Metode i filteri za filtriranje zvuka u realnom vremenu

Naučno izračunavanje - Projekat

Matei Jon Stanču, 1137/2015

2018-09-04

Sadržaj

1 Uvod

U ovom radu predstavljeno je konstruisanje filtera i njihova primena u obradi zvuka za aplikacije koje rade u realnom vremenu. Za realizaciju filtriranja u realnom vremenu bilo je neophodno koristiti razlicite spoljašnje alate koje olakšavaju implementaciju komponenti kao što su čitanje iz datoteke, dekodiranje komprimovanih podataka i upravljanje zvučnom kartom. O dizajnu i implementaciji ovih komponenti neće biti reči u ovom radu jer izlaze iz okvira teme. Od interesa su pre svega metode filtriranja, dizajn i implementacija filtera, zatim i tehnike koje omogućavaju profinjavanje kvaliteta filtriranja kao što su prozorske funkcije. U nastavku su dati detaljni opisi ovih metoda.

2 Metode filtriranja

Postoje dva opšta načina filtriranja signala, filtriranje u vremenskom domenu i filtriranje u frekvencijskom domenu. Filtriranje u vremenskom domenu podrazumeva direktnu primenu operacije konvolucije nad ulaznim signalom i filterom, dok filtriranje u frekvencijskom domenu podrazumeva računanje konvolucije ulaznog signala i filtera pomoću brze Furijeove transformacije. U nastavku su dati opisi navedenih metoda.

2.1 Konvolucija

U zavisnosti od problema mogu se koristiti različiti pristupi računanju konvolucije dva signala. Razlikujemo cirkularnu i linearnu konvoluciju, pri čemu cirkularna se koristi kada je signal koji se obrađuje periodičan ili kada se prametri filtera ne menjaju u toku filtriranja, zbog toga je za potrebe ovog rada izabrana linearna konvolucija. Konvolucija je matematička operacija definisana, u kontekstu obrade jednodimenzionih signala, sledećom formulom:

$$(f * g)_i = \sum_{j=0}^{n-1} f_j g_{i-j}$$

Prethodna formula opisuje cirkularnu konvoluciju, dok za linearnu konvoluciju postoje različite formule u zavisnosti od tačke gledišta. Jedna perspektiva algoritma bi bila iz ugla ulaza, koja opisuje doprinos svakog uzorka iz ulaznog signala uzorcima koji čine izlazni signal:

```
input_side_convolution(x, h, n, m):
    y ← array(n+m-1, 0)
    for i ← 0 to n do:
        for j ← 0 to m do:
            y[i+j] ← y[i+j] + x[i]*h[j]

    return y
```

Drugi ugao posmatranja algoritma linearne konvolucije bi bio iz ugla izlaza, koji svakom uzorku iz izlaznog signala pridružuje zbir uzorkaka iz ulaznog signala koji ga čine:

```
output_side_convolution(x, h, n, m):
    y ← array(n+m-1, 0)
    for i ← 0 to n+m-1 do:
        y[i] ← 0
        for j ← 0 to m do:
            if i-j >= 0 and i-j < 80:
                y[i+j] ← y[i+j] + x[i]*h[j]

    return y
```

U suštini linearna konvolucija se može računati cirkularnom tako što se signal proširuje za toliko nula kolika je dužina filtera i zatim nad takvim nizom primenjuje cirkularna konvolucija.

2.2 Brza Furijeova transformacija

Često su za filtriranje neophodni filteri koji imaju veću rezoluciju u frekvencijskom domenu, što onemogućava da se, zbog kvadratne složenosti algoritama računanja konvolucije, filtriranje odvija u realnom vremenu. Tada se za računanje konvolucije koristi brza Furijeova transformacija i filtriranje se vrši u frekvencijskom domenu. Filtriranje se ostvaruje tako što se brzom Furijeovom transformacijom ulazni signal i filter transformišu u njihove frekvencijske odgovore pa se zatim frekvencijski odgovori množe pokoordinatno. U nastavku je dat pseudokod algoritma brze Furijeove transformacije:

```
1  fft(f,n,s):
2      if n=1 then
3          f̂_0 ← f_0
4      else
5          f̂_0, ..., f̂_{n/2-1} ← fft(f,n/2,2s)
6          f̂_{n/2}, ..., f̂_{n-1} ← fft(f+s,n/2,2s)
7          k ← 0
8          while k < n/2 - 1 do
9              f̂_{k+n/2} ← f̂_k - w^k f̂_{k+n/2}
10             f̂_k ← f̂_k + w^k f̂_{k+n/2}
11             k ← k + s
12         end
13     end
14     return f̂_0, ..., f̂_{n-1}
```

Figure 1: Brza Furijeova transformacija

I u slučaju filtriranja u frekvencijskom domenu neophodno je proširiti ulazni signal nulama

ako bi se želela računati linearna konvolucija.

$$y_k[n] = \text{IFFT}\left(\text{FFT}\left(x_k[n]\right) \cdot \text{FFT}\left(h[n]\right)\right)$$

2.3 Preklapanje sa dodavanjem

Preklapanje sa dodavanjem je tehnika za obradu signala koja se primenjuje kada je veličina ulaznog signala prevelika da bi se primenila konvolucija direktno ili pomoću brze Furijeove transformacije na celom signalu, što cini ovu metodu adekvatnom za filtriranje u realnom vremenu. Umesto toga polazni signal se deli na niz blokova koji se nezavisno obrađuju. Preklapanje sa dodavanjem se sastoji od tri koraka: podela polaznog signala na manje delove odnosno blokove, primena filtera na svakog od blokova pojedinačno i sastavljanje obrađenih blokova kako bi se dobio izlazni signal. Postupak preklapanja sa dodavanjem je prikazan na slici 2.

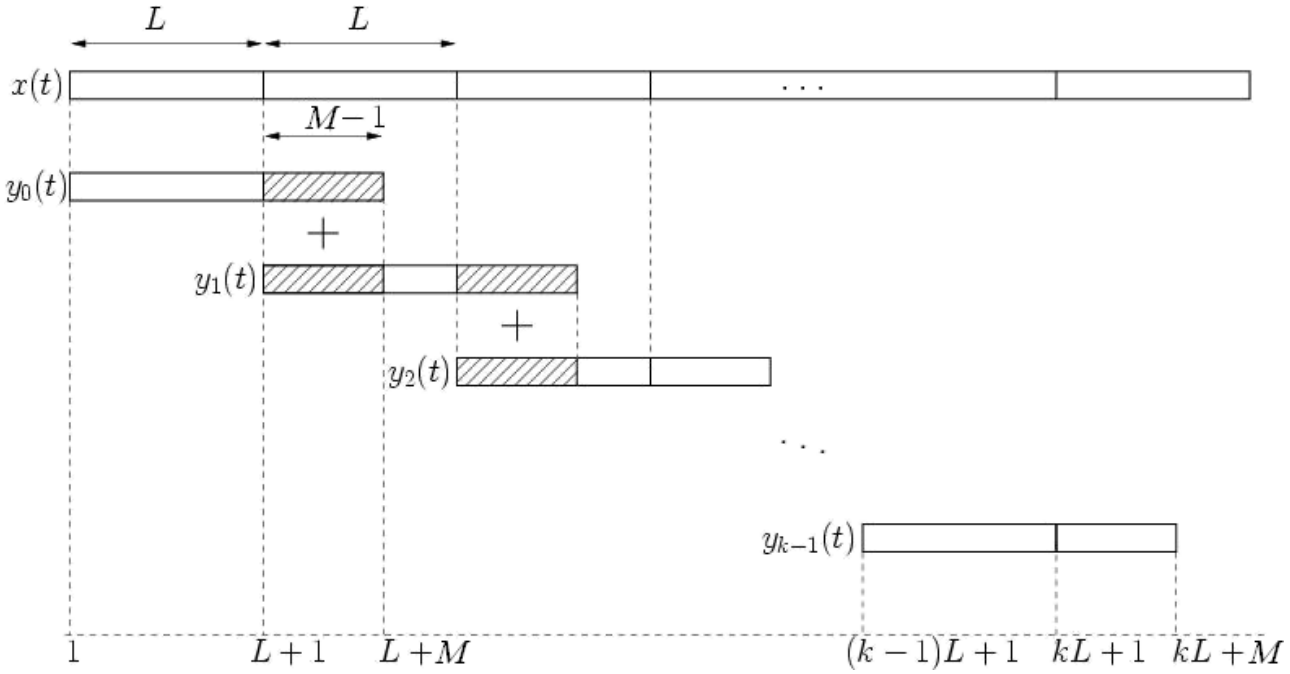

Figure 2: Preklapanje sa dodavanjem

Izdvojeni blokovi se mogu filtrirati proizvoljnom metodom, direktnom konvolucijom ili u frekvencijskom domenu pomocu FFT, pri čemu je u oba slucaja neophodno računati linearnu konvoluciju. Dakle, ako je blok veličine L a filter veličine M tada će izlazni signal biti veličine $L + M - 1$. Ako se filtriranje vrši u frekvencijskom domenu onda je potrebno proširiti ulazni signal i filter nulama tako da je njihova veličina jednaka najmanjem stepenu dvojke koji je veći ili jednak $N = L + M - 1$. Nakon računanja linearne konvolucije ostaje $M - 1$ uzoraka razlike izmedju ulaznog i izlaznog signala. Tu razliku je potrebno čuvati i nakon obrade sledećeg

bloka dodati prvim $M-1$ uzoraka iz sledećeg bolka. U nastavku je dat pseudokod metode preklapanja sa dodavanjem u slučaju filtriranja u frekvencijskom domenu.

Algorithm 1. OA for linear convolution

```
Evaluate the best value of N and L (L>0, N = M+L-1 nearest to power of 2).
Nx = length(x);
H = FFT(h,N)       (zero-padded FFT)
i = 1
y = zeros(1, M+Nx-1)
while i <= Nx  (Nx: the last index of x[n])
    il = min(i+L-1,Nx)
    yt = IFFT( FFT(x(i:il),N) * H, N)
    k  = min(i+N-1,M+Nx-1)
    y(i:k) = y(i:k) + yt(1:k-i+1)    (add the overlapped output blocks)
    i = i+L
end
```

Složenost metode preklapanja sa dodavanjem je $O(N_x log_2 N)$, gde je N_x dužina celog signala a N je najmanji stepen dvojke koji je veći ili jednak $N+M-1$, što je bolje od $O(N_x log_2 N_x)$ koliko je potrebno za računanje konvolucije na celom signalu odjednom. Na slici 3 je prikazan odnos vremena izvršavanja standardnog filtriranja u frekvencijskom domenu i filtriranja pomocu metode preklapanja sa dodavanjem u zavisnosti od različitih veličina ulaznog signala i filtera. Podebljanom plavom linijom je predstavljena granica gde metoda preklapanja sa dodavanjem počinje da bude brža od standardne FFT metode.

2.4 Preklapanje sa čuvanjem

.....

3 Filteri

Svi filteri koji su implementirani su FIR filteri (eng. finite impulse response) jer su oni adekvatni za aplikacije koje vrse filtriranje u realnom vremenu, za razliku od IIR filtera (eng. infinite impulse response) koji su dosta nestabilni i proizvode distorzije.

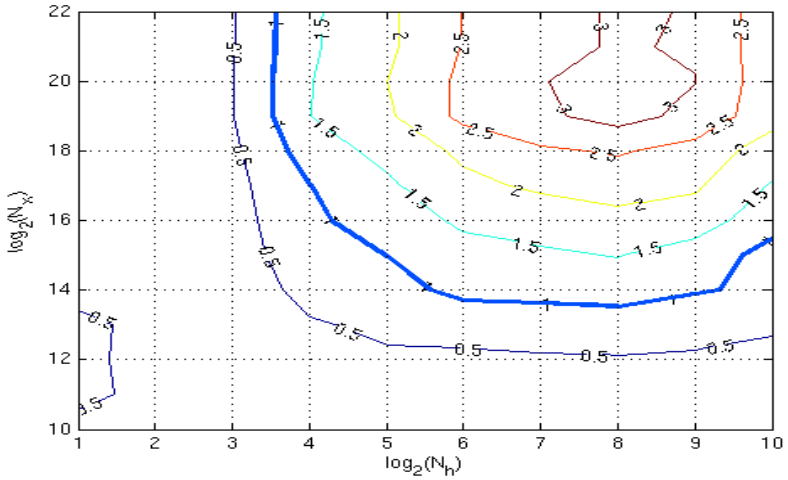

Figure 3: Preklapanje sa dodavanjem

3.1 Dizajn filtera

Za dizajniranje filtera korišćen je isključivo Batervortov niskopropusni filter različitog reda. Formula pomoću koje se računaju koeficijenti Batervortovog filtera je:

$$G^2(f) = \frac{G_0^2}{1 + (\frac{f}{f_c})^{2n}}$$

gde je G_0 gornja granica vrednosti koeficijenata koji se računaju, f je frekvencija koja odgovara koeficijentu koji se računa, a f_c frekvencija odsecanja. Pored niskopropusnog implementirani su i visokopropusni, opsegopropusni i opsegozaustavljajuci filteri. Koeficijenti visokopropusnog filtera se računaju tako što se koeficijenti niskopropusnog filtera reflektuju u odnosu na frekvenciju odsecanja na sledeći način:

$$G^2(f) = \frac{G_0^2}{1 + (\frac{f_c}{f})^{2n}}$$

Opsegopropusni i opsegozaustavljajuci filteri su malo složeniji, za njihovu implementaciju potrebni su i niskopropusni i visokopropusni filter. Opsegopropusni filter se racuna tako što se vši konvolucija niskopropusnog i visokopropusnog filtera s tim što je potrebno da frekvencija odsecanja niskopropusnog filtera bude veća od frekvencije odsecanja visokopropusnog filtera, dok je kod opsegozaustavljajućeg filtera situacija suprotna, frekvencija odsecanja niskopropusnog filtera mora biti manja od frekvencije odsecanja visokopropusnog, pri čemu se koeficijenti filtera sabiraju.

3.2 Implementacija filtera

Filteri su implementirani pomocu hijerarhije klasa. Na vrhu hijerarhije nalazi se apstraktna klasa Filter iz koje su izvedene sve ostale klase: Equalizer, LowPass, HighPass, BandPass i BandStop.

```
class Filter
{
public:
    enum Type {EQUALIZER, LOW_PASS, HIGH_PASS, BAND_PASS, BAND_STOP};
    enum Method {NONE, CONV, FFT, OA_CONV, OA_FFT};

    Filter(){}
    Filter(unsigned kernel_size, double sample_rate);
    virtual ~Filter(){std::cout << "DELETE F" << std::endl;}

    virtual Type type() const = 0;
    virtual void update_kernel() = 0;

    std::vector<double> kernel() const;
    double sample_rate() const;
    void set_kernel_size(unsigned kernel_size);
    void set_sample_rate(double sample_rate);
    void update_params(unsigned order, double cutoff, double cutoff_lp, double cutoff_hp, double gain);

    template<typename T>
    const std::vector<T> convolve(const std::vector<T> signal, Method method) const;

private:


protected:
    std::vector<double> _kernel;
    unsigned _kernel_size;
    double _sample_rate;
};
```

Figure 4: Implementacija apstraktne klase Filter

```
template<typename T>
const std::vector<T> Filter::convolve(const std::vector<T> signal, Method method) const
{
    static bool channel = false;
    std::vector<T> output = std::vector<T>(signal.size());
    static std::vector<T> overlap = std::vector<T>(_kernel_size-1);
    static std::vector<T> overlap1 = std::vector<T>(_kernel_size-1);
    static std::vector<T> overlap2 = std::vector<T>(_kernel_size-1);

    if(method == FFT)  {...}
    else if(method == CONV)  {...}
    else if(method == OA_FFT)  {...}
    else if(method == OA_CONV)  {...}
    else
    {
        std::complex<double> *isamples;
        std::complex<double> *fsamples;

        fsamples = fft_cpp(signal.data(), signal.size());


        isamples = ifft_cpp(fsamples, signal.size());
        for(unsigned i=0; i<signal.size(); i++)
            output[i] = isamples[i].real();

        delete[]  fsamples;
        delete[]  isamples;
    }

    channel = !channel;

    return output;
}
```

Figure 5: Implementacija metoda klase Filter koji računa konvoluciju

4 Korisnički interfejs

Korisnički interfejs se sastoji od dva glavna dela koja se prikazuju korisniku u zavisnosti od kontrola koje se nalaze na kontrolnoj tabli sa desne strane. Prvi deo se odnosi na ekvilajzer koji čine deset klizećih dugmadi. Svako dugme upravlja jačinom frekvencija koje se nalaze u odredjenom podintervalu celog frekvencijskog spektra. Pomeranjem dugmića korisnik može da podesi jačinu frekvencija u svakom od deset podintervala, koji zajedno pokrivaju spektar od 20 Hz do 22 KHz.

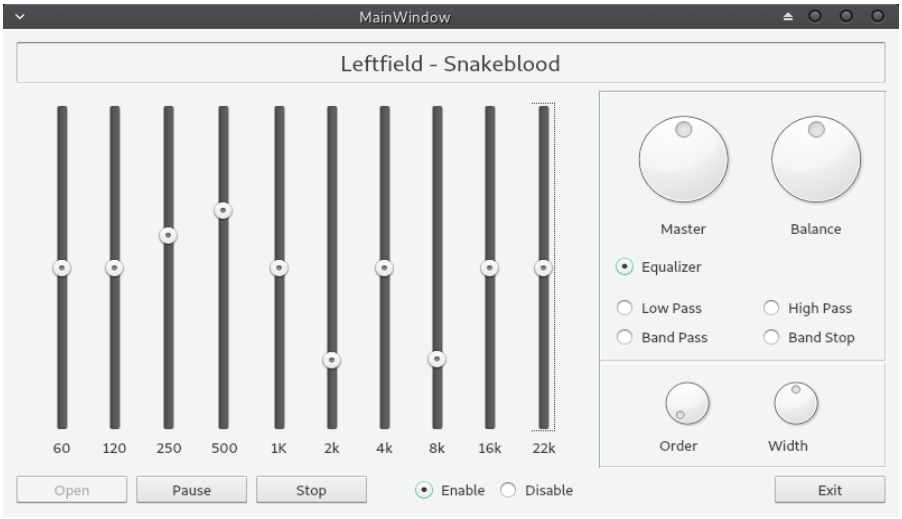

Figure 6: Ekvilajzer

Drugi deo glavne komponente interfejsa predstavlja platno po kojem se iscrtavaju filteri koje korisnik kreira. Platno takodje vrši iscrtavanje filtriranih ili nefiltriranih frekvencija koje su zastupljene u bloku ulaznog signala koji se u tom trenutku obradjuje, u zavisnosti od toga da li je filter aktivan ili ne. X-osa platna predstavlja spektar frekvencija od 0Hz do 20 KHz koje rastu eksponencijalno s leva nadesno, dok Y-osa predstavlja amplitude frekvencija od -20 dB do +3 dB koje rastu logaritamski odozdo nagore. Parametri filtera se inicijalizuju nakon što korisnik pritisne mesto na platnu gde želi da napravi filter, zatim se koordinate trenutnog položaja kursora na platnu transformišu u vrednost iz intervala frekvecija koja predstavlja frekvenciju odsecanja i u vrednost iz intervala amplituda koja predstavlja koeficijent koji pojaćava ili smanjuje odgovarajuće frekvencije.

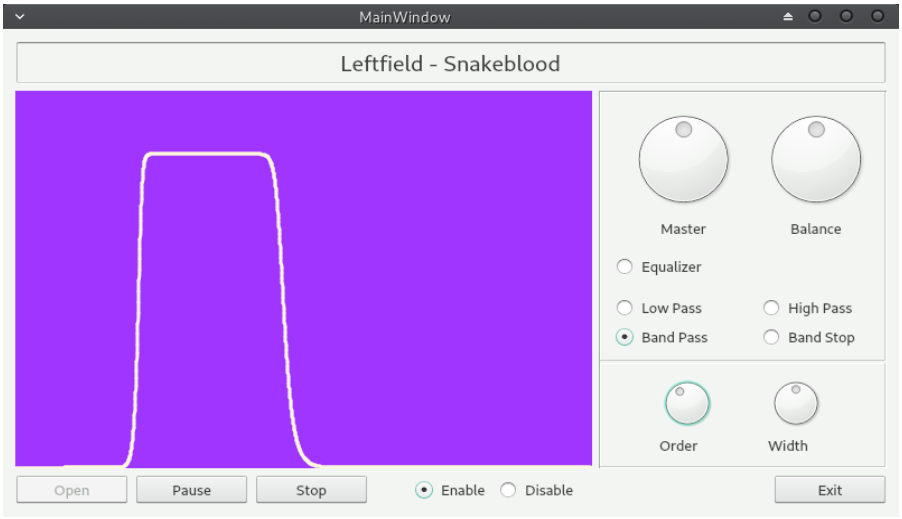

Figure 7: Opsegopropusni filter

5 Napomene, nedostaci i buduća unapređenja

- Filtriranje u realnom vremenu bez kašnjenja.
- Parametric equalizer 2.
- Vizuelna unapredjenja.

Reference

[1] Mladen Nikolić, Anđelka Zečević. *Naučno izračunavanje*. Matematički Fakultet, Univerzitet u Beogradu, 2018.

[2] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 2011.

[3] Charan Langton, Victor Levin. *The Intuitive Guide to Fourier Analysis & Spectral Estimation*. Mountcastle Company, 2016.

[4] William G. Gardner. *Efficient Convolution Without Latency*. Perceptual Computing Group MIT Media Lab, 1993.