Processamento Digital de Sinais em Pedais Digitais de Instrumentos Musicais e Voz

1st Artur M. A. Novais
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brasil
artur.matos@discente.ufg.br

2nd Davi J. Teixeira
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brasil
davijesus@discente.ufg.br

3rd Luís R. S. de Lima
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brasil
luis_.lima@discente.ufg.br

4th Marcelo H. A. Pereira Sob°.
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brasil
alvesmarcelo@discente.ufg.br

***Abstract*—This work explores the implementation of musical instrument effects - reverb, chorus, delay, tremolo, and wah-wah - in digital audio signals through digital signal processing and mathematical fundamentals, with a practical approach to programming. The study focuses on the use of signal processing concepts and algorithms to apply sound effects to a musical instrument, emphasizing the practical applicability of these concepts. It begins with a discussion on the theoretical foundations of digital signal processing and the mathematical principles underlying musical instrument effects. Subsequently, the work addresses the implementation of these concepts to simulate the desired effects. This study aims to deepen theoretical knowledge in sound engineering and digital music. The results demonstrate how the intersection of technical knowledge can be creatively applied in music, opening new paths for sound design and musical performance.**

***Index Terms*—chorus, reverb, delay, guitarra, modulação, senoides, processamento de sinais**

I. Introdução

Neste artigo, abordaremos a implementação de efeitos de guitarra e voz em sinais de aúdio digital - como *reverb*, *chorus*, *delay*, *wah-wah* e *tremolo* - utilizando os princípios de processamento digital de sinais e fundamentos matemáticos. A ênfase é colocada na aplicação de teorias de processamento de sinais para a criação e modificação de efeitos sonoros em instrumentos musicais, proporcionando uma compreensão básica de como as técnicas digitais podem ser usadas para enriquecer a experiência musical. Exploraremos os algoritmos e conceitos matemáticos por trás desses efeitos populares, demonstrando como eles moldam o som dos instrumentos de maneiras distintas e impactantes. O estudo não se limita apenas ao aspecto teórico, mas também abrange a aplicação prática desses conceitos.

II. Fundamentos Teóricos

Teoria da Amostragem e Quantização

- Teorema de Nyquist-Shannon: Fundamental para a amostragem, este teorema afirma que um sinal contínuo pode ser completamente representado e reconstruído a partir de suas amostras se a taxa de amostragem for pelo menos o dobro da frequência máxima presente no sinal. Matematicamente, se um sinal tem frequência máxima f_{max}, então a taxa de amostragem f_{s} deve satisfazer $f_{\text{s}} > 2f_{\text{max}}$;
- Quantização: No processo de quantização, cada amostra de um sinal é arredondada para o valor mais próximo dentro de um conjunto finito de valores. Esse processo pode ser modelado matematicamente como uma função de arredondamento ou truncamento, introduzindo um erro de quantização, que é a diferença entre o valor real e o valor quantizado.

Algoritmos de Efeitos de Instrumentos Musicais

- *Reverb*: O efeito de reverberação é obtido pela convolução do sinal de entrada com uma resposta ao impulso. Matematicamente, a convolução é expressa por:

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} x[k]h[n-k] \tag{1}$$

  onde $x[n]$ é o sinal de entrada, $h[n]$ é a resposta ao impulso e $y[n]$ é o sinal de saída;
- *Chorus*: O *Chorus* utiliza uma combinação de atraso, modulação e mistura. O atraso modulado pode ser expresso como:

$$y[n] = x[n] + x[n - \Delta n(t)] \tag{2}$$

  onde $\Delta n(t)$ é uma função de atraso variável no tempo, geralmente uma função senoidal ou cossenoidal;
- *Delay*: O efeito *Delay* é uma forma de eco, onde o sinal é repetido após um tempo de atraso fixo. É matematicamente simples:

$$y[n] = x[n] + \alpha x[n - N_d] \tag{3}$$

onde N_d é o número de amostras de atraso e α é o fator de atenuação;

- *Wah-Wah*: O efeito *Wah-Wah* é criado por um filtro passa-banda com frequência central variável. A equação geral para um filtro passa-banda é:

$$H(f) = \frac{1}{1 + jQ\left(\frac{f}{f_c} - \frac{f_c}{f}\right)} \tag{4}$$

onde f_c é a frequência central e Q é a qualidade do filtro;

- *Tremolo*: O *Tremolo* modula a amplitude do sinal. Seja $m(t)$ a função modeladora (geralmente senoidal), então:

$$y[n] = x[n]m[n] \tag{5}$$

onde $m[n] = 1 + \alpha \sin(2\pi f_m n)$, α é a profundidade da modulação e f_m é a frequência da modulação.

III. Metodologia

A fase inicial envolveu uma análise detalhada de diversos pedais analógicos renomados, identificando suas principais características acústicas e mecanismos de manipulação do sinal. Esta análise detalhada dos circuitos dos pedais proporcionou *insights* valiosos sobre as funções desempenhadas por diferentes componentes, como filtros, amplificadores e moduladores. Com base nisso, selecionamos algumas técnicas de processamento digital de sinais para dar prosseguimento:

- Convolução para Efeitos de *Reverb*: Empregamos respostas ao impulso pré-gravadas para simular diferentes ambientes acústicos;
- Modulação para *Chorus* e *Tremolo*: Implementamos Osciladores de Baixa Frequência (LFOs) para induzir variações periódicas no sinal de áudio;
- Filtragem Dinâmica para *Wah-Wah*: Aplicamos filtros passa-banda cujas frequências centrais são moduladas por um envelope de sinal ou LFO;
- *Delay* Digital: Desenvolvemos algoritmos para atrasar o sinal com controles de tempo e feedback.

A. Ferramentas e bibliotecas

Utilizamos o Python devido à sua vasta gama de bibliotecas para processamento de áudio e matemática e também por causa de essa linguagem de programação estar presente em todo o nosso curso de graduação. As principais bibliotecas utilizadas incluem:

- *NumPy*: Para operações matemáticas de alto desempenho, especialmente manipulação de vetores, que é crucial para o processamento de sinais digitais e também utilizaremos a *NumPy* para trabalhar com funções matemáticas;
- *SciPy*: Utilizado para operações mais avançadas de DSP, como convoluções e transformadas de Fourier;
- *Librosa*: Uma biblioteca específica para processamento de áudio, útil para leitura, escrita e análise de arquivos de som;
- *Matplotlib*: Para visualizar os sinais de áudio digital e suas transformações ao longo do processo.

B. Implementação dos efeitos

Reverb: Implementamos o *reverb* por meio da convolução do sinal de entrada com uma resposta ao impulso pré-gravada. Utilizamos a função convolve da biblioteca *SciPy* para esta operação.

```
def reverb(self, IR_path, wet_level=0.5):
    ir, sr = librosa.load(IR_path, sr=44100)
    reverbed_audio = scipy.signal.convolve(self.audio, ir, mode='full')
    return reverbed_audio
```

Fig. 1. Código em *python* do efeito *Reverb*

Chorus: Para o efeito *chorus*, implementamos um LFO (Oscilador de Baixa Frequência) que modula um sinal de *delay*. O sinal de entrada é somado com uma versão atrasada e modulada de si mesmo.

```
def chorus(self):
    print('----------------------')
    depth = float(input('Digite a profundidade do efeito Chorus (número entre 0 e 1): '))
    modfreq = float(input('Digite a frequência de modulação do Chorus (< 20Hz): '))
    delay = float(input('Digite o atraso do Chorus (em segundos): '))
    feedback = float(input('Digite o valor de feedback do Chorus (número entre -1 e 1): '))
    print('----------------------')

    output_signal = np.zeros(len(self.audio))
    delay_samples = int(delay * self.sr)

    for n in range(len(self.audio)):
        mod = depth * np.sin(2 * np.pi * modfreq * n / self.sr)
        delay_index = n - delay_samples
        if delay_index < 0:
            delay_index = 0
        delayed_sample = self.audio[delay_index]
        output_signal[n] = self.audio[n] + feedback * delayed_sample * mod

    return output_signal
```

Fig. 2. Código em *python* do efeito *Chorus*

Wah-wah: O efeito *wah-wah* é tipicamente um filtro passa-banda com uma frequência central que se move para cima e para baixo. Essa variação na frequência central é comumente controlada por um LFO (Oscilador de Baixa Frequência). A implementação envolve a criação de um filtro passa-banda que muda suas frequências de corte ao longo do tempo, guiado pelo LFO.

Tremolo: O tremolo é um efeito que modula a amplitude do sinal de entrada, criando uma sensação de ”tremulação”. Isso é feito através de um LFO que ajusta o volume do sinal ao longo do tempo.

```python
def wahwah(self):
    print('----------------------')
    damp = float(input('Digite o fator de amortecimento do wahwah (< 0.5): '))
    minf = float(input('Digite a frequência mínima de corte central (~ 500Hz): '))
    maxf = float(input('Digite a frequência máxima de corte central (~ 5000Hz): '))
    wahf = float(input('Digite a frequência "wah" (~ 2000Hz): '))
    print('----------------------')
    output_signal = np.zeros(len(self.audio))
    outh = np.zeros(len(self.audio))
    outl = np.zeros(len(self.audio))
    delta = wahf / self.sr
    centerf = np.concatenate((np.arange(minf, maxf, delta), np.arange(maxf, minf, -delta)))
    while len(centerf) < len(self.audio):
        centerf = np.concatenate((centerf, centerf))
    centerf = centerf[:len(self.audio)]
    f1 = 2 * np.sin(np.pi * centerf[0] / self.sr)
    outh[0] = self.audio[0]
    output_signal[0] = f1 * outh[0]
    outl[0] = f1 * output_signal[0]
    for n in range(1, len(self.audio)):
        outh[n] = self.audio[n] - outl[n-1] -  2 * damp * output_signal[n-1]
        output_signal[n] = f1 * outh[n] + output_signal[n-1]
        outl[n] = f1 * output_signal[n] + outl[n-1]
        f1 = 2 * np.sin(np.pi * centerf[n] / self.sr)
    return output_signal
```

Fig. 3. Código em *python* do efeito *Wah-wah*

```python
def tremolo(self):
    print('----------------------')
    alph = float((input('Digite a profundidade do tremolo (número entre 0 e 1): ')))
    modfreq = float(input('Digite a frequência de modulação (< 20Hz): '))
    print('----------------------')
    output_signal = np.zeros(len(self.audio))
    for n in range(len(self.audio)):
        trem = 1 + alph * np.sin(2 * np.pi * modfreq * n / self.sr)
        output_signal[n] = trem * self.audio[n]
    return output_signal
```

Fig. 4. Código em *python* do efeito *Tremolo*

Delay: O efeito *delay* repete o sinal de entrada após um período de tempo definido. Esse tempo é ajustável e pode incluir um fator de decaimento para simular diferentes tipos de eco.

```python
def delay(self, delay_ms, feedback):
    delay_samples = int(self.sr * delay_ms / 1000)
    delayed_audio = np.zeros(len(self.audio) + delay_samples * feedback)

    for n in range(len(self.audio)):
        delayed_audio[n] += self.audio[n]
        for i in range(1, feedback + 1):
            if n + delay_samples * i < len(delayed_audio):
                delayed_audio[n + delay_samples * i] += self.audio[n] * (0.8 ** i)

    return delayed_audio
```

Fig. 5. Código em *python* do efeito *Delay*

Os efeitos desenvolvidos foram submetidos a alguns testes com sinais de áudio de baixo e guitarra para verificar se estão de acordo com o som proposto.

IV. Resultados e Conclusões

Os testes realizados com os sinais de áudio de baixo e guitarra demonstraram que os efeitos desenvolvidos alcançaram uma reprodução sonora de média qualidade. Tecnicamente, os efeitos apresentaram características sonoras correspondentes às expectativas baseadas nos pedais analógicos estudados. Através de análises subjetivas, percebemos que:

- O *Reverb* conseguiu criar ambientes acústicos realistas, com diferentes respostas ao impulso oferecendo uma variedade de texturas sonoras;
- O *Chorus* e o *Tremolo* mostraram modulações claras e consistentes, realçando os timbres dos instrumentos com variações rítmicas e tonais;
- O *Wah-Wah* efetivamente imitou o efeito vocal característico com pouca precisão;
- O *Delay* forneceu repetições precisas, onde o controle do tempo e do decaimento permitiu simular desde ecos sutis até efeitos mais intensos.

Subjetivamente, os nós avaliadores relatamos uma experiência sonora quase satisfatória. Chegamos a conclusão de que os pedais analógicos utilizados por músicos profissionais oferecem uma experiência muito melhor que os pedais digitais implementados por algoritmos.

References

[1] FRANÇA, André Wagner. Uso de processamento digital de áudio na implementação de efeitos em instrumentos musicais. 2015.

[2] RIBEIRO, Renan Airton Batista; GOMES, Otávio de Souza Martins. Desenvolvimento de uma pedaleira de efeitos para guitarra utilizando microcontrolador de plataforma aberta.

[3] gened1080/guitar-effects. Disponível em: ¡https://github.com/gened1080/guitar-effects¿. Acesso em: 19 jan. 2024.

[4] Reverb VS Delay or Echo - What Is The Difference? Explanation, Comparison and Demonstration. Disponível em: ¡https://www.youtube.com/watch?v=SCUUu96PYjo¿.