Äänen taajuuden seuraus
Toteutusdokumentti

Roope Salmi
Tiralabra, 4. periodi 2021

Projektiin liittyen on toteutettu FFT, ristikorrelaatio, ja korrelaatiotäsmäysalgoritmi. Omana tietorakenteena käytetään rengaspuskuria. Demo-ohjelmassa esitetään, kuinka näitä voidaan käyttää oskilloskoopin kuvaajan vakauttamiseen.

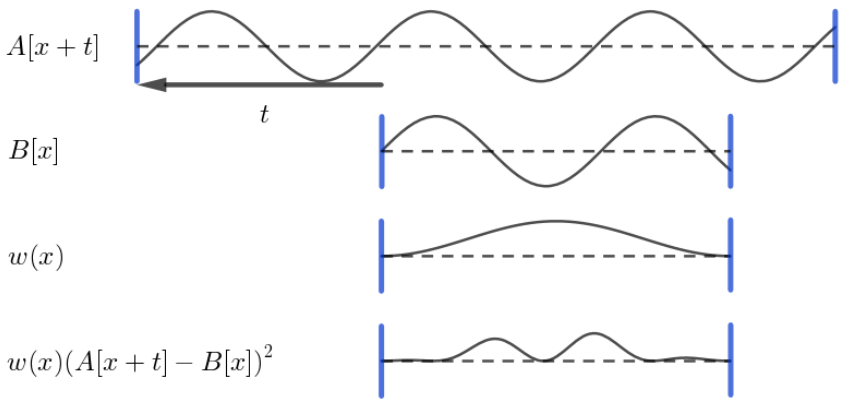
Korrelaatiotäsmäys

Algoritmi, joka etsii pidemmästä äänenpätkästä A sen kohdan, jossa lyhyempi äänenpätkä B esiintyy kaikista lähimpänä. Toteutus löytyy tiedostosta `src/correlation_match.rs`.

Olkoon signaalit $A[0..n]$ ja $B[0..m]$, $n \geq m$. Algoritmi etsii sellaisen aikasiirroksen t, jolla summa

$$d(t) = \sum_{x=0}^{m-1} w(x)(A[x+t] - B[x])^2$$

on minimaalinen. Etäisyys määräytyy siis erotuksien neliöiden kautta. Tässä w on painofunktio, jonka avulla voidaan painottaa enemmän esimerkiksi keskikohtia kuin reunoja.



Kuva 1: Korrelaatiotäsmäys

Jos tämä summa esitetään muodossa

$$\sum_{x=0}^{m-1} w(x)A[x+t]^2 - 2w(x)B[x]A[x+t] + w(x)B[x]^2,$$

nähdään, että se voidaan laskea tehokkaasti kahtena ristikorrelaationa ja yhtenä suorana tulona.

Ristikorrelaatiolla tarkoitetaan tässä siis operaatiota kahden funktion välillä, jonka tuloksena on funktio $f * g$:

$$(f * g)[t] = \sum_{x=0}^{N} f[x]g[x+t]$$

Ristikorrelaatio on toteuteuttu tiedostossa `src/cross_correlation.rs`.

Koska käsitellään diskreettejä näytteistettyjä signaaleja, ei funktion $d(t)$ minimikohtaa voida määrittää näytteenottoväliä tarkemmin. Sitä voidaan kuitenkin arvioida tarkemmin parabolisella interpolaatiolla, jossa kolmeen peräkkäiseen näytteeseen sovitetaan paraabeli, jonka minimikohta ratkaistaan analyyttisesti.

Funktiosta $d(t)$ arvioidaan lisäksi signaalin perustaajuutta. Jos sillä on monta lähes-minimikohtaa, voidaan näiden etäisyyksistä päätellä jaksonaika.

Metodi on pitkälti sama, kuin YIN-algoritmi [1], mutta absoluuttisen taajuuden tunnistamisen sijaan täsmätään erillisiä signaaleja keskenään.

Oskilloskoopin vakautus ja perustaajuuden arviointi

Korrelaatiotäsmäysalgoritmia hyödynnetään oskilloskoopin vakauttamiseen seuraavasti. Toteutus on tiedostossa `src/display.rs`, ja demon toteutus `examples/demo.rs`.

Signaaliksi A asetetaan uutta luettua signaalia. Signaaliksi B taas asetetaan vanhaa, aiemmin näytettyä signaalia. Ideana on, että algoritmi löytää signaalista A sopivan aikasiirroksen, jotta seuraavaksi näytettävä kuvaaja on mahdollisimman lähellä edellistä.

Jos oletetaan, että saapuva signaali on jaksollinen, niin taulukon A koko on valittava siten, että siihen mahtuu yksi kokonainen jakso, sekä B:n koon verran ylimääräistä signaalia. Tällöin uudesta signaalista löytyy aina sopiva kohta, joka vastaa aiemmin näytettyä signaalia täsmällisesti. Taulukon B koko on oltava korkeintaan puolet tästä.

Jotta perustaajuus voidaan arvioida, täytyy taulukkoa B vastaava kohta löytyä moneen kertaan taulukosta A. Tähän soveltuva perustaajuus on siis pienimmillään noin kaksinkertainen vakautukseen soveltuvaan nähden.

Painofunktiona w voidaan käyttää jotakin "kellokäyrän" tyyppistä funktiota, eli painotetaan näytön keskikohtia. Näin signaali pysyy keskitettynä, vaikka perustaajuus muuttuu. Toteutuksessa käytetään Hann-ikkunafunktiota [2].

Demo on suunniteltu toimimaan 44,1kHz näytteenottotaajuudella. Taulukon A koko on 1470 näytettä, ja taulukon B koko on 720 näytettä. Tällöin uusi kokonainen taulukko A luetaan reaaliajassa noin 30 kertaa sekunnissa. Matalin vakautuksen seuraama perustaajuus on siis 60Hz, ja matalin taajuusarvio 120Hz.

Demossa on lisäksi säädettävä vaimenemiskerroin ja erillinen muisti. Uutta ääntä ei verratakaan juuri näytettyyn aaltomuotoon, vaan erilliseen muistiin, jota päivitetään eksponentiaalisella vaimenemisella. Tämän tarkoituksena on saada näkymä pysymään vakaana,

vaikka ääneessä on väliaikaisia äkillisiä muutoksia.

$$muisti[x] := \alpha A[x + t] + (1 - \alpha) muisti[x]$$

FFT

FFT eli nopea Fourier-muunnos on toteutettu 2-kantaisena Cooley-Tukey -menetelmänä [3]. Sen aikavaativuus on $O(n \log n)$, mikä tekee korrelaatiotäsmäysalgoritmista yhtä tehokkaan. Toteutus on tiedostossa `src/fft.rs`.

Cooley-Tukey -menetelmä perustuu hajautua ja hallitse -tekniikkaan, mutta sen voi toteuttaa myös ilman rekursiota, kuten projektissa on tehty.

Optimointina `Fft`-olioon esilasketaan tietylle muunnoksen koolle N twiddle-kertoimet, eli kaikki kompleksiluvut muotoa $e^{2\pi ix/N}$, joita algoritmin suorituksen aikana käytetään. Esilaskeminen on projektin käyttötarkoituksessa hyödyllistä, koska on tarpeen tehdä jatkuvasti samankokoisia muunnoksia. Testausdokumentissa arvioidaan tämän optimoinnin hyötyä.

Rengaspuskuri

Rengaspuskuri, eli ring buffer, on tietorakenne, joka toimii first-in-first-out jonon tavoin. Lähettäjä ja vastaanottaja voivat olla eri säikeissä, ja atomisten kokonaislukumuuttujien avulla ei ole tarvetta lukkojen eikä järjestelmäkutsujen käytölle. Toteutus on tiedostossa `src/ring_buffer.rs`.

Jonolle varataan etukäteen vakiokokoinen taulukko. Kirjoitus- ja lukuindeksit määräävät kohdat, joista kukin operaatio tapahtuu seuraavaksi. Kun kirjoitusindeksi pääsee taulukon loppuun, aloitetaan alusta. Tästä juontuu tietorakenteen nimi — taulukko on ikään kuin rengas. On kuitenkin pidettävä huoli, ettei kirjoitusindeksi ylitä lukuindeksiä ja ylikirjoita dataa, jota ei ole vielä luettu. Jonoon mahtuu siis rajallinen määrä dataa kerrallaan.

Projektissa rengaspuskuria käytetään äänidatan siirtämiseen reaaliaikaiselta äänisäikeeltä pääsäikeelle käsittelyä varten. Yleisesti reaaliaikaisen äänen käsittelyn kanssa on tärkeää, ettei äänisäie joudu odottamaan vaikkapa lukon vapautumista.

Projektissa käytetään myös Rust-kielen standardikirjaston `Vec`-tietorakennetta, mikä on yleiskäyttöinen kasvamista tukeva taulukko. Sitä käytetään kuitenkin ainoastaan niin, että tietyn kokoinen taulukko varataan etukäteen, eikä sitä laajenneta myöhemmin. Tämän takia en katsonut tarpeelliseksi toteuttaa sitä itse.

Kirjallisuutta

[1] Alain de Cheveigné, Hideki Kawahara: "YIN, a fundamental frequency estimator for speech and music", The Journal of the Acoustical Society of America 111, 1917-1930 (2002) https://doi.org/10.1121/1.1458024

[2] “Hann function”, Wikipedia (2021), luettu 18.4. https://en.wikipedia.org/wiki/Hann_function

[3] “Cooley-Tukey FFT algorithm”, Wikipedia (2021), luettu 24.3. https://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm