Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

DINART DUARTE BRAGA

DESENVOLVIMENTO DE SOFTWARE PARA ANÁLISE GRÁFICA EM TEMPO REAL DE SISTEMAS DE COMUNICAÇÃO DIGITAL DEFINIDOS POR SOFTWARE

Campina Grande, Paraíba
Julho de 2011

DINART DUARTE BRAGA

DESENVOLVIMENTO DE SOFTWARE PARA ANÁLISE GRÁFICA EM TEMPO REAL DE SISTEMAS DE COMUNICAÇÃO DIGITAL DEFINIDOS POR SOFTWARE

Relatório de estágio supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Telecomunicações

Orientador:
Prof. Dr. Edmar Candeia Gurjão

Campina Grande, Paraíba
Julho de 2011

DINART DUARTE BRAGA

DESENVOLVIMENTO DE SOFTWARE PARA ANÁLISE GRÁFICA EM TEMPO REAL DE SISTEMAS DE COMUNICAÇÃO DIGITAL DEFINIDOS POR SOFTWARE

Relatório de estágio supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Telecomunicações

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Dr. Edmar Candeia Gurjão
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho aos meus amigos, que tanto me alegraram nestes cinco anos de curso.

AGRADECIMENTOS

Agradeço a minha família, por ter proporcionado alicerce para a minha formação pessoal e intelectual. Em especial, agradeço a minha mãe, Elita, que sempre me serviu como exemplo de humanidade, caráter, perseverança e profissionalismo, além de ter dado todo o apoio necessário para que eu concluísse o curso de engenharia elétrica.

Agradeço também a todos aqueles que fazem parte do Departamento de Engenharia Elétrica da UFCG, especialmente aos professores que exercem a atividade de docência com paixão, que entram na sala de aula com entusiasmo de ensinar algo novo aos seus alunos.

Dentre esses professores, agradeço particularmente ao professor Edmar Candeia Gurjão, que foi meu orientador, no sentido mais amplo da palavra, desde o meu primeiro ano de curso até a conclusão. Agradeço também por ter me dado a oportunidade de estagiar no Laboratório de Automação e Processamento de Sinais (LAPS) para realização deste trabalho.

Gostaria também de agradecer a todos os colegas de curso que compartilharam comigo os muitos momentos de dificuldades e também de alegrias que o curso proporcionou.

Em particular, agradeço a todos os outros integrantes do LAPS que me ajudaram, seja de forma técnica ou simplesmente conversando trivialidades entre uma seção de programação e outra.

Enfim, agradeço a todas as pessoas que contribuíram positivamente para a construção da pessoa que sou hoje.

Poeminha do Contra

Todos estes que aí estão
Atravancando o meu caminho,
Eles passarão.
Eu passarinho!

-Mario Quintana

RESUMO

O paradigma de sistemas de comunicação baseados em rádio definido por software (SDR) vivencia um rápido crescimento do interesse por parte da academia e da indústria. Uma das instituições brasileiras que atuam nesta área é a UFCG no Laboratório de Automação e Processamento de Sinais (LAPS), onde este trabalho de estágio foi desenvolvido. Uma parcela considerável dos projetos desenvolvidos do LAPS utilizam modulações digitais, mais especificamente a modulação PSK. Neste trabalho é apresentado um software para realização da avaliação do desempenho do modulador/demodulador PSK referenciado por meio da visualização gráfica das características do sistema de comunicação. Um modelo de canal realista também foi desenvolvido para utilização do software sem a necessidade de hardware de propósito específico. O modelo possui duas aplicações interessantes: análise preliminar de desempenho de sistemas SDR e utilização para fins educacionais.

Palavras-chave: Rádio, Software, PSK, Desempenho, Gráfica, GNU Radio, SDR.

ABSTRACT

The paradigm of communications systems based on software defined radio (SDR) experiments a fast interest growth from academy and industry. One of the Brazilian institutes working in this area is the UFCG in the Automation and Signal Processing Laboratory (LAPS), where this internship was developed. A great deal of the projects being developed in LAPS use digital modulation and demodulation schemes, more specifically, PSK modulations. In this work, a software is presented to perform evaluations of the PSK modulator/demodulator used trough graphic visualization of the communication system characteristics. It was also developed a fairly realistic channel model so the software can be used without the need of specific-purpose hardware. The model has two interesting applications: preliminary performance analysis of SDR systems and use for educational purposes.

Palavras-chave: Radio, Software, PSK, Perfomance, Graphical, GNU Radio, SDR.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

SDR – Software Defined Radio

UFCG – Universidade Federal de Campina Grande

LAPS – Laboratório de Processamento Digital de Sinais

PSK – Phase Shift Keying

QAM – Quadrature Amplitude Modulation

DSP – Digital Signal Processor

UAEE – Unidade Acadêmica de Engenharia Elétrica

USRP – Universal Software Radio Peripheral

RF – Radiofrequência

AD – Analógico-Digital

DA – Digital-Analógico

BPSK – Binary Phase Shift Keying

QPSK – Quaternary Phase Shift Keying

8PSK – 8 Phase Shift Keying

ISI – Inter Symbol Interference

DC – Direct Current

DBPSK – Differential Binary Phase Shift Keying

DQPSK – Differential Quaternary Phase Shift Keying

D8PSK – Differential 8 Phase Shift Keying

PLL – Phased Locked Loop

DEP – Densidade Espectral de Potência

AWGN – Additive White Gaussian Noise

OFDM – Orthogonal Frequency Division Multiplexing

BER – Bit Error Rate

LFSR – Linear Feedback Shift Register

SNR – Signal to Noise Ratio

PC – Personal Computer

GUI – Graphic User Interface

GRC – GNU Radio Companion

IDE – Integrated Development Environment

URL – Uniform Resource Location

FFT – Fast Fourier Transform

SUMÁRIO

1 INTRODUÇÃO

O rádio definido por software (SDR) é um paradigma utilizado em sistemas de comunicação por radiofrequências em que a tarefa de processar alguns é atribuída a um processador programável digital.

Atualmente, esse paradigma vem ganhando cada vez mais popularidade devido a sua grande flexibilidade e baixo custo de desenvolvimento. Percebendo essa tendência, desde 2008 o Laboratório de Automação e Processamento de Sinais (LAPS), laboratório ligado ao curso de Engenharia elétrica da UFCG, adquiriu os componentes necessários para desenvolvimento de projetos nesse campo de estudos.

Desde então, projetos como um sistema de reconhecimento automático de modulação [1] têm sido concluídos. Atualmente, estão sendo desenvolvidos alguns projetos em paralelo e boa parte deles precisa ou precisará, em etapa futura, utilizar esquemas de modulação digital.

Até o momento da apresentação deste trabalho, o único esquema de modulação digital totalmente funcional no framework de desenvolvimento da etapa de software de sistemas SDR, o GNU Radio, é o PSK diferencial. Desta forma, ele é extensivamente utilizando em muitos dos projetos do GNU Radio, no entanto, nenhum estudo mais apurado do comportamento deste esquema de modulação foi feito até então, sendo este, portanto, um ótimo tema para exploração.

Outra motivação do trabalho é o interessante campo do radio definido por software, que reúne as áreas de programação e telecomunicações de forma bastante interessante, sendo muito válido um estudo mais detalhado desse tipo de sistema de comunicação.

Para fins de análise de desempenho de um sistema de comunicação que utiliza o modulador e demodulador PSK implementado no GNU Radio, foi desenvolvido, durante o estágio, um software capaz de mostrar graficamente parâmetros de desempenho deste sistema.

Devido às limitações como ausência de antenas que operassem adequadamente com o hardware presente no laboratório e também para fins de desenvolvimento rápido, foi desenvolvido um modelo de canal que considera fontes de degradação como ruído,

limitação em largura de banda e desvanecimento. As intensidades de todas essas degradações podem ser ajustadas em tempo real pelo usuário por meio de uma interface gráfica bastante intuitiva.

A visualização gráfica dos parâmetros do sistema de comunicação, mediante alterações feitas em tempo real nas características do canal, resultou em um aspecto bastante lúdico, possibilitando que o software seja utilizado em sala de aula. Em virtude disso, o desenvolvimento do software foi permeado pela ideia de se manter a interface limpa e sem necessidade de ajustes finos feitos pelo usuário. Outra preocupação foi manter o programa leve, capaz de ser executado na maioria dos computadores atuais sem perda da qualidade.

1.1 O LABORATÓRIO

O estágio foi realizado no Laboratório de Automação e Processamento de Sinais (LAPS), que é um laboratório de pesquisa da Unidade Acadêmica de Engenharia Elétrica (UAEE) da Universidade Federal de Campina Grande (UFCG) e conta com alunos de graduação, mestrado e doutorado atuando em diferentes áreas de processamento de sinais, sendo as duas áreas de maior concentração de pesquisa:

- Utilização de Processadores Digitais de Sinais (DSPs)
- Rádio Definido por Software (SDR)

O laboratório conta com algumas placas de desenvolvimento equipados com DPSs da Texas Instruments®, que são utilizadas tanto em atividades de pesquisa como para auxílio de aprendizado dos alunos da disciplina de Processamento Digital de Sinais.

Atualmente, a linha de pesquisa com maior atividade no LAPS é a de radio definido por software. O laboratório conta com alunos de graduação desenvolvendo diversos projetos que envolvem de alguma forma sistemas SDR, seja em aplicações como rádio cognitivo, modulação adaptativa ou antenas inteligentes.

Como infraestrutura para o desenvolvimento de sistemas baseados em software, o laboratório possui algumas unidades do USRP (Universal Software Radio Peripheral), que são peças de hardware necessárias para realização da interface RF entre o programa

de computador e o meio físico. Além disso, também conta com pequenas placas filhas que são utilizadas na operação do USRP em diferentes faixas do espectro eletromagnético.

1.2 ASPECTOS GERAIS DO ESTÁGIO

O estágio a qual esse texto se refere foi um estágio supervisionado realizado durante um período de 120h. Devido à necessidade de uma abordagem teórica não vista durante a graduação, 20h foram dedicadas a estudos relacionados ao tema, alguns destes apresentados na revisão bibliográfica deste trabalho. Já para aprendizado e revisão dos conceitos relacionados à utilização do GNU Radio, software que fornece ferramentas para implementação de sistemas SDR, foram reservadas 40h. Finalmente, o desenvolvimento em si do software apresentado neste relatório foi feito durante as 60h restantes.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção é feita uma revisão bibliográfica dos pontos mais importantes da teoria que foram utilizados durante o desenvolvimento do trabalho de estágio.

2.1 RÁDIO DEFINIDO POR SOFTWARE

O termo rádio definido por software foi cunhado em 1991 por Joseph Mitola, que publicou o primeiro trabalho científico na área [2]. Neste trabalho é definido um sistema de comunicação em que praticamente todas as atividades de processamento dos sinais, recebidos ou enviados, são realizadas por um processador programável, isto é: em software.

Geralmente, os processadores utilizados são processadores digitais de sinais ou processadores de arquitetura x86, presentes em computadores pessoais [3]. O uso de software, ao invés de hardware de propósito especifico, possui algumas vantagens, como baixo custo, flexibilidade dos sistemas e possibilidade de obtenção de sistemas de comunicação totalmente diferentes somente por alterações de software.

Idealmente, um sistema com rádio definido por software é composto por uma antena, um conversor analógico digital (AD) ou digital analógico (DA), dependendo se está operando como receptor ou transmissor, e um processador digital na qual o software que realiza o processamento do sinal é instalado.

Esse sistema não é realizável por alguns motivos, talvez o mais importante seja a necessidade de um misturador (mixer) para realizar o deslocamento do espectro do sinal recebido para baixas frequências ou do sinal enviado para altas frequências. Isso é necessário, pois as frequências de RF podem ser da ordem de GHz, e conversores AD e DA não operam nessas frequências e mesmo que operassem, a taxa de bits necessária do lado digital do conversor poderia facilmente exceder a barreira de 100Gb/s, algo dificilmente alcançável em sistemas eletrônicos.

Por esse motivo, a maioria dos sistemas baseados em rádio definido por software trabalha com a representação em banda básica complexa dos sinais passa faixa a serem enviados.

Outra limitação de ordem prática dos sistemas SDR é a necessidade de inclusão de um filtro passa faixa no seu circuito de recepção, a fim de remover as componentes espectrais de ruído e interferência presentes em bandas de frequência que não pertencem ao sinal desejado. No entanto, a adição desse filtro reduz a flexibilidade do sistema, que é um dos pontos mais importantes do rádio definido por software.

Além da necessidade de um filtro específico para cada banda, antenas também são fabricadas para operar em faixas de frequências específicas. Essa característica limita ainda mais os sistemas SDR, uma vez que já existem muitas soluções de filtros sintonizáveis, mas pouco progresso na área de antenas sintonizáveis.

Em geral, um frontend de um sistema SDR capaz de operar em várias frequências possui diferentes peças de hardware para cada faixa de frequência, possibilitando que este se adeque às diferentes características de cada faixa.

O esquema de um receptor de um sistema de radio definido por software é mostrado na Figura 1. O significado dos componentes Q e I do sinal são abordados com detalhe posteriormente.

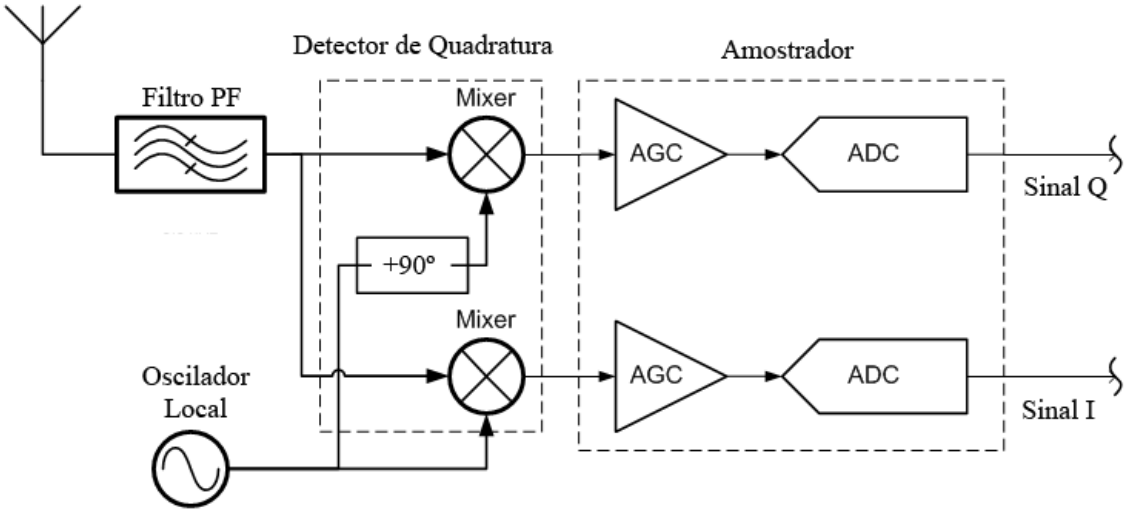

Figura 1 – Diagrama de blocos de um receptor de sistema SDR.

2.2 SINAIS EM BANDA BÁSICA COMPLEXA

Um sinal passa faixa, isto é, com conteúdo espectral concentrado em torno de um valor elevado de frequência, f_0, e nulo para as outras frequências (Figura 2), não é prático para ser processado digitalmente, pois o critério de Nyquist para amostragem

desse sinal afirma que o sinal deve ser amostrado com pelo menos o dobro da maior componente de frequência presente, que para um sinal RF pode ser da ordem de GHz.

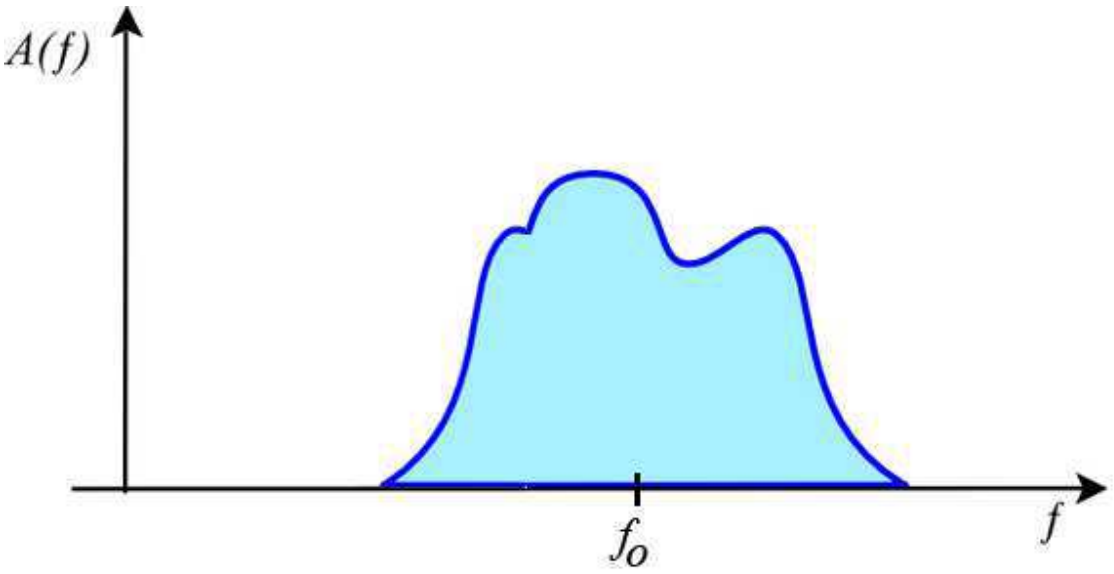

Figura 2 – Parte positiva de espectro de frequência de um sinal passa faixa.

No entanto, é possível representar esse sinal em termos de um sinal de baixa frequência e complexo [4], que é chamado equivalente passa baixa ou sinal em banda básica complexa. O mais interessante é que essa representação preserva toda a informação presente no sinal, permitindo que processadores de sinais operem nesse domínio.

Como o espectro de frequência, $A(f)$, de um sinal real $a(t)$ possui módulo par e fase ímpar, o conhecimento de $A(f)$ somente para valores $f \geq 0$ é suficiente para se obter toda a informação do sinal.

Dessa forma, é útil definir duas grandezas de interesse para essa análise, são elas:

$$A_+(f) = \begin{cases} A(f), \ \ para\ f > 0 \\ \frac{1}{2}A(0), \ \ para\ f = 0 \\ 0, \ \ para\ f < 0 \end{cases} \tag{1}$$

e

$$A_-(f) = \begin{cases} A(f), \ \ para\ f < 0 \\ \frac{1}{2}A(0), \ \ para\ f = 0 \\ 0, \ \ para\ f > 0 \end{cases}. \tag{2}$$

Isto é, $A_+(f)$ é a parte positiva da resposta em frequência e $A_-(f)$ é a parte negativa. Consequentemente, a soma das duas funções resulta em $A(f)$. Uma forma mais compacta de escrever as duas parcelas é

$$A_+(f) = A(f)u(f) \tag{3}$$

e

$$A_-(f) = A(f)u(-f) \tag{4}$$

Para um sinal real, $A_-(f) = A_+^*(-f)$, de forma que o conhecimento de apenas metade do espectro é suficiente para determinação do sinal, como comentado anteriormente. O pré-envolpe de $a(t)$ é definido como:

$$\begin{aligned} \mathrm{a}_+(t) &= \mathcal{F}^{-1}[A_+(f)] \\ &= \mathcal{F}^{-1}[A(f)u(t)] \\ &= a(t) * \left(\frac{1}{2}\delta(t) + \frac{j}{2\pi t}\right) \\ &= \frac{1}{2}a(t) + \frac{j}{2}\hat{\mathrm{a}}(t), \end{aligned} \tag{5}$$

onde $\hat{\mathrm{a}}(t)$ é a transformada de Hilbert de $a(t)$ e é definida como

$$\hat{\mathrm{a}}(t) = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{a(\tau)}{t-\tau} d\tau$$

Define-se $a_l(t)$, o equivalente passa baixa, como o sinal que tem espectro dado por $2A_+(f + f_0)$. O espectro deste sinal, em termos do sinal mostrado na Figura 2, pode ser visto na Figura 3. Note que o espectro de $A_l(f)$ não possui mais simetria par, e portanto, em geral, o sinal $a_l(f)$ não é real.

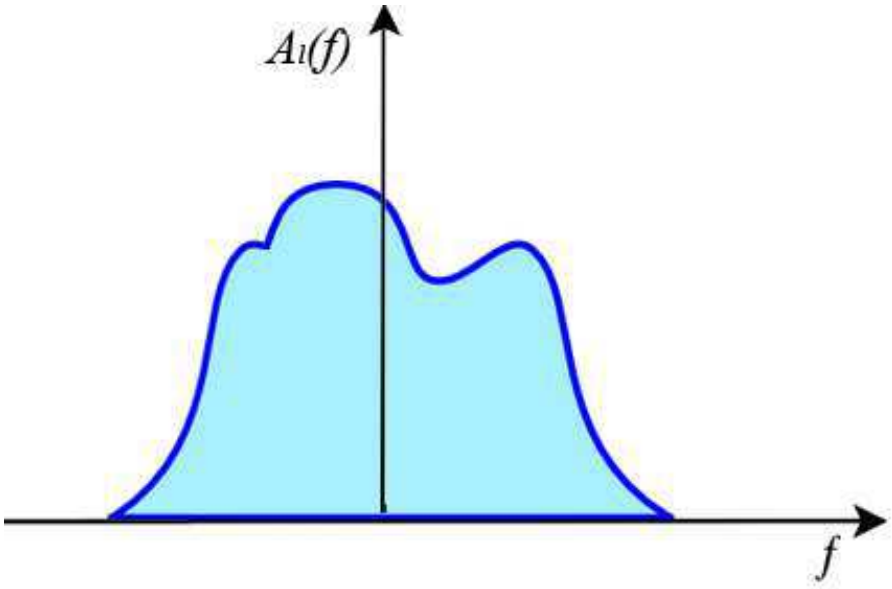

Figura 3 – Resposta em frequência de $A_l(f)$.

O sinal $a_l(t)$ no tempo pode ser determinado a partir da transformada inversa de Fourier, resultando em

$$\begin{aligned} a_l(t) &= \mathcal{F}^{-1}\{A_l(f)\} \\ &= 2a_+(t)e^{-j2\pi f_0 t} \\ &= (\, a(t) + j\hat{a}(t)\,)e^{-j2\pi f_0 t} \end{aligned} \tag{6}$$

Da Equação (6), fica claro que podemos expressar $a(t)$ em função de $a_l(t)$ da seguinte forma:

$$a(t) = Re\left[a_l(t).e^{j2\pi f_0 t}\right] \tag{7}$$

A parte real de $a_l(t)$ é comumente chamada de componente em fase, $a_i(t)$, e a parte imaginária é chamada de componente em quadratura, $a_q(t)$. De forma que $a_l(t)$ pode ser expresso como:

$$a_l(t) = a_i(t) + ja_q(t) \tag{8}$$

Já $a(t)$ pode ser expressa a partir de $a_i(t)$ e $a_q(t)$ na seguinte forma

$$a(t) = a_i(t)\cos(2\pi f_0 t) - a_q(t)sen(2\pi f_0 t) \quad (9)$$

Portanto, é possível reconstruir o sinal em passa faixa a partir do sinal em fase e em quadratura correspondente. Como esses sinais são de baixa frequência, a utilização do equivalente passa baixa é adequada em sistemas de rádio definido por software, uma vez que a sua amostragem é mais factível.

2.3 MODULAÇÃO PSK DIFERENCIAL

A modulação PSK é um esquema de modulação digital no qual a informação de uma fonte é mapeada na fase de uma onda portadora, a qual pode assumir um número finito, M, de valores. Geralmente $M = 2$, dando origem ao Binary PSK (BPSK), $M = 4$, para o Quaternary PSK (QPSK) ou $M = 8$ para 8PSK.

Cada um dos sinais utilizados por um modulador PSK é uma senóide de mesma frequência e amplitude, mas com a fase indexada pelo símbolo a qual ela está codificando.

As formas de onda utilizadas por um modulador M-PSK são dadas por:

$$s_m(t) = \left[\cos\left(\frac{2\pi}{M}(m-1)\right)\cos(2\pi f_0 t) - \operatorname{sen}\left(\frac{2\pi}{M}(m-1)\right)sen(2\pi f_0 t)\right]p(t)\,, \quad (10)$$

para $m = 1,2,\ldots,M$. Em que f_0 é a frequência da portadora.

A função $p(t)$ representa um pulso conformador de sinale é utilizado para reduzir a interferência entre símbolos (ISI) do sinal modulado, assim como reduzir a banda do sinal. Um dos filtros conformadores de pulso mais utilizados foi proposto por Nyquist e é o filtro do cosseno levantado [5].

Comparando a Equação (10) com a Equação (9) é fácil notar que as partes em fase e em quadratura de um sinal modulado digitalmente podem ser determinadas por uma simples inspeção da Equação (10) e são iguais a

$$s_{mi}(t) = cos\left(\frac{2\pi}{M}(m-1)\right)p(t) \tag{11}$$

$$s_{mq}(t) = sen\left(\frac{2\pi}{M}(m-1)\right)p(t) \tag{12}$$

em que $m = 1,2,\dots,M$.

Portanto, a tarefa de modulação PSK em um sinal expresso por seu equivalente de banda básica complexa é bastante simplificada. Se não for utilizado um pulso conformador, a modulação será apenas a mudança periódica dos dois sinais para níveis DC dados pelas Equações (11) e (12).

Se o modulador for implementado em software, como é o caso dos moduladores do GNU Radio, o sinal deixa de ser função do tempo contínuo e passa a ser representado por uma sequência numérica. Neste caso, os sinais em fase em quadratura são representados por um número de amostras, N_a, amostras estas que serão utilizadas na reconstrução dos sinais $s_{mi}(t)$ e $s_{mq}(t)$, que são utilizados para sintetização de $s_m(t)$, sinal em banda passante. Os sinais em fase e em quadratura digitais podem ser como

$$s_{mi}[n] = cos\left(\frac{2\pi}{M}(m-1)\right)p[n], \qquad n = 1,\dots,N_a \tag{13}$$

$$s_{mq}[n] = sen\left(\frac{2\pi}{M}(m-1)\right)p[n], \qquad n = 1,\dots,N_a \tag{14}$$

Os possíveis valores para as componentes em fase e em quadratura para os três tipos de modulação mais comuns estão representados nas Figuras 4,5 e 6. Considera-se um pulso retangular.

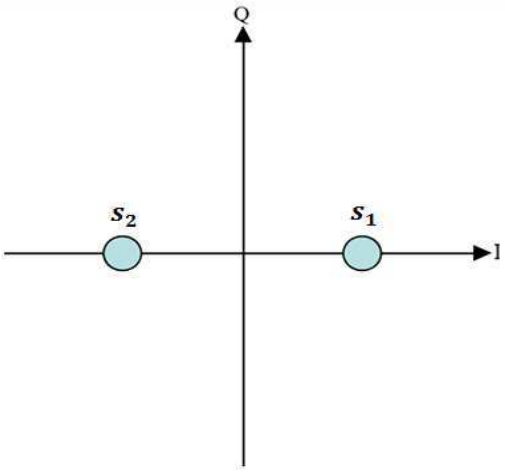

Figura 4 – Diagrama de constelação BPSK.

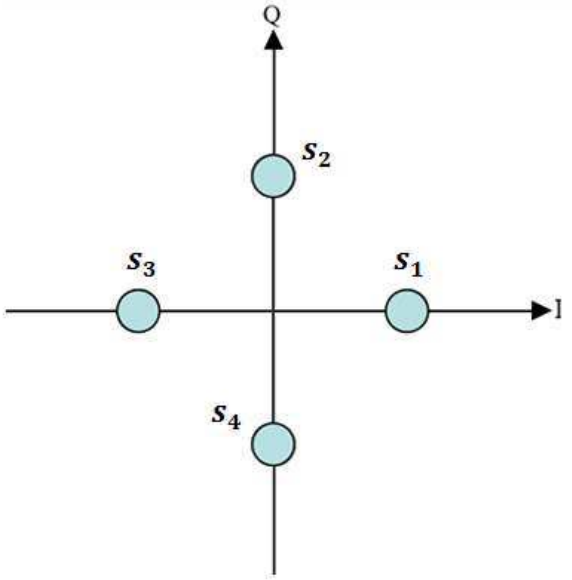

Figura 5 - Diagrama de constelação QPSK.

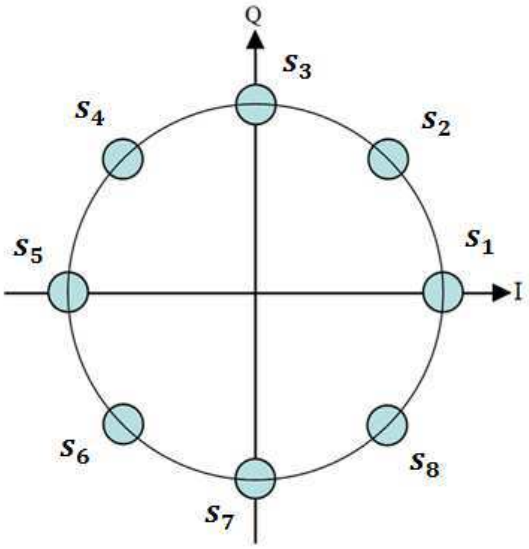

Figura 6 - Diagrama de constelação 8PSK.

Quando os símbolos são usados diretamente para indexar a fase da portadora, é necessário fazer uma reconstrução exata da portadora no receptor, ou seja, é necessário realizar uma demodulação coerente. A reconstrução da portadora não é uma tarefa fácil, sendo muitas vezes preferível utilizar uma modulação não coerente.

A modulação PSK pode se tornar não coerente se ao invés de utilizar os símbolos para indexar a fase, utilizá-los para indexar mudanças na fase. Por exemplo, na modulação BPSK, o símbolo 0 indexa 0º e o símbolo 1 indexa 180º, já na modulação BPSK diferencial (DBPSK) o 0 indica que a fase deve permanecer constante e o 1

indica que ela deve ser invertida. Procedimentos similares são utilizados para formar os símbolos das modulações DQPSK e D8PSK.

Embora pareça uma alteração sem sentido, essa alteração torna desnecessário o conhecimento da fase da portadora, uma vez que basta saber a fase relativa entre os símbolos, e também aumenta a probabilidade de erro em aproximadamente duas vezes, uma vez que o erro na detecção de um símbolo quase sempre implica na detecção do bit seguinte. O modulador do GNU Radio possui moduladores e demoduladores DBPSK, DQPSK e D8PSK.

Para realizar a demodulação do sinal DPSK, o GNU Radio utiliza um algoritmo de recuperação de fase derivado do clássico circuito PLL, o Costas Loop [6]. Ele pode ser entendido como uma versão melhorada do PLL clássico, uma vez que possui uma sensibilidade à diferença de fase estimada, θ_e, e fase recebida, θ_r, de $\text{sen}\big(2(\theta_e - \theta_r)\big)$, ao contrário da sensibilidade de $\text{sen}(\theta_e - \theta_r)$ do circuito PLL clássico, isso se traduz em uma sensibilidade duas vezes maior e uma boa estimação da fase.

É importante entender que a atividade de recuperação de sincronização é um passo crítico de um demodulador digital e qualquer falha nessa etapa da demodulação é bastante grave, como será visto mais adiante na apresentação dos resultados.

2.4 DEGRADAÇÕES DE UM CANAL DE COMUNICAÇÃO

Um canal de comunicação é caracterizado pela sua ação no sinal transmitido. São essas degradações que impõem limites à taxa de troca de informação máxima capaz de ser realizada por um sistema de comunicação [7], sendo assim de grande importância o estudo dessas características durante o projeto de um sistema de comunicação.

Algumas das degradações mais significativas em um canal de comunicação são o ruído, distorção/dispersão e o desvanecimento. Cada canal pode possuir intensidades diferentes de cada uma dessas degradações e, portanto, podem possuir características bastante distintas.

A seguir é feita uma descrição destes três limitantes e como eles podem ser modelados matematicamente.

2.4.1 RUÍDO

Um dos principais limitantes dos sistemas de comunicação é o ruído, especialmente o ruído térmico, sendo o único tipo de degradação em alguns tipos de canais como os de comunicação espacial [4].

O ruído térmico possui uma densidade espectral de potencia (DEP) função da temperatura e da frequência, que e é dada por

$$P(f) = \frac{hf}{e^{\frac{hf}{kT}} - 1}, \tag{15}$$

em que h é a constante de Planck, f é a frequência, k é a constante de Boltzmann e T é a temperatura em Kelvin.

Esta equação é praticamente constante e igual a kT para valores de frequência inferiores a 100Ghz, uma discussão mais completa desta aproximação pode ser encontrada em [8]. Como praticamente todos os sistemas de comunicação RF atuais atuam em frequências inferiores a 100Ghz, pode-se assumir que a DEP do ruído é constante, ou seja, pode-se assumir que o ruído é branco.

O modelo mais utilizado na modelagem do ruído em sistemas de comunicação é o do ruído aditivo gaussiano branco (AWGN), que considera o ruído, $n(t)$, como um processo estocástico estacionário em sentido amplo com distribuição gaussiana de média nula e variância $N_0/2$ que é somado ao sinal enviado. Essa operação é ilustrada na Figura 7.

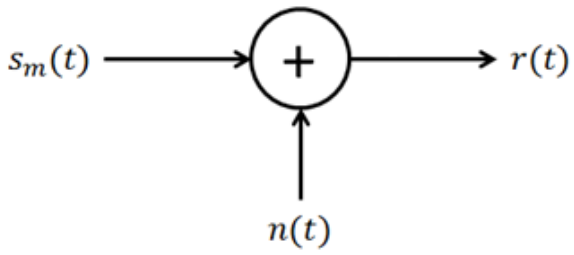

Figura 7 – Operação realizada pelo canal no sinal de entrada.

O efeito desse ruído quando visto de uma perspectiva de um sinal passa baixa equivalente (banda básica complexa), equivale à soma de dois processos estocásticos

gaussianos de média nula, variância $N_0/2$ e independentes, um na componente em fase e outro na componente em quadratura.

2.4.2 Distorção e Dispersão

A distorção linear e a dispersão, que são a caracterizadas pela deformação do sinal enviado e espalhamento do pulso básico para além do seu tempo de sinalização, respectivamente, são dois efeitos de uma mesma limitação dos canais de comunicação: a largura de banda finita e não uniforme. A presença dessas degradações é mostrada na Figura 8.

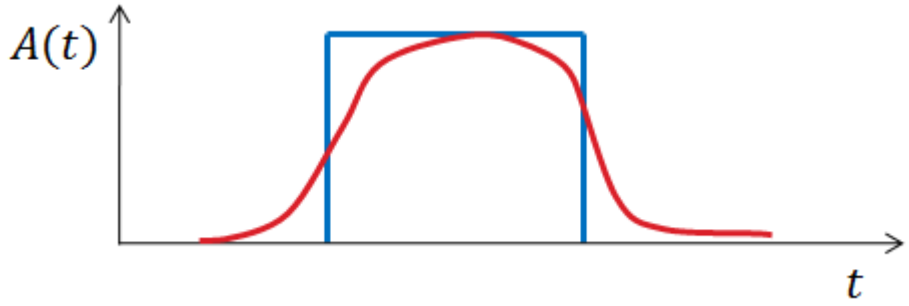

Figura 8 – Distorção de um pulso elétrico ao passar por um canal.

Os meios físicos por onde os sinais de comunicação se propagam possuem limitações com relação às faixas de frequência que podem excitar adequadamente o meio, isto é, as frequências que podem 'passar' pelo canal. A atenuação do canal para diferentes valores de frequência é comumente expressa por meio da resposta em frequência do canal, $H(f)$. Essa informação é essencial, pois é ela que fornece a capacidade de sinalização do sistema de comunicação.

Muitas vezes, não é o meio físico que impõe a limitação de banda, mas sim algum tipo de restrição técnica, especialmente em canais de radio onde o espectro é um recurso bastante caro e os engenheiros devem projetar os seus sistemas para que não utilizem uma banda maior que a adquirida, evitando a interferência em sistemas de comunicação alheios.

Uma das soluções adotadas para ajudar a diminuir a deformação e a distorção dos sinais, evitando interferência entre símbolos, e ao mesmo tempo diminuir a largura de banda dos sinais enviados é a utilização de um filtro conformador de pulso, como o filtro do cosseno levantado, que possui função de transferência dada por

$$P(f) = \begin{cases} T, & |f| \leq \frac{1-\propto}{2T} \\ \frac{T}{2}\left[1 + \cos\left(\frac{\pi T}{\propto}\left[|f| - \frac{1-\propto}{2T}\right]\right)\right], & \frac{1-\propto}{2T} < |f| \frac{1+\propto}{2T}, \\ 0, & caso\ contrário \end{cases} \quad (16)$$

em que $\propto \in [0,1]$ é um parâmetro que controla o excesso de banda utilizado. A largura de banda utilizada por um sinal com taxa de sinalização de símbolo R_s e filtrado por $P(f)$ e igual a

$$B = \frac{1}{2} R_s(\propto +1) \quad (17)$$

2.4.3 DESVANECIMENTO

Os canais de comunicações sem fio são caracterizados por possuírem uma atenuação que varia no tempo, essa característica é denominada desvanecimento. As principais causas deste tipo de degradação são a propagação multipercurso e a perda de visibilidade da onda eletromagnética devido à interposição de obstáculos no meio de propagação [9].

O desvanecimento pode ser classificado como plano ou como seletivo em frequência. No desvanecimento plano, todos os componentes de frequência do sinal transmitido são atenuados com a mesma intensidade, embora variante no tempo. Já no desvanecimento seletivo em frequência, a atenuação varia não só com o tempo, mas também com a frequência, o que gera deformações indesejadas no sinal.

Técnicas desenvolvidas para compensar o desvanecimento seletivo em frequência são utilizadas constantemente em canais de comunicação móvel, como, por exemplo, a OFDM, que consiste em utilização de várias subportadoras de pequena largura de faixa, de tal sorte que cada subportadora está sujeita um desvanecimento localmente plano.

Uma das distribuições mais utilizadas para modelar o desvanecimento é a distribuição de Rayleigh, que pode ser obtido a partir da suposição de que o sinal de rádio recebido é composto de um grande número de frentes de onda que chegam em tempos distintos e com fases diferentes. Essa suposição permite aplicar o teorema central do limite para determinar a resposta do canal, que é modelado por um processo estocástico [4].

Considerando-se um desvanecimento plano, a influência do canal sobre o sinal enviado pode ser modelado pela seguinte equação [9]

$$r_l(t) = g_l(t)s_l(t) \tag{18}$$

em que $r_l(t)$ é o equivalente passa baixa do sinal recebido, $s_l(t)$ é o equivalente passa baixa da mensagem enviada e $g_l(t)$ modela o desvanecimento.

No modelo de desvanecimento de Rayleigh, $g_l(t)$ é um processo estocástico complexo com amplitude caracterizada pela distribuição de Rayleigh e a fase dada por uma distribuição uniforme no intervalo $[0,2\pi]$.

E, em geral, a consequência mais notável nos sistemas de comunicação é a atenuação, que é dada por um processo com distribuição de Rayleigh, o nome desta distribuição também é usado para referenciar o modelo de desvanecimento.

Uma variável aleatória de Rayleigh pode ser obtida pelo cálculo do módulo de uma segunda variável aleatória complexa com parte real e imaginária dadas por variáveis aleatórias normais, independentes, de médias nulas e de mesma variância.

Na Figura 9 pode-se observar a realização de um processo estocástico de Rayleigh durante 10 segundos.

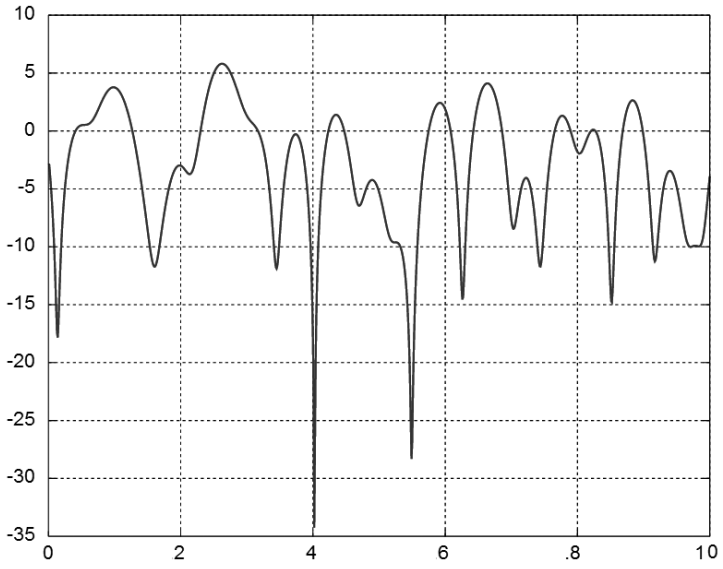

Figura 9 – Ganho do canal (dB) versus tempo (s).

A velocidade com que a atenuação do canal oscila depende da função de autocorrelação do processo $g_l(t)$. Existem muitos modelos para determinação desta função, um dos mais utilizados leva em consideração que o desvanecimento ocorre quando há movimento relativo entre transmissor e receptor, e consequentemente há um

deslocamento de frequência por efeito Doppler associado, f_d. Neste caso, a função de autocorrelação do desvanecimento é dada por

$$R(\tau) = J_0(2\pi f_d \tau) \tag{19}$$

O método de Jakes [9] é bastante utilizado na geração de funções amostradas para simulação de desvanecimento. Ele consiste na geração de uma função amostra a partir do somatório de senoides.

O sinal amostra gerado é da forma

$$g_l(t) = 2\sqrt{2}\left[\sum_{n=1}^{M}\left((\cos\beta_n + j\ sen\ \beta_n)\cos\left(2\pi f_d.\cos\frac{2\pi}{M}t\right)\right) + \frac{1}{\sqrt{2}}\cos\left(2\pi f_d.\cos\frac{2\pi}{M}t\right)\right] \tag{20}$$

Em que M é o número de senoides utilizadas e f_d é a frequência de deslocamento de Doppler.

2.5 MEDIÇÃO DA TAXA DE ERRO DE BIT

Um dos parâmetros de desempenho mais importantes em um sistema de comunicação digital é a probabilidade de erro de bit. Dessa forma, é natural que se queira aferir essa grandeza em uma situação real.

Uma das formas de estimar a probabilidade de erro de bit consiste em enviar uma sequência predeterminada através do canal de comunicação e no receptor comparar a sequência enviada com a sequência recebida, contabilizando o número de erros. A taxa de erro de bits pode ser definida como

$$BER = \frac{N_e}{N}, \tag{21}$$

em que N_e é o número de erros e N é o comprimento da sequência enviada.

Para que a estimativa seja confiável é recomendável que o valor de N seja pelo menos 10 vezes maior que o recíproco de N_e ($N > 10N_e^{-1}$), a custas de realizar uma má estimação caso essa restrição não seja respeitada.

A princípio pode-se imaginar enviar uma sequência composta inteiramente por 1s (ou 0s), de forma que a contagem de erros possa ser feita somente pela contagem dos bits 0s (ou 1s) recebidos. No entanto, essa abordagem não é interessante porque uma sequência constante prejudica a sincronização do sinal no receptor.

Para obter uma boa estimativa da probabilidade de erro de bit, deve-se enviar uma sequência aleatória (ou pseudoaleatória) e realizar o procedimento descrito. Para tal, tanto transmissor quanto receptor devem conhecer a sequência ou então conhecer um algoritmo para gerá-la.

Essa abordagem tem a desvantagem de exigir que se conheça a posição exata da sequência, para que a comparação seja feita entre posições iguais da sequência recebida e enviada. A Figura 10 ilustra uma situação em que essa posição é estimada com sucesso e a comparação é feita entre elementos equivalentes, resultando em uma boa estimativa de taxa de erro de bits. Já a Figura 11 ilustra um caso em que são comparadas sequências deslocadas de apenas 1 bit entre si, nessa situação a taxa de erro de bits estimada será incorreta, geralmente 50% se a sequência for uma boa sequência pseudoaleatória.

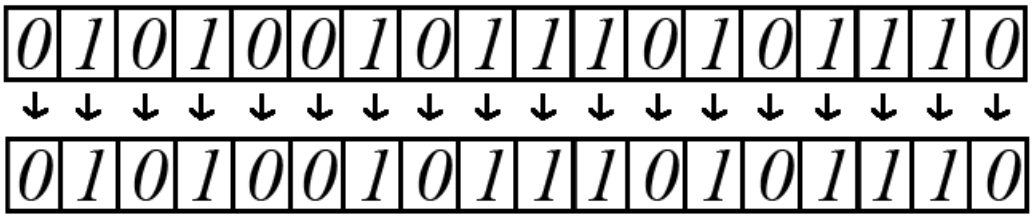

Figura 10 – Posição estimada corretamente.

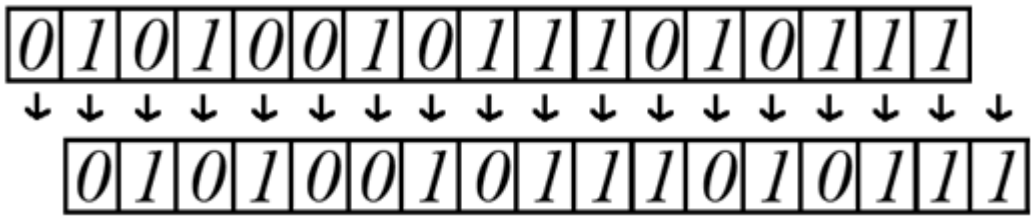

Figura 11 – Posição estimada com um bit de deslocamento.

Para relações sinal ruído (SNR) muito baixas, a sincronização da posição das sequências se torna bastante difícil, especialmente quando o demodulador gera bits

adicionais de forma aleatória para baixas SNR (caso do GNU Radio), dificultando a comparação entre as sequências.

Uma das maneiras de evitar esse problema é utilizando-se um registrador de deslocamento com realimentação linear (LFSR), que pode ser implementado em hardware ou software. Eles possuem a característica de gerar uma sequência pseudoaleatória a partir de um sinal de entrada, sendo comumente utilizados para o embaralhamento de sinais.

Para medição de taxa de erro de bits, pode-se usar um LFSR para embaralhar um sinal composto por uma sequência de 1s, gerando uma sequência pseudoaleatória. No lado receptor o sinal recebido é desembaralhado e o número de 0s na sequência pode ser utilizado para estimar o número de erros.

No entanto essa abordagem possui uma desvantagem, para que a sequência LFSR seja uma boa sequência pseudoaleatória é necessário que a saída seja função de valores passados, isto é, o algoritmo possui memória. Isso faz com que um erro na sequência embaralhada gere mais de um erro na sequência desembaralhada.

Para um pequeno valor de taxa de erro a BER pode ser calculada como o número de 0s na sequência recebida dividido pelo número de bits passados utilizado pelo LFSR para geração do bit atual vezes o número de bits total, isto é

$$BER = \frac{N_0}{N.M} \tag{22}$$

Esta aproximação é válida enquanto os erros ocorrerem de forma espaçada, no entanto, quando os erros começam a se sobrepor, essa aproximação deixa de ser válida. Felizmente um valor de $M = 3$ já é suficiente para gerar uma sequência pseudoaleatória com fins de medição de BER, isso garante uma boa estimativa de probabilidades de erro de bit para valores de até cerca de 10%, que está acima da BER da maioria dos sistemas práticos.

3 FERRAMENTAS UTILIZADAS

3.1 GNU RADIO

O GNU Radio é um software livre que fornece um conjunto de ferramentas necessárias para realização de processamento de sinais em tempo real utilizando processadores de arquitetura x86, que possuem baixo custo e são largamente utilizados em computadores pessoais (PCs). Possui licença GPLv3 e pode ser encontrado de forma gratuita [10].

Ele fornece parte das ferramentas necessárias para implementação da etapa de software em um sistema de comunicação definido por software e é largamente utilizado em ambientes acadêmicos e comerciais, em atividades de apoio ao desenvolvimento de sistemas sem fio e para implementação de diversos sistemas de comunicação utilizados comercialmente.

Alguns exemplos de aplicações desenvolvidas utilizando o GNU Radio são radares, estações rádiobase de tecnologia celular GSM e um modulador em tempo real de televisão digital do padrão DVB-T [11].

As operações de processamento de sinais do GNU Radio são encapsuladas em blocos, que são objetos que possuem entradas, saídas e realizam uma operação determinada. Os blocos podem ser classificados como blocos fonte, blocos intermediários e blocos terminais.

Os blocos fonte só possuem saídas e são responsáveis por gerar sinais, como por exemplo uma onda senoidal ou ruído gaussiano branco. Os blocos intermediários possuem uma ou mais entrada e uma ou mais saídas, são responsáveis por realizar operações nos sinais de entrada e entregar os resultados na saída. Os blocos terminais só possuem entradas e enviam o sinal recebido para algum meio externo, como uma antena, caixa de som do computador ou para visualização no monitor.

Diferentes blocos podem ser interconectados para formar um chamado “bloco hierárquico” (hierblock), é dessa forma que a maioria dos blocos que realizam atividades complexas, como modulação, são implementados no GNU Radio. Essa abordagem é interessante, pois aumenta a modularidade e capacidade de reutilização do

código desenvolvido, além de facilitar a alteração em tempo real de sistemas complexos como moduladores.

A arquitetura do GNU Radio é dividida em duas partes principais. A primeira faz o uso da linguagem de programação C++, enquanto a segunda utiliza a linguagem de script Python. Embora ambas as linguagens compartilhem a característica de serem orientadas a objeto, C++ é uma linguagem compilada enquanto o Python é uma linguagem interpretada. Devido a maior velocidade de programas compilados, o C++ é utilizada para implementação de blocos que efetuam operações de processamento de sinais computacionalmente intensas. Já Python é utilizado para instanciar e conectar os diferentes blocos. O uso de Python é bastante atrativo, pois garante grande versatilidade dos sistemas desenvolvidos, permitindo fácil implementação de sistemas de controle sofisticados.

Uma das desvantagens do GNU Radio é sua documentação é bastante escassa, sendo necessária muitas vezes a procura do código fonte de um bloco para determinar o que este bloco realiza. Esse é um dos fatores que mais dificulta o desenvolvimento utilizando esse toolkit.

3.2 GNU RADIO COMPANION

Devido a sua estrutura baseada em blocos, foi desenvolvida uma interface gráfica do usuário (GUI) para realização da ligação entre os blocos do GNU Radio. Essa interface é muito semelhante a do Simulink, presente no software Matlab®.

Denominado GNU Radio Companion (GRC), esse programa permite que o estabelecimento da estrutura básica dos blocos de um sistema seja feita de forma bastante rápida e intuitiva, na Figura 12 pode ser visto um sistema na qual dois sinais senoidais de frequência distintas são somados a um ruído gaussiano branco e enviados para a saída de áudio do computador.

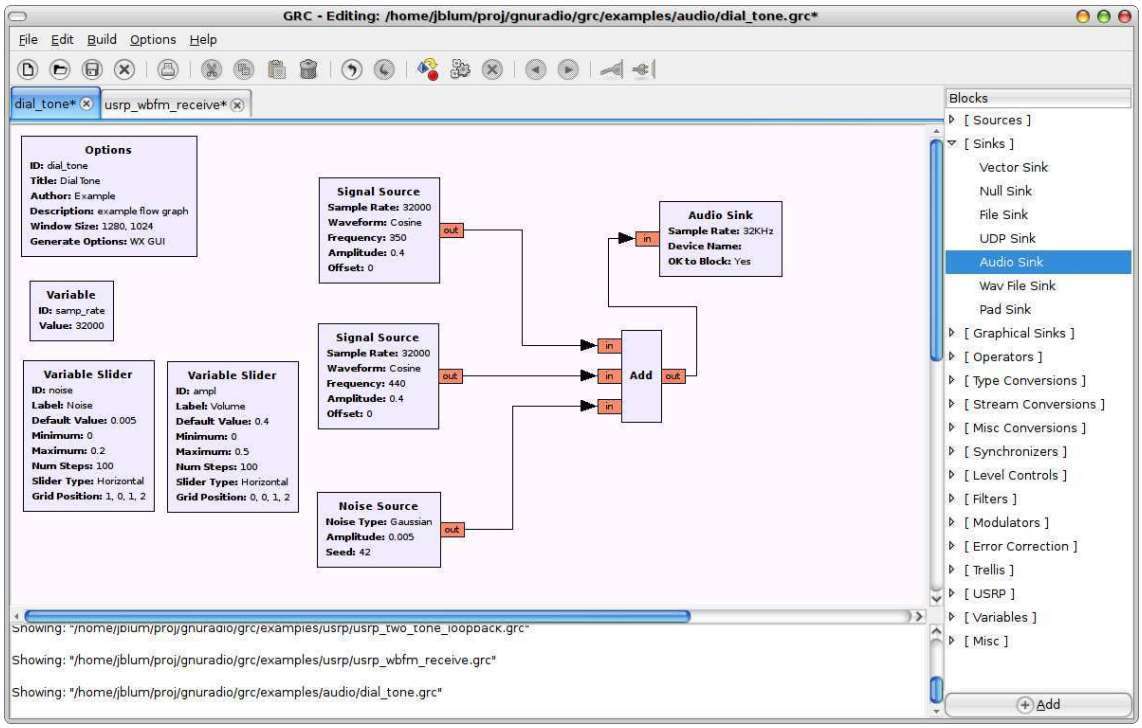

Figura 12 – Exemplo de sistema implementado com auxílio do GRC.

Para criação de um sistema utilizando Python, o usuário deve realizar a ligação entre os blocos de interesse e em seguida escolher a opção “Generate Flowgraph”, que realiza a conversão do diagrama de blocos no código em Python correspondente.

A utilização do GRC é bastante útil em dois cenários, o primeiro deles é para o usuário iniciante, que não conhece bem a estrutura do GNU Radio e está aprendendo a utilizar o toolkit. O usuário pode montar diagramas de blocos simples no GRC e visualizar os códigos em Python correspondentes.

O segundo cenário consiste no desenvolvimento do “esqueleto” de um sistema complexo, especialmente se este possuir componentes gráficos. Uma vez que o uso do GRC economiza bastante tempo, já que o código Python já é gerado com todas as bibliotecas importadas, os blocos configurados para trabalhar com o tipo correto de sinal, entre outras facilidades.

No entanto, o desenvolvimento em si de um sistema complexo e reconfigurável só pode ser feito de forma eficiente utilizando o código em Python (gerado pelo GRC ou não), uma vez que o GRC não dá suporte a reconfigurações dinâmicas do sistema e limita o alto nível de versatilidade do GNU Radio, que é exatamente o maior atrativo do radio definido por software.

3.3 ECLIPSE E PYDEV

Em tese, programas de computador podem ser escritos em qualquer editor de texto que o programador desejar, inclusive em blocos de nota. Alguns destes, como o bloco de notas da interface gnome do Linux até possuem ferramentas de realce da sintaxe, para facilitar a escrita de programas.

A utilização de programas especializados, os Ambientes Integrados de Desenvolvimento (IDE), é de grande valia durante o desenvolvimento de softwares mais complexos, pois possuem uma série de funções que auxiliam o programador.

A IDE utilizada para o desenvolvimento deste trabalho foi a Eclipse for C++, que é destinada a programação de software em C++. O suporte à linguagem de programação Python foi obtida a partir do plug-in PyDev, que permite que todas as funcionalidades do Eclipse sejam utilizadas também em códigos Python.

Entre as funcionalidades que mais ajudam o desenvolvimento está o "code browsing" que permite que com apenas um clique seja visualizado o código fonte de algum objeto do código. Por exemplo, ao instanciar um modulador PSK, com apenas um clique é possível ir diretamente à definição daquela classe, isto pode ser feito com qualquer entidade do código: classes, funções, estruturas, bibliotecas, etc. Devido a documentação de má qualidade do GNU Radio essa funcionalidade é bastante útil, pois permite uma rápida consulta ao código de cada bloco.

Outra funcionalidade bastante útil é a de autocompletar, que informa ao programador nome e lista de argumentos das funções ou classes. Desta forma, não é necessário memorizar nome de blocos ou consultar uma lista sempre que for necessário instanciar um novo bloco.

Além dessas duas grandes funcionalidades, que podem reduzir o tempo de programação e melhorar a qualidade do código desenvolvido, o Eclipse também possui uma série de ferramentas úteis como indentação automática e um ótimo depurador.

Tanto o Eclipse quanto o PyDev são softwares gratuitos e podem ser encontrados de forma gratuita [12-13]. Além de suportar C++ e Python, o Eclipse também suporta muitas outras linguagens, como Ada, C, COBOL, Java, Jython, Perl, PHP, Ruby, Scala, Clojure, e Scheme.

3.4 GITHUB

O GitHub é um serviço online de hospedagem e controle de versão para projetos de desenvolvimento de software. O GitHub utiliza o sistema git, desenvolvido por Linus Torvalds, sendo compatível com muitos outros programas de controle de versão.

Além de hospedagem do código, o GitHub fornece alguns serviços como acompanhamento do código, ferramentas para desenvolvimento cooperativo e uma forma bastante prática de visualizar o código em qualquer computador.

A versão completa e mais recente do software desenvolvida neste estágio pode ser encontrada no URL ‘github.com/dinart/psk_simu’, independente da data que este texto estiver sendo lido. Portanto, caso o leitor possua acesso a um computador, é recomendável que o código seja acessado a partir deste endereço para leitura, uma vez que os códigos em anexo podem estar desatualizados.

Além de fornecer um ótimo controle de versões (e backup), a utilização do GitHub é sugerida pelos seguintes motivos: facilitar o compartilhamento do código desenvolvido com colegas de laboratório, facilitar a cooperação entre várias pessoas desenvolvendo um mesmo sistema e o mais importante: evitar que o projeto caia no esquecimento e seja perdido.

O software descrito neste trabalho é de código aberto e continuará a receber novas funcionalidades, até que ele esteja pronto para ser submetido como exemplo para o projeto do GNU Radio. Vale salientar que a versão mais recente do código sempre estará disponível em ‘github.com/dinart/psk_simu’.

4 IMPLEMENTAÇÃO

4.1 ASPECTOS GERAIS

O software desenvolvido consiste em um analisador gráfico em tempo real de desempenho de um sistema de comunicação que utiliza moduladores e demoduladores do GNU Radio. Tendo em mente a utilização do programa em testes de sistemas definidos por software sobre diferentes condições e principalmente o uso do software em ambientes educacionais, sem a necessidade de hardware específico (USRP), foi desenvolvido um modelo de canal que pode ser ajustado para possuir diferentes tipos de degradação, com diferentes intensidades.

Os parâmetros que podem ser acompanhados em tempo real a partir da interface gráfica são: diagrama de constelação, resposta em frequência e taxa de erro de bit. As degradações ajustáveis do canal são a adição de ruído gaussiano branco, a limitação de largura de banda e a presença de desvanecimento plano.

Os moduladores e demoduladores suportados pelo programa são o DBPSK, DQPSK e D8PSK, devido ao fato de serem os únicos pares modulador-demodulador digitais disponíveis no GNU Radio atualmente. No entanto, o software é adaptável a outros tipos de modulação, como QAM, de tal sorte que assim que os demoduladores QAM forem adicionados ao GNU Radio, o software poderá ser alterado para suportá-los.

Devido ao fato de ser um programa de código aberto e intenção de futuramente submeter o software para inclusão no ‘trunk’ do GNU Radio, todo o projeto foi feito na língua inglesa, tanto o código como a interface gráfica. A tradução da interface gráfica para português pode ser feita, caso seja solicitada.

Um diagrama de blocos simplificado do programa é mostrado na Figura 13. A maioria dos blocos mostrados no diagrama é composta por outros blocos, como é o caso do modelo do canal desenvolvido.

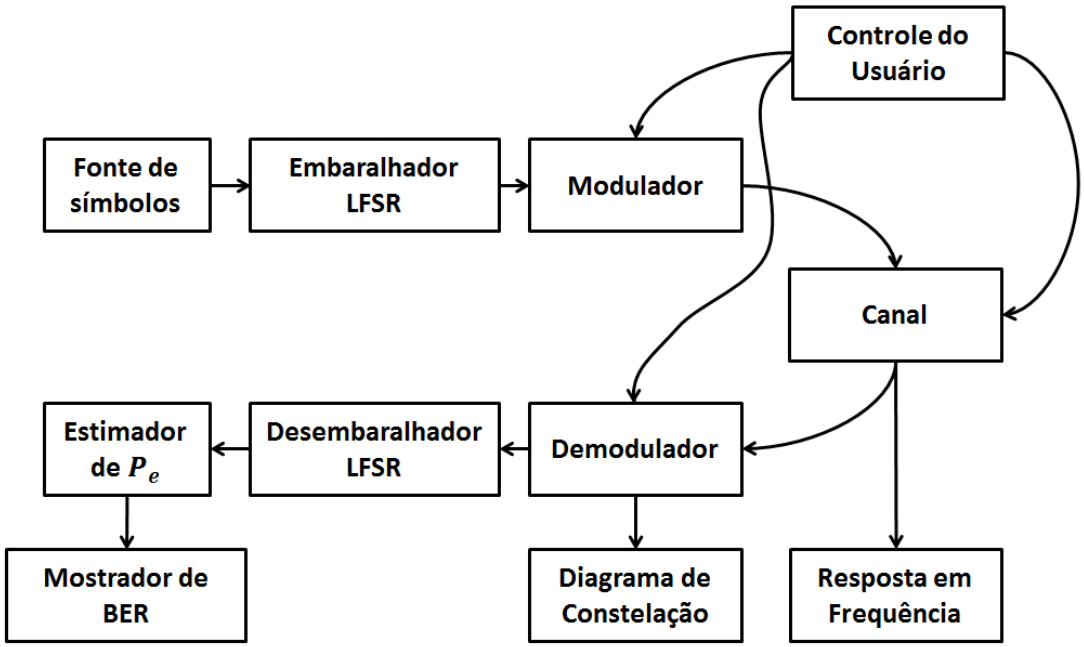

Figura 13 – Diagrama de blocos simplicado do programa desenvolvido.

O controle exercido pelo usuário consiste na escolha de esquema de modulação a ser utilizada no enlace, a relação sinal ruído do canal (sem levar em conta o desvanecimento), a largura de banda disponível e a velocidade do desvanecimento. Também é possível escolher se a medição dos parâmetros é feita na entrada do canal (transmissor) ou na saída (receptor), quando a opção 'transmissor' é escolhida, todas as degradações do canal são removidas. A barra de controles do programa é mostrada na Figura 14.

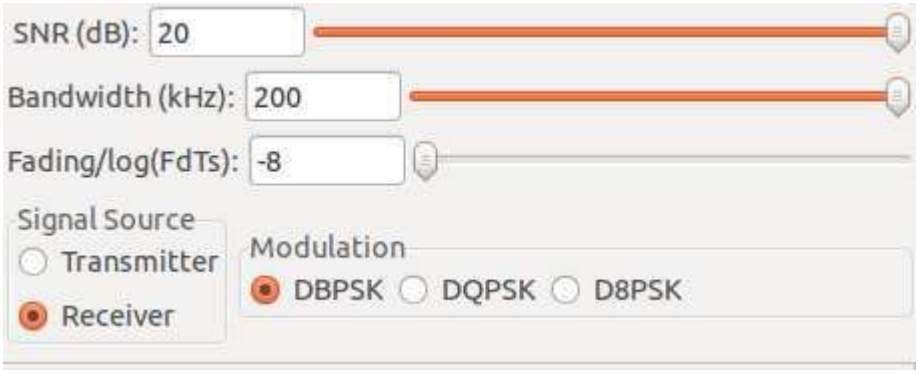

Figura 14 – Barra de controle do programa.

O software desenvolvido possui como dependências o GNU Radio e as bibliotecas padrões de Python, é composto por cinco arquivos escritos em Python. Dois

arquivos correspondem ao programa desenvolvido nesse estágio e os outros quatro são versões modificadas dos visualizadores utilizados.

O arquivo principal é o *psk_simu.py*, no qual estão implementados os componentes mais importantes do sistema, como a interligação dos blocos mostrados na Figura 13 e os algoritmos de controle do programa.

O arquivo *utils.py* contém o modelo do canal, funções utilizadas para mudar os parâmetros do canal e duas implementações distintas do estimador de BER, embora apenas uma seja utilizada.

Os arquivos *constsink.py, fftsink.py e berskin.py* são versões modificadas especialmente para este programa dos visualizadores de constelação, de resposta em frequência e de magnitude de um número real, respectivamente.

A modificação consistiu em remover os muitos botões de configuração que deixavam a interface desnecessariamente poluída e substituição da configuração manual por um controle automático das configurações, que é realizado em *psk_simu.py* a partir de funções call-back, que são evocadas sempre que ocorre uma interação do usuário com um elemento da interface gráfica, cada elemento gráfico possui sua função.

Com a utilização dessas versões modificadas dos visualizadores, a interface do programa ficou mais limpa e intuitiva, até para os menos versados em comunicações digitais, uma foto da interface completa do programa é mostrada na Figura 15.

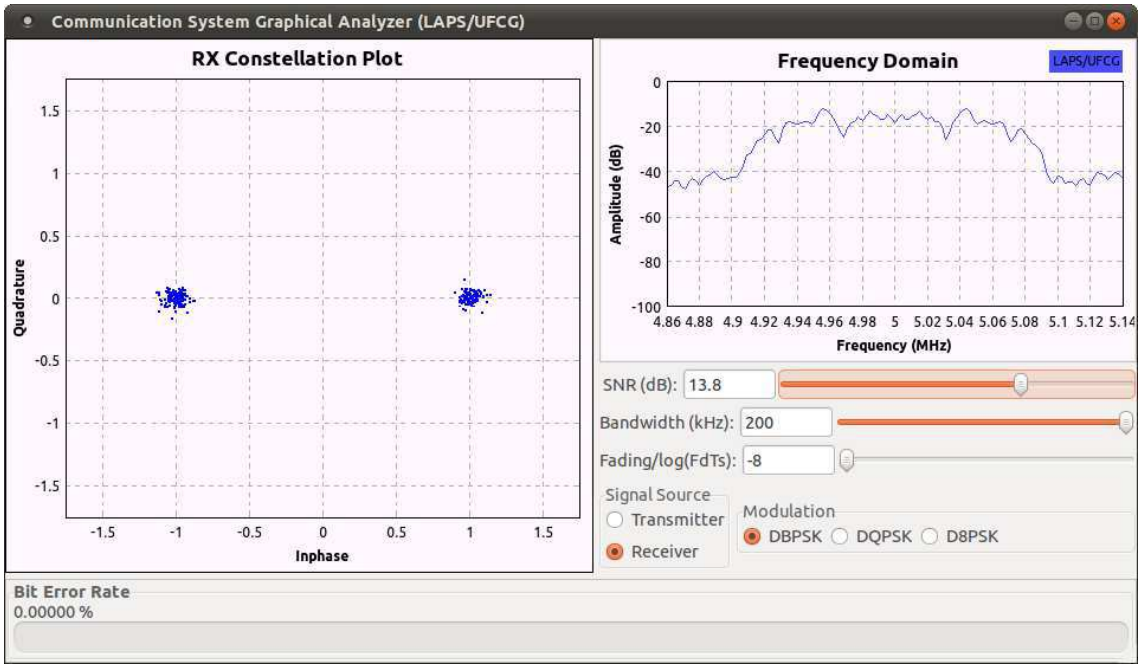

Figura 15 - Interface Gráfica do Usuário (GUI) do programa.

O programa considera uma frequência da portadora de 5Mhz, valor relativamente baixo mas necessário para evitar confusão na numeração do eixo de frequências do visualizador de espectro. A largura do espectro de frequência não se altera conforme troca-se a modulação devido ao fato do programa manter a taxa de símbolos constante, mais precisamente em 140kbauds.

A seguir é feita uma descrição mais detalhada de dois subsistemas do programa, o estimador de probabilidade de erro e o modelo do canal.

4.2 ESTIMADOR DE PROBABILIDADE DE ERRO

Para medição da taxa de erro de bits, utilizou-se o segundo método descrito na seção 2.5, que consiste na aplicação de um algoritmo de embaralhamento em uma sequência de 1s utilizando-se um LFSR e envio da sequência resultante pelo canal.

No receptor, a sequência é desembaralhada e convertida em valores de ponto flutuante, isto é: o bit 1 é transformado no ponto flutuante 1,0 e o bit 0 é transformado em um ponto flutuante 0,0. O sinal resultante passa por um filtro média móvel de 524288 pontos, suficiente para estimação confiável de probabilidades de erro maiores que 2.10^{-5}.

Devido ao fato do LFSR utilizado necessitar 3 valores passados para determinação de cada bit, um erro no canal gera 3 erros de bit na sequência desembaralhada, o que leva a implementação do primeiro estimador de BER, que implementa a seguinte função:

$$BER = \frac{|D_1 - 1|}{3} \tag{23}$$

em que D_1 é a saída do filtro média móvel. Note que D_0 representa a densidade de 1s na sequência, enquanto $|D_1 - 1|$ representa a densidade de 0s (erros).

Essa aproximação é boa para probabilidades de erro inferiores a 10%, mas é ruim em outras situações. Por exemplo, para uma SNR extremamente baixa como $-100dB$, o uso da Equação (23) resulta em uma probabilidade de erro estimada de 16,66%, bem inferior ao valor real, de aproximadamente 50%. O valor 16,66% corresponde a 50% dividido por 3, isto ocorre devido ao fato para uma SNR muito baixa a sequência que chega ao desembaralhador é completamente aleatória, gerando

outra sequência aleatória na entrada do filtro média móvel, que consequentemente terá densidade de 0s igual a 50%.

Para tentar contornar esse problema, foram realizados ensaios do modulador/demodulador PSK do GNU Radio mediante um canal AWGN através do primeiro método descrito na seção 2.5, comparação de sequências aleatórias pré-determinadas. A complexidade inerente ao processo de determinação da posição relativa das sequências enviadas, especialmente devido à criação de bits adicionais no meio da sequência para baixos valores de SNR, fez com que esse método não pudesse ser executado em tempo real. No entanto obteve-se uma curva de BER x SNR mais confiável que a obtida a partir da Equação (23), especialmente para baixos valores de SNR.

Com a curva $BER_1 \, x \, SNR$ obtida a partir do método de comparação de sequências e a curva $BER_2 \, x \, SNR$ obtida a partir do método de embaralhamento, plotou-se a curva paramétrica $BER_1 \, x \, BER_2$ e fez-se o ajuste a partir de uma equação de segundo grau. A partir desta curva, obteve-se um novo estimador de BER, mais confiável em baixas SNR. Este estimador é dado por

$$BER = 1{,}5228658|D_1 - 1|^2 + 0{,}20473967|D_1 - 1| \qquad (24)$$

As duas curvas são comparadas na Figura 16, note que as curvas apresentam um alto grau de concordância para D_0 entre 1 e 0,9, de forma que o segundo estimador preserva a característica de boa resposta para elevados valores de SNR. Além disso, ele mapeia $D_1 = 50\%$ em uma BER de aproximadamente 50%, como esperado. No entanto, se $D_1 > 50\%$, o estimador pode apontar probabilidades de erro de bits maiores que 100%.

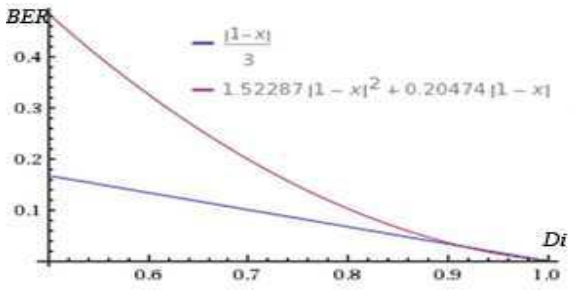

Figura 16 - Comparação entre os dois estimadores.

4.3 MODELO DO CANAL

Os modelos utilizados para determinação das degradações do canal foram aqueles apresentados na seção 2.4. Em seguida são apresentados alguns detalhes de implementação para geração de cada tipo de degradação.

4.3.1 RUÍDO AWGN

O ruído gaussiano branco pode ser gerado diretamente pelo gerador de ruídos do GNU Radio, e portanto, não houve grande complexidade para implementação desta degradação.

No entanto, é importante salientar que o ruído utilizado deve ser um ruído complexo com partes real e imaginária dadas por ruídos gaussianos brancos e independentes.

Como o GNU Radio gera ruídos com distribuição $\mathcal{N}(0,1)$ é necessário multiplicar o ruído pelo desvio padrão desejado, que é determinado pelo valor de SNR estabelecido pelo usuário e ajustado automaticamente pela função call-back correspondente aos elementos que determinam a SNR.

4.3.2 LARGURA DE BANDA

A limitação de largura de faixa do canal é conseguida a partir da utilização de um filtro passa baixa, uma vez que o sinal está representado por seu equivalente passa baixa.

O filtro passa baixa utilizado foi um filtro FIR de Kaiser que possui ganho de 0dB na faixa de passagem, atenuação de 60dB na faixa de rejeição, faixa de transição de 5kHz e frequência de corte variável, definida pelo usuário e ajustada automaticamente a partir da função call-back sempre que houver modificação do valor.

4.3.3 DESVANECIMENTO

O desvanecimento é gerado a partir do método de Jakes descrito na seção 2.4.3. A implementação considera um número arbitrário de senoides, mas atualmente o

programa utiliza 7. O parâmetro f_d não é entrado diretamente pelo usuário, uma vez que esse parâmetro isolado não diz muito sobre a rapidez do desvanecimento. O usuário entra com o parâmetro $\log(f_d T_s)$, onde T_s é o tempo de símbolo.

A escala logarítmica foi utilizada, pois permite a reserva de uma faixa de comprimentos parecidos na barra de rolagem para desvanecimentos lentos e desvanecimentos rápidos. Quando $f_d T_s$ é ajustado para 10^{-8}, valor mínimo, o programa retira automaticamente o bloco de desvanecimento do canal.

Esse parâmetro é ajustado pelo usuário através da interface gráfica e é ajustado no programa por uma função call-back de forma similar ao ajuste dos parâmetros do ruído e do desvanecimento.

5 RESULTADOS

Nas seções posteriores são mostradas capturas de tela do programa sob diferentes condições de degradação. Essa abordagem passa apenas uma ideia do comportamento deste software, é recomendado aos interessados que o instalem e experimentem o programa em tempo real. O procedimento de instalação é apresentado na seção 0.

Comentários sobre o comportamento do sistema sob diferentes condições são tecidos no final desta seção, assim como os problemas encontrados e soluções sugeridas.

5.1 INFLUÊNCIA DO RUÍDO

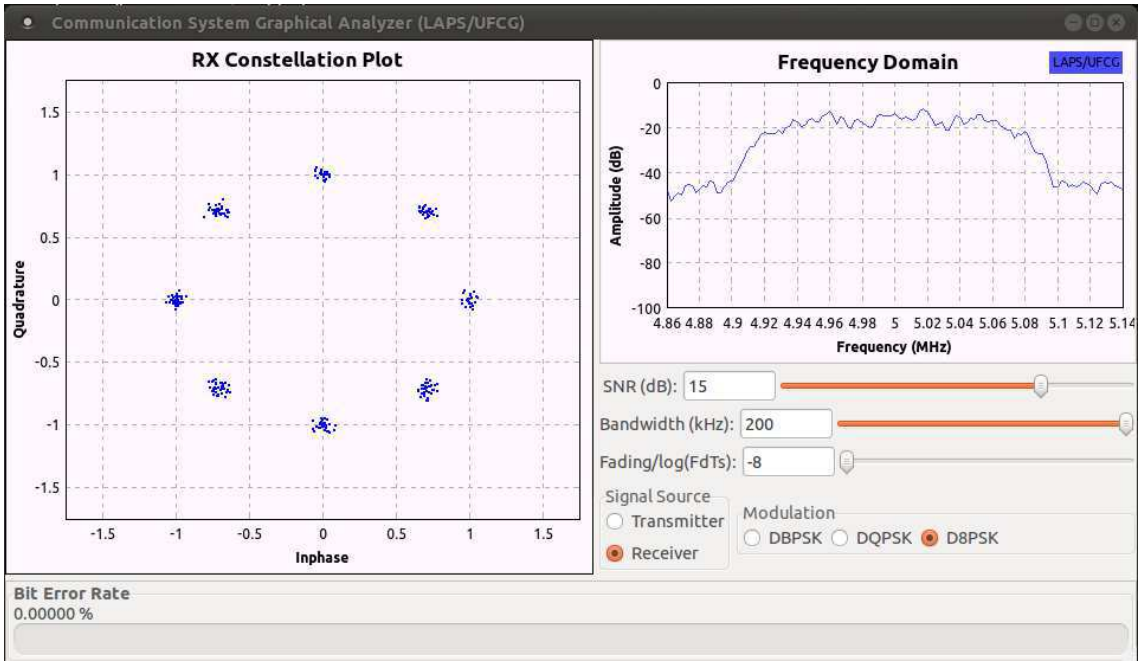

Figura 17 - Captura da tela para SNR de 15dB, banda de 200kHz, ausência de desvanecimento e modulação D8PSK..

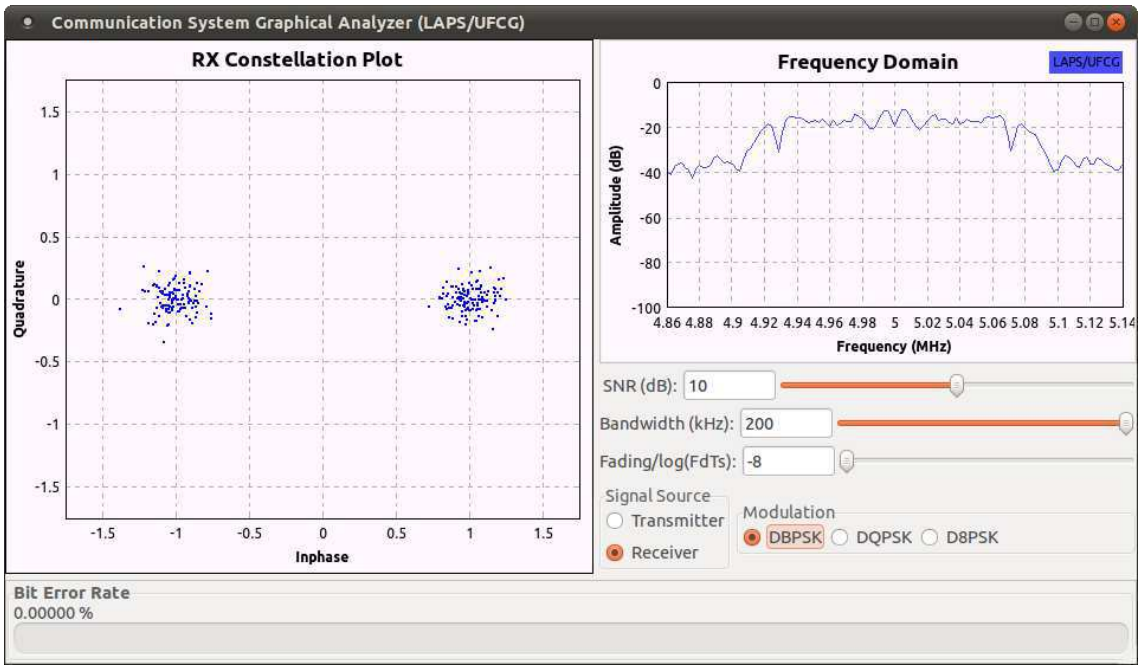

Figura 18 - Captura da tela para SNR de 10dB, banda de 200kHz, ausência de desvanecimento e modulação DBPSK.

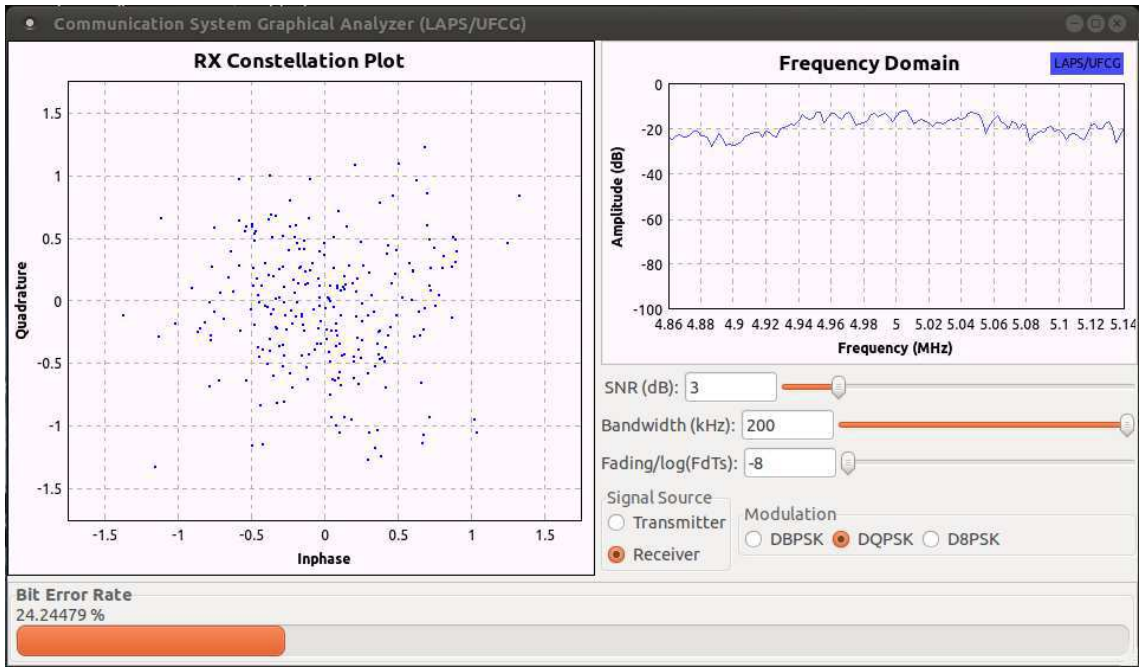

Figura 19 - Captura da tela para SNR de 3dB, banda de 200kHz, ausência de desvanecimento e modulação DQPSK.

5.2 INFLUÊNCIA DA LIMITAÇÃO DE BANDA

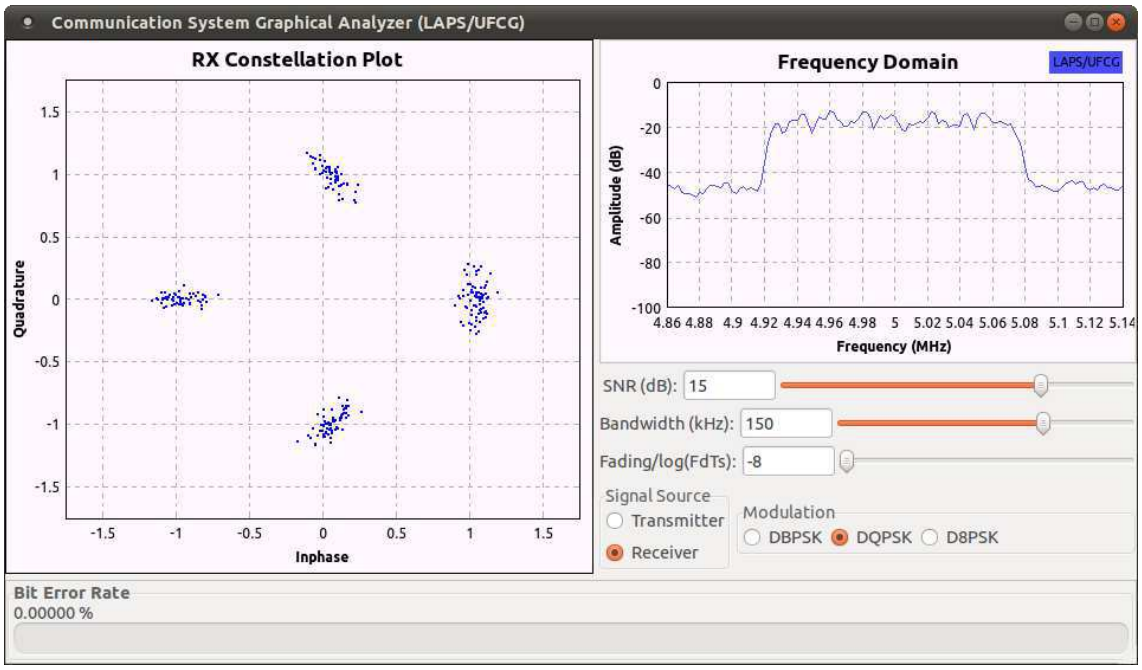

Figura 20 - Captura da tela para SNR de 15dB, banda de 150kHz, ausência de desvanecimento e modulação DQPSK.

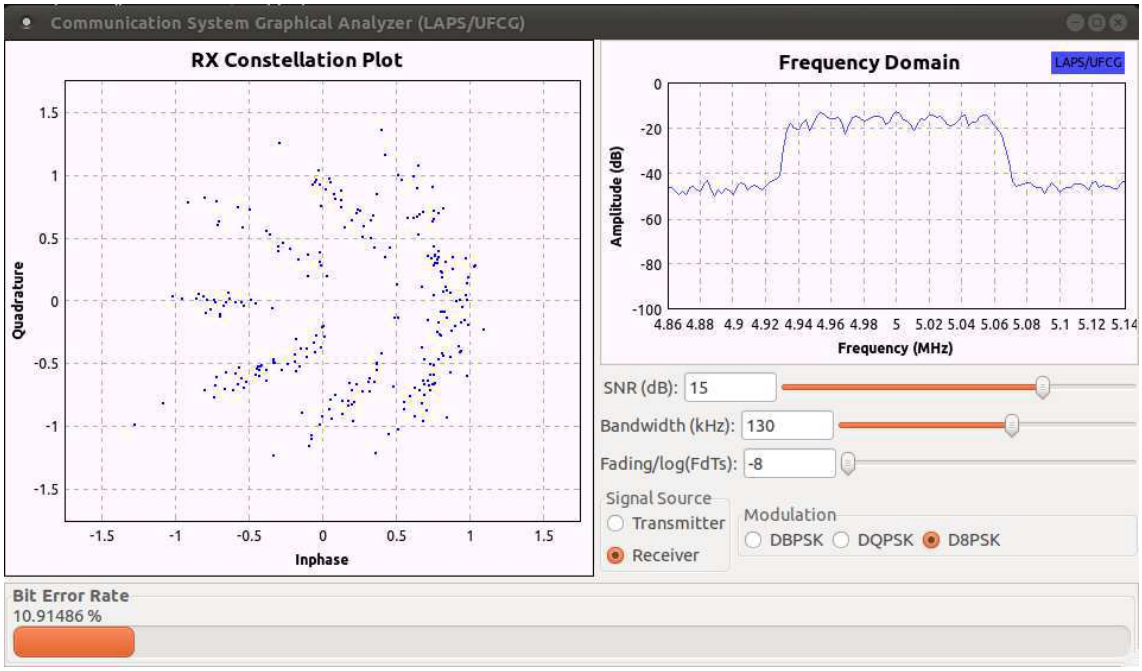

Figura 21 - Captura da tela para SNR de 15dB, banda de130kHz, ausência de desvanecimento e modulação D8PSK.

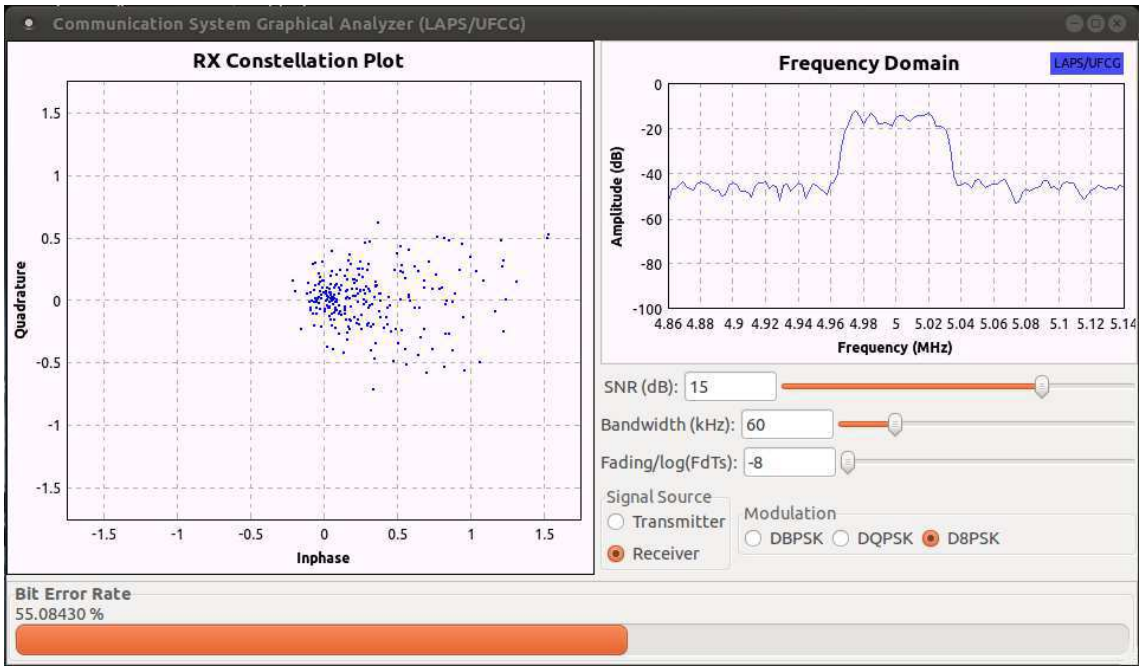

Figura 22 - Captura da tela para SNR de 15dB, banda de 60kHz, ausência de desvanecimento e modulação D8PSK..

5.3 INFLUÊNCIA DO DESVANECIMENTO

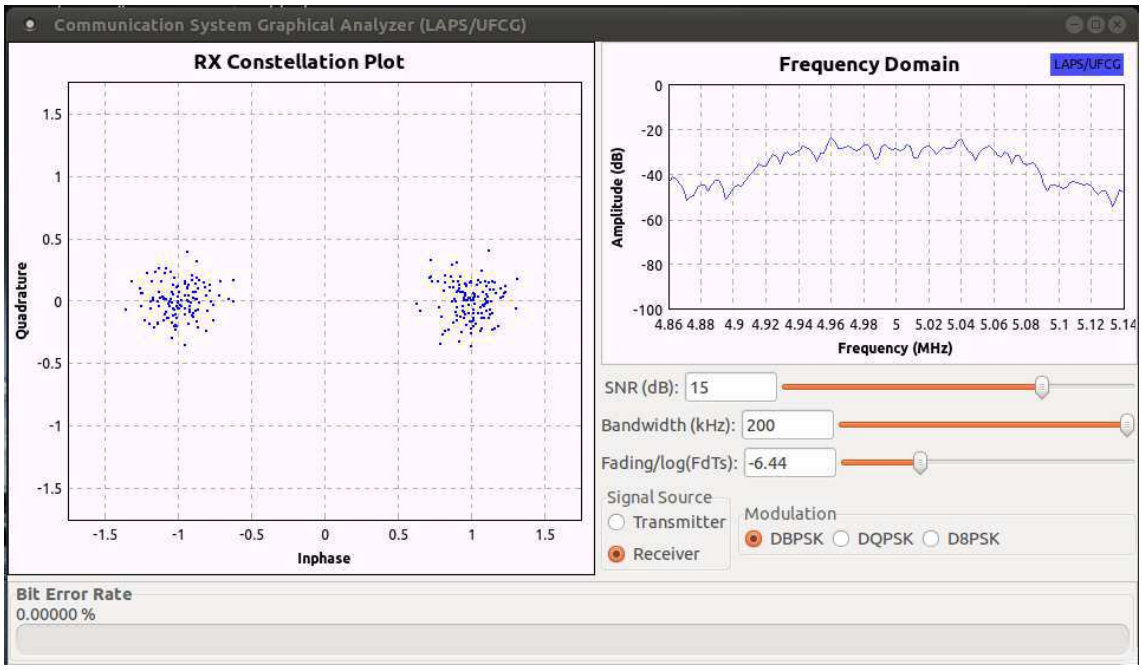

Figura 23 - Captura da tela para SNR de 15dB, banda de 200 kHz, desvanecimento com $f_d T_s = 10^{-6,44}$ e modulação DBPSK..

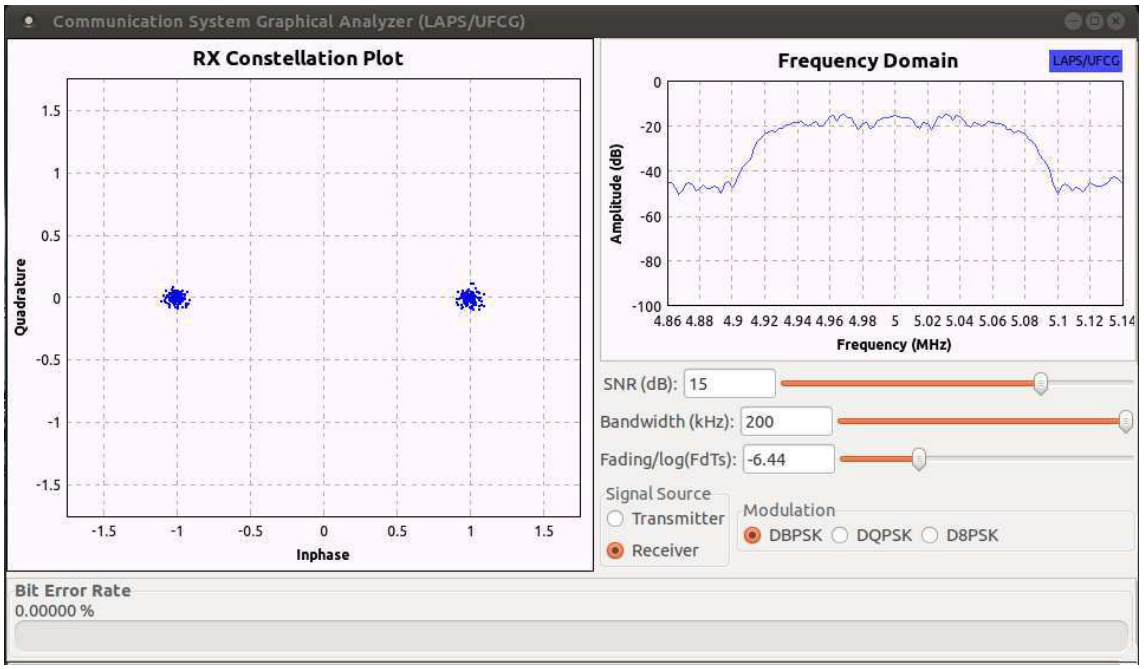

Figura 24 - Captura da tela para SNR de 15dB, banda de 200 kHz, desvanecimento com $f_d T_s = 10^{-6,44}$ e modulação DBPSK. (Mesma condição anterior)

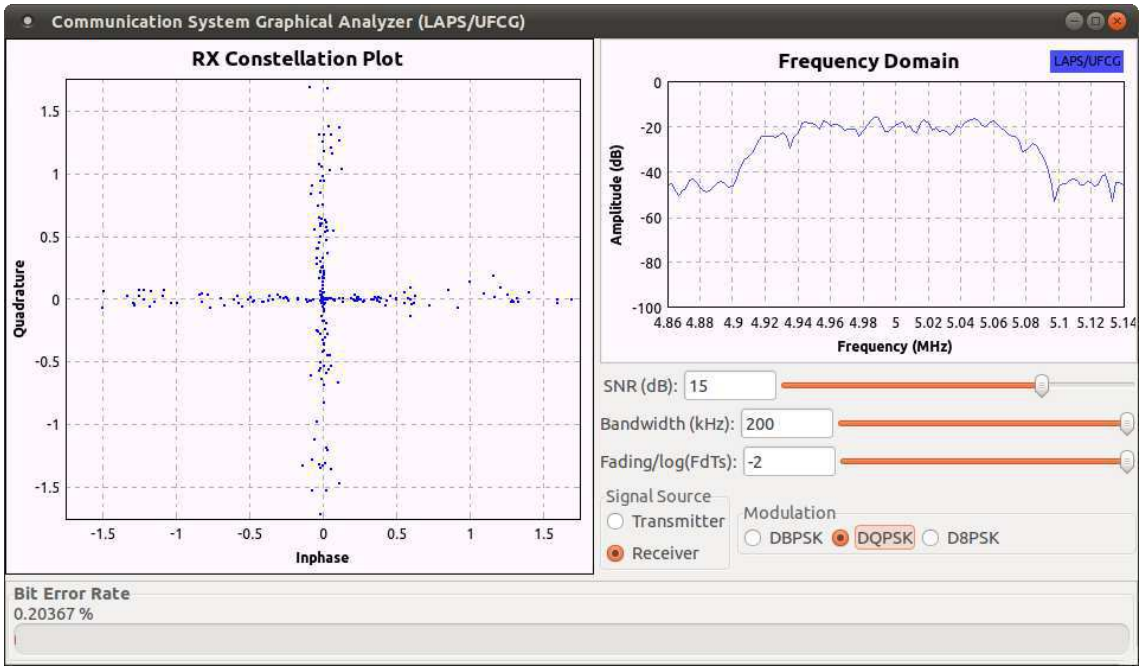

Figura 25 - Captura da tela para SNR de 15dB, banda de 200 kHz, desvanecimento com $f_d T_s = 10^{-2}$ e modulação DQPSK.

5.4 VISÃO DO SINAL NO TRANSMISSOR

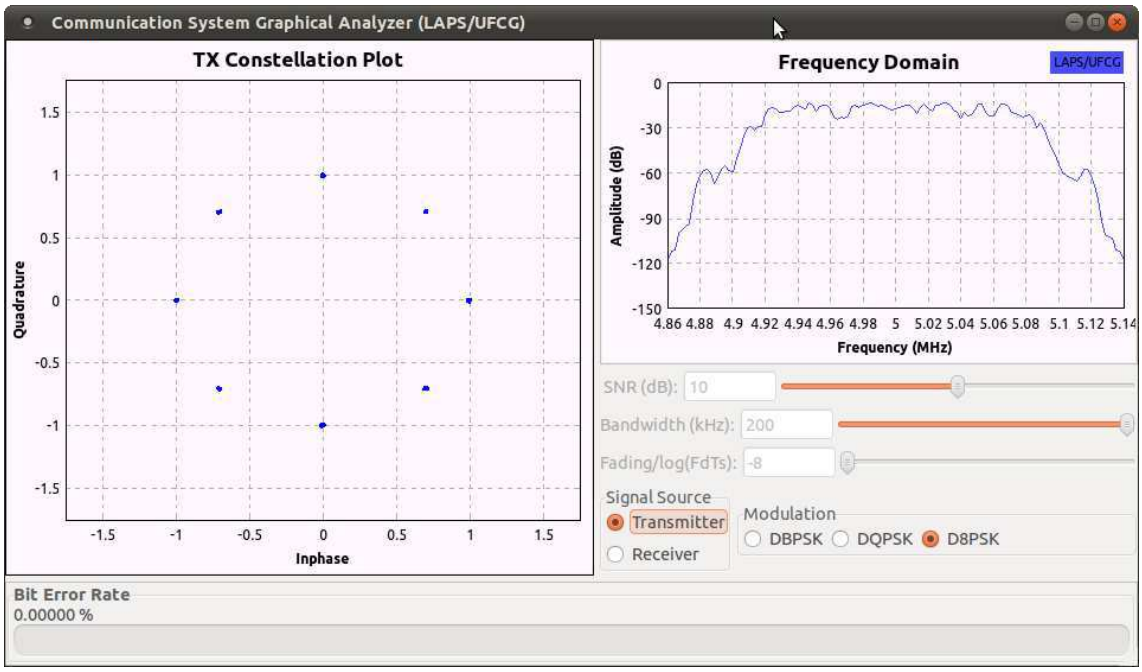

Figura 26 – Visão do sinal na saída do transmissor.

5.5 COMENTÁRIOS SOBRE OS RESULTADOS

Os resultados obtidos a partir da variação de SNR tem comportamento exatamente como os previstos em teoria e comumente mostrados em livros texto básicos, tanto com relação à constelação quanto ao comportamento no domínio da frequência.

A limitação de largura de banda tem o efeito esperado no domínio da frequência e tem um efeito bastante peculiar no diagrama de constelação, criando constelações interessantes no receptor.

O desvanecimento, assim como o ruído, apresentou o comportamento esperado, fazendo o raio das nuvens da constelação do receptor variar de tamanho com o tempo. Além disso, o comportamento na frequência também foi como esperado.

A visualização do sinal no transmissor é útil para uma rápida comparação entre como o sinal deveria ser recebido e como ele está sendo recebido para as condições estabelecidas pelo usuário.

Quanto ao desempenho dos moduladores e demoduladores, podem ser feitas duas ressalvas. A primeira diz respeito à sensibilidade do demodulador QPSK à

variação da SNR do sinal, sincronizando em ângulo deslocado para baixas SNR e permanecendo neste estado mesmo com aumento da SNR. Uma sugestão para melhora é a troca do QPSK por $\frac{\pi}{4}$QPSK, que possui maior facilidade de sincronização.

A outra ressalva diz respeito à instanciação dinâmica de demoduladores, que, ocasionalmente, apresentam problemas relacionados a um ‘assert’ de um bloco C++ que evita acesso indevido a memória. A sugestão de melhora é a troca do ‘assert’, que causa o fechamento do programa, por uma rotina de tratamento de exceção, que pode, por exemplo, reiniciar os parâmetros do demodulador.

6 INSTALAÇÃO

A instalação do software pode ser feita seguindo os seguintes passos:

- Acesse python.org/download/ e instale a versão 2.7.2 ou superior do Python para seu sistema operacional;
- Acesse GNU Radio.org/redmine/projects/GNU Radio/wiki/GettingStarted e siga o procedimento de instalação do GNU Radio versão 3.2 ou superior no seu sistema operacional;
- Acesse github.com/dinart/psk_emu e clique em "Downloads" para descarregar um arquivo .zip ou .tar com todos os arquivos necessários para rodar os programas;
- Descompacte o conteúdo em alguma pasta de seu interesse;
- Execute o arquivo *psk_simu.py*;
- Pronto, o programa está pronto para ser utilizado.

7 CONCLUSÕES

O programa desenvolvido é capaz de analisar parâmetros de desempenho de sistemas de comunicação em tempo real, como desejado inicialmente, podendo ser utilizado em conjunto com hardware específico para análise de canais de comunicação reais. Para tanto, basta substituir o bloco simulador de canal por um bloco que envia/recebe dados para/de a USRP, o hardware utilizado em conjunto com o GNU Radio.

O modelo de canal desenvolvido mostra-se bastante útil, pois pode ser utilizado em outros sistemas de rádio definido por software desenvolvidos com GNU Radio, para determinação do comportamento do sistema sob as mais diversas degradações, em tempo real.

Outra aplicação do programa é para utilização em aulas de comunicações digitais, uma vez que o aspecto visual dele o torna bastante lúdico e é capaz de proporcionar uma boa visão geral do efeito dos vários tipos de degradação em um sistema de comunicação, em particular um sistema que utiliza modulações PSK.

O desenvolvimento do programa continuará mesmo após a conclusão do estágio para que este seja submetido ao GNU Radio e acompanhe o programa como exemplo. Versões mais novas sempre estarão disponíveis online em ‘github.com/dinart/psk_simu’.

Por ser software livre e estar disponível online, qualquer pessoa é bem vinda a contribuir com alguma outra funcionalidade que ache interessante que este possua, por exemplo: adição de outros tipos de perturbação, como interferência.

Por fim, a realização deste trabalho foi útil para consolidação dos conhecimentos obtidos no curso de graduação, assim como para assimilação de muitos outros conhecimentos necessários ao desenvolvimento desta atividade de estágio.

BIBLIOGRAFIA

[1] A.V.G Menezes, D.D. Braga, E. C. Gurjão. “Automatic Modulation Recognition Using Software Defined Radio”. International Information Technology and Telecommunication Symposium. Florianópolis – Brasil. 2009

[2] J. Mitola, "The Software Radio," IEEE National Telesystems Conference, 1992

[3] J. M. III, “Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering, 3rd ed”. New York, USA: John Wiley and Sons, 1996.

[4] J. G. Proakis, M. Selehi. “Digital Communications 5th Edition”. New York, USA: McGraw-Hill Higher Education, 2008.

[5] L. M. Tavares *"Performance of Asynchronous Band-Limited DS/SSMA Systems"* . IEICE Transactions. Communications., Vol. E81-B, No. 9. 1998

[6] D. Taylor. "Introduction to `Synchronous Communications', A Classic Paper by John P. Costas". Proceedings of the IEEE 90 (8): pp. 1459–1460. 2002

[7] C.E. Shannon. “A Mathematical Theory of Communication”, Bell Systems Technology Journal. vol 27. 1948

[8] D. D. Braga. “Determinação Computacional das Curvas de Probabilidades de Erro de Símbolo das Modulações PSK E QAM em Canais de Comunicação AWGN”. Trabalho de Conclusão de Curso. Departamento de Engenharia Elétrica, UFCG. 2011

[9] Sklar, B., “Rayleigh Fading Channels in Mobile Digital Communication Systems", IEEE Communications Magazine, Vol. 35, Jul., 1997.

[10] GNU Radio website (Online). Disponível em: GNU Radio.org

[11] V. Pellegrini, et al. “Soft-DVB : A Fully-Software GNU Radio-based ETSI DVB-T Modulator”. Proc. WSR'08, Karlsruhe, Germany. 2008.

[12] Eclipse Website (Online). Disponível em: eclipse.org

[13] PyDev Website (Online). Disponível em: pydev.org

APÊNDICE A – CÓDIGO FONTE PRINCIPAL

```
#!/usr/bin/env python
##################################################
# Communication System Graphical Analyzer
# Author: Dinart Duarte Braga
# Guiding: Edmar Candeia Gurjao
# Description: PSK Channel GUI Analyzer with channel models that includes
noise, limited bandwidth and fading
##################################################


##################################################
# Imports
##################################################


import utils, constsink, bersink
from gnuradio import gr
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
import wx
#from numpy import random
import fftsink
import time

class psk_simu(grc_wxgui.top_block_gui):

    def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Communication System
Graphical Analyzer (LAPS/UFCG)")

        ##################################################
        # Default Variables
        ##################################################
        self.sps = 2
        self.snr = 20
        self.symbol_rate = 140000
        self.mod_type = "DBPSK"
        self.view = 1
        self.band= 200
        self.excess_bw=0.35
        self.fading_flag = False
        self.fdts = -8
        self.fading_state_rx = False

        ##################################################
        # Blocks Definition
        ##################################################

        #A bit stream of 1's is generated at the source, scrambled,
        #modulated and sent to the input of an AWGN channel.
        #random.seed(42)
        #self.source = gr.vector_source_b([random.randint(0, 2) for i in
range(0,10^8)], True)
        self.source = gr.vector_source_b((1,), True, 1)
        self.thottle = gr.throttle(gr.sizeof_char,10e5)
        self.scrambler = gr.scrambler_bb(0x40801, 0x92F72, 20) #Taxa de
simbolos constante
        self.pack = gr.unpacked_to_packed_bb(1, gr.GR_MSB_FIRST)
        self.modulator =
utils.mods[self.mod_type](self.sps,excess_bw=self.excess_bw)
```

```
        self.channel =
utils.channel(1/10.0**(self.snr/10.0),self.band,self.symbol_rate,self.sps)

        #The noisy signal is demodulated, descrambled and the BER
        #is estimated by the ber_estim block using the receiver
        #density of 0 bits.

        self.demodulator =
utils.demods[self.mod_type](self.sps,excess_bw=self.excess_bw)
        self.descrambler = gr.descrambler_bb(0x40801, 0x92F72, 20)
        self.char2float = gr.char_to_float()
        self.mov_average = gr.moving_average_ff(524288, 1/524288., 10000)
        self.ber = utils.ber_estim()
        #self.ber = utils.ber_estim_simple(3)


        ##################################################
        # GUI Elements Definition
        ##################################################

        #Defines an adds FFT Window to GUI
        self.fft = fftsink.fft_sink_c(self.GetWin(),
sample_rate=self.symbol_rate*self.sps, baseband_freq=5e6)
        self.GridAdd(self.fft.win, 0,3,4,3)
        self.ctr= gr.complex_to_real(1)

        #Defines and adds SNR slider to GUI
        _snr_sizer = wx.BoxSizer(wx.HORIZONTAL)
        self._snr_text_box = forms.text_box(parent=self.GetWin(),
            sizer=_snr_sizer, value=self.snr, callback=self.callback_snr,
            label=" SNR (dB)", converter=forms.float_converter(),
            proportion=0)
        self._snr_slider = forms.slider(parent=self.GetWin(),
            sizer=_snr_sizer, value=self.snr, callback=self.callback_snr,
            minimum=0, maximum=20, style=wx.RA_HORIZONTAL, proportion=1)
        self.GridAdd(_snr_sizer, 4, 3, 1, 3)

        #Defines and adds bandwidth slider to GUI
        band_sizer = wx.BoxSizer(wx.HORIZONTAL)
        self.band_text_box = forms.text_box(parent=self.GetWin(),
            sizer=band_sizer, value=self.band, callback=self.callback_band,
            label="Bandwidth (kHz)", converter=forms.float_converter(),
            proportion=0)
        self.band_slider = forms.slider(parent=self.GetWin(),
            sizer=band_sizer, value=self.band, callback=self.callback_band,
            minimum=30, maximum=200, style=wx.RA_HORIZONTAL, proportion=1)
        self.GridAdd(band_sizer, 5, 3, 1, 3)

        #Defines and adds Rayleigh GUI elements

        fading_sizer = wx.BoxSizer(wx.HORIZONTAL)
        self.fading_text_box = forms.text_box(parent=self.GetWin(),
            sizer=fading_sizer, value=self.fdts,
callback=self.callback_fading,
            label='Fading/log(FdTs)', converter=forms.float_converter(),
            proportion=0)
        self.fading_slider = forms.slider(parent=self.GetWin(),
            sizer=fading_sizer, value=self.fdts,
callback=self.callback_fading,
            minimum=-8, maximum=-2, style=wx.RA_HORIZONTAL, proportion=1)
        self.GridAdd(fading_sizer, 6, 3, 1, 3)

        #Defines and adds modulation type chooser to GUI
        self._mod_type_chooser = forms.radio_buttons(parent=self.GetWin(),
            value=self.mod_type, callback=self.set_mod_type,
            label="Modulation", choices=["DBPSK", "DQPSK", "D8PSK"],
            labels=["DBPSK", "DQPSK", "D8PSK"], style=wx.RA_HORIZONTAL)
        self.GridAdd(self._mod_type_chooser, 7, 4, 1, 2)
```

```
        #Defines and adds signal source chooser
        self.sig_src_chooser = forms.radio_buttons(parent=self.GetWin(),
            value=self.view, callback=self.callback_view,
            label="Signal Source", choices=[0,1],
            labels=["Transmitter","Receiver"], style=wx.RA_VERTICAL)
        self.GridAdd(self.sig_src_chooser, 7,3,1,1)


        #Definition of the of constellation window and attachment to the GUI
        self.constel = constsink.const_sink_c(self.GetWin(),
            title="RX Constellation Plot",  sample_rate=self.symbol_rate,
            const_size=256, mod=self.mod_type)
        self.GridAdd(self.constel.win,0,0,8,3)


        #Definition of the constellation sink window and attachment to the GUI
        self.number_sink = bersink.number_sink_f(self.GetWin(),
            sample_rate=self.symbol_rate)
        self.GridAdd(self.number_sink.win,8,0,1,6)

        ##################################################
        # Blocks Connections
        ##################################################

        #The necessary block connections to the system work as described
above.
        self.connect(self.source, self.scrambler , self.thottle, self.pack)
        #self.connect(self.source , self.thottle, self.pack)
        self.connect(self.pack, self.modulator, self.channel,
self.demodulator)
        self.connect(self.channel, self.fft)
        self.connect(self.demodulator.diffdec, self.constel)
        self.connect(self.demodulator, self.descrambler, self.char2float,
self.mov_average)
        self.connect(self.mov_average, self.ber, self.number_sink)

##################################################
# Callback Functions of GUI Elements
##################################################

#Callback function of SNR slider, sets the Signal to Noise ratio
#of the AWGN Channel

    def callback_snr(self,snr):
        #if snr > 30:
        #    self.snr = 30
        #elif snr < -10:
        #    self.snr = -10
        #else:
        self.snr = snr;
        self._snr_slider.set_value(self.snr)
        self._snr_text_box.set_value(self.snr)
        self.channel.set_snr(self.snr,self.view)

#Callback function of bandwidth slider.

    def callback_band(self,band):
        if band > 300:
            self.band = 300
        elif band < 30:
            self.band = 30
        else:
            self.band = band;
        self.band_slider.set_value(self.band)
        self.band_text_box.set_value(self.band)
        self.channel.set_band(self.band)
```

```
#Callback function that changes fading level

    def callback_fading(self, fdts):
        self.fdts=fdts
        self.channel.set_fading(fdts)
        self.fading_slider.set_value(fdts)
        self.fading_text_box.set_value(fdts)

#Callback function of the signal source chooser

    def callback_view(self,view):
        self.view = view;
        self.sig_src_chooser.set_value(self.view)
        self.callback_snr(self.snr)
        if not view:
            self.channel.set_fading(-8)
            self.channel.set_band(240)
            self._snr_slider.Disable(True)
            self._snr_text_box.Disable(True)
            self.band_slider.Disable(True)
            self.band_text_box.Disable(True)
            self.fading_text_box.Disable(True)
            self.fading_slider.Disable(True)
            self.constel.win.plotter.set_title('TX Constellation Plot')
            self.fft.win.change_yperdiv(30)
        if view:
            self.channel.set_fading(self.fdts)
            self.channel.set_band(self.band)
            self._snr_slider.Disable(False)
            self._snr_text_box.Disable(False)
            self.band_slider.Disable(False)
            self.band_text_box.Disable(False)
            self.fading_text_box.Disable(False)
            self.fading_slider.Disable(False)
            self.constel.win.plotter.set_title('RX Constellation Plot')
            self.fft.win.change_yperdiv(20)
        time.sleep(.05)

#Callback function of the modulation type chooser, it blocks the
#flowgraph and changes modulator and demodulator, it also sets
#the M parameter of the Constellation Sink.

    def set_mod_type(self, mod_type):
        self.mod_type = mod_type
        self._mod_type_chooser.set_value(self.mod_type)

        self.lock()

        self.disconnect(self.pack, self.modulator)
        self.disconnect(self.modulator, self.channel)
        self.modulator =
utils.mods[self.mod_type](self.sps,excess_bw=self.excess_bw)
        self.connect(self.pack, self.modulator)
        self.connect(self.modulator, self.channel)

        self.disconnect(self.channel, self.demodulator)
        self.disconnect(self.demodulator.diffdec, self.constel)
        self.disconnect(self.demodulator, self.descrambler)
        self.demodulator =
utils.demods[self.mod_type](self.sps,excess_bw=self.excess_bw)

        self.constel.change_mod(self.mod_type)

        self.connect(self.channel, self.demodulator)
        self.connect(self.demodulator.diffdec, self.constel)
        self.connect(self.demodulator, self.descrambler)
        self.unlock()
```

```
        time.sleep(0.1)

if __name__ == '__main__':
    tb = psk_simu()
    tb.Run()
```

APÊNDICE B – CÓDIGO FONTE DO MODELO DE CANAL E DOS ESTIMADORES DE BER

```
################################################
# Communication System Graphical Analyzer
# Author: Dinart Duarte Braga
# Guiding: Edmar Candeia Gurjao
# Description: This file contains useful classes to be used in the main
program
################################################


################################################
# Imports
################################################
from gnuradio import gr, blks2
import numpy, math


################################################
# Dictionaries used to help control functions
################################################
mods = {'DBPSK':blks2.dbpsk_mod, 'DQPSK':blks2.dqpsk_mod,
'D8PSK':blks2.d8psk_mod }
demods = {'DBPSK':blks2.dbpsk_demod, 'DQPSK':blks2.dqpsk_demod,
'D8PSK':blks2.d8psk_demod }
k = {'DBPSK':1, 'DQPSK':2, 'D8PSK':3}
M = {'DBPSK':2, 'DQPSK':4, 'D8PSK':8}
gain = {'DBPSK':1, 'DQPSK':1, 'D8PSK':270}

#Channel Model (AWGN + Filter)
#Rayleigh scattering is added dynamically.
class channel(gr.hier_block2):
    def __init__(self,ampl_i,band,symbol_rate,sps):
        gr.hier_block2.__init__(self,"Channel",
                                gr.io_signature(1,1,gr.sizeof_gr_complex),
                                gr.io_signature(1,1,gr.sizeof_gr_complex))

        self.symbol_rate = symbol_rate
        self.sample_rate=symbol_rate*sps
        self.fading = False
        self.adder = gr.add_cc()
        self.noise = gr.noise_source_c(gr.GR_GAUSSIAN, 1, -42)
        self.ampl = gr.multiply_const_cc(ampl_i)
        self.taps = gr.firdes.low_pass_2
(1,280,band/2,5,80,gr.firdes.WIN_KAISER)
        self.filter=gr.fir_filter_ccf(1,self.taps)

        #Connects
        self.connect(self,self.filter,(self.adder,0))
        self.connect(self.noise, self.ampl, (self.adder,1))
        self.connect(self.adder, self)


    def toggle_fading(self,flag,fd):
        self.lock()

        if(flag):
            self.disconnect(self,self.filter)
```

```
            self.ray=rayleigh(fd,5,self.sample_rate)
            self.connect(self,self.ray,self.filter)
            self.fading=True

        else:
            if(self.fading):
                self.disconnect(self,self.ray,self.filter)
                del(self.ray)
                self.connect(self,self.filter)
                self.fading=False

        self.unlock()

    def set_fading(self,fdts):
        fd= 10**fdts*self.symbol_rate
        #fd= fdts/100*self.symbol_rate
        if(fd > 10**-8*self.symbol_rate):
            if(not self.fading):
                self.toggle_fading(True,fd)
            else:
                self.ray.set_fd(fd)
        else:
            self.toggle_fading(False,fd)

    def set_snr(self, snr, view):
        self.ampl.set_k(view*(1/10.0**(snr/10.0)))

    def set_band(self,band):
            self.taps = gr.firdes.low_pass_2
(1,280,band/2,5,80,gr.firdes.WIN_KAISER)
            self.filter.set_taps(self.taps)


#Jakes Model for Fading Generation
class rayleigh(gr.hier_block2):
    def __init__(self,fd,M,sample_rate):
        gr.hier_block2.__init__(self,"Rayleigh Channel",
                                gr.io_signature(1,1,gr.sizeof_gr_complex),
                                gr.io_signature(1,1,gr.sizeof_gr_complex))

        self.M = M
        self.sample_rate = sample_rate
        n=range(1,M+1)
        N = 4*M+2

        f_n= [fd*math.cos(2*math.pi*x/N) for x in n]

        beta_n = [math.pi/M*x for x in n]

        a_n = [2*math.cos(x) for x in beta_n]
        a_n.append(math.sqrt(2)*math.cos(math.pi/4))
        a_n = [x*2/math.sqrt(N) for x in a_n]


        b_n= [2*math.sin(x) for x in beta_n]
        b_n.append(math.sqrt(2)*math.sin(math.pi/4))
        b_n = [x*2/math.sqrt(N) for x in b_n]

        f_n.append(fd)

        self.sin_real =
[gr.sig_source_f(self.sample_rate,gr.GR_COS_WAVE,f_n[i],a_n[i]) for i in
range(M+1)]
        self.sin_imag =
[gr.sig_source_f(self.sample_rate,gr.GR_COS_WAVE,f_n[i],b_n[i]) for i in
range(M+1)]

        self.add_real = gr.add_ff(1)
```

```
        self.add_imag = gr.add_ff(1)

        for i in range (M+1):
            self.connect(self.sin_real[i],(self.add_real,i))

        for i in range (M+1):
            self.connect(self.sin_imag[i],(self.add_imag,i))

        self.ftoc = gr.float_to_complex(1)

        self.connect(self.add_real,(self.ftoc,0))
        self.connect(self.add_imag,(self.ftoc,1))
        self.mulc = gr.multiply_const_cc((0.5))

        #self.divide = gr.divide_cc(1)
        #self.connect(self,(self.divide,0))
        #self.connect(self.ftoc,(self.divide,1))
        #self.connect(self.divide, self)
        self.prod = gr.multiply_cc(1)
        self.connect(self,(self.prod,0))
        self.connect(self.ftoc,self.mulc,(self.prod,1))
        self.connect(self.prod, self)

    def set_fd(self,fd):
        M = self.M
        n=range(1,M+1)
        N = 4*M+2

        f_n= [fd*math.cos(2*math.pi*x/N) for x in n]

        beta_n = [math.pi/M*x for x in n]

        a_n = [2*math.cos(x) for x in beta_n]
        a_n.append(math.sqrt(2)*math.cos(math.pi/4))
        a_n = [x*2/math.sqrt(N) for x in a_n]


        b_n= [2*math.sin(x) for x in beta_n]
        b_n.append(math.sqrt(2)*math.sin(math.pi/4))
        b_n = [x*2/math.sqrt(N) for x in b_n]

        f_n.append(fd)

        for i in range(M+1):
            self.sin_real[i].set_amplitude(a_n[i])
            self.sin_imag[i].set_amplitude(b_n[i])
            self.sin_real[i].set_frequency(f_n[i])
            self.sin_imag[i].set_frequency(f_n[i])


#BER estimation using a quadratic polynomial, it results on more accuracy
#approximations for BER > 10%, compared to dividing the descrambled density
#of 0's by 3.
class ber_estim(gr.hier_block2):
    def __init__(self):

        gr.hier_block2.__init__(self,"BER Estimator",
                                gr.io_signature(1,1,gr.sizeof_float),
                                gr.io_signature(1,1,gr.sizeof_float))

        #TODO Implement a polynomial block in C++ and approximate with
polynomials
        #of arbitrary order
        self.add = gr.add_const_vff((-1, ))
        self.square = gr.multiply_ff()
        self.mult_lin = gr.multiply_const_ff(-0.20473967)
        self.mult_sq = gr.multiply_const_ff(1.5228658)
        self.sum = gr.add_ff()
```

```
        self.connect(self,self.add)
        self.connect(self.add,(self.square,0))
        self.connect(self.add,(self.square,1))
        self.connect(self.square,self.mult_sq,(self.sum,0))
        self.connect(self.add,self.mult_lin,(self.sum,1))
        self.connect(self.sum,self)

#Simple BER estimator proposed by GNURadio example. Very bad for BER > 10%
class ber_estim_simple(gr.hier_block2):
    def __init__(self,k):

        gr.hier_block2.__init__(self,"BER Estimator",
                                gr.io_signature(1,1,gr.sizeof_float),
                                gr.io_signature(1,1,gr.sizeof_float))

        self.add = gr.add_const_vff((-1, ))
        self.mult= gr.multiply_const_ff(-1/k)
        self.connect(self,self.add, self.mult,self)
```