KTH Informations- och
kommunikationsteknik

Skakad men inte rörd

Accelerometer som rörelsedetektor till larm

IT1611 Mikroelektronikprojekt - KTH ICT, Kista

Fredrik Lundgren
flundg at kth.se

Joel Nilsson
joelni at kth.se

Martin Axelsson
maxels at kth.se

2008-05-14

Sammanfattning

Accelerometer har blivit billigare, mindre och har börjat dyka upp i allt fler produkter. En accelerometer är en sensor som mäter lägesförändringar. De billigaste och enklaste produceras med MEMS-teknik och mäter hur kapacitansen förändras i proportion mot accelerationen. Problemet som skulle lösas i det här projektet var att finna en ny tillämpning för en accelerometer. Tillämpningen som valdes var att göra ett larm. Målsättningen var att det skulle vara tillförlitligt och prisvärt. Först gjordes mätlaborationer för att få en uppfattning om accelerometerns funktion och mätområde. Första försöket gick ut på att använda en accelerometer för att mäta förflyttning. Olika typer av filter implementerades för att hantera bakgrundsbruset. Detta resulterade ej i någon produkt, ty datan var svårbehandlad på grund av bland annat bakgrundsbrus. Andra försöket gick ut på att använda en accelerometer som stötdetektor. Det resulterade i ett program som kan demonstrera hur en accelerometer kan användas som ett larm.

Innehåll

Figurer

Tabeller

1 Inledning

Den senaste tiden har accelerometrar blivit billigare och mindre, det har medfört att de har börjat dyka upp i allt fler produkter. En accelerometer är en sensor som känner av lägesförändringar. De kan idag finnas i allt från mobiltelefoner, spelkontroller, digitalkameror till krockkuddar. I enklare modeller mäter de i en dimension och de mer avancerade kan mäta i alla tre dimensioner. Fem parametrar är det maximala som kan fås - lutning, position, rörelse, vibration och stöt [1].

Problemet som skulle lösas i projektet var att ta fram en ny tillämpning för en accelerometer. Första steget i projektet var att ta reda på hur kraftiga accelerationer som gavs vid olika rörelser. Redan existerande tillämpningar analyserades för att ge inspiration. En tillämpningsidé specificerades med hjälp av inspirationen och ren brainstorming.

Den accelerometer som användes i projektet för att göra mätningar och tester var ett demokit från Freescale. Om tillämpningen är intressant så kommer eventuellt Freescale sedan att använda den för att demonstrera och marknadsföra accelerometrar.

Till demokitet följde det med en mjukvara där redan existerande tillämpningar kan testas samt extrahera mätdata.

Det är också meningen att projektgruppen efter projektet ska vara insatt i hur en accelerometer fungerar.

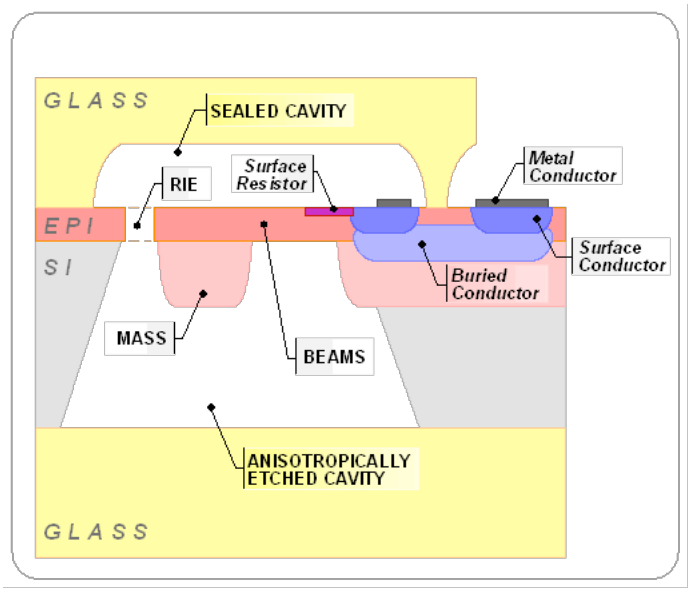
Tillämpningen som valdes var en rörelsedetektor till ett larm. Syftet var att ta fram ett litet, prisvärt och portabelt larm som skall kunna fästas på i princip vad som helst.

2 Accelerometer i teorin

Det finns några olika sätt att mäta lägesförändringarna med en accelerometer på. Den accelerometer som har använts i det här projektet mäter hur kapacitansen förändras i proportion mot accelerationen.

Accelerometrar utav den typen tillverkas med MEMS-teknik, se figur 2.1 för skiss. Hålrum med tungor etsas i kisel vilket bildar en variabel kondensator. Kapacitansen varierar med tungans läge [2]. Detta är en av de enklaste produkterna som kan tillverkas med MEMS vilket har bidragit till att priserna på accelerometrar har gått ner och tillgängligheten har ökat.

En A/D-omvandlare översätter kapacitansen till digital data som till exempel kan sändas trådlöst till en dator. Dessa värden översätts vanligen till g-kraft för att bli relevanta för en människa. Tyngdaccelerationen motsvaras till exempel av $1g$ medan en krockkuddssensor lyssnar efter accelerationer på över $50g$. Observera dock att g-kraften är relativ jorden [4].



Figur 2.1: Skiss på en MEMS-accelerometer

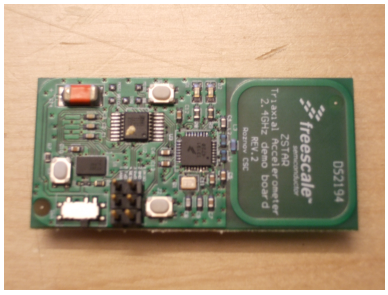
g-kraften ges av:

$$g = Value_{A/D} - \frac{Cal_{min}}{Cal_{max} - Cal_{min}} \tag{2.1}$$

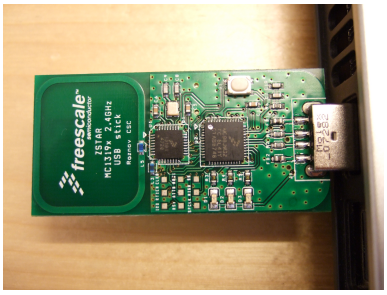
Cal_{min} och Cal_{max} är de värden accelerometern är kalibrerad till. Det är olika värden för varje axel. $Value_{A/D}$ är A/D-värdena från accelerometern. Kalibreringens syfte är att ställa in accelerometerns läge relativt jordens tyngdacceleration vid mätplatsen.

Accelerationen ges av:

$$a = g * 9.82\ m/s^2 \tag{2.2}$$



Figur 2.2: Sändarkortet i Freescales demokit.



Figur 2.3: Mottagarkortet i Freescales demokit.

2.1 Mätlaboration

Genom att göra mätningar på vissa ”vardagliga” situationer kan det skaffas en uppfattning av accelerationer och av mätområdets begränsning hos demokitets accelerometer MMA7360L. Data kan samlas in via Freescales demonstrationsprogram. Det sparar tid, A/D-värden och g-kraft för varje axel i ett Excel-dokument.

2.1.1 Hårdvarubeskrivning

Demokitet från Freescale består av två kort; sändarkortet (figur 2.2) består av accelerometern, en A/D-omvandlare och en ZigBee-sändare, mottagarkortet (figur 2.3) består av en ZigBee-mottagare och USB-interface [3]. Sändarkortet drivs av ett batteri, mottagarkortet drivs via USB. På sändarkortet finns det tre knappar, där den ena är en testknapp. Mottagarkortet har även det en testknapp. Accelerometerns mätområde kan varieras med en resistor, i demokitet är det däremot tyvärr fixerat till 1,5g.

Sensorn är en MMA7360L från Freescale och är en treaxlad accelerometer. Mikroprocessorn är en MC9S08QG8, även den från Freescale.

ZigBee är en standard för trådlös kommunikation och påminner om Bluetooth. ZigBee är framtaget för att vara energisnålt, enkelt att installera och säkert [5].

2.1.2 Utförande av mätning

Följande mätningar skall göras:

Tappa låda – Accelerometern skall släppas från olika höjder för att simulera en låda som tappas.

Sparka boll – Accelerometern skall placeras i en boll för att se vad en fotboll utsätts för.

Fäktas – Accelerometern skall fäktas runt med, förslagsvis fäst på en pinne.

Små rörelser – Små precisionsrörelser skall göras.

Promenad – En person skall promenera runt med accelerometern på sig.

Springa – En person skall jogga runt med accelerometern på sig.

Hoppa – En person skall hoppa med accelerometern på sig.

Åka kundvagn – Accelerometern skall få åka runt i en kundvagn.

Röra runt – Accelerometern skall utsättas för cirkulära rörelser.

Kast med liten boll – Accelerometern skall kastas fram och tillbaka mellan två personer.

Åka hiss i Forum – Accelerometern skall åka hiss i Forum, KTH Kista.

Studsande boll – Bollen från ”Sparka boll” skall studsas mot golvet i en båge.

Mobilvibrator – Accelerometern skall ligga på en mobiltelefon som vibrerar.

Inför varje mätning så måste accelerometern kalibreras för att ge korrekta värden. Mätningarna kräver minst två personer. En som hanterar accelerometern och en som sköter Freescales demonstrationsprogram.

2.1.3 Behandling av mätdata

Mätdatan sparas i Excel-dokument och i Excel är det sedan även möjligt att skapa grafer. För behandling av mätdata i MATLAB kan datan i Excel-dokument importeras direkt och sparas i vektorer. Importeringen till MATLAB kan göras med funktionen `xlsread`.

2.2 Mätning av förflyttning

För att bestämma hur långt accelerometern förflyttas integreras accelerationen två gånger. Första integrationen ger hastigheten. Görs detta för alla tre axlar så kan det bestämmas hur den har förflyttats. Dock måste det tas hänsyn till hur accelerometern vrids i rummet.

Integration av numeriska värden görs enkelt med verktyg som MATLAB. Problemet består således i att filtrera bort bakgrundsbrus och hantera de snabba förändringarna i systemet för att få tillförlitliga resultat.

Nedan följer några exempel på hur bakgrundsbrus kan filtreras bort.

2.2.1 Högpassfilter

Med antagandet att bakgrundsbruset är lågfrekvent så kan ett högpassfilter vara en lösning. För att ta reda på frekvensen hos bruset fouriertransformeras det från tidsplanet till frekvensplanet. Genom att studera det spektra som då fås kan frekvensen hos bruset bestämmas. Ett filter simuleras lämpligen i MATLAB. Syftet är att brusets bidrag skall minska.

2.2.2 Medelvärdesfilter

Genom att använda medelvärdet för ett givet antal samples minskas dynamiken i systemet. Det bidrar till att plötsliga förändringar kan ge mindre påverkan i efterföljande beräkningar. Syftet är att få en bättre approximation av sträckan.

2.2.3 Tröskelfilter

Alla värden som ligger under brusets maximala värde sätts till noll. Då ger bruset ingen påverkan, vilket är syftet. En bieffekt blir dock att låg acceleration inte detekteras.

2.3 Accelerometer som stötsensor

Ett objekt som förflyttas blir utsatt för en acceleration. Det kan användas för att övervaka om ett objektet förflyttas eller störs. Om så sker utlöses ett larm. Ett idealt larm kan detektera accelerationer som går mot noll. I verkligheten finns dock ett bakgrundsbrus som gör detta omöjligt.

2.3.1 Detektor

För att kunna avgöra om objektet förflyttats sätts ett tröskelvärde som ligger strax ovanför bruset. Om accelerationer över detta värde detekteras skall användaren meddelas. Detekteringen kan göras med en MATLAB-funktion som söker av värdena från accelerometern och returnerar om något har inträffat, det vill säga om acceleration över tröskelvärdet har påträffats.

2.3.2 Grafiskt användargränssnitt

För att demonstrerar hur en accelerometer kan användas som ett larm kan ett MATLAB-program användas som anropar detektorfunktionen, avsnitt 2.3.1 . MATLAB-programet bör ha ett grafiskt gränssnitt för att ge en tydlig demonstration. Parametrar som är intressanta är det Excel dokument från Freescales demonstrationsprogram som skall analyseras samt känsligheten. Känslighetens nivå måste dock ligga över bakgrundsbruset på den aktuella platsen. Ett informativt gränssnitt talar om för användaren om objektet har utsatts för en acceleration samt ger en graf för mätningen.

3 Tillämpad accelerometer

3.1 Mätlaboration

3.1.1 Utförande av mätlaboration

Accelerometern kalibrerades inför varje mätning och Freescales demonstrationsprogram användes för att samla data till ett Excel dokument.

En person utförde själva hanteringen utav accelerometern, en andra person skötte Freescales demonstrationsprogram.

Följande mätningar gjordes:

Tappa låda – Accelerometern lades i en låda som släpptes från olika höjder - 1 m, 2 m och 5 m. Den släpptes i flera lägen för att få mätningar i olika plan.

Sparka boll – Accelerometern placerades i en boll av tidningspapper och silvertejp som det sedan spelades fotboll med.

Fäktas – Accelerometern fästes på en pinne som det fäktades med.

Små rörelser – Accelerometern hölls i handen och små rörelser gjordes. Tanken var att simulera till exempel en operation.

Promenad – Accelerometern satt i en byxficka och en person promenerade med den.

Springa – Accelerometern satt i en byxficka och en person joggade med den.

Hoppa – Accelerometern satt i en byxficka och en person hoppade jämnfota - små och stora hopp.

Åka kundvagn – Accelerometern fick åka i en kundvagn.

Röra runt – Cirkulerande rörelser gjordes med accelerometern i två plan.

Kast med liten boll – Accelerometern kastades fram och tillbaka mellan två personer.

Åka hiss i Forum – Accelerometern fick åka hiss.

Studsande boll – Bollen från ”Sparka boll” användes och studsades mot golvet i en båge.

Mobilvibrator – Accelerometern låg på en vibrerande mobiltelefon.

3.1.2 Behandling av mätdata

Datan från varje mätning sparades i var sitt Excel-dokument där även grafer skapades. För behandling i MATLAB exporterades datan och sparades i vektorer i så kallade MAT-filer. I MATLAB behandlades sedan datan med funktionerna i avsnitt 3.2.1, 3.2.2, 3.2.3 och 3.3.1.

3.2 Mätning av förflyttning

Första försöket gick ut på att med MATLABs hjälp integrera data från accelerometern. På så sätt kan hastighet och sträcka fås. För att integrera numeriska värden i MATLAB användes funktionen `cumtrapz`. För att filtrera datan från bakgrundsbrus skapades nedanstående funktioner.

3.2.1 Högpassfilter

Fouriertransformeringen gjordes med funktionerna `ftfast` och `iftfast` från kursen Signaler och system, del 1 (SF1635, KTH). En vektor med data skickades in till `ftfast` som i sin tur utförde fouriertransformeringen med avseende på tiden. Spektrat analyserades för att avgöra vid vilken frekvens bakgrundsbruset låg. Ett butterworth-filter applicerades i MATLAB med funktionerna `butter` och `filter` för att dämpa bruset vid frekvensen. Den filtrerade datan återtransformerades sedan med `iftfast` med avseende på frekvensen.

För MATLAB-kod se appendix B.2.

3.2.2 Medelvärdesfilter

En MATLAB-funktion skrevs som beräknade medelvärdet för ett givet antal samples. Funktionen tog som parametrar antal samples samt data. Den beräknade sedan medelvärdet av datan för antal samples och satte dessa till medelvärdet. Utparametrar blev således den filtrerade datan som var mindre känslig för plötsliga förändringar.

För MATLAB-kod se appendix B.3.

3.2.3 Tröskelfilter

En MATLAB-funktion skrevs som sökte igenom alla värden i den inmatade datan. Om den fann ett värde som hade ett absolutbelopp under givet maximalt bakgrundsbrus sattes värdet till noll. Sökningen gjordes med en if-sats som kontrollerade om ett villkor var uppfyllt. Villkoret var antingen att värdet skulle ligga under tröskelvärdet eller över tröskelvärdets negation. Om villkoret uppfylldes sattes det aktuella värdet till noll. Satsen itererade tills alla värden i vektorn med data var kontrollerade och returnerade en vektor med behandlad data.

För MATLAB-kod se appendix B.4.

3.3 Accelerometer som stötsensor

3.3.1 Detektor

För att avgöra om accelerometern utsattes för en rörelse över ett givet tröskelvärde användes MATLAB för att analysera datan. Det gjordes med en funktion som sökte igenom varje axels vektor efter ett värde som uppfyllde villkoret. Om villkoret uppfylldes avbröts sökningen och

funktionen returnerade var den avbröts. Med hjälp av den informationen kunde tidpunkten och accelerationens amplitud bestämmas samt användaren larmas. Om villkoret inte uppfylldes returnerade funktionen även detta och användaren meddelades att inget hade inträffat.

För MATLAB-kod se appendix B.1.

3.3.2 Grafiskt användargränssnitt

För att demonstrera funktionen hos detektorn, avsnitt 3.3.1, skapades ett grafiskt användargränssnitt med hjälp av MATLAB:s GUIDE (GUI Builder). För beskrivning av hur GUIDE fungerar hänvisas till MATLAB:s dokumention då det ligger utanför det här dokumentets innehåll.

4 Resultat och analys

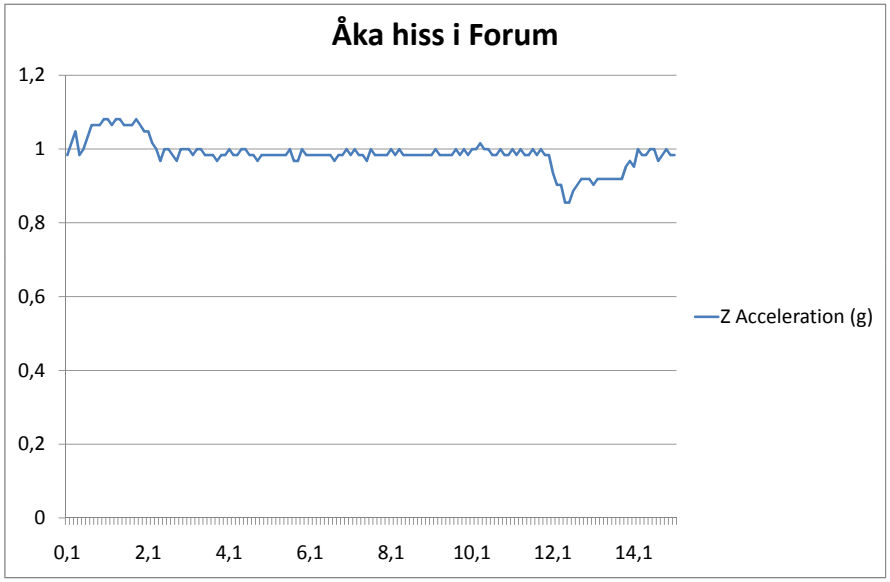
4.1 Mätlaboration

En hel del mätningar fick accelerometern att bottna, se tabell 4.1. Med bottna avses att accelerationen översteg accelerometerns mätområde ($\sim \pm 2,5g$) vilket bidrog till att mätningen blev felaktig ty information föll bort.

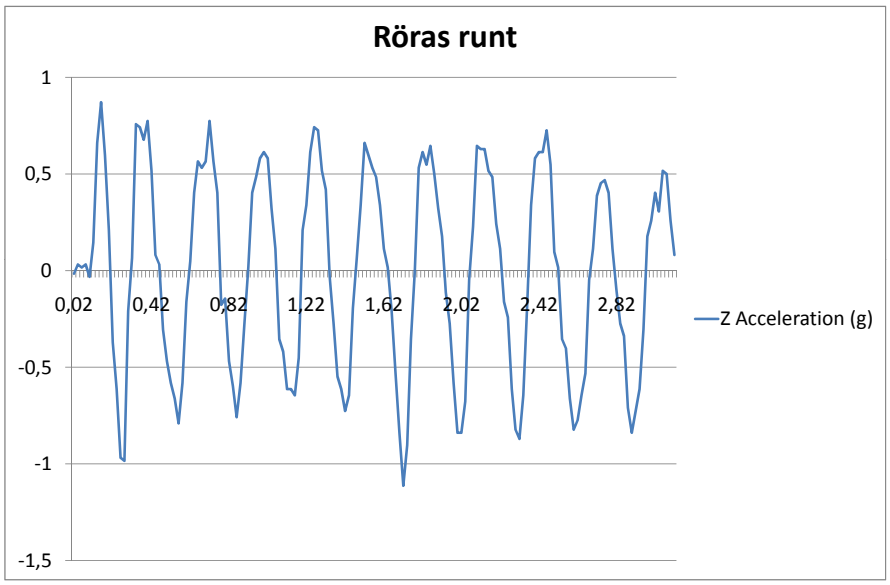
Tabell 4.1: Bottningsresultat från mätlaborationen med accelerometern.

Mätning	Bottnade
Tappa låda	Ja
Sparka boll	Ja
Fäktas	Ja
Små rörelser	Nej
Promenad	Nej
Springa	Ja
Hoppa	Ja
Åka kundvagn	Ja
Röra runt	Nej
Kast med liten boll	Ja
Åka hiss i Forum	Nej
Åka hiss i Kista galleria	Nej
Studsande boll	Ja
Mobilvibrator	Ja

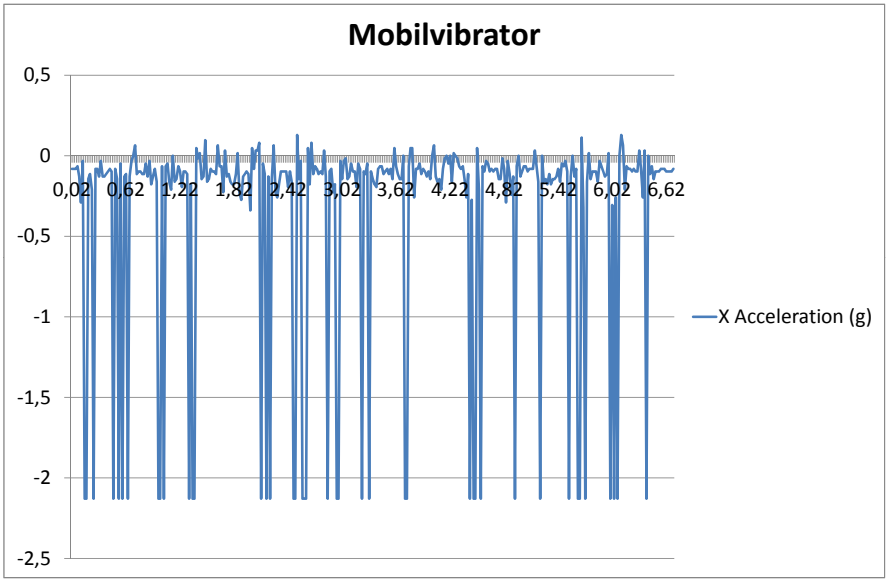
Här är de mest intressanta graferna från mätlaborationen, se avsnitt 3.1 för beskrivning av de olika mätningarna. För fler grafer se appendix A.



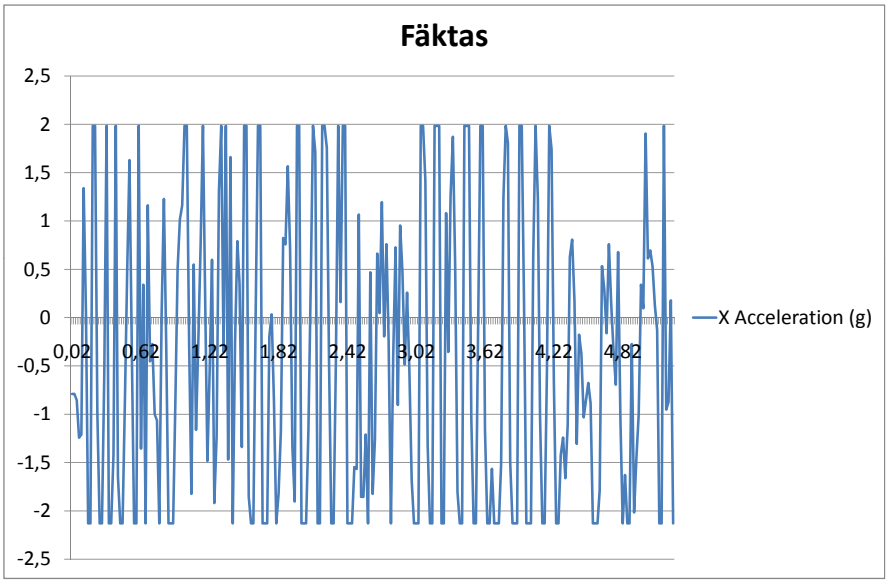
Figur 4.1: Resultat från mätning ”Åka hiss i Forum”.



Figur 4.2: Resultat från mätning ”Röra runt”.



Figur 4.3: Resultat från mätning ”Mobilvibrator”.



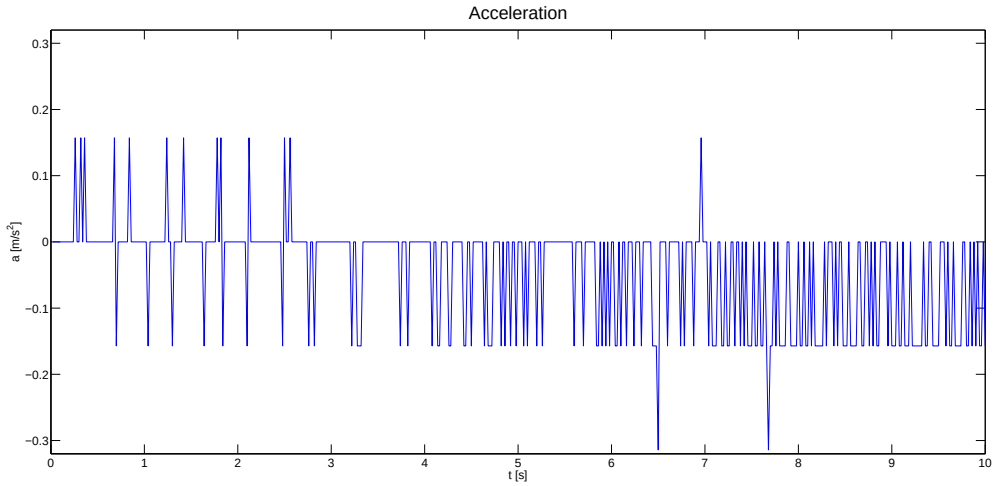
Figur 4.4: Resultat från mätning ”Fäktas”.

4.2 Mätning av förflyttning

Inga resultat som låg i närheten av den verkliga sträckan kunde räknas fram.

Till exempel gav mätningar av en sträcka på 2 m ett resultat på ungefär 0,3 m.

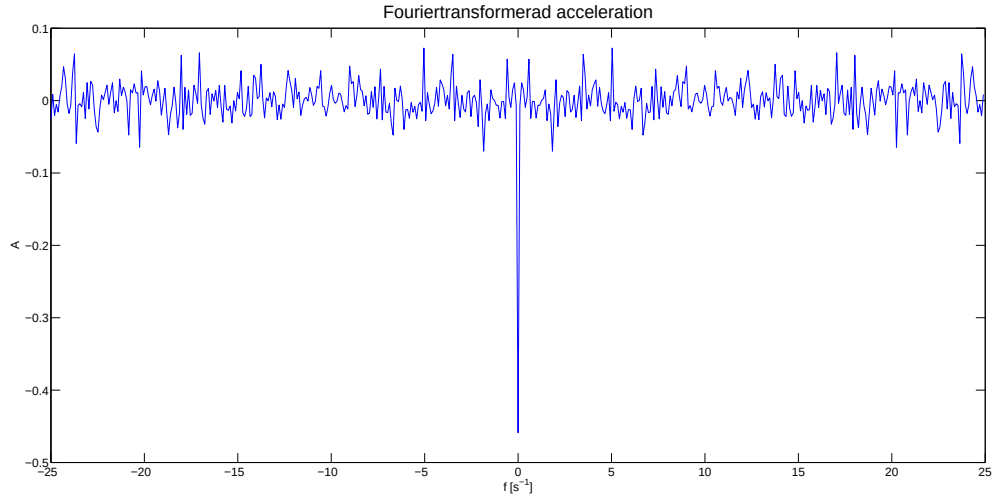
Figur 4.6 visar en mätning av bakgrundsbrus.



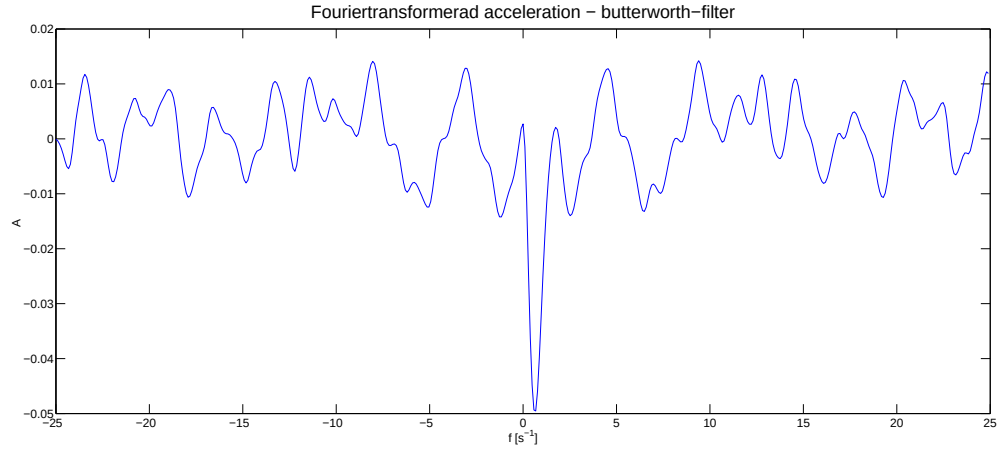
Figur 4.5: Exempel på obehandlad data bestående av bakgrundsbrus.

4.2.1 Högpassfilter

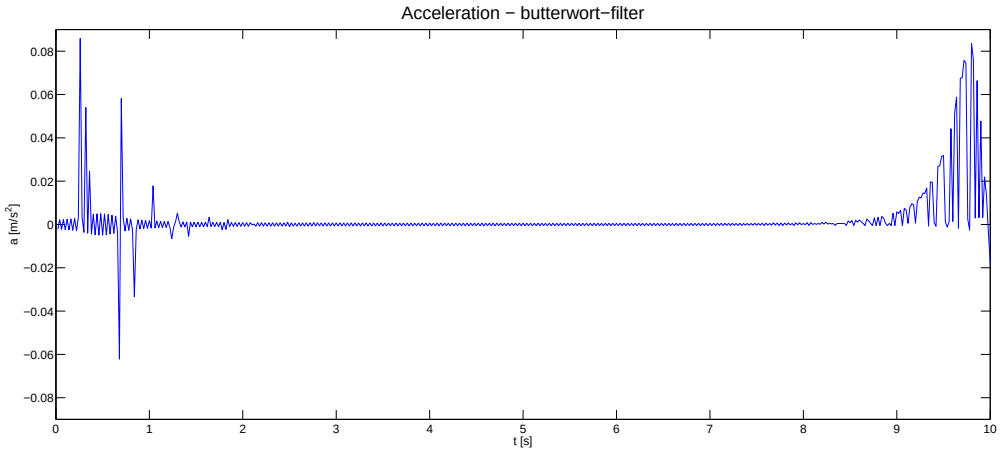
Bakgrundsbruset låg i området kring 0 Hz, se toppen i fouriertransformen i figur 4.6. Filtrerades detta området så gav det ett betydande fel, se figur 4.7 och 4.8.



Figur 4.6: Fouriertransform av bakgrundsbruset i figur 4.5 för analys av frekvensspektrat.



Figur 4.7: Bakgrundsbruset (figur 4.5), i frekvensplanet, filtrerat med ett butterworth-filter.



Figur 4.8: Bakgrundsbruset (figur 4.5), i tidsplanet, filtrerat med ett butterworth-filter.

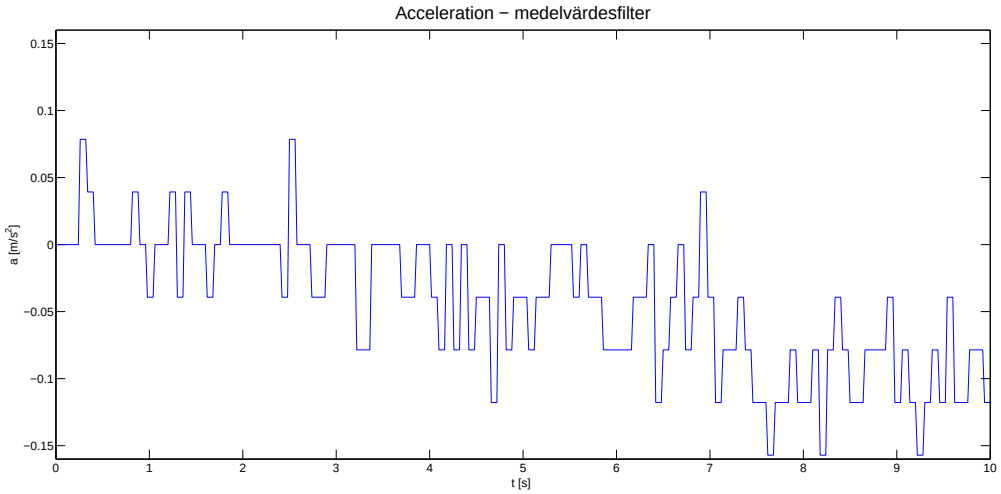
4.2.2 Medelvärdesfilter

Filtret minskade dynamiken, se figur 4.9, vilket gav mjukare kurvor efter integreringarna men påverkade inte slutvärdet.

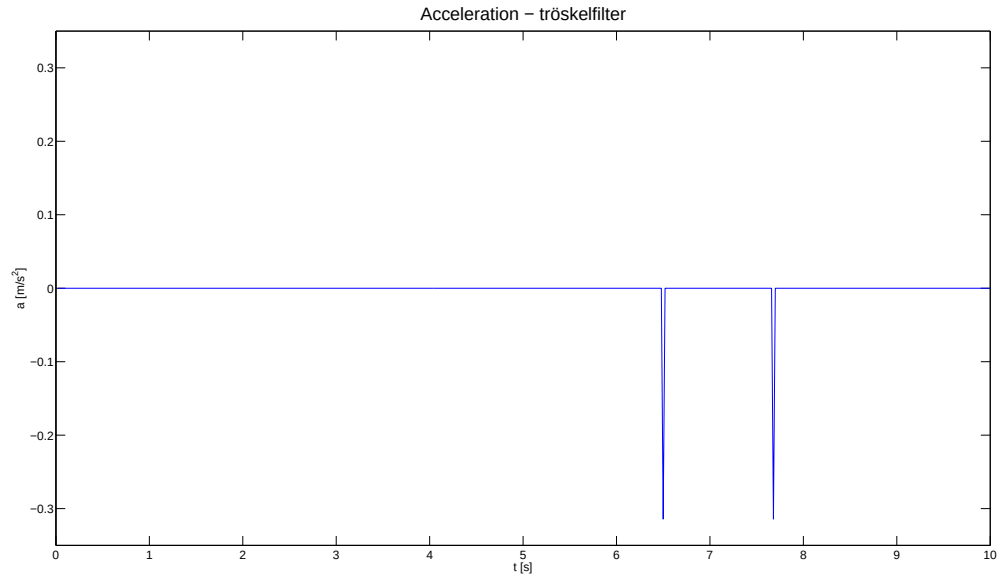
4.2.3 Tröskelfilter

Maximumvärdet för bakgrundsbruset varierade mellan $\pm 0,016$ och $\pm 0,032$.

Det visade sig att bakgrundsbruset bara var en del av felet och att filtrera bort det gav fortfarande ett stort fel på slutvärdet.



Figur 4.9: Bakgrundsbruset i figur 4.5 filtrerat med ett medelvärdesfilter.



Figur 4.10: Bakgrundsbruset i figur 4.5 filtrerat med ett tröskelfilter.

4.3 Accelerometer som stötsensor

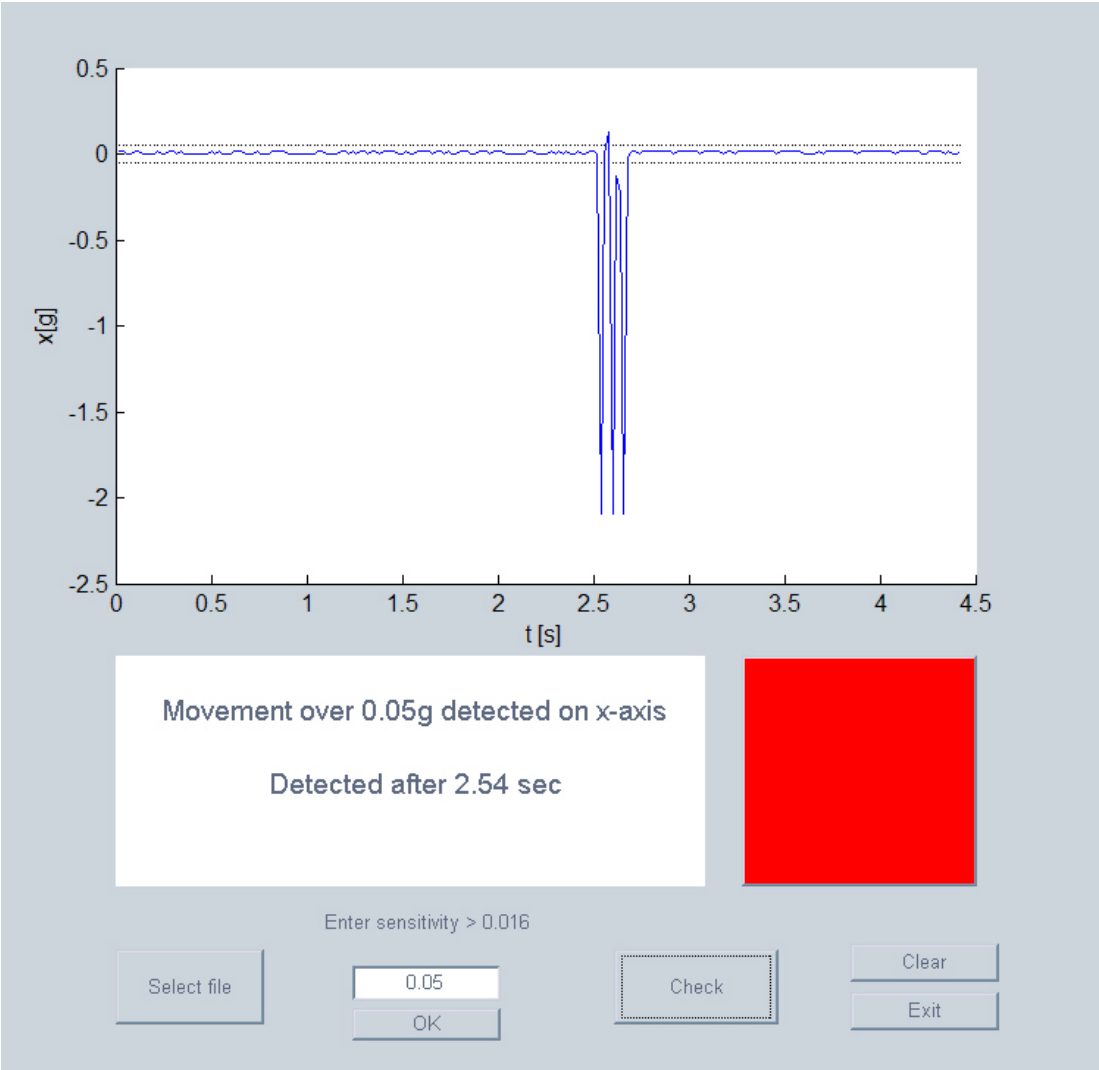
Det här försöket resulterade i en MATLAB-funktion som analyserar datan och ett MATLAB-program med grafiskt användargränssnitt.

4.3.1 Detektor

En MATLAB-funktion skrevs för att analysera datan och detektera accelerationer över tröskelvärdet. För MATLAB-kod, se appendix B.

4.3.2 Grafiskt användargränssnitt

Användaren väljer vilket Excel-dokument med data som skall analyseras och vilken känslighet som önskas. Om en rörelse över tröskelvärdet (känsligheten) detekteras så meddelar programmet användaren. Om inget detekteras så meddelas det istället. Gränssnittet vid utlöst larm kan beskådas i figur 4.11.



Figur 4.11: Larmets grafiska användargränssnitt.

5 Diskussion och slutsatser

Vi kom på flera tillämpningar för en accelerometer under projektets uppstartsfas - gitarreffekt, trummaskin, dansmaskin och vattenpass. Dessvärre visade det sig att dessa tillämpningar redan existerade. Vi begränsades också av att vår accelerometer bottnade vid endast ett par g. Vilket kan ses tydligt på flertalet av våra mätningar, se tabell 4.1 för vilka. Slutligen bestämde vi oss för att använda accelerometern till att göra ett larm.

5.1 Mätlaboration

Accelerometerns beteende skilde sig betydligt från fall till fall. Detta beror på att den accelerometer som har använts vid detta projekt var begränsad till ungefär $2g$. För en pedometer, eller stegräknare, krävs till exempel en accelerometer som klarar mellan $10g$ och $20g$.

De flesta ”vardagliga” accelerationspåverkningarna var mycket kraftigare än vi hade förutspått, se till exempel graferna för Springa (figur A.6) och Promenad (figur A.4). Vid mindre rörelser var det även svårt att särskilja dessa från bakgrundsbruset.

5.2 Mätning av förflyttning

Först tänkte vi beräkna sträckan den förflyttas och hålla koll på var i ett koordinatsystem den befann sig. Tyvärr misslyckades vi med att utföra dessa beräkningar och gav upp efter många timmars experimenterade. Misslyckandet berodde med största sannolikhet på otillräckligt filter. Efter en del undersökning visade det sig att ett Kalmanfilter kan vara lösningen på problemet. Dock saknade vi kunskaper och tid för att kunna implementera ett sådant.

5.2.1 Högpassfilter

Vi var väldigt osäkra på hur ett högpassfilter skulle implementeras i MATLAB och lyckades ej heller något vidare med detta. Det verkar visserligen dämpa datan runt 0 Hz, men även mer därtill. Dessutom ger filtret i sig fasförskjutning och andra fenomen. Med mer tid så hade vi förhoppningsvis kunnat sätta oss in i hur man gör ett lämpligt filter för digital data.

5.2.2 Medelvärdesfilter

Idén till det här filtret fick vi från Freescales demonstrationsprogram. Det hade nämligen något liknande i sitt ”Scope” med en beskrivande text. Filtret gör att accelerometern inte blir så extremt känslig och graferna vid integrering blir betydligt mjukare.

5.2.3 Tröskelfilter

Ett tappert men brutalt försök till att bli av med bakgrundsbrus och förhoppningsvis få korrektare resultat av sträckan. Nackdelen är att all information med låg acceleration försvinner vilket

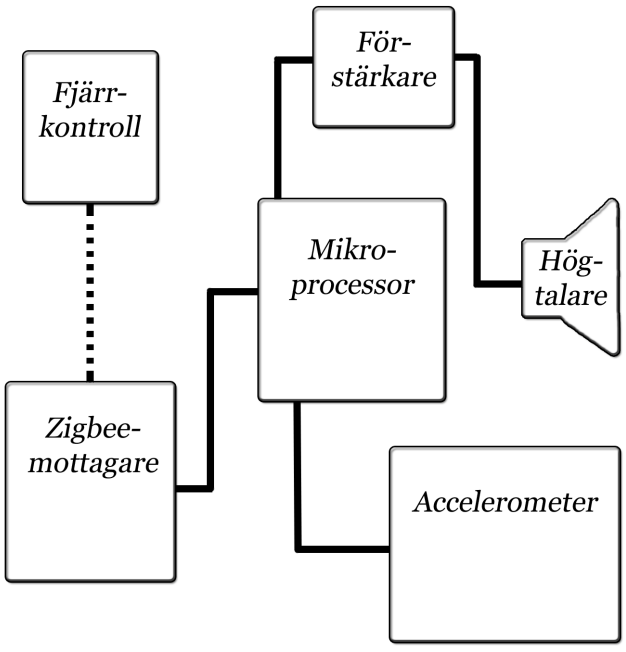
innebär att om man skulle gå mycket långsamt så skulle larmet inte detektera någon rörelse. Med andra ord olämpligt för ett larm.

5.3 Accelerometer som stötsensor

Eftersom våra försök med att beräkna förflyttning misslyckades så ändrade vi vår tillämpningsspecifikation. Istället valde vi att bara använda accelerometern som en stötsensor och övervaka objektet från rörelser över en given acceleration.

För en slutgiltig produkt som kan gå i produktion ser vi två tänkbara tillämpningar. Den första är att ha hela produkten integrerad i ett paket med accelerometer, mikroprocessor, ZigBee-mottagare, förstärkare och högtalare som kan fästas på det objekt som skall övervakas. Känsligheten och läget kan ställas in med en knapp som kalibrerar larmet för rådande bakgrundsbrus och position. När accelerationen överstiger känsligheten ljuder högtalaren. Återställning görs med en fjärrkontroll över krypterad ZigBee-länk. Figur 5.1 är ett tänkbart blockschema för denna produkt.

Den andra tänkbara tillämpningen är att ha produkten i två delar. En central som övervakar ett antal sändare placerade på olika objekt. Centralen kan till exempel vara en dator men även vara en fristående enhet. Sändarna kommunicerar med centralen över en krypterad ZigBee-länk och när ett objekt störs, till exempel ett fönster som öppnas eller krossas, larmas användaren på önskvärt sätt via centralen. Återställning sker med kod på centralen eller med en till centralen kopplad styrenhet.



Figur 5.1: Blockschema för en tänkbar produkt.

5.3.1 Detektor

Detektorn gör det som är dess huvudsakliga uppgift. Den returnerar om ett värdet över tröskelvärdet har påträffats. Men vad den även skulle kunna göra är att tala om hur stort utslag som skett samt hur många gånger. För en verklig tillämpning krävs det dessutom att den analyserar i realtid. Sen kan man fråga sig hur högt bakgrundsbrus skall hanteras, som till exempel om man befinner sig på ett tåg som vibrerar konstant.

5.3.2 Grafiskt användargränssnitt

Utöver att vara ett gränssnitt till detektorn skall även det grafiska användargränssnittet ta emot några parametrar (känslighet och Excel-dokument som skall analyseras) och ge en graf av mätdatan. Det kanske inte är ett typexempel på god designstudie och kanske ej heller världens bästa kod, men det demonstrerar hur en accelerometer i princip kan användas som larm och uppfyller därmed sitt syfte fullt tillräckligt.

Referenser

[1] Bo Göran Ericsson. Accelerometrar mäter fem parametrar. *Elektronik i norden*, (19):43–44, 2004.

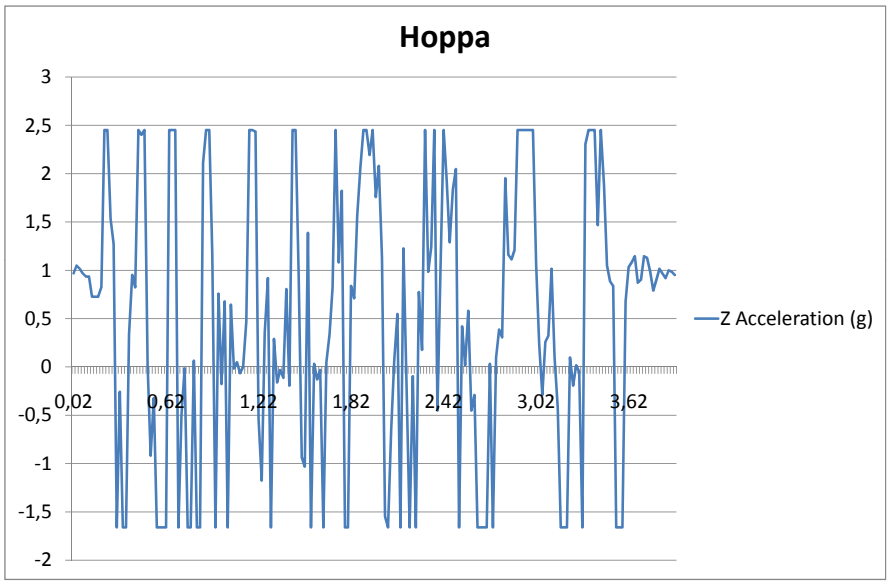
[2] Freescale Semiconductor. Mems technology. $http://www.freescale.com/files/sensors/doc/fact\_sheet/MEMSFS.pdf$, 2008-05-08.

[3] Freescale Semiconductor. Mma7360l accelerometer. $http://www.freescale.com/files/sensors/doc/fact\_sheet/ZSTAR2BOARDFS.pdf$, 2008-05-08.

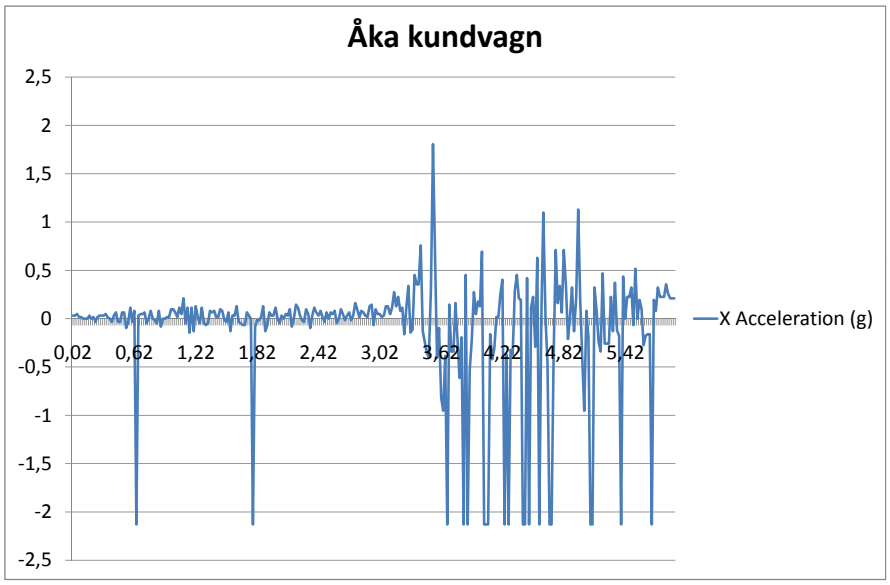
[4] Wikipedia. g-force. $http://en.wikipedia.org/wiki/Gforce$, 2008-05-08.

[5] Wikipedia. Zigbee. $http://en.wikipedia.org/wiki/ZigBee$, 2008-05-14.

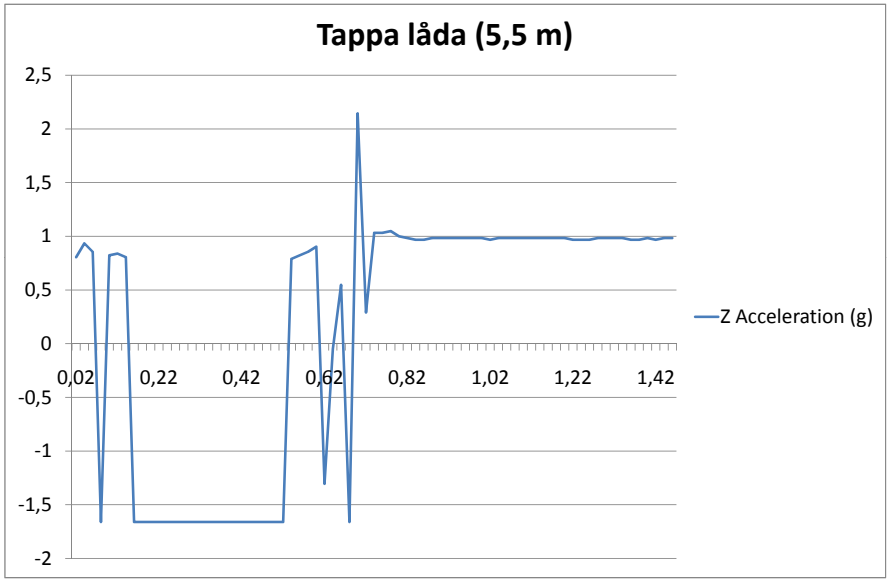
A Grafer



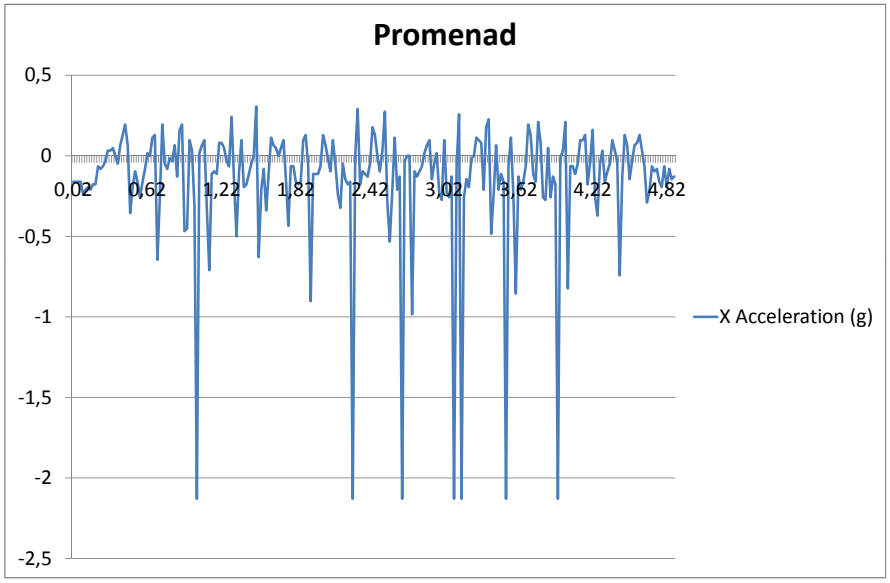
Figur A.1: Resultat från mätning ”Hoppa”.



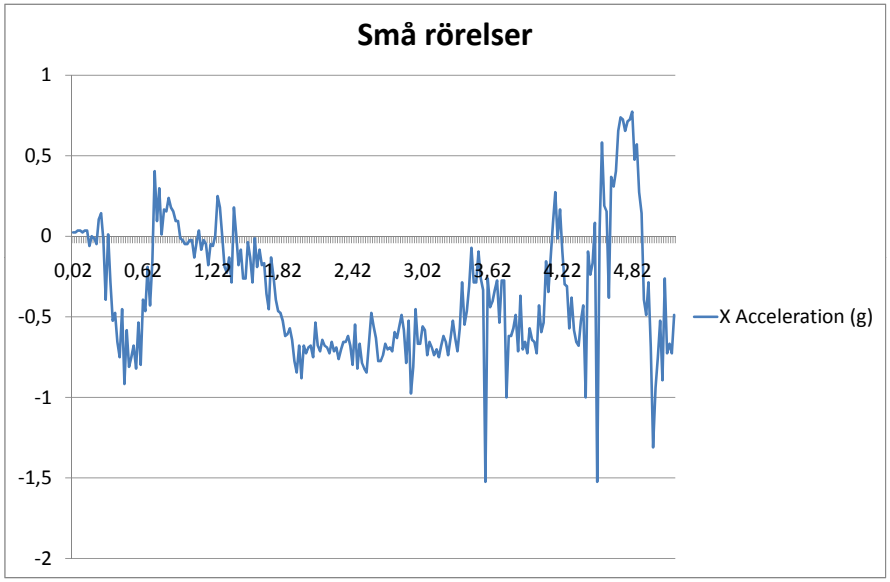
Figur A.2: Resultat från mätning ”Åka kundvagn”.



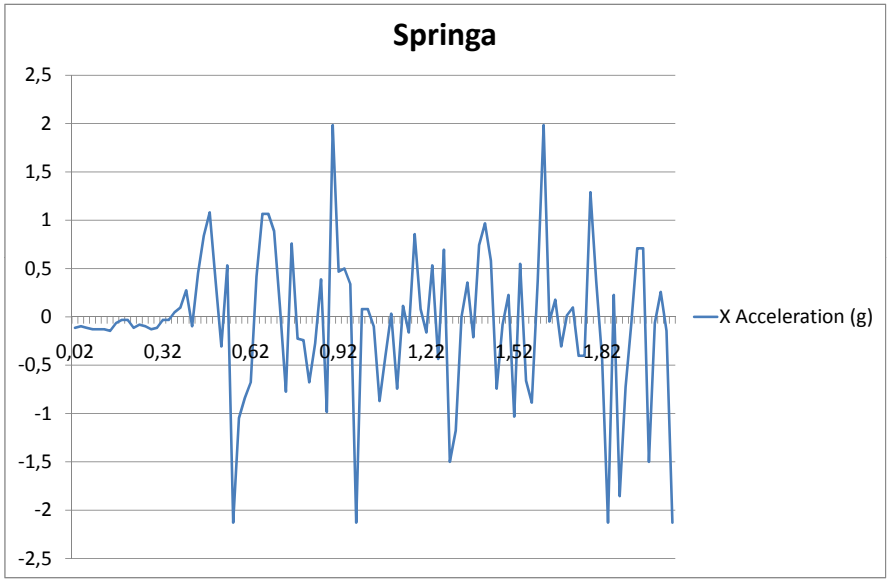
Figur A.3: Resultat från mätning ”Tappa låda”.



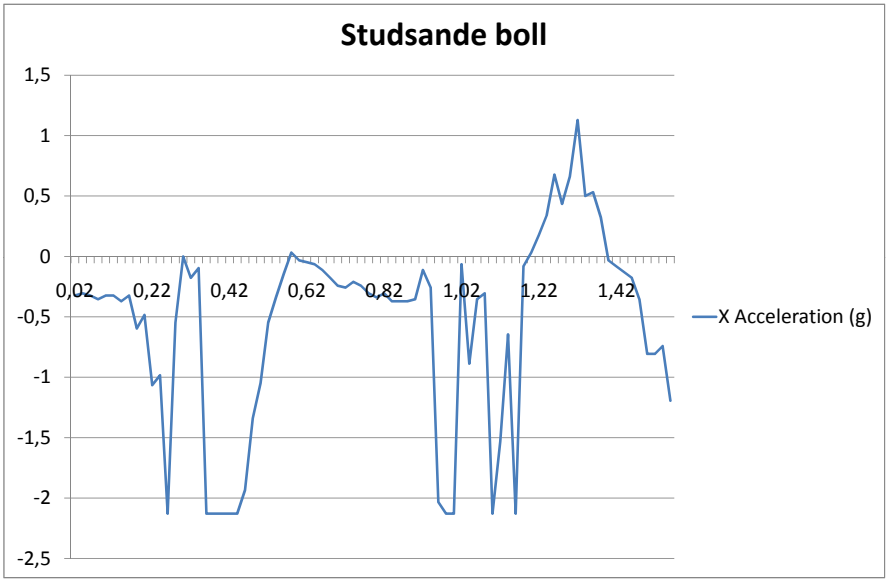
Figur A.4: Resultat från mätning ”Promenad”.



Figur A.5: Resultat från mätning ”Små rörelser”.



Figur A.6: Resultat från mätning ”Springa”.



Figur A.7: Resultat från mätning ”Studsande boll”.

B MATLAB-kod

B.1 detector.m

```
%%% Detector v. 1.1 2008-04-16
%%%
%%% KTH ICT, Kista
%%% Fredrik Lundgren, flundg@kth.se
%%% Joel Nilsson, joel@alikzus.se
%%% Martin Axelsson, maxels@kth.se
%%%
%%% INPUT: x, y, z and sens
%%% OUTPUT: bol, axis and count
%%%
%%% The function scans the x, y and z vectors for values above sens.
%%% It returns bol = 1 if there is any value above sens and bol = 0
%%% otherwise.
%%% It also returns the string axis who tells which axis the first value was
%%% deteceted at and at which index in the vector (count).

function [bol,axis,count] = detector(x,y,z,sens)
```

```
    k = 1; % Internal counter
    n = length(x); % The length of vector x (the others has the same length)

    while k ~= 0 && k < n+1

        % Searches for value above sens at the x axis and returns bol = 1,
        % axis = 'x' and count = k
        if abs(x(k)) > sens
            count = k;
            bol = 1;
            axis = 'x';
            k = 0;
        % ... y axis ...
        elseif abs(y(k)) > sens
            count = k;
            bol = 1;
            axis = 'y';
            k = 0;
        % ... z axis ...
        elseif abs(z(k)) > sens
            count = k;
            bol = 1;
            axis = 'z';
            k = 0;
        % If nothing is found it iterates the while loop
        else
            k = k+1;
        end

        % When nothing is found at all it returns bol = 0 and axis = 'none'
        if k == n+1
            count = k;
            bol = 0;
            axis = 'none';
        end

    end

end
```

B.2 bw_filter.m

```
%%% Butterworth filter v. 1.0 2008-05-14
%%%
%%% KTH ICT, Kista
%%% Fredrik Lundgren, flundg@kth.se
%%% Joel Nilsson, joel@alikzus.se
%%% Martin Axelsson, maxels@kth.se
%%%
%%% INPUT: data, time and order
%%% OUTPUT: data
%%%
%%% The function filtres the data with a butterworth filter

function [data] = bw_filter(data,time,order)

% Gravitation constant
grav = 9.82; % [m/s^2]

% Translates the g-force to acceleration
acc = zg*grav;

% Uses the ftfast function to do a fourier transform
[A,f] = ftfast(acc,t);

% Usss the butter function to calculate the vectors needed by the filter
% function for a n order butter
[b,a] = butter(order,0.1);

% Filters the signal
Af = filter(b,a,A);

% Uses the iftfast function to do an inversed fourier transform
af = iftfast(Af,f);

data = af;
```

B.3 avrg_filter.m

```
%%% Average filter v. 1.0 2008-04-16
%%%
%%% KTH ICT, Kista
%%% Fredrik Lundgren, flundg@kth.se
%%% Joel Nilsson, joel@alikzus.se
%%% Martin Axelsson, maxels@kth.se
%%%
%%% INPUT: data and samples
%%% OUTPUT: data
%%%
%%% The function takes a number of samples and calculates the average value
%%% of those.
%%% It then returns the equalized vector.

function [data] = avrg_filter(data,samples)

n = 1; % Internal counter

while n < length(data)

    sum = 0;

    % Loops through each index of the vector
    if length(data) - n < samples

        % Sums the number of samples
        for k=1:length(data)-n
            l = k+n-1;
            sum = sum + data(l);
        end

        % Calculates the average of the sum
        for k=1:length(data)-n
            avrg = sum/(length(data)-n);
            l = n+k-1;
            data(l) = avrg;
        end

    else

        % Same as above with the exception that the remaining values ain't
```

```
        % equal divideable with samples
        for k=1:samples
            l = n+k-1;
            sum = sum + data(l);
        end

        for k=1:samples
            avrg = sum/samples;
            l = k+n-1;
            data(l) = avrg;
        end

    end

    n = n+samples; % Iterates the loop

end

end
```

B.4 noise_filter.m

```
%%% Noise filter v. 1.0 2008-04-16
%%%
%%% KTH ICT, Kista
%%% Fredrik Lundgren, flundg@kth.se
%%% Joel Nilsson, joel@alikzus.se
%%% Martin Axelsson, maxels@kth.se
%%%
%%% INPUT: data and limit
%%% OUTPUT: data
%%%
%%% The function sets all values below limit to zero.
%%% It returns the modified vector.

function [data] = noise_filter(data,limit)

l = length(data); % The length of the data vector

% Loops from index 1 to l
for n=1:l
    % If value of data at index n is between zero and limit the value is
    % changed to zero
    if data(n) > 0 && data(n) <= limit
        data(n) = 0;
    % If value of data at index n is between zero and -limit the value is
    % changed to zero
    elseif data(n) <= 0 && data(n) >= -limit
        data(n) = 0;
    end
end

end
```