UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII
ȘI TEHNOLOGIA INFORMAȚIEI
SPECIALIZAREA: TEHNOLOGII MULTIMEDIA

PROIECT DE CERCETARE

Detectare rapida si eficienta a pasului

Autocorelare Bitstream

Gal Oscar

Coordonator științific:
Prof. dr. ing. Eugen LUPU

CLUJ - NAPOCA

2021

CUPRINS

1. INTRODUCERE

In cadrul acestei lucrari va fi prezentat conceptul de autocorelatie si metoda de implementare in limbajul C++. Pentru a putea intelege acest concept, va fi nevoie de a prezenta cateva fundamente teoretice.

Corelația sau dependența este orice relație statistică, indiferent dacă este cauzală sau nu, între două variabile aleatorii sau date bivariate. În sensul cel mai larg, corelația este orice asociere statistică, deși se referă de obicei la gradul în care o pereche de variabile sunt legate liniar. Exemple familiare de fenomene dependente includ corelația dintre înălțimea părinților și descendenții lor și corelația dintre prețul unui bun și cantitatea pe care consumatorii sunt dispuși să o cumpere, așa cum este descrisă în așa-numita curbă a cererii.

Corelațiile sunt utile deoarece pot indica o relație predictivă care poate fi exploatată în practică. De exemplu, o companie electrică poate produce mai puțină energie într-o zi ușoară pe baza corelației dintre cererea de energie electrică și vreme. În acest exemplu, există o relație de cauzalitate, deoarece vremea extremă determină oamenii să folosească mai multă energie electrică pentru încălzire sau răcire. Cu toate acestea, în general, prezența unei corelații nu este suficientă pentru a deduce prezența unei relații cauzale (adică, corelația nu implică cauzalitatea).

În procesarea semnalului, un semnal este o funcție care transmite informații despre un fenomen. În electronică și telecomunicații, se referă la orice tensiune, curent sau undă electromagnetică care poartă informații. Un semnal poate fi, de asemenea, definit ca o schimbare observabilă a unei calități, cum ar fi cantitatea.

Orice calitate, cum ar fi cantitatea fizică care prezintă variații în spațiu sau timp, poate fi utilizată ca semnal pentru a partaja mesaje între observatori. Conform tranzacțiilor IEEE privind procesarea semnalului, un semnal poate fi audio, video, vorbire, imagine, sonar și radar și așa mai departe. Într-un alt efort de definire a semnalului, orice este doar o funcție a spațiului, cum ar fi o imagine, este exclus din categoria semnalelor. De asemenea, se afirmă că un semnal poate conține sau nu informații.

O funcție periodică este o funcție care își repetă valorile la intervale regulate, de exemplu, funcțiile trigonometrice, care se repetă la intervale de 2π radiani. Funcțiile periodice sunt utilizate în întreaga știință pentru a descrie oscilații, unde și alte fenomene care prezintă periodicitate. Orice funcție care nu este periodică se numește aperiodică.

Autocorelația, cunoscută și sub denumirea de corelație serială, este corelarea unui semnal cu o copie întârziată a acestuia ca funcție de întârziere. În mod informal, este asemănarea dintre observații în funcție de decalajul de timp dintre ele. Analiza autocorelației este un instrument matematic pentru a găsi modele repetate, cum ar fi prezența unui semnal periodic ascuns de zgomot sau identificarea frecvenței fundamentale lipsă într-un semnal implicat de frecvențele sale armonice. Este adesea utilizat în procesarea semnalului pentru analiza funcțiilor sau a seriilor de valori, cum ar fi semnalele domeniului timp.

Diferite domenii de studiu definesc autocorelația în mod diferit și nu toate aceste definiții sunt echivalente. În unele domenii, termenul este folosit în mod interschimbabil cu autocovarianța.

Procesele radiculare unitare, procesele staționare de tendință, procesele autoregresive și procesele medii mobile sunt forme specifice de procese cu autocorelare.

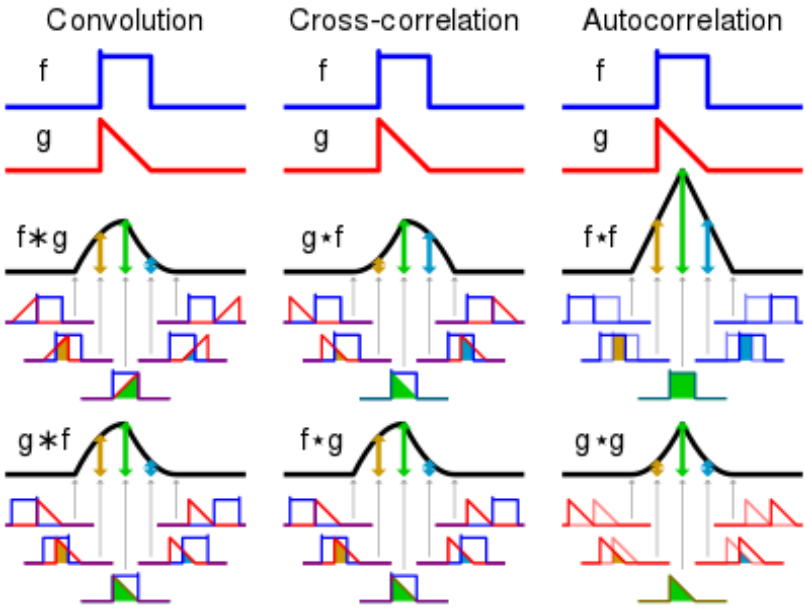

Figura 1. Comparatie intre convolutie, corelatie incrucisata si autocorelatie

In figura precedenta se poate observa o comparatie vizuala a convolutiei, corelatiei incrucisate si autocorelatiei. Pentru operațiile care implică funcția f și presupunând că înălțimea lui f este 1, valoarea rezultatului în 5 puncte diferite este indicată de zona umbrită sub fiecare punct. De asemenea, simetria lui f este motivul pentru care g * f si f ⋆ g sunt identice in acest exemplu.

1.1 Zero Crossing

Având în vedere o undă sinusoidală pură, puteți detecta perioada acesteia (și frecvența reciprocă) prin cronometrarea traversărilor zero. Oricât de naivă este, este cea mai simplă formă de detectare a înălțimii. Funcționează numai dacă aveți unde sinusoidale pure sau forme de undă destul de curate, cum ar fi triunghiuri, dinte de ferăstrău, PWM sau unde pătrate. Astfel de detectori eșuează atunci când aveți mai multe tranziții pe ciclu ca în exemplul de mai jos:

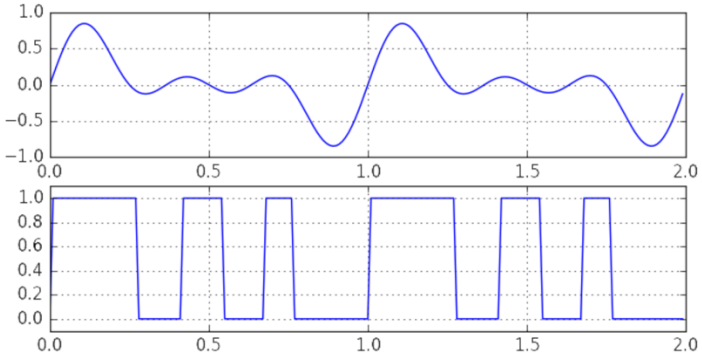

Figura 2. Exemplu de forme de unda

Acea formă de undă de mai sus conține a doua și a treia armonică puternică, chiar mai puternică decât cea fundamentală. Astfel de forme de undă sunt destul de frecvente! Fiecare ciclu conține mai multe tranziții de trecere zero. O modalitate de a „îmblânzi" forma de undă este prin filtrarea puternică a semnalului cu trecere joasă. Brut, dar probabil utilizabil, mai ales dacă știți gama de frecvențe cu care intenționați să lucrați. Din păcate pentru mine, filtrarea intensă a trecerii joase nu este o opțiune, deoarece aceasta va introduce o întârziere de fază inevitabilă.

1.2 Bitstream autocorrelation

O metoda specificata de Joel De Guzman a fost ca in loc sa aplice functia de autocorelare peste datele de intrare, sa aplice aceasta functie peste fluxul binary de treceri zero.

Pentru metoda specificata mai sus ar fi destul de greu de realizat ce este specificat, atunci daca trebuie sa se lucreze doar cu date binare, o solutie mai optima ar fi ca aceste calcule sa fie facute folosind doar operatii binare. In functia de autocorelare standard se multiplica semnalul in sine, in mod repetat (aceasta operatie de f * f). Aceasta operatie este una din cele mai costisitoare din algoritm. Tragem semnalul peste el insusi, inmulting semnalul cu o versiune intarziata a sa. Intr-o insctructiune repetitive stransa, se poate realiza acest lucru in mod repetat pentru fiecare proba din datele de intrare N /2 * N / 2 ori, unde N este numarul de probe. Este posibil sa folosim FFT pentru a calcula autocorelatia, dar acest lucru este insa destul de costisitor din punctul de vedere al timpului de executie si al memoriei, mai ales daca trebuie sa procesam mai multe canale simultan, pe un microcontroller. Chiar daca programul s-ar executa pe un calculator mai complex sim ai rapid, tot mai efficient ar fi sa folosim un algoritm mai rapid in acest scop. Operatiile DSP se folosesc destul de masiv de nucleele CPU-ului.

Și astfel, cu numai unu și zerouri cu care trebuie să se ocupe, înmulțirea poate fi simplificată:

0 * 0 = 0
0 * 1 = 0
1 * 0 = 0
1 * 1 = 1

Aceasta este în esență o operațiune ȘI (&&) ! Puteți efectua 32 de astfel de operații într-o singură fotografie cu un număr întreg de 32 de biți. 64 cu 64 de biți. Apoi, numărați toți biții care sunt setați după fiecare iterație. Cel mai înalt punct (care obține cel mai mare număr) va fi punctul în care există o corelație bună, pe măsură ce glisați semnalul peste sine. Acesta este cel mai înalt vârf (negru) din ilustrația 2 in partea cu (g*g).

Se poate observa ca daca in locul operatiei AND am folosi operatia XOR ar putea da rezultate mult mai bune. Asa ca mai jos se poate vedea in cazul XOR operatiile:

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

Spre deosebire de AND, cu ajutorul operatiei XOR primiți unul atunci când există o nepotrivire. Cu XOR, biții cu cel mai mic număr vor fi punctul în care există o corelație bună, din nou în timp ce glisați semnalul peste sine.

In urma rularii algoritmului discutat mai sus, rezultatele arata destul de promitatoare.

1.3 Codul Sursa

Codul sursa in Python:

```
from matplotlib.pyplot import figure, show
from numpy import arange, sin, pi

t = arange(0.0, 2.0, 0.01)
input = 0.3 * sin(2 * pi * t) + 0.4 * sin(4 * pi * t) + 0.3 * sin(6 * pi *
t)

fig = figure(1)

ax1 = fig.add_subplot(311)
ax1.plot(t, input)
ax1.grid(True)
ax1.set_ylim((-1, 1))


class zero_cross:
    def __init__(self):
        self.y = 0

    def __call__(self, s):
        if s < -0.1:
            self.y = 0
        elif s > 0.1:
            self.y = 1
        return self.y


zc = zero_cross()
trig = [zc(s) for s in input]

ax2 = fig.add_subplot(312)
ax2.plot(t, trig)
ax2.grid(True)
ax2.set_ylim((-0.1, 1.1))


def count_ones(l):
    r = 0
    for e in l:
        if e:
            r += 1
    return r


cross = trig
results = []
```

```
length = round(len(trig) / 2)

for i in range(length):
    x = [a ^ b for a, b in zip(trig[0:length], cross[i:i + length])]
    results.append(count_ones(x))
results.extend(results)

ax3 = fig.add_subplot(313)
ax3.plot(results)
ax3.grid(True)
ax3.set_ylim((-5, 80))

show()
```

In urma rularii am obtinut urmatorul grafic:

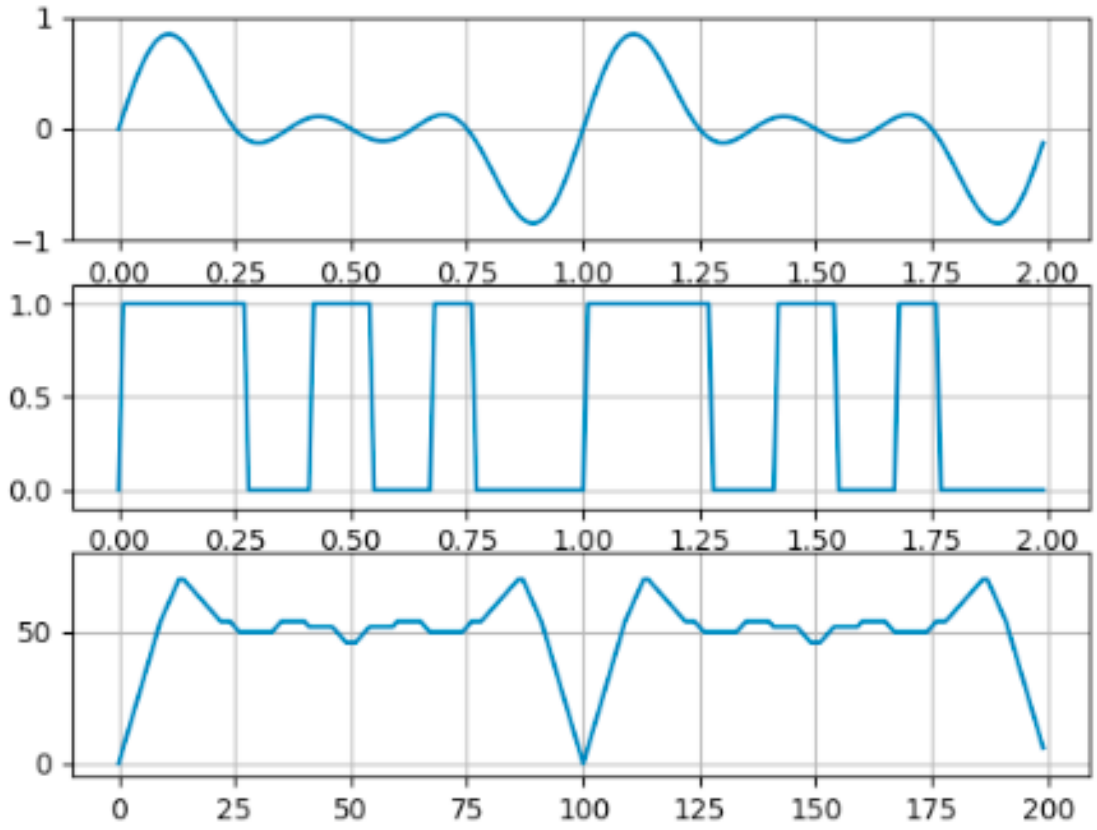

Figura 3. Rezltate dupa prima rulare

Rezultatele arata destul de promitator. Graficul din partea de jos este cel al Bitstream Autocorrelation. Punctul in care graficul atinge cel mai jos punct (la 100) este momentul in care ciclul incepe sa se repete. Distanta dintre start este perioada formei de unda.

In schimb daca schimbam datele de intrare cu:

```
input = 0.6 *sin(2 * pi *t) + 0.3 * sin(4 * pi * t) + 0.1 * sin(6 * pi * t)
```

Observam ca graficul a functiei de autocorelare nu mai este la fel de bun ca si in primul caz de executie.

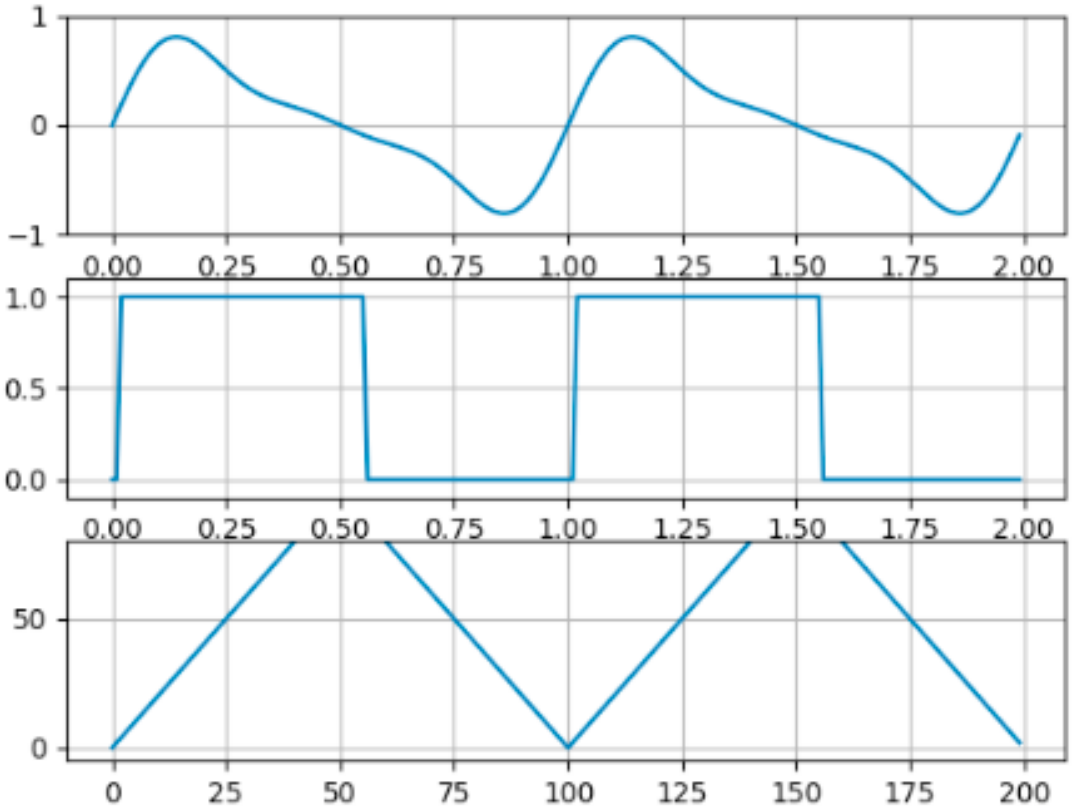

Figura 4. Rezltate dupa a doua rulare

1.4 Concluzii

Totusi, dupa toata munca depusa am ramas profund impresionat ca in anumite cazuri functioneaza atata de bine si la o fractiune din costul unei implementari complete (standard) a functiei de autocorelare. Nu exista multiplicari de functii, ci doar operatii elementare XOR ieftine, o operație inerent paralelă, procesând 32 (sau 64) biți pe pas în care fiecare bit corespunde unui eșantion

Exista si parti negative la aceasta implementare, adica trebuie sa începeți cu o margine pozitivă reală (unde forma de undă trece de punctul zero). Nu puteți începe în mijlocul unei forme de undă. Cu ACF standard, puteți începe oriunde. Dar acesta este un compromis rezonabil. Înainte de alte investigații și teste, nu se poate spune exact daca ar fi si alte probleme la acest algoritm. Aceasta este, deocamdată, doar o dovadă a conceptului.