Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

Felipe de Oliveira Campos Moreira

Sensors2Music: Contrabaixo aumentado utilizando dispositivos Android

São Paulo
2017

Felipe de Oliveira Campos Moreira

Sensors2Music: Contrabaixo aumentado utilizando dispositivos Android

Trabalho de conclusão do curso de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo.
Orientadores: Marcelo Gomes de Queiroz, Carolina Brum Medeiros, Antonio Deusany de Carvalho Junior

São Paulo

2017

Agradecimentos

Agradeço aos meus orientadores, Carol, Marcelo Queiroz e Antonio pela ajuda proporcionada durante o trabalho.

Agradeço aos meus colegas de graduação Felipe Félix e Pedro Marcondes por terem dado suporte ao longo do ano para que o trabalho fosse feito.

Agradeço ao grupo de computação musical por tecerem críticas, sugerirem novas ideias, além de tirarem dúvidas pontuais.

Resumo

Um instrumento aumentado é um instrumento musical tradicional cujas habilidades sonoras foram estendidas através de sensores. Dispositivos móveis com o sistema Android incluem diversos sensores que são normalmente utilizados para alterar configurações do celular, como brilho, e também para interação, no caso de jogos. Estes dispositivos são atualmente os mais populares no Brasil devido ao seu preço médio. Tendo isto em vista, um músico amador pode construir um instrumento aumentado usando seu próprio dispositivo telefônico sem muito esforço e sem a necessidade de adquirir um controlador ou processador de áudio externo. Neste trabalho apresentaremos diversas formas de utilizar o Android para a criação de um contrabaixo aumentado através de aplicações e tecnologias que são avaliadas e comparadas entre si. Analisaremos as formas de conexões do instrumento com o aparelho, a latência existente entre a entrada e saída do áudio e o funcionamento dos sensores do celular. Ao final será apresentada a implementação da aplicação desenvolvida durante este trabalho, a qual utiliza sensores para controle de efeitos sonoros e realiza processamento de áudio em tempo real baseando-se nos dados dos sensores.

palavras-chave: Computação Musical; instrumento aumentado; Android; Sensores; contrabaixo; latência.

Abstract

An augmented instrument is a musical instrument whose sound abilities have been extented through sensors. Mobile devices using Android operating system have embedded sensors which are used to change configurations like brightness and also for user interaction in case of gamming applications. These devices are currently the most popular in Brazil and they are more accessible due to their average price. Therefore, an amateur musician can build an augmented instrument using his own smartphone without much effort and without buying an audio controller or processing unit. In this work we show many possibilities of using the Android for creating an augmented Bass taking advantage of applications and technologies for this purpose which are evaluated and compared afterward. We analyse the ways we can connect an instrument to the smartphone, the latency between audio input and output, and the sensors functionalities. In the end, we present an application developed during this work whose main proposal was to use the embedded sensors to control sound effects and proccess audio in real-time based on the data form the sensors.

Keywords: Computer Music; Augmented Instrument; Android; Sensors; bass guitar; latency.

Sumário

Capítulo 1

Introdução

Instrumentos musicais podem ter seu som processado de diversas maneiras, neste trabalho falaremos sobre o processamento digital do som em aparelhos Android.

1.1 Contextualização

Até 2016, os dispositivos Android representavam uma fatia de 90% do mercado brasileiro de smartphones[1]. A partir da versão 5 (Lollipop), a latência de som no processamento desses dispositivos começou a diminuir consideravelmente,desde 300 ms na versão 4.3 chegando a valores menores que 10 ms[2], o suficiente para ser percebido como uma resposta instantânea para o ouvido humano.

Estes dispositivos geralmente possuem sensores como giroscópio, sensores de aceleração linear, de pressão, entre outros. Tais sensores podem ser utilizados como controladores sonoros, como é o caso do Sensors2PD. Por exemplo, ao movimentar o celular para o lado um player poderia interpretar esse gesto como um comando para trocar de música.

Um instrumento aumentado é um instrumento musical tradicional cujas habilidades sonoras foram estendidas através de sensores ligados ao instrumento e do processamento sonoro parametrizado pelos sensores (2).

Tais características, tanto a latência baixa como os sensores embarcados, podem ser levadas em consideração na criação de instrumentos aumentados. Um músico poderia utilizar seu celular tanto quanto um processador de sinais quanto como um controlador de efeitos, podendo adicionar possibilidades sonoras à performance sem precisar de equipamentos que

[1] `http://blogs.oglobo.globo.com/lauro-jardim/post/vendas-de-celulares-com-sistema-android-e-windows-c`
`html`

[2] `http://superpowered.com/superpowered-android-media-server`

não são fáceis de se adquirir.

1.2 Proposta

A proposta do trabalho é desenvolver um aplicativo para Android que visa agir tanto como um processador de sinais quanto como um controlador de efeitos sonoros. O aplicativo recebe um sinal de entrada proveniente de um instrumento musical e processa esta entrada com efeitos pré-selecionados pelo usuário, efeitos esses que são parametrizados por sensores embarcados. Também serão analisadas as tecnologias atuais em relação à conexão e os processamento para decidir quais são as mais adequadas.

1.3 Trabalhos relacionados

Para descobrir o atual estado-da-arte, fez-se um levantamento de aplicativos que utilizavam o Android em contextos musicais. As principais fontes foram a Google Play Store Store[3], o repositório F-Droid[4] e alguns grupos de pesquisa em computação musical[5], pois nestas podemos encontrar aplicativos feitos para Android que estão disponíveis para um músico sem um grande background em computação.

Dos trabalhos relacionados, podemos destacar três principais categorias: Aplicativos que utilizam sensores Android como apenas um controlador externo, os que utilizam o aparelho apenas como um processador de áudio e os que mesclam as duas funções.

1.3.1 Aplicativos de controle externo

Aplicativos de controle externo são os que utilizam o aparelho apenas como um controlador (como por exemplo o CONTROL(7), sem efetuar qualquer tipo de processamento de sinal. Os dados são enviados através de uma conexão de rede, geralmente utilizando o protocolo *Open Sound Control* (OSC). Dentre os aplicativos desse tipo, podem ser destacados dois, **TouchOSC**[6] e **Sensors2OSC**[7](1).

[3] Google Play: https://play.google.com/

[4] Repositório F-Droid: https://f-droid.org/

[5] Grupos de pesquisa de computação musical: https://compmus.ime.usp.br, http://eecs.umich.edu

[6] TouchOSC: https://hexler.net/software/touchosc

[7] Sensors2OSC: https://github.com/SensorApps/Sensors2OSC

TouchOSC envia mensagens no protocolo OSC e no protocolo *Musical Instrument Digital Interface, MIDI*. Funciona principalmente através da tela *touch*, utilizando apenas o acelerômetro como sensor integrado. Por outro lado o Sensors2OSC permite a listagem e utilização de todos os sensores disponíveis no celular, além de ser *Open Source*.

1.3.2 Aplicativos processadores de áudio

Dos aplicativos que utilizam a entrada micro USB podemos destacar o **AmpliTube UA**[8] e o **Guitar Amp and Effects - Deplike**[9]. O AmpliTube UA funciona em conjunto com (e somente com) a interface *iRig UA* pois o processamento não ocorre no aparelho celular mas no iRig UA, com o AmpliTube UA servindo apenas de interface para o usuário. Por outro lado o Deplike reconhece outras interfaces de áudio, mas parte dos efeitos que ele disponibiliza está atrás de pagamentos feitos no aplicativo.

Aplicativos que utilizam a entrada de microfone são mais comuns de serem encontrados. A qualidade de áudio assim como a resposta de latência desses aplicativos variam bastante. São encontrados desde aplicativos com grande delay e qualidade sonora ruim, como o **iRig Distortion**, passando por aqueles com qualidade boa e latência razoável, como o **Delay Effects**[10], chegando até os com qualidade boa e delay baixo, como **ToneBridged**[11], quando comparando o tempo de latência no mesmo aparelho, pois pode variar com a versão do Android e o tipo de hardware.

1.3.3 Aplicativos que mesclam os sensores com processamento

Sensors2PD[12] (6)e UrMus[13](5) são dois exemplos de aplicativos que fazem a mescla entre sensores com o processamento do áudio. Sensors2PD utiliza patches[14] em puredata para fazer o controle sonoro, porém sua utilização em Android traz alguns problemas: o patch não pode ser editado com facilidade no celular, não se pode trocar o patch com facilidade e não se pode executar múltiplos patches que fazem processamentos diferentes ao mesmo tempo.

[8] iRig UA: http://www.ikmultimedia.com/products/amplitubeua/

[9] Deplike: http://deplike.com

[10] https://play.google.com/store/apps/details?id=com.delayeffects&hl=en

[11] https://play.google.com/store/apps/details?id=com.ultimateguitar.tonebridge&hl=en

[12] `https://github.com/deusanyjunior/Sensors2PD`

[13] `http://urmus.eecs.umich.edu`

[14] *Patch* é a denominação usual para programas escritos em PureData.

UrMus é um protótipo da Universidade de Michigan voltado para apresentações multimídias. Algumas limitações desse protótipo são o fato de que o UrMus não disponibiliza todos os sensores possíveis (o que varia de dispositivo para dispositivo).

1.4 Estrutura da monografia

O capítulo 2 traz a teoria do processamento de sinais digitais, do processamento de áudio visando a criação de efeitos sonoros e uma discussão sobre diferentes tipos de conexões para o uso desses processamentos em um dispositivo Android.

O capítulo 3 traz um experimento para calcular a latência de algumas combinações interfaces/aplicativos testadas, a implementação do aplicativo desenvolvido nesse trabalho e a discussão dos resultados experimentais obtidos.

O capítulo 4 traz as conclusões sobre o trabalho e projetos futuros.

Capítulo 2

Fundamentação Teórica

2.1 *Processamento de Sinais digitais*

Será descrito um pouco do processo de digitalização de um áudio, processo fundamental para o processamento de sinais digitais.

2.1.1 *Digitalização*

Para trabalharmos com um sinal sonoro $f(t)$ dentro de um celular ou de um computador, precisamos primeiramente digitalizá-lo. O processo é feito por um dispositivo chamado conversor análogo digital, *ADC* do inglês *Analog-to-digital converter*.

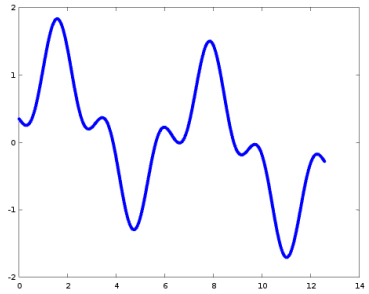

Figura 2.1: Sinal analógico $f(t) = \sin(t) - 0.5\sin(3t) + 0.35\cos(t/5)$.

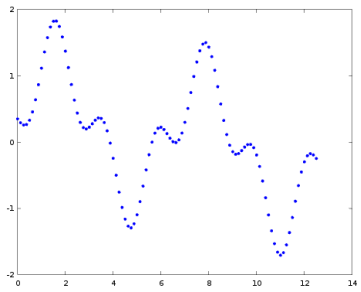

Figura 2.2: Sinal discreto de $f(t)$ com intervalo de amostragem $\Delta t = \frac{4\pi}{100}$.

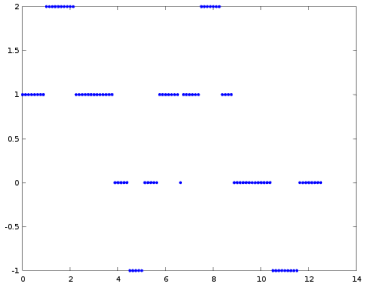

Figura 2.3: Sinal de $f(t)$, após amostragem e com quantização de 2 bits.

Um passo para a digitalização é a amostragem. Calcula-se $f(t)$ em pontos específicos de tempo. Vamos supor que $f(t)$ está definido no intervalo de tempo entre t_1 e t_2, de tal maneira que $t_1 \leq t \leq t_2$. Escolhemos um número N $\in \mathbb{N}^*$e definimos como intervalo de amostragem $\Delta t = \dfrac{t_2 - t_1}{N}$. Chama-se de taxa de amostragem ou *Sampling rate* o valor $1/\Delta t$.

Com isto podemos calcular os pontos de $f(t)$ em $t = t_1, t_1 + \Delta t, ..., t_1 + \Delta t * N$. Na figura 2.1 podemos ver um sinal contínuo $f(t)$ variando entre 0 e 4π, e na figura 2.2 vemos o mesmo sinal $f(t)$ amostrado com N = 100.

Outro passo é a quantização. Sistemas digitais não possuem precisão e nem memórias infinitas comparados a sistemas analógicos, então é necessário fazer um arredondamento dos valores de cada amostra. Considere como exemplo simplificador um computador que só consiga guardar 2 bits por amostra; nesse caso cada amostra só poderia ser representada através de 4 diferentes valores, e o sinal discreto da figura 2.2 iria ser quantizado como o da figura 2.3.

2.1.2 Aliasing e teorema da amostragem de Nyquist–Shannon

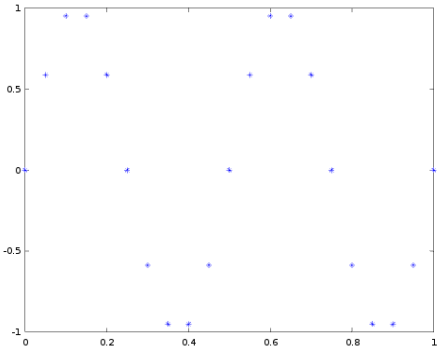

Figura 2.4: Sinal digitalizado amostrado a 20 Hz. Fonte: (3).

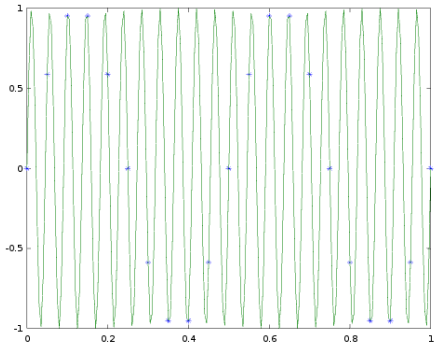

Figura 2.5: Sinal digitalizado amostrado a 200 Hz. Fonte: (3).

Aliasing ocorre quando dois sinais distintos não conseguem ser mais diferenciados após a amostragem. Por exemplo, considere as funções $f(t) = sen(4\pi t)$ e $g(t) = sen(44\pi t)$; se a amostragem for feita a 20 Hz, as mesmas amostras poderiam ser obtidas de quaisquer uma das duas funções, como podemos ver na figura 2.4, ao passo que a figura 2.5 exibe corretamente o gráfico da função $g(t)$.

Para evitar que o *Aliasing* ocorra o teorema da amostragem de Nyquist–Shannon indica que devemos fazer com que a taxa de amostragem seja pelo menos duas vezes maior do que a máxima frequência presente no sinal analógico. No exemplo anterior, a representação digital da função $g(t)$ exigiria ao menos 88 Hz de taxa de amostragem.

Um exemplo deste teorema na prática é a amostragem a 44.1 kHz de CDs de música. A frequência máxima que pode ser reproduzida nesse caso é de 22050 Hz, um pouco a mais do que a máxima frequência humanamente audível, de 20000 Hz.

2.2 Efeitos musicais

Efeito musical é uma alteração proposital no áudio usado por músicos para dar uma coloração diferente ao som. Os efeitos podem ser utilizados tanto no domínio analógico quanto no domínio digital. Analisaremos dois tipos de efeitos, os de delay e os de distorção.

2.2.1 Efeitos de delay

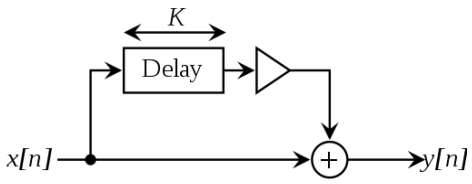

Figura 2.6: Uma implementação de filtro 'pente' (comb). Fonte da imagem: Wikipedia[1]

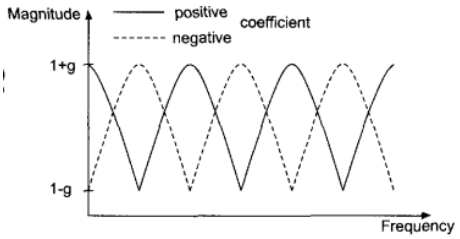

Figura 2.7: Resposta em frequência do filtro 'pente' (comb). Fonte da imagem: Digital Audio Effects (4)

Quando um músico toca, não percebemos apenas o som vindo diretamente do instrumento, mas também ouvimos o som refletido no chão, no teto ou em outras superfícies refletoras. Dependendo da distância e dos materiais de que são feitas estas superfícies temos sensações diferentes; por exemplo, quando elas estão muito distantes, tem-se a impressão de dois sons distintos, causando um efeito de eco, por outro lado quando estão próximas ouvimos um único som, podendo sofrer variações na amplitude pela alteração na fase.

A ideia do som refletido pode ser implementada através de um ou mais filtros *comb*. São filtros que somam um sinal atrasado porém idêntico ao sinal original. O nome vêm da modificação que ocorre na resposta da frequência, na qual o gráfico lembra um pente, como visto na figura 2.7. Sua equação é dada por:

$$y(n) = x(n) + g * x(n - K)$$

Sendo $x(n)$ o sinal original, $x(n - k)$ um sinal com atraso em K amostras e g uma constante que multiplica a amplitude do sinal realimentado.

[1] `https://en.wikipedia.org/wiki/File:Comb_filter_feedforward.svg`

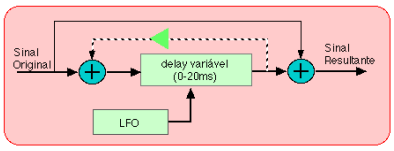

Figura 2.8: Diagrama de uma possível implementação de Flanger. Fonte da imagem: Curso da ECA/USP[2].

Para simular um efeito de eco, o atraso do sinal deve ser maior do que 50 ms. Para um efeito de Flanger (Figura 2.8) o atraso do sinal é variável, oscilando em até 20 ms, e para isto o atraso instantâneo é controlado por um oscilador de baixa frequência (LFO, do inglês Low Frequency Oscillator).

O efeito de reverberação tenta simular uma sala por completo, com todas as reflexões. Para isto utiliza-se um conjunto de filtros comb, cada um com um atraso diferente em relação ao outro.

2.2.2 Distorções

Distorção é a deformação de algum modo do áudio do som.

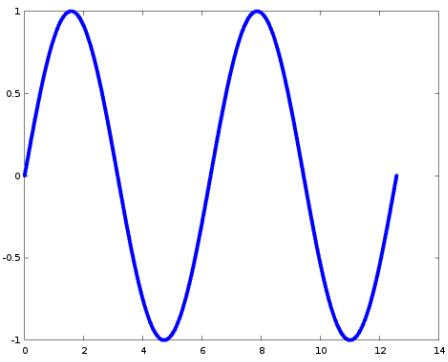

Figura 2.9: Um sinal $x(t) = sin(t)$.

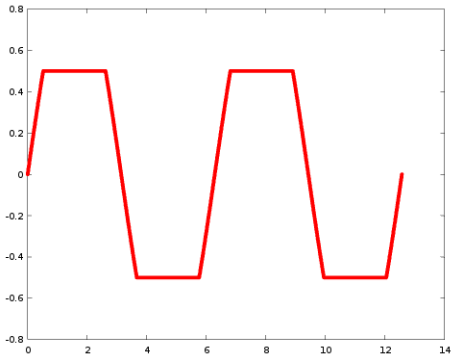

Figura 2.10: O sinal $x(t)$ com HardClip com L = 0.5.

Distorção HardClip ocorre quando se tem um limiar L e o sinal $x(t)$ de tal forma que:

$$HardClip(t) = \begin{cases} -L & x(t) \leq -L \\ x(t) & -L \leq x(t) \leq L \\ L & L \leq x(t) \end{cases}$$

[2] `http://www2.eca.usp.br/prof/iazzetta/tutor/audio/efeitos/effx.html`

Esse tipo de alteração direta da forma de onda resulta num acréscimo de harmônicos em relação ao sinal original, conferindo maior brilho ao som.

2.2.3 Processamento em bloco em Android

O processamento de áudio em Android é efetuado através de *buffers* que geralmente são pequenos, por limitações relativas ao hardware. Se uma aplicação que processa áudio demora muito para completar um buffer criará um chiado parecido com o de uma "pipoca estourando em uma panela"(popcorn Effect)[1].

Com isto os efeitos implementados têm que ser particionados em blocos, requerendo o menor número de amostras possíveis em seu processamento.

[1] `https://www.youtube.com/watch?v=PnDK17zP9BI`

Capítulo 3

Desenvolvimento

O objetivo principal do projeto é analisar uma forma de ampliar a sonoridade de contrabaixos elétricos utilizando aparelhos Android. Para isto, foram analisadas as formas nas quais um instrumento musical pode ser conectado em um celular. Em seguida, fez-se um levantamento de aplicativos que utilizam o Android como processador de sinais de áudio a fim de se descobrir o tempo médio de latência. Por fim, implementou-se uma aplicação que utiliza sensores embutidos no Android como controladores de efeitos sonoros.

3.1 Tipos de conexões de áudio

Há duas maneiras distintas de se conectar um instrumento musical em um aparelho Android. A primeira utiliza uma interface de áudio conectada à entrada micro USB. A segunda utiliza a entrada de microfone.

3.1.1 Entradas micro USB

A *iRig UA* [1] é uma interface e processador de áudio, feita para Android, que converte o sinal mono de um instrumento em uma saída micro USB. Porém há dois inconvenientes para a sua utilização. Por ser uma placa processadora acaba tendo um custo elevado de R$800,00 se comparado a outras interfaces mais simples, como a UCA22, behringer U-Phoria ou iMic griffin, que variam entre R$50,00 a R$200,00 (valores de novembro/2017). Além disso essa placa só funciona com seu programa oficial, o *AmpliTube UA*.

Estas interfaces mais simples, apesar de serem mais baratas, acabam trazendo outros dois problemas. O primeiro são suas saídas USB, que acabam requerendo um cabo

[1] `http://www.ikmultimedia.com/products/irigua//`

adaptador USB *On-the-go* (USB *OTG*). O segundo é que essas placas não possuem necessariamente um driver próprio para rodar no Android.

3.1.2 Entrada de microfone

O maior problema ao se usar a entrada de microfone para se conectar um instrumento musical em um dispositivo Android é a impedância. Para tentar contornar isto, podemos utilizar um adaptador como o iRig[2], que tem a vantagem de ser relativamente barato, custando em torno de R$20,00 (em novembro/2017).

3.2 Experimento de latência

O tempo de latência é o intervalo entre o músico tocar uma nota e o aplicativo retornar o som correspondente processado. Quanto maior a latência, pior o aplicativo, pois fará o músico perder a conexão entre gesto instrumental e som musical.

3.2.1 Sistemas e aplicativos testados

Os testes foram realizados utilizando duas versões diferentes do Android: a versão 4.3, Jelly Bean, lançada em 2012, e a versão 7.0, Nougat, lançada em 2016. Foram testados oito diferentes aplicativos: *Delay Effects, Deplike, Faust App, Guitar Effects, Guitar FX, iRig Distortion, ToneBridge, sensors2Music e Sensors2PD.* Foram utilizadas duas interfaces de áudio diferentes para a conexão micro USB, a iMic Griffin[3]. e a UCA22[4].

Os aplicativos *Delay Effects, Guitar Effects, Guitar FX, iRig Distortion* e *Sensors2PD* foram selecionados por rodar nas duas versões escolhidas do Android. O aplicativo *Deplike* foi escolhido por processar o som tanto pela entrada micro USB quanto pela entrada de microfone. *Faust app* foi selecionado porque a linguagem em que foi escrito, Faust, gera APK's diretamente. Por fim, *ToneBridge* foi escolhido por utilizar uma biblioteca chamada SuperPowered, que facilita a comunicação e utilização de funções de áudio em Android.

[2] `http://www.ikmultimedia.com/products/irig1/`

[3] `https://griffintechnology.com/us/imic`

[4] `http://www.music-group.com/Categories/Behringer/Computer-Audio/Audio-Interfaces/UCA222/p/P0A31`

3.2.2 Set Up

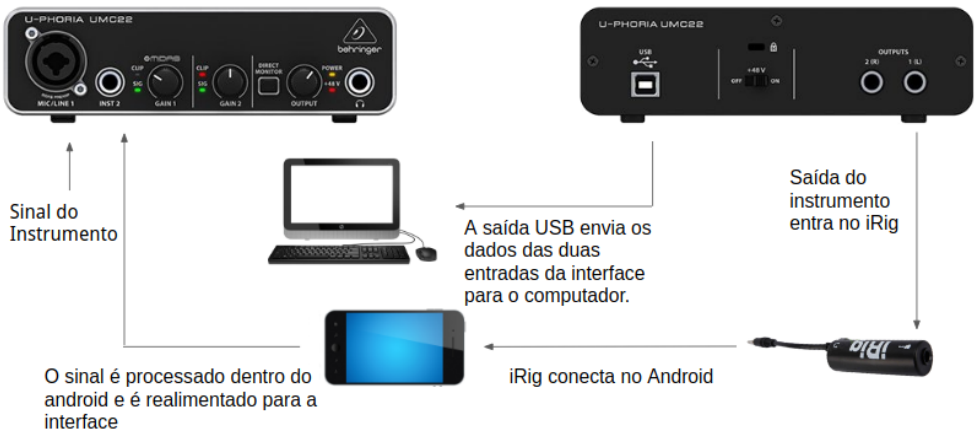

Figura 3.1: Set up do experimento, utilizando um iRig

Para a gravação dos experimentos utilizou-se uma interface de áudio *U-Phoria UMC22 Behringer*. Essa interface possui duas entradas de áudio, duas saídas (uma para cada entrada) e um conector USB.

O cabo de som do instrumento se conecta ao primeiro canal de entrada da interface. A saída deste canal alimenta a entrada do iRig, como ilustrado na figura 3.1, ou a outra interface que se conecta ao celular. A saída de som do celular passa a ser conectada ao segundo canal de entrada da interface U-Phoria.

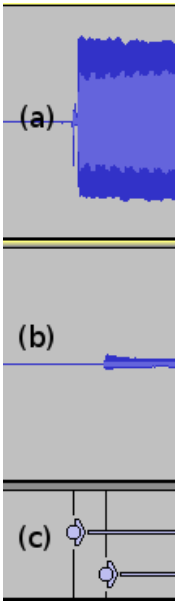

Figura 3.2: Um exemplo de gravação no Audacity. (a) A gravação do sinal do instrumento (b) O sinal advindo do celular. (c) O tempo de início de cada nota

A gravação do áudio dentro do computador é feita com o programa de gravação Audacity. Dentro do programa é possível separar as entradas da interface em duas faixas diferentes de áudio e em cada faixa passar um detector automático de notas. Na figura 3.2 podemos identificar a primeira faixa, vinda do instrumento (a), a segunda faixa, vinda do celular (b) e por último uma faixa que identifica o início de cada nota (c).

O detector de início de nota funciona bem nesta instância do problema pois o som produzido pelo contrabaixo tem um ataque rápido e cheio de energia, lembrando muito o de um instrumento percussivo.

Para a gravação, tocou-se uma nota no instrumento, e após deixá-la soar por alguns segundos, abafou-se o som para criar um intervalo

de silêncio entre cada nota. Este processo foi repetido cinco vezes para cada configuração, obtendo-se a média da latência ao final do processo.

3.3 Implementação da aplicação

Toda a implementação mencionada pode ser encontrada em: `https://github.com/felipeocm/sensors2Music`

3.3.1 Linguagens utilizadas

Java é uma linguagem orientada a objetos muito utilizada para o desenvolvimento de aplicativos em Android. Entretanto, ela possui alguns problemas ao tratar de processamento em tempo real de áudio.

O primeiro problema é que o código gerado em Java não é compilado diretamente para código nativo, mas sim transformado em bytecode para ser interpretado por uma Java Virtual Machine (JVM). Isto acaba adicionando uma camada a mais de processamento se comparado a uma linguagem que gera código diretamente para linguagem de máquina. O segundo problema deriva do fato de Java possuir um *Garbage Collector* automático, que pode ser acionado a qualquer instante independentemente de controle do usuário, atrapalhando o processamento.

Por estes problemas, optou-se por desenvolver o processamento de áudio em C/C++ através da Android Native Development Kit (NDK), utilizando Java para situações não críticas, como a interface com o usuário ou o tratamento e normalização dos sensores. Para fazer a conexão entre os dois pólos, utilizou-se a Java Native Interface (*JNI*).

3.3.2 Classes utilizadas

Foi criada uma extensão para a classe *Sensores* chamada *Musical Sensor*, pois a classe base não salva a quantidade de eixos total que um sensor possui.

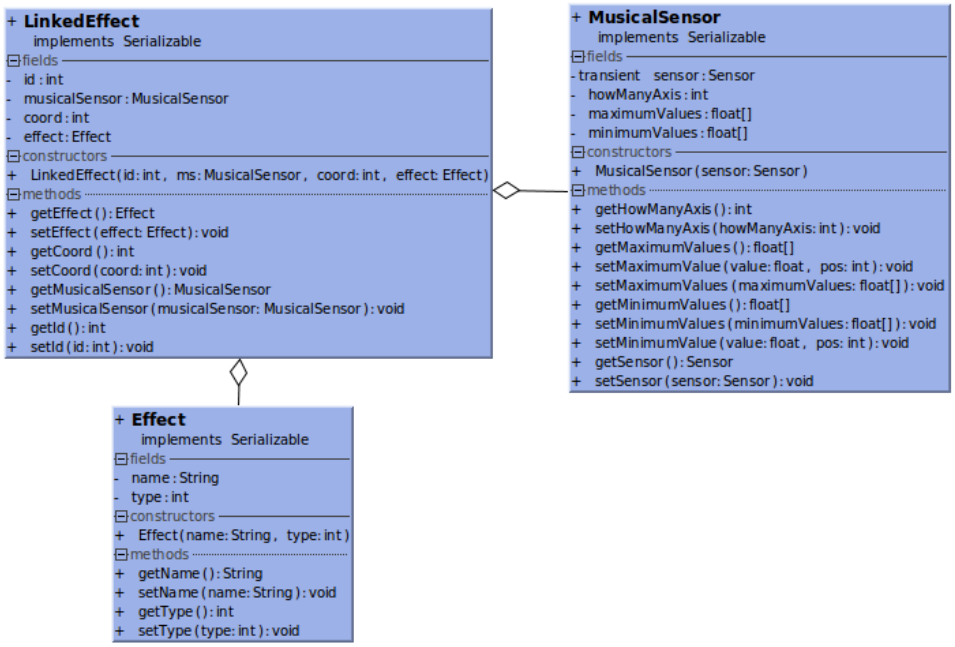

Figura 3.3: Classes criadas para o controle de efeitos por sensores

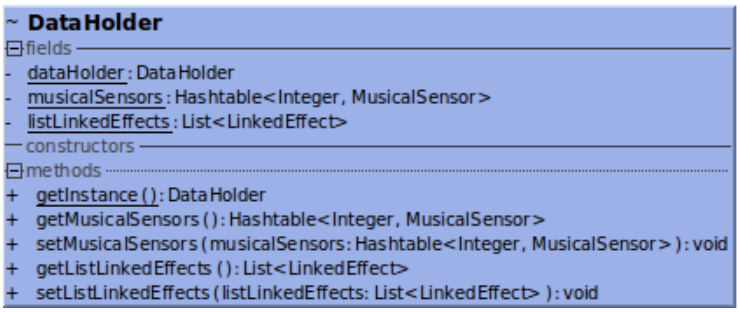

Figura 3.4: Singleton que guarda as informações enquanto o programa está funcionando

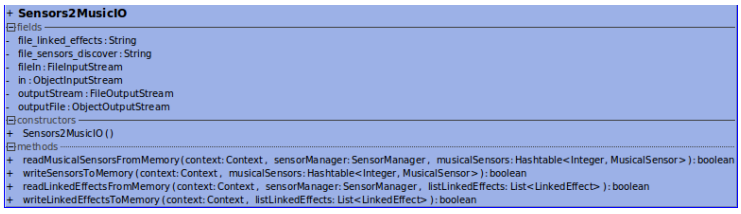

Figura 3.5: Classe que faz a comunicação entre os dados salvos no celular e no programa

3.3.3 *Tela Principal*

3.3.4 *Elementos visuais da tela*

Ao iniciar o aplicativo pela primeira vez, os botões *Link Effects* e *Calibrate* se encontram desabilitados, como mostra a figura 3.3. Isto se deve ao fato de que ainda não há dados

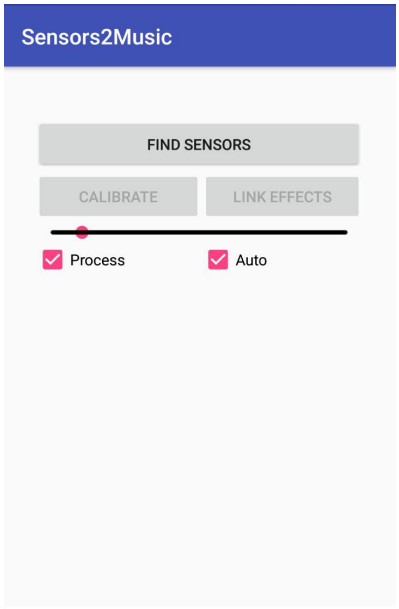

Figura 3.6: Tela principal quando não há sensores registrados

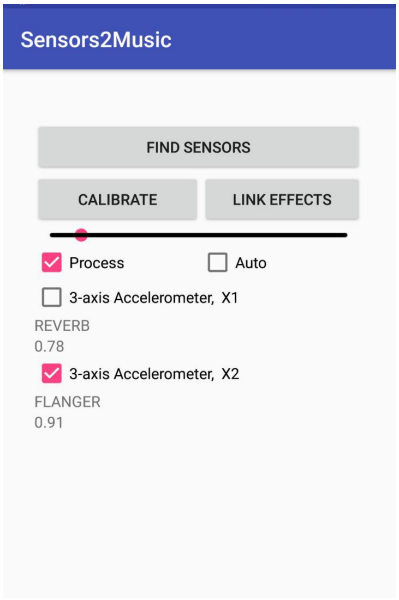

Figura 3.7: Tela principal com um sensor ativo

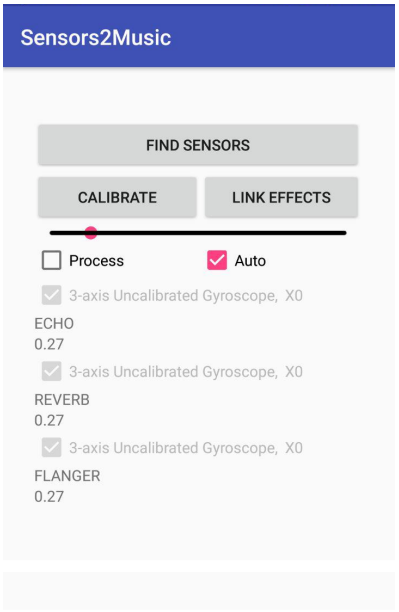

Figura 3.8: Tela principal com o bypass ligado

sobre nenhum Musical Sensor, isto é, nenhum registro contendo os sensores com o total de eixos que possuem. Para ativá-los, é necessário ir à tela *Find Sensors* ao menos uma vez.

Cada botão representa uma tela nova na aplicação: o *Find Sensors* leva à tela para encontrar novos sensores, o *Calibrate* serve para a calibração dos sensores já encontrados e *Link Effects* faz a conexão entre um eixo de um sensor e um efeito.

O *Slider* presente controla um fator de ganho, que varia entre 0 e 1000 e multiplica a amplitude de todo o sinal, permitindo o aumento ou diminuição do volume sonoro.

A caixa de seleção *Process* funciona como uma chave liga/desliga para todos os efeitos/sensores. Ao ser desativada, o áudio de saída será idêntico ao da entrada, sem qualquer tipo de processamento. Ao ser ativada, o áudio passará pelo processamento do Slider (ganho), além dos demais efeitos ativados. A caixa de seleção *Auto* modifica o processo de normalização pelo qual os sensores passarão.

Por fim, na parte inferior se localizam as caixas com as conexões dos efeitos. Nelas estão registrados o nome do sensor, o eixo sendo utilizado, o nome do efeito e qual a medição atualmente usada para o processamento do efeito.

3.3.4.1 Coleta de dados dos sensores

No Android, um sensor envia sua medições utilizando uma dentre quatro maneiras diferentes, sendo elas:*One-Shot*, *Special*, *On change* e *Continuous*:

- Sensores *One-Shot* (tiro único) enviam a medição uma única vez antes de serem desativados, precisando ser instanciados novamente para reenviarem outra medição. Um exemplo é o detector de *Signficant Motion*, que após detectar um movimento brusco se desativa para não enviar falsos positivos.

- Sensores *Special* (especiais) não geram medições de uma forma padronizada, mas cada sensor envia suas medições de uma maneira distinta. Por exemplo, o detector de passos gera dados apenas quando detecta um passo.

- Sensores *Continuous* (contínuos), como por exemplo os sensores de acelerômetro e de giroscópio, geram medições a partir de uma taxa de amostragem quase fixa.

- Sensores *On-change* (de mudança) geram medições a cada vez que o sensor detecta uma mudança de estado. Por exemplo, um sensor de proximidade gera medições conforme um usuário se aproxima ou se afasta do celular.

O Android oferece a opção de ajustar tanto a resolução quanto a amostragem. A resolução varia entre alta, média e baixa enquanto a amostragem pode variar entre *fastest, game*, normal e *user interface.* Uma taxa de amostragem rápida (fastest) com uma alta resolução implica num maior gasto de energia enquanto uma taxa de amostragem lenta (user interface) com uma resolução baixa implica em um baixo custo de energia. Porém estes parâmetros acabam sendo mais sugestões do que seguidos a risca, devido a grande variância entre aparelhos e sensores. Também não fica claro na documentação[3] se o que se modifica é a acurácia ou a resolução, porém a acurácia é geralmente limitada pelo hardware, não podendo ser configurada pelo software.

O uso de sensores *One-shot* acabou sendo descartado por serem desativados a cada relato, dificultando sua integração no contexto de efeitos de áudio. Todos os sensores instanciados são escolhidos com resolução alta, para tentar diminuir a quantidade de medições erradas.

[3] `https://developer.android.com/reference/android/hardware/SensorManager.html#SENSOR_STATUS_ACCURACY_HIGH`

Os dados coletados não são enviados diretamente para o controle do áudio, mas passam antes por um processo de filtragem passa-baixas, que tem como objetivo principal minimizar o efeito de movimentos bruscos não voluntários ou eliminar medições fora do padrão (outliers). A filtragem consiste no uso da média aritmética dos vinte ultimos relatos do sensor, atualizando o resultado a cada novo relato produzido.

3.3.4.2 Normalização

Após a filtragem dos dados dos sensores, ocorre a normalização. Esse processo é feito de duas maneiras distintas, uma que chamaremos de automática e a outra de fixada.

A normalização automática é um processo de auto-ajuste: a partir do momento em que são enviados os dados dos sensores, vão sendo salvos os valores máximo e mínimo produzidos pelo sensor. Conforme o usuário se movimenta mais, esses valores vão se dilatando, mudando assim os extremos considerados pela normalização. O processo se reinicia quando o usuário escolhe outra configuração de sensores-efeitos.

A normalização fixada precisa de um processo de pré-calibração. Na tela de calibração o usuário indica quais serão os pontos máximos e mínimos de um determinado sensor que ele pretende utilizar. Enquanto o usuário toca, estes pontos nunca serão alterados, mantendo o mapeamento até ocorrer outro processo de calibração.

Para se alternar entre estes dois modos é utilizado a caixa de seleção *Auto*: quando ela está ligada o processo utilizado é o automático, do contrário ocorre o processo de calibragem fixada.

3.3.4.3 Processamento de áudio

O processamento do áudio se inicia quando o programa é ligado e se desativa quando o usuário troca de tela. Os efeitos são processados na ordem em que são listados, levando em conta que a ordem de processamento afeta o resultado sonoro final.

3.3.5 Tela Find Sensors

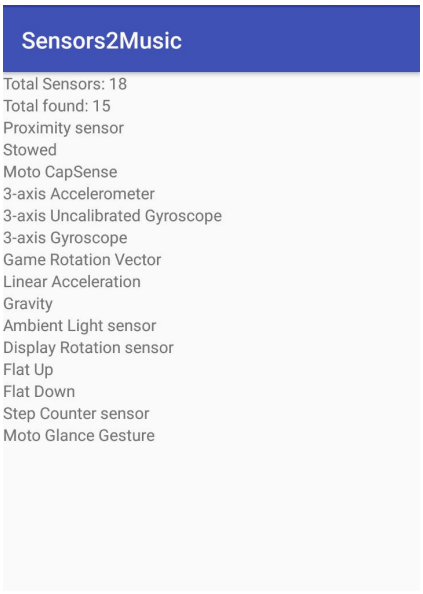

Figura 3.9: Tela do Find Sensors

3.3.5.1 Elementos visuais

A tela *Find Sensors* é visualmente simples: no topo ela mostra a quantidade total de sensores que o celular possui, já desconsiderados os sensores *One-shot*.

3.3.5.2 Encontrando sensores

Ao entrar em Find Sensors, todos os sensores que não foram encontrados em outras buscas são ativados. Quando o sensor envia uma medição pela primeira vez descobrimos a quantidade total de eixos, possibilitando a criação de um *Musical Sensor*. Após sua criação, o sensor é listado na lista de sensores descobertos e é (temporariamente) desativado.

3.3.6 Tela Calibrate

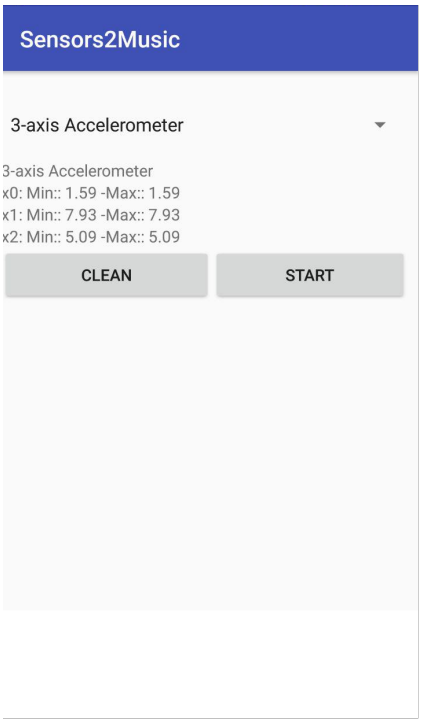

Figura 3.10: Tela para calibração dos sensores

3.3.6.1 Elementos visuais

Há um menu *drop-down* que, ao selecionar um item, mostra uma janela contendo o nome do sensor, seus eixos e os respectivos valores máximo e mínimo.

O botão *clear* limpa as medições do sensor atual, enquanto o botão *start/stop* inicializa o registro dos pontos de máximo e mínimo para a normalização automática.

É importante salientar que deve-se calibrar cada sensor separadamente, pois a calibração de um sensor pode interferir na de outro. Por exemplo, um movimento brusco não iria mudar os valores de um sensor de luminosidade mas afetará bastante um sensor de aceleração linear.

3.3.7 Link Effects

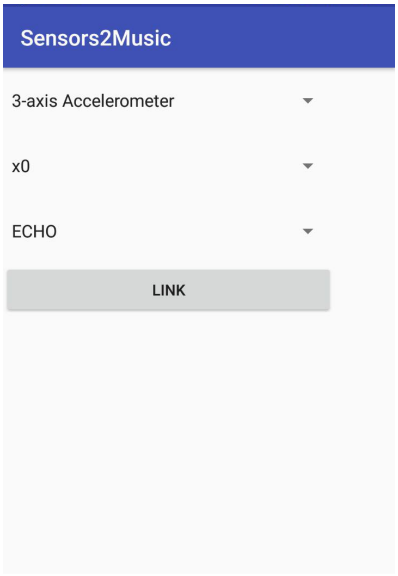

Figura 3.11: Tela Link Effects

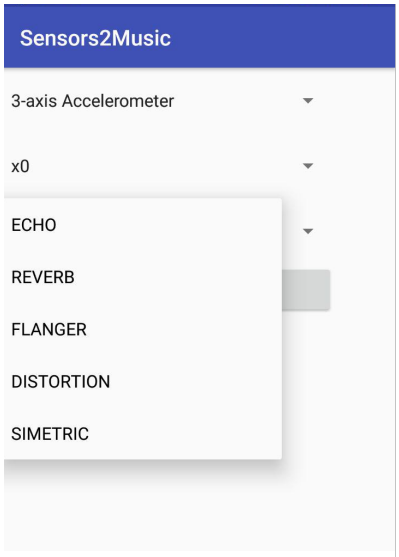

Figura 3.12: Efeitos disponíveis

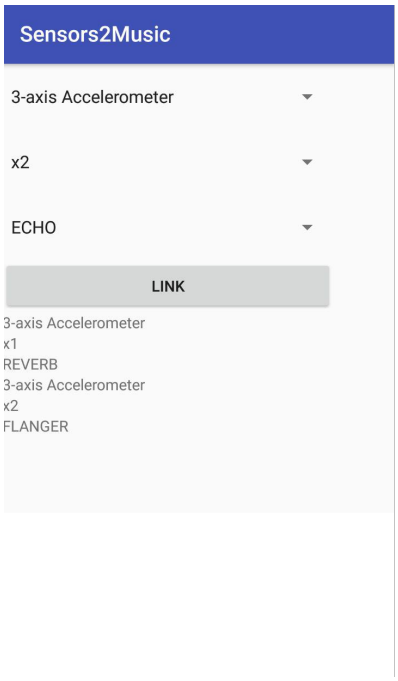

Figura 3.13: Tela link Effects com efeitos ligado a sensores

3.3.7.1 Elementos visuais

Neste tela é feita a conexão entre um sensor e um efeito musical. No primeiro menu drop-down o usuário escolhe qual sensor deseja utilizar, no segundo define o eixo do sensor e no terceiro escolhe o efeito, confirmando a ação com o botão Link. Ao confirmar a ação, aparecerá a combinação escolhida, conforme mostra a figura 3.10.

A ordem que o usuário escolher conectar os sensores/efeitos será a ordem utilizada para o processamento do áudio. Um efeito não pode ser utilizado mais de uma vez, porém um mesmo sensor pode controlar múltiplos efeitos.

3.3.8 Testes com o instrumento

Figura 3.14: Um possível setup do instrumento com o celular

Figura 3.15: Vídeo da utilização do celular como sensor

O aplicativo ficou em exposição durante a Semana Nacional de Ciência e Tecnologia realizada durante os dias 23 a 28 de outubro de 2017 na USP. Para fazer uma demonstração ao vivo, utilizou o set-up visto na figura 3.11, sendo que sua utilização foi registrada em vídeo e disponibilizada no canal do autor no YouTube[1] (figura 3.12). O set-up tentou ser o mais expressivo possível. (8)

3.3.9 Resultados

O mapeamento de aplicativos realizado está sumarizado na tabela 3.1, onde podem ser vistos os efeitos de áudio disponibilizados por cada aplicativo, as versões mínimas do Android exigidas, a medição de latência realizada e o tipo de conexão de hardware entre o contrabaixo e o celular. Alguns aplicativos aparecem em várias linhas da tabela, correspondendo a diferentes combinações entre os diversos elementos mapeados.

[1] `https://www.youtube.com/watch?v=tjaF5RwB40Y`

Aplicativo	Efeitos	Android	latência(ms)	Conexão
Sensors2Music	No Effects	7.0	50	iRig
ToneBridge		7.0	55	iRig
Sensors2PD	No Effects	7.0	67.5	iRig
Deplike	Delay	7.0	70	iRig
Sensors2Music	Delay/Reverb/Distortion	7.0	100	iRig
Guitar FX	Distortion	7.0	145	iRig
Faust App	Compression	7.0	160	iRig
Faust App	Compression/WahWah/Phaser	7.0	180	iRig
iRig Distortion	Distortion	4.3	247.5	iRig
Guitar Effects	Booster	7.0	252	iRig
iRig Distortion	Distortion	7.0	267.5	iRig
Delay Effects	Flanger	7.0	282.75	iRig
Delay Effects	Flanger	4.3	285	iRig
Guitar Effects	Booster	4.3	293,4	iRig
Guitar Effects	Normal	7.0	297,5	iRig
Guitar Effects	Normal	4.3	302,5	iRig
Guitar FX	Distortion	4.3	383,4	iRig
Delay Effects	Distortion/Flanger/Delay/Chorus	4.3	516.6	iRig
Delay Effects	Distortion/Flanger/Delay/Chorus	7.0	577.5	iRig
Deplike	Delay	7.0	628.75	IMIC
Deplike	Delay	7.0	662.5	UCA
Sensors2PD	No Effects	4.3	800	iRig

Tabela 3.1 - Resultados do experimento da seção 3.2, por ordem de latência

3.3.10 Análise dos dados do experimento

Os valores de latência na quarta coluna na tabela 3.1 não devem ser tomados como referenciais absolutos, pois o próprio setup do experimento adiciona uma latência extra às medições, produzida pela interface de áudio e o cabeamento. Isso significa que existe a possibilidade de se obter latências menores do que 50 ms no Android 7.0 (Sensors2Music sem efeitos) ou 247.5 ms no Android 4.3 (iRig Distortion) quando for realizada uma conexão direta entre instrumento e celular.

A entrada USB acabou tendo uma performance pior em comparação com a entrada de microfone, chegando a cerca de 590 ms de diferença para o mesmo aplicativo (Deplike + Delay).

De um modo geral, os aplicativos tiveram uma performance melhor na versão mais recente do Android, com uma clara exceção do *Guitar Effects*, cujos resultados foram bem parecidos nas duas versões do aparelho. Sensors2PD foi o que teve a maior diferença de performance, reduzindo a latência observada em mais de 700 ms. *ToneBridge* simula a pedaleira da música como um conjunto só, não permitindo selecionar um efeito específico. Escolheu-se a canção "Sweet child'o mine" (Guns N' Roses) para o teste realizado. Sensors2Music teve uma performance boa comparativamente com os outros aplicativos, mostrando que a conexão com os sensores não atrapalhou o processamento de áudio.

Capítulo 4

Conclusão

4.1 Conclusões

O trabalho consistiu no estudo e implementação de técnicas relacionadas ao processamento de sinais digitais, como a geração de distorções no sinal, de uma análise experimental sobre aplicativos e seus respectivos tempo de latência, e do estudo sobre a plataforma Android, partindo da criação de um protótipo até a análise e processamento dos dados dos sensores. Todos estes elementos foram utilizados na construção de um sistema que permitisse o acoplamento de um contrabaixo elétrico a um celular Android visando o uso dos sensores embutidos no celular para controlar efeitos sonoros aplicados à saída do instrumento.

Dentre os problemas encontrados, podemos destacar a dificuldade em se desenvolver um método genérico para normalizar os sensores embutidos no celular. Por serem dezenas de marcas e modelos os dispositivos explorados, não há um caminho simples que permita tratar todos de uma maneira igual.

O principal objetivo, a criação de um protótipo que utiliza os sensores embutido em um celular para o controle de um processamento de áudio, foi bem sucedido. O aplicativo foi testado tanto por músicos quanto por pessoas leigas durante a semana de tecnologia da USP de 2017 e recebeu comentários positivos do público.

4.2 Trabalhos Futuros

Durante a realização do trabalho, notou-se que a tela Touchscreen não era a maneira mais prática para se ligar ou desligar efeitos, pois exigia do músico afastar uma de suas mãos do instrumento para acioná-la. Para contornar esta situação, um dos próximos

passos do projeto é a utilização de uma pedaleira feita em Arduino, na qual, através de uma conexão bluetooth, a pedaleira se conectaria ao aplicativo e passaria a oferecer ao músico a alternativa de controlar algumas opções de processamento com seus pés.

Referências Bibliográficas

[1] de Carvalho Junior and Mayer(2015)Antonio Deusany de Carvalho Junior and Thomas Mayer.Sensors2osc. EmSound and Music Computing Conference. Maynooth University.

[2] Thibodeau, J.; Wanderley, M.M. Trumpet Augmentation and Technological Symbiosis. Comput. Music J. 2013, 37, 12–25.

[3] Broughton, S. A. and Bryan, K. (2008) , Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing, John Wiley & Sons, Inc., Hoboken, NJ, USA. doi: 10.1002/9781118032442.answ

[4] Verfaille, V., Holters, M. and Zölzer, U. (2011) DAFX: Digital Audio Effects, Second Edition (ed U. Zölzer), John Wiley & Sons, Ltd, Chichester, UK. doi: 10.1002/9781119991298.ch1

[5] Essl, Georg. (2010). Urmus - An Environment For Mobile Instrument Design And Performance. International Computer Music Conference, ICMC 2010. .

[6] de Carvalho Junior(2014) Antonio Deusany de Carvalho Junior. Sensors2pd: Mobile sensorsand wifi information as input for pure data. Em 42nd International Computer Music Conference (ICMC) joint with the 13rd Sound & Music Computing conference (SMC).InternationalComputer Music Association.

[7] C. Roberts, Control: Software for end-user interface programming and interactive performance, 2011.

[8] Sidney Fels, Ashley Gadd, and Axel Mulder. 2002. Mapping transparency through metaphor: towards more expressive musical instruments. Org. Sound 7, 2 (August 2002), 109-126.