# *Styring af objekter i 3D vha. Wii-controller*

***Bachelorprojekt, forår 2007***

*af Lasse Jon Fuglsang Pedersen*
*og Anders Sabinsky Tøgern*

*Vejleder: Jon Sporring*

*13. juni 2007*

*Datalogisk Institut*
*Københavns Universitet*

**Resumé**

Dette er et bachelorprojekt ved Datalogisk Institut ved Københavns Universitet (DIKU). I denne rapport beskriver vi hvordan vi har udviklet et klassebibliotek, WiiLib, til kommunikation med, og behandling af data fra en Wii-controller. Wii-controlleren er input-enheden til Nintendo's nyeste konsol, Wii. Da Nintendo har designet controlleren således, at den kan kommunikere med en almindelig PC over Bluetooth er det oplagt, at forsøge at anvende controlleren med en PC. Projektet indbefatter desuden også en implementation af en WiiLib-baseret udvidelse til 3DOT; en 3D-motor under udvikling på DIKU.

WiiLib gør det muligt, for en applikation, at udlæse estimater af Wii-controllerens orientering og relative position. Estimaterne kan eksempelvis anvendes til manipulation af objekter i et 3D-miljø, og beregnes på baggrund af forskellige metoder; bl.a. en adaptiv metode, som vi selv har udviklet og som beskrives i rapporten. Resultatet af den estimerede orientering er tilfredsstillende ved brug af WiiLibs adaptive metode, men falder i præcision hvis controlleren er i samtidig bevægelse. Resultatet af den estimerede position er derimod ikke tilfredsstillende, da dette påvirkes kraftigt af støj, og derfor kun fungerer under absolut perfekte forhold.

**Nøgleord** Nintendo, Wii, Wii-controller, Bluetooth, WiiLib, 3DOT

**Abstract**

This is a bachelor project at the Department of Computer Science at the University of Copenhagen (DIKU). In this report we describe how we have developed a class library – WiiLib – that enables communication with and processing of data from a Wii-controller. The Wii-controller is an input device for Nintendo's recent console – Wii. Since Nintendo has designed the controller in a way that makes it possible for it to communicate with a PC through Bluetooth, there is a motive of curiousity behind experimenting with the controller connected to a PC. The project also includes an implementation of a WiiLib-based extension to 3DOT; a 3D-engine in development at DIKU.

WiiLib makes it possible for an application to read estimates of the Wii-controller's orientation and relative position. The estimates can be used to manipulate objects in a 3D environment and are calculated using several methods; among these an adaptive method which we have developed ourselves and described in the report. The result of the estimated orientation is satisfying using WiiLib's adaptive method, but loses precision when the controller is also exposed to non-rotational movement. The result of the estimated position, on the other hand, is not satisfying because it is heavily affected by noise and therefore only works under absolutely perfect conditions.

**Keywords** Nintendo, Wii, Wii-controller, Bluetooth, WiiLib, 3DOT

# Forord

Vores implementation er ikke et direkte svar på problemformuleringen i vores synopsis, da vi ikke har konstrueret en decideret driver til en specifik 3D-motor, men istedet et klassebibliotek, der kan anvendes af enhver applikation, bl.a. en udvidelse til 3DOT.

Vores afprøvning afviger ligeledes fra den foreslåede i vores synopsis, da vi har valgt at fokusere på præcisionen af metoderne til estimaterne i stedet for praktisk anvendelse af klassebiblioteket. Vi mener at dette er en bedre afprøvning, da et mål for præcisionen af estimaterne er et bedre argument for anvendeligheden af klassebiblioteket end praktiske demoer.

Derudover har vi, som svar på problemformuleringen i vores synopsis, implementeret en udvidelse til 3DOT, som et eksempel på en driver til en 3D-motor, der indirekte faciliterer styring af objekter i 3D vha. Wii-controlleren.

Det skal nævnes at Wii er et varemærke af Nintendo og at vi kun har benyttet offentligt tilgængelig materiale i projektet.

Denne rapport er også tilgængelig som PDF: `http://qeep.dk/~volmer/bach/rapport.pdf`.

Kildekoden til klassebiblioteket, udvidelsen til 3DOT, og diverse applikationer til afprøvning, samt alle datasæt fra vores afprøvning, er tilgængelige på adressen: `http://qeep.dk/~volmer/bach/`.

# Konventioner for rapport

For at gøre rapporten konsistent og lettere at læse har vi vedtaget nogle konventioner om formattering og ordvalg.

Gennemgående benytter vi engelske fagtermer, hvor vi finder det passende, da det kan være svært at finde danske erstatninger for nogle engelske fagord, og da det som regel letter forståligheden at benytte samme ord, som i litteraturen.

Hver gang et nyt udtryk introduceres i teksten skrives det med kursiv, så det er lettere for læseren at lægge mærke til nye begreber. Yderligere findes der en forklaring af en del begreber i Appendix A.

# Indhold

# 1 Introduktion

Ofte har man i interaktive 3D-scener (eksempelvis computerspil) brug for at interagere med et eller flere objekter i det virtuelle rum. Typisk drejer det sig om at rotere eller flytte et objekt – operationer der er lette at visualisere eftersom de også sker i virkeligheden.

Traditionelle input-enheder, såsom mus og joysticks, er ofte begrænsede af deres to-dimensionelle uddata, der enten kræver oversættelse eller yderligere parametre før de kan anvendes til tre-dimensionelle operationer i det virtuelle rum. At flytte et objekt i tre dimensioner kan eksempelvis håndteres ved brug af en eller flere ekstra knapper, hvor knappernes status angiver hvilke akser i det tre-dimensionelle rum, som de to-dimensionelle uddata fra input-enheden skal anvendes på. Dette er dog næppe særligt intuitivt for brugeren, der skal interagere med scenen.

Nintendo har til deres nye spilkonsol, Wii [Nintendo, 2006], lavet en ny type input-enhed (Wii-controller). Wii-controlleren er trådløs og indeholder bl.a. et 3-akse accelerometer [Analog Devices, 2006], der gør controlleren i stand til at rapportere dens egen acceleration i tre ortogonale akser. Controlleren kommunikerer vha. Bluetooth, hvilket gør det muligt at tilslutte den til en almindelig computer.

Al kommunikation sker i et lukket pakkeformat, men siden lanceringen af controlleren er det lykkedes forskellige kilder [WiiLi, 2007, WiiBrew, 2007], at ræsonere sig frem til en delmængde af pakkeformatet, hvorfor det nu er muligt at tolke dele af input fra controlleren. Blandt andet er det muligt at aflæse controllerens acceleration i alle tre akser.

## 1.1 Problemformulering

I dette projekt undersøger vi mulighederne for at anvende Wii-controlleren til operationer i et virtuelt rum, med en forventning om, at dette kan lade sig gøre med acceptabel præcision. Vores produkt er en implementation af de gennemgåede teknikker i form af et klassebibliotek, WiiLib, der faciliterer styring af objekter i 3D vha. Wii-controlleren.

På DIKU arbejdes også på udvikling af 3D-motoren 3DOT [3DOT, 2007], hvorfor det vil være oplagt at tilbyde muligheden for at kunne benytte controlleren i denne. Vi vil derfor også tilbyde en baseline implementation af en Wii-controller-udvidelse til 3DOT.

# 2 Teknisk baggrund

I dette afsnit beskriver vi de tekniske elementer, der ligger til grund for vores analyse og implementation. Afsnittet vil primært beskæftige sig med Wii-controlleren, og vil tage udgangspunkt i hvorledes der kommunikeres med denne, helt ned på protokol-niveau.

## 2.1 Bluetooth

Bluetooth er en meget populær standard inden for trådløs kommunikation mellem mobile enheder. Bluetooth anvender radiobølger og er derfor ikke afhængig af at brugeren peger den mobile enhed mod modtageren eller senderen, som man skal ved f.eks. infrarød kommunikation. Dette giver en større bevægelsesfrihed, som er oplagt at benytte i spil.

Bluetooth har typisk en rækkevidde på 10 meters radius mellem master- og slave-enheden og er således ikke retningsbestemt.

Der findes to typer Bluetooth enheder; *master* og *slave*. Hvor op til syv slave-enheder kan forbinde til en master-enhed. Master-enheder findes ofte i computere og andre mere eller mindre stationære enheder, hvorimod slave-enheder ofte findes i mobile enheder som headsets og fjernbetjeninger.

For at forbinde en slave-enhed til en master-enhed skal slave-enheden være *discoverable*, så master-enheden kan se den. Når en enhed får øje på en anden får den tilsendt enhedens addresse og kan herefter forespørge om dennes navn, klassifikation, tilgængelige services o.a.

På denne måde ved master-enheden hvilken type slave-enheden er, og hvilke services denne tilbyder. Herefter kan master-enheden vælge at forbinde til slave-enheden, hvilket kan kræve at enhederne kender hinandens *passkey*[1], som sørger for at det kun er tilladte enheder der forbinder til hinanden.

Når to enheder er forbundet til hinanden kan disse parres, hvilket vil sige at de to enheder fremover kan forbinde automatisk til hinanden. Parring er kun muligt hvis slave-enheden har en passkey.

Alle Bluetooth-enheder har et *VendorID* og et *DeviceID*, der beskriver enhedens producent og produkt. Alle produkter fra den samme producent har således det samme VendorID og alle ens produkter fra den samme producent har det samme DeviceID.

Alle enheder der benytter Bluetooth kommunikation har en indbygget stak til at håndtere forbindelserne samt indkomne og udgående pakker. På en computer er der forskellige implementationer af Bluetooth stakke alt efter hvilket operativsystem der er installeret på computeren eller hvilken hardware dette kører på.

Windows operativsystemet er distribueret med en indbygget Bluetooth stak, som desværre kun understøtter en begrænset mængde af Bluetooth radio-enheder. Derfor findes der op til flere tredjeparts implementationer af Bluetooth stakke, hvoraf en af de mest brugte er BlueSoleil [IVT corporation, 2007], som især distribueres i bundle med Bluetooth-*dongles*.

Bluetooth stakken indeholder en *buffer*, for hver tilsluttet enhed, der sørger for at opsamle indkomne pakker og det er fra denne buffer man læser data fra en enhed. Nogle stakke tillader brugeren at ændre størrelsen af denne buffer, hvilket kan have

[1] Ofte er det kun slave-enheden der har en passkey

indflydelse på de pakker man kan læse fra bufferen.

Hvis bufferen kun kan indeholde én pakke vil man altid læse den nyeste pakke, men sandsynligheden for at der går en masse pakker tabt er ret stor. Hvis bufferstørrelsen derimod er stor kan der være et stort *lag* fra en pakke er afsendt fra en enhed til den læses fra bufferen, hvis der ikke læses med samme frekvens, eller hurtigere, som enheden afsender pakker.

## 2.2 Wii-controller

Wii-controlleren (se Figur 1) har tolv knapper, der lader brugeren styre spil afhængigt eller uafhængigt af bevægelse; fire lysdioder, *rumble*-motor og en højttaler, der giver brugeren *feedback*; en *CMOS* sensor [Pixart Imaging Inc., 2006, pressemeddelelse af 15. maj 2006], der gør det muligt at benytte controlleren som pegeredskab ved hjælp af infrarøde dioder placeret på hver side af skærmen; samt et 3-akse accelerometer [Analog Devices, 2006], der gør det muligt at benytte controlleren frit i luften. Disse komponenter giver brugeren stor bevægelsesfrihed ved interaktion med en konsol eller en computer.


**Figur 1:** Wii-controlleren set fra forskellige vinkler. *3D Studio Max model af [Grubb, 2007].*

Wii-controlleren indeholder ikke en passkey, så det er ikke muligt at parre den med en traditionel Bluetooth master-enhed, såsom en computer. Dette vil sige at controlleren manuelt skal forbindes hver gang den skal tilsluttes.

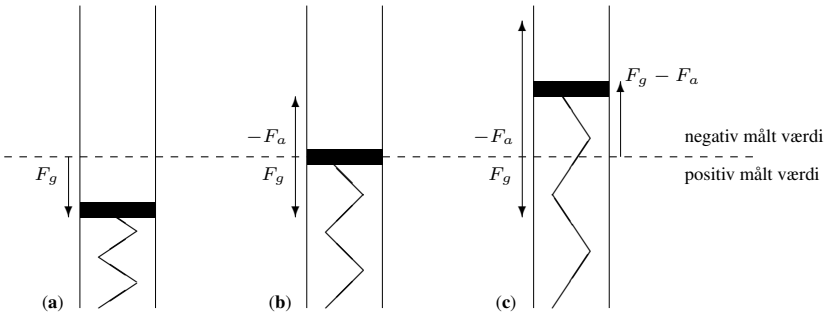
For at sætte Wii-controlleren i discoverable mode holdes knapperne *1* og *2* (se Figur 1) nede samtidigt, hvorefter controlleren vil være discoverable i et begrænset tidsrum. Der findes også en knap under batteri-dækslet til dette. Man skal derfor nå at forbinde

controlleren i dette tidsrum, eller holde knapperne nede indtil man har forbundet. Forbindelsen holdes automatisk åben indtil denne afbrydes af brugeren, hvilket kun kan ske fra master-enheden, eller hvis der sker en fejl der har indflydelse på forbindelsen, så som strømsvigt i controlleren.

### 2.2.1 Accelerometer

Et accelerometer er en enhed, der måler sin egen acceleration. Et accelerometer med 3 akser måler således sin egen acceleration i det 3-dimensionale rum. Accelerometeret i Wii-controlleren har 3 akser. Accelerometeret kan eksempelvis benyttes til at detektere hældning i forhold til tyngdekraften, eller relativ position i rum.

Accelerationen måles på baggrund af den eksterne påvirkning af accelerometeret, inklusive tyngdekraften, hvilket forklares bedst ved et eksempel: Hvis man betragter accelerometerets akser som 3 ortogonale „fjedre“, da ses rent intuitivt, at en bevægelse i én retning vil udvide eller komprimere fjedrene i den stik modsatte retning. I fjedereksemplet er den målte acceleration da udtrykt ved den kraft, hvormed fjederen udvides eller komprimeres som følge af en ekstern påvirkning (se Figur 2).




**Figur 2:** Figuren illustrerer hvordan en akse i accelerometeret opfører sig, når den påvirkes af kraften fra en reel acceleration, $F_a$, samt tyngdekraften, $F_g$, i forskellige situationer. $-F_a$ er påvirkningen af fjederen, som følge af en reel acceleration i den modsatte retning. Fra venstre mod højre: (**a**) Aksen påvirkes kun af tyngdekraften, hvilket resulterer i en positiv målt acceleration på $1g$. (**b**) Aksen i frit fald langs tyngdekraftens akse, hvorfor påvirkningen $-F_a$ fra den reelle acceleration udligner tyngdekraftens påvirkning, og den målte acceleration er nul. (**c**) Aksen påvirkes af tyngdekraften og en reel acceleration i samme retning som tyngdekraften, hvorfor påvirkningen af den reelle acceleration dominerer, og resultatet er en negativ målt acceleration $F_g - F_a$.

Tyngdekraften spiller ind som en konstant ekstern påvirkning: Uanset orienteringen af accelerometeret vil dets akser nemlig opleve en kraft på $1g$ i tyngdekraftens retning, og udfaldet vil derfor være, at „fjedrene“ udvides eller komprimeres i tyngdekraftens retning. Eftersom en almindelig acceleration i en given retning forårsager, at fjedrene udvider/komprimerer sig i den stik modsatte retning, da vil tyngdekraftens påvirkning registrere som en acceleration i stik modsat retning af selve påvirkningen.

Accelerometeret gør ikke brug af fjedre, men af mere avancerede teknikker [Analog Devices, 2006]. Princippet er dog det samme, og accelerationen aflæses derfor for alle tre akser med en konstant påvirkning fra tyngdekraften, på $1g$ i modsat retning af tyngdekraften. Den målte acceleration er derfor en sum

$$\vec{a}_{raw} = \vec{g} + \vec{a} + \vec{c} \tag{1}$$

hvor $\vec{g}$ er accelerationen som følge af tyngdekraften, $\vec{a}$ er accelerationen som følge af ændring i hastighed – den egentlige acceleration, og $\vec{c}$ er støj som følge af analog til digital konvertering.

Det ses at det er nødvendigt at tage højde for $\vec{g}$, såfremt man ønsker at finde $\vec{a}$, og omvendt. For at reducere effekten af støjen $\vec{c}$ anvendes typisk en udglattende filtrering på de aflæste data, eksempelvis et Gauss- eller eksponentialfilter.

### 2.2.2 Protokol

Wii-controlleren genkendes af en computer som en HID-enhed. HID enheder har en intern protokol, som kan benyttes til, at kommunikere med over forbindelsen til en computer. Protokollen beskriver en række virtuelle porte, som i stor grad fungerer på samme måde som en fysisk port i et netværk, men angivelsen af en port i en data-pakke bliver *parset* i software og der handles ud fra denne parsning.

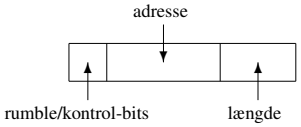
Specifikt for Wii-controlleren bliver denne protokol benyttet således, at man kan sende forskellige *requests* til forskellige porte for, at tænde og slukke LED eller rumble, sætte controlleren i forskellige modes, så den sender bestemte typer data til computeren, etc. De forskellige typer data controlleren kan sende til computeren sendes ligeledes til forskellige porte, så man ved hvordan den modtagne data skal parses. Tabel 1, side 6 viser Wii-controllerens protokol.

Det er, som tidligere nævnt, muligt at vælge forskellige modes og derved pakkeformater for data der sendes fra Wii-controlleren. De forskellige modes er egnede til forskellige situationer og man kan skifte mellem dem *on-the-fly*. Hver gang Wii-controlleren tændes sættes denne i en mode med rapport ID `0x30` uden kontinuerlig dataoverførsel, så det kun er information om knapperne der sendes til computeren og når der sker en ændring af disses tilstand.

Wii-controlleren sender pakker til computeren med $10ms$ interval, hvilket svarer til 100 pakker pr. sekund. Dette gør sig selvfølgelig gældende ved kontinuerlig dataoverførsel, men kan også forekomme uden kontinuerlig overførsel, hvis der meget ofte sker ændringer i controllerens tilstand.

### 2.2.3 Kalibrering

Når Wii-controlleren tændes skal den kalibreres for, at kompensere for usikkerheder der er opstået i produktionen af controlleren og accelerometeret. Eksempelvis kan det være tilfælde at akserne i accelerometeret ikke er helt ortogonale, eller at påvirkningen registreres en anelse forskelligt fra akse til akse, hvorfor kalibrering er nødvendigt. For at kalibrere controlleren skal der sendes en pakke til port `0x17` med en længde på 6 bytes.




**Figur 3:** Strukturen af pakken, der sendes til Wii-controlleren ved forespørgsel om kalibreringsdata.

Den første byte (se Figur 3) indeholder information om rumble-status og kontrol-

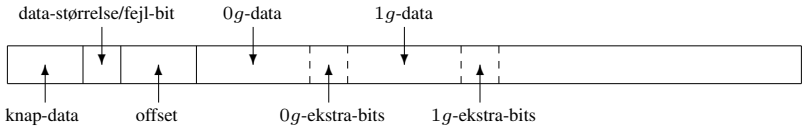
| **Output** | | | |
|---|---|---|---|
| **Port** | **Bytes** | **Funktion** | **Pakkeformat** |
| `0x11` | 1 | LED, rumble | `1L` |
| `0x12` | 2 | rapport type | `1T 1M` |
| `0x13` | 1 | IR sensor til | |
| `0x14` | 1 | højttaler til | |
| `0x15` | 1 | status | |
| `0x16` | 21 | skriv data | `1M 3F 1S 16D` |
| `0x17` | 6 | læs data | `1M 3F 2S` |
| `0x18` | 21 | højttaler data | |
| `0x19` | 1 | højttaler fra | |
| `0x1a` | 1 | IR sensor til | |

| **Input** | | | |
|---|---|---|---|
| **Port/ rapport ID** | **Bytes** | **Funktion** | **Pakkeformat** |
| `0x20` | 6 | udvidelsesport | `2N 1S 2N 1P` |
| `0x21` | 21 | læs data | `2B 1S 2F 16D` |
| `0x22` | 4 | *acknowledgment* for skrivning af data | |
| `0x30` | 2 | knapper | `2B` |
| `0x31` | 5 | knapper og motion-data | `2B 3A` |
| `0x32` | 16 | knapper, udvidelse og muligvis IR-data | `2B 8E` |
| `0x33` | 17 | knapper og motion-data | `2B 3A 12I` |
| `0x34` | 21 | knapper, udvidelse og muligvis IR-data | `2B 19E` |
| `0x35` | 21 | knapper, motion-data og udvidelse | `2B 3A 16E` |
| `0x36` | 21 | knapper, udvidelse og muligvis IR-data | `2B 10I 9E` |
| `0x37` | 21 | knapper, motion-data og udvidelse | `2B 3A 10I 6E` |
| `0x38` | 21 | knapper, motion-data og IR-data | `2B 3A 16I` |
| `0x3d` | 21 | knapper, udvidelse og muligvis IR-data | `21E` |
| `0x3e` | 21 | knapper, motion-data og evt. IR-data | `2B 1A 18I` |
| `0x3f` | 21 | knapper, motion-data og evt. IR-data | `2B 1A 18I` |

**Tabel 1:** Pakkeformatet for Wii-controllerens interne HID-protokol. Output er data sendt fra computeren til controlleren og intput er data sendt fra controlleren til computeren. Pakkeformaterne er angivet således at `A` angiver bytes til accelerationsdata, `B` angiver bytes til knapdata, `D` er databytes, `E` er bytes til udvidelsesdata, `I` er bytes til IR-data, `L` er bytes til LED-data, `M` er bytes der angiver rapport-mode, `F` er bytes der angiver offset ved læsning fra og skrivning til EEPROMen, `P` er bytes der angiver batteriniveau, `S` er bytes der angiver status for controlleren og `T` er bytes der angiver om data skal sendes kontinuerligt. De datapakker der ikke er angivet benyttes typisk til at slå features, som højttaler og IR-sensor, til og fra. *Bemærk at de fremstillede data bygger på reverse engineering og trial-and-error, så de kan ikke garanteres at være korrekte*. *Kilde: [WiiLi, 2007].*

bits, der angiver, hvorvidt der skal læses fra den indbyggede EEPROM eller fra kontrolregistrene. De næste 3 bytes indeholder adressen der skal læses fra i EEPROMen. Kalibreringsdata ligger på adresse `0x16` i EEPROMen. De sidste 2 bytes indeholder længden af den data der skal læses. Kalibreringsdata fylder 8 bytes; 3 bytes til hhv. x-, y- og z-aksernes værdi ved $0g$, 1 byte som bl.a. indeholder aksernes 9. og 10. bit ved $0g$, 3 bytes til hhv. x-, y- og z-aksernes værdi ved $1g$ og til sidst 1 byte som bl.a. indeholder aksernes 9. og 10. bit ved $1g$.

Wii-controlleren sender kalibreringsdata tilbage til computeren på port `0x21` i en pakke der indeholder knap-data, 1 fejl-bit, *offset* for data returneret i pakken samt den læste kalibreringsdata (se Figur 4).

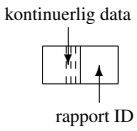



**Figur 4:** Strukturen af pakken med kalibreringsdata, der modtages fra Wii-controlleren.

Den modtagne data beskriver afvigelsen af den målte acceleration i forhold til den reelle acceleration og kan benyttes til at opnå et mere præcist resultat af målingerne. Accelerationen fra controlleren angives som et tal i intervallet $[-512; 511]$ og kan normaliseres ved at dele den målte acceleration med differensen mellem den kalibrerede acceleration ved $1g$ og $0g$.

### 2.2.4 Aflæsning af knapper

For at aflæse knap-data fra Wii-controlleren skal denne sættes i en af de modes der sender knap-data til computeren, hvilket kan være en hvilkårlig med rappport ID fra `0x30` til `0x3f` (se Tabel 1). Dette kan gøres ved at sende en pakke til port `0x12` på Wii-controlleren. Denne pakke består af 2 bytes (se Figur 5), hvor den første angiver om der kontinuerligt skal sendes data til computeren og anden byte indeholder rapport ID'et controlleren skal benytte. Hvis man f.eks. vil have controlleren til at sende både knap-data og motion-data skal anden byte sættes til `0x31`. Når der sendes data til Wii-controlleren skal status for rumble altid inkluderes, som den første bit af den første byte der sendes til controlleren.




**Figur 5:** Strukturen af pakken der sendes til Wii-controlleren for at ændre rapport ID.

Når der modtages nye knap-data vil de, alt efter hvilken mode Wii-controlleren er sat til, komme ind på en port mellem `0x30` og `0x3f`. I alle tilfælde består knap-data af 2 bytes.

For at finde ud af hvilke knapper der er trykket på skal man kende de unikke koder, eller *bitmasks*, de forskellige knapper har (se Tabel 2). Man kan kontrollere om en knap er trykket ned ved at kontrollere om dennes bitmask er repræsenteret i de modtagne knap-data.

| Knap | Bitmask |
|---|---|
| 2 | `0x0001` |
| 1 | `0x0002` |
| B | `0x0004` |
| A | `0x0008` |
| - | `0x0010` |
| Hjem | `0x0080` |
| Venstre | `0x0100` |
| Højre | `0x0200` |
| Ned | `0x0400` |
| Op | `0x0800` |
| + | `0x1000` |

**Tabel 2:** Bitmasks for Wii-controllerens knapper. *Kilde: [WiiBrew, 2007].*

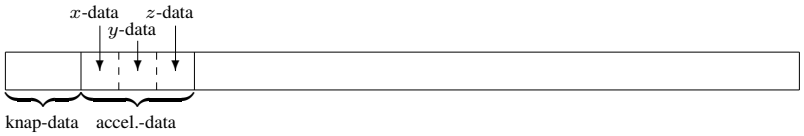
### 2.2.5 Aflæsning af accelerometer

Ved aflæsning af accelerometerdata er der, som udgangspunkt, 8 bits for hver akse, men ifølge [WiiLi, 2007] er der, i bitmasken for knapperne, gemt en 9. bit, for alle tre akser samt en 10. for x-aksen. Således kan de ekstra bits udledes af knap-dataen ved at benytte bitmasks i Tabel 3. De i tabellen angivne bitmasks kan udledes, såfremt de to bytes indeholdende knap-data sammensættes således, at den første byte udgør de mest betydende bits, og den anden byte udgør de mindst betydende bits. De ekstra bits udgør de mindst betydende bits af accelerometer-dataen for akserne.

| Akse | Bitmask |
|---|---|
| X | `0x4000` |
| X (10. bit) | `0x2000` |
| Y | `0x0020` |
| Z | `0x0040` |

**Tabel 3:** Bitmasks i knap-dataen som udgør de ekstra bits af accelerometer-dataen. *Kilde: [WiiBrew, 2007].*

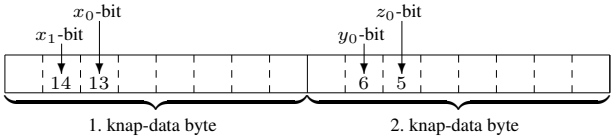
For at modtage accelerometer-data fra Wii-controlleren skal denne sættes i en af de modes der sende denne data med til computeren. Dette kunne f.eks. være mode med rapport ID `0x31`, som både sender knap- og accelerometer-data (se Figur 6). Disse data vil blive modtaget på port `0x31`.




**Figur 6:** Strukturen af pakken med knap- og accelerometer-data der modtages fra Wii-controlleren.

Da denne pakke indeholder både knap- og accelerometer-data er det muligt at udlede accelerationen i akserne med de ekstra bits præcision. Ud fra de informationer vi kender om de ekstra accelerometer-data-bits i knap-data kan vi sammensætte mere

præcise accelerometer-data og derved få en større præcision. Figur 7 viser placeringen af de ekstra bits, hvis knap-dataen betragtes som tidligere beskrevet, hvor den første byte udgør de mest betydende bits og den sidste udgør de mindst betydende.



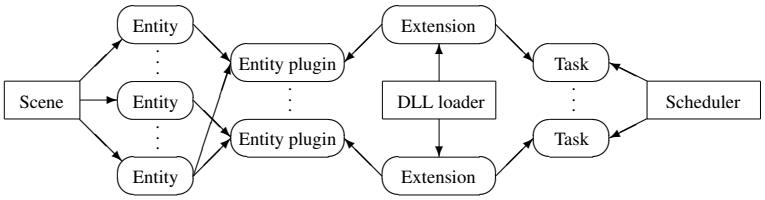

**Figur 7:** Placeringen af de ekstra bits i knap-data, for accelerometer-data. Numrene i felterne angiver den enkelte bits offset i den samlede 2-byte data.

Som det ses af figuren ligger de ekstra bits for x-, y- og z-aksen hhv. i bit 14/13, 6 og 5. Denne information kan bruges til, at udtrække de ekstra bits og efterfølgende tilføje dem til accelerations-data.

## 2.3 3DOT

3DOT er en *open source* 3D-motor der er udviklet[2] på DIKU til undervisningsbrug på bl.a. DIKU og Aalborg Universitet. Kildekoden er tilgængelig for alle, og kan hentes fra den officielle hjemmeside [3DOT, 2007] hvor der også findes en gennemgang af opsætning af udviklingsmiljøet.

3DOT implementerer et solidt udvidelses-*interface*, der gør det muligt at udvide 3D-motoren med næsten hvad som helst. Netop muligheden for at konstruere egne udvidelser (*extensions*) til 3D-motoren gør det interessant at udbyde en Wii-controller-udvidelse, således at denne kan inddrages i andre 3DOT-projekter, eksempelvis spil.




**Figur 8:** Den interne struktur af 3DOT viser hvordan forskellige entities kan benytte samme udvidelse.

### 2.3.1 Datastruktur

En *scene* i 3DOT er en slags container, indeholdende en liste af *entities* tilknyttet scenen. En entity i 3DOT er et navngivet objekt men uden form eller andre egenskaber. Det er muligt at tilføje egenskaber til en entity igennem et *entity plugin*, som udbydes af en udvidelse (se Figur 8).

Kernen af 3DOT indeholder desuden kode til serialisering af et projekt og derved hele scener. Ved serialisering af en scene pakkes al information omkring scenen, dens

[2]P.t. er 3DOT stadig under udvikling, men er i et stadie hvor den har været tilstrækkeligt færdig til at blive brugt i DADIU's [DADIU, 2007] spilprojekt, marts 2007.

status og dens entities ned som XML, således at scenen senere kan deseraliseres (rekonstrueres).

### 2.3.2 Udvidelser

Udvidelsessystemet i 3DOT er nok en af 3D-motorens vigtigste features. Faktisk kan 3DOT ikke tegne noget på skærmen uden udvidelsessystemet, da selve OpenGL-*rendereren* også er en udvidelse. Ligeledes håndteres input fra tastatur af en udvidelse, og det samme gælder for inline LUA-scripts, der også afvikles af en udvidelse.

En udvidelse til 3DOT består af flere forskellige komponenter, der alle kompileres til én DLL-fil. Den grundlæggende komponent i enhver udvidelse er selve extension-klassen, som indeholder de funktioner der kaldes når udvidelsen indlæses, samt funktioner til udbydelse af entity plugins, og andet. En mere detaljeret gennemgang af udvidelsessystemet findes på den officielle hjemmeside.

### 2.3.3 Scheduler

Kernen af 3DOT indeholder en central *scheduler*, som sørger for prioriteret sekventiel afvikling af alle registrerede begivenheder (*tasks*). Eksempelvis tegnes billedet af en scene af OpenGL-renderer-udvidelsen som følge af, at udvidelsen har registreret en begivenhed, der ved afvikling tegner scenen. Dette betyder også at det er schedulerens arbejdsbyrde der bestemmer *frameraten* og intervallet hvormed input kan læses fra f.eks. et tastatur.

Scheduleren er en vigtig komponent i 3DOT, da man ellers ville være nødsaget til at gøre brug af seperate tråde til dataopsamling og tilrettelagt afvikling af forskellige begivenheder på kryds af udvidelser. Der er ikke nogen forhindring for at benytte egne tråde i 3DOT, men det er nemmest at gøre brug af scheduleren.

# 3 Anvendelse af Wii-controller

Dette afsnit beskriver hvilke af Wii-controllerens funktioner vi vil anvende, og hvorledes vi vil anvende dem. Vores konkrete mål er at kunne benytte selve controlleren, uden udvidelser, til at rotere og flytte objekter i rum, og vi vil derfor koncentrere os om de af controllerens funktioner som er relevante i forhold til dette mål.

Vi vil derfor ikke kigge på anvendelsen af controllerens udvidelsesmoduler [Nintendo, 2007] eller funktionalitet, som den indbyggede højtaler, den indbyggede rumble-motor eller det indbyggede infrarøde kamera (herefter IR-kamera) stiller til rådighed. Generelt vil vi ikke komme nærmere ind på funktionalitet, der ikke udbyder nogen form for data, med undtagelse af lysdioderne hvis status er anvendelige i tests.

Aflæsning af IR-kameraet's data herunder koordinater på eventuelle IR-lyskilder der ligger indenfor kameraets synsfelt, er dog ganske interessant i sig selv. Disse data kan eksempelvis benyttes til direkte at styre en markør på skærmen eller til at bestemme afstanden fra kameraet til et eller flere par af lyskilder. Desværre er data fra kameraet også så afhængige af eksterne forhold (antal af lyskilder, placering af lyskilder, orientering af kamera i forhold til lyskilder, osv.), at tilgængeligheden af en eventuel løsning ville mindskes som følge af store krav til disse forhold. Vi ønsker imidlertid en løsning som er let tilgængelig, og vi fravælger derfor også IR-kameraet.

Tilbage har vi knapperne og lysdioderne på controlleren, samt det indbyggede accelerometer. Vores implementation af knapperne vil dække alle knapper med undtagelse af power-knappen, eftersom der endnu ikke eksisterer offentlig dokumentation til aflæsning af dennes status. Vi vil desuden implementere muligheden for at sætte lysdiodernes tilstand, således at vi ved at sætte forskellige tilstande kan teste direkte om controlleren rent faktisk modtager eventuelle data vi sender til den.

## 3.1 Anvendelse af knapper

Da Wii-controlleren understøtter to forskellige måder at udlæse data på, kontinuerligt eller når der sker ændringer i knappers tilstand (eller accelerometerets; hvilket næsten er det samme som kontinuerligt), vil der være mulighed for at implementere registreringen af knap-tryk vha. en *push*-metode, som gør det unødvendigt at kontrollere med fast interval om knappernes tilstand er ændret.

## 3.2 Anvendelse af accelerometer

Accelerometeret i controlleren rapporterer som nævnt dets egen (og derfor også controllerens) acceleration i dets 3 akser. Ud fra accelerationen af controlleren vil vi forsøge, at estimere både controllerens orientering og position, således at vi kan anvende disse data til at rotere og flytte på objekter i et virtuelt rum.

### 3.2.1 Bestemmelse af orientering

Hvis vi kan finde controllerens orientering kan vi anvende resultatet direkte i et 3D-miljø, eksempelvis til at vippe et plan, eller til at rotere et objekt.

Muligheden for at bestemme en orientering ud fra en acceleration kommer af, at accelerometeret i controlleren er under konstant påvirkning af tyngdekraften. Ved stil-

stand vil accelerometeret som nævnt i afsnit 2.2.1 på side 4, rapportere en konstant acceleration i modsat retning af tyngdekraften, og denne acceleration kan omdannes til en delvis beskrivelse af controllerens orientering. Accelerationen ved stilstand angiver med andre ord accelerometerets hældning i forhold til tyngdekraften.

Typisk vil controlleren dog ikke være i total stilstand, og det er derfor nødvendigt at estimere accelerationen som følge af tyngdekraften ud fra den målte acceleration. Hvis vi betragter al overskydende acceleration (den der ikke kommer fra tyngdekraften) som støj, da afhænger resultatet af hvor godt vi kan fjerne denne støj fra den målte acceleration.

Controllerens orientering kan beskrives ved begreberne *roll*, som er rotation om Wii-controllerens y-akse, når denne holdes vandret; *pitch*, som er rotation om Wii-controllerens x-akse, når denne holdes vandret; og *yaw*, som er rotation om Wii-controllerens z-akse, når denne holdes vandret; se Figur 9.

**Figur 9:** Bevægelse af Wii-controlleren svarende til hhv. roll, pitch og yaw. *3D Studio Max model af [Grubb, 2007].*

Vi kan som nævnt kun delvist bestemme controllerens orientering ud fra accelerationen: Eftersom yaw er rotationen omkring den lodrette akse kan vi ikke finde yaw ud fra en lodret acceleration, og da accelerationen ved stilstand er lodret i forhold til tyngdekraften, så er det ikke muligt at beregne yaw givet denne. Med andre ord, så ændrer det ikke tyngdekraftens påvirkning af controlleren, at vi roterer controlleren i yaw. Typisk vil man derfor i mere avancerede systemer benytte et gyroskop til at estimere yaw, som supplement til accelerometeret.

Vi kalder den delvise orientering af controlleren (roll, pitch) for controllerens $tilt$.

### 3.2.2 Bestemmelse af position

Vi ønsker desuden at bestemme controllerens position ud fra dens acceleration. Hvis vi kan finde controllerens relative position kan vi anvende resultatet direkte i et 3D-miljø, eksempelvis til at styre et objekt, med direkte forbindelse til den fysiske bevægelse, som tilføres controlleren.

Grundlæggende er en acceleration et udtryk for en ændring i hastighed over tid, ligesom en hastighed er et udtryk for en ændring i position over tid. Man kan finde en acceleration ud fra en position på et givent tidspunkt, ved først at finde hastigheden; den afledte af positionen, for derefter at finde accelerationen; den afledte af hastigheden. På samme vis kan man også finde en position ud fra en acceleration på et givent tidspunkt, ved først at integrere accelerationen; den afledte af hastigheden, for derefter at integrere hastigheden; den afledte af positionen. Resultatet er positionen.

Accelerationen udbydes af controlleren som et digitalt signal, og vi er derfor nødt

til at arbejde med numerisk integration, hvilket kan medføre at resultatet kommer til at afvige over tid. Hvor godt et resultat, der opnås, afhænger blandt andet af hvor god en metode der benyttes til numerisk integration, men også af hvor godt vi kan fjerne eventuel støj fra den målte acceleration.

En meget væsentlig støjkilde i forhold til udledning af controllerens position er hvis den også ændrer orientering mens den flyttes: En ændring i orientering vil medføre en ændring i accelerationen som følge af tyngdekraftens påvirkning, og medmindre vi kender denne ændring kan det derfor være svært, at sige noget fornuftigt om hvilken del af accelerationen der tilhører ændring i position, og hvilken del der tilhører ændring i orientering.

### 3.2.3 Samtidig bestemmelse af orientering og position

Det er tydeligt at metoderne til at bestemme orientering, og til at bestemme position, afhænger af hinandens resultater. En af de store udfordringer er derfor at finde en balance mellem de to metoder, eller med andre ord en fælles metode der tillader at estimere både orientering og position på samme tid, sådan at det ene ikke udelukker det andet.

Vores strategi i den forbindelse er at starte med at estimere tilt (den delvise orientering), for derefter at anvende resultatet derfra i estimering af position. Grunden er, at vi ikke kan fravælge accelerationen som følge af tyngdekraften, eftersom controlleren altid påvirkes af tyngdekraften, hvorimod vi godt kan fravælge accelerationen som følge af ændring i hastighed (ved at undlade at bevæge controlleren langs accelerometerets akser). Med andre ord afhænger estimering af position mere af estimering af tilt end omvendt, og det må derfor være nemmere at isolere dele af problemet omkring estimering af tilt. Vores analyse af det samlede problem følger derfor også denne strategi, hvorfor vi først behandler estimering af tilt, og derefter estimering af position.

### 3.2.4 Andre anvendelser af accelerationen

Udover orientering og position kan accelerationen også anvendes til andre ting. Vi vil kun benytte accelerationen til estimering af orientering og position i dette projekt, men det er eksempelvis værd at nævne, at accelerationen fra Wii-controlleren ofte bruges i spil til 3D-gesturegenkendelse, se eksempelvis [AiLive, 2007]. Se Bilag B, side 66 for en yderligere gennemgang af mulighederne for gestures.

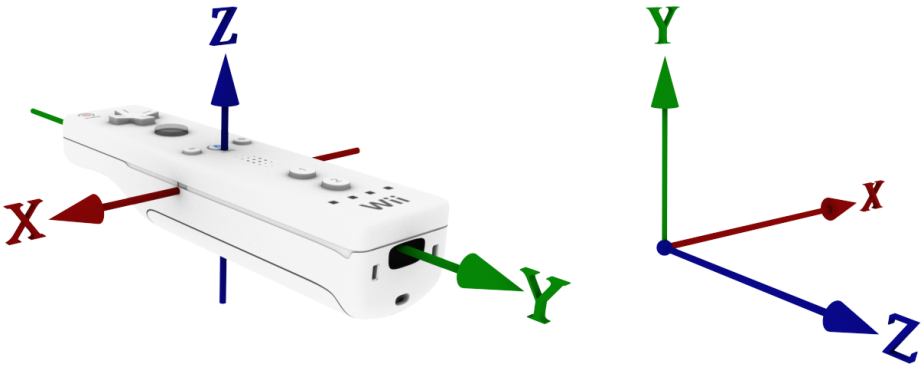
# 4 Analyse

Dette afsnit beskriver hvilke overvejelser vi har gjort os i forbindelse med, at implementere de ønskede anvendelser af Wii-controlleren. Afsnittet tager således udgangspunkt i Wii-controllerens tekniske egenskaber, som beskrevet i afsnit 2 på side 2, og hvorledes vi vil anvende disse, som beskrevet i afsnit 3 på side 11.

Specielt behandles problemerne at estimere Wii-controllerens tilt (dens delvise orientering), samt dennes position. Først vil vi dog kort redegøre for vores opfattelse af orienteringen af akserne i Wii-controllerens koordinatsystem, eller med andre ord konvertering af de målte data således, at de, i forhold til hvordan Wii-controlleren typisk anvendes, passer bedre ind med akserne i det naturlige koordinatsystem.

## 4.1 Konvertering mellem koordinatsystemer

Den naturlige måde at holde Wii-controlleren på er som et pegeredskab: Vandret med knapperne opad, og væk fra kroppen. Vi kalder denne orientering for controllerens naturlige orientering. Accelerometeret i controlleren er konfigureret således, at dets akser ved naturlig orientering af controlleren ikke stemmer overens med akserne i det almindelige højrehåndede koordinatsystem (se Figur 10), som benyttes i OpenGL m.fl.




**Figur 10:** Accelerometerets akser ved naturlig orientering af controlleren, set i forhold til det almindelige højrehåndede koordinatsystem. *3D Studio Max model af [Grubb, 2007].*

Den målte acceleration (1) udbydes af Wii-controlleren med hensyn til den naturlige basis $E$, hvor $E$ består af akserne i det almindelige højrehåndede koordinatsystem. Som inddata til beregning af tilt og position ønsker vi, at anvende den målte acceleration med hensyn til controllerens naturlige orientering, således at resultaterne af vores beregninger stemmer overens med det miljø, som de skal anvendes i. Eksempelvis bør en retning med hensyn til controllerens naturlige orientering svare til en retning med hensyn til den naturlige basis $E$.

Med andre ord vil vi gerne afbilde den målte acceleration således, at den repræsenteres med hensyn til en basis $C$, hvor $C$ er givet ved konfigurationen af akserne i controllerens naturlige orientering med hensyn til den naturlige basis $E$. $C$ kan aflæses direkte ud fra Figur 10 som en $3 \times 3$ matrix

$$C = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

At finde den målte acceleration med hensyn til basen $C$ (dvs. med hensyn til den naturlige orientering) er herefter et spørgsmål om at foretage basisskift. Givet en vektor med hensyn til basen $C$, $\vec{v}_c$, kan vi finde en vektor med hensyn til den naturlige basis $E$, $\vec{v}_e$, ved at gange $C$ på $\vec{v}_c$. Omvendt kan vi finde en vektor med hensyn til basen $C$ ved at $C^{-1}$ på $\vec{v}_e$:

$$\vec{v}_e = C\vec{v}_c \Leftrightarrow \vec{v}_c = C^{-1}\vec{v}_e.$$

I dette tilfælde er $C$ både invertibel og symmetrisk over diagonalen, hvorfor $C^{-1} = C^T = C$. Den målte acceleration med hensyn til controllerens naturlige orientering er da givet ved

$$\vec{a}_{input} = C\vec{a}_{raw} \tag{2}$$

hvor $\vec{a}_{raw}$ er den målte acceleration med hensyn til den naturlige basis.

Når vi i efterfølgende afsnit refererer til $\vec{a}_{input}$ mener vi derfor den målte acceleration med hensyn til den naturlige orientering af controlleren.

## 4.2 Estimering af tilt

Wii-controllerens tilt er den delvise orientering af controlleren i forhold til en given lodret akse, hvilket typisk vil være tyngdekraftens akse. Man kan således også sige, at controllerens tilt er den relative orientering af controlleren i forhold til den givne akse.

Ved den relative orientering af et objekt forstås den rotation, som beskriver forskellen mellem en prædefineret orientering af objektet, „reference-orienteringen“, og objektets nuværende orientering. Med andre ord er et objekts relative orientering netop den rotation som, anvendt på reference-orienteringen, resulterer i objektets nuværende orientering.

I dette afsnit redegør vi for hvordan man, givet en reference-orientering, kan estimere Wii-controllerens tilt ud fra den målte acceleration.

Førend vi kan anvende et estimat af Wii-controllerens tilt til operationer i rum, er vi dog nødt til at udlede sammenhængen mellem den delvise orientering af et objekt; objektets tilt, og den fuldstændige orientering af et objekt. Først i dette afsnit diskuteres derfor forskellige repræsentationer af rotation og orientering i rum, hvorefter vi anvender viden fra disse afsnit i løsningen af problemet, at estimere controllerens tilt.

Vi har delt problemet op således, at vi først analyserer og behandler estimering af tilt givet ideelle inddata (det ideelle tilfælde), hvorefter vi analyserer og behandler problemet, at estimere tilt givet vilkårlige inddata (det generelle tilfælde).

### 4.2.1 Rotation i $\mathbf{R}^3$

**Komposit rotation** Den nok simpleste repræsentation (at konstruere) af en rotation gør brug af en rotationsmatrix per akse: Givet tre vinkler $\theta_x$, $\theta_y$, og $\theta_z$ (en vinkel per

akse) konstrueres tre $3 \times 3$ rotationsmatricer $R_x$, $R_y$, $R_z$. Den kompositte rotation over alle tre akser er da eksempelvis givet ved matricen

$$R = R_x R_y R_z.$$

Metoden har først og fremmest den ulempe, at den ikke er kommutativ, dvs. $R_x R_y R_z \neq R_x R_z R_y$. Der ingen streng standard for rækkefølgen af $R_x$, $R_y$, $R_z$ i den kompositte rotation $R$, hvorfor denne kan variere fra implementation til implementation. Derudover kan det være svært at forudsige hvor et roteret punkt „ender“, eftersom rotationen sker relativt per akse.

Der findes dog en speciel form for komposit rotation, som er bedre end den naive. Vinklerne for denne metode kaldes *Euler-vinkler* og beskrives ofte ved tre symboler; $\phi$, $\theta$, og $\psi$, og tre tilhørende rotationsmatricer; $R_\phi$, $R_\theta$, og $R_\psi$. Euler-vinklerne og de tilhørende rotationsmatricer beskriver ikke rotationen over de originale akser, som den naive kompositte rotation, men rotationen om to af de relative akser, som opstår i løbet af den kompositte rotation. Hvis $R = R_\psi R_\theta R_\phi$, da vil matricerne kunne tolkes som at:

- $R_\phi$ beskriver den initielle rotation omkring z-aksen
- $R_\theta$ beskriver rotation omkring x-aksen efter anvendelse af $R_\phi$
- $R_\psi$ beskriver rotation omkring z-aksen efter anvendelse af $R_\theta$

Desværre er der flere konventioner for, hvordan Euler-vinkler skal tolkes; specielt hvad angår hvilke vinkler der svarer til hvilke akser, og rækkefølgen af disse. Vi har imidlertid brug for en entydig repræsentation, hvorfor vi ikke vil benytte nogen af disse metoder, men i stedet en metode der tillader rotation om en arbitrær akse.

**Arbitrær rotation** En mere direkte beskrivelse af en rotation er *axis-angle* repræsentationen: Givet en vektor $\vec{\omega} \in \mathbf{R}^3$ og en vinkel $\theta$, hvor $|\vec{\omega}| = 1$ samt $-\pi \leq \theta \leq \pi$, kan vi præcist (og entydigt) beskrive en rotation om en arbitrær akse i $\mathbf{R}^3$ ved

$$\{(\vec{\omega}, \theta) : |\vec{\omega}| = 1, -\pi \leq \theta \leq \pi\}.$$

Axis-angle representationen er generelt velset da den, udover at være entydig og forholdsvis let at forestille sig, også kan anvendes direkte som inddata til formler til konstruktion af rotationsmatricer eller quaternioner. I 3D-motorer er et objekts orientering ofte repræsenteret ved enten en matrix eller en quaternion, og for at anvende en rotation på axis-angle form vil man derfor typisk have brug for, at udtrykke den ved enten en matrix eller en quaternion.

Formlen for konstruktion af en rotationsmatrix givet en rotation $(\vec{\omega}, \theta)$, er givet ved $3 \times 3$ matricen

$$R(\vec{\omega}, \theta) =$$

$$\begin{pmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)yx - (\sin\theta)z & (1-\cos\theta)zx + (\sin\theta)y \\ (1-\cos\theta)xy + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)zy + (\sin\theta)x \\ (1-\cos\theta)xz + (\sin\theta)y & (1-\cos\theta)yz + (\sin\theta)x & \cos\theta + (1-\cos\theta)z^2 \end{pmatrix}.$$

Værdierne der indgår i axis-angle repræsentationen er tæt relateret til vektor-repræsentationen af quaternioner, hvilket også ses ud fra formlen
[Weisstein, 2007, Baker, 2007] for konstruktion af en quaternion $\vec{q}$ givet en rotation $(\vec{\omega}, \theta)$:

$$\vec{q}(\vec{\omega}, \theta) = \begin{pmatrix} \cos(\theta/2) \\ \vec{\omega}_1 \cdot sin(\theta/2) \\ \vec{\omega}_2 \cdot sin(\theta/2) \\ \vec{\omega}_3 \cdot sin(\theta/2) \end{pmatrix}.$$

Vi vil herefter kun beskæftige os med axis-angle repræsentationen og matricer, og kun overfladisk/indirekte gøre brug af quaternioner. Eksempelvis repræsenteres et objekts orientering i 3DOT ved en quaternion, og her instantieres implementationen[3] netop med $(\vec{\omega}, \theta)$ som inddata.

### 4.2.2 Orientering i $\mathbf{R}^3$

**Orientering af objekt** Et objekts orientering i $\mathbf{R}^3$ kan beskrives ved hjælp af tre ortogonale vektorer: En vektor $\vec{y}_{axis}$ der peger i retning af objektets top, dvs. y-aksen i objektets lokale koordinatsystem med hensyn til den naturlige basis, og en vektor $\vec{z}_{axis}$ der peger i retning af objektets front. Den tredje aksevektor $\vec{x}_{axis}$ er mindre relevant, da den kan udledes direkte ved krydsproduktet $\vec{x}_{axis} = \vec{y}_{axis} \times \vec{z}_{axis}$. Tilsammen danner de en basis i $\mathbf{R}^3$, der også kan skrives som en $3 \times 3$ matrix

$$B = \begin{pmatrix} \vec{x}_{axis} & \vec{y}_{axis} & \vec{z}_{axis} \end{pmatrix}.$$

En quaternion kan ligeledes repræsentere et objekts orientering, og eksempelvis 3DOT gør brug af denne type repræsentation.

Vi nævnte i forrige afsnit, at har man en rotation på axis-angle form, da kan man konstruere enten en matrix eller en quaternion ud fra denne. Muligheden for at omsætte axis-angle form til begge dele har den umiddelbare konsekvens, at uanset om man repræsenterer orienteringen af et objekt ved en matrix eller ved en quaternion, da kan man frit rotere denne orientering om en arbitrær akse.

Givet matrix-repræsentationen af en rotation $R$, og en orientering $B$, kan vi udlede rotationen af en orientering ved

$$B' = RB$$

hvor $B'$ er den roterede orientering. Hvis $B$ er invertibel ses desuden at

$$B'B^{-1} = R.$$

Med andre ord kan vi udlede matrix-repræsentationen af en rotation fra en orientering $B$ til en anden orientering $B'$, hvilket kan være praktisk hvis vi har to orienteringer, og gerne vil finde rotationen mellem disse to. Det ses desuden at $B = E \Rightarrow B' = R$ såfremt $E$ er den naturlige basis.

Lidt mere besværligt er det at finde frem til axis-angle formen af den fundne rotation, og det er ikke noget vi vil beskæftige os med her. [Baker, 2007] har dog en udmærket gennemgang af metoden.

[3] CORE_3DOT::quat

**Orientering af plan** Orienteringen af et plan er givet ved en normalvektor $\vec{n}$, også kaldet planets normal. Vi siger at et plan er orienteret vandret, og med dets positive side opad, hvis dets normalvektor er lig y-aksen i den naturlige basis $(0, 1, 0)$.

Givet en matrix-repræsentationen af en rotation $R$, og et plan givet ved dets normal $\vec{n}$, kan vi udlede rotationen af planet ved

$$\vec{n}' = R\vec{n}$$

hvor $\vec{n}'$ er normalen til det roterede plan. For at udlede rotationen fra $\vec{n}$ til $\vec{n}'$ husker vi at matricen $R$ kan beregnes ud fra axis-angle formen $(\vec{\omega}, \theta)$. Vi kan derfor nøjes med at udlede $\vec{\omega}$ og $\theta$ mellem de to normalvektorer $\vec{n}$ og $\vec{n}'$.

Krydsproduktet af to vektorer er en ny vektor der står vinkelret på begge vektorer. Krydsproduktet af to normaler vil derfor ligge i begge normalers planer, hvilket kun er muligt langs skæringen af de to planer. Vektorer langs skæringen af to planer kan betragtes som akser om hvilke de to planer (og derved også deres normaler) er roteret i forhold til hinanden. Rotationsaksen $\vec{\omega}$ er da givet ved

$$\vec{\omega} = \|\vec{n} \times \vec{n}'\|.$$

Bemærk at krydsproduktet normaliseres eftersom axis-angle repræsentationen forudsætter $|\vec{\omega}| = 1$, samt at rækkefølgen $\vec{n} \times \vec{n}'$ giver os aksen for rotationen fra $\vec{n}$ til $\vec{n}'$. Bemærk også at $\vec{\omega}$ ikke helt er entydig, eftersom krydsproduktet resulterer i nulvektoren både når den mindste vinkel mellem de to vektorer er $0$ og når den er $\pi$ (da vektorerne er parallele i begge tilfælde).

Formlen for den mindste vinkel mellem to vektorer giver os rotationsvinklen $\theta$

$$\theta = \cos^{-1}\left(\frac{\vec{n} \cdot \vec{n}'}{|\vec{n}| \cdot |\vec{n}'|}\right).$$

Hermed har vi udledt rotationen fra $\vec{n}$ til $\vec{n}'$ på axis-angle form, og som nævnt kan $R$ beregnes ud fra denne.

**Orientering af objekt givet orientering af plan** Vi ved at et objekts orientering kan defineres ved mindst to vektorer, f.eks. $\vec{y}_{axis}$ og $\vec{z}_{axis}$, hvor $\vec{y}_{axis}$ peger mod objektets top. Vi ved også at de er ortogonale, hvilket betyder at $\vec{z}_{axis}$ ligger i planet beskrevet ved normalen $\vec{y}_{axis}$ og omvendt. Vi vil nu gerne undersøge hvordan vi kan bestemme et objekts orientering ud fra orienteringen af et plan.

For at finde orienteringen $B'$ af et objekt, givet orienteringen $\vec{n}'$ af et plan, ville vi være nødt til at introducere en sekundær aksevektor, da planet kun giver os den første i form af dets normal. Vi ønsker ikke at gætte os frem, og vælger derfor en anden metode:

Antag at vi også kender reference-orienteringen $B$ af objektet, samt reference-orienteringen $\vec{n}$ af planet. Hvis $B$'s y-akse er givet ved $\vec{y}_{axis}$, og $\vec{y}_{axis} = \vec{n}$, da kan vi udregne rotationen fra $B$ til $B'$ ved at udregne rotationen fra $\vec{n}$ til $\vec{n}'$:

$$\vec{\omega} = \|\vec{n} \times \vec{n}'\|$$

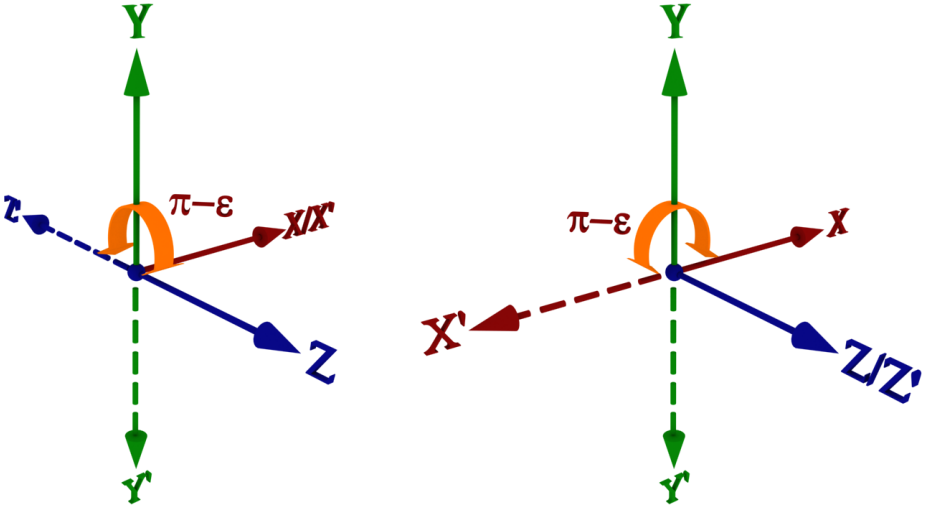
$$\theta = \cos^{-1}\left(\frac{\vec{n} \cdot \vec{n}'}{|\vec{n}| \cdot |\vec{n}'|}\right)$$

$$B' = R(\vec{\omega}, \theta)B.$$

Der er dog et par enkelte ting at bemærke ved at udlede $B'$ på denne måde:

Jo tættere $\theta$ kommer på enten 0 eller $\pi$, dvs. jo tættere $\vec{n}'$ kommer på at ligge parallelt med $\vec{n}$, desto mere vil $\vec{\omega}$ skifte for selv små ændringer i $\vec{n}'$. Det gør ikke så meget hvis $\theta$ nærmer sig 0, hvor rotationen netop er lille, men for $\theta$ der nærmer sig $\pi$ kan små skift i $\vec{n}'$, og derved store skift i $\vec{\omega}$, resultere i en markant anderledes orientering.

Eksempelvis er der stor forskel på at rotere et objekt $\theta = \pi - \epsilon$ om dets x-akse hhv. dets z-akse: Lad os antage en lille værdi for $\epsilon$, samt at objektet „peger“ langs dets y-akse. Ved rotation om $\vec{\omega}$ parallelt med dets x-akse ville objektet både vende på hovedet og pege i modsat retning. Ved rotation om $\vec{\omega}$ parallelt med dets z-akse ville objektet derimod kun vende på hovedet (se Figur 11).




**Figur 11:** Forskellen på koordinatsystemet hhv. ved rotation omkring x- og z-aksen.

Et andet problem er tvetydigheden af $\vec{\omega}$, der er nulvektoren både hvis $\theta = 0$ og $\theta = \pi$. Dette problem kan man heldigvis implementere sig ud af ved at ændre en smule på definitionen af $\vec{\omega}$:

$$\vec{\omega} = \begin{cases} \|\vec{n} \times \vec{n}'\| & \text{hvis } 0 < \theta < \pi \\ \|\vec{n} \times \vec{n}''\| & \text{ellers} \end{cases}$$

hvor $\vec{n}''$ er den sidst kendte instans af $\vec{n}'$ hvor $0 < \theta < \pi$.

Givet en orientering ved normalvektoren til et plan kan vi altså direkte overføre denne orientering til en (næsten) fuldstændig orientering af et objekt, med den ene begrænsning at objektet aldrig kan roteres om planets normal. Dette er særligt anvendeligt, eftersom vi ønsker at estimere tilt ved rotationen mellem planer, men samtidig at anvende tilt til entydigt at bestemme den fuldstændige orientering af et objekt.

### 4.2.3 Det ideelle tilfælde

Som inddata fra Wii-controlleren har vi den målte acceleration $\vec{a}_{input}$ (2). I afsnit 2.2.1 på side 4 så vi, at den målte acceleration er en sum af $\vec{g}$; accelerationen som følge af tyngdekraften, $\vec{a}$; accelerationen som følge af ændring i hastighed, samt støjen $\vec{c}$.

Lad os indtil videre antage, at controlleren ikke ændrer hastighed, hvilket medfører $\vec{a} = 0$, samt, at støjen fra digital til analog konverteringen er ikke-eksisterende, hvilket medfører $\vec{c} = 0$. Accelerationen som følge af tyngdekraften udgør da netop den målte acceleration

$$\vec{g} = \vec{a}_{input}$$

hvor $|\vec{g}| = 1$.

Den konstante længde af $\vec{g}$ er et direkte resultat af, at tyngdekraften opfattes som konstant: Så længe måleenhederne i accelerometeret udsættes for tyngdekraften, da vil disse blive påvirket med en kraft der svarer til en acceleration på $1g$ i den modsatte retning af tyngdekraften. Uanset accelerometerets orientering har vi altså en enhedsvektor $\vec{g}$, der peger direkte væk fra tyngdefeltet som accelerometeret befinder sig i (jordens, højst sandsynligt).

Vi beskrev tidligere hvordan normalen til et plan er et udtryk for planets orientering, samt hvordan man kan beregne rotationen mellem to planer givet deres normalvektorer. Definitionen af tilt var jo netop, at givet en orientering og en reference-orientering, da er tilt den rotationen, som anvendt på reference-orienteringen resulterer i orienteringen (dvs. forholdet mellem de to orienteringer).

Hvis vi kan finde en reference-orientering for accelerometeret, samt accelerometerets nuværende orientering, så kan vi benytte denne strategi til at bestemme rotationen mellem dem på axis-angle form, og derved Wii-controllerens tilt.

Lad os antage, at reference-orienteringen af accelerometeret er givet ved normalvektoren $\vec{n}_{ref}$ til det plan der beskriver den neutrale orientering (orienteringen der ikke medfører nogen rotation). Hvis $\vec{g}_{ref}$ er accelerationen som følge af tyngdekraften i den neutrale orientering, så har vi at

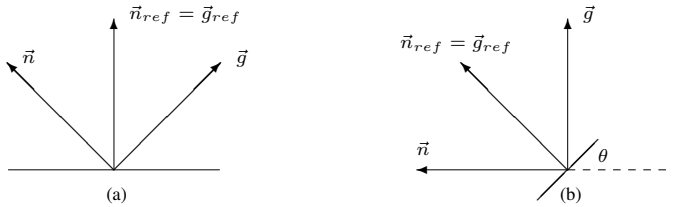
$$\vec{n}_{ref} = \vec{g}_{ref}.$$

Lad os endvidere antage, at den nuværende orientering af accelerometeret er givet ved normalvektoren $\vec{n}$, samt at $\vec{g}$ er den nuværende acceleration som følge af tyngdekraften. Eftersom $\vec{g}$ afhænger af tyngdekraftens påvirkning, og derfor altid drejer modsat den egentlige rotation (se Figur 12), da kan vi finde $\vec{n}$ ud fra $\vec{g}$ ved at spejle $\vec{g}$ i $\vec{g}_{ref}$. Først projiceres $\vec{g}$ ind på $\vec{g}_{ref}$ og projektionen skaleres til dobbelt længde, hvorefter $\vec{g}$ trækkes fra den skalerede projektion:

$$\vec{n} = 2(\vec{g} \cdot \vec{g}_{ref})\vec{g}_{ref} - \vec{g}.$$

Vi kan nu udregne tilt ved at finde rotationen fra $\vec{n}_{ref}$ til $\vec{n}$. Dette kan eksempelvis gøres som beskrevet i afsnittet omkring rotation i $\mathbf{R}^3$ (se afsnit 4.2.1, side 15); resultatet vil i da tilfælde være tilt givet ved en akse $\vec{\omega}$ samt en vinkel $\theta$.

Vi ved desuden, at givet en orientering ved normalvektoren til et plan, da kan vi direkte overføre orienteringen til den (næsten) fuldstændige orientering af et objekt (se




**Figur 12:** Forholdet mellem en reference-orientering $\vec{n}_{ref} = \vec{g}_{ref}$, en acceleration $\vec{g}$ som følge af tyngdekraften, og en orientering $\vec{n}$. Det ses at er vinklen fra $\vec{g}_{ref}$ til $\vec{g}$ er $-\theta$, og at vinklen fra $\vec{n}_{ref}$ til $\vec{n}$ er $\theta$.

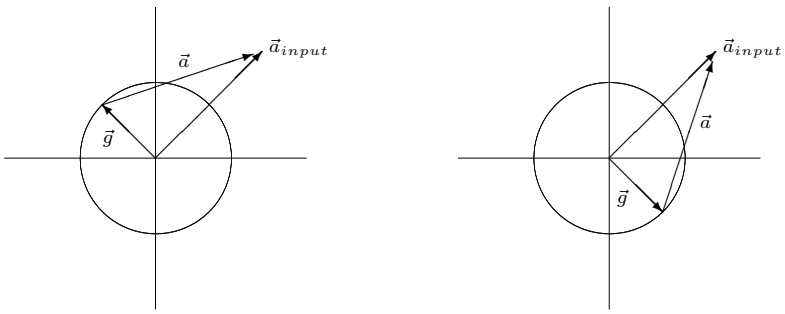
afsnit 4.2.2, side 18), hvorfor vi kan anvende estimatet af tilt til f.eks. at rotere et objekt i et virtuelt rum.

I dette tilfælde fandt vi tilt ud fra den nuværende orientering $\vec{n}$ givet $\vec{g}$; den nuværende acceleration som følge af tyngdekraften, samt $\vec{g}_{ref}$; accelerationen som følge af tyngdekraften i den neutrale orientering. Før vi kan sige noget om den nuværende orientering er vi nødt til at have kendskab til accelerationen som følge af tyndekraften, og hvis vi ikke kender denne er vi nødt til at udlede den. Værdien af $\vec{g}_{ref}$ er til gengæld et valg, som frit kan foretages, alt efter hvad man anser for at være en neutral orientering. Vi kan konkludere, at så længe vi kender acceleration som følge af tyngdekraften, da kan vi også udlede orienteringen af Wii-controlleren og derved Wii-controllerens tilt.

### 4.2.4 Det generelle tilfælde

I forrige afsnit beskrev vi hvordan man, givet den målte acceleration (2), kan bestemme tilt ud fra antagelsen $\vec{g} = \vec{a}_{input}$, hvor $\vec{g}$ er accelerationen som følge af tyngdekraften. I dette afsnit behandles problemet at estimere $\vec{g}$. Vi kalder estimatet af $\vec{g}$ for $\vec{g}'$.

I langt de fleste tilfælde vil $\vec{a}_{input} \neq \vec{g}$, eftersom vi må beregne en vis mængde støj $\vec{c}$ fra digital til analog konverteringen i Wii-controlleren, samt en skiftende acceleration $\vec{a}$ som følge af små rystelser eller større bevægelser, specielt hvis controlleren er håndholdt. Da $\vec{a}_{input}$ er en sum af vektorer er det tydeligt (se Figur 13), at vi for $\vec{a}_{input} \neq \vec{g}$ ikke entydigt kan fastslå værdien af $\vec{g}$.




**Figur 13:** Problemet med dekomponering af den målte acceleration. Her ses to tilfælde hvor den målte acceleration $\vec{a}_{input}$ er identisk, men hvor komponenterne $\vec{g}$ og $\vec{a}$, hvis sum udgør den målte acceleration, er vidt forskellige.

Med andre ord kan $\vec{g}$ tolkes på uendelig mange måder indenfor rammen $|\vec{g}| = 1$,

eftersom man for et hvilket som helst $\vec{g}$ vil kunne fremmane et passende $\vec{a}$. Dette gælder dog ikke for det omvendte tilfælde, da vi godt kan vælge $\vec{a}$ således at $\vec{a}_{input} - \vec{a} \neq \vec{g} + \vec{c}$. Sidstnævnte betyder også, at valget af $\vec{a}$ begrænses af $\vec{a}_{input}$.

Selvom vi ikke med sikkerhed kan fastslå $\vec{g}$, så kan vi måske forbedre et eventuelt estimat ved at begrænse indflydelsen af accelerationen $\vec{a}$. Logisk set, jo mindre indflydelse $\vec{a}$ har i forhold til $\vec{g}$, på $\vec{a}_{input}$, desto mere sandsynligt er det at $\vec{g}$ bidrager væsentligt til værdien af $\vec{a}_{input}$. Da $|\vec{g}| = 1$ må der gælde at

$$|\vec{a}_{input}| \leq 1 \Leftrightarrow |\vec{g} + \vec{a} + \vec{c}| \leq 1 \Leftrightarrow 0 \leq |\vec{a} + \vec{c}| \leq 2. \tag{3}$$

Hvis $|\vec{a}_{input}| \leq 1$, og derved $0 \leq |\vec{a} + \vec{c}| \leq 2$, er der med andre ord større chance for. at finde et godt estimat for $\vec{g}$, end hvis $|\vec{a}_{input}|$, og derved $|\vec{a}|$, var ubegrænset.

Vi kan præcisere et estimat yderligere ved at antage at Wii-controlleren anvendes håndholdt. For at præcisere estimatet vil vi gøre os to relaterede antagelser om værdierne $\vec{g}$ og $\vec{a}$, som følge af håndholdt anvendelse af controlleren:

1. Så længe controlleren holdes relativt stille vil den målte acceleration ligge forholdsvist tæt på $\vec{g}$. Hvis controlleren kun accelererer sagte som følge af ændring i hastighed er det sandsynligt, at $|\vec{a}| < |\vec{g}|$. Jo mere $\vec{g}$ vejer i forhold til $\vec{a}$, desto mindre vil fejlen være ved at estimere $\vec{g}$ ved blot $\vec{g}' = \|\vec{a}_{input}\|$.

2. Selv om controlleren anvendes på en sådan måde, at $\vec{a}$ svinger kraftigt, da vil der stort set altid være perioder hvor $|\vec{a}| < |\vec{g}|$. Grunden er, at $\vec{a}$ er udtryk for accelerationen som følge af en ændring i hastighed: Selv om controlleren bevæges frem og tilbage, op og ned, eller svinges fra venstre mod højre, da vil der næsten altid (med undtagelse af centrifugering og konstant bevægen i cirkler) forekomme perioder hvor ændringen i hastighed er meget lille, hvilket medfører en acceleration tæt på nul (se også afsnit 4.3.1, side 30).

Hvis der altid forekommer perioder hvor ændringen i hastighed er meget lille, dvs. perioder hvor $\vec{a}_{input} \approx \vec{g}$, da kan vi estimere $\vec{g}$ ved $\vec{g}' = \|\vec{a}_{input}\|$. Omvendt ses det, at jo kortere der er mellem disse perioder med lille ændring i hastighed, desto større chance er der for, at $|\vec{g}| - k \leq |\vec{a}_{input}| \leq |\vec{g}| + k$ medfører, at $\vec{a}_{input} \approx \vec{g}$, hvor $k$ er en konstant der angiver den tilladte afvigelse.

Lad os derfor antage at

$$1 - k \leq |\vec{a}_{input}| \leq 1 + k \Rightarrow \vec{g}' = \|\vec{a}_{input}\| \Rightarrow \vec{g}' \approx \vec{g}. \tag{4}$$

Med andre ord siger vi at $\vec{a}_{input}$ approksimerer $\vec{g}$ hvis $1 - k \leq |\vec{a}_{input}| \leq 1 + k$.

Vi mangler da at estimere $\vec{g}$ for tilfældene $|\vec{a}_{input}| < 1 - k$ og $|\vec{a}_{input}| > 1 + k$. I den forbindelse kigger vi først på anvendelse af interpolering og lavpasfiltrering, og i næste afsnit giver vi så vores bud på en adaptiv metode til løsning af problemet.

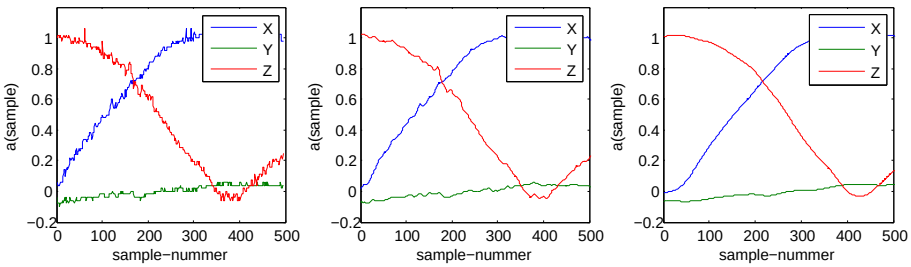
**Interpolering** Lad os først betragte $\vec{a}_{input}$ som et signal med længde $n$, hvor $\vec{a}_{input}[0]$ er den nyeste sample, og $\vec{a}_{input}[n-1]$ er den ældste sample. Det er oplagt at man vil kunne udvælge to samples $\vec{a}_{input}[i]$ hhv. $\vec{a}_{input}[j]$, der begge approksimerer $\vec{g}$ ved $\vec{g}'_i$ hhv. $\vec{g}'_j$. $\vec{g}'$ kan da beskrives for samples i intervallet $[i+1; j-1]$ ved at interpolere mellem $\vec{g}'_i$ og $\vec{g}'_j$.

Problemet med at lade $\vec{g}'$ afhænge af interpolering er, at vi for en ny sample $\vec{a}_{input}[0]$ ikke altid vil kunne udlede $\vec{g}'$ øjeblikkeligt: Hvis den nye sample ikke selv approksimerer $\vec{g}$, så vil $\vec{g}'$ for den nye sample afhænge af $\vec{g}'$ for en fremtidig sample, og hvis der da går $m$ samples indtil vi støder på en sample der approksimerer $\vec{g}$, så vil vores estimat af $\vec{g}'$ være $m$ samples bagud.

Eftersom Wii-controlleren rapporterer accelerationen ca. 100 gange i sekundet (se afsnit 2.2.2, side 5), og da det er ganske muligt at bevæge controlleren således, at den konstant ændrer hastighed over et interval på mere end 1 sekund, så kan vi f.eks. godt komme ud for $m = 100$. Derudover har vi ikke noget mål for hvor stort et $m$ vi rent faktisk kan støde på, og estimatet $\vec{g}'$ vil derfor være et variabelt antal samples bagud.

Vi er interesserede i et estimat af tilt der indenfor rimelig tid afspejler controllerens nuværende orientering. Eksempelvis er det meget tydeligt, hvis der er 1 sekunds forskel mellem controllerens nuværende orientering i virkeligheden, og afbildningen af estimatet af tilt på skærmen. Interpolering mellem estimater af $\vec{g}$ er derfor ikke attraktivt i vores tilfælde.

**Lavpasfiltrering** En anden fremgangsmåde er at betragte $\vec{a}_{input}$ som et signal der består af høje og lave frekvenser. Hvis $\vec{g}$ er lavfrekvent i forhold til $\vec{a}$ og $\vec{c}$, så kan vi anvende et lavpasfilter til at mindske effekten af $\vec{a}$ og $\vec{c}$, for derefter at estimere $\vec{g}$ direkte ud fra den filtrerede værdi.




**Figur 14:** Figuren viser effekten af et let lavpasfilter og et kraftigt lavpasfilter på et ufiltreret signal. Filteret anvendt i dette eksempel er et en-dimensionelt Gaussfilter med standardafvigelse $\sigma$. Fra venstre mod højre ses: (a) det ufiltrerede signal, (b) det let filtrerede signal; $\sigma = 3$, og (c) det kraftigt filtrerede signal; $\sigma = 15$.

Jo kraftigere et lavpasfilter der anvendes, desto mere udglattes det oprindelige signal (se Figur 14). Ifølge tidligere antagelser (3,4) om præcisionen af et estimat har dette den effekt, at for værdier af $\vec{a}_{input}$ hvor $1 - k \leq |\vec{a}_{input}| \leq 1 + k$, da vil estimatet $\vec{g}'$ være mindre præcist i forhold til $\vec{g}$, end ved en mindre kraftig lavpasfiltrering. Med andre ord risikerer vi med et kraftigt lavpasfilter at udglatte værdier af $\vec{a}_{input}$, som uden filtrering ville medføre $\vec{g}' = \|\vec{a}_{input}\| \Rightarrow \vec{g}' \approx \vec{g}$.

Til gengæld vil en kraftig lavpasfiltrering sikre, at $\vec{a}$ og $\vec{c}$ får mindre indflydelse på den filtrerede værdi, og derved stiger den gennemsnitlige præcision af $\vec{g}'$. Vælger vi derfor et tilpas kraftigt lavpasfilter $F$, så kan vi estimere $\vec{g}$ ved

$$\vec{g}' = \|F(\vec{a}_{input})\|.$$

Vi kalder ovenstående metode, at normalisere de eventuelt lavfiltrerede inddata, for

den *naive metode* til estimering af tilt.

Der er dog tydelige problemer ved udelukkende at benytte lavpasfiltrering til at estimere $\vec{g}$. Uanset hvor godt man tilpasser styrken af sit lavpasfilter (uanset hvor kraftigt eller let det gøres), vil der altid være en pris der skal betales. Benyttes et kraftigt filter betales prisen i form af præcisionen af $\vec{g}'$ for værdier af $\vec{a}_{input}$, der i forvejen approksimerer $\vec{g}$, hvilket, som med interpolering, giver udslag i form af dårligere responstid. Benyttes et mindre kraftigt filter betales prisen i form af dårligere udelukkelse af $\vec{a}$ og $\vec{c}$, hvilket reducerer præcisionen af $\vec{g}'$ for værdier af $\vec{a}_{input}$, der ikke approksimerer $\vec{g}$.

Som et alternativ til lavpasfiltrering har vi derfor udviklet en *adaptiv metode*. Metoden beskrives i det efterfølgende afsnit (4.2.5).

### 4.2.5 En adaptiv metode for det generelle tilfælde

Vi så i forrige afsnit hvilke problemer der er forbundet med, at estimere tilt for det generelle tilfælde. I dette afsnit beskrives en adaptiv metode, ESTIMATEGRAVITY, til at beregne et estimat $\vec{g}'$ af $\vec{g}$; accelerationen som følge af tyngdekraften, ud fra den målte acceleration $\vec{a}_{input}$ (2).

Metoden bygger på endnu en antagelse om brugen af Wii-controlleren: Ved normal håndholdt brug eksisterer der en øvre grænse for vinklen, hvormed controllerens orientering kan ændre sig mellem to målinger af accelerationen. Vi kalder den øvre grænse for vinklen for $\beta$. Eksempel: Hvis controlleren rapporterer accelerationen ca. 100 gange i sekundet (se afsnit 2.2.2, side 5), da vil $\beta$ være den vinkel hvormed $\vec{g}$ maksimalt kan ændre sig på 10 millisekunder.

**Formål** Den grundlæggende ide er at benytte antagelsen om en maksimal forskel i vinklen mellem to orienteringer til, at afgøre hvor meget et estimat $\vec{g}'$ kan afvige fra et tidligere estimat $\vec{g}'_{base}$, således at antallet af mulige værdier for estimatet $\vec{g}'$ reduceres. Herudover ønsker vi at stabilisere estimatet for „usikre“ værdier af $\vec{a}_{input}$, således at effekten af f.eks. en pludselig acceleration som følge af ændring i hastighed ikke kommer til at påvirke estimatet voldsomt. Ved adaptivt at begrænse værdien af estimatet, samt indflydelsen fra usikre inddata, håber vi, at kunne opnå gode resultater i forhold til en simpel lavpasfiltrering af signalet.

Givet $\vec{a}_{input}$ ønsker vi med andre ord at beregne $\vec{g}'$ på baggrund af aksen og vinklen $(\vec{\omega}, \theta)$ mellem en reference-vektor $\vec{g}'_{base}$ og inddata $\vec{a}_{input}$. Estimatet $\vec{g}'$ kan derfor betragtes som en vægtet rotation af $\vec{g}'_{base}$, hvor aksen hhv. vinklen for rotationen er givet ved $(\vec{\omega}, \theta \cdot W(\vec{a}_{input}))$, hvor funktionen $W$ angiver vægten af dens argument som en værdi mellem 0 og 1.

**Definitioner** Vægten af $\vec{a}_{input}$ er udtryk for anvendeligheden af disse data, og bestemmes ud fra tidligere antagelser (3,4) om chancen for at $\vec{a}_{input}$ approksimerer $\vec{g}$. Vi antog tidligere at chancen for at $\vec{g}$ udgør en betydelig del af værdien af $\vec{a}_{input}$ er højst når $1 - k \leq |\vec{a}_{input}| \leq 1 + k$, hvorfor vi definerer vægtfunktionen $w$ ved:

$$W(\vec{v}) = \begin{cases} |\vec{v}| & hvis |\vec{v}| \leq 1 \\ 1/|\vec{v}| & ellers \end{cases} .$$

En høj vægt er således udtryk for en mere anvendelig værdi, og vægtfunktionen

benyttes i den forstand til at begrænse hvor meget indflydelse $\vec{a}_{input}$ har på værdien af et eventuelt estimat $\vec{g}'$, i forhold til reference-vektoren $\vec{g}'_{base}$.

Vi siger at $\vec{a}_{input}$ er *kandidat til* reference-vektoren $\vec{g}'_{base}$ hvis

$$w(\vec{a}_{input}) \geq 1 - k$$

hvor $k$ er en konstant, $0 \leq k \leq 1$, der angiver den tilladte afvigelse i vægt.

Lad vinklen mellem to vektorer være givet ved $\theta(\vec{a}, \vec{b})$. Vinklen $\alpha$ angiver den tilladte afvigelse i vinklen mellem reference-vektoren $\vec{g}'_{base}$ og inddata $\vec{a}_{input}$. Vi siger at $\vec{a}_{input}$ *overholder vinkelbegrænsningen* såfremt

$$\theta(\vec{g}'_{base}, \vec{a}_{input}) \leq \alpha.$$

Eftersom $\vec{g}'$ er resultatet af den vægtede rotation fra $\vec{g}'_{base}$ til $\vec{a}_{input}$ vil det altid gælde, at

$$\theta(\vec{g}'_{base}, \vec{g}') \leq \theta(\vec{g}'_{base}, \vec{a}_{input})$$

hvilket med andre ord betyder, at et estimat $\vec{g}'$ overholder vinkelbegrænsningen hvis blot dets inddata $\vec{a}_{input}$ overholder vinkelbegrænsningen.

Vi kan nu definere reference-vektoren $\vec{g}'_{base}$. Reference-vektoren $\vec{g}'_{base}$ er det seneste tidligere estimat $\vec{g}'$, hvor inddata $\vec{a}_{input}$ både overholdte vinkelbegrænsningen og var kandidat til den forrige reference-vektor $\vec{g}'_{base}$.

**Beskrivelse af metoden** Vi ved at ethvert estimat $\vec{g}'$ skal overholde vinkelbegrænsningen. Vi ved også ud fra definitionen, at overholder $\vec{a}_{input}$ vinkelbegrænsningen, så overholder estimatet den også. Hvis $\vec{a}_{input}$ derimod ikke overholder vinkelbegrænsningen, dvs. hvis $\theta(\vec{g}'_{base}, \vec{a}_{input}) > \alpha$, da begrænser vi det resulterende $\theta$, således at vinklen igen er opadtil begrænset af vinklen $\alpha$.

Ideen bag begrænsningen af $\theta$ er at mindske mængden af falske positiver. Med falske positiver mener vi de tilfælde hvor $\vec{a}_{input}$ har høj nok vægt til, at være kandidat til $\vec{g}'_{base}$ uden at værdien af $\vec{a}_{input}$ rent faktisk afspejles af accelerationen som følge af tyngdekraften. Eksempel: Hvis controlleren holdes vandret og derefter accelereres i løbet af én sample med 2g i modsat retning af $\vec{g}$, da vil den målte acceleration være $-\vec{g}$. Uden vinkelbegrænsningen ville dette forårsage et estimat svarende til, at controlleren pludselig vendte på hovedet.

Det resulterende $\theta$ vægtes desuden efter anvendeligheden af $\vec{a}_{input}$, således at mindre anvendelige input automatisk forårsager en mindre værdi for $\theta$. Herved beskyttes $\vec{g}'$ mod påvirkning fra f.eks. en pludselig acceleration som følge af ændring i hastighed.

Efter at have begrænset og vægtet værdien af $\theta$ er vi sikre på, at en rotation fra $\vec{g}'_{base}$ til $\vec{a}_{input}$ vil ligge indenfor $\alpha$, den tilladte afvigelse i vinklen, og så kan vi udregne estimatet $\vec{g}'$. Det er dog ikke nok bare at udregne estimatet, da vi også skal huske at opdatere reference-vektoren; ellers ville metoden jo være låst til altid at finde estimater indenfor $\alpha$ af den første reference-vektor.

Hvis $\vec{a}_{input}$ både er kandidat til $\vec{g}'_{base}$ og overholder vinkelbegrænsningen, da sættes $\vec{g}'_{base}$ lig det fundne estimat $\vec{g}'$, hvorefter den tilladte afvigelse i vinklen mellem reference-vektoren og fremtidige inddata nulstilles ($\alpha = \beta$). Reference-vektoren opdateres således kun hvis inddata både overholder vinkelbegrænsningen, og vægter over eller lig $1 - k$.

Hvis $\vec{a}_{input}$ er kandidat til $\vec{g}'_{base}$ men *ikke* overholder vinkelbegrænsningen, så betragtes det i stedet som tegn på, at $\vec{g}$; accelerationen som følge af tyngdekraften, måske har flyttet sig i retning af $\vec{a}_{input}$. I da tilfælde udvides $\alpha$ med $\beta$, således at den tilladte afvigelse for vinklen er større for næste iteration. Udvidelsen af $\alpha$ bevirker, at inddata til næste iteration af metoden vil have større chance for at overholde vinkelbegrænsningen.

Man kan diskutere om det er bedre at udvide vinklen $\alpha$ mellem hver iteration, uanset om $\vec{a}_{input}$ er kandidat til $\vec{g}'_{base}$ eller ej. Vi er imidlertid meget interesserede i hvor godt vi kan lukke af for påvirkninger, som ikke stammer fra accelerationen som følge af tyngdekraften, og i da tilfælde vil det være en fordel at holde $\alpha$ nede, således at det kun er kandidater til reference-vektoren der får lov at forøge området for vinkelbegrænsningen. Hvis vi valgte at forøge $\alpha$ i hver iteration, uanset inddata, så ville vi ganske vist få en mere korrekt vinkelbegrænsning, men samtidig ville vi miste stabiliteten som begrænsningen giver os.

Algoritme 1 viser udkast til implementation af metoden:

**Algorithm 1** ESTIMATEGRAVITY($\vec{a}_{input}$)

```
1:  g' ← 0 // Nulstil estimatet
2:  (ω, θ) ← AXISANGLE(g'_base, a_input) // Beregn akse og vinkel
3:  if a_input overholder vinkelbegrænsningen then
4:      θ ← θ · w(a_input)
5:      if a_input er kandidat til g'_base then
6:          g' ← R(ω, θ) g'_base
7:          g'_base ← g' // Opdater reference-vektoren
8:          α ← β // Nulstil den tilladte afvigelse
9:      else
10:         g' ← R(ω, θ) g'_base
11:     end if
12: else
13:     θ ← α · w(a_input) // θ begrænses her opadtil af α
14:     if a_input er kandidat til g'_base then
15:         g' ← R(ω, θ) g'_base
16:         α ← α + β // Forøg den tilladte afvigelse
17:     else
18:         g' ← R(ω, θ) g'_base
19:     end if
20: end if
21: return g' // Returner estimatet
```

Efter initialisering kan metoden umiddelbart efter benyttes til at beregne estimater. Under kørslen opdateres værdierne for $\vec{g}'_{base}$ og $\alpha$, og disse værdier ændrer sig således fra iteration til iteration (de kan betragtes som globale variable). Vi betragter metoden som adaptiv, da estimatet afhænger af disse selvopdaterende værdier.

**Initialisering af metoden** Metoden skal initialiseres før brug, eftersom $\vec{g}'$ afhænger af reference-vektoren $\vec{g}'_{base}$, samt indirekte af værdierne $\beta$ og $k$. $\beta$ angiver som nævnt den maksimale vinkel mellem to samples, og $k$ angiver den tilladte afvigelse i vægten af inddata før inddata udelukkes fra at være kandidat til $\vec{g}'_{base}$.

Algoritme 2 viser udkast til implementation af initialisering af metoden:

**Algorithm 2** INITESTIMATEGRAVITY($\vec{g}_{ref}$,$\beta_{choice}$,$k_{choice}$)

1: $\vec{g}'_{base} \leftarrow \vec{g}_{ref}$ // Initialisér reference-vektoren
2: $\beta \leftarrow \beta_{choice}$ // Initialisér $\beta$ til den angivne værdi
3: $k \leftarrow k_{choice}$ // Initialisér $k$ til den angivne værdi
4: $\alpha \leftarrow \beta$ // Nulstil den tilladte afvigelse

Som første reference-vektor $\vec{g}'_{base}$ er det oplagt, at benytte accelerationen for den første kendte orientering af controlleren, som er givet ved $\vec{g}_{ref}$ (se afsnit 4.2.3, side 20). I så tilfælde vil reference-vektoren nemlig beskrive en reel orientering fra første iteration.

**Valg af $\beta$** Konstanten $\beta$ beskriver som sagt den maksimale forskel i vinklen mellem to samples. Jo lavere en værdi der vælges for $\beta$, desto bedre vil metoden være til at udelukke effekten af pludselig acceleration som følge af ændring i hastighed. Omvendt kan en lav værdi for $\beta$ potentielt give anledning til falske negativer: Hvis brugeren formår at rotere Wii-controlleren hurtigere end $\beta$ og derfor end $\alpha$ tillader, da vil der gå et vist antal iterationer førend inddata igen overholder vinkelbegrænsningen, også selvom inddata $\vec{a}_{input}$ for hver iteration formår at være kandidat til reference-vektoren $\vec{g}'_{base}$.

Eksempel: Lad os antage at $\beta = \frac{1}{4}\pi$, at $\alpha = \frac{1}{4}\pi$, samt at reference-vektoren er sat til accelerationen ved reference-orienteringen af Wii-controlleren. Hvis brugeren inden næste iteration (dvs. mellem to samples) formår, at rotere (ren rotation, ingen anden bevægelse, dvs. $W(\vec{a}_{input}) = 1$) controlleren med en vinkel $\theta = \pi$, for derefter at holde controlleren stille, da vil de efterfølgende iterationer være:

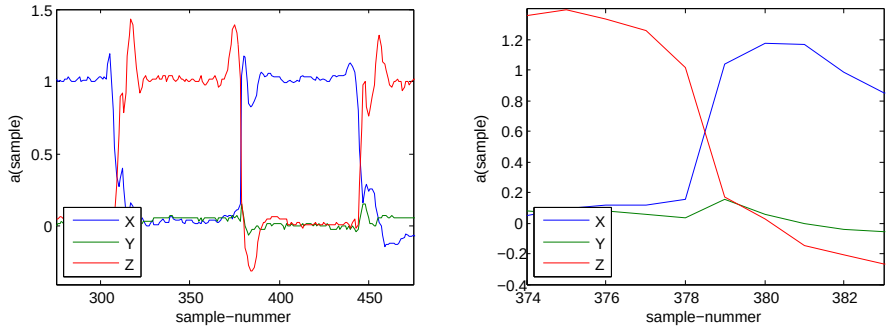
1. $\theta > \alpha$: Vinkelbegrænsningen er ikke overholdt og estimatet begrænses af $\alpha$, hvorefter $\alpha$ forøges; $\alpha = \alpha + \beta = \frac{1}{2}\pi$.
2. $\theta > \alpha$: Vinkelbegrænsningen er ikke overholdt og estimatet begrænses af $\alpha$, hvorefter $\alpha$ forøges; $\alpha = \alpha + \beta = \frac{3}{4}\pi$.
3. $\theta \geq \alpha$: Vinkelbegrænsningen er nu overholdt og estimatet begrænses ikke. Referencevektoren opdateres og $\alpha$ nulstilles; $\alpha = \beta$.

Det er tydeligt ud fra eksemplet, at hvis $\beta$ ikke svarer nogenlunde til vinklen, hvormed brugeren typisk formår at rotere Wii-controlleren mellem to samples, da introduceres et efterslæb som følge af begrænsning af estimatet. I dette tilfælde tog det 3 iterationer at nå frem til et ubegrænset estimat. Hvis accelerometeret i controlleren samples 100 gange i sekundet (se afsnit 2.2.2, side 5) vil der være 10 millisekunder mellem hver iteration, hvilket i dette tilfælde svarer til en forsinkelse på 20 millisekunder.

Vælges $\beta$ derimod for højt, da vil metoden som nævnt være dårligere til at udelukke pludselig acceleration som følge af ændring i hastighed. $\beta$ bør derfor vælges i forhold til vinklen, hvormed man regner med at controlleren vil blive roteret på 10 millisekunder under brug.

For at finde ud af hvor højt $\beta$ bør vælges har vi derfor udført et lille eksperiment. Vi ved at den målte acceleration ved nogenlunde stilstand er en approksimation til accelerationen som følge af tyngdekraften, samt at orienteringen er givet ved sidstnævnte.

Ved først at holde controlleren vandret for derefter at rotere mellem to orienteringer, og over kun én akse, da kan vi se på signalet hvordan accelerationen skifter mellem to akser. Ved at foretage rotationen så hurtigt som muligt over én akse vil vi få et skift i accelerationen over to akser, hvor den største forskel mellem to samples er udtryk for det størst mulige skift i vinklen mellem to samples.




**Figur 15:** Figuren viser to plots af udsnit af data fra eksperimentet med hurtig rotation: Til venstre ses udsnit af de målte data indeholdende flere forsøg med rotation omkring y-aksen, og til højre ses det udvalgte stykke, hvor accelerationen, fordelt på x-aksen og z-aksen, krydser stærkest som følge af hurtig rotation i y-aksen.

I vores eksperiment valgte vi at holde controlleren med begge hænder, og at udføre rotationen hurtigst muligt om controllerens y-akse, hvorfor skiftet i acceleration foregik mellem z-aksen og x-aksen. Vi aflæste den målte acceleration ca. 100 gange per sekund. På Figur 15 ses et udsnit af data fra eksperimentet (venstre), samt et udsnit af data fra den rotation hvor skiftet i vinklen mellem to samples var størst (højre). Ud fra de målte data fandt vi, at den største numeriske forskel mellem to samples var en acceleration på mellem $0.88g$ (x-aksen) og $0.85g$ (z-aksen), dvs. $0.88g$. Vinklen for den største forskel er da givet ved

$$\sin^{-1}(0.88) \approx 1.075 \approx \frac{1}{3}\pi.$$

Resultatet af eksperimentet, $\frac{1}{3}pi$, viser kun hvor hurtigt *vi* har formået at rotere controlleren, hvilket er yderst subjektivt. På den anden side viser resultatet også, at det er *muligt* at rotere controlleren $\frac{1}{3}\pi$ på ca. 10 millisekunder (jvf. ca. 100 samples per sekund). Det var dog temmelig anstrengende at opnå denne hastighed, hvorfor vi ikke regner med at controlleren ved normal brug vil blive roteret med $\frac{1}{3}\pi$ per sample.

$\beta$ bør vælges så lavt som muligt, hvilket stemmer godt overens med ovenstående observation. Hvis vi antager at controlleren ved normal brug maksimalt vil blive udsat for en rotation, per sample, på halvdelen eller mindre af $\frac{1}{3}\pi$, så ved vi i det mindste, at vi bør vælge $\beta$ således at

$$0 \leq \beta \leq \frac{1}{6}\pi. \tag{5}$$

**Valg af $k$** Konstanten $k$ beskriver hvor meget vægten af $\vec{a}_{input}$ må afvige fra den maksimale vægt, førend inddata afvises som kandidat til reference-vektoren. Jo højere

værdi vi vælger for $k$, desto mere følsomt vil estimatet blive overfor støj og acceleration som følge af ændring i hastighed. På den anden side er det også vigtigt at vælge en værdi for $k$, som ikke fejlagtigt udelukker nogenlunde acceptable værdier af $\vec{a}_{input}$ fra at blive kandidater til $\vec{g}'_{base}$:

- Hvis inddata $\vec{a}_{input}$ ikke indeholder nogen form for signalstøj, dvs. $\vec{c} = 0$, da bør $k$ vælges i forhold til den acceleration som følge af hastighed; $\vec{a}$, der opstår, som følge af f.eks. rystelser på hånden eller anden menneskelig „støj“, når Wii-controlleren benyttes håndholdt og hovedsageligt roteres. Hvis $\vec{a}$ udelukkende kommer fra små rystelser o.lign., da er det oplagt at vælge $k$ i nærheden af $|\vec{a}|$. Vi ønsker at vælge en konstant værdi for $k$, samt en værdi der udelukker falske positiver så godt som muligt. Hvis $|\vec{a}|$ svinger meget kan det derfor være interessant at betragte middelværdien af nogle målinger af $\vec{a}$, hvorfor vi typisk vil have at $k_{choice} = mean(|\vec{a}|)$.
- Hvis inddata $\vec{a}_{input}$ derimod indeholder støjen $\vec{c}$, da bør $k$ vælges således, at støjens indflydelse på vægten tages i betragtning. I praksis vil dette betyde, at man er nødt til at vælge $k$ højere end ellers antaget. Vægtfunktionen $W$ er længdebaseret, hvorfor støjens maksimale indflydelse på vægten kan betragtes som den maksimale længde af $\vec{c}$. Typisk vil vi derfor for støjfyldte inddata have, at $k_{choice} = k_{choice} + max(|\vec{c}|)$.

Valget af $k$ afhænger med andre ord af, om $\vec{a}_{input}$ indeholder støjen $\vec{c}$ eller ej. Hvis man i forvejen har karakteriseret $\vec{c}$ (se afsnit 4.3.4, side 34) kan det være en fordel, at benytte et filter $F$ til at fjerne støjen fra $\vec{a}_{input}$ før der gøres brug af den adaptive metode, da man så kan vælge en strammere værdi for $k$.

I så tilfælde bør metoden anvendes med den filtrerede værdi af $\vec{a}_{input}$, og estimatet $\vec{g}'$ er da givet ved:

$$\vec{g}' = \text{ESTIMATEGRAVITY}(F(\vec{a}_{input}))$$

hvor $F$ er filteret der begrænser støjen.

**Køretid** Køretiden af den adaptive metode er væsentligt højere end køretiden for den naive metode. Begge metoders køretid er asymptotisk konstant, $O(1)$, men da metoderne typisk vil blive anvendt i *realtime*-applikationer kan det være hensigtsmæssigt, at undersøge køretiden ved, at tælle antallet af operationer der skal udføres af metoderne ved beregning af et estimat.

Den naive metode normaliserer blot dens inddata, hvilket kræver 3 multiplikationer, 2 additioner, én kvadratrodsoperation, og 3 divisioner. Den adaptive metode er derimod mere kompleks og består af flere skridt:

1. Beregning af aksen og vinklen mellem to vektorer, hvilket kræver 12 multiplikationer, 7 additioner, 4 divisioner, 2 kvadratrodsoperationer, og én cosinusoperation.
2. Vægtning af inddata, hvilket kræver én multiplikation, og i mange tilfælde én division.
3. Beregning af rotationsmatrix ud fra akse og vinkel, hvilket kræver 24 multiplikationer, 10 additioner, én sinusoperation, og én cosinusoperation.

4. Beregning af roteret vektor ud fra rotationsmatrix, hvilket kræver 9 multiplikationer, og 6 additioner.

Alt i alt kræver den adaptive metode 46 multiplikationer, 23 additioner, 5 divisioner, 2 kvadratrodsoperationer, 2 cosinusoperationer, og én sinusoperation. Det totale antal operationer for den naive metode er 9, hvor det totale antal operationer for den adaptive metode er 79.

En moderne processor vil ikke umiddelbart have problemer med, at afvikle hverken den naive eller den adaptive metode mange tusinde gange i sekundet, hvorfor begge metoder er ganske velegnede til brug i realtime-applikationer.

## 4.3 Estimering af position

I dette afsnit behandler vi hvorledes man kan estimere den relative position af Wii-controlleren ud fra den modtagne accelerometer-data. Vi vil først gennemgå forskellige metoder til at udlede positionen fra accelerationen, samt beskrive hvorledes disse kan anvendes på de kendte acceleration (det ideelle tilfælde), hvorefter vi vil analysere og behandle problemet, at estimere positionen ud fra den målte acceleration (det generelle tilfælde).

### 4.3.1 Forhold mellem acceleration, hastighed og position

Der eksisterer en sammenhæng mellem accelerationen ($\vec{a}$), hastigheden ($\vec{v}$) og positionen ($\vec{x}$) af et objekt, således at det er muligt at udlede den relative hastighed fra accelerationen og den relative position fra hastigheden. Figur 16 viser plots af en acceleration og den relative afledte hastighed og position heraf.

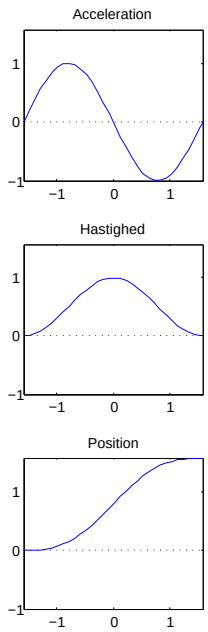
Det ses, af figuren, at accelerationen er et udtryk for ændring i hastigheden, idet accelerationen har sine positive og negative ekstremer ved samme $t$, som hvor hastigheden stiger eller falder mest. Tillige ses at hastigheden er et udtryk for ændring i positionen, idet hastigheden har sit positive ekstrem ved samme $t$, som hvor positionen ændrer sig mest.

For at udlede positionen fra accelerationen, som vi skal i vores tilfælde, er det muligt, at foretage en dobbeltintegration af accelerationen, således at man efter første integration har hastigheden, og efter anden integration, som er på hastigheden, har positionen.

Da Wii-controlleren udlæser samplet data fra accelerometeret vil vi være nødt til, at benytte numerisk integration, for at udlede positionen. Dette skaber en mindre usikkerhed, idet vi ikke har det komplette signal fra accelerometeret, og derfor vil den udledte position kun være et estimat af den reelle position af Wii-controlleren.

### 4.3.2 Metoder til numerisk integration

Der findes forskellige metoder til numerisk integration, hvoraf en del er afarter af den konstante Eulerintegration. Eulerintegratoren antager, at værdien af signalet er konstant i intervallet fra $a$ til $b$. For *left endpoint* Eulerintegration er denne værdi $f(a)$, hvor $f(t)$ er en funktion der returnerer værdien af det modtagne signal ved tiden $t$. $a$ og $b$ begrænser således det vindue integratoren arbejder ud fra (se figur 17).




**Figur 16:** Figuren viser sammenhængen mellem hhv. acceleration, hastighed og position, som kan udledes af hinanden vha. integration.

Vi vil tage udgangspunkt i den konstante Eulerintegration (herefter Eulerintegration) og sammenligne denne med hhv. trapez- og midpointintegration, som begge er varianter af Eulerintegration.

**Eulerintegration** Som antydet interpolerer den konstante Eulerintegration ikke mellem tiderne $a$ og $b$ og vil således være behæftet med en relativt stor negativ afvigelse i forhold til den ideelle integration.

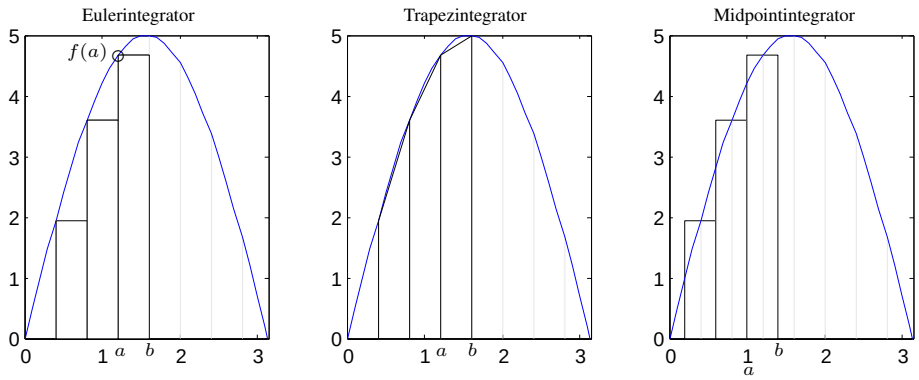
Ved Eulerintegration er tiderne $a$ og $b$ bestemt ud fra hhv. den forrige sample og den seneste sample og sampelængden for denne periode er således $\Delta ab = b - a$. På grund af det potentielt store tab af information, hvis kurven mellem $a$ og $b$ er meget stejl, er Eulerintegration en middelmådig tilnærmelse til den ideelle integration:

$$\int_a^b f(t)\,dt \approx I_{Euler} = \Delta ab \cdot f(a).$$

Eulerintegration har således ikke nogen forsinkelse idet den seneste sample direkte bliver brugt i integratoren.

**Trapezintegration** Trapezintegration er, som nævnt, en variation af den konstante Eulerintegration, men hvor Eulerintegration benytter et 0.-ordens polynomie benytter trapezintegration et 1.-ordens polynomie hvilket giver en lineær interpolering mellem $f(a)$ og $f(b)$.

Som det ses i Figur 17 minder trapezintegration meget om Eulerintegration, da de




**Figur 17:** Figuren viser hvordan vinduet dannet af $a$ og $b$ er forskelligt mellem de tre integrations-metoder; Eulerintegration, trapezintegration og midpointintegration.

har samme placering mellem samples, men trapezintegration foretager en lineær interpolering mellem $f(a)$ og $f(b)$ ved at tage højde for forskellen mellem de to punkter.

Lineær interpolering siger dog ikke ret meget om hvad der sker mellem $f(a)$ og $f(b)$, og kan således „overse“ ændringer i signalet i tidsintervallet $a$ til $b$. På trods af dette er det tydeligt, at Trapezintegration er en bedre tilnærmelse til den ideelle integration end Eulerintegration:

$$\int_a^b f(t)\,dt \approx I_{Trapez} = \Delta ab \cdot \frac{f(a)+f(b)}{2}.$$

Trapezintegratione har, ligesom Eulerintegratione ingen forsinkelse, da den nyeste sample benyttes til integration.

**Midpointintegration** Midpointintegration minder mere om Eulerintegration end trapezintegration, hvis man ser på selve vinduets form (se figur 17). Dette gør sig dog ikke gældende for metodens virkemåde, idet vinduet er forsinket en halv samplelængde, hvilket gør at en del af vinduet ligger over signalet, såfremt værdien af signalet ikke er konstant i det pågældende vindue. Dette gør at integrationen er et tilnærmet gennemsnit over signalet fra $f(a)$ til $f(b)$.

Ligesom Eulerintegration benytter midpointintegration et 0.-ordens polynomie, men integrationen ligger tættere på den ideelle integration end Eulerintegration, på grund af det forskudte vindue. Midpointintegration er givet ved

$$\int_a^b f(t)\,dt \approx I_{Midpoint} = \Delta ab \cdot f(\frac{a+b}{2}).$$

Midpointintegration er, i modsætning til de to tidligere nævnte integratorer, forsinket 0.5 samples, idet området der integreres pr. sample er forskudt 0.5 samples tilbage i tiden.

**Usikkerhed** Ved alle numeriske integratorer er der en risiko for, at miste betydningsfulde detaljer af det signal der integreres. Eulerintegratoren kan „overse“ detaljer i signalet, hvilket kan give en afvigelse, som i vores tilfælde kan medføre *drift*.

Vi kan forestille os et udsnit af et signal, hvor accelerationen, som udgangspunkt er $0$ og der eksisterer to forskellige peaks; en positiv og en negativ, hvor det eksakte integrale af den positive peak er $x$ og det tilsvarende integrale af den negative peak er $-x$.

Ved ideel integration vil de to integraler udligne hinanden, og hastigheden efter de to peaks vil være $0$, men ved numerisk integration, som Eulerintegration, er det usandsynligt, at integralerne af de to peaks udligner hinanden, og således vil der opstå en usikkerhed, $y$, som påvirker hastigheden efter de to peaks. Således kan der opstå en falsk hastighed, som vil akkumulere ved integration til position og medvirke til et forkert positionsestimat. Det er derfor vigtigt, at mindske denne tendens så meget som muligt. Lignende usikkerhed kan opstå ved trapez- og midpont-integration.

For at undgå, eller mindske, denne fejl af numerisk integration er det vigtigt at have en høj samplerate, så det er så lidt som muligt, af signalet, der går tabt mellem samples. Det er desuden også muligt, at mindske denne fejl ved at benytte integratorer, der bedre tilnærmer sig signalets oprindelige form, mellem samples, ved interpolation eller ved højere orden. Eksempler på sådanne integratorer er Verlet, der benytter et 1.-ordens polynomie, men kan foretage den dobbelte integration i ét skridt; og RK4 (Runge-Kutta 4), der benytter et 4.-ordens polynimie.

### 4.3.3 Det ideelle tilfælde

Som inddata fra Wii-controlleren har vi den målte acceleration $\vec{a}_{input}$ (2). Denne er som nævnt en sum af accelerationen som følge af tyngdekraften, accelerationen som følge af ændring i hastighed, samt en støj; $\vec{g}$, $\vec{a}$, og $\vec{c}$. Lad os indtil videre antage at støjen er ikke-eksisterende, hvilket medfører $\vec{c} = 0$, samt at vi på et hvilket som helst tidspunkt kender $\vec{g}$, accelerationen som følge af tyngdekraften. Vi har da at

$$\vec{a} = \vec{a}_{input} - \vec{g}.$$

Vi så tidligere hvordan der eksisterer et forhold mellem accelerationen, hastigheden, og positionen, og vi kan derfor anvende accelerationen $\vec{a}$ til at estimere controllerens hastighed og position. Ved at integrere accelerationen to gange, f.eks. ved hjælp af en af de tre tidligere beskrevede integratorer; Euler, trapez eller midpoint, vil resultatet være et estimat af hhv. hastigheden og positionen af controlleren.

Hvis vi f.eks. vælger den simple Eulerintegrator vil estimatet af Wii-controllerens hastighed ($\vec{v}$) hhv. position ($\vec{x}$) i tiden $t_i$ være givet ved:

$$v(t_{i+1}) = v(t_i) + a(t_i) * (t_{i+1} - t_i)$$

$$x(t_{i+1}) = x(t_i) + v(t_i) * (t_{i+1} - t_i).$$

Eftersom ingen af de tre integratorer er perfekte, og da det modtagne signal ikke indeholder nogen støj, da er det kun den valgte integrator der vil være fejlkilde til resultatet af den dobbelte integration (se afsnittet om usikkerhed på side 32). Fejl der opstår under dobbelt numerisk integration har som nævnt den uheldige egenskab, at de akkumulerer.

Vi kan konkludere, at så længe vi kender accelerationen som følge af ændring i hastighed; $\vec{a}$, da kan vi også estimere controllerens hastighed og position. Præcisionen af estimatet vil variere efter den numeriske integrations-metode der benyttes, og forudsætter også en acceptabel præcision i Wii-controllerens udlæsning af accelerometeret og efterfølgende intern behandling af disse data. Eksempelvis er det vigtigt at kalibreringen (se afsnit 2.2.3, side 5) af accelerometeret fungerer korrekt, således at den målte acceleration ikke konsekvent hælder i en given retning.

Hvis vi ikke direkte kender accelerationen som følge af ændring i hastighed; $\vec{a}$, da er vi nødt til først at finde denne.

### 4.3.4 Det generelle tilfælde

I en realistisk situation, hvor Wii-controlleren og accelerometeret bliver påvirket af forskellige kilder, kan vi ikke længere antage, at vi kender $\vec{g}$, eller at $\vec{c} = 0$. Dette gør, at vi ikke umiddelbart kender $\vec{a}$ ud fra vores inddata $\vec{a}_{input}$, og vi er derfor nødt til at estimere $\vec{a}$ for at kunne estimere controllerens positionen.

Da tyngdekraften påvirker Wii-controllerens accelerometer på alle tre akser kan vi, i de fleste tilfælde, ikke være sikre på, hvordan tyngdekraften er fordelt i forhold til brugerens påvirkning af accelerometeret. Således bliver vi nødt til først at estimere $\vec{g}$ sådan, at vi senere kan anvende dette estimat til at estimere summen af $\vec{a}$ og $\vec{c}$.

For at estimere $\vec{g}$ kan vi benytte algoritmen ESTIMATEGRAVITY (se Algoritme 1, side 26) med $\vec{a}_{input}$ som argument. Dette vil give os et estimat, $\vec{g}'$, af accelerationen som følge af tyngdekraften:

$$\vec{g}' = \text{ESTIMATEGRAVITY}(\vec{a}_{input}).$$

Da $\vec{g}' \approx \vec{g}$ kan vi nu trække $\vec{g}'$ fra $\vec{a}_{input}$, og resultatet vil være et estimat af af summen af $\vec{a}$ og $\vec{c}$:

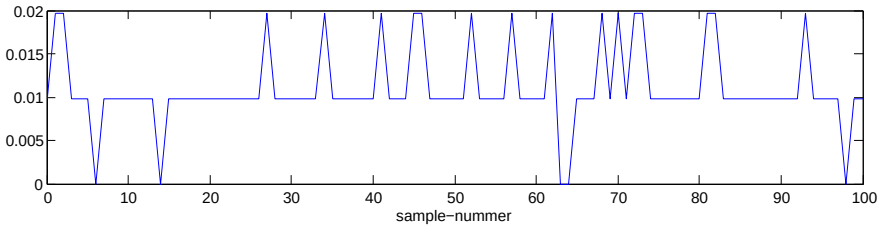
$$\vec{a}_{input} - \vec{g}' = \vec{a}' + \vec{c}'$$

hvor $\vec{a}'$ er et estimat af accelerationen som følge af ændring i hastighed; $\vec{a}$, og $\vec{c}'$ er et estimat af støjen $\vec{c}$.

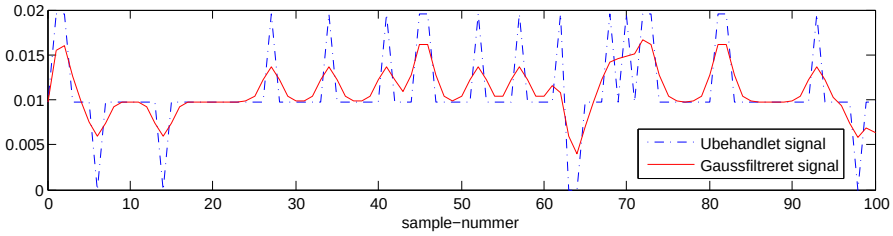
Eftersom vi ønsker at finde frem til $\vec{a}'$ kan vi udnytte, at vi nu kender værdien af summen af $\vec{a}' + \vec{c}'$. Hvis vi på en eller anden måde kan udlede eller mindske værdien af $\vec{c}'$, så kan vi også finde $\vec{a}'$. Vi ved desuden at $\vec{c}$ er et estimat af $\vec{c}$, hvorfor vi måske kan sige noget om $\vec{c}'$ ud fra egenskaberne ved $\vec{c}$.

Vi kan karakterisere $\vec{c}$ ved, at kigge på det signal, der sendes fra Wii-controlleren, når denne er i stilstand således, at den kun påvirkes af tyngdekraften. Figur 18 viser et plot af 100 samples, over x-aksen, fra Wii-controlleren, i stilstand. Vi har valgt x-aksen, da denne akse har højst præcision (se afsnit 2.2.5, side 8). Af dette plot ses det, at støjen er højfrekvent og har omtrent samme værdi omkring basis for signalet, samt at støjen ligger ujævnt fordelt over signalet. Det er derfor ikke muligt, at udlede støjen på en effektiv måde, og vi kan derfor ikke isolere det rene signal.

På baggrund af ovenstående observationer om støjen mener vi derfor, at et lavpasfilter, vil være et godt valg for, at mindske støjen i signalet, eftersom lavpasfilteret lader de lave frekvenser passere. Vi mener, at Gaussfilteret er et godt valg, da dette ikke




**Figur 18:** „Optagelse“ af 100 samples over x-aksen, hvor Wii-controlleren ligger normalt på en fast vandret flade.




**Figur 19:** 100 samples fra accelerometerets x-akse ubehandlet og filtreret med et Gaussfilter med $\sigma = 1$.

decideret ødelægger det originale signal, men forsøger at udglatte støjen. Gaussfiltrering er desuden hurtig nok til, at det ikke vil påvirke ydelsen af positionsestimeringen uacceptabelt.

For at finde et Gaussfilter, med en passende standardafvigelse, har vi prøvet, at filtrere det „optagede“ signal fra accelerometerets x-akse, med forskellige standardafvigelser, og er kommet frem til resultatet i Figur 19, hvor vi har benyttet en standardafvigelse på 1 (hvilket svarer til en filterlængde på 7).

Lad $F$ være et Gaussfilter med standardafvigelse på 1. Vi kan da benytte $F$ til, at approksimere $\vec{a}'$ ved

$$F(\vec{a}' + \vec{c}') \approx \vec{a}'.$$

Da estimatet af $\vec{a}'$ baserer sig på tilnærmelser af andre værdier er der en del usikkerheder, som kan være årsag til store eller små afvigelser i estimatet af Wii-controllerens position ud fra $\vec{a}'$, i forhold til estimatet af controllerens positionen i det ideelle tilfælde, hvor vi netop kender $\vec{a}$. Disse usikkerheder er

1. estimering af $\vec{g}$, accelerationen som følge af tyngdekraften. Muligvis den værste fejlkilde, eftersom det er umuligt i alle tilfælde, at bestemme, med fuld sikkerhed, hvordan $\vec{g}$ er fordelt på accelerometerets tre akser (se afsnit 4.2.4, side 21). Således kan påvirkningen fra tyngdekraften ødelægge alle muligheder for, at give en realistisk gengivelse af brugerens påvirkning af Wii-controlleren, hvis accelerationen som følge af tyngdekraftens påvirkning estimeres til, at være i en anden retning end den reelt er;

2. støj på signalet og bortfiltrering af denne, da det eneste Gaussfilteret gør er, at

udglatte signalet så det er mere jævnt, og således ikke fjerner den ekstra information støjen tilfører signalet;

3. den dobbelte integration, da eventuelle fejl vil akkumulere og således forstærke enhver afvigelse fra det originale signal. Problemet opstår også i det ideelle tilfælde, men skal her ses specielt i forhold til, at integrationen udføres på et estimat $\vec{a}$', hvilket vil bidrage indirekte til fejlen.

Vi kan konkludere, at når vi er kommet frem til en tilnærmet værdi for den acceleration, $\vec{a}'$, som brugeren påfører Wii-controlleren, da kan vi, som tidligere beskrevet (se afsnit 4.3.3, side 33), estimere controllerens relative position ved integrere $\vec{a}'$ to gange.

Problemet med metoden er, at vi beregner et estimat baseret på et estimat: I det ideelle tilfælde kendte vi $\vec{a}$, og vi estimerede således positionen ud fra den kendte acceleration, hvorfor fejlen opstod udelukkende i den numeriske integrator. I dette tilfælde estimerer vi derimod positionen ud fra $\vec{a}'$, estimatet af $\vec{a}$, hvorfor fejlen opstår både som følge af forskellen mellem $\vec{a}'$ og $\vec{a}$, og i den numeriske integrator.

Forskellen mellem $\vec{a}'$ og $\vec{a}$ vil typisk bevirke, at integratoren anvendt på $\vec{a}'$ beregner en hastighed og en position, som reelt set ikke forekommer, og eftersom hastigheden akkumulerer ved integration til position vil dette betyde, at der opstår drift (se afsnit 4.3.2, side 32). Med andre ord vil den estimerede position blive ustabil.

For at undgå vedvarende ustabilitet i positionen kan man dog introducere en simpel begrænsning, der forudsætter at Wii-controlleren anvendes håndholdt: Eftersom controlleren i da tilfælde ikke kan bevæges over ubegrænsede afstande, da kan vi definere en begrænsning i form af en periode $t$, der angiver i hvor tid lang brugeren typisk vil være i stand til at holde controlleren i bevægelse med konstant hastighed. Begrænsningen udføres da, såfremt hastigheden har været konstant i en længere periode end $t$, ved at nulstille estimatet af controllerens hastighed, og næste integrations-skridt vil således tage udgangspunkt i en hastighed på $0$, hvilket stabiliserer positionen.

# 5 Implementering

I dette afsnit beskriver vi opbygningen og virkemåden af klassebiblioteket, *WiiLib*, der kan benyttes af en applikation til kommunikation med Wii-controlleren. WiiLib indeholder blandt andet en implementation af de gennemgåede metoder til estimering af tilt (se afsnit 4.2, side 15) og estimering af position (se afsnit 4.3, side 30). Bemærk at afsnittet ikke har til formål at dokumentere selve koden i detaljer, eftersom denne er vedlagt som bilag, men derimod at give et overordnet indtryk af hvordan klassebiblioteket fungerer. Vi beskriver derudover også en baseline implementation af en udvidelse til 3DOT, der gør brug af WiiLib.

Til implementering har vi benyttet programmeringssproget `C++`. Vi har primært arbejdet på Windows-platformen, hvorfor vi har gjort brug af `C++`-compileren i Visual Studio 2005. Det burde være trivielt at oversætte koden med en anden compiler, f.eks. GCC, men på grund af tekniske komplikationer med Bluetooth under Linux understøtter WiiLib's `HID`-klasse (se afsnit 5.2.1, 38) indtil videre kun Windows.

Først beskrives hvilke konventioner vi har bestræbt os på at overholde for koden. Herefter beskrives WiiLib, og til sidst beskrives udvidelsen til 3DOT.

## 5.1 Konventioner for kode

Vi overholder, i det omfang vi har fundet det praktisk, de samme konventioner for vores kode, som der benyttes i 3DOT-projektet [3DOT, 2007].

Alle variabelnavne starter med et lille bogstav. Hvis et variabelnavn består af flere ord er disse konkateneret, så de ikke er adskilt af noget tegn, og første bogstav i alle ord, undtaget det første ord, er med stort begyndelsesbogstav. For eksempel `int birdCount`.

Alle membervariable starter med lille „m“ efterfulgt af variabelnavn med stort begyndelsesbogstav. For eksempel `mCount`.

Alle klassenavne starter med et stort bogstav og følger ellers konventionerne for variabelnavne. For eksempel `class CorrectClass`.

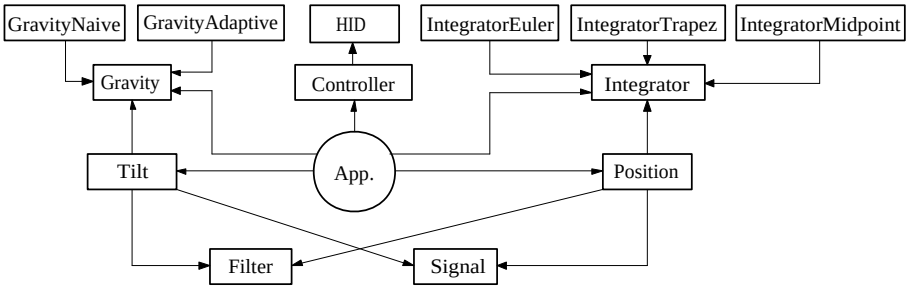
Konstanter og defines skrives med store bogstaver og såfremt de består af flere ord er disse adskilt af underscores. For eksempel `#define ARRAY_SIZE 22` og `const int ARRAY_SIZE = 22`.

Pointeroperatorer skrives uden adskillelse fra det variabelnavn de anvendes i forbindelse med. For eksempel `int *cnt` og `&cnt`.

Paranteser skrives uden adskillelse fra deres indhold, og uden adskillelse fra funktionsnavne, men med et mellemrum fra `if`- og `for`-statements, m.v. For eksempel `write(buffer, bufferSize)` og `for (int i=0; i<count; i++)`.

## 5.2 WiiLib

Klassebiblioteket WiiLib er den del af implementeringen, som afsnittene Teknisk baggrund samt Analyse har lagt op til. Det er således i WiiLib, at vi faciliterer kommunikation med Wii-controlleren, samt forskellige former for behandling af data fra Wii-controlleren. En vilkårlig host-applikation kan på denne vis gøre brug af klasser i WiiLib til, for eksempel, at estimere Wii-controllerens orientering og position.




**Figur 20:** Diagram over klasse-sammenhænge i WiiLib.

Da WiiLib er et klassebibliotek, og ikke en selvstændig applikation, er WiiLib dog ikke selv i stand til, at koordinere kommunikationen med Wii-controlleren. Dette kræver at host-applikationen jævnligt, og med tilstrækkeligt korte intervaller, instruerer WiiLib i aflæsning og behandling af data fra Wii-controlleren således, at der er samples nok til at give acceptable tilt- og positions-estimater.

WiiLib består af en samling af par af `C++` headerfiler (`.h`) og tilhørende `C++` implementationsfiler (`.cpp`). Filerne er navngivet alt efter hvilke klasser de repræsenterer. En applikation gør brug af WiiLib ved at inkludere den overordnede headerfil `WiiLib.h`, som automatisk sørger for at inkludere de resterende headerfiler.

Alle klasser i WiiLib er lagt under samme *namespace*; `WiiLib`, og det er derfor tilrådeligt at importere dette namespace såfremt man ønsker at benytte WiiLib i sin applikation. Man kan dog også anvende klasser i WiiLib uden at importere WiiLibs namespace – i da tilfælde skal de blot refereres til som `WiiLib::<klassenavn>`.

For at lette beskrivelsen af klasserne i WiiLib tager vi udgangspunkt i Figur 20, som viser relationerne mellem de vigtigste klasser i biblioteket.

De efterfølgende underafsnit beskriver de vigtigste klasser i WiiLib. Hjælpeklasser, såsom vektorer og matricer, beskrives undervejs (hvor de bliver brugt), da vi ikke finder dem vigtige nok til, at blive fremhævet i seperate afsnit.

### 5.2.1 `HID`

`HID`-klassen implementerer kommunikation med Wii-controlleren gennem Bluetooth og HID-protokollen. Klassen giver mulighed for at forbinde til en Wii-controller, og at sende og modtage data mellem denne og en computer. Den udbyder desuden mulighed for at lukke forbindelsen, og for at kontrollere om en Wii-controller er forbundet.

Ved forbindelse til en Wii-controller er det nødvendigt, at angive VendorID og DeviceID for denne, men da Wii-controllerens specifikke værdier for disse er vedlagt, så kan der let forbindes til en Wii-controller. Ved forbindelse til andre typer enheder kan funktionaliteten af klassens funktioner ikke garanteres at være korrekt.

På grund af store forskelle i den måde forskellige Bluetooth-stakke og deres A-PI'er er implementeret på understøtter `HID`-klassen kun Windows-platformen. Yderligere understøtter vi kun Bluetooth-stakken der følger med Windows, samt BlueSoleil [IVT corporation, 2007]. BlueSoleil er en alternativ Bluetooth-stak, som understøtter størstedelen af eksisterende Bluetooth radio-enheder, og som udbyder omtrent samme

API som den, i Windows, indbyggede stak.

Der er dog nogle implementeringsfejl i BlueSoleil, som gør, at man ikke kan skelne mellem denne stak og Windows-staken ved run-time, og derfor har vi indført et compile-time flag, der angiver om BlueSoleil-stakken skal benyttes (se `HIDconfig.h` i Bilag C.1 på side 67). Således er det nødvendigt, at have en version af WiiLib der er prækonfigureret til den specifikke Bluetooth-stak man benytter.

Understøttelsen af disse Bluetooth-stakke gør, at WiiLib er afhængig af eksterne filer fra Windows DDK. Disse filer er `hidsdi.h`, `hidpi.h`, `hidusage.h`, `setupapi.h`, biblioteket `setupapi.lib`, samt biblioteket `hid.dll`.

Når der oprettes forbindelse til en Wii-controller sættes Bluetooth-stakkens bufferstørrelse til 2, hvilket medfører en stor risiko for pakketab, da bufferen hurtigt fyldes op, men sikrer, at det altid er de nyeste pakker der læses fra bufferen.

I implementeringen er der ikke taget højde for, om forbindelsen til en Wii-controller mistes, f.eks. som følge af flade batterier eller at controlleren bliver ført uden for rækkevidden af Bluetooth radioen. Det er således ikke muligt, at benytte en Wii-controller efter forbindelsen er blevet brudt, uden at lukke forbindelsen helt og åbne den igen.

`HID`-klassen er baseret på tilsvarende klasse, `cHIDDevice`, i *public domain*-projektet cWiiMote [Forbes, 2006]. Koden er vedlagt i Bilag C.1 på side 67.

### 5.2.2 `Controller`

`Controller`-klassen (se Bilag C.1 på side 72) benytter `HID`-klassen til, at implementere kommunikation vha. Wii-controllerens indbyggede protokol 2.2.2. `Controller`-klassen har således defineret alle benyttede konstanter for rapport ID, knap-masker osv. Klassen udbyder funktioner til, at åbne og lukke forbindelsen til en Wii-controller, til at kalibrere en forbundet Wii-controller, til at ændre controllerens rapport-tilstand, ændre status for rumble, tænde og slukke lysdioderne, samt at aflæse data fra controlleren.

Klassen benytter en datastruktur, der indeholder knap-data, den målte acceleration på alle akser, den normaliserede acceleration på alle akser, en vektor med accelerationen på akserne angivet korrekt i forhold til controllerens naturlige orientering (se afsnit 4.1 på side 14), samt antallet af cycles CPU'en har afviklet siden forrige pakke fra Wii-controlleren.

Den nævnte vektor er en simpel implementation (se Bilag C.1 på side 99), som udbyder de mest anvendelige operationer på vektorer, samt en funktion der returnerer vinklen mellem to vektorer og hvordan denne er fordelt på de tre akser.

Ved hjælp af den sidste værdi, i den nævnte datastruktur, samt CPU'ens klokfrekvens (antal cycles pr. sekund) er det muligt, at udregne den præcise tid, $\Delta t$, mellem den forrige og den nye pakke. Da alle CPU'er ikke har samme klokfrekvens er det op til applikationen, at foretage denne udregning ud fra klokfrekvensen af den CPU applikationen afvikles på.

Antallet af cycles siden sidste pakke kan udregnes ved, at læse værdien af en `RDTSC`-instruktion i CPU'en for hver pakke og trække denne værdi for den forrige pakke fra den tilsvarende for den nye pakke. Dette kræver at CPU'en understøtter denne instruktion. `RDTSC` eller ReaD Time-Stamp Counter indeholder antallet af cycles, CPU'en har afviklet siden computeren blev tændt, og kan benyttes til en højopløselig timer, hvis man kender CPU'ens clockfrekvens, og er meget benyttet inden for computerspil-verdenen.

Når applikationen har oprettet forbindelse til en Wii-controller, gennem WiiLib, sørger `Controller`-klassen for, at kalibrere controlleren automatisk. På denne måde er det sikkert, at data der modtages fra controlleren er kalibreret.

Den nok vigtigste funktion i `Controller`-klassen er `Sample`, der sørger for, at læse og behandle en pakke fra Bluetooth-stakkens buffer. Funktionen kontrollerer rapport ID'et for den læste pakke og sørger for, at behandle den læste data korrekt. Det er op til applikationen, der benytter WiiLib, at kalde `Sample` tilstrækkeligt ofte, så der modtages nok samples fra Wii-controlleren til, at der kan foretages acceptable tilt- og positions-estimater.

Efter hvert kald til `Sample` findes de nyeste data i variablen `mStatus`. Applikationen kan således til enhver tid læse de nyeste data fra klassen gennem variablen `mStatus`.

Vi benytter denne form for timer, da vi ikke fandt de indbyggede timere, i Windows' API, tilstrækkelige. Timerne i Windows' API angav $\Delta t$ til at være mellem $0$ og $30$ millisekunder. Ved at aflæse data fra Wii-controlleren så hurtigt som muligt, har vi konstateret, at controlleren ikke sender pakker oftere end hvert $10$. millisekund. Derfor finder vi denne måling af $\Delta t$ urealistisk, da både $0$ og mere end $20$ millisekunders interval mellem pakkerne er for stor afvigelse i forhold til det reelle interval.

Det viser sig dog, at `RDTSC`, på trods af sin højere opløsning, ikke giver et bedre resultat. Vi anbefaler derfor ikke, at applikationer lægger alt for stor vægt på $\Delta t$, men i stedet antager $\Delta t$ ud fra hvor ofte der udføres kald til `Sample`.

### 5.2.3 `Filter`

`Filter`-klassen beskriver et vilkårligt en-dimensionelt foldningsfilter. Klassen instantieres med en filterkerne, en værdi der beskriver længden af filterkernen, samt et „centerindex“, der angiver midten af filterkernen ved foldning. Klassen anvendes internt i forbindelse med `Signal`-klassen, men typisk vil det være op til applikationen at udvælge eller konstruere filteret førend det kan anvendes som en parameter til andre klasser, som f.eks. `Tilt`-klassen og `Position`-klassen.

Klassen indeholder derfor også en række statiske funktioner, der kan benyttes til generering af forskellige typer filtre. Eksempelvis er det muligt at generere et Gaussfilter ved blot at kalde den statiske funktion `Filter::CreateGaussFilter`($\sigma$), der er implementeret ud fra den en-dimensionelle Gaussfunktion:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

men applikationen kan dog også vælge at konstruere sine helt egne filterkerner, hvilket i den forstand gør klassen meget fleksibel.

Kildekoden er vedlagt i Bilag C.1 side 81.

### 5.2.4 `Signal`

`Signal`-klassen beskriver et (udsnit af et) en-dimensionalt signal, hvor signalets længde begrænses af længden af en intern cyklisk buffer; en såkaldt *ringbuffer*. At bufferen er cyklisk gør, at nye samples kan tilføjes signalet i konstant tid, hvilket er yderst praktisk eftersom vi, per signal, ønsker at behandle op til 100 samples per sekund (se afsnit 2.2.2, side 5).

Typisk vil en applikation indirekte anvende op til 6 samtidige instanser af `Signal`-klassen; 3 i `Tilt`-klassen, samt 3 i `Position`-klassen, hvilket resulterer i tilføjelse og behandling af op til 600 samples per sekund.

Behandling af samples der tilføjes et signal sker på baggrund af en binding mellem signalet og en instans af `Filter`-klassen. Signalet indeholder en funktion `SetFilter` til binding af et filter, samt en funktion `Convolve` der folder en kopi af den interne ringbuffer med det nuværende bundne filter.

Klassen udbyder således, på baggrund af det bundne filter, forskellig funktionalitet til udtrækning af rå hhv. filtrerede værdier fra det rå signal hhv. filtrerede signal, hvor det filtrerede signal beregnes automatisk i det øjeblik der efterspørges en filtreret værdi, og alt efter hvilket filter der er bundet til signalet.

Kildekoden er vedlagt i Bilag C.1 side 83.

### 5.2.5 `Gravity`

`Gravity`-klassen specificerer hvilken funktionalitet der skal til for at estimere $\vec{g}$, accelerationen som følge af tyngdekraften, ud fra den målte acceleration $\vec{a}_{input}$. Klassen indeholder en *pure virtual* funktion, og kan derfor ikke instantieres direkte, men beskriver i stedet kun kravet til en eventuel implementation. Mere specifikt skal enhver klasse der ønsker at nedarve fra `Gravity` implementere funktionen `GetEstimate`($\vec{a}_{input}$), således at denne returnerer et estimat af $\vec{g}$.

Grunden til strukturen er, at vi som parameter til `Tilt`-klassen ønsker at kunne angive en vilkårlig metode til estimering af $\vec{g}$. Eksempelvis kunne det tænkes, at det i nogle situationer er rigeligt at benytte en metode der beregner et meget naivt estimat, hvor det i andre situationer er bedre at benytte en metode der beregner et adaptivt estimat. Det kunne også meget vel tænkes, at man i stedet ønskede at benytte en helt tredje metode til beregning af estimatet, men stadig som parameter til klassen til estimering af tilt.

Dette opnås ved netop at gøre brug af nedarvning og virtuelle funktioner, da parameteren til `Tilt`-klassen blot kan angives som værende af typen `Gravity`, hvorefter alle implementationer af `Gravity` vil være gyldige argumenter. Dette kaldes også dynamisk polymorfi, og gør det med andre ord muligt, i applikationen, at implementere en alternativ metode til estimering af $\vec{g}$, for derefter at lade `Tilt`-klassen benytte denne uden at det kræver ændringer i WiiLib.

I WiiLib har vi på forhånd implementeret to metoder der begge nedarver fra `Gravity`-klassen. Den første implementation, `GravityNaive`, er baseret på en naiv metode som blot normaliserer dets inddata. Den anden anden implementation, `GravityAdaptive`, er baseret på den adaptive metode, som vi beskrev i analyseafsnittet 4.2.5, side 24.

Kildekoden til grundklassen og begge implementationer er vedlagt i Bilag C.1 side 87.

### 5.2.6 `Integrator`

`Integrator`-klassen er grundklassen for alle numeriske integratorer (se afsnit 4.3.2, side 4.3.2) i WiiLib. På samme måde som `Gravity`-klassen indeholder klassen en pure virtual funktion `Integrate`($\vec{a}$, $\Delta t$), hvilket medfører at klassen ikke direkte

kan instantieres, men i stedet skal implementeres (eller med andre ord nedarves fra) i en anden klasse.

Udover den virtuelle funktion indeholder `Integrator`-klassen også tre membervariable; `mA`, `mV`, og `mX`, der efter hvert kald til den virtuelle funktion `Integrate` skal opdateres med nye værdier for hhv. accelerationen, hastigheden, og positionen. `Integrate`-funktionen har således til formål at dobbelt-integrere accelerationen $\vec{a}$ i forhold til $\Delta t$, for derefter at opdatere klassens tre membervariable.

Vi har implementeret tre forskellige numeriske integratorer; `IntegratorEuler`, `IntegratorTrapez`, samt `IntegratorMidpoint`, der alle nedarver fra `Integrator`-klassen, men eftersom `Position`-klassen gør brug af en vilkårlig implementation af `Integrator`-klassen er det dog helt og aldeles muligt, at lade applikationen definere en alternativ numerisk integrator til brug ved positionsestimering.

Dette er specielt interessant med henblik på senere udvidelser, da der som nævnt (se afsnit 4.3.2, side 32) findes mere præcise numeriske integratorer end lige netop de tre; Euler-integration, Trapez-integration og Midpoint-integration, som vi har valgt at implementere.

Kildekoden til grundklassen og de tre implementationer er vedlagt i Bilag C.1 side 93.

### 5.2.7 `Tilt`

`Tilt`-klassen (se Bilag C.1 på side 90) implementerer en tiltestimator, der vha. `Filter`- og `Gravity`-klasserne kan udlede Wii-controllerens tilt, i axis-angle repræsentation, for en given sample. Det er op til applikationen, at benytte `Tilt`-klassens `ProcessSample`-funktion på de samples, som den vil have et estimat for.

Det er muligt, at angive reference-orienteringen af Wii-controlleren, i form af en `Vec3`-vektor, således, at man kan „nulstille“ controlleren til en bestemt position. Det er desuden muligt, at angive et nyt filter og en ny gravityestimator efter at `Tilt`-klassen er instantieret, således at man kan ændre disse efter skiftende forhold.

`Tilt`-klassen benytter `Signal`-klassen til at filtrere hver akse. Når et nyt filter vælges til tiltestimering opdateres instansen af `Signal`-klassens bufferstørrelse, så den reflekterer det nye filter, og `Signal`-instansen sættes til at benytte det nye filter. Dette sker for hver af de tre akser.

Applikationen kalder klassens funktion `ProcessSample`($\vec{a}_{input}$). `ProcessSample` er en implementation af det generelle tilfælde for estimering af tilt (se afsnit 4.2.4, side 21), og benytter 3 instanser af `Signal`-klassen til, at filtrere den nye sample $\vec{a}_{input}$, hvorefter $\vec{g}'$, estimatetet af accelerationen som følge af tyngdekraften, beregnes vha. instansen af `Gravity`-klassen. Normalen for Wii-controlleren beregnes da ud fra $\vec{g}'$, og da kan axis-angle repræsentationen beregnes ud fra normalen og reference-orienteringen.

Nu kan applikationen kalde klassens funktioner `GetGravityEstimate`, `GetNormalEstimate` og `GetAxisAngle` for, at få hhv. estimatet af accelerationen som følge af tyngdekraften, estimatet af normalen til Wii-controlleren og tiltestimatet, som alle lægges ind i et variabelt argument af typen `Vec3`.

### 5.2.8 `Position`

`Position`-klassen (se Bilag C.1 på side 97) implementerer en positionsestimator, der vha. `Filter`- og `Integrator`-klasserne kan udlede hastigheden og den relative position for hver ny sample. Det er, ligesom med `Tilt`-klassen, op til applikationen, at benytte `Position`-klassens `ProcessSample`-funktion på hver sample for, at få et acceptabelt positions-estimat.

Når applikationen skal have positionsestimatet af en ny sample skal applikationen både angive $\vec{a}_{input}$, den målte acceleration, og $\vec{g}'$, estimatet af accelerationen som følge af tyngdekraften; begge i form af en `Vec3`-vektor, samt $\Delta t$, tiden mellem den forrige og den nye sample. Denne opbygning gør det muligt, at angive en anden acceleration som følge af tyngdekraften, end den estimerede, hvis man f.eks. kender den præcise fordeling af tyngdekraften på de tre akser.

Det er desuden muligt for applikationen, at skifte filter og integrator efter `Position`-klassen er instantieret således, at man kan ændre disse efter skiftende forhold. `Position`-klassen benytter `Signal`-klassen til at filtrere hver akse. Når et nyt filter vælges til positionsestimering opdateres instansen af `Signal`-klassens bufferstørrelse, så den reflekterer det nye filter, og `Signal`-instansen sættes til at benytte det nye filter. Dette sker for hver af de tre akser.

Applikationen kalder klassens funktion `ProcessSample`($\vec{a}_{input}$,$\vec{g}'$,$\Delta t$). `ProcessSample` er en implementation af det generelle tilfælde for estimering af position (se afsnit 4.3.4, side 34), og benytter $\vec{g}'$ til at finde et estimat af $\vec{a}$, Wii-controllerens reelle acceleration, hvorefter estimatet af $\vec{a}$ filtreres ved 3 instanser af `Signal`-klassen. Til sidst benyttes integratoren til at udlede den relative hastighed og position, givet den filtrerede acceleration og $\Delta t$.

Nu kan applikationen aflæse Wii-controllerens relative hastighed og position via. klassens funktioner `GetVelocity` og `GetPosition`, der begge lægger resultatet ind i et variabelt argument af typen `Vec3`.

## 5.3 Udvidelse til 3DOT

Vores baseline implementation af en udvidelse til 3DOT består af 3 klasser: En *extension*, en *task* (der afvikles af *scheduleren*), samt et *entity plugin* (der kan tilknyttes entities i en scene). Vi har døbt klasserne `WiiControllerExtension` for extension-klassen, `WiiControllerTask` for task-klassen, samt `WiiControlled` for entity plugin-klassen.

Som tidligere beskrevet (se afsnit 2.3, side 9) er det extension-klassen der er udgangspunktet for udvidelsen som en helhed, da det er denne som sørger for, at udbyde udvidelsens entity plugin til entities i en scene, samt at registrere udvidelsens task i 3DOT's scheduler. Se desuden Figur 8 for en illustration af sammenhængene mellem komponenter i 3DOT.

De efterfølgende underafsnit beskriver opbygningen og virkemåden af de 3 klasser.

### 5.3.1 `WiiControllerExtension`

`WiiControllerExtension`-klassen implementerer en extension, der udbyder en task; `WiiControllerTask`-klassen, samt et entity plugin; `WiiControlled`-klassen. Derudover har extension-klassen også til formål at håndtere og sørge for al

kommunikation med Wii-controlleren. Klassen gør derfor direkte brug af WiiLib, og indeholder således flere membervariable, der er instanser af klasser i WiiLib (se afsnit 5, side 37), heriblandt en instans af `Controller`-klassen og en instans af `Tilt`-klassen.

`WiiControllerExtension`-klassen udbyder desuden forskellige *event handles*, således at andre objekter i 3DOT-miljøet kan koble sig på klassen og modtage *events*, f.eks. når Wii-controllerens orientering ændrer sig, eller når der bliver trykket på en knap på Wii-controlleren. De udbudte event handles er:

- `ButtonStatesChangedEvent`, affyres når knappernes status ændrer sig.
- `OrientationChangedEvent`, affyres når orienteringen ændrer sig.
- `PositionChangedEvent`, affyres når positionen ændrer sig.

Specielt `WiiControlled`-klassen gør brug af disse event handles til at „lytte efter“, om der er sket ændringer i controllerens orientering eller position. Typisk vil en lytter implementere en *event handler* til at håndtere et enkelt event. Se eksempelvis `mOrientationChangedEventHandler` samt `OnOrientationChanged` i `WiiControlled.cpp` (Bilag C.5, side 123).

Den vigtigste funktion i extension-klassen er `Process`, der har til formål at instruere `Controller`-klassen, i WiiLib, i at sample data fra Wii-controlleren, samt at anvende disse samplede data til estimering af tilt og position. Estimering af tilt og position sker ved brug af de dertil indrettede klasser fra WiiLib, og `Process` sørger således blot for at lede og fordele arbejdet, ganske som enhver anden applikation der gør brug af WiiLib.

I tilfælde af at de behandlede data, f.eks. estimaterne af controllerens orientering og position, ændrer sig mellem to kald til `Process`, da sørger funktionen for at „affyre“ relaterede events således, at alle objekter der lytter på de pågældende event handles får besked om ændringen.

`Process` sørger desuden for at forsøge at genforbinde til Wii-controlleren, skulle forbindelsen gå tabt.

### 5.3.2 `WiiControllerTask`

`WiiControllerTask`-klassen implementerer en scheduler task, der bliver registreret i 3DOT's scheduler ved instantiering af `WiiControllerExtension`-klassen. `WiiControllerTask`-klassen implementerer desuden en funktion, `Execute`, der sørger for at kalde funktionen `Process` i `WiiControllerExtension`-klassen og er således andet led i kaldet til funktionen `Sample` i WiiLib (se 5.2.2, side 39).

Første led er selve scheduleren, der sørger for afvikle alle tasks én gang for hver frame. Det er således scheduleren der indirekte bestemmer hyppigheden for læsning af pakker fra Wii-controlleren, og det er derfor vigtigt at have en framerate, der ligger så tæt på 100 som muligt, for at opnå gode tilt- og positions-estimater. For at mindske denne risiko har vi sat bufferstørrelsen for Bluetooth-stakken til 2 pakker, således at det altid er den nyeste bevægelse der behandles i WiiLib.

### 5.3.3 `WiiControlled`

`WiiControlled`-klassen implementerer et entity plugin, der bliver registreret i 3DOT ved instantiering af `WiiController`-klassen. `WiiControlled`-klassen indeholder blandt andet to funktioner, der registreres som *event handlers* til de to event handles; `OrientationChangedEvent` og `PositionChangedEvent`, og påvirker vha. disse den entity der er bundet til entity plugin'et, når disse events aktiveres.

Entity plugin'et udbyder desuden en række metoder til den bundne entity, som gør det muligt, at angive hvilke af de to events der reelt skal følges op på, således at den bundne entity kan reagere på enten begge events, et af de to events, eller ingen af dem. På denne måde kan man vha. 3DOT's level-editors brugerinterface styre, per entity, hvordan den bundne entity skal påvirkes af Wii-controlleren, eftersom brugerinterfacet automatisk genererer context-menuer med alle bundne funktioner. Hvis der, for eksempel, er specificeret i den bundne entity at entity'en skal påvirkes af Wii-controllerens orientering, da sørger entity plugin'et for at påvirke orienteringen af den bundne entity, hver gang `Process`-funktionen i `WiiControllerExtension`-klassen affyrer det relaterede event handle.

Ved hjælp af den udvidelse af brugerinterfacet, som entity plugin'et medfører, er det muligt for brugeren af 3DOT's level-editor, at binde en instans af `WiiControlled`-klassen til en entity, og angive om denne skal påvirkes af Wii-controllerens orientering, position, orientering og position, eller ingen af delene.

# 6 Afprøvning

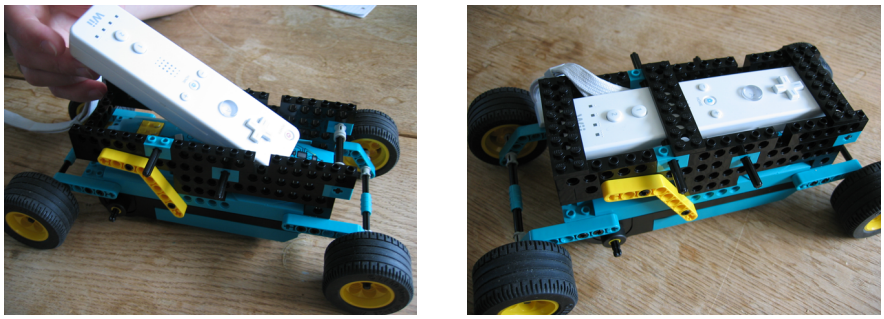
Succeskriteriet for vores implementation er, at den lader Wii-controlleren fungere til operationer i et virtuelt rum med acceptabel præcision, hvor operationer er enten rotationer eller translationer. Vi så tidligere hvordan tilt er udtryk for en rotation, og det er indlysende at skift i position oversætter direkte til translation. For at afprøve vores implementation er vi derfor interesserede i, at finde ud af hvor præcist vi kan estimere Wii-controllerens tilt, samt hvor præcist vi kan estimere dennes position.

For de komponenter i WiiLib som er baserede på eksisterende metoder har vi gjort brug af fabrikeret afprøvning i form af unittests. Ingen af vores unittests meldte fejl. Kildekoden til alle unittests er vedlagt i Bilag C.2 på side 104.

**Strategi** Det er umiddelbart svært at sige noget konkret om præcisionen af et estimat hvis man ikke har den reelle værdi at holde estimatet op imod. Vores strategi er derfor at teste præcisionen af estimaterne under kontrollerede forhold, hvilket vi vil gøre ved at konstruere de enkelte tests af estimaterne således, at de ud fra visse antagelser om inddata til estimatet kan give et mål for hvor godt estimatet nærmer sig den reelle værdi.

Eksempelvis kan vi få et mål for præcisionen af et tiltestimat, hvis vi under hele testen kender værdien af $\vec{g}$, accelerationen som følge af tyngdekraften, eftersom tiltestimatet er baseret på værdien af $\vec{g}'$, estimatet af $\vec{g}$. Et mål for præcisionen af et estimat er da eksempelvis givet ved vinklen $\theta(\vec{g}, \vec{g}')$, hvor en lavere vinkel er udtryk for et mere præcist estimat. Ligeledes kan vi få et mål for præcisionen af et positionsestimat, hvis vi har en reel position at sammenligne med, og hvis vi under hele testen kender værdien af $\vec{g}$ og derved $\vec{a} + \vec{c}$.

Problemet er i begge tilfælde, at vi kun kender værdien af $\vec{g}$ under hele testen såfremt vi ved, at Wii-controlleren ikke ændrer orientering under selve testen. Ved håndholdt brug vil controllerens orientering typisk variere en smule, også selvom den holdes relativt stille, hvorfor vi ikke kender $\vec{g}$ under hele testen hvis controlleren er håndholdt. På den anden side er det ikke særligt spændende at analysere data fra en Wii-controller der kun ligger stille.



**Figur 21:** Figuren viser vores hjemmelavede vogn til stabilisering af Wii-controlleren. Wii-controlleren placeres først i holderen, hvis længde passer med, at controlleren sættes i spænd på langs. Herefter låses den fast af tre overliggende klodser, og sættes til sidst i spænd på tværs vha. de fire justerbare stænger. Wii-controlleren stabiliseres således i alle tre akser.

For at løse dette problem har vi konstrueret en LEGO-vogn (se Figur 21) til sta-

bilisering af Wii-controlleren, som tilnærmelsesvist sikrer, at controlleren ikke ændrer orientering under testen, samt kun har mulighed for at accelerere langs en enkelt akse. At accelerationen kun sker over en enkelt akse ændrer ikke ved muligheden for, at teste præcisionen af et estimat, men gør det en smule enklere, at generere gode inddata til tests.

**Generering af datasæt** Som inddata til de enkelte tests vil vi gøre brug af forskellige datasæt, hvor hvert datasæt består af en værdi for $\vec{g}$, samt et antal samples. Idéen med dette er, at kunne sammenligne præcisionen af to forskellige metoder til beregning af et estimat, da dette bedst lader sig gøre hvis begge metoder beregner deres respektive estimat ud fra samme inddata.

Det er vigtigt at bemærke, at selvom vi kun har én reel metode til beregning af f.eks. tiltestimater (`Tilt`-klassen; se afsnit 5.2.7, side 42) i WiiLib, så er den parameteriseret således, at metoden varierer alt efter dens parametre. Man kan derfor betragte f.eks. estimering af tilt med to forskellige valg af parametre, som to forskellige metoder, eller to forskellige afarter af samme metode. Det samme gør sig gældende for positionsestimering (`Position`-klassen; se afsnit 5.2.8, side 43).

Vi genererer alle datasæt med en lille applikation (`WiiLib2MatLab`; se Bilag C.3, side 106), som for et givet $n$ aflæser $n$ samples af den målte accelerationen, og skriver disse ud i en fil (datasættet), som bl.a. kan læses af Matlab. Vi sampler altid ca. 100 gange i sekundet. Derudover sørger applikationen også for at finde en approksimativ værdi for $\vec{g}$, eftersom et mål for præcisionen kræver denne: Baseret på en antagelse om, at Wii-controlleren, over en periode på 1000 samples, holdes i samme orientering, som selve testen skal udføres i, sættes $\vec{g}$ til at være gennemsnittet af disse 1000 samples.

Alle de datasæt vi har benyttet under afprøvning kan findes på adressen: `http://qeep.dk/~volmer/bach/WiiLibTestsDatasets.tgz`.

**Udførelse af tests** Til udførelse af tests har vi skrevet en applikation (`WiiLibTests`; se Bilag C.4, side 108), der, givet et datasæt, er i stand til at udføre forskellige parameteriserede tests. Applikationen er bygget op således, at den ud fra parametre på kommandolinien udvælger en parameteriseret test og udfører denne på det angivne datasæt.

Hver test er defineret som en klasse i applikationen, og returværdien af en test er altid en liste af værdier, som er testens resultatet i forhold til hver sample i det angivne datasæt. Eksempelvis har vi skrevet en parameteriseret test til estimering af tilt, og resultatet af denne er en liste af vinkler mellem $\vec{g}$, aflæst direkte fra det angivne datasæt, og $\vec{g}'$, som udregnes per sample fra det angivne datasæt, hvor lavere vinkler er udtryk for mere præcise estimater.

Vi har valgt at udføre vores afprøvning af tilt- og positionsestimering i samme orden, som vi har behandlet tilt- hhv. positionsestimering i alle tidligere afsnit. Først udføres altså tests af estimering af tilt, og derefter udføres tests af estimering af position. Vi udfører både tests af estimering af tilt og tests af estimering af position på flere forskellige datasæt.

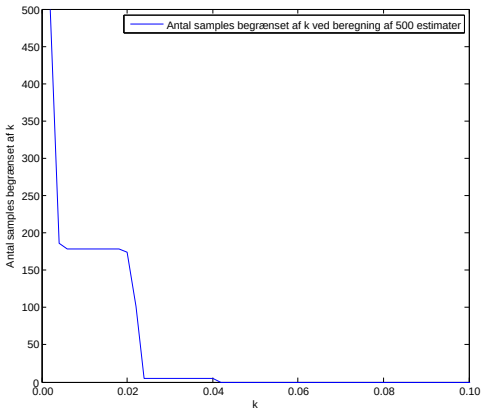
## 6.1 Tests af estimering af tilt

For at teste præcisionen af estimering af tilt har vi valgt at sammenligne vinklen mellem $\vec{g}$; den tilnærmelsesvis kendte acceleration som følge af tyngdekraften, og $\vec{g}'$; estimatet af $\vec{g}$. Et tiltestimat beregnes ud fra en enkelt sample af den målte acceleration, og præcisionen af et estimat er derfor også givet per sample. En lille vinkel mellem $\vec{g}$ og $\vec{g}'$ er således udtryk for et godt estimat for netop den givne sample.

For at kunne sammenligne med $\vec{g}$ for hver sample er vi nødt til at antage, at Wii-controlleren ikke ændrer orientering i løbet af testen. Dette bevirker, at vores test af præcisionen af tiltestimatet faktisk er en test af afvigelsen af tiltestimatet, eftersom vi ønsker at finde den metode, hvis tiltestimat $\vec{g}'$ fjerner sig mindst muligt fra den reelle værdi $\vec{g}$.

Tiltestimatet beregnes af en af de to implementerede metoder: Den naive metode, der blot normaliserer dens inddata, eller vores egen adaptive metode, som vi udviklede i analysefasen (se afsnit 4.2.5, side 24). Ved at anvende metoderne på identiske datasæt vil vi undersøge hvilken af metoderne der bedst egner sig til at estimere tilt. Den bedste metode er metoden med den mindste gennemsnitlige vinkel over alle samples i datasættet.

**Valg af $k$** Førend vi kan sammenligne de to metoder er vi nødt til først, at finde gode værdier for parametrene til den adaptive metode; $\vec{g}'_{base}$, $k$ og $\beta$. $\vec{g}'_{base}$ sættes umiddelbart til $\vec{g}$, og varierer således fra datasæt til datasæt. $k$ og $\beta$ skal derimod findes ved afprøvning.



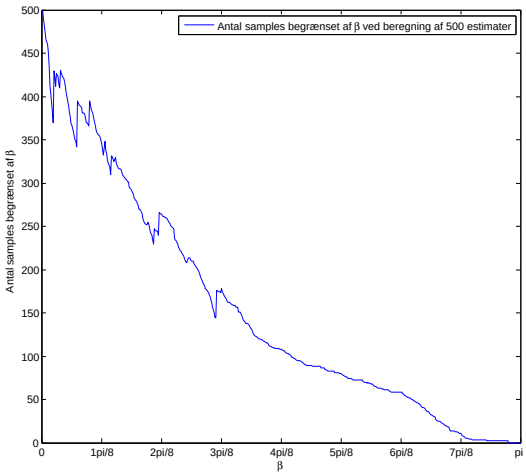

**Figur 22:** Figuren viser antallet af samples, der, som følge af $k$, ikke vægtede højt nok til, at blive taget i betragtning som kandidater til $\vec{g}'_{base}$ i den adaptive metode. Samples stammer fra det anvendte datasæt, der er en optagelse af den målte acceleration ved håndholdt stilstand af controlleren.

$k$ er værdien, som begrænser antallet af kandidater til $\vec{g}'_{base}$ ud fra deres vægt. Ved at generere et specielt datasæt (`handStill.dat`), hvori vi mener at alle samples vægter højt nok til at være kandidater til $\vec{g}'_{base}$, kan vi benytte en speciel version af den adaptive metode til at bestemme en udmærket værdi for $k$. Typisk vil det være interessant at sætte $k$ tilpas lavt til, at den adaptive metode ikke udelukker samples fra at være kandidater, bare fordi brugeren ryster en smule på hånden. Datasættet vi har

brugt i dette tilfælde er således genereret ved håndholdt brug af controlleren, hvor vi ikke har udsat controlleren for nogen form for bevidst påvirkning.

Ved at anvende den adaptive metode på det specielle datasæt, samt tælle hvor mange gange, for et givent k, den adaptive metode udelukker samples fra at være kandidat til $\vec{g}'_{base}$, har vi fundet frem til værdien 0.05 (se Figur 22). Når $k = 0.05$ udelukker den adaptive metode nemlig kun samples, hvis vægt er mindre end de samples der ikke optrådte i det specielle datasæt, hvilket også må betyde, at $k$ da kun udelukker samples hvis lave vægt følger af mere end blot ubevidste rystelser.

**Valg af $\beta$** $\beta$ er værdien, som begrænser vinklen mellem estimatet og $\vec{g}'_{base}$. Ved at generere et specielt datasæt (`handStationaryRotation.dat`), indeholdende samples af udelukkende stationær rotation af Wii-controlleren, samt anvende den fundne værdi for $k$, bør vi kunne finde en acceptabel værdi for $\beta$ på samme måde som vi finder en god værdi for $k$.

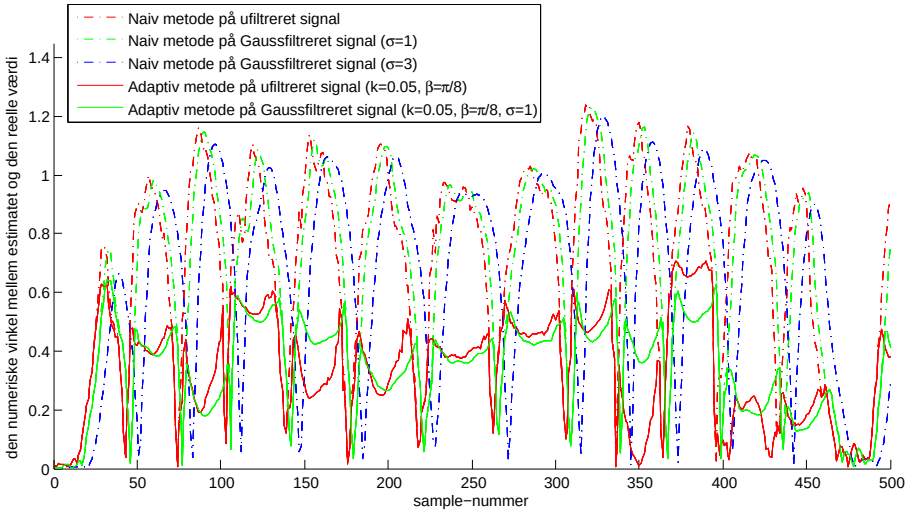



**Figur 23:** Figuren viser antallet af samples, der, som følge af $\beta$, blev begrænset i deres indflydelse på vinklen af estimatet af $\vec{g}$. Samples stammer fra det anvendte datasæt, der er en optagelse af den målte acceleration ved håndholdt stationær rotation af controlleren.

Ud fra Figur 23 ses ikke overraskende, at lave værdier af $\beta$ generelt er mere begrænsende end høje værdier. Der opstår dog en sjov effekt visse steder, da nogle værdier begrænser flere samples end tilsvarende høje værdier. Dette sker højst sandsynligt fordi begrænsningen ikke sker direkte som følge af værdien af $\beta$, men derimod som følge af $\alpha$ og $\vec{g}'_{base}$ (se Algoritme 1, side 26). Vi vil gerne vælge $\beta$ så lavt som muligt, men det er tydeligt ud fra figuren at antallet af begrænsede samples ikke just er stabilt indtil ca. $\frac{1}{8}\pi$, hvorfor vi har valgt $\beta = \frac{1}{8}\pi$.

Nu da vi har fundet værdier for både $k$ og $\beta$ kan vi afprøve og sammenligne præcisionen af den naive og den adaptive metode.

**Præcision af tiltestimatet ved hurtige lineære bevægelser** Vi starter med at teste på et datasæt med 500 samples (`carForthBack.dat`), hvor hurtige stærke skift i accelerationen har til formål at afsløre præcisionen af de anvendte metoder. Wii-



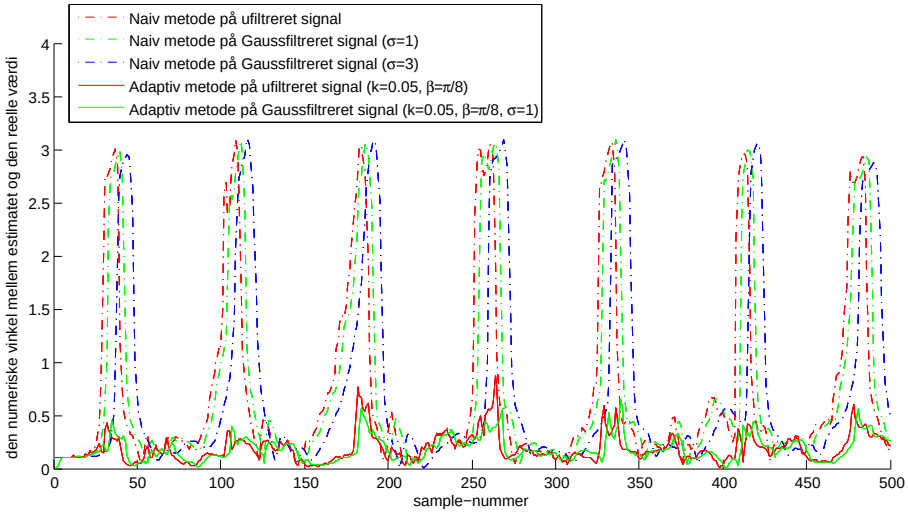

**Figur 24:** Figuren viser forskellen i vinklen mellem estimatet $\vec{g}'$ og den reelle værdi $\vec{g}$, ved hurtige lineære bevægelser. Estimatet er beregnet for både 500 ufiltrerede og filtrerede samples, og med både den naive og den adaptive metode. Lave værdier for vinklen er således udtryk for en mere præcis metode.

controlleren blev placeret i vores LEGO-vogn (se Figur 21), og vognen blev da accelereret skiftevis i positiv og negativ retning i kun én akse. Testen blev udført på både ufiltrerede og filtrerede varianter af datasættet med både den naive og den adaptive metode.

I Figur 24 ses resultatet af testen, der meget tydeligt vindes af den adaptive metode. Ingen af metoderne er dog tæt på at være perfekte, men vinklen af estimatet fra den adaptive metode er, for en stor del af signalet, betydeligt mindre end vinklen af estimatet fra den naive metode, og derved er estimatet fra den adaptive metode mindst afvigende i forhold til den reelle værdi. Det ses desuden at nogle af metodernes resultater er forskudte, hvilket skyldes at disse metoder anvendes på et filtreret, og derved „forsinket“, signal.

| **Gennemsnitlig forskel i vinklen ved hurtige lineære bevægelser** | |
|---|---|
| 0.3312 | Adaptiv metode på ufiltreret signal |
| 0.3376 | Adaptiv metode på Gaussfiltreret signal ($\sigma = 1$) |
| 0.6522 | Naiv metode på Gaussfiltreret signal ($\sigma = 3$) |
| 0.6719 | Naiv metode på Gaussfiltreret signal ($\sigma = 1$) |
| 0.6802 | Naiv metode på ufiltreret signal |

**Præcision af tiltestimatet ved bevægelse langs tyndekraftens akse** Det næste datasæt (`handUpDown.dat`) er baseret på håndholdt brug af Wii-controlleren, hvor controlleren holdes vandret mens den bevæges op og ned langs tyndekraftens akse. Det kan diskuteres at $\vec{g}$ i da tilfælde ikke er udtryk for den reelle acceleration som følge af tyngdekraften under hele testen, men i dette tilfælde vil det alligevel være anvendeligt at sammenligne med $\vec{g}$, eftersom controlleren ikke udsættes for betydelig rotation under testen. Testen har til formål at teste om de anvendte metoder kender forskel på,




**Figur 25:** Figuren viser forskellen i vinklen mellem estimatet $\vec{g}'$ og den reelle værdi $\vec{g}$, ved bevægelse langs tyngdekraftens akse, og uden bevidst rotation. Estimatet er beregnet for både 500 ufiltrerede og filtrerede samples, og med både den naive og den adaptive metode. Lave værdier for vinklen er således udtryk for en mere præcis metode.

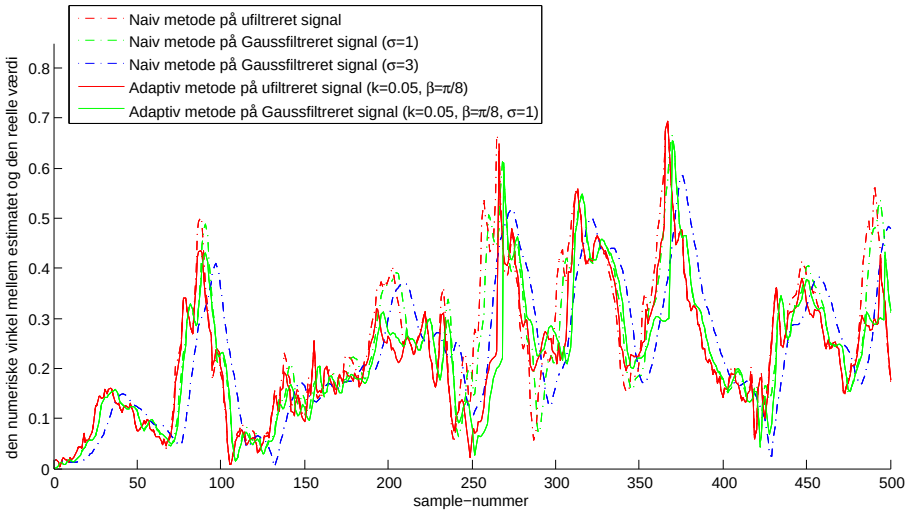
om controlleren rent faktisk vendes på hovedet, eller om den blot udsættes for kraftig acceleration langs tyngdekraftens akse. Datasættet indeholder 500 samples.

I Figur 25 ses resultatet af den naive og den adaptive metode på det ufiltrerede og filtrerede datasæt. Det ses at den adaptive metode er langt overlegen den naive metode i denne test. Den naive metode antager en rotation på næsten $\pi$, som følge af at den målte acceleration i visse tilfælde svinger mellem $-1g$ og $1g$, mens den adaptive metode gør brug af vinkelbegrænsningen (med udgangspunkt i $\beta$) til at eliminere de pludselige udsving.

Resultatet af den adaptive metode er desuden en lille smule mere præcist ved anvendelse på det filtrerede signal, i forhold til anvendelse på det ufiltrerede signal, specielt omkring sample 200 og sample 250. Det kan dog diskuteres om det er dét værd, at ofre den gode responstid for så lille en forbedring i den gennemsnitlige præcision over hele signalet.

| **Gennemsnitlig forskel i vinklen ved bevægelse langs tyngdekraftens akse** | |
|---|---|
| 0.1803 | Adaptiv metode på Gaussfiltreret signal ($\sigma = 1$) |
| 0.1894 | Adaptiv metode på ufiltreret signal |
| 0.7302 | Naiv metode på ufiltreret signal |
| 0.7314 | Naiv metode på Gaussfiltreret signal ($\sigma = 1$) |
| 0.7368 | Naiv metode på Gaussfiltreret signal ($\sigma = 3$) |

**Præcision af tiltestimatet ved moderat bevægelse i alle akser** Det tredje og sidste datasæt (`handLevelMovement.dat`) er også baseret på håndholdt brug af Wii-controlleren. Denne gang holdes controlleren nogenlunde vandret, mens den flyttes stille og roligt, men tilfældigt, rundt i rummet i alle tre akser. Datasættet indeholder således en moderat skiftende acceleration i alle tre akser, og eftersom controlleren hol-




**Figur 26:** Figuren viser forskellen i vinklen mellem estimatet $\vec{g}'$ og den reelle værdi $\vec{g}$, ved moderat bevægelse langs alle akser, og uden bevidst rotation. Estimatet er beregnet for både 500 ufiltrerede og filtrerede samples, og med både den naive og den adaptive metode. Lave værdier for vinklen er således udtryk for en mere præcis metode.

des nogenlunde vandret under hele testen går det an, ganske som i forrige test, at teste præcisionen af estimatet i forhold til $\vec{g}$.

I Figur 26 ses resultatet af testen, der ikke overraskende viser, at den naive metode i dette tilfælde er omtrent lige så præcis som den adaptive metode. Faktisk ser det ud som om, at den naive metode anvendt på et Gaussfiltreret datasæt, hvor $\sigma = 3$, giver et lidt bedre resultat, end begge tilfælde af den adaptive metode. Dette er dog ikke tilfældet når man kigger på gennemsnittet af vinklerne, over hele signalet, for de forskellige metoder:

| **Gennemsnitlig forskel i vinklen ved moderat bevægelse i alle akser** | |
|---|---|
| 0.2202 | Adaptiv metode på Gaussfiltreret signal ($\sigma = 1$) |
| 0.2237 | Adaptiv metode på ufiltreret signal |
| 0.2327 | Naiv metode på Gaussfiltreret signal ($\sigma = 3$) |
| 0.2434 | Naiv metode på Gaussfiltreret signal ($\sigma = 1$) |
| 0.2464 | Naiv metode på ufiltreret signal |

Den adaptive metode må derfor også i dette tilfælde regnes for den bedste metode.

### 6.1.1 Opsummering

Generelt giver den adaptive metode et mere præcist tiltestimat end den naive metode. Dette skyldes primært, at den adaptive metode er bedre til, at udelukke fejlagtige udsving i tiltestimatet, som følge af høj pludselig acceleration. Den adaptive metode var dog i et enkelt tilfælde (test af præcisionen ved moderat bevægelse af Wii-controlleren) kun marginalt bedre end den naive, hvilket skyldes at datasættet kun indeholdt relativ lav acceleration, der derfor ikke kunne udelukkes af den adaptive metode. I de resterende tests lagde den adaptive metode dog tydeligt afstand til den naive metode.

Når det er sagt, så skal det også siges, at ingen af de to metoder giver et perfekt estimat af tilt, og det er uanset om Wii-controlleren bevæges hurtigt eller langsomt, eller slet ikke: Estimatet $\vec{g}'$ svinger uanset metoden, og ligger relativt længere fra den reelle værdi $\vec{g}$ når controlleren er i anden bevægelse end rotation. Dette medfører også, at det er svært, når controlleren er i sådan bevægelse, at anvende estimatet som parameter til andre metoder, der kræver et forholdsvist præcist estimat af $\vec{g}$ for at fungere korrekt.

Man kan dog sagtens anvende tiltestimatet, og derved Wii-controlleren, til at udføre rotationer i rum. Man skal blot være opmærksom på, at præcisionen af aksen og vinklen for rotationen generelt falder, jo mere controlleren udsættes af brugeren for anden påvirkning end blot rotation. Når controlleren er i relativ stilstand, og kun udsættes for rotation, vil tiltestimatet omvendt være både responsivt og relativt præcist.

Succeskriteriet for vores implementation om muligheden for rotation i rum er således opfyldt så længe Wii-controlleren kun udsættes for rotation, og begrænses kun af, at vi ikke ud fra data fra accelerometeret alene kan estimere yaw (se afsnit 3.2.1, side 11), og derved controllerens fulde orientering.

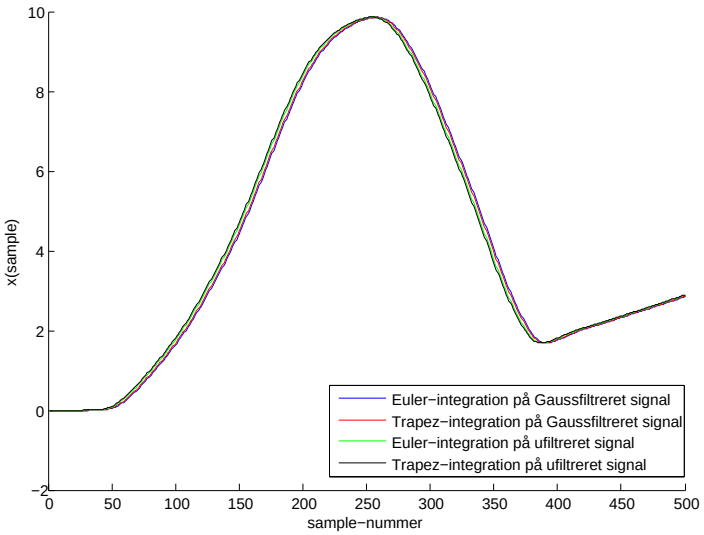
## 6.2 Tests af estimering af position

For at teste estimering af position har vi optaget tre datasæt `car1mBackForth.dat`, `car1mBackForthSlow.dat` og `carStill.dat`; to hvor vi skubbede LEGO-vognen frem og tilbage over en afstand på 1 meter, således at de bageste hjuls akser startede og stoppede ved hhv. 0- og 1-metermarkeringerne (se Figur 27); og et hvor vognen holdt helt stille på en plan, vandret flade.


**Figur 27:** Figuren viser anvendelsen af vognen til måling af accelerationen langs en enkelt akse over en afstand på 1 meter.

For det første datasæt kørte vi vognen langsomt, så det tog 10 sekunder at køre ruten og for det andet kørte vi vognen hurtigere, så det tog 5 sekunder at køre ruten. For det sidste datasæt lod vi vognen stå uberørt, så der ikke var acceleration i nogen af accelerometerets akser, ud over acceleration som følge af tyngdekraften i y-aksen.

Ud fra de to første datasæt er det muligt, at kontrollere præcisionen af positionsestimeringen, idet vognen kører lige langt frem og tilbage, så vognen gerne skulle ende med, at have flyttet sig 0.0 meter fra dens startposition. Den kørte afstand skulle til gengæld gerne være 2.0 meter, idet vognen reelt tilbagelægger en afstand på 2.0 meter. I det sidste eksempel, hvor vognen holder stille skulle der ikke være nogen acceleration, ud over den nævnte acceleration som følge af tyngdekraften i y-aksen. Ved at have




**Figur 28:** Plot af resultatet af integration med hhv. Euler og trapez, med og uden Gaussfiltrering, ved høj acceleration.

støjen i et signal uden anden påvirkning vil vi være i stand til, at genkende et eventuelt drift, som følge af akkumuleret støj i signalet, hvilket bør medføre at den endeligt kørte afstand er forskellig fra $0.0$.

Da accelerationen udlæses fra Wii-controlleren i $g$ er afstanden, som er resultat af dobbelt-integrationen, ikke et udtryk for reelle meter, men et udtryk for meter i forhold til tyngdeaccelerationen, således at $1g \approx 9.82m$. Dette kan vi benytte til, at omregne resultatet til meter ved

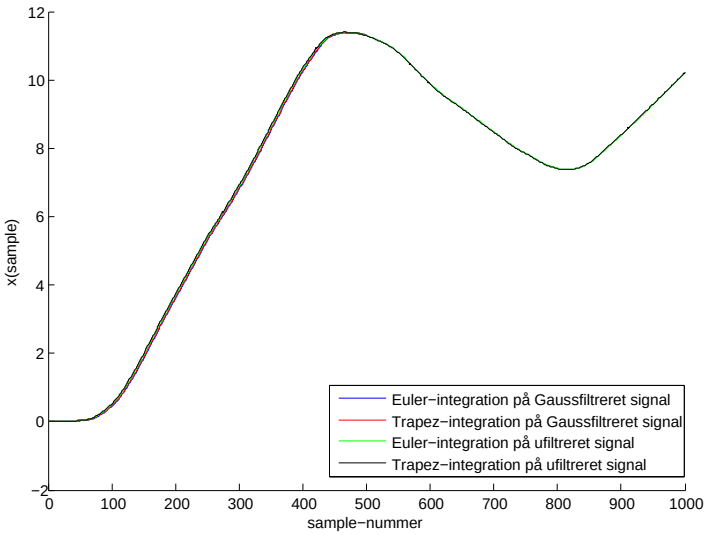
$$d = \sum_{i=1}^{N} (\frac{a_i}{9.82} \Delta t),$$

hvor $a$ er resultatet af dobbelt-integrationen, $\Delta t$ er tiden mellem hver sample i sekunder, og $d$ er den samlede afstand.

**Præcision af positionsestimatet ved høj acceleration** I Figur 28 ses resultatet af den dobbelte integration på det anvendte datasæt, hvor vognen har kørt hurtigt. Figuren viser resultatet af to forskellige integratorer, Euler og trapez, med og uden Gaussfiltrering med $\sigma = 1$. Det ses tydeligt, at de fire resultater ligger meget tæt op ad hinanden. Figuren viser et tydeligt drift, idet signalet starter i $0.0$, men ender, ved sample $400$, i omkring $1.68$, hvor vognen reelt har nået sit mål. Efterfølgende forekommer et tydeligt drift, der trækker afstanden i vejret med forholdsvist høj hastighed.

Når vognen når sit mål ved sample $400$ har den, ifølge positionsestimeringen, kørt $\sum_{i=1}^{400} \frac{a_i}{9.82} 0.01 = 1.8947m$, hvor den i realiteten har kørt $2m$, hvilket svarer til et positivt drift på $0.1053m$. Dette drift må anses for, at være betydeligt i forhold til den korte afstand og lave hastighed.

Det ses desuden, at vognen kører tilbage lidt over halvvejs inde i signalet, hvor afstanden nærmer sig $10$, hvilket svarer til, at vognen har kørt $\sum_{i=1}^{258} \frac{a_i}{9.82} 0.01 =$




**Figur 29:** Plot af resultatet af integration med hhv. Euler og trapez, med og uden Gaussfiltrering, ved lav acceleration.

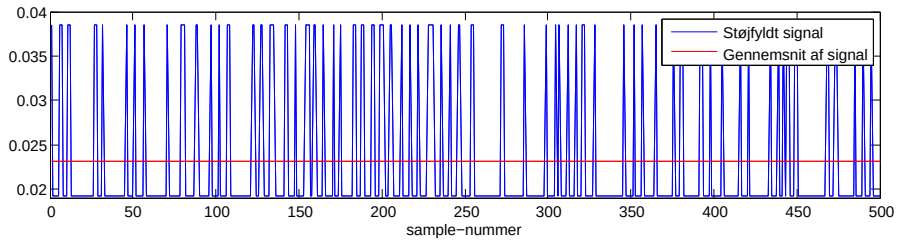
$1.0647m$, der er lidt over $6cm$ længere end den reelt har kørt. Allerede her ses det positive drift, som forværres betydeligt på vejen tilbage.

**Præcision af positionsestimatet ved lav acceleration** Figur 29 viser resultatet af den dobbelte integration på det anvendte datasæt, hvor vognen har kørt langsomt. Ligesom ved høj acceleration er der ikke den store afvigelse mellem de fire resultater, hvorimod det er meget tydeligt, at se hvordan et meget kraftigt drift, forårsaget af støj, påvirker positionsestimatet så voldsomt, at det i praksis er ubrugeligt. Når vognen kører tilbage bliver dens tilbagelagte afstand kun estimeret til omtrent $\frac{1}{3}$ af den reelle afstand, hvilket er helt uacceptabelt.

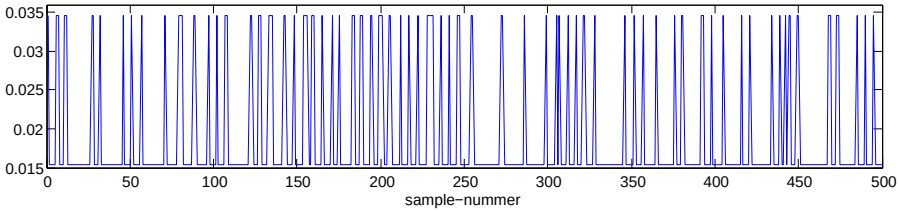
Ud fra de to resultater, i Figur 28 og Figur 29, er det tydeligt, at se hvordan hastigheden af vognen har en stor indflydelse på størrelsen af den drift der påvirker signalet.

Det skal dog nævnes, at det ikke udelukkende er støj der er skyld i de store afvigelser, i form af drift. Det kan ligesåvel være fejlkilder i form af, at vognen ikke er blevet kørt præcist 1 meter frem og 1 meter tilbage, men måske lidt over en meter frem og lidt mindre tilbage. En anden fejlkilde kan være, at den overflade hvorpå vi kørte vognen ikke er helt plan. Til sidst skal nævnes, at også kalibreringen af Wii-controlleren kan være årsag til afvigelser i positionsestimatet, da vi ikke kan være sikre på, at kalibreringsdata er perfekt.

**Præcision af positionsestimatet ved stilstand** For at få en ide om hvor meget accelerometeret er påvirket af støj analyserer vi det datasæt, der er optaget hvor LEGO-vognen holder stille på en plan flade. Figur 30 viser hvordan signalet er påvirket af udelukkende positiv støj, ved stilstand. Tillige ses at basis for signalet, og dermed støjen, har et offset på $0.0192$, hvilket giver en yderligere usikkerhed i positionsestimeringen.




**Figur 30:** Figuren viser datasættet optaget, hvor LEGO-vognen holder stille. Det er tydeligt, at se at signalet er støjfyldt, samt at der er et offset i forhold til den reelle acceleration af vognen.




**Figur 31:** Figuren viser en korrektion af datasættet optaget, hvor LEGO-vognen holder stille. Der er kompenseret for støjens indflydelse på signalet på signalet ved at trække et bias fra signalet.

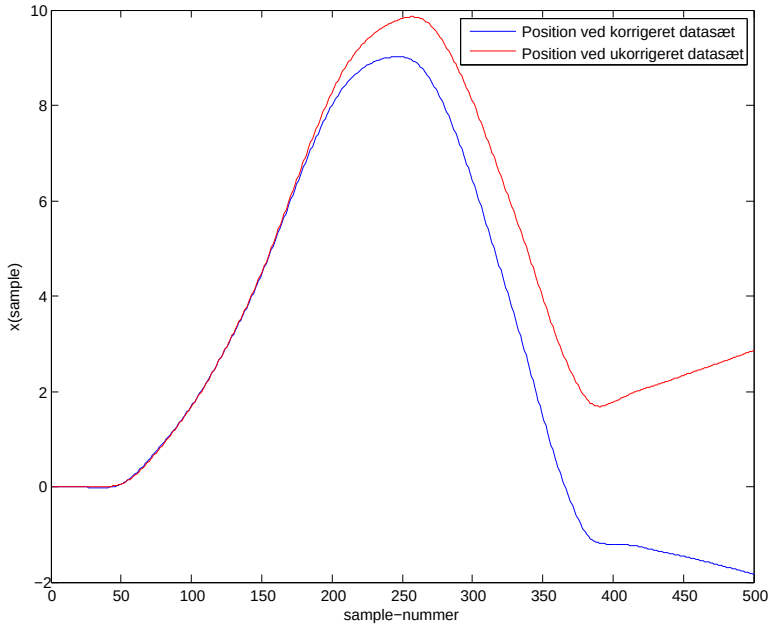
Da vi ikke er tilfredse med resultatet af den metode for reducering af støjen, som vi beskrev i analysen (se afsnit 4.3.4, side 34) har vi overvejet en anden metode, der tager udgangspunkt i accelerationen ved stilstand. Ved hjælp af den normaliserede information fra z-aksen i datasættet (se Figur 30), kan vi finde gennemsnittet for støjen, hvilket kan benyttes til, at mindske støjens påvirkning på det rene signal.

Hvis vi antager, at signalets offset også er gældende for de andre akser, og når Wii-controlleren er i bevægelse, kan vi forsøge at kompensere for støjen ved at udlede et bias, der er forskellen mellem gennemsnittet af støjen og signalets basisværdi. Denne forskel kan trækkes fra hele signalet, hvilket svarer til, at støjen fordeles jævnt omkring signalets basis, hvormed der bliver kompenseret for forskellen mellem den optagne acceleration og den reelle acceleration. Dette bias er i tilfældet i Figur 30 på $0.0039$.

Figur 31 viser signalet fra Figur 30, hvor der er kompenseret for støjen ved at trække det fundne bias fra signalet. Det ses at signalet, som følge af korrektionen med bias, ligger lavere end det oprindeligt gjorde.

Figur 32 viser denne fremgangsmåde benyttet på det første datasæt, hvor vognen er kørt $1$ meter frem og tilbage på $5$ sekunder. Det er tydeligt, at se at det fratrukne bias har en indvirkning på signalet, men det er reelt for stort i forhold til hvor meget der skal korrigeres. Dog ligger det nye signal tættere på det ideelle end det ukorrigerede signal, da afvigelsen er hhv. omkring $-1.1793$ og $1.7714$. Dette betyder, at der er lidt at vinde ved at anvende bias til at korrigere for støjen og muligvis ville der være endnu mere at vinde, hvis kalibreringen af akserne var mere præcis.

Denne form for støjreduktion forudsætter, at støjen er additiv således, at den ikke ændrer karakter i forskellige situationer. Vi har testet dette ved, at optage et datasæt, hvor Wii-controlleren er i stilstand, ligesom før, men roteret $\frac{\pi}{2}$ radianer, således at




**Figur 32:** Plot af Gaussfiltreret trapez-integration udført på hhv. korrigeret og ukorrigeret data.

z-aksen påvirkes med $1g$ af tyngdekraften. I dette tilfælde havde støjen samme karakteristik, som den tidligere test og må derfor være additiv.

### 6.2.1 Opsummering

Som vi så ved resultatet af de to første datasæt er vores positionsestimering præget af støj, der medfører et meget kraftigt positivt drift således, at det over længere tid er umuligt, at få et positionsestimat der er bare en lille smule anvendeligt. Dette gør sig især gældende ved langsomme bevægelser, der giver en lav acceleration, som har stor risiko for, at „drukne“ i støj og således ikke kan estimeres tilfredsstillende.

Vi testede derfor en anden teori om hvorledes støjen kan reduceres, så det er muligt at få et tilfredsstillende positionsestimat, hvilket til dels lykkedes. Resultatet af testen med støjreduktion, baseret på bias, viste sig at være en anelse bedre end den vi havde forudset i analysen (se afsnit 4.3.4 på side 34). Vi har desværre ikke nået at implementere denne løsning, så vi kan ikke eftergive den i praksis i WiiLib.

Et af de største problemer vi har tilbage er stadig drift, som påvirker signalet meget voldsomt. Da drift dels skyldes støj, numerisk integration og usikkerheder i kalibreringsdata er der ikke nogen sikker måde, at undgå dette på. Man kan tage højde for drift, som følge af numerisk integration, ved at indbygge en form for bremse, der sørger for at registrere konstante hastigheder i alle tre akser og kompensere for disse, men dette løser dog ikke problemet for drift som følge af støj og usikkerheder i kalibreringsdata.

Et andet meget stort problem er, at vi ikke kan estimere $\vec{g}$ præcist, når Wii-controlleren bevæges, hvorfor vi ikke kan tage fuldstændigt højde for acceleration som

følge af tyngdekraften, når controlleren er håndholdt. Dette efterlader spor af accelerationen som følge af tyngdekraften på alle tre akser og påvirker inddata til integratoren på uforudsigelig vis. Således er det svært at foretage et perfekt tilt-estimat. Således vil et håndholdt positionsestimat være yderligere forværret i forhold til vores resultater i denne afprøvning, og brugbarheden af vores estimater vil derfor være begrænset jf. alle de problemer vi har med drift.

Hvis man skal lave et godt positionsestimat ud fra de data vi modtager fra controlleren er det nødvendigt, at controlleren styres således, at dens orientering aldrig ændrer sig, men det kun er positionen der ændrer sig. I dette tilfælde kan vi regne med en konstant acceleration fra tyngdekraften der ikke ændrer retning, og det er således ikke nødvendigt, at estimere accelerationen som følge af tyngdekraften.

Et problem med denne løsning er at der stadig vil være støj på signalet og at vi ikke helt kan regne med kalibreringen af controlleren, så der vil stadig være et drift, som skal elimineres. Løsningen har desuden et andet, meget stort problem: At controlleren bliver nødt til, at være spændt fast i en eller anden form for holder, der sikrer, at controllerens orientering ikke ændrer sig, hvilket fjerner hele ideen med, at controlleren håndholdt skal kunne bruges til, at flytte på objekter i virtuelt rum.

Vi er, som nævnt, ikke tilfredse med vores løsning for positionsestimering, da denne ikke lever op til vores forventninger om at anvende Wii-controlleren håndholdt til, at flytte objekter i rum. Problemet opstår pga. kraftig drift som følge af usikkerheder i proceduren, der gør at positionsestimatet bliver ekstremt upræcist.

Succeskriteriet for vores implementation om muligheden for translation i rum er dog opfyldt, idet vi godt kan anvende vores metode til, at registrere translation af Wii-controlleren. På grund af støj, og derved drift, er translationen desværre meget upræcis og derfor mindre anvendelig i praksis.

# 7 Konklusion

Vi har udviklet et klassebibliotek, WiiLib, der sørger for kommunikationen med en Wii-controller samt behandling af data fra denne. Ud fra viden om accelerometeret i Wii-controlleren har vi udledt og udviklet forskellige metoder til estimering af controllerens tilt (dennes delvise orientering) og controllerens relative position. Blandt andet har vi udviklet en adaptiv metode med henblik på, at kunne estimere controllerens tilt mere præcist. Vi har derudover udviklet en udvidelse til 3D-motoren 3DOT, der gør direkte brug af metoderne i WiiLib.

Ved afprøvning af metoderne til estimering af tilt fandt vi, at vi er i stand til at estimere Wii-controllerens delvise orientering med meget acceptabel præcision, når blot controlleren udelukkende udsættes for rotation. Når controlleren også udsættes for anden påvirkning end tyngdekraftens, da falder præcisionen af tiltestimatet som følge af, at tyngdekraftens påvirkning ikke længere kan udledes entydigt fra den målte acceleration. Ved anvendelse af den adaptive metode får vi dog stadig, i de fleste tilfælde, et estimat med acceptabel præcision.

Eftersom vi kan estimere Wii-controllerens tilt med acceptabel præcision, og da tiltestimatet kan oversættes til orienteringen af et objekt i et virtuelt rum, da kan vi afgjort anvende controlleren til at rotere objekter i et virtuelt rum, og med acceptabel præcision.

Ved afprøvning af metoderne til estimering af Wii-controllerens position fandt vi, at faktorer som signal-støj, og usikkerhed i kalibreringsdata, medfører et meget stort „drift“ i den resulterende position. Vi forsøgte for så vidt muligt under afprøvningen, at skabe ideelle forhold for afprøvning af positionsestimering, blandt andet ved at stabilisere Wii-controlleren, og kun påvirke denne langs en enkelt akse. Dette var desværre ikke tilstrækkeligt, hvilket måske til dels skyldtes, at vi først sent i projektet overvejede muligheden for et additivt bias i forhold til støjen per akse i accelerometeret.

Eftersom vores positionsestimat er offer for drift, selv under relativt ideelle forhold (hvor Wii-controlleren ikke udsættes for rotation), er det kun mindre anvendeligt hvis man ønsker at overføre positionen direkte til et objekt i et virtuelt rum. Vi mener dog, at det i højere grad er en uheldig egenskab ved accelerometre; at de udbyder summen af alle påvirkninger, end det er en decideret programfejl i vores implementation.

Vi kan konkludere, at vi godt kan anvende Wii-controlleren til rotationer i et virtuelt rum, specielt når Wii-controlleren kun udsættes for rotation, men at vi i mindre grad kan anvende Wii-controlleren til overførsel af dennes position, til en partikel eller et objekt i det virtuelle rum.

# 8 Perspektivering

Hvis Wii-controlleren indeholdt et gyroskop, en sensor til måling af vinkelhastigheden i controllerens 3 akser, da ville man kunne gøre brug af denne hastighed til, at estimere Wii-controllerens fuldstændige orientering, inklusive rotationen om tyngdekraftens akse. Derudover ville man muligvis også kunne bruge vinkelhastigheden til, specielt i tilfælde hvor controlleren udsættes for andre påvirkninger end blot tyngdekraften, at udregne et endnu mere præcist estimat af accelerationen som følge af tyngdekraften, og derved et mere præcist estimat af Wii-controllerens orientering i forhold til denne.

Wii-controlleren indeholder som nævnt en udvidelsesport til tilslutning af forskellige enheder, hvorfor et oplagt fremtidigt projekt ville være, at designe et udvidelsesmodul med et gyroskop til controlleren, og implementere brugen af dette i WiiLib.

Eftersom et mere præcist estimat af acceleration som følge af tyngdekraften er et bedre udgangspunkt for estimering af position, da ville positionsestimatet muligvis også forbedres ved anvendelse af et gyroskop til estimering af Wii-controllerens orientering. Man kan dog yderligere forestille sig, at implementere støjreduktion ved, at trække et bias fra signalet, som beskrevet i afsnit 6.2.1 på side 57, før signalet kalibreres med kalibreringsdata fra controlleren, da disse måske ikke er helt præcise.

Da vi har implementeret en WiiLib-baseret udvidelse til 3DOT, og da 3DOT er multiplatform, kunne det være en god ide, at lægge lidt mere arbejde i også, at gøre selve WiiLib multiplatform. I WiiLib er det `HID`-klassen der begrænser muligheden for at kompilere til andre operativsystemer end Windows, da HID-klassen udelukkende supporterer HID-laget i Windows. At gøre WiiLib multiplatform kræver derfor hovedsageligt, at funktionaliteten af `HID`-klassen implementeres for hver platform man ønsker at supportere.

# Referencer

[3DOT, 2007] 3DOT (2007). 3DOT.
http://3dot.dk.

[AiLive, 2007] AiLive (2007). LiveMove.
http://www.ailive.net/liveMove.html.

[Analog Devices, 2006] Analog Devices (2006). ADXL330.
http://www.analog.com/UploadedFiles/Data_Sheets/ADXL330.pdf.

[Baker, 2007] Baker, M. J. (2007). Rotation conversions.
http://www.euclideanspace.com/maths/geometry/rotations/conversions/.

[DADIU, 2007] DADIU (2007). Det Danske Akademi for Digital, Interaktiv Underholdning.
http://dadiu.dk.

[Forbes, 2006] Forbes, K. (2006). cWiiMote 0.2.
http://simulatedcomicproduct.com/2006/12/cwiimote-02.php.

[Grubb, 2007] Grubb, P. (2007). Wii-controller 3D Studio Max model.
http://wiinintendo.net/2007/01/30/free-3d-model-of-wiimote-your-welcome.

[IVT corporation, 2007] IVT corporation (2007). BlueSoleil.
http://www.bluesoleil.com.

[Nintendo, 2006] Nintendo (2006). Wii.
http://www.nintendo.com/channel/wii.

[Nintendo, 2007] Nintendo (2007). Wii-controllers.
http://wii.nintendo.com/controller.jsp#nunchuk.

[OpenGL, 2007] OpenGL (2007). Open Graphics Library.
http://opengl.org.

[Pixart Imaging Inc., 2006] Pixart Imaging Inc. (2006). Pixart teams with Nintendo for sensor tracking technology for „Wii“ controller.
http://www.pixart.com.tw/investor.asp?sort=4&learname=level03.

[PUC-Rio, 2007] PUC-Rio (2007). The Programming Language LUA.
http://lua.org.

[Weisstein, 2007] Weisstein, E. W. (2007). „Euler Parameters“ From Mathworld–A Wolfram Web Resource.
http://mathworld.wolfram.com/EulerParameters.html.

[WiiBrew, 2007] WiiBrew (2007). WiiBrew.
http://wiibrew.org.

[WiiLi, 2007] WiiLi (2007). WiiLi.
http://www.wiili.org.

# Figurer

# Tabeller

# A Definitioner

Dette afsnit beskriver nogle af de begreber der bruges i teksten og som måske er ukendte for læseren. Begreberne er sorteret i den rækkefølge de optræder i teksten.

**Rumble-motor** Denne enhed i Wii-controlleren gør det muligt at give brugeren feedback på samme måde, som vibratoren i en mobiltelefon.

**CMOS** Complementary Metal Oxide Semiconductor er en meget populær energieffektiv elektronisk komponent af typen havleder.

**HID** Human Interface Device er en standard for enheder til brugerinput på en computer.

**EEPROM** Electrically Erasable Programmable Read Only Memory er en speciel type hukommelseschip, der tillader læsning og skrivning og kan gemme skrevet data uden at være tilsluttet en strømkilde.

**XML** eXtensible Markup Language er en standard for, at lagre data på en struktureret og let læselig måde.

**OpenGL** Open Graphics Library [OpenGL, 2007] er et open-source multi-platforms bibliotek til udvikling af 2D- og 3D-miljøer og benyttes ofte i spil.

**LUA** Programmeringssproget LUA [PUC-Rio, 2007] er meget anvendt i computerspilverdenen og tillader nem *real-time* manipulation med objekter i scenen.

**DLL** Dynamic Link Library er en fil-type, der indeholder eksekverbar kode, som kan indlæses efter behov under afviklingen af et program.

# B Gesturegenkendelse

Det er muligt, at lave gesturegenkendelse på flere måder: en måde er at „optage“ de signaler, der kommer fra Wii-controlleren og samenligne dem med tiligere optagede signaler, som er bundet til forskellige handlinger. Denne måde gør det muligt, at kunne lave den samme gesture på mere end én måde, da den samme data kan genereres af accelerometeret ud fra to forskellige bevægelser (MISSING eksempel). Metoden kræver desuden, at man gemmer alle data, der er modtaget fra accelerometeret, hvilket kan være en stor del for en kompliceret gesture.

En anden måde er at benytte positionsestimering så man er sikker på, at en gesture er lavet ud fra en bestemt bevægelse. Denne metode er lettere at repræsentere, da man kan nøjes med, at gemme start- og slut-punkter for bevægelserne, hvilket også gør den hurtigere, da der er færre data der skal sammenlignes.

I begge tilfælde er der et problem med, at gestures skal kunne overlappe, hvilket vil sige, at flere gestures kan have samme mønster i starten og kun afvige lidt i slutningen. Dette lægger op til, at gestures skal gemmes som segmenter, der matches mod det modtagne mønster. På denne måde får man opbygget en sekvens af gesture-segmenter, som matcher det modtagne mønster. Denne gesture-sekvens matcher muligvis flere gestures, men til sidst vil det matche højst en gesture, hvis bundne handling udføres.

# C Kildekode

## C.1 WiiLib



### WiiLib

**Listing 1:** /WiiLib/WiiLib.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_H__
#define __WIILIB_H__

// Include everything necessary to connect
#include "Controller.h"

// Include utility classes
#include "Tilt.h"
#include "Position.h"

// Include integrators
#include "IntegratorEuler.h"
#include "IntegratorTrapez.h"
#include "IntegratorMidpoint.h"

// Include gravity models
#include "GravityNaive.h"
#include "GravityAdaptive.h"

#endif
```

### HID

**Listing 2:** /WiiLib/HIDConfig.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
```

```
    The WiiLibHID class is based on the HID-class of cWiiMote 0.2:
    http://simulatedcomicproduct.com/2006/12/cwiimote-02.php
    by Kevin Forbes
*/
#ifndef __WIILIB_HIDCONFIG_H__
#define __WIILIB_HIDCONFIG_H__

#define USE_BLUESOLEIL  0
#define USE_DEBUG       0

#endif
```

**Listing 3:** /WiiLib/HID.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>

    The WiiLibHID class is based on the HID-class of cWiiMote 0.2:
    http://simulatedcomicproduct.com/2006/12/cwiimote-02.php
    by Kevin Forbes
*/
#ifndef __WIILIB_HID_H__
#define __WIILIB_HID_H__

#include <windows.h>

extern "C" {
    #include "hidsdi.h"
}

// Device information for bluetooth handshake
#define VENDOR_ID   0x057e
#define DEVICE_ID   0x0306

namespace WiiLib
{
    typedef VOID    (__stdcall *PHidD_GetHidGuid)(LPGUID);
    typedef BOOLEAN (__stdcall *PHidD_GetAttributes)(HANDLE, PHIDD_ATTRIBUTES);
    typedef BOOLEAN (__stdcall *PHidD_SetOutputReport)(HANDLE, PVOID, ULONG);
    typedef BOOLEAN (__stdcall *PHidD_SetNumInputBuffers)(HANDLE, ULONG);

    class WiiLibHID
    {
    public:

        WiiLibHID();
        ~WiiLibHID();

    public:

        bool Connect(unsigned short deviceId, unsigned short vendorId,
                     int index=0);
        bool Disconnect();

        bool Write(unsigned const char *buffer, int numBytes);
        bool Read(unsigned char *buffer, int maxBytes, int &numBytes,
                  int timeout=50);

        bool IsConnected() const { return mConnected; }

    private:

        HMODULE hHID;
        HANDLE  hReadWrite;

        bool    mConnected;

        PHidD_GetHidGuid            HidD_GetHidGuid;
        PHidD_GetAttributes         HidD_GetAttributes;
        PHidD_SetOutputReport       HidD_SetOutputReport;
```

```
        PHidD_SetNumInputBuffers    HidD_SetNumInputBuffers;

    private:

        bool OpenDevice(int index);
        void Debug(unsigned const char *buffer, int numBytes, bool read,
                   bool fail);

    };
}
// End namespace WiiLib

#endif
```

**Listing 4:** /WiiLib/HID.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>

    The WiiLibHID class is based on the HID-class of cWiiMote 0.2:
    http://simulatedcomicproduct.com/2006/12/cwiimote-02.php
    by Kevin Forbes
*/
#include "HID.h"
#include "HIDConfig.h" /* Must define USE_DEBUG and USE_BLUESOLEIL */
#include <setupapi.h>
#include <stdio.h>

#pragma comment(lib, "setupapi.lib")

namespace WiiLib
{
    /*
     * Setup
     */

    // Constructor
    WiiLibHID::WiiLibHID()
    {
        hReadWrite = NULL;

        hHID = LoadLibraryA("hid.dll");
        if (!hHID)
        {
            printf("Could not load 'hid.dll'.\n");
        }

        HidD_GetHidGuid = (PHidD_GetHidGuid)
            GetProcAddress(hHID, "HidD_GetHidGuid");
        HidD_GetAttributes = (PHidD_GetAttributes)
            GetProcAddress(hHID, "HidD_GetAttributes");
        HidD_SetOutputReport = (PHidD_SetOutputReport)
            GetProcAddress(hHID, "HidD_SetOutputReport");
        HidD_SetNumInputBuffers = (PHidD_SetNumInputBuffers)
            GetProcAddress(hHID, "HidD_SetNumInputBuffers");

        if (!HidD_GetHidGuid || !HidD_GetAttributes || !HidD_SetOutputReport ||
            !HidD_SetNumInputBuffers)
        {
            FreeLibrary(hHID);
            printf("Could not find one or more HID entry points.");
        }

        mConnected = false;
    }

    // Disconnect on destruct
    WiiLibHID::~WiiLibHID()
    {
        if (mConnected)
```

```
58          {
59              Disconnect();
60          }
61          if (hHID)
62          {
63              FreeLibrary(hHID);
64          }
65      }

67      // Disconnect by closing existing handles
68      // Returns true if successful
69      bool WiiLibHID::Disconnect()
70      {
71          if (mConnected)
72          {
73              mConnected = !CloseHandle(hReadWrite);
74          }
75          return !mConnected;
76      }

78      // Connect to device with specific device id, vendor id, and index
79      // Returns true if successful
80      bool WiiLibHID::Connect(unsigned short deviceId, unsigned short vendorId,
81                              int index)
82      {
83          bool result = false;
84          int devIndex = 0;
85          int devCount = 0;

87          // Attempt to disconnect and reconnect if already connected
88          if (mConnected && !Disconnect())
89          {
90              return false;
91          }

93          // Loop through devices until matching item is found or end is reached
94          while (true)
95          {
96              HIDD_ATTRIBUTES attrib;

98              // Abort if a device cannot be opened
99              // This will happen when there are no more items to check
100             if (!OpenDevice(devIndex)) {
101                 break;
102             }

104             // Note that when OpenDevice succeeds it creates a handle hReadWrite

106             // Read attributes of device
107             // Skip device if its attributes cannot be read
108             if (HidD_GetAttributes(hReadWrite, &attrib))
109             {
110                 // Check that device id's and vendor id's match
111                 if (attrib.ProductID == deviceId && attrib.VendorID == vendorId)
112                 {
113                     // If index matches device number we have a valid connection
114                     if (devCount == index)
115                     {
116                         mConnected = true;
117                         break;
118                     }
119                     devCount++;
120                 }
121             }

123             // Free unused handles
124             CloseHandle(hReadWrite);

126             // Next device
127             devIndex++;
128         }

130         return mConnected;
131     }
```

```
133     // Open device by specific index
134     // If successful it creates the handle hReadWrite
135     // Returns true if successful
136     bool WiiLibHID::OpenDevice(int index)
137     {
138         bool result = false;

140         GUID guid;
141         HDEVINFO devInfo;
142         SP_DEVICE_INTERFACE_DATA devIntData;

144         // Get device list
145         HidD_GetHidGuid(&guid);
146         devInfo = SetupDiGetClassDevs(&guid, NULL, NULL,
147                                       DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);
148         devIntData.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);

150         // If index exists in device list
151         if (SetupDiEnumDeviceInterfaces(devInfo, NULL, &guid, index,
152                                         &devIntData) == TRUE)
153         {
154             DWORD size;
155             PSP_INTERFACE_DEVICE_DETAIL_DATA detail;

157             // First read size of device details
158             SetupDiGetDeviceInterfaceDetail(devInfo, &devIntData, NULL, 0,
159                                             &size, 0);

161             // Then get device details according to size
162             detail = (PSP_INTERFACE_DEVICE_DETAIL_DATA)malloc(size);
163             detail->cbSize = sizeof(SP_INTERFACE_DEVICE_DETAIL_DATA);

165             // Then create device handles and mark successful
166             if (SetupDiGetDeviceInterfaceDetail(devInfo, &devIntData, detail,
167                                                 size, NULL, NULL))
168             {
169                 hReadWrite = CreateFile(detail->DevicePath,
170                                         GENERIC_READ | GENERIC_WRITE,
171                                         FILE_SHARE_READ | FILE_SHARE_WRITE,
172                                         NULL,
173                                         OPEN_EXISTING,
174                                         NULL/* | FILE_FLAG_OVERLAPPED*/,
175                                         NULL);

177                 // Setting number of buffers low requires less fast reading,
178                 // but increases chance of missing packages that interleave
179                 // with acceleration data (calibration data, for example)
180                 HidD_SetNumInputBuffers(hReadWrite, 2);

182                 result = true;
183             }

185             // Free device details
186             free(detail);
187         }

189         // Free device list
190         SetupDiDestroyDeviceInfoList(devInfo);

192         return result;
193     }

195     /*
196      * Read/write
197      */

199     // Write
200     // Returns true if successful
201     bool WiiLibHID::Write(unsigned const char *buffer, int numBytes)
202     {
203         bool result = false;

205         // Write buffer to device
```

```
        if (mConnected) {
#if USE_BLUESOLEIL
            // BlueSoleil stack
            unsigned char* long_buffer = new unsigned char[22];
            DWORD bytes_written = 0;

            memset(long_buffer, 0, 22);
            memcpy(long_buffer, buffer, numBytes);

            result = WriteFile(hReadWrite, long_buffer, 22, &bytes_written,
                               NULL);

            delete[] long_buffer;
#else
            // Windows XP stack
            result = HidD_SetOutputReport(hReadWrite, (PVOID)buffer, numBytes);
#endif
        }
#if USE_DEBUG
        // Debugging
        Debug(buffer, numBytes, false, !result);
#endif

        return result;
    }

    // Read
    // Returns true if successful
    bool WiiLibHID::Read(unsigned char *buffer, int maxBytes, int &numBytes,
                         int timeout)
    {
        bool result = false;

        // Read into buffer from device
        if (mConnected)
            result = ReadFile(hReadWrite, (LPVOID)buffer, maxBytes,
                              (LPDWORD)&numBytes, NULL) != 0;

#if USE_DEBUG
        // Debugging
        Debug(buffer, numBytes, true, !result);
#endif

        return result;
    }

    /*
     * Debugging
     */

    // Prints some debug information
    void WiiLibHID::Debug(unsigned const char *buffer, int numBytes, bool read,
                          bool fail)
    {
        printf("%s%s", read ? "READ" : "WRITE", fail ? "FAIL" : "");
        for (int i = 0; i < numBytes; i++)
        {
            printf(" %2x", buffer[i]);
        }

        printf("\n");
    }
}
// End namespace WiiLib
```

## Controller

**Listing 5:** /WiiLib/Controller.h

```
/*
    This file is part of WiiLib
```

```
4     Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
5     Author: Sabinsky Tøgern <ast@hoast.dk>
6 */
7 #ifndef __WIILIB_CONTROLLER_H__
8 #define __WIILIB_CONTROLLER_H__

10 #include "HID.h"
11 #include "Vec3.h"
12 #include <time.h>
13 #include <sys/timeb.h>
14 #include <fstream>

16 // LED bits
17 #define LED_1                          0x10    // fmt: SS => S0
18 #define LED_2                          0x20
19 #define LED_3                          0x40
20 #define LED_4                          0x80
21 #define LED_MASK                       0xF0    // all led bits set

23 // Button bits
24 #define BUTTON_TWO                     0x0001  // fmt: BB BB => 00 BB
25 #define BUTTON_ONE                     0x0002
26 #define BUTTON_B                       0x0004
27 #define BUTTON_A                       0x0008
28 #define BUTTON_MINUS                   0x0010
29 #define BUTTON_HOME                    0x0080
30 #define BUTTON_LEFT                    0x0100  // fmt: BB BB => BB 00
31 #define BUTTON_RIGHT                   0x0200
32 #define BUTTON_DOWN                    0x0400
33 #define BUTTON_UP                      0x0800
34 #define BUTTON_PLUS                    0x1000
35 #define BUTTON_MASK                    0x1F9F  // all button bits set

37 #define STATUS_INTERVAL_MS             5
38 #define SYNC_TIMEOUT_MS                2

40 #define HB_SUCCESS                     0
41 #define HB_ERROR_DISCONNECT            1
42 #define HB_ERROR_UNKNOWN_REPORT        2
43 #define HB_ERROR_READ_FAIL             3

45 namespace WiiLib
46 {
47     class Controller
48     {
49     public:

51         enum ReportMode
52         {
53             Report_Buttons,         // report only button presses
54             Report_ButtonsMotion    // report button presses and acceleration
55         };

57         struct StatusReport
58         {
59             short   buttons;    // bitmask of buttons pressed
60             int     accRawX;    // acceleration in x, range -255..256
61             int     accRawY;    // acceleration in y, range -255..256
62             int     accRawZ;    // acceleration in z, range -255..256
63             double  accX;       // calibrated accel. in x where 1.0 = 1g
64             double  accY;       // calibrated accel. in y where 1.0 = 1g
65             double  accZ;       // calibrated accel. in z where 1.0 = 1g
66             Vec3    accNatural; // calibrated AND rotated accel, according
67                                 // to the controller's natural orientation
68                                 // (buttons facing up, and front facing away
69                                 // from holder's body), where 1.0 = 1g
70             double  deltaT;     // time since last status report (in ms)
71             StatusReport() : buttons(0), accX(0), accY(0), accZ(0) {}
72         } mStatus, mStatus0G, mStatus1G;

74         struct ExpansionReport
75         {
76             bool            attachment; // true if expansion port in use
```

```
            bool            camera;     // true if camera (ir) enabled
            bool            speaker;    // true if speaker enabled
            unsigned char   leds;       // bitmask of lit leds
            unsigned char   battery;    // battery level (0-200 [wiili.org])
            ExpansionReport() : attachment(false), camera(false), speaker(false),
                                leds(0), battery(255) {}
        } mExpansion;

    public:

        Controller();
        ~Controller();

    public:

        bool Connect(int index = 0);
        bool Disconnect();
        void CalibrateAsync();

        bool IsConnected();
        bool IsCalibrated();
        bool HasButtonData();
        bool HasMotionData();

        bool SetLEDs(unsigned char mask);
        bool SetRumble(bool state);
        bool SetReportMode(ReportMode reportMode, bool continuous);

        int Sample(int timeout = 50);
        int GetLastSampleResult();

        void PrintStatus();

    private:

        WiiLibHID       mDevice;        // device connection
        int             mLastIndex;     // device index
        unsigned char   mReportMode;    // report mode hex code
        bool            mContinuous;    // true if continuous reporting is enabled
        bool            mCalibrated;    // true if calibrated
        bool            mHasMotionData; // true if there is valid motion data
        bool            mHasButtonData; // true if there is valid button data
        unsigned char   mRumble;        // bitmask for rumble
        unsigned char   mLeds;          // bitmask for lit leds
        int             mLastSampleR;   // last sample result
        __int64         mPrevT;         // last sample time
        std::ofstream   mLog;           // log output stream

        static const int    INPUT_BUFFER_SIZE = 22;         // buffer size
        unsigned char       inputBuffer[INPUT_BUFFER_SIZE]; // input buffer

    private:

        void LogEvent(char _event[]);

        bool SendExpansionDataRequest();
        bool SendLEDs();
        bool SendReportMode();
        bool SendCalibrationDataRequest();

        void ParseButtonData(const unsigned char *data);
        void ParseMotionData(const unsigned char *data);
        void ParseExpansionData(const unsigned char *data);
        void ParseCalibrationData(const unsigned char *data);
    };
}
// End namespace WiiLib

#endif
```

**Listing 6:** /WiiLib/Controller.cpp

```
/*
```

```
2     This file is part of WiiLib

4     Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
5     Author: Sabinsky Tøgern <ast@hoast.dk>
6 */
7 #include "Controller.h"
8 #include <stdio.h>

10 /*
11  * Definitions
12  */

14 // Data reporting definitions
15 #define OUTPUT_SET_LEDS                 0x11
16 #define OUTPUT_SET_REPORT_MODE          0x12
17 #define OUTPUT_REQ_STATUS_INFO          0x15
18 #define OUTPUT_REQ_READ_MEMORY          0x17

20 #define INPUT_REQ_STATUS_INFO           0x20
21 #define INPUT_REQ_READ_MEMORY           0x21

23 #define REPORT_MODE_BUTTONS             0x30    // fmt: BB BB
24 #define REPORT_MODE_BUTTONS_MOTION      0x31    // fmt: BB BB XX YY ZZ

26 #define REPORT_CHANGES_ONLY             0x00
27 #define REPORT_CONTINUOUSLY             0x04

29 // Rumble motor states
30 #define RUMBLE_OFF                      0x00
31 #define RUMBLE_ON                       0x01

33 // Embedded acceleration bits (LSBs)
34 // Button byte 1 = 00 X1 X0 00 00 00 00 00
35 // Button byte 2 = 00 Z1 Y1 00 00 00 00 00
36 #define LSB_ACC_X1                      14      // 1<<14 = 0x4000 = accel LSB X1
37 #define LSB_ACC_X0                      13      // 1<<13 = 0x2000 = accel LSB X0
38 #define LSB_ACC_Y1                      5       // 1<<5  = 0x0020 = accel LSB Y1
39 #define LSB_ACC_Z1                      6       // 1<<6  = 0x0040 = accel LSB Z1

41 // Memory addresses
42 #define CALIBRATION_DATA_ADDRESS        0x16
43 #define CALIBRATION_DATA_LENGTH         8

45 namespace WiiLib
46 {
47     /*
48      * Setup
49      */

51     // Constructor
52     Controller::Controller()
53     {
54         mReportMode         = REPORT_MODE_BUTTONS;
55         mCalibrated         = false;
56         mContinuous         = false;
57         mRumble             = RUMBLE_OFF;
58         mLeds               = 0;

60         mLog.open("c:\\WiiLib.log", std::ofstream::trunc);
61         if (!mLog.is_open())
62         {
63             printf("Error opening log file!\n");
64         }
65         else
66         {
67             printf("Log file opened\n");
68         }
69     }

71     // Destructor
72     Controller::~Controller()
73     {
74         Disconnect();
75         if (mLog.is_open())
```

```
76          {
77              mLog.close();
78          }
79      }

81      void Controller::LogEvent(char _event[])
82      {
83          if (mLog.is_open())
84          {
85              mLog << time(NULL) << ": " << _event;
86              mLog.flush();
87          }
88      }

90      // Connect
91      bool Controller::Connect(int index)
92      {
93          if (mDevice.Connect(DEVICE_ID, VENDOR_ID, index))
94          {
95              memset(&mStatus, 0, sizeof(mStatus));

97              mLastIndex = index;
98              SendCalibrationDataRequest();
99              SendReportMode();
100             SendLEDs();

102             LogEvent("Connection established\n");

104             return true;
105         }
106         else
107         {
108             LogEvent("Could not connect\n");
109             return false;
110         }
111     }

113     // Disconnect
114     bool Controller::Disconnect()
115     {
116         if (mDevice.Disconnect())
117         {
118             mCalibrated = false;
119             LogEvent("Disconnected\n");
120             return true;
121         }
122         else
123         {
124             LogEvent("Failed disconnect attempt\n");
125             return false;
126         }
127     }

129     // Just a public wrapper for SendCalibrationDataRequest
130     void Controller::CalibrateAsync()
131     {
132         SendCalibrationDataRequest();
133     }

135     /*
136      * Queries
137      */

139     // True if connected
140     bool Controller::IsConnected()
141     {
142         return mDevice.IsConnected();
143     }

145     // True if calibrated
146     bool Controller::IsCalibrated()
147     {
148         return mCalibrated;
149     }
```

```
151     // True if there is valid button data to be read
152     bool Controller::HasButtonData()
153     {
154         return mHasButtonData;
155     }

157     // True if there is valid motion data to be read
158     bool Controller::HasMotionData()
159     {
160         return mHasMotionData;
161     }

163     /*
164      * Output
165      */

167     // Set LED status
168     bool Controller::SetLEDs(unsigned char mask)
169     {
170         mLeds = mask & LED_MASK;
171         return SendLEDs();
172     }

174     // Set rumble status
175     bool Controller::SetRumble(bool state)
176     {
177         mRumble = (state ? RUMBLE_ON : RUMBLE_OFF);
178         return SendLEDs();
179     }

181     // Request status report
182     bool Controller::SendExpansionDataRequest()
183     {
184         unsigned char data[2];
185         data[0] = 0x15;
186         data[1] = mRumble;
187         return mDevice.Write(data, 2);
188     }

190     // Send LED status
191     bool Controller::SendLEDs()
192     {
193         unsigned char data[2];
194         data[0] = OUTPUT_SET_LEDS;
195         data[1] = mLeds | mRumble;
196         return mDevice.Write(data, 2);
197     }

199     // Set report mode
200     bool Controller::SetReportMode(ReportMode reportMode, bool continuous)
201     {
202         switch (reportMode)
203         {
204             case Report_Buttons:
205                 LogEvent("SetReportMode(): buttons\n");
206                 mReportMode = REPORT_MODE_BUTTONS;
207                 break;
208             case Report_ButtonsMotion:
209                 LogEvent("SetReportMode(): buttons_motion\n");
210                 mReportMode = REPORT_MODE_BUTTONS_MOTION;
211                 break;
212         }
213         mContinuous = continuous;
214         return SendReportMode();
215     }

217     // Send report mode
218     bool Controller::SendReportMode()
219     {
220         unsigned char data[3];
221         data[0] = OUTPUT_SET_REPORT_MODE;
222         data[1] = (mContinuous ? REPORT_CONTINUOUSLY : REPORT_CHANGES_ONLY)
223                 | mRumble;
```

```
224         data[2] = mReportMode;
225         return mDevice.Write(data, 3);
226     }

228     // Sends calibration data request
229     // Format of read request is: AA AA AA AA SS SS where A's
230     // denote read address and S's denote read length
231     bool Controller::SendCalibrationDataRequest()
232     {
233         unsigned char data[7];
234         data[0] = OUTPUT_REQ_READ_MEMORY;
235         data[1] = mRumble;
236         data[2] = 0;
237         data[3] = 0;
238         data[4] = CALIBRATION_DATA_ADDRESS;
239         data[5] = 0;
240         data[6] = CALIBRATION_DATA_LENGTH;
241         return mDevice.Write(data, 7);
242     }

244     /*
245      * Input
246      */

248     // Parse button data
249     // Assumes two-byte sequence of button data, format: BB BB
250     void Controller::ParseButtonData(const unsigned char *data)
251     {
252         // Set button mask
253         mStatus.buttons = ((data[0]<<8) | data[1]) & BUTTON_MASK;
254         // Button data is now available
255         mHasButtonData = true;
256     }

258     inline static unsigned __int64 _RDTSC()
259     {
260         _asm _emit 0x0F
261         _asm _emit 0x31
262     }

264     // Parse motion data
265     // Assumes five-byte sequence of button and motion data where button data
266     // contains LSB of motion data, format: BB BB XX YY ZZ
267     void Controller::ParseMotionData(const unsigned char *data)
268     {
269         short mask = (data[0]<<8) | data[1];

271         // Get raw x-acceleration with 10 bits precision
272         mStatus.accRawX = data[2]<<2;
273         mStatus.accRawX |= ( (mask>>LSB_ACC_X1)&0x1 )<<1;
274         mStatus.accRawX |= ( (mask>>LSB_ACC_X0)&0x1 );
275         mStatus.accRawX -= mStatus0G.accRawX;

277         // Get raw y-acceleration with 9 bits precision
278         mStatus.accRawY = data[3]<<2;
279         mStatus.accRawY |= ( (mask>>LSB_ACC_Y1)&0x1 )<<1;
280         mStatus.accRawY -= mStatus0G.accRawY;

282         // Get raw z-acceleration with 9 bits precision
283         mStatus.accRawZ = data[4]<<2;
284         mStatus.accRawZ |= ( (mask>>LSB_ACC_Z1)&0x1 )<<1;
285         mStatus.accRawZ -= mStatus0G.accRawZ;

287         // Get normalized (calibrated) acceleration
288         mStatus.accX = (double)(mStatus.accRawX) /
289                        (double)(mStatus1G.accRawX - mStatus0G.accRawX);
290         mStatus.accY = (double)(mStatus.accRawY) /
291                        (double)(mStatus1G.accRawY - mStatus0G.accRawY);
292         mStatus.accZ = (double)(mStatus.accRawZ) /
293                        (double)(mStatus1G.accRawZ - mStatus0G.accRawZ);

295         // Get normalized (calibrated) AND rotated acceleration, according to
296         // controller's natural orientation (buttons facing up, and front facing
297         // away from holder's body)
```

```
298         Vec3::GetNaturalAccelerationVector(mStatus.accX, mStatus.accY,
299                                            mStatus.accZ, mStatus.accNatural);

301         // Calculate time since last status report
302         __int64 now = _RDTSC();
303         mStatus.deltaT = now - mPrevT;

305         // Update the previous time structure
306         mPrevT = now;

308         // Motion data is now available
309         mHasMotionData = true;
310     }

312     // Parse expansion data
313     // Assumes four-byte sequence, format: DD DD DD DD
314     void Controller::ParseExpansionData(const unsigned char *data)
315     {
316         mExpansion.attachment   = (data[0] & 0x02) != 0;
317         mExpansion.camera       = (data[0] & 0x08) != 0;
318         mExpansion.speaker      = (data[0] & 0x04) != 0;
319         mExpansion.leds         = (data[0] & LED_MASK);
320         mExpansion.battery      = data[3];

322         // Re-send report mode
323         SendReportMode();
324     }

326     // Parse calibration data
327     // Assumes eight-byte sequence, format: X0 Y0 Z0 ?? X1 Y1 Z1 ??
328     void Controller::ParseCalibrationData(const unsigned char *data)
329     {
330         int mask = 0x03;

332         // Get raw acceleration at zero-g
333         mStatus0G.accRawX = (data[0]<<2) | ((data[3]>>4)&mask);
334         mStatus0G.accRawY = (data[1]<<2) | ((data[3]>>2)&mask);
335         mStatus0G.accRawZ = (data[2]<<2) | ((data[3]>>0)&mask);

337         // Get raw acceleration at 1 g
338         mStatus1G.accRawX = (data[4]<<2) | ((data[7]>>4)&mask);
339         mStatus1G.accRawY = (data[5]<<2) | ((data[7]>>2)&mask);
340         mStatus1G.accRawZ = (data[6]<<2) | ((data[7]>>0)&mask);

342         // Done calibrating
343         mCalibrated = true;
344     }

346     // Sample
347     // This function reads data from device and distributes this data to the
348     // different input parsers for button states, motion reports, etc.
349     int Controller::Sample(int timeout)
350     {
351         int result = HB_SUCCESS;
352         int numBytes = 0;
353         unsigned char reportId;

355         // Clear input buffer
356         memset(&inputBuffer, 0, INPUT_BUFFER_SIZE);

358         // Read from device to input buffer
359         if (mDevice.Read(inputBuffer, INPUT_BUFFER_SIZE, numBytes, timeout) &&
360             numBytes > 0)
361         {
362             reportId = inputBuffer[0];
363             switch (reportId)
364             {
365                 // Expansion data
366                 case INPUT_REQ_STATUS_INFO:
367                     ParseExpansionData(&inputBuffer[3]);
368                     break;
369                 // Memory read data, most likely calibration data
370                 // Also contains button data
371                 case INPUT_REQ_READ_MEMORY:
```

```
                    ParseButtonData(&inputBuffer[1]);
                    ParseCalibrationData(&inputBuffer[6]);
                    break;
                // Button data
                case REPORT_MODE_BUTTONS:
                    ParseButtonData(&inputBuffer[1]);
                    break;
                // Button and motion data
                case REPORT_MODE_BUTTONS_MOTION:
                    ParseButtonData(&inputBuffer[1]);
                    // Process motion data only once calibrated
                    if (mCalibrated)
                    {
                        ParseMotionData(&inputBuffer[1]);
                    }
                    break;
                // Unknown data
                default:
                    LogEvent("Sample: Unknown report received\n");
                    result = HB_ERROR_UNKNOWN_REPORT;
                    break;
            }
        }
        else
        {
            LogEvent("Sample: Could not read from Bluetooth stack\n");
            // An error occured reading from the Bluetooth stack. This is not
            // catastrophic and will happen many consecutive times when the
            // controller is ungracefully disconnected before the actual
            // disconnect is discovered
            result = HB_ERROR_READ_FAIL;
        }

        // Store result for threaded use
        mLastSampleR = result;

        // Return result
        return result;
    }

    // Returns last sample result
    int Controller::GetLastSampleResult()
    {
        return mLastSampleR;
    }

    /*
     * Debugging
     */

    // Print status
    void Controller::PrintStatus()
    {
        // Acceleration
        printf("[%d\t%d\t%d\t%+1.2f\t%+1.2f\t%+1.2f] ",
            mStatus.accRawX, mStatus.accRawY, mStatus.accRawZ,
            mStatus.accX, mStatus.accY, mStatus.accZ
        );
        // Buttons
        if (mStatus.buttons & BUTTON_TWO)
            printf("2");
        if (mStatus.buttons & BUTTON_ONE)
            printf("1");
        if (mStatus.buttons & BUTTON_B)
            printf("B");
        if (mStatus.buttons & BUTTON_A)
            printf("A");
        if (mStatus.buttons & BUTTON_MINUS)
            printf("-");
        if (mStatus.buttons & BUTTON_HOME)
            printf("H");
        if (mStatus.buttons & BUTTON_LEFT)
            printf("L");
        if (mStatus.buttons & BUTTON_RIGHT)
```

```
            printf("R");
        if (mStatus.buttons & BUTTON_DOWN)
            printf("D");
        if (mStatus.buttons & BUTTON_UP)
            printf("U");
        if (mStatus.buttons & BUTTON_PLUS)
            printf("+");
        printf("\n");
    }
}
// End namespace WiiLib
```

## Filter

**Listing 7:** /WiiLib/Filter.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_FILTER_H__
#define __WIILIB_FILTER_H__

typedef unsigned int uint;

namespace WiiLib
{
    class Filter
    {
    public:

        Filter(double *kernel, int kernelSize, int kernelCenterIndex);
        ~Filter();

    public:

        const double    *GetKernel() const;             // return kernel pointer
        int             GetKernelSize() const;          // return kernel size
        int             GetKernelCenterIndex() const;   // return center index

    private:

        double          *mKernel;           // pointer to filter kernel
        int             mKernelSize;        // filter kernel size
        int             mKernelCenterIndex; // filter kernel center index

    public:

        // Creates a pass-through filter
        static Filter *CreatePassThroughFilter();
        // Creates a low-pass filter using using a 1D gaussian distribution
        static Filter *CreateGaussFilter(double deviation);
        // Creates a low-pass gaussian filter with specified kernel radius
        static Filter *CreateGaussFilterWithRadius(uint kernelRadius);
        // Creates a low-pass filter using an array of exponential coefficients
        static Filter *CreateExponentialFilter(uint kernelRadius, double alpha);

    };
}
// End namespace WiiLib

#endif
```

**Listing 8:** /WiiLib/Filter.cpp

```
/*
    This file is part of WiiLib
```

```
4     Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
5     Author: Sabinsky Tøgern <ast@hoast.dk>
6 */
7 #define _USE_MATH_DEFINES

9 #include "Filter.h"
10 #include <cmath>
11 #include <iostream>

13 namespace WiiLib
14 {
15     /*
16      * Setup
17      */

19     Filter::Filter(double *kernel, int kernelSize, int kernelCenterIndex)
20         :
21         mKernel(kernel),
22         mKernelSize(kernelSize),
23         mKernelCenterIndex(kernelCenterIndex)
24     {
25     }

27     Filter::~Filter()
28     {
29         if (mKernelSize != 0)
30         {
31             delete[] mKernel;
32         }
33     }

35     /*
36      * Kernel
37      */

39     const double *Filter::GetKernel() const
40     {
41         return mKernel;
42     }

44     int Filter::GetKernelSize() const
45     {
46         return mKernelSize;
47     }

49     int Filter::GetKernelCenterIndex() const
50     {
51         return mKernelCenterIndex;
52     }

54     /*
55      * Factories
56      */

58     // Creates a filter that simply lets original data pass through
59     Filter *Filter::CreatePassThroughFilter()
60     {
61         int kernelSize = 1;
62         int kernelCenterIdx = 0;
63         double *kernel = new double[kernelSize];

65         for (int i = 0; i < kernelSize; i++)
66         {
67             kernel[i] = 1;
68         }

70         return new Filter(kernel, kernelSize, kernelCenterIdx);
71     }

73     // Creates a low-pass filter using a one-dimensional Gaussian distribution
74     Filter *Filter::CreateGaussFilter(double deviation)
75     {
76         int kernelRadius = (int)std::ceil(3*deviation);
```

```
        int kernelSize = 1+2*kernelRadius;
        int kernelCenterIdx = kernelRadius;
        double *kernel = new double[kernelSize];
        double x;

        for (int i = 0; i < kernelSize; i++)
        {
            // Formula for the Gaussian distribution (1D) is:
            // G(x) = exp(-x^2/(2*deviation^2)) / (deviation*sqrt(2*pi));
            x = i-kernelRadius;
            kernel[i] = exp(-(x*x)/(2*deviation*deviation)) /
                        (deviation*sqrt(2*M_PI));
        }

        return new Filter(kernel, kernelSize, kernelCenterIdx);
    }

    // Creates a low-pass Gaussian filter with specified kernel radius
    Filter *Filter::CreateGaussFilterWithRadius(uint kernelRadius)
    {
        double deviation = (double)kernelRadius/3.0;
        return Filter::CreateGaussFilter(deviation);
    }

    // Creates a low-pass filter using an exponential distribution based on alpha
    Filter  *Filter::CreateExponentialFilter(uint kernelRadius, double alpha)
    {
        int kernelSize = 1+2*kernelRadius;
        int kernelCenterIdx = kernelRadius;
        double *kernel = new double[kernelSize];

        for (int i = kernelSize-1; i >= 0; i--)
        {
            kernel[i] = std::pow(1 - alpha, i) * alpha;
            //printf("%0.4f\n", kernel[i]);
        }

        return new Filter(kernel, kernelSize, kernelCenterIdx);
    }
}
```

## Signal

**Listing 9:** /WiiLib/Signal.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_SIGNAL_H__
#define __WIILIB_SIGNAL_H__

#include "Filter.h"

namespace WiiLib
{
    class Signal
    {
    public:

        Signal(int bufferSize);
        Signal(int bufferSize, Filter *fi);
        ~Signal();

    public:

        void    SetBufferSize(int n);       // sets ringbuffer size,
                                            // NOTE: destroys buffer contents!
        int     GetBufferSize() const;      // returns ringbuffer size
```

```
        const double *GetBuffer() const;     // returns pointer to ringbuffer data
        int     GetBufferOffset() const;     // returns ringbuffer offset
        int     GetBufferNumInUse() const;   // returns number of ringbuffer
                                             // indices in use
        bool    HasFullBuffer() const;       // returns true if ringbuffer is full
        void    Append(double sample);       // appends sample to signal ringbuffer

    public:

        void    SetFilter(Filter *fi);       // sets attached filter kernel,
                                             // NOTE: invalidates filtered data!
        const Filter *GetFilter() const;     // returns pointer to attached filter
        const double *GetFiltered();         // returns pointer to filtered data
        double  GetFilteredValue(int index);// returns filtered value by index

    private:

        double  *mBuffer;           // ringbuffer, contains most recent
                                    // data from signal
        int     mBufferSize;        // length of segment stored in ringbuffer
        int     mBufferOffset;      // start of segment stored in ringbuffer
                                    // (oldest sample)
        int     mBufferNumInUse;    // number of ringbuffer indices in use

        Filter  *mFilter;           // pointer to attached filter
        double  *mFiltered;         // normal 0-indexed buffer containing filtered
                                    // data from convolve
        bool    mFilteredDirty;     // true if convolve hasn't run since last
                                    // append or change of filter

    private:

        void    Convolve();         // convolves ringbuffer with attached filter

    };
}
// End namespace WiiLib

#endif
```

**Listing 10:** /WiiLib/Signal.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "Signal.h"
#include <string.h>
#include <stdio.h>

namespace WiiLib
{
    /*
     * Setup
     */

    Signal::Signal(int bufferSize)
        :
        mBufferSize(0),
        mBufferOffset(0),
        mBufferNumInUse(0),
        mFilteredDirty(false)
    {
        SetBufferSize(bufferSize);
    }

    Signal::Signal(int bufferSize, Filter *fi)
        :
        mBufferSize(0),
        mBufferOffset(0),
        mBufferNumInUse(0),
```

```
32          mFilteredDirty(false)
33      {
34          SetBufferSize(bufferSize);
35          SetFilter(fi);
36      }

38      Signal::~Signal()
39      {
40          if (mBufferSize != 0)
41          {
42              delete[] mBuffer;
43              delete[] mFiltered;
44          }
45      }

47      /*
48       * Ringbuffer
49       */

51      void Signal::SetBufferSize(int n)
52      {
53          if (mBufferSize != 0)
54          {
55              delete[] mBuffer;
56              delete[] mFiltered;
57          }

59          mBufferSize = n;
60          mBufferOffset = 0;
61          mBufferNumInUse = 0;
62          mBuffer = new double[mBufferSize];
63          mFiltered = new double[mBufferSize];
64          mFilteredDirty = true;

66          memset(mBuffer, 0, mBufferSize*sizeof(double));
67          memset(mFiltered, 0, mBufferSize*sizeof(double));
68      }

70      int Signal::GetBufferSize() const
71      {
72          return mBufferSize;
73      }

75      const double *Signal::GetBuffer() const
76      {
77          return mBuffer;
78      }

80      int Signal::GetBufferOffset() const
81      {
82          return mBufferOffset;
83      }

85      int Signal::GetBufferNumInUse() const
86      {
87          return mBufferNumInUse;
88      }

90      bool Signal::HasFullBuffer() const
91      {
92          return (mBufferNumInUse == mBufferSize);
93      }

95      void Signal::Append(double sample)
96      {
97          // Add sample to ringbuffer
98          mBuffer[mBufferOffset] = sample;
99          mBufferOffset = (mBufferOffset+1)%mBufferSize;

101         // Add to number of ringbuffer indices in use
102         if (mBufferNumInUse < mBufferSize)
103         {
104             mBufferNumInUse++;
105         }
```

```
107         // Make dirty
108         mFilteredDirty = true;
109     }

111     /*
112      * Filtering
113      */

115     void Signal::SetFilter(Filter *fi)
116     {
117         // Make dirty if filter changes
118         if (mFilter != fi)
119         {
120             mFilteredDirty = true;
121         }

123         // Attach new filter
124         mFilter = fi;
125     }

127     const Filter *Signal::GetFilter() const
128     {
129         return mFilter;
130     }

132     const double *Signal::GetFiltered()
133     {
134         // Convolve if dirty
135         if (mFilteredDirty)
136         {
137             Convolve();
138         }

140         // Return buffer
141         return mFiltered;
142     }

144     double Signal::GetFilteredValue(int index)
145     {
146         // Convolve if dirty
147         if (mFilteredDirty)
148         {
149             Convolve();
150         }

152         // Return filtered value at index
153         return mFiltered[index];
154     }

156     /*
157      * Convolution
158      */

160     void Signal::Convolve()
161     {
162         int kernelSize, kernelOffset;
163         const double *kernel;
164         int ringIndex;

166         // Abort if no filter is attached
167         if (mFilter == 0)
168         {
169             return;
170         }

172         // Get kernel properties
173         kernelSize = mFilter->GetKernelSize();
174         kernelOffset = -mFilter->GetKernelCenterIndex();//-kernelSize/2;
175         kernel = mFilter->GetKernel();

177         // Calculate convoluted value for each buffer element
178         for (int i = 0; i < mBufferSize; i++)
179         {
```

```
            mFiltered[i] = 0;

            // Apply filter kernel
            for (int j = 0; j < kernelSize; j++)
            {
                // Check boundaries
                if (0 > i+j+kernelOffset || i+j+kernelOffset > mBufferSize-1)
                {
                    continue;
                }

                // Get ringbuffer index from 0-offset index
                ringIndex = (i+j+kernelOffset+mBufferOffset)%mBufferSize;

                // Convolve
                mFiltered[i] += kernel[j]*mBuffer[ringIndex];
            }
        }

        // Mark not dirty
        mFilteredDirty = false;

        // Filtered buffer now contains convolution of buffer and filter kernel
    }

}
// End namespace WiiLib
```

## Gravity

**Listing 11:** /WiiLib/Gravity.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_GRAVITY_H__
#define __WIILIB_GRAVITY_H__

#include "Vec3.h"

namespace WiiLib
{
    // Gravity estimator base class
    class Gravity
    {
    public:

        virtual void GetEstimate(const Vec3 &v, Vec3 &result) = 0;

    };
}
// End namespace WiiLib

#endif
```

**Listing 12:** /WiiLib/GravityNaive.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_GRAVITY_NAIVE_H__
#define __WIILIB_GRAVITY_NAIVE_H__
```

```
10 #include "Gravity.h"

12 namespace WiiLib
13 {
14     // Naive gravity estimation method
15     class GravityNaive : public Gravity
16     {
17     public:

19         virtual void GetEstimate(const Vec3 &v, Vec3 &result);

21     };
22 }
23 // End namespace WiiLib

25 #endif
```

**Listing 13:** /WiiLib/GravityNaive.cpp

```
1 /*
2     This file is part of WiiLib

4     Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
5     Author: Sabinsky Tøgern <ast@hoast.dk>
6 */
7 #include "GravityNaive.h"

9 namespace WiiLib
10 {
11     void GravityNaive::GetEstimate(const Vec3 &v, Vec3 &result)
12     {
13         result = v;
14         result.Normalize();
15     }
16 }
```

**Listing 14:** /WiiLib/GravityAdaptive.h

```
1 /*
2     This file is part of WiiLib

4     Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
5     Author: Sabinsky Tøgern <ast@hoast.dk>
6 */
7 #ifndef __WIILIB_GRAVITY_ADAPTIVE_H__
8 #define __WIILIB_GRAVITY_ADAPTIVE_H__

10 #include "Gravity.h"

12 namespace WiiLib
13 {
14     // Adaptive gravity estimation method
15     class GravityAdaptive : public Gravity
16     {
17     public:

19         GravityAdaptive(const Vec3 &base, double beta, double k);

21     public:

23         virtual void GetEstimate(const Vec3 &v, Vec3 &result);

25     public:

27         bool IsWithinBounds(const Vec3 &v) const;
28         bool IsBaseCandidate(const Vec3 &v) const;

30     private:

32         Vec3            mBase;
33         double          mAlpha;
34         const double    mBeta;
```

```
        const double    mK;

    private:

        double GetWeight(const Vec3 &v) const;

    };
}
// End namespace WiiLib

#endif
```

**Listing 15:** /WiiLib/GravityAdaptive.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "GravityAdaptive.h"
#include "Mat3.h"

namespace WiiLib
{
    /*
     * Setup
     */

    GravityAdaptive::GravityAdaptive(const WiiLib::Vec3 &base, double beta,
                                     double k)
        :
        mBase(base),
        mBeta(beta),
        mK(k)
    {
        mAlpha = mBeta;
    }

    /*
     * Adaptive gravity estimation method
     */

    void GravityAdaptive::GetEstimate(const Vec3 &v, Vec3 &result)
    {
        Vec3 omega;
        double theta;
        double weight;
        Mat3 rotation;

        // Zero the estimate
        result = Vec3(0,0,0);

        // Get axis-angle from base to v
        mBase.GetAxisAngle(v, omega, theta);

        // Get weight of v
        weight = GetWeight(v);

        // This algorithm computes the estimate (see the analysis chapter in
        // our report for further details, and a breakdown of why it works)
        if (theta <= mAlpha) // True if v is within bounds
        {
            theta = theta * weight;
            rotation.SetRotation(omega, theta);
            rotation.Transform(mBase, result);

            if (weight >= 1-mK) // True if v is base candidate
            {
                mBase = result; // Shift base vector
                mAlpha = mBeta; // Reset alpha
            }
        }
```

```
        else
        {
            theta = mAlpha * weight;
            rotation.SetRotation(omega, theta);
            rotation.Transform(mBase, result);

            if (weight >= 1-mK) // True if v is base candidate
            {
                mAlpha += mBeta; // Increase alpha
            }
        }

        // Estimate has now been computed in result
    }

    double GravityAdaptive::GetWeight(const Vec3 &v) const
    {
        double n = v.GetLength();

        if (n > 1)
        {
            return 1.0/n;
        }
        else
        {
            return n;
        }
    }

    /*
     * Auxilary functions for external queries
     */

    // True if vector v is within current bounds
    bool GravityAdaptive::IsWithinBounds(const Vec3 &v) const
    {
        Vec3 omega;
        double theta;

        mBase.GetAxisAngle(v, omega, theta);

        return theta <= mAlpha;
    }

    // True if vector v is base candidate
    bool GravityAdaptive::IsBaseCandidate(const Vec3 &v) const
    {
        return GetWeight(v) >= 1-mK;
    }
}
```

## Tilt

**Listing 16:** /WiiLib/Tilt.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_TILT_H__
#define __WIILIB_TILT_H__

#include "Vec3.h"
#include "Signal.h"
#include "Filter.h"
#include "Gravity.h"

namespace WiiLib
{
```

```
    class Tilt
    {
    public:

        Tilt(Filter *filter, Gravity *gravityModel);

    public:

        void    SetReference(const Vec3 &acc);          // Set reference accel.
                                                        // (accel. when facing up)
        void    SetFilter(Filter *filter);              // set smoothing filter
        void    SetGravityModel(Gravity *gravityModel); // set gravity model

        void    ProcessSample(const Vec3 &acc); // process current acceleration
                                                // (this triggers filtering etc.)

    public:

        // Returns reference gravity vector
        void    GetGravityReference(Vec3 &result) const;
        // Returns current gravity vector estimate
        void    GetGravityEstimate(Vec3 &result) const;
        // Returns current normal vector estimate
        void    GetNormalEstimate(Vec3 &result) const;
        // Returns axis-angle representation (angle in radians)
        void    GetAxisAngle(Vec3 &axis, double &angle) const;

    private:

        Vec3    mGravityReference;  // reference gravity vector
        Vec3    mGravityEstimate;   // current gravity vector estimate
        Vec3    mNormalReference;   // reference normal vector
        Vec3    mNormalEstimate;    // current normal vector estimate

        Signal  mSigX;              // x-readings
        Signal  mSigY;              // y-readings
        Signal  mSigZ;              // z-readings

        Filter  *mFilter;           // current noise filter
        Gravity *mGravityModel;     // current gravity model

    };
}
// End namespace WiiLib

#endif
```

**Listing 17:** /WiiLib/Tilt.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "Tilt.h"
#include <iostream>

namespace WiiLib
{
    /*
     * Setup
     */

    // Constructor
    Tilt::Tilt(Filter *filter, Gravity *gravityModel)
        :
        mSigX(0), mSigY(0), mSigZ(0),
        mFilter(0),
        mGravityModel(gravityModel)
    {
        SetReference(Vec3(0, 1, 0));
        SetFilter(filter);
```

```
25      }

27      // Set reference acceleration (acceleration when facing up)
28      void Tilt::SetReference(const Vec3 &acc)
29      {
30          // Get reference gravity vector (then normalize it, just in case)
31          mGravityReference = acc;
32          mGravityReference.Normalize();

34          // Get reference normal vector
35          mNormalReference = mGravityReference;
36      }

38      // Set filter
39      void Tilt::SetFilter(Filter *filter)
40      {
41          int kernelSize = filter->GetKernelSize();

43          // Update filter
44          mFilter = filter;

46          // Update signal lengths
47          mSigX.SetBufferSize(kernelSize);
48          mSigY.SetBufferSize(kernelSize);
49          mSigZ.SetBufferSize(kernelSize);

51          // Attach filter to signals
52          mSigX.SetFilter(mFilter);
53          mSigY.SetFilter(mFilter);
54          mSigZ.SetFilter(mFilter);
55      }

57      // Set gravity model
58      void Tilt::SetGravityModel(Gravity *gravityModel)
59      {
60          mGravityModel = gravityModel;
61      }

63      /*
64       * Processing
65       */

67      // Process current acceleration
68      void Tilt::ProcessSample(const Vec3 &acc)
69      {
70          int kernelCenterIndex = mFilter->GetKernelCenterIndex();
71          Vec3 accFiltered;

73          // Split sample into signals
74          mSigX.Append(acc.mX);
75          mSigY.Append(acc.mY);
76          mSigZ.Append(acc.mZ);

78          // Get filtered value
79          accFiltered.mX = mSigX.GetFilteredValue(kernelCenterIndex);
80          accFiltered.mY = mSigY.GetFilteredValue(kernelCenterIndex);
81          accFiltered.mZ = mSigZ.GetFilteredValue(kernelCenterIndex);

83          // Estimate gravity using current gravity model
84          // (then normalize it, just in case)
85          mGravityModel->GetEstimate(accFiltered, mGravityEstimate);
86          mGravityEstimate.Normalize();

88          // Get current normal vector estimate
89          //  n = 2*(g \cdot g_ref)*g_ref - g
90          mNormalEstimate = mGravityReference;
91          mNormalEstimate.Scale(
92              2*mGravityEstimate.GetDotProduct(mGravityReference));
93          mNormalEstimate.Subtract(mGravityEstimate);
94          // Alternative approach:
95          //  n = g + 2*((g \cdot g_ref)*g_ref - g)
96          //mNormalEstimate = mGravityReference;
97          //mNormalEstimate.Scale(
98          //  mGravityEstimate.GetDotProduct(mGravityReference));
```

```
        //mNormalEstimate.Subtract(mGravityEstimate);
        //mNormalEstimate.Scale(2);
        //mNormalEstimate.Add(mGravityEstimate);
    }

    /*
     * Gravity estimate and axis-angle representation
     */

    // Returns reference gravity vector
    void Tilt::GetGravityReference(Vec3 &result) const
    {
        result = mGravityReference;
    }

    // Returns current gravity vector estimate
    void Tilt::GetGravityEstimate(Vec3 &result) const
    {
        result = mGravityEstimate;
    }

    // Returns current normal vector estimate
    void Tilt::GetNormalEstimate(Vec3 &result) const
    {
        result = mNormalEstimate;
    }

    // Returns axis-angle representation (angle in radians)
    void Tilt::GetAxisAngle(Vec3 &axis, double &angle) const
    {
        mNormalReference.GetAxisAngle(mNormalEstimate, axis, angle);
    }
}
// End namespace WiiLib
```

## Integrator

**Listing 18:** /WiiLib/Integrator.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_INTEGRATOR_H__
#define __WIILIB_INTEGRATOR_H__

#include "Vec3.h"

namespace WiiLib
{
    // Numeric integrator base class
    class Integrator
    {
    public:

        Vec3 mA;    // acceleration
        Vec3 mV;    // velocity
        Vec3 mX;    // position

    public:

        Integrator() : mA(0,0,0), mV(0,0,0), mX(0,0,0) {}
        // Integrates acceleration to velocity and position
        virtual void Integrate(const Vec3 &acc, double delta) = 0;
    };
}
// End namespace WiiLib

#endif
```

**Listing 19:** /WiiLib/IntegratorEuler.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_INTEGRATOR_EULER_H__
#define __WIILIB_INTEGRATOR_EULER_H__

#include "Integrator.h"

namespace WiiLib
{
    // Performs Euler integration (left end-point) of segment of acceleration
    class IntegratorEuler : public Integrator
    {
    public:

        // Integrates acceleration to velocity and position
        virtual void Integrate(const Vec3 &acc, double delta);

    private:

        Vec3 mA_;   // previous acceleration
        Vec3 mV_;   // previous velocity
        Vec3 mX_;   // previous position

    };
}
// End namespace WiiLib

#endif
```

**Listing 20:** /WiiLib/IntegratorEuler.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "IntegratorEuler.h"

namespace WiiLib
{
    void IntegratorEuler::Integrate(const Vec3 &acc, double delta)
    {
        // Shift the previous values
        mA_ = mA;
        mV_ = mV;
        mX_ = mX;

        // Set acceleration
        mA = acc;

        // Integrate acceleration to velocity
        //  v(t+1) = v(t) + a(t) * dt
        mV.mX = mV_.mX + mA_.mX * delta;
        mV.mY = mV_.mY + mA_.mY * delta;
        mV.mZ = mV_.mZ + mA_.mZ * delta;

        // Integrate velocity to position
        //  x(t+1) = x(t) + v(t) * dt
        mX.mX = mX_.mX + mV_.mX * delta;
        mX.mY = mX_.mY + mV_.mY * delta;
        mX.mZ = mX_.mZ + mV_.mZ * delta;
    }
}
```

**Listing 21:** /WiiLib/IntegratorTrapez.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_INTEGRATOR_TRAPEZ_H__
#define __WIILIB_INTEGRATOR_TRAPEZ_H__

#include "Integrator.h"

namespace WiiLib
{
    // Performs Trapez integration of segment of acceleration
    class IntegratorTrapez : public Integrator
    {
    public:

        // Integrates acceleration to velocity and position
        virtual void Integrate(const Vec3 &acc, double delta);

    private:

        Vec3 mA_;   // previous acceleration
        Vec3 mV_;   // previous velocity
        Vec3 mX_;   // previous position

    };
}
// End namespace WiiLib

#endif
```

**Listing 22:** /WiiLib/IntegratorTrapez.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "IntegratorTrapez.h"

namespace WiiLib
{
    void IntegratorTrapez::Integrate(const Vec3 &acc, double delta)
    {
        // Shift the previous values
        mA_ = mA;
        mV_ = mV;
        mX_ = mX;

        // Set acceleration
        mA = acc;

        // Integrate acceleration to velocity
        //  v(t+1) = v(t) + ((a(t)+a(t+1))/2) * dt
        mV.mX = mV_.mX + ((mA_.mX + mA.mX)/2.0) * delta;
        mV.mY = mV_.mY + ((mA_.mY + mA.mY)/2.0) * delta;
        mV.mZ = mV_.mZ + ((mA_.mZ + mA.mZ)/2.0) * delta;

        // Integrate velocity to position
        //  x(t+1) = x(t) + ((v(t)+v(t+1))/2) * dt
        mX.mX = mX_.mX + ((mV_.mX + mV.mX)/2.0) * delta;
        mX.mY = mX_.mY + ((mV_.mY + mV.mY)/2.0) * delta;
        mX.mZ = mX_.mZ + ((mV_.mZ + mV.mZ)/2.0) * delta;
    }
}
```

**Listing 23:** /WiiLib/IntegratorMidpoint.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_INTEGRATOR_MIDPOINT_H__
#define __WIILIB_INTEGRATOR_MIDPOINT_H__

#include "Integrator.h"

namespace WiiLib
{
    // Performs Midpoint integration of segment of acceleration
    class IntegratorMidpoint : public Integrator
    {
    public:

        // Integrates acceleration to velocity and position
        virtual void Integrate(const Vec3 &acc, double delta);

    private:

        Vec3 mA_;   // previous acceleration
        Vec3 mV_;   // previous velocity
        Vec3 mX_;   // previous position
        double mDelta_; // previous delta

    };
}
// End namespace WiiLib

#endif
```

**Listing 24:** /WiiLib/IntegratorMidpoint.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "IntegratorMidpoint.h"

namespace WiiLib
{
    void IntegratorMidpoint::Integrate(const Vec3 &acc, double delta)
    {
        // Get midpoint delta from current and previous delta
        double deltaMidpoint = (mDelta_ + delta)/2.0;

        // Shift the previous values
        mA_ = mA;
        mV_ = mV;
        mX_ = mX;

        // Set acceleration
        mA = acc;

        // Integrate acceleration to velocity
        //  v(t+1) = v(t) + a(t) * ((dt + dt+1)/2)
        mV.mX = mV_.mX + mA_.mX * deltaMidpoint;
        mV.mY = mV_.mY + mA_.mY * deltaMidpoint;
        mV.mZ = mV_.mZ + mA_.mZ * deltaMidpoint;

        // Integrate velocity to position
        //  x(t+1) = x(t) + v(t) * ((dt + dt+1)/2)
        mX.mX = mX_.mX + mV_.mX * deltaMidpoint;
        mX.mY = mX_.mY + mV_.mY * deltaMidpoint;
        mX.mZ = mX_.mZ + mV_.mZ * deltaMidpoint;

        // Store delta for next midpoint delta
        mDelta_ = delta;
```

```
    }
}
```

## Position

**Listing 25:** /WiiLib/Position.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_POSITION_H__
#define __WIILIB_POSITION_H__

#include "Vec3.h"
#include "Signal.h"
#include "Filter.h"
#include "Integrator.h"

namespace WiiLib
{
    class Position
    {
    public:

        Position(Filter *filter, Integrator *integrator);

    public:

        void    SetFilter(Filter *filter);              // set filter
        void    SetIntegrator(Integrator *integrator);  // set integrator
                                                        // NOTE: resets velocity
                                                        // and position!

        void    ProcessSample(const Vec3 &acc, const Vec3 &accGravity,
                              double delta); // process current acceleration
                                             // (this triggers filtering etc.)

    public:

        void    GetVelocity(Vec3 &result) const;    // returns current velocity
        void    GetPosition(Vec3 &result) const;    // returns current position

    private:

        Vec3    mVelocity;          // current velocity
        Vec3    mPosition;          // current position

        Signal  mSigX;              // x-readings
        Signal  mSigY;              // y-readings
        Signal  mSigZ;              // z-readings

        Filter      *mFilter;       // current filter
        Integrator  *mIntegrator;   // current integrator

    };
}
// End namespace WiiLib

#endif
```

**Listing 26:** /WiiLib/Position.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
```

```
5     Author: Sabinsky Tøgern <ast@hoast.dk>
6 */
7 #include "Position.h"

9 namespace WiiLib
10 {
11     /*
12      * Setup
13      */

15     // Constructor
16     Position::Position(Filter *filter, Integrator *integrator)
17         :
18         mSigX(0), mSigY(0), mSigZ(0),
19         mFilter(0),
20         mIntegrator(integrator)
21     {
22         SetFilter(filter);
23     }

25     // Set filter
26     void Position::SetFilter(Filter *filter)
27     {
28         int kernelSize = filter->GetKernelSize();

30         // Update filter
31         mFilter = filter;

33         // Update signal lengths
34         mSigX.SetBufferSize(kernelSize);
35         mSigY.SetBufferSize(kernelSize);
36         mSigZ.SetBufferSize(kernelSize);

38         // Attach filter to signals
39         mSigX.SetFilter(mFilter);
40         mSigY.SetFilter(mFilter);
41         mSigZ.SetFilter(mFilter);
42     }

44     // Set integrator
45     void Position::SetIntegrator(Integrator *integrator)
46     {
47         mIntegrator = integrator;
48     }

50     /*
51      * Processing
52      */

54     // Process current acceleration
55     void Position::ProcessSample(const Vec3 &acc, const Vec3 &accGravity,
56                                  double delta)
57     {
58         int kernelCenterIndex = mFilter->GetKernelCenterIndex();
59         Vec3 accReal;

61         // Subtract gravity from measured acceleration => ~real acceleration
62         accReal = acc;
63         accReal.Subtract(accGravity);

65         // Split partial acceleration into signals
66         mSigX.Append(accReal.mX);
67         mSigY.Append(accReal.mY);
68         mSigZ.Append(accReal.mZ);

70         // Get filtered value (filtered real acceleration, hopefully less noisy)
71         accReal.mX = mSigX.GetFilteredValue(kernelCenterIndex);
72         accReal.mY = mSigY.GetFilteredValue(kernelCenterIndex);
73         accReal.mZ = mSigZ.GetFilteredValue(kernelCenterIndex);

75         // Estimate position by integrating the filtered real acceleration
76         // Delta is given in ms, so the real delta must be delta/1000
77         mIntegrator->Integrate(accReal, delta/1000.0);
```

```
        // Update velocity and position vectors
        mVelocity = mIntegrator->mV;
        mPosition = mIntegrator->mX;
    }

    /*
     * Gravity estimate and axis-angle representation
     */

    // Returns current velocity
    void Position::GetVelocity(Vec3 &result) const
    {
        result = mVelocity;
    }

    // Returns current position
    void Position::GetPosition(Vec3 &result) const
    {
        result = mPosition;
    }
}
// End namespace WiiLib
```

## Vec3

**Listing 27:** /WiiLib/Vec3.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_VEC3_H__
#define __WIILIB_VEC3_H__

namespace WiiLib
{
    class Vec3
    {
    public:

        double mX;
        double mY;
        double mZ;

    public:

        Vec3();
        Vec3(double x, double y, double z);

    public:

        inline double   GetLength() const;
        inline void     Normalize();

        inline void     Add(const Vec3 &v);
        inline void     Subtract(const Vec3 &v);
        inline void     Scale(double f);

        inline double   GetDotProduct(const Vec3 &v) const;
        inline void     GetCrossProduct(const Vec3 &v, Vec3 &result) const;

        inline double   GetAngle(const Vec3 &v) const;
        inline void     GetAxisAngle(const Vec3 &v, Vec3 &axis,
                                     double &angle) const;

    public:

        static void     GetNaturalAccelerationVector(double accX, double accY,
                                                     double accZ, Vec3 &result);
```

```
    };
}
// End namespace WiiLib

#include "Vec3.inl"

#endif
```

**Listing 28:** /WiiLib/Vec3.inl

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_VEC3_INLINE__
#define __WIILIB_VEC3_INLINE__

#include <cmath>

namespace WiiLib
{
    // Returns length of vector relative to length of unit vector (1.0)
    inline double Vec3::GetLength() const
    {
        return sqrt(mX*mX + mY*mY + mZ*mZ);
    }

    // Normalize this vector
    inline void Vec3::Normalize()
    {
        double f = GetLength();
        mX /= f;
        mY /= f;
        mZ /= f;
    }

    // Adds another vector to this vector
    inline void Vec3::Add(const Vec3 &v)
    {
        mX += v.mX;
        mY += v.mY;
        mZ += v.mZ;
    }

    // Subtracts another vector from this vector
    inline void Vec3::Subtract(const Vec3 &v)
    {
        mX -= v.mX;
        mY -= v.mY;
        mZ -= v.mZ;
    }

    // Scales this vector
    inline void Vec3::Scale(double f)
    {
        mX *= f;
        mY *= f;
        mZ *= f;
    }

    // Returns dot product (this vector \dot vector v)
    inline double Vec3::GetDotProduct(const Vec3 &v) const
    {
        return (mX*v.mX + mY*v.mY + mZ*v.mZ);
    }

    // Returns cross product (this vector \cross vector v) in result vector
    inline void Vec3::GetCrossProduct(const Vec3 &v, Vec3 &result) const
    {
        result.mX = mY*v.mZ - mZ*v.mY;
```

```
        result.mY = mZ*v.mX - mX*v.mZ;
        result.mZ = mX*v.mY - mY*v.mX;
    }

    // Returns unsigned angle between vectors in radians
    inline double Vec3::GetAngle(const Vec3 &v) const
    {
        // Angle formula: cos^-1 (a dot b)/(|a|*|b|)
        return acos( GetDotProduct(v) / (GetLength()*v.GetLength()) );
    }

    // Returns axis and angle of rotation (from this vector to vector v)
    // in second and third argument
    inline void Vec3::GetAxisAngle(const Vec3 &v, Vec3 &axis, double &angle) const
    {
        GetCrossProduct(v, axis);

        // If the cross product is (0,0,0) the angle must also be 0,
        // and in that case we return the vector itself as there will
        // be no actual rotation
        if (axis.GetLength() == 0)
        {
            axis = *this;
            angle = 0;
        }
        else
        {
            // Normalize axis
            axis.Normalize();

            // Angle formula: cos^-1 (a dot b)/(|a|*|b|)
            angle = acos( GetDotProduct(v) / (GetLength()*v.GetLength()) );
        }
    }
}
// End namespace WiiLib

#endif
```

**Listing 29:** /WiiLib/Vec3.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "Vec3.h"

namespace WiiLib
{
    /*
     * Setup
     */

    Vec3::Vec3()
        :
        mX(0), mY(0), mZ(0)
    {
    }

    Vec3::Vec3(double x, double y, double z)
        :
        mX(x), mY(y), mZ(z)
    {
    }

    // Returns a vector (in result) where the Wii-controller's coordinate system
    // has been rotated, so that the natural orientation of the controller forms
    // the base of the measured acceleration
    void Vec3::GetNaturalAccelerationVector(double accX, double accY, double accZ,
                                            Vec3 &result)
    {
```

```
        result.mX = -accX;
        result.mY = accZ;
        result.mZ = accY;
    }
}
// End namespace WiiLib
```

## Mat3

**Listing 30:** /WiiLib/Mat3.h

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_MAT3_H__
#define __WIILIB_MAT3_H__

#include "Vec3.h"

namespace WiiLib
{
    class Mat3
    {
    public:

        double m11, m12, m13;
        double m21, m22, m23;
        double m31, m32, m33;

    public:

        Mat3();

    public:

        inline void     SetIdentity();
        inline void     SetRotation(const Vec3 &axis, double angle);
        inline void     SetAccelerationBaseChange();

        inline void     Transform(const Vec3 &v, Vec3 &result) const;

    };
}
// End namespace WiiLib

#include "Mat3.inl"

#endif
```

**Listing 31:** /WiiLib/Mat3.inl

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#ifndef __WIILIB_MAT3_INLINE__
#define __WIILIB_MAT3_INLINE__

#include <cmath>

namespace WiiLib
{
    inline void Mat3::SetIdentity()
    {
```

```
        m11 = 1; m12 = 0; m13 = 0;
        m21 = 0; m22 = 1; m23 = 0;
        m31 = 0; m32 = 0; m33 = 1;
    }

    inline void Mat3::SetRotation(const Vec3 &axis, double angle)
    {
        double s = sin(angle);
        double c = cos(angle);
        double t = 1.0-c;

        m11 = t*(axis.mX*axis.mX) + c;
        m12 = t*(axis.mX*axis.mY) - axis.mZ*s;
        m13 = t*(axis.mX*axis.mZ) + axis.mY*s;

        m21 = t*(axis.mX*axis.mY) + axis.mZ*s;
        m22 = t*(axis.mY*axis.mY) + c;
        m23 = t*(axis.mY*axis.mZ) - axis.mX*s;

        m31 = t*(axis.mX*axis.mZ) - axis.mY*s;
        m32 = t*(axis.mY*axis.mZ) + axis.mX*s;
        m33 = t*(axis.mZ*axis.mZ) + c;
    }

    inline void Mat3::Transform(const Vec3 &v, Vec3 &result) const
    {
        result.mX = v.mX*m11 + v.mY*m12 + v.mZ*m13;
        result.mY = v.mX*m21 + v.mY*m22 + v.mZ*m23;
        result.mZ = v.mX*m31 + v.mY*m32 + v.mZ*m33;
    }
}
// End namespace WiiLib

#endif
```

**Listing 32:** /WiiLib/Mat3.cpp

```
/*
    This file is part of WiiLib

    Author: Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
    Author: Sabinsky Tøgern <ast@hoast.dk>
*/
#include "Mat3.h"

namespace WiiLib
{
    /*
     * Setup
     */

    Mat3::Mat3()
        :
        m11(0), m12(0), m13(0),
        m21(0), m22(0), m23(0),
        m31(0), m32(0), m33(0)
    {
    }
}
// End namespace WiiLib
```

## C.2 WiiLibUnittests

**Listing 33:** /WiiLibUnittests/main.cpp

```
#include <iostream>
#include <cassert>
#include "WiiLib.h"

#pragma comment(lib, "WiiLib.lib")

using namespace std;
using namespace WiiLib;

void TestConvolution(double *signal, int signalLength,
                     double *kernel, int kernelLength, int kernelCenterIndex,
                     double *result)
{
    Filter *f = new Filter(kernel, kernelLength, kernelCenterIndex);
    Signal s(signalLength, f);
    const double *t;

    // Fill signal
    for (int k = 0; k < signalLength; k++)
    {
        s.Append(signal[k]);
    }

    // Compare results (indirectly performs convolution)
    t = s.GetFiltered();

    for (int k = 0; k < signalLength; k++)
    {
        assert(t[k] == result[k]);
    }
}

void TestIntegrator(double *signal, int signalSize, double *result,
                    Integrator &integrator)
{
    double *t = new double[signalSize];

    // Perform all steps of integration
    for (int k = 0; k < signalSize; k++)
    {
        integrator.Integrate(Vec3(signal[k],0,0),1);

        // Store integrated signal in t
        t[k] = integrator.mV.mX;
    }

    // Compare results
    for (int k = 0; k < signalSize; k++)
    {
        assert(t[k] == result[k]);
    }
}

int main(int argc, char *argv[])
{
    /*
     * Unittests: Convolution with custom filter kernel
     */

    // With centered kernel
    double  sigA[5] = { 1,1,1,1,1 };
    int     sigAsiz = 5;
    double  kerA[3] = { 0.25,0.5,0.25 };
    int     kerAsiz = 3;
    int     kerAidx = 1;
    double  resA[5] = { 0.75,1,1,1,0.75 };

    TestConvolution(sigA, sigAsiz, kerA, kerAsiz, kerAidx, resA);
```

```
70      // With uncentered kernel
71      double  sigB[5] = { 1,1,1,1,1 };
72      int     sigBsiz = 5;
73      double  kerB[3] = { 0.25,0.5,0.25 };
74      int     kerBsiz = 3;
75      int     kerBidx = 0; // note: center index is not the middle element
76      double  resB[5] = { 1,1,1,0.75,0.25 };

78      TestConvolution(sigB, sigBsiz, kerB, kerBsiz, kerBidx, resB);

80      // With typical filter
81      double  sigC[7] = { 8,4,2,1,2,4,8 };
82      int     sigCsiz = 7;
83      double  kerC[3] = { 0.1,0.75,0.1 };
84      int     kerCsiz = 3;
85      int     kerCidx = 1;
86      double  resC[7] = { 6.4,4,2,1.15,2,4,6.4 };

88      TestConvolution(sigC, sigCsiz, kerC, kerCsiz, kerCidx, resC);

90      // With negative values in filter
91      double  sigD[7] = { 8,4,2,1,2,4,8 };
92      int     sigDsiz = 7;
93      double  kerD[3] = { -1,0,1 };
94      int     kerDsiz = 3;
95      int     kerDidx = 1;
96      double  resD[7] = { 4,-6,-3,0,3,6,-4 };

98      TestConvolution(sigD, sigDsiz, kerD, kerDsiz, kerDidx, resD);

100     /*
101      * Unittests: Correctness of the different numeric integrators
102      */

104     // Euler
105     double  signalEul[10]   = { 0,1,2,3,0,-1,0,-2,-3,0 };
106     int     signalEulSiz    = 10;
107     double  resultEul[10]   = { 0,0,1,3,6,6,5,5,3,0 };

109     TestIntegrator(signalEul, signalEulSiz, resultEul, IntegratorEuler());

111     // Trapez
112     double  signalTpz[10]   = { 0,1,2,3,0,-1,0,-2,-3,0 };
113     int     signalTpzSiz    = 10;
114     double  resultTpz[10]   = { 0,0.5,2,4.5,6,5.5,5,4,1.5,0 };

116     TestIntegrator(signalTpz, signalTpzSiz, resultTpz, IntegratorTrapez());

118     // Midpoint (in this case midpoint == euler, since delta t is constant)
119     double  signalMid[10]   = { 0,1,2,3,0,-1,0,-2,-3,0 };
120     int     signalMidSiz    = 10;
121     double  resultMid[10]   = { 0,0,1,3,6,6,5,5,3,0 };

123     TestIntegrator(signalMid, signalMidSiz, resultMid, IntegratorMidpoint());

125     /*
126      * End of unittests
127      */

129     return 0;
130 }
```

## C.3 WiiLib2MatLab

**Listing 34:** /WiiLib2MatLab/main.cpp

```
1 #include <iostream>
2 #include <cmath>
3 #include <ctime>
4 #include <cstdlib>

6 #include "WiiLib.h"

8 #pragma comment(lib, "WiiLib.lib")

10 using namespace std;
11 using namespace WiiLib;

13 int main(int argc, char *argv[])
14 {
15     Controller  c;
16     ofstream    fout;
17     int         samples;
18     Vec3        accGravity;
19     int         accGravitySamples;

21     // Read arguments
22     if (argc < 3)
23     {
24         cout << "Syntax: " << argv[0] << " <filename> <#samples>" << endl;
25         return 0;
26     }

28     // Create data file
29     cout << "Creating data file... ";
30     fout.open(argv[1], ofstream::trunc);
31     if (!fout.is_open())
32     {
33         cout << "Error: Could not create '" << argv[1] << "'." << endl;
34         return 0;
35     }
36     else
37     {
38         fout.precision(16);
39         cout << "OK!" << endl;
40     }

42     // Get number of samples
43     cout << "Getting number of samples... ";
44     samples = atoi(argv[2]);
45     if (samples == 0)
46     {
47         cout << "Error: Could not parse argument." << endl;
48         return 0;
49     }
50     else
51     {
52         cout << "OK!" << endl;
53     }

55     // Connect controller
56     cout << "Connecting... ";
57     if (!c.Connect())
58     {
59         cout << "Error: Could not connect to controller." << endl;
60         return 0;
61     }
62     else
63     {
64         cout << "OK!" << endl;
65     }

67     // Calibrate controller
68     cout << "Calibrating... ";
69     while (!c.IsCalibrated())
```

```
70      {
71          c.CalibrateAsync();
72          c.Sample();
73      }
74      cout << "OK!" << endl;

76      // Wait for motion data
77      cout << "Waiting for motion data... ";
78      c.SetReportMode(WiiLib::Controller::Report_ButtonsMotion, true);
79      while (!c.HasMotionData())
80      {
81          c.Sample();
82      }
83      cout << "OK!" << endl;

85      // Calculate average gravity vector
86      cout << "Ready to calculate static gravity." << endl;
87      cout << "HIT RETURN TO CONTINUE"; getchar();
88      cout << "Calculating average gravity vector... ";
89      accGravitySamples = 1000;
90      for (int i = 0; i < accGravitySamples; i++)
91      {
92          c.Sample();
93          accGravity.Add(c.mStatus.accNatural);
94      }
95      accGravity.mX /= accGravitySamples;
96      accGravity.mY /= accGravitySamples;
97      accGravity.mZ /= accGravitySamples;
98      accGravity.Normalize();
99      fout << scientific  // gravity average
100         << accGravity.mX << "\t"
101         << accGravity.mY << "\t"
102         << accGravity.mZ << endl;
103     fout << scientific  // number of samples
104         << samples << "\t"
105         << 0 << "\t"
106         << 0 << endl;
107     cout << "OK!" << endl;

109     // Record <#samples> samples from controller
110     cout << "Ready to record " << samples << " samples to '" << argv[1] << "'."
111         << endl;
112     cout << "HIT RETURN TO CONTINUE"; getchar();
113     cout << "Recording... ";
114     for (int i = 0; i < samples; i++)
115     {
116         c.Sample();
117         fout << scientific
118             << c.mStatus.accNatural.mX << "\t"
119             << c.mStatus.accNatural.mY << "\t"
120             << c.mStatus.accNatural.mZ << endl;
121         if((i+1)%(samples/5) == 0)
122         {
123             cout << (i+1) << "... ";
124         }
125     }
126     cout << "OK!" << endl;

128     // Close file and exit
129     fout.close();
130     cout << "Done." << endl;

132     return 0;
133 }
```

## C.4 WiiLibTests



### main

**Listing 35:** /WiiLibTests/main.cpp

```
#include <fstream>
#include <iostream>
#include <cmath>
#include <time.h>
#include <String>

#include "Vec3.h"
#include "Experiment.h"
#include "KExperiment.h"
#include "BExperiment.h"
#include "TiltExperiment.h"
#include "PositionExperiment.h"

#pragma comment(lib, "WiiLib.lib")

using namespace std;
using namespace WiiLib;

bool ReadSamples(const char *file, Vec3 &gravity, int &samples, Vec3 *&container)
{
    ifstream fin;

    // Create data file
    fin.open(file);
    if (!fin.is_open())
    {
        return false;
    }

    fin >> gravity.mX;
    fin >> gravity.mY;
    fin >> gravity.mZ;

    fin >> samples;
    double dummy;
    fin >> dummy;
    fin >> dummy;

    //Try catch?
    container = new Vec3[samples];
    for (int i = 0; i < samples; ++i)
    {
        container[i] = Vec3();
        fin >> container[i].mX;
        fin >> container[i].mY;
        fin >> container[i].mZ;
    }

    fin.close();
    return true;
}

bool WriteSamples(const char *file, const Vec3 gravity, const int samples,
                  const double *data)
```

```
55 {
56     ofstream fout;

58     fout.open(file, ofstream::trunc);
59     if (!fout.is_open())
60     {
61         return false;
62     }
63     else
64     {
65         fout.precision(16);
66     }

68     for (int i = 0; i < samples; i++)
69     {
70         fout << scientific << data[i] << endl;
71     }

73     fout.close();

75     return true;
76 }

78 int main(int argc, char *argv[])
79 {
80     Vec3 gravity;
81     Vec3 *data;
82     int samples = 0;
83     int argCnt = argc - 4;
84     bool success = false;
85     double *output;
86     string outfile;

88     if (argc < 3)
89     {
90         cout << "Syntax: " << argv[0]
91             << " <input filename> <output filename> <test-type> [options]"
92             << endl;
93         return 0;
94     }

96     if (!ReadSamples(argv[1], gravity, samples, data))
97     {
98         return 0;
99     }

101     //Work start

103     output = new double[samples];
104     memset(output, 0, sizeof(double)*samples);
105     if (strcmp(argv[3], "k-test") == 0)
106     {
107         KExperiment kex = KExperiment();
108         success = kex.DoExperiment(&argv[4], argCnt, &gravity, samples, data,
109                                    output);
110     }
111     else if (strcmp(argv[3], "b-test") == 0)
112     {
113         BExperiment bex = BExperiment();
114         success = bex.DoExperiment(&argv[4], argCnt, &gravity, samples, data,
115                                    output);
116     }
117     else if (strcmp(argv[3], "tilt") == 0)
118     {
119         TiltExperiment TE = TiltExperiment();
120         success = TE.DoExperiment(&argv[4], argCnt, &gravity, samples, data,
121                                    output);
122     }
123     else if (strcmp(argv[3], "position") == 0)
124     {
125         PositionExperiment PE = PositionExperiment();
126         success = PE.DoExperiment(&argv[4], argCnt, &gravity, samples, data,
127                                    output);
128     }
```

```
    else
    {
        delete [] output;
        return 0;
    }

    outfile = argv[2];
    for (int i = 3; i < argc; ++i)
    {
        outfile += "_";
        outfile += argv[i];
    }
    outfile += ".mat";

    if (success && WriteSamples(outfile.c_str(), gravity, samples, output))
    {
        cout << "Successfully done!" << endl;
    }
    else
    {
        cout << "What arrrrrr' yo' doin'?!" << endl;
    }

    //Work end

    delete [] data;
    delete [] output;

    return 0;
}
```

## Experiment

**Listing 36:** /WiiLibTests/Experiment.h

```
#ifndef __EXPERIMENT_H__
#define __EXPERIMENT_H__

#include "Vec3.h"

using namespace WiiLib;

class Experiment
{
public:

    virtual bool DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
        const int samples, const Vec3 *data, double *output) = 0;

};

#endif
```

## KExperiment

**Listing 37:** /WiiLibTests/KExperiment.h

```
#ifndef __KEXPERIMENT_H__
#define __KEXPERIMENT_H__

#include "Experiment.h"

class KExperiment: public Experiment
{
public:

    virtual bool DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
```

```
        const int samples, const Vec3 *data, double *output);

};

#endif
```

**Listing 38:** /WiiLibTests/KExperiment.cpp

```
#define _USE_MATH_DEFINES

#include <cmath>
#include <iostream>
#include "KExperiment.h"
#include "GravityAdaptive.h"

bool KExperiment::DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
                               const int samples, const Vec3 *data,
                               double *output)
{
    double k = 0.0;
    double kStep = 1.0/(double)samples;
    Vec3 dummy;

    for (int i = 0; i < samples; i++)
    {
        GravityAdaptive gm(*gravity, M_PI/6, k);
        output[i] = 0;

        for (int j = 0; j < samples; j++)
        {
            if (!gm.IsBaseCandidate(data[j]))
            {
                output[i] += 1.0;
            }
            gm.GetEstimate(data[j], dummy);
        }

        k += kStep;
    }

    return true;
}
```

## BExperiment

**Listing 39:** /WiiLibTests/BExperiment.h

```
#ifndef __BEXPERIMENT_H__
#define __BEXPERIMENT_H__

#include "Experiment.h"

class BExperiment: public Experiment
{
public:

    virtual bool DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
        const int samples, const Vec3 *data, double *output);

};

#endif
```

**Listing 40:** /WiiLibTests/BExperiment.cpp

```
#define _USE_MATH_DEFINES

#include <cmath>
```

```
#include <iostream>
#include "BExperiment.h"
#include "GravityAdaptive.h"

bool BExperiment::DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
                               const int samples, const Vec3 *data,
                               double *output)
{
    double b = 0.0;
    double bStep = M_PI/(double)samples;
    Vec3 dummy;

    if (argc < 1)
    {
        return false;
    }

    for (int i = 0; i < samples; i++)
    {
        GravityAdaptive gm(*gravity, b, atof(argv[0]));
        output[i] = 0;

        for (int j = 0; j < samples; j++)
        {
            if (!gm.IsWithinBounds(data[j]))
            {
                output[i] += 1.0;
            }
            gm.GetEstimate(data[j], dummy);
        }

        b += bStep;
    }

    return true;
}
```

## TiltExperiment

**Listing 41:** /WiiLibTests/TiltExperiment.h

```
#ifndef __TILTEXPERIMENT_H__
#define __TILTEXPERIMENT_H__

#include "Experiment.h"

class TiltExperiment: public Experiment
{
public:

    virtual bool DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
        const int samples, const Vec3 *data, double *output);

};

#endif
```

**Listing 42:** /WiiLibTests/TiltExperiment.cpp

```
#include <iostream>
#include "TiltExperiment.h"
#include "Tilt.h"
#include "GravityAdaptive.h"
#include "GravityNaive.h"

using namespace WiiLib;
using namespace std;

bool TiltExperiment::DoExperiment(char *argv[], const int argc,
```

```
                                    const WiiLib::Vec3 *gravity, const int samples,
                                    const WiiLib::Vec3 *data, double *output)
{
    if ((argc < 3) || ((strcmp(argv[0], "adaptive") == 0) && (argc < 5)))
    {
        return false;
    }

    // Create convolution filter
    Filter *f;

    if (strcmp(argv[3], "gauss") == 0)
    {
        cout << "Gauss filter" << endl;
        cout << "  sigma = " << atof(argv[4]) << endl;
        f = Filter::CreateGaussFilter(atof(argv[4]));
    }
    else if (strcmp(argv[3], "pass") == 0)
    {
        cout << "No filter" << endl;
        f = Filter::CreatePassThroughFilter();
    }
    else
    {
        return false;
    }

    // Create tilt estimator
    Tilt tilt(f, 0);
    tilt.SetReference(*gravity);
    GravityAdaptive gmAdaptive(*gravity, atof(argv[2]), atof(argv[1]));
    GravityNaive gmNaive;

    if (strcmp(argv[0], "adaptive") == 0)
    {
        cout << "Adaptive method" << endl;
        cout << "  beta = " << atof(argv[2]) << endl;
        cout << "  k = " << atof(argv[1]) << endl;
        tilt.SetGravityModel(&gmAdaptive);
    }
    else if (strcmp(argv[0], "naive") == 0)
    {
        cout << "Naive method" << endl;
        tilt.SetGravityModel(&gmNaive);
    }
    else
    {
        return false;
    }

    // Run test
    Vec3 dummy;
    double angle = 0.0;

    cout << "Running test" << endl;
    for (int i = 0; i < samples; i++)
    {
        tilt.ProcessSample(data[i]);
        tilt.GetAxisAngle(dummy, output[i]);
    }

    return true;
}
```

## PositionExperiment

**Listing 43:** /WiiLibTests/PositionExperiment.h

```
#ifndef __POSITIONEXPERIMENT_H__
#define __POSITIONEXPERIMENT_H__
```

```
#include "Experiment.h"

class PositionExperiment: public Experiment
{
public:

    virtual bool DoExperiment(char *argv[], const int argc, const Vec3 *gravity,
        const int samples, const Vec3 *data, double *output);

};

#endif
```

**Listing 44:** /WiiLibTests/PositionExperiment.cpp

```
#include <iostream>
#include "PositionExperiment.h"
#include "Position.h"
#include "IntegratorEuler.h"
#include "IntegratorTrapez.h"

using namespace std;

bool PositionExperiment::DoExperiment(char *argv[], const int argc,
                                      const WiiLib::Vec3 *gravity,
                                      const int samples,
                                      const WiiLib::Vec3 *data,
                                      double *output)
{
    if (argc < 4)
    {
        return false;
    }

    // Create convolution filter
    Filter *f;

    if (strcmp(argv[1], "gauss") == 0)
    {
        cout << "Gauss filter" << endl;
        cout << "  sigma = " << atof(argv[2]) << endl;
        f = Filter::CreateGaussFilter(atof(argv[2]));
    }
    else if (strcmp(argv[1], "pass") == 0)
    {
        cout << "No filter" << endl;
        f = Filter::CreatePassThroughFilter();
    }
    else
    {
        return false;
    }

    // Create position estimator
    Position pos(f, 0);
    IntegratorEuler integEuler;
    IntegratorTrapez integTrapez;

    if (strcmp(argv[0], "euler") == 0)
    {
        cout << "Euler integrator" << endl;
        pos.SetIntegrator(&integEuler);
    }
    else if (strcmp(argv[0], "trapez") == 0)
    {
        cout << "Trapez integrator" << endl;
        pos.SetIntegrator(&integTrapez);
    }
    else
    {
        return false;
    }
```

```
    // Run test
    double delta = atof(argv[3]);
    Vec3 result;

    cout << "Running test" << endl;
    for (int i = 0; i < samples; i++)
    {
        pos.ProcessSample(data[i], *gravity, delta);
        //Udlæs kun z-aksen, da kun denne påvirkes af bevægelse i vores "vogn"
        pos.GetPosition(result);
        output[i] = result.mZ;
    }

    return true;
}
```

## C.5 Udvidelse til 3DOT



### WiiControllerExtension

**Listing 45:** /trunk/extensions/wiicontroller/wiicontrollerextension.h

```
1 #ifndef EXTENSIONS_WIICONTROLLER_WIICONTROLLEREXTENSION_H_3DOT
2 #define EXTENSIONS_WIICONTROLLER_WIICONTROLLEREXTENSION_H_3DOT
3 /*
4 // \file extensions/wiicontroller/wiicontrollerextension.h
5 //
6 // Description:
7 // Connects to- and exposes data and events from a Wii-controller.
8 //
9 // \Author Anders Sabinsky Tøgern <ast@hoast.dk>
10 // \Author Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
11 //
12 // \Copyright See COPYING file that comes with the 3DOT distribution
13 */

15 #include <3DOT/extensions/wiicontroller/iwiicontrollerextension.h>
16 #include <extension/baseextension.h>
17 #include <entity/event.h>
18 #include <WiiLib.h>

20 using namespace WiiLib;

22 namespace EXT_WIICONTROLLER_3DOT
23 {
24     /*
25      * Class definition: WiiControllerExtension
26      */

28     class WiiControllerExtension
29         : public CORE_3DOT::BaseExtension<IWiiControllerExtension>
30     {
31     public:

33         WiiControllerExtension(APPLICATION_3DOT::HApplication app);
34         ~WiiControllerExtension();

36     public:

38         virtual void        Initialize(APPLICATION_3DOT::HApplication app,
39                                        EDITOR_3DOT::HEditor editor);

41     public:

43         void                Process();

45         CORE_3DOT::quat     GetOrientation() const;
46         CORE_3DOT::vec3     GetPosition() const;

48         CORE_3DOT::HEvent   GetButtonStatesChangedEvent();
49         CORE_3DOT::HEvent   GetOrientationChangedEvent();
50         CORE_3DOT::HEvent   GetPositionChangedEvent();

52     private:

54         Controller          mController;        // WiiLib controller class
55         Tilt                mEstTilt;           // WiiLib tilt estimator
56         Position            mEstPosition;       // WiiLib position estimator

58         int                 mButtonStates;      // last known button data
```

```
        CORE_3DOT::quat     mOrientation;       // last known orientation
        CORE_3DOT::vec3     mPosition;          // last known position

        CORE_3DOT::HEvent   mButtonStatesChangedEvent;
        CORE_3DOT::HEvent   mOrientationChangedEvent;
        CORE_3DOT::HEvent   mPositionChangedEvent;

    private:

        bool                WiiControllerExtension::ConnectAndCalibrate();
        CORE_3DOT::vec3     WiiControllerExtension::ConvWiiLibVec3(Vec3 &v);

    };
}

#endif
```

**Listing 46:** /trunk/extensions/wiicontroller/wiicontrollerextension.cpp

```
/*
// \file extensions/wiicontroller/wiicontrollerextension.cpp
//
// Description:
// Connects to- and exposes data and events from a Wii-controller.
//
// \Author Anders Sabinsky Tøgern <ast@hoast.dk>
// \Author Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
//
// \Copyright See COPYING file that comes with the 3DOT distribution
*/

#define _USE_MATH_DEFINES

#include "wiicontrollerextension.h"
#include "wiicontrollertask.h"
#include "entityplugins/wiicontrolled.h"
#include <entity/event.h>
#include <3DOT/extensions/base_interfaces/interfaces/itransformable.h>
#include <cmath>

#define DELTA_T 33 // 30 frames per second ~= 33ms between samples

namespace EXT_WIICONTROLLER_3DOT
{
    // Exported symbol (3DOT DLL entry point)
    DLLEXPORT_C CORE_3DOT::IExtension*
    getExtension(APPLICATION_3DOT::IApplication* app)
    {
        return new WiiControllerExtension(app);
    }

    /*
     * Setup
     */

    // Constructor, exports
    WiiControllerExtension::WiiControllerExtension(
        APPLICATION_3DOT::HApplication app)
        :
        CORE_3DOT::BaseExtension<IWiiControllerExtension>(app),
        mButtonStatesChangedEvent(new CORE_3DOT::BaseEvent()),
        mOrientationChangedEvent(new CORE_3DOT::BaseEvent()),
        mPositionChangedEvent(new CORE_3DOT::BaseEvent()),
        mEstTilt(WiiLib::Filter::CreateGaussFilter(1),
                 new WiiLib::GravityNaive()),
        mEstPosition(WiiLib::Filter::CreateGaussFilter(1),
                     new WiiLib::IntegratorMidpoint())
    {
        // Requirements
        AddRequiredEntityPlugin<EXT_BASEINTERFACES_3DOT::ITransformable>();

        // Exports
        AddExportedScheduleTaskWithFactory<IWiiControllerTask,
```

```
                                        WiiControllerTask>( );
        AddExportedEntityPluginWithFactory<IWiiControlled,
                                        WiiControlled>( );
    }

    // Destructor
    WiiControllerExtension::~WiiControllerExtension()
    {
        // Disconnect if connected
        if (mController.IsConnected())
        {
            mController.Disconnect();
        }
    }

    // Initialize
    void WiiControllerExtension::Initialize(APPLICATION_3DOT::HApplication app,
                                        EDITOR_3DOT::HEditor editor)
    {
        // Attempt to connect on startup
        ConnectAndCalibrate();
    }

    /*
     * Processing
     */

    bool WiiControllerExtension::ConnectAndCalibrate()
    {
        if (!mController.IsConnected())
        {
            // Stop sending anything but button changes for now
            mController.SetReportMode(Controller::Report_Buttons, false);

            // Abort if unable to connect
            if (!mController.Connect())
            {
                return false;
            }

            LOG("WiiControllerExtension") << "Connected." << LOGEND();

            // Wait for calibration once connected
            while (!mController.IsCalibrated())
            {
                mController.Sample();
                mController.CalibrateAsync();
            }

            LOG("WiiControllerExtension") << "Calibrated." << LOGEND();

            // Start sending button changes and motion data continuously
            mController.SetReportMode(Controller::Report_ButtonsMotion, true);
            mController.SetLEDs(LED_1|LED_2|LED_3|LED_4);
        }

        // Connected and calibrated
        return true;
    }

    // Returns a CORE_3DOT::vec3 from a WiiLib::Vec3
    CORE_3DOT::vec3 WiiControllerExtension::ConvWiiLibVec3(WiiLib::Vec3 &v)
    {
        return CORE_3DOT::vec3(v.mX, v.mY, v.mZ);
    }

    // This function polls controller class for new data and fires all events
    void WiiControllerExtension::Process()
    {
        int             wiiLibCode;
        WiiLib::Vec3    wiiLibAxis;
        double          wiiLibAngle;
        WiiLib::Vec3    wiiLibPosition;
        WiiLib::Vec3    wiiLibGravity;
```

```
130         CORE_3DOT::quat newOrientation;
131         CORE_3DOT::vec3 newPosition;

133         // Attempt to connect if not connected
134         if (!mController.IsConnected())
135         {
136             // Abort process if unable to connect
137             if (!ConnectAndCalibrate())
138             {
139                 return;
140             }
141         }

143         // Instruct controller class to read next sample
144         wiiLibCode = mController.Sample();

146         // Check result of sample instruction
147         if (wiiLibCode == HB_SUCCESS)
148         {
149             // Check for updated button states
150             if (mController.HasButtonData() &&
151                 mController.mStatus.buttons != mButtonStates)
152             {
153                 // Update button states
154                 mButtonStates = mController.mStatus.buttons;

156                 // Fire button states changed event
157                 mButtonStatesChangedEvent->Fire(this, NULL);
158             }

160             // Check for other updates if motion data is present
161             if (mController.HasMotionData())
162             {
163                 // Process sample in tilt class
164                 mEstTilt.ProcessSample(mController.mStatus.accNatural);

166                 // Get new orientation
167                 mEstTilt.GetAxisAngle(wiiLibAxis, wiiLibAngle);
168                 newOrientation = CORE_3DOT::quat(ConvWiiLibVec3(wiiLibAxis),
169                                                  wiiLibAngle);

171                 // Check if orientation has changed
172                 if (newOrientation.GetS() != mOrientation.GetS() ||
173                     newOrientation.GetX() != mOrientation.GetX() ||
174                     newOrientation.GetY() != mOrientation.GetY() ||
175                     newOrientation.GetZ() != mOrientation.GetZ())
176                 {
177                     // Update orientation
178                     mOrientation = newOrientation;

180                     // Fire orientation changed event
181                     mOrientationChangedEvent->Fire(this, NULL);
182                 }

184                 // Process sample in position class
185                 mEstTilt.GetGravityEstimate(wiiLibGravity);
186                 mEstPosition.ProcessSample(mController.mStatus.accNatural,
187                                            wiiLibGravity, DELTA_T);

189                 // Get new position
190                 mEstPosition.GetPosition(wiiLibPosition);
191                 newPosition = ConvWiiLibVec3(wiiLibPosition);

193                 // Check if position has changed
194                 if (newPosition.GetX() != mPosition.GetX() ||
195                     newPosition.GetY() != mPosition.GetY() ||
196                     newPosition.GetZ() != mPosition.GetZ())
197                 {
198                     // Update position
199                     mPosition = newPosition;

201                     // Fire position changed event
202                     mPositionChangedEvent->Fire(this, NULL);
```

```
                }
            } // if (mController.HasMotionData())
        }
        else // if (wiiLibCode == HB_SUCCESS)
        {
            LOG("WiiControllerExtension") << "Controller::Sample() Errorcode: "
                << wiiLibCode << LOGEND();
        }
    }

    /*
     * Exposed methods
     */

    // Returns orientation of controller (angle-axis => quaternion)
    CORE_3DOT::quat WiiControllerExtension::GetOrientation() const
    {
        return mOrientation;
    }

    // Returns position of controller
    CORE_3DOT::vec3 WiiControllerExtension::GetPosition() const
    {
        return mPosition;
    }

    /*
     * Event handles
     */

    // Returns button states changed event handle
    CORE_3DOT::HEvent WiiControllerExtension::GetButtonStatesChangedEvent()
    {
        return mButtonStatesChangedEvent;
    }

    // Returns orientation changed event handle
    CORE_3DOT::HEvent WiiControllerExtension::GetOrientationChangedEvent()
    {
        return mOrientationChangedEvent;
    }

    // Returns position changed event handle
    CORE_3DOT::HEvent WiiControllerExtension::GetPositionChangedEvent()
    {
        return mPositionChangedEvent;
    }
}
```

**Listing 47:** /trunk/3DOT/extensions/wiicontroller/iwiicontrollerextension.h

```
#ifndef DOT_EXTENSIONS_WIICONTROLLER_IWIICONTROLLEREXTENSION_H_3DOT
#define DOT_EXTENSIONS_WIICONTROLLER_IWIICONTROLLEREXTENSION_H_3DOT

#include <3DOT/extension/iextension.h>

namespace EXT_WIICONTROLLER_3DOT
{
    /*
    GUID info:
        http://www.3dot.dk/wiki/index.php/EXTGUIDs%2C_CLSIDs%2C_and_IIDs
    GUID generator:
        http://emanuelgreisen.dk/stuff/uuid.php?num=3

    GUIDFactory<0x9d40314f, 0xc429, 0x49d5, 0xb26f6c429395a810LL>()
                ^^ used by IWiiControllerExtension
    GUIDFactory<0x406381ed, 0x7e86, 0x4bf8, 0xbe12d5cad67e948bLL>()
                ^^ used by IWiiControllerTask
    GUIDFactory<0x5a3f909a, 0xb500, 0x4b0c, 0xa37a97ad24ed2f80LL>()
                ^^ used by IWiiControlled
    GUIDFactory<0x87d409f6, 0x5043, 0x4965, 0x9a4793dea0875b4bLL>()
    */
    #define IID_IWiiControllerExtension GUIDFactory<0x9d40314f, 0xc429, 0x49d5,\
```

```
                                                0xb26f6c429395a810LL>()
    struct DLLEXPORT IWiiControllerExtension : public CORE_3DOT::IExtension
    {
        // Updates and processes new controller data
        INTF_METH( void Process() );

        // Returns orientation of controller (axis-angle => quaternion)
        INTF_METH( CORE_3DOT::quat GetOrientation() const );
        // Returns position of controller
        INTF_METH( CORE_3DOT::vec3 GetPosition() const );

        // Returns button states changed event handle
        INTF_METH( CORE_3DOT::HEvent GetButtonStatesChangedEvent() );
        // Returns orientation changed event handle
        INTF_METH( CORE_3DOT::HEvent GetOrientationChangedEvent() );
        // Returns position changed event handle
        INTF_METH( CORE_3DOT::HEvent GetPositionChangedEvent() );

        // RTTI block
        DECLARE_RTTI_BEGIN(IWiiControllerExtension, CORE_3DOT::IExtension)
            DECLARE_RTTI_METHOD("Process", IWiiControllerExtension::Process)

            DECLARE_RTTI_METHOD("GetOrientation",
                IWiiControllerExtension::GetOrientation)
            DECLARE_RTTI_METHOD("GetPosition",
                IWiiControllerExtension::GetPosition)

            DECLARE_RTTI_METHOD("GetButtonStatesChangedEvent",
                IWiiControllerExtension::GetButtonStatesChangedEvent)
            DECLARE_RTTI_METHOD("GetOrientationChangedEvent",
                IWiiControllerExtension::GetOrientationChangedEvent)
            DECLARE_RTTI_METHOD("GetPositionChangedEvent",
                IWiiControllerExtension::GetPositionChangedEvent)
        DECLARE_RTTI_END();
    };
    typedef CORE_3DOT::ExtensionHandle<IWiiControllerExtension>
        HWiiControllerExtension;
}

#endif
```

## WiiControllerTask

**Listing 48:** /trunk/extensions/wiicontroller/wiicontrollertask.h

```
#ifndef EXTENSIONS_WIICONTROLLER_WIICONTROLLERTASK_H_3DOT
#define EXTENSIONS_WIICONTROLLER_WIICONTROLLERTASK_H_3DOT
/*
// \file extensions/wiicontroller/wiicontrollertask.h
//
// Description:
// A task that calls WiiControllerExtension::Process as often as possible.
//
// \Author Anders Sabinsky Tøgern <ast@hoast.dk>
// \Author Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
//
// \Copyright See COPYING file that comes with the 3DOT distribution
*/

#include <3DOT/extensions/wiicontroller/iwiicontrollertask.h>
#include <engine/basescheduletask.h>

namespace EXT_WIICONTROLLER_3DOT
{
    /*
     * Class definition: WiiControllerTask
     */

    class WiiControllerTask
        : public CORE_3DOT::BaseScheduleTask<IWiiControllerTask>
    {
```

```
    public:

        WiiControllerTask(CORE_3DOT::HScene scene);

    public:

        void    Execute(double time_global, double time_step);
        double  FirstExecutionTime() const;
        double  ExecutionPeriod() const;
        bool    IsSingleton();

    };
}

#endif
```

**Listing 49:** /trunk/extensions/wiicontroller/wiicontrollertask.cpp

```
/*
// \file extensions/wiicontroller/wiicontrollertask.cpp
//
// Description:
// A task that calls WiiControllerExtension::Process as often as possible.
//
// \Author Anders Sabinsky Tøgern <ast@hoast.dk>
// \Author Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
//
// \Copyright See COPYING file that comes with the 3DOT distribution
*/

#include "wiicontrollertask.h"
#include <3DOT/extensions/wiicontroller/iwiicontrollerextension.h>
#include <3DOT/handles/extensionhandle.h>

namespace EXT_WIICONTROLLER_3DOT
{
    // Constructor
    WiiControllerTask::WiiControllerTask(CORE_3DOT::HScene scene)
        :
        CORE_3DOT::BaseScheduleTask<IWiiControllerTask>(scene)
    {
    }

    // Called by scheduler (args not in use as we only want to execute Process)
    void WiiControllerTask::Execute(double time_global, double time_step)
    {
        HWiiControllerExtension()->Process();
    }

    // This function specifies first execution time
    double WiiControllerTask::FirstExecutionTime() const
    {
        return 0;
    }

    // This function specifies execution interval
    double WiiControllerTask::ExecutionPeriod() const
    {
        return 0;
    }

    // This function specifies only one execution per frame
    bool WiiControllerTask::IsSingleton()
    {
        return true;
    }
}
```

**Listing 50:** /trunk/3DOT/extensions/wiicontroller/iwiicontrollertask.h

```
#ifndef DOT_EXTENSIONS_WIICONTROLLER_IWIICONTROLLERTASK_H_3DOT
#define DOT_EXTENSIONS_WIICONTROLLER_IWIICONTROLLERTASK_H_3DOT
```

```
#include <3DOT/engine/ischeduletask.h>

namespace EXT_WIICONTROLLER_3DOT
{
    /*
    GUID info:
        http://www.3dot.dk/wiki/index.php/EXTGUIDs%2C_CLSIDs%2C_and_IIDs
    GUID generator:
        http://emanuelgreisen.dk/stuff/uuid.php?num=3

    GUIDFactory<0x9d40314f, 0xc429, 0x49d5, 0xb26f6c429395a810LL>()
                ^^ used by IWiiControllerExtension
    GUIDFactory<0x406381ed, 0x7e86, 0x4bf8, 0xbe12d5cad67e948bLL>()
                ^^ used by IWiiControllerTask
    GUIDFactory<0x5a3f909a, 0xb500, 0x4b0c, 0xa37a97ad24ed2f80LL>()
                ^^ used by IWiiControlled
    GUIDFactory<0x87d409f6, 0x5043, 0x4965, 0x9a4793dea0875b4bLL>()
    */
    #define IID_IWiiControllerTask GUIDFactory<0x406381ed, 0x7e86, 0x4bf8,\
                                               0xbe12d5cad67e948bLL>()
    struct DLLEXPORT IWiiControllerTask : public CORE_3DOT::IScheduleTask
    {
        // RTTI block
        DECLARE_RTTI_BEGIN(IWiiControllerTask, CORE_3DOT::IScheduleTask)
        DECLARE_RTTI_END();
    };
    typedef HUnknown<IWiiControllerTask> HWiiControllerTask;
}

#endif
```

## WiiControlled

**Listing 51:** /trunk/extensions/wiicontroller/entityplugins/wiicontrolled.h

```
#ifndef EXTENSIONS_WIICONTROLLER_ENTITYPLUGINS_WIICONTROLLED_H_3DOT
#define EXTENSIONS_WIICONTROLLER_ENTITYPLUGINS_WIICONTROLLED_H_3DOT
/*
// \file extensions/wiicontroller/entityplugins/wiicontrolled.h
//
// Description:
// An entity plugin that maps the orientation and position of the connected
// Wii-controller to selected entities in the scene.
//
// \Author Anders Sabinsky Tøgern <ast@hoast.dk>
// \Author Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
//
// \Copyright See COPYING file that comes with the 3DOT distribution
*/

#include <3DOT/extensions/wiicontroller/entityplugins/iwiicontrolled.h>
#include <entity/baseentityplugin.h>

namespace EXT_WIICONTROLLER_3DOT
{
    /*
     * Class definition: WiiControlled
     */

    class WiiControlled : public CORE_3DOT::BaseEntityPlugin<IWiiControlled>
    {
    public:

        WiiControlled(CORE_3DOT::HEntityPluginFactory factory);

    public:

        void OnAddedToEntity();
        void Serialize(CORE_3DOT::HSerializeStream stream);
        void Update();
```

```
    public:

        void SetWiiControlledOrientation(bool flag);
        bool GetWiiControlledOrientation() const;

        void SetWiiControlledPosition(bool flag);
        bool GetWiiControlledPosition() const;

        void OnOrientationChanged(CORE_3DOT::HRTTIUnknown sender,
                                  CORE_3DOT::HRTTIUnknown data);
        void OnPositionChanged(CORE_3DOT::HRTTIUnknown sender,
                               CORE_3DOT::HRTTIUnknown data);

    private:

        bool mWiiControlledOrientation;
        bool mWiiControlledPosition;

        CORE_3DOT::HEventHandler mOrientationChangedHandler;
        CORE_3DOT::HEventHandler mPositionChangedHandler;

    };
}

#endif
```

**Listing 52:** /trunk/extensions/wiicontroller/entityplugins/wiicontrolled.cpp

```
/*
// \file extensions/wiicontroller/entityplugins/wiicontrolled.cpp
//
// Description:
// An entity plugin that maps the orientation and position of the connected
// Wii-controller to selected entities in the scene.
//
// \Author Anders Sabinsky Tøgern <ast@hoast.dk>
// \Author Lasse Jon Fuglsang Pedersen <fuglsang@diku.dk>
//
// \Copyright See COPYING file that comes with the 3DOT distribution
*/

#include "wiicontrolled.h"
#include <3DOT/extensions/wiicontroller/iwiicontrollerextension.h>
#include <3DOT/handles/extensionhandle.h>
#include <entity/event.h>
#include <3DOT/extensions/base_interfaces/interfaces/itransformable.h>

namespace EXT_WIICONTROLLER_3DOT
{
    /*
     * Setup
     */

    // Constructor
    WiiControlled::WiiControlled(CORE_3DOT::HEntityPluginFactory factory)
        :
        CORE_3DOT::BaseEntityPlugin<IWiiControlled>(factory),
        mOrientationChangedHandler(new CORE_3DOT::BaseEventHandler(this,
            &WiiControlled::OnOrientationChanged)),
        mPositionChangedHandler(new CORE_3DOT::BaseEventHandler(this,
            &WiiControlled::OnPositionChanged)),
        mWiiControlledOrientation(false),
        mWiiControlledPosition(false)
    {
    }

    // This function is called when an instance of the
    // entity plugin is added to an entity
    void WiiControlled::OnAddedToEntity()
    {
        HWiiControllerExtension()->GetOrientationChangedEvent()
            ->AddHandler(mOrientationChangedHandler);
```

```
45          HWiiControllerExtension()->GetPositionChangedEvent()
46              ->AddHandler(mPositionChangedHandler);
47      }

49      // This function is called when serialization is run
50      void WiiControlled::Serialize(CORE_3DOT::HSerializeStream s)
51      {
52          this->BaseEntityPlugin<IWiiControlled>::Serialize(s);
53          // Her sker spændende ting:
54          // - Hvis der loades sættes de angivne variable.
55          // - Hvis der gemmes aflæses de angivne variable.
56          //s->attr("position", mPosition, true);
57          //s->attr("orientation", mOrientation, true );
58          //s->attr("scale", mScale, true);
59      }

61      // This function is called after serialization of ALL objects has run
62      void WiiControlled::Update()
63      {
64          HWiiControllerExtension()->GetOrientationChangedEvent()
65              ->AddHandler(mOrientationChangedHandler);
66          HWiiControllerExtension()->GetPositionChangedEvent()
67              ->AddHandler(mPositionChangedHandler);
68      }

70      /*
71       * Flags
72       */

74      void WiiControlled::SetWiiControlledOrientation(bool flag)
75      {
76          mWiiControlledOrientation = flag;
77      }

79      bool WiiControlled::GetWiiControlledOrientation() const
80      {
81          return mWiiControlledOrientation;
82      }

84      void WiiControlled::SetWiiControlledPosition(bool flag)
85      {
86          mWiiControlledPosition = flag;
87      }

89      bool WiiControlled::GetWiiControlledPosition() const
90      {
91          return mWiiControlledPosition;
92      }

94      /*
95       * Event handlers
96       */

98      // This function is called by WiiControllerExtension;
99      // more specifically the orientation changed event
100     void WiiControlled::OnOrientationChanged(CORE_3DOT::HRTTIUnknown sender,
101                                              CORE_3DOT::HRTTIUnknown data)
102     {
103         // If entity plugin is set to make use of reported orientation
104         if (mWiiControlledOrientation)
105         {
106             // Get transformable entity
107             EXT_BASEINTERFACES_3DOT::HTransformable t = GetEntity();

109             // Get reported orientation
110             CORE_3DOT::quat orientation = HWiiControllerExtension()
111                 ->GetOrientation();

113             // Map reported orientation
114             t->SetOrientationLocal(orientation);
115         }
116     }

118     // This function is called by WiiControllerExtension;
```

```
    // more specifically the position changed event
    void WiiControlled::OnPositionChanged(CORE_3DOT::HRTTIUnknown sender,
                                          CORE_3DOT::HRTTIUnknown data)
    {
        // If entity plugin is set to make use of reported position
        if (mWiiControlledPosition)
        {
            // Get transformable entity
            EXT_BASEINTERFACES_3DOT::HTransformable t = GetEntity();

            // Get reported position
            CORE_3DOT::vec3 position = HWiiControllerExtension()->GetPosition();

            // Map reported position
            t->SetPositionLocal(position);
        }
    }
}
```

**Listing 53:** /trunk/3DOT/extensions/wiicontroller/entityplugins/iwiicontrolled.h

```
#ifndef DOT_EXTENSIONS_WIICONTROLLER_ENTITYPLUGINS_IWIICONTROLLED_H_3DOT
#define DOT_EXTENSIONS_WIICONTROLLER_ENTITYPLUGINS_IWIICONTROLLED_H_3DOT

#include <3DOT/entity/ientityplugin.h>

namespace EXT_WIICONTROLLER_3DOT
{
    /*
    GUID info:
        http://www.3dot.dk/wiki/index.php/EXTGUIDs%2C_CLSIDs%2C_and_IIDs
    GUID generator:
        http://emanuelgreisen.dk/stuff/uuid.php?num=3

    GUIDFactory<0x9d40314f, 0xc429, 0x49d5, 0xb26f6c429395a810LL>()
                ^^ used by IWiiControllerExtension
    GUIDFactory<0x406381ed, 0x7e86, 0x4bf8, 0xbe12d5cad67e948bLL>()
                ^^ used by IWiiControllerTask
    GUIDFactory<0x5a3f909a, 0xb500, 0x4b0c, 0xa37a97ad24ed2f80LL>()
                ^^ used by IWiiControlled
    GUIDFactory<0x87d409f6, 0x5043, 0x4965, 0x9a4793dea0875b4bLL>()
    */
    #define IID_IWiiControlled GUIDFactory<0x5a3f909a, 0xb500, 0x4b0c,\
                                           0xa37a97ad24ed2f80LL>()
    struct DLLEXPORT IWiiControlled : public CORE_3DOT::IEntityPlugin
    {
        // Sets/gets whether or not to make use of the reported orientation
        INTF_METH( void SetWiiControlledOrientation(bool) );
        INTF_METH( bool GetWiiControlledOrientation() const );

        // Sets/gets whether or not to make use of the reported position
        INTF_METH( void SetWiiControlledPosition(bool) );
        INTF_METH( bool GetWiiControlledPosition() const );

        // RTTI block
        DECLARE_RTTI_BEGIN(IWiiControlled, CORE_3DOT::IEntityPlugin)
            DECLARE_RTTI_METHOD("SetWiiControlledOrientation",
                IWiiControlled::SetWiiControlledOrientation)
            DECLARE_RTTI_METHOD("GetWiiControlledOrientation",
                IWiiControlled::GetWiiControlledOrientation)
            DECLARE_RTTI_METHOD("SetWiiControlledPosition",
                IWiiControlled::SetWiiControlledPosition)
            DECLARE_RTTI_METHOD("GetWiiControlledPosition",
                IWiiControlled::GetWiiControlledPosition)
        DECLARE_RTTI_END();
    };
    typedef HUnknown<IWiiControlled> HWiiControlled;
}

#endif
```

# D Synopsis

Synopsis følger fra næste side.

# Indhold

# 1 Indledning

Nintendo har til deres spilkonsol, Wii, valgt at benytte en ny slags controller, der bl.a. indholder et 3-akse accelerometer [Analog Devices, 2006]. Den fungerer trådløst via Bluetooth, hvilket åbner op for handshake med en almindelig PC.

DIKU's game-track gør brug af 3D-motoren „3Dot" til forskellige projekter. Vores mål er at tilføje input, til 3Dot, vha. Wii-controlleren og dens accelerometer.

# 2 Problemformulering

Hvordan kan vi konstruere en driver til en 3D-motor (3Dot), der sørger for kommunikation med og behandling af data fra Wii-controlleren, og faciliterer styring af objekter i 3D?

# 3 Afgrænsning

Vi vil, som udgangspunkt, kun aflæse og behandle data fra knapper samt accelerometeret i controlleren.

Mht. data fra accelerometeret vil vi først og fremmest koncentrere os om at estimere tilt (pitch, roll). Andre anvendelser, f.eks. 3D-gestures eller positionsestimering, vil vi beskæftige os med såfremt vi har tid til overs.

Vi vil tage udgangspunkt i et eksisterende kommunikations-bibliotek, [Forbes, 2006], til formidling af data fra Bluetooth-driveren og udvide dette så det passer til vores behov.

Gennemgående i projektet vil vi benytte C++ til implementation af de forskellige komponenter og vi vil fokusere på Windows som platform for implementationen, da det er denne platform 3Dot primært er lavet til.

Vi vil, i projektet, kun lægge vægt på implementeringen.

# 4 Begrundelse

Wii-controlleren har potentiale for at være et bedre værtøj til styring af objekter i 3D, end de gængse input-devices i brug i dag (mus, joystick). Da controlleren holdes i fri luft giver den mulighed for større bevægelsesfrihed for brugeren, end f.eks. en traditionel mus, som er afhængig af et fast underlag af en vis kvalitet. Samtidig er dens uddata i 3D hvor musens uddata er i 2D.

# 5 Evaluering

Grundlæggende for vores implementation er vores kommunikations-bibliotek.

For at afprøve vores implementation vil vi konstrurere demoer der viser forskellige anvendelser af Wii-controlleren med vores implementation.

For at teste præcisionen af vores tilt-estimat vil vi lave en demo, hvor controlleren er mappet direkte til et plan i 3D. Man kan vha. controlleren vippe planet og få en kugle til at rulle på dette. Vores mål er at få estimeret tilt således, at planets hældning svarer til controllerens fysiske hældning i forhold til tyngdekraften.

En anden demo, som skal vise anvendeligheden af vores implementation, gør det muligt at styre flaps på et fly, hvor roll kontrollerer forholdet mellem flaps i højre og venstre vinge, og hvor pitch kontrollerer afvigelsen fra deres neutrale position.

Evaluering af andre anvendelser af controlleren afhænger af om vi når til dem i projektet. For positionsestimering kunne tænkes en demo hvor et punkt i 3D mappes direkte til controlleren - kan man bevæge controlleren frem og tilbage og ende samme sted? For 3D-gestures kunne tænkes en simpel applikation der blot udskriver den genkendte gesture for et givent bevægelsesmønster. Vi vil således minimere antallet af falske positiver (hvis en bevægelse, som ikke er en gesture, genkendes) og falske negativer (hvis en gesture ikke genkendes, selvom den skulle).

# 6 Arbejdsopgaver

Følgende er en liste over de arbejdsopgaver vi forestiller os i projektforløbet. Punkter mærket med (evt.) indgår ikke i tidsplanen, da vi kun vil arbejde med disse så fremt der er tid til overs.

- Kommunikations-bibliotek
  - Research
  - Bluetooth HID
  - Aflæsning af Wii-controller
- Driver til 3Dot
  - Research
  - Estimering af tilt
  - Positionsestimering (evt.)
  - 3D gestures (evt.)

  - Andet (evt.)
- Demoer
  - Kugle-labyrint (tilt)
  - Flaps (tilt)
  - Andet (evt.)
- Rapport
  - Sammenfatning
  - Konklusion
  - Korrektur

# 7 Tidsplan

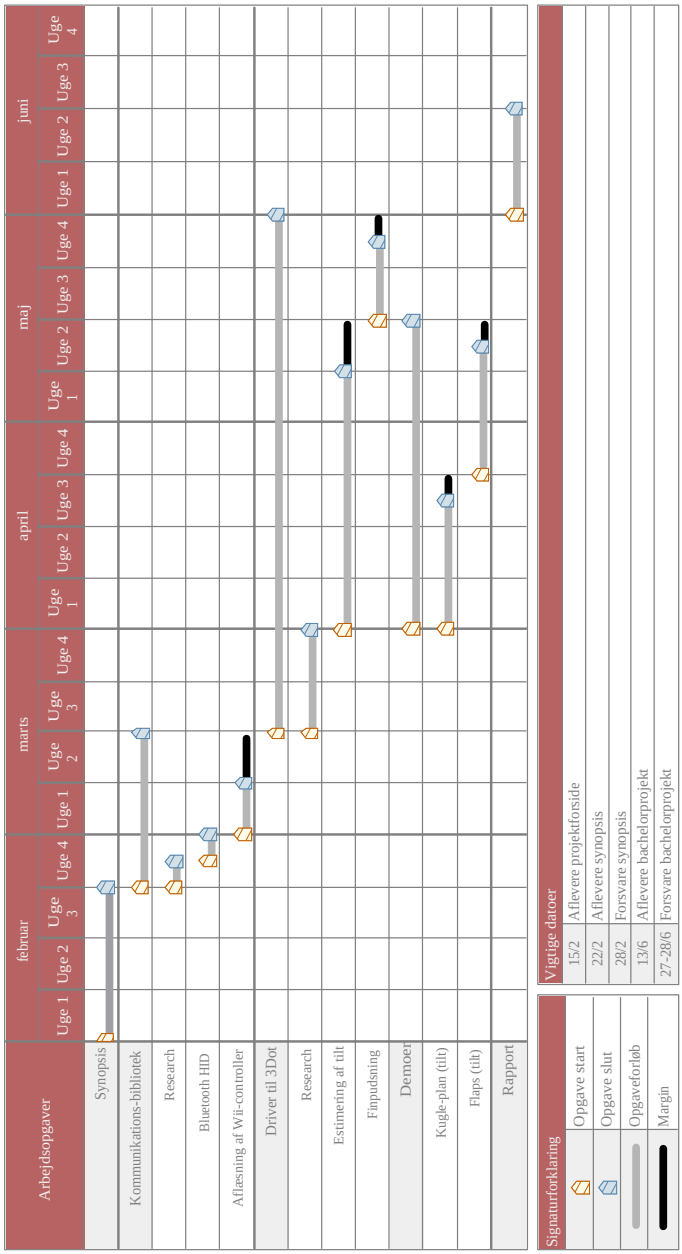
Projektet forløber over 2 blokke. Der er forskellige deadlines i starten og slutningen af projektet. Til selve opgaveløsningen er der 15 uger. Bilag A, på side 6 viser vores foreløbige tidsplan for projektet.

# Litteratur

[Analog Devices, 2006] Analog Devices (2006). ADXL330. http://www.analog.com/UploadedFiles/Data_Sheets/ADXL330.pdf.

[Forbes, 2006] Forbes, K. (2006). cWiiMote 0.2. http://simulatedcomicproduct.com/2006/12/cwiimote-02.php.

# A Tidsplan


Arbejdsopgaver
februar
marts
april
maj
juni
Uge 1
Uge 2
Uge 3
Uge 4
Synopsis
Kommunikations-bibliotek
Research
Bluetooth HID
Aflæsning af Wii-controller
Driver til 3Dot
Research
Estimering af tilt
Finpudsning
Demoer
Kugle-plan (tilt)
Flaps (tilt)
Rapport
Signaturforklaring
Opgave start
Opgave slut
Opgaveforløb
Margin
Vigtige datoer
15/2 Aflevere projektforside
22/2 Aflevere synopsis
28/2 Forsvare synopsis
13/6 Aflevere bachelorprojekt
27-28/6 Forsvare bachelorprojekt