

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETROTÉCNICA
CURSO DE ENGENHARIA ELÉTRICA

RENAN ANTONIO CORRÊA MEDEIROS

PROCESSAMENTO DIGITAL DE SINAIS DE ULTRASSOM
EMBARCADO NO RASPBERRY PI

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2019

RENAN ANTONIO CORRÊA MEDEIROS

**PROCESSAMENTO DIGITAL DE SINAIS DE ULTRASSOM
EMBARCADO NO RASPBERRY PI**

Trabalho de Conclusão do Curso de Graduação em Engenharia Elétrica apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR) como requisito para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Amauri Amorin Assef

CURITIBA

2019

Renan Antonio Corrêa Medeiros

**PROCESSAMENTO DIGITAL DE SINAIS DE ULTRASSOM EMBARCADO NO
RASPBERRY PI**

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro Eletricista, do curso de Engenharia Elétrica do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 27 de novembro de 2019.

Prof. Antonio Carlos Pinho, Dr.
Coordenador de Curso de Engenharia Elétrica

Profa. Annemarle Gehrke Castagna, Mestre
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia Elétrica do DAELT

ORIENTAÇÃO

Amauri Amorin Assef, Dr.
Universidade Tecnológica Federal do Paraná
Orientador

BANCA EXAMINADORA

Amauri Amorin Assef, Dr.
Universidade Tecnológica Federal do Paraná

Marcelo de Oliveira Rosa, Dr.
Universidade Tecnológica Federal do Paraná

Fábio Kurt Schneider, Dr.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia Elétrica

*Dedico este trabalho aos meus pais Gilberto e
Neila e ao meu irmão Rogério*

AGRADECIMENTO

Primeiramente agradeço a Deus, pela força em todos os momentos e por ter me abençoado e proporcionado chegar até aqui.

Aos meus pais José Gilberto Sousa Medeiros e Neila Ruth Corrêa Medeiros, que me deram a oportunidade de estudar e sempre me apoiaram, incentivaram e acreditaram em mim.

Ao meu irmão Rogério Corrêa Medeiros, que além de tudo é meu grande amigo e sempre me ajudou e apoiou nessa trajetória.

Ao orientador professor Dr. Amauri Amorin Assef, um grande profissional, que sempre foi atencioso e dedicado durante a orientação e, principalmente, pela oportunidade e confiança em mim depositada durante a realização deste trabalho.

Aos meus avós e demais familiares que estão no Pará, que apesar da distância, sempre estiveram rezando e torcendo por mim.

A todos os meus amigos e colegas que fizeram parte dessa fase da minha vida e, que com certeza, me auxiliaram nesta caminhada.

A todos os professores da UTFPR que contribuíram com a minha formação.

A UTFPR, que durante anos me acolheu.

RESUMO

MEDEIROS, Renan Antonio Corrêa. **Processamento digital de sinais de ultrassom embarcado no Raspberry Pi**. 2019. 76 f. Trabalho de Conclusão de Curso (Graduação – Curso de Engenharia Elétrica). Universidade Tecnológica Federal do Paraná, Curitiba, 2019.

A técnica do ultrassom em Modo B representa uma das principais modalidades de geração de imagem para auxílio ao diagnóstico médico. A grande vantagem de utilizar a ultrassonografia na medicina está na técnica não-invasiva e na radiação não-ionizante, que tornam o método indolor e seguro. Para melhorar a qualidade da imagem gerada, são esperadas novas abordagens e técnicas de processamento digital de sinais baseadas em plataformas de *hardware* e *software*. Neste trabalho são abordados o estudo e a implementação dos algoritmos de processamento de sinais de ultrassom no sistema embarcado Raspberry Pi. As etapas desenvolvidas incluem: filtragem digital, somatório coerente, demodulação com detecção de envoltória e compressão logarítmica, sendo a etapa de conversão de varredura realizada no software Matlab, após os dados serem processados no Raspberry Pi. Para validação do algoritmo implementado foram utilizados dados amostrados com frequência de 40 MHz, obtidos por simulação e através da plataforma de pesquisa ULTRA-ORS. As análises qualitativas e quantitativas utilizando as funções de custo da raiz quadrada do erro quadrático médio normalizado (NRMSE) e da soma residual dos quadrados normalizado (NRSS), demonstram que o algoritmo desenvolvido em linguagem de programação Python, implementado no Raspberry Pi, apresenta resultados compatíveis com o modelo de referência adotado no Matlab e validado em estudos prévios. Todos os resultados do NRMSE foram menores que 10% e do NRSS foram próximos de zero, indicando uma excelente concordância com o modelo do Matlab.

Palavras-chave: Ultrassom. Processamento digital de sinais. Raspberry Pi. Python.

ABSTRACT

MEDEIROS, Renan Antonio Corrêa. **Ultrasound digital signal processing embedded on Raspberry Pi**. 2019. 76 f. Trabalho de Conclusão de Curso (Graduação – Curso de Engenharia Elétrica). Universidade Tecnológica Federal do Paraná, Curitiba, 2019.

The B-mode ultrasound technique represents one of the main modalities of imaging to aid medical diagnosis. The great advantage of using ultrasound in medicine is the non-invasive technique and non-ionizing radiation, which make the method painless and safe. To improve the quality of the generated image, new approaches and techniques for digital signal processing based on hardware and software platforms are expected. This work deals with the study and implementation of ultrasound signal processing algorithms in the Raspberry Pi embedded system. The developed steps include: digital filtering, coherent summation, demodulation and envelope detection, and logarithmic compression, where the scan conversion step was performed in the Matlab software after the data has been processed in Raspberry Pi. To validate the implemented algorithm, we used data sampled with frequency of 40 MHz, obtained by simulation and through the ULTRA-ORS research platform. Qualitative and quantitative analysis using the cost functions of the Normalized Root Mean Squared Error (NRMSE) and the Normalized Residual Sum of Squares (NRSS) show that the Python algorithm implemented in Raspberry Pi presents results compatible with the reference adopted in Matlab and validated in previous studies. All NRMSE results were less than 10% and NRSS results were close to zero, indicating excellent agreement with the Matlab model.

Keywords: Ultrasound. Digital signal processing. Raspberry Pi. Python.

LISTA DE FIGURAS

Figura 1 - Diagrama em blocos das principais funções de processamento digital para geração de imagem no Modo B.	16
Figura 2 - Representação da geração de ondas de US pelo método pulso-eco para determinação e caracterização de diferentes meios.	19
Figura 3 - Exemplificação da sequência de aberturas das linhas de varredura em um transdutor matricial linear de 128 elementos no método Beamforming.....	21
Figura 4 - Representação de um equipamento de US para formação da imagem em Modo B.	21
Figura 5 - Representação da modulação do brilho no Modo B de acordo com a amplitude no Modo A.....	22
Figura 6 - Varredura retangular e amostras do sinal coerente para uma determinada linha da imagem.	23
Figura 7 - Varredura setorial e amostras do sinal coerente para um determinado setor da imagem.	23
Figura 8 - Estrutura convencional generalizada de um filtro FIR.	25
Figura 9 - Exemplo de Resposta ao Impulso de um Filtro FIR com simetria ímpar.....	26
Figura 10 - Estrutura de um filtro FIR com simetria ímpar de coeficientes.....	26
Figura 11 - Exemplo de Resposta ao Impulso de um Filtro FIR com simetria par.	27
Figura 12 - Estrutura de um filtro FIR com simetria par de coeficientes.....	27
Figura 13 - Exemplo de Correção de atraso e Somatório Coerente.	28
Figura 14 - Diagrama de blocos para detecção de envoltória.....	30
Figura 15 - Estrutura da Transformada de Hilbert baseada em filtro FIR para obtenção das componentes em quadratura e fase.	30
Figura 16 - Exemplo de Resposta ao Impulso de um Filtro FIR Hilbert <i>Transformer</i>	31
Figura 17 - Exemplo de detecção da envoltória de um sinal de US a partir da Transformada de Hilbert.....	31
Figura 18 - Exemplo de compressão logarítmica com adequação da faixa dinâmica para visualização.	32
Figura 19 - Exemplo de conversão de varredura do sistema de coordenadas polares para o sistema de coordenadas cartesianas.	33
Figura 20 - Módulo Raspberry Pi 3 Modelo B.....	35
Figura 21 - Tela do terminal de comando para acesso remoto do Raspberry Pi.	37

Figura 22 - Tela do terminal de comando com o IP do Raspberry Pi.	37
Figura 23 - Tela inicial da interface gráfica do Raspberry Pi com aplicativo Python 3 (IDLE).	38
Figura 24 - (a) Vista frontal e (b) vista traseira da Plataforma ULTRA-ORS.	39
Figura 25 - <i>Phantom</i> de US com suporte para o transdutor utilizado na obtenção dos dados de US com a plataforma ULTRA-ORS.....	39
Figura 26 - Sinais de teste gerados a partir do Simulink.....	41
Figura 27 - Detecção de envoltória a partir do Simulink.	41
Figura 28 - Tela inicial de exibição do Filter Designer.....	42
Figura 29 - Respostas em frequência da Magnitude e Fase do filtro FIR passa-baixa projetado.	43
Figura 30 - Resposta ao Impulso do Filtro FIR passa-baixa projetado.	43
Figura 31 - Respostas em frequência da Magnitude e Fase do filtro FIR Hilbert <i>Transformer</i> projetado.....	45
Figura 32 - Resposta ao Impulso do Filtro FIR Hilbert <i>Transformer</i> projetado.	45
Figura 33 - (a) Sinal Senoidal com ruído e (b) filtrado. (c) FFT do sinal Senoidal com ruído e do (d) sinal senoidal filtrado.....	49
Figura 34 - (a) Sinal do tipo Chirp original e (b) filtrado. (c) FFT do sinal Chirp original e do (d) sinal Chirp filtrado.....	50
Figura 35 - (a) Sinal de US com ruído e (b) filtrado. (c) FFT do sinal de US com ruído e do (d) sinal de US filtrado.....	51
Figura 36 - Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Raspberry Pi/Python.....	52
Figura 37 - Destaque da Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Raspberry Pi/Python.....	53
Figura 38 - Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Matlab.....	53
Figura 39 - Destaque da Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Matlab.....	54
Figura 40 - Comparação da Envoltória obtida a partir do Matlab/Simulink com a Envoltória obtida a partir do Raspberry Pi/Python.....	55
Figura 41 - Comparação da Linha 1 da <i>Scanline</i> 1 obtida a partir do Raspberry Pi com a obtida a partir do Matlab.	56
Figura 42 - Comparação da <i>Scanline</i> 1 obtida a partir do Matlab e do Raspberry Pi.	57

Figura 43 - Comparação da Envoltória da <i>Scanline</i> 1 sem deslocamento de amostras no tempo, obtida a partir do Matlab e do Raspberry Pi.....	58
Figura 44 - Comparação da Envoltória da <i>Scanline</i> 1 obtida a partir do Matlab e do Raspberry Pi.....	59
Figura 45 - Comparação da Compressão Logarítmica da <i>scanline</i> 1 obtida a partir do Matlab e do Raspberry Pi com -30 dB de faixa dinâmica.....	60
Figura 46 - Imagem Final de US sem a conversão de varredura obtida a partir do Raspberry Pi.	61
Figura 47 - Comparação da imagem final de US obtida a partir do (a) processamento no Matlab e (b) no Raspberry Pi.....	62

LISTA DE TABELAS

Tabela 1 - Especificações do Filtro Digital.	42
Tabela 2 - Coeficientes do Filtro FIR passa-baixa projetado.	44
Tabela 3 - Especificações do Filtro FIR Hilbert <i>Transformer</i>	44
Tabela 4 - Coeficientes do Filtro FIR Hilbert <i>Transformer</i> projetado.	46
Tabela 5 - Resultado das Funções de Custo para comparação da Detecção de Envoltória.	54
Tabela 6 - Resultado das Funções de Custo para comparação da Filtragem Digital entre o Matlab e o Rapsberry Pi.	56
Tabela 7 - Resultado das Funções de Custo para comparação das <i>Scanlines</i> obtidas no Matlab e no Rapsberry Pi.	57
Tabela 8 - Resultado das Funções de Custo para comparação da detecção da envoltória sem deslocamento a partir do Matlab e do Raspberry Pi.	58
Tabela 9 - Resultado das Funções de Custo para comparação da detecção da envoltória.	59
Tabela 10 - Resultado das Funções de Custo para comparação da compressão logarítmica. ...	60
Tabela 11 - Resultado das Funções de Custo para a Imagem Final.	63

LISTA DE ABREVIATURAS

A	Amplitude
ADC	<i>Analog-to-Digital Converters</i>
B	Brilho
CPU	<i>Central Processing Unit</i>
DSP	<i>Digital Signal Processor</i>
FFT	<i>Fast Fourier Transform</i>
FIR	<i>Finite Impulse Response</i>
IDE	<i>Integrated Development Environment</i>
IIR	<i>Infinite Impulse Response</i>
IP	<i>Internet Protocol</i>
LUS	Laboratório de Ultrassom
M	Movimento
NRMSE	<i>Normalized Root Mean Squared Error</i>
NRSS	<i>Normalized Residual Sum of Squares</i>
RAM	<i>Random Access Memory</i>
RF	Radiofrequência
ROI	<i>Region of Interest</i>
SSH	<i>Secure Shell</i>
TCC	Trabalho de Conclusão de Curso
ULTRA-ORS	<i>Ultrasound Open Research System</i>
US	Ultrassom
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1	INTRODUÇÃO.....	14
1.1	DELIMITAÇÃO DO TEMA	15
1.2	PROBLEMAS E PREMISSAS	16
1.3	OBJETIVOS	17
1.4	JUSTIFICATIVA	17
1.5	PROCEDIMENTOS METODOLÓGICOS	18
1.6	ESTRUTURA DO TRABALHO	18
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	IMAGEAMENTO POR ULTRASSOM NO MÉTODO PULSO-ECO	19
2.2	IMAGEAMENTO POR ULTRASSOM NO MÉTODO <i>BEAMFORMING</i>	20
2.3	FORMAÇÃO DA IMAGEM EM MODO B	21
2.4	VARREDURA SETORIAL E RETANGULAR DE TRANSDUTORES.....	22
2.5	PROCESSAMENTO DIGITAL DE SINAIS	23
2.6	FILTRAGEM DIGITAL	24
2.6.1	FILTROS FIR.....	25
2.7	ATRASO DE FOCALIZAÇÃO E SOMATÓRIO COERENTE	28
2.8	DEMODULAÇÃO.....	28
2.8.1	TRANSFORMADA DE HILBERT E DETECÇÃO DE ENVOLTÓRIA.....	29
2.9	COMPRESSÃO LOGARÍTIMICA	31
2.10	CONVERSÃO DE VARREDURA (<i>SCAN CONVERSION</i>).....	32
3	MATERIAIS E MÉTODOS	35
3.1	RASPBERRY PI	35
3.1.1	CONFIGURAÇÃO DO RASPBERRY PI.....	36
3.2	PLATAFORMA ULTRA-ORS.....	38
3.3	SIMULINK - MATLAB	40
3.4	FILTER DESIGNER - MATLAB.....	41
3.5	PROJETO DO FILTRO DIGITAL	42
3.6	PROJETO DE DEMODULAÇÃO E DETECÇÃO DA ENVOLTÓRIA	44
3.7	FUNÇÃO DE CUSTO NRMSE	46
3.8	FUNÇÃO DE CUSTO NRSS	47
4	RESULTADOS.....	48
4.1	VALIDAÇÃO DO FILTRO DIGITAL PROJETADO.....	48

4.2	VALIDAÇÃO DA DEMODULAÇÃO E DETECÇÃO DA ENVOLTÓRIA.....	52
4.3	COMPARAÇÃO DO MODELO DO MATLAB COM O MODELO DO RASPBERRY PI.....	55
5	DISCUSSÕES E CONCLUSÕES	64
	REFERÊNCIAS.....	66
	APÊNDICE A.....	70

1 INTRODUÇÃO

No final da década de 40, a visualização de imagens a partir da técnica de ultrassom (US) passou a ser aplicada em testes não destrutivos de materiais e em diagnósticos médicos. Na medicina, a ultrassonografia vem desempenhando um papel fundamental para o diagnóstico médico por imagem, devido as aplicações nas diversas especialidades médicas. Portanto, existe uma alta demanda por essa modalidade de imageamento, que constituem a segunda técnica mais utilizada para o auxílio do diagnóstico por imagem (DEMARRE et al., 1983; SHUNG et al., 1992).

O US é definido como uma onda mecânica e longitudinal, com frequência superior a 20 kHz. A geração das ondas ultrassônicas ocorre a partir da excitação por corrente elétrica de cristais piezoelétricos, presentes nos transdutores de US. Quando os feixes de US incidem nos tecidos do corpo humano, que possuem diferentes impedâncias acústicas, ecos são refletidos, permitindo identificar as superfícies e diferenciar os tecidos. Tipicamente, são utilizados modernos sistemas de US, que figuram entre os mais sofisticados equipamentos de processamentos de sinais na atualidade. Os sistemas de US operam transmitindo pulsos e captando os ecos através de um único transdutor (*probe*). Por este motivo, são conhecidos como equipamentos de aquisição pulso-eco (WELLS, 1993; EVANS et al., 2000).

A grande vantagem de utilizar a ultrassonografia na medicina está na técnica não-invasiva e na radiação não-ionizante, que tornam o método indolor e seguro. O baixo custo quando comparado com outros métodos similares de diagnóstico por imagem, a facilidade de manuseio, a portabilidade e a possibilidade de geração das imagens em tempo real, que permite o diagnóstico mais rápido, são fundamentais para consolidação do US no diagnóstico por imagem (CHRISTENSEN, 1988; SHUNG, 2006).

Com o desenvolvimento de conversores analógicos e digitais nos anos 70, a forma de exibição das imagens mudou significativamente. A partir desse momento, tornou-se possível gerar imagens relacionando as amplitudes dos ecos refletidos com a diferentes tonalidades de escala de cinza, caracterizando assim o princípio utilizado no modo de exibição conhecido como Modo B (brilho). Neste modo, as imagens geradas são bidimensionais e a posição dos pontos na tela é definida a partir da orientação dos ecos e pelo tempo entre eles. Além do Modo B, existem outros métodos de exibição da imagem, destacando-se o Modo A (Amplitude), Modo M (Movimento), Modo Doppler Contínuo e Modo Doppler Pulsátil (FISH, 1990; OTAKE et al., 2003; HEDRICK et al., 2005).

1.1 DELIMITAÇÃO DO TEMA

A geração de imagens por US ocorre através do processamento dos dados recebidos pelo transdutor. Para realizar o processamento digital dos sinais de US, e conseqüentemente gerar as imagens no Modo B, é necessário converter os sinais analógicos em sinais digitais. Os conversores analógicos para digital (ADCs - *Analog-to-Digital Converters*) são os componentes eletrônicos que realizam esta função. A partir do momento em que os sinais são digitalizados, técnicas avançadas de processamento digital podem ser aplicadas em um processador digital de sinal (DSP - *Digital Signal Processor*).

O processamento digital de um sinal tem como vantagens a flexibilidade e repetibilidade. Um único *hardware* pode ser utilizado para implementar diferentes funções descritas matematicamente, que dependem exclusivamente da instrução em *software*. Isto permite que novas atualizações e alterações possam ser realizadas, mantendo o mesmo *hardware*. A geração das imagens em tempo real não é garantida, pois depende da capacidade e velocidade de execução do sistema (HAYKIN, 2001). Entretanto, novos sistemas de US vêm sendo desenvolvidos pela comunidade científica e empresas especializadas, como por exemplo, os equipamentos SonixMDP (ULTRASONIX MEDICAL CORP., CANADÁ) e ACUSON X300 Premium Edition (SIEMENS CORP., ALEMANHA).

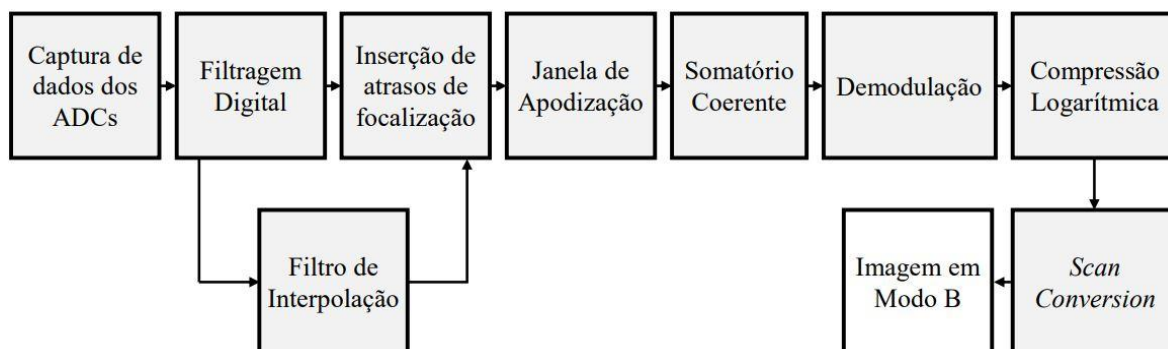
Segundo Assef (2013), as principais etapas de processamento digital dos sinais de US para imageamento em Modo B, são divididas conforme o diagrama de blocos apresentado na Figura 1.

Neste Trabalho de Conclusão de Curso (TCC) serão abordados o estudo e a implementação dos seguintes algoritmos de processamento em um sistema embarcado: filtragem digital, atraso de focalização e somatório coerente, demodulação com detecção de envoltória e compressão logarítmica. A etapa de *scan conversion* (conversão de varredura) foi realizada com os dados processados no sistema de *hardware* embarcado e exportado para o *software* Matlab (THE MATHWORKS INC., EUA) para visualização.

Dessa forma, este trabalho tem a finalidade de avaliar o módulo Raspberry Pi 3 Modelo B para realizar o processamento de sinais de US. A família Raspberry Pi é composta de minicomputadores, com o tamanho de um cartão de crédito, que suportam distribuições de Linux e Windows. Com capacidade de fazer o processamento de texto e reprodução de vídeos em alta definição, o módulo pode conter conexão de rede sem fio e Bluetooth, dependendo da versão (RASPBERRYPI, 2018; ROBOCORE, 2018). O módulo utilizado

está disponível no Laboratório de Ultrassom (LUS) da Universidade Tecnológica Federal do Paraná (UTFPR), câmpus Curitiba.

Figura 1 - Diagrama em blocos das principais funções de processamento digital para geração de imagem no Modo B.



Fonte: Assef (2013).

1.2 PROBLEMAS E PREMISSAS

Os *scanners*, como também são conhecidos os aparelhos de ultrassonografia, são produtos que necessitam de um alto investimento. Além do mais, os sistemas atuais não permitem a pesquisa de técnicas alternativas e inovadoras de processamento digital para geração da imagem, pois possuem uma arquitetura fechada e fixa.

O grande desafio encontra-se na geração de imagem em tempo real a partir do processamento massivo em altas taxas de transferência de dados, que necessita ser realizado tanto em *software* quanto em *hardware*.

Importante destacar que existem poucas pesquisas ou trabalhos conhecidos na literatura científica que realizem o processamento digital de sinais de US embarcado em um módulo da família Raspberry Pi. Assim, surge uma questão: é possível utilizar o módulo Raspberry Pi para processar dados de US para pesquisa sobre geração de imagens? Dessa forma, este TCC buscou avaliar o desempenho da plataforma Raspberry Pi em relação a qualidade dos resultados e velocidade de processamento.

1.3 OBJETIVOS

O objetivo geral deste trabalho foi realizar o processamento digital de sinais de US para geração de imagem no Modo B, a partir de dados brutos de radiofrequência (RF) disponíveis, utilizando um módulo Raspberry Pi 3 Modelo B.

Além do objetivo geral, este trabalho tem como objetivos específicos:

- 1) Compreender as principais características dos sinais de US para formação de imagem no Modo B;
- 2) Pesquisar sobre o funcionamento do Raspberry Pi 3 Modelo B, destacando as principais características, ferramentas, plataforma e linguagem de programação para implementação de algoritmos;
- 3) Projetar um filtro digital para os dados de US amostrados a 40 MHz, estabelecendo os parâmetros e coeficientes;
- 4) Desenvolver a etapa de Demodulação com Detecção de Envoltória utilizando estruturas de filtro digital;
- 5) Implementar no Raspberry Pi 3 Modelo B, o Filtro Digital projetado, o Somatório Coerente, a Demodulação com Detecção de Envoltória e a Compressão Logarítmica;
- 6) Avaliar as etapas de processamento desenvolvidas utilizando dados obtidos por simulação e experimentalmente a partir da plataforma de pesquisa ULTRA-ORS (ASSEF, 2013);
- 7) Transferir os dados processados do Raspberry Pi 3 para um computador para reconstrução de imagens utilizando o *software* Matlab (THE MATHWORKS INC., EUA);
- 8) Gerar imagens de US em Modo B, avaliando a qualidade da imagem e o desempenho do Raspberry Pi 3 Modelo B.

1.4 JUSTIFICATIVA

O crescente desenvolvimento tecnológico computacional auxilia na implementação de novos algoritmos de processamento digital em novas plataformas de *hardware* e *software*, permitindo aprimorar a qualidade e a pesquisa dos métodos de reconstrução de imagens de US.

No Brasil existe um grande potencial para desenvolver no âmbito acadêmico o setor de equipamentos de imagem médica e, conseqüentemente, contribuir para o desenvolvimento científico, tecnológico e inovador na área de engenharia biomédica, especificamente no diagnóstico por imagem a partir do US.

Futuramente, este trabalho poderá servir de base para estudo de novas pesquisas para o desenvolvimento de um equipamento de ultrassonografia. Tem-se como meta buscar utilizar uma nova plataforma aberta para integrar inúmeras técnicas de processamento digital. O Raspberry Pi 3 Modelo B, surge como possibilidade de *hardware* acessível, portátil e com ótimo custo benefício, apto a realizar as diferentes tarefas do processamento digital, a partir da implementação de algoritmos que descrevam matematicamente as funções desejadas.

1.5 PROCEDIMENTOS METODOLÓGICOS

Para a realização do trabalho foram utilizados dados brutos de US que podem ser gerados por simulação ou através do sistema de pesquisa do US, ULTRA-ORS (*Ultrasound Open Research System*), disponível no LUS (ASSEF, 2013). Após as pesquisas sobre as técnicas de geração de imagens por US, foram avaliadas as ferramentas de programação do Raspberry Pi para o desenvolvimento dos algoritmos de processamento. Os resultados de cada etapa de processamento do Raspberry Pi foram comparados com resultados de referência gerados através do *software* Matlab.

1.6 ESTRUTURA DO TRABALHO

Este trabalho está organizado em cinco capítulos. No Capítulo 1 é apresentada a introdução, que aborda principalmente o tema e os objetivos do trabalho. No Capítulo 2 são abordados os principais conceitos necessários para o desenvolvimento da pesquisa. O Capítulo 3 apresenta os materiais e métodos utilizados na pesquisa. No Capítulo 4 são apresentados os resultados das etapas de processamento digital de sinais realizadas no Raspberry Pi em comparação com os resultados obtidos no Matlab/Simulink. No Capítulo 5 são abordadas as discussões e conclusões deste TCC.

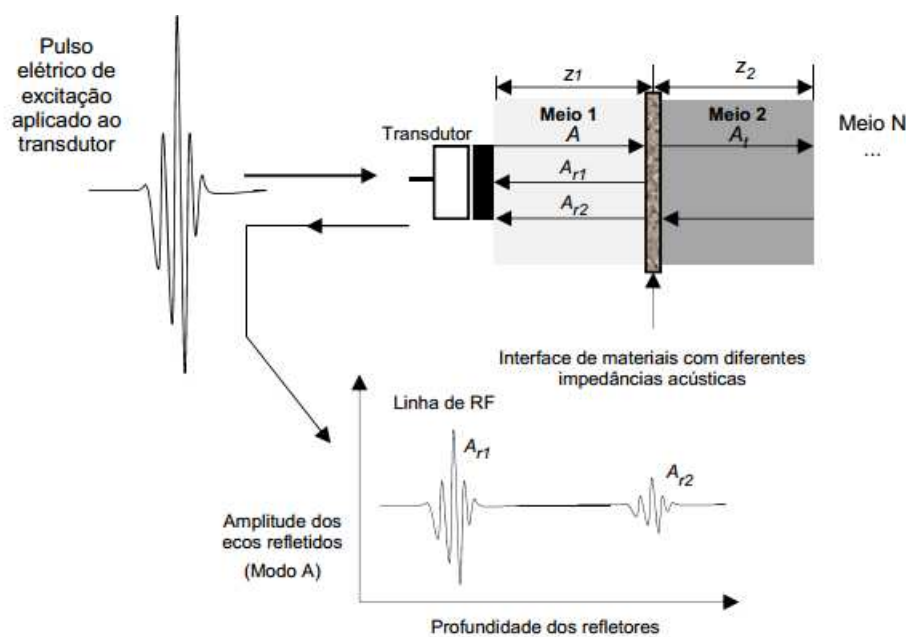
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão descritos os principais conceitos para geração de imagens por US no Modo B. Esses conceitos são importantes para uma melhor compreensão das etapas do processamento digital desenvolvido.

2.1 IMAGEAMENTO POR ULTRASSOM NO MÉTODO PULSO-ECO

O método pulso-eco serve de base para obtenção de imagens por US. Pulsos elétricos estreitos e de alta tensão excitam o cristal piezoelétrico presente no transdutor e são convertidos em pulsos de ondas ultrassônicas. As ondas de US são refletidas e espalhadas nas interfaces entre os meios e tecidos com diferentes propriedades acústicas para posteriormente serem captadas pelo mesmo cristal piezoelétrico que, dessa vez, faz a conversão no sentido contrário, dando origem ao sinal elétrico de RF. O sinal recebido de RF contém as informações sobre a posição e densidade dos objetos refletores, e quando representados como função do tempo, caracteriza a formação da imagem em Modo A, conforme ilustrado na Figura 2 (HEDRICK; HYKES; STARCHMAN, 2005; JINBO, 2007).

Figura 2 - Representação da geração de ondas de US pelo método pulso-eco para determinação e caracterização de diferentes meios.



Fonte: Adaptado de Ferreira (2017).

A Figura 2 apresenta dois meios com diferentes impedâncias acústicas e diferentes profundidades. A onda de US transmitida proveniente do transdutor se propaga no Meio 1 e incide na interface entre os dois meios. Parte da onda é refletida ($Ar1$) e outra parte propaga-se no Meio 2 (At). De forma similar, o sinal At é refletido, originando o sinal $Ar2$. As ondas refletidas $Ar1$ e $Ar2$ representam o sinal de RF que carrega as informações sobre as características do meio. Através da Equação (1), pode-se calcular a distância z ou a velocidade de propagação da onda c no meio a partir de um tempo t decorrido entre a aplicação do pulso de excitação e a recepção do eco (CHRISTENSEN, 1988).

$$c = \frac{2 \cdot z}{t} \quad (1)$$

2.2 IMAGEAMENTO POR ULTRASSOM NO MÉTODO *BEAMFORMING*

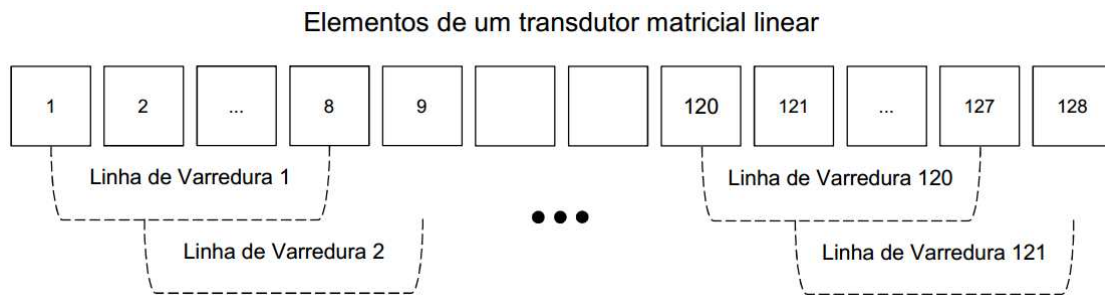
No método *beamforming*, como é conhecido o processo de utilização de múltiplas ondas acústicas para orientar e focalizar o campo acústico em uma determinada região de interesse (ROI - *Region of Interest*), um conjunto de elementos é excitado com tempo de atraso programável – cada elemento de uma abertura é excitado com um tempo de atraso específico para realizar a focalização sintética. Após a recepção, que ocorre de maneira similar ao método pulso-eco, a abertura sintética é deslocada em um elemento e a sequência é repetida, até o último elemento da matriz (JENSEN et al, 2006; HASSAN; YOUSSEF; KADAH, 2011).

Como exemplo, na Figura 3 é ilustrada a sequência de aberturas de cada linha de varredura (*scanline*) com oito elementos ativos, para um transdutor matricial linear com 128 elementos. O número resultante de *scanlines* (n_{sc}) pode ser calculado através da Equação (2):

$$n_{sc} = ne - na + 1, \quad (2)$$

no qual ne é o número total de elementos do transdutor e na é o número de elementos ativos da abertura. Neste caso, o número resultante de *scanlines* é 121.

Figura 3 - Exemplificação da sequência de aberturas das linhas de varredura em um transdutor matricial linear de 128 elementos no método Beamforming.

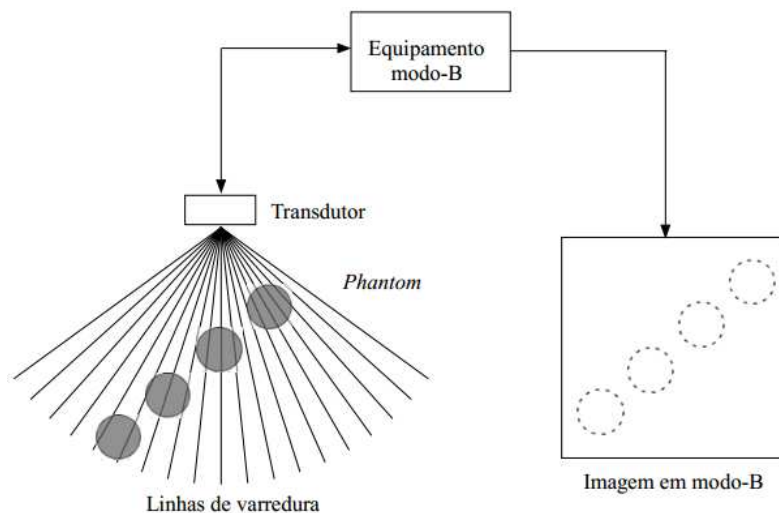


Fonte: Ferreira (2017).

2.3 FORMAÇÃO DA IMAGEM EM MODO B

O Modo B é composto por um conjunto de sinais em Modo A, provenientes de uma linha de varredura concentrada em um ponto focal, para formar uma imagem bidimensional. Os tempos de atraso de recepção e a posições das linhas de varredura determinam a disposição espacial do ponto. Cada linha de varredura é representada por um sinal de RF no Modo A, conforme mostrado na Figura 4 (HEDRICK; HYKES; STARCHMAN, 2005).

Figura 4 - Representação de um equipamento de US para formação da imagem em Modo B.



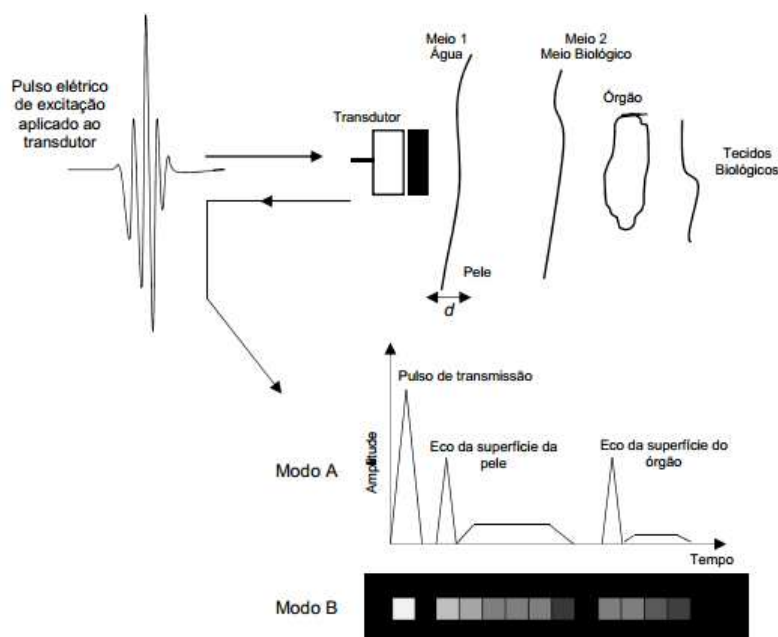
Fonte: Jinbo (2007).

A amplitude do sinal de RF é utilizada para modular a intensidade do brilho de cada *pixel* (elemento de imagem) em tonalidades de cinza, conforme ilustrado na Figura 5.

As diferentes amplitudes dos sinais de RF refletidos que compõem o Modo A ocorre devido os fatores geométricos como, por exemplo, dimensões da fonte sonora, comprimento

de onda e presença de superfícies refletoras. Os mecanismos de absorção, nos quais parte da energia é convertida em calor, e a profundidade da região de interesse também influenciam a amplitude do sinal recebido (WELLS, 1999; OTAKE et al., 2003; HEDRICK; HYKES; STARCHMAN, 2005; SHUNG, 2006).

Figura 5 - Representação da modulação do brilho no Modo B de acordo com a amplitude no Modo A.



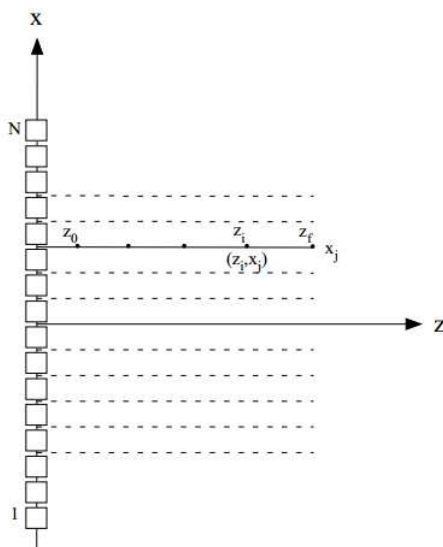
Fonte: Adaptado de Shung (2006).

2.4 VARREDURA SETORIAL E RETANGULAR DE TRANSDUTORES

Os transdutores matriciais, utilizados na formação da imagem por US no Modo B, podem apresentar dois tipos de varredura: retangular ou setorial. A Figura 6 apresenta uma matriz linear com N elementos e varredura retangular, na qual a formação da imagem é dada em coordenadas cartesianas (x,z) que constituirão diretamente uma linha na composição da imagem final. A coordenada x representa o afastamento em relação ao eixo central e z representa a profundidade (JINBO, 2007).

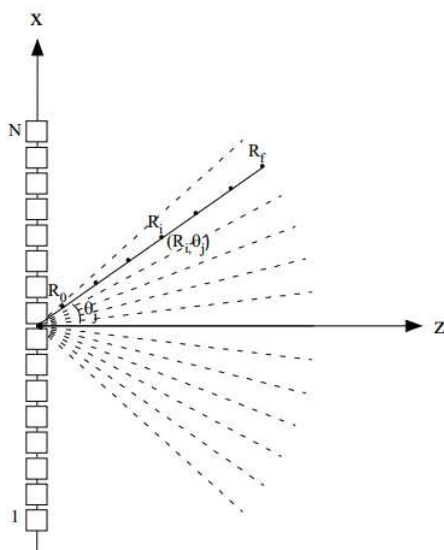
Já na Figura 7, a matriz linear com N elementos realiza uma varredura setorial. O setor da imagem é composto pela soma coerente dos sinais em coordenadas polares (r,θ) , com profundidade radial r e ângulo de deflexão θ (JINBO, 2007).

Figura 6 - Varredura retangular e amostras do sinal coerente para uma determinada linha da imagem.



Fonte: Jinbo (2007).

Figura 7 - Varredura setorial e amostras do sinal coerente para um determinado setor da imagem.



Fonte: Jinbo (2007).

2.5 PROCESSAMENTO DIGITAL DE SINAIS

Define-se o processamento digital de sinais como a matemática, equacionamentos, algoritmos e técnicas implementadas para representar, transformar e manipular sinais que são convertidos do domínio de tempo contínuo para o domínio de tempo discreto (SMITH, 2003).

O processamento digital de sinais surgiu como uma alternativa para o processamento analógico; tem por vantagem a reconfiguração do sistema e fácil atualização do processamento com a alteração da programação, sem a necessidade de alteração de *hardware* (PROAKIS; MANOLAKIS, 2007). Por esses motivos, em diversas aplicações de telecomunicações, instrumentação e engenharia biomédica, o processamento digital é utilizado para modificar algumas características dos sinais ou extrair informações sem perdas ou distorções (OPPENHEIM; SCHAFER, 1975).

Na geração de imagens por US no Modo B, conforme descrito na seção anterior, várias etapas de processamento digital são utilizadas. A filtragem digital para eliminar frequências de bandas indesejadas é a primeira. Em seguida, a inserção de atrasos para alinhar e possibilitar a soma coerente que irá compor uma linha de varredura. Posteriormente, a demodulação e detecção de envoltória do sinal coerente para extração do espectro em Modo A. Em sequência, a compressão logarítmica para reduzir a faixa dinâmica e adequar os dados na escala de cinza para visualização em Modo B. Por fim, a conversão de varredura para apresentação da imagem em um monitor (JINBO, 2007; ASSEF, 2013; HASSAN; KADAH, 2013).

Esses processos são comuns na geração e melhoria da qualidade da imagem de US, e serão esclarecidos a seguir.

2.6 FILTRAGEM DIGITAL

A filtragem digital é uma das técnicas de processamento digital mais significativas para melhorar a qualidade das imagens reconstruídas por US. É utilizada para eliminar a componente contínua e os ruídos da radiofrequência de eco. Os ruídos gerados a partir da estrutura não homogênea do tecido, diminuem a qualidade da imagem, principalmente o contraste (JINBO, 2007).

Um filtro digital pode ser representado no domínio da frequência através da resposta de magnitude e fase para uma entrada do tipo impulso. Assim, divide-se a classificação em dois tipos de filtros. Os filtros de resposta ao impulso finita (FIR – *Finite Impulse Response*), no qual a resposta ao impulso tem um número finito de amostras, e os filtros de resposta ao impulso infinita (IIR – *Infinite Impulse Response*), em que a resposta ao impulso possui um número infinito de amostras. Nos filtros FIR, essas amostras são referidas como os coeficientes do filtro (RABINER; SCHAFER, 1978; MADISSETTI; WILLIAMS, 1998).

Por apresentar inerente estabilidade, fácil implementação, resposta de fase linear e não necessitar de realimentação, os filtros FIR geralmente são preferidos no processamento digital de sinais (JI-FENG et al., 2015).

Além da classificação citada acima, um filtro pode ter quatro configurações básicas distintas em relação à função executada, podendo ser passa-baixa, passa-alta, passa-faixa ou rejeita-faixa. As próprias denominações já apontam as características de cada um conforme as frequências de banda passante ou atenuante (OPPENHEIM; WILLSKY, 2010).

O trabalho de Assef et al. (2016) propõe o projeto de filtro digital FIR passa-baixa para aplicação em processamento de sinais de US. Por esse motivo, serão apresentados a seguir o equacionamento e as características de um filtro FIR passa-baixa.

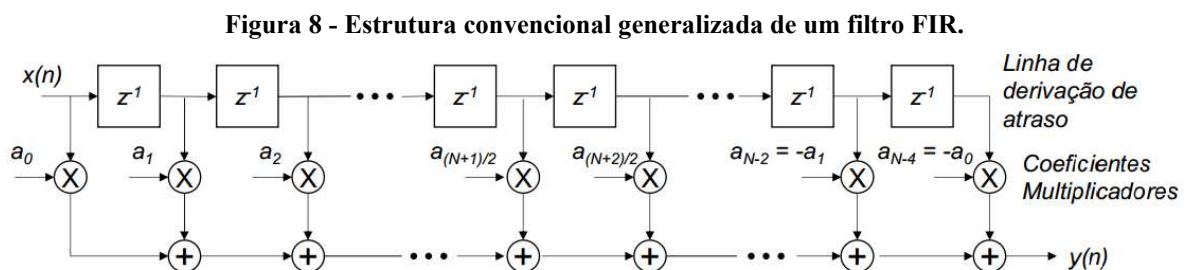
2.6.1 FILTROS FIR

Os filtros FIR são representados matematicamente por uma sequência de multiplicações e somas, conforme a Equação (3):

$$y[n] = \sum_{k=0}^{M-1} a_k x[n - k], \quad (3)$$

em que $x[n]$ é o sinal de entrada, $y[n]$ é o sinal de saída, n é o índice da amostra, a_k são os coeficientes do filtro e M é a ordem do filtro (RABINER; SCHAFER, 1978; JI-FENG et al., 2015).

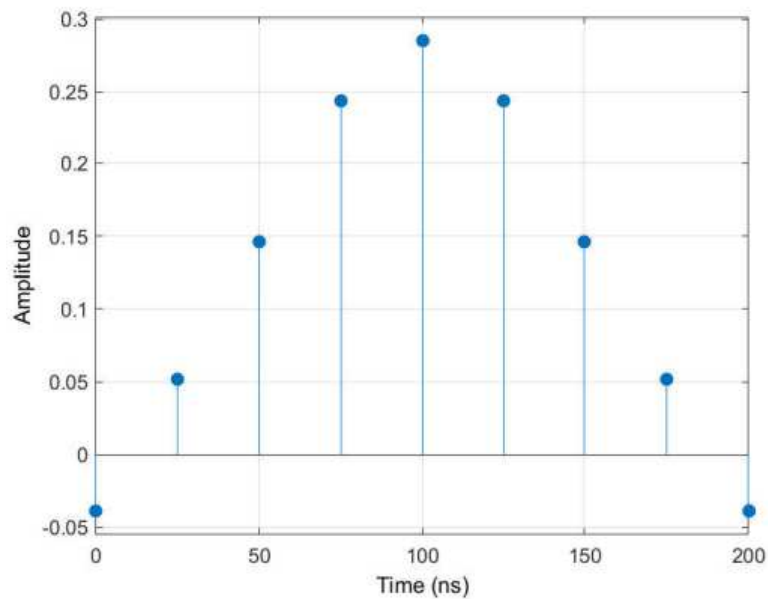
A partir da interpretação da Equação (3), define-se a estrutura da forma direta de um filtro FIR, conforme o diagrama em blocos da Figura 8, no qual z^{-1} é o fator de atraso.



Fonte: Ferreira (2017).

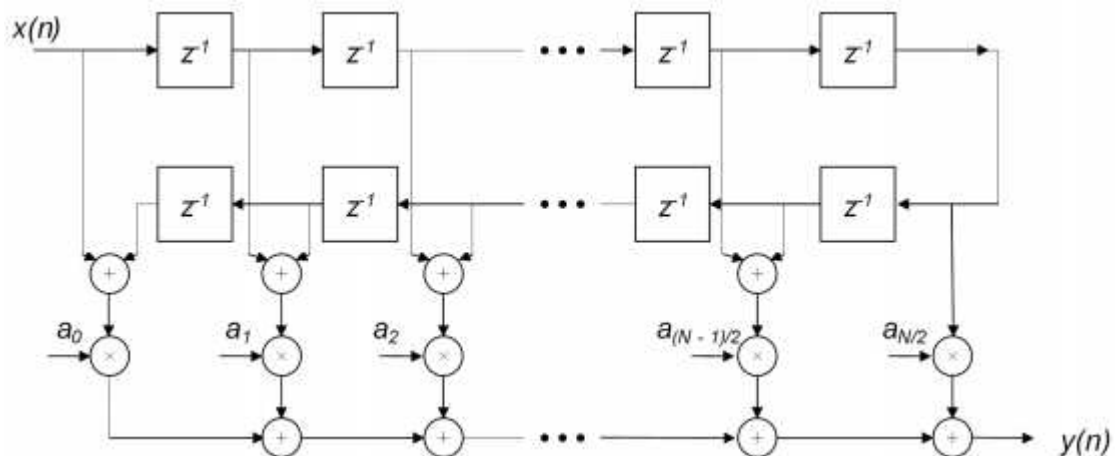
De acordo com a resposta ao impulso do filtro, é possível determinar o tipo de simetria existente (ímpar ou par). Conseqüentemente, pode-se explorar cada tipo de simetria e reduzir os requisitos do sistema, diminuindo o número de multiplicadores e somadores (FERREIRA, 2017). Exemplificando, na Figura 9 é ilustrado a resposta ao impulso para um filtro FIR com simetria ímpar de 9 coeficientes. Assim, o diagrama mostrado na Figura 8 pode ser otimizado e substituído pelo diagrama ilustrado na Figura 10.

Figura 9 - Exemplo de Resposta ao Impulso de um Filtro FIR com simetria ímpar.



Fonte: Autoria Própria (2019).

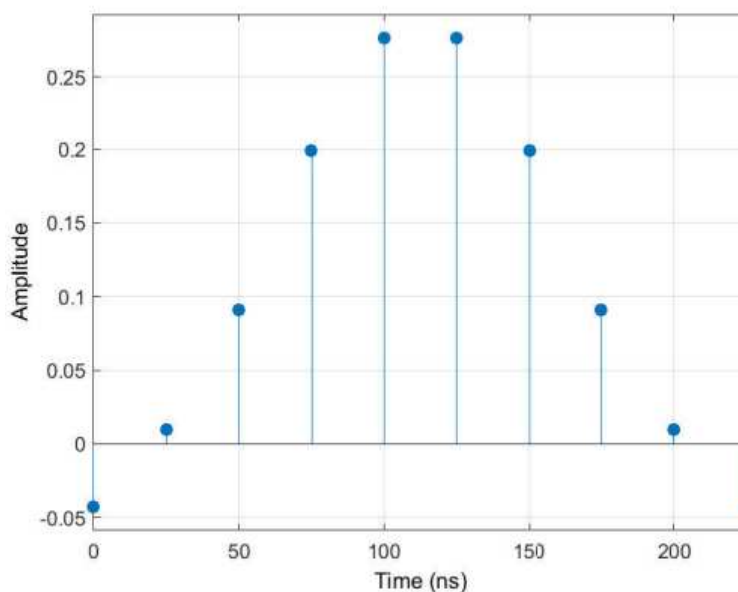
Figura 10 - Estrutura de um filtro FIR com simetria ímpar de coeficientes.



Fonte: Ferreira (2017).

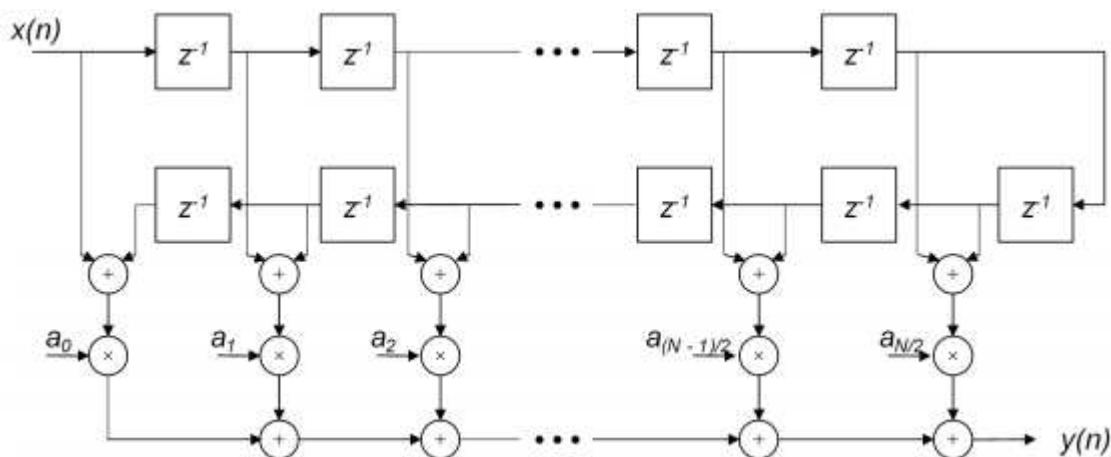
O mesmo raciocínio pode ser utilizado para a simetria par. Por exemplo, na Figura 11 é apresentada a resposta ao impulso para um filtro FIR com simetria par de 10 coeficientes. Neste caso, o diagrama mostrado na Figura 8 também pode ser otimizado e substituído pelo diagrama ilustrado na Figura 12.

Figura 11 - Exemplo de Resposta ao Impulso de um Filtro FIR com simetria par.



Fonte: Autoria Própria (2019).

Figura 12 - Estrutura de um filtro FIR com simetria par de coeficientes.



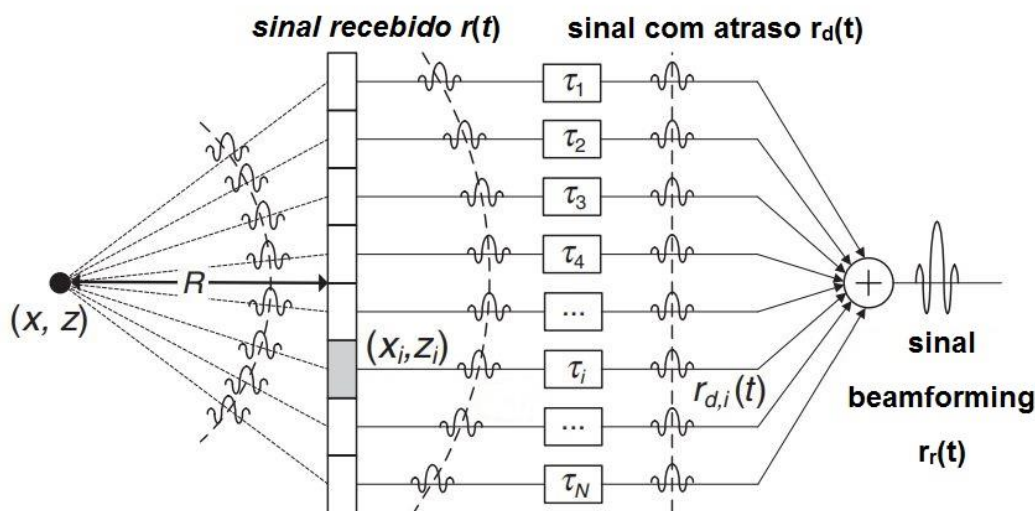
Fonte: Ferreira (2017).

Conseqüentemente, enquanto a estrutura convencional utiliza M multiplicações, o diagrama explorando a simetria ímpar requer $(M+1)/2$ multiplicações e o diagrama explorando a simetria par requer $M/2$ multiplicações (FERREIRA, 2017).

2.7 ATRASO DE FOCALIZAÇÃO E SOMATÓRIO COERENTE

Na técnica de focalização *beamforming* para reconstrução de imagens por US, os elementos piezoelétricos são excitados com tempos de atraso distintos. Consequentemente, os sinais de RF recebidos possuem a mesma característica de atraso. Por esse motivo, o mesmo perfil de atraso de focalização das ondas utilizado na transmissão ultrassônica também deve ser aplicado na recepção. A correção é realizada aplicando coeficientes de atraso temporal (*delay*) nos ecos refletidos. Após o alinhamento dos sinais, realiza-se o somatório coerente destes sinais para compor uma linha de varredura (HASSAN; KADAH, 2013). Na Figura 13 é exemplificado esse procedimento, no qual $r(t)$ é o eco refletido e captado por cada elemento i do transdutor, $r_d(t)$ é o sinal após o ajuste de atraso e $r_r(t)$ é o resultado do somatório coerente dos sinais após a correção do atraso para formação da *scanline*.

Figura 13 - Exemplo de Correção de atraso e Somatório Coerente.



Fonte: Adaptado de Sohn et al. (2011).

2.8 DEMODULAÇÃO

Para geração de imagens por US em Modo B, cujo brilho é modulado de acordo com a amplitude do sinal em Modo A, são necessárias as operações de demodulação seguida pela detecção de envoltória para extrair do sinal soma coerente a linha de varredura final da faixa de frequência da portadora do eco (JINBO, 2007; ASSEF, 2013).

Segundo Zhou e Zheng (2015), tipicamente são aplicados três métodos com esse propósito: demodulação baseada em filtragem, demodulação de quadratura e demodulação baseada na Transformada de Hilbert. O primeiro método caracteriza-se pela retificação do sinal seguida da filtragem passa-baixa para detectar o envelope. No segundo método, o sinal é multiplicado por portadores de senos e cossenos e filtrado por um filtro passa-baixa. O terceiro método utiliza a Transformada de Hilbert discreta para criar um sinal analítico de US.

A demodulação baseada em filtragem é a mais simples e possui o melhor custo benefício. Porém, o seu desempenho é significativamente deteriorado em aplicações de imagens dinâmicas. Por outro lado, a demodulação através da Transformada de Hilbert é mais eficiente e precisa (LEVESQUE; SAWAN, 2009). Por esses motivos, foi abordado neste trabalho a detecção de envoltória a partir da Transformada de Hilbert.

2.8.1 TRANSFORMADA DE HILBERT E DETECÇÃO DE ENVOLTÓRIA

A Transformada de Hilbert para demodulação e detecção de envoltória do sinal de US gera um sinal analítico complexo. A componente imaginária (em quadratura) representa a Transformada de Hilbert do sinal original, produzida a partir de um filtro FIR Hilbert *Transformer* com resposta de fase linear. A componente real (em fase) representa o sinal original atrasado, para compensar o atraso de fase durante a obtenção da componente em quadratura (LEVESQUE; SAWAN, 2009; OPPENHEIM; SCHAFER, 1975).

A envoltória do sinal é obtida através do valor absoluto da Transformada de Hilbert do sinal (OPPENHEIM; SCHAFER, 1975), conforme a Equação (4):

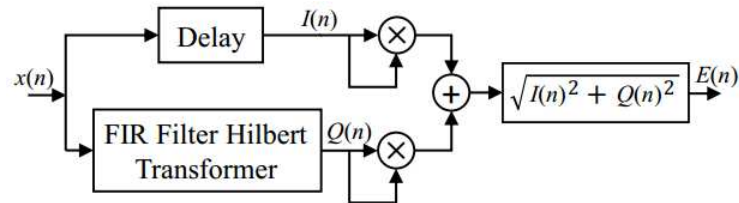
$$E(n) = |\text{Hilbert}(x(n))|, \quad (4)$$

na qual $E(n)$ é a envoltória do sinal e $x(n)$ é o sinal que se deseja obter a envoltória. Matematicamente, o valor absoluto da Transformada de Hilbert é calculado pela raiz quadrada da soma dos quadrados das componentes em fase $I(n)$ e em quadratura $Q(n)$ (OPPENHEIM; SCHAFER, 1975), de acordo com a Equação (5):

$$|\text{Hilbert}(x(n))| = \sqrt{I(n)^2 + Q(n)^2} \quad (5)$$

Todo processo descrito anteriormente pode ser representado pelo diagrama em blocos ilustrado na Figura 14.

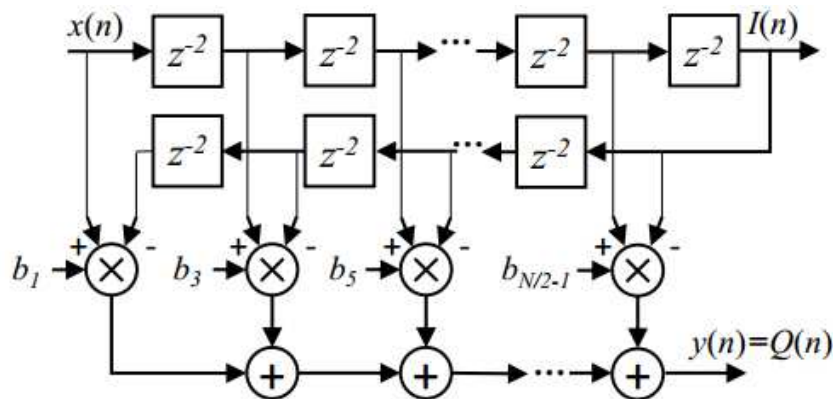
Figura 14 - Diagrama de blocos para detecção de envoltória.



Fonte: Assef et al. (2019).

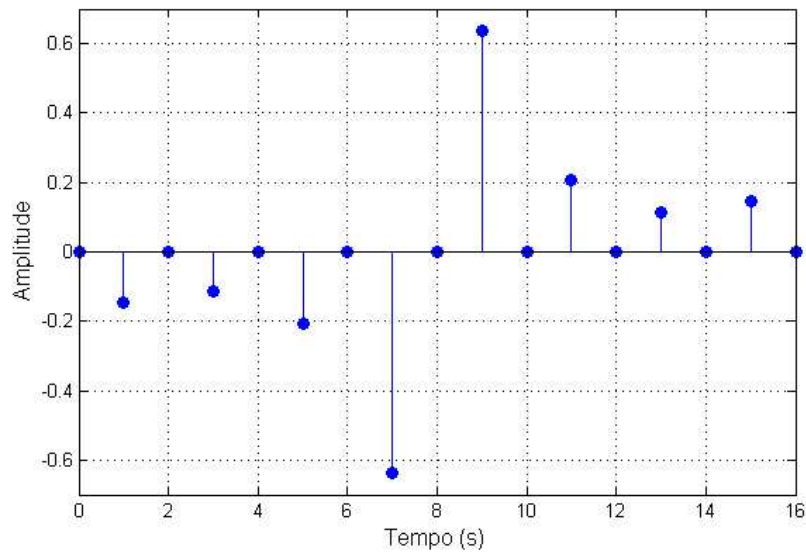
A Transformada de Hilbert baseada em filtro FIR utiliza a mesma estrutura de convolução de um filtro FIR convencional, conforme é ilustrado na Figura 15. No entanto, os coeficientes b_N são obtidos a partir da Transformada de Hilbert. Esse tipo de filtro tem como características a resposta ao impulso antissimétrica e os coeficientes de índice par com valor zero, que podem ser observadas na Figura 16 (ZHOU; ZHENG, 2015; ASSEF et al., 2018).

Figura 15 - Estrutura da Transformada de Hilbert baseada em filtro FIR para obtenção das componentes em quadratura e fase.



Fonte: Adaptado de Assef et al. (2018).

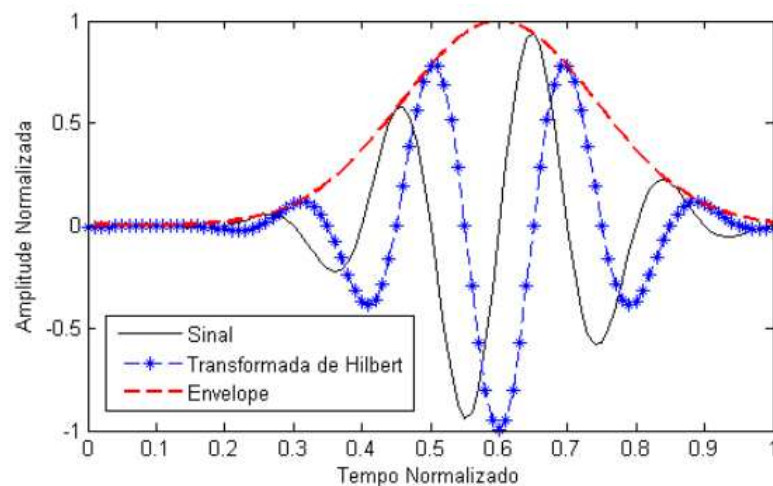
Figura 16 - Exemplo de Resposta ao Impulso de um Filtro FIR Hilbert Transformer.



Fonte: Autoria Própria (2019).

Para exemplificar a aplicação da Transformada de Hilbert para detecção de envoltória (envelope) de um sinal de US, apresenta-se a Figura 17.

Figura 17 - Exemplo de detecção da envoltória de um sinal de US a partir da Transformada de Hilbert.



Fonte: Assef (2013).

2.9 COMPRESSÃO LOGARÍTIMICA

A alta variação na amplitude dos dados de US faz com que a faixa dinâmica de uma linha de varredura possa exceder os níveis da escala de cinza que podem ser exibidos por um monitor de computador. No caso de a imagem ser mapeada linearmente, pontos com

amplitudes elevadas ofuscam pontos com amplitudes menores, prejudicando o contraste da imagem. Dessa maneira, a compressão logarítmica é utilizada para adequar a faixa dinâmica da linha de varredura nos níveis da escala de cinza para visualização no Modo B, melhorando o contraste da imagem (FERREIRA, 2017). Conseqüentemente, a compressão logarítmica também pode ser aplicada para ajustar o brilho da imagem (ALI; MAGEE; UDAYAN, 2008).

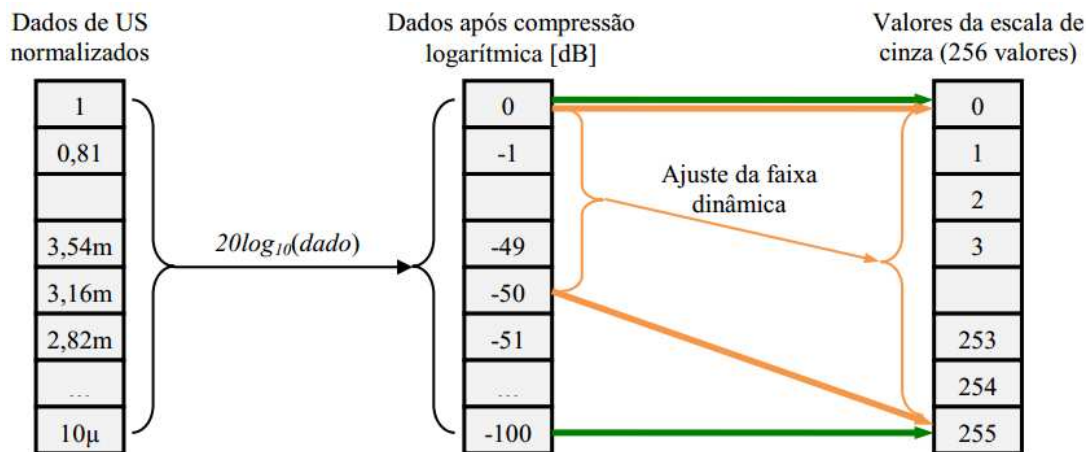
Na compressão logarítmica, os valores dos dados de US são mapeados não linearmente por uma função logarítmica, através da Equação (6):

$$Ec(n) = 20 \log_{10}(E(n)) \quad (6)$$

na qual $E(n)$ é o sinal da envoltória e $Ec(n)$ é o sinal da envoltória comprimido, dado em decibéis (dB) (JINBO, 2007).

Um exemplo de compressão logarítmica com ajuste da faixa dinâmica é ilustrado na Figura 18. Considerando um monitor de 8 bits, com 256 níveis da escala de cinza, ajusta-se a faixa dinâmica dos dados comprimidos de -100 dB para -50 dB.

Figura 18 - Exemplo de compressão logarítmica com adequação da faixa dinâmica para visualização.



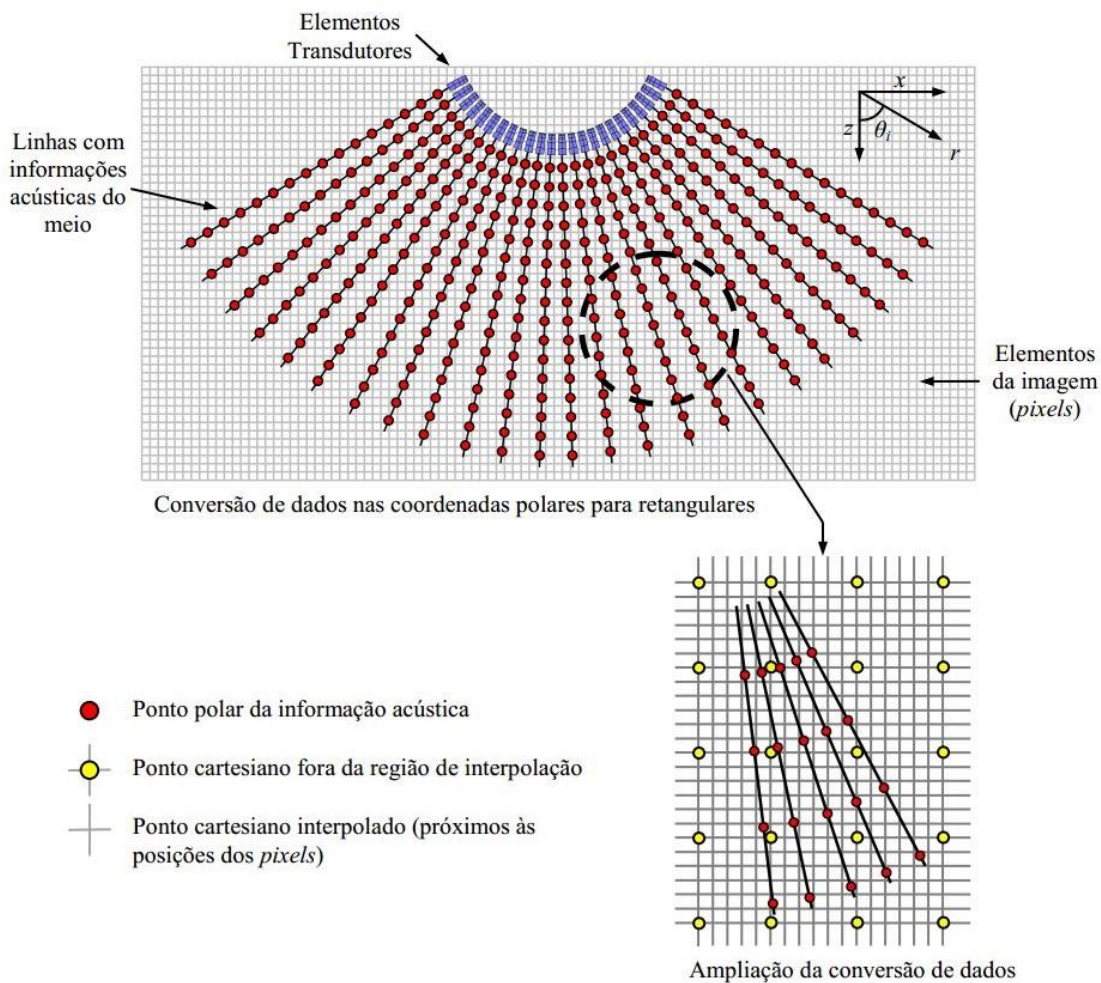
Fonte: Assef (2013).

2.10 CONVERSÃO DE VARREDURA (SCAN CONVERSION)

Na geração de imagens no Modo B, as imagens são apresentadas em coordenadas cartesianas. Já os dados podem ser adquiridos em coordenadas cartesianas, através da inspeção por varredura linear, ou adquiridos em coordenadas polares, através da inspeção por varredura setorial (JINBO, 2007).

A incompatibilidade do sistema de coordenadas da varredura setorial, normalmente utilizada nos sistemas de US, com o sistema de coordenadas de visualização, necessita a realização da etapa de conversão de varredura ou *Scan Conversion*. Na Figura 19 é exemplificado o funcionamento desta etapa.

Figura 19 - Exemplo de conversão de varredura do sistema de coordenadas polares para o sistema de coordenadas cartesianas.



Fonte: Assef (2013).

A relação entre as coordenadas (x, z) do sistema cartesiano e as coordenadas (r, θ) do sistema polar, são determinadas pelas Equações (7) e (8):

$$x = r \sin(\theta) \quad (7)$$

$$z = r \cos(\theta) \quad (8)$$

Verifica-se na Figura 19 que a região mais próxima aos elementos transdutores possuem uma densidade maior dos feixes em relação as regiões mais afastadas. Nessas regiões distantes, os feixes não cobrem todos os *pixels* e, portanto, não endereçam as amostras (ASSEF, 2013).

Segundo Jinbo (2007), nesse tipo de aplicação, a técnica mais comum e adequada para converter as coordenadas dos dados e solucionar esse problema é a interpolação bilinear, por proporcionar uma imagem com boa qualidade. Em geral, os pontos interpolados são calculados com base nos pontos vizinhos para aproximar os endereços dos elementos da imagem da posição original dos sinais discretizados.

Normalmente os sistemas utilizam a interpolação 2x2, baseada em quatro vizinhos mais próximos. Entretanto, alguns sistemas também utilizam a interpolação 4x4, baseada em 16 vizinhos mais próximos (ALI; MAGEE; UDAYAN, 2008).

Neste trabalho, todas as etapas anteriores, exceto a correção de atraso, foram realizadas na placa Raspberry Pi, sendo que a correção de coordenadas e apresentação da imagem reconstruída foram realizadas em um computador com o *software* Matlab. A inserção de atraso não foi implementada, pois foram utilizados dados de entrada pré-processados com a correção de atraso. Entretanto, essa etapa pode ser realizada facilmente no Raspberry Pi através de um deslocamento nas amostras após a filtragem.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentados os materiais e ferramentas utilizadas, a configuração da plataforma Raspberry Pi, as especificações de projeto das etapas de filtragem digital, detecção de envoltória e as métricas quantitativas para análise posterior dos resultados e validação do processamento de sinais de US embarcado no Raspberry Pi.

3.1 RASPBERRY PI

O Raspberry Pi é um dispositivo desenvolvido por uma organização inglesa sem fins lucrativos, chamada Raspberry Pi *Foundation* (Fundação Raspberry Pi), cujo principal objetivo é estimular o ensino da ciência da computação.

Neste trabalho, o Raspberry Pi Modelo B, ilustrado na Figura 20, é utilizado para realizar o processamento embarcado de sinais de US para reconstrução de imagens. O módulo apresenta as seguintes especificações principais: Processador Quad Core de 1,2 GHz Broadcom BCM2837, CPU (*Central Processing Unit*) de 64 bits e Memória RAM (*Random Access Memory*) de 1 GB (RASPBERRYPI, 2018).

Figura 20 - Módulo Raspberry Pi 3 Modelo B.



Fonte: RaspberryPi (2018).

O próprio desenvolvedor da plataforma disponibiliza gratuitamente o sistema operacional oficial Raspbian para o dispositivo. Integrado a este sistema, encontra-se o Python 3 IDE (*Integrated Development Environment*), que permite o desenvolvimento de algoritmos usando a linguagem de programação Python (RASPBERRYPI, 2018).

O Python é uma linguagem de programação que permite trabalhar mais facilmente e integrar diversos sistemas com eficácia. Atualmente, é uma das principais linguagens utilizadas e difundidas (PYTHON, 2019).

3.1.1 CONFIGURAÇÃO DO RASPBERRY PI

A inicialização e configuração da plataforma Raspberry Pi foi realizada conforme a documentação disponível no site do fabricante. Para o armazenamento de todos os arquivos e do sistema operacional foi utilizado um cartão de memória com capacidade de armazenar 16 GB. A instalação do sistema operacional Raspbian foi efetuada através do instalador NOOBS, que é disponibilizado pelo fabricante.

Neste trabalho, o acesso ao Raspberry Pi foi feito remotamente a partir de um computador. Para isso, usou-se o aplicativo VNC *Viewer* no computador que permite acessar a interface gráfica através da conexão dos dispositivos na mesma rede. Uma vantagem do sistema operacional Raspbian é que já está incluso o aplicativo VNC *server*, necessário ser executado no Raspberry Pi para que a conexão seja efetuada com sucesso.

Primeiramente, é preciso habilitar a SSH (*Secure Shell*) do Raspberry Pi, que por padrão é desativado. O SSH é um protocolo usado para gerenciar sistemas e aplicativos de maneira remota a partir de um terminal, cuja a transferência de dados é criptografada. Esse processo pode ser feito adicionando um arquivo, sem nenhuma extensão, com o nome “ssh” no cartão de memória. Quando o arquivo for encontrado durante a inicialização do Raspberry Pi, o SSH é ativado e o arquivo é excluído.

Com o auxílio do aplicativo Pi Finder da Adafruit Industries, que permite encontrar o Raspberry Pi na rede e se conectar através da SSH, foi possível executar o VNC *server* no Raspberry Pi a partir da instrução “vncserver” no terminal de comando, conforme ilustrado na Figura 21.

Figura 21 - Tela do terminal de comando para acesso remoto do Raspberry Pi.

```

pi@raspberrypi: ~
Linux raspberrypi 4.19.75-v7+ #1270 SMP Tue Sep 24 18:45:11 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Nov 18 17:38:39 2019 from 10.22.10.6

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $ vncserver

```

Fonte: Autoria Própria (2019).

Após enviar a instrução, o terminal de comando retorna o endereço de IP (*Internet Protocol*) mostrado na Figura 22, que representa a identificação do Raspberry Pi na rede.

Figura 22 - Tela do terminal de comando com o IP do Raspberry Pi.

```

pi@raspberrypi: ~
RealVNC e VNC são marcas registradas da RealVNC Ltd que estão protegidas por
registros de marca e/ou pedidos pendentes de marca registrada na União
Europeia, nos Estados Unidos da América e em outras jurisdições.
Marcas protegidas pela patente 2481870 no Reino Unido, 8760366 nos EUA e
2652951 na União Europeia.
Consulte https://www.realvnc.com para obter informações sobre o VNC.
Para ver reconhecimentos de terceiros, consulte:
https://www.realvnc.com/docs/6/foss.html
OS: Raspbian GNU/Linux 10, Linux 4.19.75, armv7l

Em algumas distribuições (particularmente no Red Hat), você pode ter uma
experiência melhor executando o vncserver-virtual juntamente com o servidor Xorg
do sistema, em vez da versão antiga integrada ao Xvnc. Outros ambientes e
aplicativos de desktop provavelmente serão compatíveis. Para obter mais
informações sobre essa implementação alternativa, consulte:
https://www.realvnc.com/doclink/kb-546

Executando aplicativos em /etc/vnc/xstartup

Frase do VNC Server: "Radical reverse senior. Avenue voodoo learn."
assinatura: d7-77-98-67-6e-5a-86-e0

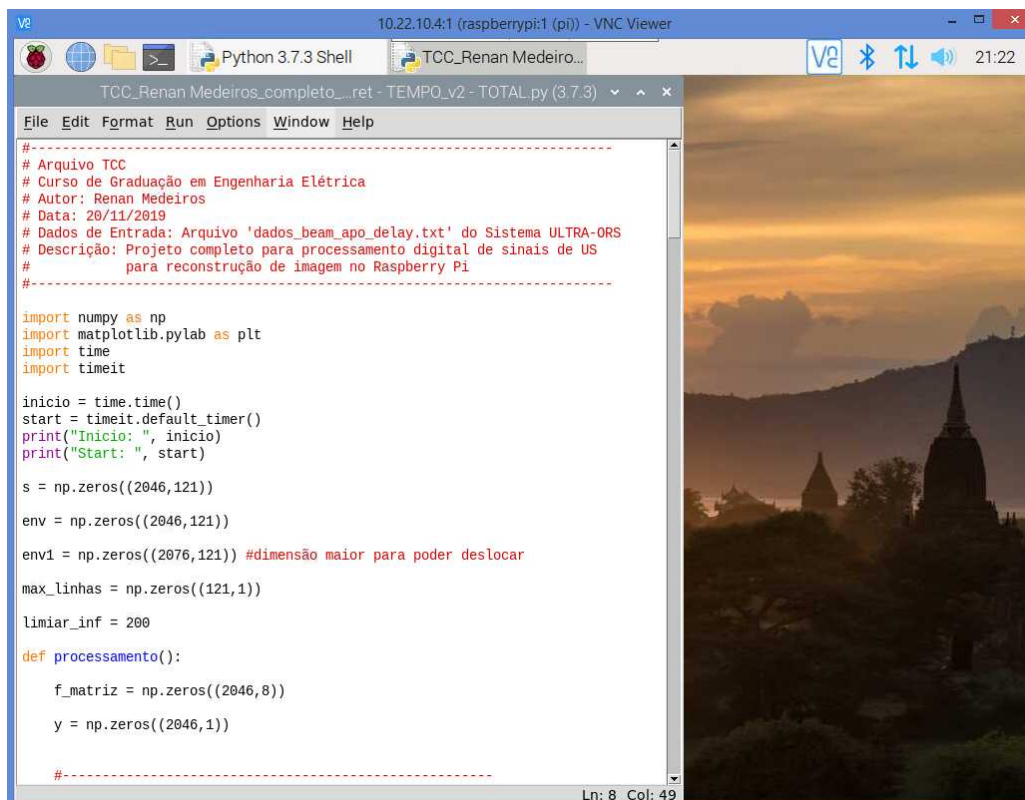
O arquivo de log é /home/pi/.vnc/raspberrypi:1.log
O novo desktop é raspberrypi:1 (10.22.10.4:1)
pi@raspberrypi:~ $

```

Fonte: Autoria Própria (2019).

A partir da obtenção do número do IP, basta inseri-lo no VNC *Viewer* e realizar a autenticação da conexão colocando o nome de usuário e senha do Raspberry Pi. Como resultado, têm-se o acesso à interface gráfica do Raspberry Pi. Na Figura 23 é apresentada a tela da interface gráfica do Raspberry Pi com aplicativo Python 3 (IDLE), versão 3.7.3. O código completo que foi implementado é apresentado no Apêndice A.

Figura 23 - Tela inicial da interface gráfica do Raspberry Pi com aplicativo Python 3 (IDLE).



```

v2 10.22.10.4:1 (raspberrypi:1 (pi)) - VNC Viewer
Python 3.7.3 Shell TCC_Renan Medeiros...
TCC_Renan Medeiros_completo...ret - TEMPO_v2 - TOTAL.py (3.7.3)
File Edit Format Run Options Window Help
#-----
# Arquivo TCC
# Curso de Graduação em Engenharia Elétrica
# Autor: Renan Medeiros
# Data: 20/11/2019
# Dados de Entrada: Arquivo 'dados_beam_apo_delay.txt' do Sistema ULTRA-ORS
# Descrição: Projeto completo para processamento digital de sinais de US
#             para reconstrução de imagem no Raspberry Pi
#-----

import numpy as np
import matplotlib.pyplot as plt
import time
import timeit

inicio = time.time()
start = timeit.default_timer()
print("Inicio: ", inicio)
print("Start: ", start)

s = np.zeros((2046,121))
env = np.zeros((2046,121))
env1 = np.zeros((2076,121)) #dimensão maior para poder deslocar
max_linhas = np.zeros((121,1))
limiar_inf = 200

def processamento():
    f_matriz = np.zeros((2046,8))
    y = np.zeros((2046,1))

#-----
Ln: 8 Col: 49

```

Fonte: Autoria Própria (2019).

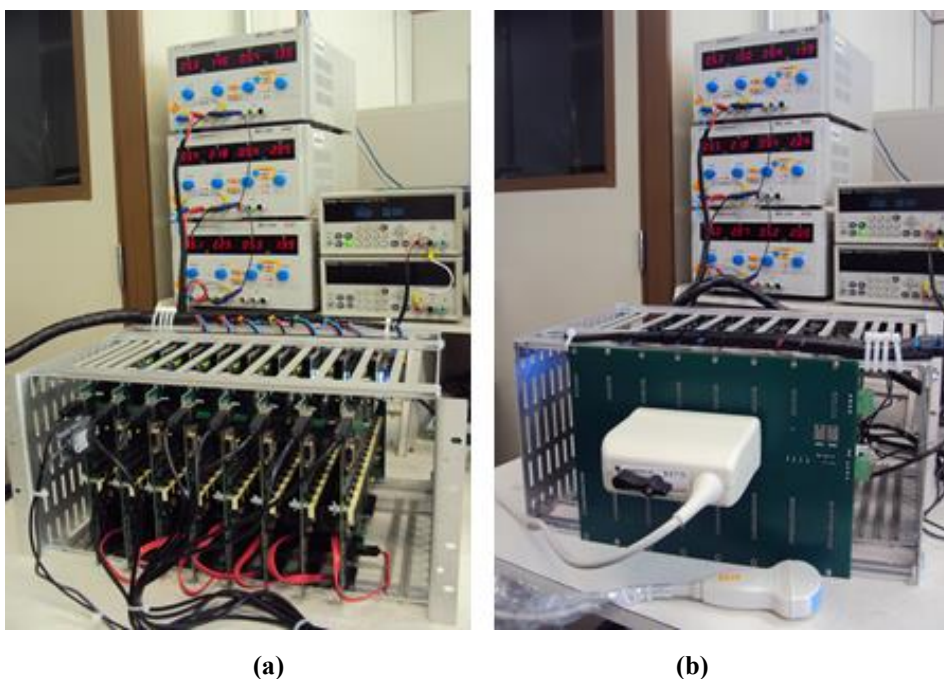
Os dados de entrada utilizados para a avaliação das etapas de processamento digital de sinais implementadas neste trabalho foram gerados por duas maneiras: simulação no Matlab/Simulink e pela plataforma de pesquisa ULTRA-ORS.

3.2 PLATAFORMA ULTRA-ORS

A plataforma ULTRA-ORS é o protótipo de um sistema de transmissão e aquisição de sinais de US desenvolvido na UTFPR. Na Figura 24(a) é mostrado a visão frontal da plataforma conectada com as fontes de alimentação, e na Figura 24(b) é ilustrado a visão traseira da plataforma conectada a um transdutor de US.

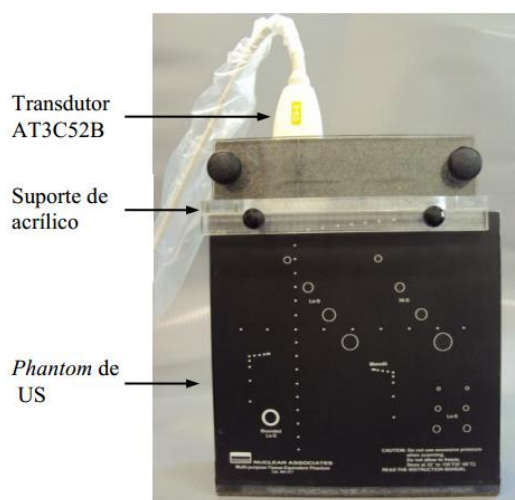
Neste trabalho, os dados brutos de US foram capturados e digitalizados com uma frequência de 40 MHz para um computador pelo sistema ULTRA-ORS adquiridos através de um *phantom* mimetizador de tecidos biológicos modelo 84-317 e um transdutor multielemento convexo AT3C52B, conforme a configuração apresentada na Figura 25.

Figura 24 - (a) Vista frontal e (b) vista traseira da Plataforma ULTRA-ORS.



Fonte: Assef (2013).

Figura 25 - Phantom de US com suporte para o transdutor utilizado na obtenção dos dados de US com a plataforma ULTRA-ORS.



Fonte: Assef (2013).

O transdutor utilizado contém 128 elementos e opera com frequência de 3,2 MHz no método *beamforming* com 8 elementos ativos em cada abertura. O procedimento para aquisição dos dados é semelhante ao que foi exemplificado na Figura 3, resultando em 121 *scanlines*, conforme a Equação (2). Para cada um dos 8 sinais que compõe a *scanline* foram obtidas 2046 amostras com resolução de 12 *bits*.

3.3 SIMULINK - MATLAB

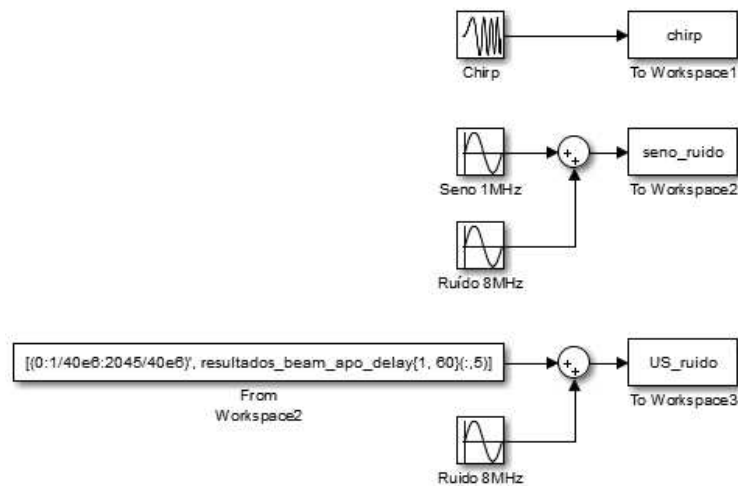
O Simulink é um ambiente de desenvolvimento, avaliação e simulação, integrado ao Matlab, que permite o projeto de sistemas a partir de uma estrutura de diagrama de blocos, com possibilidade de geração automática de códigos (THE MATHWORKS INC., EUA).

Neste trabalho, o Simulink foi utilizado para gerar os sinais (Figura 26) utilizados na verificação do funcionamento do filtro FIR passa-baixa projetado. Foram obtidos três tipos de sinais diferentes, todos amostrados com uma frequência de 40MHz e 2046 amostras no total.

O primeiro tipo de sinal de teste utilizado para a avaliação da etapa de filtragem digital foi uma onda senoidal com frequência de 1 MHz, que foi adicionada de outra onda senoidal com frequência de 8 MHz para simular um ruído. O respectivo bloco no Simulink é o “Sine Wave”. Em sequência, obteve-se um sinal do tipo *chirp*, com a frequência variando de 1 MHz até 10 MHz, através do bloco “Chirp Signal”. Por último, foi utilizado um sinal típico de US obtido a partir da plataforma ULTRA-ORS, conforme a configuração da Figura 25. No exemplo da Figura 26, utilizou-se a linha 5 da *scanline* 60. Assim como no sinal senoidal, foi adicionado ao sinal de US um sinal senoidal com frequência de 8 MHz com a finalidade de reproduzir um ruído. Em todos os casos, os sinais resultantes são exportados primeiramente para a área de trabalho do Matlab e posteriormente para o algoritmo em Python.

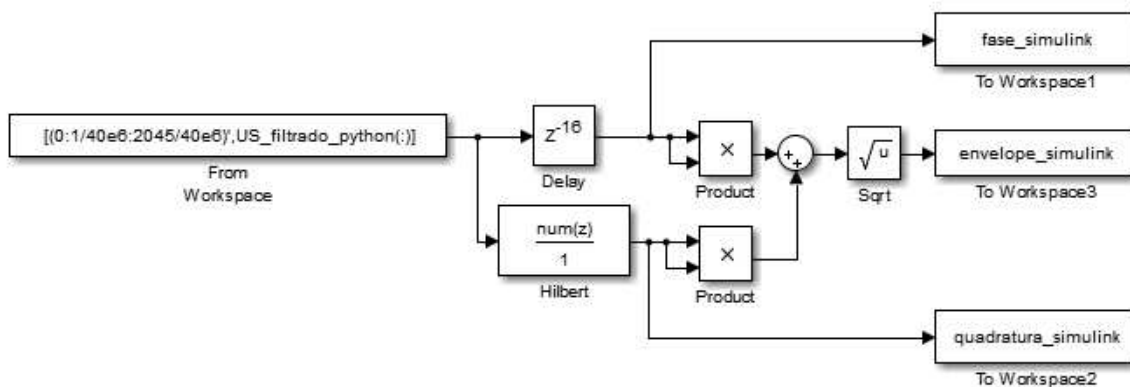
Na Figura 27 é apresentado o projeto da etapa de demodulação e detecção da envoltória no Simulink. Esse projeto representa o mesmo diagrama de blocos apresentado na Figura 14. Os sinais resultantes são exportados para o Matlab com o intuito de posterior visualização e comparação com o resultado do processamento embarcado no Raspberry Pi.

Figura 26 - Sinais de teste gerados a partir do Simulink.



Fonte: Autoria Própria (2019).

Figura 27 - Detecção de envoltória a partir do Simulink.

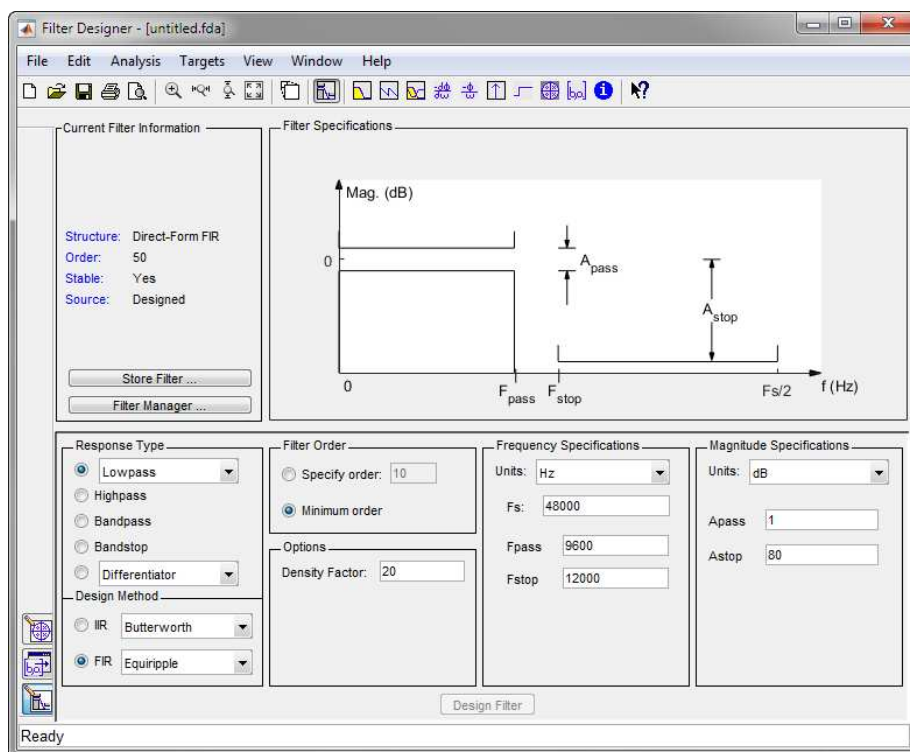


Fonte: Autoria Própria (2019).

3.4 FILTER DESIGNER - MATLAB

O *Filter Designer* é uma ferramenta para projeto de filtros digitais que está integrada ao Matlab (THE MATHWORKS INC., EUA). A ferramenta possibilita a obtenção dos coeficientes de um filtro, resposta de magnitude e fase, resposta ao impulso, entre outros dados, a partir da inclusão das especificações de um filtro. No *Filter Designer*, conforme ilustrado na Figura 28, é possível definir o tipo de resposta desejada, o método de projeto, a ordem do filtro, a frequência de amostragem, faixas de frequência, frequência de rejeição, atenuação da banda de rejeição, entre outras opções (JI-FENG et al., 2015). A ferramenta foi utilizada nos projetos de filtragem digital e demodulação.

Figura 28 - Tela inicial de exibição do Filter Designer.



Fonte: The Mathworks Inc. (2019).

3.5 PROJETO DO FILTRO DIGITAL

O projeto do filtro digital foi desenvolvido na ferramenta *Filter Designer* do Matlab. As especificações utilizadas são apresentadas na Tabela 1, levando em conta que os dados de US são digitalizados com uma frequência de amostragem de 40 MHz e frequência central do transdutor é de 3,2 MHz.

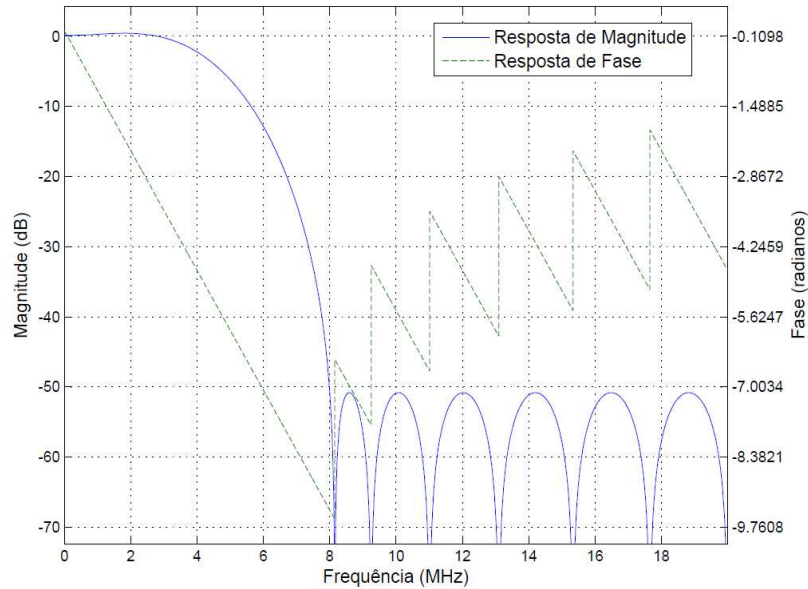
Tabela 1 - Especificações do Filtro Digital.

Parâmetro	Especificação
Frequência de amostragem	40 MHz
Método do projeto	FIR - Generalized Equiripple
Tipo de resposta	Passa-baixa
Frequência da banda passante	3,2 MHz
Frequência da banda de rejeição	8,0 MHz
Atenuação na banda passante	-1 dB
Atenuação na banda de rejeição	-50 dB

Fonte: Autoria Própria (2019).

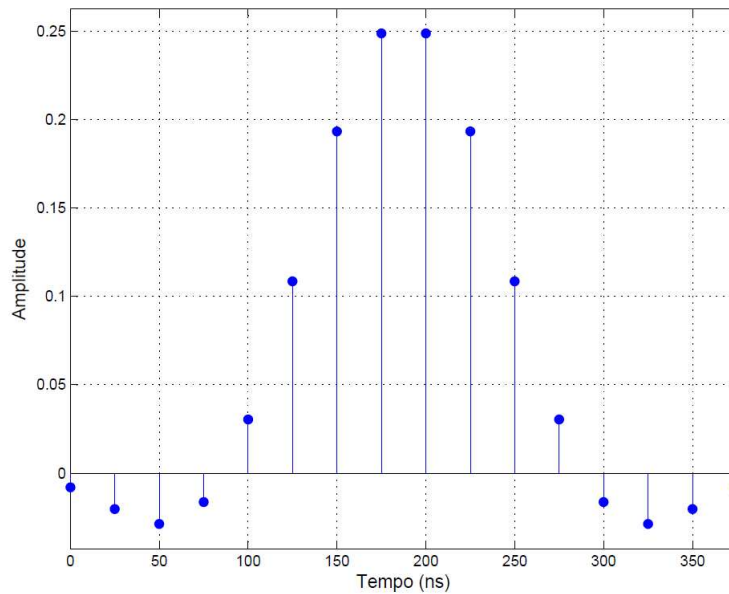
A partir dessas especificações, determinou-se um filtro com a menor ordem. As respostas de magnitude e fase são ilustradas na Figura 29, e a resposta ao impulso é mostrado na Figura 30.

Figura 29 - Respostas em frequência da Magnitude e Fase do filtro FIR passa-baixa projetado.



Fonte: Autoria Própria (2019).

Figura 30 - Resposta ao Impulso do Filtro FIR passa-baixa projetado.



Fonte: Autoria Própria (2019).

Com o auxílio da ferramenta, calculou-se os coeficientes do filtro. Como resultado, obteve-se 16 coeficientes, com os valores descritos na Tabela 2.

Tabela 2 - Coeficientes do Filtro FIR passa-baixa projetado.

Nº do coeficiente	Valor do coeficiente	Nº do coeficiente	Valor do coeficiente
1	-0,00805246535535765	9	0,24892007978955700
2	-0,02031883006153170	10	0,19336131713264000
3	-0,02880053432836590	11	0,10853961742278900
4	-0,01643518329923500	12	0,03036968142921930
5	0,03036968142921930	13	-0,01643518329923500
6	0,10853961742278900	14	-0,02880053432836590
7	0,19336131713264000	15	-0,02031883006153170
8	0,24892007978955700	16	-0,00805246535535765

Fonte: Autoria Própria (2019).

3.6 PROJETO DE DEMODULAÇÃO E DETECÇÃO DA ENVOLTÓRIA

Para realizar a etapa de demodulação e detecção da envoltória, projetou-se um filtro FIR com os coeficientes obtidos através da Transformada de Hilbert, desenvolvido na ferramenta *Filter Designer* do Matlab. As principais especificações utilizadas são apresentadas Tabela 3.

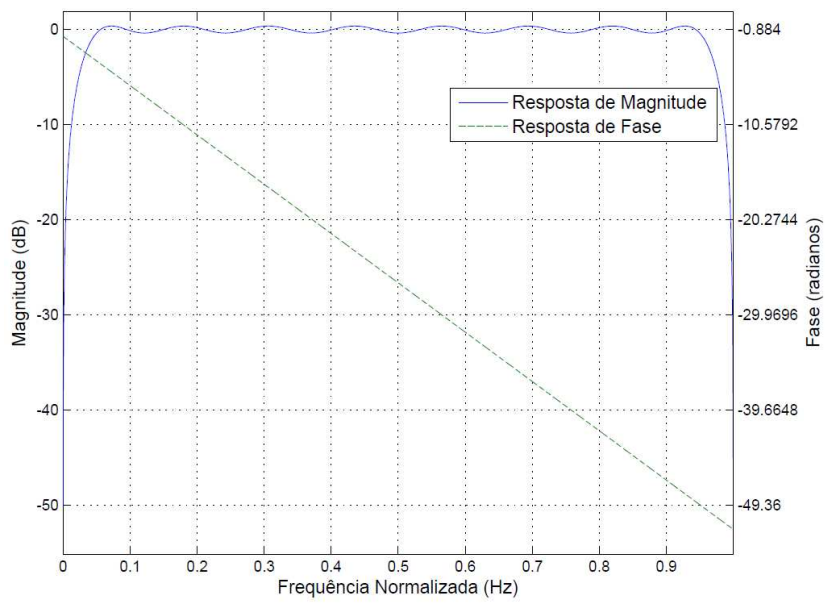
Tabela 3 - Especificações do Filtro FIR Hilbert *Transformer*.

Parâmetro	Especificação
Método do projeto	FIR – Equiripple
Tipo de resposta	Hilbert <i>Transformer</i>
Ordem do filtro	32

Fonte: Autoria Própria (2019).

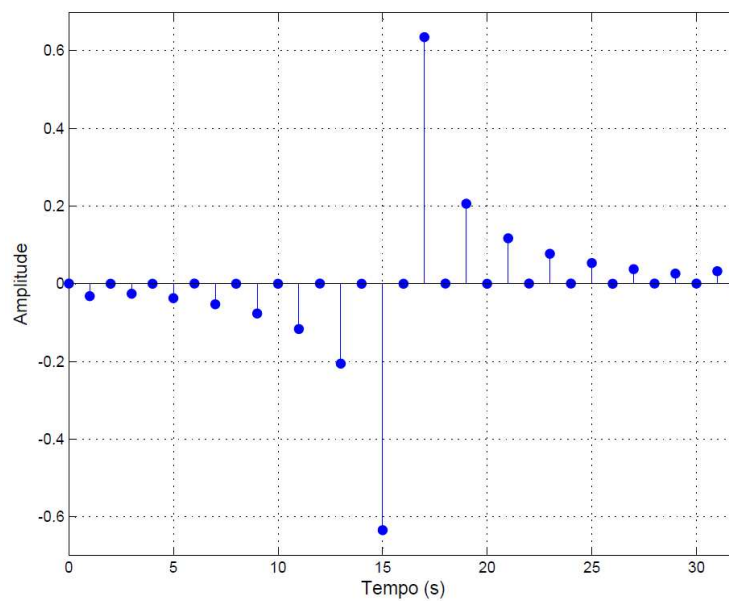
Nas demais especificações necessárias foram utilizados os valores padrão da ferramenta. Assim, obteve-se as respostas de magnitude e fase, conforme são ilustradas na Figura 31, e a resposta ao impulso, conforme é mostrado na Figura 32.

Figura 31 - Respostas em frequência da Magnitude e Fase do filtro FIR Hilbert *Transformer* projetado.



Fonte: Autoria Própria (2019).

Figura 32 - Resposta ao Impulso do Filtro FIR Hilbert *Transformer* projetado.



Fonte: Autoria Própria (2019).

A partir da ferramenta, obteve-se como resultado 33 coeficientes, com os valores descritos na Tabela 4.

Tabela 4 - Coeficientes do Filtro FIR Hilbert *Transformer* projetado.

Nº do coeficiente	Valor do coeficiente	Nº do coeficiente	Valor do coeficiente
0	0	17	0,63447262939206328
1	-0,031913604561168388	18	0
2	0	19	0,2058058100572413
3	-0,026047835887445595	20	0
4	0	21	0,11685382676685013
5	-0,037328402135740679	22	0
6	0	23	0,076699537501351389
7	-0,053110745993753178	24	0
8	0	25	0,053110745993753178
9	-0,076699537501351389	26	0
10	0	27	0,037328402135740679
11	-0,11685382676685013	28	0
12	0	29	0,026047835887445595
13	-0,2058058100572413	30	0
14	0	31	0,031913604561168388
15	-0,63447262939206328	32	0
16	0	-	-

Fonte: Autoria Própria (2019).

3.7 FUNÇÃO DE CUSTO NRMSE

A avaliação quantitativa dos resultados obtidos no trabalho para o processamento proposto foi realizada a partir da função de custo da raiz quadrada do erro quadrático médio normalizado - NRMSE (*Normalized Root Mean Squared Error*). Essa métrica é utilizada por diferentes autores da comunidade científica para avaliar a comparação entre dois modelos distintos (FERREIRA, 2017; ASSEF et al., 2019).

Segundo Kadiyala et al. (2015), o modelo é considerado excelente, bom, justo ou pobre baseado respectivamente nos seguintes valores de NRMSE: <10%, 10–20%, 20–30%, ou >30%. O cálculo da função de custo NRMSE é realizado conforme a Equação (9):

$$\text{NRMSE} = \sqrt{\frac{\sum_{n=1}^M |R(n) - S(n)|^2}{\sum_{n=1}^M |S(n) - \bar{S}|^2}} * 100, \quad (9)$$

no qual n é o índice da amostra, M é o número total de amostras, $R(n)$ representa a amostra do modelo utilizado de referência e $S(n)$ representa a amostra do modelo que se deseja comparar e \bar{S} é a média das amostras do modelo que se deseja comparar.

3.8 FUNÇÃO DE CUSTO NRSS

Assim como a função NRMSE, a função de custo da soma residual dos quadrados normalizado - NRSS (*Normalized Residual Sum of Squares*) foi usada para avaliar quantitativamente os resultados obtidos neste trabalho. Essa métrica também é aplicada por diferentes autores da comunidade científica para avaliar a comparação entre dois modelos distintos. Como exemplo, pode-se citar os trabalhos de Levesque e Sawan (2009) e de Zhou e Zheng (2015). O cálculo da função de custo NRSS é feito conforme a Equação (10):

$$\text{NRSS} = \frac{\sum_{n=1}^M |R(n) - S(n)|^2}{\sum_{n=1}^M |S(n)|^2}, \quad (10)$$

no qual n é o índice da amostra, M é o número total de amostras, $R(n)$ representa a amostra do modelo utilizado de referência e $S(n)$ representa a amostra do modelo que se deseja comparar.

4 RESULTADOS

Neste capítulo são apresentados os resultados qualitativos e quantitativos obtidos através de simulações e de forma experimental para verificação e validação de cada etapa que compõe o processamento digital proposto, além do resultado final do processamento digital para reconstrução de imagem por US em Modo B.

4.1 VALIDAÇÃO DO FILTRO DIGITAL PROJETADO

Para verificar o funcionamento do filtro projetado, os valores dos coeficientes foram exportados para um algoritmo desenvolvido em Python na plataforma Raspberry Pi. No algoritmo são implementados os somatórios e as multiplicações necessárias para descrever a estrutura do filtro FIR. O código

Na Figura 33 a Figura 35 são apresentados os resultados antes e após a aplicação do algoritmo de filtragem digital. Realizou-se a filtragem de três tipos de formas de onda diferentes e a obtenção da *Fast Fourier Transform* (FFT) de cada sinal para verificar o espectro no domínio da frequência.

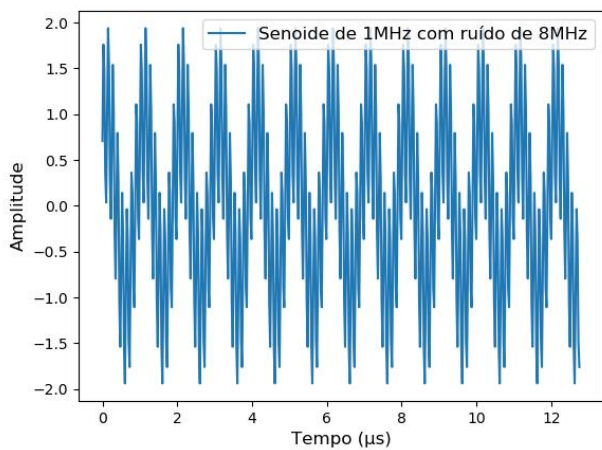
Na Figura 33(a) e Figura 33(c) são mostrados o sinal senoidal de 1 MHz adicionado de um ruído de 8 MHz e a respectiva FFT. Na Figura 33(b) e Figura 33(d) são indicados o resultado da filtragem digital e FFT, respectivamente.

Na Figura 34(a) e Figura 34(c) são ilustrados o sinal do tipo *chirp* com frequência de 1 MHz a 10 MHz e a respectiva FFT. Na Figura 34(b) e Figura 34(d) são apresentados o resultado da filtragem digital do sinal *chirp* e a respectiva FFT.

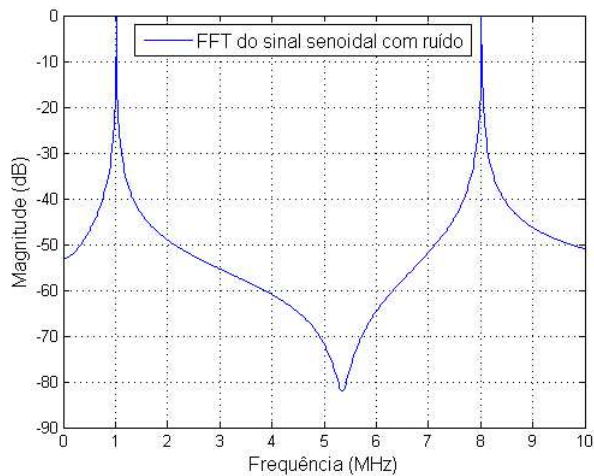
Por último, na Figura 35(a) e Figura 35(c) são apresentados o sinal de US adicionado de um ruído de 8 MHz e a respectiva FFT. Na Figura 35(b) e Figura 35(d) são mostrados o resultado da filtragem digital e FFT, respectivamente.

Cada sinal possui 2046 amostras com frequência de amostragem de 40 MHz. No entanto, para melhor visualização, o sinal senoidal foi limitado em 510 amostras e o sinal de US limitado em 1023 amostras.

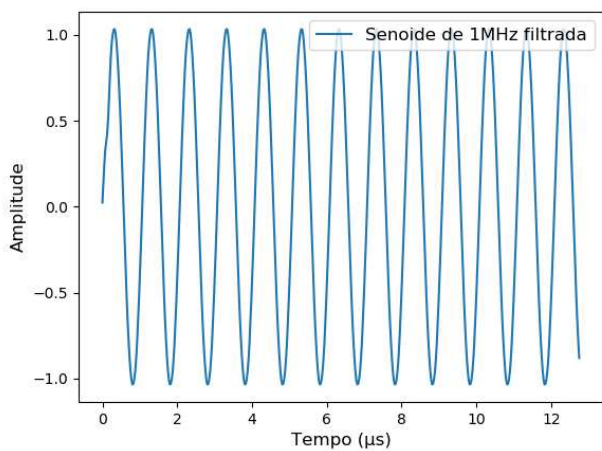
Figura 33 - (a) Sinal Senoidal com ruído e (b) filtrado. (c) FFT do sinal Senoidal com ruído e do (d) sinal senoidal filtrado.



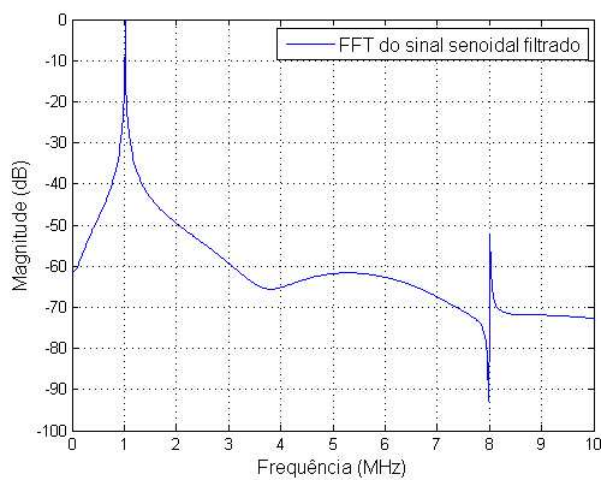
(a)



(c)



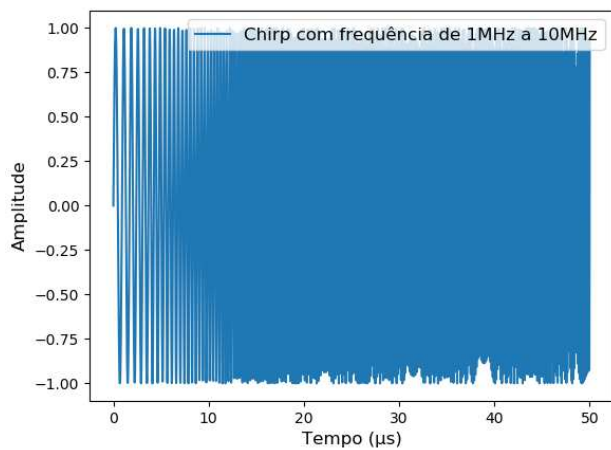
(b)



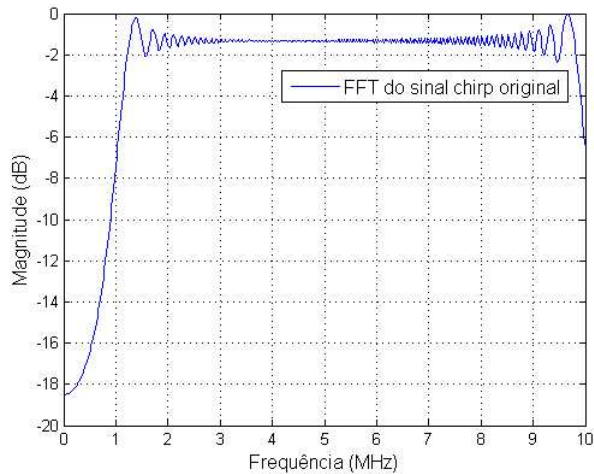
(d)

Fonte: Autoria Própria (2019).

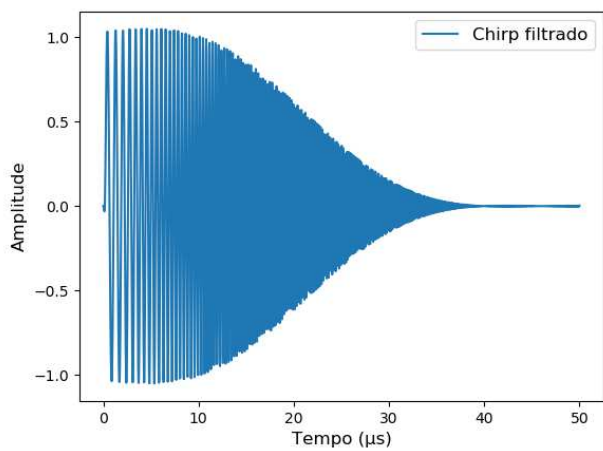
Figura 34 - (a) Sinal do tipo Chirp original e (b) filtrado. (c) FFT do sinal Chirp original e do (d) sinal Chirp filtrado.



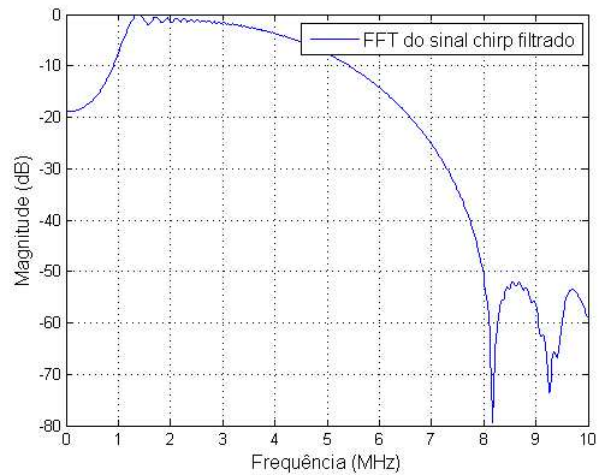
(a)



(c)



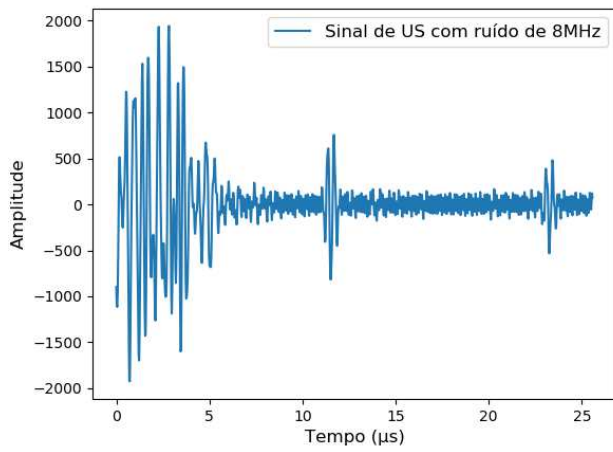
(b)



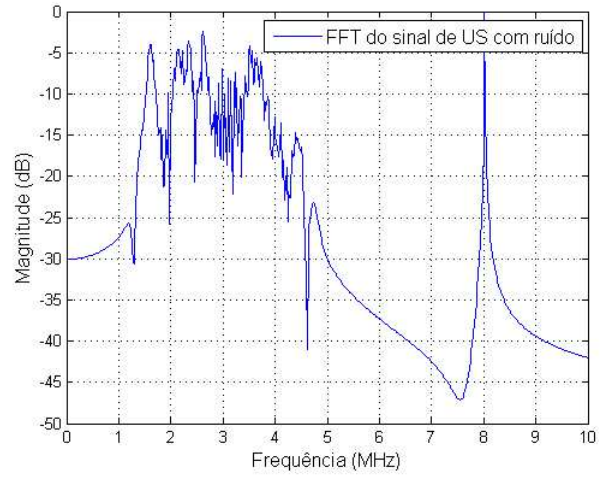
(d)

Fonte: Autoria Própria (2019).

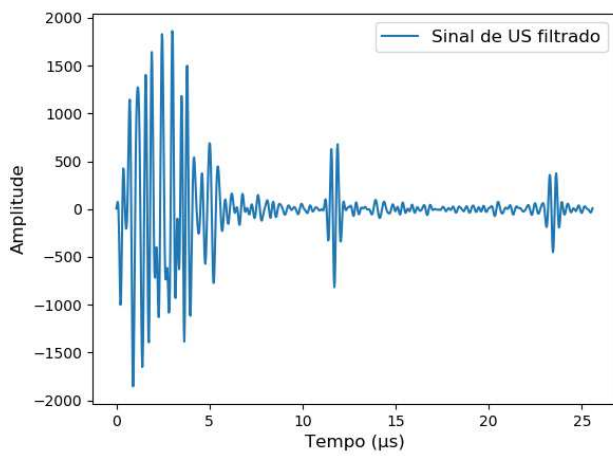
Figura 35 - (a) Sinal de US com ruído e (b) filtrado. (c) FFT do sinal de US com ruído e do (d) sinal de US filtrado.



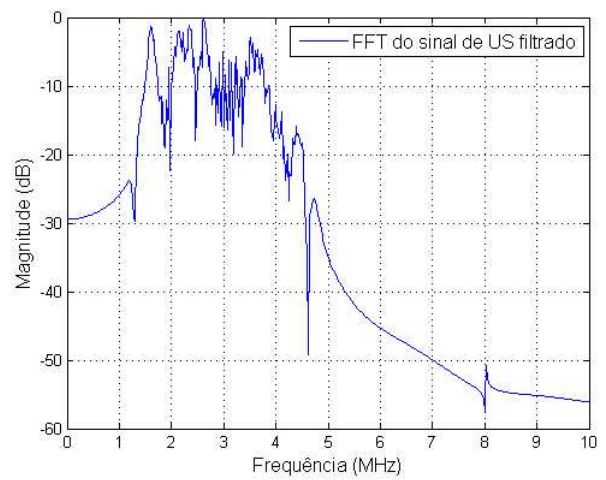
(a)



(c)



(b)



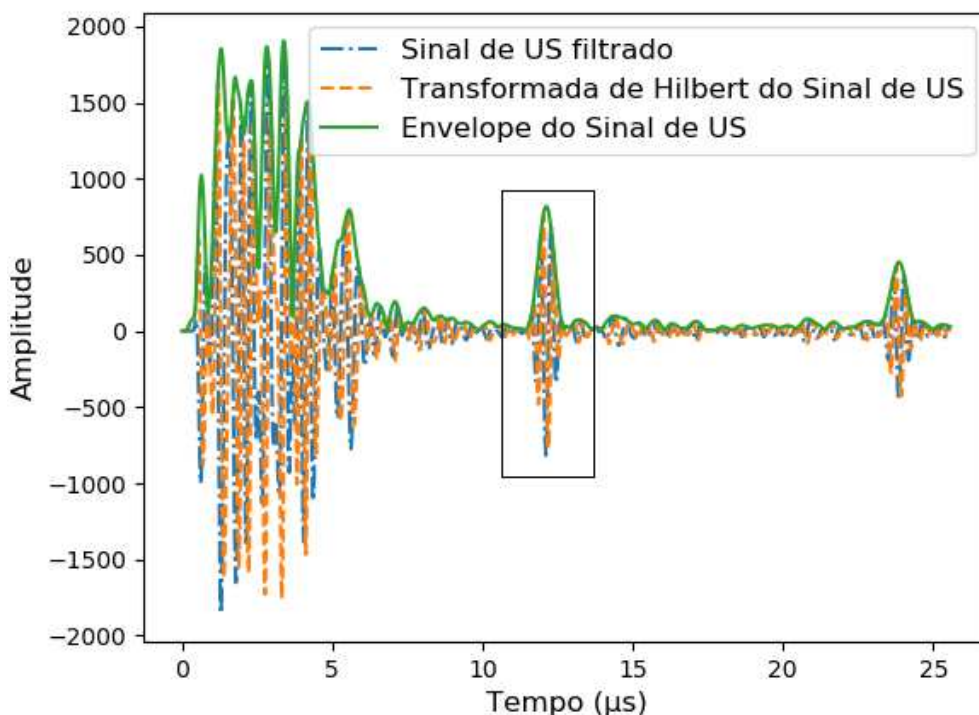
(d)

Fonte: Autoria Própria (2019).

4.2 VALIDAÇÃO DA DEMODULAÇÃO E DETECÇÃO DA ENVOLTÓRIA

Para verificar o funcionamento da detecção de envoltória, os valores dos coeficientes foram exportados para o mesmo algoritmo desenvolvido no Raspberry Pi em Python do filtro digital, dando continuidade ao processamento. No algoritmo, implementou-se a estrutura da Transformada de Hilbert baseada no filtro FIR e o equacionamento matemático necessário. Obteve-se o resultado conforme mostrado na Figura 36, na qual são ilustrados o sinal de US filtrado, a Transformada de Hilbert do sinal de US e o Envelope resultante do processamento.

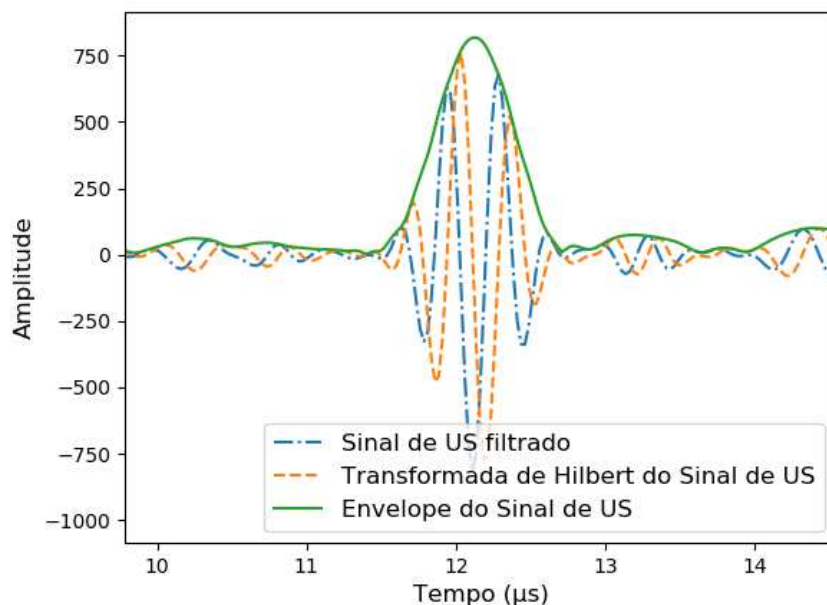
Figura 36 - Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Raspberry Pi/Python.



Fonte: Autoria Própria (2019).

Com o intuito de melhorar a visualização, realizou-se uma ampliação da região em destaque da Figura 36. O resultado é ilustrado na Figura 37.

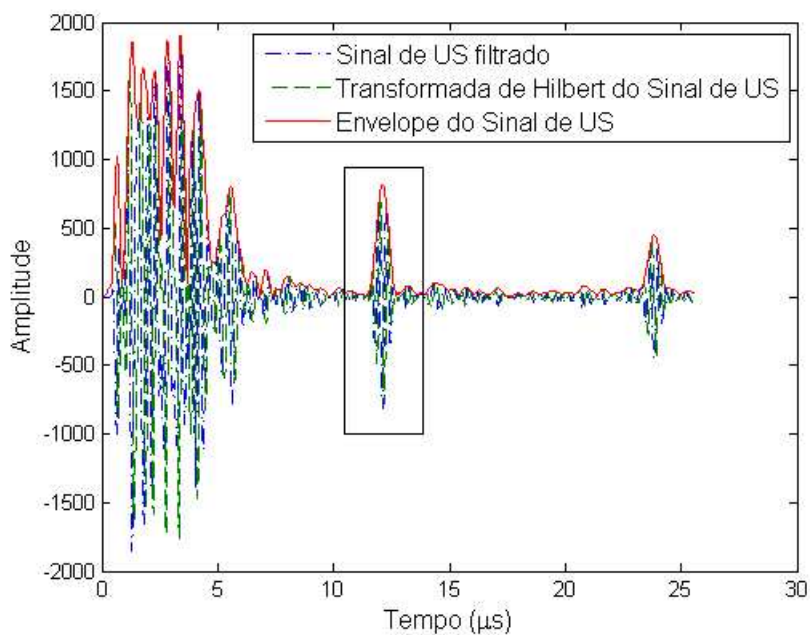
Figura 37 - Destaque da Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Raspberry Pi/Python.



Fonte: Autoria Própria (2019).

Para obter um parâmetro de comparação, realizou-se a detecção de envoltória do mesmo sinal de US filtrado utilizando o Matlab/Simulink, conforme o modelo da Figura 27. Na Figura 38 é ilustrado o resultado obtido.

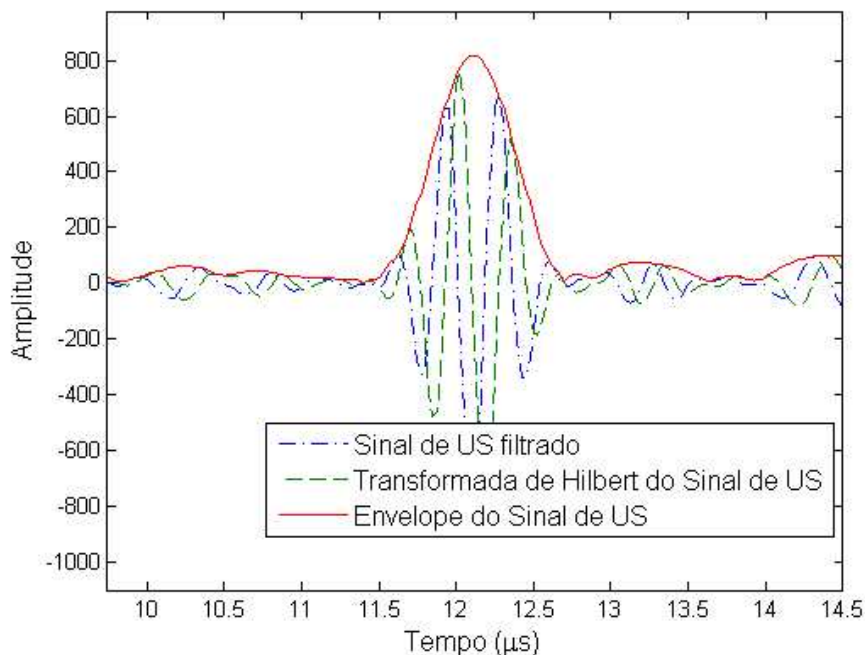
Figura 38 - Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Matlab.



Fonte: Autoria Própria (2019).

Realizando o mesmo procedimento para melhorar a visualização da região em destaque da Figura 38, obtém-se o resultado ilustrado na Figura 39.

Figura 39 - Destaque da Envoltória do Sinal de US obtida a partir da Transformada de Hilbert no Matlab.



Fonte: Autoria Própria (2019).

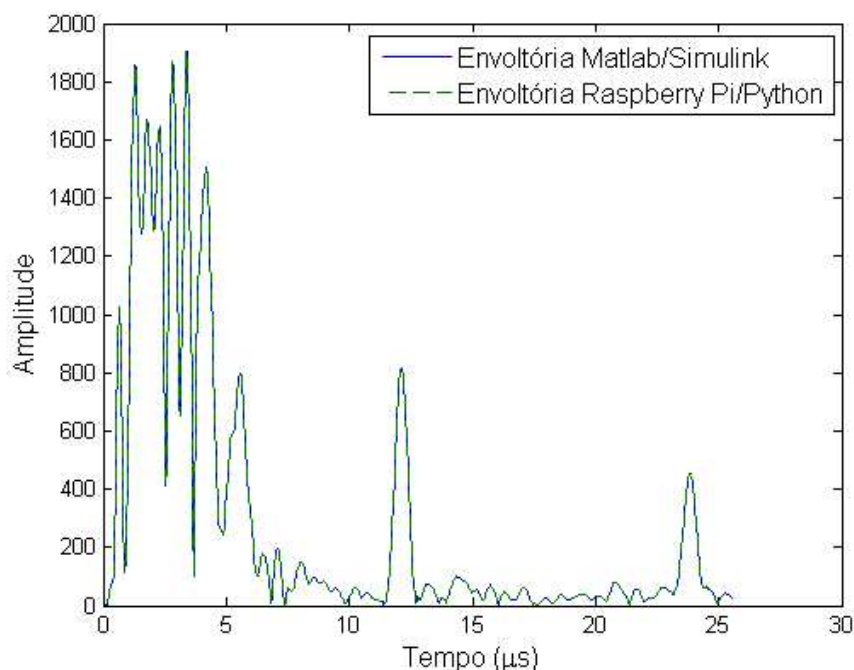
Por fim, foi realizado a plotagem da envoltória obtida no Matlab/Simulink juntamente com a envoltória obtida no Raspberry Pi/Python. O resultado é ilustrado na Figura 40. Também foi realizado o cálculo das funções custo para a etapa de detecção da envoltória. O resultado é apresentado na Tabela 5.

Tabela 5 - Resultado das Funções de Custo para comparação da Detecção de Envoltória.

Função Custo	Valor
NRMSE	1,1499e-09%
NRSS	1,0202e-20

Fonte: Autoria Própria (2019).

Figura 40 - Comparação da Envoltória obtida a partir do Matlab/Simulink com a Envoltória obtida a partir do Raspberry Pi/Python.



Fonte: Autoria Própria (2019).

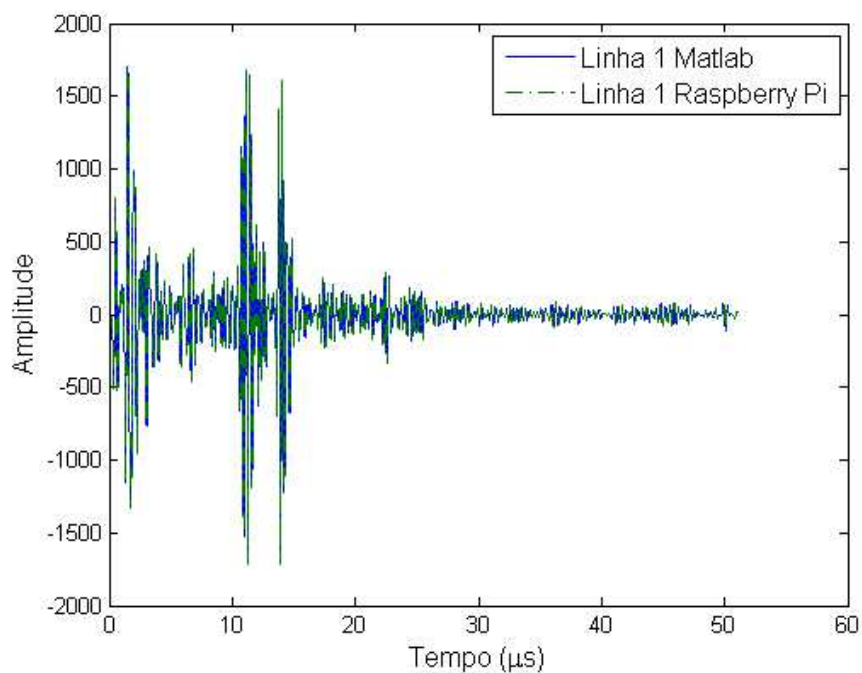
4.3 COMPARAÇÃO DO MODELO DO MATLAB COM O MODELO DO RASPBERRY PI

Após os testes individuais das etapas de filtragem digital e detecção de envoltória, realizou-se o processamento completo dos dados de US para geração de imagem. Comparou-se cada etapa do processamento completo realizado no Raspberry Pi com a respectiva etapa do processamento realizado no Matlab.

Após os dados brutos de RF serem importados para o programa em Python desenvolvido neste trabalho, verificou-se a primeira etapa do processamento, representada pela filtragem digital.

De acordo com o método *beamforming*, as *scanlines* são resultados do somatório de 8 linhas de dados de US. Cada linha é filtrada separadamente para o posterior somatório coerente e formação da *scanline*. A Figura 41 ilustra o resultado da comparação entre os dados da linha 1 da *scanline* 1 após serem filtrados no Raspberry Pi e após serem filtrados no Matlab.

Figura 41 - Comparação da Linha 1 da *Scanline 1* obtida a partir do Raspberry Pi com a obtida a partir do Matlab.



Fonte: Autoria Própria (2019).

O resultado do cálculo das funções de custo para a etapa de filtragem digital é apresentado na Tabela 6.

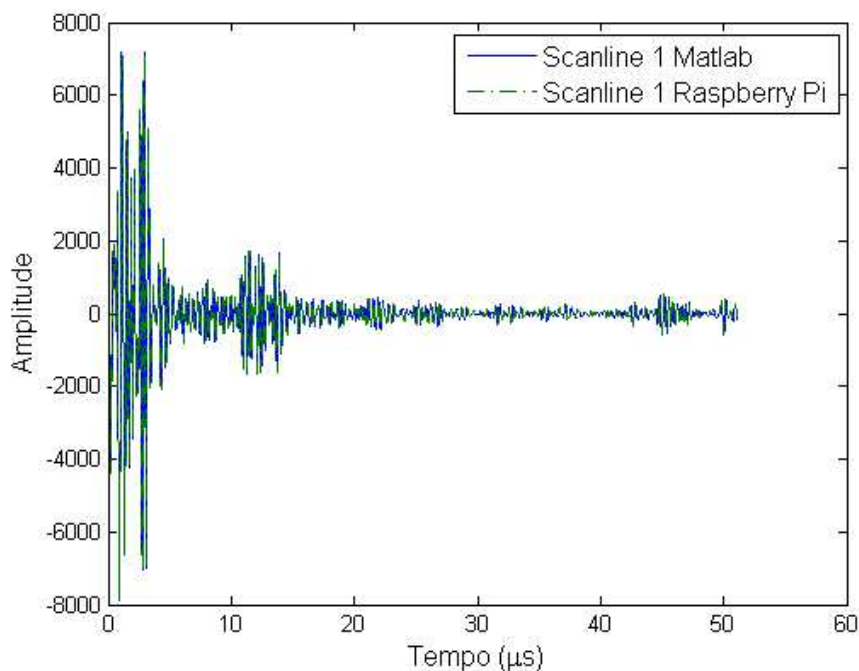
Tabela 6 - Resultado das Funções de Custo para comparação da Filtragem Digital entre o Matlab e o Raspberry Pi.

Função Custo	Valor
NRMSE	5,2696e-10%
NRSS	2,7768e-21

Fonte: Autoria Própria (2019).

Na Figura 42 é ilustrado o resultado da comparação entre a *scanline 1* obtida a partir do Raspberry Pi com a *scanline 1* obtida a partir do Matlab.

Figura 42 - Comparação da *Scanline 1* obtida a partir do Matlab e do Raspberry Pi.



Fonte: Autoria Própria (2019).

O resultado do cálculo das funções custo para a comparação entre a *scanline 1* obtida a partir do Raspberry Pi com a obtida a partir do Matlab é apresentado na Tabela 7.

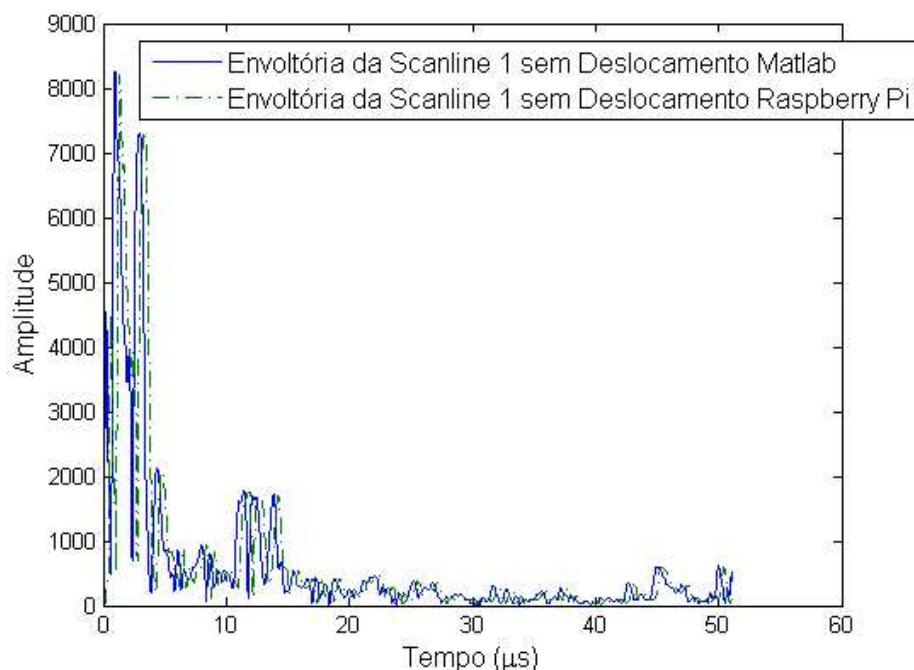
Tabela 7 - Resultado das Funções de Custo para comparação das *Scanlines* obtidas no Matlab e no Raspberry Pi.

Função Custo	Valor
NRMSE	4,8918e-10%
NRSS	2,3929e-21

Fonte: Autoria Própria (2019).

Na Figura 43 é ilustrado o resultado da comparação entre a detecção da envoltória da *scanline 1* sem deslocamento de amostras no tempo, obtida a partir do Matlab e do Raspberry Pi. Nota-se que o resultado do Matlab não apresenta atraso de processamento do sinal.

Figura 43 - Comparação da Envoltória da *Scanline 1* sem deslocamento de amostras no tempo, obtida a partir do Matlab e do Raspberry Pi.



Fonte: Autoria Própria (2019).

O resultado do cálculo das funções de custo para comparação entre a detecção da envoltória da *scanline 1* sem deslocamento, obtida a partir do Matlab e do Raspberry Pi, é apresentado na Tabela 8.

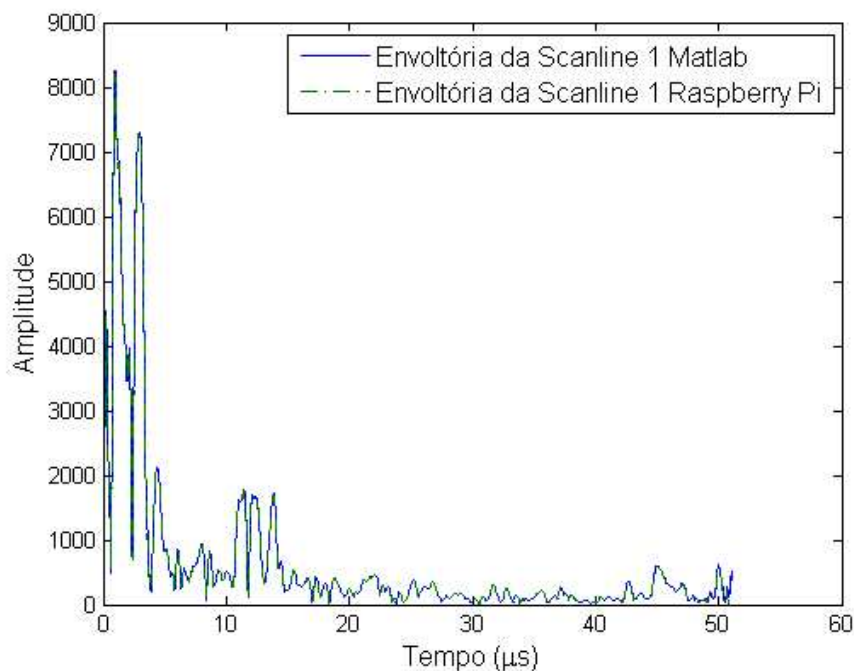
Tabela 8 - Resultado das Funções de Custo para comparação da detecção da envoltória sem deslocamento a partir do Matlab e do Raspberry Pi.

Função Custo	Valor
NRMSE	74,1744%
NRSS	44,3645

Fonte: Autoria Própria (2019).

Na Figura 44 são mostradas as envoltórias da *scanline 1* obtidas a partir do Matlab e do Raspberry Pi para fins de comparação qualitativa.

Figura 44 - Comparação da Envoltória da Scanline 1 obtida a partir do Matlab e do Raspberry Pi.



Fonte: Autoria Própria (2019).

Assim, o resultado do cálculo das funções de custo para comparação entre a detecção da envoltória da *scanline* 1 obtida a partir do Matlab e do Raspberry Pi tiveram os valores reduzidos, conforme é apresentado na Tabela 9.

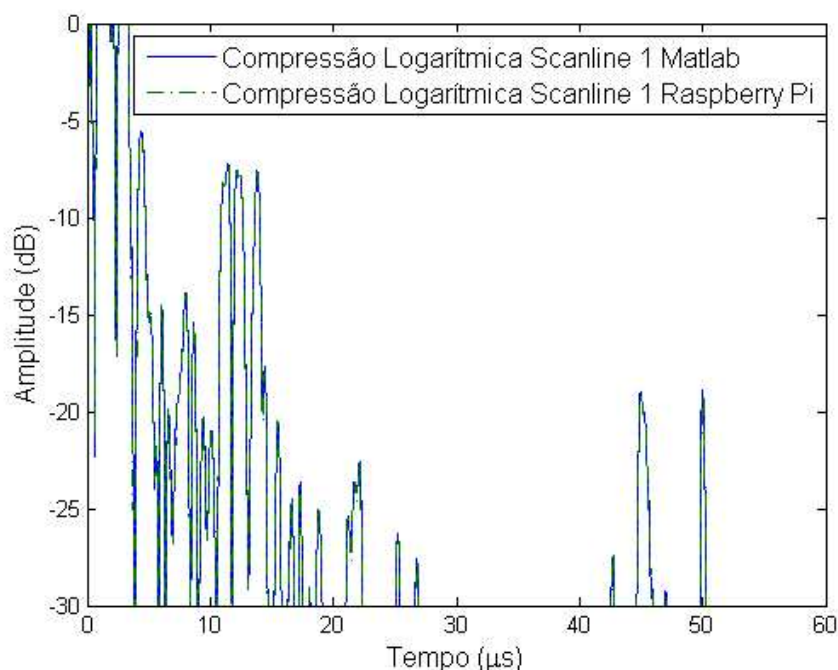
Tabela 9 - Resultado das Funções de Custo para comparação da detecção da envoltória.

Função Custo	Valor
NRMSE	3,0839%
NRSS	0,0767

Fonte: Autoria Própria (2019).

Na Figura 45 é ilustrado o resultado da comparação entre a compressão logarítmica da *scanline* 1 obtida a partir do Matlab e do Raspberry Pi. Neste trabalho os resultados foram obtidos com faixa dinâmica de -30 dB.

Figura 45 - Comparação da Compressão Logarítmica da *scanline* 1 obtida a partir do Matlab e do Raspberry Pi com -30 dB de faixa dinâmica.



Fonte: Autoria Própria (2019).

O resultado do cálculo das funções de custo para comparação entre a compressão logarítmica da *scanline* 1 obtida a partir do Matlab e do Raspberry Pi é ilustrado na Tabela 10.

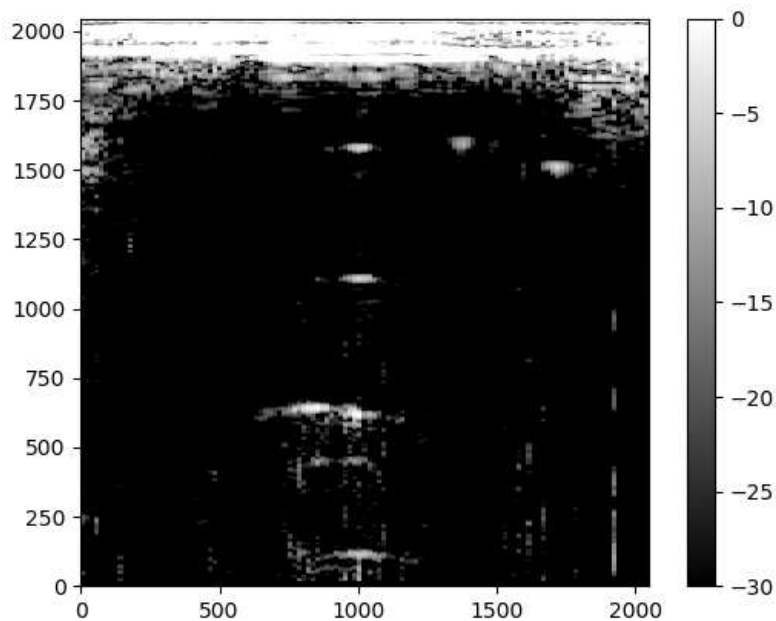
Tabela 10 - Resultado das Funções de Custo para comparação da compressão logarítmica.

Função Custo	Valor
NRMSE	3,5575%
NRSS	0,0138

Fonte: Autoria Própria (2019).

Na Figura 46 é ilustrado o resultado da imagem final de US sem a conversão de varredura obtida a partir do Raspberry Pi. Conforme a barra de cor lateral, a faixa dinâmica é de -30 dB.

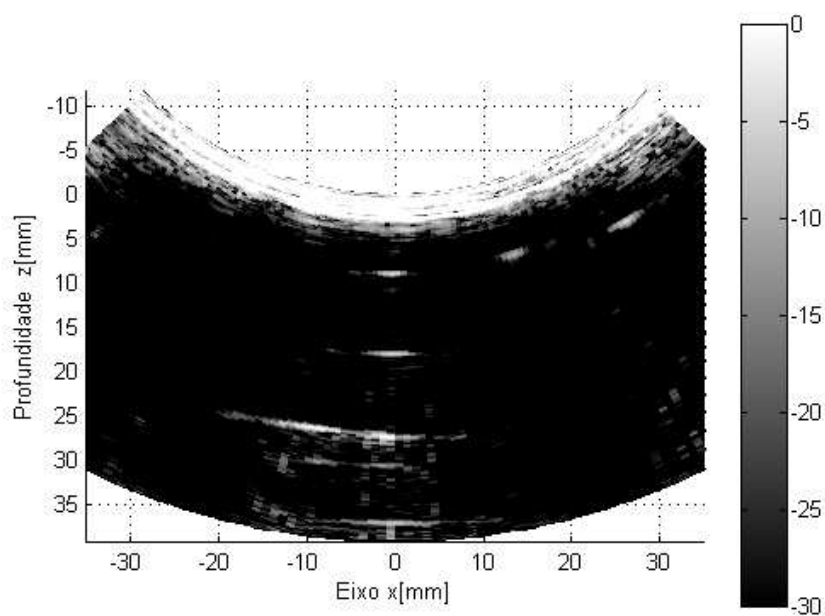
Figura 46 - Imagem Final de US sem a conversão de varredura obtida a partir do Raspberry Pi.



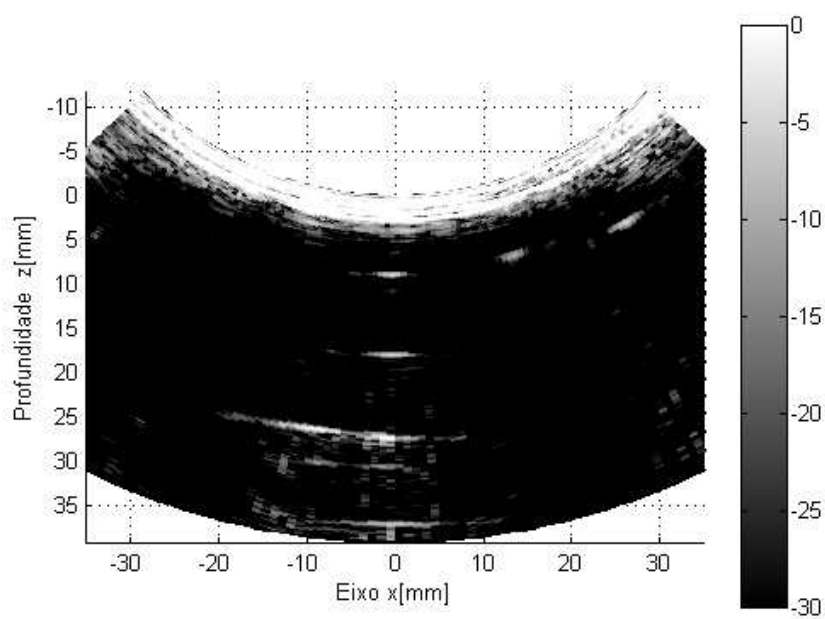
Fonte: Autoria Própria (2019).

Após os dados brutos de US serem processados completamente no Raspberry Pi, os sinais resultantes foram exportados para o Matlab. Posteriormente, realizou-se a etapa de conversão de varredura no Matlab, seguida pela geração da imagem final. Na Figura 47(a) e Figura 47(b) são mostradas as imagens finais de US obtidas a partir do processamento no Matlab e no Raspberry Pi, respectivamente.

Figura 47 - Comparação da imagem final de US obtida a partir do (a) processamento no Matlab e (b) no Raspberry Pi.



(a)



(b)

Fonte: Autoria Própria (2019).

O resultado do cálculo das funções de custo para comparação entre as imagens finais obtidas com o processamento no Matlab e no Raspberry Pi é apresentado na Tabela 11. São apresentados como resultados o valor médio e o desvio padrão para as 121 *scanlines*.

Tabela 11 - Resultado das Funções de Custo para a Imagem Final.

Etapa	Função de Custo	Valor Médio	Desvio Padrão
Somatório Coerente (<i>scanline</i>)	NRMSE	3,4333e-10%	6,9869e-11%
	NRSS	1,2271e-23	5,0142e-24
Detecção de Envoltória	NRMSE	3,5806%	0,7490%
	NRSS	0,0012	5,0203e-04
Compressão Logarítmica	NRMSE	4,3501%	2,3107%
	NRSS	2,2203e-04	2,5724e-04

Fonte: Autoria Própria (2019).

5 DISCUSSÕES E CONCLUSÕES

O algoritmo implementado no Raspberry Pi, que realiza a etapa de filtragem digital, envolve a aplicação de um filtro FIR passa-baixa com 16 coeficientes. O resultado obtido foi satisfatório, apresentando atenuação das componentes com frequência superior à frequência da banda de rejeição. Verificou-se principalmente a atenuação acima de 50 dB da componente de 8 MHz, conforme é apresentado na Figura 33, Figura 34 e Figura 35.

No algoritmo que descreve a etapa de detecção de envoltória baseada no filtro FIR a partir da Transformada de Hilbert, o envelope do sinal de US obtido no Raspberry Pi/Python quando comparado ao envelope obtido no Simulink (Figura 40) obteve o resultado excelente, uma vez que, os valores das funções custo apresentados Tabela 5 confirmam o resultado.

Entretanto, quando a etapa de detecção de envoltória dos dados processados no Raspberry Pi foi comparada com a etapa de detecção de envoltória dos dados processados no Matlab, verificou-se que o envelope resultante do Raspberry Pi estava adiantado em 16 amostras com relação ao envelope do Matlab, resultando em valores elevados para as funções de custo (Tabela 8). O número de amostras atrasadas corresponde exatamente à ordem do filtro FIR baseado na Transformada de Hilbert dividido por 2 ($M/2$). A partir dessa análise, foi inserido o atraso no algoritmo em Python que resulta na Figura 44. Deste modo, os resultados dos valores das funções de custo foram reduzidos, conforme mostrado na Tabela 9. Esse atraso pode ser explicado, pois no processamento completo realizado no Matlab utiliza-se a função “hilbert()”. No arquivo “hilbert.m”, verificou-se que após a etapa de processamento utilizando comandos como “fft” e “ifft”, é aplicado a instrução “shiftdim” que corrige os atrasos resultantes das funções empregadas.

Através dos resultados obtidos, constatou-se que não é possível a geração de imagens em tempo real, devido às limitações do Raspberry Pi 3 Modelo B – principalmente frequência de operação e capacidade de memória. O tempo total para todas as etapas de processamento propostas e embarcadas no Raspberry Pi foi de aproximadamente 27 minutos. Entretanto, abre-se uma nova possibilidade de processamento de sinais de US embarcado no Raspberry Pi com a utilização da linguagem de Python, haja vista que não foram encontrados na literatura e demais fontes pesquisadas, até o momento, códigos de programação para essa finalidade. Dessa forma, este trabalho apresenta resultados que podem ser explorados em outros trabalhos futuros pela comunidade científica.

Conclui-se que o objetivo principal deste trabalho, que foi realizar o processamento digital de sinais para geração de imagem no Modo B, a partir de dados brutos de US, utilizando um módulo Raspberry Pi 3 Modelo B, foi obtido com sucesso. As análises qualitativas e quantitativas utilizando as funções de custo NRMSE e NRSS demonstram que o algoritmo em Python implementado no Raspberry Pi, disponível no Apêndice A, apresenta resultados compatíveis com o modelo de referência adotado e validado em estudos prévios (ASSEF, 2013). Todos os resultados do NRMSE foram menores que 10% e do NRSS foram próximos de zero, indicando uma excelente concordância com o modelo do Matlab.

Como trabalhos futuros, sugere-se:

- Implementação da correção dos atrasos utilizando dados brutos de US sem pré-processamento;
- Desenvolvimento da etapa de conversão de varredura para geração da imagem final de US considerando diferentes tipos de transdutores (convexo, linear, *phased array*, dentre outros);
- Estudos visando a otimização para redução do tempo de processamento, por exemplo, alterar a função logarítmica “log” por uma *Look Up Table*;
- Comparação do contraste entre as imagens geradas a partir do processamento no Matlab e no Raspberry Pi;
- Utilização de um *display* do tipo LCD TFT *Touch* de 3,5 polegadas para apresentação da imagem reconstruída.

REFERÊNCIAS

ALI, M.; MAGEE, D.; DASGUPTA, U. **Signal Processing Overview of Ultrasound Systems for Medical Imaging**. Texas Instruments Incorporated, White Paper, nov. 2008.

ASSEF, A. A.; FERREIRA, B. M.; MAIA, SERAFIN, H. S.; BASSAN, G.M.; MORAES FILHO, A. F.; COSTA, E. T. **Projeto de um filtro digital FIR passa-baixa em FPGA para aplicações de processamento de sinais de ultrassom**. XXV Congresso Brasileiro de Engenharia Biomédica – CBEB, 2016.

ASSEF, AMAURI A. **Arquitetura de Hardware Multicanal Reconfigurável com Excitação Multinível para Desenvolvimento e Testes de Novos Métodos de Geração de Imagens por Ultrassom**. Universidade Tecnológica Federal do Paraná, 2013.

ASSEF, AMAURI A.; OLIVEIRA, JONATHAN DE.; MAIA, JOAQUIM M.; COSTA, EDUARDO T. **FPGA implementation and evaluation of an approximate Hilbert Transform-based envelope detector for ultrasound imaging using the DSP builder development tool**. 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Berlin, Germany, p. 2813-2816, 2019.

ASSEF, AMAURI A.; FERREIRA, BRENO M.; MAIA, JOAQUIM M.; ZIMBICO, ACÁCIO J.; COSTA, EDUARDO T. **Modeling and FPGA-based implementation of an efficient and simple envelope detector using a Hilbert Transform FIR filter for ultrasound imaging applications**. Research on Biomedical Engineering, v. 34, n. 1, p. 87-92, 2018.

CHRISTENSEN, DOUGLAS A. **Ultrasonic Bioinstrumentation**. New York: J. Wiley, 1988.

DEMARRE, DEAN A.; MICHAELS, DAVID. **Bioelectronic Measurements**. Englewood Cliffs: Prentice-Hall, 1983. 291p.

EVANS, D. H.; MCDICKEN, W. N. **Doppler Ultrasound: Physics, Instrumentation, and Signal Processing**. 2nd ed. Chichester, ENG; New York, NY: J. Wiley & Sons, 2000.

FERREIRA, BRENO M. A. **Modelagem e Implementação de um Sistema de Processamento Digital de Sinais baseado em FPGA para Geração de Imagens por**

Ultrassom usando o Simulink. Dissertação de Mestrado (PPGSE) – Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, 2017.

FISH, PETER. **Physics and Instrumentation of: Diagnostic Medical Ultrasound.** Chichester: John Wiley & Sons, 1990, 250p.

HASSAN MAWIA A.; YOUSSEF, ABDOL-BAKR M.; KADAH, YASSER M. **Digital Signal Processing Methodologies for Conventional Digital Medical Ultrasound Imaging System.** American Journal of Biomedical Engineering, v. 3, n. 1, p. 14-30, 2013.

HASSAN MAWIA A.; YOUSSEF, ABDOL-BAKR M.; KADAH, YASSER M. **Modular FPGA-Based Digital Ultrasound Beamforming.** 1st Middle East Conference on Biomedical Engineering, United States: IEEE, 2011.

HAYKIN, SIMON S.; VAN VEEN, BARRY. **Sinais e Sistemas.** Porto Alegre, RS: Bookman, 2001, Xvii, 668 p.

HEDRICK, WAYNE R.; HYKES, DAVID L.; STARCHMAN, DALE E. **Ultrasound Physics and Instrumentation.** 4th ed. St. Louis, Mo.: Elsevier Mosby Inc., 2005, 445 p.

JENSEN, J. A.; NIKOLOV, S. I.; GAMMELMARK, K. L.; PEDERSEN, M. H. **Synthetic aperture ultrasound imaging.** Ultrasonics, v 22, n. 44, p. 5-15, dec. 2006.

JI-FENG, DING; SHUANG, XU; ZHANG, JUN-XING; YA-NING, YANG. **Filter Design Based on DSP Builder.** Open Electrical & Electronic Engineering Journal, v. 9, p. 15-21, 2015.

JINBO, M. **Imagens ultrassônicas em modo-B com técnicas de abertura focal sintética–SAFT.** Dissertação (Mestrado em Engenharia) – Universidade de São Paulo – USP, São Paulo, 2007.

KADIYALA, M. D. M., JONES, J. W., MYLAVARAPU, R. S., LI, Y. C., & REDDY, M. D. **Identifying irrigation and nitrogen best management practices for aerobic rice–maize cropping system for semi-arid tropics using CERES-rice and maize models.** Agricultural water management, v. 149, p. 23-32, 2015.

LEVESQUE, P.; SAWAN, M. **Real-Time Hand-Held Ultrasound Medical-Imaging**

Device Based on a New Digital Quadrature Demodulation Processor. IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control, v.56, n. 8, p. 1654-1665, 2009.

MADISETTI, VIJAY K.; WILLIAMS, DOUGLAS. **Digital Signal Processing Handbook.** Boca Raton, Fla.: CRC Press, 1998.

OPPENHEIM, A. V.; SCHAFER, R. W. **Digital signal processing.** New Jersey: Prentice-Hall, 1975.

OTAKE, T.; KAWANO, T.; SUGIYAMA, T.; MITAKE, T.; UMEMURA, S. **High-quality/High-resolution Digital Ultrasound Diagnostic Scanner.** Hitachi, 2003. Disponível em: http://www.hitachi.com/ICSFiles/afiedfile/2004/06/01/r2003_04_106.pdf. Acesso em: 21 out. 2018.

PROAKIS, JOHN G.; MANOLAKIS, DIMITRIS G. **Digital Signal Processing: Principles, Algorithms and Applications.** 4th ed. Prentice Hall, 2007.

PYTHON BRASIL. **Empresas que usam Python.** Disponível em: <https://python.org.br/empresas/>. Acesso em: 02 jul. 2019.

PYTHON. **Python: About.** Disponível em: <https://www.python.org/about/>. Acesso em: 02 jul. 2019.

RABINER, LAWRENCE R.; SCHAFER, RONALD W. **Digital processing of speech signals.** New Jersey: Prentice-Hall Englewood Cliffs, 1978.

RASPBERRYPI. **Raspberry Pi 3 Model B.** Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acesso em: 20 out. 2018.

ROBOCORE. **Descrição do Raspberry Pi 3 - Model B.** Disponível em: <https://www.robocore.net/loja/embarcados/raspberry-pi-3-model-b>. Acesso em: 22 out. 2018.

SHUNG, K. KIRK. **Diagnostic Ultrasound Imaging and Blood Flow Measurements.** Taylor & Francis Group, 2006, 202p.

SHUNG, K. KIRK; SMITH, MICHAEL B.; TSUI, BENJAMIN M.W. **Principles of Medical Imaging**. San Diego: Academic Press Inc., 1992. 289p.

SMITH, STEVEN W. **Digital Signal Processing: A Practical Guide For Engineers and Scientists**. Burlington: Newnes, 2003.

SOHN, H. Y.; KANG, J.; CHO, J.; SONG, T. K.; YOO, Y. **Time-sharing bilinear delay interpolation for ultrasound dynamic receive beamformer**. *Electronics Letters*, v. 47, n. 2, p. 89, 2011.

THE MATHWORKS INC. **Matlab and Introduction to Filter Designer**. Disponível em: <https://www.mathworks.com/help/signal/examples/introduction-to-filter-designer.html>. Acesso em: 02 jul. 2019.

WELLS, PETER N.T. (Ed.) **Advances in Ultrasound Techniques and Instrumentation**. New York: Churchill Livingstone Inc., 1993. 192p.

ZHOU, HAO; ZHENG, YIN-FEI. **An efficient quadrature demodulator for medical ultrasound imaging**. *Frontiers of Information Technology & Electronic*, v. 16, n. 4, p. 301-310, 2015.

APÊNDICE A

```

#-----
# Arquivo TCC
# Curso de Graduação em Engenharia Elétrica
# Autor:Renan Medeiros
# Data: 20/11/2019
# Dados de Entrada: Arquivo 'dados_beam_apo_delay.txt' do Sistema ULTRA-ORS
# Descrição: Projeto completo para processamento digital de sinais de US
#           para reconstrução de imagem no Raspberry Pi
#-----

import numpy as np
import matplotlib.pyplot as plt
import time
import timeit

inicio = time.time()
start = timeit.default_timer()
print("Inicio: ", inicio)
print("Start: ", start)

s = np.zeros((2046,121))

env = np.zeros((2046,121))

env1 = np.zeros((2076,121)) #dimensão maior para poder deslocar

max_linhas = np.zeros((121,1))

limiar_inf = 200

def processamento():

    f_matriz = np.zeros((2046,8))

    y = np.zeros((2046,1))

    #-----
    # Importando entrada do sinal de US a partir de um TXT
    #-----

    with open('dados_beam_apo_delay.txt', 'r') as arquivo:
        z = arquivo.readlines()
    #print(z[0])

    w = np.zeros((2046,1))

    for k in range(0, 121):

        #-----
        # Filtragem
        #-----

        for r in range(0, 8):
            for i in range(0,2046):
                w[i] = z[i+r*2046+k*16368]

```

```

x = w

'''
if k==0 and r==0:
    # Exportando dados para TXT
    with open('teste_python_entrada1.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
        for i in range(0, 2046, 1):
            arquivo.write(str(x[i]) + '\n')
#print(x[0])
'''

# Vetor de saida (FIR)
f = np.zeros((2046,1)) # dimensao 2046x1 - vetor de saida do filtro

# Vetor de coeficientes (FIR)
Coef_FIR = np.array ([-0.0080524653553576461, -0.020318830061531748, -0.028800534328365872,
-0.01643518329923499, 0.030369681429219274, 0.10853961742278931, 0.19336131713263965,
0.24892007978955652, 0.24892007978955652, 0.19336131713263965, 0.10853961742278931,
0.030369681429219274, -0.01643518329923499, -0.028800534328365872, -0.020318830061531748, -
0.0080524653553576461]);

# 'A' recebe os coeficientes transpostos FIR
a = Coef_FIR.T;

# Convolucao
f[0]=x[0]*a[0];
f[1]=x[1]*a[0] + x[0]*a[1];
f[2]=x[2]*a[0] + x[1]*a[1]+ x[0]*a[2];
f[3]=x[3]*a[0] + x[2]*a[1]+ x[1]*a[2]+ x[0]*a[3];
f[4]=x[4]*a[0] + x[3]*a[1]+ x[2]*a[2]+ x[1]*a[3]+ x[0]*a[4];
f[5]=x[5]*a[0] + x[4]*a[1]+ x[3]*a[2]+ x[2]*a[3]+ x[1]*a[4]+ x[0]*a[5];
f[6]=x[6]*a[0] + x[5]*a[1]+ x[4]*a[2]+ x[3]*a[3]+ x[2]*a[4]+ x[1]*a[5]+ x[0]*a[6];
f[7]=x[7]*a[0] + x[6]*a[1]+ x[5]*a[2]+ x[4]*a[3]+ x[3]*a[4]+ x[2]*a[5]+ x[1]*a[6]+ x[0]*a[7];
f[8]=x[8]*a[0] + x[7]*a[1]+ x[6]*a[2]+ x[5]*a[3]+ x[4]*a[4]+ x[3]*a[5]+ x[2]*a[6]+ x[1]*a[7]+
x[0]*a[8];
f[9]=x[9]*a[0] + x[8]*a[1]+ x[7]*a[2]+ x[6]*a[3]+ x[5]*a[4]+ x[4]*a[5]+ x[3]*a[6]+ x[2]*a[7]+
x[1]*a[8]+ x[0]*a[9];
f[10]=x[10]*a[0] + x[9]*a[1]+ x[8]*a[2]+ x[7]*a[3]+ x[6]*a[4]+ x[5]*a[5]+ x[4]*a[6]+ x[3]*a[7]+
x[2]*a[8]+ x[1]*a[9]+ x[0]*a[10];
f[11]=x[11]*a[0] + x[10]*a[1]+ x[9]*a[2]+ x[8]*a[3]+ x[7]*a[4]+ x[6]*a[5]+ x[5]*a[6]+ x[4]*a[7]+
x[3]*a[8]+ x[2]*a[9]+ x[1]*a[10]+ x[0]*a[11];
f[12]=x[12]*a[0] + x[11]*a[1]+ x[10]*a[2]+ x[9]*a[3]+ x[8]*a[4]+ x[7]*a[5]+ x[6]*a[6]+ x[5]*a[7]+
x[4]*a[8]+ x[3]*a[9]+ x[2]*a[10]+ x[1]*a[11]+ x[0]*a[12];
f[13]=x[13]*a[0] + x[12]*a[1]+ x[11]*a[2]+ x[10]*a[3]+ x[9]*a[4]+ x[8]*a[5]+ x[7]*a[6]+ x[6]*a[7]+
x[5]*a[8]+ x[4]*a[9]+ x[3]*a[10]+ x[2]*a[11]+ x[1]*a[12]+ x[0]*a[13];
f[14]=x[14]*a[0] + x[13]*a[1]+ x[12]*a[2]+ x[11]*a[3]+ x[10]*a[4]+ x[9]*a[5]+ x[8]*a[6]+
x[7]*a[7]+ x[6]*a[8]+ x[5]*a[9]+ x[4]*a[10]+ x[3]*a[11]+ x[2]*a[12]+ x[1]*a[13]+ x[0]*a[14];

for i in range(15, 2046, 1):

    f[i]=x[i]*a[0] + x[i-1]*a[1]+ x[i-2]*a[2]+ x[i-3]*a[3]+ x[i-4]*a[4]+ x[i-5]*a[5]+ x[i-6]*a[6]+ x[i-
7]*a[7]+ x[i-8]*a[8]+ x[i-9]*a[9]+ x[i-10]*a[10]+ x[i-11]*a[11]+ x[i-12]*a[12]+ x[i-13]*a[13]+x[i-
14]*a[14]+x[i-15]*a[15];

# monta matriz de uma scanline com 8 elementos na varredura (abertura)

```



```

for i in range(0, 2046):
    f_matriz[i][r] = f[i]

'''
if k==0 and r==0:
# Exportando dados para TXT
    with open('teste_python_linha1.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
        for i in range(0, 2046, 1):
            arquivo.write(str(f_matriz[i][r]) + '\n')
'''
'''
if k==120:
# Exportando dados para TXT
    with open('teste_valor_apodizado_python_sc121.txt', 'w') as arquivo: #com 'w': sobrescreve o
anterior
        for r in range(0, 8, 1):
            for i in range(0, 2046, 1):
                arquivo.write(str(f_matriz[i][r]) + '\n')
            #print(f_matriz)
'''
#-----
# Soma coerente
#-----

for i in range(0, 2046):
    y[i] = f_matriz[i][0] + f_matriz[i][1] + f_matriz[i][2] + f_matriz[i][3] + f_matriz[i][4] + f_matriz[i][5]
+ f_matriz[i][6] + f_matriz[i][7];

'''
if k==0:
# Exportando dados para TXT
    with open('teste_python_sc1.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
        for i in range(0, 2046, 1):
            arquivo.write(str(y[i]) + '\n')
#print(y[0])
'''
#-----
# Envoltória com ordem 32
#-----

# Vetor de saída em Quadratura Q(t) - imaginário (Transformada de Hilbert)
q = np.zeros((2046,1)) # dimensao 2046x1

# Vetor de saída em Fase I(t) - real
j = np.zeros((2046,1)) # dimensao 2046x1

# Vetor de coeficientes (Hilbert)
#Coef_hilbert = np.array ([0.000045448643348697757, -0.0319136045611684, 0, -0.0260478358874456,
0, -0.0373284021357407, 0, -0.0531107459937532, 0, -0.0766995375013514, 0, -0.116853826766850, 0, -
0.205805810057241, 0, -0.634472629392063, 0, 0.634472629392063, 0, 0.205805810057241, 0,
0.116853826766850, 0, 0.0766995375013514, 0, 0.0531107459937532, 0, 0.0373284021357407, 0,
0.0260478358874456, 0, 0.0319136045611684, 0]);
#Coef_hilbert = np.array ([4.54486433486978e-05, -0.0319136045611684, -3.31277815638996e-05, -
0.0260478358874456, -1.59427170184714e-06, -0.0373284021357407, 3.46942587573067e-06, -
0.0531107459937532, -1.03378562693070e-06, -0.0766995375013514, -5.96683448033613e-06, -
0.116853826766850, 1.66624629077077e-05, -0.205805810057241, -1.89302802971109e-05, -
0.634472629392063, 0, 0.634472629392063, 1.89302802971109e-05, 0.205805810057241, -
1.66624629077077e-05, 0.116853826766850, 5.96683448033613e-06, 0.0766995375013514,

```

```
1.03378562693070e-06, 0.0531107459937532, -3.46942587573067e-06, 0.0373284021357407,
1.59427170184714e-06, 0.0260478358874456, 3.31277815638996e-05, 0.0319136045611684, -
4.54486433486978e-05];
```

```
Coef_hilbert = np.array ([ 0.000045448643348697757, -0.031913604561168388, -
0.00003312778156389956, -0.026047835887445595, -0.0000015942717018471429, -
0.037328402135740679, 0.0000034694258757306669, -0.053110745993753178, -
0.0000010337856269306998, -0.076699537501351389, -0.0000059668344803361263, -
0.11685382676685013, 0.00001666246290770766, -0.2058058100572413, -0.00001893028029711089, -
0.63447262939206328, 0, 0.63447262939206328, 0.00001893028029711089, 0.2058058100572413, -
0.00001666246290770766, 0.11685382676685013, 0.0000059668344803361263, 0.076699537501351389,
0.0000010337856269306998, 0.053110745993753178, -0.0000034694258757306669,
0.037328402135740679, 0.0000015942717018471429, 0.026047835887445595, 0.00003312778156389956,
0.031913604561168388, -0.000045448643348697757]);
```

```
# 'B' recebe os coeficientes transpostos
b = Coef_hilbert.T;
```

```
#Envoltória
q[0]=y[0]*b[0];
q[1]=y[1]*b[0]+ y[0]*b[1];
q[2]=y[2]*b[0]+ y[1]*b[1]+ y[0]*b[2];
q[3]=y[3]*b[0]+ y[2]*b[1]+ y[1]*b[2]+ y[0]*b[3];
q[4]=y[4]*b[0]+ y[3]*b[1]+ y[2]*b[2]+ y[1]*b[3]+ y[0]*b[4];
q[5]=y[5]*b[0]+ y[4]*b[1]+ y[3]*b[2]+ y[2]*b[3]+ y[1]*b[4]+ y[0]*b[5];
q[6]=y[6]*b[0]+ y[5]*b[1]+ y[4]*b[2]+ y[3]*b[3]+ y[2]*b[4]+ y[1]*b[5]+ y[0]*b[6];
q[7]=y[7]*b[0]+ y[6]*b[1]+ y[5]*b[2]+ y[4]*b[3]+ y[3]*b[4]+ y[2]*b[5]+ y[1]*b[6]+ y[0]*b[7];
q[8]=y[8]*b[0]+ y[7]*b[1]+ y[6]*b[2]+ y[5]*b[3]+ y[4]*b[4]+ y[3]*b[5]+ y[2]*b[6]+ y[1]*b[7]+
y[0]*b[8];
q[9]=y[9]*b[0]+ y[8]*b[1]+ y[7]*b[2]+ y[6]*b[3]+ y[5]*b[4]+ y[4]*b[5]+ y[3]*b[6]+ y[2]*b[7]+
y[1]*b[8]+ y[0]*b[9];
q[10]=y[10]*b[0]+ y[9]*b[1]+ y[8]*b[2]+ y[7]*b[3]+ y[6]*b[4]+ y[5]*b[5]+ y[4]*b[6]+ y[3]*b[7]+
y[2]*b[8]+ y[1]*b[9]+ y[0]*b[10];
q[11]=y[11]*b[0]+ y[10]*b[1]+ y[9]*b[2]+ y[8]*b[3]+ y[7]*b[4]+ y[6]*b[5]+ y[5]*b[6]+ y[4]*b[7]+
y[3]*b[8]+ y[2]*b[9]+ y[1]*b[10]+ y[0]*b[11];
q[12]=y[12]*b[0]+ y[11]*b[1]+ y[10]*b[2]+ y[9]*b[3]+ y[8]*b[4]+ y[7]*b[5]+ y[6]*b[6]+ y[5]*b[7]+
y[4]*b[8]+ y[3]*b[9]+ y[2]*b[10]+ y[1]*b[11]+ y[0]*b[12];
q[13]=y[13]*b[0]+ y[12]*b[1]+ y[11]*b[2]+ y[10]*b[3]+ y[9]*b[4]+ y[8]*b[5]+ y[7]*b[6]+ y[6]*b[7]+
y[5]*b[8]+ y[4]*b[9]+ y[3]*b[10]+ y[2]*b[11]+ y[1]*b[12]+ y[0]*b[13];
q[14]=y[14]*b[0]+ y[13]*b[1]+ y[12]*b[2]+ y[11]*b[3]+ y[10]*b[4]+ y[9]*b[5]+ y[8]*b[6]+ y[7]*b[7]+
y[6]*b[8]+ y[5]*b[9]+ y[4]*b[10]+ y[3]*b[11]+ y[2]*b[12]+ y[1]*b[13]+ y[0]*b[14];
q[15]=y[15]*b[0]+ y[14]*b[1]+ y[13]*b[2]+ y[12]*b[3]+ y[11]*b[4]+ y[10]*b[5]+ y[9]*b[6]+
y[8]*b[7]+ y[7]*b[8]+ y[6]*b[9]+ y[5]*b[10]+ y[4]*b[11]+ y[3]*b[12]+ y[2]*b[13]+ y[1]*b[14]+ y[0]*b[15];
q[16]=y[16]*b[0]+ y[15]*b[1]+ y[14]*b[2]+ y[13]*b[3]+ y[12]*b[4]+ y[11]*b[5]+ y[10]*b[6]+
y[9]*b[7]+ y[8]*b[8]+ y[7]*b[9]+ y[6]*b[10]+ y[5]*b[11]+ y[4]*b[12]+ y[3]*b[13]+ y[2]*b[14]+ y[1]*b[15]+
y[0]*b[16];
q[17]=y[17]*b[0]+ y[16]*b[1]+ y[15]*b[2]+ y[14]*b[3]+ y[13]*b[4]+ y[12]*b[5]+ y[11]*b[6]+
y[10]*b[7]+ y[9]*b[8]+ y[8]*b[9]+ y[7]*b[10]+ y[6]*b[11]+ y[5]*b[12]+ y[4]*b[13]+ y[3]*b[14]+
y[2]*b[15]+ y[1]*b[16]+ y[0]*b[17];
q[18]=y[18]*b[0]+ y[17]*b[1]+ y[16]*b[2]+ y[15]*b[3]+ y[14]*b[4]+ y[13]*b[5]+ y[12]*b[6]+
y[11]*b[7]+ y[10]*b[8]+ y[9]*b[9]+ y[8]*b[10]+ y[7]*b[11]+ y[6]*b[12]+ y[5]*b[13]+ y[4]*b[14]+
y[3]*b[15]+ y[2]*b[16]+ y[1]*b[17]+ y[0]*b[18];
q[19]=y[19]*b[0]+ y[18]*b[1]+ y[17]*b[2]+ y[16]*b[3]+ y[15]*b[4]+ y[14]*b[5]+ y[13]*b[6]+
y[12]*b[7]+ y[11]*b[8]+ y[10]*b[9]+ y[9]*b[10]+ y[8]*b[11]+ y[7]*b[12]+ y[6]*b[13]+ y[5]*b[14]+
y[4]*b[15]+ y[3]*b[16]+ y[2]*b[17]+ y[1]*b[18]+ y[0]*b[19];
q[20]=y[20]*b[0]+ y[19]*b[1]+ y[18]*b[2]+ y[17]*b[3]+ y[16]*b[4]+ y[15]*b[5]+ y[14]*b[6]+
y[13]*b[7]+ y[12]*b[8]+ y[11]*b[9]+ y[10]*b[10]+ y[9]*b[11]+ y[8]*b[12]+ y[7]*b[13]+ y[6]*b[14]+
y[5]*b[15]+ y[4]*b[16]+ y[3]*b[17]+ y[2]*b[18]+ y[1]*b[19]+ y[0]*b[20];
```

q[21]=y[21]*b[0]+ y[20]*b[1]+ y[19]*b[2]+ y[18]*b[3]+ y[17]*b[4]+ y[16]*b[5]+ y[15]*b[6]+ y[14]*b[7]+ y[13]*b[8]+ y[12]*b[9]+ y[11]*b[10]+ y[10]*b[11]+ y[9]*b[12]+ y[8]*b[13]+ y[7]*b[14]+ y[6]*b[15]+ y[5]*b[16]+ y[4]*b[17]+ y[3]*b[18]+ y[2]*b[19]+ y[1]*b[20]+ y[0]*b[21];

q[22]=y[22]*b[0]+ y[21]*b[1]+ y[20]*b[2]+ y[19]*b[3]+ y[18]*b[4]+ y[17]*b[5]+ y[16]*b[6]+ y[15]*b[7]+ y[14]*b[8]+ y[13]*b[9]+ y[12]*b[10]+ y[11]*b[11]+ y[10]*b[12]+ y[9]*b[13]+ y[8]*b[14]+ y[7]*b[15]+ y[6]*b[16]+ y[5]*b[17]+ y[4]*b[18]+ y[3]*b[19]+ y[2]*b[20]+ y[1]*b[21]+ y[0]*b[22];

q[23]=y[23]*b[0]+ y[22]*b[1]+ y[21]*b[2]+ y[20]*b[3]+ y[19]*b[4]+ y[18]*b[5]+ y[17]*b[6]+ y[16]*b[7]+ y[15]*b[8]+ y[14]*b[9]+ y[13]*b[10]+ y[12]*b[11]+ y[11]*b[12]+ y[10]*b[13]+ y[9]*b[14]+ y[8]*b[15]+ y[7]*b[16]+ y[6]*b[17]+ y[5]*b[18]+ y[4]*b[19]+ y[3]*b[20]+ y[2]*b[21]+ y[1]*b[22]+ y[0]*b[23];

q[24]=y[24]*b[0]+ y[23]*b[1]+ y[22]*b[2]+ y[21]*b[3]+ y[20]*b[4]+ y[19]*b[5]+ y[18]*b[6]+ y[17]*b[7]+ y[16]*b[8]+ y[15]*b[9]+ y[14]*b[10]+ y[13]*b[11]+ y[12]*b[12]+ y[11]*b[13]+ y[10]*b[14]+ y[9]*b[15]+ y[8]*b[16]+ y[7]*b[17]+ y[6]*b[18]+ y[5]*b[19]+ y[4]*b[20]+ y[3]*b[21]+ y[2]*b[22]+ y[1]*b[23]+ y[0]*b[24];

q[25]=y[25]*b[0]+ y[24]*b[1]+ y[23]*b[2]+ y[22]*b[3]+ y[21]*b[4]+ y[20]*b[5]+ y[19]*b[6]+ y[18]*b[7]+ y[17]*b[8]+ y[16]*b[9]+ y[15]*b[10]+ y[14]*b[11]+ y[13]*b[12]+ y[12]*b[13]+ y[11]*b[14]+ y[10]*b[15]+ y[9]*b[16]+ y[8]*b[17]+ y[7]*b[18]+ y[6]*b[19]+ y[5]*b[20]+ y[4]*b[21]+ y[3]*b[22]+ y[2]*b[23]+ y[1]*b[24]+ y[0]*b[25];

q[26]=y[26]*b[0]+ y[25]*b[1]+ y[24]*b[2]+ y[23]*b[3]+ y[22]*b[4]+ y[21]*b[5]+ y[20]*b[6]+ y[19]*b[7]+ y[18]*b[8]+ y[17]*b[9]+ y[16]*b[10]+ y[15]*b[11]+ y[14]*b[12]+ y[13]*b[13]+ y[12]*b[14]+ y[11]*b[15]+ y[10]*b[16]+ y[9]*b[17]+ y[8]*b[18]+ y[7]*b[19]+ y[6]*b[20]+ y[5]*b[21]+ y[4]*b[22]+ y[3]*b[23]+ y[2]*b[24]+ y[1]*b[25]+ y[0]*b[26];

q[27]=y[27]*b[0]+ y[26]*b[1]+ y[25]*b[2]+ y[24]*b[3]+ y[23]*b[4]+ y[22]*b[5]+ y[21]*b[6]+ y[20]*b[7]+ y[19]*b[8]+ y[18]*b[9]+ y[17]*b[10]+ y[16]*b[11]+ y[15]*b[12]+ y[14]*b[13]+ y[13]*b[14]+ y[12]*b[15]+ y[11]*b[16]+ y[10]*b[17]+ y[9]*b[18]+ y[8]*b[19]+ y[7]*b[20]+ y[6]*b[21]+ y[5]*b[22]+ y[4]*b[23]+ y[3]*b[24]+ y[2]*b[25]+ y[1]*b[26]+ y[0]*b[27];

q[28]=y[28]*b[0]+ y[27]*b[1]+ y[26]*b[2]+ y[25]*b[3]+ y[24]*b[4]+ y[23]*b[5]+ y[22]*b[6]+ y[21]*b[7]+ y[20]*b[8]+ y[19]*b[9]+ y[18]*b[10]+ y[17]*b[11]+ y[16]*b[12]+ y[15]*b[13]+ y[14]*b[14]+ y[13]*b[15]+ y[12]*b[16]+ y[11]*b[17]+ y[10]*b[18]+ y[9]*b[19]+ y[8]*b[20]+ y[7]*b[21]+ y[6]*b[22]+ y[5]*b[23]+ y[4]*b[24]+ y[3]*b[25]+ y[2]*b[26]+ y[1]*b[27]+ y[0]*b[28];

q[29]=y[29]*b[0]+ y[28]*b[1]+ y[27]*b[2]+ y[26]*b[3]+ y[25]*b[4]+ y[24]*b[5]+ y[23]*b[6]+ y[22]*b[7]+ y[21]*b[8]+ y[20]*b[9]+ y[19]*b[10]+ y[18]*b[11]+ y[17]*b[12]+ y[16]*b[13]+ y[15]*b[14]+ y[14]*b[15]+ y[13]*b[16]+ y[12]*b[17]+ y[11]*b[18]+ y[10]*b[19]+ y[9]*b[20]+ y[8]*b[21]+ y[7]*b[22]+ y[6]*b[23]+ y[5]*b[24]+ y[4]*b[25]+ y[3]*b[26]+ y[2]*b[27]+ y[1]*b[28]+ y[0]*b[29];

q[30]=y[30]*b[0]+ y[29]*b[1]+ y[28]*b[2]+ y[27]*b[3]+ y[26]*b[4]+ y[25]*b[5]+ y[24]*b[6]+ y[23]*b[7]+ y[22]*b[8]+ y[21]*b[9]+ y[20]*b[10]+ y[19]*b[11]+ y[18]*b[12]+ y[17]*b[13]+ y[16]*b[14]+ y[15]*b[15]+ y[14]*b[16]+ y[13]*b[17]+ y[12]*b[18]+ y[11]*b[19]+ y[10]*b[20]+ y[9]*b[21]+ y[8]*b[22]+ y[7]*b[23]+ y[6]*b[24]+ y[5]*b[25]+ y[4]*b[26]+ y[3]*b[27]+ y[2]*b[28]+ y[1]*b[29]+ y[0]*b[30];

q[31]=y[31]*b[0]+ y[30]*b[1]+ y[29]*b[2]+ y[28]*b[3]+ y[27]*b[4]+ y[26]*b[5]+ y[25]*b[6]+ y[24]*b[7]+ y[23]*b[8]+ y[22]*b[9]+ y[21]*b[10]+ y[20]*b[11]+ y[19]*b[12]+ y[18]*b[13]+ y[17]*b[14]+ y[16]*b[15]+ y[15]*b[16]+ y[14]*b[17]+ y[13]*b[18]+ y[12]*b[19]+ y[11]*b[20]+ y[10]*b[21]+ y[9]*b[22]+ y[8]*b[23]+ y[7]*b[24]+ y[6]*b[25]+ y[5]*b[26]+ y[4]*b[27]+ y[3]*b[28]+ y[2]*b[29]+ y[1]*b[30]+ y[0]*b[31];

for i in range(32, 2046, 1):

q[i]=y[i]*b[0]+ y[i-1]*b[1]+ y[i-2]*b[2]+ y[i-3]*b[3]+ y[i-4]*b[4]+ y[i-5]*b[5]+ y[i-6]*b[6]+ y[i-7]*b[7]+ y[i-8]*b[8]+ y[i-9]*b[9]+ y[i-10]*b[10]+ y[i-11]*b[11]+ y[i-12]*b[12]+ y[i-13]*b[13]+ y[i-14]*b[14]+ y[i-15]*b[15]+ y[i-16]*b[16]+ y[i-17]*b[17]+ y[i-18]*b[18]+ y[i-19]*b[19]+ y[i-20]*b[20]+ y[i-21]*b[21]+ y[i-22]*b[22]+ y[i-23]*b[23]+ y[i-24]*b[24]+ y[i-25]*b[25]+ y[i-26]*b[26]+ y[i-27]*b[27]+ y[i-28]*b[28]+ y[i-29]*b[29]+ y[i-30]*b[30]+ y[i-31]*b[31]+ y[i-32]*b[32];

for i in range(0, 15, 1):

j[i]=0;

for i in range(16, 2046, 1):

j[i]=y[i-16];

```

# Calculando o envelope

for i in range(0,2046):

    env1[i][k] = np.sqrt((q[i]**2)+(j[i]**2))

for i in range(0,2046):
    env[i][k] = env1[i+16][k]; #deslocando vetor

'''
if k==0:
    # Exportando dados para TXT
    with open('teste_python_sc1_env.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
        for i in range(0, 2046, 1):
            arquivo.write(str(env[i][k]) + '\n')
if k==0:
    # Exportando dados para TXT
    with open('teste_python_sc1_env_sem_desloc.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
        for i in range(0, 2046, 1):
            arquivo.write(str(env1[i][k]) + '\n')
'''

'''
# Exportando dados para TXT
with open('teste_env_completo_python.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
    for k in range(0, 121, 1):
        for i in range(0, 2046, 1):
            arquivo.write(str(env[i][k]) + '\n')
'''

value = 3.823465047158225e+03 #valor máximo para todas as linhas e colunas
print(value)

for k in range(0, 121):

    #-----
    # Limite Superior, Inferior e Compressão Log
    #-----

    for i in range(0,2046):
        if env[i][k] > value: #limite superior
            env[i][k] = value;

        if env[i][k] < limiar_inf: #limite inferior
            env[i][k] = limiar_inf;

        env[i][k] = env[i][k] - limiar_inf;
        aux = value - limiar_inf;

        s[i][k] = 20*(np.log10(env[i][k]/aux)) #compressão log

        if s[i][k] < -30: #ajuste de faixa dinâmica
            s[i][k] = -30;

# Exportando dados para TXT
with open('processamento_completo_env32_v2.txt', 'w') as arquivo: #com 'w': sobrescreve o anterior
    for k in range(0, 121, 1):
        for i in range(0, 2046, 1):
            arquivo.write(str(s[i][k]) + '\n')

```

```
    return s

processamento()
end = timeit.default_timer()
fim = time.time()

print("End: ", end)
print("Fim: ", fim)
print("Time: ", end-start)
print("Duração: ", fim-inicio)

#-----
# Plotando imagem de US
#-----

plt.imshow(s, extent=[0,2046,0,2046], cmap=plt.get_cmap('gray'))
plt.colorbar()
plt.show()
```