar mais adequado possível, pois pudemos perceber que um sistema, sem ajustes que minimizem os impactos da incerteza de ruído, pode ter seu desempenho fortemente prejudicado.

Resumindo, dado que algoritmos tradicionais de detecção de energia são baseados em um limiar fixo e pudemos verificar que, nesse caso, o desempenho diminui de acordo com o aumento do grau de incerteza de ruído, isso indica que a escolha de um limiar fixo não é uma solução válida quando há incerteza sob o ruído, e sendo assim, devem ser escolhidos limiares flexíveis de forma a compensar essa incerteza. Alguns autores [21, 22], propõem soluções como um limiar dinâmico que se baseia principalmente em um fator de compensação a ser aplicado na determinação do limiar e que possa minimizar o fator de incerteza.

De acordo com um trabalho<sup>[5]</sup> empregando detecção de energia no domínio do tempo em amostras experimentais de portadoras OFDM na banda de 3.5 GHz, conclui-se como um ponto fraco desse algoritmo de detecção a exigência de um conhecimento exato e preciso da potência do ruído durante o processamento das amostras em função da incerteza. Para isso foram realizados ajustes visando ajustar o limiar durante as etapas de processamento, inclusive, foi referido que se um único limiar fosse utilizado para todas as etapas de processamento, o erro geral de detecção (5.56%) aumentaria em 2%.

## 8.5. Parâmetros de projeto

Portanto ficou claro pelos resultados apresentados que os principais parâmetros de projeto do detector de energia são:

- Limiar ( $\tau$ ); e
- Número de amostras (N).

Embora o desempenho do detector de energia dependa também da relação sinal-ruído e da variância do ruído, os projetistas em geral têm um controle muito



limitado sobre a SNR e  $\sigma_w^2$  porque estes parâmetros dependem sobretudo do comportamento do canal sem fio [6].

### 8.5.1. Limiar

O controle adequado do limiar [6] é um dos aspectos mais críticos na detecção de energia, visto que é com base do valor do limiar que se decidirá se o sinal alvo está ausente ou presente. Como ficou claro pelas expressões de desempenho e pelo impacto da incerteza de ruído, o limiar determina todas as métricas de desempenho: probabilidade de detecção (Pd), probabilidade de falso alarme (Pfa) e probabilidade de perda de detecção (ou *miss detection*, Pmd).

Como o limiar varia de 0 a  $\infty$ , a seleção do limiar operacional é importante. O limiar em operação, portanto, pode ser determinado com base no valor alvo da métrica de desempenho que se pretende. Quando o limiar aumenta, tanto Pfa como Pd diminuem (e vice-versa).

Para um número de amostras (N) e uma variância do ruído ( $\sigma_w$ ) conhecidos, e o critério mais comum de definir o limiar é baseado em uma probabilidade constante de falso alarme, o CFAR, inclusive como foi citado, o método de Neyman-Pearson tem preferência justamente por visar fixar a probabilidade de falso alarme e maximizar a probabilidade de detecção, que é a abordagem mais usual.

### 8.5.2. Número de amostras

O número de amostras (N) também é um parâmetro de projeto importante para alcançar os requisitos de probabilidades de detecção e de falso alarme [6]. Se especificarmos um desempenho alvo definindo uma probabilidade de falso alarme Pfa e probabilidade de detecção Pd, o número mínimo exigido de amostras passa a ser definido em função da SNR.

Como foi visto nos resultados, o fato é que, quanto mais baixa é a relação sinal-ruído, maior terá de ser o número de amostras requerido. Isso claramente faz aumentar o tempo de sensoriamento e representa um claro compromisso e uma limitação prática do detector de energia para ambientes de SNR muito baixa.

Devido à propriedade de diminuição monótona da função  $Q^{-1}(\cdot)$ , podemos constatar (ver resultados) que o sinal pode ser detectado mesmo em uma região de SNR muito baixa se aumentarmos  $N$  (quando a potência do ruído é perfeitamente conhecida).

Além disso, o número necessário aproximado de amostras para alcançar um dado desempenho em termos de probabilidades de detecção e falso alarme varia com  $SNR^{-2}$ , isto é, se a SNR cair pela metade, será preciso quadruplicar o número de amostras para manter o desempenho, desde que, devido à incerteza de ruído, não seja atingida a SNR mínima imposta pela barreira de SNR.

Além disso, uma vez que  $N \approx T f_a$  onde  $T$  é o tempo de detecção e  $f_a$  é a frequência de amostragem, quando é requerido um  $N$  maior (em geral para superar uma SNR mais adversa) o tempo de detecção também aumenta.

Esta é a principal desvantagem na detecção de espectro pra valores baixos de SNR, ou seja, por causa da limitação do tempo máximo de detecção admissível (por exemplo, o IEEE 802.22 especifica que o tempo de detecção deve ser inferior a 2 s). Portanto, a seleção de  $N$  também é um problema de otimização, porém balizado pelos critério de limite de tempo de processamento para efetuar a detecção, sob pena de não pode acessar o canal, mesmo o usuário primário estando ausente.

## 9

## Conclusões e Trabalhos Futuros

### 9.1.

### Conclusões

De acordo com aquilo que era o objetivo principal do trabalho, o detector de energia foi criado e implementado com sucesso no GNU Radio, assim como foi também construído e implementado um esquema de análise de desempenho que permitiu a realização de simulações e ensaios com dispositivos reais em uma plataforma de estudo e desenvolvimento, como é o caso GNU Radio, utilizado neste trabalho.

Outro aspecto fundamental neste trabalho consistiu em fazer-se uma comparação dos resultados obtidos em simulação e os resultados obtidos em ensaios com as USRP com os resultados previstos de acordo com os modelos teóricos consagrados para a detecção de energia, com o fim de aferir o grau de confiabilidade da plataforma a pesquisas a serem desenvolvidas e implementadas com base no GNU Radio na área emergente de sensoriamento de espectro.

Pelo grau de concordância dos resultados obtidos tanto em relação às simulações como aos ensaios com as USRP empregadas, podemos concluir que a plataforma atendeu com sucesso as expectativas de servir como uma plataforma de pesquisa e desenvolvimento na área de sensoriamento de espectro, principalmente devido aos recursos já disponibilizados na forma de *toolboxes* e blocos de processamento de sinal que podem ser aplicados na pesquisa e desenvolvimentos de novas soluções ou na validação de soluções já apresentadas.

Quanto aos resultados do detector de energia construído no GNU Radio, foi possível concluir que os requisitos de sensibilidade para um detecção minimamente eficiente (acima de 90%) mesmo para sinais fracos (-116 dBm) e ainda imersos no ruído (com baixas relações sinal ruído, da ordem de -21 dB), acarretam numa necessidade de maior rapidez (taxa de amostragem e capacidade de cálculo) e capacidade de memória do detector, em virtude do aumento necessário no número

de amostras, o que em geral, tende consequentemente a acarretar um maior custo de produção.

Foi possível observar, por exemplo, que para satisfazer os critérios de desempenho da IEEE 802.22 de um desempenho de no mínimo 90% para a probabilidade de detecção e uma tolerância de no máximo 10% para a probabilidade de falso alarme com uma taxa constante, se quisermos atingir esse desempenho mesmo com um sinal primário bastante deteriorado pelo ruído com uma SNR de -21 dB, precisaremos em torno de 250 mil amostras. Se por acaso quisermos ser ainda mais intolerantes em relação à probabilidade de alarme falso, para no máximo uma ocorrência em cada 100 ( $P_{fa}=0.01$ ), teremos de praticamente dobrar o número de amostras para atingir o mesmo desempenho.

Podemos concluir então que o detector de energia, apesar das vantagens de maior facilidade e simplicidade de implementação em relação às demais técnicas consagradas como o detector ciclo estacionário ou baseado em matrizes de covariância, e além de necessitar de uma estimativa precisa da potência do ruído para a determinação do limiar (geralmente de acordo com o critério CFAR) apresenta como desvantagem uma sensibilidade elevada à degradação do sinal pelo ruído, o que se traduz na necessidade de um aumento considerável no número mínimo de amostras para poder compensar essa degradação, tornando o detector cada vez mais custoso em termos de recurso computacional e de memória disponível para processamento.

## **9.2. Trabalhos futuros**

Com base no trabalho desenvolvido e apresentado aqui, fica em aberto a utilização desta plataforma (GNU Radio) para criação e implementação de outros detectores para sensoriamento espectral e inclusive aproveitando a estrutura de análise de desempenho para comparar os resultados obtidos com os modelos teóricos em estudo.

## 10 Glossário

|       |   |
|-------|---|
| ASIC  | Application Specific Integrated Circuits        |
| GRC   | GNU Radio Companion                             |
| OOT   | Out of Tree module                              |
| PA    | Power Amplifier                                 |
| PAPR  | Peak to Average Power Ratio                     |
| USRP  | Universal Software Radio Peripheral             |
| WiMAX | Worldwide Interoperability for Microwave Access |
| WRAN  | Wireless Regional Area Network                  |
| SDR   | Software Defined Radio                          |
| SNR   | Signal-to-Noise Ratio                           |
| ST-DC | Software-Tunable-Downconverter                  |
| ST-UC | Software-Tunable-Upconverter                    |
| SWIG  | Simpler Wrapper and Interface Grabber           |
| TDD   | Time Division Duplex                            |
| FDD   | Frequency Division Duplex                       |
| H-FDD | Half-Frequency Division Duplex                  |

- [1] J. MITOLA et al., *Cognitive radio: Making software radios more personal*, IEEE Pers. Commun., vol. 6, no. 4, pp. 13–18, Aug. 1999.
- [2] S. Haykin, *Cognitive radio: Brain-empowered wireless communications*, IEEE Journal on Selected Areas in Communications, vol. 23, pp. 201–220, Fev-2005
- [3] R. Pfeifer and C. Scheier, *Understanding Intelligence*. Cambridge, MA: MIT Press, 1999
- [4] H. Urkowitz, *Energy Detection of Unknown Deterministic Signals*, Proceedings of the IEEE, vol. 55, no. 4, pp. 523 – 531, April 1967
- [5] C. V. R. Ron, M. S. Pontes, L. S. Mello e R. P. David, *Experimental Evaluation of Time Domain Energy Detection for Cognitive Radio Applications*, Microwave & Optoelectronics Conference (IMOC), 2011 SBMO/IEEE MTT-S International, 2011
- [6] S. Attapatu, C. Tellambura, and H. Jiang, *Energy Detection for Spectrum Sensing in Cognitive Radio*, Springer Briefs in Computer Science, 2014
- [7] H. V. Poor, *An Introduction to Signal Detection and Estimation*, 2nd edition, Springer 1994
- [8] S. M. KAY, *Fundamentals of Statistical Signal Processing Vol. II: Detection Theory*, Prentice-Hall PTR, 1998

- [9] A. M. Wyglinski, M. Nekovee, Y. T. Hou, *Cognitive Radio Communications and Networks: Principles and Practice*, ELSEVIER, 2010
- [10] PROAKIS, John G., *Digital Communications*, 3ª ed., McGraw-Hill, 1995
- [11] M. A. Abdulsattar e Z. A. Hussein, *Energy detector with Baseband Sampling for Cognitive Radio: Real-Time Implementation*, Wireless Engineering and Technology, Vol.3 No.4, Published Online, October 2012.
- [12] M. A. Abdulsattar e Z. A. Hussein, *Energy detection technique for Spectrum sensing in cognitive radio: a survey*, International Journal of Computer Networks & Communications (IJCNC) Vol.4, No.5, September 2012
- [13] M. Barkat, *Signal Detection and Estimation*, 2<sup>nd</sup> ed., Artech House, 2005
- [14] M. D. Srinath, P. K. Rajasekaran e R. Viswanathan, *Introduction to Statistical Signal Processing with Applications*, Prentice-Hall, 1996
- [15] J.P.A. Albuquerque, J. M. P. Fortes e W. A. Finamore, *Probabilidade, Variáveis Aleatórias e Processos Estocásticos*, Editora PUC-Rio e Editora Interciência, 2008
- [16] A. Leon-Garcia, *Probability, Statistics and Random Processes for Electrical Engineering*, 3<sup>rd</sup> ed., Prentice-Hall, 2008
- [17] P. S. Coutinho, Prof. J. F. Rezende, *Detecção de Energia para Rádios Cognitivos usando GNU Radio e USRP2*, UFRJ-COOPE, 2011

- [18] H. L. Van Trees, *Detection, Estimation, and Modulation Theory, Part I. Detection, Estimation and Linear Modulation Theory*, John Wiley & Sons, Inc, 2001
- [19] H. ARSLAN e H. CELEBI, *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*, Springer, 2007
- [20] FETTE, Bruce A., *Cognitive Radio Technology*, 1st ed., ELSEVIER, 2006
- [21] C. Korumilli e C.I. Hemalatha, *Performance Analysis of Energy Detection Algorithm in Cognitive Radio*, International Journal of Engineering Research and Applications (IJERA), Vol. 2, Issue 4, pp.1004-1009, 2012
- [22] T. L. Jinbo Wu, Guicai e G. Yue, *The performance merit of dynamic threshold energy detection algorithm in cognitive radio systems*, The 1st International Conference on Information Science and Engineering (ICISE2009), IEEE Computer Society, 2009.
- [23] K. Minseok, *Prototyping and Evaluation of Software Defined Radio using GNU Radio-USRP*, Technical report of IEICE
- [24] [https://en.wikipedia.org/wiki/Spectrum\\_Task\\_Force](https://en.wikipedia.org/wiki/Spectrum_Task_Force), acessado em 26/03/2018
- [25] [https://kb.ettus.com\\_Series\\_Quick\\_Start\\_\(Daughterboard\\_Installation\)](https://kb.ettus.com_Series_Quick_Start_(Daughterboard_Installation)), acessado em 26/03/2018
- [26] <https://www.ettus.com/product/category/Daughterboards>, acessado em 26/03/2018
- [27] <https://wiki.gnuradio.org/index.php>, acessado em 26/03/2018



- [28] [https://kb.ettus.com/About USRP Bandwidths and Sampling Rates](https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates),  
acessado em 26/03/2018
  
- [29] <https://wiki.gnuradio.org/index.php/InstallingGRFromSource>,  
acessado em 26/03/2018
  
- [30] <https://wiki.gnuradio.org/index.php/OutOfTreeModules>,  
acessado em 26/03/2018
  
- [31] <https://wiki.gnuradio.org/index.php/TutorialsWritePythonApplications>,  
acessado em 26/03/2018

## A Instalação do GNU Radio e GNU Radio Companion

O GNU Radio foi instalado utilizando o *script* de instalação *build-gnuradio* desenvolvido e disponibilizado por Marcus Leech [29]. Esse *script* é próprio para os sistemas recentes Fedora e Ubuntu e possui uma série de opções que podemos usar para instalar diferentes versões do GNU Radio. Apenas executando o *build-gnuradio* sem opções adicionais, ele busca e constrói a última versão lançada da série 3.7.

Podemos utilizar o *script* abrindo uma janela de terminal, entrando na pasta onde se deseja que os arquivos sejam armazenados e executando o comando:

```
wget http://www.sbrac.org/files/build-gnuradio && chmod a + x build-  
gnuradio && ./build-gnuradio
```

Este comando faz o *download* do instalador (*build-gnuradio*) e o torna executável. Em seguida, baixa e instala todas as dependências, baixa tanto o UHD quanto o GNU Radio do Git (o que significa que ele irá instalar automaticamente a versão mais recente do ramo 'master'), executa o processo *make* e instala o GNU Radio no sistema.

Na maioria dos casos, simplesmente executar o *script* fará tudo o que é necessário para se obter um sistema de GNU Radio executado a partir da fonte. Além disso, teremos todo o código-fonte no disco rígido e, portanto, disponível para futuras modificações. O *script* combina a flexibilidade de instalação da fonte com a facilidade de usar binários e é recomendado para a maioria dos usuários do Ubuntu e Fedora. É possível instalar ainda o GNU Radio utilizando o PyBOMBS ou até manualmente a partir da fonte porém, o *script* é o meio mais recomendado.

Os *scripts* python desenvolvidos no âmbito deste trabalho foram gerados através de uma funcionalidade do GNU Radio Companion (GRC), que é a ferramenta de interface gráfica para construção de fluxogramas para execução no GNU Radio.

## B

### Criação do bloco do detector de energia no GNU Radio

É importante destacar aqui que todo o procedimento para criação bloco do detector de energia realizado neste trabalho se baseou e será apresentado passo a passo seguindo o clássico tutorial “*Out-of-tree modules - Extending GNU Radio with own functionality and blocks*” disponibilizado na Wiki do GNU Radio [30]. Esse tutorial, por outro lado, se baseia muito do tutorial original "Como escrever um bloco?" escrito por *Eric Blossom*. O tutorial original foi praticamente reescrito porque acabou ficando muito desatualizado, em função das atualizações e mudanças na estrutura do GNU Radio.

Outro tutorial muito importante, também disponível na Wiki do GNU Radio, e que foi fundamental para a construção dos *scripts* de simulação e ensaio deste trabalho, é o “*Tutorials Write Python Applications*” [31].

#### B.1

##### Os módulos fora da árvore do GNU Radio – *OOT modules*

Sendo assim, e já no âmbito dos tutoriais citados, um módulo fora da árvore é um componente GNU Radio que não mora na árvore de origem do GNU Radio. Normalmente, se você quiser estender o GNU Radio com suas próprias funções e blocos, esse módulo é o que você cria (ou seja, você normalmente não adiciona material à árvore de origem do GNU Radio, a menos que você esteja planejando enviá-lo para o desenvolvedores para a integração a montante). Isso permite que você mantenha o código você mesmo e tenha funcionalidade adicional ao lado do código principal.

De acordo com os criadores do GNU Radio, muitos dos projetos OOT estão hospedados na CGRAN. Os projetos CGRAN estão disponíveis através da ferramenta PyBOMBS. Na verdade, quando você adiciona seu projeto ao *PyBOMBS recipe repo* (repositório), ele atualizará automaticamente o site CGRAN. Para obter mais informações sobre como configurar seu projeto e adicioná-lo ao PyBOMBS, podemos consultar o tutorial sobre *Configuring OOT*

*Projects*. Isso irá ajudá-lo a criar seus arquivos CMake para obter as melhores chances de instalação, editar o arquivo MANIFEST de seu projeto para exibir corretamente informações sobre CGRAN e criar o arquivo de receita para adicionar o projeto a PyBOMBS.

Um exemplo de tal módulo é o *spectral estimation toolbox*, que estende o GNU Radio com características de estimação espectral. Quando instalado, você tem mais blocos disponíveis (no caso, no GRC) que se comportam como o resto do GNU Radio, no entanto, os desenvolvedores são pessoas diferentes.

No caso deste trabalho em particular, foi criado um módulo OOT denominado *myblocks* e nele foi implementado em C++ o bloco *energy\_detector\_ff*. O sufixo *ff* é uma espécie de convenção para a nomenclatura dos blocos em que a primeira letra especifica o tipo usado à entrada e a segunda letra especifica o tipo utilizado à saída do bloco. Como este bloco toma à entrada amostras com valores do tipo *float* (ou vírgula flutuante), e também produz amostras do tipo *float* à saída, então o sufixo fica sendo *ff*. Se ele utilizasse amostras à entrada e à saída do tipo complexo, por exemplo, o sufixo seria *cc* e o nome do bloco seria *energy\_detector\_cc*.

## B.2

### **gr\_modtool – A ferramenta de criação de um módulo**

Neste trabalho, portanto, foi necessário criar um módulo (neste caso, o módulo *myblocks*) para o bloco do detector de energia (*energy\_detector\_ff*), porém, e de acordo com o tutorial “*OOT Modules*” citado, ao construirmos um módulo no GNU Radio, há muito trabalho monótono envolvido, tal como seções de código, geralmente referente a estruturas, e que têm de ser incluídas em muitos lugares com pouca ou nenhuma alteração. Isso é muito frequente quando se refere a linguagens que são consideradas detalhadas descritivamente, ou seja, o programador deve escrever um monte de código para fazer trabalhos mínimos. Além disso, é necessário cuidar da edição do *makefile*, e etc. O *gr\_modtool* é um *script* que já vem com o GNU Radio e que visa ajudar com todas essas coisas, editando automaticamente *makefiles*, usando modelos e fazendo o máximo de trabalho possível para o desenvolvedor de tal forma que é possível “pular” diretamente para a codificação DSP que de fato interessa.

Ao utilizar o *gr\_modtool*, podemos observar que ele faz diversas suposições sobre o como o código deve parecer. Quanto mais o seu módulo for personalizado e tiver alterações específicas, menos útil será o *gr\_modtool*, mas provavelmente continua sendo o melhor lugar para começar com qualquer novo módulo ou bloco. O *gr\_modtool* agora está disponível na árvore de origem do GNU Radio e é instalado por padrão.

Podemos visualizar as funcionalidades de criação ou exclusão de módulos e blocos por meio da opção *help*:

```
elizeu@elizeu-PC-Inmetro:~/Tese$ gr_modtool help

Usage:
gr_modtool <command> [options] -- Run <command> with the given options.
gr_modtool help -- Show a list of commands.
gr_modtool help <command> -- Shows the help for a given command.

List of possible commands:
```

| Name    | Aliases      | Description   |
|---------|--------------|---|
| disable | dis          | Disable block (comments out CMake entries for files)    |
| info    | getinfo, inf | Return information about a given module                 |
| remove  | rm, del      | Remove block (delete files and remove Makefile entries) |
| makexml | mx           | Make XML file for GRC block bindings                    |
| add     | insert       | Add block to the out-of-tree module.                    |
| newmod  | nm, create   | Create a new out-of-tree module                         |
| rename  | insert       | Add block to the out-of-tree module.                    |

```
elizeu@elizeu-PC-Inmetro:~/Tese$
```

Figura 66 - *gr\_modtool*: a ferramenta de criação de módulos e blocos

O *gr\_modtool* foi utilizado tanto para criar o bloco *energy\_detector\_ff* como o módulo OOT *myblocks* no qual o bloco está contido.

Aqui está uma lista rápida de todas as etapas necessárias para criar e editar blocos e módulos OOT, as quais foram utilizadas neste trabalho, conforme mostrado em anexo:

1. Criar um módulo (faça isso apenas uma vez por módulo): **gr\_modtool create MÓDULO**
2. Adicionar um bloco ao módulo: **gr\_modtool add BLOCO**
3. Criar um pasta de compilação: **mkdir build/**
4. Invocar o processo make:
 

```
cd build
cmake ../
make
```

(Note que só tem que se chamar cmake se os arquivos CMake sofreram alteração)
5. Invocar o teste: **ctest** (ou **ctest -V** para mais verbosidade)
6. Instalar (somente quando tudo funciona e nenhum teste falhar): **sudo make install**
7. Para usuários do Ubuntu - recarregar as libs: **sudo ldconfig**
8. Para excluir blocos da árvore de origem: **gr\_modtool rm REGEX**
9. Desabilitar blocos removendo-os dos arquivos CMake: **gr\_modtool disable REGEX**

Figura 67 - Lista rápida de comandos para criar módulos e blocos

Tendo seguido esses passos, podemos ver o módulo OOT *myblocks* e o bloco *energy\_detector\_ff* disponível na árvore do GRC para utilização nos fluxogramas pretendidos.

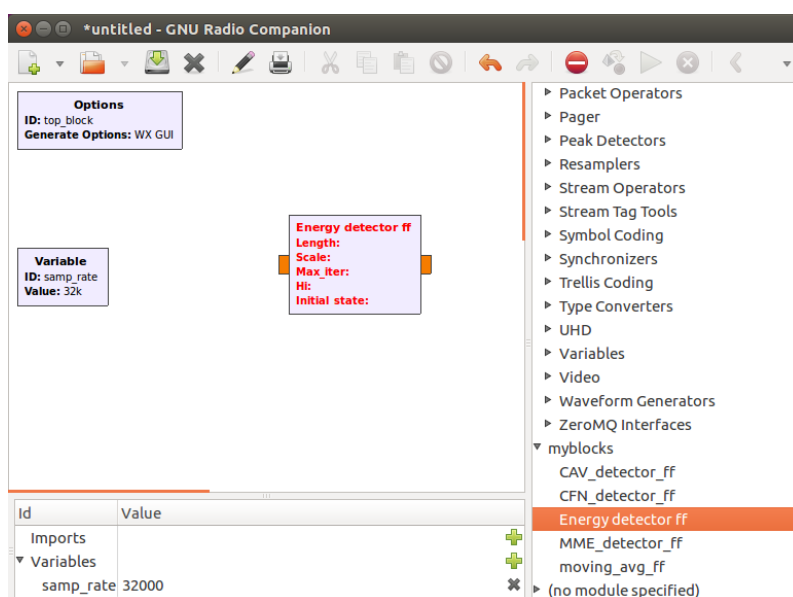


Figura 68 - Novo módulo e novo bloco na árvore do GNU Radio

## B.3

**CMake, make, etc.**

O GNU Radio usa o CMake como um sistema de compilação. Construir um módulo, portanto, exige que você tenha o *cmake* instalado, e qualquer gerenciador de construção que você preferir (na maioria das vezes isso é feito com o *make*, mas você também pode estar usando Eclipse, como foi o caso deste trabalho, ou o MS Visual Studio).

## B.4

**Criando o módulo OOT *myblocks***

Com a ferramenta *gr\_modtool*, basta apontar a linha de comando para onde quiser a nova pasta do módulo OOT (em geral a pasta raiz do GNU Radio) e fazer da seguinte forma:

```
% gr_modtool newmod myblocks
Creating out-of-tree module in ./gr-myblocks... Done.
Use 'gr_modtool add' to add a new block to this currently empty
module.
```

Agora, dentro da pasta *gnuradio* temos uma subpasta chamada *gr-myblocks* que contém a definição do módulo OOT *myblocks*. Podemos ir direto para o módulo *gr-myblocks* criado e ver de que ele é composto:

```
gr-myblocks % ls
apps  cmake  CMakeLists.txt  docs  examples  grc  include  lib
python  swig
```

Vemos que ele consiste em várias subpastas. Qualquer coisa que será escrita em C ++ (ou C, ou qualquer linguagem que não seja Python) é colocado em *lib/*. Para arquivos C ++, geralmente temos cabeçalhos que são colocados em *include/* (se forem exportados) ou também em *lib/* (se forem relevantes somente durante a compilação, mas não forem instalados posteriormente, como arquivos *\_impl.h*).

O material Python, como é esperado, entra na pasta *python/*. Isso inclui *unit tests*, ou *testes de unidade* (que não estão instalados) e partes do módulo *python*, que já estão instalados.

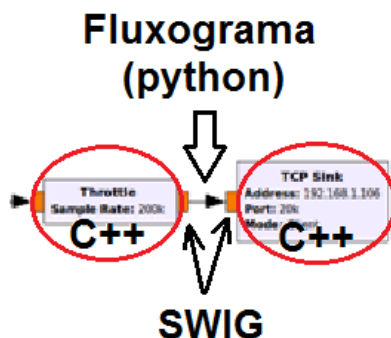


Figura 69 - SWIG cuida das interfaces (parâmetros) entre python e C++

É interessante observar que os blocos do GNU Radio estão disponíveis para importação e utilização no Python (como é o caso do *script* de um fluxograma .grc) mesmo se eles foram escritos em C ++. Isso é feito com a ajuda do SWIG (*Simplified Wrapper and Interface Generator*, ou SWIG, que é uma ferramenta de desenvolvimento de *software* que conecta programas escritos em C e C ++ com uma variedade de linguagens de programação de alto nível, como é o caso do Python). Ele cria automaticamente o código de “cola” para tornar isso possível. O SWIG precisa de algumas instruções sobre como fazer isso, que são colocadas no subpasta *swig/*. A menos que se pretenda fazer algo mais específico com seu bloco, em geral não é necessário entrar na pasta *swig/*; a ferramenta *gr\_modtool* lida com tudo isso automaticamente.

Para os blocos estarem disponíveis no GRC, a interface gráfica do GNU Radio, é preciso adicionar descrições XML dos blocos e colocá-los em *grc/*.

Quanto à documentação do novo módulo e os blocos associados, *docs/* contém algumas instruções sobre como extrair documentação dos arquivos C ++ e arquivos Python (usamos Doxygen e Sphinx para isso) e também certifique-se de que eles estão disponíveis como *docstrings* em Python. O criador do bloco pode adicionar documentação personalizada aqui também.

O subpasta *apps/* contém todas as aplicações completas (tanto para GRC como para executáveis independentes) que são instaladas no sistema juntamente com os blocos.

A pasta, *examples/* pode ser usada para salvar exemplos, que são um adendo importante à documentação, pois outros desenvolvedores podem simplesmente olhar diretamente para o código para ver como seus blocos podem ser usados.



O sistema **build** (compilador) traz também alguma bagagem extra para a implementação do bloco, como por exemplo, o arquivo *CMakeLists.txt* (um dos que está presente em todos os subpastas) e a pasta *cmake*. É possível ignorar a última por enquanto, pois traz principalmente instruções para o CMake sobre como encontrar as bibliotecas de GNU Radio necessárias durante a compilação e etc. Os arquivos *CMakeLists.txt* precisam ser editados, principalmente para garantir que seu módulo seja construído corretamente. Se estiver havendo algum problema na compilação referente a algum módulo ou biblioteca que esteja faltando provavelmente essa referência deverá ser incluída aqui.

## B.5

### Escrevendo o bloco `energy_detector_ff` em C ++

Neste caso em particular, pretende-se criar um bloco que faz detecção de energia, ou seja, calcula uma média móvel ponderada do quadrado das N amostras consideradas na sua entrada e efetua uma decisão considerando um determinado limiar. Esse bloco aceitará um fluxo de entrada de *single float* e produzirá um fluxo de saída *single float*, isto é, para cada item de *single float* de entrada, será emitido um item de *single float* que reflete a decisão referente aos N últimos itens considerados e uma amostra *single float* de valor 1.0, o que representa a decisão pela presença do sinal do usuário primário, ou o valor 0.0, caso contrário. Portanto, seguindo as convenções de nomenclatura, o bloco será chamado *energy\_detector\_ff* porque tem entradas de do tipo *single float* e saídas de *single float*.

Vamos ter que fazer alguns arranjos neste bloco, assim como em outros que escrevemos neste artigo, acabando no módulo Python *myblocks*. Isso nos permitirá acessá-lo a partir do Python assim:

```
import myblocks
ed = myblocks.energy_detector_ff()
```

### Gerando os arquivos modelo para o bloco

O primeiro passo é criar arquivos vazios para o bloco e editar os arquivos de *CMakeLists.txt*.

Novamente, *gr\_modtool* faz o trabalho. Na linha de comando, vá para a pasta *gr-myblocks* e digite:

```
gr-myblocks % gr_modtool add -t sync energy_detector_ff
GNU Radio module name identified: myblocks
Language: C++
Block/code identifier: energy_detector_ff
Enter valid argument list, including default arguments:
Add Python QA code? [Y/n]
Add C++ QA code? [y/N]
Adding file 'energy_detector_ff_impl.h'...
Adding file 'energy_detector_ff_impl.cc'...
Adding file 'energy_detector_ff.h'...
Editing swig/myblocks_swig.i...
Adding file 'qa_energy_detector_ff.py'...
Editing python/CMakeLists.txt...
Adding file 'myblocks_energy_detector_ff.xml'...
Editing grc/CMakeLists.txt...
```

Na linha de comando, especificamos que estamos adicionando um bloco, seu tipo é '*sync*' (falaremos a seguir sobre quais são os tipos de blocos) e é chamado *energy\_detector\_ff*.

*gr\_modtool* então pergunta se o seu bloco tem argumentos, se queremos ou não queremos código QA (*quality assurance*) para Python e para C ++.

Agora, podemos observar os diferentes arquivos de *CMakeLists.txt* e ver o que *gr\_modtool* fez. Podemos ver também diversos arquivos novos que o *gr\_modtool* criou e que agora precisam ser editados se quisermos que o bloco funcione.

## Programação do bloco dirigida por teste

Poderíamos começar lidando logo com o código C++, mas em geral sempre é interessante escrever o código de teste primeiro. Afinal, temos uma boa especificação para o processamento do bloco: tomar um único *stream* de *floats* como entrada e produzir um único *stream* de *floats* como a saída, onde a saída é uma decisão (1.0 ou 0.0) baseada no fato da energia calculada ultrapassar ou não o limiar estabelecido. Vamos então abrir *python/ qa\_energy\_detector\_ff.py*, que foi editado para ficar desta forma:

```
from gnuradio import gr, gr_unittest
from gnuradio import blocks
import myblocks

class qa_energy_detector_ff (gr_unittest.TestCase):
    def setUp (self):
        self.tb = gr.top_block ()
    def tearDown (self):
        self.tb = None
    def test_001_t (self):
        (...)
    def test_002_t (self):
        (...)
if __name__ == '__main__':
    gr_unittest.run(qa_energy_detector_ff, qa_energy_detector_ff.xml")
```

Figura 70 - Script de teste do bloco (*qa\_energy\_detector\_ff.py*)

Onde as funções de teste estão definidas a seguir. Vemos que foram implementadas duas funções de teste. A primeira função (*test\_001\_t*) testa efetivamente se as decisões do novo bloco (*energy\_detector\_ff*) correspondem de fato ao que seria esperado para um dado vetor de amostras e um limiar específico, e a segunda (*test\_002\_t*), realiza o mesmo teste porém com uma conjunção dos blocos *Multiply*, *Moving Average* e *Threshold* do GNU Radio e que também implementam a detecção de energia de maneira similar ao bloco criado.

```

def test_001_t (self):
    # Verifica as decisões obtidas do Detector de Energia com as decisões
    esperadas
    thres_value = 65
    src_data = (8.0, -4.0, 12.0, 8.0, 4.0, -12.0, 8.0, -4.0, 12.0, 16.0)
    expected_result = (0, 0, 0, 1, 0, 1, 1, 0, 1, 1)
    src = blocks.vector_source_f (src_data)
    ed = myblocks.energy_detector_ff (4, 0.25, 4000, thres_value,
    thres_value, 0)
    snk = blocks.vector_sink_f ()
    self.tb.connect (src, ed)
    self.tb.connect (ed, snk)
    self.tb.run ()
    result_data = snk.data ()
    self.tb.run ()

    # check data
    self.assertFloatTuplesAlmostEqual(expected_result, result_data, 6)

```

Figura 71 - 1ª função de teste do bloco (test\_001\_t)

Ao implementar o teste são utilizadas algumas funções de apoio para validação dos resultados obtidos. O *gr\_unittest* é uma extensão para o módulo Python *unittest* padrão. *gr\_unittest* adiciona suporte para verificar a igualdade aproximada de tuplas de *floats* e números complexos. O *unittest* usa o mecanismo de reflexão do Python para encontrar todos os métodos que começam com *test\_* e os executa. O *unittest* envolve cada chamada para *test\_\** com chamadas correspondentes a *setUp* e *tearDown*. Consulte a [Python unittest documentation](#) para obter mais detalhes.

Quando executamos o teste, *gr\_unittest.main* vai chamar *setUp*, *test\_001\_t*, *test\_002\_t* e *tearDown*, nessa ordem. A função *test\_001\_t* constrói um pequeno fluxograma que contém três nós. O bloco *blocks.vector\_source\_f (src\_data)* irá gerar os elementos de *src\_data* a serem processados e depois dizer que está concluído. Quanto a *myblocks.energy\_detector\_ff* é o bloco que estamos testando do módulo *myblocks* recém-criado, e o bloco *blocks.vector\_sink\_f* coleta as decisões à saída do detector de energia *myblocks.energy\_detector\_ff*.

Uma vez que *tb* é o objeto que representa o fluxograma de teste, o método *run ()* executa o fluxograma até que todos os blocos indiquem que estão com seu processamento concluído. Finalmente, verificamos se o resultado da execução de *energy\_detector\_ff* em *src\_data* corresponde ao que esperamos em *expected\_result*).

```
def test_002_t (self):
    #Compara as decisões do Detector de Energia implementado
    #com as decisões do detector composto pelo multiply (para
    #fazer o quadrado das amostras),
    #o Moving_Average e o Threshold do módulo Blocks
    thres_value = 65
    src_data = (8.0, -4.0, 12.0, 8.0, 4.0, -12.0, 8.0, -4.0, 12.0, 16.0)
    expected_result = (0, 0, 0, 1, 0, 1, 1, 0, 1, 1)
    src = blocks.vector_source_f (src_data)
    mt = blocks.multiply_vff(1)
    ma = blocks.moving_average_ff(4, 0.25)
    th = blocks.threshold_ff(thres_value, thres_value, 0)
    snk = blocks.vector_sink_f ()
    self.tb.connect((src, 0), (mt, 0))
    self.tb.connect((src, 0), (mt, 1))
    self.tb.connect (mt, ma)
    self.tb.connect (ma, th)
    self.tb.connect (th, snk)
    self.tb.run ()
    result_data = snk.data ()
    self.tb.run ()

    # check data
    self.assertFloatTuplesAlmostEqual (expected_result, result_data, 6)
```

Figura 72 - 2ª função de teste do bloco (test\_002\_t)

A função *test\_002\_t* faz um teste similar ao da função *test\_001\_t*, porém utilizando os blocos do GNU Radio *multiply\_ff*, *moving\_average\_ff* e *threshold\_ff* para averiguar se os resultados são os mesmos em relação ao bloco criado.

Chama a atenção o fato de que este teste normalmente é chamado antes de instalar o módulo. Isso significa que precisamos de alguma estratégia para poder carregar os blocos ao testar. *CMake* cuida da maioria das coisas, alterando *PYTHONPATH* adequadamente. Além disso, importamos *myblocks\_swig* em vez de *myblocks* neste arquivo.

Para que **CMake** realmente saiba que este teste existe, **gr\_modtool** modificou `python/CMakeLists.txt` com estas linhas:

```
#####
# Handle the unit tests
#####
include(GrTest)

set(GR_TEST_TARGET_DEPS gnuradio-myblocks)
set(GR_TEST_PYTHON_DIRS ${CMAKE_BINARY_DIR}/swig)
GR_ADD_TEST(qa_energy_detector_ff ${PYTHON_EXECUTABLE}
${CMAKE_CURRENT_SOURCE_DIR}/qa_energy_detector_ff.py)
```

## B.6

### Tipos de blocos quanto ao fluxo de dados

É muito comum no GNU Radio que os blocos de processamento de sinal tenham uma relação fixa entre a taxa de entrada e a taxa de saída. Muitos são 1:1, enquanto outros têm relações 1: N ou N:1.

Os blocos são construídos por meio de herança (um conceito muito utilizado em linguagens de programação orientada a objetos) de classes padronizadas nesse aspecto essencial para a construção do bloco e a forma como ele lida com o fluxo de amostras. A classe base abstrata para todos os blocos de processamento é a `gr::block..`

De acordo com o manual *doxygen* do GNU Radio disponibilizado em [https://gnuradio.org/doc/doxygen/classgr\\_1\\_1block.html](https://gnuradio.org/doc/doxygen/classgr_1_1block.html), constrói-se um fluxo de processamento de sinal criando uma árvore de blocos hierárquicos, que em qualquer nível também podem conter nós que implementam funções de processamento de sinal. A classe `gr::block` entretanto, é a classe base para todos esses nós de folha.

Os blocos têm um conjunto fixo de fluxos (ou *streams*) de entrada e de saída. O `input_signature` e o `output_signature` definem o número de fluxos de entrada e o número de fluxos de saída, respectivamente, e o tipo de itens de dados em cada fluxo. Embora os blocos possam consumir dados em cada fluxo de entrada em uma taxa diferente, todos os fluxos de saídas devem produzir dados na mesma taxa, e essa taxa pode ser diferente de qualquer uma das taxas de entrada.

Os blocos derivados criados pelo usuário substituem dois métodos, `forecast()` e `general_work()`, para implementar o comportamento pretendido em termos de processamento de sinal. O método `forecast()` é invocado pelo gerenciador

de processos (*scheduler*) do sistema para determinar quantos itens são necessários em cada fluxo de entrada para produzir um determinado número de itens de saída, enquanto o método *general\_work()* é invocado para executar o processamento do sinal específico pretendido para o bloco. Ele lê os itens de entrada, os manipula e em seguida grava os itens de saída.

O ***gr::block*** permite uma grande flexibilidade no que diz respeito ao consumo de *streams* de entrada e à produção de *streams* de saída. O uso inteligente de *forecast()* e *consume()* permite a construção de blocos de taxa variável. É possível, por exemplo, construir blocos que consomem dados em taxas diferentes em cada entrada e produzem saída a uma taxa que é uma função do conteúdo dos dados de entrada.

Por outro lado, é muito comum que os blocos de processamento de sinal tenham uma relação fixa entre a taxa de entrada e a taxa de saída. A grande maioria possui uma relação tipo 1:1, enquanto outros têm relações 1: N ou N:1.

Existem três tipos de blocos quanto ao aspecto dessa relação entre a taxa de entrada e a taxa de saída:

- *gr::sync\_block*;
- *gr::decimator\_block*; e
- *gr::interpolator\_block*.

Outra exigência comum é a necessidade de examinar mais de uma amostra de entrada para produzir uma única amostra de saída. Esse aspecto não tem qualquer relação direta entre taxa de entrada e saída conforme foi abordado no parágrafo anterior. Por exemplo, um filtro FIR não-decimador e não-interpolador precisa examinar N amostras de entrada para cada amostra de saída que produz, onde N é o número de *taps* no filtro. No entanto, ele consome apenas uma única amostra de entrada para produzir uma única amostra de saída. Esse conceito normalmente é chamado de "histórico".

## B.7

### O ***gr::sync\_block***

O ***gr::sync\_block*** é derivado do bloco ***gr::*** e implementa um bloco 1:1 com histórico opcional. Dado que sabemos a taxa da entrada para a saída, certas

simplificações são possíveis. Do ponto de vista do implementador, a principal alteração é que definimos um método *work()* em vez de *general\_work()*. O método *work()* tem uma sequência de chamada ligeiramente diferente; ele omite o parâmetro desnecessário *ninput\_items* e faz um arranjo para que *consume\_each()* seja chamado por nossa conta.

No caso do detector de energia implementado neste trabalho, o bloco é do tipo 1:1, ou seja, para cada amostra que entra no bloco, este produz como resultado uma única amostra na saída, e portanto, é do tipo *gr::sync\_block*.

Sendo assim, o construtor em *energy\_detector\_ff\_impl.cc* é feito de forma que a classe pai é *gr::sync\_block*, e, nesse caso, a função *work()* é a diferença real em relação ao caso geral *gr::block* (além disso, também não temos mais uma função *forecast()*).

Essa concepção mais simples do *gr::sync\_block()* tem a vantagem de requerer menos coisas para nos preocuparmos e menos código para escrevermos.

Outro aspecto importante também é que, se o bloco requer histórico maior que 1 (como é o caso do detector de energia), podemos invocar o método *set\_history()* no construtor ou a qualquer momento que o requisito for alterado.

Em resumo:

- *gr::sync\_block* nos dá uma versão de *forecast()* que lida com o requisito de histórico.
- *gr::sync\_decimator* é derivado de *gr::sync\_block* e implementa um bloco N:1 com histórico opcional.
- *gr::sync\_interpolator* é derivado de *gr::sync\_block* e implementa um bloco 1:N com histórico opcional.

Agora que temos esse conhecimento dos tipos de blocos, fica claro que *energy\_detector\_ff* deve ser um *gr::sync\_block* e com histórico. Esses aspectos ficam claros no código do construtor do bloco, conforme apresentado a seguir.



```

        /*
        * The private constructor
        */
energy_detector_ff_impl::energy_detector_ff_impl(int length, float scale,
        int max_iter, float lo, float hi, float initial_state)
        : gr::sync_block("energy_detector_ff",
        gr::io_signature::make(1, 1, sizeof(float)),
        gr::io_signature::make(1, 1, sizeof(float))),
        d_length(length),
        d_scale(scale),
        d_max_iter(max_iter),
        d_new_length(length),
        d_new_scale(scale),
        d_updated(false),
        d_lo(lo), d_hi(hi), d_last_state(initial_state)
        {
            set_history(length);
        }

        /*
        * Our virtual destructor.
        */
energy_detector_ff_impl::~energy_detector_ff_impl()
{ }

```

Figura 73 - O bloco `energy_detector_ff` como um `sync_block` com histórico

## B.8

### O código C++ do bloco do detector de energia

Agora que vimos a questão do teste para o bloco e os arquivos modelos para o novo bloco implementados pelo `gr_modtool`, podemos escrever o código C++ do novo bloco `energy_detector_ff`. Como vimos anteriormente, todos os blocos de processamento de sinal são derivados de **`gr::block`** ou uma de suas subclasses. Isso pode ser verificado no [block documentation](#) no manual gerado por *Doxygen*.

Nesta altura, **`gr_modtool`** já nos forneceu os três arquivos modelo que definem o bloco:

- `lib/energy_detector_ff_impl.h`
- `lib/energy_detector_ff_impl.cc`
- `include/myblocks/energy_detector_ff.h`

Agora só nos resta modificá-los para o nosso caso em questão. A primeira abordagem deverá ser focada nos arquivos de cabeçalho. Se o bloco que estivermos escrevendo for simples, em geral nem precisam realmente de mudanças pois o arquivo de cabeçalho em *include/* é muitas vezes completo o bastante depois de executar *gr\_modtool*, a menos que precisemos adicionar alguns métodos públicos como métodos de mudança, ou seja, *get's* e *set's*, como foi feito neste caso, e segue destacado aonde houveram as modificações.

Podemos observar portanto que neste caso, as únicas mudanças foram as inclusões de métodos de mudança (*set's* e *get's*) dos parâmetros do bloco.

```

#ifndef INCLUDED_MYBLOCKS_ENERGY_DETECTOR_FF_IMPL_H
#define INCLUDED_MYBLOCKS_ENERGY_DETECTOR_FF_IMPL_H

#include <myblocks/energy_detector_ff.h>

namespace gr {
  namespace myblocks {
    class energy_detector_ff_impl : public energy_detector_ff
    {
    private:
      int d_length;
      float d_scale;
      int d_max_iter;
      int d_new_length;
      float d_new_scale;
      bool d_updated;

      float d_lo, d_hi;    // the constant
      float d_last_state;

    public:
      energy_detector_ff_impl(int length, float scale, int max_iter,
float lo, float hi, float initial_state);
      ~energy_detector_ff_impl();

      int length() const { return d_new_length; }
      float scale() const { return d_new_scale; }
      void set_length_and_scale(int length, float scale);
      void set_length(int length);
      void set_scale(float scale);

      float lo() const { return d_lo; }
      void set_lo(float lo) { d_lo = lo; }
      float hi() const { return d_hi; }
      void set_hi(float hi) { d_hi = hi; }
      float last_state() const { return d_last_state; }
      void set_last_state(float last_state) { d_last_state =
last_state;

      // Where all the action really happens
      int work(int noutput_items,
        gr_vector_const_void_star &input_items,
        gr_vector_void_star &output_items);
    };
  }
}

```

Figura 74 - Mudanças no arquivo de cabeçalho energy\_detector\_ff\_impl.h

Isso nos deixa portanto com o arquivo *lib/energy\_detector\_ff\_impl.cc*. A ferramenta **gr\_modtool** sugere onde devemos alterar o código adicionando como marcadores os símbolos <+>.

Vamos analisar essas alterações por partes.

```
energy_detector_ff_impl::energy_detector_ff_impl(int length, float scale, int
max_iter, float lo, float hi, float initial_state)
    : gr::sync_block("energy_detector_ff",
        gr::io_signature::make(<+MIN_IN+>, <+MAX_IN+>, sizeof(<+ITYPE+>)),
        gr::io_signature::make(<+MIN_OUT+>, <+MAX_OUT+>, sizeof(<+OTYPE+>)))
{

    /*
     * Our virtual destructor.
     */
    energy_detector_ff_impl::~energy_detector_ff_impl()
    {
    }
}
```

Figura 75 - energy\_detector\_ff\_impl.cc - versão original do gr\_modtool

Observamos que o próprio construtor está vazio, pois o bloco não precisa de configurar nada. A única parte interessante é a definição das assinaturas de entrada e saída: na entrada, temos uma única porta que permite entradas de *floats* e na porta de saída uma única saída de floats, logo basta substituir os textos referentes aos tipos de entrada e saída <+ITYPE+> e <+OTYPE+> por *float*, e substituir os textos referentes aos números de fluxos de dados de entradas e de saídas <+MIN\_IN+>, <+MAX\_IN+>, <+MIN\_OUT+> e <+MAX\_OUT+> por 1.

```
energy_detector_ff_impl::energy_detector_ff_impl(int length, float scale, int
max_iter, float lo, float hi, float initial_state)
    : gr::sync_block("energy_detector_ff",
        gr::io_signature::make(1, 1, sizeof(float)),
        gr::io_signature::make(1, 1, sizeof(float))),
    (...)
}
```

Figura 76 - energy\_detector\_ff\_impl.cc - versão modificada

## B.9

A função *work()*

Logo após criar um bloco do tipo *gr::sync\_block*, o “código esqueleto” gerado pelo *gr\_modtool* em geral se apresenta aproximadamente da seguinte forma:

```
int
my_block_name::work(int noutput_items,
                    gr_vector_const_void_star &input_items,
                    gr_vector_void_star &output_items)
{
    const float *in = (const float *) input_items[0];
    float *out = (float *) output_items[0];

    // Do <+signal processing+>

    // Tell runtime system how many output items we produced.
    return noutput_items;
}
```

Figura 77 - A função *work()* - versão original criada pelo *gr\_modtool*

Portanto, considerando que temos de cuidar de coisas como histórico, vetores, múltiplas portas de entrada e etc, podemos observar que, como os blocos **sync** têm taxa de entrada para saída fixa, tudo o que é preciso saber é o número de itens de saída e assim é possível calcular quantos itens de entrada estão disponíveis, o que torna bem mais simples o código do bloco.

```
int
energy_detector_ff_impl::work(int noutput_items,
                             gr_vector_const_void_star &input_items,
                             gr_vector_void_star &output_items)
{
    if(d_updated)
    {
        d_length = d_new_length;
        d_scale = d_new_scale;
        set_history(d_length);
        d_updated = false;
        return 0; // history requirements might have changed
    }

    const float *in = (const float *)input_items[0];
    float *out = (float *)output_items[0];

    (continua...)
```

Figura 78 - A função *work()* - versão modificada (1ª parte)

Podemos agora então nos concentrar na função *work()* que é efetivamente onde o processamento das amostras ocorre. A tarefa consiste basicamente em calcular a média móvel do quadrado de N amostras e comparar o resultado com um determinado limiar. O código da função *work()* fica então da seguinte forma.

**(continuação...)**

```
// Do <+signal processing+>
// Obtendo média móvel e armazenando no vetor out[] para posterior decisão
float sum = 0;
float energy = 0;
int num_iter = (noutput_items>d_max_iter) ? d_max_iter : noutput_items;
for(int i = 0; i < d_length-1; i++) {
    sum += in[i]*in[i];          // Soma os quadrados das amostras
}

for(int i = 0; i < num_iter; i++) {
    sum += in[i+d_length-1]*in[i+d_length-1];
    out[i] = sum * d_scale;
    sum -= in[i]*in[i];          //Subtrai o quadrado da amostra anterior
}
(continua...)

// vetor out[] possui agora os valores das médias móveis na saída,
// logo esse vetor é que será comparado com o threshold para decisão
```

Figura 79 - A função *work()* - versão modificada (2ª parte)

Na primeira parte da função *work()*, temos apenas a possível atualização dos parâmetros do bloco, caso tenham sido alterados durante a execução, a configuração do *set\_history()* e a inicialização dos arrays referentes aos buffers de entrada e saída do bloco. Na segunda parte temos o processamento relativo ao efetuar o quadrado das amostras e o somatório desses quadrados ponderado por um fator de escala (tipicamente  $1/N$ ).

## B.10

### O método `set_history()`

Quanto ao método `set_history()`, se o bloco precisa de um histórico (ou seja, algo como um filtro FIR ou um acumulador que calcula uma média móvel, como é o caso do detector de energia), basta invocar isso no construtor, especificando o número de amostras em atraso (histórico) necessárias. A partir daí, o GNU Radio, certifica-se de que você tenha sempre o número solicitado de itens 'antigos' disponíveis.

O menor histórico que é possível ter é 1, ou seja, para cada item de saída, é preciso de 1 item de entrada. Se escolhermos um valor maior,  $N$ , isso significa que o item de saída é calculado a partir do item de entrada atual e dos  $N-1$  itens de entrada anteriores. O próprio gerenciador de processos do sistema (*scheduler*) cuida disso.

Se definirmos o histórico como comprimento  $N$ , os  $N$  primeiros itens no buffer de entrada incluirão os  $N-1$  anteriores (mesmo até que já tenham sido consumidos).

## B.11

### Implementando a detecção de energia

Como foi visto, na segunda parte da função `work()` em análise, é feita uma soma móvel dos quadrados das  $N$  amostras e que vai sendo atualizada considerando as amostras seguintes enquanto que as anteriores vão sendo descartadas. Essa soma é ponderada pelo fator de escala ( $1/N$ ) obtendo assim a média móvel dos quadrados das amostras pretendida, ou seja, a energia considerando as  $N$  amostras especificadas.

Para finalizar, resta apenas comparar a energia calculada com o limiar pre-estabelecido e colocar o valor da decisão respectiva na amostra do vetor correspondente ao *buffer* de saída do bloco.

Agora que temos o código do detector de energia (`energy_detector_ff`) implementado e as rotinas de teste, podemos compilar e testar o bloco.

*(continuação...)*

```
for(int i = 0; i < noutput_items; i++) {  
    energy = out[i];  
    if(energy > d_hi) {  
        out[i] = 1.0;  
        d_last_state = 1.0;  
    }  
    else if(energy < d_lo) {  
        out[i] = 0.0;  
        d_last_state = 0.0;  
    }  
    else  
        out[i] = d_last_state;  
}  
  
    // Tell runtime system how many output items we produced.  
    return num_iter;  
}
```

Figura 80 - A função work() - versão modificada (3ª parte)



## B.12

**Compilação do bloco: usando CMake**

De acordo com o que consta no manual do Ubuntu na internet em <http://manpages.ubuntu.com/manpages/xenial/man1/cmake.1.html>, o *CMake* é usado para configurar projetos de compilação por meio de *scripts*. É um gerador de sistema de construção (*build*) multiplataforma.

Os projetos especificam seu processo de compilação através de um arquivo de lista denominado *CMakeList.txt*, incluído em cada pasta de uma árvore de origem. Dessa forma, um usuário consegue criar um projeto usando *CMake* para gerar um sistema de compilação para uma ferramenta nativa em sua plataforma (como o *make*, por exemplo).

Em resumo, o *CMake* cuida da configuração da lista de caminhos para arquivos e bibliotecas que serão necessárias durante a compilação. O fluxo de trabalho típico de um projeto baseado em *CMake* como visto a partir da linha de comando em geral é o seguinte:

```
$ mkdir build      # Cria a pasta de compilação do módulo (só uma vez)
$ cd build/
$ cmake ../        # Diz ao CMake que seus arquivos de configuração
                  # estão na pasta acima
$ make            # Inicia a compilação
```

Figura 81 - Comandos típicos para compilação de um módulo

Agora temos uma nova pasta ***build/*** na pasta do nosso módulo e toda a compilação do módulo é baseada nesta pasta, e dessa forma a árvore de origem real do GNU Radio não fica cheia de arquivos temporários.

Se mudarmos quaisquer dos arquivos citados em *CMakeLists.txt*, devemos voltar a executar o *cmake ../* (embora na verdade, o *cmake* detecte essas alterações e reinicia automaticamente quando for executado da próxima vez). Durante a compilação, as bibliotecas são copiadas para a árvore de compilação. Somente durante a instalação, os arquivos são instalados na árvore de instalação, tornando então nossos blocos disponíveis para os aplicativos do GNU Radio.

Geralmente escrevemos nossos módulos de forma que eles acessem o código e as bibliotecas na árvore de instalação. Por outro lado, queremos que nosso código

de teste seja executado na árvore de compilação, onde podemos detectar problemas antes da instalação.

### B.13

#### Controle de qualidade: executando o *make test*

Como escrevemos o código de controle de qualidade (teste) do código C++ do bloco, podemos ver imediatamente se o que fizemos estava correto.

Usamos *make test* para executar nossos testes (execute isso a partir do subpasta *build/*, depois de chamar *cmake* e *make*). Isso invoca um *script* de shell que configura a variável de ambiente `PYTHONPATH` para que nossos testes usem as versões de árvore de compilação de nosso código e bibliotecas. Ele então executa todos os arquivos que têm nomes do formulário *qa\_\*.py* e relata o sucesso ou falha geral. Há um pouco de “ação de bastidores” necessária para usar as versões não instaladas do nosso código (podemos perceber isso consultando a pasta *cmake/*).

Se o bloco *energy\_detector\_ff* foi escrito corretamente, podemos verificar testando o bloco da seguinte forma:

```
gr-myblocks/build % make test
Running tests...
Test project /home/braun/tmp/gr-myblocks/build
  Start 1: test_myblocks
1/2 Test #1: test_myblocks ..... Passed    0.01 sec
  Start 2: qa_energy_detector_ff
2/2 Test #2: qa_energy_detector_ff ..... Passed    0.38 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) = 0.39 sec
```

Figura 82 - Testando o código do bloco com o *make test*

Se algo falhar durante os testes, podemos voltar ao código do bloco e corrigir os erros. Quando executamos o *make test*, estamos realmente invocando o programa *ctest* do CMake, que tem um número de opções que podemos passar a ele para obter informações mais detalhadas. Digamos que esquecemos de multiplicar *in[i] \* in[i]* e, portanto, não são obtidos realmente os quadrados das amostras. Se nós apenas executarmos *make test* ou até mesmo apenas *ctest*, teríamos algo semelhante a isso:

```

gr-myblocks/build $ ctest
Test project /home/braun/tmp/gr-myblocks/build
Start 1: test_myblocks
1/2 Test #1: test_myblocks ..... Passed    0.02sec
Start 2: qa_energy_detector_ff
2/2 Test #2: qa_energy_detector_ff .....***Failed    0.21 sec

    50% tests passed, 1 tests failed out of 2

Total Test time (real) = 0.23 sec

The following tests FAILED:
    2 - qa_energy_detector_ff (Failed)
Errors while running CTest

```

Figura 83 - Exemplo de erro ao testar o código de um bloco

Para descobrir o que aconteceu com o nosso teste *qa\_energy\_detector\_ff*, podemos executar *ctest -V -R energy\_detector\_ff*. A flag '-V' nos dá uma saída detalhada e a flag '-R' é uma expressão regex (regular expression tester) para executar apenas os testes que correspondem à expressão *energy\_detector\_ff* especificada.

Também podemos depurar o código colocando instruções com *print's* em nosso código de QA que esteja apresentando falhas, como por exemplo, imprimir *expected\_result* e *result\_data* para compará-las visualmente de forma a entender melhor o problema.

## B.14

### Flexibilidade do *gr::block*

Apesar da vantagem da simplicidade conferida pelo *gr::sync\_block*, *gr::block* permite uma tremenda flexibilidade no que diz respeito ao consumo de fluxos de dados de entrada e à produção de fluxos de dados de saída. Além disso, o uso inteligente dos métodos *forecast()* e *consume()* permite a construção de blocos de taxa variável. É possível, por exemplo, construir blocos que consomem dados em taxas diferentes em cada entrada e produzem saída a uma taxa que é uma função do conteúdo dos dados de entrada.

Um aspecto interessante no uso do método *set\_history()*, é que, se você usar um histórico de comprimento *k*, o GNU Radio manterá *k-1* entradas do buffer de entrada em vez de descartá-las. Isso significa que se o GNU Radio disser a você que o buffer de entrada possui *N* itens, ele realmente tem *N + k-1* itens que é

possível usar.

Vejamos o caso em que precisamos apenas de um item anterior, então o histórico é definido como  $k = 2$ . Se formos inspecionar o *loop for* de perto de um bloco que implementa isso, veremos que, de *noutput\_items* itens, *noutput\_items+1* itens são realmente lidos. Isso é possível porque há um item extra no buffer de entrada do histórico. Depois de consumir *noutput\_items* itens, a última entrada não é descartada e estará disponível para a próxima chamada da função *work()*.

## B.15

### Tornando o bloco disponível no GRC

Agora que o código do bloco foi desenvolvido, testado e compilado com sucesso, é possível instalar seu módulo, porém ele ainda não estará disponível no GRC. Isso ocorre porque o *gr\_modtool* não pode criar arquivos XML válidos até que se tenha escrito um bloco. O código XML gerado quando se executa a ferramenta *gr\_modtool add* é apenas um código esqueleto.

Uma vez que se tenha terminado de escrever o bloco, *gr\_modtool* tem uma função para auxiliar a criar o código XML. Para o exemplo *myblocks*, é possível invocá-la no bloco *energy\_detector2\_ff* através da opção *makexml*.

```
gr-myblocks % gr_modtool makexml energy_detector_ff
GNU Radio module name identified: myblocks
Warning: This is an experimental feature. Don't expect any magic.
Searching for matching files in lib/:
Making GRC bindings for lib/energy_detector_ff_impl.cc...
Overwrite existing GRC file? [y/N] y
```

Figura 84 - Opção *makexml* para construir código xml de um bloco

Note que *gr\_modtool add* cria um arquivo GRC inválido, logo podemos sobrescrevê-lo. Na maioria dos casos, *gr\_modtool* não consegue descobrir todos os parâmetros por si só e você terá que editar o arquivo XML apropriado manualmente. O site wiki GRC tem uma descrição disponível.

Quando o bloco é bem simples, às vezes o XML realmente é válido. Podemos conferir o arquivo XML original gerado *myblocks\_energy\_detector\_ff.xml* dentro da pasta *grc/* e o arquivo modificado.

```
<?xml version="1.0"?>
<block>
  <name>energy_detector_ff</name>
  <key>myblocks_energy_detector_ff</key>
  <category>[myblocks]</category>
  <import>import myblocks</import>
  <make>myblocks.energy_detector_ff($length, $scale, $max_iter, $lo, $hi,
$initial_state)</make>
  <!-- Make one 'param' node for every Parameter you want settable from the GUI.
    Sub-nodes:
      * name
      * key (makes the value accessible as $keyname, e.g. in the make node)
      * type -->
  <param>
    <name>...</name>
    <key>...</key>
    <type>...</type>
  </param>

  <!-- Make one 'sink' node per input. Sub-nodes:
    * name (an identifier for the GUI)
    * type
    * vlen
    * optional (set to 1 for optional inputs) -->
  <sink>
    <name>in</name>
    <type><!-- e.g. int, float, complex, byte, short, xxx_vector, ...--></type>
  </sink>

  <!-- Make one 'source' node per output. Sub-nodes:
    * name (an identifier for the GUI)
    * type
    * vlen
    * optional (set to 1 for optional inputs) -->
  <source>
    <name>out</name>
    <type><!-- e.g. int, float, complex, byte, short, xxx_vector, ...--></type>
  </source>
</block>
```

Figura 85 - Arquivo XML gerado pelo *gr\_modtool* - versão original

```

<block>
  <name>Energy detector ff</name>
  <key>myblocks_energy_detector_ff</key>
  <category>[myblocks]</category>
  <import>import myblocks</import>
  <make>myblocks.energy_detector_ff($length, $scale, $max_iter, $lo, $hi,
  $initial_state)</make>
  <param>
    <name>Length</name>
    <key>length</key>
    <type>int</type>
  </param>
  <param>
    <name>Scale</name>
    <key>scale</key>
    <type>float</type>
  </param>
  <param>
    <name>Max_iter</name>
    <key>max_iter</key>
    <type>int</type>
  </param>
  <param>
    <name>Lo</name>
    <key>lo</key>
    <type>float</type>
  </param>
  <param>
    <name>Hi</name>
    <key>hi</key>
    <type>float</type>
  </param>
  <param>
    <name>Initial_state</name>
    <key>initial_state</key>
    <type>float</type>
  </param>
  <sink>
    <name>in</name>
    <type>float</type>
  </sink>
  <source>
    <name>out</name>
    <type>float</type>
  </source>
</block>

```

Figura 86 - Arquivo XML gerado pelo gr\_modtool - versão modificada

Se executarmos os comandos *sudo make install* e *sudo ldconfig* a partir da linha de comando dentro da pasta de compilação (*build/*), é possível usar o bloco no GRC. O comando *sudo ldconfig* é necessário pois, caso contrário, corremos o risco de receber uma mensagem de erro informando que a biblioteca que acabou de instalar não pôde ser encontrada. Se o GRC já estiver em execução, é possível clicar no botão *Reload Blocks* na barra de ferramentas do GRC, é uma seta circular azul no lado direito.

Agora deverá ser possível ver uma categoria *MYBLOCKS* na árvore do GRC.

## B.16

**Métodos adicionais do *gr::block*****Forecast ()**

No caso geral do *gr::block*, em que não há uma relação de 1:1 na taxa de entrada em relação à taxa de saída, o sistema precisa saber quantos dados são necessários para garantir a validade em cada um dos *arrays* de entrada. Como mencionado anteriormente, o método *forecast()* fornece essas informações e, portanto, é preciso substituí-las sempre que escrever um bloco derivado de *gr::block* (para blocos *sync*, isso é implícito).

A implementação padrão de *forecast()* diz que há uma relação de 1:1 entre *noutput\_items* e os requisitos para cada *stream* de entrada. O tamanho dos itens é definido por *gr::io\_signature::make* no construtor do *gr::block*. Os tamanhos dos itens de entrada e saída podem, naturalmente, diferir; isso ainda se qualifica como uma relação 1:1. Claro, se você tivéssemos esse relacionamento, não iríamos querer usar um *gr::block*, uma vez que o *gr::sync\_block* é mais simples e intuitivo.

```
// default implementation: 1:1

void
gr::block::forecast(int noutput_items,
                    gr_vector_int &ninput_items_required)
{
    unsigned ninputs = ninput_items_required.size ();
    for(unsigned i = 0; i < ninputs; i++)
        ninput_items_required[i] = noutput_items;
}
```

Figura 87 - A função *forecast()* do *gr::block()*

Apesar da implementação 1:1 atender a construção do *energy\_detector\_ff*, esta não seria apropriada para interpoladores, decimadores ou blocos com uma relação mais complicada entre *noutput\_items* e os requisitos de entrada. Dito isto, derivando suas classes de *gr::sync\_block*, *gr::sync\_interpolator* ou *gr::sync\_decimator* em vez de *gr::block*, é possível muitas vezes evitar a implementação de *forecast()*, como foi o caso do detector de energia implementado em que o bloco é do tipo *gr::sync\_block*.

## Set\_output\_multiple ()

Ao implementarmos a rotina de *general\_work()*, ocasionalmente é conveniente para o sistema em execução garantir que só se é solicitado a produzir um número de itens de saída que é múltiplo de um determinado valor. Isso pode ocorrer se o algoritmo naturalmente se aplica a um bloco de dados de tamanho fixo. Nesse caso, podemos invocar *set\_output\_multiple()* em seu construtor para especificar esse requisito. O múltiplo padrão de saída é 1. No caso da construção do detector de energia deste trabalho também não houve necessidade desta função uma vez que o bloco é do tipo *gr::sync\_block*.

## B.17

### Tudo de uma só vez – referência de comandos

Apresentamos aqui está uma lista rápida de todas as etapas necessárias para criar e editar blocos OOT no GNU Radio:

2. Criar o módulo para o bloco (faça isso uma vez por módulo): **gr\_modtool create NOMEMODULO**
3. Adicionar um bloco ao módulo: **gr\_modtool add NOMBLOCO**
4. Criar um diretório de compilação: **mkdir build/**
5. Invocar o processo make: **cd build && cmake <OPÇÕES> ../ && make**  
(Note que você só tem que chamar cmake se você mudou os arquivos CMake)
6. Invocar o teste: **ctest** (ou **ctest -V** para mais verbosidade)
7. Instalar (somente quando tudo funciona e se nenhum teste falhar): **sudo make install**
8. (para usuários do ubuntu) recarregar as libs: **sudo ldconfig**
9. Excluir blocos da árvore de origem: **gr\_modtool rm REGEX**
10. Desabilitar os blocos removendo suas referências dos arquivos CMake: **gr\_modtool disable REGE**



## C

## Criação da estrutura de análise de desempenho

Antes de criarmos a estrutura de análise, é necessário gerar os sinais de usuário primário e secundário. Admitamos que se queira então gerar um ruído gaussiano de média nula e variância unitária que será somado a um outro sinal gaussiano (também AWGN) correspondendo ao um usuário primário com média nula e variância definida de acordo com uma dada relação sinal-ruído especificada. O bloco *Noise Source* pode gerar tanto as amostras do sinal do usuário primário como a do ruído.

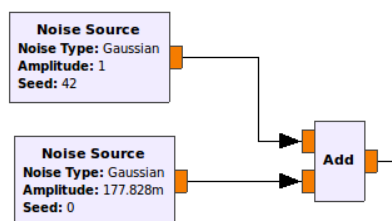


Figura 88 - Gerando amostras de sinal gaussiano com o *Noise Source*

Em seguida, pretende-se fazer a detecção de energia desse sinal (já com o ruído somado) e exibir tanto o sinal como a energia do mesmo e o resultado com as decisões obtidas. Para isso podemos utilizar a seguinte estrutura.

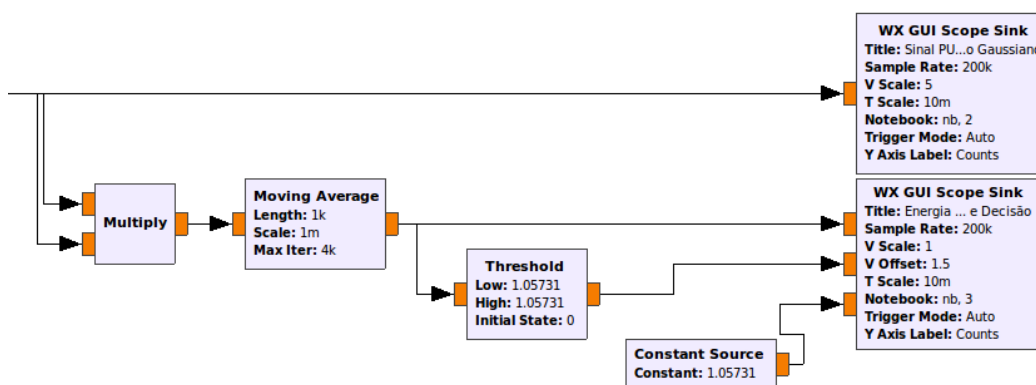


Figura 89 - Detecção de energia com os blocos nativos do GNU Radio

O bloco *Constant Source* apenas gera amostras com o valor do limiar definido de forma a constar na figura e facilitar a identificação do processo de decisão. O resultado do processo de detecção é possível visualizar na Figura 90, onde em azul temos a energia do sinal (métrica  $T(x)$ ), em vermelho o limiar, e em verde as decisões sobre a presença ou não do usuário primário

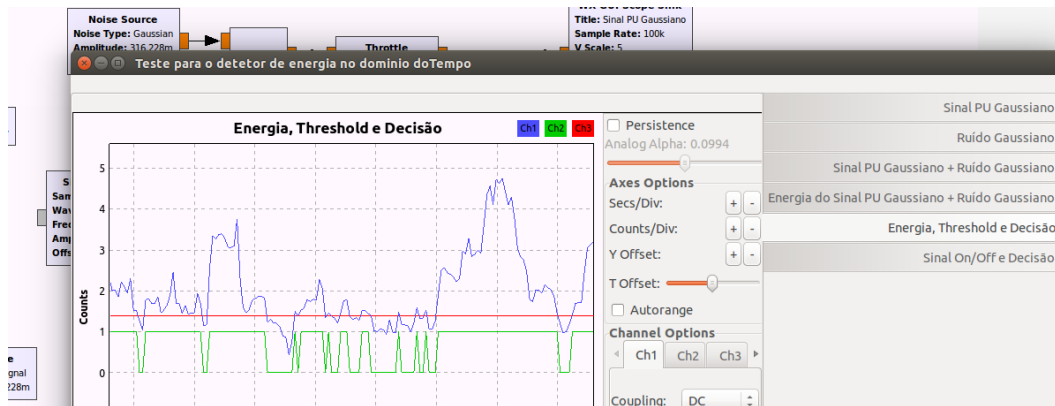


Figura 90 - Energia do sinal primário com o ruído, limiar e decisões

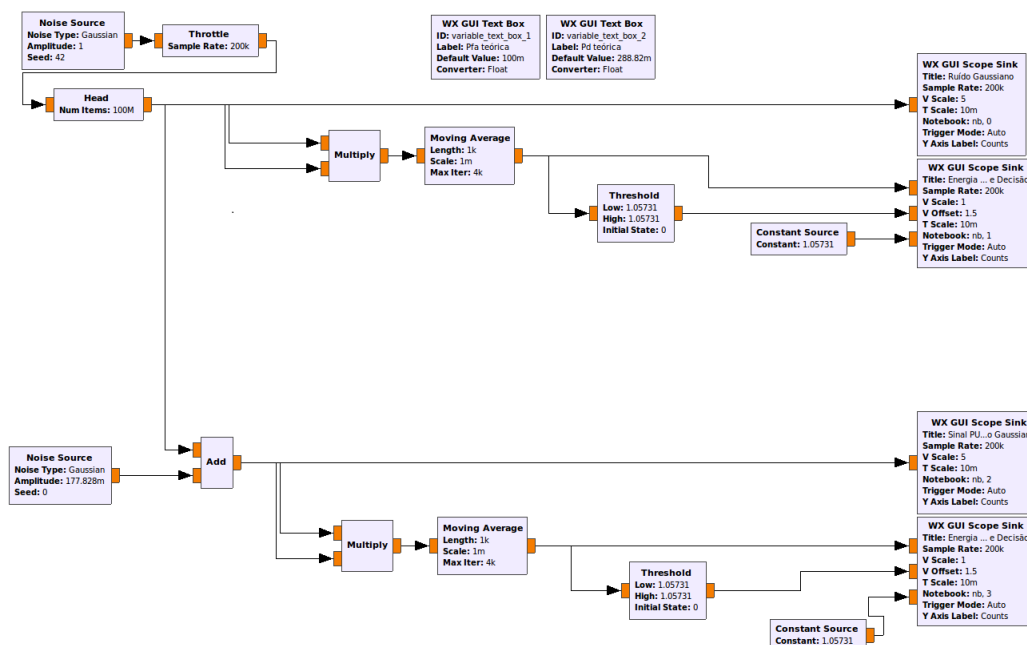


Figura 91 - Estrutura para visualizar apenas o ruído e o sinal+ruído

Se quisermos visualizar simultaneamente os sinais do ruído e do usuário primário simultaneamente assim como as decisões tomadas quando está presente apenas o ruído (ramo de cima) e o sinal mais o ruído (ramo de baixo), podemos utilizar a estrutura criada.

Agora, podemos imaginar um detector de energia sendo utilizado em uma estrutura que permita fazer a contagem das ocorrências de decisões do tipo '1' no ramo de cima e obter a taxa dessas ocorrências, a qual irá representar uma estimativa da probabilidade de falso alarme, uma vez que no ramo de cima apenas o ruído está presente.

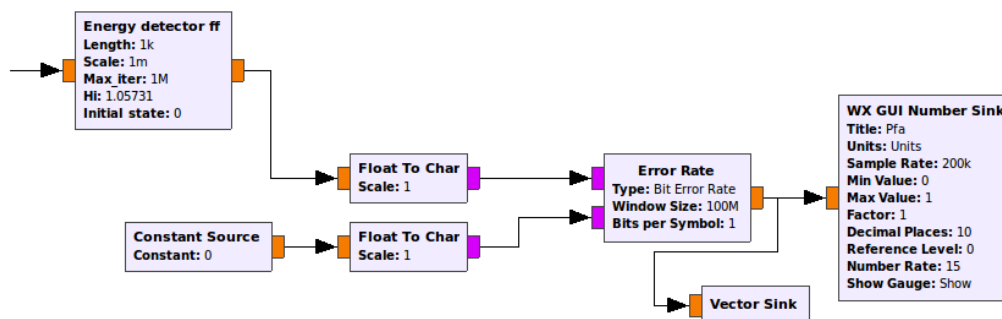


Figura 92 - Esquema para contagem de ocorrências visando Pfa

O bloco que permite obter a taxa de ocorrências equivocadas (decisão igual a '1') no ramo de cima, aonde apenas o ruído está presente, é o bloco *Error Rate*, que compara as amostras resultantes da decisão do detector de energia (que idealmente deveriam ser todas '0') com o sinal de comparação do bloco *Constant Source*, permitindo assim obter a taxa de ocorrências de decisões erradas do detector de energia, e assim temos uma estimativa da probabilidade de alarme falso, pois dessa forma o bloco *Error Rate* obtém justamente a taxa de ocorrência de decisões erradas indicando a presença do sinal do usuário primário quando temos apenas o ruído.

Da mesma forma podemos proceder no ramo de baixo aonde temos a presença do sinal do usuário primário somado à contaminação do ruído. Sendo assim, comparando a saída do detector com o sinal de comparação do bloco *Constant Source*, o bloco *Error Rate* agora nos permite obter a taxa de ocorrências de decisões erradas do detector de energia, ou seja, indicar que o sinal do usuário está

ausente quando isso é falso, ou seja a saída do bloco *Error Rate* nos dá a taxa de perda de detecção (ou “*miss detection*”), que é complementar à taxa de decisões acertadas. Sendo assim, utilizamos os blocos *Constant Source* e *Subtract*, para obter a taxa complementar e assim temos uma estimativa da probabilidade de detecção, pois dessa forma a obtenção da taxa complementar do bloco *Error Rate* nos fornece justamente a taxa de ocorrência de decisões acertadas, ou seja, indicando com sucesso a presença do sinal do usuário primário.

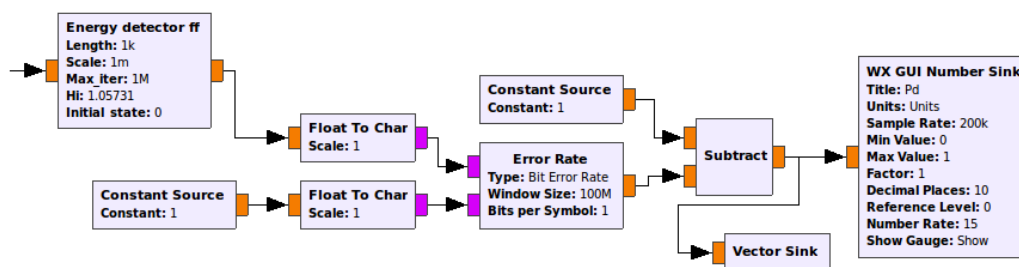


Figura 93 - Esquema para contagem de ocorrências visando Pd

Dessa forma, fica claro que com o uso dessa estrutura em dois ramos com a presença apenas do ruído no ramo de cima e a soma do ruído com o sinal do usuário primário no ramo de baixo, juntamente com o detector de energia e uma estrutura que, através do bloco *Error Rate* permite obter a taxa de falso alarme no ramo de cima e a taxa de detecção no ramo de baixo, temos portanto uma estrutura que nos permite estimar o desempenho de um dado detector (neste caso o detector de energia, mas poderia ser de outro tipo) em termos de uma estimativa das suas probabilidades de detecção e falso alarme, podendo inclusive obter uma estimativa das curvas ROC (*Receiver Operating Characteristic*) do detector em estudo.

## D

### Automatização dos Fluxogramas de Simulação

Pois bem, após essa pequena introdução ao Eclipse, podemos voltar o foco à solução (fluxograma GRC) construído para obter as estimativas para as probabilidades de detecção e falso alarme que servirão como critério de desempenho ao analisar o detector em causa.

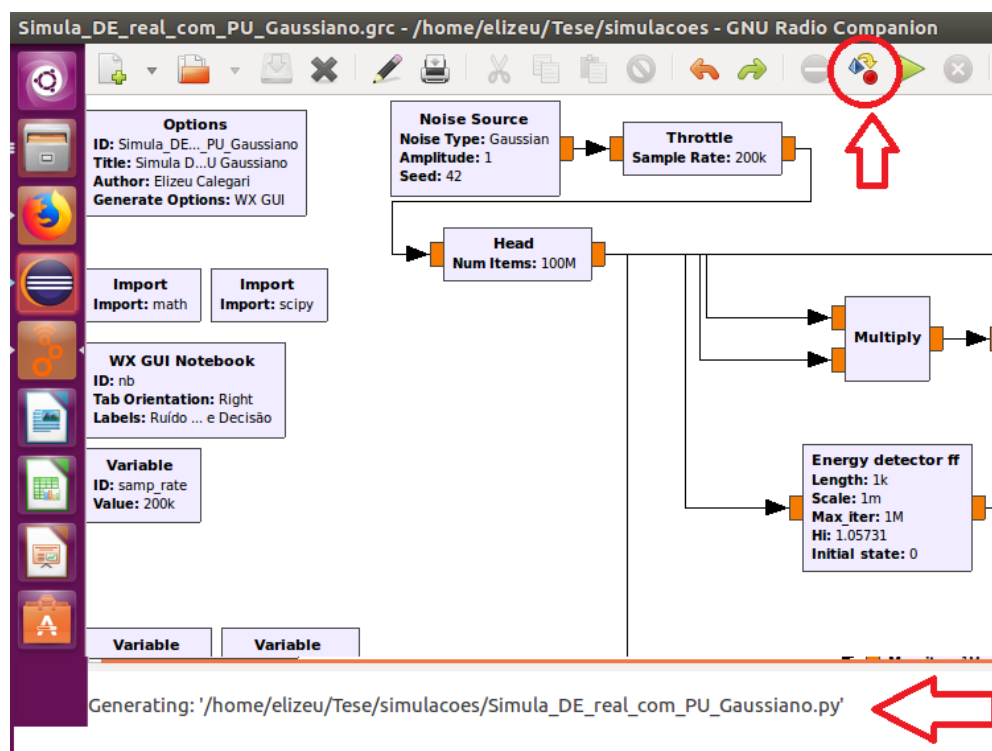
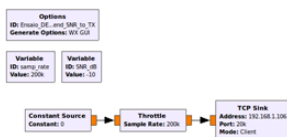


Figura 94 - O botão "Generate the Flowgraph" no GRC

Ao observar o ambiente do GRC utilizado para implementação do fluxograma de análise do detector, destacamos imediatamente o botão “*Generate the flowgraph*” e que, ao ser acionado, gera de imediato o *script* referente ao fluxograma implementado. Inclusive surge também uma mensagem no console abaixo do tipo “*Generating: ...(caminho e nome do script .py gerado)*” e podemos em seguida inspecionar esse *script* com um editor de texto, ou melhor, fazer uma cópia e importá-la para o eclipse para análise e edição.

## Fluxograma GRC



## Fluxograma Python

```

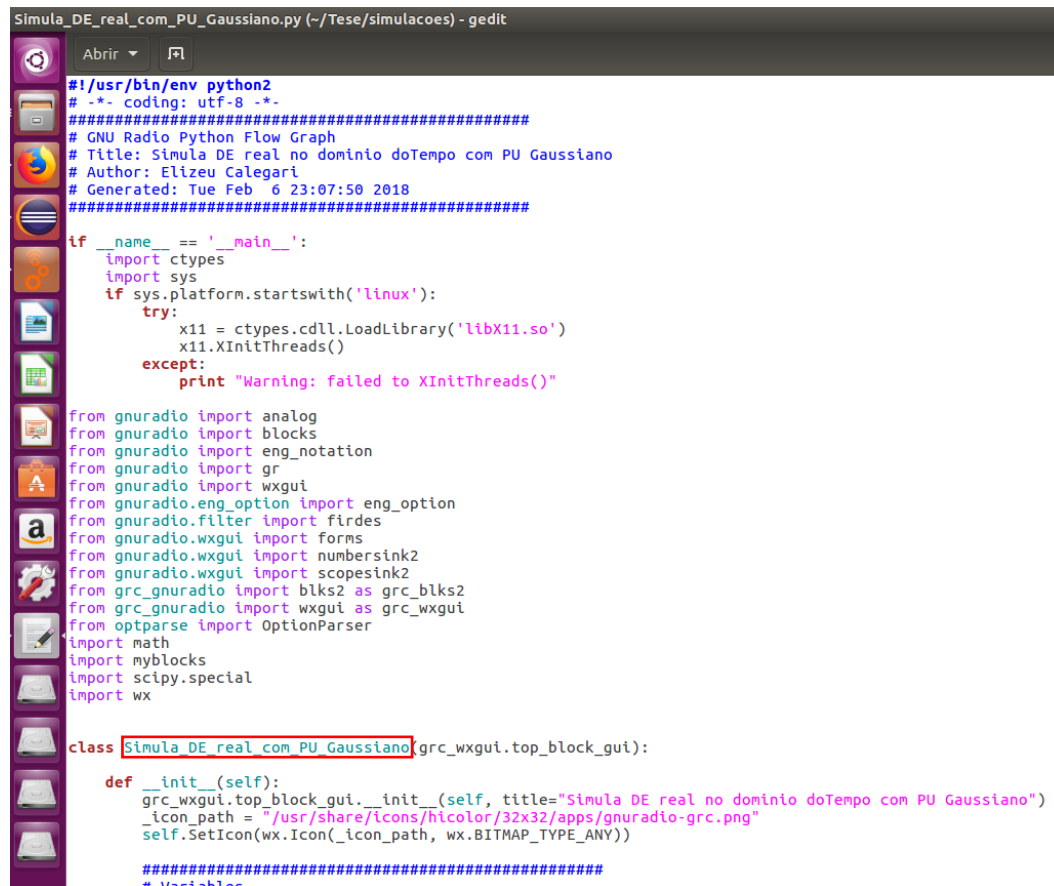
446 Ensaio_DE_real_Pd_x_SNR_com_PU_Gaussiano
447 if __name__ == '__main__':
448     Pfa_range = [0.1]
449     SNR_db_range = [-25.0, -22.5, -20]
450     N_range = [100, 1000, 2500, 5000]
451     print "===== Ensaio via USRPs ..."
452     print "\nSNR_db      N"
453
454     tb_send_SNR_to_TX = Ensaio_DE_real_Pd_x_SNR_to_TX.start()
455     tb_send_SNR_to_TX.start()
456
457     for p in Pfa_range:
458         for n in N_range:
459             Pd_teorica = [Pd_teorica]
460             Pfa_Simulada = []
461             Pd_Simulada = []
462             for s in SNR_db_range:
463                 s_corr=s
464                 # ...

```

Figura 95 - Fluxograma do GRC gera um arquivo .grc em linguagem python

Ao analisar o *script* .py gerado, observamos de pronto que ele inicia com a importação das bibliotecas necessárias para compilação e execução do código, inclusive algumas bibliotecas do próprio GNU Radio e do python, para manipulação de expressões e funções empregadas no fluxograma.

Um aspecto fundamental é a definição do objeto (no caso, “*Simula\_DE\_real\_com\_PU\_Gaussiano*”, da classe *grc\_wxgui.top\_block\_gui* cuja a instância representa o próprio fluxograma em execução, com todo o código referente aos blocos e controles incorporados durante a implementação do fluxograma, assim como a parte do código das bibliotecas importadas e necessárias. A função de membro *\_\_init\_\_* (), que é o construtor desta classe, ou seja, a função que inicializa o objeto fluxograma, que é a instância da classe em questão.



```

Simula_DE_real_com_PU_Gaussiano.py (~/Tese/simulacoes) - gedit
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
#####
# GNU Radio Python Flow Graph
# Title: Simula DE real no dominio doTempo com PU Gaussiano
# Author: Elizeu Calegari
# Generated: Tue Feb  6 23:07:50 2018
#####

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 gnuradio import analog
    from gnuradio import blocks
    from gnuradio import eng_notation
    from gnuradio import gr
    from gnuradio import wxgui
    from gnuradio.eng_option import eng_option
    from gnuradio.filter import firdes
    from gnuradio.wxgui import forms
    from gnuradio.wxgui import numbersink2
    from gnuradio.wxgui import scopesink2
    from grc_gnuradio import blks2 as grc_blks2
    from grc_gnuradio import wxgui as grc_wxgui
    from optparse import OptionParser
    import math
    import myblocks
    import scipy.special
    import wx

    class Simula_DE_real_com_PU_Gaussiano(grc_wxgui.top_block_gui):
        def __init__(self):
            grc_wxgui.top_block_gui.__init__(self, title="Simula DE real no dominio doTempo com PU Gaussiano")
            _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
            self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

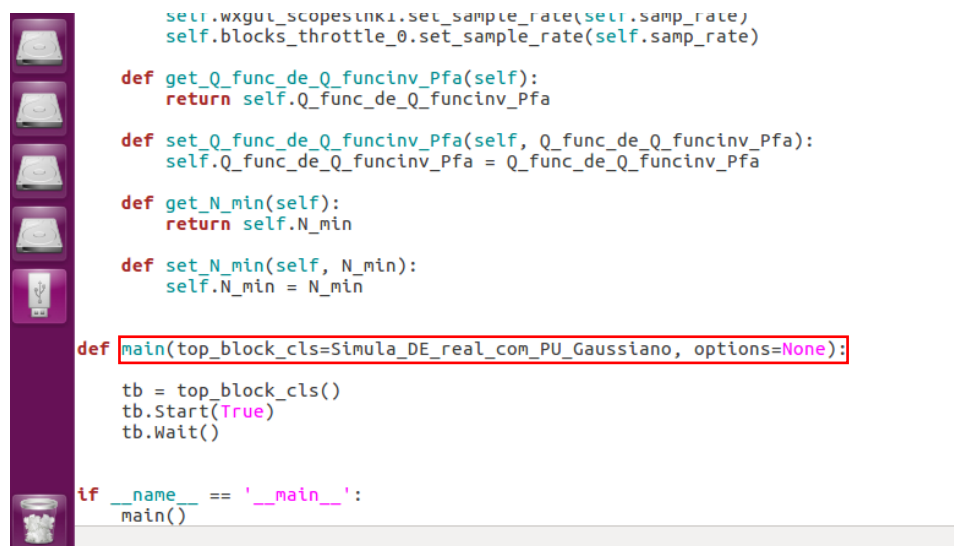
            #####
            # Variables

    def main(top_block_cls=Simula_DE_real_com_PU_Gaussiano, options=None):
        tb = top_block_cls()
        tb.Start(True)
        tb.Wait()

    if __name__ == '__main__':
        main()

```

Figura 96 - Script .py gerado pelo GRC a partir do fluxograma .grc



```

self.wxgui_scopesink1.set_sample_rate(self.samp_rate)
self.blocks_throttle_0.set_sample_rate(self.samp_rate)

def get_Q_func_de_Q_funcinv_Pfa(self):
    return self.Q_func_de_Q_funcinv_Pfa

def set_Q_func_de_Q_funcinv_Pfa(self, Q_func_de_Q_funcinv_Pfa):
    self.Q_func_de_Q_funcinv_Pfa = Q_func_de_Q_funcinv_Pfa

def get_N_min(self):
    return self.N_min

def set_N_min(self, N_min):
    self.N_min = N_min

def main(top_block_cls=Simula_DE_real_com_PU_Gaussiano, options=None):
    tb = top_block_cls()
    tb.Start(True)
    tb.Wait()

    if __name__ == '__main__':
        main()

```

Figura 97 - A função main() no script .py gerado pelo fluxograma

Após o código de definição do objeto do fluxograma e seus métodos de interface aos parâmetros do bloco associados (funções *sets* e *gets*), temos a definição da função *main()* onde o fluxograma é de fato instanciado e executado, portanto o foco para customização da execução do fluxograma GRC de forma a automatizar a coleta dos resultados pretendidos (probabilidades de detecção e falso alarme para cada cenário de Pfa, SNR e N) será exatamente nessa função *main()*.

```

42 from wx import Sleep, Millisleep
43 import pylab
44
45
46 class Simula_DE_real_com_PU_Gaussiano(grc_wxgui.top_block_gui):
47
48     def __init__(self, valor_Pfa, valor_SNR_dB, valor_N):
49         grc_wxgui.top_block_gui.__init__(self, title="Simula DE real no dominio
50         _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
51         self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))
52
53         #####
54         # Variables
55         #####
56         self.SNR_dB = SNR_dB = valor_SNR_dB
57         self.Pfa = Pfa = valor_Pfa
58         self.N = N = valor_N
59
60         self.snr = snr = math.pow(10.0, SNR_dB/10.0)
61         self.sigma_noise = sigma_noise = 1.0
62         self.Q_funcinv_Pfa = Q_funcinv_Pfa = math.sqrt(2)*scipy.special.erfinv(1
63         self.threshold_GFAF_nss_normalized = threshold_GFAF_nss_normalized = math

```

Figura 98 - Inclusão dos parâmetros Pfa, SNR\_dB e N na inicialização

Após importar a cópia do *script* .py gerado pelo GRC para o Eclipse, adotamos como estratégia acrescentar como parâmetros de inicialização do fluxograma os valores de Pfa, SNR e N especificados, de forma que os valores desses parâmetros são repassados às variáveis do fluxograma a ser executado, e dessa forma, podemos criar as instâncias que serão executadas para cada cenário (Pfa, SNR e N) pretendido.



```

335
336
337 def Pd_teoria(Pfa, SNR_dB, N):
338     """ Calcula a probabilidade de detecção teórica """
339     sigma_noise = 1.0
340     snr = math.pow(10, SNR_dB/10.0)
341     sigma_signal = math.sqrt(snr)*sigma_noise
342
343     Q_funcinv_Pfa = math.sqrt(2)*scipy.special.erfinv(1.0-2.0*Pfa)
344     threshold_CFAR = (1.0/N)*math.pow(sigma_noise,2)*N*(math.sqrt(2.0/N)*Q_funcinv_Pfa)
345     threshold_CFAR_nao_normalizd = math.pow(sigma_noise,2)*N*(math.sqrt(2.0/N)*Q_funcinv_Pfa)
346
347     Pd_teorica = 0.5 * math.erfc( ( threshold_CFAR_nao_normalizd - N*(math.pow(sigma_signal,2)) ) / ( N*(math.pow(sigma_noise,2)) ) )
348
349     # print "Pd teoria = %f" % Pd_teorica
350     # print "Pfa = %f" % Pfa
351     # print "SNR_dB = %f" % SNR_dB
352     # print "snr = %f" % snr
353     # print "sigma_signal = %f" % sigma_signal
354     #
355     # print "threshold_CFAR = %f" % threshold_CFAR
356     # print "threshold_CFAR_nao_normalizd = %f" % threshold_CFAR_nao_normalizd
357     # print "sigma_noise = %f" % sigma_noise
358     # print "sigma_signal = %f" % sigma_signal
359     # print "-----"
360     return Pd_teorica
361
362 def simular_Pd(Pfa, SNR_dB, N):
363     tb = Simula_DE_real_com_PU_Gaussiano(Pfa, SNR_dB, N)
364
365     tb.start()
366     Millisleep(N*10)
367     tb.stop()
368
369     Pfa_data = tb.blocks_vector_sink_x_0.data()
370     Pd_data = tb.blocks_vector_sink_x_1.data()
371
372     #print ("\nN      SNR_dB      => Pfa teo   Pfa sim   Pd teo   Pd sim")
373     print ("%1f      %.0f => %.2f      %f %f %f \n" % (SNR_dB, N, Pfa, Pfa_data[-1], Pd_data[-1], Pd_data[-1]))
374
375     return Pfa_data[-1], Pd_data[-1]
376
377

```

Figura 99 - Funções *Pd\_teoria()* e *simular\_Pd()* e a função *Millisleep()*

Além da customização na inicialização do objeto principal do fluxograma, acrescentamos também duas funções auxiliares. *Pd\_teoria()* como o nome indica, calcula as probabilidades de detecção teóricas para a configuração (Pfa, SNR, N) especificada vir a traçar a curva teórica de referência. Já a *simular\_pd()* é a que cria cada instância de fluxograma com a configuração desejada para obter os valores de Pd e Pfa pretendidos na simulação. Esses valores são repassados em um bloco *Vector Sink* e como a simulação demora algum tempo a estabilizar os esses valores devido à contabilização de cenários anteriores até se atingir as médias pretendidas, buscamos obter então o último valor dessas sequências de valores de Pd e Pfa através do índice -1, o qual retorna a última posição dos vetores com os valores simulados para Pd e Pfa.

É importante observar que, como o cálculo das probabilidades de detecção e falso alarme depende de uma média móvel que precisa convergir para o valor

pretendido, cada ponto é obtido gerando o respectivo fluxograma e executando, sendo que o tempo de execução até que os valores das probabilidades sejam considerados estáveis o suficiente para serem coletados é de dez vezes o número de amostras ( $N \times 10$ ) milissegundos. Essa definição é feita dentro da função *simular\_Pd()* invocando a função *Millisleep()*, a qual faz pausar a execução do *script* de simulação  $N \times 10$  milissegundos permitindo aguardar a estabilização do cálculo das probabilidades pretendidas.

Podemos observar também na função *simular\_Pd()* que o método *start()* aciona a inicialização e execução do fluxograma e o método *stop()* finaliza a execução do mesmo, de forma a podermos coletar as probabilidades pretendidas.

```

370
379
380 if __name__ == '__main__':
381
382     Pfa_range = [0.1]
383     SNR_dB_range = [-25.0, -22.5, -20.0, -19.0, -18.0, -17.0, -16.0, -15.0, -14.0]
384     N_range = [100, 1000, 2500, 5000, 10000]
385
386
387
388 print "==== Simulando ... ====\n"
389 print "\nSNR_dB   N   => Pfa teo   Pfa sim   Pd teo   Pd sim"
390
391
392 for p in Pfa_range:
393     for n in N_range:
394         Pd_teorica = [Pd_teoria(p,s,n) for s in SNR_dB_range]
395         Pfa_Simulada = []
396         Pd_Simulada = []
397         for s in SNR_dB_range:
398             Pfa_aux, Pd_aux = simular_Pd(p,s,n)
399             Pfa_Simulada.append(Pfa_aux)
400             Pd_Simulada.append(Pd_aux)
401
402         pylab.figure(1)
403
404         pylab.plot(SNR_dB_range, Pd_teorica, ':', label='Pd teorica' + ' P')
405         pylab.plot(SNR_dB_range, Pd_Simulada, 'v', label='Pd simulada' + ' P')
406         pylab.legend(loc='upper left')
407
408         pylab.figure(2)
409         pylab.plot(SNR_dB_range, Pfa_Simulada, '^', label='Pfa simulada' + ' P')
410
411
412
413 pylab.figure(1)
414 pylab.xlabel('SNR_dB')
415 pylab.ylabel('Pd')
416 pylab.title('Pd x SNR_dB')
417 axes = pylab.gca()
418 axes.set_ylim([0.0, 1.0])
419 pylab.grid()
420
421 pylab.figure(2)
422 Pfa_teorica = [Pfa_range for s in SNR_dB_range]
423 pylab.plot(SNR_dB_range, Pfa_teorica, ':', label='Pfa teorica')
424 pylab.legend(loc='upper right')
425 pylab.xlabel('SNR_dB')
426 pylab.ylabel('Pfa')
427 pylab.title('Pfa x SNR_dB')
428 axes = pylab.gca()
429 #axes.set_xlim([-30.0, 0.0])
430 axes.set_ylim([0.0, 1.0])
431 pylab.grid()
432
433 pylab.show()
434

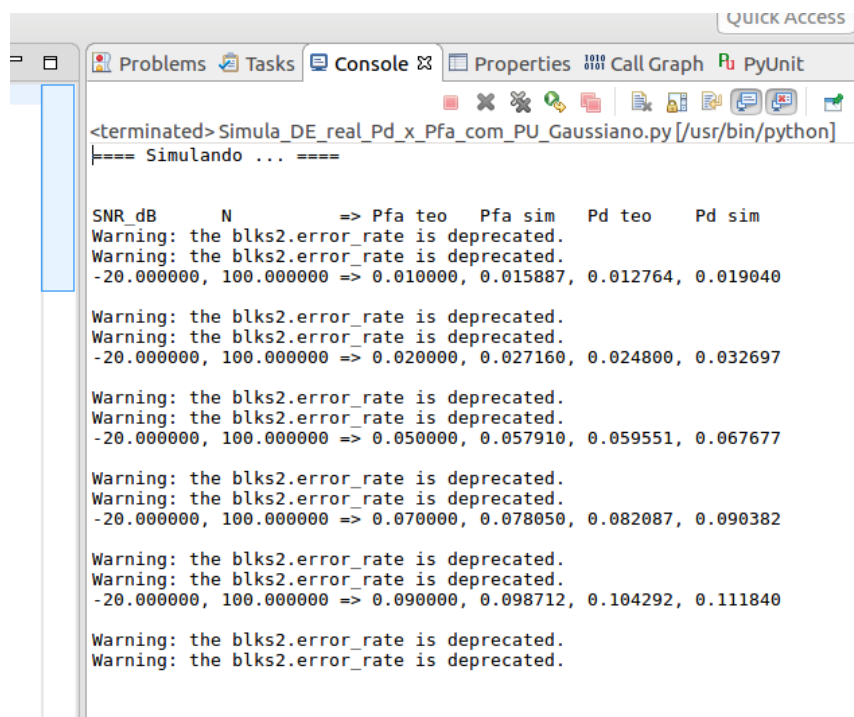
```

Figura 100 - Script que automatiza o varrimento dos cenários (Pfa, SNR, N)

No *script* editado, vemos que o trabalho de automação propriamente dito é realizado no método `__main__` onde são definidos os valores dos cenários a serem obtidos ( $P_{fa}$ , SNR e N) para as faixas a serem especificadas (por exemplo, podemos pretender os resultados de  $P_d$  teórica e simulada apenas para o caso de  $P_{fa}=0.1$  e cinco curvas de variação de  $P_d$  em função de SNR para  $N=100, 1000, 2500, 5000$  e  $10000$  amostras) e é realizado um loop simulando um a um os fluxogramas para cada valor de SNR definido e para cada curva de N amostras, obtendo as  $P_d$  teórica e simulada.

Efetando o varrimento para as configurações pretendidas, podemos obter os vetores com os resultados pretendidos e traçar as curvas teóricas e de simulação que são apresentadas na seção de resultados.

Destacamos também que na função `simular_Pd()` são imprimidos os valores obtidos para a console do Eclipse e dessa forma obtemos uma espécie de *log* da simulação com a apresentação numérica dos resultados referente às curvas obtidas.



```

<terminated> Simula_DE_real_Pd_x_Pfa_com_PU_Gaussiano.py [/usr/bin/python]
==== Simulando ... ====

SNR_dB      N      => Pfa teo  Pfa sim  Pd teo  Pd sim
Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.010000, 0.015887, 0.012764, 0.019040

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.020000, 0.027160, 0.024800, 0.032697

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.050000, 0.057910, 0.059551, 0.067677

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.070000, 0.078050, 0.082087, 0.090382

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.
-20.000000, 100.000000 => 0.090000, 0.098712, 0.104292, 0.111840

Warning: the blks2.error_rate is deprecated.
Warning: the blks2.error_rate is deprecated.

```

Figura 101 - Janela console exibindo o Log com os resultados da simulação

Portanto, tendo customizado o *script* gerado para automatizar a coleta dos resultados pretendidos, podemos ordenar a sua execução e visualizar os resultados

obtidos por meio das mensagens no console e dos gráficos gerados, como por exemplo, o gráfico de uma simulação para obter a Probabilidade de detecção em função da relação sinal-ruído para os vários cenários das  $N$  amostras consideradas (no caso para  $N=100, 250, 500, 1000$  e  $5000$  amostras).

## E Automação dos Fluxogramas de Ensaio

Tal como no caso dos *scripts* de simulação, podemos também neste caso voltar o foco à solução (fluxograma GRC) criado para se obter as estimativas para as probabilidades de detecção e falso alarme que servirão como critério de desempenho ao analisar o detector em causa em condição de testes reais.

Da mesma forma que no caso dos *scripts* de simulação, utilizamos o botão “Generate the flowgraph” e que, ao ser acionado, gera de imediato o *script* referente ao fluxograma implementado no receptor. Em seguida podemos, também neste caso, inspecionar esse *script* com um editor de texto, ou melhor, fazer uma cópia e importá-la para o eclipse para análise e edição.

```

Ensaio_DE_real_Pd_x_SNR_com_PU_Gaussiano_RX
25 from gnuradio import wxgui
26 from gnuradio.eng_option import eng_option
27 from gnuradio.fft import window
28 from gnuradio.filter import firdes
29 from gnuradio.wxgui import fftsink2
30 from gnuradio.wxgui import forms
31 from gnuradio.wxgui import numbersink2
32 from grc_gnuradio import blks2 as grc_blks2
33 from grc_gnuradio import wxgui as grc_wxgui
34 from optparse import OptionParser
35
36 import math
37 import numpy
38 import myblocks
39 import scipy.special
40 import wx
41 import pylab
42
43 from wx import Sleep, Millisleep
44 import pylab
45
46
47 class Ensaio_DE_real_com_PU_Gaussiano_RX(grc_wxgui.top_block_gui):
351
352
353
354
355 class Ensaio_DE_real_com_PU_Gaussiano_RX_send_SNR_to_TX(grc_wxgui.top_block_gui):
359
360
400
401
402 def Pd_teorica(Pfa, SNR_dB, N):
426
427 def ensaiar_Pd(Pfa, SNR_dB, N):
443
444
445
446 if __name__ == '__main__':
447
448     Pfa_range = range(1, 11)

```

Figura 102 - *Script* de ensaio em RX para automatizar a coleta de Pd e Pf

Após construir o *script* de ensaio no receptor, podemos observar de imediato a definição de dois objetos da classe *grc\_wxgui.top\_block\_gui* instanciados, que são respectivamente:

- *Ensaio\_DE\_real\_com\_PU\_Gaussiano\_RX* – fluxograma de análise de desempenho, tal como tínhamos no caso da simulação; e
- *Ensaio\_DE\_real\_com\_PU\_Gaussiano\_RX\_send\_SNR\_to\_TX* – fluxograma que envia ao notebook que executa o fluxograma de transmissão as amostras com o valor da SNR especificada em cada ponto durante o ensaio.

```

446 if __name__ == '__main__':
447
448     Pfa_range = [0.1]
449     SNR_dB_range = [-25.0, -22.5, -20.0, -19.0, -18.0, -17.0, -16.0, -15.0, -14.0, -13.0]
450     N_range = [100, 1000, 2500, 5000, 10000]
451
452     print "==== Ensaio via USRPs ... ====\n"
453     print "\nSNR_dB      N      => Pfa teo   Pfa ens.   Pd teo   Pd ensaio   PU estia
454
455
456     tb_send_SNR_to_TX = Ensaio_DE_real_com_PU_Gaussiano_RX_send_SNR_to_TX()
457     tb_send_SNR_to_TX.start()
458
459
460     for p in Pfa_range:
461         for n in N_range:
462             Pd_teorica = [Pd_teoria(p,s,n)          for s in SNR_dB_range]
463             Pfa_Simulada = []
464             Pd_Simulada = []
465             for s in SNR_dB_range:
466
467                 s_corr=s
468                 if (s== -9.0):
469                     s_corr=-8.9
470                 if (s== -8.0):
471                     s_corr=-7.8
472                 if (s== -7.0):
473                     s_corr=-6.4
474                 if (s== -6.0):
475                     s_corr=-4.7
476                 if (s== -5.0):
477                     s_corr=-2.0
478
479                 tb_send_SNR_to_TX.set_SNR_dB(s_corr)
480
481
482             if s_corr==s:
483                 print ("Mudando para SNR=%.1f no PU" % (s))
484             else:
485                 print ("Mudando para SNR=%.1f (%.1f) no PU" % (s, s_corr))

```

Figura 103 - Fluxograma envia SNR requerida a TX com correção de ganho

A execução do ensaio propriamente dito se inicia definido o *range* dos cenários considerados, por exemplo, definindo uma  $Pfa=0.1$  e traçando cinco curvas de  $Pd$  versus SNR para  $N=100, 1000, 2500, 5000$  e  $10000$  amostras..

Conforme a Figura 103, o fluxograma que envia demanda do valor de SNR para o fluxograma TX no transmissor é instanciado e colocado em execução com o método *start()* e o valor do SNR especificado é enviado para por meio do parâmetro em *s\_corr*.

Percebe-se também no código que há um fator de correção aplicado à SNR solicitada, que ocorre devido a não linearidade no ganhos da USRPs. Como exemplo, podemos observar que para atingir no receptor um sinal com uma SNR de -5 dB é necessário solicitar uma SNR de -2 dB ao transmissor, ou seja, praticamente o dobro da potência. Esse fator em geral ocorre devido a fenômenos como compressão de ganho o que é uma característica intrínseca e até uma figura de mérito própria dos amplificadores de sinal, como os que são utilizados nas USRPs.

O restante da execução do ensaio corre praticamente da mesma forma como no caso das simulações, onde ocorre o varrimento dos parâmetros Pfa, SNR ou N em função das curvas ROC demandadas para análise do detector, e portanto, efetuando esse varrimento para os cenários (Pfa, SNR e N) especificados, podemos obter os vetores com os resultados de Pd e Pfa pretendidos e traçar as curvas teóricas e de ensaio que são apresentadas nos resultados.

Tal como no caso da simulação, destacamos também que na função *simular\_Pd()* são imprimidos os valores obtidos para a console do Eclipse e dessa forma obtemos uma espécie de *log* do ensaio com a apresentação numérica dos resultados referente às curvas obtidas. Esses *logs* também são disponibilizados em anexo exibindo os resultados obtidos.

Em resumo, partindo dos fluxogramas projetados para os ensaios, estes diferem dos fluxogramas das simulações fundamentalmente:

- na separação dos fluxogramas em um para transmissão e outro para recepção/detecção/análise do sinal recebido;
- na inclusão dos blocos *UHD USRP Source* e *UHD USRP Sink* referentes à utilização das USRP como dispositivos de transmissão e recepção; e
- na inclusão dos blocos *TCP Sink* e *TCP Source* para fornecer a partir do *script* de ensaio/análise que é executado no receptor, indicar ao fluxograma de geração do sinal de usuário primário qual deve ser a SNR utilizada em função da configuração (Pfa, SNR, N) demandada pelo *script* no receptor.

Podemos observar também que a mesma estratégia é adotada tanto para os *scripts* de simulação como para os de ensaio para os diversos cenários analisados, os quais são transmissão e recepção via duas USRP conectadas por meio de:

- cabo coaxial e atenuador de 30 dB (conforme especificação do próprio fabricante é necessário usar um atenuador de no mínimo 30 dB à entrada, em caso de conexão via cabo coaxial, de forma a proteger a entrada de RF da USRP de valores acima dos -10dBm de potência máxima do sinal à entrada permitida);
- antenas com blindagem externa para prevenir interferências; e
- antenas em aberto.