

Nopeuden määrittäminen äänen perusteella

Fysiikan vikсутutkielma 2005

Juho Roponen

Valkeakosken aikuislukio/

Päivölän kansanopisto

Tiivistelmä

Tutkimuksen ensisijaisena tavoitteena oli kehittää passiivinen sensori, joka pystyy pelkästään äänen perusteella selvittämään liikkuvan äänilähteen nopeuden, esimerkkinä auto.

Autojen äänet koostuvat aina useista eri taajuuksista. Tutkimuksen työläin työvaihe oli järkevä taajuusjakumaa tutkivan algoritmin kehittäminen. Ohjelmoin Java-kielisen ääntä analysoivan ohjelman, joka saa syötteenä ainoastaan wav-muotoisen äänitiedoston, jossa on tutkittava ääninäyte. Äänen taajuudet saadaan esille FFT-algoritmillä, joka on tavallinen äänen digitaalisen signaalinkäsittelyn työkalu. Tutkimuksen suurin haaste oli kuitenkin FFT:llä lasketun taajuusjakauman analysointi.

Ohjelmaa testattiin kolmella eri tavalla. Ensin testattiin generoidulla siniaallolla, seuraavaksi tutkittiin läheisellä maantiellä liikkuvien autojen nopeuksia ja lopuksi tutkittiin, miten oikein ohjelma antoi tunnetulla nopeudella liikkuvan auton nopeuden.

Testauksen tulokset olivat toivotunlaisia. Ohjelma pystyy laskemaan äänilähteen nopeuden oikein kaikissa yksinkertaisissa tapauksissa ja suurimmassa osassa vaikeampia tapauksia. Muutamissa tilanteissa ohjelma antoi melko suuren virheen, mutta tutkimuksen voidaan kuitenkin sanoa saavuttaneen tavoitteensa.

Sisällysluettelo

1 Johdanto.....	1
2 Aineisto ja menetelmät.....	1
2.1 Fysiikka.....	1
2.2 Saadun äänidatan tutkiminen.....	2
2.3 Taajuusjakauman analysointi.....	3
2.4 Ohjelman toiminta.....	4
3 Testaus.....	5
1. Koe: Siniaalto.....	6
2. Koe: Läheinen maantie.....	6
3. Koe: Tunnetulla nopeudella liikuva auto.....	8
4 Tulosten tarkastelu.....	9
5 Pohdinta.....	12
6 Lähdeluettelo.....	13
7 Liitteet.....	14
7.1 Lähdekoodi.....	14

1 Johdanto

Tutkimuksen ensisijaisena tavoitteena oli kehittää passiivinen sensori, joka pystyy pelkästään äänen perusteella selvittämään liikkuvan äänilähteen nopeuden, esimerkkinä auto.

Tutkielma on hyvin ajankohtainen, koska tällä hetkellä on käynnissä useita tutkimuksia taajuusjakauman muodon analysointiin liittyen. Esimerkiksi Nokialla on taajuusjakumien muotoihin liittyviä tutkimuksia (http://www.nokia.com/library/files/docs/A_Spatial_Audio_User_Interface_for_Generating_Music_Playlists.pdf). Myös nopeuden selvittämiseksi äänen perusteella on olemassa käyttötarkoituksia. Esimerkiksi hyvin pienten kappaleiden nopeuksia on lähestulkoon mahdotonta selvittää perinteisillä dopplertutkilla.

Myös se, että kyseessä on täysin passiivinen sensori, lisää tehdyn ohjelman käyttöarvoa, koska aina ei haluta tutkittavan kohteen tietävän olevansa tutkimuksen kohteena.

Ohjelma toteutettiin Java-kielellä ja se saa syötteenä ainoastaan wav-muotoisen äänitiedoston, jossa on tutkittava ääninäyte. Äänen taajuusjakauma oli tarkoitus selvittää käyttäen FFT-algoritmia, joka on tavallinen äänen digitaalisen signaalinkäsittelyn työkalu. Tutkimuksen suurin haaste oli kuitenkin FFT:llä lasketun taajuusjakauman analysointi. Autojen äänet koostuvat aina useista eri taajuuksista. Tutkimuksen työläin työvaihe oli järkevän taajuusjakumaa tutkivan algoritmin kehittäminen.

2 Aineisto ja menetelmät

2.1 Fysiikka

Dopplerin ilmiöön liittyvällä yleisesti tunnetulla kaavalla (kaava 1) (Peltonen, Perkkiö, Vierinen 2004, s.112), jossa f on kuultava taajuus, f_0 todellinen taajuus ja c äänen nopeus, voidaan laskea auton nopeus, v .

$$(1) \quad f = f_0 \cdot \frac{c}{c \pm v} ,$$

Sijoittamalla kaavaan taajuudet f_1 , joka taajuus, kun auto tulee kohti ja f_2 , joka saadaan kun auto menee pois päin saadaan kaavat 2a ja 2b.

$$(2a) \quad f_1 = f_0 \cdot \frac{c}{c - v} \quad \text{ja} \quad (2b) \quad f_2 = f_0 \cdot \frac{c}{c + v}$$

Ratkaistaan kaavojen 2a ja 2b muodostamasta yhtälöparista nopeus v .

$$(c-v) \cdot f_1 = (c+v) \cdot f_2$$

$$(f_1 + f_2) \cdot v = (f_1 - f_2) \cdot c$$

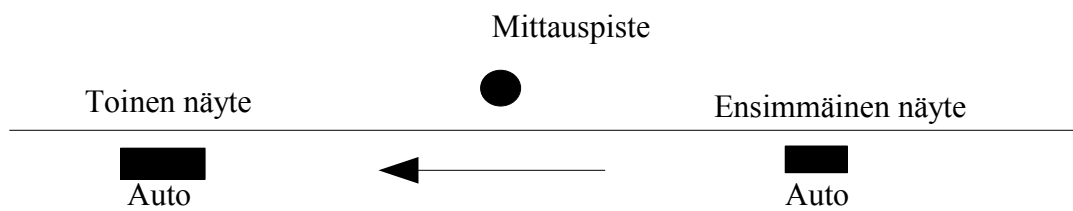
$$(3) \quad v = c \cdot \frac{f_1 - f_2}{f_1 + f_2}$$

Kaavassa 3 oletetaan, että auto tulee suoraan kohti ja menee suoraan pois päin, mutta kun auto kulkee mittauspisteen ohi, todellinen nopeus saadaan kaavasta

$$(4) \quad v = \cos(\alpha) \cdot v ,$$

jossa α on auton kulkusuunnan ja mittauspisteen välinen kulma autosta katsottuna. Käytännössä tämä ero on merkityksettömän pieni, jos mittauspiste on lähellä auton kulkulinjaa ja äänen analysoinnissa tutkitaan vain tilannetta, jossa auto hieman kauempana mittauspisteestä. Tällöin $\cos\alpha$ on likimain yksi, jolloin äänestä laskettu nopeus vastaa melko pitkälti todellista nopeutta.

Auton äänen tutkiminen auton ollessa aivan kohdalla ei toimi hyvin, koska äänen voimakkuus kasvaa todella kovaksi ja autosta kuuluva ääni on mahdollisesti sekavampaa, kuin hieman kauempaa kuultuna.



Piirros 1: Kuva mittaustilanteesta

2.2 Saadun äänidatan tutkiminen

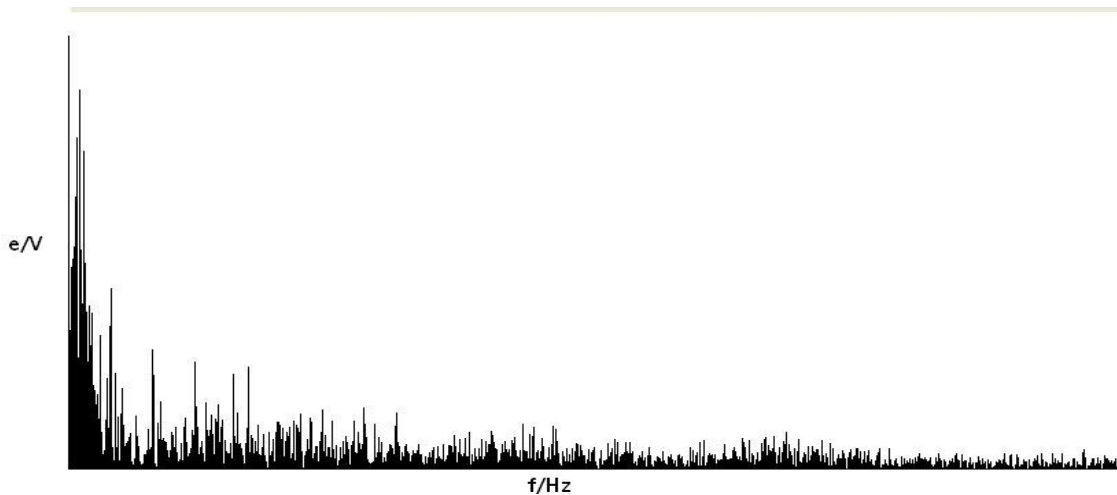
Äänen taajuusjakauman selvittämiseen ohjelma käyttää Fourier'n muunnosta, joka on tavallisimmin käytettyjä algoritmeja äänen digitaalisessa signaalinkäsittelyssä. Lähestyvän ja loittonevan auton taajuusjakaumista voidaan laskea taajuuden muutos. Fourier'n muunnoksen toiminta perustuu siihen, että mikä tahansa jaksollinen funktio voidaan esittää sinifunktioiden summana. Fourier'n muunnoksella saadaan selville näiden eri komponenttien suuruudet. Fourier'n muunnoksen melko tarkkaan numeeriseen laskemiseen käytetään yleensä diskreettiä Fourier'n muunnosta (DFT). (<http://mathworld.wolfram.com/DiscreteFourierTransform.html>) DFT ei suoraan sovi kuitenkaan kovin suurien aineistojen käsittelyyn, koska algoritmin nopeus on $O(n^2)$ riippuva. Tästä syystä suurien aineistojen käsittelyyn käytetään nopeampia algoritmeja.

Koska aineisto voi ääntä analysoidessa kasvaa huomattavan suureksi käytän Fourier'n muunnoksen

laskemiseen niin sanottua nopeaa Fourier'n muunnosta (FFT eli Fast Fourier Transform). Sen suoritusaika on vain $O(n \log_2 n)$ riippuva, joten se on suuria aineistoja käsitellessä huomattavasti nopeampi kuin suora DFT. FFT:n suurempi nopeus johtuu siitä, että se käyttää funktion jaksollisuutta hyväkseen DFT:n laskemisessa. Tavallisimmin käytetty ja minun käyttämäni FFT on niin sanottu Cooley/Tukey algoritmi, joka hajottaa Fourier'n muunnoksen laskemisen useaksi pienemmäksi DFT:ksi. (<http://mathworld.wolfram.com/FastFourierTransform.html>) Tämä algoritmi voidaan toteuttaa useammalla tavalla, joista käyttämäni on melko tavallinen FFT esimerkki.

2.3 Taajuusjakauman analysointi

Tutkimuksen suurimpia haasteita on taajuuden muutoksen havaitseminen FFT:llä saadusta taajuusjakaumasta. Tämä johtuu siitä, että auton ääni koostuu hyvin monista eri taajuuksista. Tällöin Fourier'n muunnoksella laskettavan taajuusjakauman kuvaajasta tulee usein melko epäselvä kuten kuvaajasta 1 näkyy.

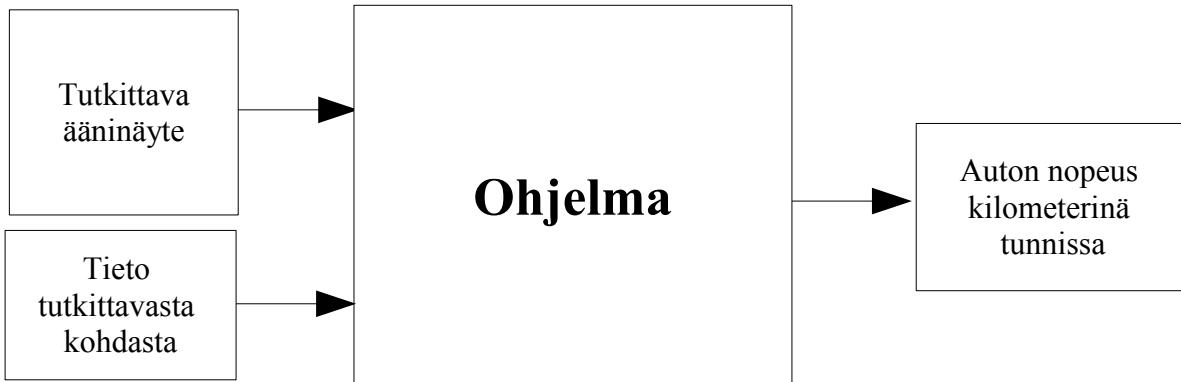


Kuvaaja 1: Erään auton äänen taajuusjakauma

Kuvaajan 1 kaltaisissa tapauksissa taajuuden keskiarvon muutoksen laskemisesta ei ole mitään hyötyä. Tästä syystä päätin analysoida taajuuden muutosta tutkimalla voimakkaimmin kuuluvan taajuuden muutosta.

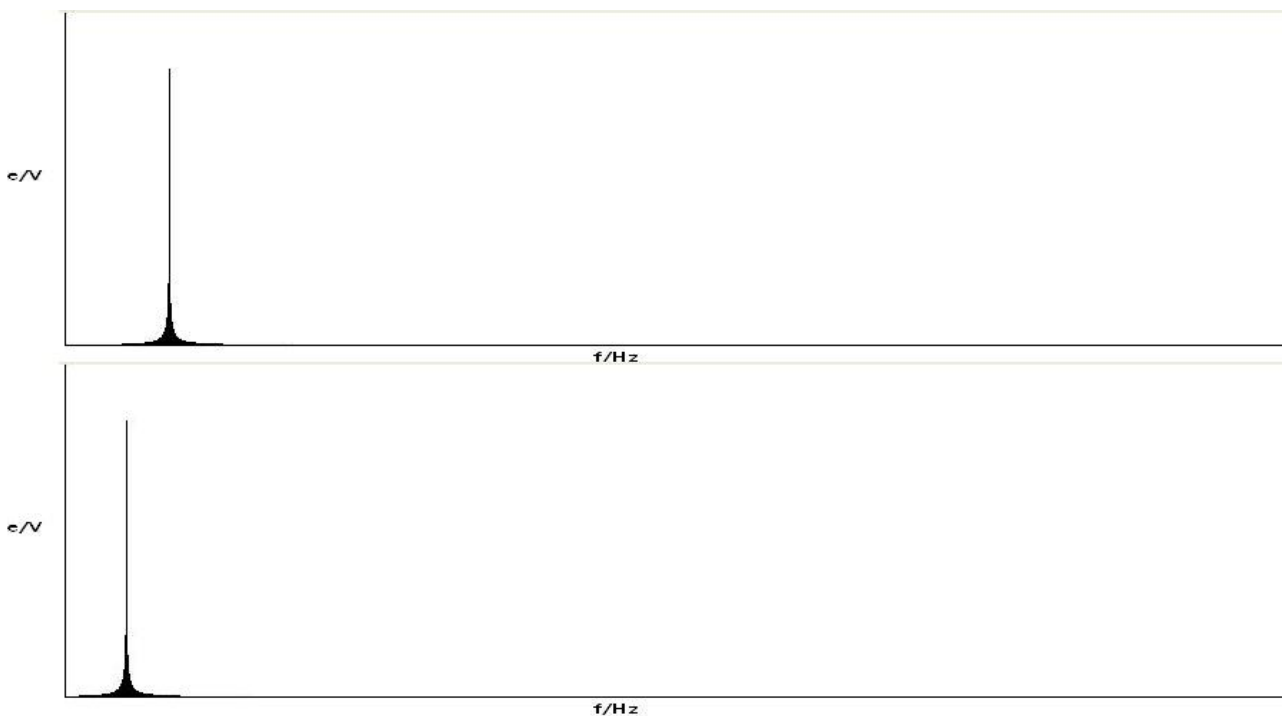
Käytännössä tämä toimii siten, että ohjelma hakee sekä lähestyvän että loittonevan auton äänen taajuusjakaumista suurimman arvon ja vertaa näitä toisiinsa. Aivan pienimmät taajuudet jätetään huomiotta, koska niillä laskettaessa virheet ovat suurempia.

2.4 Ohjelman toiminta



Piirros 2: Yksinkertainen kaavakuva ohjelmasta

Piirros 2 esittää yksinkertaistetusti, mitä ohjelma tekee. Ensimmäiseksi ohjelma hakee annettun äänitiedoston kahdesta valitusta kohdasta liukulukumuodossa olevan äänisignaalin sisältävät taulukot. Annettujen kohtien tulisi olla hieman ennen kuin auto on mittauspisteen kohdalla ja hieman mittauspisteen jälkeen. Tämän jälkeen ohjelma tekee FFT muunnoksen molemmille saaduille signaaleille, jolloin saadaan laskettua niiden taajuusjakaumat. Seuraavaksi ohjelma etsii taajuusjakaumista huippuja, joiden siirtymisestä signaalinäytteiden välillä ohjelma laskee auton nopeuden kaavan 3 perusteella käyttäen löydettyjä huipputaajuuksia.

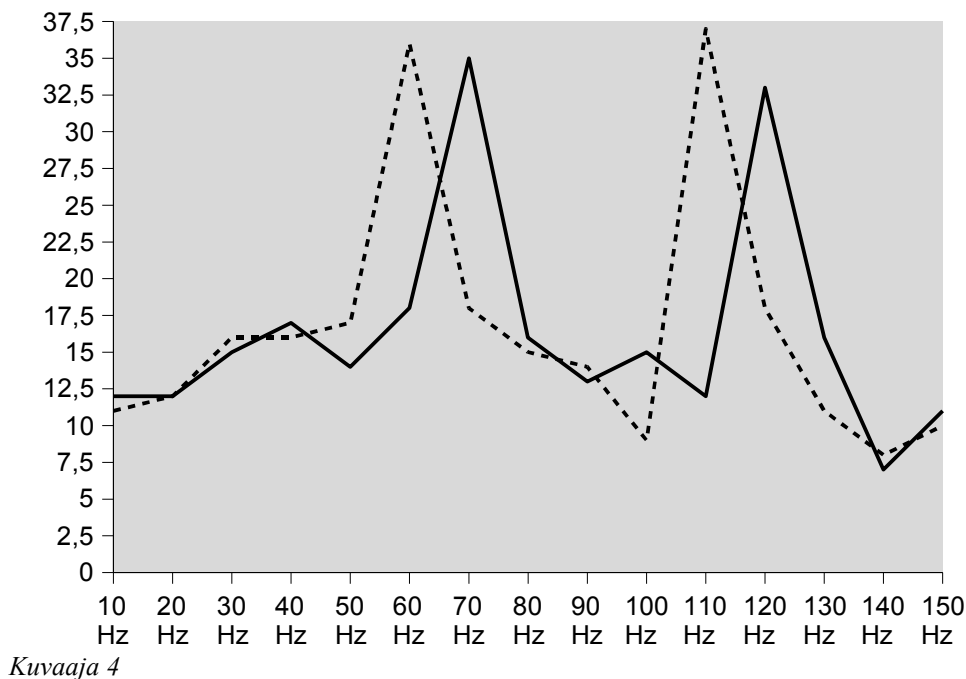


Kuvaajat 2 ja 3: Generoidun siniaallon taajuusjakauma.

Kuvaajissa 2 ja 3 on esimerkki tapauksesta, jossa voimakkaimmin kuuluvat taajuudet on todella helppo selvittää. Tällaisissa tapauksissa ohjelman on helppoa laskea auton nopeus.

Aina huippujen löytäminen ei kuitenkaan ole helppoa. Esimerkiksi kuvaajan 4 kaltaisesta kaaviosta ihminen huomaa selvästi, että katkoviivalla merkityn kuvaajan huiput ovat hiukan enemmän vasemmalla kuin jatkuvalla viivalla merkityn, mutta tämä tietokoneelle ei ole läheskään yhtä yksinkertaista. Esimerkkikuvaajan tapauksessakin on kaksi huippua, jotka ovat lähestulkoon yhtä suuria. Tämä voi aiheuttaa ongelmia tietokoneelle varsinkin, jos huippujen keskinäinen korkeusjärjestys vaihtelee kuvaajien välillä. Käytännössä kuvaajat eivät myöskään koskaan ole yhtä yksinkertaisia kuin esimerkkikuvaajassa. Jotkut taajuusjakaumat ovat liian monimutkaisia yksinkertaisella algoritmilla tutkittaviksi.

Esimerkki



3 Testaus

Ohjelmaa testattiin kolmella eri tavalla. Ensin testattiin generoidulla siniaallolla. Seuraavaksi tutkittiin läheisellä maantiellä liikkuvien autojen nopeuksia. Viimeiseksi tutkittiin, miten oikein ohjelma antoi tunnetulla nopeudella liikkuvan auton nopeuden.

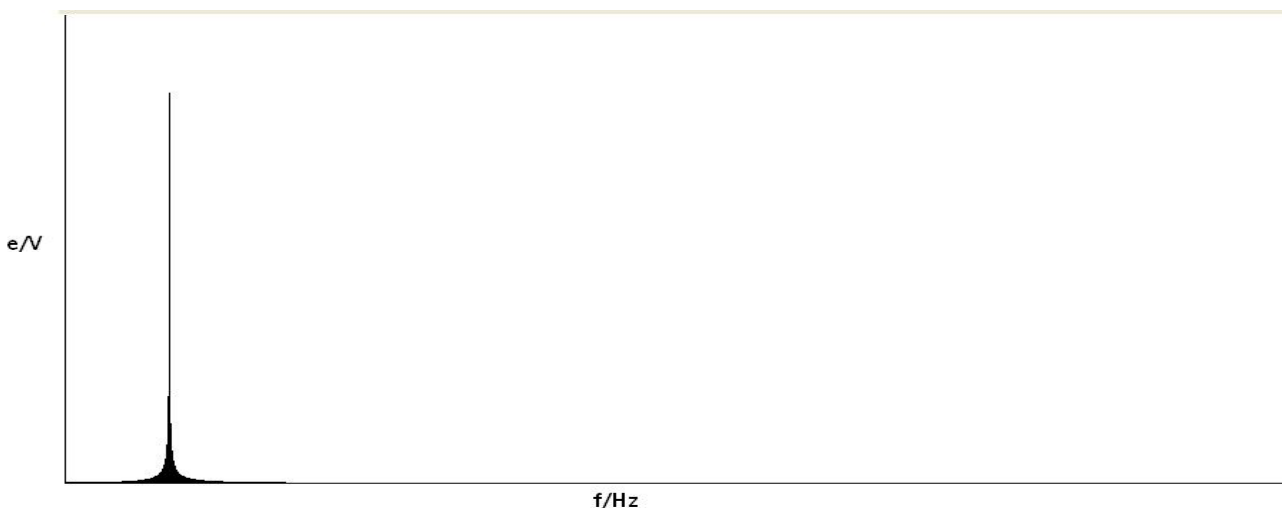
1. Koe: Siniaalto

Aivan ensimmäiseksi käytin generoitua siniaaltoa. Tulokset ovat taulukossa 1.

Siniaallon 1 taajuus/Hz:	200	200	200	200	150	150
Siniaallon 2 taajuus/Hz:	200	150	120	100	120	100
Ohjelman antama nopeus/km/h:	0,000000	154,285721	267,692308	360,000011	120,000001	216,000003
Laskennallinen nopeus/km/h:	0,000000	154,285714	270,000000	360,000000	120,000000	216,000000

Taulukko 1: Ohjelman siniaalloista laskemat nopeudet

Tuloksista nähdään, että ohjelma pystyy laskemaan nopeuden oikein ainakin siniaalloista. Ensimmäisen kokeen tulokset osoittavat, että ohjelma pystyy ratkaisemaan nopeuden ainakin yksinkertaisimmassa mahdollisessa tapauksessa.



Kuvaaja 5: Siniaallon taajuusjakauma FFT:llä laskettuna

Kuvaajassa 5 näkyy, että FFT löytää jopa siniaallosta useita taajuuksia. Tämä on eräs FFT:n ongelmia. (<http://www.broadcastpapers.com/tvtran/TeracomMobileDVB-T04.htm>)

2. Koe: Läheinen maantie

Seuraavassa vaiheessa nauhoitin läheisellä maantiellä liikkuvien autojen ääniä. Tulokset ovat näkyvissä taulukossa 2.

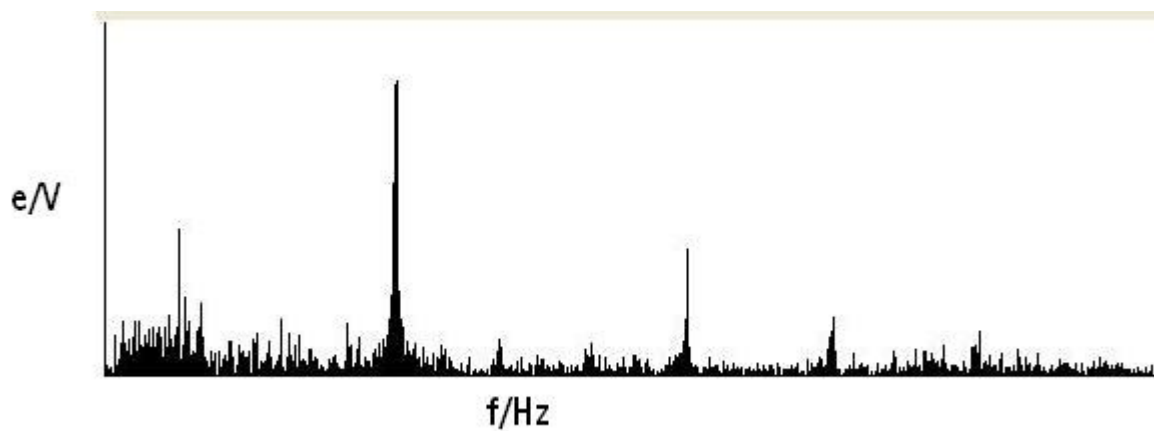
	Auto 1	Auto 2	Auto 3	Linja-auto
Ohjelman antama nopeus/km/h:	91,91	110,69	93,7	70,95

Taulukko 2: Läheisellä maantiellä liikkuvien autojen nopeuksia.

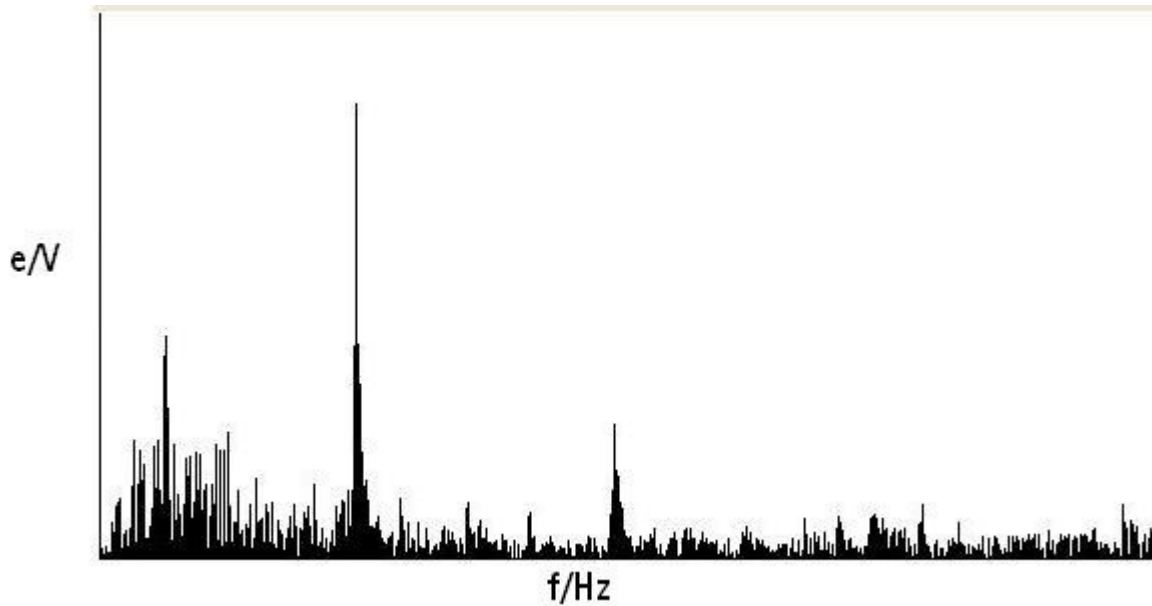
Paikallinen nopeusrajoitus on 80km/h. Autojen nopeuksiksi tuli jonkin verran nopeusrajoituksen ylittäviä arvoja, mutta tämä oli valitettavasti odotettavissa. Jopa 110km/h nopeus on kyseisellä tieosuudella realistinen. Linja-auton hieman nopeusrajoitusta alempi nopeus johtuu luultavasti siitä, että linja-autot ajavat melko usein kyseisellä tienpätkällä hieman alle nopeusrajoitusten, koska lähistöllä on myös useita linja-autopysäkkejä.

Kuvaajissa 6 ja 7 on esitetty Fourier'n muunnoksella saatujen taajuusjakaumien kuvaajat, kun linja-auto on tulossa kohti ja menossa poispäin. Kuvissa näkyvät taajuusjakaumat ovat esimerkkejä ohjelmalle helpposta tapauksesta, joissa on yksi huomattavasti muita suurempi selkeä huippu.

Auton 2 suuri nopeus voi myös johtua laskennan epäonnistumisesta, koska ääninäyte oli hiukan liian lyhyt, jolloin en päässyt valitsemaan näytekohtaa aivan riittävän kaukaa hetkestä, jolloin auto oli mittauspisteen kohdalla. Kun ääninäyte on liian läheltä mittauspistettä, kuuluu auton moottorin ääni erittäin voimakkaasti. Tämä haittaa laskentaa, koska moottoriäänet voivat vaihdella paljonkin riippuen siitä, mistä suunnasta niitä kuunnellaan. Esimerkiksi pakoputki voi pitää häiritsevää ääntä auton ajaessa poispäin.



Kuvaaja 6: Linja-auto tulossa kohti



Kuvaaja 7: Linja-auto menossa pois päin

3. Koe: Tunnetulla nopeudella liikkuva auto

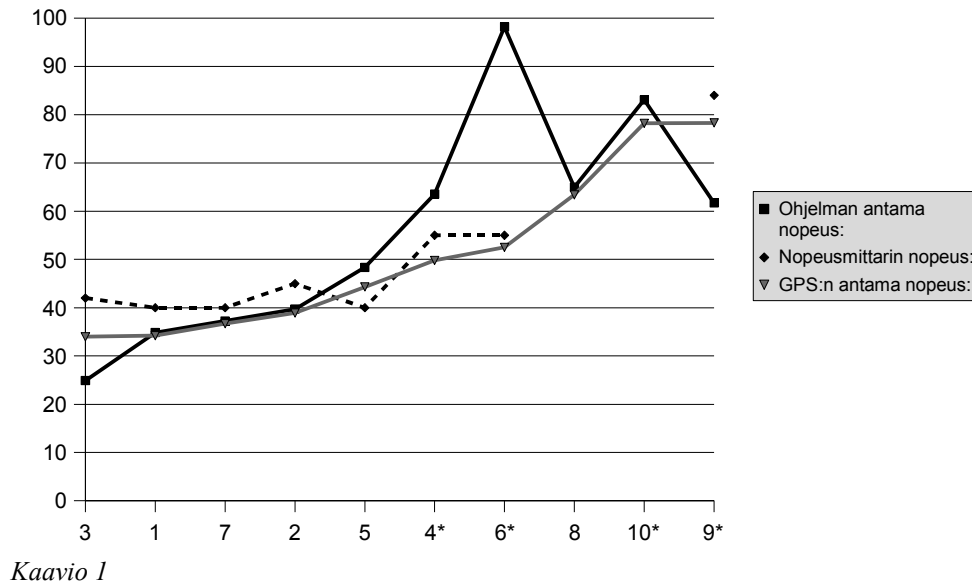
Viimeisessä testausvaiheessa nauhoitin tunnetulla nopeudella liikkuvan auton (Volkswagen Caravell Syncro, vm 1988) ääntä. Auto ajoi ohi kymmenen kertaa eri nopeuksilla. Auton oikea nopeus katsottiin GPS:stä (Benefon ESC) ja nopeusmittarista, joka antaa aiemmin tarkistetun noin 10% virheen. Nopeusmittarin lukemat voivat myös olla epätarkkoja, koska niitä lukeneen henkilön piti keskittyä myös auton ajamiseen. GPS:n kanssa tätä ongelmaa ei ole. Mittaushetkellä tuuli hieman. Taulukossa 3 ja kaaviossa 1 näkyvät auton nopeudet eri ohiajokerroilla.

Mittaus:	1	2	3	4*	5
Ohjelman antama nopeus:	34,84	39,71	24,92	63,53	48,36
Nopeusmittarin nopeus:	40	45	42	55	40
GPS:n antama nopeus:	34,2	38,9	34	49,8	44,3

Mittaus:	6*	7	8	9*	10*
Ohjelman antama nopeus:	98,18	37,24	64,96	61,71	83,08
Nopeusmittarin nopeus:	55	40		84	
GPS:n antama nopeus:	52,5	36,7	63,4	78,3	78,2

Taulukko 3

Mittaustulokset kaaviossa



Tähdellä on merkitty mittaukset, joiden taajuusjakauma oli niin epäsäännöllinen, että nopeudet jouduttiin laskemaan hyvin pienistä taajuuksista. Tämä vaatii pienen muutoksen koodiin, koska normaalisti ohjelma jättää pienimmät taajuudet huomiotta. Pienimpien taajuuksien mukaan ottaminen automaattisesti voidaan helposti toteuttaa siten, että ne huomioidaan, kun nopeudeksi saadaan jotain ilmeisen järjetöntä. Todellisessa tilanteessa on kuitenkin yleensä olemassa arvio siitä, mitä tulokseksi tulee. Tällöin järjettömät tulokset voidaan helposti karsia.

Testeissä havaittiin, että lähes kaikissa autojen äänissä on ainakin yksi hyvin voimakas matala taajuus. Matalien taajuuksien analysointi kuitenkin lisää virhettä saatuun nopeuteen merkittävästi, koska saatujen taajuuksien prosentuaalinen virhe kasvaa tällöin suuremmaksi. Tästä syystä pienet taajuudet jätetään normaalisti tarkastelun ulkopuolelle.

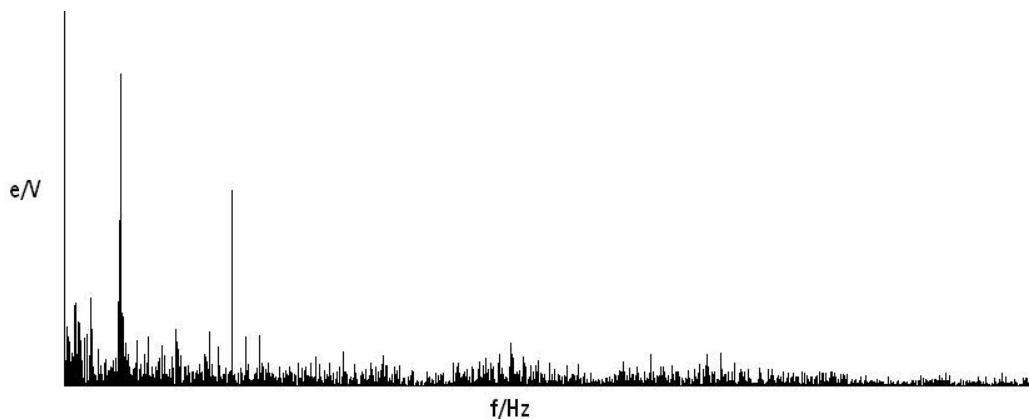
4 Tulosten tarkastelu

Testien perusteella voidaan sanoa, että ohjelma toimii hyvin, kun laskentaan voidaan käyttää suuria taajuuksia. Pienet virheet suurilla taajuuksilla lasketuissa nopeuksissa voivat johtua virheestä joko nauhoitetussa äänitiedostossa tai äänen analysoinnissa. Pienillä taajuuksilla lasketuissa nopeuksissa pienikin virhe voimakkaan taajuuden selvittämisessä kasvaa moninkertaiseksi nopeutta laskettaessa. Yksi mahdollinen virhelähde äänitiedostossa muualta kuin autosta tulevat äänet kuten esimerkiksi tuuli. Toinen mahdollinen virhelähde on yksinkertaisesti mikrofoniin huono asento tai heikko toiminta. Tätä virhelähdettä pystyy kontrolloimaan aika hyvin. Itse analyysissa virhe tulee todennäköisimmin taajuusjakauman monimutkaisesta rakenteesta. Ohjelma ei selviä tapauksesta,

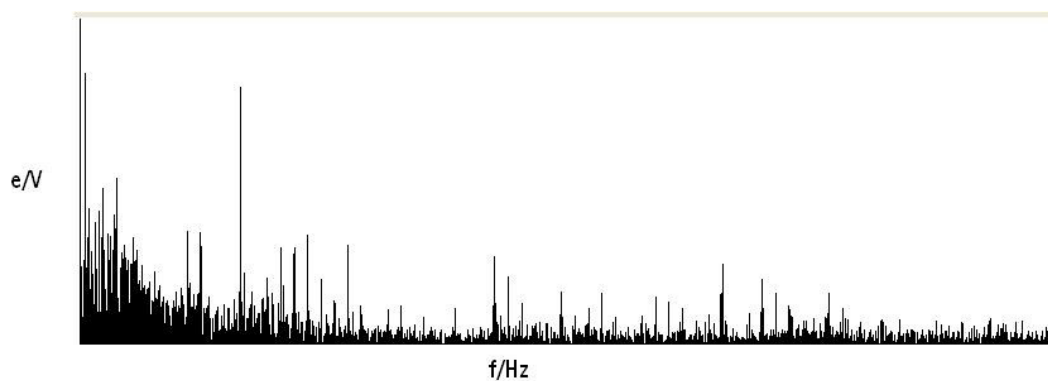
jossa taajuusjakaumassa on useita huippuja, joiden suuruusjärjestys syystä tai toisesta muuttuu auton mentyä ohi.

Erittäin monimutkaiseen taajuusjakaumaan auttaa usein jonkin verran taajuusjakauman analysointi uudestaan aivan pienistä taajuuksista, mutta tällöin virhe voi kasvaa niin suureksi, että nopeudeksi voi tulla ihan mitä vain. Näin tapahtuu, koska taajuuksien ollessa pieniä myös niiden väliset erot ovat pieniä. Tällöin ohjelma joutuu vähentämään kaksi lähes yhtäsuurta lukua toisistaan, mikä heikentää merkittävästi tuloksen tarkkuutta. Pieniä taajuuksia tutkittaessa myös käytetty FFT on epätarkempi. (http://www.lids-group.com/docs/site_documents/AN006.pdf)

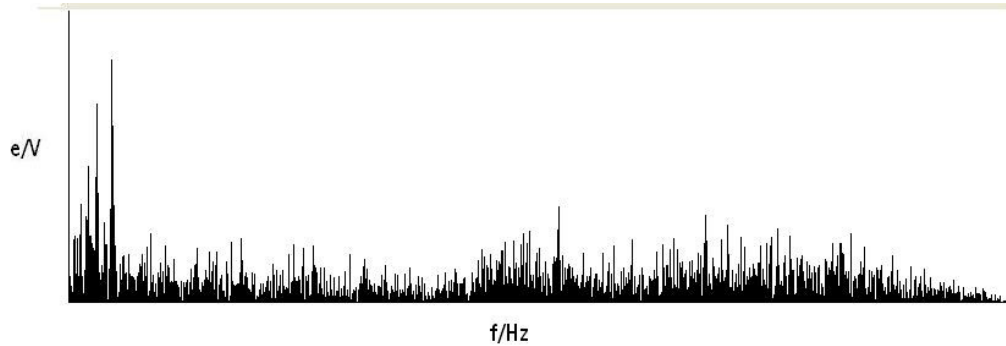
Kuvaajat 8 ja 9 esittävät helppoa tapausta, jossa on selkeä huippu melko suurella taajuudella. Kuvaajissa 10 ja 11 taas on tapaus, jossa ainoat selkeät huiput ovat matalilla taajuuksilla.



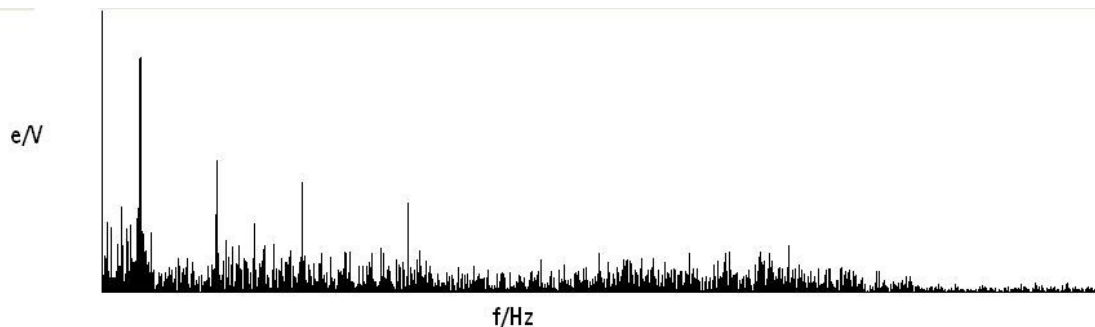
Kuvaaja 8: Auto tulossa kohti



Kuvaaja 9: Auto menossa poispäin



Kuvaaja 10: Auto tulossa kohti



Kuvaaja 11: Auto menossa pois päin

Ohjelmaa voisi vielä jatkokehittää entistäkin paremmaksi. Eräs tapa analysoida monimutkaisia taajuusjakaumia on etsiä useita huippuja joiden muutoksia vertailemalla saadaan tarkempia tuloksia, mutta tämä vaatisi huomattavasti kehittyneemmän algoritmin, jonka toteutus veisi paljon aikaa. Kehittyneempi algoritmi myös hidastaisi ohjelman suoritusta. Täydellinen taajuusjakaumien tutkiminen onkin äärimmäisen haastavaa ja hidasta.

Toinen tapa parantaa ohjelmaa olisi käytettävän FFT algoritmin parantaminen. Suoraan taulukoituna taajuusjakaumasta voi kadota osa huipuista. Tämän ongelman saa ratkaistua muuttamalla taajuusjakauman taulukointitapaa.

(<http://www.teemach.com/FFTProp/FFTProperties/FFTProperties.htm>)

Pieni puute ohjelmassa on myös se, että itsenäisesti oikeiden tutkittavien kohtien löytäminen äänitiedostosta on ohjelmalle käytännössä mahdotonta. Tämä ei kuitenkaan ole ongelma, koska samaa koodia voidaan yhtä hyvin soveltaa reaaliajassa saatavaan äänisignaaliin, jolloin ohjelma tallentaa muistiin äänisignaalit vain, kun auto on oikeassa kohdassa. Tällöin ohjelman ei tarvitse etsiä oikeaa kohtaa. Itse käytin nauhoitettua ääntä puhtaasti testausteknisistä syistä. Ohjelmaa on miellyttävämpää testata, kun mittaukset voidaan tehdä ulkona ja varsinainen analysointi voidaan suorittaa sisätiloissa.

5 Pohdinta

Taajuusjakaumien tunnistuksen ei suinkaan tarvitse rajoittua autojen nopeuden laskemiseen. Koska jokaisen auton taajuusjakauma on melko yksilöllinen, voitaisiin ohjelmaa jatkokehittää tunnistamaan auton nopeuden lisäksi se, mikä auto on kyseessä. Ohjelmaa voitaisiin myös soveltaa saman auton äänen muutoksen tutkimiseen esimerkiksi konevian tapauksessa.

Suoraan ohjelmaa voitaisiin soveltaa myös muiden äänilähteiden kuin autojen nopeuksien määrittämiseen. Esimerkiksi hyönteisten nopeuksia lienee käytännössä vaikeaa selvittää tavallisella tutkalla. Äänen perusteella hyönteisen nopeuden selvittäminen ei kuitenkaan ole haastavampaa kuin auton. Monien muidenkin pienten tutkimuskohteiden nopeuksien selvittämisessä ääni voi olla monia muita tapoja parempi.

6 Lähdeluettelo

”Discrete Fourier Transform” sivustolla mathworld.wolfram.com 29.11.2005.

”Fast Fourier Transform” sivustolla mathworld.wolfram.com 29.11.2005.

”FFT Properties 3.5 The basic tutorial on frequency analysis”. Steema Softwaren sivusto (<http://www.teemach.com/FFTProp/FFTProperties/FFTProperties.htm>) 29.11.2005

Hiipakka, Jarmo ja Lorho, Gaëtan ([[jarmo.hiipakka](mailto:jarmo.hiipakka@nokia.com), [gaetan.lorho](mailto:gaetan.lorho@nokia.com)]@nokia.com ”A Spatial Audio User Interface For Generating Music Playlists”. Nokian sivuilla (www.nokia.com) 29.11.2005.

Peltonen, Hannu, Perkkiö, Juha ja Vierinen, Kari 2004. Insinöörin (AMK) Fysiikka osa II. Saarijärvi: Saarijärven OFFSET OY.

Stare, Erik. ”MOBILE RECEPTION OF 2K AND 8K DVB-T SIGNALS ”. (<http://www.broadcastpapers.com/tvtran/TeracomMobileDVB-T04.htm>) 29.11.2005.

”Variable Resolution Random Control”. LDS-groupin sivusto. (http://www.lds-group.com/docs/site_documents/AN006.pdf) 29.11.2005.

7 Liitteet

7.1 Lähdekoodi

```

public final class FastFourierTransform {
//luokka, joka toteuttaa nimensä mukaisesti FFT:n

    private int n, nu;

    private int bitrev(int j) {

        int j2;
        int j1 = j;
        int k = 0;
        for (int i = 1; i <= nu; i++) {
            j2 = j1/2;
            k = 2*k + j1 - 2*j2;
            j1 = j2;
        }
        return k;
    }

    public final float[] fftMag(float[] x)
    {
        // oletetaan, että n on kahden potenssi
        n = x.length;
        nu = (int) (Math.log(n)/Math.log(2));
        int n2 = n/2;
        int nul = nu - 1;
        float[] xre = new float[n];
        float[] xim = new float[n];
        float[] mag = new float[n2];
        float tr, ti, p, arg, c, s;
        for (int i = 0; i < n; i++)
        {
            xre[i] = x[i];
            xim[i] = 0.0f;
        }
        int k = 0;

        for (int l = 1; l <= nu; l++)
        {
            while (k < n)
            {
                for (int i = 1; i <= n2; i++)
                {
                    p = bitrev (k >> nul);
                    arg = 2 * (float) Math.PI * p / n;
                    c = (float) Math.cos (arg);
                    s = (float) Math.sin (arg);
                    tr = xre[k+n2]*c + xim[k+n2]*s;
                    ti = xim[k+n2]*c - xre[k+n2]*s;
                }
            }
        }
    }
}

```

```

        xre[k+n2] = xre[k] - tr;
        xim[k+n2] = xim[k] - ti;
        xre[k] += tr;
        xim[k] += ti;
        k++;
    }
    k += n2;
}
k = 0;
n1--;
n2 = n2/2;
}
k = 0;
int r;
while (k < n)
{
    r = bitrev (k);
    if (r > k)
    {
        tr = xre[k];
        ti = xim[k];
        xre[k] = xre[r];
        xim[k] = xim[r];
        xre[r] = tr;
        xim[r] = ti;
    }
    k++;
}

mag[0] = (float) (Math.sqrt(xre[0]*xre[0] +
xim[0]*xim[0]))/n;
for (int i = 1; i < n/2; i++)
    mag[i]= 2 * (float) (Math.sqrt(xre[i]*xre[i] +
xim[i]*xim[i]))/n;
mag[0]=0;
return mag;
}
}

```

```

public class SpeedAnalyzer {
// Luokka, joka laskee äänilähteen nopeuden, kun sille annetaan
kaksi taajuusjakaumaa sisältävää
//taulukkoa

float[] array1, array2;
float f1, f2;
float fr1, fr2;
final double speedOfSound = 300;

public SpeedAnalyzer(float fool, float foo2){
    fr1=fool;
    fr2=foo2;
}
}

```

```

public double analyze(){
    double speed;
    f1=0.5f*getMaxValue(array1)*fr1;
    f2=0.5f*getMaxValue(array2)*fr2;

    speed=((f1-f2)/(f1+f2))*speedOfSound;
    return speed;
}

public double analyze(float[] array1,float[] array2){
    double speed;
    f1=(float)getMaxPoint(array1)*0.5f*fr1/array1.length;
    f2=(float)getMaxPoint(array2)*0.5f*fr2/array2.length;
    speed=((f1-f2)/(f1+f2))*speedOfSound;
    return speed*3.6;
}

public int getMaxPoint(float[] array){
    int max=0;
    float maxValue=0;
    for(int i=10;i<array.length;i++){
        if(array[i]>maxValue){
            max=i;
            maxValue=array[i];
        }
    }

    return max;
}
}

public class testailu {
//luokka, joka ainoastaan kutsuu muita luokkia
    static float[] fool, foo2;
    static float[] mag1, mag2;

    public static void main(String[] args){
        //Signalreader on luokka, joka hakee äänitiedostosta
        //äänien liukulukumoudossa
        SignalReader sr1=new SignalReader("V001.wav");
        SignalReader sr2=new SignalReader("V001.wav");
        SpeedAnalyzer sa=new SpeedAnalyzer(sr1.getSamplingRate(),
            sr2.getSamplingRate());
        FastFourierTransform fft=new FastFourierTransform();

        //getSignal-metodi hakee halutun mittaisen pätkän
        //äänisignaalia tiedoston halutusta kohdasta
        fool=sr1.getSignal(16384,150000);
        foo2=sr2.getSignal(16384,550000);

        //Tehdään FFT-muunnos molemmille äänisignaaleille
        mag1=fft.fftMag(fool);
        mag2=fft.fftMag(foo2);
    }
}

```

```
        //Tulostetaan konsoliin taajuusjakauman analysoinnin
        //tulos
        System.out.println("nopeus: "+sa.analyze(mag1,mag2));
    }
}
```