UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**BRUNO DAL BÓ SILVA**

# LOCALIZAÇÃO DE SOM COM SINAIS BINAURAIS

Porto Alegre
2011

**BRUNO DAL BÓ SILVA**

# LOCALIZAÇÃO DE SOM COM SINAIS BINAURAIS

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

ORIENTADOR: Prof. Dr. Marcelo Götz

Porto Alegre
2011

**BRUNO DAL BÓ SILVA**

# LOCALIZAÇÃO DE SOM COM SINAIS BINAURAIS

Este Projeto foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universidade de Paderborn – Paderborn, Alemanha

Banca Examinadora:

Prof. Dr. Adalberto Schuck Jr., UFRGS
Doutor pela UFRGS – Porto Alegre, Brasil

Eng. Msc. Javier Aprea, IMS Power Quality
Mestre pela UFRGS – Porto Alegre, Brasil.
Mestre pela TU/e – Eindhoven, Holanda

Chefe do DELET: _____
Altamiro Amadeu Susin

Porto Alegre, dezembro de 2011.

# DEDICATÓRIA

Dedico este projeto aos meus pais, pela perseverança que me transmitiram; ao meu irmão, que caminha, a passos largos, para se tornar um engenheiro com uma criatividade sem precedentes e aos meus amigos e colegas, pela camaradagem e *feedback* a qualquer hora.

# AGRADECIMENTOS

Agradeço a todos que contribuíram para este Projeto e que proporcionaram, mesmo que inconscientemente, momentos de iluminação. Aos professores do Departamento de Engenharia Elétrica, que me orientaram e indicaram o caminho das boas práticas. A todos que apoiaram os momentos difíceis. Ao meu chefe e mentor. A paciência de todos. Obrigado.

# RESUMO

Este projeto apresenta uma proposta de implementação de localização de sons com sinais binaurais usando modelos matemáticos. O relatório compreende o planejamento de gestão durante a execução do projeto; uma abordagem básica do assunto com análise de modelos; a verificação dos modelos usando o software Matlab e a descrição da implementação em código de máquina.

**Palavras-chave: Som, Binaural, DSP.**

# ABSTRACT

This project presents a proposal for implementing sound localisation through mathematical models that produce binaural signals. This report comprehends the management and planning during implementation and execution; s simple view on the general subject and model analysis; model verification with Matlab and the description of the final implementation in machine code.

**Keywords: Sound, Binaural, DSP.**

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTAGENS

# LISTA DE ABREVIATURAS

| | |
|---|---|
| HRTF | *Head-Related Transfer Function* |
| HRIR | *Head-Related Impulse Response* |
| ITD | *Interneural Time Difference* |
| ILD | *Interneural Level Difference* |
| HS | *Head Shadowing* |
| HAT | *Head and Torso* |
| DFT | *Discrete Fourier Transform* |
| FFT | *Fast Fourier Transform* |
| DSP | *Digital Signal Processing* |
| MTB | *Motion-Tracked Binaural* |

# LISTA DE SÍMBOLOS

$a_h$ Raio da cabeça

$\theta$ Ângulo azimutal

$\Theta_l$ Posição da orelha esquerda no plano azimutal

$\Theta_r$ Posição da orelha direita no plano azimutal

$\phi$ Ângulo de elevação

$\overrightarrow{p}$ Ponto (vetor de localização) virtual de origem do som em coordenadas esféricas

$\overrightarrow{e_r}$ Ponto (vetor de localização) representando a orelha direita em coordenadas esféricas

$\overrightarrow{e_r}$ Ponto (vetor de localização) representando a orelha esquerda em coordenadas esféricas

$d$ Distância radial do ponto de origem do som até o centro da cabeça

$c$ Velocidade do som no ar ($\approx$343m/s)

$F_s$ Frequência de amostragem

$T_s$ Período de amostragem

# SUMÁRIO

# 1 INTRODUÇÃO

A humanidade começou a desfrutar das vantagens da informação pelo som em 1877, com a invenção do Gramofone[38] e, desde então, inúmeras tecnologias surgiram para aperfeiçoar o som gravado e reproduzido. Inicialmente, éramos capazes de gravar e reproduzir apenas um canal de som, chamado de sinal *monofônico* ou *mono*. Em 1881, foi introduzido o conceito de som estereofônico ou estéreo, onde dois canais eram usados para gravar e reproduzir o som. No entanto, até 1931[41], este conceito não foi patenteado e introduzido ao público. A introdução do som estéreo foi o primeiro passo para a marcha rumo à localização de sons no espaço. A palavra estéreo, ou, em inglês, *stereo*, vem do grego *stereos* e significa "firme" ou "sólido", pela sensação maior do ouvinte de estar em outro lugar comparado ao som mono. A partir desta filosofia, muitos outros padrões seguiram, sempre tentando aumentar a sensação de localização no espaço. Surgiram, então, diversas tecnologias de *surround sound* que valem-se do princípio da utilização de múltiplas caixas de som em torno do ouvinte para dar a sensação de localização no plano 2-D[42]. Alguns padrões prevaleceram como o 3.1 (*left , right, rear* + *woofer*) e o 5.1 (*left , right, rear-left, rear-right, center* + *woofer*). No entanto, nas últimas duas décadas, com a avenção e difusão do som digital, muitos pesquisadores começaram a implementar teorias da psicoacústica (um ramo da psicofísica) que datam do início do século XX para emular melhor esta localização de som. Introduz-se, então, o conceito de som *binaural*, referente às duas "entradas" de áudio que o sistema de percepção humano possui. Teoricamente, se é possível enviar ao sistema auditivo um sinal igual ao que este receberia de uma fonte de som no espaço, podemos criar a falsa sensação de espacialização e – o que é o grande trunfo desta tecnologia – sem a discretização de canais como nas tecnologias de *surround sound*. As técnicas de gravação e reprodução deste tipo de sinais variam bastante e vão desde usar microfones em ouvidos de manequins até tratar o som com filtros digitais. Herdando as características físicas da nossa percepção sonora, existem algoritmos para tratar sons simples (geralmente monofônicos) e dividir este sinal tratado diferentemente para o ouvido esquerdo ou direito. Este projeto explora esses modelos de filtros e propõe uma implementação em máquina para validar a viabilidade de distribuição em massa de um padrão usando esta filosofia.

# 2 CONCEITOS BÁSICOS

O estudo da percepção do som pelo sistema nervoso data de quase um século e existem muitas teorias, às vezes divergentes, sobre o assunto.Antes de entrar na implementação, faz-se necessário um estudo dos modelos existentes assim como das técnicas mais empregadas para a localização virtual de som.

## 2.1 Convenções e geometria

Antes de prosseguir, introduzimos os termos comuns e altamente empregados nesta área. A parametrização do espaço se faz em coordenadas esféricas com o ponto {0,0,0} sendo o centro de uma esfera de raio $a_h$, o raio da cabeça supostamente esférica. O ponto $\overrightarrow{p}$ que indica a fonte de som é representado por:

- $d$ : a distância radial a partir do centro do sistema
- $\theta$ : o ângulo azimutal contando a partir do plano médio (frontal) e positivo em direção à orelha esquerda
- $\phi$ : a elevação contando a partir do plano horizontal (plano dos olhos/orelhas) e positivo para cima

A figura 1 ilustra a esfera de localização. Neste relatório, nos interessaremos principalmente pelo ângulo azimutal $\theta$. Vale lembrar, também, as coordenadas da orelha direita : $\overrightarrow{e_r} = \{a_h, -90°, 0°\}$ e da orelha esquerda: $\overrightarrow{e_l} = \{a_h, 90°, 0°\}$

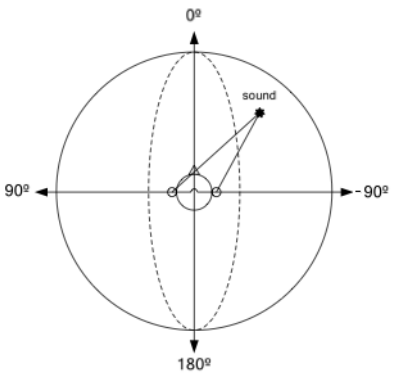



Figura 1: Representação do espaço esférico centrado na cabeça[31]

## 2.2 As abordagens clássicas

A primeira maneira – e a mais intuitiva – de abordar o problema da localização do som é observar os sinais recebidos diretamente no ouvido interno. Uma técnica altamente empregada é a de instalar pequenos microfones nos ouvidos de pessoas ou até mesmo de bonecos (*dummies*) com uma forma de cabeça realista. Capturando este sinal e comparando aos que chegam em cada ouvido, podemos ter uma ideia de como funciona o sistema de localização que todos possuímos como capacidade inata. Desta técnica, advém uma visão mais moderna, que conta com filtros digitais para obter os resultados esperados: tendo o resultado destes testes (normalmente feitos em salas anecóicas[1]), usam-se técnicas de processamento digital de sinais para desconvoluir o sinal e obter a resposta impulsiva do sistema. Esta resposta é referida como HRIR: *Head-Related Impulse Respose* ou Resposta Impulsiva Relacionada à Cabeça[46]. Logo, pelo princípio de sistemas lineares, podemos convoluir no tempo qualquer sinal de som e obter o sinal binaural através da resposta impulsiva obtida empiricamente. O problema desta abordagem está na grande base de dados de respostas impulsivas que se deve dispor para a obtenção de todas as respostas possíveis do sistema. Por não existir um único sinal de referência, estas respostas impulsivas variam com a posição do som, com o formato da cabeça, da orelha e do torso do ouvinte e, além disso, visto que os sinais são obtidos de maneira experimental, não é possível ter todo o espaço mapeado – este é discretizado em incrementos de ângulo. Para se conseguir um efeito contínuo a partir deste método, utilizam-se interpolações da resposta impulsiva, o que é computacionalmente muito caro e não muito preciso. Existem bancos de dados livres com informações deste tipo de medidas, os mais conhecidos são o *CIPIC HRTF Database* e o *MIT Dummy Kemar HRTF Measurements* encontrados em [9] e [10]. Algazi et Al. descrevem a base CIPIC em [14].

Aplicando transformadas de Fourier na resposta impulsiva obtemos a HRTF: *Head-Related Transfer Function*, ou a Função de Transferência Relacionada à Cabeça. Em [25], temos um exemplo de algoritmo de interpolação para estas respostas no domínio da frequência. Existem discussões sobre qual domínio seria mais adequado para abordar o problema, ou seja, se o sistema nervoso conta com a frequência ou com o tempo para distinguir origens de som [17][18][22].

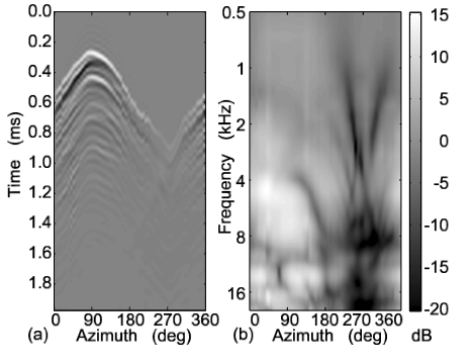



Figura 2: Representação no tempo e frequência das respostas relacionadas à cabeça para um ouvido[14]

[1] salas projetadas de modo a minimizar fenômenos de eco e reverberação.

Uma segunda vertente da área aposta em modelos matemáticos para gerar filtros digitais que transformam um som mono em dois sinais destinados aos ouvidos esquerdo e direito. Esta visão sobre o fenômeno vale-se de estudos de psicoacústica, física de ondas e acústica digital para gerar esses modelos. Existem diferentes tentativas para modelar a resposta impulsiva da cabeça com fórmulas matemáticas e filtros contínuos. Temos desde modelos muito simples, que consideram apenas características evidentes do sistema como a distância entre os ouvidos e a diferença de caminho da onda até modelos que compreendem a forma da orelha externa (pina) e as reflexões no torso (*Head-and-Torso Model* – HAT). Como discutido anteriormente, temos também diferentes modelagens que visam aproximar a resposta temporal ou em frequência do sistema real. Em [44] e [43], encontramos modelos complexos sugeridos por Zhang, onde temos uma expansão modal em séries e polinômios de Bessel com aproximações muito boas da HRIR medida nos bonecos Kemar do banco de dados disponibilizado pelo MIT. Estes modelos são, no entanto, extremamente complexos e de alto custo computacional e vamos nos focar nos modelos modulares compostos de filtros lineares propostos por Brown e Duda, em [17]. Este modelo proposto é amplamente aceito e reconhecido por sua simplicidade e resultados assaz satisfatórios.

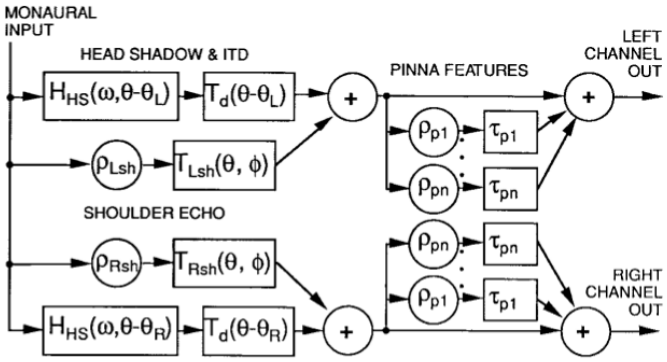



Figura 3: Modelo de blocos do sistema proposto por Brown e Duda[17]

Brown e Duda propuseram o sistema mostrado na figura 3. Eles decompuseram as chamadas *cues* (“deixas” ou “dicas”, do inglês) de localização em vários fatores. Temos que a impressão do ângulo azimutal $\theta$ é, basicamente, devido a dois fatores: a diferença de tempo que o som leva para chegar em cada ouvido (ITD : *Interneural Time Diference*) e a diferença de energia com que chega (ILD : *Interneural Level Diference*). A ITD se dá basicamente graças à propagação do som no ar e naturalmente existe uma diferença de percurso entre as duas orelhas, logo, essa é modelada como um simples atraso ($T_d$, na figura). A ITD acontece por um fenômeno chamado *Head-Shadowing* (HS), literalmente, “sombreamento da cabeça” que é o efeito da difração que sofre a onda em torno da cabeça ($H_{hs}$, na figura)[20]. A modelagem da espacialização no plano horizontal é comumente referida como *Duplex Theory*, porém, essa modelagem apresenta algumas limitações que veremos adiante. Adicionalmente temos que a noção de elevação é dada pelo eco no ombro – *shoulder echo* – , que seria função basicamente da elevação, e dos reflexos e interferências na pina, a parte externa da orelha. Verifica-se, experimentalmente, que o eco do ombro não adiciona uma grande noção de elevação e é frequentemente ignorado

nos modelos[17]. A modelagem sugerida para os efeitos da pina consiste em uma série de ecos com tempos e amortecimentos diferentes e, de acordo com o modelo proposto, essas seriam as principais *cues* do ângulo de elevação $\phi$ para o sistema. Veremos a expansão matemática e experimentação destes modelos no capítulo 4.

### 2.2.1 Discussão dos modelos

Apresentamos o modelo da figura 3 como sendo confiável e suficientemente realista. No entanto, faz-se necessária uma discussão sobre o quanto isto seria verdade. Verifica-se, experimentalmente, que as ITDs, na verdade, só fornecem uma boa localização para frequências muito baixas ($f \leq 1.5kHz$) e, além disso, em uma sala com muita reverberação as interferências tornam a ITD irreconhecível e inútil na localização da fonte de som[20]. Por este fato, considera-se que usamos as ITDs para localizar sons de baixas frequências e a ILD para sons mais agudos pela característica dos filtros, pois a ILD costuma ser plana para frequências baixas e a ITD é basicamente insensível para frequência [15]. Adicionalmente, os modelos de elevação disponíveis são fracos e de baixa performance. Visto que a ITD só funciona para baixas frequências e a ILD tem um espectro mais colorido, porém não compreende toda a resposta, existem estudos sobre o impacto da PRTF *Pinna-Related Transfer Function*, ou a função de transferência relacionada à pina. Como veremos adiante, o modelo de Brown e Duda conta com uma abordagem no domínio do tempo e é provado que o sistema nervoso conta com muitas deixas espectrais para a localização, advindas da resposta da pina[35][21][36]. Complementarmente, em [36], temos um estudo de psicoacústica sobre indivíduos que sofrem de deficiência auditiva em um ouvido ou audição monoaural crônica. Estes indivíduos são capazes de localizar a origem de um som no espaço sem contar com a ITD, visto que dispõem de apenas uma entrada de sinal. O que nos faz pensar que a resposta em frequência do sistema conta fortemente para a sensação de localização.

A teoria *Duplex* apresenta duas grandes ambiguidades que são comumente observadas: o fenômeno de lateralização e a inversão frente-trás. A lateralização consiste em um *loophole* do algoritmo em que o ouvinte tem a impressão de que o som está dentro da cabeça, diminuindo o parâmetro – subjetivo – de *externalização*. Em [18], Brown et Al. propõem a introdução de um eco simples para aumentar a sensação de externalização causada pelo algoritmo simples baseado na *Duplex Theory*. Parece evidente, no entanto, que apenas um eco não representa de maneira realista uma sala. Temos, então, que a localização pode ser melhorada por uma boa modelização dos efeitos da sala por filtros denominados RIR, *Room Impulse Response* ou Resposta Impulsiva da Sala [15]. A inversão frente-trás advém de uma igualdade matemática nos modelos quando localizamos a fonte de som na frente ou no seu ponto-espelho, atrás da cabeça. Apesar de a RIR contribuir para a redução da lateralização, a inversão frente-trás continua um mistério e temos diferentes propostas para reduzir este problema, que incluem a modelização do ouvido interno usando a escala de Bark, herdada de estudos de psicoacústica[31]. É um consenso geral que o cérebro conta com a ajuda de outros sentidos para mimizar estas confusões, incluindo o chamado "cone de confusão", que se trata de um cone de caminhos iguais para o ouvido quando leva-se em conta a elevação[2]. Para a indústria de entretenimento audiovisual, este problema é facilmente corrigido com o uso de imagens. Ao jogar um *video game* que dispõe desta tecnologia, por exemplo, o jogador conta constantemente com di-

[2]O cone de confusão é um fenômeno recorrente e presente mesmo nos modelos convolutivos. Neste trabalho damos mais atenção, no entanto, aos efeitos sobre o plano horizontal considerando apenas a variação do ângulo azimutal.

cas visuais da origem do som e a compensação dos canais ao mover o personagem na tela modifica os filtros e ajuda na sensação de localização. Para aplicações de imersão sonora, no entanto, não dispomos de guias visuais para as fontes sonoras e, neste caso, introduz-se o conceito de MTB, *Motion-Tracked Binaural* ou Binaural Guiado por Movimento. Nestas aplicações usa-se um sensor de movimento na cabeça do ouvinte para compensar o movimento da cabeça e o usuário pode eliminar a confusão frente-trás movimentando o pescoço. Prova-se que o erro por inversão é fortemente diminuído com o uso de tal técnica e que este tipo de abordagem se tornou uma grande área de pesquisa[12][10].

Finalmente devemos levar em consideração todas as não idealidades dos sistemas de gravação e de reprodução de som. É comum usar filtros de pré-ênfase para compensar a resposta dos fones de ouvido usados para reproduzir o som, assim como em sistemas convolutivos com base de dados, as respostas dos microfones usados devem ser bem conhecidas[29].

## 2.3 Conclusão

Vimos alguns modelos e filtros para introduzir os conceitos da área antes de passarmos aos testes e implementação. Temos, então, um breve resumo dos pontos principais blocos de operação e termos que serão revisitados adiante:

- Ângulo azimutal ($\theta$) : Representa o ângulo em um sistema de coordenadas esféricas contado a partir do plano médio e positivo para o lado da orelha esquerda – principal parâmetro explorado neste projeto;
- ITD – *Interneural Time Diference*: Representa a diferença de tempo que o som leva para chegar em cada ouvido. Muito efetivo na localização para baixas freqüências, porém não representa uma grande adição na localização para frequências superiores a $1.5kHz$;
- ILD – *Interneural Level Diference*: Representa a diferença de energia do som chegando em cada ouvido. Maior influência devido à sombra da cabeça (*Head-Shadow* – HS); Usualmente modelado pela equação da difração para uma cabeça esférica;
- PRTF – *Pinna-Related Transfer Function*: Representa o filtro da pina, parte externa da orelha, representado por uma série de ecos que causam interferências construtivas e destrutivas;
- RIR – *Room Impulse Response*: Representa o filtro que modela a sala onde o som é reproduzido. Pode aumentar significativamente a sensação de espacialização e elimina o problema de lateralização;
- MTB – *Motion-Tracked Binaral*: Técnica que usa um agente de compensação em forma de sensor de movimento para acrescentar o movimento da cabeça no aperfeiçoamento do sistema de localização;

Veremos no capítulo 4 a modelização matemática destes blocos encontrada nas diferentes referências.

# 3 GESTÃO DE PROJETO

Antes de começar o modelo matemático, gostaria de abordar a parte administrativa do projeto, com algumas técnicas de gestão. Uma vez que as linhas gerais do assunto estão assimiladas, pude prever tarefas e atribuir alocação de recursos (tempo, neste caso) para cada uma.

## 3.1 Discriminação de objetivos

Decompus o projeto em objetivos principais e secundários. Os objetivos principais fazem parte da implementação fundamental do conceito inicial e os secundários seriam atividades extras, porém não fundamentais para a realização dos objetivos.

Tabela 1: Objetivos Principais

| **Nome** | **Descrição** |
|---|---|
| Modelo Matlab | Implementar os filtros encontrados na bibliografia em Matlab para verificar os modelos e obter os gráficos das respostas espectrais e temporais. |
| Configuração de sistema embarcado | Configurar corretamente um sistema embarcado para suportar os algoritmos necessários para o processamento. |
| Implementação em Máquina | Transporte do modelo Matlab para código de máquina (C/C++). |

Tabela 2: Objetivos Secundários

| **Nome** | **Descrição** |
|---|---|
| Compensação de movimento | Compensação do movimento da cabeça no modelo MTB |
| Implementação Tempo Real | Obtenção de processamento em tempo real com o mínimo atraso possível. |

## 3.2 Gráfico de Gantt

Uma vez os objetivos estabelecidos, pude estabelecer tarefas e subtarefas pertinentes a cada grande área científica. O método de Pert-Gantt é bastante utilizado em gestão de projeto para organizar grupos com objetivos comuns e tarefas compartilhadas. Como este projeto é desenvolvido por apenas uma pessoa, o gráfico de Gantt não é um grande foco e pode ser encontrado no apêndice A.

# 4 MODELO MATEMÁTICO

Neste capítulo, aprofundaremos os estudos das equações matemáticas que regem os modelos matemáticos avaliados no capítulo 2. Iniciamos descrevendo as equações encontradas para em seguida apresentar o código em Matlab usado como prova dos modelos e, por fim, os resultados obtidos.

## 4.1 ITD

Como vimos, a ITD é a diferença de tempo da chegada do som de um ouvido ao outro. Este efeito é modelado considerando os parâmetros $\theta$, $\overrightarrow{er}$, $\overrightarrow{el}$ e $\overrightarrow{p}$. A diferença de tempo de chegada está diretamente relacionada à distância que o som deve percorrer. Adotando uma cabeça supostamente esférica e reutilizando as definições do capítulo 2, podemos deduzir esta diferença de tempo de acordo com Miller, em [32], pelos seguintes procedimentos. Considere o plano horizontal descrito pela figura 4.

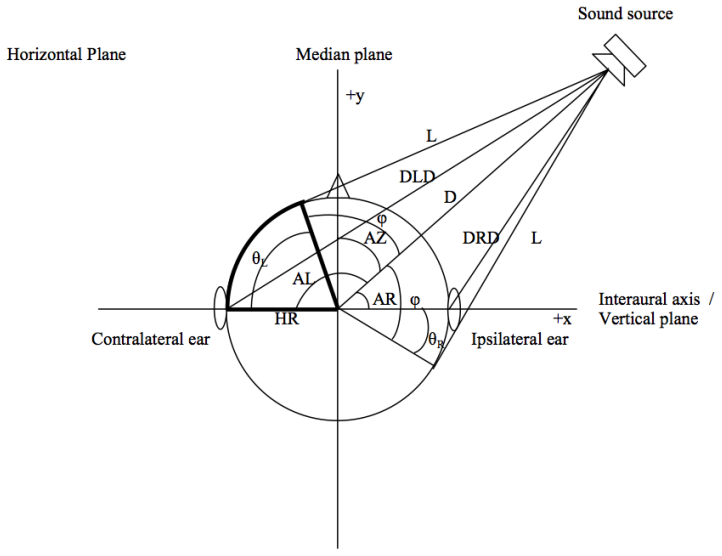



Figura 4: Esquematização para cálculo da ITD por Miller[32]

Explicando a notação empregada pelo autor:

- **L** Distância da fonte ao ponto tangente;
- **D** Distância da fonte ao centro da cabeça ($d$);

- **DLD** Distância linear para a orelha esquerda;
- **DRD** Distância linear para a orelha direita;
- **DLA** Comprimento do arco do ponto tangente à orelha esquerda;
- **DRA** Comprimento do arco do ponto tangente à orelha direita;
- **AZ** Ângulo Azimutal ($\theta$);
- **HR** Raio da cabeça ($a_h$);
- $\theta_l$ Ângulo entre a posição da orelha esquerda e o ponto tangente;
- $\theta_r$ Ângulo entre a posição da orelha direita e o ponto tangente;
- $\phi$ Ângulo entre a reta **D** e o ponto de tangência;
- **AL** Ângulo entre a reta **D** e o ponto de tangência esquerdo;
- **AR** Ângulo entre a reta **D** e o ponto de tangência direito;
- $D_l$ Distância total para orelha esquerda;
- $D_r$ Distância total para orelha direita;
- $\Delta D$ Diferença de caminho do som;
- **ITD** Diferença temporal buscada.

Resolvendo o problema dado $\overrightarrow{p} = \{AZ, D\}$ ou $\overrightarrow{p} = \{x_s, y_s\}$ em coordenadas cartesianas.

Resolvemos para a distância principal $L$:

$$\begin{aligned} HR^2 + L^2 &= D^2 \\ L &= \sqrt{D^2 - HR^2} \end{aligned} \tag{1}$$

Resolvemos para $\phi$:

$$\begin{aligned} \cos(\phi) &= \frac{HR}{D} \\ \phi &= \cos^{-1}\left(\frac{HR}{D}\right) \end{aligned} \tag{2}$$

Definimos os ângulos necessários $\theta_l$ e $\theta_r$

$$\begin{aligned} AL &= \frac{\pi}{2} + AZ \\ AR &= \frac{\pi}{2} - AZ \\ \theta_l &= AL - \phi \qquad (3) \\ \theta_r &= AR - \phi \qquad (4) \end{aligned}$$

$DLA$ e $DRA$:

$$\begin{aligned} DLA &= \theta_l \cdot HR \quad (5) \\ DRA &= \theta_r \cdot HR \quad (6) \\ & \quad (7) \end{aligned}$$

e $DLD$:

$$\begin{aligned} DLD &= \sqrt{(x_s + HR)^2 + y_s^2} \\ DLD &= \sqrt{(D \cdot \sin(AZ) + HR)^2 + (D \cdot \cos(AZ))^2} \\ DLD &= \sqrt{D^2 + HR^2 + 2D \cdot HR \cdot \sin(AZ)} \quad (8) \end{aligned}$$

analogamente $DRD$:

$$\begin{aligned} DRD &= \sqrt{(x_s - HR)^2 + y_s^2} \\ DRD &= \sqrt{(D \cdot \sin(AZ) + HR)^2 + (D \cdot \cos(AZ))^2} \\ DRD &= \sqrt{D^2 + HR^2 - 2D \cdot HR \cdot \sin(AZ)} \quad (9) \end{aligned}$$

Agora o que temos são duas distâncias possíveis para cada direção em pares independentes. Temos as distâncias direta e de arco que são decididas de acordo com as seguintes equações:

$$DL = \begin{cases} DLD & , \quad DLD < L \\ L + DLA & , \quad DLD \geq L \end{cases} \quad (10)$$

$$DR = \begin{cases} DRD & , \quad DRD < L \\ L + DRA & , \quad DRD \geq L \end{cases} \quad (11)$$

Logo, a diferença entre os caminhos é dada por:

$$\Delta D = DL - DR \quad (12)$$

e a diferença de tempo $ITD$ é obtida com a velocidade do som $c$:

$$ITD = \frac{\Delta D}{c} \quad (13)$$

Por último, como trabalhamos em domínio de tempo discreto, nossa diferença de tempo deve ser em amostras, definida por $ITD_s$. Logo, conhecendo a frequência de amostragem $F_s$ temos, dado o operador $round$ que arredonda o argumento de entrada:

$$ITD_s = round(ITD \cdot F_s) \quad (14)$$

Para avaliar o algoritmo de Miller, implementei a lógica matemática descrita nas equações acima em Matlab. O código pode ser visto na listagem 4.1. Para confirmar a continuidade e validade do retorno desta função, a traçamos para o intervalo $\theta = \{-180°, 180°\}$. O resultado pode ser visto na figura 5. Como podemos ver, para $\theta > 0°$, a fonte de som está do lado esquerdo da cabeça e, logo, temos uma diferença de tempo positiva, pois $DL > DR$ e o sinal vem da equação 12. A forma de onda se aproxima muito de uma

senóide perfeita. No entanto, isto só se dá pelos argumentos $a_h = 0.08m$ e $d = 1.0m$ usados para traçar o gráfico. Para $d$ muito próximo do raio da cabeça $a_h$, surge uma singularidade perto do ponto $\theta = 0°$ devido à pequena distância da fonte sonora. O raio da cabeça $a_h$ é um parâmetro que pode ser mudado a qualquer momento na execução das funções, mas é preferido manter este valor fixo em $8cm$ para os testes[13]. Usa-se, também, $F_s = 48kHz$ – taxa de amostragem comum para placas de som. Vale notar que os "degraus" observados na saída são devidos à função $round$ usada para obter o número de amostras. Como a amplitude da onda fica em no máximo 30 amostras, podemos deduzir que a máxima ITD com este algoritmo para os parâmetros fixados é de aproximadamente $625\mu s$.

```
function u = itdDelay ( a, D, theta, Fs )
  % a = head radius
  % D = source distance
  % theta = azimuth angle

  % thetaR = azimuth angle in radians
  % c = speed of sound

  % problem symmetry for theta > 90 degrees:
  if abs(theta) > 90
      theta = sign(theta) * (180 - abs(theta));
  end

  thetaR = theta * (pi/180);
  c = 343;

  % L = distance from source to tangent point
  L = sqrt(D^2  - a^2);

  % phi = angle from source direction to tangent point (
      simetrical )
  phi = acos( a/D );

  % AL = angle from source direction to left ear
  % AR = angle from source direction to right ear
  AL = pi/2 + thetaR;
  AR = pi/2 - thetaR;

  % first position is left ear, second position is right ear
  A = [AL AR];

  % T = tangent angles from source to tangent points
  T = A - phi;

  % DE = arc distance from tangent point to ear
  DE = T * a;


  % DA = Direct path distance

  DA = [ sqrt(D^2 + a^2 + 2*D*a*sin(thetaR)) , sqrt( D^2 + a
      ^2 - 2*D*a*sin(thetaR) ) ];

  %DL = distance to left ear
  %DR = distance to right earl =
```

```
44
45    if AL - phi < 0
46        DL = DA(1);
47    else
48         DL = DE(1) + L;
49    end
50
51
52    if AR - phi < 0
53        DR = DA(2);
54    else
55        DR = DE(2) + L;
56    end
57
58    % ITD = interneural time difference
59    ITD = ( DL - DR ) / c;
60
61    % u = interneural time difference, in samples
62
63    % negative delay menans the sound is on the right side of
          the head
64    % positive delay means the sound is on the left side of the
          head
65    u = round(ITD * Fs);
66
67
68 end
```

Listagem 4.1: (*itdDelay.m*) Implementação da diferença de tempo internaural

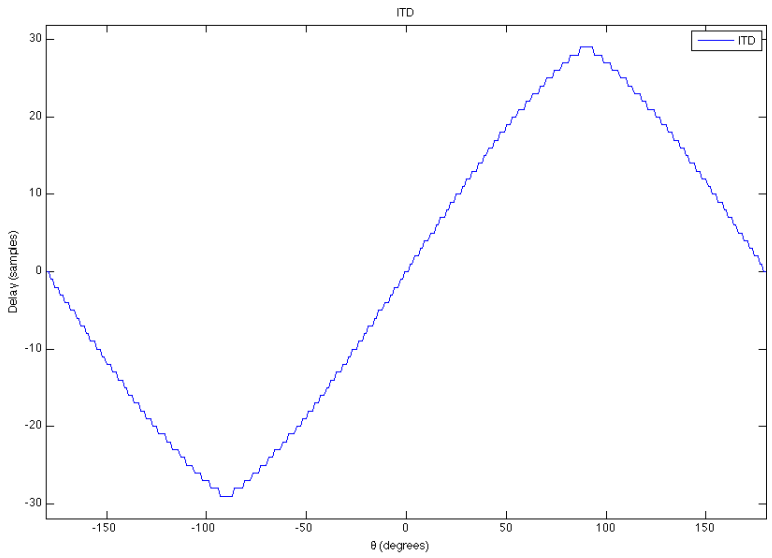



Figura 5: Forma da saída de *itdDelay* para $\theta = \{-180°, 180°\}$

## 4.2 ILD

Como vimos, a ILD representa a diferença de energia com que o som chega em cada ouvido. O maior fator para esta diferença de energia é a difração em torno da cabeça. Nos

apropriamos do modelo proposto por Brown em, [17], baseado na solução da difração para uma cabeça esférica pelo sombreamento da cabeça (HS) proposta por Lord Rayleigh [34], ilustrada na figura 6. Modela-se o filtro HS com um simples modelo de fase-mínima. Um sistema é dito de fase-mínima se este e sua inversa são estáveis e causais[46]. Do modelo proposto por Brown, temos:

$$H_{HS}(\omega, \theta) = \frac{1 + j\frac{\alpha\omega}{2\omega_0}}{1 + j\frac{\omega}{2\omega_0}} \tag{15}$$

onde $\omega_0$ está relacionado ao raio da esfera por

$$\omega_0 = \frac{c}{a_h} \tag{16}$$

e o parâmetro $\alpha$ localiza o zero do filtro e depende do ângulo azimutal $\theta$:

$$\alpha(\theta) = \left(1 + \frac{\alpha_{min}}{2}\right) + \left(1 - \frac{\alpha_{min}}{2}\right) \cdot \cos\left(\pi\frac{\theta}{\theta_0}\right) \tag{17}$$

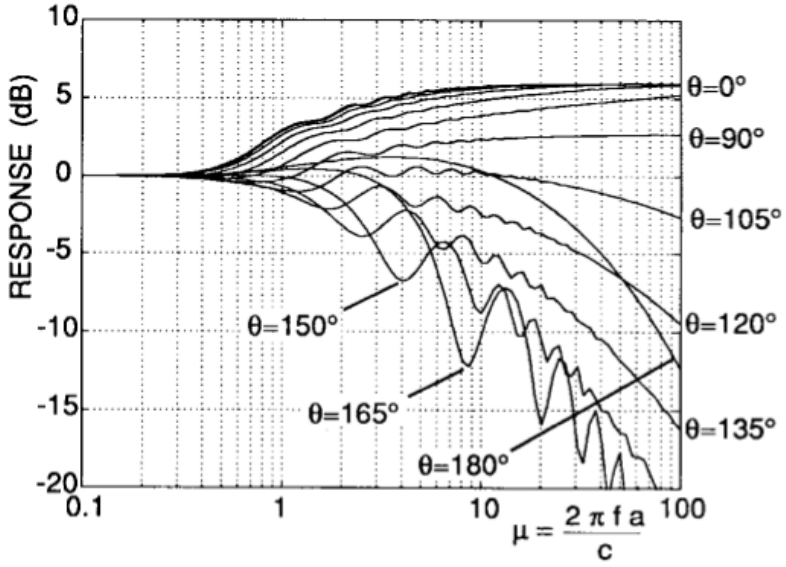



Figura 6: Efeito do sombreamento da cabeça de acordo com Rayleigh[34]

O parâmetro $\theta_0$ indica o ângulo onde o ganho será mínimo e o parâmetro $\alpha_{min}$ controla o ganho mínimo. Em [17], os valores usados são $\alpha_{min} = 0.1$ e $\theta_0 = 150°$. No entanto, ao verificar o valor de $\theta_0$, nota-se que com o valor de $150°$ cria-se uma assimetria nas respostas quanto ao plano médio. Foi verificado que essa assimetria gera inversão dos ângulos para $\theta > 150°$. Visto que este ângulo se situa atrás da cabeça – que já é uma zona ambígua na localização – preferi fixar $\theta_0 = 180°$. Esta decisão afasta o modelo simplificado daquele proposto por Lord Rayleigh em [34], no entanto, elimina o dito ponto de inversão e, veremos adiante, possibilita uma resposta contínua para qualquer $\theta$, o que é desejável.

Em uma nota relacionada, note que o filtro $H_{HS}$ proposto depende de dois parâmetros: $\omega$ e $\theta$. Isto implica que a resposta do sistema não depende apenas da frequência do sinal, mas também de outro parâmetro externo – esse variável no tempo. Neste ponto, o sistema

deixa de ser LTI (linear e invariante no tempo), o que eliminaria as hipóteses e a carga matemática de sistemas e sinais usuais. No entanto, doravante diremos que o parâmetro $\theta$ varia com frequências muito menores que a dos sinais de entrada do sistema e podemos, assim, considerar este como sendo LTI em espaços de tempo. Feita esta observação, retornamos à equação 15.

O sistema proposto está definido para o domínio de tempo contínuo. Logo, faz-se necessário uma transposição deste filtro para o domínio de tempo discreto empregando a transformada Z. Assumindo $j\omega = s$, da transformada de Laplace, temos que[46]

$$s = j\omega = \frac{2}{T_s}\frac{Z-1}{Z+1} \tag{18}$$

onde $T_s = \frac{1}{F_s}$ é o período de amostragem. Aplicando a equação 18 na equação 15 temos, já na forma normalizada:

$$H_{HS}(Z,\theta) = \frac{\left(\frac{\alpha(\theta)+\mu}{1+\mu}\right) + \left(\frac{\mu-\alpha(\theta)}{1+\mu}\right)Z^{-1}}{1+\left(\frac{\mu-1}{1+\mu}\right)Z^{-1}} \tag{19}$$

onde $\mu = \frac{\omega_0}{F_s} = \frac{c}{a_h \cdot F_s}$.

Podemos, então, aplicar este filtro com uma decalagem de ângulo para cada orelha. Novamente, a literatura propõe que as orelhas não estão nos ângulos azimutais de $90°$, mas sim mais próximas de $110°$. Preferi, no entanto, manter a proposta intuitiva e fixar $\Theta_r = -90°$ e $\Theta_l = 90°$, optando por maior facilidade na leitura das respostas. Logo, sendo $H^l_{HS}$ o filtro aplicado à orelha esquerda e $H^r_{HS}$ o aplicado à orelha direita. Temos:

$$H^l_{HS}(Z,\theta) = H_{HS}(Z,\theta-\Theta_l) \tag{20}$$
$$H^r_{HS}(Z,\theta) = H_{HS}(Z,\theta-\Theta_r) \tag{21}$$

Passamos adiante à implementação em código Matlab do algoritmo de ILD proposto. O código da função *ildFilter* é mostrado na listagem 4.2. Faz-se uso da função *filter* disponível com a ferramenta, que toma como argumentos os coeficientes dos polinômios de numerador e denominador da função de transferência no domínio de tempo discreto. A figura 7 mostra o resultado para $\theta = \{0°, 180°\}$ desconsiderando a decalagem de orelha. Nota-se, nesta figura, uma semelhança com a figura 6, como desejado até ângulos próximos de $150°$, quando a simplificação do modelo muda a resposta. Na figura 8 temos a comparação de $H^l_{HS}$ e $H^r_{HS}$ para o mesmo intervalo de $\theta$.

```
1  function  u = ildFilter( a , theta , ear , input, Fs)
2      % a is the head radius
3      % theta is the azimuth angle, in degrees
4      % ear is an integer for offset to ( ear > 0 = left ear ),
5      %                                 ( ear < 0 = right ear )
       ,
6      %                                 ( ear = 0 = no offset)
7      % input is the signal to be filtered
8      % Fs is the sampling Frequency
9
10     % c is the speed of sound
11     c = 343;
12
13     %earPos is the ear position in the azimuth plane
```

```
14      earPos = 90;
15      thetaEar = earPos* sign(ear);
16
17      theta = theta - thetaEar;
18
19
20      % theta0 is the angle with minimum gain
21      theta0 = 180;
22
23
24      %alfa_min dictates gain
25
26      alfa_min = 0.1;
27
28
29      %w0 is the frequency relation for the head size
30      w0 = c/a;
31
32
33      alfa = 1+ alfa_min/2 + (1- alfa_min/2)* cos(theta/ theta0
            * pi) ;
34      mu = w0/Fs;
35
36      B = [(alfa+mu)/(1+mu), (-alfa+mu)/(1+mu)] ;
37          % numerator of Transfer Function
38      A = [1, -(1-mu)/(1+mu)] ;
39          % denominator of Transfer Function
40
41      u = filter(B, A, input);
42
43
44
45  end
```

Listagem 4.2: (*ildFilter.m*) Implementação dos filtros de fase-mínima para ILD

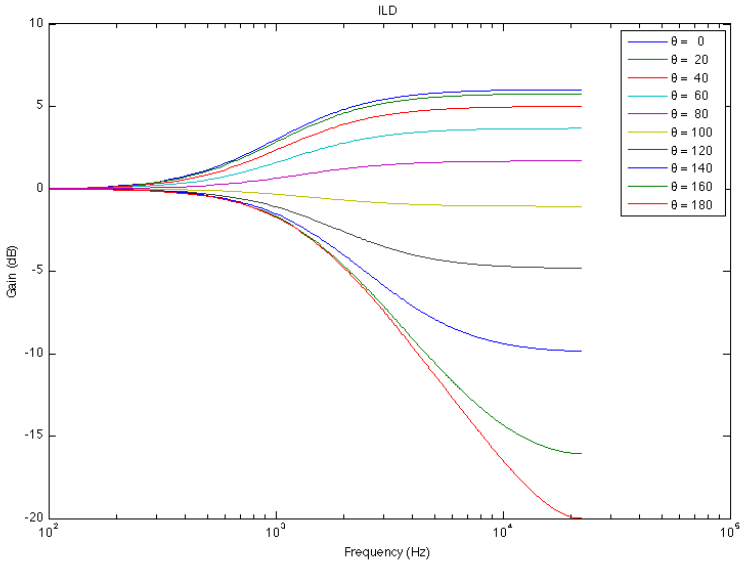



Figura 7: Resultado do filtro de HS proposto para $\theta = \{0°, 180°\}$

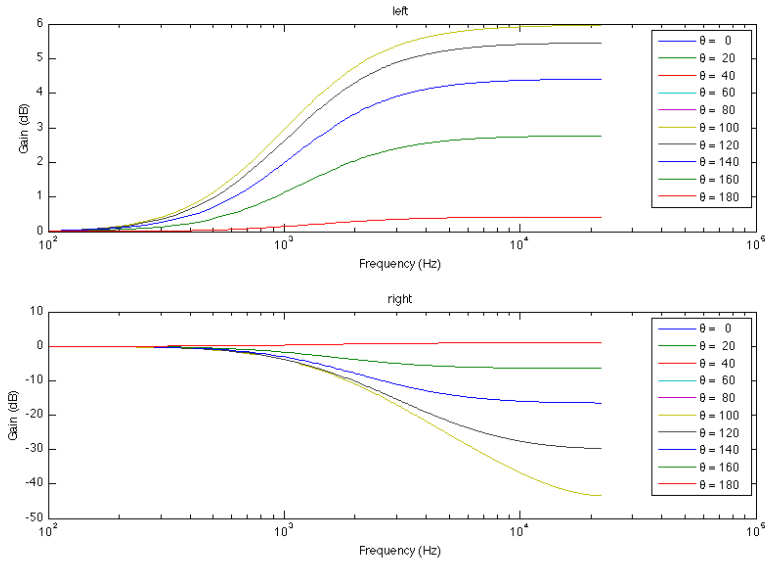
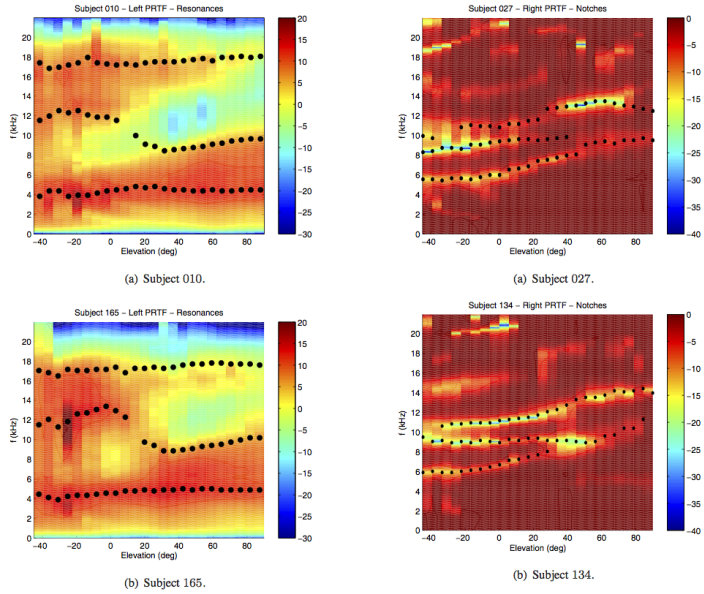



Figura 8: Comparação de $H^l_{HS}$ e $H^r_{HS}$ propostos para $\theta = \{0°, 180°\}$

## 4.3 PRTF

Para acrescentar um tom de realidade à resposta dos filtros, adicionamos os efeitos da pina – parte externa da orelha. Como discutido anteriormente, acredita-se que a pina tenha uma contribuição maior na resposta em frequência do sinal que na sua resposta no tempo. Por esta razão, abandono o modelo de Brown nesta parte, apesar de a implementação proposta se assemelhar fortemente à citada por ele. A pina cria uma série de reflexões da onda sonora, mudando sua "cor". Estas reflexões são dependentes da forma da pina e variam significativamente de uma pessoa para outra. No entanto, Spagnol Et Al., em [35], usam algoritmos para detetar respostas espectrais comuns das PRTFs. O artigo propõe, então, um modelo em filtros de tipo ressonância e *notch* para simular as zonas de interferências positivas e negativas. Os resultados dos algoritmos preditores são mostrados na figura 9, onde os pontos pretos mostram os caminhos de maior influência. Note que o objetivo das linhas de predição é encontrar as respostas para diferentes elevações. O único ponto de interesse para a aplicação em discussão é o ponto de elevação $\phi = 0°$. Um exemplo dos modelos tratados por Spagnol para uma elevação de $11°$ é mostrado na figura 10.




(a) Subject 010. (a) Subject 027.

(b) Subject 165. (b) Subject 134.

Figura 9: Predição dos filtros de ressonância e *notch* por Spagnol[35]

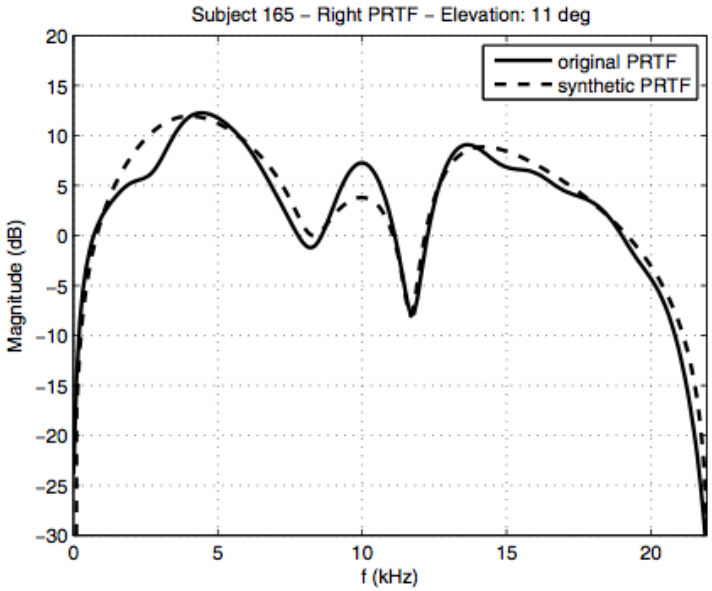



Figura 10: Exemplo do resultado da predição por Spagnol a $\phi = 11°$ [35]

Com base nestes dados, o grupo propõe um modelo estrutural ilustrado na figura 11.

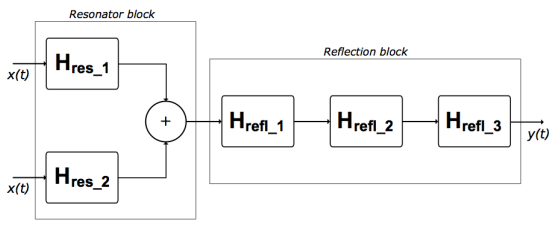



Figura 11: Modelo estrutural para PRTF proposto por Spagnol[35]

Onde $H_{res}$ são filtros de ressonância e $H_{refl}$ são filtros de reflexão tipo *notch*. Os filtros de ressonância são modelados na forma:

$$H_{res}(Z) = \frac{V_0(1-h)(1-Z^{-2})}{1+2dhZ^{-1}+(2h-1)Z^{-2}} \tag{22}$$

onde

$$h = \frac{1}{1+\tan\left(\pi\frac{f_B}{f_C}\right)} \tag{23}$$

$$d = -\cos\left(2\pi\frac{f_C}{F_s}\right) \tag{24}$$

$$V_0 = 10^{\frac{G}{20}} \tag{25}$$

tal que $f_C$ é a frequência central da ressonância e $G$ seu ganho em dB retirados a partir das linhas pontilhadas pretas da figura 9 para ressonância. $f_B$ é a largura de banda fixada em $5kHz$.

Os filtros de reflexão são modelados na forma:

$$H_{refl}(Z) = \frac{1+(1+k)\frac{H_0}{2}+d(1-k)Z^{-1}+(-k-(1+k)\frac{H_0}{2})Z^{-2}}{1+d(1-k)Z^{-1}-kZ^{-2}} \tag{26}$$

onde

$$V_0 = 10^{-\frac{G}{20}} \tag{27}$$

$$H_0 = V_0 - 1 \tag{28}$$

$$d = -\cos\left(2\pi\frac{f_C}{F_s}\right) \tag{29}$$

$$k = \frac{\tan\left(\pi\frac{f_B}{F_s}\right)-V_0}{\tan\left(\pi\frac{f_B}{F_s}\right)+V_0} \tag{30}$$

tal que $f_C$ é a frequência central da reflexão destrutiva e $G$ é a profundidade da reflexão em dB retirados a partir das linhas pontilhadas pretas da figura 9 para reflexão. $f_B$ é a largura de banda fixada em $1kHz$.

Seguindo o modelo proposto na figura 11, montamos a tabela 3 para retirar os coeficientes necessários para cada reflexão e ressonância.

Tabela 3: Coeficientes para os filtros de PRTF

| **Número** | **Tipo** | $f_C$ [kHz] | $G$ [dB] |
|---|---|---|---|
| 1 | *res* | 4 | 5 |
| 2 | *res* | 12 | 5 |
| 3 | *refl* | 6 | 5 |
| 4 | *refl* | 9 | 5 |
| 5 | *refl* | 11 | 5 |

A listagem 4.3 mostra a implementação em código Matlab dos filtros, usando a função *filter* do Matlab novamente. Traçando a resposta deste filtro no domínio da frequência,

obtemos o mostrado na figura 12. Observe na figura os dois picos localizados nas frequências de ressonância especificados e os três vales nas frequências de reflexão. Compare com a figura 10 e observamos nitidamente as semelhanças. Note, também, que o filtro da PRTF escolhido é insensível para o ângulo azimutal $\theta$, supondo que a PRTF muda apenas com a elevação. Este modelo pode ser usado para dar mais vida espectral à resposta, mas do modo como foi implementado não acrescenta sensação de elevação. Supõe-se que a pina age de forma *monoaural* em que os resultados dos outros filtros passam separadamente (para orelha esquerda e direita) por este filtro.

```
function u = prtfFilter( input, Fs )
    %input is the signal to process
    %Fs is the sampling frequency

    fC = [4e3 12e3 6e3 9e3 11e3]; %resonance and notch center
         freq;
    G = 5; %resonance and notch magnitude ;

    u1 = prtfRes ( input, Fs, fC(1), G); %1st resonance
    u2 = prtfRes (input, Fs, fC(2), G);  %2nd resonance

    u = u1+u2; % addition block


    u = prtfRefl( u , Fs , fC(3), G ); %1st notch
    u = prtfRefl( u , Fs , fC(4), G ); %2nd notch
    u = prtfRefl( u , Fs , fC(5), G  ); %3rd notch
    function u = prtfRes( input , Fs, fC , G) % resonance
         filter
        fB = 5e3; % fixed bandiwidth 5kHz
        h = 1/(1+ tan(pi*fB/Fs));
        d = -1*cos(2*pi*fC/Fs);
        V0 = 10 ^ (G/20);

        B = (V0*(1-h))*[1 0 -1];
        A = [1 2*d*h (2*h-1)];

        u = filter( B, A, input );
    end

    function u = prtfRefl( input, Fs, fC,  G ) %reflexion
         filter
        fB = 1e3; % fixed bandiwidth 1kHz
        V0 = 10^(-G/20);
        H0 = V0-1;
        d = -cos(2*pi*fC/Fs);
        k = (tan(pi*fB/Fs) - V0)/( tan(pi*fB/Fs) + V0);
        B = [(1+(1+k)*H0/2) d*(1-k) (-k-(1+k)*H0/2)];
        A = [1 d*(1-k) -k];

        u = filter(B, A, input);
    end

end
```

Listagem 4.3: (*prtfFilter.m*) Implementação dos filtros de ressonância e reflexão da PRTF

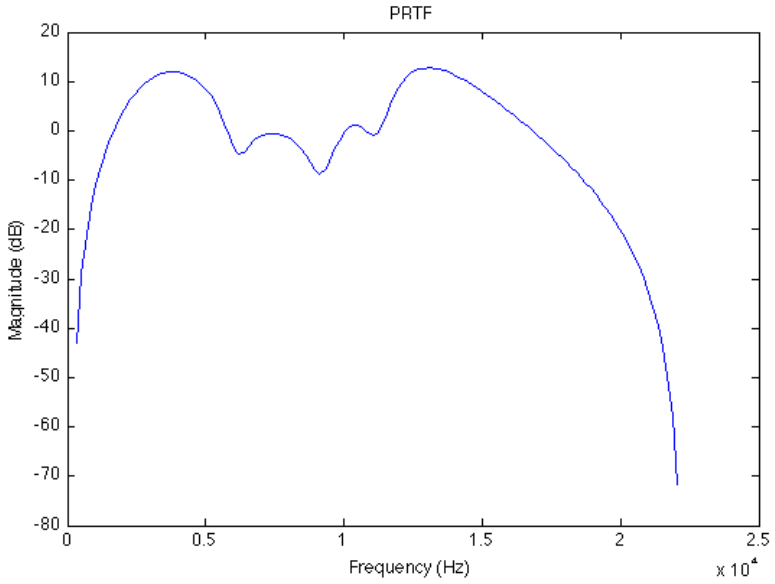



Figura 12: Resultado do filtro usado para PRTF

## 4.4 RIR

Por último, para eliminar o efeito de lateralização discutido anteriormente, podemos adicionar um filtro que modele a resposta impulsiva da sala. Pela simplicidade do modelo proposto, seguimos o modelo de Brown anteriormente citado, onde temos apenas um eco não tratado e somado a cada sinal de saída (esquerdo e direito) que melhora fortemente a sensação de espacialização. Para isto, criamos apenas um filtro com atenuação de 15dB e um atraso no tempo de 15ms. Verificou-se que aplicando um filtro passa-baixas de um pólo em 10kHz tornou a resposta mais agradável. A listagem 4.4 mostra o código Matlab para a RIR escolhida.

```
1  function u = roomEcho ( input, Fs)
2    % input is the original monoaural sound
3    % Fs is the sampling frequency
4
5    %x is the time delay (15ms)
6    %k is the gain (-15dB)
7     x = 15e-3;
8     K = -10^(-15/20);
9
10    %delay in samples
11    T = round(x * Fs);
12
13    %apply filter
14    u = K*delay(input, T);
15
16    %low pass to eliminate unpleasent high frequency reverb.
17    u = lowPass(u, Fs, 10e3,1);
18
19 end
```

Listagem 4.4: (*roomEcho.m*) Eco simples da sala para eliminar a lateralização

## 4.5 Modelo Completo

Juntando todos os filtros criados, chegamos a um resultado assaz satisfatório de espacialização no plano azimutal. O diagrama de blocos da figura 13 mostra o sistema final. Para aplicar todos os efeitos, criamos uma função que aplica os filtros na ordem dos blocos da figura 13 e analisamos as respostas temporais e espectrais do impulso na entrada.

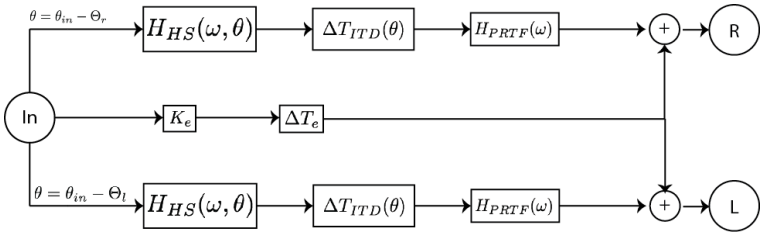



Figura 13: Diagrama de blocos final

```
function u = spacialise( r, a , theta, input , Fs, mkecho)

    % r is source distance
    % a is head radius
    % theta is the azimuth angle in degrees
    % input is the input signal
    % Fs is the sampling frequency
    % mk echo = boolean to add room echo


    %BNS is the binaural signal


    %echo is the delayed and attenuated input -- the RIR
    echo = roomEcho( input, Fs);


    % apply ILD filter with arg 1 for left ear and arg -1 for
          right ear
    BNS = [ ildFilter( a, theta, 1, input , Fs ) ildFilter(
          a, theta, -1, input , Fs ) ];


    % apply binaural delay ( uses ILD delay )
    BNS = binauralDelay( BNS, Fs, a , r , theta );

    % apply Pinna-related effects
    BNS = prtfFilter( BNS  , Fs );

    L = BNS(:,1);
    R = BNS(:,2);

    % prepare output;
    u = [ L R ];

    %if mkecho is TRUE
    if( mkecho )
```

```
36          u = u + [echo echo]; %add the same echo to both output
                channels
37      end
38
39
40  end
```

Listagem 4.5: (*spacialise.m*) Função *spacialise* que aplica todos os filtros no som de entrada

Com a função *spatialise* da listagem 4.5 podemos testar o processamento de som pelos filtros implementados e ouvir o resultado usando entradas e saídas de arquivos *wav* no Matlab. Seguindo os algoritmos de [37], traçamos a resposta impulsiva do sistema completo (sem a RIR) para validar a resposta contra respostas conhecidas. Primeiramente, mostramos a resposta no tempo para os canais esquerdo e direito. Em um gráfico polar temos que o raio é o tempo (onde o centro do círculo é $t = 0$) e o ângulo é o ângulo azimutal. Podemos notar na figura 14 que as respostas não possuem descontinuidades e são consistentemente simétricas se compararmos os dois canais de saída. Note, também, que a parte significativa da resposta impulsiva dura aproximadamente 1.5ms ou 66 amostras a 44.1kHz. Temos uma resposta imediata e forte, relacionada aos efeitos do *Head Shadow*, o distanciamento da resposta conforme $\theta$ varia é dado principalmente pela ITD e temos os vários picos de menor energia, que são contribuição dos reflexos da pina. Se considerarmos o eco deste gráfico, lembramos que ele leva um atraso de 15ms da entrada, logo está 75 vezes mais atrasado do que o pico de mais energia. Este seria um indicativo de que a resposta da sala faz parte de uma lista de *cues* diferente da HRTF e por isso não é incluído nesta verificação.

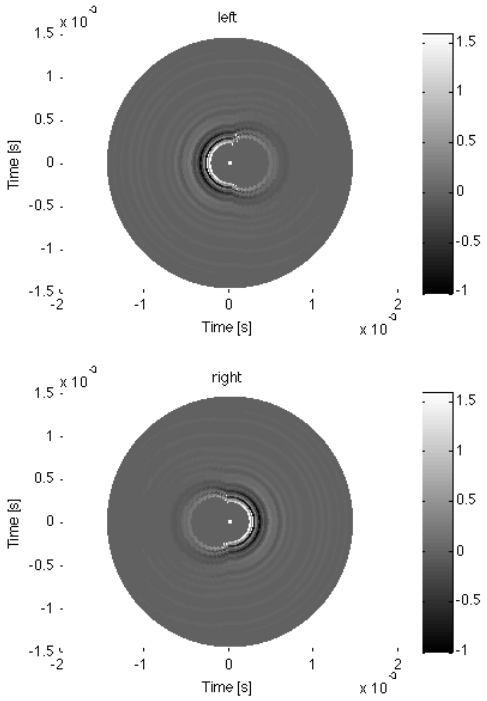



Figura 14: Resposta no tempo em coordenadas polares

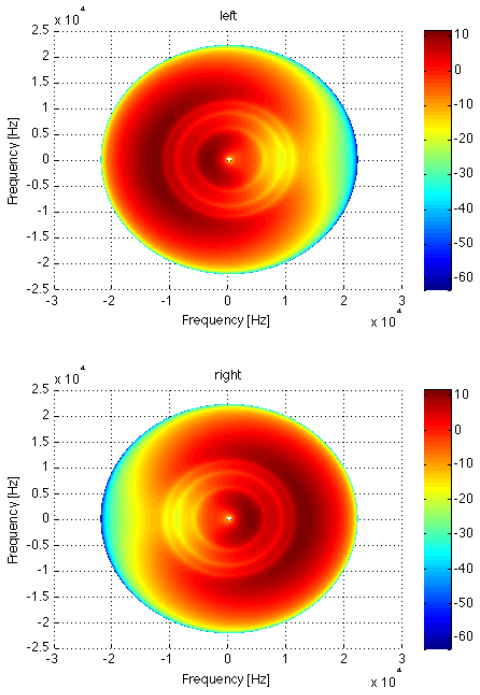



Figura 15: Resposta em frequência em coordenadas polares

# 5 CONFIGURACÃO DO SISTEMA

Neste capítulo, veremos a configuração do ambiente de trabalho, a escolha da plataforma embarcada utilizada e a determinação de bibliotecas utilizadas na implementação. Inesperadamente, esta etapa do projeto durou mais do que o inicialmente planejado, pois houve um longo período de adaptação à tecnologias até então inéditas. Como veremos, sempre que possível, soluções *Open Source* foram preferidas.

## 5.1 Placa de Desenvolvimento

Para embarcar o sistema, foi escolhida a plataforma de prototipação *Open Hardware* chamada *Beagleboard* [11]. A revisão da placa utilizada foi a *Beagleboard-xM revC* cujas características são:

- Super-scalar ARM Cortex TM -A8
- 512-MB LPDDR RAM
- High-speed USB 2.0 OTG port optionally powers the board
- On-board four-port high-speed USB 2.0 hub with 10/100 Ethernet
- DVI-D (digital computer monitors and HDTVs)
- S-video (TV out)
- Stereo audio out/in
- High-capacity microSD slot and 4-GB microSD card
- JTAG
- Camera port

O processador em questão se trata de um *DM3730* da *Texas Instruments*[2]. Este processador conta com um ARM com clock de 1GHz mais um DSP da família DaVinci com clock de 800MHz. Trata-se de um chip muito utilizado para tecnologias embarcadas de alto processamento e multimídia como telefones celulares de última geração. É um chip relativo aos da família OMAP (*Open Multimedia Application Platform*), porém com processamento de ponto-fixo.

### 5.1.1 Sistema Operacional

A *Beagleboard* dispõe de um leitor de cartões SD que pode ser usado como sistema de arquivos para o sistema operacional. Assim, não é necessário programar a placa pelo dispositivo JTAG, podemos programar diretamente dentro de um sistema operacional que é armazenado no cartão. Um *bootloader* chamado "U-Boot" vem pré-gravado na ROM do processador e podemos controlar suas diretivas de lançamento através de arquivos texto de configuração. Seguindo exemplos do site [11], configurei o *bootloader* (ver apêndice D) para carregar o sistema operacional *Angstrom*, uma distribuição gratuita de Linux embarcado. Esta *distro* dispõe de um website com montador de pacotes onde se pode descarregar de uma vez só o ambiente de desenvolvimento para a plataforma assim como o kernel e utilitários pré-compilados. Criei uma configuração mínima para o sistema operacional através deste sistema, chamado *Narcissus* [7], removendo todos os pacotes que não seriam necessários para a aplicação em questão, como o cliente de ambiente gráfico do Linux, o "X".

### 5.1.2 Preparação para Co-Processamento

Tratando-se de uma aplicação de alto nível de processamento e de cunho de processamento de sinais, parece intuitivo fazer uso do DSP contido na placa. Porém, para isto, foi necessário prever a preparação do sistema operacional para ativar este segundo processador. A distribuição *Angstrom* faz uso apenas do núcleo principal (ARM) e teríamos todo o poder de processamento do segundo núcleo (DSP) para processar o áudio. A *TI* disponibiliza dois módulos do kernel fundamentais para esta comunicação entre processadores:

- *CMEM* : responsável pela criação de um *pool* virtual de memória compartilhado entre os dois processadores;
- *DSPLINK* : responsável pelas caixas de mensagens (*mailbox*) utilizadas para as trocas de mensagens;

Adicionalmente, foi inserida uma diretiva de *boot* para limitar a memória utilizada pelo sistema operacional, deixando uma parte livre para uso comum para o co-processamento.

## 5.2 Compiladores

Naturalmente, não se pode criar os binários desta apliação com qualquer compilador. Para tanto foram necessários dois compiladores diferentes: um para o núcleo ARM e um segundo para o DSP. O compilador do ARM usado foi o distribuído pelo *Narcissus* chamado *arm-gcc* ("*ARM GNU C Compiler*"). No entanto, a escolha de compilador para o DSP não é tão direta. Tratando-se de um projeto de duração relativamente curta, desenvolver toda a biblioteca de gerenciamento de troca de mensagens entre os processadores pareceu inviável. Logo, recorri à uma ferramenta distribuída gratuitamente pela *TI* chamada *C6EZRun*[1].

A biblioteca *C6EZRun* disponibiliza um compilador integrado ao compilador do lado ARM para criar aplicações que podem ser lançadas normalmente como qualquer binário do sistema operacional, mas esta usa os módulos do kernel citados anteriormente para executar funções no lado do DSP. Esta ferramenta se divide em dois sabores: *C6RunApp*, que permite fazer uma aplicação completa processada pelo DSP; e a *C6RunLib*, que permite a criação de um arquivo ".lib" que constitui uma biblioteca de funções como qualquer

outra. Mas todas elas são operadas pelo DSP. Optei pela segunda pois, como veremos adiante, parte do processamento deve obrigatoriamente ser feito pelo núcleo principal. Para colocar esta ferramenta em operação são necessários alguns módulos adicionais, em especial[1]:

- *TI-CGT* (*TI Code Generator Tools*): compilador em si para o DSP;
- *DSP-BIOS*: sistema operacional construído especialmente para DSPs da *TI*.

Com estes compiladores em mãos, escrevi um *Makefile* que distribui os arquivos de código fonte entre os dois, dependendo do prefixo deste. Arquivos com prefixo "arm_" são enviados para o compilador *arm-gcc*, enquanto aqueles com prefixo "dsp_" são enviados para o compilador *c6runlib-cc*. Ao final de tudo, o *arm-gcc* entra novamente, mas desta vez apenas juntando todas as partes compiladas – atuando como *linker*. Diretivas de pré-compilação foram criadas para possibilitar compilar todo o projeto apenas para o núcleo principal (facilitando o processo de depuragem) assim como desabilitar algumas bibliotecas. Uma nota importante: ambos os compiladores citados são binários para máquinas Linux-i386, veremos na próxima seção como a instalação na placa é feita.

## 5.3 Ambiente de Desenvolvimento

Nesta seção veremos como se fez o processo de desenvolvimento, os softwares utilizados, a instalação na placa e os padrões de escrita de código utilizados.

### 5.3.1 Softwares de Apoio

Para desenvolver a aplicação foram usados os seguintes softwares:

- VIM: editor de texto em linha de comando com grande extensibilidade e personalização através de uma linguagem própria[8];
- GIT: controle de subversão utilizado em larga escala em softwares *Open Source*[5];
- *Doxygen*: padrão de documentação de código e gerador de documentação automático[3][2].

### 5.3.2 Distribuição de Tarefas

Tratando-se de vários binários compilados para diferentes arquiteturas, foi necessário o uso de uma máquina virtual com Ubuntu i386 para compilar o código. A figura 16 ilustra como foram distribuídas as tarefas entre as diferentes máquinas utilizadas, compreendendo o fato de que parte do desenvolvimento foi feito fora do local físico destas máquinas usando um laptop. Toda a comunicação entre as máquinas é feita por SSH (túneis representados por linhas azuis na figura), inclusive a instalação na placa. Em resumo, o trabalho seguiu os seguintes passos:

1. Desenvolvimento na máquina principal (iMac);
2. Enviar código para máquina virtual Ubuntu[SSH];

[1] Recorrer ao apêndice C para as versões específicas dos pacotes utilizados.

[2] A geração de documentação automática pode ser feita em páginas HTML ou em LaTeX, no apêndice B temos um exemplo de documentação de função gerada automaticamente.

3. Compilar o código, gerar documentação[make+doxygen];
4. Enviar o binário para a placa, enviar a documentação de volta para a máquina principa[SSH]l;
5. Publicar a documentação pela internet na máquina principal[HTTP];
6. Testar a aplicação na placa[SSH].

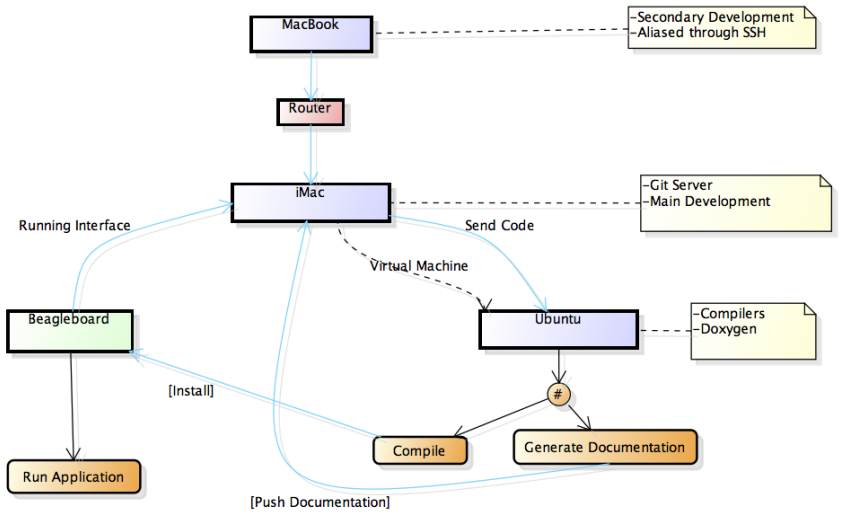



Figura 16: Descrição da distribuição de tarefas entre as diferentes máquinas

### 5.3.3 Bibliotecas

Ao desenvolver software, é uma boa prática informar-se das bibliotecas disponíveis (principalmente as em *Open Source*) para evitar o desperdício de tempo ao reescrever grandes funcionalidades que são de uso geral. Nesta seção, descrevemos brevemente as bibliotecas utilizadas na implementação.

#### *5.3.3.1 JACK Audio*

O *JACK Audio*, ou simplesmente *JACK*[6], é um conjunto de ferramentas que compreendem um servidor de som para o sistema operacional e uma API (*Application Programming Interface*) para interagir com as interfaces de áudio de um sistema de maneira fácil e rápida. A maior vantagem desse sistema é a integração fácil com plataformas Linux e drivers de som ALSA (*Advanced Linux Sound Architecture*). A ferramenta permite a obtenção e o envio de pacotes de áudio em baixa latência e processamento de tempo real, a solução ideal para este projeto.

#### *5.3.3.2 NCurses*

O *NCurses*[39], ou *New Curses*, é uma biblioteca que perimite a criação de interfaces de usuário em console que "imitam" o comportamento de uma interface em um ambiente gráfico trabalhando com *buffers* de texto que são trocados constantemente. Dentre as capacidades da ferramenta, estão a criação de pseudojanelas, a reconstrução periódica da tela e a interação com o usuário. Visto que o objetivo do projeto não foi o de criar uma interface completa e agradável, mas apenas uma comprovação de algoritmos, esta ferramenta permite a criação de interfaces informativas e de rápida construção.

#### *5.3.3.3 Posix Threads*

Todo programa que propõe processamento em tempo real deve utilizar o conceito de *threads*. Para implementar as tarefas do programa, foram utilizadas as chamadas *Posix Threads*[40], ou *PThreads*, uma biblioteca de gerenciamento de tarefas extremamente difundida, de código aberto e largamente documentada. As funções disponíveis foram utilizadas para disponibilizar tarefas simultâneas assim como a sincronização e troca de mensagens entre elas.

#### *5.3.3.4 DSPLIB*

Complementarmente a todas as ferramentas já citadas que são oferecidas pela *Texas Instruments*, adicionamos à lista mais uma: *DSPLIB*[4], que consiste em uma biblioteca de funções matemáticas comuns a diferentes algoritmos de DSP implementadas de maneira otimizada para diversas versões de processadores da *TI*. Visto que o DSP utilizado é um ponto-fixo, a versão da biblioteca empregada no projeto é a compatível com processadores C64x e possui operações em ponto fixo implementadas em *assembler*. As funções de rápido processamento disponibilizadas são, no entanto, para registradores de 16 bits com codificação Q15 (ou *Rational 15-bits*), o que quer dizer que os números tratados são expressados em números inteiros, representando um ponto flutuante de $0.0$ a $1.0$ discretizados em $2^15$ valores assim como seus negativos (o décimo-sexto bit representa o sinal). Felizmente, a bilbioteca traz consigo funções de conversão de ponto-flutuante normatizado (módulo inferior a $1.0$) para Q15 e vice-versa. Além das funções de conversão, a maior utilidade da biblioteca para o projeto foram as funções de filtros de resposta impulsiva finita (FIR).

# 6 DESENVOLVIMENTO DA APLICAÇÃO

Enfim, podemos passar para a descrição de como a aplicação em si foi pensada para avaliar a realização de sons binaurais. Inicialmente, vemos os pré-requisitos estabelecidos para o programa – já adiantando os que não foram implementados. Em seguida, uma análise de como passar todo o estudo do capítulo 4 para código de máquina. E, finalmente, uma descrição completa de como foi feita a arquitetura do programa, a comunicação entre tarefas e o mais importante: a garantia do tempo real.

## 6.1 Requisitos

A aplicação foi pensada para ser capaz de cumprir as tarefas a seguir. Os itens marcados com [NOK] não foram implementados e seguem com justificativa. O programa deve:

- possuir uma interface atualizada periodicamente, dando informações sobre os parâmetros utilizados;
- ter a possibilidade de modificar alguns parâmetros em tempo de execução:
  - Ângulo azimutal ($\theta$);
  - Volume;
  - Tipo de entrada (ruído, tom, arquivo *wave*, microfone [NOK] [1]).
- ter possibilidade de ler um arquivo de configuração para os parâmetros estáticos:
  - Parâmetros da fonte sonora;
  - Parâmetros do filtro de eco;
  - Parâmetros do filtro PRTF [NOK] [2];
  - Informação do arquivo *wave* que deve ser lido.
- implementar os quatro filtros descritos no capítulo 4 mais um filtro passa-baixas para eliminar alguns efeitos do processamento;
- baixo processamento para o núcleo principal quando compilado para o DSP;

[1] A entrada com microfone não foi implementada pois o arquivo *wave* foi o suficiente para a avaliação do algoritmo e a operação em tempo real não precisou desta entrada para ser comprovada.

[2] Os filtros da PRTF ficam extremamente complicados, como veremos na próxima seção, tornando os parâmetros destes desnecessários.

- permitir comandos para eliminar individualmente cada um dos filtros: ITD, ILD, PRTF, Eco e Passa-Baixas;

## 6.2 Transporte para Código de Máquina

Até agora, toda a validação de algoritmos foi feita no Matlab. No entanto, ao implementar um programa em código de máquina (C, neste caso), não dispomos da vasta biblioteca de funções do software matemático. Logo, já considerando a otimização para garantia de tempo real, analisamos cada um dos filtros para transformá-los em funções escritas em C.

### 6.2.1 ITD

A figura 17 mostra como é implementado o algoritmo da ITD. Temos um primeiro momento em que o filtro deve receber o novo ângulo azimutal e ajustar seu atraso no tempo caso esse tenha mudado desde a última operação de filtragem. A função de filtro em si que segue implementa uma memória circular que guarda todas as amostras que passaram e faz a decisão de qual canal deve ser atrasado a partir do sinal do atraso calculado. Então, a função escreve na saída o valor atrasado. A equação 31 descreve o comportamento do filtro no domínio discreto, onde $\Delta n_l$ e $\Delta n_r$ são os atrasos (em amostras) para cada um dos canais calculados a partir da equação 32. Lembrando que sempre que um for não nulo o outro o será.

$$\begin{bmatrix} y_l[n] \\ y_r[n] \end{bmatrix} = \begin{bmatrix} x_l[n - \Delta n_l] \\ x_r[n - \Delta n_r] \end{bmatrix} \tag{31}$$

$$\begin{aligned} \Delta n_l &= \begin{cases} |\Delta n| & , \quad \Delta n > 0 \\ 0 & , \quad \Delta n \leq 0 \end{cases} \\ \Delta n_r &= \begin{cases} |\Delta n| & , \quad \Delta n < 0 \\ 0 & , \quad \Delta n \geq 0 \end{cases} \end{aligned} \tag{32}$$

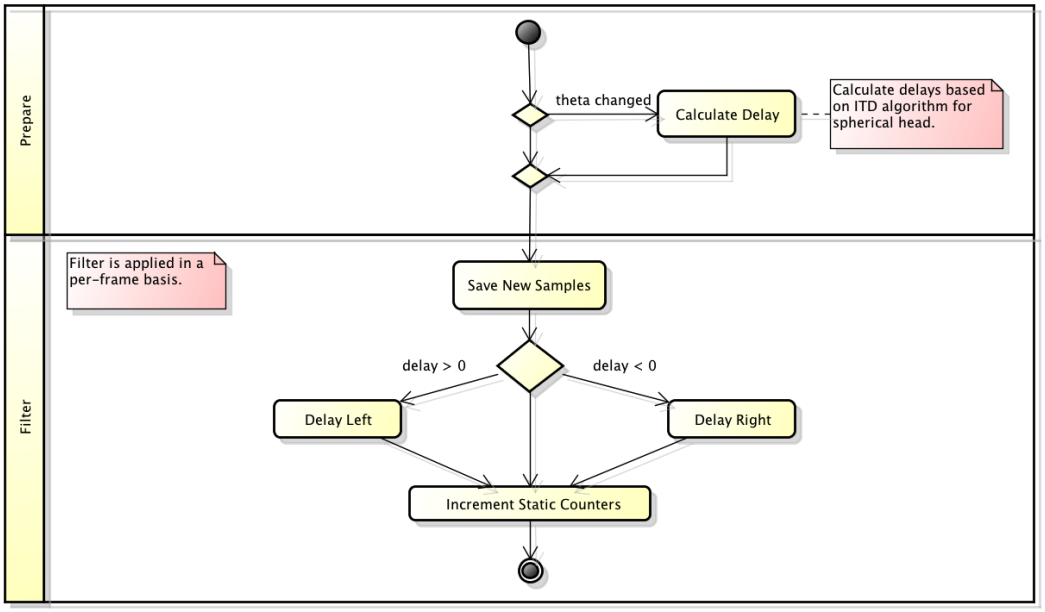



Figura 17: Fluxograma da operação do filtro de ITD

### 6.2.2 ILD

No capítulo 4, avaliamos a equação que rege o comportamento chamado *head shadowing*. No entanto, utilizamos funções prontas do Matlab para simular estes efeitos e no momento de passar para código de máquina não dispomos destas funções. Para garantir a linearidade dos filtros, podemos usar filtros FIR para implementar a ILD. Se relembrarmos o equacionamento do filtro, vemos que existe um zero que depende do ângulo azimutal, impossibilitando uma aproximação *hard-coded*. Faz-se necessário avaliar simbolicamente como se comportam os coeficientes da resposta impulsiva. Para isto, passamos as mesmas equações pelo Matlab, mas desta vez procurando uma resposta simbólica no domínio do tempo. Eis o que obtivemos:

$$\overrightarrow{h} = \begin{bmatrix} \frac{\alpha+\mu}{\mu+1} \\ -\frac{2\mu(\alpha-1)}{(\mu+1)^2} \\ \frac{2\mu(\alpha-1)(\mu-1)}{(\mu+1)^3} \\ -\frac{2\mu(\alpha-1)(\mu-1)^2}{(\mu+1)^4} \\ \frac{2\mu(\alpha-1)(\mu-1)^3}{(\mu+1)^5} \\ -\frac{2\mu(\alpha-1)(\mu-1)^4}{(\mu+1)^6} \\ \frac{2\mu(\alpha-1)(\mu-1)^5}{(\mu+1)^7} \\ -\frac{2\mu(\alpha-1)(\mu-1)^6}{(\mu+1)^8} \\ \frac{2\mu(\alpha-1)(\mu-1)^7}{(\mu+1)^9} \\ -\frac{2\mu(\alpha-1)(\mu-1)^8}{(\mu+1)^{10}} \\ \frac{2\mu(\alpha-1)(\mu-1)^9}{(\mu+1)^{11}} \\ -\frac{2\mu(\alpha-1)(\mu-1)^{10}}{(\mu+1)^{12}} \\ \frac{2\mu(\alpha-1)(\mu-1)^{11}}{(\mu+1)^{13}} \\ -\frac{2\mu(\alpha-1)(\mu-1)^{12}}{(\mu+1)^{14}} \\ \frac{2\mu(\alpha-1)(\mu-1)^{13}}{(\mu+1)^{15}} \\ -\frac{2\mu(\alpha-1)(\mu-1)^{14}}{(\mu+1)^{16}} \end{bmatrix} \tag{33}$$

O vetor da equação 33 apresenta uma simetria extremamente interessante se observado de perto. O primeiro termo destoa dos outros e podemos fixá-lo em

$$h[0] = \frac{\alpha+\mu}{\mu+1} \tag{34}$$

Observando os componentes que seguem, podemos definí-los iterativamente a partir do primeiro coeficiente:

$$h[1] = -\frac{2\mu\,(\alpha-1)}{(\mu+1)^2} \tag{35}$$

$$h[n] = h[n-1] \cdot \frac{(1-\mu)}{(\mu+1)} \tag{36}$$

É assim que os coeficientes da ILD são construídos. Um último detalhe deve ser posto em foco: pela natureza do filtro, o primeiro coeficiente será sempre o maior. Se o avaliarmos para o maior valor de $\alpha$ possível (ver equação 17) obtemos

$$h[0] = \frac{2+\mu}{\mu+1} \tag{37}$$

O que não seria um problema se não fosse o fato que temos que transformar este vetor para o formato Q15 – que não aceita valores ponto-flutuante superiores a $1.0$. Logo, temos que normalizar a resposta do filtro para que este valor máximo seja $1.0$, aplicando o seguinte ganho estático:

$$M = \frac{1+\mu}{2+\mu} \tag{38}$$

garantindo, assim, que todos os coeficientes são compatíveis com a transformação. A performance deste cálculo é muito importante, pois lembramos que temos efetivamente *dois* vetores de resposta impulsiva, um para cada canal sendo operado, que devem ser gerados a cada mudança do ângulo azimutal. Esta configuração iterativa é ótima e consome pouco processamento. Como sugerido pela equação 33, utilizamos um filtro de décima-sexta ordem.

Tendo os coeficientes, a operação do filtro ILD é relativamente simples está ilustrada na figura 18. Primeiro preparamos os coeficientes e depois passamos o vetor de entrada por um filtro FIR. Note na figura que temos blocos de decisão em azul. Estes blocos não representam uma decisão durante operação do programa, mas estão ali para ilustrar a possibilidade de compilar o código com a biblioteca *DSPLIB*. Os algoritmos usados para operar filtros FIR com e sem a biblioteca são descritos nas seções 6.2.6 e 6.2.7.

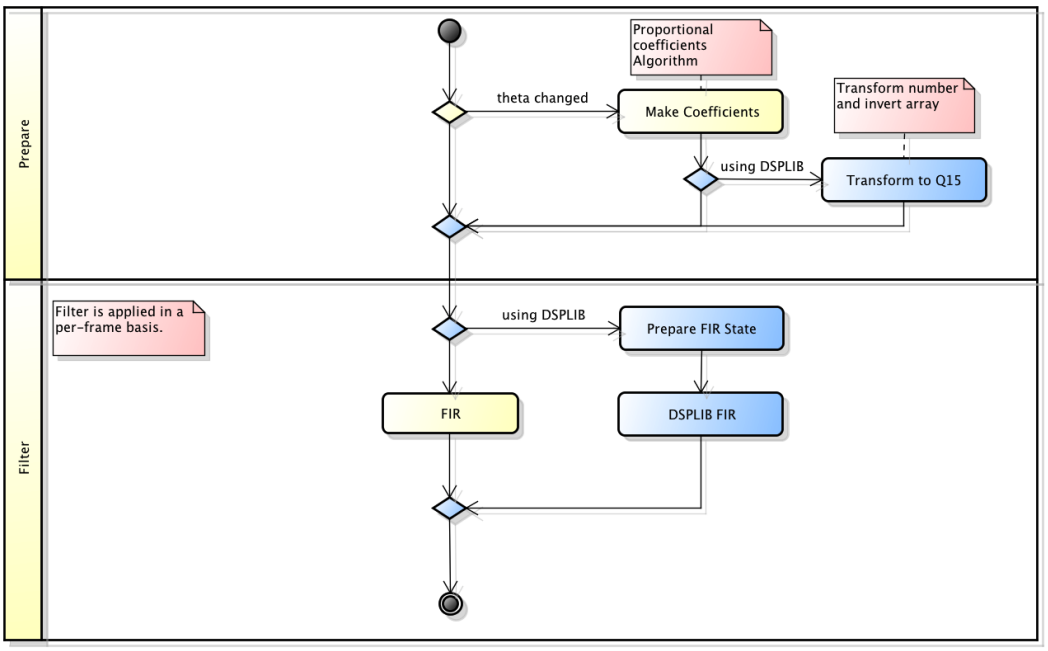



Figura 18: Algoritmo usado para o filtro ILD

### 6.2.3 PRTF

Como discutido anteriormente, a PRTF para este trabalho não contribui para a localização em si da fonte sonora, mas sim para fixar esta fonte no plano azimutal de elevação zero. Tal efeito é obtido utilizando o filtro visto no capítulo 4. Como este filtro é estático e não muda com a variação do ângulo azimutal, obtivemos os coeficientes da resposta

impulsiva até a décima-sexta ordem utilizando o Matlab e estes coeficientes ficaram no código do programa.[3]. Com os coeficientes, seguimos um algoritmo semelhante ao da ILD, ilustrado na figura 19, à diferença que a etapa de preparo é executada apenas uma vez durante a inicialização.

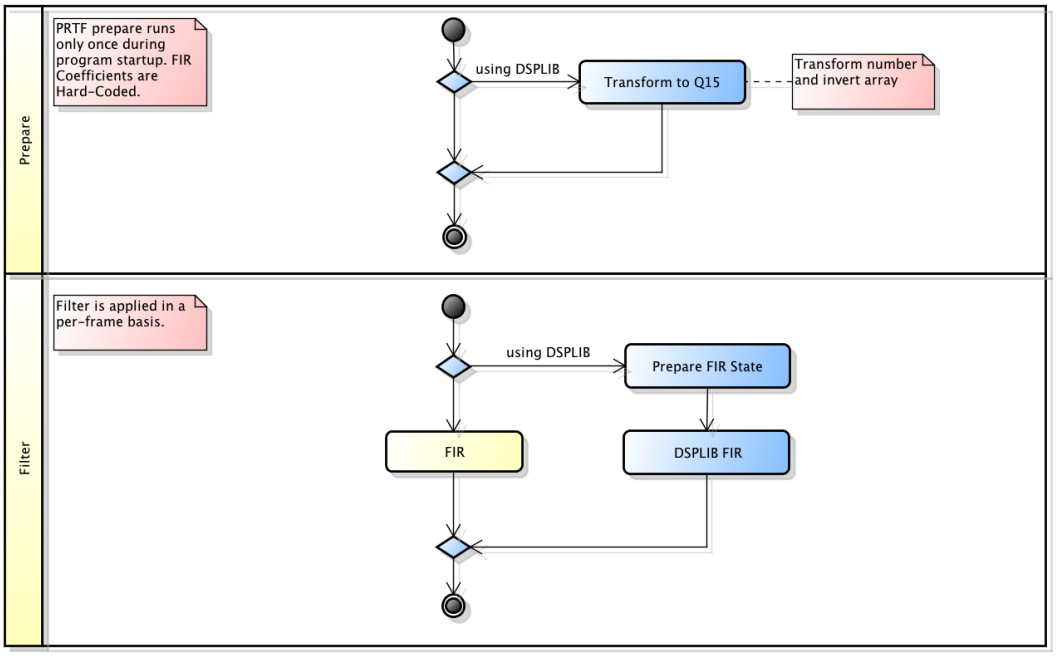



Figura 19: Algoritmo usado para o filtro PRTF

### 6.2.4 Passa-Baixas

Até então, não tínhamos mencionado a necessidade de um filtro passa-baixas neste sistema. Ele se faz necessário para eliminar a não linearidade dos filtros durante uma mudança do ângulo azimutal. Como as respostas impulsivas são muito mais rápidas que a mudança de $\theta$, os efeitos da variação dos filtros no tempo foram negligenciados até aqui. A figura 20 ilustra o efeito da não linearidade (da ITD, neste caso) em um sinal de entrada senoidal de 440Hz.

[3] Avaliando o filtro apresentado no capítulo 4, notamos que ele depende da frequência de amostragem. Foi feita uma tentativa de obter uma forma simbólica para este como para a ILD, para aumentar a portabilidade do código. No entanto, este filtro se mostrou consideravelmente mais complexo de ser obtido na forma simbólica e, também por esta razão, foi decidido deixá-lo fixo.

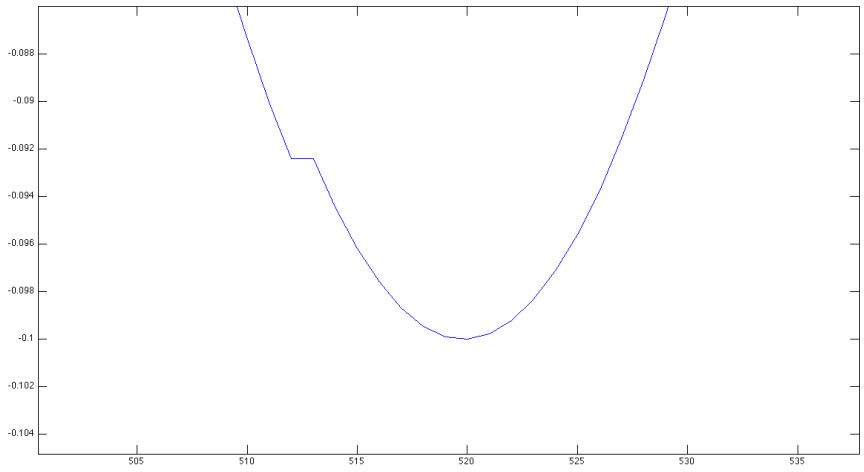



Figura 20: Não linearidade provocada pela mudança do ângulo azimutal

Como vemos na figura, a diferença de atraso causada pela mudança do ângulo azimutal gera um "dente" no tom. Mesmo este detalhe ínfimo causa uma sensação desagradável durante a operação do programa, percebido como "cliques" no som até que os filtros se estabilizem no novo ponto de operação. No caso ilustrado, a singularidade tem duração de uma amostra apenas, mas é mais que o suficiente para causar desconforto. Então, adiciona-se um filtro passa-baixas projetado no Matlab para frequência de corte a 60% da frequência de amostragem e atenuação de 40dB a 80% de $F_s$. O Matlab disponibiliza diretamente os coeficientes do filtro que foram *hard-coded* e operado exatamente como a PRTF que vimos anteriormente, no entanto com *tap* superior: 32. A figura 21 ilustra o algoritmo utilizado.

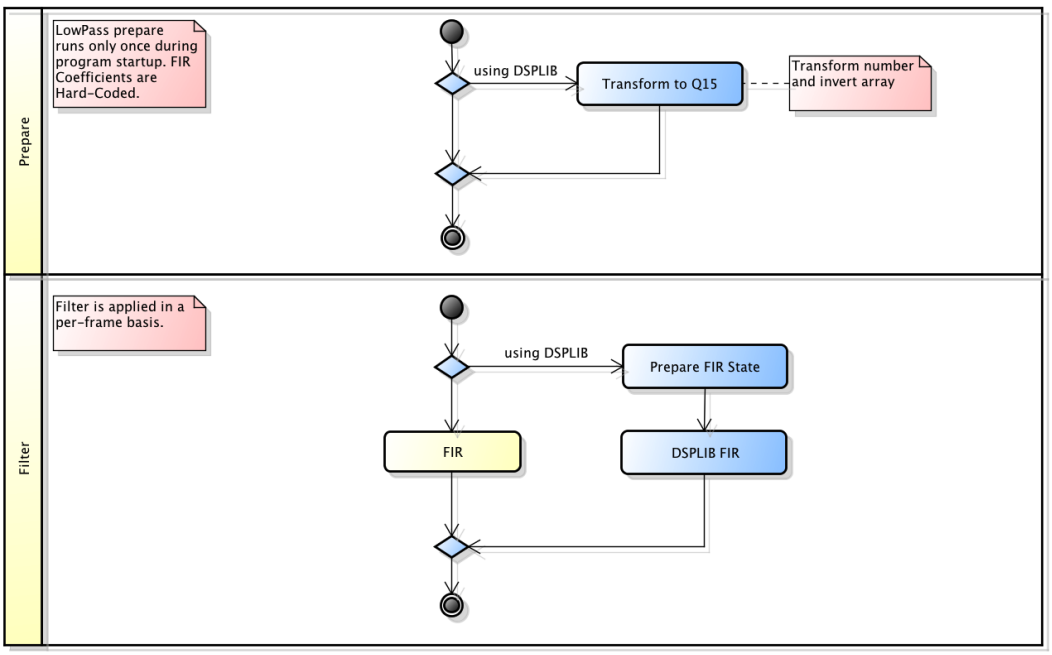



Figura 21: Algoritmo utilizado no filtro passa-baixas

### 6.2.5 Eco

No capítulo 4, foi sugerido que um eco simples do sinal original contribui para a sensação de localização para ângulos próximos aos ouvidos. A implementação deste em código de máquina é bem simples: guardamos todas as amostras em um vetor circular de tamanho igual ao atraso e aplicamos um ganho antes de retornar o valor. O algoritmo é ilustrado na figura 22. A etapa de preparação é necessária pois o programa carrega o atraso do eco durante a inicialização e temos que alocar a memória necessária para guardar todas as amostras.

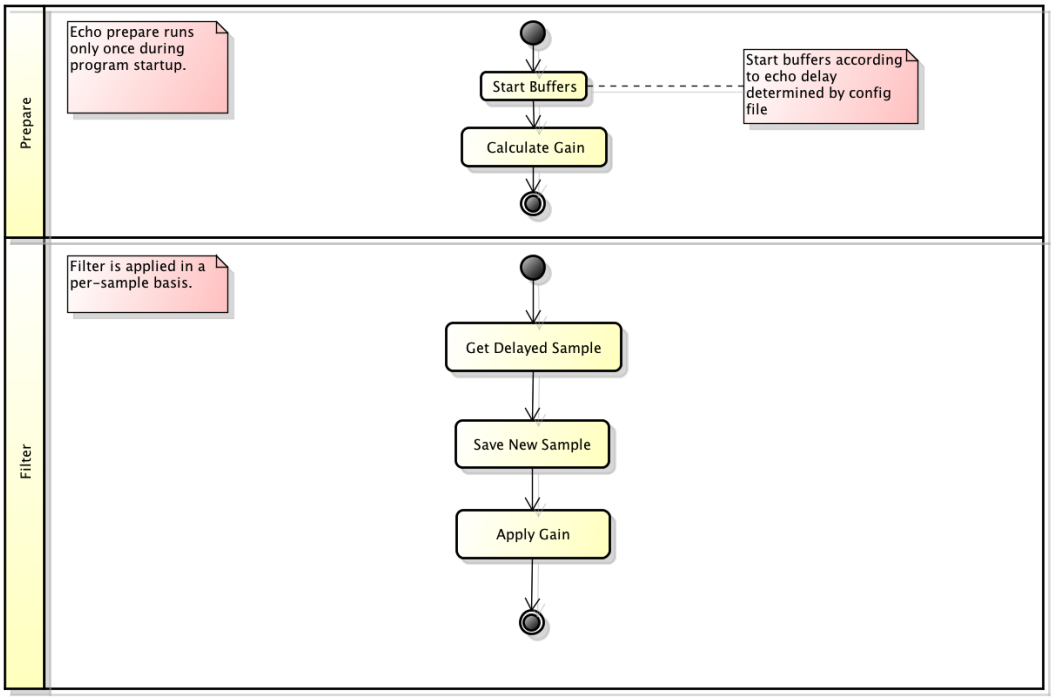



Figura 22: Algoritmo utilizado no filtro de Eco

### 6.2.6 FIR em ponto-flutuante

Para executar o processamento de um filtro FIR com as variáveis expressas em ponto-flutuante usamos a equação que descreve este tipo de filtro:

$$y[n] = \sum_{i=0}^{N-1} x[n-i] \cdot h[i] \tag{39}$$

Onde $y[n]$ é a saída, $x[n]$ é o vetor de entrada no mesmo momento e $h$ é um vetor de tamanho $N$ contendo os valores da resposta impulsiva do filtro. Note que a equação 39 expressa uma convolução simples. O valor $N$ normalmente é chamado de ordem ou *tap* do filtro. É uma prática comum usar *buffers* circulares para a memória deste bloco. Como vemos na listagem 6.1 podemos fazer esta operação para os dois canais ao mesmo tempo, neste caso ilustramos a operação com o filtro da ILD. A figura 23 mostra um fluxograma explicando o algoritmo utilizado.

```
1 void ildFilter( aSample * pL, aSample * pR )
2 {
3    // stuff for fir filter:
4    static float x[ 2 ][ ILD_TAP ]; // fir memory
```

```
5     static int n = 0; // position
6     int i;
7     float y[2] = { 0.0 , 0.0 }; // output
8
9     // copy states
10    x[ channelL ][ n ] = *pL;
11    x[ channelR ][ n ] = *pR;
12
13    // convolute
14    for( i = 0 ; i < ILD_TAP ; i++ )
15    {
16       y[ channelL ] += x[ channelL ][ ((unsigned int) n - i)
             % ILD_TAP  ] * ildCoeff[ channelL ][ i ]; // left
17       y[ channelR ] += x[ channelR ][ ((unsigned int) n - i)
             % ILD_TAP  ] * ildCoeff[ channelR ][ i ]; // right
18    }
19
20    // copy to output
21    *pL = y[ channelL ];
22    *pR = y[ channelR ];
23
24    // next
25    n = ( n + 1 )  % ILD_TAP;
26
27 }
```

Listagem 6.1: (*fir.c*) Implementação de um filtro FIR para dois canais simultâneos (ILD)

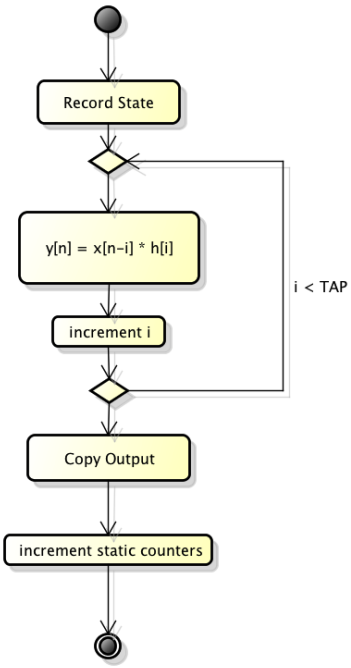



Figura 23: Algoritmo para calcular um filtro FIR

### 6.2.7 FIR em Q15 (DSPLIB)

Tendo a possibilidade de usar a biblioteca citada no capítulo anterior que contém funções otimizadas de processamento de sinais, podemos fazer uso destas para desfrutar de

maior performance na operação destes filtros. No entanto, o filtro FIR genérico oferecido pela *DSPLIB* implementa um algoritmo levemente diferente, regido pela equação 40. Note que o vetor $x$ é percorrido para frente, em contraste à equação 39. Complementarmente, o vetor da resposta impulsiva $\overline{h}$ deve – além de estar expresso no formato Q15 – estar ao contrário. A equação 41 mostra a conversão de um tipo de vetor resposta impulsiva para outro, onde $\overline{h}$ é o vetor usado na biblioteca *DSPLIB*.

$$y[n] = \sum_{j=0}^{N-1} x[n+j] \cdot \overline{h}[j] \tag{40}$$

$$\overline{h}[j] = h[(N-1)-j] \tag{41}$$

A listagem 6.2 mostra o algoritmo utilizado para converter o vetor $h$ em ponto-flutuante para o vetor $\overline{h}$ em Q15.

```
1  void makeShortCoeffFilter( const float * fpIn , short * sOut
       , short tap )
2  {
3     short  * helper  = malloc( tap * sizeof( short ) );
4
5     short * pS =  & (helper[ tap - 1 ]);
6
7     DSP_fltoq15( fpIn , helper, tap );
8
9
10    while( tap > 0 )
11    {
12       *sOut++ = *pS--;
13
14       tap--;
15    }
16
17    free( helper );
18 }
```

Listagem 6.2: (*makeCoeff.c*) Função que converte um vetor de resposta impulsiva de ponto-flutuante para formato Q15 recebido pela biblioteca *DSPLIB*

Lembrando que também o vetor $x$ deve entrar na função fornecida pela biblioteca de uma forma diferente que o simples *buffer* circular visto na seção 6.2.6. A listagem 6.3 mostra o código da macro utilizada para preparar um vetor $x$ a compatível. Note que este exemplo já prepara duas entradas para os dois canais processados. Construímos o vetor $x$ que será enviado para a função *de trás para frente*, diferentemente dos algoritmos vistos antes. A figura 24 mostra um fluxograma deste algoritmo de "preparação".

```
1 #define FIR_DOUBLE_PREPARE( pL, pR, tap )
2    static short m[ 2 ][ tap - 1 ];
3    const short msize = tap - 1;
4    static short n = 0;
5    short x[ 2 ][ FIR_BATCH_SIZE + tap  ];
6    short j , k  ;
7
8    for( k = 0 ; k < msize ; k ++ )
9    {
```

```
10          x[ channelL ][ k ] = m[ channelL ][ ( n + k ) % msize
                ]; /* getMemory */
11          x[ channelR ][ k ] = m[ channelR ][ ( n + k ) % msize
                ];
12      }
13
14      for( j = 0 ; j < FIR_BATCH_SIZE ; j++ )
15      {
16          m[ channelL ][ n ] = pL[ j ]; /* save memory */
17          m[ channelR ][ n ] = pR[ j ];
18
19          x[ channelL ][ k + j ] = pL[ j ]; /* build most recent
                */
20          x[ channelR ][ k + j ] = pR[ j ];
21
22          n = ( n + 1 )  % msize;
23      }
```

Listagem 6.3: (*firPrepare.c*) Macro utilizada para preparar o vetor de entrada para a função *DSP_fir_gen*

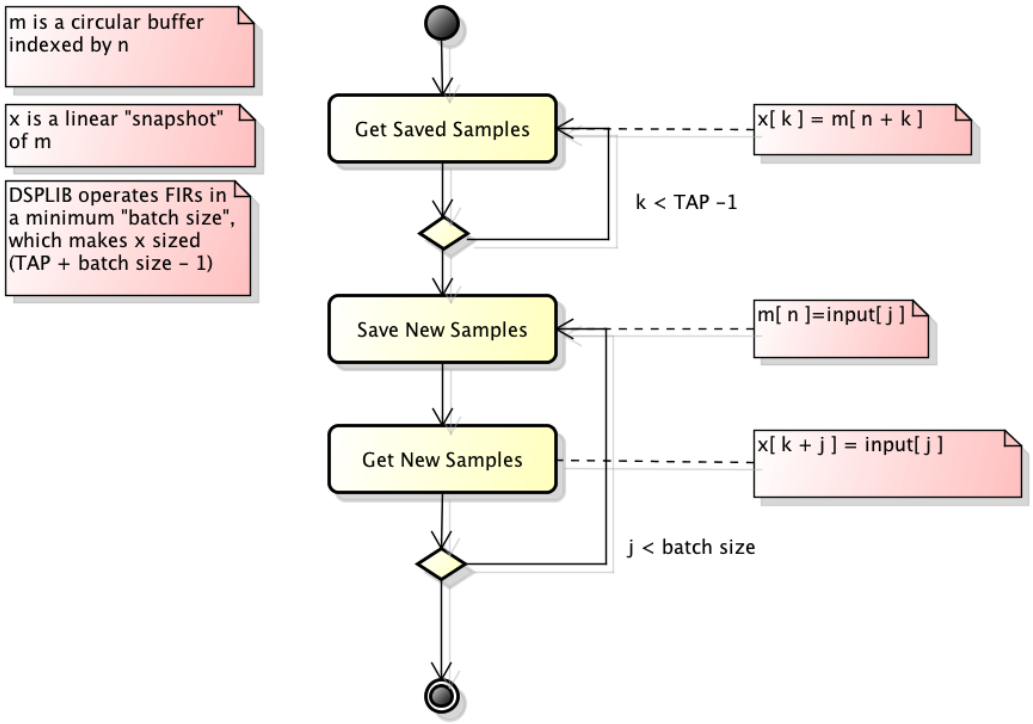



Figura 24: Algoritmo de preparação do vetor de entrada para a função *DSP_fir_gen*

Note a utilização da diretiva *FIR_BATCH_SIZE*. Um último detalhe é que a função *DSP_fir_gen* fornecida pela biblioteca não aceita operar o filtro por amostra, temos que enviar para a função um número mínimo $k \geq 4$ de amostras para que a função opere normalmente. Logo, surge esta diretiva que implementa um *batch* (“conjunto”) de amostras a ser operadas a cada chamada, ao invés de chamar a função a cada nova amostra.

## 6.3 Arquitetura de Tarefas

Nesta seção, descrevemos a organização das tarefas criadas pelo programa. Seguiremos o esquema da figura 25. O programa utiliza 3 *threads* (além da principal) para operar, das quais duas são criadas pela árvore principal e uma terceira é apenas ligada ao programa, porém, pertence efetivamente ao *daemon* do JACK que funciona com privilégios de tempo real no sistema operacional.

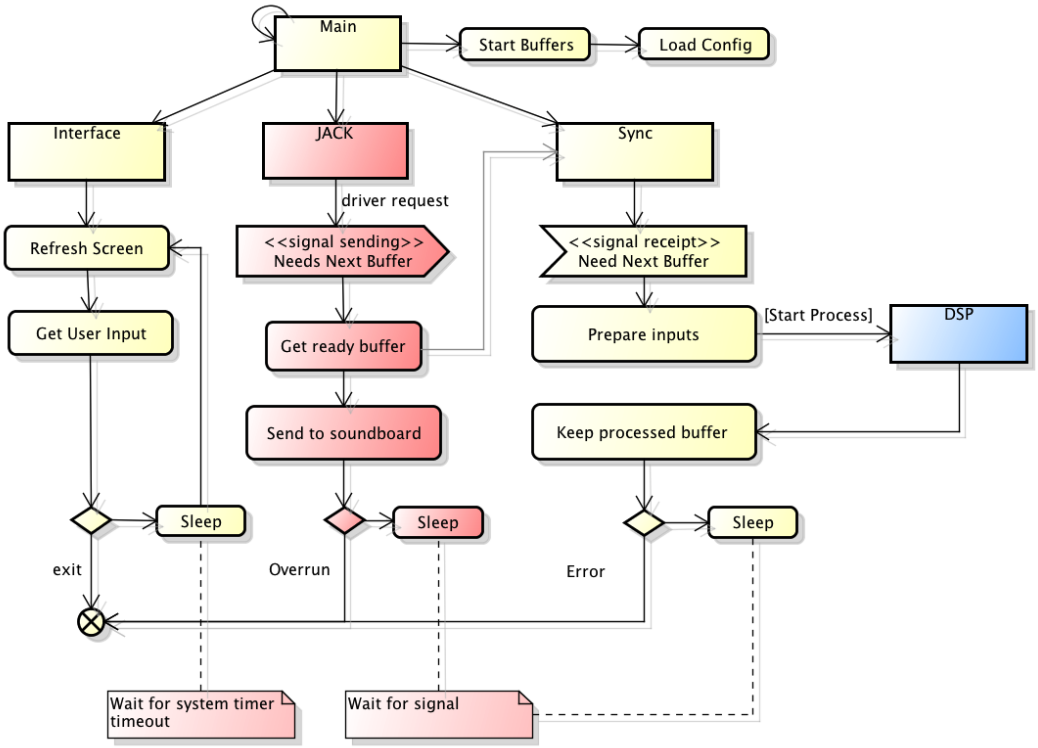



Figura 25: Esquema das tarefas da aplicação

### 6.3.1 Main

A tarefa principal é responsável pela inicialização dos espaços de memória que guardam os parâmetros que devem ser acessíveis tanto do lado ARM quanto do lado DSP. Em seguida, ela lança três tarefas "filhas" e fica em estado de espera até que algum erro ocorra ou o usuário termine o programa através da interface. Quando recebe o sinal de finalização, a tarefa envia sinais de término para as tarefas filhas que ainda vivem no sistema, aguarda os seus términos e finaliza a operação do programa. Esta tarefa é também responsável pela primeira chamada a alguma função no DSP (*dummie*). Isto é necessário pela latência que existe na primeira chamada de função de co-processamento, pois o núcleo auxiliar deve ser ligado, inicializado com o sistema DSPBIOS e só então aguardar comandos. Para esta inicialização foi criada uma função que calcula $\pi$ e imprime o resultado no console, apenas para lançar e validar a comunicação com o segundo núcleo. Cabe salientar que todos os espaços de memória que serão acessíveis do lado DSP devem ser alocados usando funções especiais do módulo de kernel *cmem* e, devido ao caráter assíncrono do programa, tentamos garantir que todos os ponteiros sejam válidos antes de lançar as outras tarefas que usaram este espaço de memória comum para guardar e buscar parâmetros para a sua operação.

### 6.3.2 JACK

Como já discutido, esta tarefa não pertence ao programa em si. A tarefa principal cria um cliente que se conecta ao servidor do JACK, que está ocioso no sistema operacional. Informamos todas as informações de conexão e, uma vez esta estabelecida, podemos obter as informações pertinentes para o processamento como tamanho dos *buffers* da placa de som e a frequência de amostragem utilizada. Para esta aplicação, a unidade de processamento do JACK é de 256 amostras e a frequência de amostragem é de 48kHz. Por fim, informamos uma função de *callback* que é chamada a cada vez que a placa de som (gerenciada pelo ALSA, no nosso caso) precisa de novas amostras. Esta função de *callback* apesar de viver no código dentro do programa é chamada dentro da *thread* do JACK e, logo, possui prioridades de tempo real no sistema. De volta à figura 25, vemos como opera a função de *callback*: ao receber o sinal da placa de som, enviamos um aviso à tarefa de sincronização que pode operar a próxima leva e, em seguida, enviamos à placa de som os *buffers* de saída já disponibilizados pela tarefa de sincronização. Para garantir um acoplamento entre os pedidos da placa de som e as respostas processadas, foi implementado um esquema de "assinaturas" de pergunta e resposta. Basicamente, existem duas variáveis inteiras definidas no código que indicam a última assinatura de demanda e a assinatura da resposta que está pronta. A função de *callback* só será operada com sucesso se ela pedir uma assinatura que representa um número acima da resposta que está pronta. Esta verificação simples perimitiu a depuragem para atrasos e perdas de pacote.

### 6.3.3 Sync

A tarefa de sincronização entre a placa de som e as operações do DSP foi apelidada de *sync*. Ela integra o sistema de "assinaturas", descrito anteriormente, e funciona através das paradas condicionais definidas pelas *pThreads*. Cada vez que a tarefa recebe o sinal de execução, esta prepara o vetor de entrada (de acordo com as opções selecionadas pelo usuário descritas no início deste capítulo) , envia todas as informações ao segundo núcleo e adormece até que receba uma resposta de fim de processamento. Do lado do DSP seguimos o algoritmo, descrito na figura 26, que trata de operar todos os filtros que estudamos até aqui em sequência e algumas verificações básicas quanto à validade dos valores encontrados.

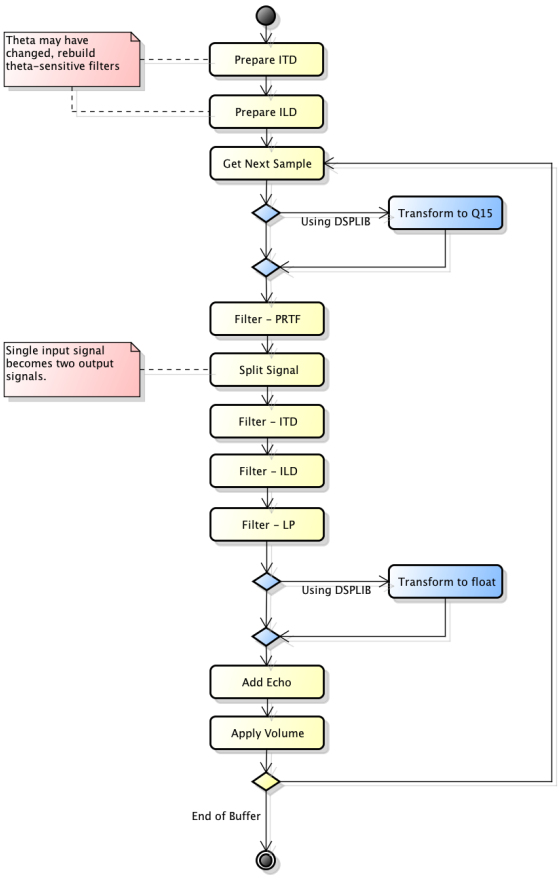



Figura 26: Fluxograma da operação principal de processamento de sinal

Quando o DSP encerra o processamento daquele pacote de amostras, a tarefa *sync* volta à ativa. Então ela disponibiliza o *buffer* se saída do DSP utilizando uma técnica bastante conhecida, chamada de *ping-pong*. Temos dois vetores de resposta idênticos e para evitar reescrever a cada vez todos os dados apenas trocamos os ponteiros destes. Onde um ponteiro sempre representa uma zona de memória que está em operação e o outro uma zona que já está pronta. Ao fazer esta operação de *ping-pong*, a rotina incrementa a assinatura da resposta, informando, de maneira assíncrona, ao cliente JACK que a sua próxima pergunta será válida. Caso ocorra algum erro durante a chamada do segundo núcleo, a tarefa finalizará sua execução e a rotina principal entrará em operação de finalização automaticamente. Caso contrário, a tarefa volta para uma condição ociosa, aguardando o próximo pacote.

Toda esta troca de pacotes e verificações entre as tarefas *sync* e JACK deve ser feita com atraso mínimo, pois se analisarmos os dados retornados pelo cliente JACK temos 256 amostras operadas a 48kHz, o que nos indica que cada pacote de dados deve ser operado a cada $5ms$, aproximadamente. Se as restrições temporais não são obedecidas, temos primariamente efeitos indesejados no áudio de saída e, caso o atraso seja grande demais, o cliente do JACK lançará um erro e a aplicação terminará para preservar a memória. Por isso, é de extrema importância manter a sincronia entre estas duas tarefas e, sobretudo, garantir que o algoritmo seja leve o suficiente para ser operado no tempo máximo. Com a sincronia mantida, provamos que o sistema é capaz de operar em tempo real.

Note na figura 26 as ações marcadas em azul. Se compilamos o programa com a *DSPLIB*, temos um pequeno desvio na execução do algoritmo de DSP, em que os *buffers* são

transformados para o formato Q15 antes de serem operados e transformados de volta para ponto-flutuante ao final da operação. Isto permite uma execução mais rápida dos filtros FIR, garantindo um tempo mínimo de execução por pacote de dados. Adicionalmente, reparamos que a PRTF é executada antes e apenas a um canal, contrariamente ao apresentado no capítulo 4. Isto se dá pois, como vimos, a PRTF não é sensível ao ângulo azimutal e, logo, se trata de um filtro fixo, como o passa-baixas, por exemplo. Como todos os filtros são lineares (considerando as restrições apresentadas), eles também são permutáveis pela definição de sistema linear. Logo, a PRTF não precisa ser operada para cada canal e economizamos processamento passando o sinal apenas uma vez por este, um passo antes de dividi-lo. Considerando esta mudança, o gráfico de processamento se torna o ilustrado na figura 27 – já acrescentando os filtros passsa-baixas e o ganho de volume $V$.

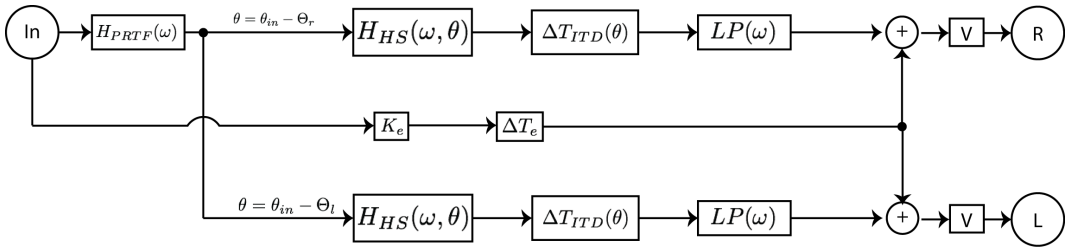



Figura 27: Diagrama de blocos de processamento de sinal pelo DSP

### 6.3.4 Interface

Não perderemos muito tempo na descrição desta tarefa em particular, pois ela serviu apenas de referência quanto aos parâmetros utilizados pelo sistema durante o processo de desenvolvimento. Como dito anteriormente, foi utilizada a biblioteca *ncurses* que possibilita uma pseudointerface com janelas no console de texto. A tarefa aguarda uma entrada de caractere único do usuário até um tempo limite *timeout* definido. Quando o tempo de espera é excedido, a interface reescreve as janelas e informações da tela com a biblioteca citada. A figura 28 mostra esta interface em operação.

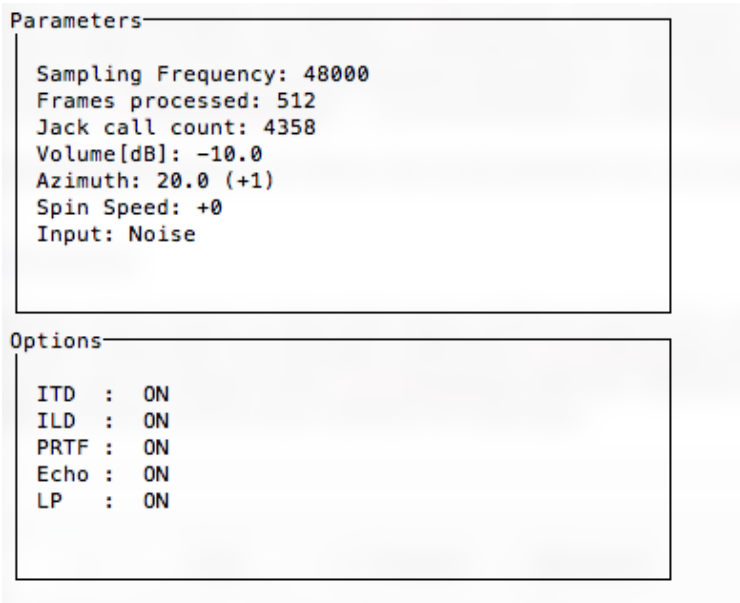



Figura 28: Captura de tela da interface do programa em operação

## 6.4 Imprevistos

Todo projeto apresenta alguns momentos em que a previsão de esforço não representa o esforço efetivamente empregado para determinadas tarefas. Para este projeto, encontramos dois destes grandes imprevistos – ambos relacionados à utilização do DSP em co-processamento.

O primeiro ponto em questão foi o correto emprego da biblioteca de compartilhamento de memória *CMEM*. Como dito, foi criado um *Makefile* que possibilita compilar o código para os dois processadores ou apenas para o lado ARM, permitindo depuragem em código nativo. Depois de todo o código pronto, quando compilado para o DSP, ele, estranhamente, não funcionava. A fim de descobrir o problema, deixamos ele processar um tom puro e plotamos a resposta no Matlab que pode ser vista na figura 29. Vemos na figura que quando o programa opera a partir do DSP existe um desalinhamento dos tons. Este fato chamou a atenção para a alocação de memória pelo módulo de kernel citado.

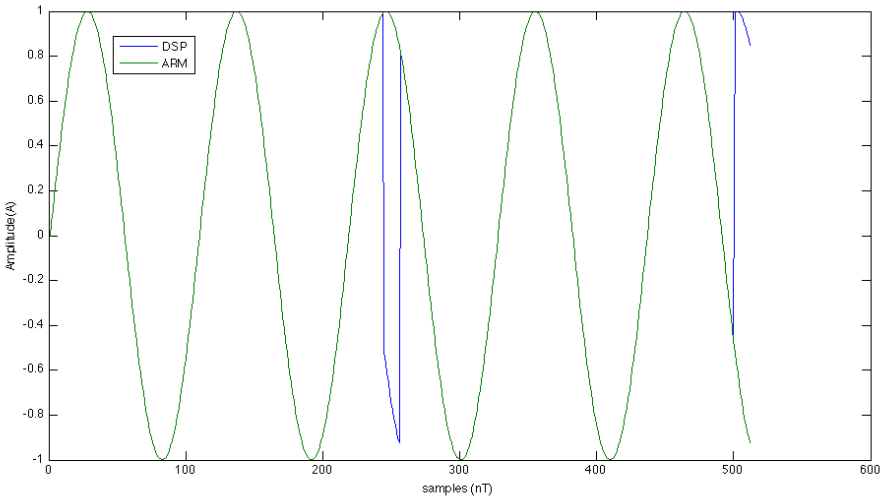



Figura 29: Problemas de alinhamento de memória compartilhada

Ao fim de inúmeras tentativas e aproximações de depuragem, foi constatado que se *todos* os *buffers* fossem alocados com o dobro da memória, o programa funcionava perfeitamente no DSP. Este fato pôs em evidência que a figura destorcida que vemos acima se dá porque os ponteiros não estão percorrendo o vetor corretamente ou este está sendo corrompido. Recorrendo à documentação da ferramenta *C6EZRun* encontramos um detalhe quanto ao alinhamento mínimo de memória dever ser feito em 8 bits, não 4 – o que é mais comum. Espantosamente, forçando um alinhamento a 8 bits não resolveu o este problema. Em uma última tentativa foi forçado um alinhamento de 16 bits nas alocações de memória e eis que esta solução, efetivamente, resolveu o desencontro dos ponteiros.

Uma vez os *buffers* sendo percorridos corretamente, ativamos os filtros de processamento um a um. Neste ponto, um novo inconveniente se apresentou: a chamada do segundo processador apresenta um *overhead* muito grande – o que introduz uma latência da resposta e impede o tempo real. Para resolver este novo empecilho foi acrescentada uma opção de pré-compilador em que os vetores de processamento são múltiplos inteiros dos vetores esperados pela placa de som. Naturalmente, esta solução atrasa a resposta mas não gera “travamentos” no momento do processamento, pois uma vez que a função

está operando no lado DSP, ela é efetivamente operada com mais rapidez. Os multiplicadores foram inseridos no algoritmo de detção de assinaturas, em que só enviamos uma nova mensagem de processamento para o segundo processador quando temos assinaturas que são múltiplas inteiras deste multiplicador de *buffer*. Na prática, um fator de 2 foi o suficiente para que o *overhead* não afetasse a performance do sistema.

# 7 RESULTADOS

Em uma etapa de testes, avaliamos a capacidade de emprego deste sistema em artigos de consumo. Dois critérios principais foram levados em conta: a velocidade de processamento e a percepção de espacialização. Quanto à velocidade de processamento, foram levados em conta três possibilidades:

1. Programa apenas no ARM
2. Processamento no DSP sem *DSPLIB*
3. Processamento no DSP com *DSPLIB*

No primeiro caso, verificamos que não houve *overrun* na placa de som. No entanto, a aplicação toma aproximadamente 47% de CPU do núcleo principal. O que é extremamente alto e comprova as preocupações encontradas na literatura quanto à usabilidade destes sistemas em aplicações de consumo. No segundo caso, para que o programa funcione sem *overruns*, faz-se necessário desligar *no mínimo* dois dos filtros, pois o segundo núcleo não opera em ponto-fixo neste caso e não se trata de sua forma natural de processamento. Este caso não foi levado em conta para a análise de localização, pois os "travamentos" causados pela lentidão geram uma saída de som desagradável. Para o terceiro caso, supomos[1] o segundo núcleo em dedicação funcionando a aproximadamente 50% da carga. O núcleo principal ficou livre para operar o sistema operacional – a aplicação consome menos de 4% de CPU neste.

Quanto à sensação de localização da fonte sonora convidamos três voluntários para testar o sistema. Eles puderam alterar o ângulo azimutal manualmente ou programar uma velocidade angular fazendo a fonte sonora girar em torno da cabeça do ouvinte. Primeiramente foi constatado que o tom fundamental a 440Hz não é localizável (nem se em movimento), reforçando as teorias de que dependemos muito do envelope de espectro para fazer esta localização. O ruído branco fica em segundo lugar quanto à sensação de posição, permitindo uma grande impressão de espacialização em movimento e uma resolução levemente superior a $5^{\circ}$. Foi utilizado um trecho da música *Tom's Diner*, deSuzanne Vega, para o terceiro teste. Esta faixa é sugerida em [46] por sua característica de ser apenas cantada, sem o acompanhamento de nenhum instrumento musical. Neste caso em especial, a sensação desejada ficou notavelmente mais forte, sugerindo que o sistema neurológico é especialmente treinado para localizar voz.

Para todos os testes, foi verificado inversão frente-trás para fontes estáticas próximas do ângulo frontal e seu inverso, a fonte em movimento elimina este problema, pois a

[1] A versão das ferramentas de DSP utilizadas não possuem funções para verificar a real carga do processador.

simetria que causa a inversão é rapidamente quebrada. Em uma segunda iteração foram ensaiados dois diferentes tipos de fones de ouvido: externos e internos. Para ambos os casos a resolução estática de ângulo ficou em torno de $5^{\circ}$, no entanto, a PRTF apresenta respostas melhores para o fone interno. Contrariamente ao antecipado, para todos os casos a sensação de espacialização foi maior quando o usuário não teve controle sobre o ângulo azimutal, assim como passos menores que $5^{\circ}$ não são perceptíveis.

# 8 FUTURO DO PROJETO

Este projeto apresenta muitos níveis e abordagens de estudo e, infelizmente, não foi possível avaliar todas as possibilidades ou explorar mais ferramentas de processamento. Listamos abaixo algumas melhorias que poderiam ser feitas nesta aplicação:

- Utilizar a entrada de som da placa para localizar a entrada de um microfone P2;
- Localizar $n$ fontes sonoras monofônicas simultaneamente – lidas de um arquivo de áudio codificado em 5.1, por exemplo;
- Mostrar, em interface gráfica avançada, as respostas dos filtros para depuragem;
- Acoplar o processamento a um ambiente virtual 3D para provar a interdependência dos sentidos;
- Introduzir um acelerômetro acoplado aos fones de ouvido para obter uma compensação do movimento da cabeça.

Por estas e outras possíveis evoluções do sistema, este será disponibilzado, gratuitamente, na internet através do site [11] e de outras ferramentas de compartilhamento de código aberto. Devido à intenção de abrir o código fonte desde o princípio, todo ele foi feito em inglês e com meticulosa documentação, permitindo, assim, que valores possam ser agregados ao projeto.

# 9 CONLUSÃO

Neste projeto, avaliamos a viabilidade de um sistema de localização de sons usando sinais binaurais e sistemas de filtragem analíticos. Verificamos que, com filtros simples, podemos criar aplicações que – mesmo que computacionalmente pesadas – reproduzem com alguma fidelidade os efeitos de localização de som. A utilização de filtros analíticos para tal é o diferencial em foco, pois todos surgem de equações matemáticas parametrizáveis, fazendo com que apenas estes parâmetros sejam – ou não – empíricos.

Em um segundo objetivo, verificamos as extensões que atingem as novas tecnologias de processamento portátil. Com a avenção de *smartphones* e *tablets*, temos à disposição ferramentas de alto processamento e de baixo consumo que permitem aplicações extremamente avançadas. Complementarmente, os movimentos de *Open Hardware* e *Open Source* trazem as tecnologias de ponta ao alcance de todos, transformando usuários em potenciais desenvolvedores. O casamento da alta capacidade de processamento com algoritmos de manipulação de sinais possibilitam aplicações de grande complexidade, como verificado neste projeto. Enfim, a execução desta aplicação foi um exercício de pesquisa, projeto e desenvolvimento, fechando um ciclo completo de desenvolvimento de produto em pequena escala.

# REFERÊNCIAS


[1] C6ezrun. URL http://processors.wiki.ti.com/index.php?title=C6EZRun.

[2] Dm 3730 texas instruments product details. URL http://www.ti.com/product/dm3730.

[3] Doxygen – documentation from source code. URL www.doxygen.org/.

[4] Dsplib. URL http://processors.wiki.ti.com/index.php/DSPLIB.

[5] git – the fast version control system. URL http://git-scm.com/.

[6] Jack audio connection kit. URL http://jackaudio.org/.

[7] Narcissus angstrom distribution. URL http://narcissus.angstrom-distribution.org/.

[8] Vim online. URL http://www.vim.org/.

[9] Hrtf measurements of a kemar dummy-head microphone, 200. URL http://sound.media.mit.edu/resources/KEMAR.html.

[10] The cipic hrtf database, 2004. URL http://interface.cipic.ucdavis.edu/sound/hrtf.html.

[11] Beagleboard.org, 2011. URL http://beagleboard.org.

[12] Robert J. Jr.; Duda Richard O.; Thompson Dennis M. Algazi, V. Ralph; Dalton. Motion-tracked binaural sound for personal music players. In *Audio Engineering Society Convention 119*, 10 2005. URL http://www.aes.org/e-lib/browse.cfm?elib=13312.

[13] V. Ralph Algazi, Carlos Avendano, and Richard O. Duda. Estimation of a spherical-head model from anthropometry. *National Science Foundation*, 2001.

[14] V. Ralph Algazi, Richard O. Duda, D. M. Thompson, and Carlos Avendano. The cipic hrtf database. *CIPIC U.C. Davis*, 2001.

[15] Frederico Avanzini. *Algorithms for Sound and Music Computing*, chapter 4, Sound in Space. 2009.

[16] Carlos Avendano, V. Ralph Algazi, and Richard O. Duda. A head-and-torso model for low-frequency binaural elevation effects. *IEEE Workshop on APplications of Signal Processing to Audio and Accoustics*, 1999.

[17] C. Philip Brown and Richard O. Duda. A sructural model for binaural sound synthesis. *IEE Transactions on Speech and Audio Processing*, 6, 1988.

[18] C. Philip Brown and Richard O. Duda. An efficient hrtf model for 3-d sound. Master's thesis, University Of Mariyland, San Jose State University, 1997.

[19] David A. Burgess. *Techniques for Low Cost Spatial Audio*. PhD thesis, Georgia Institute of Technology.

[20] Simon Carlie. *Virtual Auditory Space: Generation and Applications*, chapter 2. Landes Company, 1996.

[21] Jiashu Chen, Barry D. Van Veen, and Kurt E. Hecox. External ear transfer function modeling. *J. Acoust. Soc. Am.*, 1992.

[22] Corey I. Cheng and Gregory H. Wakefield. Introduction to head-related transfer functions (hrtfs): Representations of hrtfs in time, frequency and space. *J Audio Eng Soc*, 49(4), 2001.

[23] DELET. Deletex, classe latex do delet, 2011. URL `http://www.ece.ufrgs.br/~fetter/deletex/`.

[24] Ramani Duraiswami. Introduction to hrtfs. PPT.

[25] Fábio P. Freeland, Luiz Wagner P. Biscainho, and Paulo Sergio Diniz. Efficient hrtf interpolation in 3d moving sound. In *AES 22nd International Conference on Virtual, Synthetic and Entertainment Audio*, 2002.

[26] Michele Geronazzo, Simone Spagnol, and Federico Avanzini. Estimation and modeling of pinna-related transfer functions. In *Int. Conference on Digital Audio Effets*, volume 13, 2010.

[27] Nail Gurnerov and Ramani Duraiswami. Physics based approaches to hrtf computation and understanting. PPT.

[28] Jyri Huopaniemi and Matti Karljalainen. Review of digital filter design and implementation methods for 3-d sound. Master's thesis, Helsinki University of Technology.

[29] Jyri Huopaniemi and Matti Karljalainen. Hrtf filter design based on auditory criteria. *Nordic Accoustical Meeting*, 1996.

[30] Doris J. Kistler and Frederic L. Wightman. A model of head-related transfer functions based on principal components analysis and minimum-phase reconstruction. *J. Acoust. Soc. Am.*, 1992.

[31] Kyosik Koo and Hyungtai Cha. Enhancement of a 3d sound using pshychoacoustics. *World Academy of Science, Engineering and Technology*, 2008.

[32] Joel David Miller. Modeling interneural time difference assuming a spherical head. Master's thesis, Stanford University, 2001.

[33] Wang Peng, Wee Ser, and Ming Zhang. Bark scale equalizer design using warped filter. Master's thesis, School of Electrical and Electronic Engineering Nanyang Technological University.

[34] J.W.S. Rayleigh. *The theory of sound*. Number v. 1 in The Theory of Sound. Macmillan, 1894. URL http://books.google.com/books?id=EGQSAAAAIAAJ.

[35] Simone Spagnol, Michele Geronazzo, and Federico Avanzini. Structural modeling of pinna-related transfer functions. 2010.

[36] Marc M. Van Wanrooij and A. John Van Opstal. Contribution of head shadow and pinna cues to chronic monaural sound localization. *Journal of Neuroscience*, 2004.

[37] György Wersényi. Representations of hrtfs using matlab: 2d and 32 plots of accurate dummy-head measurements. In *Proceedings of 20th International Congress on Acoustics*, 2010.

[38] Wikipedia. Phonograph — wikipedia, the free encyclopedia, 2011. URL http://en.wikipedia.org/w/index.php?title=Phonograph&oldid=444663711. [Online; accessed 17-August-2011].

[39] Wikipedia. Ncurses — wikipedia, the free encyclopedia, 2011. URL http://en.wikipedia.org/w/index.php?title=Ncurses&oldid=460380131. [Online; accessed 14-November-2011].

[40] Wikipedia. Posix threads — wikipedia, the free encyclopedia, 2011. URL http://en.wikipedia.org/w/index.php?title=POSIX_Threads&oldid=460581886. [Online; accessed 14-November-2011].

[41] Wikipedia. Stereophonic sound — wikipedia, the free encyclopedia, 2011. URL http://en.wikipedia.org/w/index.php?title=Stereophonic_sound&oldid=444500670. [Online; accessed 17-August-2011].

[42] Wikipedia. Surround sound — wikipedia, the free encyclopedia, 2011. URL http://en.wikipedia.org/w/index.php?title=Surround_sound&oldid=445325953. [Online; accessed 18-August-2011].

[43] Wen Zhang, Thusara D. Abhayapala, Rodney A. Kennedy, and Ramani Duraiswami. Modal expasion of hrtfs: Continuous representation in frequency-range-angle. *IEE Xplore*, 2009.

[44] Wen Zhang, Rodney A. Kennedy, and Thusara D. Abhayapala. Efficient continous hrtf model using data independent basis functions: Experimentally guided approach. *IEE Transactions on Audio, Speech and Language Processing*, 2009.

[45] Wen Zhang, Thusara D. Abhayapala, and Rodney A. Kennedy. Insights into head-related transfer fucntion: Spatial dimensionality and countinuous representation. *J. Acoust. Soc. Am.*, 2010.

[46] U. Zölzer and X. Amatriain. *DAFX: digital audio effects*. Wiley, 2002. ISBN 9780471490784. URL http://books.google.com/books?id=h90HIV0uwVsC.

# APÊNDICE A GRÁFICO DE GANTT

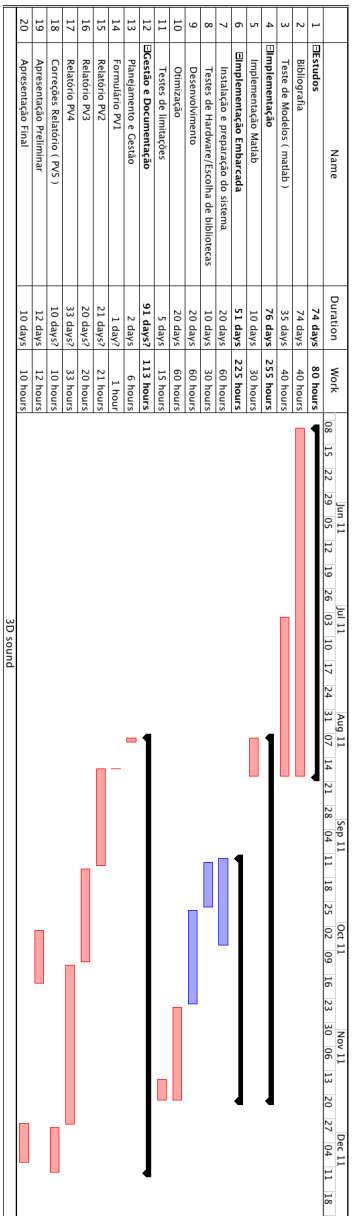



Figura 30: Gráfico de Gantt usado na gestão de projeto

# APÊNDICE B DOXYGEN - EXEMPLO DE DOCUMENTAÇÃO AUTOMÁTICA

#### 4.21.2 Function Documentation

##### 4.21.2.1 void lowPass ( float * *pL*, float * *pR* )

low pass for both channels simultaneously

**Parameters**

| | |
|---|---|
| *pL* | pointer to left channel sample |
| *pR* | pointer to right channel sample |

Definition at line 121 of file dsp_lowpass.c.

Here is the caller graph for this function:

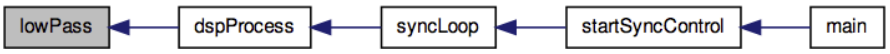



Figura 31: Exemplo de documentação automática gerada em LaTeX pelo Doxygen

# APÊNDICE C VERSIONAMENTO DE SISTEMA

O lado ARM do sistema utiliza:

- OS Angstrom *Linux beagleboard 2.6.32*;
- O compilador utiliza as seguintes opções:

```
1 Target: arm-angstrom-linux-gnueabi
2 Configured with: /OE/angstrom/build/tmp-angstrom_2008_1/
      work/i686-armv7a-sdk-angstrom-linux-gnueabi/gcc-
      cross-sdk-4.3.3-r23.4/gcc-4.3.3/configure --build=
      i686-linux --host=i686-linux --target=arm-angstrom-
      linux-gnueabi --prefix=/usr/local/angstrom/arm --
      exec_prefix=/usr/local/angstrom/arm --bindir=/usr/
      local/angstrom/arm/bin --sbindir=/usr/local/angstrom
      /arm/bin --libexecdir=/usr/local/angstrom/arm/
      libexec --datadir=/usr/local/angstrom/arm/share --
      sysconfdir=/usr/local/angstrom/arm/etc --
      sharedstatedir=/usr/local/angstrom/arm/share/com --
      localstatedir=/usr/local/angstrom/arm/var --libdir=/
      usr/local/angstrom/arm/lib --includedir=/usr/local/
      angstrom/arm/include --oldincludedir=/usr/local/
      angstrom/arm/include --infodir=/usr/local/angstrom/
      arm/share/info --mandir=/usr/local/angstrom/arm/
      share/man --enable-largefile --disable-nls --enable-
      ipv6 --with-gnu-ld --enable-shared --enable-
      languages=c,c++,objc,fortran --enable-threads=posix
      --disable-multilib --enable-c99 --enable-long-long
      --enable-symvers=gnu --enable-libstdcxx-pch --
      program-prefix=arm-angstrom-linux-gnueabi- --enable-
      target-optspace --enable-cheaders=c_std --enable-
      libssp --disable-bootstrap --disable-libgomp --
      disable-libmudflap --with-sysroot=/usr/local/
      angstrom/arm/arm-angstrom-linux-gnueabi --with-build
      -time-tools=/OE/angstrom/build/tmp-angstrom_2008_1/
      sysroots/i686-linux/usr/armv7a/arm-angstrom-linux-
      gnueabi/bin --with-build-sysroot=/OE/angstrom/build/
      tmp-angstrom_2008_1/sysroots/armv7a-angstrom-linux-
      gnueabi --disable-libunwind-exceptions --disable-
      libssp --disable-libgomp --disable-libmudflap --with
      -mpfr=/OE/angstrom/build/tmp-angstrom_2008_1/
      sysroots/i686-linux/usr --enable-__cxa_atexit
3 Thread model: posix
4 gcc version 4.3.3 (GCC)
```

Para o lado DSP foi compilada a ferramenta *C6Run* versão *0.97.03.03* utilizando os seguintes pacotes:

- *DSPLINK* v 1.65.00.03
- *LPM* v 1.24.02.09
- *DSPBIOS* v 5.41.07.24
- *SYSLINK* v 02.00.00.68b1
- *IPC* v 1.22.04.25
- *XDCTOOLS* v 3.20.05.76
- *LINUXUTILS* v 2.25.05.11
- *TI-CGT* v 7.3.0

O servidor de áudio JACK usado é a versão *jackd 0.118.0* e é inicializado (por usuário com permissões de tempo real) com a seguinte linha de comando:

```
1 jackd -t5000 -R -d alsa -p 256 -n 4 -P hw:0 -C hw:0 -S -s -r
      48000 > jack_log &
```

# APÊNDICE D DIRETIVAS DE BOOT

O *bootloader* U-Boot permite a criação de um arquivo chamado *uEnv.txt* que carrega em tempo de inicialização para definir algumas diretivas para o kernel. Abaixo, as linhas utilizadas para personalizar a inicialização. As áreas de memória indicadas devem estar de acordo com as linhas utilizadas para lançar o módulo de kernel *CMEM*.

```
1 optargs="mem=99M@0x80000000 mem=128M@0x88000000"
2 dvimode="640x480MR-16@60"
```