Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Elektronica en informatiesystemen

# Ontwerp en implementatie van een mobiel sensorsysteem voor het traceren van menselijke houding

# Design and Implementation of a Mobile Sensor System for Human Posture Tracking

Benoît Huyghe

Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Elektrotechniek
Academiejaar 2010-2011

ISBN nummer:
NUR code: 959
Depot nummer:

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Elektronica en Infromatiesystemen

Promotoren: Prof. dr. ir. Jan Doutreloigne
Prof. dr. ir. Jan Vanfleteren

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Elektronica en Informatiesystemen
Centrum voor Microsysteemtechnologie (CMST)
Technologiepark 914-A
B-9052 Gent-Zwijnaarde, België

Tel.: +32-9-264.53.50
Fax.: +32-9-264.53.74

Dit werk kwam tot stand in het kader van een aspirantenmandaat van het FWO-Vlaanderen (Fonds Wetenschappelijk Onderzoek – Vlaanderen).

Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Elektrotechniek
Academiejaar 2010-2011

# Dankwoord

Op 1 oktober 2007 begon ik er aan. Een doctoraat. Het zou 4 jaar duren en het leek op dat ogenblik lang, heel lang. Maar achteraf is alles anders, de jaren vlogen en het werk kreeg vorm. Voor ik het besefte was er een werkend systeem, een tastbaar resultaat waarmee ik kon experimenteren en waarover ik het boek heb geschreven dat u nu in uw handen houdt. Dit resultaat was er uiteraard nooit gekomen zonder de hulp en steun van een veelheid aan mensen.

Eerst en vooral dank ik graag mijn beide promotoren, Jan Doutreloigne en Jan Vanfleteren. Elk op hun eigen manier waren ze steeds enthousiast over de geboekte resultaten en lieten ze me de vrijheid om het onderzoek zelf te sturen in welke richting dan ook. Hun eindeloze steun vertaalde zich dan ook in een optimisme dat vaak groter was dan het mijne. Hun vertrouwen stelde mij in staat dit onderzoek te voeren en uiteindelijk de titel van doctor the behalen.

Onze voorzitter André Van Calster verdient mijn dank voor zijn bekommernis om het lot van CMST, inclusief de perikelen met bureauruimte. Verder ook voor zijn expertise op zo goed als alle vlakken, alsook voor tal van sociale activiteiten zoals het jaarlijkse etentje en de legendarische barbecue.

Verschillende mensen hebben een bijdrage geleverd aan het uiteindelijke resultaat en dienen hiervoor zeker en vast vermeld te worden. Nadine voor het uitdenken en uitvoeren van een schitterende choreografie die leidde tot een geslaagd experiment. Michiel van IPEM voor het gebruik van hun tracking systeem en het leveren van het bijhorend videomateriaal. Ine, Veerle en Joeri van de sporwetenschappen die ons graag ontvingen om tests uit te voeren met hun optische opstelling en zich hierbij flink in het zweet werkten.

Naast de technische bijdragen ben ik de hele CMST ploeg enorm dankbaar voor de opperbeste sfeer zowel op als naast de werkvloer. In het bijzonder vemeld ik graag de designers voor de aangename ribbetjes events die talrijk bijgewoond werden en de wekelijkse babbel die zowel Jan als elkaar op de hoogte hield. Ook dank aan de bureaugenootjes, zowel zij die er van in het begin bij waren, Thomas, Jeroen, Rik en David die samen met mij genoodzaakt waren om de kleine Brody te bezetten, als de talrijke anderen die er na de verhuis bijgekomen zijn, Jindrich, Tom, Amir, Sandeep, Sheila en Sanjeev. Verder wil ik ook graag Ann bedanken voor het nalezen van dit boek toen de eerste hoofdstukken op hun plaats vielen, alsook Pietro voor zijn bereidheid om in de jury te zetelen en mij de laatste paar jaar te helpen bij de ontwikkeling van het systeem.

Naast de werkgerelateerde bijdrages zijn er ook een hele boel mensen die mij op persoonlijk vlak heel nauw aan het hart liggen. Niet in het minst, mijn beide ouders. Dankzij jullie was ik er in de eerste plaats letterlijk niet geweest, en had ik dit nooit kunnen bereiken. De onvoorwaardelijke steun die ik tijdens mijn volledige bestaan mocht ervaren is onbeschrijfelijk. Ook mijn broer en zus, ik ben blij dat jullie er zijn.

In tijden van ontspanning kan ik ook steeds rekenen op de talrijke groep mensen die ik tot mijn vrienden mag rekenen. Ik doe hier geen poging om alle namen op te sommen, jullie weten wel wie jullie zijn. Bedankt voor alle verjaardagsfeestjes, de instuif bij iedere verhuis, spelletjesavonden, snookermatchen, passieve en actieve voetbalbijeenkomsten, avonden op café, tripjes en zelfs reizen naar waar-dan-ook, etentjes bij elkaar of op restaurant, concertbezoeken,... Kortom, dank voor alle plezante bijeenkomsten.

Tenslotte moet er nog één persoon speciaal vermeld worden. Eén persoon die elke dag doet opfleuren, simpelweg door er te zijn. Liesbet, merci voor de vele vreugdevolle dagen die we al samen hebben beleefd en voor al diegene die er ongetwijfeld nog zullen komen.

*Gent, november 2011*
*Benoît Huyghe*

# Table of Contents

# List of Figures

# List of Tables

# List of Code Segments

# List of Acronyms

## A

| | |
|---|---|
| AC | Alternating Current |
| ADC | Analogue to Digital Convertor |
| AKF | Adaptive Kalman Filter |
| API | Application Programming Interface |

## B

| | |
|---|---|
| BAN | Body Area Network |
| BCB | Benzocyclobutene |

## C

| | |
|---|---|
| CDKF | Central Difference Kalman Filter |
| CLR | Common Language Runtime |
| CMST | Centre for Microsystems Technology |
| CPU | Central Processing Unit |
| CRC | Cyclic Redundancy Check |

## D

| | |
|---|---|
| DAC | Digital to Analogue Convertor |
| DC | Direct Current |
| DCO | Digitally Controlled Oscillator |
| DOF | Degrees Of Freedom |

## E

| | |
|---|---|
| EEG | Electroencephalogram |
| EKF | Extended Kalman Filter |

## F

| | |
|---|---|
| FPU | Floating Point Unit |

## G

| | |
|---|---|
| GUI | Graphical User Interface |

## H

| | |
|---|---|
| HCI | Host Controller Interface |
| HWM | Hardware Multiplier |

## I

| | |
|---|---|
| IC | Integrated Circuit |
| I$^2$C | Inter-Integrated Circuit |
| ID | Identification |
| IEEE | Institute of Electrical and Electronics Engineers |
| IMU | Inertial Measurment Unit |
| IP | Internet Protocol |
| IPEM | Institute for Psychoacoustics and Electronic Music |
| ISM | Industrial, Scientific and Medical |
| ISR | Interrupt Service Routine |

## J

| | |
|---|---|
| JTAG | Joint Test Action Group |

## K

| | |
|---|---|
| KF | Kalman Filter |

## L

| | |
|---|---|
| LDO | Low Dropout |
| LED | Light Emitting Diode |
| LoS | Lign of Sight |
| LPM | Low Power Mode |

## M

| | |
|---|---|
| MAC | Media Access Control |
| MARG | Magnetic, Angular Rate and Gravitational |
| MEMS | Microelectromechanical System |
| MFG | Magnetic Field and Gravitational |
| ML | Maximum Likelyhood |

## O

| | |
|---|---|
| OpenGL | Open Graphics Library |

## P

| | |
|---|---|
| PC | Personal Computer |
| PDA | Personal Digital Assistant |

## Q

| | |
|---|---|
| QoS | Quality of Service |

# R

| | |
|---|---|
| RF | Radio Frequency |
| RISC | Reduced Instruction Set Computer |
| RMS | Root Mean Square |
| RX | Receive |

# S

| | |
|---|---|
| SN | Sensor Network |
| SoC | System on a Chip |
| SP | Sigma Points |
| SPI | Serial Peripheral Interface |
| SPKF | Sigma Points Kalman Filter |

# T

| | |
|---|---|
| TDMA | Time Division Multiple Access |
| TX | Transmit |

# U

| | |
|---|---|
| UART | Universal Asynchronous Receiver/Transmitter |
| UDP | User Datagram Protocol |
| UKF | Unscented Kalman Filter |
| USB | Universal Serial Bus |
| USCI | Universal Serial Communication Interface |
| UTCP | Ultra Thin Chip Package |
| UV | Ultraviolet |

# V

| | |
|---|---|
| VB | Visual Basic |

## W

| | |
|---|---|
| WBAN | Wireless Body Area Network |
| WDT | Watchdog Timer |
| WSN | Wireless Sensor Network |

# Samenvatting

De reconstructie van menselijke houding en het traceren van bewegingen kan voor vele applicaties een meerwaarde betekenen. Bewegingen van acteurs kunnen worden opgenomen en later gebruikt worden om een digitaal personage te animeren zodat een realistische visualisatie wordt bekomen. Het is een onmisbare technologie in toepassingen met virtuele realiteit gezien de handelingen van de gebruiker moeten getraceerd worden om interactiviteit in de omgeving in te bouwen. Door atleten te volgen tijdens het uitoefenen van hun sport kan de efficiëntie van hun bewegingen geanalyseerd worden en het risico op kwetsuren worden ingeschat en beperkt. Artsen kunnen door biomechanische analyse van patiënten in revalidatie bepalen welke oefeningen moeten worden uitgevoerd voor een beter en sneller herstel.

De combinatie van een steeds snellere evolutie in de ontwikkeling van microsensoren en de opkomst van draadloze sensor netwerken als een gedistribueerde oplossing heeft ertoe geleid dat inertiaalsensoren bruikbaar worden voor het traceren van oriëntatie. Sensor nodes voorzien van accelerometers, magnetometers en gyroscopen leveren drie dimensionale metingen die toelaten om driftvrije absolute oriëntatie te bepalen. Door het menselijke lichaam te benaderen door een set van starre structuren die onderling verbonden zijn door gewrichten kan de houding gereconstrueerd worden wanneer elk van de individuele lichaamsdelen voorzien wordt van een sensor node.

Het complementaire karakter van de sensoren wordt bij inertiële traceersystemen gebruikt om absolute oriëntatie te berekenen. Eerst wordt het signaal van de gyroscopen geïntegreerd om een gegist bestek te verkrijgen van de oriëntatie. Kleine systematische fouten zullen echter snel leiden tot drift op deze schatting. Daarom wordt een correctie toegepast gebaseerd op de metingen van de accelero- en magnetometer. Deze sensoren worden immers verwacht om respectievelijk de gravitatieversnelling en het aardmagnetisch veld te meten. Beiden kunnen gezien worden als een statische referentie die gebruikt kan worden om de initiële schatting te corrigeren. Het bronloze karakter van inertiële traceersystemen is hun grootste pluspunt. Terwijl het bereik van optische systemen beperkt is tot de oppervlakte waar de camera's voldoende zicht op hebben en magnetische systemen enkel bruikbaar zijn waar het gegenereerde veld voldoende sterk is, wordt het bereik van inertiaalsystemen enkel beperkt door de draadloze communicatie.

In dit werk wordt de volledige ontwikkeling van een inertieel traceersysteem zonder gyroscopen toegelicht. Een traceringsalgoritme dat toelaat ori-

ëntatie te schatten van sensormetingen behept met ruis wordt geïntroduceerd. Het gehele systeemontwerp bestaande uit de ontwikkeling van zowel hardware als ingebedde software wordt besproken gaande van de keuze voor de componenten, lay-out van het bord, een draadloos protocol op maat en het inbedden van het schattingsalgoritme. Computer software die toelaat om de gereconstrueerde menselijke houding in real time te visualiseren en een methode om anatomisch incorrecte houdingen te corrigeren op basis van een vereenvoudigd model van de gewrichten worden toegelicht. Uiteindelijk wordt aan de hand van een rotationele positioneringseenheid de traceringsperformantie van een individuele node kwantitatief geanalyseerd en de functionaliteit van het gehele systeem wordt geverifieerd met experimenten waar een persoon volledig wordt gevolgd.

In hoofdstuk 2 wordt een grondig overzicht gegeven van Euler hoeken en quaternionen als representatiemiddel voor drie dimensionale oriëntatie en de Kalman filter met al zijn varianten wordt geïntroduceerd als een algoritme om de informatie van verschillende sensoren te combineren. Zowel de uitgebreide als sigma punt filterarchitectuur wordt toegepast op het probleem gebruik makende van zowel Euler hoeken als quaternionen als voorstellingswijze. Om de performantie van de filters te verbeteren bij de afwezigheid van informatie over de hoeksnelheid worden terugkoppeling en adaptieve ruiscovariantie toegevoegd. Alle parameters van de filters worden geschat gebruik makende van data afkomstig van een traceersessie met een optisch systeem zodat een adequaat model van het probleem wordt verkregen. Een derde orde invers Chebychev filter wordt eveneens voorgesteld om hoog frequente ruis op de uitgangssignalen van de sensoren te verminderen. Simulaties die de kenmerken van het schattingsalgoritme testen en toelaten om de adaptieve parameters te schatten worden aangebracht op het einde van het hoofdstuk. Het stapantwoord illustreert dat de filters zoals verwacht anders reageren op plotse veranderingen in tilt en koers gezien de Z-as samenvalt met de vector van het gravitatieveld. Ruis simulaties bevestigen dat de digitale voorfilter de invloed van hoog frequente ruis op de schattingen vermindert. Ten slotte kan de efficiëntie van de adaptieve aanpak in de filters bepaald worden aan de hand van simulaties waar een sinusoïdaal stoorsignaal wordt toegevoegd aan de uitgang van de accelerometer.

Hoofdstuk 3 behandelt het systeemontwerp bestaande uit zowel hardware als ingebedde software. De hardware kant bevat een overzicht van alle generaties van traceersystemen die ontwikkeld werden doorheen de voorbije jaren. Ieder ontwerp introduceert een nieuw aspect dat bijdraagt tot de finale functionaliteit. De eerste generatie legt de basis van de sensor node architectuur die gebruikt wordt om een volledig operationeel sensor netwerk te bouwen met de tweede generatie. De derde generatie introduceert een laag vermogenverbruik en de mogelijkheid om het netwerk uit te breiden met meer nodes, terwijl in de vierde en laatste generatie vooral wordt gelet op het comfort van de gebruiker door gebruik te maken van geavanceerde bord- en verpakkingstechnieken. Het deel over de ingebedde software handelt zowel over een volledig plug-

and-play, draadloos ad hoc netwerk protocol en een implementatie van het oriëntatie schattingsalgoritme. Het protocol gebruikt een hiërarchie met een meester en verschillende slaven waar data om beurten verzonden wordt. Het ontvangen van een pakket van de meester wordt door de slaven gebruikt om te bepalen wanneer zij zelf hun data mogen verzenden. Door de rol van een sensor node slechts te bepalen bij het opstarten en slaven zelf te laten zoeken naar een beschikbaar tijdslot ontstaat een dynamische implementatie. Gebaseerd op de resultaten van de performantie verkregen in hoofdstuk 2, wordt de quaternion uitgebreide Kalman filter gekozen voor implementatie in ingebedde software. Hiertoe wordt een digitaal formaat met vaste komma geïntroduceerd waarvan de vermenigvuldiging wordt geïmplementeerd door gebruik te maken van de hardware vermenigvuldigingsmodule en de vierkantswortel benaderd wordt met de methode van Newton. De implementatie van de filter maakt verder maximaal gebruik van de voordelen van symmetrische matrices: enkel de beneden driehoek wordt berekend en inversie wordt bekomen door Cholesky decompositie. Uiteindelijk zijn nodes in staat om een schatting te berekenen van de oriëntatie in de beschikbare termijn van 10 ms en kan één basisstation data van maximaal 19 nodes van de derde generatie ontvangen terwijl elk van hen slechts een gemiddeld stroomverbruik van 6.5 mA laat noteren.

De ontwikkeling van computer software wordt behandeld in hoofdstuk 4, beginnende met een overzicht van de structuur van het programma en de gegevensstroom erdoorheen. Belangrijker echter is het tweede deel van dit hoofdstuk waar een menselijk model wordt toegelicht dat gebruikt wordt om anatomisch onmogelijke houdingen te corrigeren naar haalbare houdingen. Een boomstructuur wordt gebruikt om recursief een stokfiguur op te bouwen bestaande uit verschillende botten die overeenkomen met lichaamsdelen en de zogenaamde swing-twist parameterisatie wordt geïntroduceerd om de limieten te beschrijven die eigen zijn aan elk van de gewrichten tussen de botten van het menselijk lichaam. Botten worden onderverdeeld in vier categorieën op basis van de bewegingsvrijheid die toegestaan wordt door het gewricht dat hen verbindt met hun ouder. Vrije botten worden nooit gecorrigeerd en nemen eenvoudigweg de oriëntatie van hun toegekende sensor node over, terwijl vaste botten de oriëntatie van hun ouder overnemen. Het gewricht tussen een bot met correctie in één enkel vlak en zijn ouder wordt gemodelleerd als een scharnier zoals het geval is voor ellebogen en knieën. Botten met correctie in meerdere vlakken worden gekarakteriseerd door een bolgewricht zoals bijvoorbeeld de schouders of heupen. Ten slotte wordt een model voor de omgeving beschreven waar het bot met het laagste eindpunt op een volledig vlakke vloer wordt geplaatst.

In hoofdstuk 5 wordt de performantie van het systeem geanalyseerd met een rotationeel positioneringssysteem en een optisch traceersysteem. Eerst worden de lineariteit, het stapantwoord, de maximale snelheid en de vertraging van het algoritme bepaald door de schattingen te vergelijken met de hoek van het positioneringssysteem. Zowel tilt als koers worden beschouwd gezien de verschillen die blijken uit de simulaties in hoofdstuk 2. Nadien worden dezelfde metingen herhaald met de ingebedde versie van de filter en met een schattingsalgoritme dat wel gebruik maakt van gyroscopen. Uiteraard zal de informatie over de hoeksnelheid er voor zorgen dat de respons van dit laatste filter consistenter en sneller is, maar dat meer tijd nodig is om de correcte eindwaarde te bereiken. In het laatste deel van dit hoofdstuk wordt het systeem getest in experimenten waar een volledig persoon wordt getraceerd en een optisch systeem als referentie wordt gebruikt. Deze experimenten onthullen dat schattingen van het inertiële systeem globaal gezien goed overeenkomen met de optische resultaten zolang de stoorsignalen door beweging op de accelerometer data beperkt blijven.

# Summary

Human posture reconstruction and motion tracking is of interest for many different applications. In animation, captured motion sequences from actors can be mapped to a digital character in order to obtain a realistic visualisation. It is a key technology for building virtual reality since the actions of the user must be captured such that an interactive synthetic environment can be built. Captures of performing athletes allow the analysis of the efficiency of their actions and provide them with feedback for achieving even better results. In rehabilitation, biomechanical analysis enables physicians to determine which exercises should be executed for a better and faster recovery.

The combination of the increasingly fast evolution in the development of micromachined sensors and the rise of wireless sensor networks as a distributed solution has allowed inertial sensors to become a fast emerging technology for orientation tracking. Sensor nodes equipped with accelerometers, magnetometers and gyroscopes supply three dimensional readings that can be used to determine driftfree absolute orientation. By approximating the human body by a set of rigid structures interconnected by joints, posture reconstruction is made possible when each of the individual bodyparts is equipped with a sensor node.

Inertial tracking systems use the complementary nature of the sensors to determine absolute orientation. First, the gyroscope signal is integrated in order to obtain a dead reckoning estimate of the orientation. Since small bias errors will quickly result in drift errors in the estimate, a correction is applied based on the accelerometer and magnetometer readings. The first of these sensors is expected to measure gravity while the latter senses earth's magnetic field. Both are static references that can be used to correct the initial estimate. The sourceless nature of inertial tracking systems forms their main asset. Where the range of optical systems is bound by the area that is adequately covered by the cameras and magnetic systems can only be used where the generated field is strong enough, the range of inertial systems is only limited by the wireless communication link.

In this work, the entire design of a gyroless inertial motion tracking system is outlined. A tracking algorithm is presented that estimates the orientation from noisy sensor readings. The entire system design, covering both hardware and embedded software development is discussed including component choice, board layout, a custom wireless protocol and estimation algorithm embedding. Computer software allowing realtime visualisation of human posture and a method for correcting anatomically incorrect postures using a simplified joint model are presented. Finally, the single node tracking performance is analysed quantitatively using a rotational stage and the entire system functionality is verified with full body tracking experiments.

In chapter 2, a thorough overview is given of Euler angles and quaternions as a means for representing three dimensional orientation and the Kalman filter with all of its variations is introduced as a sensor fusion algorithm. Several flavours of the filter are applied to the problem at hand based on either an extended or a sigma point architecture and using either Euler or quaternion representation. Additional features, such as feedback and adaptive noise covariance are added to improve the performance of the filters under the absence of angular rate information. Using real life motion captures from an optical tracking system, all parameters of the filters are estimated to ensure that an adequate modeling of the problem is obtained. A third order inverted Chebychev filter is also proposed in order to reduce the amount of high frequency output noise of the sensors. Simulations of the estimation algorithm are presented in the final part of the chapter in order to test its characteristics and determine an estimate for the adaptive parameters in the system. Step response simulations show that the filter behaves differently for tilt and heading steps as is expected due to the coincidence of the Z-axis and the gravity vector. Noise response simulations confirm that the sensor output digital pre-filter reduces the influence of the high frequency noise on the filter output. And finally, motion disturbance simulations, where a disturbing sinusoidal signal is added to one of the accelerometer outputs, determine the efficiency of the adaptive filtering approach.

Chapter 3 deals with the system design comprising both hardware and embedded software. On the hardware side, an overview of all generations of motion tracking systems that have been developed throughout the years is given. Each of the designs introduces a new aspect that provides added value compared to the previous system. The first generation lays out the basics of the sensor node architecture which is used to obtain a fully operational sensor network in the second generation. The third generation introduces lower power consumption and the possibility to extend the network with more nodes, while the fourth and final generation concentrates on unobtrusiveness using advanced board and packaging techniques. On the embedded software side, both a fully plug and play, wireless ad hoc network protocol is introduced and an implementation of the orientation tracking filter is discussed. The protocol uses a hierarchy of a master and several slaves where data is transmitted using a turn based approach. The master acts as a synchronisation beacon for the slaves who time their transmission relative to the reception of master packages. A dynamic implementation results when the role of the sensor nodes is determined at runtime and slaves choose their own timeslot according to its availability. Based on the performance determined in chapter 2, the quaternion type extended Kalman filter is selected to be implemented in embedded software. To this extend, a fixed point number format is introduced, multiplication is implemented using the hardware multiplier and the square root is approximated based on the Newton method. The filter implementation makes maximal use of the advantages of symmetrical matrices: only lower triangular parts are calculated and inversion is completed using Cholesky decomposition. Finally, nodes are capable of calculating the orientation estimate within the available 10 ms timeframe and a single base station accommodates a maximum of 19 third generation nodes while each of them consumes an average current of 6.5 mA.

The computer software development is handled in chapter 4 which starts with an overview of the structure and dataflow of the program. More important however, is the second part of this chapter where the human model is outlined that is used to correct anatomically impossible postures into more feasible ones. A tree structure is used to recursively build a stickman consisting of several bones corresponding to bodyparts and the swing-twist parameterisation is introduced to describe the limits that are inherent to each of the joints between the bones in the human body. Bones are divided into four categories depending on the amount of freedom allowed by the joint that connects them to their parent. Free bones are never corrected and simply copy the orientation from their associated sensor node, while fixed bones copy their parent's orientation. The joint between a single plane constraint bone and its parent is modeled as a hinge as is the case for elbows and knees. Multiple constraint bones are characterised by a ball-and-socket type joint as e.g. shoulders and hips. Finally, a world model is described where the bone with the lowest endpoint is clipped to an entirely flat floor.

In chapter 5, the performance of the system is analysed using a rotational stage and an optical tracking system. First, the linearity, step response, maximum speed and delay of the tracking filter is determined by comparing the estimations with the angular position of a rotational stage. Both the tilt and heading case are considered given the difference found in the simulations of chapter 2. Afterwards, these performance measures are compared to the ones obtained with the embedded version of the filter and using an estimation filter with gyroscopes. Naturally, angular rate information makes that the latter displays a more consistent and faster response, yet take more time to reach the correct final value. In the final part of this chapter, the system is tested in full body tracking experiments where an optical tracking system is used as golden standard. The experiments reveal that the output of the inertial system generally corresponds well to the optical output as long as motion disturbance in the accelerometer data is limited.

# 1 Introduction

The introductory chapter outlines the frame in which the research for this work is situated and gives an overview of the existing tracking systems and technologies. An attempt is made to list some of the numerous applications that benefit from this type of technology.

## 1.1 Sensors

In a modern day environment, devices called *sensors* may be found everywhere [1]. The name covers an extremely large family of instruments that are capable of measuring a certain physical quantity and converting it into a signal which can be read by an observer. The measured quantity should be of interest to the user and can be almost anything ranging from temperature [2], pressure [3] or speed [4] to oxygen level [5], toxic particle count [6] or earthquake intensity [7]. The format of the output signal needs to be adapted to the considered observer. In case of a human observer, a visual representation is needed while an observer consisting of another device might be able to handle electrical or mechanical signals.

More and more, sensors are built into everyday products which contribute to a higher life standard. Cars have evolved from fully mechanical machines to electronically controlled integrated systems where additional features are added continuously to increase safety and comfort on one side and meet more stringent environmental restrictions on the other. In this evolution, sensors play a crucial role in providing information for these new control systems

[8]. Accelerometers determine if airbags need to be deployed [9], gas sensors allow selective catalytic reduction systems to control the emission of exhaust fumes [10], parking aid and collision avoidance systems use information from radar sensors to detect the presence of nearby objects or cars [11], the list is virtually endless [12].

The integration of an increasing number of sensors has finally lead to the concept of sensor networks. Many small microsystems, commonly referred to as sensor nodes, are networked in order to provide distributed information to control a system. These sensor nodes consist of the actual sensor and some additional components that allow read-out, data processing, external communication and perhaps even immediate feedback [13]. With the rise of wireless communication, the nodes quickly adopted a wireless interface emphasising their mainly mobile nature. The elimination of wires also meant that a local power supply needed to be present, which in turn implies that power availability is limited. Given these characteristics, dedicated wireless protocols need to be designed to meet the strict requirements of ultra-low power consumption and restricted bandwidth.

Nowadays, sensor networks have found their way to many applications in various fields. In disaster management [14], sensor nodes can be deployed from rescue helicopters to obtain valuable information on the current ground situation. In structural monitoring [15], the state of buildings or bridges can be assessed from sensor readings of nodes embedded inside the structure or attached to its surface. In public transport [16], doors or passenger seats can be monitored easily by sensor nodes. In meteorology and environmental monitoring [17], several parameters can be measured and communicated to monitoring stations.

Aside from the integration of sensors into objects or the environment, the human body is also targeted as a possible host for sensor networks. So-called Body Area Networks (BANs) consist of many different sensors measuring vital functions and are centered around a mobile device, mostly a Personal Digital Assistant (PDA) or smartphone, which operates as an access point to the outside world [18]. Naturally, an important aspect of these BANs is unobtrusiveness, since the system must be virtually invisible to the user [19]. Vital signs that are mostly monitored with BANs include heart pulse rate, blood pressure, glucose level and brain activity [20]. Of special interest for this work however, is the use of a BAN for tracking human motion and posture.

## 1.2 Tracking Systems

Many different types of tracking systems have been developed using a variety of sensor types and physical principles [21]. Since the systems have mostly been designed with a specific application in mind, their performance characteristics are tailored and they may not be suited for a different purpose. Most systems measure a certain signal and reference this to the quantity that must be measured, namely the orientation or position of an object. By approximating the human body by a set of rigid structures interconnected by joints, full body tracking can be obtained by tracking each of these rigid links individually.

A clear distinction must be made between orientation and position tracking. In the first, only the relative orientation referenced to a certain zero direction is determined. In the latter, the location of an object referenced to a certain position in space is traced. Orientation tracking applied to humans leads to human posture, while position tracking yields the location of the person in the room. Some tracking systems are capable of providing both types of tracking, while others can only supply one of them due to technical restrictions.

In general, tracking systems are mostly divided in five classes: mechanical, optical, magnetic, acoustic or inertial [22]. Additionally, systems may also utilise a combination of technologies and form some kind of hybrid class system where the weakness of one type of system may be compensated by another [23]. Although this thesis handles the design of an inertial tracking system, the working principle of each of the five basic types is discussed here, as well as the advantages and disadvantages associated to it. This allows to understand the limits of each of these systems and how inertial tracking inherently differs from other methods.

### 1.2.1 Mechanical Systems

In mechanical motion capture systems, the user is equipped with an exo-skeleton consisting of metal or plastic rods [24]. This exo-skeleton is capable of moving with the user and directly measuring the body joint angles using goniometers within the links between the rods. The measured data can then be stored locally or transmitted via a wireless link to a processing unit where a kinematic algorithm recombines all data to full body posture [25]. Since the system is carried around by the user, no location tracking is provided unless an additional system is present. Figure 1.1 displays a picture of a person equipped with the Gypsy 7 mechanical system designed by Animazoo [26].

The advantages of mechanical tracking include a relatively low cost and occlusion free tracking output since each joint is measured individually. If the system is equipped with a wireless interface, the range of motion is only limited by the range of the interface. Whether or not the system can be used in realtime also depends on the presence of an interface. Tracking more than one person simply requires more exo-skeletons.

A clear disadvantage is the fact that the exo-skeleton limits the freedom of

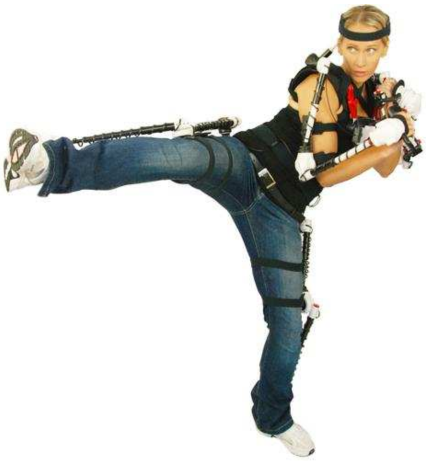


*Figure 1.1: A person equipped with a mechanical tracking system.*

movement of the test person. Rolling over the floor or holding objects e.g. are no longer an option. Also, moving around with the system equipped might hamper the user and result in the user adapting to the system. This in turn leads to less natural behaviour while the purpose of the tracking system is to capture normal motions.

Since the system is placed outside of the body, the center of the goniometers never corresponds to the joint origin. Moreover, human joints never correspond to actual hinges or spherical joints, which makes aligning the goniometers to the body very difficult. This is especially true for complex multiple Degrees Of Freedom (DOF) joints as e.g. shoulders. Furthermore, due to differences in anthropological build, these systems need to be recalibrated for different users.

### 1.2.2 Optical Systems

A variety of different systems may be categorised as optical trackers. The common property among them is that they perform tracking by sensing some sort of light using cameras. The most popular version tracks the location of markers placed on the subject's body with multiple cameras. The markers may be either passive, reflecting light generated by separate sources, or active, by transmitting a certain amount of light themselves. The position of the markers in space is calculated by triangulating the coordinates in each two dimensional capture of the working volume from the cameras. Figure 1.2 shows a picture of a person equipped with markers and a computer generated character adopting the posture of the tracked person. In this case, the markers are spherical reflectors and the light that is used originates from infrared Light Emitting Diodes (LEDs) that are placed next to the cameras.

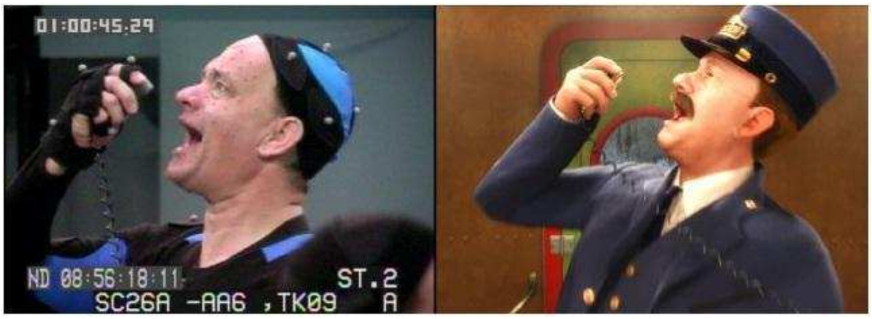


*Figure 1.2: The posture of the person on the left is captured using an optical system and mapped to a computer generated character on the right.*

Full body tracking with markers is obtained by placing at least three markers on hinge locations. The spacial angle between the markers can then be retrieved when the location of the markers in space is known [27]. In order to allow the system to distinguish different bodyparts, unique patterns of markers can be used and associated to a certain bodypart in software [28]. This principle may also be applied to track multiple subjects at once.

This type of technology makes a good effort at being unobtrusive since the markers are almost invisible to the user and allow them to move freely. However, the biggest advantage is found in the high degree of accuracy that can be obtained. Furthermore, not only orientation, but also absolute position is known since the orientation is derived from position.

Several disadvantages are also associated with these systems. First of all, a certain minimum number of cameras need to have Lign of Sight (LoS) to a marker, in order for the software to reconstruct the location of this marker [29]. Whenever this condition is not met, data loss may occur. Second, the tracking space in which tracking can be performed correctly is limited, in fact only a region of a couple of square meter can be covered properly. Third, setting up the equipment is a tedious process where all cameras must be set up to face the tracking space from different angles and an extensive calibration procedure must be executed. Finally, since the entire estimation is based on measuring light intensity, additional light sources can disturb the measurements.

Lately, much effort is also put in markerless optical tracking. Here, camera images are analysed using pattern recognition algorithms in order to track a certain object or bodypart. These systems require a large amount of processing power but remove the need to equip the test subject with markers or a specialised suit at the cost of reduced accuracy. An example of this type of system is Kinect developed by Microsoft for their game console, the XBox 360 [30].

### 1.2.3 Magnetic Systems

Magnetic tracking requires the user to wear several magnetic sensors that measure a set of artificially generated magnetic fields [31]. The sensors consist of three mutually orthogonal coils bearing an induced current due to a changing magnetic field according to electromagnetic theory. The fields are in turn generated by a source also consisting of three mutually perpendicular coils. A full measurement thus results in nine induced currents that can be used to calculate the orientation of the sensor and also the position relative to the emitter since the magnitude of the field is proportional to the distance of the sensor to the source [32].

In general, the source coils may be driven by either Direct Current (DC) or Alternating Current (AC). When DC is used, each coil is driven at a different moment in time in order to distinguish the source of the induced current in the sensor [33]. Also, the effect of the earth magnetic field must be removed by calibration, since this is also a constant field. Using AC allows to differentiate between the different source coils by using different frequencies that can be separated at the sensor side [34].

The advantages of magnetic systems include occlusion free tracking since the human body is almost invisible for the applied magnetic fields and the fact that these systems are relatively cheap compared to optical tracking systems.

Since magnetic fields are used for tracking, the downsides of these systems are immediately related to them. The strength of such a field is inversely proportional to the square of the distance between the source and the sensor. Hence the tracking volume is limited to the size of a small room. Also disturbances gain more interest when the distance increases since the power of the useful field is already weakened. Hence the accuracy of the systems depends on the tracking volume [35].

### 1.2.4 Acoustic Systems

In acoustic systems, the location of an acoustic sensor is determined by either time-of-flight of sound pulses and triangulation [36] or by comparing the phase of the received signal with the phase of a reference signal [37]. The advantage of the first is found in the absolute measure it provides. The latter uses the difference between the current and the previous measurement to determine change in position which is then integrated. The initial position must in this case be known or be determined by other means and drift might occur as a problem.

The accuracy, range and update rate of these systems entirely depend on the physics of sound [38]. This implies a larger range for acoustic systems compared to magnetic ones [39], but a lower sampling frequency due to the limited speed of sound. Clearly, the sensors require LoS to the sound source to provide correct estimates and reflections of sound on walls or objects can disturb the performance.

### 1.2.5 Inertial Systems

With the recent advances in miniaturisation of micromachined sensors, inertial tracking systems have rapidly gained interest. Tracking human posture using these types of systems requires applying sensor nodes on each body segment that needs to be traced [40]. Each of the nodes is equipped with sensors measuring angular rate, linear acceleration and magnetic field in three dimensions and is commonly referred to as a Magnetic, Angular Rate and Gravitational (MARG) sensor node.

A naive approach of determining orientation information from MARG nodes is to simply integrate the angular rate signal. This dead reckoning approach can in fact be used with very large and expensive gyroscopes, yet with cheap micromachined sensors the resulting orientation will quickly exhibit drift due to small bias errors [41]. Therefore, continuous corrections are applied to the orientation based on the measurements supplied by the other sensors. Accelerometers can be used to correct tilt errors based on measurements of earth's gravity and magnetometers sensing earth's magnetic field supply additional information concerning the heading.

If accelerometers would not exhibit any noise or bias, position tracking would be possible by double integration of the signal after removing the contribution of gravity. However, given the errors on the output signal, drift will clearly make this approach impossible for extended periods of time as the error grows quadratically. On short term however, reports have been made of successfully applying double integration by estimating bias drift during phases of zero velocity [42]. This way, a position estimate relative to the starting point can be given.

The biggest advantage of inertial tracking is found in the sourceless nature of the systems. No external source of any kind is needed to provide orientation estimates, since the earth's gravity and magnetic field are omnipresent. This implies that the range of the system is only limited by the communication to the backend and the precision is independent of the location of the subject. Compared to mechanical systems, the main advantage is the fact that sensor nodes can be designed to be very small, whereas the rods with goniometers must essentially cover the entire body.

The downsides of the system include the susceptibility to magnetic interference from active electromagnetic sources as e.g. cell phones and shielding of the magnetic field by metals. Some effort is put in trying to make inertial systems less prone to magnetic disturbance by temporarily relying more on the other sensors [43]. Generally, the accuracy of inertial systems is regarded to be lower than e.g. optical ones, since all estimations are based on the output of very low cost sensors.

## 1.3 Applications

Numerous application may benefit from human posture reconstruction technology or orientation tracking in general. In the following, a far from exhaustive list of applications is given.

### 1.3.1 Animation

Perhaps the most well known application is found in animation. Actors are equipped with technology that allows either their physical movements or facial expressions to be captured digitally. Later on, this data is used to animate a digital avatar such that it mimics the actor's actions.

Obviously, animation is widely used in film industry and examples of the use of motion tracking are not hard to find: *Gollum* in *The Lord of the Rings*, *Davy Jones* in *Pirates of the Caribbean* and the *Na'vi* from *Avatar* are commonly known. Entirely computer animated movies also benefit, think of the tap dancing sequences in *Happy Feet* and the fact that a single actor may pose for several characters as was the case for Tom Hanks in *The Polar Express*.

The technique is also applied in video games for animating certain actions that must be performed by the digital character in order to obtain a realistic animation. Typical examples are found in action games where fighting sequences are taken from martial arts specialists and sports games where the movements of actual athletes are used.

Although the use of motion capture has many advantages including the mentioned possibility to capture a single actor for multiple characters, creating realistic animations in situations that are too dangerous to actually film and the fact that costumes e.g. no longer need to be made, there is also an objection to it. Contrary to the case of *Beowulf* for example, where a digital version of the actors was built to their image, one could hire cheap, unknown actors to control the actions of a digital version of famous actors.

### 1.3.2 Virtual Reality

Taking animation one step further leads to the concept of virtual reality where a user is able to interact with a synthetic environment. This powerful concept provides mankind with a different view to the world and allows to think about a whole new way of how society should function. In education, difficult concepts may be explained using visualisations and actually experiencing the effects. Training of athletes in difficult or fundamentally different environments is made possible without having to travel large distances or exposing them to possible hazards. Coworkers are able to work together on a problem even if they are physically separated from each by hundreds of kilometers reducing travel costs. The entertainment industry can build worlds that only existed in their imagination and people could actually walk around in them and experience it with all their senses.

In order to provide full immersion into a virtual world, the ability to interact with the synthetic environment is a key factor. Most of the interaction is achieved by simple body motion, which in turn leads to changes in the environment that can be observed through each of our five senses. Hence, the first requirement for interaction is the ability to measure or capture body posture and movement. When expanding the concept to multiple users at once in a wide area environment, inertial sensing is probably the only technology allowing virtual reality to be implemented.

### 1.3.3 Biomechanical Analysis

The analysis of human motion during various tasks or exercises is of interest for different fields. In the medical world, motion capture helps in both rehabilitation after an injury or stroke [44] and diagnosis where the detection of oncoming problems is of major interest [45]. In sport science, accurately tracking athletes offers insight in the effectiveness of the execution and the potential risk of injuries.

Aside from orientation estimation, inertial sensors themselves may already be used in medical applications. Accelerometer signals for example are used to measure breathing activity [46] or monitor the possibility of occurrence of the sudden infant death syndrome [47]. Accelerometry has also been applied to improve the detection of epileptic seizures [48].

The mobile nature of inertial sensing also opens the path to home monitoring of patients. This way, therapists can determine if the patient is actually performing the home exercises that have been prescribed. Furthermore, information from the tracking system can provide much faster and more accurate feedback allowing the therapist to alter the exercises such that process of rehabilitation is accelerated or adjusted to the needs of that specific patient.

### 1.3.4 Industrial Applications

Aside from tracking human motion, orientation tracking in general also has many applications for monitoring objects. In robotics, feedback of the orientation is crucial for stabilisation of walking robots. This principle is also widespread among hobbyists building miniature planes and helicopters in order to avoid their work from crashing down. But also the originals may benefit from tracking: ship and plane movements, as well as race car roll can be monitored.

## 1.4 Inertial Motion Tracking

Given their proven track record and years of development, optical systems are generally regarded as the *golden standard* in human motion analysis. Although their accuracy is very high, the limitations of these systems mostly require tracking sessions to take place in specialised laboratories due to the long setup time and expensive nature of the systems. This approach does not allow captures of everyday life activities to be taken in a regular environment as many of the applications listed in the previous section might require. In this regard, the portability of the inertial systems resulting from their sourceless nature is clearly an important asset. Furthermore, with the vast amount of effort that is put in the development of better Microelectromechanical System (MEMS) sensors, inertial tracking has a bright future ahead.

### 1.4.1 Inertial Sensors

An accelerometer consists of a very small mass element that is suspended by spring-like structures. Acceleration causes the mass to exhibit displacement from its neutral position, which is mostly measured using capacitive sensing techniques. Given the fact that a mass is used to measure acceleration explains the fact that an accelerometer measures earth's gravity whenever it is in a stationary position.

A practical implementation of the sensing principle is shown in figure 1.3. The sensor consists of a solid mass that is suspended on either side with a spring to an anchor contact. The mass bears an electrode comb with fingerlike structures that fit into complementary fingers of a fixed comb. When the mass is subjected to accelerations, the distance between the fingers changes proportionally according to Hooke's law, which is in turn sensed capacitively by the comb electrodes [49].

Several types of magnetometers exist, though integrated sensors are commonly based on the Hall effect [50]. The Hall effect dictates that when a

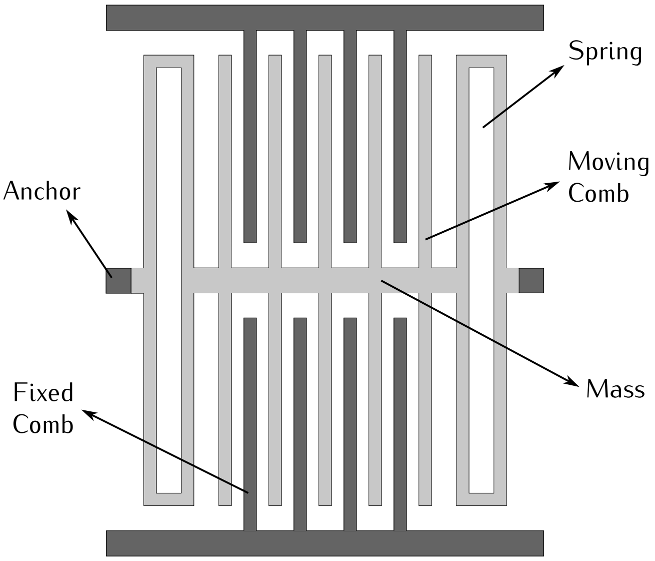


*Figure 1.3: Schematical view of a MEMS accelerometer implementation.*

magnetic field is present perpendicular to a current bearing conductor, a voltage is generated across this conductor in the direction perpendicular to both the magnetic field and the current. This voltage also reflects the polarity of the magnetic field. Strictly speaking no mechanical parts are present in these sensors, yet the signals originating from Hall sensors are very low and require proper amplification, which indicates that extra electronics are also integrated aside from the sensor itself.

Recently, MEMS magnetometers have been built using the Lorentz force. According to electromagnetic theory, a current bearing wire placed in a magnetic field experiences a force perpendicular to both the current and the field:

$$\boldsymbol{F}_{Lorentz} = I\,\boldsymbol{l} \times \boldsymbol{B}, \tag{1.1}$$

where $I$ represents the current, $\boldsymbol{l}$ the length of the conductor, $\boldsymbol{B}$ the magnetic field strength and × denotes the vector cross product. A practical implementation is shown in figure 1.4. The coil shaped conductor bears a constant current and is prone to a Lorentz force when an out-of-plane magnetic field is present. A field directed outward causes both sides of the coil to move closer to the sensing electrodes, an inward field causes a force in the opposite direction. The displacement of the wires then serves as a measure for the magnetic field strength [51].

MEMS gyroscopes are miniature versions of vibrating structure gyroscopes

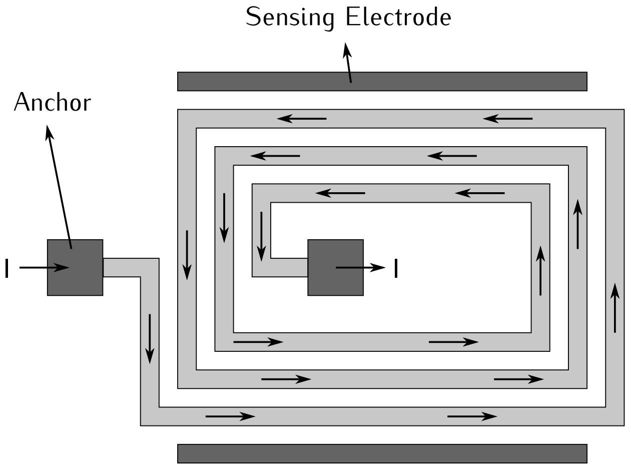


*Figure 1.4: Schematical view of a MEMS magnetometer implementation.*

and use Coriolis acceleration to detect angular speed. Mostly tuning fork type vibrating bodies are used. When this tuning fork is set to vibrate at its fundamental frequency and is subjected to an external rotation about its axis, the Coriolis effect introduces an out-of-plane sinusoidal movement with an amplitude proportional to the angular rate of rotation.

Figure 1.5 shows an implementation of such a tuning fork gyroscope. Both masses are set to oscillate by the driving electrodes and display in plane vibrations in antiphase in the direction of the arrows drawn on them. When the structures are subjected to a rotation around the vertical axis denoted by $\Omega$, movement is introduced by the Coriolis effect where the masses start vibrating out-of-plane. This displacement is then measured by sensing electrodes located underneath the masses [52].

## 1.4.2 State of the Art

At the time of writing, several research institutions, as well as commercial companies have reported activities related to inertial motion tracking. The work ranges from the design of sensor network protocols to entire motion capturing suits.

### 1.4.2.1 Wireless Sensor Networks

The increasing number of applications for sensor networks is reflected in the amount of different protocols that have been designed. This evolution results directly from the fact that each application requires a different kind of sensor

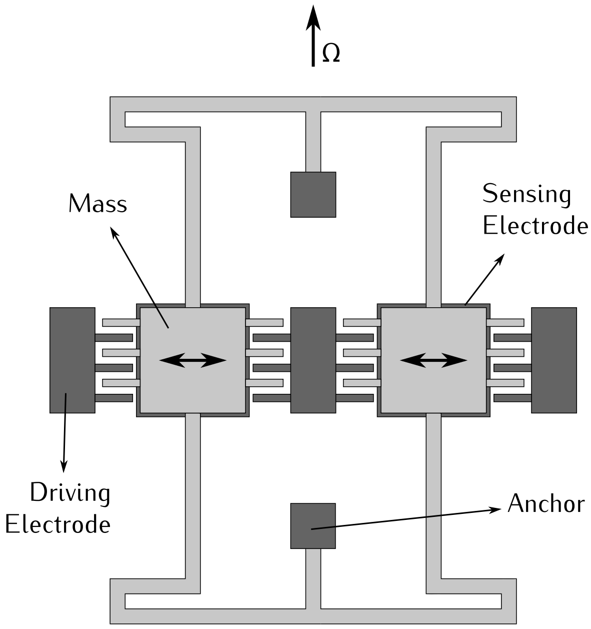


*Figure 1.5: Schematical view of a MEMS gyroscope implementation.*

network since other performance characteristics are stressed. Various parameters need to be taken into account when designing a sensor network and a wireless protocol [53]:

- Range: Some applications only require a very short range of several meters, while others will demand ranges of up to several kilometers.
- Scale: Sensor networks can range from a couple of sensor nodes up to vast amounts of several thousands.
- Latency: Depending on the application, data might need to reach the base station within a very short time or a certain delay could be allowed.
- Loss: Loss of data could be tolerated in some applications, while others demand 100 % of the collected data to be delivered at the base station.
- Dynamics: Nodes might be moving around while measuring, requiring the network to adapt data routing.

Much research effort is put in building powerful protocols that can be used to network many sensors in a harsh and widespread environment [54]. As a result, the focus of the protocols lies in the ad hoc nature where dynamic reorganisation is important and the ability to support multi-hop communication in a mesh network while maintaining a low power consumption is desired [55]. Also, the presented protocols are mostly designed for generic sensor networks, independent of the actual sensor that is used [56].

#### 1.4.2.2 Motion Tracking Sensor Nodes

In sensor node hardware, a distinction must be made between generic sensor network platforms and specifically designed motion tracking nodes.

General purpose sensor nodes allow the user to choose which sensors need to be incorporated on the nodes. Examples of this approach found in the research field are *TinyNode* from Lausanne in Switzerland [57] and UC Berkeley's *Telos* [58] and from a more commercial angle, Sun *SPOT* [59]. Although this offers an extensive amount of freedom and enables the design to be sold to several end users for different applications, the final system is not the most efficient solution. In general, sensors are added by connecting several sensor boards to a main board containing the processing and communications unit. The result is a multiboard sensor node where large connectors are required for interconnection between the boards. Furthermore, given the broad scope of these platforms, the nodes implement too much functionality for any given task. Therefore, the current consumption will always be higher compared to tailor made sensor nodes. The advantage however, is that quick prototyping and a short time to market can be achieved and the cost is logically lower.

Sensor nodes designed specifically for motion tracking are found in both the commercial market and the research world. Restricting the overview to wireless devices, the *MTw* by XSens [60], Intersense's *InertiaCube* [61] and the *ProMove-2* from Inertia Technology [62] are only a couple of the motion tracking nodes available on the market. Each of them is equipped with sensors for measuring acceleration, angular rate and magnetic field in three dimensions, has an on board rechargeable battery and is capable of transmitting data wirelessly to a custom base station. Similar solutions have been built in research facilities, two examples are *Orient* from the University of Edinburgh [63] and *WIMS* from Tyndall National Institute [64]. The main advantage of these systems is their compact size compared to the commercial ones.

#### 1.4.2.3 Inertial Tracking Algorithms

The inertial tracking algorithms can essentially be divided into two major groups: the estimators based on a Kalman filter [65] and the ones based on complementary filtering [66]. All of them exploit the complementary nature of the inertial sensors where gyroscopes are more suited for fast rotations of short duration and accelerometer and magnetometer signals are used to capture low frequency variations allowing driftless estimation.

Since the introduction of this technique, some interesting variations have been elaborated in order to improve the performance. A first point of focus is reducing the sensitivity of the system to disturbances. By including an estimate of the magnetic disturbance in the sensor model, Roetenberg et al. have managed to reduce the output error significantly during magnetic interference [43]. Young et al. determined that the main source of errors in human motion tracking is due to linear accelerations and by using a body model, these accelerations may be estimated [67]. Harada et al. have focused on the compensation of both magnetic and motion disturbance by switching models in the Kalman filter [68].

Other researchers have attempted to reduce the amount of sensors needed for tracking. Luinge et al. describe a method for estimating the orientation of the upper arm with respect to the lower arm using only accelerometers and gyroscopes. The resulting heading error between the two segments is corrected using knowledge about the limitations of the elbow joint [69]. Yun et al. suggest to perform tracking without the use of a gyroscope. The proposed algorithm sequentially determines the orientation quaternion and only uses accelerometer values to determine tilt such that magnetic interference only influences the heading [70].

#### 1.4.2.4 Motion Tracking Suit

There are two companies offering full body motion tracking suits: the *MVN* from XSens [71] and the *IGS* by Animazoo [72]. Both systems use multiple tracking nodes that are connected through a wired network to a wireless communication module. The entire system is embedded in a Lycra suit that comes in different sizes. Batteries power the nodes and the wireless module allowing continuous functionality for about three hours.

Although these suits are very well suited for human motion tracking, this is the only application the system can be used for. When thinking of tracking animals or objects for example, they are no longer usable. Also, since the location of the sensors on the human body is fixed, these suits are not fit for applications where precise knowledge of the distributed orientation of a bodypart is required. In rehabilitation for example, accurate information of the spinal orientation in any given point may be of interest.

## 1.5 Scope and Goals

The final objective of this work is to create an inertial tracking system based on a wireless sensor network topology. Given that a mobile application is targeted, wearability and user comfort should be maximised, while power consumption and cost should be minimised. Primarily targeting human motion capturing, the system should consist of at least 15 nodes considering this is the amount of bodyparts that is mostly used for modeling [73]. Furthermore, a sensor sampling rate of 100 Hz is targeted assuming the bandwidth of human movement can be restricted to 5 Hz and applying the rule of thumb that calls for 20 times oversampling of noisy data [66].

An important step in minimising the current consumption is trying to remove as much components as possible without compromising the functionality. One of the goals of this work is to determine the performance that can be achieved when omitting the gyroscope in the sensor nodes. Since these devices consume much more current compared to accelerometers and magnetometers, and driftless orientation can in fact be determined from these sensors alone, the functionality is not jeopardised while the reduction in current consumption should be significant. However, since the angular rate data supplies valuable information about high frequency movements, it is expected that removing the gyroscope will have some effect on the eventual performance of the system when looking at the dynamic behaviour.

Aside from the gyroscope, the wireless interface contributes the biggest part to the consumption. As this component is clearly needed to fulfill the required functionality, a protocol should be used where the amount of transferred information is restricted to the absolute minimum. On the other hand, the fact that the system must work in realtime implies that the delay should be very low and data cannot be accumulated over time. This indicates that the application of sensor networks for inertial tracking requires a very specific protocol that is mainly data driven. Therefore, the design of a protocol able to handle the transmission of sensor data from at least 15 different nodes at a rate of 100 Hz with minimal delay and current consumption forms another goal.

In order to satisfy the wearability requirement, the hardware design should be approached with caution. With the increasingly fast evolving market of MEMS sensors, angular rate sensors are now becoming available as three dimensional integrated devices. As this is only a recent development, all of the sensor nodes listed in the state of the art consist of multiple boards that are interconnected in a perpendicular fashion. The result is a multiboard sensor node with a considerable height. This was an additional reason to attempt to create a gyroless tracking system as it would allow the design of single board nodes. A flat solution would indeed offer a higher degree of comfort to the user. The goal for the hardware design has not changed however, nodes should consist of a single board solution and size should be minimised as much as possible.

## 1.6 Outline

Chapter two starts with an introduction to mathematical representations of orientation and the Kalman filter as a sensor fusion algorithm. Then, both principles are used to design an orientation tracking algorithm where only accelerometer and magnetometer data is utilised for estimation. Finally, the optimal values for the parameters used in the algorithm are estimated and simulations are performed to verify the functionality and to determine the performance.

The system design is handled in the third chapter. The different hardware designs that have been made throughout the course of this thesis are outlined and the design of future generations is shown. A fully dynamic, ad hoc wireless protocol is conceived to reliably transfer information from the sensor nodes to the backend and its principles are implemented on different hardware generations. The final section of this chapter covers the embedded implementation of the tracking estimator designed in the second chapter.

The fourth chapter describes the real time software aspects of the system. The first part gives an overview of the software structure and the data flow through the program. The second part handles the implementation of a human model for the stickman visualisation in order to correct anatomically incorrect postures to feasible ones.

Chapter five covers the measurement results of conducted experiments. The performance of a single node is analysed by comparing the estimated orientation with the angular position of a rotational stage to which the node is attached. Afterwards, the functionality of the entire system is validated through several full body tracking experiments.

Finally, a general conclusion is formulated in chapter six.

## 1.7 Publications

### Papers published in an SCI-journal

- B. Huyghe, H. Rogier, J. Vanfleteren and F. Axisa, *Design and Manufacturing of Stretchable High-Frequency Interconnects*, IEEE Transactions on Advanced Packaging, Vol. 31(4), November 2008, pp. 802-808.

### Papers submitted to an SCI-journal

- B. Huyghe, J. Doutreloigne and J. Vanfleteren, *A Wireless Sensor Network Protocol for an Inertial Motion Tracking System*, Wireless Personal Networks, *Submitted - Under Review.*

- B. Huyghe, P. Salvo, J. Doutreloigne and J. Vanfleteren, *Feasibility Study and Performance Analysis of a Gyroless Orientation Tracker*, IEEE Transactions on Instrumentation and Measurement, *Submitted - Under Review.*

**Papers presented at international conferences listed as P1-publications**

- B. Huyghe, J. Doutreloigne and J. Vanfleteren, *3D Orientation Tracking Based on Unscented Kalman Filtering of Accelerometer and Magnetometer Data*, Proceedings of the IEEE Sensors Applications Symposium, February 2009, New Orleans, Louisiana, USA, pp. 148-152.

- B. Huyghe, J. Vanfleteren and J. Doutreloigne, *Design of Flexible, Low-Power and Wireless Sensor Nodes for Human Posture Tracking Aiding Epileptic Seizure Detection*, Proceedings of the 8th IEEE Conference on Sensors, October 2009, Christchurch, New Zealand, pp. 1963-1966.

**Papers presented at international conferences without proceedings**

- J. Vanfleteren, F. Axisa, D. Brosteaux, F. Bossuyt, E. De Leersnyder, T. Vervust, B. Huyghe, J. Missinne, R. Verplancke and M. Gonzalez, *Elastic electronic circuits and systems using Moulded Interconnect Device (MID) technology*, Abstracts of the MRS Spring Meeting, 2008, San Francisco, California, USA.

**Papers presented at national conferences**

- B. Huyghe, *Human body posture tracking using flexible, wireless and low-power sensor nodes* , 10th FirW PhD Symposium, Faculty of Engineering, Ghent University, December 2009.

# References

[1] J. Wilson. *Sensor Technology Handbook*. Elsevier, December 2004.

[2] L. Michalski, K. Eckersdorf, J. Kucharski, and J. McGhee. *Temperature Measurement*. John Wiley & Sons, London, UK, 2001.

[3] R.P. Benedict. *Fundamentals of Temperature, Pressure and Flow Measurements*. Wiley, September 1984.

[4] W.-C. Wang. *A Motor Speed Measurement System Based on Hall Sensor*. In Ran Chen, editor, Intelligent Computing and Information Science, volume 134 of *Communications in Computer and Information Science*, pages 440–445. Springer Berlin Heidelberg, 2011.

[5] J.G. Webster. *Design of Pulse Oximeters*. IOP Publishing, Bristol, UK, 1997.

[6] P. Metilda, K. Prasad, R. Kala, J.M. Gladis, T. Prasada Rao, and G.R.K. Naidu. *Ion Imprinted Polymer Based Sensor for Monitoring Toxic Uranium in Environmental Samples*. Analytica Chimica Acta, 582(1):147–153, 2007.

[7] S. Stein and M. Wysession. *An Introduction to Seismology, Earthquakes and Earth Structure*. Blackwell Publishing, 2003.

[8] P. Kleinschmidt and F. Schmidt. *How Many Sensors Does a Car Need?* Sensors and Actuators A: Physical, 31(1–3):35–45, 1992.

[9] K.H. Kim, J.S. Ko, Y.-H. Cho, K. Lee, B.M. Kwak, and K. Park. *A Skew-Symmetric Cantilever Accelerometer for Automotive Airbag Applications*. Sensors and Actuators A: Physical, 50(1–2):121–126, 1995.

[10] R. Moos, R. Müller, C. Plog, A. Knezevic, H. Leye, E. Irion, T. Braun, K.-J. Marquardt, and K. Binder. *Selective Ammonia Exhaust Gas Sensor for Automotive Applications*. Sensors and Actuators B: Chemical, 83(1–3):181–189, 2002.

[11] M. Klotz and H. Rohling. *24 GHz Radar Sensors for Automotive Applications*. In 13th International Conference on Microwaves, Radar and Wireless Communications, volume 1, pages 359–362, 2000.

[12] W.J. Fleming. *Overview of Automotive Sensors*. Sensors Journal, IEEE, 1(4):296–308, December 2001.

[13] G. Meijer. *Smart Sensor Systems*. Wiley, September 2008.

[14] S. Sana and M. Matsumoto. *A Wireless Sensor Network Protocol for Disaster Management*. In Proceedings of Information, Decision and Control, pages 209–213, Adelaide, Australia, February 2007.

[15] X. Ning, S. Rangwala, K.K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. *A Wireless Sensor Network for Structural Monitoring*. In Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pages 13–24, Baltimore, MD, USA, 2004.

[16] S. Kootkar and Z. Al-Ars. *Design and Implementation of Reliable Wireless Sensor Networks - A Case Study in Commuter Trains*. In Proceedings of the ProRISC Workshop, pages 201–204, Veldhoven, Netherlands, November 2007.

[17] J.D. Lundquist, D.R. Cayan, and M.D. Dettinger. *Meteorology and Hydrology in Yosemite National Park: A Sensor Network Application*. In Proceedings of the 2nd International Conference on Information Processing in Sensor Networks, pages 518–528, Palo Alto, CA, USA, 2003.

[18] E. Monton, J.F. Hernandez, J.M. Blasco, T. Herve, J. Micallef, I. Grech, A. Brincat, and V. Traver. *Body Area Network for Wireless Patient Monitoring*. Communications, IET, 2(2):215–222, February 2008.

[19] B. Huyghe, H. Rogier, J. Vanfleteren, and F. Axisa. *Design and Manufacturing of Stretchable High-Frequency Interconnects*. IEEE Transactions on Advanced Packaging, 31(4):802–808, November 2008.

[20] B. Gyselinckx, C. Van Hoof, J. Ryckaert, R.F. Yazicioglu, P. Fiorini, and V. Leonov. *Human++: Autonomous Wireless Sensors for Body Area Networks*. In Proceedings of the IEEE Custom Integrated Circuits Conference, pages 13–19, September 2005.

[21] A. Menache. *Understanding Motion Capture for Computer Animation*. Elsevier, 2011.

[22] G. Welch and E. Foxlin. *Motion Tracking: No Silver Bullet, But a Respectable Arsenal*. Computer Graphics and Applications, IEEE, 22(6):24–38, November 2002.

[23] T. Auer and A. Pinz. *Building a Hybrid Tracking System: Integration of Optical and Magnetic Tracking*. In Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality, pages 13–22, 1999.

[24] I.E. Sutherland. *A head-mounted three dimensional display*. In Proceedings of the Joint Computer Conference, Part I, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA, 1968. ACM.

[25] J. Yang, D. Bai, S. Bai, Y. Li, and S. Wang. *Design of Mechanical Structure and Tracking Control System for Lower Limbs Rehabilitative Training Robot*. In International Conference on Mechatronics and Automation, pages 1824–1829, August 2009.

[26] Animazoo. *Gypsy 7*. http://www.animazoo.com/motion-capture-systems/gypsy-7-motion-capture-system/.

[27] A.G. Kirk, J.F. O'Brien, and D.A. Forsyth. *Skeletal Parameter Estimation from Optical Motion Capture Data*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 2, page 1185, June 2005.

[28] N. Miyata, M. Kouchi, T. Kurihara, and M. Mochimaru. *Modeling of Human Hand Link Structure from Optical Motion Capture Data*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 3, pages 2129–2135, September 2004.

[29] L. Herda, P. Fua, R. Plankers, R. Boulic, and D. Thalmann. *Skeleton-based Motion Capture for Robust Reconstruction of Human Motion*. In Proceedings of the Conference on Computer Animation, pages 77–83, 2000.

[30] Microsoft. *Kinect for Xbox 360*. http://www.xbox.com/en-GB/kinect/.

[31] F.H. Raab, E.B. Blood, T.O. Steiner, and H.R. Jones. *Magnetic Position and Orientation Tracking System*. IEEE Transactions on Aerospace and Electronic Systems, AES-15(5):709–718, September 1979.

[32] J.S. Day, D.J. Murdoch, and G.A. Dumas. *Calibration of Position and Angular Data from a Magnetic Tracking Device*. Journal of Biomechanics, 33(8):1039–1045, 2000.

[33] M. Schneider and C. Stevens. *Development and Testing of a New Magnetic Tracking Device for Image Guidance*. In SPIE Medical Imaging 2007: Visualization and Image-Guided Procedures, pages 6509–6517, February 2007.

[34] Y. Liu, Y. Wang, D. Yan, and Y. Zhou. *DPSD Algorithm for AC Magnetic Tracking System*. In IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems, pages 101–106, July 2004.

[35] G. Zachmann. *Distortion Correction of Magnetic Fields for Position Tracking*. In Proceedings of the International Conference on Computer Graphics, pages 213–220, 251, June 1997.

[36] E. Foxlin, M. Harrington, and G. Pfeifer. *Constellation: a Wide-Range Wireless Motion-Tracking System for Augmented Reality and Virtual Set Applications*. In Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, pages 371–378, New York, NY, USA, 1998. ACM.

[37] J.H. Kim, J.S. Hur, H.R. Lee, D.S. Kim, I.S. No, and K.M. Kim. *Performance Enhancement of Target Tracking for an Underwater Vehicle Using Synthetic Sensor Technique*. In MTS/IEEE Conference and Exhibition OCEANS, volume 3, pages 1693–1696, 2001.

[38] Q. Wang, W.-P. Chen, R. Zheng, K. Lee, and L. Sha. *Acoustic Target Tracking Using Tiny Wireless Sensor Devices*. In Proceedings of the 2nd International Conference on Information Processing in Sensor Networks, pages 642–657, Berlin, Heidelberg, 2003. Springer-Verlag.

[39] F. Packi, F. Beutler, and U.D. Hanebeck. *Wireless Acoustic Tracking for Extended Range Telepresence*. In International Conference on Indoor Positioning and Indoor Navigation, pages 1–9, September 2010.

[40] R. Zhu and Z. Zhou. *A Real-Time Articulated Human Motion Tracking using Tri-Axis Inertial/Magnetic Sensors Package*. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 12(2):295–302, June 2004.

[41] H. Chung, L. Ojeda, and J. Borenstein. *Accurate Mobile Robot Dead-Reckoning with a Precision-Calibrated Fiber-Optic Gyroscope*. IEEE Transactions on Robotics and Automation, 17(1):80–84, February 2001.

[42] X. Yun, E.R. Bachmann, H. Moore, and J. Calusdian. *Self-Contained Position Tracking of Human Movement Using Small Inertial/Magnetic Sensor Modules*. In Transactions of the IEEE International Conference on Robotics and Automation, pages 2526–2533, April 2007.

[43] D. Roetenberg, H.J. Luinge, C.T.M. Baten, and P.H. Veltink. *Compensation of Magnetic Disturbances Improves Inertial and Magnetic Sensing of Human Body Segment Orientation*. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 13(3):395–405, September 2005.

[44] C. Goodvin, E. Park, K. Huang, and K. Sakaki. *Development of a Real-Time Three-Dimensional Spinal Motion Measurement System for Clinical Practice*. Medical and Biological Engineering and Computing, 44(12):1061–1075, 2006.

[45] G. Ebersbach, M. Heijmenberg, L. Kindermann, T. Trottenberg, J. Wissel, and W. Poewe. *Interference of Rhythmic Constraint on Gait in Healthy*

*Subjects and Patients with Early Parkinson's Disease: Evidence for Impaired Locomotor Pattern Generation in Early Parkinson's Disease*. Movement Disorders, 14(4):619–625, 1999.

[46] P.D. Hung, S. Bonnet, R. Guillemaud, E. Castelli, and P.T.N. Yen. *Estimation of Respiratory Waveform Using an Accelerometer*. In 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, pages 1493–1496, May 2008.

[47] P. Mistiaen. *Design of a Breathing Sensor using Accelerometers*. Master's thesis, Ghent University, Ghent, Belgium, 2009.

[48] T.M.E. Nijsen, J.B.A.M. Arendsa, P.A.M. Griepa, and P.J.M. Cluitmansa. *The Potential Value of Three-Dimensional Accelerometry for Detection of Motor Seizures in Severe Epilepsy*. Epilepsy and Behavior, 7:74–Ű84, 2005.

[49] H. Luo, G.K. Fedder, and L.R. Carley. *A 1 mG Lateral CMOS-MEMS Accelerometer*. In The Thirteenth Annual International Conference on Micro Electro Mechanical Systems, pages 502–507, January 2000.

[50] E.H. Hall. *On a New Action of the Magnet on Electric Currents*. American Journal of Mathematics, 2:287–292, 1879.

[51] J. Kyynäräinen, J. Saarilahti, H. Kattelus, A. Kärkkäinen, T. Meinander, A. Oja, P. Pekko, H. Seppä, M. Suhonen, H. Kuisma, S. Ruotsalainen, and M. Tilli. *A 3D Micromechanical Compass*. Sensors and Actuators A: Physical, 142(2):561–568, 2008.

[52] M.S. Weinberg and A. Kourepenis. *Error Sources in In-Plane Silicon Tuning-Fork MEMS Gyroscopes*. Journal of Microelectromechanical Systems, 15(3):479–491, June 2006.

[53] S. Tilak, N.B. Abu-Ghazaleh, and W. Heinzelman. *A Taxonomy of Wireless Microsensor Network Models*. Mobile Computing and Communications Review, 3(2):28–36, 2002.

[54] C. Alippi, R. Camplani, C. Galperti, and M. Roveri. *A Robust, Adaptive, Solar-Powered WSN Framework for Aquatic Environmental Monitoring*. IEEE Sensors Journal, 11(1):45–55, January 2011.

[55] A. Manjeshwar and D.P. Agrawaly. *APTEEN: A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks*. In Proceedings of the International Parallel and Distributed Processing Symposium, volume 2, page 0195b, Fort Lauderdale, Florida, April 2002.

[56] W. Ye, J. Heidemann, and D. Estrin. *An Energy-Efficient MAC Protocol for Wireless Sensor Networks*. In Proceedings of the 21st IEEE International Conference on Computer Communications, volume 3, pages 1567–1576, New York, NY, USA, June 2002.

[57] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler. *TinyNode: a Comprehensive Platform for Wireless Sensor Network Applications*. In Proceedings of the 5th International Conference on Information Processing in Sensor Networks, pages 358–365, 2006.

[58] J. Polastre, R. Szewczyk, and D. Culler. *Telos: Enabling Ultra-Low Power Wireless Research*. In Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, pages 364Ű–369, 2005.

[59] R.B. Smith. *SPOTWorld and the Sun SPOT*. In Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pages 565–566, 2007.

[60] XSens Technologies. *MTw 3DOF Orientation Tracker*. http://www.xsens.com/images/stories/products/PDF_Brochures/mtwleaflet.pdf.

[61] Intersense Inc. *Intersense Wireless InertiaCube3*. http://www.intersense.com/uploads/documents/Wireless_InertiaCube3_Datasheet.pdf.

[62] Inertia Technology. *ProMove-2*. http://inertia-technology.com/wp-content/uploads/2010/12/promove2_datasheet_preliminary.pdf.

[63] A. D. Young, M. J. Ling, and D. K. Arvind. *Orient-2: a Realtime Wireless Posture Tracking System using Local Orientation Estimation*. In Proceedings of the 4th Workshop on Embedded Networked Sensors, pages 53–57, Cork, Ireland, 2007.

[64] A. Lynch, B. Majeed, J. Barton, F. Murphy, K. Delaney, and S. OŠMathuna. *A Wireless Inertial Measurement System (WIMS) for an Interactive Dance Environment*. Journal of Physics: Conference Series, 15:95Ű–100, 2005.

[65] X. Yun and E.R. Bachmann. *Design, Implementation, and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking*. IEEE Transactions on Robotics, 22(6):1216–1227, 2006.

[66] E.R. Bachman. *Inertial and Magnetic Tracking of Limb Segment Orientation for Inserting Humans into Synthetic Environments*. PhD thesis, Naval Postgraduate School, Monterey, California, USA, 2000.

[67] A.D. Young, M.J. Ling, and D.K. Arvind. *Distributed Estimation of Linear Acceleration for Improved Accuracy in Wireless Inertial Motion Capture*. In Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks, pages 256–267, April 2010.

[68] T. Harada, T. Mori, and T. Sato. *Development of a Tiny Orientation Estimation Device to Operate under Motion and Magnetic Disturbance*. The International Journal of Robotics Research, 26(6):547–559, 2007.

[69] H.J. Luinge, P.H. Veltink, and C.T.M. Baten. *Ambulatory Measurement of Arm Orientation*. Journal of Biomechanics, 40(1):78–85, 2007.

[70] X. Yun, E.R. Bachmann, and R.B. McGhee. *A Simplified Quaternion-Based Algorithm for Orientation Estimation From Earth Gravity and Magnetic Field Measurements*. IEEE Transactions on Instrumentation and Measurement, 57(3):638–650, March 2008.

[71] XSens Technologies. *MVN Inertial Motion Capture*. http://www.xsens.com/images/stories/products/PDF_Brochures/mvnleaflet.pdf.

[72] Animazoo. *IGS-190 Mobile Motion Capture System*. http://www.animazoo.com/motion-capture-systems/.

[73] E.P. Hanavan. *A Mathematical Model of the Human Body*. Technical Report TR-64-102, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, OH, USA, 1964.

# 2 Filter Design

This chapter describes the design of an orientation estimating filter. How sensor output data is transformed from raw measurements to useful information and which additional mechanisms are needed to obtain reliable estimates.

## 2.1 Introduction

Before the actual filter design can be described, some important principles must be outlined. First, mathematical representations of rigid body orientation are introduced. These formalisms form the basis of the process model formulas used in the orientation estimating filter. Second, the basics of the used filter architecture is explained and some derived versions developed for non-linear systems are described.

When the basic ideas have been introduced, the principles can be applied to the problem at hand. All equations necessary for estimation are derived and the entire procedure is outlined. This procedure consists of different steps of preprocessing of the sensor values, estimation using the Kalman filter framework and adjusting the parameters to ensure robustness and accuracy.

In order for the estimator to perform adequately, several parameters will need to be estimated. Real life captures of movements help to model the process and tune the parameters. Later, simulations are executed to test the performance of the filter and to determine the influence certain parameters might have on the filter characteristics.

## 2.2 Orientation of a Rigid Body

The concept of rigid body orientation is defined as the relation between a moving coordinate system attached to the body and a fixed, earth-bound reference frame. Conventionally, a fixed reference frame is chosen with positive axes pointing to the local east, north and up direction, forming a right-handed base. The moving coordinate system consists of three orthogonal axes pointing along distinct geometric features of the rigid body at hand. Considering for example an airplane, the axes could be defined along the length axis of the plane, the left wing and straight up.

Various methods are available to describe orientation mathematically [1]. Each method has advantages and disadvantages which should be taken into account [2]. Euler angles and quaternions are most common and will be discussed further in this section.

### 2.2.1 Euler Angles

Euler angles describe the orientation of a rigid body as three subsequent rotations of the reference frame around well defined axes. Depending on the choice of these axes, many different conventions exist, where the axes can be chosen in the reference or the moving coordinate system. Intrinsic rotations are executed around moving axes of the object-bound coordinate system, while extrinsic rotations are performed around fixed, earth-bound reference axes [3].

#### 2.2.1.1 Convention

The convention used in this work is known as *Tait-Bryan* angles or, more specific, the *Roll-Pitch-Yaw* convention. This convention uses three subsequent intrinsic rotations around three different axes. The order used is Z-Y-X, where the Z-rotation is designated as *Yaw*, Y-rotation corresponds to *Pitch* and X-rotation equals *Roll* [4]. Figure 2.1 visualises the subsequent rotations of the convention applied to an airplane.

Commonly, the Greek symbols $\phi$, $\theta$ and $\psi$ denote the roll, pitch and yaw angles respectively. Positive values of the angles refer to a counter-clockwise rotation around the corresponding axis. This convention is known as the right-hand rule. Note the negative sign of the $\theta$ angle in Figure 2.1 as here clockwise rotation around the Y-axis is indicated.

In order to describe a certain orientation with a unique set of angles, limited ranges apply. Both $\phi$ and $\psi$ are defined modulo $2\pi$ radians or $360°$, while $\theta$ is only allowed to cover $\pi$ radians or $180°$. The most commonly used ranges are $[0,\ 360°[$ for roll and yaw angles and $[-90°,\ 90°]$ for the pitch angle.

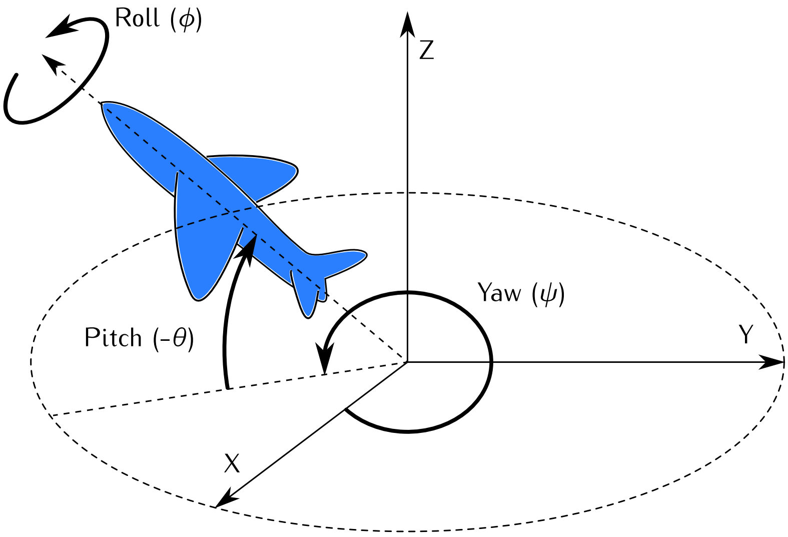


*Figure 2.1: Roll, pitch and yaw convention applied to an airplane.*

#### 2.2.1.2 Rotation Matrices

An easy way to apply a certain rotation to a vector consists of multiplying the original vector coordinates in the fixed reference system with a rotation matrix, which results in new vector coordinates in the rotated reference system:

$$\boldsymbol{v}' = \mathrm{R}\,\boldsymbol{v} \tag{2.1}$$

The rotation matrix associated with a roll operation around the X-axis over an angle $\phi$ is given by:

$$\mathrm{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.2}$$

It is easily understood that the resulting vector will retain the same X-coordinate as the original vector as this is the axis around which the rotation takes place. The Y- and Z-coordinate however, change in value according to the sine and cosine of the angle.

Similarly, the rotation matrices for pitch and yaw rotations over respectively $\theta$ and $\psi$ are given by:

$$\mathrm{R}_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.3}$$

$$\mathrm{R}_z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Euler angles compose three individual rotations in a certain order to obtain the final orientation. Likewise, all three rotation matrices may be multiplied to obtain the combined rotation in a single matrix. Note that the matrix positioned at the right will effect the vector first. This implies that the matrices must be left multiplied according to the Euler angle order.

$$\mathrm{R} = \mathrm{R}_x\,\mathrm{R}_y\,\mathrm{R}_z =$$
$$\begin{bmatrix} \cos\theta\cos\psi & -\cos\theta\sin\psi & \sin\theta \\ \cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi - \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\theta \\ \sin\phi\sin\psi - \cos\phi\sin\theta\cos\psi & \sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \cos\phi\cos\theta \end{bmatrix} \tag{2.5}$$

#### 2.2.1.3 Relation to Body Rates

Due to the fact that the rotation axes used to obtain the orientation of a rigid body do not correspond to the axes of the reference system bound to the object, body rates do not correspond with Euler rates. Consequently, body rates may not be integrated over time to obtain Euler angles. The angular rate in the earth bound fixed reference system $\boldsymbol{\omega}_{ref}$ can be expressed in terms of the Euler rates ($\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$) and the corresponding rotation matrices [5]:

$$\boldsymbol{\omega}_{ref} = \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + \mathrm{R}_z^{-1} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathrm{R}_z^{-1}\,\mathrm{R}_y^{-1} \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{2.6}$$

The Z-axis of the reference system corresponds to the $\psi$-axis, so it can be added up directly. The other two rates must first be rotated in order to align both axes. The body rates $\boldsymbol{\omega}_{body}$ are a result of the full rotation of the angular rates in the reference system:

$$\boldsymbol{\omega}_{body} = \mathrm{R}\,\boldsymbol{\omega}_{ref} = \mathrm{R}_x\,\mathrm{R}_y\,\mathrm{R}_z\,\boldsymbol{\omega}_{ref} \tag{2.7}$$

Substituting (2.6) in (2.7) yields:

$$\boldsymbol{\omega}_{body} = \mathrm{R}_x\,\mathrm{R}_y\,\mathrm{R}_z\left(\begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} + \mathrm{R}_z^{-1}\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + \mathrm{R}_z^{-1}\,\mathrm{R}_y^{-1}\begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix}\right) \tag{2.8}$$

$$= \mathrm{R}_x\,\mathrm{R}_y\,\mathrm{R}_z\begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} + \mathrm{R}_x\,\mathrm{R}_y\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + \mathrm{R}_x\begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} \tag{2.9}$$

Using (2.2), (2.3) and (2.4), the above equation can be reduced to:

$$\boldsymbol{\omega}_{body} = \begin{bmatrix}p\\q\\r\end{bmatrix} = \begin{bmatrix}\dot{\phi} + \dot{\psi}\sin\theta\\ \dot{\theta}\cos\phi - \dot{\psi}\sin\phi\cos\theta\\ \dot{\theta}\sin\phi + \dot{\psi}\cos\phi\cos\theta\end{bmatrix} \tag{2.10}$$

The inverse of (2.10) can easily be deduced [6]:

$$\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix} = \begin{bmatrix}p - q\sin\phi\tan\theta + r\cos\phi\tan\theta\\ q\cos\phi + r\sin\phi\\ -q\sin\phi\sec\theta + r\cos\phi\sec\theta\end{bmatrix} \tag{2.11}$$

#### 2.2.1.4 Performance

Rotating a single vector requires this vector to be subsequently multiplied by all three rotation matrices. Using the fact that four of the elements in the matrices are zero and one element equals unity, this requires four multiplications and two additions per matrix multiplication, resulting in a total of 12 multiplications and 6 additions. It is important to note however that before these operations can be executed, sines and cosines of all three angles must be determined.

An important drawback of Euler angles however, is a singularity called the *gimbal lock*. When the pitch angle approaches its extreme values (±90°), the resulting roll axis and original yaw axis coincide. This leads to the fact that each orientation with an extreme pitch can be represented by an infinite number of Euler angle sets. One degree of freedom is essentially lost in the gimbal lock. For example, the simple upright position with zero roll and yaw and 90° pitch can be obtained by any set of angles where the roll and yaw angles are each other's negative.

### 2.2.2 Quaternion

Quaternions are a mathematical extension of complex numbers where the imaginary part consists of a vector with three components rather than a single real number. The quaternion set thus forms a four-dimensional vector space with elements in $\mathcal{R}^4$. One number forms the real or scalar part of the quaternion, while the other three form the imaginary or vector part [7].

#### 2.2.2.1 Notation

Generally, a quaternion can be written as a linear combination of the four base vectors of the quaternion space:

$$\mathbf{q} \;=\; w \;+\; x\,\mathrm{i} \;+\; y\,\mathrm{j} \;+\; z\,\mathrm{k}, \tag{2.12}$$

where i, j and k denote the imaginary base vectors. Other common notations write quaternions as a scalar/vector pair or as a four dimensional vector:

$$\mathbf{q} \;=\; [w,\; \mathbf{v}] \;=\; [w,\; x,\; y,\; z] \tag{2.13}$$

#### 2.2.2.2 Operations

Several operations are defined within the quaternion space. The addition of two quaternions simply results in a quaternion whose elements equal the sum of the base elements:

$$\mathbf{q}_1 \;+\; \mathbf{q}_2 \;=\; (w_1+w_2) \;+\; (x_1+x_2)\,\mathrm{i} \;+\; (y_1+y_2)\,\mathrm{j} \;+\; (z_1+z_2)\,\mathrm{k}. \tag{2.14}$$

A quaternion may also be multiplied with a scalar number leading to each of the elements being multiplied by this scalar:

$$a\,\mathbf{q} \;=\; a\,w \;+\; a\,x\,\mathrm{i} \;+\; a\,y\,\mathrm{j} \;+\; a\,z\,\mathrm{k}. \tag{2.15}$$

This also leads to the definition of the negative or additive inverse of a quaternion:

$$-\,\mathbf{q} \;=\; -\,w \;-\; x\,\mathrm{i} \;-\; y\,\mathrm{j} \;-\; z\,\mathrm{k}. \tag{2.16}$$

By defining the products of the base elements, quaternion multiplication can be determined using distributive law. In 1843, Sir William Rowan Hamilton defined quaternion multiplication [8] by carving the following equations into the stone of the Brougham Bridge in Dublin:

$$\mathrm{i}^2 \;=\; \mathrm{j}^2 \;=\; \mathrm{k}^2 \;=\; \mathrm{i}\,\mathrm{j}\,\mathrm{k} \;=\; -1 \tag{2.17}$$

From these equations, the following relations can further be derived:

$$\begin{aligned} \mathrm{i}\,\mathrm{j} &= \mathrm{k} & \mathrm{j}\,\mathrm{k} &= \mathrm{i} & \mathrm{k}\,\mathrm{i} &= \mathrm{j} \\ \mathrm{j}\,\mathrm{i} &= -\mathrm{k} & \mathrm{k}\,\mathrm{j} &= -\mathrm{i} & \mathrm{i}\,\mathrm{k} &= -\mathrm{j} \end{aligned}$$

The above equations clearly indicate that quaternion multiplication is not commutative. Three dimensional rotations also exhibit this property, hinting to a possible relation between both.

The multiplication of two quaternions can now be determined by obeying the above equations:

$$\begin{aligned} \mathbf{q}_1 \otimes \mathbf{q}_2 = & \, w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \\ & + (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2)\, \mathrm{i} \\ & + (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2)\, \mathrm{j} \\ & + (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2)\, \mathrm{k} \end{aligned} \tag{2.18}$$

The above product is also referred to as the Hamilton product and will further be denoted by the $\otimes$-sign. Using the scalar/vector notation, the Hamilton product can also be expressed as:

$$\mathbf{q}_1 \otimes \mathbf{q}_2 = [w_1 w_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2,\ w_1 \boldsymbol{v}_2 + w_2 \boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2], \tag{2.19}$$

where $\cdot$ and $\times$ respectively denote the inner and vector product of three dimensional vectors:

$$\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 = x_1 x_2 + y_1 y_2 + z_1 z_2 \tag{2.20}$$

$$\boldsymbol{v}_1 \times \boldsymbol{v}_2 = [y_1 z_2 - z_1 y_2,\ z_1 x_2 - x_1 z_2,\ x_1 y_2 - y_1 x_2] \tag{2.21}$$

Conjugation of a quaternion simply requires the inversion of the complex part, as is the case with complex numbers:

$$\mathbf{q}^* = w - x\,\mathrm{i} - y\,\mathrm{j} - z\,\mathrm{k}. \tag{2.22}$$

The norm of a quaternion is defined using the above definition of the conjugate and results to be the square root of the summation of the square of each of the quaternion elements:

$$|\mathbf{q}| = \sqrt{\mathbf{q} \otimes \mathbf{q}^*} = \sqrt{w^2 + x^2 + y^2 + z^2}. \tag{2.23}$$

The multiplicative inverse can easily be derived from the above equations:

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{|\mathbf{q}|^2}. \tag{2.24}$$

#### 2.2.2.3 Quaternions and Three Dimensional Rotation

Each orientation of a rigid body can be described by a rotation over a certain angle $\alpha$ around a certain axis with unit direction vector $\boldsymbol{u}$. The quaternion associated to this rotation is defined as follows [9]:

$$\mathbf{q} \;=\; \cos\frac{\alpha}{2} \;+\; \boldsymbol{u}\sin\frac{\alpha}{2}. \tag{2.25}$$

The norm of the above quaternion clearly equals unity. Remark that reversing the axis direction and negating the angle results in the exact same quaternion:

$$\mathbf{q} \;=\; \cos\left(-\frac{\alpha}{2}\right) \;-\; \boldsymbol{u}\sin\left(-\frac{\alpha}{2}\right) \;=\; \cos\frac{\alpha}{2} \;+\; \boldsymbol{u}\sin\frac{\alpha}{2}, \tag{2.26}$$

which is expected as both resulting orientations are also equal. Furthermore, also note that the negative of $\mathbf{q}$ also corresponds to the same orientation:

$$\begin{aligned} -\mathbf{q} &= -\cos\frac{\alpha}{2} \;-\; \boldsymbol{u}\sin\frac{\alpha}{2} \\ &= \cos\left(180^\circ \;-\; \frac{\alpha}{2}\right) \;-\; \boldsymbol{u}\sin\left(180^\circ \;-\; \frac{\alpha}{2}\right) \\ &= \cos\left(\frac{360^\circ \;-\; \alpha}{2}\right) \;-\; \boldsymbol{u}\sin\left(\frac{360^\circ \;-\; \alpha}{2}\right) \\ &= \cos\left(\frac{-\,\alpha}{2}\right) \;-\; \boldsymbol{u}\sin\left(\frac{-\,\alpha}{2}\right) \\ &= \cos\frac{\alpha}{2} \;+\; \boldsymbol{u}\sin\frac{\alpha}{2}, \end{aligned} \tag{2.27}$$

using the fact that $\alpha$ is defined modulo 360°. This implies that every orientation can in fact be represented by either of the two quaternions.

In order to use the defined quaternion to calculate rotations of a vector $\boldsymbol{r}$, one must first define the quaternion $\mathbf{q}_r$, which consists of the vector components as the imaginary part:

$$\mathbf{q}_r \;=\; x_r\,\mathrm{i} \;+\; y_r\,\mathrm{j} \;+\; z_r\,\mathrm{k}. \tag{2.28}$$

The rotated version of $\boldsymbol{r}$, $\boldsymbol{r'}$, is then found as the complex part of the quaternion resulting from the following product:

$$\mathbf{q}_r' \;=\; \mathbf{q} \;\otimes\; \mathbf{q}_r \;\otimes\; \mathbf{q}^{-1}. \tag{2.29}$$

The real part of the resulting quaternion $\mathbf{q}_r'$ will always equal zero, resulting in a pure vector quaternion result. Note that, as $\mathbf{q}$ represents a unit quaternion, the inverse may also be replaced by the conjugate as follows from

(2.24). This can easily be understood as any rotation can be inverted by either negating the rotation angle or reversing the axis direction. Both of these operations will negate the vector part of the quaternion.

Concatenation of rotations can be executed by multiplying the respective quaternions before applying (2.29) with the resulting quaternion. Let $\mathbf{q}_1$ and $\mathbf{q}_2$ correspond to subsequent rotations:

$$\begin{aligned}(\mathbf{q}_1 \otimes \mathbf{q}_2) \otimes \mathbf{q}_r \otimes (\mathbf{q}_1 \otimes \mathbf{q}_2)^{-1} &= \mathbf{q}_1 \otimes \mathbf{q}_2 \otimes \mathbf{q}_r \otimes \mathbf{q}_2^{-1} \otimes \mathbf{q}_1^{-1} \\ &= \mathbf{q}_1 \otimes \left(\mathbf{q}_2 \otimes \mathbf{q}_r \otimes \mathbf{q}_2^{-1}\right) \otimes \mathbf{q}_1^{-1},\end{aligned}$$

which shows that a rotation with the product quaternion equals a rotation with $\mathbf{q}_2$ followed by one with $\mathbf{q}_1$. As is the case with rotation matrices, quaternions must be left multiplied to be concatenated.

#### 2.2.2.4 Relation to body rates

Assuming small changes in angular position, an equation can be derived linking quaternion rate to body rates [10]:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \mathbf{q}_{\boldsymbol{\omega}_{ref}} = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix}0 & p & q & r\end{bmatrix} \tag{2.30}$$

The above equation allows to relate angular body rates to quaternion rates without using any trigonometric functions.

#### 2.2.2.5 Performance

Using quaternions as a representation for orientation requires keeping track of four doubles compared to three with Euler angles. Also, evaluation of (2.29) requires 24 multiplications and 18 additions when keeping in mind that some of the components are zero. This is twice the amount of multiplications and three times the amount of additions required to calculate a rotation using Euler angles.

However, quaternions do not require any trigonometric calculations as Euler angles do and, more importantly, do not suffer from any singularities. Due to the fact that an orientation is described by a single rotation over a certain angle and around a certain axis, no gimbal lock occurs.

### 2.2.3 Conversion

Orientations expressed in either of the forms presented can be converted in the other representation. Both transformation formulas are derived here.

#### 2.2.3.1 Euler Angles to Quaternion

The conversion from Euler angles to quaternion is easily derived. Each of the three individual rotations can be represented by a single quaternion and multiplied in the same order as the rotation matrices in (2.5). The yaw rotation is applied first and consists of a rotation around the Z-axis over $\psi$. Applying (2.25) quickly results in:

$$\mathbf{q}_\psi = \begin{bmatrix} \cos\frac{\psi}{2} & 0 & 0 & \sin\frac{\psi}{2} \end{bmatrix} \tag{2.31}$$

Similar expressions are found for the pitch and roll rotations:

$$\mathbf{q}_\theta = \begin{bmatrix} \cos\frac{\theta}{2} & 0 & \sin\frac{\theta}{2} & 0 \end{bmatrix} \tag{2.32}$$

$$\mathbf{q}_\phi = \begin{bmatrix} \cos\frac{\phi}{2} & \sin\frac{\phi}{2} & 0 & 0 \end{bmatrix} \tag{2.33}$$

The quaternion describing the entire orientation can then be found by multiplying all of the quaternions in the correct order:

$$\mathbf{q} = \mathbf{q}_\phi \otimes \mathbf{q}_\theta \otimes \mathbf{q}_\psi \tag{2.34}$$

Or in expanded form:

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} - \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{bmatrix} \tag{2.35}$$

#### 2.2.3.2 Quaternion to Euler Angles

The conversion from quaternion form to Euler angles is derived via rotation matrices. Expanding (2.29) and rewriting the result in matrix notation gives:

$$\mathrm{R}_\mathbf{q} = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2\,(x\,y - w\,z) & 2\,(w\,y + x\,z) \\ 2\,(x\,y + w\,z) & w^2 - x^2 + y^2 - z^2 & 2\,(y\,z - w\,x) \\ 2\,(w\,y - x\,z) & 2\,(y\,z + w\,x) & w^2 - x^2 - y^2 + z^2 \end{bmatrix} \tag{2.36}$$

Given the fact that the quaternion must be normalised, this can be rewritten as:

$$\mathrm{R_q} = \begin{bmatrix} 1 - 2\left(y^2 + z^2\right) & 2\left(x\,y - w\,z\right) & 2\left(w\,y + x\,z\right) \\ 2\left(x\,y + w\,z\right) & 1 - 2\left(x^2 + z^2\right) & 2\left(y\,z - w\,x\right) \\ 2\left(w\,y - x\,z\right) & 2\left(y\,z + w\,x\right) & 1 - 2\left(x^2 + y^2\right) \end{bmatrix} \tag{2.37}$$

Comparison of (2.37) with (2.5) finally results in the following conversion formulas:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan\left(\dfrac{2\left(w\,x - y\,z\right)}{1 - 2\left(x^2 + y^2\right)}\right) \\ \arcsin\left(2\,\left(w\,y + x\,z\right)\right) \\ \arctan\left(\dfrac{2\left(w\,z - x\,y\right)}{1 - 2\left(y^2 + z^2\right)}\right) \end{bmatrix} \tag{2.38}$$

## 2.3 Kalman Filter

The Kalman Filter (KF) is a recursive data processing algorithm that estimates the state of a dynamic system from a series of incomplete and noisy measurements [11]. It is a predictor-corrector type filter where the final estimate of a single step is a weighted average of the predicted and the measured value. The weights are derived from the covariance, which is a measure for the uncertainty of the present values. The filter also allows multiple measurements from the same state to be fused into a single estimate, as long as the statistics of each of these measurements are known. Basically, the filter is a Maximum Likelyhood (ML) estimator that minimizes the error covariance under the assumption that some conditions are met [12].

### 2.3.1 System Model

The KF is based on a linear dynamic system model that describes the problem at hand using vectors and matrices. The system state is represented by a vector and it is assumed that this state follows a certain process model equation where the current state is a linear combination of the state in the previous step, the current input and a random noise term. Denoting the current time step by $k$, the general form of the process model equation is:

$$\boldsymbol{x}_k = \mathrm{A}_k\,\boldsymbol{x}_{k-1} + \mathrm{B}_k\,\boldsymbol{u}_k + \boldsymbol{w}_k, \tag{2.39}$$

where $\boldsymbol{x} \in \mathcal{R}^n$ represents the state vector, $\mathrm{A} \in \mathcal{R}^{n \times n}$ is the state transition matrix describing the change from the previous state, $\boldsymbol{u} \in \mathcal{R}^l$ is the input vector term, $\mathrm{B} \in \mathcal{R}^{n \times l}$ is the input transition matrix modeling the relation between the state and the input, and $\boldsymbol{w} \in \mathcal{R}^{n}$ corresponds to the process noise. It is assumed that the noise term is time invariant, Gaussian white noise with zero mean and covariance matrix $\mathrm{Q} \in \mathcal{R}^{n \times n}$:

$$\mathcal{P}(\boldsymbol{w}_k) \ \sim \ \mathcal{N}(\mathbf{0},\ \mathrm{Q}_k) \tag{2.40}$$

Aside from the process model, a measurement model is required which describes the relation between the state and the measurements. The general form of the measurement model equation is given by:

$$\boldsymbol{z}_k \ = \ \mathrm{H}_k \, \boldsymbol{x}_k \ + \ \boldsymbol{v}_k, \tag{2.41}$$

where $\boldsymbol{z} \in \mathcal{R}^{m}$ is the measurement vector, $\mathrm{H} \in \mathcal{R}^{m \times n}$ is the measurement transition matrix describing the relation between the state and the measurements and $\boldsymbol{v} \in \mathcal{R}^{m}$ represents the measurement noise. As with the process noise, it is assumed that the noise term is time invariant, Gaussian white noise with zero mean and covariance matrix $\mathrm{R} \in \mathcal{R}^{m \times m}$:

$$\mathcal{P}(\boldsymbol{v}_k) \ \sim \ \mathcal{N}(\mathbf{0},\ \mathrm{R}_k) \tag{2.42}$$

Furthermore, process and measurement noise terms are assumed to be independent. Note that in the above equations, all of the variables are allowed to change over time (hence the $k$ subscript), which is however not necessarily the case.

### 2.3.2 Filter Algorithm

As already mentioned before, the KF consists of two major steps: prediction and correction. In the prediction step, a first estimate of the current state is predicted based on the process model equation and thus only using the previous state and the current input. This predicted estimate is mostly referred to as the *a priori* estimate and is denoted by $\hat{\boldsymbol{x}}^{-}$. The correction step combines the estimate from the prediction step with the measurement result to give the final estimate for the current time step. This final estimate is known as the *a posteriori* estimate and is denoted by $\hat{\boldsymbol{x}}$.

Aside from providing an estimate of the state, both steps also supply the estimate error covariance. Defining the a priori and a posteriori error as

$$\boldsymbol{e}_k^{-} \ = \ \boldsymbol{x}_k \ - \ \hat{\boldsymbol{x}}_k^{-} \tag{2.43}$$

$$\boldsymbol{e}_k \ = \ \boldsymbol{x}_k \ - \ \hat{\boldsymbol{x}}_k, \tag{2.44}$$

allows to define both covariances as well:

$$\begin{aligned} \mathrm{P}_k^- &= \mathrm{E}\left[\boldsymbol{e}_k^- \, \boldsymbol{e}_k^{-T}\right] && (2.45) \\ \mathrm{P}_k &= \mathrm{E}\left[\boldsymbol{e}_k \, \boldsymbol{e}_k^T\right]. && (2.46) \end{aligned}$$

In each cycle, the KF will thus propagate the first two moments of the system state distribution. The a posteriori estimate reflects the mean of this distribution, while the a posteriori estimate error covariance reflects the variance. If both (2.40) and (2.42) are satisfied, the state is actually normally distributed:

$$\mathcal{P}(\boldsymbol{x}_k) \sim \mathcal{N}(\hat{\boldsymbol{x}}_k, \, \mathrm{P}_k) \tag{2.47}$$

### 2.3.2.1 Prediction

The state prediction equation is easily obtained by assuming zero noise and substituting the previous a posteriori estimate in (2.39). The a priori estimate covariance incorporates the uncertainty of the a posteriori estimate and the noise by adding up both covariances appropriately.

$$\begin{aligned} \hat{\boldsymbol{x}}_k^- &= \mathrm{A}_k \, \hat{\boldsymbol{x}}_{k-1} + \mathrm{B}_k \, \boldsymbol{u}_k && (2.48) \\ \mathrm{P}_k^- &= \mathrm{A}_k \, \mathrm{P}_{k-1} \, \mathrm{A}_k^T + \mathrm{Q}_k && (2.49) \end{aligned}$$

### 2.3.2.2 Correction

The correction step starts by calculating the Kalman gain $\mathrm{K}_k$, which is chosen such that the a posteriori estimate error covariance (2.46) is minimized. The a posteriori estimate then results from a weighted sum of the a priori estimate and the current innovation, which is defined as the mismatch between the predicted ($\mathrm{H}\,\hat{\boldsymbol{x}}^-$) and the actual ($\boldsymbol{z}$) measurement. Finally the a posteriori estimate error covariance can be computed.

$$\begin{aligned} \mathrm{K}_k &= \mathrm{P}_k^- \, \mathrm{H}_k^T \left(\mathrm{H}_k \, \mathrm{P}_k^- \, \mathrm{H}_k^T + R_k\right)^{-1} && (2.50) \\ \hat{\boldsymbol{x}}_k &= \hat{\boldsymbol{x}}_k^- + \mathrm{K}_k \left(\boldsymbol{z}_k - \mathrm{H}_k \, \hat{\boldsymbol{x}}_k^-\right) && (2.51) \\ \mathrm{P}_k &= \left(\mathrm{I}_{n \times n} - \mathrm{K}_k \, \mathrm{H}_k\right) \mathrm{P}_k^- && (2.52) \end{aligned}$$

In the above formula $\mathrm{I}_{n \times n}$ represents the identity matrix of size $n$.

The Kalman gain determines which component weighs more in the final estimate according to their level of uncertainty. If the measurement covariance matrix R is much smaller than the a priori estimate error covariance $\mathrm{P}^-$, the actual measurement will be trusted more than the predicted estimate, while the roles are reversed in case R turns out to be much bigger than $P^-$. This can easily be understood when looking at some extreme cases:

$$\lim_{R \to 0} K = H^{-1} \tag{2.53}$$

$$\lim_{P^- \to 0} K = 0 \tag{2.54}$$

#### 2.3.2.3 Recursion

The filter algorithm starts by choosing an appropriate initial value of the state vector and the estimate error covariance. These values should accurately reflect the distribution of the state in order to obtain a well functioning filter. Afterwards, the prediction and correction steps are recursively applied, as depicted in figure 2.2. Strictly speaking however, both steps are not required to alternate. A predicted value may be corrected multiple times before a new prediction is calculated. Multiple predictions are also allowed. This is e.g. the case when a new measurement is not available at the appropriate time.

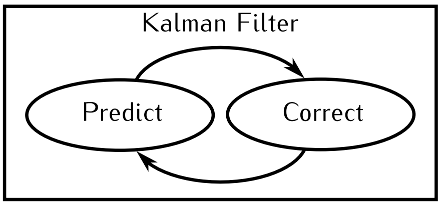


*Figure 2.2: Basic Kalman filter outline with two recursive steps.*

### 2.3.3 Extended Kalman Filter

In standard form, the KF requires a system to be linear in order to estimate the state in an optimal way. Many systems however, do not fit the linear model described by (2.39) and (2.41). The Extended Kalman Filter (EKF) provides a solution for estimating the state in these types of systems by linearising the non-linear equations as first order approximations around the current estimates [13].

#### 2.3.3.1 System Model

A more general system model using a state vector that applies to non-linear systems uses differentiable functions:

$$\begin{aligned} \boldsymbol{x}_k &= f_k\left(\boldsymbol{x}_{k-1},\ \boldsymbol{u}_k\right) + \boldsymbol{w}_k & (2.55)\\ \boldsymbol{z}_k &= h_k\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k. & (2.56) \end{aligned}$$

Both noise terms must again satisfy (2.40) and (2.42).

#### 2.3.3.2 Linearisation

The standard Kalman equations can no longer be applied as the matrices are replaced by non-linear functions. The solutions consists of computing the partial derivatives of the functions resulting in Jacobian matrices and evaluating them at the current state estimate.

$$\begin{aligned} \mathrm{A}_k &= \left.\frac{\partial f_k}{\partial \boldsymbol{x}}\right|_{(\hat{\boldsymbol{x}}_{k-1},\ \boldsymbol{u}_k)} & (2.57)\\ \mathrm{H}_k &= \left.\frac{\partial h_k}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_k^-} & (2.58) \end{aligned}$$

#### 2.3.3.3 Filter Algorithm

Using the defined Jacobians, the standard filter equations can easily be rewritten. For the prediction step:

$$\begin{aligned} \hat{\boldsymbol{x}}_k^- &= f_k\left(\hat{\boldsymbol{x}}_{k-1},\ \boldsymbol{u}_k\right)\\ \mathrm{P}_k^- &= \mathrm{A}_k\,\mathrm{P}_{k-1}\,\mathrm{A}_k^T + \mathrm{Q}_k & (2.59) \end{aligned}$$

and for the correction step:

$$\begin{aligned} \mathrm{K}_k &= \mathrm{P}_k^-\,\mathrm{H}_k^T\left(\mathrm{H}_k\,\mathrm{P}_k^-\,\mathrm{H}_k^T + \mathrm{R}_k\right)^{-1}\\ \hat{\boldsymbol{x}}_k &= \hat{\boldsymbol{x}}_k^- + \mathrm{K}_k\left(\boldsymbol{z}_k - h_k\left(\hat{\boldsymbol{x}}_k^-\right)\right) & (2.60)\\ \mathrm{P}_k &= \left(\mathrm{I}_{n\times n} - \mathrm{K}_k\,\mathrm{H}_k\right)\mathrm{P}_k^- \end{aligned}$$

By using Jacobians evaluated at the current state estimate however, the EKF only approximates the optimal estimator, unlike the standard version. Furthermore, errors in the estimations result in a linearisation around the wrong center point and could lead to instability of the filter.

### 2.3.4 Sigma Point Kalman Filters

By linearizing the system equations, the EKF ignores the fact that the state is actually a random variable with an inherent uncertainty, resulting in a suboptimal estimation of the state. Moreover, an EKF will only take into account the first derivative, ignoring higher order moments of the distribution, which results in poor performance for highly non-linear systems [14].

Sigma Points Kalman Filters (SPKFs), based on statistical linearisation, were developed to address this issue. The sigma point approach consists of approximating the statistical distribution by a discrete set of points, which are called sigma points. These sigma points are then transformed by applying the non-linear system equations, resulting in a transformed set of points, from which the propagated mean and variance can be calculated. It can been shown that the resulting mean and variance are accurately calculated to at least the second order [15].

#### 2.3.4.1 The Sigma Point Approach

Suppose a random variable with a known statistic undergoes a non-linear transformation, resulting in a new variable:

$$\boldsymbol{y} \;=\; g\left(\boldsymbol{x}\right) \tag{2.61}$$

Using the sigma point approach, the statistic of the resulting variable can be approximated. First, sigma points must be chosen in such a way that their distribution reflects the statistic of a random variable $\boldsymbol{x} \in \mathcal{R}^n$. Therefore, all of the points are chosen symmetrically around the mean $\bar{\boldsymbol{x}}$ with a spread that captures the covariance $\mathsf{P}_{\boldsymbol{x}}$. A set of $2n+1$ sigma points $\mathcal{X}$ that fulfills these requirements is given by:

$$\mathcal{X}|_i \;=\; \begin{cases} \bar{\boldsymbol{x}} & i = 0 \\ \bar{\boldsymbol{x}} \;+\; \sqrt{n}\;\mathsf{S}_{\boldsymbol{x}}|_i - 1 & 1 \le i \le n \\ \bar{\boldsymbol{x}} \;-\; \sqrt{n}\;\mathsf{S}_{\boldsymbol{x}}|_i - n - 1 & n < i \le 2n \end{cases} \tag{2.62}$$

where $.|_i$ denotes the $i$-th column of a matrix and $\mathsf{S}_{\boldsymbol{x}}$ represents the square root matrix of the covariance $\sqrt{\mathsf{P}_{\boldsymbol{x}}}$ as defined by the Cholesky decomposition [16]:

$$\mathsf{P}_{\boldsymbol{x}} \;=\; \mathsf{S}_{\boldsymbol{x}}\,\mathsf{S}_{\boldsymbol{x}}^T. \tag{2.63}$$

This decomposition is only defined for positive-definite matrices and can be calculated by fairly straight-forward algorithms [17]. The square root matrix $\mathsf{S}_{\boldsymbol{x}}$ is a lower triangular matrix and, given the fact that the covariance matrix is symmetrical, it only contains real elements.

A new set of sigma points is now calculated by transforming the original set of sigma points using the given non-linear function:

$$\mathcal{Y}_i \;=\; g\left(\mathcal{X}_i\right) \tag{2.64}$$

From this new set, the resulting mean $\bar{\boldsymbol{y}}$ and covariance of the transformed variable $\mathrm{P}_{\boldsymbol{y}}$, as well as the cross-covariance between the input and output variable $\mathrm{P}_{\boldsymbol{xy}}$, can be approximated:

$$\bar{\boldsymbol{y}} \;\approx\; \frac{1}{2n+1}\sum_{i=0}^{2n}\mathcal{Y}_i \tag{2.65}$$

$$\mathrm{P}_{\boldsymbol{y}} \;\approx\; \frac{1}{2n+1}\sum_{i=0}^{2n}\left[\mathcal{Y}_i \;-\; \bar{\boldsymbol{y}}\right]\left[\mathcal{Y}_i \;-\; \bar{\boldsymbol{y}}\right]^T \tag{2.66}$$

$$\mathrm{P}_{\boldsymbol{xy}} \;\approx\; \frac{1}{2n+1}\sum_{i=0}^{2n}\left[\mathcal{X}_i \;-\; \bar{\boldsymbol{x}}\right]\left[\mathcal{Y}_i \;-\; \bar{\boldsymbol{y}}\right]^T \tag{2.67}$$

#### 2.3.4.2 Filter Implementation

Two popular SPKF implementations exist known as the Unscented Kalman Filter (UKF) [18] and the Central Difference Kalman Filter (CDKF) [19, 20]. Both filters use sigma points to approximate the statistical distribution of the state estimate. The difference between them is that the UKF applies the Scaled Unscented Transformation [21], while the CDKF implements the Stirling polynomial interpolation [22]. This difference is mainly reflected in the values of the scalar coefficients and the approximation of the a posteriori estimate error covariance. In practical implementations however, there is little difference in the performance of both filter implementations.

**Prediction Sigma Points**

The first step in SPKFs consists of linearisation of the process model equation (2.55) by defining a set of sigma points. Using a shorthand notation for (2.62), the sigma point collection is given by:

$$\mathcal{X}_k^- \;=\; \left[\hat{\boldsymbol{x}}_{k-1},\; \hat{\boldsymbol{x}}_{k-1} \;+\; \eta\sqrt{\mathrm{P}_{k-1}},\; \hat{\boldsymbol{x}}_{k-1} \;-\; \eta\sqrt{\mathrm{P}_{k-1}}\right], \tag{2.68}$$

where

$$\eta \;=\; \begin{cases}\sqrt{n \;+\; \lambda} & UKF \\ h & CDKF\end{cases} \tag{2.69}$$

$n$ is the state dimension, and $\lambda$ and $h$ determine the spread of the sigma points for respectively a UKF and a CDKF.

The size of the spreading parameter dictates the radius of the sphere that bounds the sigma points around the statistical mean. When the spreading is set too high, local non-linearities may not be incorporated in the sigma point distribution after transformation, keeping the sigma points to close to the mean might not capture any non-linear effects at all.

For the UKF, $\lambda$ is mostly expressed as:

$$\lambda = \alpha^2 (n + \kappa) - n, \tag{2.70}$$

where $\kappa \geq 0$ determines the additional spread to the state vector size $n$ and $0 \leq \alpha \leq 1$ allows to resize the radius of the bounding sphere. Originally $\alpha$ was introduced to avoid problems of sampling non-local effects in the sigma points when $n$ is high [21]. A negative $\kappa$ value could result in an unstable filter, so $\alpha$ would be used to reduce the spreading.

The optimal value for the $h \geq 1$ parameter in CDKFs is dictated by the distribution of the state variable. It turns out that $h^2$ should approach the kurtosis of this distribution in order to minimize estimation errors [23].

### Prediction

The non-linear process model equation can now be used to determine a new set of transformed sigma points whose distribution approximates the prediction statistic:

$$\mathcal{Y}_k = f\left(\mathcal{X}_k^-, \boldsymbol{u}_k\right) \tag{2.71}$$

The a priori estimation can now be determined by applying (2.65) with modified weighing coefficients:

$$\boldsymbol{x}_k^- = \sum_{i=0}^{2n} W_i^m \left.\mathcal{Y}_k\right|_i, \tag{2.72}$$

where for the UKF

$$W_i^m = \begin{cases} \dfrac{\lambda}{n + \lambda} & i = 0 \\ \\ \dfrac{1}{2(n + \lambda)} & 1 \leq i \leq 2n \end{cases} \tag{2.73}$$

and for the CDKF

$$W_i^m = \begin{cases} \dfrac{1}{2\,h^2} & i = 0 \\ \\ \dfrac{h^2 - n}{h^2} & 1 \le i \le 2n \end{cases} \tag{2.74}$$

The estimate error covariance is calculated in a significantly different manner in both filter implementations. The UKF uses the calculated new mean as a basis, while the CDKF only relies on the sigma points themselves. Application of (2.66) quickly yields the UKF equation:

$$\mathrm{P}_k^- = \sum_{i=0}^{2n} W_i^c \left[\mathcal{Y}_k|_i - \boldsymbol{x}_k^-\right] \left[\mathcal{Y}_k|_i - \boldsymbol{x}_k^-\right]^T + \mathrm{Q}_k \tag{2.75}$$

where the weights $W_i^c$ are given by:

$$W_i^c = \begin{cases} \dfrac{\lambda}{n + \lambda} + \left(1 - \alpha^2 + \beta\right) & i = 0 \\ \\ \dfrac{1}{2\,(n + \lambda)} & 1 \le i \le 2n \end{cases} \tag{2.76}$$

Compared to the weighting coefficients for the mean $W_i^m$, the zeroth order coefficient includes an extra term containing the scaling parameter $\alpha$ that was already introduced in (2.70) and a new parameter $\beta \ge 0$. When additional information is known about the higher order moments of the statistical distribution of the state (e.g. the kurtosis), $\beta$ may be used to incorporate this knowledge and avoid errors in higher order terms.

The CDKF version follows from the second order Stirling interpolation approximation [20], which resembles a Taylor series expansion where the derivatives are replaced by central divided differences [22].

$$\begin{aligned} \mathrm{P}_k^- = \sum_{i=1}^{n} \Big[ & W_i^{c1} \left(\mathcal{Y}_k|_i - \mathcal{Y}_k|_{i+n}\right)^2 \\ & + W_i^{c2} \left(\mathcal{Y}_k|_i + \mathcal{Y}_k|_{i+n} - 2\,\mathcal{Y}_k|_0\right)^2 \Big] + \mathrm{Q}_k \end{aligned} \tag{2.77}$$

where $(.)^2$ is used as a shorthand for the vectorial outer product $[.][.]^T$ and

$$\begin{aligned} W_i^{c1} &= \frac{1}{4\,h^2} & 1 \le i \le n \\ W_i^{c2} &= \frac{h^2 - 1}{4\,h^4} & 1 \le i \le n \end{aligned} \tag{2.78}$$

### Correction Sigma Points

Linearisation of the measurement model (2.56) is obtained by defining a second set of sigma points centered around the a priori estimate:

$$\mathcal{X}_k \;=\; \left[\hat{\boldsymbol{x}}_k^-,\; \hat{\boldsymbol{x}}_k^- \;+\; \eta\sqrt{\mathrm{P}_k^-},\; \hat{\boldsymbol{x}}_k^- \;-\; \eta\sqrt{\mathrm{P}_k^-}\right], \tag{2.79}$$

The spreading parameters are again defined by (2.69). However, it is not necessary that both sigma points use the same spreading. As the non-linear measurement transition function $h\,(.)$ could exhibit very different characteristics compared to the state transition function $f\,(.)$, different spreading values might even improve the filter performance.

### Measurement Prediction

Before the actual correction can be calculated, a measurement prediction must be determined based on the transformation of the sigma points:

$$\mathcal{Z}_k \;=\; h\,(\mathcal{X}_k) \tag{2.80}$$

Both the measurement prediction and the error covariance of this measurement can now be determined using equations very similar to (2.72), (2.75) and (2.77):

$$\boldsymbol{z}_k^- \;=\; \sum_{i=0}^{2n} W_i^m\; \mathcal{Z}_{i,k} \tag{2.81}$$

$$\mathrm{P}_{\boldsymbol{z}_k} \;=\; \begin{cases} \sum_{i=0}^{2n} W_i^c\; \left(\mathcal{Z}_k|_i - \boldsymbol{z}_k^-\right)\left(\mathcal{Z}_k|_i - \boldsymbol{z}_k^-\right)^T \;+\; \mathrm{R}_k & UKF \\[2ex] \sum_{i=1}^{n}\Big[W_i^{c1}\;\left(\mathcal{Z}_k|_i \;-\; \mathcal{Z}_k|_{i+n}\right)^2 & \\ \quad +W_i^{c2}\;\left(\mathcal{Z}_k|_i \;+\; \mathcal{Z}_k|_{i+n} - 2\;\mathcal{Z}_k|_0\right)^2\Big] \;+\; \mathrm{R}_k & CDKF \end{cases} \tag{2.82}$$

where all of the weights are defined as in (2.73), (2.76), (2.74) and (2.78) and $(.)^2$ is used as a shorthand for the vectorial outer product $[.][.]^T$.

Aside from the measurement prediction covariance, the cross-covariance matrix between the measurement prediction and the a priori estimate must also be determined. In the UKF implementation the formula for this matrix directly follows from (2.67):

$$\mathrm{P}_{\boldsymbol{x}_k\boldsymbol{z}_k} = \sum_{i=0}^{2n} W_i^c\;\left(\mathcal{X}_k|_i - \boldsymbol{x}_k^-\right)\left(\mathcal{Z}_k|_i - \boldsymbol{z}_k^-\right)^T \tag{2.83}$$

As with the covariance matrix, the CDKF does not make use of previously calculated means, yet only uses the sigma points to obtain the cross-covariance. The formula again finds its origin in the second order Stirling approximation [20].

$$\mathrm{P}_{\boldsymbol{x}_k \boldsymbol{z}_k} = \sqrt{W_i^{c1}\, \mathrm{P}_k^-}\ \left[ \mathcal{Z}_k|_{1:n} \ - \ \mathcal{Z}_k|_{n+1:2n} \right]^T \tag{2.84}$$

**Correction**

The final step of the SPKF algorithm uses all of the results of the previous steps to obtain the a posteriori estimate and its error covariance. First, the Kalman gain is determined:

$$\mathrm{K}_k \ = \ \mathrm{P}_{\boldsymbol{x}_k \boldsymbol{z}_k} \mathrm{P}_{\boldsymbol{z}_k}^{-1} \tag{2.85}$$

The gain is then used to calculate the final estimate and the estimate error covariance using formulas that resemble the standard KF equations (2.51) and (2.52):

$$\hat{\boldsymbol{x}}_k \ = \ \hat{\boldsymbol{x}}_k^- \ + \ \mathrm{K}_k \left( \boldsymbol{z}_k \ - \ \boldsymbol{z}_k^- \right) \tag{2.86}$$

$$\mathrm{P}_k \ = \ \mathrm{P}_k^- - \mathrm{K}_k \, \mathrm{P}_{\boldsymbol{z}_k} \, \mathrm{K}_k^T \tag{2.87}$$

#### 2.3.4.3 Performance

As a general rule, more sigma points mean a better approximation of the statistic to a higher order. However, extra sigma points also increase the numerical complexity and number of operations to be performed. The additional cost of using sigma points is countered by the fact that no troublesome analytical derivatives of complex non-linear systems need to be calculated.

Although the introduced spreading parameters have some influence on the outcome of the SPKFs, their influence is rather limited. A global optimum has not yet been found and it seems more likely that the optimum is problem specific. It is however important to mention that some combinations result in numerically unstable filters, which must be avoided at all cost.

### 2.3.5 Hybrid Kalman Filters

The described filter algorithms can also be mixed together to build an estimator. In some cases, the process model is a linear equation, while the measurement model is highly non-linear. In this case, the prediction can be implemented by standard KF equations and, depending on the non-linearity, the correction either adopts an EKF or SPKF shape.

### 2.3.6 Adaptive Kalman Filters

The performance of the Kalman filter as an estimator greatly depends on the accuracy by which the process is described. More specifically, the quality of the a priori information about the process noise and measurement noise directly influences the performance of the estimator [24]. However, this information depends on factors that are difficult to obtain, such as process dynamics and surrounding environment. In some cases, these factors could even be subject to changes during estimation. An adaptive estimation scheme where a learning process dynamically adjusts the parameters tackles these issues and makes the estimator less sensitive to errors in the a priori values.

The adaptive Kalman filter uses the innovation sequence to adjust the noise covariance matrices $\mathrm{Q}_k$ and $\mathrm{R}_k$ in order to suit the current situation [25]. Based on the whiteness of the innovation sequence, the covariance matrices are adapted as follows:

$$\mathrm{R}_k = \mathrm{C}_{\boldsymbol{v}_k} - \mathrm{H}_k \mathrm{P}_k^- \mathrm{H}_k^T \tag{2.88}$$

$$\mathrm{Q}_k = \mathrm{K}_k \mathrm{C}_{\boldsymbol{v}_k} \mathrm{K}_k^T \tag{2.89}$$

where $\mathrm{C}_{\boldsymbol{v}_k}$ is an approximation of the covariance matrix of the innovation, calculated as a moving average from the previous innovation values:

$$\mathrm{C}_{\boldsymbol{v}_k} = \frac{1}{N} \sum_{i=k-N+1}^{k} \boldsymbol{v}_i \boldsymbol{v}_i^T = \frac{1}{N} \sum_{i=k-N+1}^{k} \left(\boldsymbol{z}_i - \mathrm{H}_i \hat{\boldsymbol{x}}_i^-\right) \left(\boldsymbol{z}_i - \mathrm{H}_i \hat{\boldsymbol{x}}_i^-\right)^T \tag{2.90}$$

The window size $N$ should at least exceed the number of states and the number of update measurements, as filter instability is likely to occur otherwise. A large window size will be unable to track high frequency changes in process dynamics, while a short window size might adapt parameters too quickly to adjust to local or temporary effects. A trade-off clearly needs to be made, depending on the expected rate of change in dynamics and the update rate of the filter.

The adaptive principle can be applied to either type of Kalman filter, as the innovation sequence is a vital part of the algorithm. Using adaptive schemes however, care must be taken to avoid instability of the filter.

## 2.4 Orientation Estimator Design

In this section, the actual design of the orientation estimation filter is highlighted. Where the previous sections can be seen as an introduction to the principles that are used, this section covers some of the actual realisations. The heart of the estimator consists of a Kalman filter, yet several additional elements are present around the filter to improve the performance. Although the main focus of this work is on the design of a gyro-less tracking system (Magnetic Field and Gravitational (MFG)), a nine DOF attitude filter (MARG) will also be proposed and used in further chapters as a reference basis for comparison. The future development of this estimator within the Centre for Microsystems Technology (CMST) lab is in the hands of Pietro Salvo.

First, an overview of the estimators is given by describing the architecture with block diagrams. Then, more details are supplied about MEMS sensors and necessary steps to validate their data. Next, the Kalman filter principle is applied to the problem at hand and mathematical expressions are derived describing the system model. Finally, the orientation estimation procedure is summarised and the adaptive aspect is highlighted.

## 2.4.1 Filter Architecture

Two filter architectures are introduced in this section, though both are very much alike. The main difference is found in the presence of the gyroscope, which provides additional information for the MARG version of the filter.

### 2.4.1.1 MARG Filter

Figure 2.3 depicts the block diagram of the MARG filter. The heart of the filter consists of an adaptive Kalman filter providing the actual orientation estimate. Three vectors containing three dimensional sensor information forms the input, as well as the covariance matrices, while the output consists of the orientation in either Euler angles or as a quaternion.

All of the sensor outputs are processed before being used by the filter. The accelerometer and magnetometer output are combined in a single vector and form the measurement vector $\boldsymbol{z}$ of the Kalman filter. The gyroscope input is added in the prediction step of the Kalman filter as the input term $\boldsymbol{u}$. This way, all of the modeling equations of the correction step are essentially equal in both the MARG and the MFG filter. There is also a possibility for optional post-processing of the output sequence.

### 2.4.1.2 MFG Filter

The block diagram for the MFG filter is displayed in figure 2.4. A Kalman filter is again at the center of the diagram. The input now only consists of processed accelerometer and magnetometer output vectors.

Two major differences are immediately clear. First, as the gyroscope is not present, an approximation of the orientation rate by discrete derivation

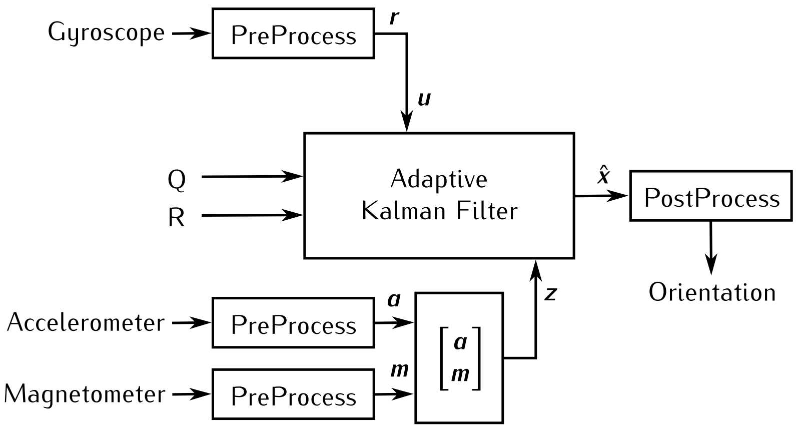


*Figure 2.3: Block diagram of the MARG type filter.*

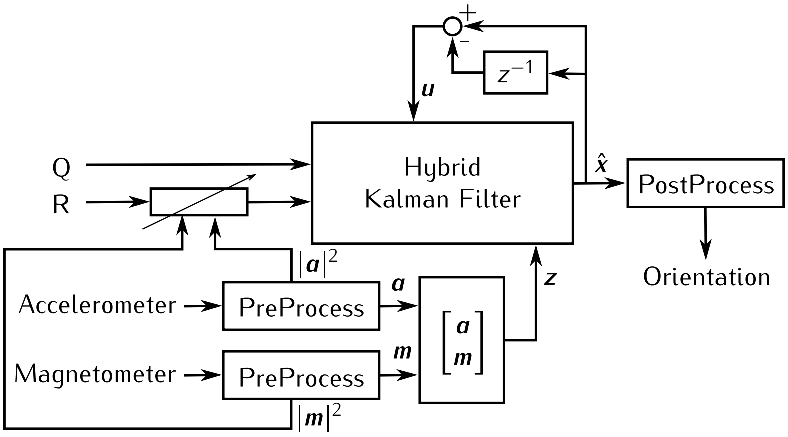


*Figure 2.4: Block diagram of the MFG type filter.*

of the state is fed back to the filter as the input term $\boldsymbol{u}$. Second, the filter is no longer adaptive, yet the magnitude of the measured acceleration and magnetic field will influence the measurement noise covariance matrix. Under normal circumstances, both measured vectors should approximately have a fixed magnitude, however, when disturbances occur, this will no longer be the case and the covariance should be adjusted. The adaptive Kalman filter has been replaced by this custom adaptation as the averaging cannot track the disturbance quickly enough, while direct adaptation can.

### 2.4.2 Sensor Signals

The output of several MEMS sensors is combined to obtain full three dimensional orientation. The absolute minimum for a drift-free estimator requires an accelerometer and a magnetometer. A gyroscope is usually added to obtain short-term information which can be integrated to obtain an estimate via dead reckoning. Internally, all of these sensors consist of mechanical elements that are subject to the quantity they are designed to measure. Electrical circuits are added to provide a value indicating the size of this quantity and possibly to implement an interface or processing unit.

Although the output is proportional to the desired quantity, offset errors and noise still distort the readout values. Therefore, a sensor output model is proposed and calibration procedures exist to match the output to the quantity at hand.

#### 2.4.2.1 Sensor Output Model

The output of a three dimensional MEMS sensor is usually modeled according to equation (2.91). The vectors $\boldsymbol{u} \in \mathcal{R}^3$ and $\boldsymbol{q} \in \mathcal{R}^3$ respectively stand for the sensor output and the actual quantity to be measured along each of the three orthogonal sensitivity axes X, Y and Z. The bias vector is denoted by $\boldsymbol{b} \in \mathcal{R}^3$, $\mathsf{K} \in \mathcal{R}^{3\times3}$ corresponds to the gain matrix containing the scale factors on its diagonal and the cross sensitivity values as off-diagonal elements and $\boldsymbol{w} \in \mathcal{R}^3$ stands for a Gaussian white noise term.

$$\boldsymbol{u} \;=\; \mathsf{K}\boldsymbol{q} \;+\; \boldsymbol{b} \;+\; \boldsymbol{w}, \tag{2.91}$$

where

$$\mathsf{K} \;=\; \begin{bmatrix} k_{xx} & k_{xy} & k_{xz} \\ k_{yx} & k_{yy} & k_{yz} \\ k_{zx} & k_{zy} & k_{zz} \end{bmatrix} \qquad\qquad \boldsymbol{b} \;=\; \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \tag{2.92}$$

The covariance matrix of the noise $\mathsf{P}_{\boldsymbol{w}} \in \mathcal{R}^{3\times3}$, the gain matrix and bias vector depend on the internal construction of the sensor and on environmental factors. In the following, it will be assumed that all of these variables do not change over time, and thus performing calibration can eliminate the effect of both the bias and the gain. The noise however, is a random signal that will be dealt with in the actual estimation procedure.

#### 2.4.2.2 Calibration

Accelerometers and magnetometers can be calibrated in a similar way, however, gyroscopes require a different approach. The reason lies in the measured quantity of the sensors. Gyroscopes measure rotational speed and thus measure zero whenever the component is stationary. Accelerometers and magnetometers however, measure a fixed vector value, respectively earth's gravity and magnetic field. When these sensors are stationary, their output will depend on the overall orientation. Therefore, these sensors can be calibrated by positioning them in well known directions while gyroscopes must be in motion to perform calibration.

### Accelerometer

The calibration procedure describes a method to estimate the bias vector and the gain matrix. In order to perform the calibration, six measurements must be executed [26]. Three measurements with gravity successively parallel to one of the three axes, resulting in an acceleration of 1 g along this particular axis, and three measurements with gravity anti parallel. The resulting six measured vectors can be calculated according to equation (2.91) under the assumption that the noise term equals zero. This assumption is strengthened by the fact that each measurement is actually an average of multiple samples, reducing the noise term to approximate its mean. For gravity along the X-axis, the following expressions result:

$$\begin{aligned}
\boldsymbol{u}^{x+} &= \begin{bmatrix} u_x^{x+} \\ u_y^{x+} \\ u_z^{x+} \end{bmatrix} = \mathrm{K}^a \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{b}^a = \begin{bmatrix} k_{xx}^a + b_x^a \\ k_{xy}^a + b_y^a \\ k_{xz}^a + b_z^a \end{bmatrix} \\
\boldsymbol{u}^{x-} &= \begin{bmatrix} u_x^{x-} \\ u_y^{x-} \\ u_z^{x-} \end{bmatrix} = \mathrm{K}^a \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{b}^a = \begin{bmatrix} -k_{xx}^a + b_x^a \\ -k_{xy}^a + b_y^a \\ -k_{xz}^a + b_z^a \end{bmatrix}
\end{aligned} \qquad (2.93)$$

The signed superscript denotes to which axis gravity is parallel (+) or anti parallel (-). The superscript $a$ indicates these values apply to the accelerometer. Similar equations can be derived for gravity along the Y- and Z-axis:

$$\boldsymbol{u}^{y+} = \begin{bmatrix} k_{yx}^a + b_x^a \\ k_{yy}^a + b_y^a \\ k_{yz}^a + b_z^a \end{bmatrix} \qquad \boldsymbol{u}^{y-} = \begin{bmatrix} -k_{yx}^a + b_x^a \\ -k_{yy}^a + b_y^a \\ -k_{yz}^a + b_z^a \end{bmatrix} \qquad (2.94)$$

$$\boldsymbol{u}^{z+} = \begin{bmatrix} k_{zx}^a + b_x^a \\ k_{zy}^a + b_y^a \\ k_{zz}^a + b_z^a \end{bmatrix} \qquad \boldsymbol{u}^{z-} = \begin{bmatrix} -k_{zx}^a + b_x^a \\ -k_{zy}^a + b_y^a \\ -k_{zz}^a + b_z^a \end{bmatrix} \qquad (2.95)$$

The different parameters can now be estimated by applying the following formulas to the measured values:

$$\boldsymbol{b}^a = \frac{1}{6}\left[\sum_{t=x,y,z} \boldsymbol{u}^{t+} + \sum_{t=x,y,z} \boldsymbol{u}^{t-}\right] \tag{2.96}$$

$$k^a_{ij} = \frac{1}{2}\left(u^{i+}_j - u^{i-}_j\right), \; \forall i,j \in \{x,y,z\} \tag{2.97}$$

Using these estimations, the acceleration can be calculated from the measured values of the accelerometer by inverting equation (2.91) under the assumption that the noise is zero:

$$\boldsymbol{a} = (\mathrm{K}^a)^{-1}\left(\boldsymbol{u}^a - \boldsymbol{b}^a\right), \tag{2.98}$$

where $\boldsymbol{q}$ has been replaced by $\boldsymbol{a}$ to indicate acceleration.

### Magnetometer

Although the calibration procedure resembles the one used for accelerometers, one important difference exists. The actual direction of the magnetic field vector is much harder to find than the direction of gravity. If a magnetometer is positioned according to gravity, equations (2.93) through (2.95) will contain extra terms due to the horizontal component of the magnetic field and no longer offer a straight forward solution if the exact same heading is not employed in each measurement. There are three possible solutions to this problem that could still lead to a valid calibration.

The first solution neglects the cross sensitivity terms in the gain matrix K. When all of these terms are set to zero, the procedure described above can be applied and the magnetometer must be positioned parallel and anti parallel to gravity with each axis. The bias vector follows from equation (2.96) and scale factors from (2.97) with $i = j$.

The second solution requires the user to search for the maximal and minimal output values on each axis by attempting to align the axes to the magnetic field. This way, equations (2.93) through (2.95) are valid and all of the calibration variables can be determined. However, in this way, the noise term could have an influence on the final result. The measurements used to perform the calibration would no longer be an average of multiple samples, but rather represent an extreme output value, indicating that the noise term cannot be assumed to approach its mean value.

A final way to solve the problem is to take measurements at different headings and take the average over a number of full circle rotations. When aligning an axis to the gravity vector and rotating the magnetometer around the vertical, the output on the axis parallel to gravity should remain constant. The output on the other two axes however, describes one full period of a sine wave with each full rotation. So the average output over a number of full circle rotations yields a fixed value on one axis and zero on the other two. The calibration procedure for the accelerometer can now be applied to the average values obtained while rotating to estimate the gain matrix $\mathrm{K}^m$ and the bias vector $\boldsymbol{b}^m$.

#### Gyroscope

The calibration procedure for gyroscopes requires a more elaborate setup. Due to the fact that the sensors measure speed, a known angular rate must be applied to the gyroscope in order to determine the gain and offset parameters. Yet again, six measurements must be performed [27]. The sensor is placed parallel and anti parallel to gravity with each of its axes and a rotation around the vertical axis with a fixed known angular rate is applied. Although it is of course only required to perform the rotations around the actual sensor axes, gravity is chosen as a reference in order to allow all of the calibration procedures to be combined. The resulting measurements can be used in a similar manner as has been described above to determine the calibration values. The only difference lies in the gain matrix where the known angular rate $\omega$ should also be included:

$$k_{ij}^{g} = \frac{1}{2\,\omega}\left( u_j^{i+} - u_j^{i-} \right), \; \forall i,j \in \{x,y,z\} \tag{2.99}$$

Generally, the gyroscope must be rotated around each axis for approximately 10 to 15 minutes [28] and the known angular rate should be chosen somewhere in the middle of the sensor's range.

### 2.4.2.3 Sensor Output Processing

Before the sensor output can be used by the attitude filter, pre-processing steps are needed. These steps include naturally the calibration as described in the previous section, but also low pass filtering of the output data and normalisation, if required.

#### Accelerometer

Figure 2.5 is a block diagram of the pre-processing steps to which the accelerometer output is subjected. The first step implements (2.98) and thus completes the calibration. Then, the resulting signal is filtered by a digital low pass filter to eliminate high frequency noise. The cut-off frequency of this filter must be chosen such that relevant signals corresponding to human movement remain untouched. The final step normalises the signal. Under normal conditions, the filter expects the accelerometer to measure only gravity, yet movement also introduces accelerations that distort the signal. Normalising reduces the effect this distortion has on the eventual orientation estimate.

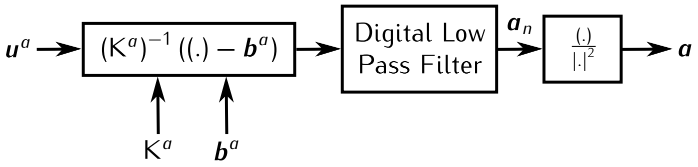


*Figure 2.5: Block diagram of accelerometer output processing.*

### Magnetometer

The magnetometer signal is subjected to similar pre-processing as the accelerometer output. Figure 2.6 displays the block diagram. Normalisation of the signal is performed not only to reduce the effect of disturbances but also originates in the fact that the magnitude of the earth magnetic field highly depends on the environment. Large quantities of iron in a building structure e.g. can reduce this magnitude significantly. By normalising, this effect is negated. However, normalisation also means that the output of the magnetometer will need to be compared to the normalised earth magnetic field vector, and not to the nominal vector.

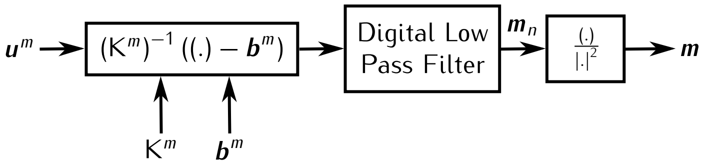


*Figure 2.6: Block diagram of magnetometer output processing.*

### Gyroscope

The pre-processing of the gyroscope signal is displayed in figure 2.7. Clearly, no normalisation is present here as the measured quantity is supposed to be integrated to obtain angular position.

## 2.4.3 Kalman Filter System Model

The system model must be defined for each type of filter, MARG and MFG with either Euler angles or quaternion representation. After selecting a suitable state vector, process and measurement model equations may be derived.

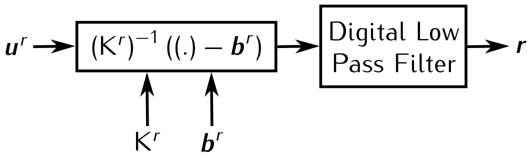


*Figure 2.7: Block diagram of gyroscope output processing.*

#### 2.4.3.1 State Vector

The state vector $\boldsymbol{x}$ of the KF must keep track of the full three dimensional orientation of the sensor node. Two possible representations will be investigated: Euler angles in the roll-pitch-yaw convention and quaternions.

#### Euler Angles

Obviously the state must contain all three Euler angles, Roll, Pitch and Yaw, as defined in section 2.2.1:

$$\boldsymbol{x} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.100}$$

#### Quaternion

In the quaternion case, the state must incorporate all four components of the quaternion as defined in section 2.2.2:

$$\boldsymbol{x} = \mathbf{q}_x = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} \tag{2.101}$$

#### 2.4.3.2 Process Model

The process model predicts the evolution of the state vector $\boldsymbol{x}$. In the MARG version, the gyroscope output is added as the input term $\boldsymbol{u}$, which means that the process model is different for both filter types.

#### MFG Process Model

As only three dimensional orientation is present in the state, the process model will be a rather crude approximation of the actual process. In order to actually predict a certain movement, rather then static orientation, an approximation of the Euler angular velocity or the quaternion rate will be made using previous state estimations and adding a fraction of this approximation to the state vector:

$$\boldsymbol{x}_k \;=\; \boldsymbol{x}_{k-1} \;+\; \tau\;(\boldsymbol{x}_{k-1} \;-\; \boldsymbol{x}_{k-2}) \;+\; \boldsymbol{w}_k, \tag{2.102}$$

where $\tau$ will further be referred to as the feedback gain and $\boldsymbol{w}$ represents the process noise with covariance matrix Q. As movement is assumed to be random, all of the vector components are uncorrelated and Q is modeled as:

$$\mathrm{Q} \;=\; q\;\mathrm{I}_{n\times n}, \tag{2.103}$$

where $\mathrm{I}_{n\times n}$ represents the identity matrix of size three for the Euler and size four for the quaternion version.

Note that the process model equation is clearly linear and that the matrices are not changing over time, hence no $k$ subscript is present.

### MARG Process Model

The gyroscope offers useful information about the the current angular rate, which can be used to provide an accurate prediction of the state. The gyroscope output is considered as the input vector $\boldsymbol{u}$:

$$\boldsymbol{u} \;=\; \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{2.104}$$

As the rate sensor is attached to the sensor node, it will deliver body rates which must still be converted. In case of a Euler angles state, conversion is executed using (2.11):

$$f\left(\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}\right) \;=\; \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} + \Delta T \begin{bmatrix} r_x \;-\; r_y \sin\phi \tan\theta \;+\; r_z \cos\phi \tan\theta \\ r_y \cos\phi \;+\; r_z \sin\phi \\ -\; r_y \sin\phi \sec\theta \;+\; r_z \cos\phi \sec\theta \end{bmatrix} + \boldsymbol{w}_k. \tag{2.105}$$

In the case of a quaternion state, (2.30) completes the conversion:

$$f\left(\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}, \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}\right) \;=\; \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} \;+\; \frac{\Delta T}{2}\left(\mathbf{q}_{\boldsymbol{x}_{k-1}} \;\otimes\; \mathbf{q}_r\right) \;+\; \boldsymbol{w}_k. \tag{2.106}$$

In both equations, $\Delta T$ equals the time between two updates. As movement is assumed to be random, the noise covariance equation (2.103) still applies. However, the source of the noise can now be found in the gyroscope output and not in the crude approximation of motion that was used in the MFG filter. It is thus assumed that the covariance will be significantly lower in this model.

Although the process model for the MFG filter is linear, this is clearly not the case for the MARG filter. A direct result is that linearisation will be necessary using either EKF or SPKF methods.

#### 2.4.3.3 Measurement Model

The measurement model relates the measured value $\boldsymbol{z}$ to the value of the state vector $\boldsymbol{x}$. The measured value consists of two parts: $\boldsymbol{z}_{acc}$ and $\boldsymbol{z}_{mag}$, which correspond to respectively the output of the accelerometer and the magnetometer. The accelerometer is expected to measure earth's gravity field in the coordinate system of the sensor node. Similarly, the magnetometer is expected to measure earth's magnetic field in sensor coordinates. Let $\boldsymbol{g}$ and $\boldsymbol{h}$ be the vectors respectively representing earth's gravity field and earth's magnetic field in earth coordinates. The measurement model for both representations can then be determined using the formalisms introduced in section 2.2.

#### Euler

Let $\mathrm{R}_x$, $\mathrm{R}_y$, and $\mathrm{R}_z$ be the rotation matrices corresponding to rotations of respectively $\phi$ around the X-axis, $\theta$ around the Y-axis and $\psi$ around the Z-axis as described in (2.2) – (2.4). The expected sensor output can then be written as:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{z}_{acc} \\ \boldsymbol{z}_{mag} \end{bmatrix} = h(\boldsymbol{x}) = \begin{bmatrix} \mathrm{R}_x\,\mathrm{R}_y\,\mathrm{R}_z\,\boldsymbol{g} \\ \mathrm{R}_x\,\mathrm{R}_y\,\mathrm{R}_z\,\boldsymbol{h} \end{bmatrix} + \boldsymbol{v} \tag{2.107}$$

When choosing the earth bound reference system corresponding to a zero state in such a way that gravity coincides with the Z-axis and the Y-axis points to the magnetic North, the gravity and magnetic field vector become:

$$\boldsymbol{g} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \qquad \boldsymbol{h} = \begin{bmatrix} 0 \\ h_H \\ h_V \end{bmatrix} \tag{2.108}$$

Where $g$ equals the gravitational acceleration of 9.81 m/s$^2$ or 1 g and $h_H$ and $h_V$ represent the local horizontal and vertical component of the magnetic field. These components are subject to change over time and vary all over the globe, yet can be calculated using geomagnetic models. In the area of Ghent, Belgium, in March 2011, the horizontal component approximately equaled 19.6 nT and the vertical component 44.5 nT [29].

Using the expanded product of rotation matrices (2.5) and substituting (2.108) in (2.107), the measurement model can be rewritten as:

$$h\left(\begin{bmatrix}\phi\\\theta\\\psi\end{bmatrix}\right) = \begin{bmatrix} g\sin\theta \\ -g\sin\phi\cos\theta \\ g\cos\phi\cos\theta \\ -h_H\cos\theta\sin\psi + h_V\sin\theta \\ h_H\left(\cos\phi\cos\psi - \sin\phi\sin\theta\sin\psi\right) - h_V\sin\phi\cos\theta \\ h_H\left(\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi\right) + h_V\cos\phi\cos\theta \end{bmatrix} + \boldsymbol{v}. \tag{2.109}$$

An evaluation of these formulas requires a minimum of six trigonometric functions, 17 multiplications and five additions when reusing identical terms.

### Quaternion

Let $\mathbf{q}_g$ and $\mathbf{q}_h$ be the quaternions associated with the gravity and magnetic field vector as defined by (2.28). The expected sensor output can then be calculated as the rotation of these quaternions as described in (2.29):

$$\boldsymbol{z} = \begin{bmatrix} z_{acc} \\ z_{mag} \end{bmatrix} = h(\boldsymbol{x}) = \begin{bmatrix} \mathbf{q}_x \otimes \mathbf{q}_g \otimes \mathbf{q}_x^{-1} \\ \mathbf{q}_x \otimes \mathbf{q}_h \otimes \mathbf{q}_x^{-1} \end{bmatrix} + \boldsymbol{v} \tag{2.110}$$

Choosing the same earth-bound reference system as described in the Euler angle version above and substituting the expressions from (2.108) in (2.110) results in:

$$h\left(\begin{bmatrix}w\\x\\y\\z\end{bmatrix}\right) = \begin{bmatrix} 2g\left(wy + xz\right) \\ 2g\left(yz - wx\right) \\ g\left(w^2 - x^2 - y^2 + z^2\right) \\ 2h_H\left(xy - wz\right) + 2h_V\left(wy + xz\right) \\ h_H\left(w^2 - x^2 + y^2 - z^2\right) + 2h_V\left(yz - wx\right) \\ 2h_H\left(wx + yz\right) + h_V\left(w^2 - x^2 - y^2 + z^2\right) \end{bmatrix} + \boldsymbol{v}, \tag{2.111}$$

which requires a minimum of 22 multiplications and 12 additions to evaluate when reusing as many terms as possible.

### Measurement Noise

In all the above equations, $\boldsymbol{v}$ represents the sensor noise with covariance matrix R. The output noise is assumed to be uncorrelated between the two sensors and between the different axes. Furthermore, an even spread of the noise is expected for every axis of each sensor, resulting in:

$$\mathrm{R} = \begin{bmatrix} r_{acc}\,\mathrm{I}_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & r_{mag}\,\mathrm{I}_{3\times3} \end{bmatrix}. \tag{2.112}$$

The covariance of the noise present on the accelerometer output of a single axis is denoted by $r_{acc}$, for the magnetometer $r_{mag}$ is used. The identity matrix with three rows and columns is represented by $\mathrm{I}_{3\times3}$ and the zero matrix of the same dimensions by $0_{3\times3}$.

#### 2.4.3.4 Linearisation

When using an EKF for estimation in a non-linear system, partial derivatives of the model equations are required as described in section 2.3.3. Both the measurement and the MARG process model equations are non-linear.

**Euler angle MARG Process Model**

Application of (2.57) to (2.105) results in:

$$\begin{aligned}
\frac{\partial f(\boldsymbol{x},\ \boldsymbol{u})}{\partial \phi} &= \begin{bmatrix} 1 + \Delta T\left(-r_y \cos\phi \tan\theta - r_z \sin\phi \tan\theta\right) \\ \Delta T\left(-r_y \sin\phi + r_z \cos\phi\right) \\ \Delta T\left(-r_y \cos\phi \sec\theta - r_z \sin\phi \sec\theta\right) \end{bmatrix} \\
\frac{\partial f(\boldsymbol{x},\ \boldsymbol{u})}{\partial \theta} &= \begin{bmatrix} \Delta T\left(-r_y \dfrac{\sin\phi}{\cos^2\theta} + r_z \dfrac{\cos\phi}{\cos^2\theta}\right) \\ 1 \\ \Delta T\left(-r_y \dfrac{\sin\phi \tan\theta}{\cos\theta} + r_z \dfrac{\cos\phi \tan\theta}{\cos\theta}\right) \end{bmatrix} \qquad (2.113) \\
\frac{\partial f(\boldsymbol{x},\ \boldsymbol{u})}{\partial \psi} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\end{aligned}$$

**Quaternion MARG Process Model**

Using 2.57, the linearisation of (2.106) becomes:

$$\frac{\partial f\left(\boldsymbol{x},\ \boldsymbol{u}\right)}{\partial w} = \begin{bmatrix} 1 \\ -\frac{\Delta T}{2}\, r_x \\ -\frac{\Delta T}{2}\, r_y \\ -\frac{\Delta T}{2}\, r_z \end{bmatrix} \qquad \frac{\partial f\left(\boldsymbol{x},\ \boldsymbol{u}\right)}{\partial x} = \begin{bmatrix} -\frac{\Delta T}{2}\, r_x \\ 1 \\ -\frac{\Delta T}{2}\, r_z \\ \frac{\Delta T}{2}\, r_y \end{bmatrix}$$

$$\frac{\partial f\left(\boldsymbol{x},\ \boldsymbol{u}\right)}{\partial y} = \begin{bmatrix} -\frac{\Delta T}{2}\, r_y \\ \frac{\Delta T}{2}\, r_z \\ 1 \\ -\frac{\Delta T}{2}\, r_x \end{bmatrix} \qquad \frac{\partial f\left(\boldsymbol{x},\ \boldsymbol{u}\right)}{\partial z} = \begin{bmatrix} -\frac{\Delta T}{2}\, r_z \\ -\frac{\Delta T}{2}\, r_y \\ \frac{\Delta T}{2}\, r_x \\ 1 \end{bmatrix} \tag{2.114}$$

### Euler Angle Measurement Model

Applying (2.58) to (2.107) yields the following linearisation:

$$\begin{aligned}
\frac{\partial h\left(\boldsymbol{x}\right)}{\partial \phi} &= \begin{bmatrix} 0 \\ -g\cos\phi\cos\theta \\ -g\sin\phi\cos\theta \\ 0 \\ -h_H\left(\sin\phi\cos\psi\ +\ \cos\phi\sin\theta\sin\psi\right)\ -\ h_V\cos\phi\cos\theta \\ h_H\left(\cos\phi\cos\psi\ -\ \sin\phi\sin\theta\sin\psi\right)\ -\ h_V\sin\phi\cos\theta \end{bmatrix} \\
\frac{\partial h\left(\boldsymbol{x}\right)}{\partial \theta} &= \begin{bmatrix} g\cos\theta \\ g\sin\phi\sin\theta \\ -g\cos\phi\sin\theta \\ h_H\sin\theta\sin\psi\ +\ h_V\cos\theta \\ -h_H\sin\phi\cos\theta\sin\psi\ +\ h_V\sin\phi\sin\theta \\ h_H\cos\phi\cos\theta\sin\psi\ +\ h_V\cos\phi\sin\theta \end{bmatrix} \\
\frac{\partial h\left(\boldsymbol{x}\right)}{\partial \psi} &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ -h_H\cos\theta\cos\psi \\ -h_H\left(\cos\phi\sin\psi\ +\ \sin\phi\sin\theta\cos\psi\right) \\ h_H\left(-\sin\phi\sin\psi\ +\ \cos\phi\sin\theta\cos\psi\right) \end{bmatrix}
\end{aligned} \tag{2.115}$$

**Quaternion Measurement Model**

In the quaternion case, (2.58) must be applied to (2.110) to result in the following equations:

$$\frac{\partial h\left(\boldsymbol{x}\right)}{\partial w} = \begin{bmatrix} 2\,g\,y \\ -2\,g\,x \\ 2\,g\,w \\ -2\,h_H\,z \;+\; 2\,h_V\,y \\ 2\,h_H\,w \;-\; 2\,h_V\,x \\ 2\,h_H\,x \;+2\,h_V\,w \end{bmatrix} \qquad \frac{\partial h\left(\boldsymbol{x}\right)}{\partial x} = \begin{bmatrix} 2\,g\,z \\ -2\,g\,w \\ -2\,g\,x \\ 2\,h_H\,y \;+\; 2\,h_V\,z \\ -2\,h_H\,x \;-\; 2\,h_V\,w \\ 2\,h_H\,w \;-2\,h_V\,x \end{bmatrix}$$

$$\frac{\partial h\left(\boldsymbol{x}\right)}{\partial y} = \begin{bmatrix} 2\,g\,w \\ 2\,g\,z \\ -2\,g\,y \\ 2\,h_H\,x \;+\; 2\,h_V\,w \\ 2\,h_H\,y \;+\; 2\,h_V\,z \\ 2\,h_H\,z \;-2\,h_V\,y \end{bmatrix} \qquad \frac{\partial h\left(\boldsymbol{x}\right)}{\partial z} = \begin{bmatrix} 2\,g\,x \\ 2\,g\,y \\ 2\,g\,z \\ -2\,h_H\,w \;+\; 2\,h_V\,x \\ -2\,h_H\,z \;+\; 2\,h_V\,y \\ 2\,h_H\,y \;+2\,h_V\,z \end{bmatrix} \tag{2.116}$$

## 2.4.4 Orientation Estimation Procedure

Initially ($k = 0$) the state $\boldsymbol{x}$ is set to zero and the covariance P is chosen to be a unitary matrix. These initial values will mostly differ from the actual initial state of the sensor node, but, after a few iterations, the estimations will quickly converge to the actual state.

### 2.4.4.1 Prediction

The first step in the KF is the prediction, which is based upon the process model equation.

**MFG Filter**

Since (2.102) is actually a linear equation, standard discrete KF equations (2.48) and (2.49) can be applied:

$$\hat{\boldsymbol{x}}_k^- \;=\; \hat{\boldsymbol{x}}_{k-1} \;+\; \tau\;(\hat{\boldsymbol{x}}_{k-1} \;-\; \hat{\boldsymbol{x}}_{k-2}) \tag{2.117}$$

$$\mathrm{P}_k^- \;=\; \mathrm{P}_{k-1} \;+\; \mathrm{Q} \tag{2.118}$$

Note that in the above equations $(\hat{\boldsymbol{x}}_{k-1} - \hat{\boldsymbol{x}}_{k-2})$ represents the $\boldsymbol{u}_k$ term in (2.39). For that reason, this term is not reflected in the equation for the a priori estimate error covariance. Any errors in the process model are incorporated in the process noise term $\boldsymbol{w}_k$ and its covariance matrix Q.

### MARG Filter

Both process model equations, (2.105) for Euler angles and (2.106) for quaternions, are non linear. Therefore, linearisation is needed by either calculating the Jacobian matrices and applying EKF procedures or computing sigma points and using SPKF equations.

**EKF** The partial derivative of the state transition equation, (2.113) for a Euler state or (2.114) for a quaternion state, must be evaluated at the previous a posteriori estimate $\hat{\boldsymbol{x}}_{k-1}$ and current input $\boldsymbol{u}_k$ to give the Jacobian matrix $\mathrm{A}_k$ as described by the linearisation procedure of the EKF (2.57):

$$\mathrm{A}_k^{Euler} = \left[ \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial \phi}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \quad \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial \theta}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \quad \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial \psi}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \right] \tag{2.119}$$

or

$$\mathrm{A}_k^{Quaternion} = \left[ \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial w}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \quad \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial x}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \quad \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial y}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \quad \left.\frac{\partial f(\boldsymbol{x},\, \boldsymbol{u})}{\partial z}\right|_{\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k} \right] \tag{2.120}$$

The above expressions can then be used in the standard EKF prediction equations (2.59).

**SPKF** First, sigma points reflecting the current state distribution are determined based on (2.68). Then, the points are transformed using the non linear state update equations (2.113) or (2.114) according to the SPKF procedure (2.68). The prediction and its error covariance can now be determined as its distribution is reflected by the new set of sigma points.

### 2.4.4.2 Correction

The second step in the KF procedure consists of the correction. Both the MARG and MFG filter use the exact same measurement model, so the correction procedure is also equal. As this model is clearly non linear, either EKF or SPKF equations must be used.

### Extended Kalman Filter

The Jacobian state-to-measurement transition matrix $\mathrm{H}_k$ must be determined by evaluating the partial derivatives of the measurement model equation (2.115) for Euler state or (2.116) for quaternion state at the a priori state estimation $\hat{\boldsymbol{x}}_k^-$:

$$\mathrm{H}_k^{Euler} = \left[ \left.\frac{\partial h(x)}{\partial \phi}\right|_{\hat{x}_k^-} \quad \left.\frac{\partial h(x)}{\partial \theta}\right|_{\hat{x}_k^-} \quad \left.\frac{\partial h(x)}{\partial \psi}\right|_{\hat{x}_k^-} \right] \tag{2.121}$$

or

$$\mathrm{H}_k^{Quaternion} = \begin{bmatrix} \left.\frac{\partial h(x)}{\partial w}\right|_{\hat{x}_k^-} & \left.\frac{\partial h(x)}{\partial x}\right|_{\hat{x}_k^-} & \left.\frac{\partial h(x)}{\partial y}\right|_{\hat{x}_k^-} & \left.\frac{\partial h(x)}{\partial z}\right|_{\hat{x}_k^-} \end{bmatrix} \tag{2.122}$$

The correction can then be executed using the EKF equations (2.60).

#### Sigma Point Kalman Filter

The correction procedure starts by computing sigma points reflecting the a priori estimate distribution as defined by (2.79). After transformation by the measurement model equation (2.107) or (2.110), the measurement prediction can be obtained as the weighted mean of the sigma points. The correction is then completed by applying the SPKF equations.

### 2.4.5 Adaptive Filtering

This section only discusses the adaptive filtering technique for the MFG filter, the MARG version of the tracking filter already implements the standard adaptive filtering principles discussed in section 2.3.6.

In the MFG version, all estimations are based on the output of two sensors. When this output is distorted, the estimates of the filter also deteriorate. Aside from circuit noise, the output of the sensors will also be distorted by other effects. The accelerometer measures other accelerations besides gravity caused by movement and shocks while the magnetic field of the earth might be distorted by nearby cell phones or large quantities of metal.

In order to reduce the influence of these errors, a certain measure of the error must first be determined. This measure is found in the norm of both sensor outputs. Since, under normal circumstances, this norm should remain approximately constant, any deviation from the expected value can be associated with an error signal due to disturbances.

The next step is to find a way to reduce the influence a sensor output has on the overall filter estimate. As the measurement covariance matrix R dictates the uncertainty about the measurements, it is the perfect candidate for adaptation. Denoting the variance of the circuit noise on the accelerometer respectively magnetometer output by $r_{acc}^0$ and $r_{mag}^0$, the adaptive formulas are given by:

$$\begin{aligned} r_{acc} &= r^0_{acc}\left(1 + \zeta\left||\boldsymbol{a}_n|^2 - |\boldsymbol{g}|^2\right|\right) \\ &= r^0_{acc}\left(1 + \zeta\left||\boldsymbol{a}_n|^2 - 1\right|\right) \qquad (2.123) \\ r_{mag} &= r^0_{mag}\left(1 + \xi\left||\boldsymbol{m}_n|^2 - |\boldsymbol{h}|^2\right|\right) \\ &= r^0_{mag}\left(1 + \xi\left||\boldsymbol{m}_n|^2 - 1\right|\right) \qquad (2.124) \end{aligned}$$

Both values can then be substituted in the expression for the measurement noise covariance matrix (2.112).

In the above formulas, the measurement noise covariance is increased such that the Kalman filter will rely more on the predicted estimate with its lower process noise covariance and the other non disturbed sensor to determine its a posteriori estimate.

## 2.5 Parameter Estimation

The orientation estimator algorithm uses several parameters to perform its task. In order for the filter to deliver estimates of descent quality, all of these parameters must be assigned values that reflect the process dynamics as adequate as possible. Actual data of motion captures of humans performing several tasks provides vital information for the estimation of the parameter values as it represents the process to be estimated. Therefore, data analysis of captures with an optical tracking system will be used to determine the optimal parameter values.

First, the digital pre-filters used in the sensor output preprocessing are designed. Then Kalman filter parameters need to be chosen. Mainly the noise covariance matrices Q and R form an important aspect, yet, also the orientation rate feedback gain $\tau$ used in the MFG filter needs to be determined.

### 2.5.1 Digital Pre-Filter

The sensor outputs contain an amount of high frequency noise generated within the readout circuits and internal structure of the devices, but may also be induced by outside sources. In order to remove this high frequency component, low pass filters can be used. Given the fact that the sensor outputs are delivered in digital form, these filters will also be digital.

#### 2.5.1.1 Filter Concept

The general form of realistic digital filters is given by [30]:

$$y(n) \;=\; -\sum_{m=1}^{M} a_m\, y(n-m) \;+\; \sum_{k=0}^{K} b_k\, x(n-k), \qquad (2.125)$$

where $x(n)$ and $y(n)$ denote the input and output of the filter and all coefficients are constant and real numbers. Note that any output value $y(n)$ only requires previous values of the output and previous and current values of the input to be calculated, making this filter actually implementable. In the frequency domain, filters are often described by their transfer function allowing a quick look at how the signal will be affected after filtering. For (2.125), the corresponding transfer function is given by:

$$H(z) \;=\; \frac{\sum_{k=0}^{K} b_k\, z^{-k}}{1 \;+\; \sum_{m=1}^{M} a_m\, z^{-m}} \;=\; \frac{B(z)}{A(z)} \qquad (2.126)$$

The frequency response of the filter can be determined by evaluating $H(z)$ on the unit circle in $z = e^{j2\pi f T_s}$, where $T_s = f_s^{-1}$ represents the sample period. This response is a periodic function and is entirely determined by its values in the frequency band $[0, f_s/2]$.

Depending on the values of the coefficients, digital filters can be made to be high pass, low pass, band pass or band stop. The order of the filter, which is determined by the size of $M$ and $K$, influences the steepness of the frequency response. However, a higher order also increases the filter latency, which is inherent to digital filters. It is the time which a signal takes to propagate through a system.

When designing a digital filter, measures are used to describe the filter characteristics in the frequency domain. The *stop band* and *pass band* are the frequency bands where the filter respectively attenuates and transmits the input signal. The *cutoff frequency* is where the pass and stop band meet. The *stop band attenuation* defines the minimal reduction in signal strength required in the pass band. The *pass band ripple* refers to the maximal ripple in the pass band transmission.

Several standard filter forms are readily available and can easily be designed using specialised software. Each of these filter types exhibits different behaviour that should be accounted for when choosing an architecture. Butterworth filters e.g. are designed to have a maximally flat frequency response in the pass band and they roll off slowly and smoothly at the cutoff frequency [31]. Chebychev filters show a ripple in either the pass band (Type I) or the stop band (Type II or inverted Chebychev) and show a steeper roll off. Elliptical filters show a ripple in both the pass and the stop band, yet the transition between both bands is the fastest available. Figure 2.8 displays frequency responses of fifth order low pass filters of each type. The list of existing architectures is endless though, many more can be found in literature [32].

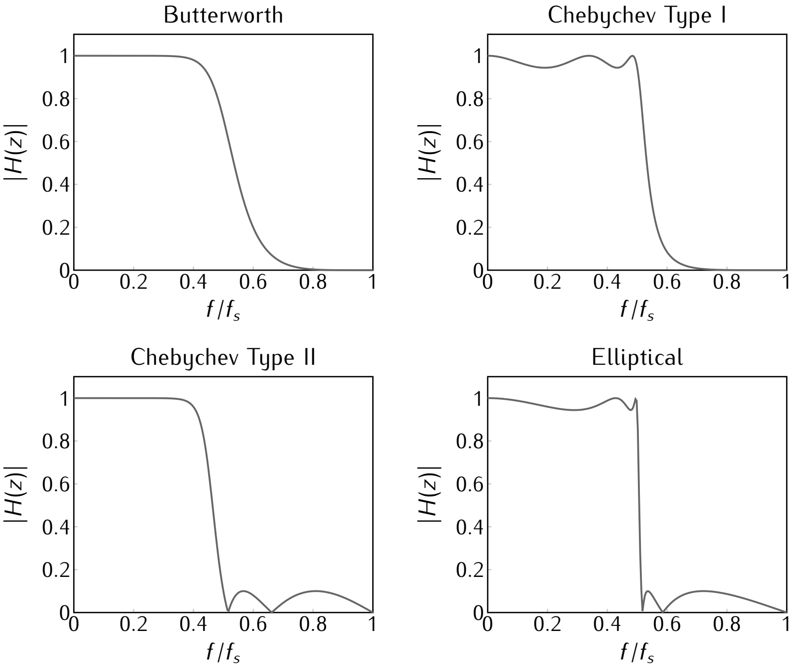


*Figure 2.8: Digital filter architecture comparison by frequency response.*

#### 2.5.1.2 Sensor Output Filter Design

The sensor output signals should be low passed as this part of the frequency band contains useful information on the orientation. A flat response of 0 dB in the pass band is advisable in order to have equal transmission of low frequencies. Furthermore, the latency of the filter should remain limited. Inverse Chebychev filters are able to deliver all of these properties and offer a steeper roll off at a lower order than Butterworth types can. Finally a choice of cut off frequency must still be made based on the knowledge gained from signal analysis.

The power spectrum of a signal gives an indication of which frequency bands contain the most signal power. This spectrum can be calculated from the Fourier transform. Given the discrete nature of the signals, the Fast Fourier Transform will be used:

$$S_{xx}(f) \;=\; |X(k)|^2 \;=\; \left|\sum_{n=0}^{N-1} x(n)\, e^{-\frac{j2\pi nk}{N}}\right|^2, \tag{2.127}$$

where $X(k)$ corresponds to the Fourier transform at frequency $kf_s/N$, $N$ is the length of the data series and $f_s$ corresponds to the sample frequency.

### Accelerometer

Figure 2.9 depicts the power spectrum of accelerometer output noise. The spectrum reaches until 50 Hz as the accelerometer utilises a 100 Hz sampling frequency. It is clear that much of the noise is concentrated in the frequency band from 20 to 30 Hz. Furthermore, human motion is limited to very low frequencies around a couple of Hertz. The cut off frequency could thus be chosen around e.g. 5 Hz, with the stop band starting at least at 15 Hz.

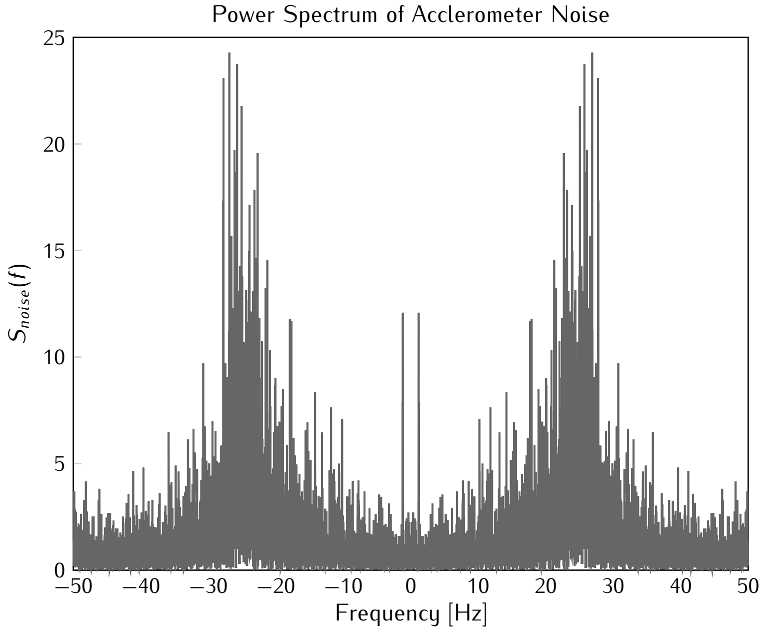


*Figure 2.9: Power spectrum of accelerometer output noise.*

A filter with the given characteristics can easily be designed using the *Filter Design and Analysis Tool* of the *Signal Processing* toolbox in Matlab [33]. Figure 2.10 shows the frequency response of the designed filter for a 100 Hz digital signal. The corresponding transfer function is given by:

$$H(z) = \frac{0.059 - 0.018\,z^{-1} - 0.018\,z^{-2} + 0.059\,z^{-3}}{1 - 2.049\,z^{-1} + 1.507\,z^{-2} - 0.377\,z^{-3}} \qquad (2.128)$$

It is a third order inverted Chebychev with a stop band starting at 12 Hz and an attenuation of more then 20 dB. The pass band stretches out to 6 Hz where the magnitude of the frequency response drops to −1 dB.

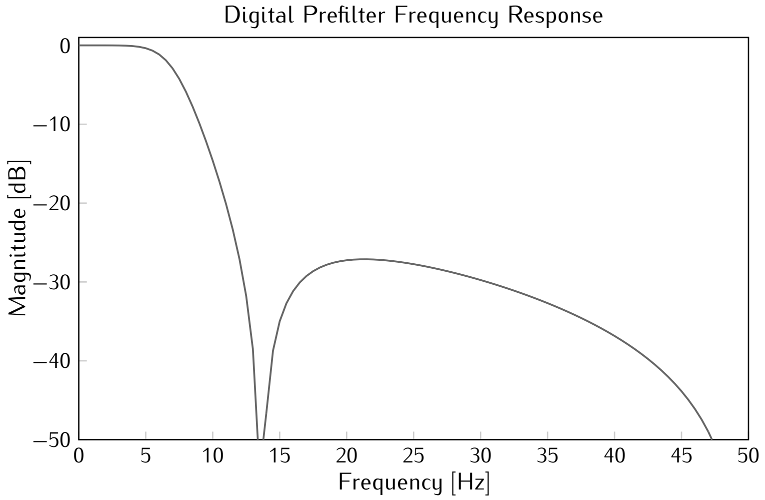


*Figure 2.10: Frequency response of the accelerometer pre-filter.*

### Magnetometer

Figure 2.11 shows the power spectrum of noise on the magnetometer output. The signal was also sampled at 100 Hz. In contrast to the case of accelerometer noise, the spectrum is much flatter and evenly spread. However, since the same signal is meant to be measured, the same low pass filter can be used.

### Gyroscope

The power spectrum of gyroscope output noise resembles the spectrum of magnetometer output noise. The same filter can thus be reused for gyroscope output pre-filtering.

## 2.5.2 Kalman Filter Parameters

Two important parameters that define the Kalman filter estimation procedure are the process and measurement noise covariance matrices Q and R. The measurement noise covariance is easily obtained as the sensors themselves are readily available for testing. The process noise covariance however requires data from the process that will be estimated in order to find a descent estimate.

Aside from these two covariances, three other parameters still need to be determined: the feedback gain $\tau$ present in the MFG process model and the adaptation parameters $\zeta$ and $\xi$. The first will again require analysis of data sequences of actual motion as it involves the process model. The other two will determine the suppression of disturbances on the sensor outputs and will

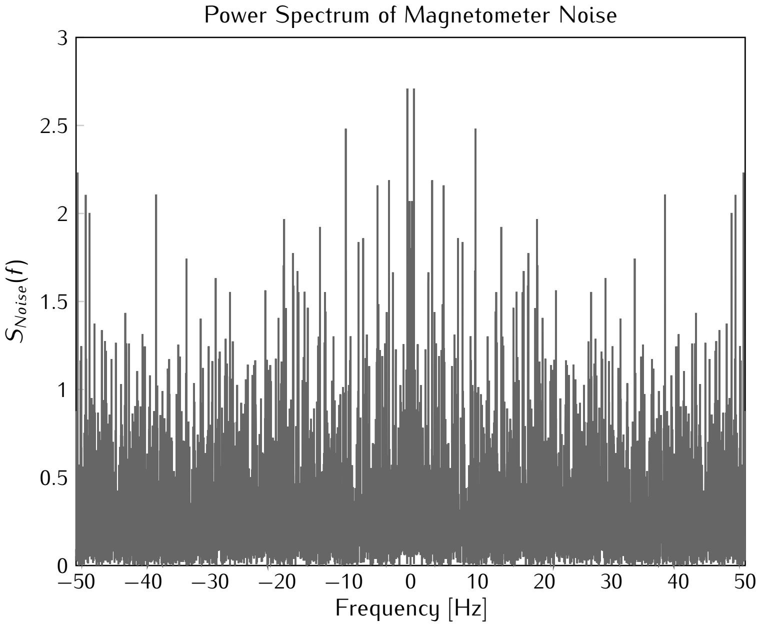


*Figure 2.11: Power spectrum of magnetometer output noise.*

be estimated in simulation.

#### 2.5.2.1 Measurement Noise Covariance

Determining the measurement noise covariance is in most cases fairly straightforward since it corresponds to the noise each of the sensor outputs exhibit. This noise can easily be obtained by placing the sensors in an environment where it is expected that their output is constant. In the case of inertial sensors this naturally means no movement is involved. Any deviation from the mean can then be designated as noise and the variance is quickly obtained.

**Accelerometer Noise**

An output sequence measured on the X-axis of an accelerometer placed on a table in an office environment is displayed in figure 2.12. Note that the raw output has first been subjected to calibration and pre-filtering before being saved, as these steps also precede the Kalman filter. Care has been taken that the digital pre-filter has reached steady state conditions, so that the transient cannot influence the covariance.

Similar sequences were captured from various sensors on each of their axes. The mean variance measured across 15 different sensors equals $2 \times 10^{-5}$.

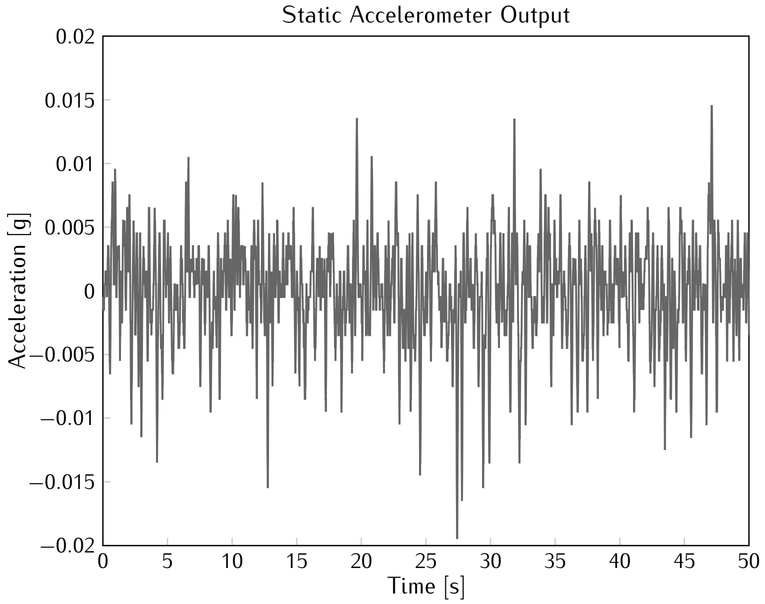


*Figure 2.12: Static accelerometer output capture of* 50 s.

### Magnetometer Noise

Figure 2.13 displays the output sequence measured on the X-axis of the magnetometer on the same sensor node during the same time period as the sequence of the graph in figure 2.12. The magnetic field is presented in arbitrary units as the output is normalised to the earths magnetic field strength during calibration.

Again, the covariance was measured on 15 different nodes, which resulted in an average of approximately $1 \times 10^{-5}$.

### Covariance Matrix

In section 2.4.3.3, certain assumptions were made involving the measurement noise covariance matrix. It was assumed that measurement data was uncorrelated between both sensors and that no cross covariance existed between different sensitivity axes. Measurements revealed that sensor cross covariance values are 100 times smaller than the variances stated in the previous paragraphs and that cross axis covariance is more than 10 times smaller. It is safe to say that these values are negligible compared to the diagonal elements of the covariance matrix.

Substituting the values determined in the previous paragraphs in the de-

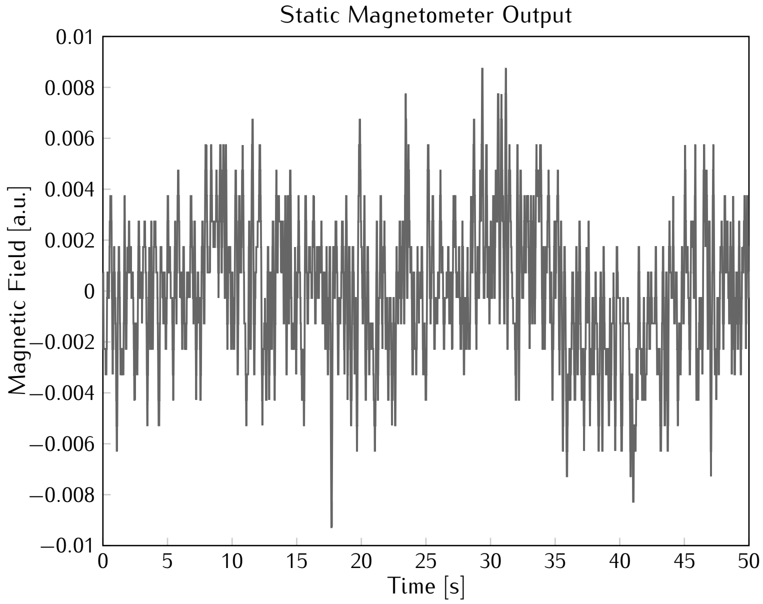


*Figure 2.13: Static magnetometer output capture of* 50 s.

rived form of the measurement noise covariance matrix (2.112) finally results in:

$$R = \begin{bmatrix} 2\times10^{-5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 2\times10^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 2\times10^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1\times10^{-5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1\times10^{-5} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1\times10^{-5} \end{bmatrix}. \tag{2.129}$$

### 2.5.2.2 Process Noise Covariance

The process noise covariance matrix can only be determined when data is available that is representative for the process that will be estimated. When this is the case, the process model can be applied to the data sequence and the covariance of the error can be determined. This procedure can easily be applied to the optical motion captures for the MFG filter, yet for the MARG filter, the gyroscope output noise will dictate the process noise. In general however, the process noise covariance should be lower in the MARG case, as the gyroscope will provide much more qualitative information than the discrete derivation of the orientation estimate will ever give.

### MFG

The process model equation (2.102) clearly depends on the choice of the feedback gain parameter $\tau$. A logic consequence is that the covariance matrix of the noise will also depend on this parameter. Furthermore, the representation used for the state also effects the covariance, Euler angles are allowed to change over a larger interval than quaternion elements.

**Euler** Figure 2.14 displays two graphs of the prediction error variance for different values of the feedback gain parameter $\tau$. The dark grey graph was obtained by applying the process model equation (2.102) to the available motion captured sequence of a walking and running person, while the light grey graph used a capture of various actions ranging from simple arm lifting tasks to jumping around. The plotted variance is an average over all limbs and over all three Euler angles.

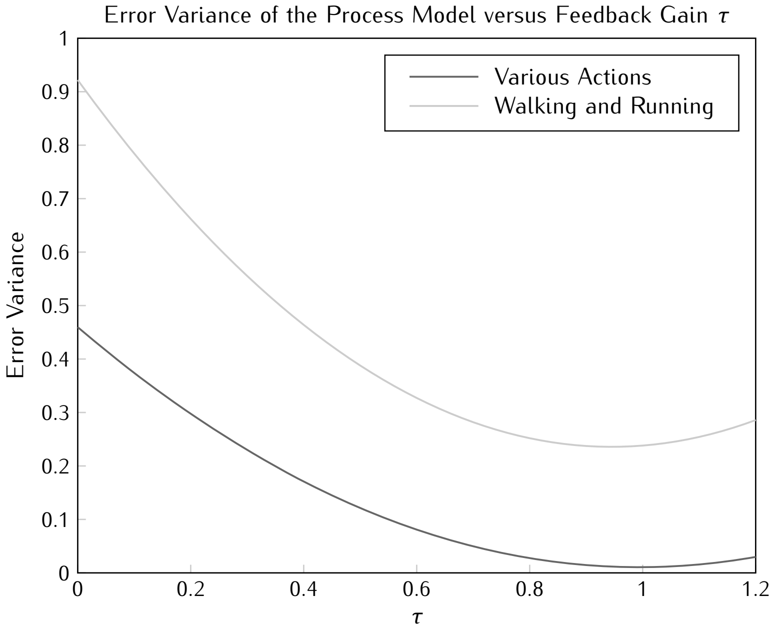


*Figure 2.14: Variance of the a priori estimate error versus the feedback gain parameter $\tau$ in case of a Euler angle state.*

The graphs clearly exhibit a minimum at high $\tau$ values where the variance reaches 0.23 for walking and running and as low as 0.01 for random tasks, while at the origin the variance still amounts to much higher values. This clearly indicates that their is a strong correlation between the current and the previous orientation. Note that the graphs also indicate that the variance highly depends on the performed tasks.

The proposed process noise covariance matrix of section 2.4.3.2 assumed that each Euler angle has a similar distribution and that no cross covariance exists. The cross covariance turns out to be at least ten times smaller than the variances plotted in the graphs, which makes this assumption rather realistic. However, differences between the variances on the individual axes do exist. Figure 2.15 displays the variance on each of the axes for the data set with random tasks, which would approximate real life situations as best as possible. It is clear that the variance on the Y-angle is much lower and on the Z-axis is much higher. For the walking and running sequence, the X-angle variance dominated the others as these movements provoke large rotations around the X-axis.

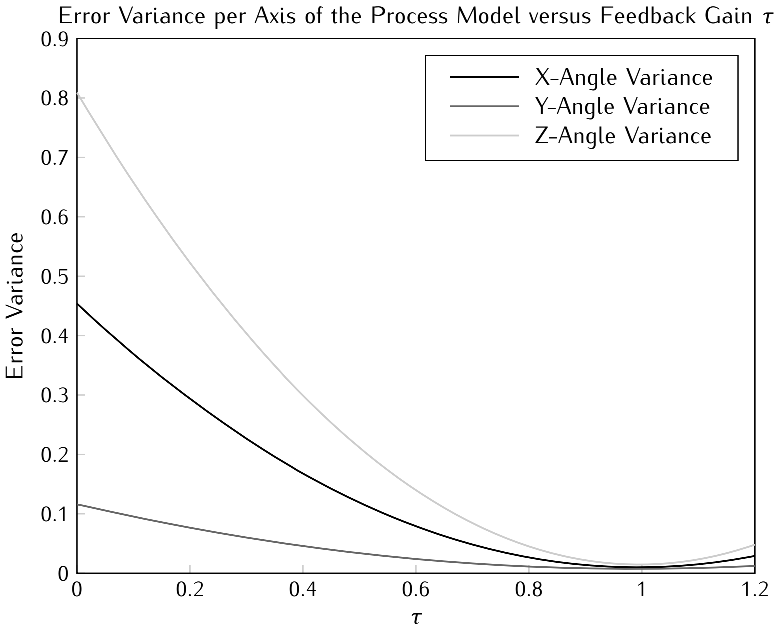


*Figure 2.15: Variance on each of the individual axes of the process model error versus the feedback gain parameter $\tau$ in case of a Euler angle state.*

**Quaternion** Applying the same procedure to the quaternion version of the data sequence results in the graph displayed in figure 2.16. Again, a clear minimum is present at approximately the same location as was the case for a Euler state. The error variance reaches 3.8 × $10^{-6}$ for the walking and running and 9.5 × $10^{-7}$ for various random tasks. The cross covariance is again negligible compared to the diagonal elements of the covariance matrix and, contrary to the Euler case, the variance is very much alike for each of the state elements. This means that the postulated process error covariance matrix form of equation (2.103) is valid for the quaternion case.

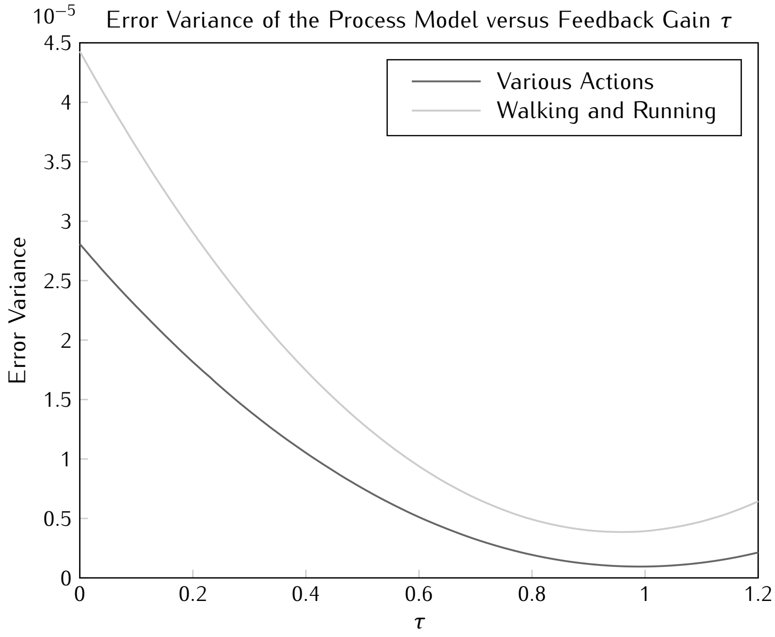


*Figure 2.16: Variance of the a priori estimate error versus the feedback gain parameter $\tau$ in case of a quaternion state.*

### MARG

The process noise covariance matrix in the MARG filter is a direct result of the gyroscope output noise. Therefore, the covariance matrix of the gyroscope output noise must first be determined. A data sequence captured from the X-output of a gyroscope in static conditions is displayed in figure 2.17. The mean variance measured across several sensors is 1 × $10^{-2}$. As was the case with the other sensors, all axes demonstrate similar noise variance and cross covariance is negligible.

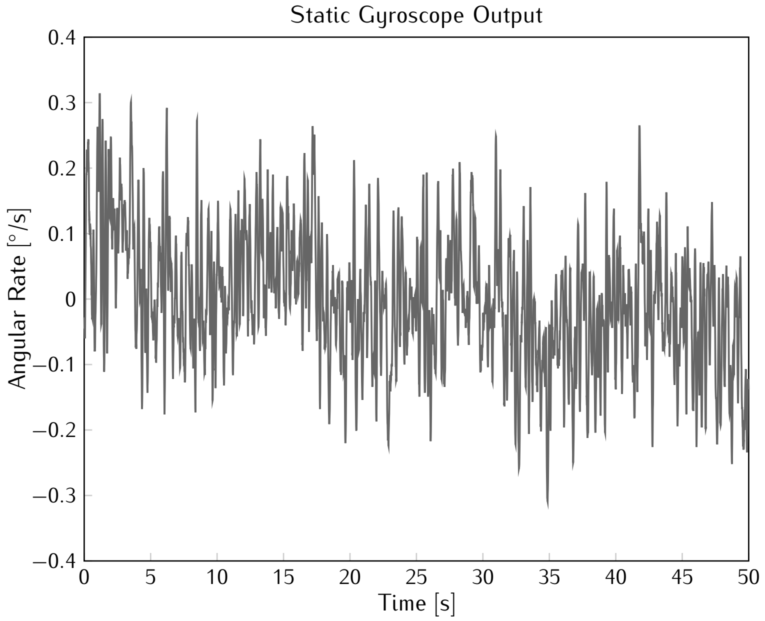


*Figure 2.17: Static gyroscope output capture of* 50 s.

**Euler** In order to find the relation between the gyroscope and process noise covariance, equation (2.11) must be used to convert the body rates to Euler rates. This conversion is linear, at least from the point of view of the body rates. The covariance of the process noise now still depends on the orientation, yet each Euler angle can be swept across its possible values and the resulting covariance can be calculated. Assuming each possible combination of angles has an equal possibility of occurring, the process noise covariance is found as the average of all of the resulting matrices. Applying the described method results in the following covariance matrix:

$$Q = \Delta T^2 \begin{bmatrix} 0.6 & 0 & 0.0001 \\ 0 & 0.01 & 0.0001 \\ 0.0001 & 0.0001 & 0.6 \end{bmatrix} \qquad (2.130)$$

Cross covariance terms are yet again negligible. Note however that due to the singular orientations of the gimbal lock, $\theta$ has not been averaged over its entire range as this results in an infinite variance. This immediately explains the higher values on the X- and Z-angle.

**Quaternion** Applying the same method as described above, yet using equation (2.30) as transformation equation, the process noise covariance matrix equals:

$$Q \;=\; \Delta T^2 \begin{bmatrix} 1.9 & 0 & 0 & 0 \\ 0 & 1.9 & 0 & 0 \\ 0 & 0 & 1.9 & 0 \\ 0 & 0 & 0 & 1.9 \end{bmatrix} \times 10^{-3} \qquad (2.131)$$

This result is obviously a reflection of the fact that the transformation in the quaternion case doesn't include any trigonometric functions or singularities. Cross axis terms are very close to zero and diagonal terms are all equal.

#### 2.5.2.3 Feedback Gain

As the feedback gain is used to describe the process in the MFG filter, data captures will again provide very useful information. The process model equation (2.102) attempts to estimate a future value based on previous outputs. In signal processing, this technique is often referred to as autoregressive modeling [30], where a signal value $s(n)$ is approximated by a linear combination of $K$ previous values:

$$s_p(n) \;=\; \sum_{k=1}^{K} a_k\, s(n-k) \qquad (2.132)$$

Denoting $\boldsymbol{d}$ as the difference between two subsequent estimates $\hat{\boldsymbol{x}}$, the prediction equation (2.117) can be rewritten in the form of the autoregressive model with $K = 1$:

$$\boldsymbol{d}_p(k) \;=\; \tau\, \boldsymbol{d}(k-1) \qquad (2.133)$$

According to autoregressive modeling, $\tau$ should optimally be chosen as follows:

$$\tau_{opt} \;=\; \frac{R_{dd}(1)}{R_{dd}(0)}, \qquad (2.134)$$

where $R_{dd}$ represents the autocorrelation function:

$$R_{dd}(i) \;=\; \sum_{k} d(k)\, d(k-i) \qquad (2.135)$$

This procedure can easily be applied to the available captures describing human motion.

### Euler

Calculation of the mean over all available datasets yields an optimal value for the feedback parameter of 0.91. However, this value greatly depends on the executed movement and on the body part which is observed. Figure 2.18 displays the results of the autoregressive analysis on three captures of walking and running sequences. Each sequence consists of nine different orientations corresponding to several body parts.

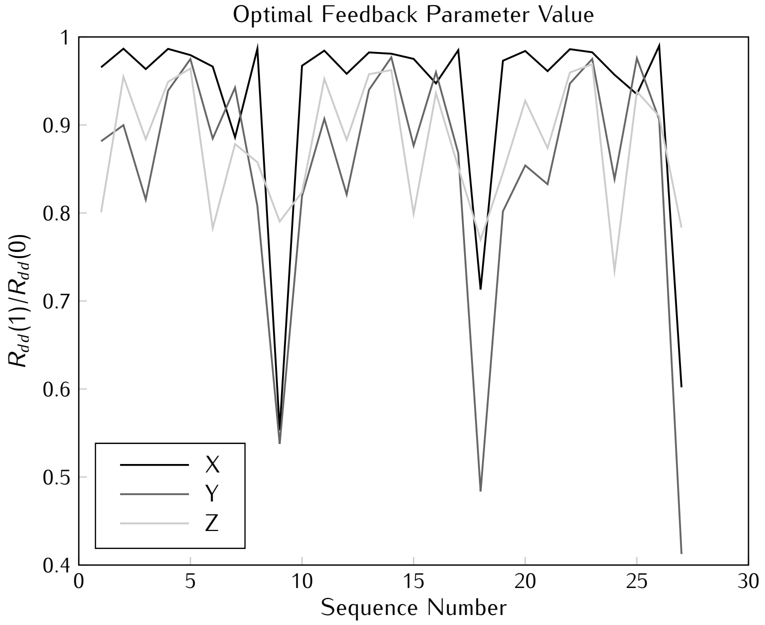


*Figure 2.18: Optimal feedback parameter estimate per Euler axis calculated on the walking and running sequence.*

All three Euler angle series clearly follow a similar pattern, highs and lows clearly match. Furthermore, body parts give similar results for the feedback value across the three captures. The torso e.g. , present at sequence numbers 9, 18 and 27, exhibits very low correlation.

### Quaternion

Autoregressive analysis on the quaternion version of the movement captures indicate an optimal value of 0.89. The same trends that were found in the Euler version also apply here. The fact that the correlation turns out to be slightly lower is related to the fact that the quaternion must be normalised in order to represent an orientation.

## 2.6 Filter Simulation

As the focus of this work lies on gyroless orientation tracking, only simulations of the MFG filters will be presented in this section. All of the simulations will assume that the sensors are sampled at a rate of 100 Hz and, as a result, the output will have the same sample frequency. Three types of simulations will be discussed: step response, noise response and the influence of motion disturbance.

### 2.6.1 Step Response

The step response describes the output of a filter when a sudden change is applied at the input. In this case, a sudden change in sensor output values corresponding to a rotation will be applied. The filter should respond to this and the output will evolve towards the new orientation.

Two different orientation steps will be simulated: a 90° tilt rotation around the X-axis and a 90° heading rotation. In both situations, the starting point is the neutral position where all of the Euler angles equal zero or where the orientation quaternion has zero on its vector part and one on the scalar part. For the tilt rotation, the end point corresponds to $[\phi, \theta, \psi] = [90, 0, 0]$ for Euler angles and $q = [\sqrt{2}/2, \sqrt{2}/2, 0, 0]$ for the quaternion filter. The heading step will end with $[\phi, \theta, \psi] = [0, 0, 90]$ or $q_x = [\sqrt{2}/2, 0, 0, \sqrt{2}/2]$. The reason that both are simulated lies in the fact that the gravity vector coincides with the Z-axis. A heading rotation will thus form a bigger challenge for the filter as the accelerometer will not provide any information about the change in orientation.

All of the simulations in this section will receive noiseless sensor data to calculate their output from. The influence of noise on the sensor outputs is analysed in the subsequent section.

#### 2.6.1.1 Tilt Step

Both the Euler and the quaternion version of the filter are simulated with sensor output sequences corresponding to an instant rotation of 90° around the X-axis. At first, the feedback value $\tau$ is set to zero. Later, the influence of an increasing feedback value is shown.

#### Euler

Figure 2.19 shows the response of the tracking filters to a tilt step. Note that the outputs of the UKF and CDKF almost match exactly, therefore only one curve is given for the SPKFs. It is clear that both filter types manage to track the step response, although the EKF takes a lot more time to achieve its steady state value. The transient period lasts up to almost 3 s. Figure 2.20 shows more detail of the transient on the SPKF output signal. The input signal has also been plotted as a reference. Clearly, the transition here is extremely fast and takes only 3 samples or 30 ms to complete. This is one hundredth of the transition time seen in the EKF response.

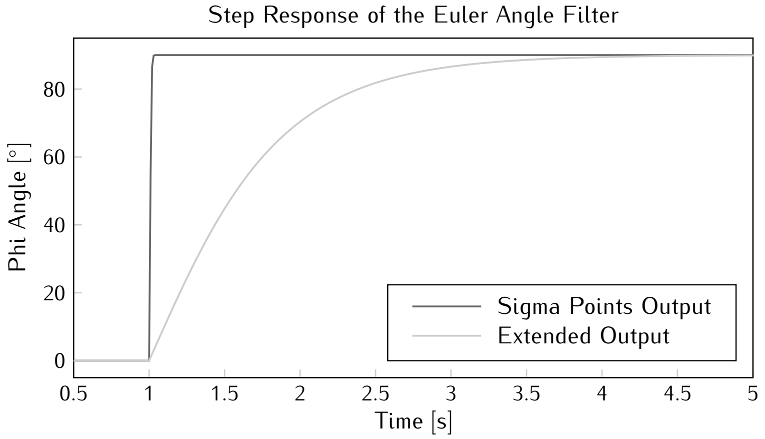


*Figure 2.19: Response of the Euler angle state filters to a sudden* 90° *rotation around the X-axis.*

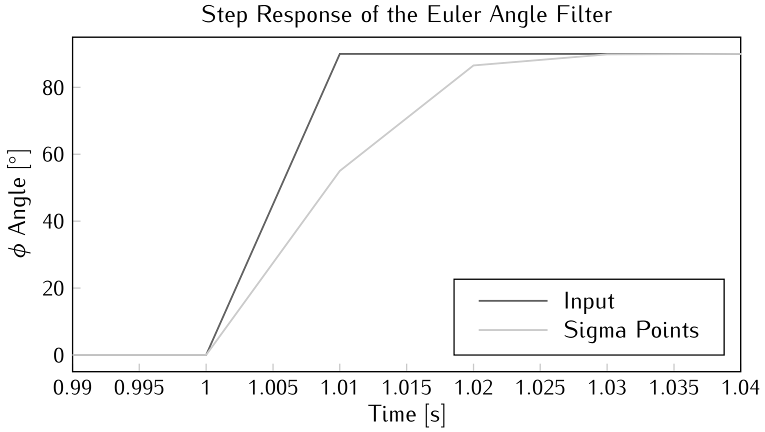


*Figure 2.20: Zoom of the step input and sigma point filter response.*

### Quaternion

The tilt responses of the quaternion version of the filters is shown in figure 2.21. In this case, all three filters show a similar transient time. The EKF takes 20 ms, while the SPKFs take 30 ms. However, it is important to mention that both sigma point filters suffer from a slight overshoot, before settling to the final value.

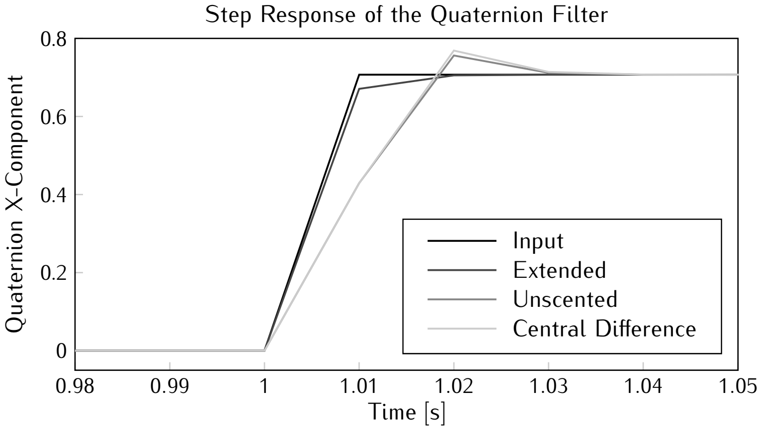


*Figure 2.21: Response of the quaternion state filters to a sudden* 90° *rotation around the X-axis.*

### Feedback

Figure 2.22 shows several graphs of the tilt response of the EKF with Euler state at different values of the feedback parameter $\tau$. With increasing $\tau$, the settling time of the filter output clearly decreases. However, at a certain point, overshoot occurs and ringing is present when $\tau$ is increased even further. The settling time as a function of $\tau$ is plotted in figure 2.23. The optimal value for the feedback parameter from this graph lies around 0.8, which is slightly lower than the value determined in section 2.5.2.3.

As the settling time on the other filter outputs is already very short, no improvement was found by increasing the feedback value. Only very small differences were found and no ringing occurred.

### 2.6.1.2 Heading Step

The filters are presented with a sudden rotation of 90° around the Z-axis. A slower response to this change is expected as the estimation will now only be based upon the magnetometer output.

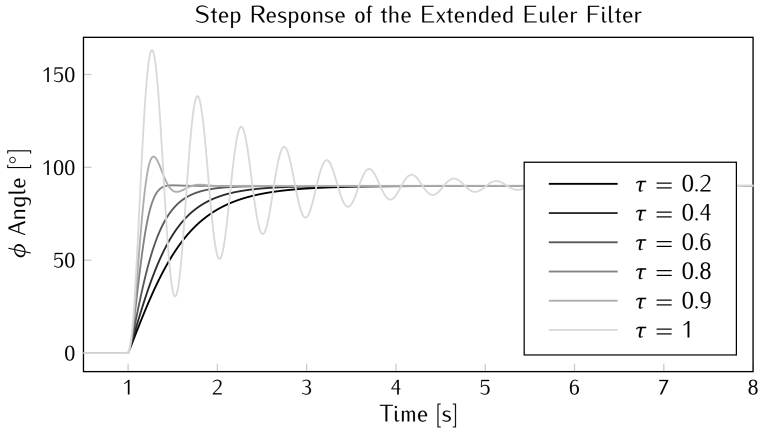


*Figure 2.22: Step response of the extended Euler type filter with increasing value of feedback parameter $\tau$.*

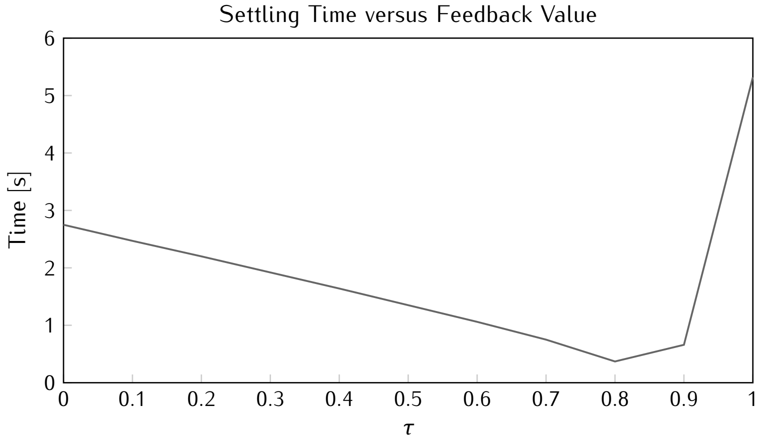


*Figure 2.23: Settling time of the tilt response of the EKF with Euler state to* 1 % *of the final value for varrying value of the feedback parameter $\tau$.*

### Euler

The response of the EKF to a heading step almost equals the response to a tilt step. However, the SPKF response shows increased settling time as can be seen in the graphs displayed in figure 2.24. The settling time for heading steps is approximately 60 ms, twice the settling time simulated for a tilt step.

More important to mention though, is the fact that during the transition on the heading angle $\psi$, both tilt angles $\phi$ and $\theta$ show transient responses with peaks up to 14°. The cause of these transients is found in the high sensibility of the filter and the fact that the heading rotation takes place around the gravity vector. The filter immediately responds to the change in the sensor values and attempts to find the matching orientation. Due to the fact that the accelerometer output has not changed, no information is gained from this sensor and the corresponding part of the innovation will also equal zero. Later on, when wrong values of the tilt angles $\phi$ and $\theta$ propagate through the filter, the accelerometer output will contribute and help correcting these errors.

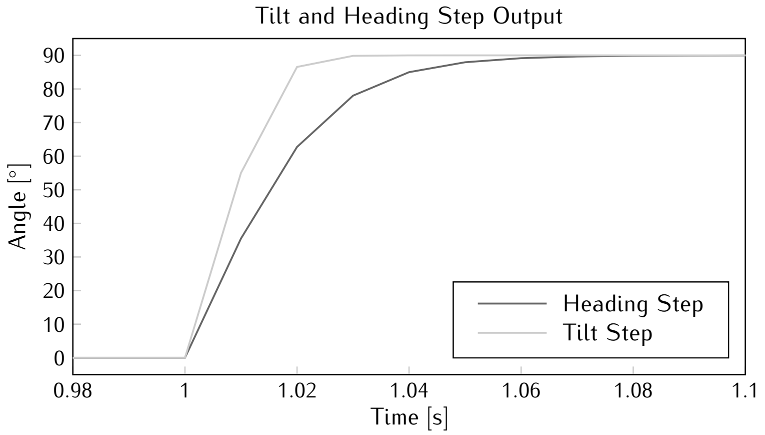


*Figure 2.24: Tilt and heading step response of the Euler SPKFs.*

### Quaternion

The response of the quaternion filters to a heading step is displayed in figure 2.25. For the EKF, the settling time increases to 70 ms compared to the 20 ms settling time for a tilt step, while for the UKF overshoot occurs without any increase in settling time. As was the case for the Euler SPKFs, transients are visible on the other quaternion components reaching almost 20 % of the step size.

## 2.6.2 Noise Response

The noise response is the output of the filter to noisy sensor data corresponding to a fixed orientation. The neutral orientation is chosen as this is also the starting value for the filter. This way, no transient will be present at the start of the output sequence.

Three different noise sequences will be considered: simulated noise created by a random generator, real noise taken from the output of stationary

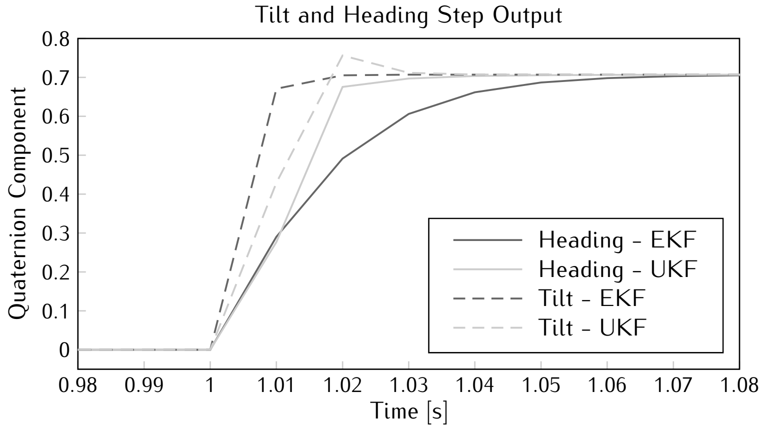


*Figure 2.25: Tilt and heading step response of the quaternion EKF and UKF.*

sensors and real noise which has been filtered by a digital pre-filter.

#### 2.6.2.1 Simulated Noise

Gaussian white noise signals are generated with variances equal to the ones measured on the sensor output signals and given in section 2.5.2.1. These sequences are then added to static sensor signals corresponding to neutral orientation and the output of the filter is analysed.

##### Euler

Figure 2.26 displays graphs of the variance on each axis, for each filter type versus several values of the feedback parameter. As expected by the graphs for the step response, the EKF has much lower variances due to slower response and longer settling. The variance on this filter also increases strong with increasing feedback value, as can be expected from the results in the previous section. On both filters, the $\psi$ variance is ten times higher than the variance on the tilt angles. This is again a confirmation to the fact that heading is harder to track due to the coincidence of the Z-axis with the gravity vector.

##### Quaternion

The variance on the output of the quaternion state filters also confirms the results from the previous section. Both the EKF and the SPKFs show similar variance on their noise response. The main difference lies in the fact that the variance on the scalar component of the quaternion is much lower than the variance on the vector components. This is a direct result of the normalisation

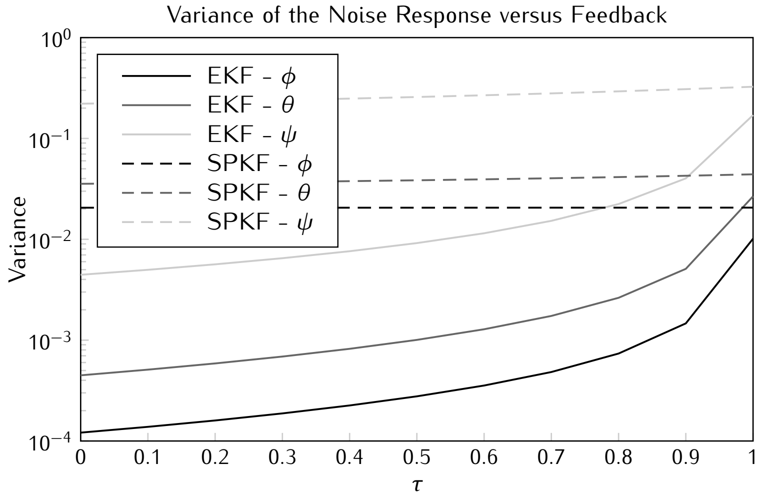


*Figure 2.26: Variance of the Euler filter response to simulated noise for different values of the feedback parameter.*

that takes place every cycle and the fact that the scalar component must equal 1 in the neutral position, while the vector part remains zero. Within the vector component, the same relations are seen between components, the variance on the heading component is ten times the variance on the tilt components.

#### 2.6.2.2 Real Noise

Real noise samples are taken from the sensor outputs and added to the simulated neutral position outputs. Once again, the output of the filter is analysed and compared to the simulated noise case. Furthermore, the noise response of the filter will also be determined when the sensor data is first filtered using the digital filter designed in section 2.5.1.2.

**Euler**

Figure 2.27 displays the variance on the noise response of the EKF response to simulated and real noise. Both graphs exhibit the same behaviour with increasing feedback. The real noise response variance lies approximately ten times higher than the simulated noise response. The reason for this difference lies in the distribution of the real sensor noise. Contrast to the simulated noise, the real noise does not exactly follow a normal distribution. This is especially true for the accelerometer noise as follows from its power spectrum displayed in figure 2.9.

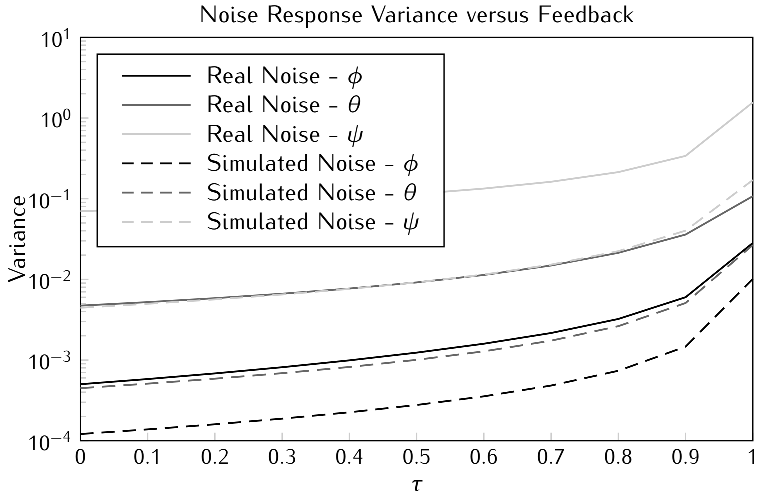


*Figure 2.27: Variance of the Euler EKF response to simulated and real noise for different values of the feedback parameter.*

The SPKFs show a similar increase in variance as can be deducted from table 2.1. Since the feedback does not greatly affect these filters, only the values for zero feedback are given.

| | $\phi$ | $\theta$ | $\psi$ |
|---|---|---|---|
| Simulated Noise | 0.0206 | 0.0355 | 0.221 |
| Real Noise | 0.162 | 0.418 | 2.034 |

*Table 2.1: Variance on the noise response of the Euler SPKFs with zero feedback value.*

### Quaternion

Table 2.2 lists the variances on the noise response of the filters when zero feedback is applied. As with the Euler case, the real noise introduces variances that are approximately ten times higher than with simulated noise. The only exception is the scalar component, where the increase is even higher. Yet again, the normalisation is to blame for this.

### Pre-Filter

Tables 2.3 and 2.4 list the variance on the filtered noise response of all of the filters when zero feedback is applied. It is clear that aside from the Euler EKF, all filters exhibit improved responses with much lower variances. In the

| Filter | Component | Simulated Noise | Real Noise |
|---|---|---|---|
| EKF | w | $1.29 \times 10^{-10}$ | $4.47 \times 10^{-8}$ |
| | x | $1.53 \times 10^{-6}$ | $1.22 \times 10^{-5}$ |
| | y | $2.55 \times 10^{-6}$ | $2.96 \times 10^{-5}$ |
| | z | $1.52 \times 10^{-5}$ | $1.39 \times 10^{-4}$ |
| SPKF | w | $1.29 \times 10^{-10}$ | $4.99 \times 10^{-7}$ |
| | x | $1.61 \times 10^{-6}$ | $2.09 \times 10^{-5}$ |
| | y | $2.61 \times 10^{-6}$ | $3.57 \times 10^{-5}$ |
| | z | $1.52 \times 10^{-5}$ | $1.75 \times 10^{-4}$ |

*Table 2.2: Variance on the noise response of the quaternion filters with zero feedback value.*

Euler EKF case, no improvement is found as this filter already reacts to slow to notice any difference. These results indicate that the digital pre-filters should definitely be applied to the sensor output in order to improve the noise response.

| Filter | Component | Real Noise | Filtered Real Noise |
|---|---|---|---|
| EKF | $\phi$ | $5.02 \times 10^{-4}$ | $4.70 \times 10^{-4}$ |
| | $\theta$ | $4.73 \times 10^{-3}$ | $4.59 \times 10^{-3}$ |
| | $\psi$ | $6.97 \times 10^{-2}$ | $6.83 \times 10^{-2}$ |
| SPKF | $\phi$ | 0.162 | 0.0145 |
| | $\theta$ | 0.418 | 0.0533 |
| | $\psi$ | 2.034 | 0.496 |

*Table 2.3: Variance on the real and filtered noise response of the Euler filters with zero feedback value.*

| Filter | Component | Real Noise | Filtered Real Noise |
|---|---|---|---|
| EKF | w | $4.47 \times 10^{-8}$ | $8.59 \times 10^{-10}$ |
| | x | $1.22 \times 10^{-5}$ | $1.10 \times 10^{-6}$ |
| | y | $2.96 \times 10^{-5}$ | $3.99 \times 10^{-6}$ |
| | z | $1.39 \times 10^{-4}$ | $3.71 \times 10^{-5}$ |
| SPKF | w | $4.99 \times 10^{-7}$ | $8.54 \times 10^{-10}$ |
| | x | $2.09 \times 10^{-5}$ | $1.36 \times 10^{-6}$ |
| | y | $3.57 \times 10^{-5}$ | $3.94 \times 10^{-6}$ |
| | z | $1.75 \times 10^{-4}$ | $3.72 \times 10^{-5}$ |

*Table 2.4: Variance on the real and filtered noise response of the quaternion filters with zero feedback value.*

### 2.6.3 Motion Disturbance

Simulation of motion disturbance requires the addition of error signals that correspond to motion on the output of the accelerometer. More specifically, the influence on the output orientation is analysed when a certain sinusoidal disturbance is present on one of the sensor axes of the accelerometer. Note that a similar analysis could also be executed for disturbance on the magnetometer signal, though the conclusions should generally correspond.

The simulated accelerometer signal that is fed to the filter is shown in figure 2.28. It corresponds to static, neutral orientation and includes a part where the X-axis output is disturbed by a sinusoidal signal with an amplitude equal to 1 g and a frequency of 1 Hz.

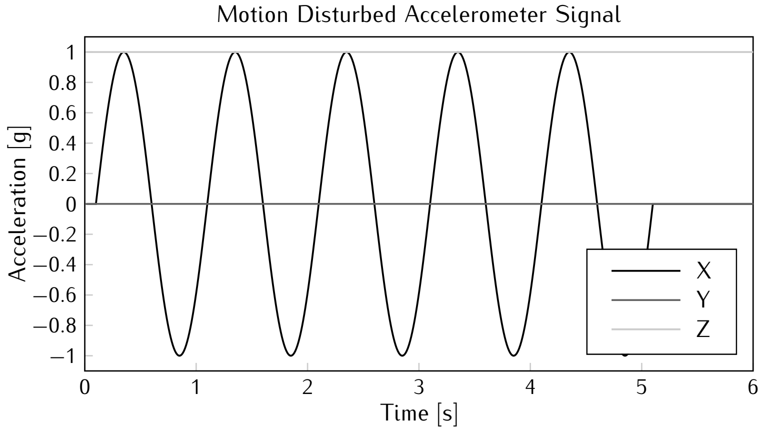


*Figure 2.28: Motion disturbed accelerometer signal.*

The output of the filter is analysed by determining the mean square error of the output signal with respect to the expected output of static, neutral orientation. When no feedback is applied ($\tau = 0$) and no noise adaptation is present ($\zeta = 0$), the output of the Euler UKF follows the graphs in figure 2.29.

The mean square error amounts to 4.5°, 16.5° and 35° on roll, pitch and yaw angles respectively. In the following, the influence of the adaptive filtering and feedback parameter is varied in an attempt to reduce this error.

Note that by placing the disturbance on the X-axis, the worst possible distortion is seen on the output angles. Figures 2.30 and 2.31 show the output orientation of the filter for motion disturbance on respectively the Y- and Z-axis. Remark that the $\theta$ graph lies underneath the $\psi$ graph in both figures. Given the fact that gravity coincides with the Z-axis, any disturbance in this direction does not really influence the output of the filter. The reason

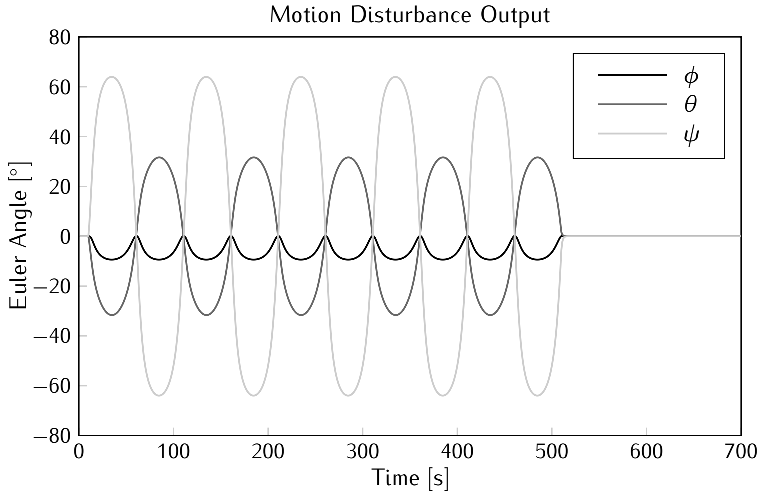


*Figure 2.29: Distorted output signal on the Euler UKF due to motion disturbance.*

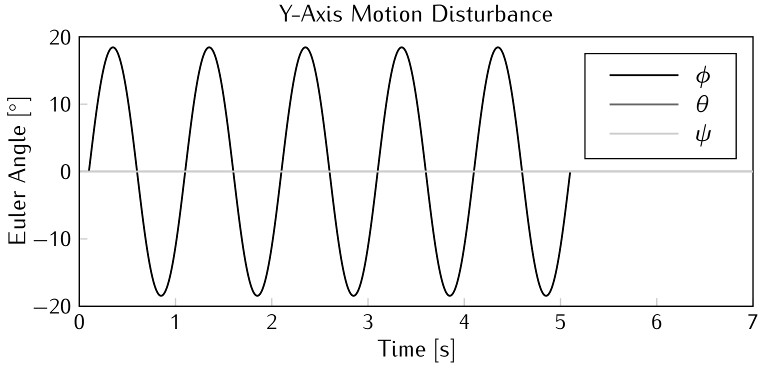


*Figure 2.30: Distorted output signal on the Euler UKF due to motion disturbance on the Y-axis.*

that a disturbance on the Y-axis results in less distortion lies in the choice of aligning the North direction with the Y-axis. Rotating the sensor node around the X-axis will rearrange the magnitude of the magnetic field on both Y and Z-axis, giving the filter a reasonable error on both sensor readings. Yet a rotation around the Y-axis introduces an expected magnetic measurement on the X-axis which is not reflected in the sensor readings. This triggers the filter to drastically change the orientation and introduces large errors.

For the quaternion filters, similar graphs are obtained. Figure 2.32 shows the EKF output due to motion disturbance, which greatly resembles the graphs

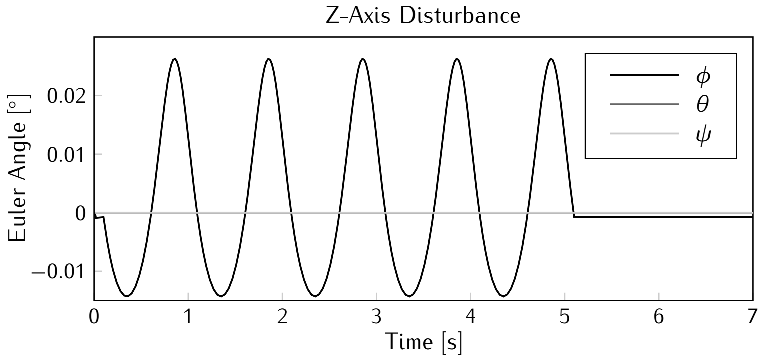


*Figure 2.31: Distorted output signal on the Euler UKF due to motion disturbance on the Z-axis.*

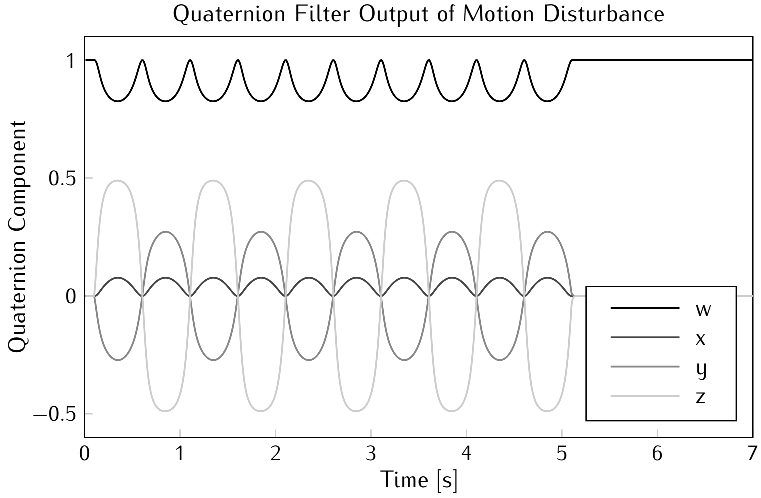


*Figure 2.32: Distorted output signal on the quaternion EKF due to motion disturbance.*

in figure 2.29. When the disturbance is generated on the Z-axis however, no distortion is seen on the quaternion output.

#### 2.6.3.1 Adaptive Filtering

The adaptive filtering parameter $\zeta$ was introduced in order to correct distortion caused by motion disturbance. It is thus expected that increasing this parameter will reduce the error in the output signal.

### Euler

**Extended** Figure 2.33 shows the evolution of the mean square error on the heading angle when the adaptive parameter $\zeta$ is increased. The heading was chosen as this angle displays the highest distortion. Several graphs are given for different frequencies of the disturbance signal. Note that with increasing frequency, the error reduces significantly due to the slow response of the Euler type EKF. Furthermore, higher frequencies can be filtered with an additional digital output filter. Also note that the sensor signal has not been preprocessed, which means that the mean square output error in the figure corresponds to the worst case scenario.

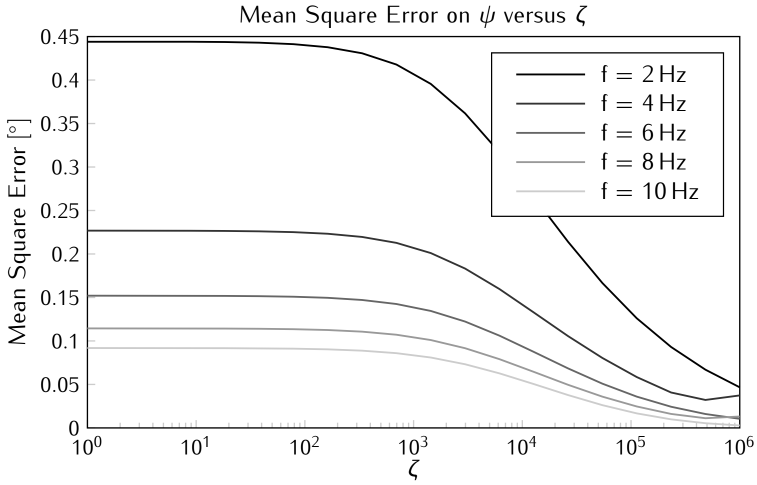


*Figure 2.33: Mean square error on the heading angle output of the EKF for increasing values of the adaptive parameter and different motion disturbance frequencies.*

The graph clearly indicates that below $\zeta = 1000$, no significant reduction is seen on the output error. From that point on however, improved filter performance is visible. At $\zeta = 1 \times 10^6$, saturation seems to occur and higher values are no longer offering any better behaviour. It must however be stressed that values this high lead to poor performance as the measurement noise covariance will already be adjusted due to output noise and small calibration errors. This is easily understood when examining (2.123). With a high $\zeta$, the second term between the brackets will quickly overpower the first term and increase $r_{acc}$ when no motion is present. This way, the filter will be slowed even more.

**Sigma Point** The mean square error on the heading for the SPKFs versus the value for the adaptive parameter is given in figure 2.34. A big difference with the EKF graphs is that the frequency of the disturbance barely influences the mean square error when $\zeta$ is almost zero. The reason lies of course in the higher sensitivity of the sigma point filters which makes them react equally to any frequency of motion disturbance.

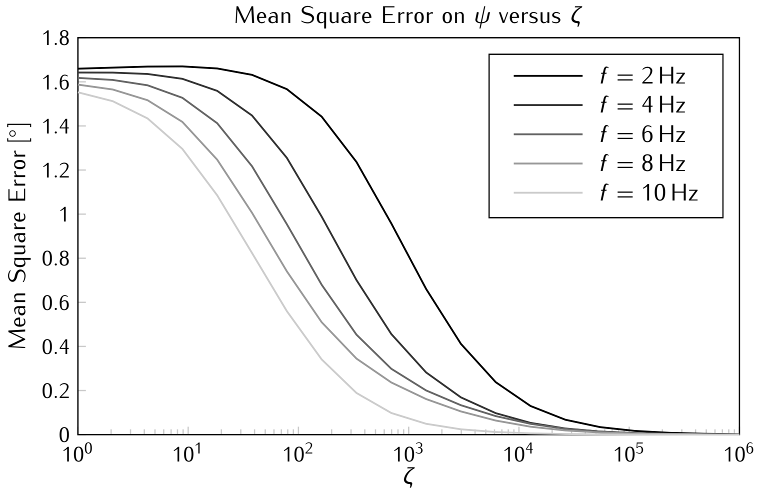


*Figure 2.34: Mean square error on the heading angle output of the SPKFs for increasing values of the adaptive parameter.*

Generally, the curves follow a similar pattern as in figure 2.33. Low values of $\zeta$ have almost no influence, suddenly the mean square error drops and finally saturation occurs. The transition period runs roughly from $\zeta = 10$ to $\zeta = 1 \times 10^4$, which covers much more acceptable values for the adaptation. For the graphs in the figure, $\zeta = 1000$ manages to reduce the mean square error to at least half the initial value.

Contrast to the EKF case, the value of $\zeta$ at which these changes in behaviour occur depend on the frequency of the disturbance. A lower frequency requires a much higher adaptation to obtain the same mean square error. The reason lies in the update nature of the Kalman filter. Each new estimate depends on the previous estimate and the noise covariances. A high motion disturbance frequency causes the measurement noise covariance to rise quicker and manages to keep the error on the output low, while a low frequency allows the filter to change its estimates before having a higher noise covariance block the error.

### Quaternion

In the quaternion case, the difference between the EKF and SPKFs is very small. The graphs follow a similar trend as in the case of the Euler SPKFs. Figure 2.35 displays the mean square error on each of the quaternion components when the disturbance has a frequency of 5 Hz. Choosing $\zeta = 1000$ again reduces the output error significantly.

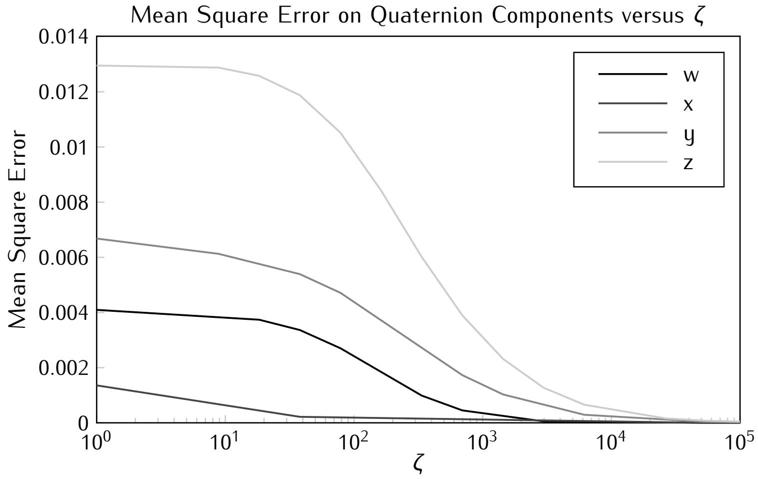


*Figure 2.35: Mean square error on the quaternion output of the EKF for increasing values of the adaptive parameter and for a disturbance with a* 5 Hz *frequency.*

### 2.6.3.2 Feedback Parameter

At first glance, one would expect that a higher feedback parameter will increase the error due to motion disturbance as it enhances the error even further by adding a fraction of the error in the prediction equation. This is indeed correct, however, as was shown by figures 2.14 and 2.16 in section 2.5.2.2, the process noise covariance can be reduced when $\tau$ is increased. A reduction of Q would allow the filter to trust more on the prediction, which would automatically mean that the sensor outputs will be weighed less and the motion disturbance signal would have less influence on the output. However, a decrease of the process noise covariance also means that the step response settling time will increase. A trade off is imminent.

### Euler

**Extended** Figure 2.36 shows the resulting mean square error of the Euler EKF with increasing feedback and for different values of the process noise covariance diagonal element. In the simulations to obtain the figure, the adaptive parameter $\zeta$ has been set to 1000 and the disturbance had a frequency of 5 Hz. The above theory is reflected in the graphs as the mean square error clearly rises when the feedback is increased. However, this increase can be countered by reducing the process noise covariance. When looking at the graph of optimal process noise covariance versus the feedback in figure 2.14, $q$ may easily be reduced from 0.5 at $\tau = 0$ to 0.05 at $\tau = 0.8$. According to the graphs in figure 2.36, changing both values will have little influence on the mean square error.

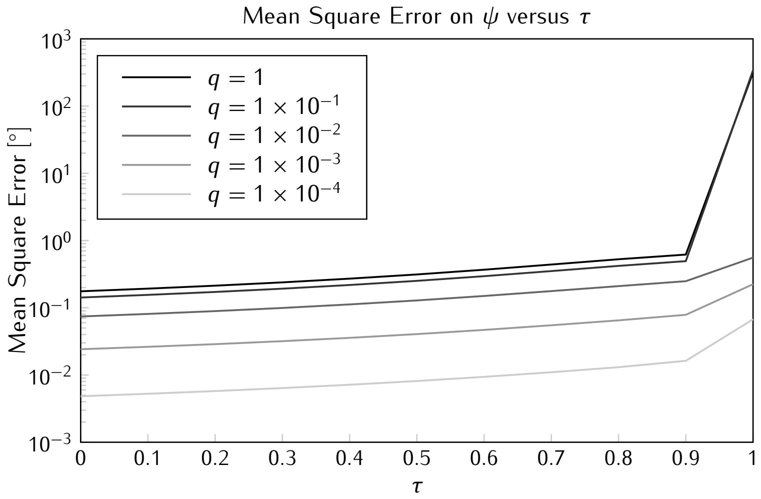


*Figure 2.36: Mean square error on the heading angle output of the EKF for increasing values of the feedback parameter and different process noise covariance.*

Contrast to the expectations, decreasing the value of $q$ has little or no influence on the settling time of the step response. This is again a consequence of the low sensitivity of the Euler EKF resulting in a fairly slow response time and the fact that increasing $\tau$ has a bigger influence on the settling time.

**Sigma Point** Figure 2.37 shows the output graphs of the Euler SPKFs in the same conditions as mentioned above. The graphs point to the same conclusion as was the case for the EKF: decreasing $q$ when increasing $\tau$ allows a similar mean square error on the output.

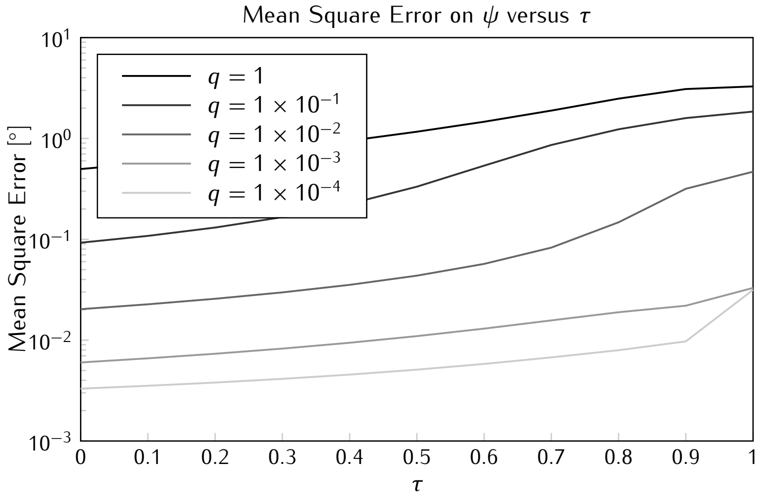


*Figure 2.37: Mean square error on the heading angle output of the SPKFs for increasing values of the feedback parameter and different process noise covariance.*

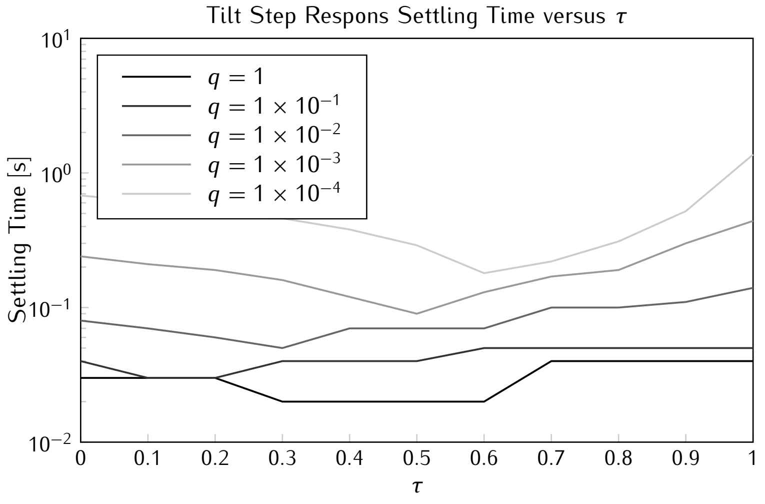


*Figure 2.38: Tilt step response settling time of the Euler SPKFs for increasing values of the feedback parameter and different process noise covariance.*

In the case of the SPKFs, the choice of $q$ has a bigger influence on the settling time of the step response as can be seen in figure 2.38. With increasing $\tau$, all of the graphs show a decrease in settling time before increasing afterwards. The reason is that at a certain point, overshoot occurs which causes the settling time to increase. Furthermore, it is clear that a reduction of $q$ will in any case increase the settling time. Setting the upper bound to 0.1 s, $q$ should remain above $1 \times 10^{-2}$.

### Quaternion

The graphs resulting from simulations of the quaternion state filter follow similar trends as was the case for the Euler SPKFs. Increasing $\tau$ must be coupled with a decrease in $q$ to obtain a similar mean square error on the output. In order to keep the settling time on the tilt step response below 0.1 s, $q$ should now be larger than $1 \times 10^{-6}$.

## 2.7 Conclusion

After giving a thorough overview of Euler angles and quaternions as a means for representing three dimensional orientation and introducing the Kalman filter and all of its variations, an orientation tracking algorithm was presented. Several flavours of the filter were implemented supporting either an MFG or MARG type sensor node, based on either an EKF or SPKF architecture and using either Euler or quaternion representation. Additional features, such as feedback and adaptive noise covariance were added to improve the MFG type filter's performance.

Using real life motion captures from an optical tracking system, all parameters of the filters were estimated to ensure an adequate modeling of the problem is obtained. The process noise covariance depends upon the newly introduced feedback parameter $\tau$, whose optimal value was obtained through auto regression analysis of the captures. A third order inverted Chebychev filter was also proposed in order to reduce the amount of high frequency output noise of the sensors. The pass band of this digital filter extends to 6 Hz with the stop band starting at 12 Hz and offering a minimum of 20 dB attenuation.

Finally, the filter was simulated in order to test its characteristics and determine an estimate for the adaptive parameters in the system. From these simulations it is clear that the EKF Euler type filter has a much slower response than all other filters. Clearly, the first order approximation by calculation of the Jacobian matrix does not suffice for the highly non-linear measurement model containing several trigonometric functions. For this reason, this filter will not be considered in the following chapters, as it will always be outperformed by the other filters.

Different step response behaviour was obtained for tilt and heading steps. This is a clear reflection of the fact that the gravity vector coincides with the Z-axis. Settling times range from 30 ms for tilt to 60 ms for heading changes.

Noise response simulations confirmed that the sensor output digital pre-filter reduces the influence of the high frequency noise on the filter output. Also, simulated white noise turned out to be a bad approximation of sensor noise due to the poor whiteness of the latter. The variance on the output signals introduced by the noise is ten times higher with actual sensor noise. However, pre-filtering reduces this variance back to approximately the same level as the variance of the simulated noise response.

Finally, motion disturbance simulations were executed where a disturbing sinusoidal signal was added to one of the accelerometer outputs. It has been shown that a value of 1000 for the adaptive filtering parameter $\zeta$ significantly reduces the mean square error on the output of both the Euler and quaternion type filters. Furthermore, the increase of this error due to higher values of the feedback parameter $\tau$ can be countered by reducing the process noise covariance at the cost of an increased step response settling time.

# References

[1] E.R. Bachmann, R.B. McGhee, X. Yun, and M.J. Zyda. *Rigid Body Dynamics, Inertial Reference Frames, and Graphics Coordinate Systems: A Resolution of Conflicting Conventions and Terminology*. In Proceedings of the IEEE Symposium on Computational Intelligence in Robotics and Automation, 2000.

[2] J. Diebel. *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*, October 2006.

[3] H. Goldstein. *Classical Mechanics*, chapter 4-4 The Euler Angles, pages 143–148. Addison-Wesley, 1980.

[4] W.T. Thomson. *Introduction to Space Dynamics*. Wiley, 1961.

[5] L.D. Landau and E.M. Lifschitz. *Mechanics*. Pergamon Press, Oxford, England, 1976.

[6] H. Goldstein. *Classical Mechanics*, chapter Appendix B: Euler Angles in Alternate Conventions, pages 606–610. Addison-Wesley, 1980.

[7] P. Kelland and P.G. Tait. *Introduction to Quaternions*. Macmillan, London, 1904.

[8] W.R. Hamilton. *On a New Species of Imaginary Quantities, Connected with the Theory of Quaternions*. In Proceedings of the Royal Irish Academy, volume 2, pages 424–434, 1844.

[9] D. Hearn and M.P. Baker. *Computer Graphics: C Version*, pages 419–420 and 617–618. Prentice-Hall, Englewood Cliffs, NJ, 1996.

[10] J.M. Cooke, M.J. Zyda, D.R. Pratt, and R.B. McGhee. *NPSNET: Flight Simulation Modeling Using Quaternions*. Presence, 1(4):404–420, 1992.

[11] R.E. Kalman. *A New Approach to Linear Filtering and Prediction Problems*. Transaction of the ASME Journal of Basic Engineering, 82D:35–45, 1960.

[12] Peter S. Maybeck. *Stochastic Models, Estimation and Control*, volume 1, chapter 1. Academic Press, 111 Fifth Avenue, New York, New York 10003, USA, 1979.

[13] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*. ACM, 2001.

[14] S. Julier and J. Uhlmann. *A New Extention of the Kalman Filter to Nonlinear Systems*. In Proceedings of SPIE International Society of Optical Engineers, pages 182–193, Orlando, Florida, USA, April 1997.

[15] R. Van Der Merwe. *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. PhD thesis, OGI School of Science & Engineering, Health & Science University, Oregon, USA, 2004.

[16] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Steward. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[17] W.H. Press, S. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[18] S. Julier, J. Uhlmann, and H. Durrant-Whyte. *A New Approach for Filtering Nonlinear Systems*. In Proceedings of the American Control Conference, pages 1628–1632, 1995.

[19] K. Ito and K. Xiong. *Gaussian Filters for Nonlinear Filtering Problems*. IEEE Transactions on Advanced Control, 45(5):910–927, May 2000.

[20] M. Norgaard, N. Poulsen, and O. Ravn. *New Developments in State Estimation for Nonlinear Systems*. Automatica, 36(11):1627–1638, November 2000.

[21] S. Julier. *The Scaled Unscented Transformation*. In Proceedings of the American Control Conference, volume 6, pages 4555–4559, May 2002.

[22] J. Stirling. *Methodus Differentialis, Sive Tractatus de Summation et Interpolation Serierum Infinitarium*, 1730.

[23] M. Norgaard, N. Poulsen, and O. Ravn. *Advances in Derivative-Free State Estimation for Nonlinear Systems*. Technical Report IMM-REP-1998-15, Department of Mathmatical Modeling, Technical University of Denmark, 28 Lyngby, Denmark, April 2000.

[24] A. Gelb. *Applied Optimal Estimation*. MIT Press, Cambridge, MA, 1988.

[25] A.H. Mohamed and K.P. Schwarz. *Adaptive Kalman Filtering for INS/GPS*. Journal of Geodesy, 73(4):193–203, 1999.

[26] F. Ferraris, U. Grimaldi, and M. Parvis. *Procedure for Effortless In-Field Calibration of Three Axis Rate Gyros and Accelerometers*. Sensors and Materials, 7(5):311–330, 1995.

[27] D. Titterton and J. Weston. *Strapdown Inertial Navigation Technology*. The Institution of Engineering and Technology, 2004.

[28] H. Hou. *Modeling Inertial Sensors Errors Using Allan Variance*. Technical Report 20201, Department of Geomatics Engineering, The University of Calgary, Alberta, Canada, 2004.

[29] National Geophysical Data Center. *Online Magnetic Field Calculator*. http://www.ngdc.noaa.gov/geomagmodels/IGRFWMM.jsp.

[30] J.P. Martens. *Signal Processing*. Lecture Notes, 2004.

[31] S. Butterworth. *On the Theory of Filter Amplifiers*. Wireless Engineer, 7:536–541, 1930.

[32] A.B. Williams and F.J. Taylors. *Electronic Filter Design Handbook*. McGraw-Hill, New York, 1988.

[33] Mathworks. *Signal Processing Toolbox*. http://www.mathworks.com/products/signal/.

# 3
# System Design

The actual system design is presented in this chapter. It consists of several aspects ranging from the hardware design with the choice of components, schematics design and layout of the circuit to the development of embedded software to have all of the subsystems function individually and allow communication between them.

## 3.1 Introduction

Before the system design is fully explained, the requirements for these systems are outlined and an overview is given of the systems that have been designed throughout the years.

### 3.1.1 System Requirements

First of all, the system should naturally be able to supply the necessary data to perform the task at hand. Given the filter design of the previous chapter, this implies that at least three dimensional acceleration and magnetic field measurements should be present. Additionally, this data can also be complemented with gyroscope readings. Aside from the sensors themselves, additional devices should be present to transfer all of the information to a processing unit, in this case a Personal Computer (PC), where the data may be processed to an orientation and where the result can be visualised.

Second, an inertial motion tracking system must also fulfill a range of requirements that are normally associated to mobile systems. It should be as unobtrusive as possible. It should be light in order for the user to be able to carry it around and it should be small so that it doesn't hamper the user in his movements. Generally speaking, it should be invisible to the user such that it has no influence on his or her behaviour.

Finally, the system should be portable. This implies that wireless technology should be used to transfer measured data and that the system will need to be powered by batteries. From the second requirement immediately follows that, given the limited power capacity, the system should consume as little power as possible. This will be reflected in both the choice of components and the implementation of the functionality in firmware.

### 3.1.2 Available Systems Overview

Over the past few years, several generations of motion tracking systems have been developed. Initially, an existing system was already present as the result of the master thesis of ir. Niels Decraene [1]. Within the scope of the thesis, several types of sensor nodes were designed containing a variety of sensor types and brands [2–5]. This system was the first step of CMST into the world of inertial sensing and thus mainly consisted of test boards. As the goal of the thesis was to explore the possibilities of accelerometers and perhaps other sensors, none of the boards were specifically designed for actual orientation tracking. This is clearly reflected in the fact that none of them contain a means for measuring the earth's magnetic field in three dimensions, which is a crucial part of the tracking process.

The initial system however, has set the basis for future generations of systems. Wireless communication was implemented successfully on some of the system's nodes, although the transfer rate still needed improvement. The world's first two dimensional integrated gyroscope [4] was used together with a more conventional, uniaxial sensor [3] to form a flat board that could measure angular rate in three dimensions. Previously, several boards had to be interconnected and placed perpendicular to each other to obtain this. Throughout the years, a fully integrated, three dimensional gyroscope [6] became available, making the design even more compact. Finally, the system build-up consisting of a microcontroller [7] interconnected with several sensors and a wireless transceiver chip [8] has been reused in newer designs.

After one year, a new system became available designed by two master students in the scope of their master thesis [9]. It consisted of several MARG sensor nodes capable of transmitting their data at a rate of 100 Hz to a central receiver via a wireless interface. The system had grown over several prototypes of smaller nodes in fewer amounts until the final version. This system was used extensively for several tests and helped develop the tracking filter to what it is today. The system has also determined the final network layout where multiple nodes send data to a base station in a turn-based approach [10].

Finally, a fully optimised system has been designed combining all of the best aspects of the preceding versions. The system exists in both an MFG version as well as a MARG version. The microcontroller and wireless transceiver have been retained from the initial design, while the sensors and some of the network aspects have been copied from the second generation. Extra features were then added to make the system outperform its predecessors. A plug-n-play wireless ad hoc network was coded into the firmware supporting a large number of sensor nodes to function simultaneously. One of the tracking filters was also implemented in firmware, relieving the backend computer from this task. Fully flat, single board MARG nodes were conceived using three fully integrated three dimensional sensors. And all of this was realised while minimising the power consumption.

Aside from the developments on rigid boards, other technologies were also explored. A flexible version of the MFG nodes was designed and produced at CMST. Careful and creative routing allowed to interconnect all of the components within a single conductive layer. Finally the first steps have been taken towards the development of nodes with the CMST patented Ultra Thin Chip Package (UTCP) technology [11] where chips are embedded within the flexible substrate. This way, the whole node becomes even more unobtrusive and, combined with flexible batteries, a fully flexible node can be made.

## 3.2 Hardware

An overview of the available hardware platforms used to obtain the results of this work is given in this section. First, a general overview of the system is given. Then several generations of designs as they have been outlined in the introduction are described. An overview is given of each of the subsystems present in that generation and all of the components used to create the hardware.

The first generation designed by Niels Decraene [1] will not be described here as none of the designed nodes contained all of the necessary sensors to perform true driftless three dimensional orientation tracking. The key features of these designs have already been mentioned in the introduction.

### 3.2.1 General System Layout

The system consists of two major parts: the sensor nodes and the base station. Multiple sensor nodes are needed to track an entire human person as each limb should be equipped with a node. Generally, the human body can be approximated by a rigid body with 15 interconnected links [12]. Although a single base station should suffice, multiple base stations could improve the system performance by decreasing the data loss.

#### 3.2.1.1 Sensor Node Build-up

Sensor nodes need to be attached to the tracking subject. Logically this calls for a subsystem that is very light, small and comfortable, but also wireless and low power. In general, the circuit of a sensor node should consist of three major parts: a microcontroller, sensors and a wireless transceiver. The block diagram displayed in figure 3.1 shows the general layout of the sensor node subsystem.

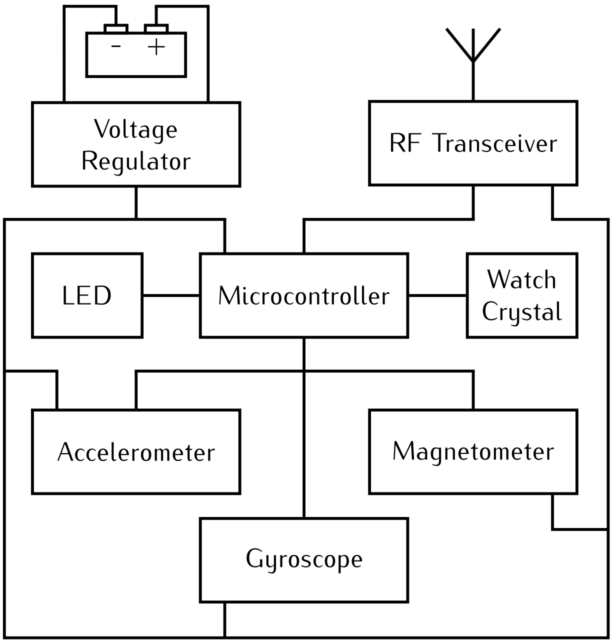


*Figure 3.1: Block diagram of the standard sensor node build-up.*

The microcontroller forms the heart of the node, controls all of the events and manages the signals. The sensors are connected through a shared digital interface for communication or via the Analogue to Digital Convertor (ADC) pins of the microcontroller in case it concerns an analogue sensor. The Radio Frequency (RF) transceiver is connected via another digital interface. Power is supplied from a battery and converted to a steady voltage by a Low Dropout (LDO) voltage converter. A LED is added for visual feedback of the node's functionality and a low frequency watch crystal allows accurate timing of the actions taken by the system.

#### 3.2.1.2 Base Station Build-up

The base station is designed for data throughput and should be optimised to perform this task as quickly as possible. Consequently, the build-up consists of a receiving interface, in this case the wireless transceiver, and a transmitting interface connected to a computer. This transmitting interface can be either of the available standard computer interfaces e.g. Universal Serial Bus (USB) [13] or Ethernet [14] or even a wireless interface as e.g. Wi-Fi [15] or Bluetooth [16]. The block diagram of this type of circuit is displayed in figure 3.2.

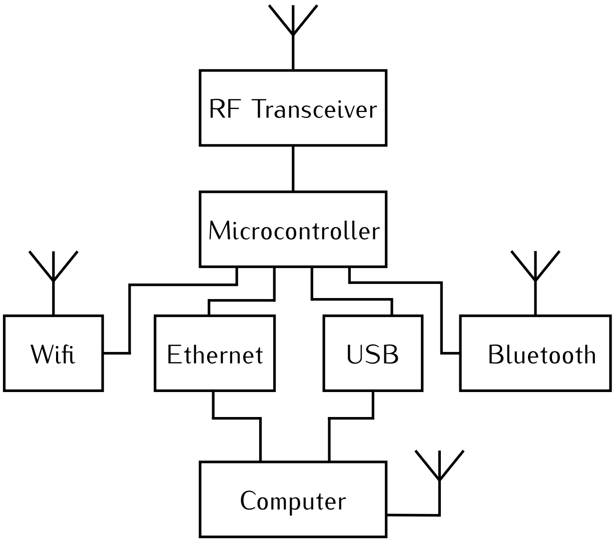


*Figure 3.2: Block diagram of the standard base station build-up.*

Aside from the interfaces, the power circuit should also be considered. Several sources should be supported, but, if present, power can also be taken from the computer via the USB interface as this does not require any additional battery or power source.

### 3.2.2 Second Generation

The second generation tracking system has been designed by ir. Wouter Verstichel and ir. Bart Kuyken in the frame of their master thesis [9]. This system consisted of several sensor nodes that are able to transmit measurement data to a base station via wireless communication.

#### 3.2.2.1 Sensor Node

The sensor nodes of this generation consist of several boards that must be interconnected in order to obtain a fully functional subsystem. As can be seen in figure 3.3, three boards are needed, the main board, the wireless board and the gyroscope board. The wireless board was designed in such a way that it could also be replaced by a commercially available, more powerful version [17]. The outside dimensions and weights of the boards are listed in table 3.1, the last line gives the overall dimensions when all boards have been connected together.

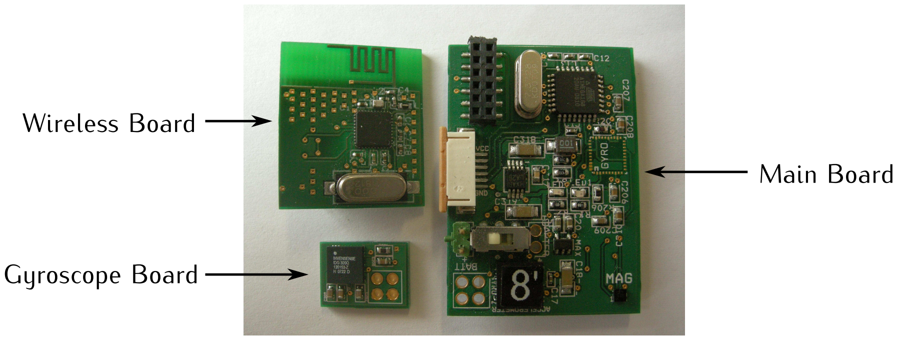


*Figure 3.3: Picture of the individual boards of the second generation sensor node.*

| | Length [mm] | Width [mm] | Thickness [mm] | Weight [g] |
|---|---|---|---|---|
| Main Board | 42 | 30 | 5 | 8.1 |
| Wireless Board | 26 | 22 | 11 | 3.1 |
| Gyroscope Board | 14 | 11 | 3 | 0.8 |
| Assembled Node | 55 | 30 | 14 | 12 |

*Table 3.1: Second generation sensor node boards dimensions and weights.*

A fully assembled sensor node with all boards connected together and a battery [18] attached to the bottom is displayed in figure 3.4. The battery weighs 18.9 g and measures 60 by 34 by 3.8 mm and has been attached to the backside of the main board using Velcro.

### Accelerometer

The fully integrated LIS3LV02DQ accelerometer from ST Microelectronics was used [19]. This sensor offers acceleration measurements in three dimensions and allows internal conversion to a digital signal that can be obtained through either an Inter-Integrated Circuit (I$^2$C) bus or a Serial Peripheral Interface

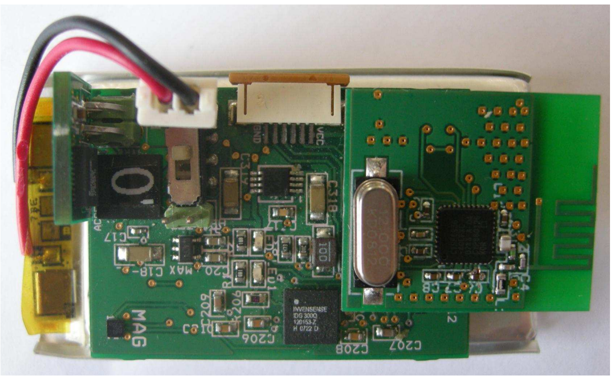


*Figure 3.4: Picture of an assembled second generation sensor node.*

(SPI). Furthermore, it features a programmable range of ±2 or ±6 g and an output data rate ranging from 40 Hz to 2.5 kHz. Especially the lower range is of intrest, since only gravity is supposed to be measured. When configured this way, the device has a resolution of about 1 mg and a non linearity error that is limited to 60 mg. This translates in possible orientation errors below 3° and a resolution of 0.05°.

### Gyroscopes

Two gyroscopes were needed in order to obtain fully three dimensional angular rate information as three dimensional integrated gyroscopes did not exist yet. The world's first two dimensional gyroscope, the IDG300 from Invensense [4] with a range of 500 °/s, is used twice on each node. One gyro is placed on the main board and measures the tilt rates and another one is placed on the vertical gyroscope board to measure the yaw rate. The reason that no yaw rate gyroscope is placed on the main board to complement the two dimensional gyro is that no single axis gyroscope was found that functioned with a 3 V supply. Otherwise, a second 5 V supply had to be added to feed this component.

### Magnetometer

The magnetometer used is the YAS529 from Yamaha [20]. This device is the smallest 3D integrated magnetometer available at the moment, measuring only 2 x 2 x 1 mm. It is equipped with an I$^2$C compatible digital interface and has a built-in temperature sensor and initialization coils. The geomagnetic sensor also features an automatic power down control which keeps the sensor in a stand-by state once a measurement is completed. This component is very suited for the application at hand not only for its very compact size and restricted current consumption, but also due to its high sensitivity which makes the sensor sensitive to earth's magnetic field.

### Microcontroller

The ATmega168 [21] from Atmel's AVR family of 8 bit low power microcontrollers is used. It features a Reduced Instruction Set Computer (RISC) architecture with many programmable peripherals ranging from counters and analog comparators to digital interfaces and ADCs. The low power aspect is reflected in the 0.1 µA current consumption in stand-by mode and 250 µA in active mode when fed by a 1.8 V supply and operating at a 1 MHz clock frequency.

### RF Transceiver

The RF communication is provided by the Cypress CYRF6936 [22] single chip transceiver operating in the unlicensed 2.4 GHz Industrial, Scientific and Medical (ISM) frequency band. The chip takes care of all signal modulation and demodulation and communicates to the outside world via an SPI. With a Transmit (TX) and Receive (RX) current consumption of respectively 34.1 mA and 21.2 mA, the RF part of the chip takes up the biggest part of the power consumption. Therefore, low power modes exist where only the digital interface is active leading which leads to a consumption of only 1 mA. This way, data can be clocked in or out of the chip while the frontend is switched off.

## 3.2.2.2 Base Station

The data from the sensor nodes needs to be received by a base station that is connected to a computer in order to allow processing of the data and visualisation of the result on the screen. Several options exist to transfer data to a computer as multiple interfaces are readily available. The base station in the second generation allows communication via either USB or Ethernet. A picture of a fully assembled base station is displayed in figure 3.5, it utilises the same platform of microcontroller and RF transceiver as the sensor nodes. The board measures 85 mm in length, 29 mm in width and is 26 mm thick.

### USB

The USB interface consists of a Universal Asynchronous Receiver/Transmitter (UART) to USB converter chip, the CP2102 from Silicon Laboratories [23]. A microcontroller can send data serially to this chip, which will then convert it in valid USB signals. The inverse data direction is of course also available.

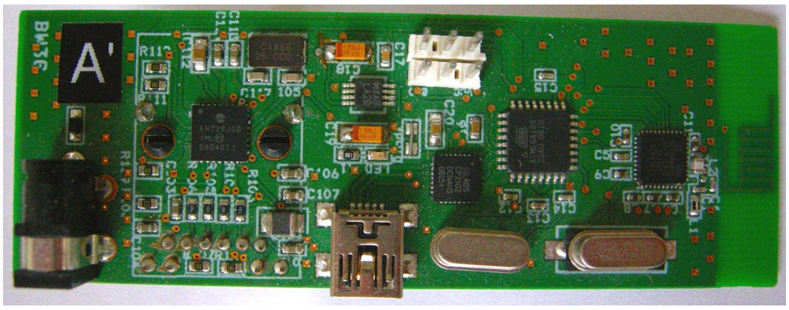


*Figure 3.5: Picture of an assembled second generation base station.*

At the computer side, a driver is installed that mounts a virtual serial port from which data can be read.

A major advantage of the USB interface is the fact that power can be drawn from the computer, which can be used to feed the entire base station circuit as the bridge chip also incorporates a regulator. This avoids the need for an external battery or power source.

### Ethernet

The ENC28J60 from Microchip [24] provides the base station of an Ethernet controller. This stand-alone device is equipped with an SPI through which the device can be set up and data can be transferred or received. The Media Access Control (MAC) and physical layer are implemented on the chip, other layers as e.g. transport or network, need to be implemented in the microcontroller.

The Ethernet interface has several advantages over USB. First, any computer can be added to the network containing the base station and can start receiving and visualising the sensor data. With a USB connection, an extra base station is needed. Second, multiple networked base stations can easily be placed at various locations in order to obtain lower data loss. Wireless communication is prone to changes in the environment which can be very local, placing multiple base stations might counter the effects of these changes. At the computer side, all packages need to be combined and duplicates have to be removed. Finally, this type of interface does not require any drivers to be installed since the network interface is expected to be present and active in a modern day computer.

### 3.2.3 Third Generation

In the third generation, two types of sensor nodes were created, one with and one without a gyroscope. The multiple interconnected board structure from the second generation was abandoned as the connectors proved to be a major source of problems and random stalling. Given the fact that a different RF transceiver was chosen to offer a bigger range with an available antenna design, the base station also needed to be redesigned.

#### 3.2.3.1 MFG Node

A picture of a fully assembled MFG sensor node is displayed in figure 3.6. The same magnetometer from the second generation nodes is used and the accelerometer is the LIS302DL from ST Microelectronics [25], a component with similar performance to the one selected for the second generation. The outside dimensions are a length of 55 mm, a width of 24 mm and a thickness of 7 mm. The board including all components weighs 6 g.

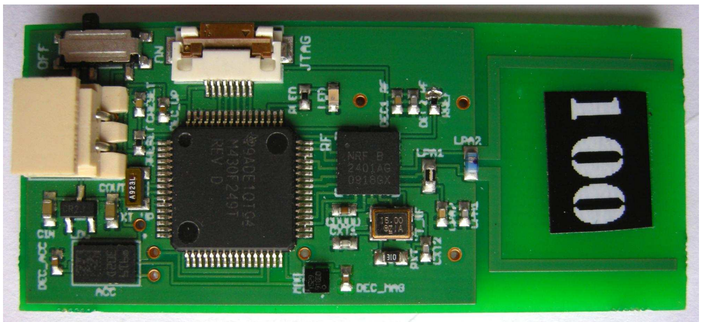


*Figure 3.6: Picture of an assembled third generation MFG sensor node.*

#### Microcontroller

The MSP430f249 [26] from Texas Instruments' ultra low power microcontroller family is used. This family of controllers incorporate a 16-bit RISC Central Processing Unit (CPU) in a von Neumann architecture [27]. Several peripherals such as digital interfaces (UART, SPI, etc.), ADCs, Digital to Analogue Convertors (DACs), timers and Hardware Multipliers (HWMs) can be present depending on the family member. Ultra low power functionality is obtained through the availability of numerous low power modes where parts of the system can be temporarily disabled and many interrupt sources are possible. The current consumption of this component is very comparable to the AVR, the biggest difference however is found in the 16-bit functionality that is key to the embedded implementation of the orientation tracking algorithm.

### RF Transceiver

The RF transceiver used in this design is the Nordic Semiconductor nRF2401 [8]. The functionality of this device is comparable to that of the Cypress chip used in the second generation. It is a single chip RF solution with a power saving function called *Shockburst*, where data is first acquired via the SPI in sleep mode before being transmitted in a fast active burst. The biggest difference however, lies in the *DuoCeiver* functionality. In this mode, the RF transceiver is able to receive data in two separate channels simultaneously through a single antenna at maximum data rate. These channels must be spaced exactly 8 MHz apart to benefit from this functionality and received data is available through separate digital interfaces on the chip. The actual current consumption of this device is also significantly lower with 13 mA and 23 mA being consumed during respectively TX and RX operation and as low as 12 µA during stand-by.

### 3.2.3.2 MARG Node

The MARG node is identical to the MFG node aside from the added angular rate sensor. A picture of an assembled MARG node is displayed in figure 3.7. The node measures 55 mm in length, is 26 mm wide and has a thickness of 7 mm. It weighs 6.3 g.

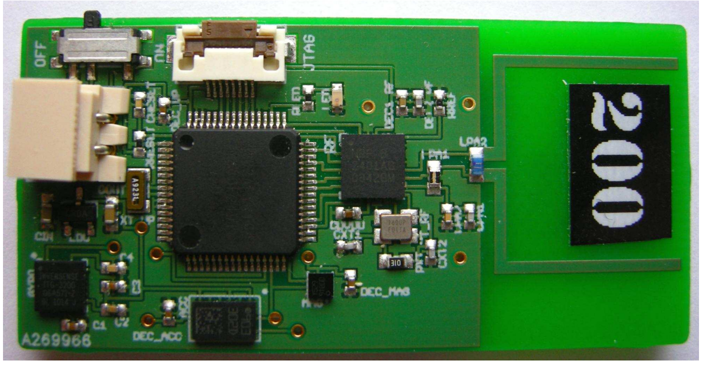


*Figure 3.7: Picture of an assembled third generation MARG sensor node.*

### Gyroscope

The ITG3200 from Invensense [6] has been selected as angular rate sensor. It is the world's first fully integrated triaxial gyroscope and features 16 bit ADCs, an $I^2C$ interface and a full scale range of ±2000 °/s. This higher range compared to the IDG300 was necessary since saturation was seen when using the second generation nodes during experiments. Thanks to this brand new device, the MARG sensor node is very compact and only consists of a single board.

### 3.2.3.3 Base Station

The design of the base station was part of the work executed by ir. Bert Vanhoutte in the frame of his master thesis [28]. It is based on the same platform as the third generation sensor nodes and thus contains the same microcontroller and RF transceiver. As was the case with the second generation receiver, it features a USB interface based on a UART to USB bridge [29] and an Ethernet connection [24]. Additionally, a Bluetooth interface is also available. Figure 3.8 shows a picture of a fully assembled base station. The outer dimensions are a length of 71 mm, a width of 57 mm and a thickness of 15 mm.

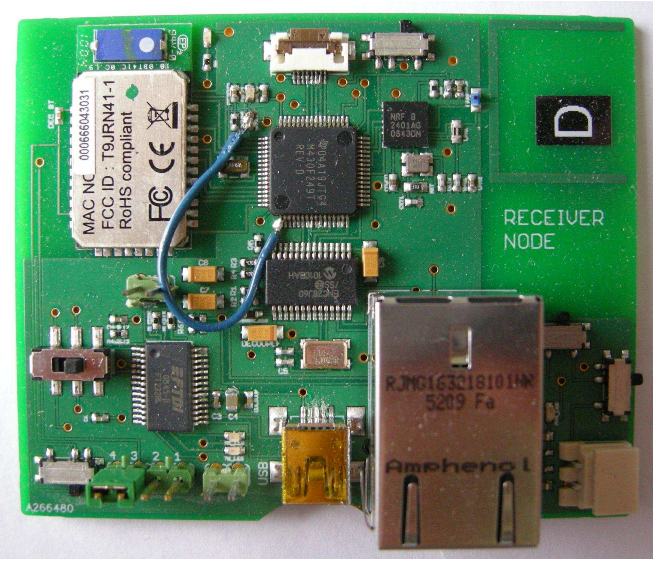


*Figure 3.8: Picture of an assembled third generation base station.*

### Bluetooth

The integration of a Bluetooth interface proved a bigger challenge then expected. Although single chip solutions do exist, these devices are mostly optimised for very specific tasks and require input sequences corresponding to Host Controller Interface (HCI) layer commands. This means that the Bluetooth stack also needs to be implemented in the microcontroller, which is clearly beyond the scope of this work. An alternative is the use of a Bluetooth module which can be addressed using simple commands via a UART interface. Eventually the RN41 from Roving Networks [30] was chosen.

The use of a wireless interface also opens up new possibilities for the network topology. In terms of scalability, it could be interesting to have each user carry a base station that collects all of the data from the sensor nodes placed on that person's body and have these devices function in some sort of BAN. A second network consisting of all these BANs and one or more computers could then be formed by dedicating the base stations as access points. This functionality has however not been fully investigated as it turns out that the Bluetooth module used in the design is fairly limited in capabilities since the data rate did not meet up to the expectations.

## 3.2.4 Fourth Generation

In the fourth generation, the focus has turned to the production technology used for the sensor nodes. In a first step, the design was transferred to an in house flexible board technology with a single conductive layer. Afterwards, the CMST patented UTCP technology is used to make even the Integrated Circuits (ICs) flexible.

### 3.2.4.1 Flexible Board

Only an MFG type sensor node has been designed for realisation with flexible technology.

### Process

The process flow of the CMST flexible board technology is displayed in figure 3.9. It is based on a 50 µm thick polyimide substrate with a single 18 µm thick layer of copper on top (a). Patterning of the copper is done by standard lithography steps: resist layer deposition through spin coating (b), exposure to Ultraviolet (UV) light through a pattern mask (c), removal of unexposed resist (d), etching of the copper parts that are no longer protected by resist (e) and stripping of the remaining resist (f). Afterwards, a 20 µm thick solder mask is applied (g) and a Nickel Gold finish is plated on the parts of the copper

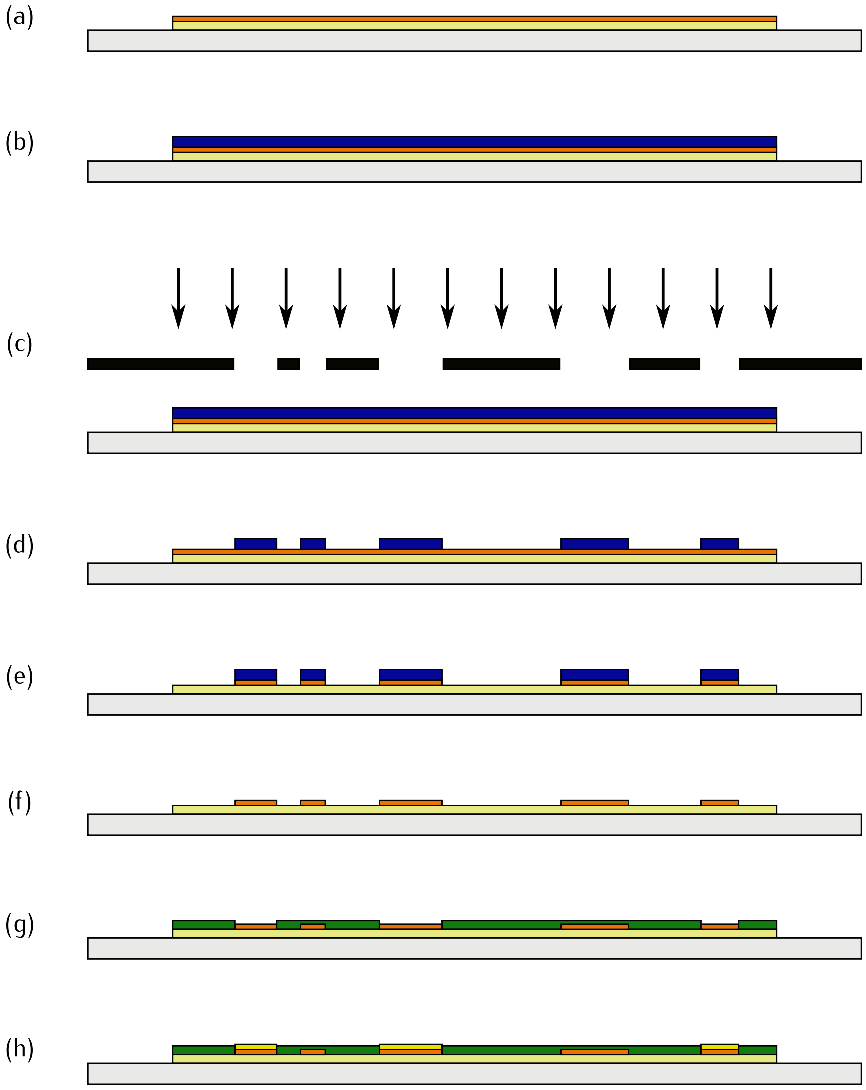


*Figure 3.9: Process flow of the flexible board technology.*

that are still open to avoid corrosion and improve soldering (h). Finally, components can be soldered on the surface using a standard reflow process. Note that during all of these steps, the substrate is glued to a temporary rigid carrier in order to ease handling and avoid curling of the substrate.

### MFG Node

A top view of a fully assembled flexible MFG sensor node is displayed in figure 3.10. The node contains the same sensors and RF transceiver as the third generation MFG node, but a different microcontroller, the MSP430f2132 [31] is chosen. The reason lies in the smaller package of this component. The wires at the top of the picture allow connection to a battery and also programming of the microcontroller firmware. Figure 3.11 shows a side view of the flexible node. The low thickness of the flexible substrate board is clearly visible.

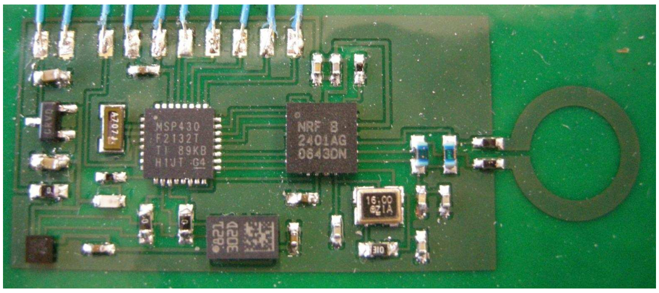


*Figure 3.10: Picture of an assembled fourth generation flexible MFG sensor node.*

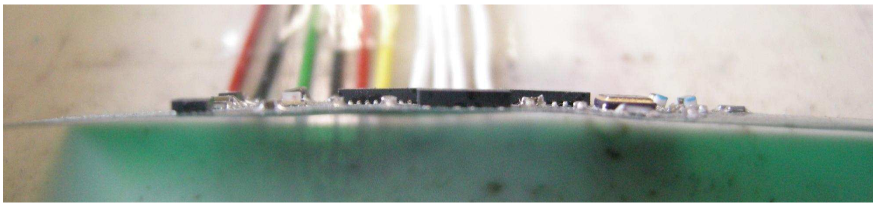


*Figure 3.11: Side view of an assembled fourth generation flexible MFG sensor node.*

Due to the availability of only a single conductive layer, careful routing of the signals had to be performed. Unused pins of the microcontroller provided handy shortcuts to route otherwise blocked signals. Despite all creative routing however, zero Ohm resistors had to be used to bridge one signal over another. Note however that the nodes in the pictures still have three of these bridges, while future versions can be made with only one.

### 3.2.4.2 UTCP

The UTCP technology has been developed by dr. ir. Wim Christiaens in the scope of his PhD research [32]. The process of the technology includes the thinning of individual chips down to approximately 15 µm and the embedding of these chips in flexible substrates. The total thickness of the package is lower then 50 µm and thanks to the very low thickness of the chip, the entire structure is even bendable. Currently, the production technology and process flow are being refined and steps are made towards stacking of thinned dies.

### Process

The UTCP process consists mainly of two parts: chip thinning and embedding.

The chip thinning technology is optimized on a PM5 Precision Lapping and Polishing Machine [33] and is a combination of a lapping and a polishing process. The first stage is a lapping process which has a much higher material removal rate and produces a non reflective matt surface. The second stage is the polishing process where surface damage usually created during lapping is removed and a reflective surface is obtained. The surface roughness produced in the lapping stage depends on the abrasive particle size and the hardness of the material. In the polishing stage, the final roughness only depends on the polishing method. Details and optimization for the different process steps are available in [32].

The process flow of the embedding part is displayed in figure 3.12. The base substrate consists of a polyimide layer as was the case for the flexible technology but without the copper layer (a). A layer of photo definable polyimide is spun on top of the first layer (b). Then, a cavity is made in this second layer using photo lithography (c) and a drop of Benzocyclobutene (BCB) is dispensed inside the cavity to serve as adhesive material (d). Afterwards, the thinned chip may be placed, face up, inside the cavity and, together with the BCB, the cavity is now entirely filled (e). The whole stack is now covered with a second layer of photo definable polyimide to obtain a flat substrate (f). Next, via's are defined in the top polyimide layer via a second photo lithography step in order to make the contacts of the chip accessible (g). Finally, the via's are metallised and a third lithography step allows the metal to be patterned (h). Note that, as with the flexible technology, a rigid carrier substrate is used the entire process.

### Nodes

Although no actual sensor nodes using the UTCP technology have been fabricated, some initial designs have been made for first tests and future production. Since a lot of effort has been made for packaging microcontrollers and RF transceivers as UTCPs, these are the devices that will be embedded. In the presented designs, the chips are embedded in a flexible board as described by the UTCP process and these packages are then mounted on top of the base circuit flexible board which contains the other components and

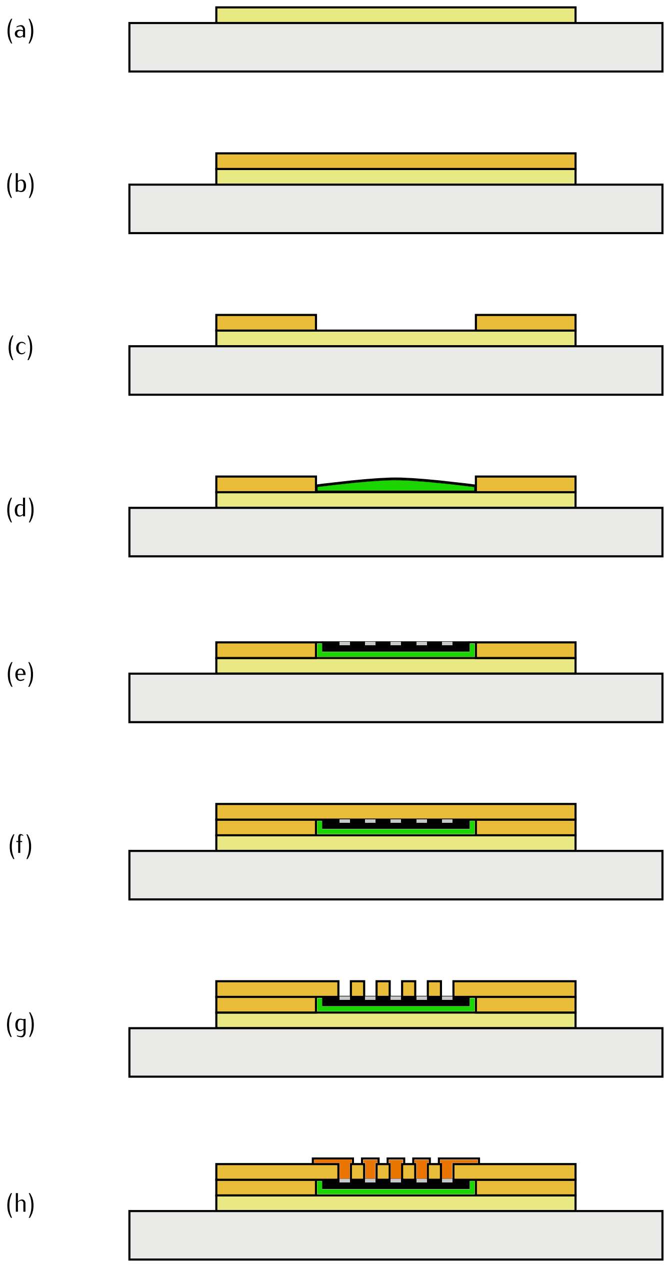
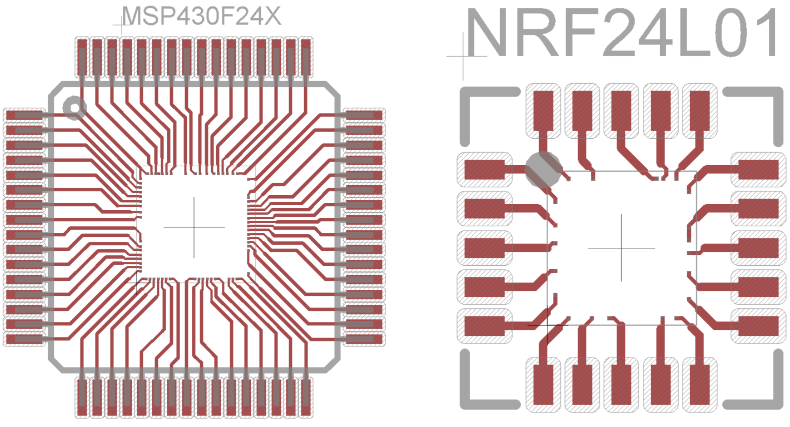


*Figure 3.12: Process flow of the UTCP technology embedding part.*

the circuit routing. In a later stage, all UTCP components can be embedded inside the base circuit board and conventional components can be mounted on top of the embedded devices. This way, an even more compact result can be obtained by exploiting all available space.

The UTCP sensor node designs represent an MFG type sensor node containing the exact same components as the third generation MFG nodes. The only difference is that a newer version of the RF transceiver is used, the nRF24L01 [34], which is entirely backwards compatible with its predecessor.

The first design consists of a base circuit board that resembles the flexible sensor node design and two separate flexible boards containing the UTCP devices. Both the UTCP boards are displayed in figure 3.13. The metal layer is used to form a taper from the chip contact pads towards a footprint compatible to the actual packaged devices.



(a) MSP430F249 UTCP design

(b) nRF24L01 UTCP design

*Figure 3.13: Flexible boards for UTCP devices.*

The base board design is shown in figure 3.14 and accommodates space for all of the components in the circuit, including both of the UTCP boards. Note that due to the fact that the thinned chips are placed face up in the cavity and thus face down on the base board, the footprints are actually mirrored compared to the normal packaged devices.

In the second design, both thin chips are supposed to be packaged in the same flexible board forming a combined UTCP. The challenge is found in the need for accurate placing of both components, as the interconnections between both are now realised by the metal layer of the UTCP technology. Figure 3.15 shows the design of the combined UTCP. Note that not all of the

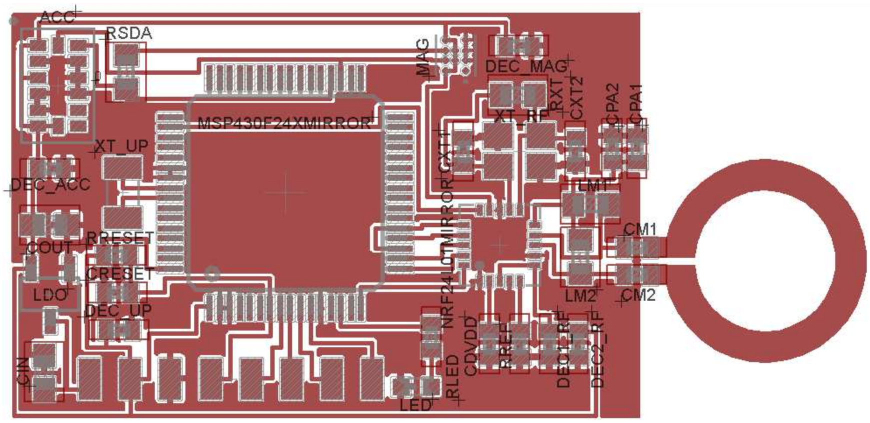


*Figure 3.14: Fourth generation base circuit flexible board design for separate UTCP devices.*

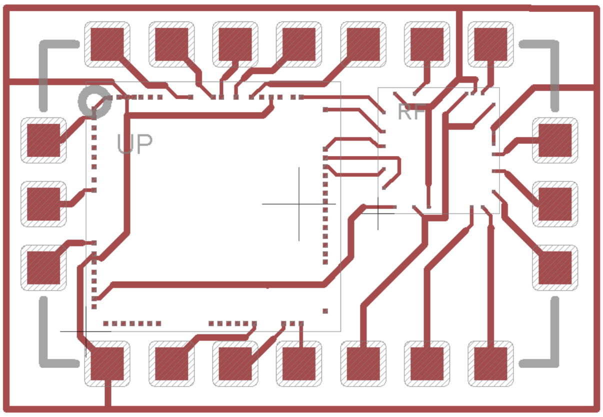


*Figure 3.15: Flexible board combining both UTCP devices.*

contact pads are actually interconnected as these are also not needed for the correct functionality of the sensor node.

Figure 3.16 displays the base circuit flexible board design compatible with the combined UTCP design from figure 3.15.

## 3.3 Network Protocol

The sensor nodes of the inertial tracking system will need to transmit the captured sensor data to a base station connected to a computer for further

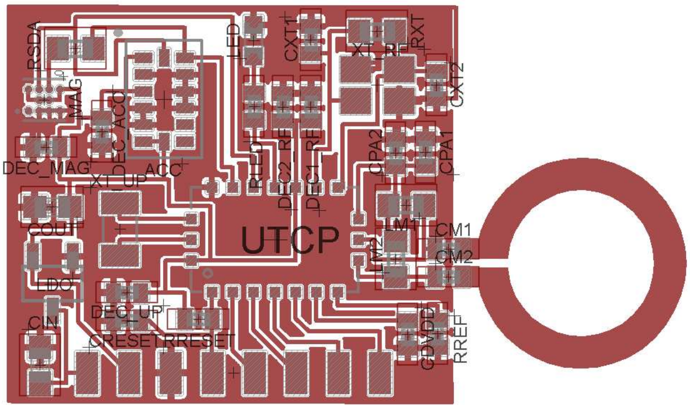


*Figure 3.16: Fourth generation base circuit flexible board design for combined UTCP devices.*

processing. As in every communication system, a certain protocol should determine when each node is allowed to transmit information and what information should be transferred in which order. Otherwise, the result is total chaos where nodes disturb each other's communication with the base station and none of the received information can be processed.

In this section, a wireless ad hoc network protocol is presented that allows simultaneous operation of many inertial sensor nodes. First the requirements for the protocol are discussed and some conclusions are drawn about the features that the protocol should support. Then, a framework is chosen for the protocol determining general functionality and applied principles. Afterwards, the protocol itself is introduced step by step, starting with the two possible roles nodes can take on: master or slave. Dynamic functionality where nodes can change their behaviour is introduced next, followed by the description of additional control mechanisms that increase robustness. The description is concluded by the base station functionality and some possible measures to increase the supported node count. Finally, the current consumption of the nodes is analysed during events coupled to the protocol and the performance is rounded up.

Note that the protocol was first designed for the third generation MFG type nodes. Hence the entire protocol description is based on this tracking system. At the end of the section, the adjustments are discussed that allowed the implementation of the same protocol on the second generation sensor system and on the third generation MARG type nodes.

### 3.3.1 Protocol Requirements

Inertial motion tracking of persons requires several sensor nodes to be attached to the body [35]. Each of these nodes will need to deliver sensor data at a regular rate to allow full human body posture tracking. Visualisation and further processing of the data is carried out by a computer which will obtain the information from a base station. As all of the nodes are clustered onto the body, the distance from each of the nodes to the base station is approximately equal no matter the posture of the traced person. When the person moves out of the reach of the base station, all nodes will loose the ability to communicate almost simultaneously.

In order to provide full body motion tracking, a human body model is adopted using 15 interconnected segments [12]. This implies that at least 15 nodes should be able to function simultaneously. Furthermore, a sample frequency of 100 Hz is chosen to allow smooth and accurate tracking. All of the nodes must consequently transmit their data within a 10 ms timeframe.

When motion capturing is performed in realtime, delay is considered as an important aspect and should always be minimised. Therefore, latency within the network should be close to zero. Combining this with a relatively high sampling rate where data will quickly become outdated, leads to the conclusion that retransmission of lost data will not be needed as new data will already be available when a retransmit is requested. This way, nodes can concentrate on the present data and no acknowledgments or retransmission requests need to be transmitted by the base station. The result is that the base station will purely function as a receiver, no data is transmitted from the base station.

The mainly fast data driven nature of the network shows the need for a robust network protocol. Malfunctioning nodes should not result in a collapsing network or hinder other nodes in their communication with the base station. Whenever a node is activated, data transmission should start as soon as possible and nodes should adapt to the state of the network to insure data integrity. Moreover, nodes should be able to determine their place in the network and detect problematic situations based on own observations. The use of metadata should be avoided due to the fact that this approach requires additional information to be transmitted by the nodes or even the base station, which in turn implies that less time is available and more power is consumed.

The above statements clearly imply that the network will adopt a star topology where data is only transmitted directly from the nodes to the base station. Although shielding by the human body can lead to unexpected packet loss in non LoS situations, multi-hop communication will not provide any benefit due to the fact that new data is generated regularly. If data is to travel via a multi-hop path, chances are that new data has already reached the base station making this older data obsolete. Furthermore, since no diagnostic metadata will be used, nodes can never determine if their data actually reaches the base station. A better approach to avoid unexpected package loss due to shielding consists of placing additional base stations on either side of the tracked subject.

### 3.3.2 Protocol Framework

Wireless communication will contribute most to the total power consumption of the sensor nodes. The presented protocol allows each of the components in the circuit to remain in sleep mode as long as possible, maximizing the battery life. Care has been taken to implement the protocol using an interrupt driven approach.

The custom designed wireless protocol is based upon the principle of Time Division Multiple Access (TDMA), where data transmissions are separated in time to avoid collision [36]. Nodes are assigned a certain timeslot in which they are allowed to transmit freely without interrupting the data transmission of any other node. Each timeslot is assigned a number in the temporal order in which they appear. The length of the timeslots is chosen in such a way that it allows a node to transmit the data and the base station to process it. Furthermore, a small amount of time is added to avoid overlap due to slight clock differences originating from the tolerance on the watch crystal frequency. Given the 20 ppm accuracy of the crystals and the 10 ms timeframe length, the worst case drift per frame amounts to 0.4 μs.

Using a time driven approach however, requires the presence of some sort of time reference. To this extend, a hierarchy is introduced involving a master and several slaves. The master node will provide all of the slaves with a synchronising point to which they are able to time their transmission. It is important to note that the master and slaves all consist of exactly the same hardware and that the difference is only noticeable in software.

In order to be able to identify a certain node in the network, each of the nodes is assigned a unique Identification (ID) number. This ID is also programmed within a separate part of the flash memory of the microcontroller, allowing the node to be reprogrammed without changing this fixed sensor node ID. It is also visible on the pictures of the sensor nodes shown in figures 3.4, 3.6 and 3.7 from section 3.2 as a label with this ID is applied to each node.

The data package transmitted by each of the MFG nodes consists of 11 bytes as can be seen in figure 3.17. The first byte depends on the current function of the node in the network, if it is the master, the byte is a package counter, if it is a slave, the byte instead equals the number of the timeslot that is used by the node. The second byte is the ID that has been assigned to the node. Then three bytes follow with respectively X, Y and Z accelerometer output data in 2's complement form [37]. The final six bytes contain the output of the magnetometer as three fixed point digital sequences of two bytes. More information on the sensor output data formats is given in section 3.4.

| Master: Packet Counter<br>Slave: Timeslot Number | Node ID | Accelerometer Data | Magnetometer Data |
|---|---|---|---|
| 1 Byte | 1 Byte | 3 Bytes | 6 Bytes |

*Figure 3.17: Contents of the RF data package. The master sends a packet counter, while slaves send their timeslot number.*

The base station combines all of the received RF data packages from the sensor nodes in a single large package for transmission to the computer. The structure of this package is displayed in figure 3.18. In total, ten bytes per active node and two additional counter bytes are transmitted. Each node adds an ID byte, three accelerometer output bytes and six bytes corresponding to the magnetometer output. The package is concluded with the counter byte found in the master package and an additional overflow counter that is maintained on the base station.

| Master ID + Sensor Data | Slave 1 ID + Sensor Data | ... | Slave x ID + Sensor Data | Master and BS Counter |
|---|---|---|---|---|
| 10 Bytes | 10 Bytes | | 10 Bytes | 2 Bytes |

*Figure 3.18: Contents of the base station data package.*

### 3.3.3 Master Operation

The implementation of the master sensor node operation is fairly straightforward. A flowchart illustrating its functionality is given in figure 3.19.

At startup, an initialisation phase sets up each of the components in the circuit and prepares them for the first acquisition and transmission of data. The output data rate of the accelerometer is set to 100 Hz and its range to 2 g as it is expected to measure only gravity. The magnetometer's initialisation coils are activated in order to avoid saturation of the sensing device, the factory calibration is acquired and an offset measurement is completed to set the range of the sensor. The RF channel is set in the transceiver, functionality is set to *Shockburst* at 250 kbps and all of the registers are configured to use two address bytes and a single byte Cyclic Redundancy Check (CRC) for packet filtering. For the microcontroller, interfaces are configured to match

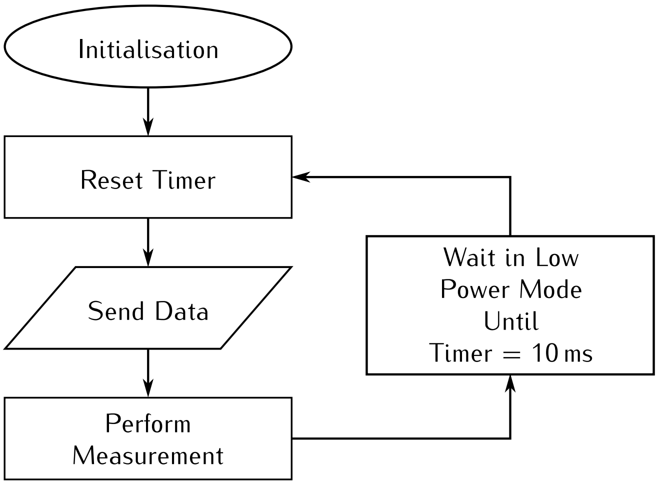


*Figure 3.19: Flowchart of the master sensor node operation.*

the peripheral devices and unused parts are disabled. Furthermore, one of the internal timers is set to a 10 ms period and programmed to generate a time-out interrupt on each completed cycle. It is driven by a clock signal that is generated by the external watch crystal present on the sensor node board and thus provides a stable reference.

Afterwards, the master starts an infinite program loop where, in each iteration, data is collected from the sensors and transmitted to the base station via the wireless interface. When both steps are completed, the sensor node is put in sleep mode, conserving as much power as possible. The only component that is kept active during the sleep period is the accelerometer due to its long turn-on time of 30 ms. Given the timer setup, the node will wake up at a rate of 100 Hz to repeat the cycle.

### 3.3.4 Slave Operation

The slave sensor node operation is more complex because in this case the transmission of data must be synchronised with the master. Figure 3.20 depicts a flowchart representing the slave node implementation.

The initialisation procedure mostly sets up the components as the master node does, yet there is one important difference. Two different time-out interrupts are associated with the 10 ms timer driven by the watch crystal. One which corresponds to the timeslot that has been assigned to the slave and another is set to approximately 9 ms. This second interrupt will allow the slave to wake-up on time for packet reception from the master.

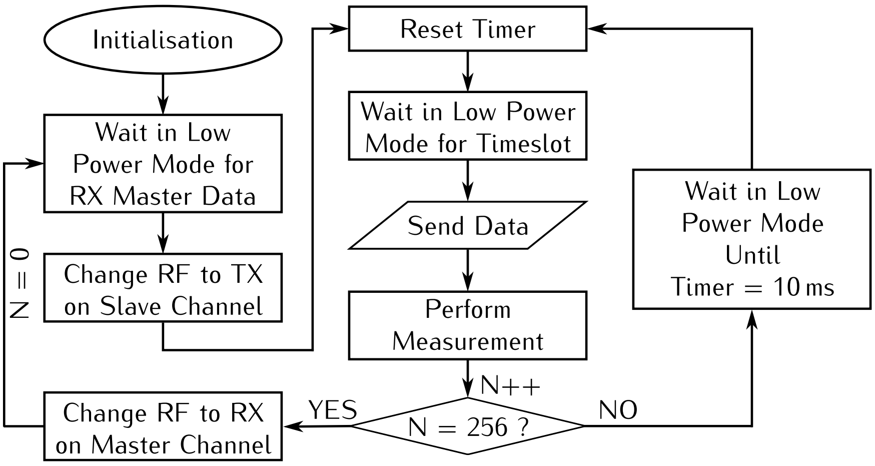

Initialisation
Wait in Low Power Mode for RX Master Data
Change RF to TX on Slave Channel
N = 0
Change RF to RX on Master Channel
Reset Timer
Wait in Low Power Mode for Timeslot
Send Data
Perform Measurement
N++
N = 256 ?
YES
NO
Wait in Low Power Mode Until Timer = 10 ms

*Figure 3.20: Flowchart of a slave sensor node operation.*

After initialisation, the RF transceiver is configured for reception of a packet from the master. In order to allow easy detection if a package originates from a slave or from the master, slaves utilise a different RF channel for data transmission. This way, slaves can simply listen on the master channel and synchronise to any packet reception without having to unpack or even read out the received package. Once a packet is detected, the RF transceiver operation is switched to transmit mode and its frequency is set to the slave RF channel. Also, the timer is forced to reset to complete the synchronisation process. The slave is then put in sleep mode only to be awoken by the timer interrupt when its timeslot is reached. At this point, data is transmitted wirelessly to the base station and new data is acquired from the sensors.

The above procedure could easily be repeated in an infinite loop to obtain a working TDMA-like master slave protocol. However, this approach would incur two important drawbacks. Firstly, a slave would have to activate the frontend of its transceiver twice every cycle: once to receive a synchronising package from the master and a second time to transmit its own data. Since the RF transceiver is clearly the biggest contributor to the power consumption of a sensor node, this would result in slaves that consume approximately twice as much as the master. Secondly, whenever a slave would miss a package from the master due to e.g. environmental effects, it would also not transmit a package of its own. This would in turn result in a higher packet loss for slaves.

The suggested approach, which is also illustrated in figure 3.20, involves reducing the number of times a slave will synchronise with the master to only once every 256 transmitted packages. When no synchronisation is performed, the node will rely on the internal timer driven by the local watch crystal to provide the interrupts at the appropriate time. A result of this approach is that power consumption in slave nodes will drastically be reduced and that the transmission of 256 subsequent packages is guaranteed with each successful synchronisation. Note however, that the worst case clock drift increases to 0.1024 ms since it must also be multiplied by the same amount.

### 3.3.5 Dynamic Implementation

The protocol as it has been proposed so far still suffers from one major flaw. It depends fully on the existence and the correct functionality of the master node. Once this node malfunctions or its battery runs out, the whole network collapses as no more synchronisation is present. In order to avoid this catastrophic situation, nodes must be able to determine their role in the network at runtime and react to sudden changes when needed. By adding this functionality, the network attains a plug-and-play nature where nodes can be switched on or off at the user's discretion. A flowchart illustrating this dynamic implementation is displayed in figure 3.21.

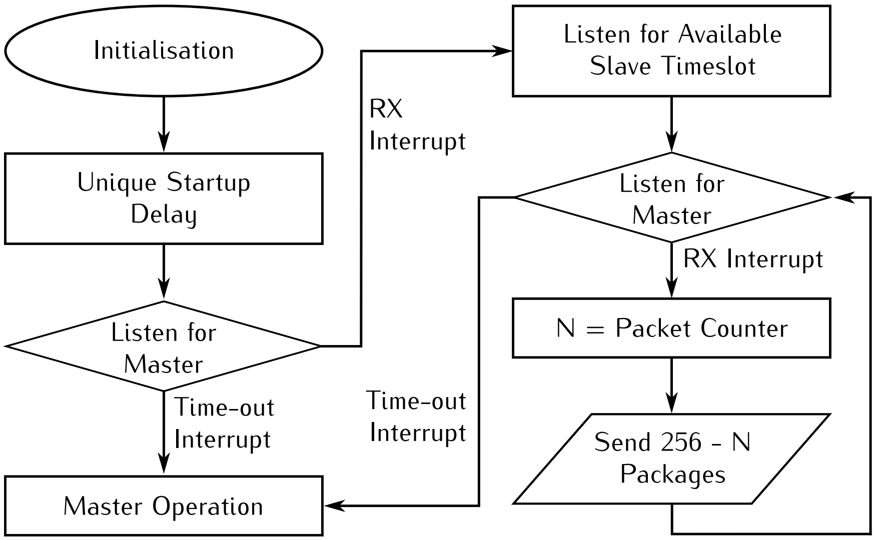


*Figure 3.21: Flowchart of a dynamic node operation.*

An important part of the startup phase is the unique startup delay. This delay ensures good functionality of the protocol when multiple nodes are powered on at the same time. This situation can easily arise if these nodes share the same power supply. In the initialisation phase, a timer is configured to deliver a time-out after approximately 90 ms. This fixed delay will then be repeated a number of times equal to the sensor node's ID. The individual timespan is chosen such that a node can complete the required steps up until the transmission of its first package as a master. This way, when multiple nodes are powered at the same time, only the node with the lowest assigned ID will become master.

After the initial delay, a node will figure out if a master is present in the network. To accomplish this, the RF transceiver must be set to receive data in the master's RF frequency channel. At the same time, a timer is started, programmed to trigger a time-out interrupt after 60 ms. This means that, if a master is present, the node can receive a total of six RF packages before the timer generates the time-out interrupt. If indeed a package is received, this will trigger a different interrupt indicating that the node should become a slave. Otherwise, the time-out interrupt will eventually occur and the node will start acting as the master within the network. In this case it will operate according to the flowchart depicted in figure 3.19.

If the node has detected a master within the network, it should now determine which timeslot it will use for wireless communication. It switches the RF transceiver to receive mode in the slave channel and sets a timer to trigger an interrupt after 60 ms. Until this time-out occurs, the slave will now process every received package to determine which timeslots are already in use. This can be accomplished given the fact that the timeslot number is also included in the data package as can be seen in figure 3.17. When the time-out finally occurs, the node will choose the lowest available timeslot and start transmitting data of its own according to the scheme displayed in figure 3.20.

There is however one major change in the behaviour of a slave in the dynamic implementation. Whenever a slave needs to synchronise with the master, it starts a timer which will generate a time-out interrupt. This time-out period however, depends on the timeslot that the slave is using according to the following formula:

$$Period\,[\mathrm{ms}] = 40\,\mathrm{ms} + Timeslot * 20\,\mathrm{ms}. \tag{3.1}$$

If the master is still alive, the slaves will be awoken by an interrupt generated by the RF transceiver chip, meaning that they can continue their operation as a slave within the assigned timeslot. However, if the master has encountered a problem, the slave node with the lowest timeslot number will receive a time-out interrupt before all the other slaves will. This slave must then reconfigure itself as the new master and start operating according to the flowchart depicted in figure 3.19. The other slaves will receive a package from this new master node and continue transmitting packages in their timeslot as if no changes have occurred in the network. Due to the fact that the time-out period increases with increasing timeslot number, only the slave with the lowest timeslot number will eventually take over the master role.

As in the startup procedure, the slave using the first timeslot will only decide to become a master if at least six subsequent packages from the master are lost. Each of the following nodes will require two additional dropped packages compared to the preceding node. If a slave becomes master, the subsequent slave must receive at least one of the first two packets sent out by the new master, otherwise, this slave will also decide to take up the master role. A solution to this situation will be discussed in section 3.3.6.

The system that has been described in the above is only able to work if one important condition is met: all of the slaves must synchronise with the master at the same time. If this is not the case, time shifts between synchronisations could easily result in multiple masters. To avoid this issue, synchronisation will be executed according to a packet counter. As can be seen in figure 3.17, the master transmits a single byte package counter. This modulo 256 counter is unpacked by the slaves and locally stored with each synchronisation. Each time an RF packet is transmitted by a slave, the local counter is incremented and whenever it resets to zero, the slave will synchronise with the master. As all of the slaves keep their counter aligned with the master's counter, all of them will perform the synchronisations at the same time.

### 3.3.6 Additional Control Mechanisms

By introducing dynamic features in the protocol, the system has clearly improved in robustness as the failure of one node no longer means that the whole network collapses. However, up until this point, no backup mechanisms are present to correct possible anomalies in the network. As already stated before, multiple masters could be active at the same time due to unexpected package loss. Since RF communication is highly susceptible to environmental conditions, this scenario must be taken into account.

When looking at the master operation depicted in figure 3.19, it is clear that the master node will blindly transmit data packages to the base station. This implies that multiple active masters would never notice each other and that the user would have to intervene by switching off the redundant nodes.

Solving this problem requires the master to scan for other possible active masters in the surroundings. From time to time, a master will listen in the master RF channel instead of transmitting a package. If a package is received before a time-out interrupt is generated after 10 ms, the master will reconfigure itself as a slave and look for an available timeslot. Otherwise, the master will continue its normal operation in the following timeframe.

Not only the master suffers from this problem though. Analysing the operation of the slaves depicted in figure 3.20 more thoroughly shows that slaves will never notice other slaves transmitting within the same timeslot. A similar procedure as has been described above is used to avoid this issue.

These conflict checks must however be performed on a random base, avoiding the situation where all conflicting nodes are listening for each other at the same time. Therefore, each node will randomly choose a period between 5 and 13 s to perform the conflict check. After each check, the period is again randomised.

Note that nodes will deliberately skip transmission of a package while performing a conflict check. This of course results in loss of packets at the base station side, yet, due to the chosen intervals, it is restricted to a maximum of 0.2 %. Not a large cost considering the highly valued gain in robustness.

### 3.3.7 Base Station Operation

Figure 3.22 depicts a flowchart representing the base station operation. In the initialisation step, the base station is prepared for reception and retransmission of data. The RF transceiver is configured for reception of data on the master channel with a two byte address and a single byte CRC. Each of the other peripheral interface devices are initialised to retransmit data from the microcontroller over their supported connection. On the microcontroller, the digital interfaces are set up to match the abilities of the peripherals and a timer based on the watch crystal clock is initialised with a 10 ms period and an interrupt after 8 ms. Afterwards, the base station is put in sleep mode until data is received from the master node. Upon reception, the timer is reset for synchronisation, the data is read from the RF chip and stored in an array. Then, the RF transceiver is reconfigured for reception of data in the slave channel and an infinite loop starts where the base station waits for data and reads it out upon reception. At a certain point, the timer interrupt will break the loop operation and the base station transmits all of the received sensor data to either of its interfaces connected to the computer. Finally, the RF chip is again configured to receive data in the master channel and the cycle starts over again.

Note that the base station operation is independent of the number of slaves at any point in the network. Failing or out of reach slaves will simply keep the base station waiting in low power mode during their timeslot.

Although this flow of working does work well, it is particularly vulnerable to a failing master node. Since the entire cycle of operation depends on the success of master node data reception, a failing master could jam the base station. However, since this is carefully monitored by several controlling mechanisms in the sensor node operation, this problem is very unlikely to occur. A disadvantage that still remains is the fact that when the master

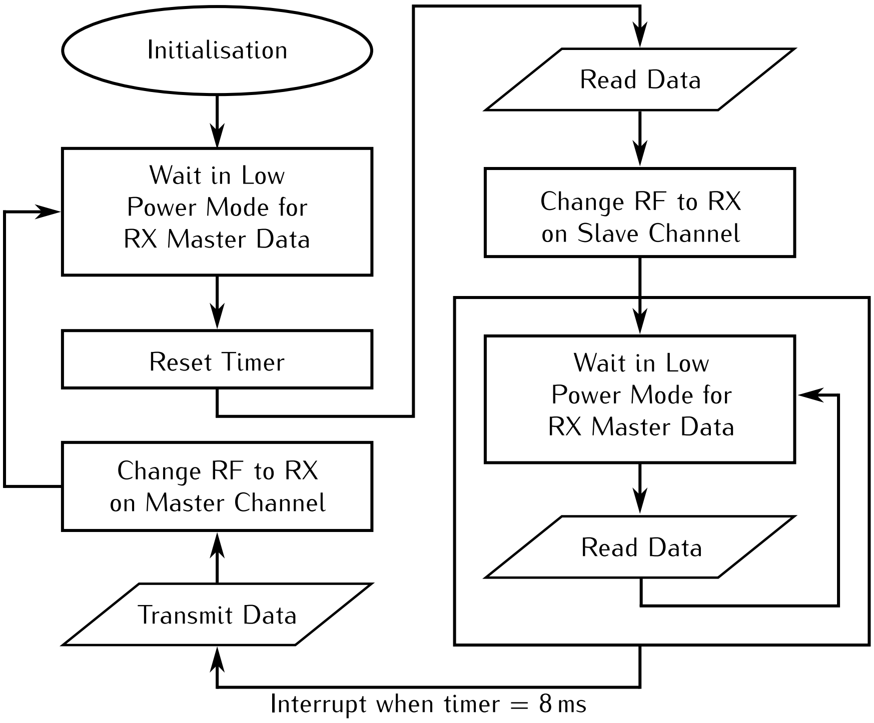

Initialisation
Wait in Low Power Mode for RX Master Data
Reset Timer
Change RF to RX on Master Channel
Transmit Data
Read Data
Change RF to RX on Slave Channel
Wait in Low Power Mode for RX Master Data
Read Data
Interrupt when timer = 8 ms

*Figure 3.22: Flowchart of the base station operation.*

packet is not received due to changes in the environment, no slave data will be received either.

### 3.3.8 Extending Node Count

The number of nodes that can operate within the network is currently limited by the available number of timeslots. However, methods are available to extend the node count by making better use of the available hardware in the base station or by adding additional hardware.

As mentioned in section 3.2.3.1, the RF transceiver used in the third generation hardware design features a dual receiver functionality. Activating this feature when receiving in the slave channel allows for additional slaves transmitting in the second receive channel in the exact same timeslots as the other slaves. These slaves must also synchronise their transmission to the same master as the other slaves do, and implement the same functionality as described in the previous sections. At startup, nodes will now start by looking for an active master. If this is the case, they will first scan for an available timeslot in the lowest RF channel. If all of the timeslots are in use, they will change to the second RF channel and scan for an available timeslot there. This method almost doubles the amount of nodes that can be active within the network.

Another way to allow more nodes into the network is to add extra base stations. Each base station can be programmed to utilize different RF channels for communication. By doing so, multiple distinct networks will be created. Nodes will have to start by scanning the first network for an available spot. If this is not the case, they can move on to the next network. Adding extra nodes is now just a matter of providing additional hardware and supplying enough processing power to use the measured data.

### 3.3.9 Node Current Consumption

The current consumption of sensor nodes is analysed during several possible situations: normal master and slave operation, dynamic reconfiguring of a slave to a master and collision detection. Each of the graphs given in this section was obtained by measuring the voltage across a small resistor placed in the feedline of the battery with an oscilloscope.

#### 3.3.9.1 Master Node

Figure 3.23 shows a time graph of the current consumption in the master node during one frame period. Only a single node acting as the master was active during measurement. Indicated on the graph are several clearly discernible regions where the current consumption can be associated to functionality of a certain component. The large peak with a duration of 0.85 ms and an absolute maximal value of 21.5 mA corresponds to the RF transceiver sending a package. Following the peak is a 3 ms period where a steady current consumption of about 3.75 mA is visible. This plateau is a result of the magnetometer performing a new measurement after being activated by the microcontroller. In the final part, lasting 6.15 ms and where an approximate current of 0.75 mA is consumed, all components are kept in low-power mode, except for the accelerometer as its wake-up period is too long for the application at hand. The resulting time period clearly matches the required frame length of 10 ms to obtain a 100 Hz sampling rate. In normal operation, the average current consumption of the master was measured to be below 3 mA. This means that with the batteries that are currently used [18], a lifetime of approximately 2 weeks can be achieved.

#### 3.3.9.2 Slave Nodes

The current consumption in several slave nodes and one master is displayed as a time graph in figure 3.24. Five nodes were used simultaneously during this

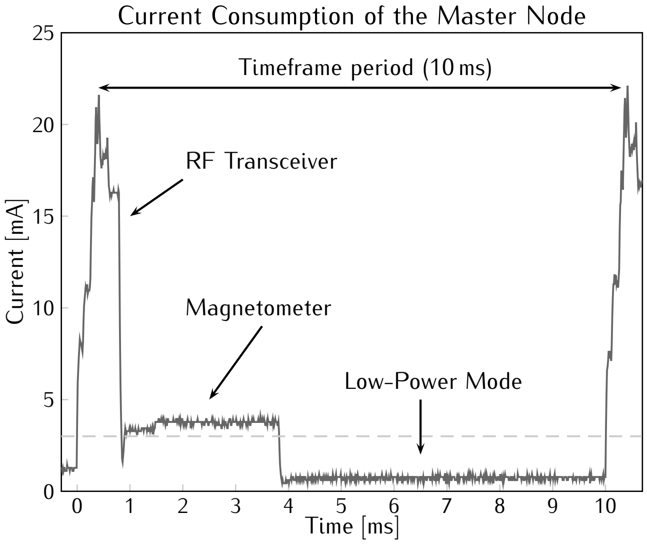


*Figure 3.23: Time graph of the current consumption in the master node during one timeframe period. The dashed line indicates the average consumption over time.*

experiment. The slave indices in the graph's legend indicate which timeslot is used for RF transmission. All of the transmission peaks of the different slaves are clearly separated in time as outlined by the protocol. In the center of the graph, an additional peak with a value and duration of respectively 28.5 mA and 0.9 ms is present in each of the slave nodes current consumption. This peak is aligned to the transmission of a master packet and is due to the slave's RF transceiver receiving this packet for synchronisation purposes. As slaves will only synchronise with the master every 256 packets, this extra receive peak will only add about 10 µA to the average current consumption of the slaves in respect to the master's consumption.

Note that the current consumption in the slaves seems higher on the graphs due to the fact that the LED present on the board has been switched on. Slaves will blink this LED a number of times equal to their timeslot number just after synchronising with the master. Each blink will add about 21 µA to the average current consumption. Also note that the master will blink the LED continuously yet slower in order to provide visual feedback of the system state. This consumes about 170 µA and has already been included in the 3 mA average current consumption.

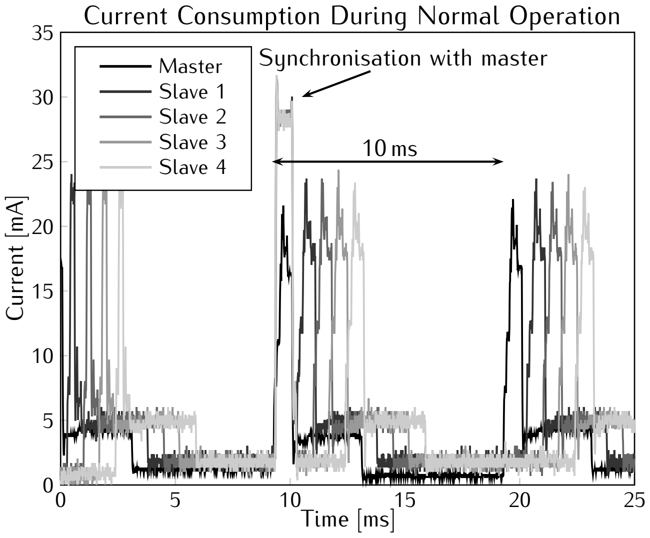


*Figure 3.24: Time graph of the current consumption during normal operation of master and slave nodes.*

#### 3.3.9.3 Dynamic Operation

Figure 3.25 depicts a time graph of the current consumption when a slave takes over the master role. Two other slaves are active when the master suddenly fails. When the counters in the slave nodes are reset, a synchronisation action is initiated and all slaves begin to listen for a package from the master. According to (3.1), the node using timeslot 1 receives a time-out after 60 ms, which is also indicated in the graph. At that point the slave decides to take over the master role and immediately transmits an RF package. This transmission is also visible in the graph as a very small peak. An other active slave using a timeslot with a higher number will pick up this master packet and resume normal operation afterwards. From this node's point of view, nothing has changed in the network, it still fulfills the same role.

#### 3.3.9.4 Collision Detection

The time graph in figure 3.26 represents the current consumption in the event of multiple active masters. In order to trigger this condition, one node has been programmed to simply assume the master role at startup and never even check if another master is present. After activating three other nodes, of which only one becomes the master, this additional node was powered and the role change in the master was observed. According to the wireless protocol, a master will

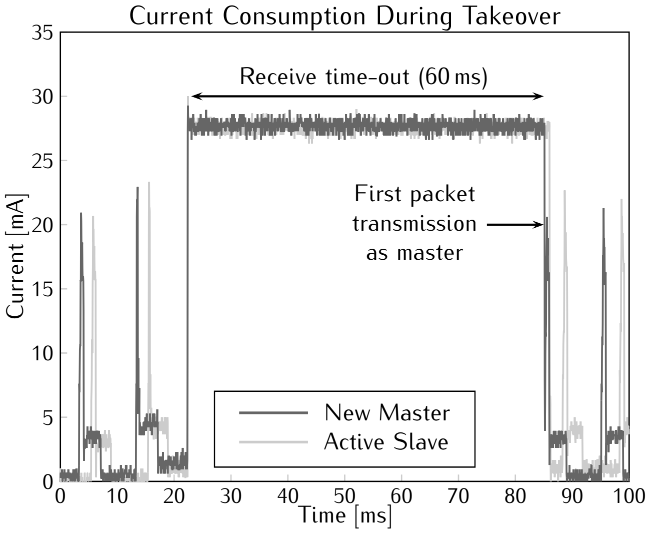


*Figure 3.25: Time graph of the current consumption in two slaves in the event that one of the slaves takes over the master role after a time-out occurs.*

check on a random basis if another master is present in the network. This event occurs in the graph and after approximately 5 ms another master is detected. The node will then switch to the slave RF channel and start scanning for an available timeslot. This procedure lasts about 60 ms and several spikes are visible where data from an active slave is received. When all reconfiguration is completed, a free timeslot is chosen and the new slave starts by synchronising its transmissions with the other master. Note that the spikes caused by data reception from active slaves in the first two timeslots are not synchronised with the active master, but with the reconfiguring master as they appear before the active master's transmit peaks. The synchronisation with the active master will only occur when the internal counter of the slaves is reset.

A similar timegraph of the current consumption where a slave detects a conflicting node operating in the same timeslot is displayed in figure 3.27. To trigger the requested situation, one node was programmed specifically to become a slave using the first timeslot without checking for conflicts or availability. The normal slave goes through the reconfiguration procedure after detecting the conflict and finds a different available timeslot. It starts operating in the new timeslot by synchronising its transmission with the master and transmits its first packet in the new timeslot 92 ms after the collision detection. Note that the collision detection only takes place at the very end of the conflict detection period as the receiver is not yet fully activated when

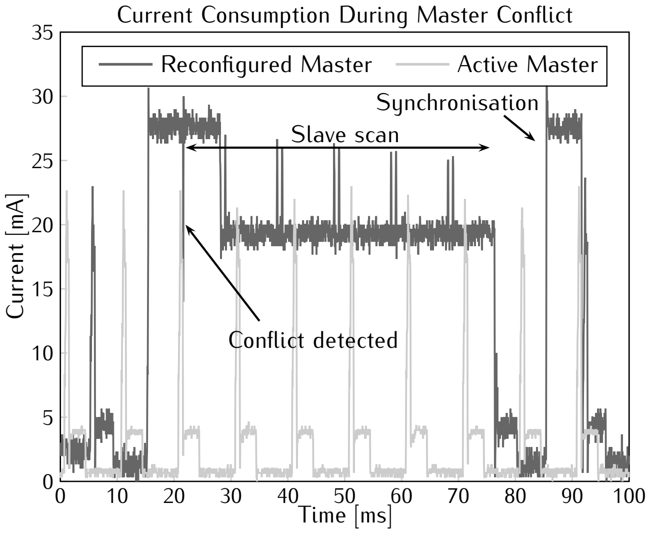


*Figure 3.26: Time graph of the current consumption in two conflicting master nodes in the event that one of them detects a conflict and reconfigures itself as a slave.*

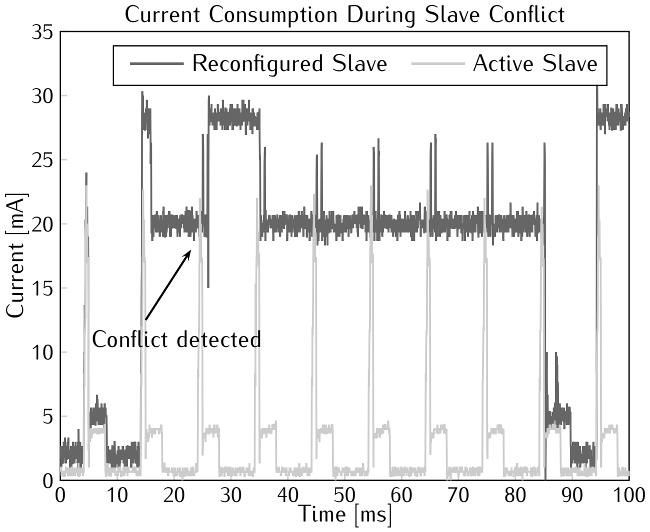


*Figure 3.27: Time graph of the current consumption in two conflicting slave nodes in the event that one of them detects a conflict and reconfigures itself to use a different timeslot.*

the conflicting slave transmits a package.

The extra conflict detection mechanism also adds to the current consumption of the nodes. However, as this check is performed on a random basis, no fixed value can be associated with it. Activating the receiver will result in a current consumption peak with a duration of 10 ms and a value of 26.5 mA. Detection is performed every 5 to 13 s and will thus add anything between 53 µA and 20 µA to the average current consumption of a node.

### 3.3.10 Protocol Performance

The graphs in Figures 3.23 and 3.24 allow to estimate the maximum number of nodes that would be able to operate within the available 10 ms timeframe. However, extra time within this timeframe has to be reserved to allow the base station to fulfill its entire function, i.e. the transmission of the received data to a computer via one of the available interfaces. Tests have been executed with varying timeslot lengths to find the minimal allowed period required to avoid data loss or collision. The results showed that with a timeslot length of 0.85 ms only sporadically packets are lost, indicating that these losses can be attributed to the inherent sensitivity of wireless communication to the environment. Note that this also indicates that the effect of clock drift is included in this period, since bursts of packet loss would otherwise occur. Furthermore, the maximal number of simultaneously active slaves has been determined. With nine slaves active in each of the two available RF channels, the base station is still capable of communicating all of the data through within the remaining timespan. This means that a total of 19 nodes (one master and eighteen slaves) are capable of communicating at a rate of 100 samples per second with a single base station.

Given the very application specific nature of the protocol, only a comparison with wireless protocols used in other orientation tracking systems is relevant. A maximum of 32 commercially available wireless sensor nodes by XSens [38] can communicate at a rate of 20 Hz with a single base station. When reducing the amount of nodes to 12, 6 and 1, the datarate can be increased to respectively 50 Hz, 75 Hz and 120 Hz. The Orient system [39] designed by the University of Edinburgh supports the use of 15 nodes at a rate of 64 Hz. Clearly, the protocol presented here performs better than both of these systems.

### 3.3.11 Alternative Implementations

The described protocol was also implemented on other sensor node systems. However, the restrictions and required functionality of these systems needs to be taken into account when porting the protocol.

#### 3.3.11.1 Third Generation MARG Nodes

As the MARG nodes contain the same platform as the MFG nodes, the required adjustments are limited. Actually the only difference originates in the presence of the gyroscope. The data of the gyroscope consists of six additional bytes that need to be transferred to the base station. These bytes are added to the data package of figure 3.17 and enlarge it to a size of 15 bytes. The result is that the used timeslot size will also need to be increased as more data is transferred and needs to be processed on the base station side. This in turn leads to less nodes that can function with a single base station.

As with the MFG case, tests have indicated that a timeslot length of 1.12 ms leads to occasional package loss. The result is that only seven slaves can be active at the same time in each RF channel. Luckily, this results in a total of 15 simultaneously active nodes for each base station, which is exactly the amount that was put up front in the requirements.

The current consumption graph of MARG nodes is very similar to those given in section 3.3.9. The only difference is an offset current consumed by the gyroscope which amounts to approximately 6.5 mA. As the 50 ms turn-on time of this component also exceeds the timeframe length, it is never put in sleep mode. Note that the average current consumption of a MARG sensor node turns out to be more than three times the consumption of MFG nodes. Hence, with the use of the same battery as mentioned before [18], a lifetime of only four and a half day can be reached.

#### 3.3.11.2 Second Generation

An important drawback of the second generation nodes is that the chosen RF transceiver does not offer the dual receiver feature. Hence, only one slave channel can be used and the number of nodes per base station will be lower. Tests have shown that the timeslot lengths determined for the third generation can be reused for this generation. Therefore, the maximum number of supported nodes per base station equals eight when the nodes transmit all of their information and ten when the gyroscope data is omitted.

Figure 3.28 shows a time graph of the current consumption in a second generation node operating as a master. Note that it concerns a node where the gyroscopes were not assembled to the board. The general form of the curve resembles the third generation current consumption graph of figure 3.23, yet the overall average consumption lies somewhat higher at about 8 mA. A fully assembled MARG node consumes an extra fixed current of approximately 17.5 mA due to both active gyroscopes. This translates in a lifetime of five days for an MFG node powered by the Varta PoliFlex battery [18], while a MARG node only lasts two and a half days with the same power source.

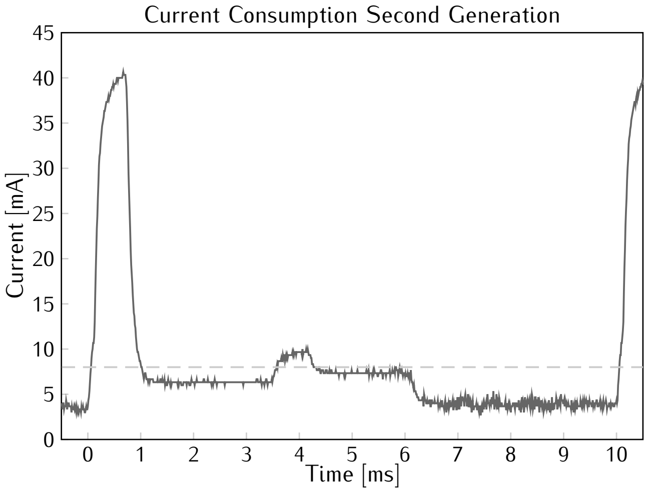


*Figure 3.28: Time graph of the current consumption in a second generation master node without gyroscopes. The dashed line indicates the average consumption over time.*

## 3.4 Firmware Filter Implementation

When a large amount of sensors are active simultaneously, lots of data is generated and delivered to the backend computer. The result is that this computer is no longer capable of calculating the orientation of each of these nodes within the restricted timeframe using the filters described in section 2.4. The solution is to divide the workload of orientation tracking to the available processing power in the system, namely the microcontrollers.

Transferring the implementation of the filter from PC software to embedded firmware is however not a simple copying task. The microcontrollers are not equipped with the same amount of processing power that modern computers have available. While modern everyday computers run at a clock frequency of several GHz, contain large quantities of memory and feature multiple processing cores capable of handling 32 bit or even 64 bit operations in parallel, a low power microcontroller is much more restricted in its resources.

In this section, the different steps in the implementation of the orientation filter in firmware for the third generation nodes is described. The focus is on the MFG nodes, yet an extension for the MARG nodes is also discussed. The third generation is chosen over the second generation since the selected TI microcontroller features 16 bit operations and a hardware multiplier while the Atmel is only an 8 bit device.

First, the filter is chosen that will be implemented in firmware. Next, a fixed point number format is defined that will be used to represent the real numbers encountered in the filter implementation and sensor output processing in the embedded version is discussed. Then the actual Kalman filter algorithm is converted and a number of optimisations that reduce the number of operations to an absolute minimum are presented. Afterwards, the data package format is shown and the resulting impact of the embedded calculations on the current consumption is demonstrated. Finally, some solutions are suggested to possible issues for embedding of a MARG type filter.

### 3.4.1 Filter Choice

Choosing one of the filters from section 2.4 for implementation in firmware is mostly based on two important criteria: the number of operations and the type of operations used in the algorithm. The filter with the least amount of operations, the Euler type EKF, has already been ruled out in chapter 2 due to its slow step response. An SPKF mostly requires more operations than an EKF, yet the state dimension is different for both representation types. However, in the end the trigonometric functions associated with Euler angles have to be taken into account, as well as the gimbal lock singularity. A quick test showed that calculating a cosine requires approximately 200 times more time than a single multiplication. Therefore, the quaternion EKF is finally chosen as the candidate for embedded implementation.

### 3.4.2 Fixed Point Notation

On modern computers, real numbers are mostly represented by a floating point system. A number is then approximated by a significant, scaled by a certain exponent [40]:

$$Number \ \approx \ Significant \ \times Base^{Exponent} \tag{3.2}$$

The term *floating* refers to fact that the location of the digital point can be anywhere within the significant of the sequence. The advantage of this type of representation lies in the high dynamic range that is obtained. In other words, the distance between two subsequent numbers is coupled to the magnitude of these numbers. The downside is the execution time and complexity of mathematical operations, which is a direct result of the very nature of the representation. Addition and subtraction cannot be executed using standard integer operations. The exponents must first be equal, then the significants can be added or subtracted and the final result is obtained. For multiplication, significants are multiplied, yet exponents must be summed and the result must then be normalised to conform to the notation.

An alternative to floating point numbers is found in the fixed point representation format. Contrast to the floating point format, the decimal point is now placed at a fixed place within the bit stream. A fixed point bitstream $\boldsymbol{b}$ is thus separated in two parts, the *integer* bits left and the *fractional* bits right of the decimal:

$$\boldsymbol{b} \;=\; b_1 b_2 b_3 \ldots b_{m-1} b_m \;.\; b_{m+1} b_{m+2} \ldots b_{n-1} b_n, \tag{3.3}$$

where $n$ is the number of bits in the sequence and $m < n$ is the number of integer bits. Similar to the integer case, the real number $x$ represented by an unsigned bitsequence can be reconstructed by adding up the corresponding powers of two:

$$x \;=\; \sum_{i=1}^{n} b_i \; 2^{n-m-i-1} \tag{3.4}$$

Due to this similarity with integer numbers, the summation of fixed point numbers does not require any additional function definitions. Multiplication is easily implemented using the integer multiplication followed by bitshift operations to validate the result. The disadvantage of fixed point formats lies in the fact that the accuracy and range fully depend on the chosen number of bits and the location of the decimal point. Therefore, the distance between two subsequent numbers is always equal to the power of two associated with the least significant bit, much like the distance between two integers is always 1. The result is a much lower dynamic range and higher rounding error with multiplications. Note that signed fixed point numbers also exist, the 2's complement formalism [37] is mostly used to achieve this.

Given the fact that many matrix operations need to be executed to implement the tracking filter, the speed of operations is of major importance for the choice of number format. Moreover, since none of the devices are equipped with a Floating Point Unit (FPU) providing fast and efficient floating point operations, the fixed point format is clearly the best option. This choice is also validated by tests showing that the execution time of a summation of two floats is about 100 times the execution time of the summation of two integers of the same bitlength.

#### 3.4.2.1 Format

When choosing the fixed point format, the range and precision form the important criteria that determine the size of the integer and fractional part. The total number of bits should preferably equal one of the standard integer lengths supported by the compiler, such as 16 for a short or 32 for a long. The maximal precision obtainable with a 16 bit fixed point number format is

```
#define BASE24   16777216
typedef long     fixed8_24;
```

*Code Segment 3.1: Fixed point format*

approximately $3 \times 10^{-5}$, when only a single integer bit is chosen. This is not low enough considering the recommended value for the covariance matrices in the quaternion version of the tracking filter. Moreover, in the entire filter calculation, numbers should then not exceed unity, which is highly unlikely. Therefore, a 32 bit length is put up front.

A good trade-off between a descent range and a fine precision is a choice of 24 fractional and 8 integer bits. Given the fact that signed numbers are used, this choice offers an integer range from $-128$ to 127 and a precision of $5.96 \times 10^{-8}$. In code, the two lines given in code segment 3.1 were added to reflect this choice. The *BASE24* variable equals $2^{24}$, which is the binary representation of unity in the fixed point format. The *typedef* defines a new type for the fixed point format which is stored as a *long*.

### 3.4.2.2 Multiplication

Fixed point multiplication is based on underlying integer multiplication followed by a bit shift operation to obtain a result in the same fixed point format. However, multiplication of bitsequences commonly results in overflow and the product might require a bitsequence of twice the arguments' length. Also, since many bits represent values below the decimal point, rounding will be needed as the result will contain bits far below the range of the chosen fixed point format.

Multiplication is an inherently more complex operation to perform on microcontrollers compared to summation. Therefore, many of them are equipped with a Hardware Multiplier (HWM) to shorten the execution time. Considering the fact that the MSP430 family only features a 16 bit HWM, the multiplication of fixed point numbers according to the chosen format will have to be executed in several steps. The results from each of the steps should then be added together in the appropriate manner. Code segment 3.2 shows the implementation.

Since the bitsequences are to be split in two for multiplication, problems occur when an operand represents a negative number. Therefore, this is first tested and negative numbers are negated to their positive counterpart. If only one of the operands is a negative number, a boolean is set to indicate that the result should also be negated afterwards. Both operands are now positive and can be multiplied in four subsequent multiplications using the HWM.

First, the most significant sixteen bits of each operand are multiplied.

```
//Multiply two fixed point numbers
fixed8_24 Mult8_24(fixed8_24 a, fixed8_24 b){

  //Declare return variable
  fixed8_24 RES_LSB = 0;
  fixed8_24 RES_MSB = 0;
  char negative = 0;

  //Sign trouble
  if (a < 0){
    if (b < 0){
      b = -b;
    }
    else{
      negative = 1;
    }
    a = -a;
  }
  else{
    if (b < 0){
      negative = 1;
      b = -b;
    }
  }

  //Use Hardware multiplier for 4 separate multiplications
  MPYS = a >> 16;
  OP2 = b >> 16;
  RES_MSB = RESHI << 8;
  RES_MSB += RESLO >> 8;
  RES_LSB = RESLO << 8;
  MPY = b >> 16;
  OP2 = a;
  MAC = a >> 16;
  OP2 = b;
  RES_LSB += RESHI << 8;
  RES_LSB += RESLO >> 8;
  RES_MSB += RESHI >> 8;
  MPY = a;
  OP2 = b;
  RES_LSB += RESHI >> 8;

  //Shift result into fixed point form
  RES_MSB = RES_MSB << 16;
  RES_MSB += RES_LSB;

  //Sign trouble
  if (negative == 1)
    RES_MSB = -RES_MSB;

  //Return result
  return RES_MSB;
}
```

*Code Segment 3.2: Fixed point multiplication function*

```
//Square root function by conversion to floating point
fixed8_24 Sqrt8_24(fixed8_24 x){

  //Convert to floating point
  float t = x;

  //Calculate sqrt
  t = sqrt(t/BASE24);

  //Convert back to fixed point representation
  x = (fixed8_24)(t * BASE24);

  //Return the result
  return x;
}
```

*Code Segment 3.3: Fixed point square root through floating point*

These bits actually form a fixed point number with eight integer and eight factional bits. The result consists of 32 bits, of which 16 are integer bits and the other 16 are fractional. Overflow occurs when one of the eight most significant bits of this result is not zero, since the result should be a positive number and the integer value should fall below the upper boundary of the fixed point representation. The second and third multiplication are between the 16 most significant bits of one operand and the 16 least of the other. Here, the multiply accumulate functionality can be used since both results are a fixed point number with only fractional bits. The last multiplication is between the least significant 16 bits of each operand and only the highest eight bits of this result are actually retained in the result as all others fall below the precision of the chosen fixed point format.

The three results are stored in two intermediate variables, one for each half. Note however that both of these variables are actually 32 bits in size to account for shift operations outside of the 16 bit range and to support possible overflow in the lowest part. The full result is finally shifted in place just before a possible negation of the result.

#### 3.4.2.3 Square Root

The quaternion EKF algorithm uses a couple of square root evaluations. Sensor data and quaternion output e.g. need to be normalised each step. Furthermore, square roots will also need to be calculated for the matrix inversion present in the Kalman filter algorithm. Unlike addition or multiplication, a square root operation is not linear and cannot be implemented directly. Several options exist for fixed point square root calculation.

The first and most straightforward option is shown in code segment 3.3.

```
//Constants
DEFINE  FRACBITS         24
DEFINE  ITERS     15+(FRACBITS>>1)

//Recursive square root of a fixed point number
fixed8_24 RecSqrt(fixed8_24 x){

  unsigned long root, remHi, remLo, testDiv, count;

  //Clear root
  root = 0;

  //Clear high part of partial remainder
  remHi = 0;

  //Get argument into low part of partial remainder
  remLo = x;

  //Load loop counter
  count = ITERS;
  do {

    //get 2 bits of arg
    remHi = (remHi << 2) | (remLo >> 30); remLo <<= 2;

    //Get ready for the next bit in the root
    root <<= 1;

    //Test radical
    testDiv = (root << 1) + 1;
    if (remHi >= testDiv) {
      remHi -= testDiv;
      root += 1;
    }
  } while (count-- != 0);
  return(root);
}
```

*Code Segment 3.4: Fixed point recursive square root calculation*

The code uses the standard command *Math.sqrt()* defined in the math library to calculate the square root of the given number after converting it to a float. The final result is then obtained by converting the square root float number back to a fixed point number.

A second solution is to use a well known recursive algorithm as described by [41]. The application of this algorithm to the specific fixed point format is given in code segment 3.4. No conversion to or from a floating point number is now needed to calculate the square root. Note that the outcome of this algorithm yields the exact same result as the floating point conversion method in code segment 3.3.

The third and final option is known as the *Quake square root approxima-*

```
//Calculate the inverse of the square root of a fixed point number
fixed8_24 InvSqrt8_24(fixed8_24 x){

  //Define a help variable half the size of x
  fixed8_24 xhalf = x >> 1;

  //Convert x to a float variable
  float t = x;

  //Store floating-point bits in long
  long i = *(long*)&t;

  //Initial guess for Newton's method
  i = 0x5f3759d5 - (i >> 1);
  x = (fixed8_24)(*(float*)&i * BASE24 * 4096);

  //One round of Newton's method
  x = Mult8_24(x,(0x01800000-Mult8_24(xhalf,Mult8_24(x,x))));

  //Return the result
  return x;
}
```

*Code Segment 3.5: Fixed point inverse square root*

*tion*, as it appeared in the source code of the first person shooter game *Quake III* [42]. The implementation is given by code segment 3.5. It is based on Newton approximation [43] and a magic constant that is used to create a very good initial guess [44]. This way, the Newton method must only be run once to obtain a result with a maximal error of 0.175 %. Note that the algorithm requires the fixed point number to be converted to a floating point number as was the case in the first code segment. When the floating point number is converted back into a fixed point number, it needs to be multiplied by the base and by the square root of the base as it is an initial guess of the inverted square root in fixed point notation. Furthermore, the inverse of the square root rather than the square root itself is in fact calculated. This is however not an issue as the code is mostly used to normalise vectors. In this case the vector components should be divided by the square root of the norm, requiring a division operation. With the inverse of the square root, multiplication can be used.

The performance of all three code segments was compared in a test program where the square root of 1000 randomly generated numbers is calculated. A timer sourced by the watch crystal was used as a time reference. The standard math method required an average time of 201 µs to complete one square root calculation when the microcontroller runs at its maximal clock frequency of 16 MHz. Running at the same settings, the recursive and the *Quake* code respectively take 103 µs and 64 µs.

### 3.4.3 Sensor Output Processing

As all of the filter calculations will be executed using the fixed point representation, all of the sensor data must first be converted to the appropriate format. Furthermore, calibration constants will need to be present on the sensor node as they must be applied before the data is actually valid and can be used for tracking purposes. Finally, the digital filter must also be applied in order to provide additional high frequency noise filtering.

#### 3.4.3.1 Calibration

The calibration procedure itself is still executed as described in section 2.4.2.2 and requires the nodes to be programmed to transmit their sensor data to the backend computer. The resulting constants must however be available for the microcontroller to apply them to the sensor outputs. Instead of including the values in the firmware, they are stored in a protected part of the flash memory called the information memory, which is not necessarily erased when a new program is loaded to the device. Recall that the sensor node ID is also programmed in this part of the memory. Although this requires manual programming from the designer, this way of working allows all of the nodes to be programmed with the same code and without any magic numbers.

#### 3.4.3.2 Accelerometer

The accelerometer output is a digital sequence of a single byte. The conversion to a fixed point number representing acceleration in g is obtained by first converting the byte value to a signed char and copying this value into a long. Then the bits in this long are shifted to the left such that the most significant bit of the original byte is in the position of the least significant integer bit of the fixed point representation. Finally, the resulting number is multiplied by the range of the accelerometer in fixed point format to obtain the final value. The code implementation is given by code segment 3.6. Note that although the sensor's range has been configured to ±2 g since the other alternative of ±6 g is not useful for the application at hand, the real maximum and minimum output value typically correspond to ±2.5 g.

#### 3.4.3.3 Magnetometer

The magnetometer output consists of two parts that need to be added together: a rough offset value and the fine measurement result. The rough offset is obtained at startup, when the sensor is ordered to perform a measurement over

```
//Convert accelerometer output
fixed8_24* ConvertAcc(unsigned char adata[]){

  //Declare return variable
  fixed8_24 acc[3];

  //Convert signed
  acc[0] = (signed char)adata[0];
  acc[1] = (signed char)adata[1];
  acc[2] = (signed char)adata[2];

  //Shift in place
  acc[0] <<= 17;
  acc[1] <<= 17;
  acc[2] <<= 17;

  //Multiply by range (=2.5g)
  acc[0] = Mult8_24(acc[0], 0x0280000000);
  acc[1] = Mult8_24(acc[1], 0x0280000000);
  acc[2] = Mult8_24(acc[2], 0x0280000000);
}
```

*Code Segment 3.6: Accelerometer output conversion to fixed point format*

its entire range which returns a 5 bit number. From then on, fine measurements are done in a part of the sensor's range around the rough offset result, giving a 10 bit output. Both results must however be combined to obtain a number that indicates the real value of the measurement. Furthermore, a factory calibration has been included in the memory of the device which must be applied to the resulting value. Finally, this result must be converted to the fixed point implementation in order for it to be used by the tracking filter. The code implementing all of this is given by segment 3.7.

Before the output is converted to the final fixed point format, an intermediate fixed point representation is used with ten fractional and six integer bits. The decimal point is chosen such that the rough offset precision equals unity, thus at least six integer bits were needed to allow a signed representation. The fine measurements are taken over a range of four rough intervals and divide this space into 1024 intervals given the 10 bit precision. This means that the two most significant bits also contribute to the integer bits of the fixed point format, while the other eight will correspond to the most significant fractional bits. Also note that 15 is subtracted from the resulting value to obtain a symmetric range.

Since the intermediate fixed point representation is used for the sensor output, the calibration values are also represented this way. Multiplication of both sequences can easily be implemented using the HWM and only requires one step given the fact that the numbers now only consist of 16 bits. Finally,

```
//Convert magnetometer output
fixed8_24* ConvertMag(unsigned char fine[]){

  //Declare return variable
  fixed8_24 mag[3];

  //Declare temporary 16 bit fixed point variables
  short x,y,z;
  short mdata[3];

  //Calculate fixed point representation and add rough offset value
  x = ((rough[0] + (fine[4] & 0x03) - 15) << 10) | (fine[5] << 2);
  y = ((rough[1] + (fine[2] & 0x03) - 15) << 10) | (fine[3] << 2);
  z = ((rough[2] + (fine[0] & 0x03) - 15) << 10) | (fine[1] << 2);

  //Apply factory calibration
  mdata[0] = x                + Mult6_10(b[1],y) + Mult6_10(b[2],z);
  mdata[1] = Mult6_10(b[3],x) + Mult6_10(b[4],y) + Mult6_10(b[5],z);
  mdata[2] = Mult6_10(b[6],x) + Mult6_10(b[7],y) + Mult6_10(b[8],z);

  //Copy to long
  mag[0] = mdata[0];
  mag[1] = mdata[1];
  mag[2] = mdata[2];

  //Shift in place
  mag[0] <<= 14;
  mag[1] <<= 14;
  mag[2] <<= 14;
}
```

*Code Segment 3.7: Magnetometer output conversion to fixed point format*

a shift operation of 14 bits to the left converts the intermediate fixed point number into the final representation with eight integer and 24 fractional bits.

#### 3.4.3.4 Digital Filter

The digital filter is implemented using the proposed fixed point format. Since it is a third order inverse Chebychev type filter, it requires seven multiplications and six summations per application. There are three values on each of the sensors, resulting in a total of 42 multiplications and 36 summations each cycle.

### 3.4.4 Kalman Filter Algorithm

The execution time of the Kalman filter algorithm implementation can be reduced by optimising some of the matrix operations that are used. First of all, several simplifications can be introduced to reduce the number of matrix operations needed for the implementation of the algorithm. Second, many matrices are in fact symmetric as they represent a covariance, which means only the upper or lower triangular should be calculated. And third, the matrix inversion can also be accelerated when taking into account the symmetric form of the matrix to invert.

Note that the adaptive feature of the filter will not be implemented in the embedded version. The reason lies in the fact that the R matrix cannot be chosen as small as has been put forward in section 2.5.2.1, since this causes overflow to occur at the point where matrix inversion is needed. Therefore, the increased value will already make the filter less sensitive to disturbances.

#### 3.4.4.1 Simplifications

The extended Kalman filter equations used in the quaternion MFG filter are repeated here for convenience:

$$\begin{aligned}
\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_{k-1} + \tau\,(\hat{\mathbf{x}}_{k-1} - \hat{\mathbf{x}}_{k-2}) \\
\mathrm{P}_k &= \mathrm{A}_k\,\mathrm{P}_{k-1}\,\mathrm{A}_k^T + \mathrm{Q}_k \\
\mathrm{K}_k &= \mathrm{P}_k\,\mathrm{H}_k^T\left(\mathrm{H}_k\,\mathrm{P}_k\,\mathrm{H}_k^T + \mathrm{R}_k\right)^{-1} \\
\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k + \mathrm{K}_k\,(\mathbf{z}_k - h_k\,(\hat{\mathbf{x}}_k)) \\
\mathrm{P}_k &= (\mathrm{I}_{n\times n} - \mathrm{K}_k\,\mathrm{H}_k)\,\mathrm{P}_k
\end{aligned} \tag{3.5}$$

First of all, it should be noted that the a priori and a posteriori estimate and corresponding error covariance can be stored in the same variable as when one of them is calculated, there is no need to store the other. Therefore, the superscript minuses have been dropped in the above equations.

The prediction step can further be simplified when taking into account that A is actually an identity matrix and Q is diagonal. Therefore, the prediction error covariance can be determined using only four summations.

The term $\mathrm{P}\,\mathrm{H}^T$ actually appears twice in the calculation for K and can thus be saved as a temporary variable further denoted as PHT. The transpose of this term can also be found in the last equation when it is rewritten:

$$\begin{aligned}
\mathrm{P}_k &= (\mathrm{I}_{n\times n} - \mathrm{K}_k\,\mathrm{H}_k)\,\mathrm{P}_k \\
&= \mathrm{P}_k - \mathrm{K}_k\,\mathrm{H}_k\,\mathrm{P}_k \\
&= \mathrm{P}_k - \mathrm{K}_k\left(\mathrm{P}_k^T\,\mathrm{H}_k^T\right)^T \\
&= \mathrm{P}_k - \mathrm{K}_k\left(\mathrm{P}_k\,\mathrm{H}_k^T\right)^T
\end{aligned}$$

where the symmetrical form of P has been used and the fact that the transpose of a product of matrices equals the product of the transposed terms in reverse order.

```
//PHT = P * Transpose(H)
for(i=0; i<n; i++){
  for(j=0; j<m; j++){
    PHT[i][j] = Mult8_24(P[i][0], H[j][0]);
    for(k=1; k<i+1; k++){
      PHT[i][j] = PHT[i][j] + Mult8_24(P[i][k], H[j][k]);
    }
    for(k=i+1; k<n; k++){
      PHT[i][j] = PHT[i][j] + Mult8_24(P[k][i], H[j][k]);
    }
  }
}

//HPHT = H * PHT = H * P * transpose(H)
for(i=0; i<m; i++){
  for(j=0; j<i+1; j++){
    HPHT[i][j] = Mult8_24(H[i][0], PHT[0][j]);
    for(k=1; k<n; k++){
      HPHT[i][j] = HPHT[i][j] + Mult8_24(H[i][k], PHT[k][j]);
    }
  }
}
```

*Code Segment 3.8: Symmetric matrices*

### 3.4.4.2 Symmetric Matrices

All of the matrices representing a covariance are symmetric. Furthermore, left multiplication of a symmetric matrix with an arbitrary matrix and right multiplication with its transpose also yields a symmetric matrix. Therefore, the following matrices are symmetric:

$$\begin{array}{ccc} P & Q & R \\ HPH^T & HPH^T + R & KHP \end{array} \tag{3.6}$$

From all of these matrices, only the lower triangular part is stored as this covers all necessary information. This should however be taken into account when using the matrices in further calculations. Both aspects can be found in code segment 3.8, where PHT is calculated from the symmetric matrix P and HPHT is a symmetric matrix by itself. In the fist part, care has been taken not to address any elements from P that are in the upper triangular part, as these are invalid. The for loop has therefore been separated into two loops. In the second part, it is clear that only the lower triangular part is calculated since the index of the second for loop never exceeds the index of the first.

#### 3.4.4.3 Symmetric Inversion

In general, matrix inversion is a very complex and time consuming operation to perform. It can be accomplished by using either the Gauss-Jordan elimination algorithm or LU decomposition [45]. In the first, a matrix is augmented with an identity matrix, linear operations are used to reduce the initial matrix to an identity matrix and the inverse is then found in the augmented part. The latter method first decomposes a matrix into an upper and lower triangular matrix, inverts both these matrices and multiplies them again in reverse order to find the inverse of the original matrix.

The inversion of a symmetric matrix is slightly easier to perform since these matrices can be decomposed using Cholesky decomposition [46], which yields a single triangular matrix such that the original matrix equals the product of this triangular matrix with its transpose. The inverse matrix can now be calculated by inverting only one triangular matrix.

$$M^{-1} = \left(L\,L^{T}\right)^{-1} = L^{-1^{T}}\,L^{-1} \tag{3.7}$$

**Cholesky Decomposition**

Code segment 3.9 implements the Cholesky decomposition. In each step, one row of the triangular matrix is calculated recursively by substitution. Note that some handy tricks are incorporated to avoid having to recalculate certain values. The array variable *t[]* is used to store all the inverse square roots of the diagonal elements of the original matrix, which is reused in the calculation of the off diagonal elements. Note however that the diagonal elements equal the square root itself and thus an inversion is needed. Since the fixed point division has not been implemented, this is accomplished by converting the number to a float representation, inverting the float and reverting to the fixed point notation. This method turns out to be faster than computing the square root directly using any of the other methods proposed in section 3.4.2.3.

**Triangular Matrix Inversion**

Triangular matrix inversion is implemented by code segment 3.10 and is easily obtained through substitution. Important to know is that the inverse of a triangular matrix is also triangular. From this quickly follows that the diagonal elements of the inverse matrix equal the inverse of the diagonal elements. This again requires a division that is done via floating point representation. The off diagonal elements can be calculated using the resulting diagonal elements.

```
//Cholesky decomposition of M with side m
fixed8_24* Cholesky(fixed8_24* M, int m){

  //Declare return and temporary variables
  fixed8_24 L[m][m];
  fixed8_24 sum;
  fixed8_24 t[m];

  //Decomposition
  for (i=0; i<m; i++){

    //Off diagonal elements
    for (j=0; j<i; j++){
      sum = 0;
      for (k=0; k<j; k++){
        sum = sum + Mult8_24(L[i][k], L[j][k]);
      }
      L[i][j] = Mult8_24((M[i][j] - sum), t[j]);
    }

    //Diagonal elements
    sum = 0;
    for (k=0; k<i; k++){
      sum = sum + Mult8_24(L[i][k], L[i][k]);
    }
    t[i] = InvSqrt8_24(M[i][i] - sum);
    float x = t[i];
    x = BASE24 / x;
    x = x * BASE24;
    L[i][i] = (fixed8_24)x;
  }

  //Return result
  return L;
}
```

*Code Segment 3.9: Cholesky decomposition*

### 3.4.5 Wireless Data Package

Since the functionality of the wireless protocol and the settings of the RF transceiver chip change when the size of the wireless data packet is altered, it would be beneficial to keep this length fixed. The MFG packet format has already been explained in section 3.3.2 and is shown in figure 3.17. The total size of this package is 11 bytes, where the first two are vital for the correct functioning of the protocol and the recognition of the sensor node data at the base station and backend computer side. That leaves us with nine available bytes for the resulting orientation quaternion consisting of four times four bytes, 16 in total.

As has been shown in section 2.2.2, an orientation quaternion needs to be

```
//Inversion of a lower triangular matrix with side m
fixed8_24* InvertTriangular(fixed8_24* L, int m){

  //Declare return and temporary variables
  fixed8_24 Linv[m][m];
  fixed8_24 sum;

  //Calculate inverse
  for (i=0; i<m; i++){

    //Linv[i][i] = 1 / L[i][i]
    float x = L[i][i];
    x = BASE24 / x;
    x = x * BASE24;
    Linv[i][i] = (fixed8_24)x;

    //Linv[i][j] = -Linv[i][i] * sum(L[i][k] * Linv[k][j], k=j..i-1)
    for (j=0; j<i; j++){
      sum = 0;
      for (k=j; k<i; k++){
        sum = sum - Mult8_24(L[i][k], Linv[k][j]);
      }
      Linv[i][j] = Mult8_24(Linv[i][i], sum);
    }
  }

  //Return result
  return Linv;
}
```

*Code Segment 3.10: Triangular matrix inversion*

normalised, which implies that one of the components, e.g. the scalar part, can be reconstructed apart from its sign:

$$t \;=\; \pm\sqrt{1 \;-\; \left(x^2 \;+\; y^2 \;+\; z^2\right)} \tag{3.8}$$

However, both **q** and −**q** represent the same orientation as shown by (2.27). Thus, by choosing the quaternion with positive scalar part as the standard, reconstruction can indeed be applied at the backend side and only three components will need to be transmitted. Since all three are equally important and nine bytes are available, they will need to be repacked into three bytes each.

From the normalisation condition also follows that the quaternion only contains elements that are bound between 1 and −1. This clearly indicates that the most significant six bits of the fixed point representation used in the above are actually superfluous as they will either equal zero if it concerns a positive number or one in the case of a negative number. Given the fact that eight bits need to be dropped, the least significant two bits of the fractional part will also be omitted resulting in a new fixed point representation of the quaternion parts in the data package consisting of two integer and 22 fractional bits. The resulting data package is displayed in figure 3.29.

| Master: Packet Counter<br>Slave: Timeslot Number | Node<br>ID | Quat<br>X | Quat<br>Y | Quat<br>Z |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 3 Bytes | 3 Bytes | 3 Bytes |

*Figure 3.29: Contents of the RF data package when nodes perform embedded orientation tracking.*

### 3.4.6 Current Consumption

The current consumption of a third generation MFG node running the orientation estimation filter in firmware is displayed in figure 3.30. When the graph was taken, the node was operating as the network master. In general, the resulting graph resembles the normal consumption graph displayed in figure 3.23 aside from the consumption due to the active microcontroller. This corresponds to a constant consumption of approximately 7.5 mA for almost 7.5 ms. The resulting average current consumption amounts to 6.5 mA leading to a lifetime of six days when using a Varta PoliFlex battery [18].

The result shown here was obtained using the inverted square root approximation which exhibits the lowest execution time. Since almost 1.5 ms remain available, the recursive implementation could also be used, yet no improved behaviour was found, only an increase in current consumption. Important to mention is the fact that the multiplication procedure described in section 3.4.2.2 has been simplified by removing the final multiplication between the lowest significant bits as its influence on the output result seemed minimal.

At this point, the current consumption per node can be compared to other known systems since these all incorporate on board orientation estimation. Note however that it concerns MARG systems, which inherently consume more power. The Orient system from the University of Edinburgh [39] has a lifetime of approximately two hours on a 120 mAh battery. Hence the average current consumed by a single node is around 60 mA. Unfortunately, XSens only mention that their wireless sensor nodes [38] have an autonomy of three and a half hours without mentioning the battery capacity. Hence the only conclusion that can be drawn is that the system presented in this dissertation can last a lot longer than the existing solutions given the battery choice that is made.

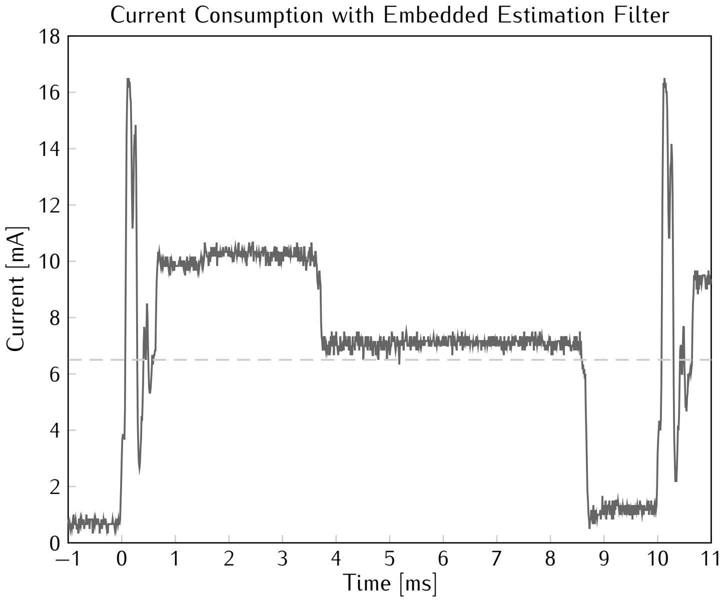


*Figure 3.30: Time graph of the current consumption in a third generation MFG node running the embedded version of the orientation estimation filter and operating as a master. The dashed line indicates the average consumption over time.*

### 3.4.7 MARG Extension

The MARG filter has not yet been actually implemented in firmware, yet some considerations are highlighted here.

The main challenge will be in the implementation of the adaptive part of the filter, the other aspects are mostly covered in the implementation of the MFG filter. This adaptive part requires the calculation of the average over a certain window length of the innovation covariance. This moving average can best be calculated by keeping track of all the matrices in the window and determining the new average by first subtracting the oldest matrix and then adding the new one. This way, only two matrix additions need to be executed to calculate the new average. The resulting measurement and process covariance matrices can then easily be determined using (2.88) and (2.89).

Although the data package of the MARG nodes could accommodate more bits of the orientation result, this advantage should not be used. It is much more interesting to reduce the size of the data packets and construct them according to figure 3.29. This way, more MARG nodes are capable of transmitting their information to a single base station. Furthermore, no more distinction will exist between both node types, they can coexist in the same network and communicate to the same base station.

## 3.5 Conclusion

In the first section, an overview of all types of hardware has been given. Each of the designs introduced a new aspect that provided added value compared to the previous system. Early, first generation test systems carefully scanned the world of inertial sensing and wireless communication to lay the basics of the sensor node architecture. In the second generation, these aspects were combined into a fully operational sensor network and the base station architecture was defined. The third generation introduced lower power consumption and the possibility to extend the network with more nodes. Finally, the fourth generation was meant to make the system more unobtrusive using advanced board and packaging techniques.

A fully plug and play, wireless ad hoc network protocol was introduced in the following section. Nodes can take the role of a master or a slave in a TDMA like network scheme where data is short lived and only sent to the base station. The master acts as a synchronisation beacon for the slaves who time their transmission relative to the reception of master packages. A dynamic implementation results when the role of the sensor nodes is determined at runtime and slaves choose their own timeslot according to its availability. Slaves also watch the health of the master since its functionality is crucial for the survival of the network and they take over whenever the master fails. Additional control mechanisms have been implemented to counter colliding data transfer. By careful implementation of the protocol, an average current consumption of less than 3 mA is obtained while a maximum of 19 MFG nodes are capable of transmitting their data to a single base station.

Finally, the implementation of the orientation tracking filter in firmware is discussed. The quaternion type extended Kalman filter is chosen due to the absence of trigonometric functions, the number of operations required for estimation and the performance determined in chapter 2. A fixed point number format using eight integer and 24 fractional bits was chosen and multiplication was implemented using the hardware multiplier. For the square root, an approximation based on the Newton method was found that offers very fast calculation of the inverse of the square root. The filter implementation was realised by making maximal use of the advantages of symmetrical matrices: only lower triangular parts are calculated and inversion is completed using Cholesky decomposition. Due to all these simplifications, nodes are capable of calculating the orientation estimate within the available 10 ms timeframe while consuming an average current of 6.5 mA.

# References

[1] N. Decraene. *Electronics for Movement Analysis in Medical Applications.* Master's thesis, Ghent University, Ghent, Belgium, 2007.

[2] Kionix Inc., Ithaca, NY, USA. *2g Tri-Axis Accelerometer Specifications*, December 2006. http://www.kionix.com/Product-Specs/KXPS5-2050%20Specifications%20Rev%204.pdf.

[3] Analog Devices, Norwood, MA, USA. ±75 °/s *Single Chip Yaw Rate Gyro with Signal Conditioning*, 2004. http://www.analog.com/en/mems-sensors/gyroscopes/adxrs401/products/product.html.

[4] Invensense. *IDG-300, Integrated Dual-Axis Gyro*, August 2007. http://invensense.com/mems/gyro/documents/PS-IDG-0300B-00-03.pdf.

[5] Honeywell, Plymouth, MN, USA. *Digital Compass Solution HMC6352*, January 2006. www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Missiles-Munitions/HMC6352.pdf.

[6] Invensense. *ITG-3200, Product Specification*, March 2010. http://invensense.com/mems/gyro/documents/PS-ITG-3200-00-01.4.pdf.

[7] Texas Instruments. *MSP430F149, 16-bit Ultra-Low-Power Microcontroller*, June 2004. http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=msp430f149&fileType=pdf.

[8] Nordic Semiconductor. *nRF2401, Single chip 2.4 GHz Transceiver*, June 2004. http://www.nordicsemi.no/files/Product/data_sheet/nRF2401rev1_1.pdf.

[9] B. Kuyken and W. Verstichel. *Realisation of a Headbanging Orchestra using Movement Sensors and Wireless Links.* Master's thesis, Ghent University, Ghent, Belgium, 2008.

[10] B. Kuyken, W. Verstichel, F. Bossuyt, J. Vanfleteren, M. Demey, and M. Leman. *The HOP sensor: Wireless Motion Sensor.* In Proceedings of the International Conference on New Interfaces for Musical Expression, Genova, Italy, 2008.

[11] W. Christiaens, B. Vandevelde, and J. Vanfleteren. *Ultra-Thin Chip Package (UTCP) for Flexible Electronics Applications.* In Proceedings of the IMAPS Nordic Conference, pages 7–11, Gothenburg, Sweden, September 2006.

[12] E.P. Hanavan. *A Mathematical Model of the Human Body*. Technical Report TR-64-102, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, OH, USA, 1964.

[13] *USB.org, Universal Serial Bus*. http://www.usb.org/.

[14] Digital Equipment, Intel, and Xerox. *The Ethernet, A Local Area Network, Data Link Layer and Physical Layer Specifications*, September 1980.

[15] *Wi-Fi Alliance*. http://www.wi-fi.org/.

[16] *Bluetooth.org, Official Bluetooth Info Site*. http://www.bluetooth.com/English/Pages/default.aspx.

[17] Artaflex, Markham, Ontario, Canada. 2.4 GHz *DSSS Radio Module with Integrated Power Amplifier*, October 2008. http://artaflexmodules.com/sites/default/files2/DataSheet%20AWA24S.pdf.

[18] Varta. *LPP423566, Rechargeable Lithium Polymer*, September 2008. http://www.varta-microbattery.com/en/oempages/product_data/poductdata_types.php?output=typedata&segment=RechLiFlatPoly.

[19] ST Microelectronics. *LIS3LV02DQ, 3-Axis - ±2g/±6g Digital Output Low Voltage Linear Accelerometer*, October 2005. http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00047926.pdf.

[20] Yamaha. *YAS529, Magnetic Field Sensor*, June 2006. http://www.yamaha.co.jp/english/product/lsi/prod/pdf/sensor/BAS529A20.pdf.

[21] Atmel. *ATMega168, 8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash*, September 2007. http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf.

[22] Cypress. *CYRF6936, WirelessUSB LP 2.4 GHz Radio SoC*, April 2007. http://www.cypress.com/?docID=17202.

[23] Silicon Laboratories. *CP2102, Single-Chip USB to UART Bridge*, October 2004. http://www.silabs.com/pages/DownloadDoc.aspx?FILEURL=Support%20Documents/TechnicalDocs/cp2102.pdf&src=DocumentationWebPart.

[24] Microchip. *ENC28J60, Stand-Alone Ethernet Controller with SPI Interface*, January 2008. http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf.

[25] ST Microelectronics. *LIS302DL, 3-Axis - ±2g/±8g Smart Digital Output Piccolo Accelerometer*, October 2008. http://www.st.com/stonline/products/literature/ds/12726.pdf.

[26] Texas Instruments. *MSP430f249, 16-bit Ultra-Low-Power Microcontroller*, April 2009. http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=msp430f249&fileType=pdf.

[27] R. Rojas and U. Hashagen. *The First Computers: History and Architectures*. MIT Press, 2000.

[28] B. Vanhoutte. *Design and Implementation of a Wireless Communication Protocol for Sensor Networks*. Master's thesis, Ghent University, Ghent, Belgium, 2010.

[29] Future Technology Devices International. *FT232R, USB UART IC*, January 2006. http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf.

[30] Roving Networks. *RN-41: Class 1 Bluetooth Module*. Los Gatos, CA, USA, August 2009. http://www.rovingnetworks.com/documents/RN-41.pdf.

[31] Texas Instruments. *MSP430F2132, 16-bit Ultra-Low-Power Microcontroller*, April 2009. http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=msp430f2132&fileType=pdf.

[32] W. Christiaens. *Active and Passive Component Integration in Polyimide Interconnection Substrates*. PhD thesis, CMST, Elis Department, Ghent University, November 2008.

[33] Logitech. *PM5 Auto Precision Lapping and Polishing Machine*. http://www.logitech.uk.com/pm5autolap.asp.

[34] Nordic Semiconductor. *nRF24L01, Single chip 2.4 GHz Transceiver*, July 2007. http://www.nordicsemi.com/eng/content/download/2730/34105/file/nRF24L01_Product_Specification_v2_0.pdf.

[35] R. Zhu and Z. Zhou. *A Real-Time Articulated Human Motion Tracking using Tri-Axis Inertial/Magnetic Sensors Package*. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 12(2):295–302, June 2004.

[36] R. Rom and M. Sidi. *Multiple Access Protocols: Performance and Analysis*. Springer-Verlag, 1990.

[37] J.F. Wakerly. *Digital Design Principles & Practices*. Prentice Hall, 2000.

[38] XSens Technologies. *MTw 3DOF Orientation Tracker.* http://www.xsens.com/images/stories/products/PDF_Brochures/mtwleaflet.pdf.

[39] A. D. Young, M. J. Ling, and D. K. Arvind. *Orient-2: a Realtime Wireless Posture Tracking System using Local Orientation Estimation.* In Proceedings of the 4th Workshop on Embedded Networked Sensors, pages 53–57, Cork, Ireland, 2007.

[40] IEEE. *IEEE Standard for Floating-Point Arithmetic.* IEEE Std 754-2008, August 2008.

[41] K. Turkowski. *Fixed Point Square Root.* Technical Report 96, Apple, 1994.

[42] id Software. http://www.quake.com/.

[43] C.T. Kelley. *Solving Nonlinear Equations with Newton's Method.* Siam, 2003.

[44] C. Lomont. *Fast Inverse Square Root.* Technical report, Department of Mathematics, Purdue University, West Lafayette, Indiana, USA, 2003.

[45] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins, 1996.

[46] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Steward. *LINPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, 1979.

# 4 Software

This chapter contains a description of the key features that are implemented in the computer software that allows data processing of the sensor signals. The emphasis lies with the visualisation as this is the most important task to be fulfilled.

## 4.1 Introduction

The software that has been developed throughout the years found its origin in the work conducted by ir. Niels Decraene in the scope of his master thesis [1]. The initial applications allowed to receive data from one of the first generation systems and visualise it on the screen. A first attempt for orientation tracking was already implemented by simple integration of the gyroscope signals and periodic correction using the accelerometer output. Naturally, the orientation drifted due to noise and the heading was never corrected as the accelerometer does not supply any information about this angle. However, as the first generation system was readily available for initial testing, the software was also adopted as a starting point.

The original code was written in Visual Basic (VB) version 6.0 [2]. Since the main stream support for this programming language ended in March 2005, the first task was to transfer the code to a newer language. Finally, the .NET version of VB [3] was chosen as it is closely related and would require the least amount of modifications. The language is also part of Microsoft's .NET framework which consists of a large library that can be used by any of the supported languages and a Common Language Runtime (CLR) that provides services as security, memory management and exception handling [4]. Programs can even be compiled from source files written in different languages, as long as they are part of the framework. Another advantage of migrating to the framework is found in the fact that a free implementation is available in the form of Microsoft Visual Studio Express [5].

Gradually, the program has been expanded with new features starting with graphs for visualisation of sensor data and ending with a stickman for full body posture tracking. With the parallel developments in the hardware of the system, the program also gained support for all different generations of systems, although the support for the first generation systems has now been abandoned since none of these devices offered full three dimensional driftless tracking.

## 4.2 Application Overview

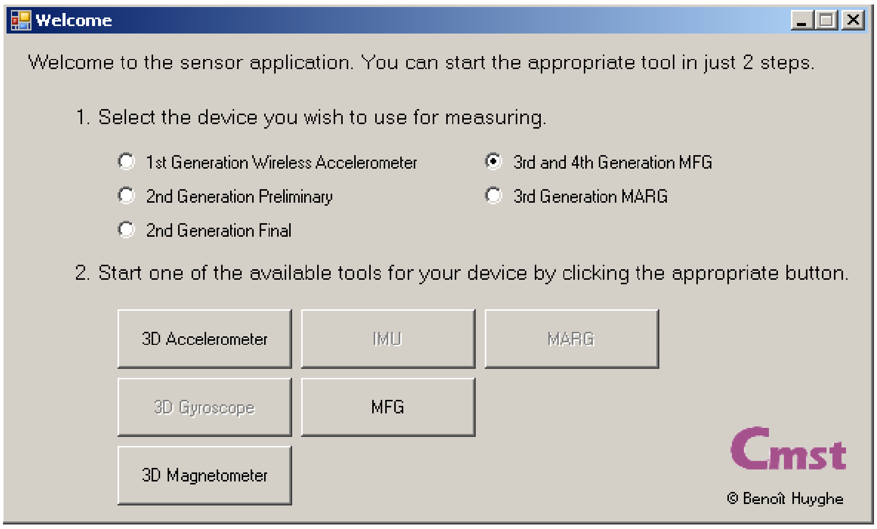
In this section, an overview of the entire software application is given. First, the general structure of the software is outlined and each of the windows that form the Graphical User Interface (GUI) of the application are described. Then, the data flow through the software is shown and the appropriate classes are highlighted.

### 4.2.1 Software Structure

The software GUI consists of several windows that are linked to each other. At startup, the *Welcome* window is the first to be displayed and prompt the user for input. Then, depending on the choices made by the user, a *Device* window is loaded which allows to visualise sensor data and orientation. Once this is active, a *Calibration* window may be started in case a calibration needs to be executed. When shutting down the software, the order of appearing windows is reversed.

#### 4.2.1.1 Welcome Window

The welcome window is shown by default when the application is started. As can be seen in figure 4.1, it allows the user to select which tracking system is currently in use followed by the choice of which device window to run. The buttons are only activated when that specific device window is available for the selected system.



Welcome

Welcome to the sensor application. You can start the appropriate tool in just 2 steps.

1. Select the device you wish to use for measuring.

○ 1st Generation Wireless Accelerometer
○ 2nd Generation Preliminary
○ 2nd Generation Final
● 3rd and 4th Generation MFG
○ 3rd Generation MARG

2. Start one of the available tools for your device by clicking the appropriate button.

*Figure 4.1: Screen capture of the welcome window.*

#### 4.2.1.2 Device Window

There are six device windows each supporting a different set of sensors. Three of them support a single three dimensional sensor, either an accelerometer, a gyroscope or a magnetometer. Two support a combination of two sensors, the MFG combines acceleration and magnetic field measurements and the Inertial Measurment Unit (IMU) is based on an accelerometer and gyroscope combination. Finally, the MARG device window supports the combination of all three sensors.

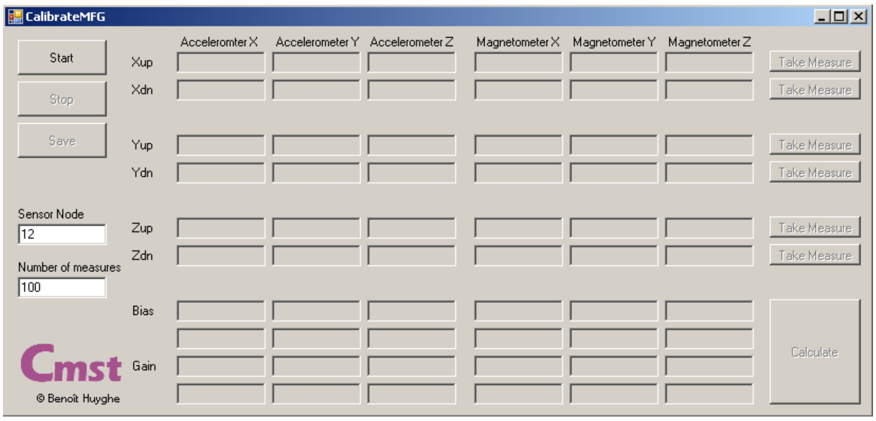
The device windows are the actual work horses of the application as they implement the main functionality of the software. Before starting however, several radiobuttons, textboxes and checkboxes need to be set up. The communication interface with the receiver can be set to either a serial port or User Datagram Protocol (UDP) packages and all of the parameters associated with it can be adjusted. The type of sensor node output is selected to be either sensor output data or orientation from the embedded filter implementation. The digital pre-filter for sensor data can be activated or deactivated and all of the coefficients can be adjusted using a dialog window. All of the filters described in chapter 2 can be used to perform orientation tracking using either Euler angle or quaternion representation. Moreover, all of the parameters used by the tracking filters can be adjusted using textbox entries. Finally, checkboxes allow the user to have the application write data to log files.

The windows support four types of visualisation placed in different tabs.

The first tab contains graphs where sensor output data can be visualised. The second features a beam drawn using Open Graphics Library (OpenGL) [6] that visualises the orientation and several graphs for both the sensor data and the orientation. The orientation of up to six nodes can be visualised by individual beams in the third tab. Finally, full body posture can be shown in the last tab by means of a stickman. On each tab, the ID numbers of the operating nodes can be entered in textboxes.

#### 4.2.1.3 Calibration Window

A calibration window can be started by selecting the sensors to calibrate in the device window and clicking the calibrate button. The type of calibration window depends on which sensors were selected for calibration. Figure 4.2 shows the calibration window for both the accelerometer and the magnetometer.



Sensor Node 12

Number of measures 100

*Figure 4.2: Screen capture of the calibration window.*

First, the ID of the sensor node in use must be entered. Also, the number of measures that will be averaged to obtain a valid output value can be adjusted, although the standard value of 100 samples should suffice. Then, by pressing the start button, samples are acquired from the sensor node and it may be positioned in any of the required standard orientations as defined by the calibration procedure described in section 2.4.2.2. Once the positioning is complete and the sensor node is steady, a measurement is initiated by pressing the appropriate button on the right hand side of the window. Upon completion, the average measurements are displayed in the textboxes. After completing the procedure for all six positions, the sensor readout can be stopped and the bias and gain values can be calculated. Finally, the newly acquired calibration result can be saved to a text file. The application will then automatically load the available calibration values when the same sensor nodes is used for tracking.

## 4.2.2 Data Flow

The data flow through the software is displayed in figure 4.3. The origin of the data is the base station that is connected to the computer which receives the sensor data wirelessly. It then propagates through the program passing several classes until it reaches its final goal which can be one of the many visualisations implemented in the software.

In this section, the data flow of normal sensor data is described. For the case where the embedded estimation filter is running on the nodes, small adjustments need to be made that even simplify the flow as conversion to orientation is no longer needed.

### 4.2.2.1 Data Collection

Data can be received from the base station using either a UDP socket or a COM port object.

**UDP Socket**

When the base station is connected to the computer via Ethernet [7, 8], it will transmit the data in UDP type packages which must then be unpacked [9]. This task is executed by a piece of code that represents the UDP socket and which determines if a package conforms to the given protocol, checks if the Internet Protocol (IP) address [10] is correct and if the port number corresponds. If all of the conditions are met, the package payload is unwrapped and handed to the network class in the form of a byte array.

**COM Port**

When either a UART to USB interface [11–13] or Bluetooth [14, 15] are used for data transfer, packages are received via a serial port [16]. In both cases, this serial port is actually a virtual COM port mounted in software by a driver. This way, existing software objects that were frequently used in the past for data communication using the serial port can be recycled. These objects are capable of translating the received signals into byte arrays when the baudrate and number of control bits are known.

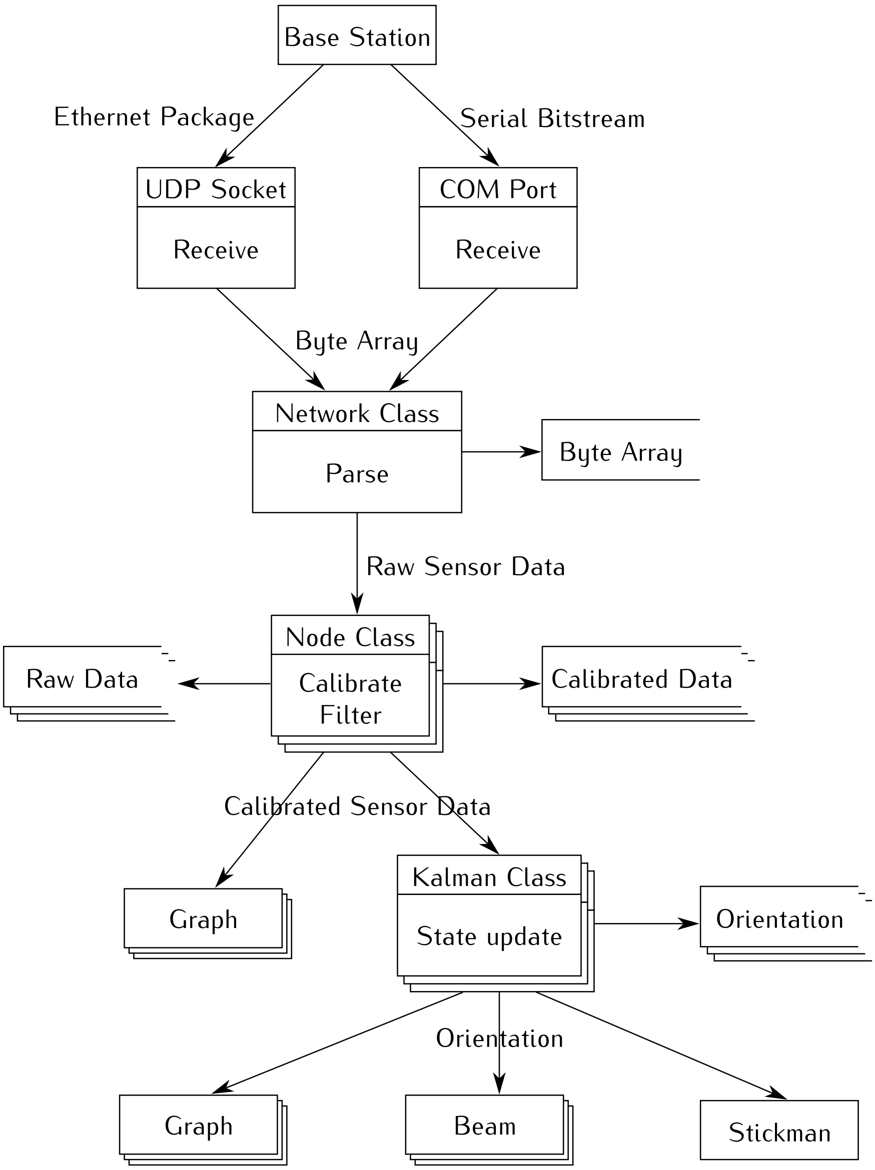


*Figure 4.3: Software data flow diagram.*

#### 4.2.2.2 Network Class

The network class is responsible for managing all of the data packages received from a base station. All of these packets will conform to the form described in section 3.3.2 and displayed in figure 3.18. Optionally, the received byte array can first be written to a log file. Then, the data is unpacked 10 bytes at a time and parsed to floating point numbers representing the raw output values from the sensors of a single node. Depending on the ID number, this raw data is subsequently passed on to the corresponding node class object. If no object exists with the encountered ID number, the data is discarded. The procedure is naturally repeated a number of times equal to the number of RF packages that have been received by the base station in the present timeframe.

#### 4.2.2.3 Node Class

There are as many node class objects as there are nodes whose data should be visualised. Each individual object will validate the data corresponding to its assigned ID number. Before any operations are performed on the raw data, it can optionally be written to a log file. In this case, multiple log files will exist as each node object will create its own. Then, processing of the raw data starts by applying the calibration constants which are recovered from a file when the node object is created. Afterwards, the data is filtered by an object of the digital filter class which implements the pre-filter designed in chapter 2. At this point, the newly acquired data can again optionally be written to a log file, before being transferred to either a graph object or a Kalman filter.

#### 4.2.2.4 Kalman Filter Class

Each node object has an associated Kalman filter class object that performs estimation of the orientation. It requires the calibrated sensor output to perform an update of the internal state which can be either a set of Euler angles or a quaternion. Optionally, this state can be written to a log file for future reference. Also, an optional post-filter can be applied to the orientation at this point.

#### 4.2.2.5 Visualisation

The final destination of the data is some kind of visualisation. Three types exist: a graph, a beam or a stickman. Note that although the data arrives at a rate of 100 Hz, the visualisation is updated at a slower rate of 25 Hz to reduce the workload on the computer.

The beam and stickman class are drawn using OpenGL [6], which is an Application Programming Interface (API) for 2D and 3D graphics. It includes many functions for rendering, texture mapping and special effects for all popular platforms. The functions are made available for the .NET programming languages through the TAO framework [17], which is a library that allows access to many graphics libraries including OpenGL. As a base, the *visual* class is defined as an extension of a TAO included class called *SimpleOpenGLControl*. This newly defined class supports general functions as mouse controlled rotation of the drawn objects and resizing of the window.

### Graph Class

The graph class extends the .NET picturebox class and accepts data in numeric form. With each update, a line is drawn between the previous sample and the current to form a piecewise linear graph. Multiple data entries are drawn using different line colors.

### Beam Class

Either Euler angles or quaternion orientation can be loaded into the beam class. A refresh of the visual causes the OpenGL frame to be rotated according to the given orientation and a multicolored beam to be drawn. The result is an on screen beam that positions itself in realtime according to the sensor node's orientation.

### Stickman Class

The stickman class features several bone class objects that are connected to each other when drawn. The orientation of each of these bones corresponds to the output of a Kalman filter associated to a node active in the network. Although in general 15 bones are required for a realistic human approximation [18], hands and feet are omitted to form a stickman consisting of only 11 parts: two for each arm and leg, one for the head and two for the torso. When an update is performed, a human correction model is applied and the stickman is drawn recursively starting at a root. More details on the human model that is used to correct impossible postures is given in the next section.

## 4.3 Human Model

When the orientation output of sensor nodes is mapped directly to a stickman figure, chances are that the posture of the stickman is in general impossible for any human to take on. The reason for this lies in the errors that are present in the orientation output and originate from slight calibration mismatches to the influence of motion disturbance on the sensor readings. In an attempt to correct these anatomically incorrect postures to more feasible ones, a human model is included in the software.

The basics of the human model has been laid out by two students, ir. Wim Mistiaen and ir. Sander Van Schoote, in the scope of their master thesis [19]. The thesis was the result of a cooperation between CMST and Multimedialab and also contained a part about high level recognition of certain actions or gestures from the output sequences of the orientation tracking filter. While this second part is not outlined here as it is beyond the scope of this work, the first part consisting of the human model is of major importance. Therefore several improvements have been added to the model since the initial version which will be discussed in this section.

This section starts with a description of how the stickman figure is animated at each iteration and how the corrections are performed in general. Then an overview is given of the offsets that must be applied to the bone orientations in order to correct errors due to misplacement of the sensor nodes on the body of the test subject. Subsequently, the procedure for correcting the bone orientation according to the human model is outlined. Finally, a floor is added to the visualisation, allowing the stickfigure to move around in the virtual world.

### 4.3.1 Stickman Build-up

The stickman figure naturally consists of several bones that are interconnected by joints [20]. Since the endpoint of a bone is mostly the starting point of one or more other bones, the drawing and updating process will be implemented in a recursive way. Therefore, bones are arranged in a tree structure where each bone is assigned a parent whose endpoint equals the startpoint of the considered bone and possibly multiple children whose startpoint is connected to its endpoint. In general, a tree always starts with a *root* which can be either an object itself or a dummy version that is not displayed and only serves as an anchorpoint [21]. The tree of bones that is used for drawing the stickman is displayed in figure 4.4. In this case, the root is a dummy bone that is not drawn and is located between the upper and lower torso. Aside from the root, 15 bones are modeled, although only 11 of them are assumed to be equipped with sensor nodes. The shoulder and hip bones will be attached to the torso when drawn.

Each time the stickman is rendered, the calculated orientations of the sensor nodes are first assigned to their respective bones. Then the orientation is corrected recursively in the order defined by the tree structure and according to a defined set of correction rules. However, these rules can only be defined when the subset of anatomically correct orientations is known, which in turn entirely depends on the type of link that exists between the parent and the considered bone as it is assumed that the bones can be approximated by rigid bodies. Therefore, some sort of parameterisation of realistic human joint limits

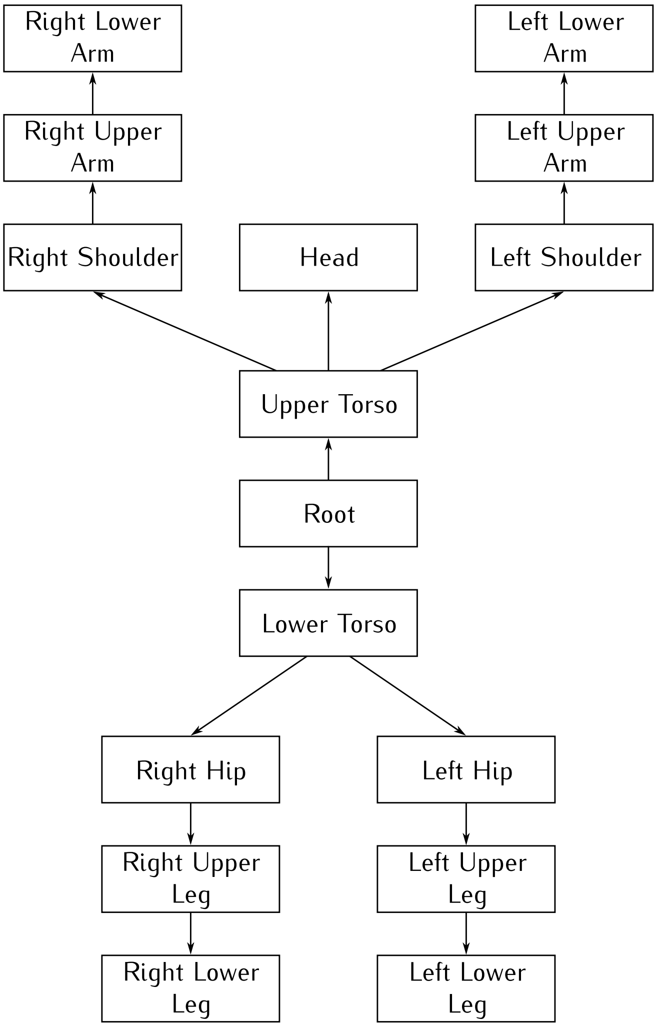


*Figure 4.4: Stickman bones tree structure.*

needs to be chosen that allows a definition of the feasible range of motion. A commonly used method is to describe the orientation using the *swing-twist* formalism that partitions an arbitrary orientation into two components [22]. The first component is called the *swing* and corresponds to the spherical motion where the bone is pointed in any arbitrary direction. The second component is an axial rotation around this final direction, also called the *twist*. The anatomical limits that are imposed by the presented model will bound both of these components separately. Although both components are in fact coupled to some extend [23], this way of working allows to define restrictions in an intuitive way with sufficient precision [24].

Since the model limits are defined using the swing-twist formalism, the orientation of the bone must be represented using this format. With Euler angles, the formalism can easily be obtained when the twist corresponds to the angle of the final rotation. However, care must be taken when using this type of conversion as the zero twist must in this case be well defined, otherwise, induced twist can easily occur [25]. For the quaternion case, a decomposition must be used where a single quaternion is split into a swing and twist component. This can be realised using the fact that the twist axis is in fact known and the swing axis is perpendicular to it. Appendix A describes this decomposition mathematically.

## 4.3.2 Bone Offset

Under normal circumstances, the software expects that a neutral orientation corresponds to the test person standing up straight and facing the north direction. However, when a sensor node is applied to a part of the human body, small misalignment errors between the sensor node and the limbs are inevitable. Tilt errors arise from the fact that the body is not flat and clothes are worn underneath the nodes. However, an offset is also generated by a person not exactly facing the North direction, which is mostly hard to avoid. Furthermore, this heading error can also differ between body parts since it is possible that not all nodes are facing the same direction when the reference position is assumed. Both of these offsets, tilt and heading, are bound to a different reference frame. The facing offset must be corrected in the earth bound reference frame and the misplacement of the nodes must be corrected in the sensor frame. This must be taken into account when determining the offsets.

Figure 4.5 shows a screen capture of the stickman visualisation before and after offset correction. It is clear that the original stickman is facing away from the screen and has limbs with tilt errors due to misalignment of the sensor nodes and roundness of the human body. The offsets correct this nicely to a stickman standing up straight with arms beside the body.

### 4.3.2.1 Euler Angles

Since the heading is the first rotation to be applied according to the roll-pitch-yaw convention introduced in section 2.2.1, this correction is automatically realised in the earth bound frame. The person should thus simply assume the reference position which, for each bone, will result in a certain set of Euler angles ($\phi_r$, $\theta_r$, $\psi_r$) that generally does not correspond to the neutral position

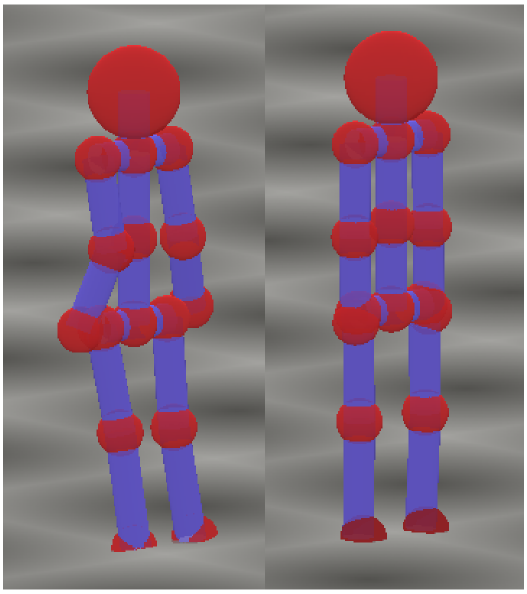


*Figure 4.5: Screen capture of the stickman before (left) and after (right) offset corrections are applied.*

given by $(0,0,0)$. Since the visualisation expects that the first set of angles should actually equal the latter when the reference position is adopted, the offsets can be set equal to the negative of $(\phi_r, \theta_r, \psi_r)$. From then on, all output angles can be validated by adding the offsets to the output of the filter.

#### 4.3.2.2 Quaternion

In the quaternion case, finding the offsets is more complex given the fact that the correction is performed by multiplication and the final orientation is obtained through a single rotation. When a certain quaternion $\mathbf{q}_r$ is found to be the output of the filter when the reference position is assumed, it needs to be corrected to the neutral orientation given by a unit scalar part and a zero vector part. Since the heading error is corrected in the earth bound frame and the tilt error in the sensor bound frame, the correction is performed by right multiplication of the quaternion with a heading correction and left multiplication with a tilt correction:

$$\mathbf{q}_{tilt} \otimes \mathbf{q}_r \otimes \mathbf{q}_{heading} = 1, \tag{4.1}$$

The reference output quaternion $\mathbf{q}_r$ should now be decomposed as follows to solve (4.1):

$$\mathbf{q}_r \;=\; \mathbf{q}_{tilt}^{-1} \otimes \mathbf{q}_{heading}^{-1} \;=\; \mathbf{q}_{tilt}^{*} \otimes \mathbf{q}_{heading}^{*}, \tag{4.2}$$

where the fact is used that the inverse of a unit quaternion equals its conjugate. Note that this decomposition can be accomplished by using the swing-twist decomposition [25] since the heading quaternion fulfills the role of a twist component corresponding to a rotation around the Z-axis and the tilt quaternion can be seen as a swing component with a rotation around an axis in the XY-plane. The mathematical derivation of this decomposition is given in the appendix section A.2. Application of (A.13) yields for the heading quaternion:

$$w_{heading} \;=\; \frac{\pm\, w_r}{\sqrt{w_r^2 \;+\; z_r^2}} \tag{4.3a}$$

$$z_{heading} \;=\; \frac{\mp\, z_r}{\sqrt{w_r^2 \;+\; z_r^2}} \tag{4.3b}$$

Where the duality in sign clearly reflects the fact that both a unit quaternion and its negative represent the same rotation. Note that contrast to (A.13), the signs of both coefficients must be different, reflecting the inversion present in (4.2). Also note that a singularity is present when both $w_r$ and $z_r$ equal zero, which must be taken into account when using the above equations. However, this situation occurs when $\mathbf{q}_r$ corresponds to a 180° rotation around an axis in the XY-plane which in turn means that the test person is standing upside down. Given the chosen reference stance, it is clear that the singularity is highly unlikely to occur.

The tilt quaternion can be found by rearranging terms in (4.2):

$$\mathbf{q}_{tilt} \;=\; \mathbf{q}_{heading}^{*} \otimes \mathbf{q}_r^{*} \tag{4.4}$$

Finally, quaternion orientations coming from the estimation filter can be validated by applying:

$$\mathbf{q}_{tilt} \otimes \mathbf{q}_{raw} \otimes \mathbf{q}_{heading} \;=\; \mathbf{q}_{corrected}. \tag{4.5}$$

### 4.3.3 Bone Corrections

At this point in the posture update, all bone orientations are converted to quaternion representation. The reason lies in the choice of correction method that is applied where the direction of the bone is corrected independently from the torsion. If Euler angles were used, this torsion would have to be applied as the final rotation, which would complicate the process model equations of the Kalman filter as gravity is also parallel to this initial torsion axis and the rotation around gravity should in fact be applied first. Otherwise, all three angles appear in the equation while the accelerometer readings should only control two of them given the fact that rotations around gravity cannot be detected.

The corrections that a bone undergoes depend on the constraints that the bone is subjected to. There are in fact four categories: free bones which remain uncorrected, fixed bones that follow their parent's orientation, single plane constraint bones whose connection to the parent is modeled by a hinge joint and multiple plane constraint bones that are connected through a ball-and-socket joint.

Although the idea of limiting swing and twist separately and the applied classification of bones is widely used, the actual correction strategies presented in this section are mainly new.

#### 4.3.3.1 Free Bones

Free bones have no restrictions on the orientation they adopt. When all bones are designated as free bones, no restrictions apply and each individual bone simply copies the orientation of the sensor node associated with it. This way, an unconstrained stickman is built.

In the human model presented here, only the root bone is left uncorrected. As this bone defines the overall stance of the stickman, it should be defined free to allow any possible position in general. Note that this way of working inherently means that errors in the orientation of the root are automatically included in the stickman posture. This can only be adjusted by changing the build-up of the stickman and correcting all of the bones together to a generally more feasible position instead of relying on recursive correction.

#### 4.3.3.2 Fixed Bones

The orientation a fixed bone takes on is copied from its parent bone. In general, these bones have no sensor node assigned to them because their movement is not included in the used human model. Although more elaborate human models might require a lot more bones to construct a better approximation of the human body, some of these bones can just be assigned to be fixed bones if less sensor nodes are available.

Both shoulder and hip bones are defined as fixed bones in the model. The shoulder bones take over the orientation of the upper torso, while the hips follow the lower torso. The lower torso itself is also defined as a fixed bone. Strictly speaking, a sensor node is assigned to this bone, yet the orientation from this sensor node is actually assigned to the root and is then transferred to the lower torso by defining it as a fixed bone.

#### 4.3.3.3 Single Plane Constraint Bones

Bones that are categorised as single plane constraint bones are connected to their parent via a hinge type joint. These types of joints only allow a single DOF in rotation around a well known axis. As can be seen in figure 4.6, valid orientations of these bones are only found in the plane that contains the parent and is perpendicular to the hinge axis. Furthermore, the angle between the parent and the bone, here denoted as $\gamma$, is also restricted to the range $[\gamma_{min}, \gamma_{max}]$. In the human body, the anatomical restrictions on both the knee and elbow joints comply with this type of model. Hence, both the lower legs and arms are classified as single plane constraint bones. Note that for both of these joints, a range of approximately 180° is valid, stretching from $\gamma_{min} = 0°$ to $\gamma_{max} = 180°$. This transforms the valid plane into a half-plane.

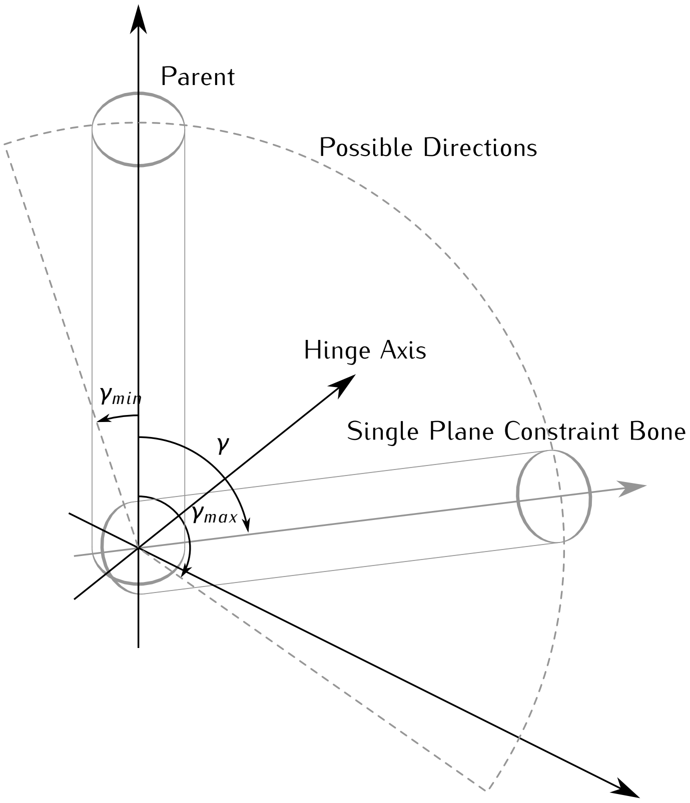


*Figure 4.6: Single plane constraint visualisation.*

As outlined in section 4.3.1, the correction is performed separately for swing and twist. The swing correction will attempt to place the bone inside the allowed plane, while the twist correction simply restricts the axial rotation of the limb. Since the lower arm consists of two bones, the radius and ulna [26], which are connected in a pivot joint at the wrist allowing a person to turn his or her hand to face different directions, a certain amount of twist is allowed for this bone. In the lower legs, the knee allows slight rotations around the axial direction. Hence, although the joints are modeled as hinges, the twist will actually model the joints with two DOF.

### Swing Correction

When the orientation of the corresponding sensor node is assigned to a single plane constraint bone, the swing component will generally not place the bone in the required half-plane. A correction method is needed to change this faulty direction to a valid one. Although strictly speaking, a hinge joint would indicate that the bone is part of the plane as defined in figure 4.6, a certain amount of tolerance will be allowed in order to make the correction smoother. This tolerance translates in the fact that the corrected bone may be part of any half-plane located in a wedge around the initially defined half-plane containing the possible directions according to the hinge. The tolerance is therefore defined as the angle between the half-plane perpendicular to the hinge axis and the most extreme members of the wedge. It will further be denoted as $\delta$ and is chosen to be 20°.

Figure 4.7 displays the correction of a bone when the swing component does not place it in one of the half-planes of the defined wedge. The direction of the parent $\boldsymbol{d}_p$, the associated hinge axis $\boldsymbol{a}_h$ and the direction of the corresponding sensor node $\boldsymbol{d}_s$ form the inputs of the correction method. The corrected direction $\boldsymbol{d}_c$ is then calculated by projecting it on a correction plane with normal vector $\boldsymbol{n}_c$ according to the following procedure:

1. Calculate the sensor axis $\boldsymbol{a}_s$ perpendicular to both the parent direction $\boldsymbol{d}_p$ and the sensor node direction $\boldsymbol{d}_s$:

$$\boldsymbol{a}_s = \boldsymbol{d}_p \times \boldsymbol{d}_s \tag{4.6}$$

2. If the angle $\alpha_s$ between $\boldsymbol{a}_s$ and $\boldsymbol{a}_h$ is smaller than $\delta$, the sensor node direction $\boldsymbol{d}_s$ lies within one of the half-planes of the wedge and the corrected direction $\boldsymbol{d}_c$ is found to be simply $\boldsymbol{d}_s$. In this case the procedure ends, otherwise, more steps are required.

3. The normal of the correction plane $\boldsymbol{n}_c$ can now easily be determined by remarking that the hinge axis $\boldsymbol{a}_h$ is in fact the normal to the theoretical plane that should contain the bone according to the hinge. However, given the fact that some tolerance will be incorporated, $\boldsymbol{a}_h$ will be rotated

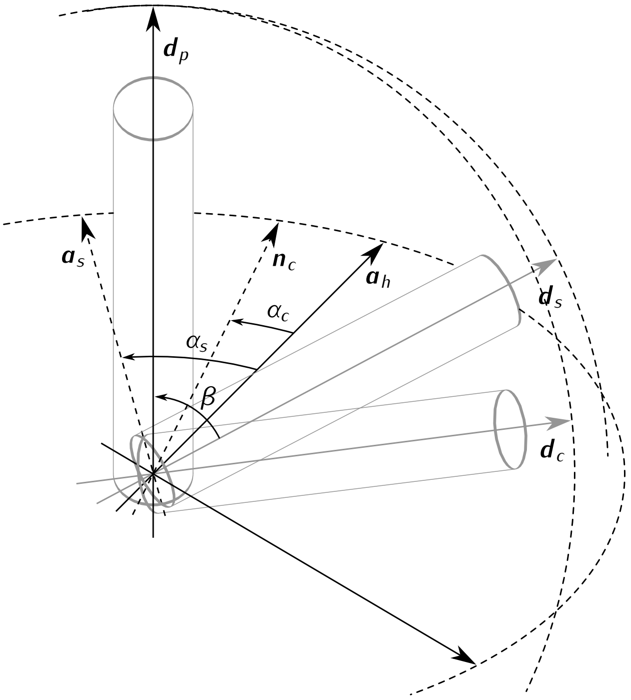


*Figure 4.7: Single plane constraint correction.*

towards $\boldsymbol{a}_s$ around $\boldsymbol{d}_p$ and over a correction angle $\alpha_c$ to find the normal to the correction plane $\boldsymbol{n}_c$. In order to ensure a smooth correction, $\alpha_c$ will also depend on the actual error angle $\alpha_s$ between $\boldsymbol{a}_s$ and $\boldsymbol{a}_h$:

$$\alpha_c \; = \; \delta \; + \; (\alpha_s \; - \; \delta) \; \epsilon, \tag{4.7}$$

where $\epsilon$ represents the fraction of the error angle $\alpha_s$ that is added to the correction angle $\alpha_c$. This parameter is referred to as the *elasticity* and equals 0.2.

4. The sensor direction $\boldsymbol{d}_s$ can now be projected onto the correction plane to yield the corrected direction $\boldsymbol{d}_c$:

$$\boldsymbol{d}_c \; = \; \boldsymbol{d}_s \; - \; (\boldsymbol{d}_s \cdot \boldsymbol{n}_c) \, \boldsymbol{n}_c \tag{4.8}$$

   This direction can finally be normalised if desired.

5. In the case that the angle between the sensor axis $\boldsymbol{a}_s$ and the correction plane normal $\boldsymbol{n}_c$ is larger than 90°, projecting the sensor direction $\boldsymbol{d}_s$

onto the correction plane results in a direction located in the wrong half-plane. The resulting $\gamma$ angle will in this case not be located in the bounds that were put forward and the visualisation will show an overstretched elbow or knee. This case occurs whenever the following inequality holds:

$$\alpha_s - \alpha_c > 90° \tag{4.9}$$

Substituting the expression for $\alpha_c$ from (4.7) and solving for $\alpha_s$ yields:

$$\alpha_s > \frac{90° + \delta\,(1 - \epsilon)}{1 - \epsilon} = 132.5° \tag{4.10}$$

Hence a large orientation error should be present to trigger such a wrongful projection. The corrected bone direction $\boldsymbol{d}_c$ will in this case either be set to $\boldsymbol{d}_p$ when the angle $\beta$ between $\boldsymbol{d}_p$ and $\boldsymbol{d}_s$ is smaller than 90°, or $-\boldsymbol{d}_p$ otherwise.

Figure 4.8 shows a screen capture of the stickman before and after swing corrections are applied. The upper right arm and leg are drawn in the standard upright position and are facing forward. In this case, the orientations visualised on the left are anatomically impossible. The errors are corrected and shown on the right giving a posture that is much more feasible. Note that corrected bones are drawn in green for visual feedback. Also, the constraints are visualised by drawing the hinge axis $\boldsymbol{a}_h$ in turquoise and the correction plane normal $\boldsymbol{n}_c$ in purple. For the knees, the hinge axis is located in the plane of the stickman figure, while for the elbows, it forms an angle of 30° with this plane since a person in the reference stance also shows an outward rotation of the elbows.

### Twist Correction

After the swing correction has been executed, the twist still needs to be validated. However, the boundaries of the twist also depend on the twist of the parent bone, much as the direction of the hinge axis also depends on the swing of the parent. Furthermore, in order to assure smooth visualisation, the elasticity factor $\epsilon$ is also used here. The following procedure is finally obtained:

1. Calculate the difference in twist between the parent and the bone under consideration. If the absolute value of this difference exceeds the twist bound, the twist must be corrected, otherwise no adjustments are required and the procedure ends.

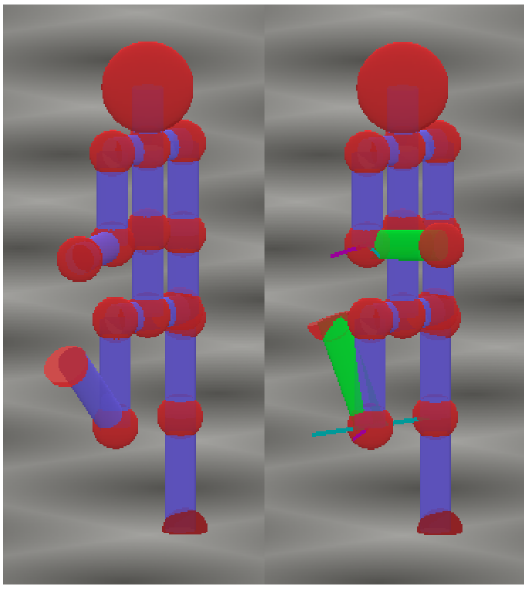


*Figure 4.8: Screen capture of the stickman before (left) and after (right) corrections to the swing of the lower right arm and leg have been applied.*

2. Denoting the corrected twist angle by $\tau_c$, the parent twist angle by $\tau_p$, the original twist angle by $\tau_s$ and the twist bound by $\tau_b$, the following formulas are applied to correct the twist angle:

$$\tau_c = \begin{cases} \tau_p - \tau_b + \epsilon \left(\tau_s - \tau_p + \tau_b\right) : & \tau_s < \tau_p \\ \tau_p + \tau_b + \epsilon \left(\tau_s - \tau_p - \tau_b\right) : & \tau_s > \tau_p \end{cases} \tag{4.11}$$

   The twist angle is corrected by starting from the parent's twist and moving towards the original twist angle over the bound value and a fraction of the twist error.

Since none of the single plane constraint bones have any child bones associated to them and the visualisation of the bones is circular symmetric in the axial direction, this correction is barely visible. Therefore, the feet of the stickman have been drawn in such a way that they are no longer circular symmetric and clearly show the twist on the lower leg bones. Figure 4.9 shows the stickman before and after twist correction on the lower right leg. On the left, the foot is clearly pointing backwards, which is a very unnatural pose. By limiting the axial rotation as described by (4.11), the foot assumes an anatomically correct direction. Note that when the twist is corrected, the end joint of the bone is drawn in green to offer clear visual indication of the correction. For the knees, the torsion bound is chosen to be 20°, while the elbows have a more liberal 90° limit given the higher amount of axial rotation allowed by the wrist.

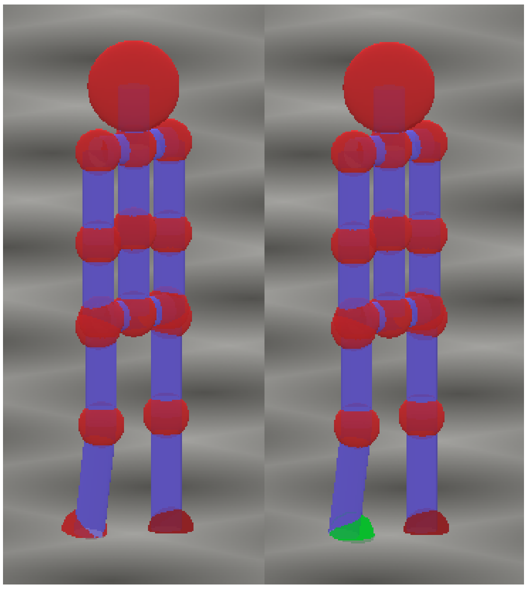


*Figure 4.9: Screen capture of the stickman before (left) and after (right) corrections to the twist of the lower right leg have been applied.*

#### 4.3.3.4 Multiple Plane Constraint Bones

The multiple plane constraint bones are connected to their parent through a ball-and-socket type joint. Therefore, the child bone is capable of rotation around an infinite number of axes who have a common center in the middle of the ball and the joint exhibits three DOF. However, not all orientations are feasible as the joint also restricts the movement. The directions that are anatomically impossible are therefore modeled by a cone, as can also be seen in figure 4.10. This cone is characterised by its center axis direction and the angle at the top. Given the fact that the joints have three DOF, they also allow a certain amount of twist. This twist must however also be bound as was the case for the single plane constraint bones.

In this human model, both the upper arms and legs, as well as the upper torso and the neck with the head are categorised as multiple plane constraint bones. The hips and shoulders clearly conform to the ball-and-socket joint

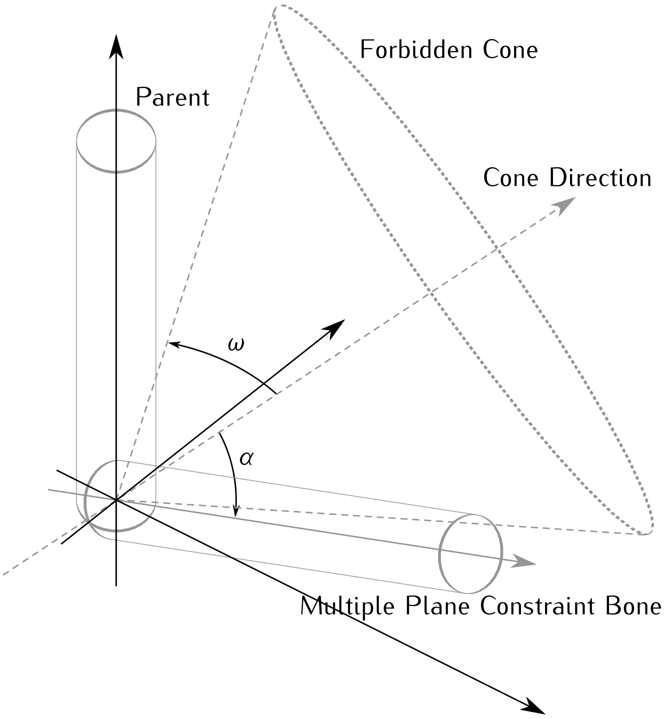


*Figure 4.10: Multiple plane constraint visualisation.*

definition. The neck joint and the joint between the upper and lower torso are in fact more complex as they depend on joints in the vertebral column, but they can both be approximated by the model presented here.

### Swing Correction

Figure 4.11 shows the swing correction of a bone whose sensor node direction lies within the forbidden cone. The correction method disposes of the parent direction $\boldsymbol{d}_p$, the corresponding cone axis $\boldsymbol{a}_c$ and the direction of the associated sensor node $\boldsymbol{d}_s$ to calculate the corrected direction $\boldsymbol{d}_c$:

1. Calculate the angle between the sensor direction $\boldsymbol{d}_s$ and the center axis of the cone $\boldsymbol{a}_c$:

$$\alpha_s = \arccos(\boldsymbol{d}_s \cdot \boldsymbol{a}_c) \tag{4.12}$$

   If the resulting angle is smaller than the top angle of the cone $\omega$, correction of the swing is required since the bone is located within the

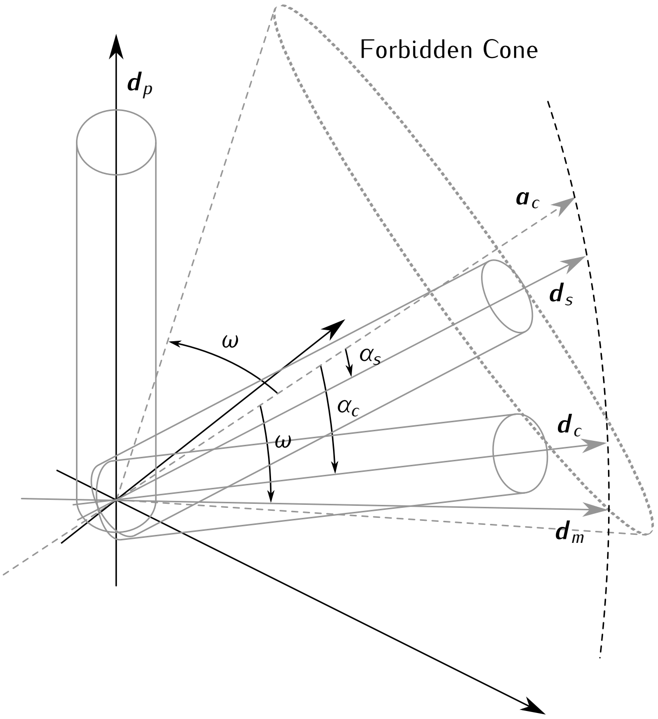


*Figure 4.11: Multiple plane constraint correction.*

forbidden cone. Otherwise, the procedure ends and the corrected direction $\boldsymbol{d}_c$ equals $\boldsymbol{d}_s$.

2. The projection of the sensor node direction $\boldsymbol{d}_s$ on the cone mantle is calculated by rotating the cone axis $\boldsymbol{a}_c$ over the cone angle $\omega$ around the axis that is perpendicular to both of them:

$$\boldsymbol{d}_m = \mathrm{R}^{\omega}_{\boldsymbol{a}_c \times \boldsymbol{d}_s}(\boldsymbol{a}_c) \tag{4.13}$$

3. In order to provide smoother correction rather than enforcing a strict boundary, the elasticity is used to drag the projected direction into the forbidden area by a fraction of the error:

$$\boldsymbol{d}_c = \boldsymbol{d}_m + \epsilon\,(\boldsymbol{d}_s - \boldsymbol{d}_m) \tag{4.14}$$

Finally, this corrected direction may be normalised if desired.

Figure 4.12 shows a screen capture of the stickman before and after corrections have been applied to the upper torso and the upper right leg. On the left capture, the upper torso forms a 90° angle with the lower torso which is anatomically impossible. Moreover, the upper right leg is directed way to far backwards when considering the upright position of the pelvis and lower torso. The right capture shows the corrected version where the upper right leg has been dragged forward and the upper torso is forced more upright, giving the stickman a more feasible pose. The lower right leg has also been corrected as described in the previous section to avoid overstretching of the knee joint. Furthermore, the forbidden cones are shown in yellow in the right part of the figure. Note that the top angle of the cone can also exceed 90°, as is the case for the upper torso. The angles and axis directions of the cones for each of the multiple constraint bones used in the human model are listed in table 4.1. The axis coordinates are given in a reference frame where the Z-axis points upward, the Y-axis forward and the X-axis to the left when looking at the front side of the stickman.

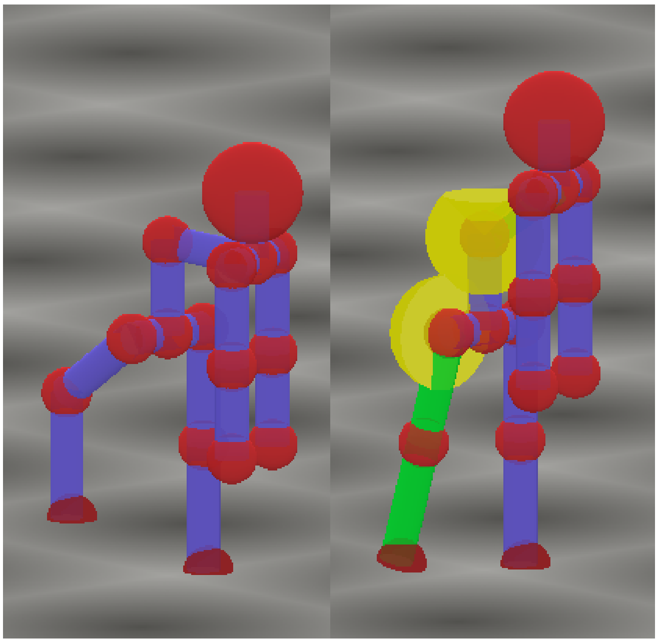


*Figure 4.12: Screen capture of the stickman before (left) and after (right) corrections to the swing of the upper right leg and the upper torso have been applied.*

| Bone | Top angle $\omega$ | Axis direction $\boldsymbol{a}_c$ |
|---|---|---|
| Head | 90° | [0, 0, -1] |
| Upper torso | 135° | [0, 0, -1] |
| Left Upper Leg | 85° | [0, -sin 85°, cos 85°] |
| Right Upper Leg | 85° | [0, -sin 85°, cos 85°] |
| Left Upper Arm | 67.5° | [cos 67.5°, -sin 67.5°, 0] |
| Right Upper Arm | 67.5° | [-cos 67.5°, -sin 67.5°, 0] |

*Table 4.1: Cone top angles and axis directions given in the reference frame with Z-axis pointing upward, Y-axis forward and X-axis left when looking at the front of the stickman.*

#### Twist Correction

The twist correction of multiple constraint bones follows the same procedure as was outlined for single constraint bones since the restrictions on twist are in fact the same for both types of bones. The twist bound $\tau_b$ is set to a rather liberal 90° value for all of the bones designated as multiple constraint bones.

### 4.3.4 World Model

At first, the root of the stickman was always drawn in the middle of the screen and the entire stickman was drawn by recursively adding all of the child bones and connecting them to their parents. The result however, is a stickman that looks as if it is hung up at the root point. When e.g. the tracked person is bowing, the stickman will rotate around the root point, moving both its upper and lower body in opposite directions. A better approach is to add a world model to create a more realistic visualisation of the movements.

The model that is used is fairly simple and consists only of a flat surface drawn at a fixed height representing a floor. It is then assumed that at least one of the bones should touch the floor and that this must be the bone whose endpoint has the lowest vertical component in its coordinates. Note that the endpoint is chosen rather than the starting point as each bone starts at the endpoint of its parent, while not every endpoint corresponds to the starting point of another node since some of the bones do not have any child bones.

The update procedure for the world model is as follows. Before the stickman is drawn, the bone with the lowest endpoint is first determined, as well as the coordinates of this endpoint in the case that the root is drawn in the middle of the screen. The vertical component of this position is then used to adapt the height of the point where the root is drawn in order to clip the endpoint of the lowest bone to the floor. The horizontal components are used to adapt the texture mapping on the floor such that the location where the endpoint touches the floor does not change. This way, it looks as if the lowest endpoint is stuck to the floor at the point where it first coincided with it.

This simple world model introduces a higher degree of realism to the visualisation of the stickman as e.g. walking will actually move the stickman around the floor. However, it also imposes some important restrictions to the movements that are correctly reconstructed and to the real world where the tracked person is allowed to move in. The person should indeed always have at least one contact point with the floor which quickly rules out jumping and even running to some extend. Furthermore, the floor should be entirely flat and fully horizontal, an inclined floor or differences in level could imply that the lowest bone is not always the one that is in contact with the floor.

## 4.4 Conclusion

The general structure of the program was first outlined by giving an overview of the different windows that make up the entire program. The welcome window allows the user to select the correct generation of hardware that will be used and start the appropriate device window. These windows allow data to be visualised using either a graph, a rotating cube or an animated stickman. They also allow to start a calibration window when required. Following this overview, the data flow through the application is analysed. The received data is passed on from one class to the next while several operations convert the received bytes into a valid orientation.

The second part of this chapter describes the human model that has been implemented in order to correct anatomically impossible postures into feasible ones. The recursive nature and tree structure of the bones that make up the stickman are discussed and the swing-twist parameterisation for joints is introduced as a means for correcting orientations. Next, the problem of misalignment errors between the sensor node and the bone is tackled by applying offsets to the orientation in either Euler angle or quaternion form. Bones are divided into four categories depending on the amount of freedom allowed by the joint that connects them to their parent. Free bones are never corrected and simply copy the orientation from their associated sensor node, while fixed bones copy their parent's orientation. The joint between a single plane constraint bone and its parent is modeled as a hinge and thus only allows the swing to move the bone into a single half-plane. Tolerance on this output transforms it to a wedge of half-planes smoothing the visualisation. Multiple constraint bones are connected to their parent through a ball-and-socket joint where swing restrictions are modeled using a cone of forbidden directions. In both cases the twist of the bone is also smoothly bound using an elasticity parameter. Finally, a world model is described where the bone with the lowest endpoint is clipped to an entirely flat floor.

# References

[1] N. Decraene. *Electronics for Movement Analysis in Medical Applications*. Master's thesis, Ghent University, Ghent, Belgium, 2007.

[2] F. Balena. *Programming Microsoft Visual Basic 6.0*. Microsoft Press, May 1999.

[3] D. Grundgeiger. *Programming Visual Basic .NET*. O'Reilly Media, May 2002.

[4] H. Lam and T.L. Thai. *.NET Framework Essentials*. O'Reilly Media, February 2002.

[5] Microsoft. *Microsoft Brings Programming to the Masses With Visual Studio Express*, April 2006. http://www.microsoft.com/presspass/press/2006/apr06/04-19VSExpressFreePR.mspx.

[6] R.S. Wright, B. Lipchak, N. Haemel, and G. Sellers. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley, July 2010.

[7] Microchip. *ENC28J60, Stand-Alone Ethernet Controller with SPI Interface*, January 2008. http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf.

[8] Digital Equipment, Intel, and Xerox. *The Ethernet, A Local Area Network, Data Link Layer and Physical Layer Specifications*, September 1980.

[9] J. Postel. *User Datagram Protocol*. RFC 768 (Standard), August 1980.

[10] J. Postel. *Internet Protocol*. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[11] Silicon Laboratories. *CP2102, Single-Chip USB to UART Bridge*, October 2004. http://www.silabs.com/pages/DownloadDoc.aspx?FILEURL=Support%20Documents/TechnicalDocs/cp2102.pdf&src=DocumentationWebPart.

[12] Future Technology Devices International. *FT232R, USB UART IC*, January 2006. http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf.

[13] *USB.org, Universal Serial Bus*. http://www.usb.org/.

[14] Roving Networks. *RN-41: Class 1 Bluetooth Module*. Los Gatos, CA, USA, August 2009. http://www.rovingnetworks.com/documents/RN-41.pdf.

[15] *Bluetooth.org, Official Bluetooth Info Site*. http://www.bluetooth.com/English/Pages/default.aspx.

[16] *EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Data Interchange*, August 1969.

[17] *The Tao Framework*. http://sourceforge.net/projects/taoframework/.

[18] E.P. Hanavan. *A Mathematical Model of the Human Body*. Technical Report TR-64-102, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, OH, USA, 1964.

[19] W. Mistiaen and S. Van Schoote. *Motion Recognition and Adaptation for 3D Visualisation using Wearable MEMS Sensors*. Master's thesis, Ghent University, Ghent, Belgium, 2010.

[20] U.Y. Usta. *Comparison of Quaternion and Euler Angle Methods for Joint Angle Animation of Human Figure Models*. Master's thesis, Naval Postgraduate School, Monterey, California, USA, 1999.

[21] D. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, chapter Section 2.3: Trees, pages 308–Ű423. Addison-Wesley, 1997.

[22] F.S. Grassia. *Practical Parameterization of Rotations using the Exponential Map*. Journal of Graphics Tools, 3(3):29–48, 1998.

[23] X.G. Wang, F. Mazet, N. Maia, K. Voinot, J.P. Verriest, and M. Fayet. *Three-Dimensional Modelling of the Motion Range of Axial Rotation of the Upper Arm"*. Journal of Biomechanics, 31(10):899–908, 1998.

[24] L. Herda, R. Urtasun, P. Fua, and A. Hanson. *An Automatic Method for Determining Quaternion Field Boundaries for Ball-and-Socket Joint Limits*. In Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition, pages 88–93, may 2002.

[25] P. Baerlocher and R. Boulic. *Parameterization and Range of Motion of the Ball-and-Socket Joint*. In Proceedings of Avatars, 2000.

[26] H. Gray. *Anatomy of the Human Body*. Lea & Febiger, Philadelphia, 1918.

# 5 Measurements

The performance of the tracking system is analysed in this chapter. Quantitative results are obtained using a single axis rotational stage and full body experiments validate the functionality of the entire system.

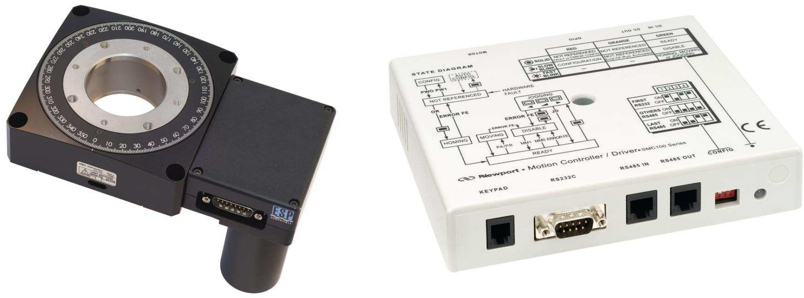
## 5.1 Introduction

After outlining all of the aspects involved in the design of an inertial sensor based tracking system, the performance can finally be analysed. To this extend, the actual orientation to which a sensor node is subjected must be known. One approach to achieve this is to position the node according to a well known orientation and to compare this to the output of the tracking filter. A second approach is to compare it to the output of another tracking system that serves as a golden standard since it has proven functionality. In the following, both approaches are used to characterise the performance.

In the first part, the individual performance of a single node is quantified by placing it on a uniaxial rotational stage and comparing the output of the filter with the angular position of the stage. In the second part, full body tracking experiments are outlined where the test subject is simultaneously equipped with sensor nodes and with the reflective markers to allow tracking by an optical camera system. The latter will in this case be used as a reference given the maturity these systems have gained throughout the years.

## 5.2 Individual Node Performance

The motorised rotational stage that is used for measurements is displayed in figure 5.1a. The Newport URB100CC [1] is a belt driven rotational stage powered by a DC servo motor. It features a continuous full circle travel range and allows speeds of up to 720 °/s or two revolutions per second. A controller is required in order to drive the stage and supply the correct input voltage. The SMC100CC [2] depicted in figure 5.1b is the single axis controller that was chosen for the task. It allows the stage to be controlled from a PC or by an optional remote control.



(a) Rotational stage URB100CC

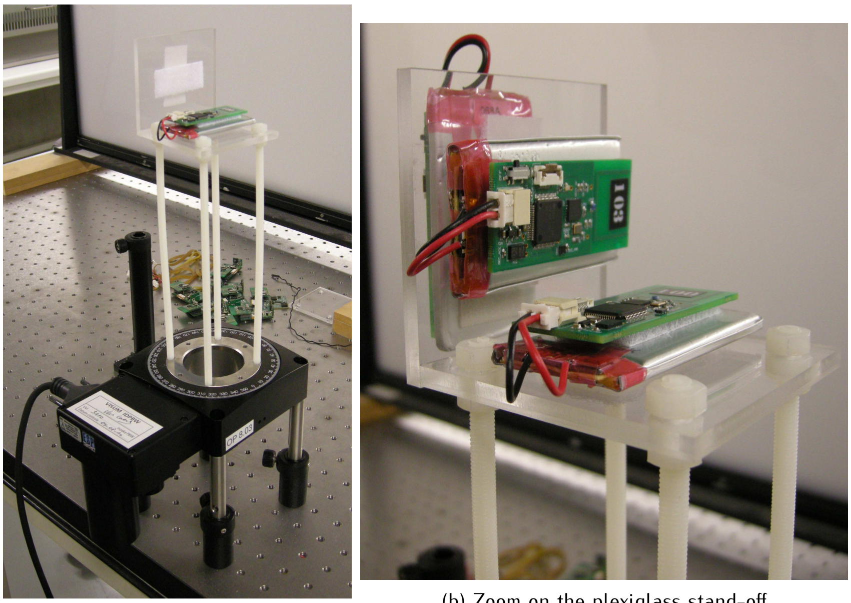
(b) Stage controller SMC100CC

*Figure 5.1: Rotational stage and controller from Newport.*

The measurement setup is shown in figure 5.2. A plexiglass stand-off was constructed in order keep the magnetometer at a certain distance from the rotational stage since it is made entirely out of metal. The stand-off consists of two plates atop four nylon screws of 20 cm length. One of the plates is attached to the screws using hexnuts while the other is glued perpendicular to the side of the first in order to allow a node to be placed vertical as well. Several velcro strips have been applied to the stand-off which can accommodate one or more nodes.

The rotational stage is connected via the driver to a PC allowing direct control of the stage as well as retrieving information from it such as current position, speed or state. The base station of the tracking system is also connected to the same PC such that both information sequences can easily be merged. A program has been written in VB.NET [3] to accomplish this and at the same time control the stage's movement.

In this section, several performance characteristics of the tracking system, such as linearity, step response and maximal angular speed are determined



(a) Rotational stage setup

(b) Zoom on the plexiglass stand-off

*Figure 5.2: Measurement setup with the rotational stage.*

when the node is rotated around one of its sensitivity axes. A distinction is made between heading and tilt response since it is expected that heading changes will be harder to track as was also reflected by the simulation results in chapter 2.

First, the performance of the different MFG filters is analysed and compared to each other. Quaternion outputs are converted to Euler angles to allow meaningful comparison. Afterwards, the embedded filter implementation is tested in the exact same way as the real time software version. Finally, some characteristics of the reference MARG filter are deducted.

Before the sensor outputs are used for estimation, they are processed according to figures 2.5 and 2.6. The calibration values are obtained using the procedure outlined in section 2.4.2.2 using the rotational stage. Also, the digital filter designed in section 2.5.1.2 is used. Unless explicitly mentioned otherwise, the parameters of the MFG filters used to obtain the results in this section are set to the optimal values determined in sections 2.5 and 2.6. Specifically, the measurement noise covariance matrix is given by (2.129):

$$R = \begin{bmatrix} 2\times10^{-5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 2\times10^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 2\times10^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1\times10^{-5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1\times10^{-5} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1\times10^{-5} \end{bmatrix}. \tag{5.1}$$

The feedback gain parameter $\tau$ is set to 0.8, which means that the process noise needs to be adapted to this chosen value as was seen by the graphs in figures 2.14 and 2.16. Therefore, the following matrices have been selected:

$$Q_{Euler} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \quad Q_{Quaternion} = \begin{bmatrix} 10^{-6} & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 \\ 0 & 0 & 0 & 10^{-6} \end{bmatrix} \tag{5.2}$$

Finally, the adaptive parameters $\zeta$ and $\xi$ are set to 1000 although it is expected that, since only rotations will be executed in a controlled environment, these parameters will have little influence on the output of the filters.

### 5.2.1 Linearity

In order to obtain the linearity over the entire travel range of 360° a stepping test was used. This test consists of a single revolution performed in several smaller steps. In between the steps, the stage is kept stationary for a certain amount of time. This test can also be used to determine the offset the output of the filter exhibits in comparison with the stage's angular position output. In the heading case, this offset is of major importance as the zero position of the rotational stage might not correspond to the sensor node pointing north.

The stepping test used to obtain the subsequent results has a step size of 10° and a stationary period of approximately 2 s.

Considering the linearity of the sensor devices present on the boards, a linearity error of a couple of degrees would be normal. As a realistic target specification, a 5° non linearity can be set.

#### 5.2.1.1 Heading

Figure 5.3 displays a time graph of the output of the Euler type UKF versus the rotational stage position in a stepping test. At first sight, the linearity looks pretty good, although there are some small deviations that can be seen. The graph is slightly bent in such a way that the estimation lies above the rotational stage output in the beginning and at the end of the sequence and underneath it in the middle.

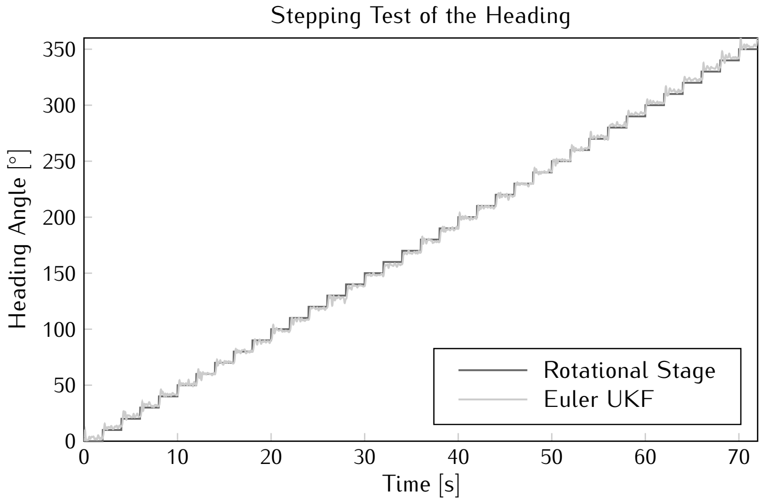


*Figure 5.3: Stepping test output sequence for the Euler type UKF.*

The linearity of the filter output can be analysed by applying linear regression analysis [4]. First, the graph given in figure 5.3 is reduced to points by averaging the output of the filter over the intervals of static orientation. In order to avoid any influence of the transient response due to the steps, the mean is calculated only over the second half of these periods. Figure 5.4 shows the resulting samples as dots together with a line corresponding to the linear approximation. The equation representing the line is given by:

$$y = 1.0056\,x - 0.5602. \tag{5.3}$$

This shows that the rotational stage output and the filter estimation match really well. The error between the linear approximation and the actual output is given in figure 5.5. From this graph it is easily understood that the error is systematic given the sinusoidal form. This implies that the source for the error depends on the orientation and must be present in the input of the filter. Indeed, the origin is found in the calibration of the sensor outputs, since the filter's estimations are based on the comparison of these sensor readings to known values. It turns out that the linearity of the filters in the heading rotation entirely depend on the calibration of the magnetometer, which is logic given the fact that the accelerometer does not supply any information about heading and its output is independent from the heading angle.

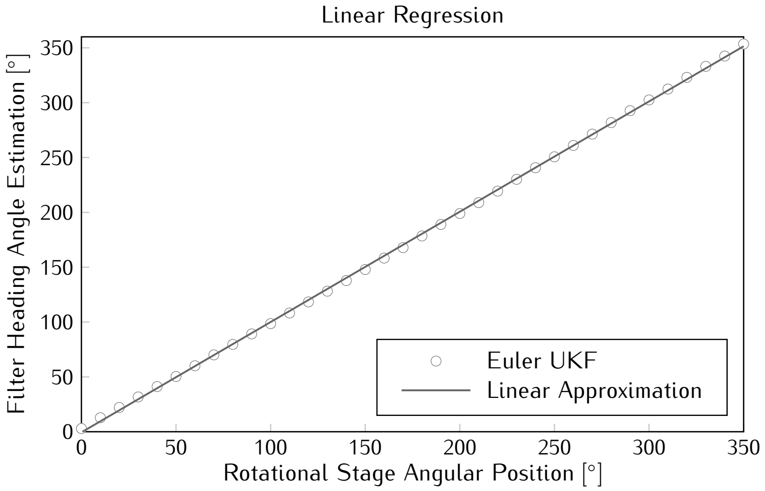


*Figure 5.4: Linear regression applied to the mean output of the filter during a stepping test on the heading angle.*

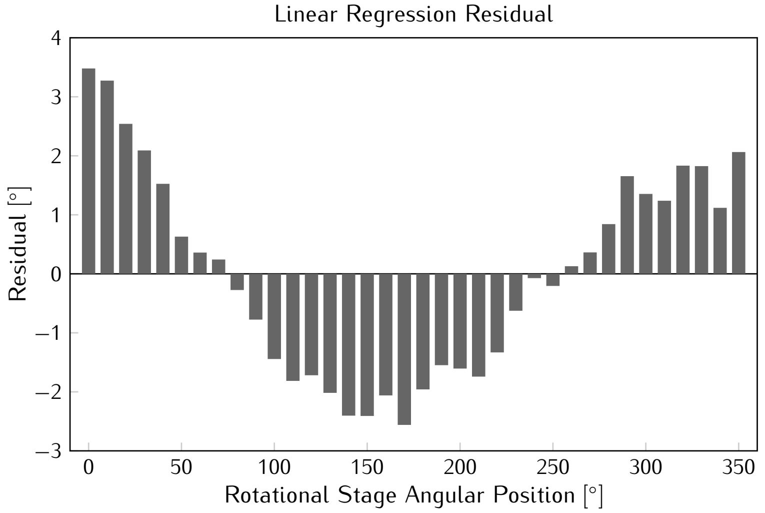


*Figure 5.5: Linear regression residual on heading.*

Figure 5.6 shows the linear regression analysis when an error is introduced in the magnetometer calibration on the X and Y-axis bias value. The bias was increased by 0.05, a dimensionless number given the fact that the output is normalised to unity. Important to mention is that the resulting residual has increased significantly and shows peaks up to 15°, even though the introduced bias error is only very small. It is clear that a careful calibration procedure is required to obtain valid tracking with good linearity.

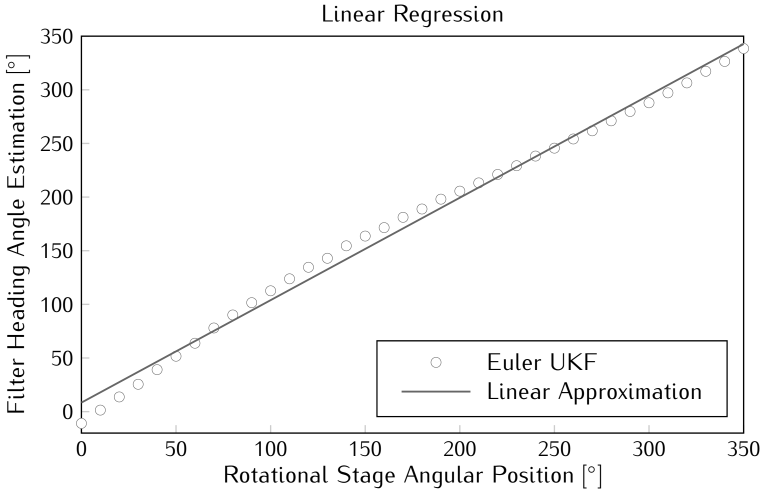


*Figure 5.6: Linear regression of the filter output when calibration errors were introduced.*

### 5.2.1.2 Tilt

Applying linear regression to a tilt stepping test results in the graph shown in figure 5.7. The equation corresponding to the linear approximation is given by:

$$y = 0.9962\,x - 0.9805. \tag{5.4}$$

The corresponding residual is depicted by the bar graph of figure 5.8. The residual also displays a certain sinusoidal form, yet it is less obvious than in the heading case. Also, the mean absolute error is only a third of the heading linearisation mean absolute error. All these observations are more or less expected since the tilt change is reflected in both sensors' output signals and the magnitude of change is also larger. This means that errors in the calibration now have less effect on the linearity as one sensor can compensate for the other.

## 5.2.2 Step Response

The step response is analysed using a block type test. This test is related to the stepping test but in this case a forward step is followed by a backward

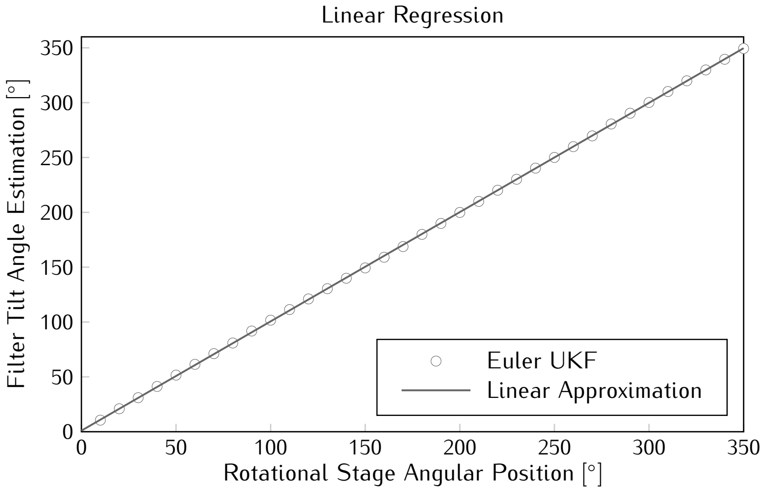


*Figure 5.7: Linear regression applied to the mean output of the filter during a stepping test on the tilt angle.*

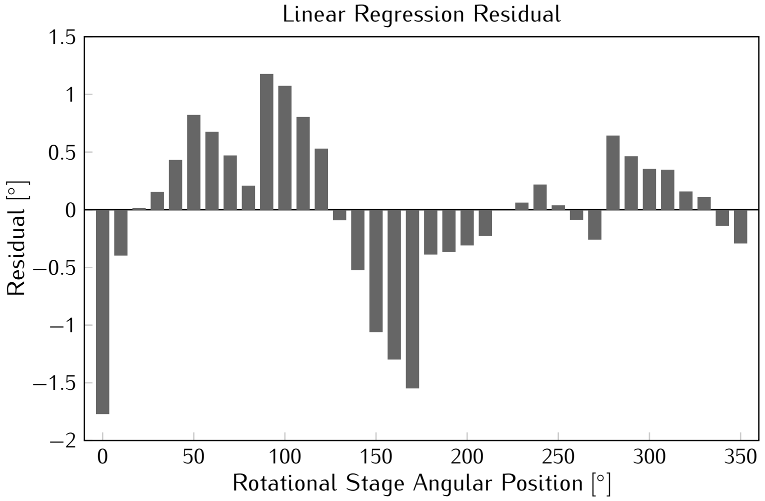


*Figure 5.8: Linear regression residual on tilt.*

motion over the same travel range. Again, in between both movements, the stage is kept stationary. Note however that since the stage will actually be moving, the step is not instant as was the case in simulation. Therefore, larger settling times are to be expected. As a realistic target for the delay between the rotational stage and the filter estimations, 100 ms or about 10 update cycles seems reasonable.

In order to relate easily to the simulations, the travel range of the block is set to 90°. In between motions, a rest period of approximately 4 s is added to allow settling of the filter output. The block is performed five times in each test and a total of five individual tests are analysed to obtain an average settling time for both forward and backward motion.

#### 5.2.2.1 Heading

**Euler Angle**

Figure 5.9 shows a time graph of both the rotational stage position and the Euler type filters' output. The graphs for both the UKF and CDKF coincided, so only one graph is drawn. The step itself is initiated at the 4 s marker, yet it takes some time until the stage starts moving. The figure clearly indicates that the rotational stage itself takes approximately 400 ms after this starting point to reach the final position. This must be taken into account when comparing the resulting settling times with the simulation results obtained in section 2.6.1.2, since this delay is not present there.

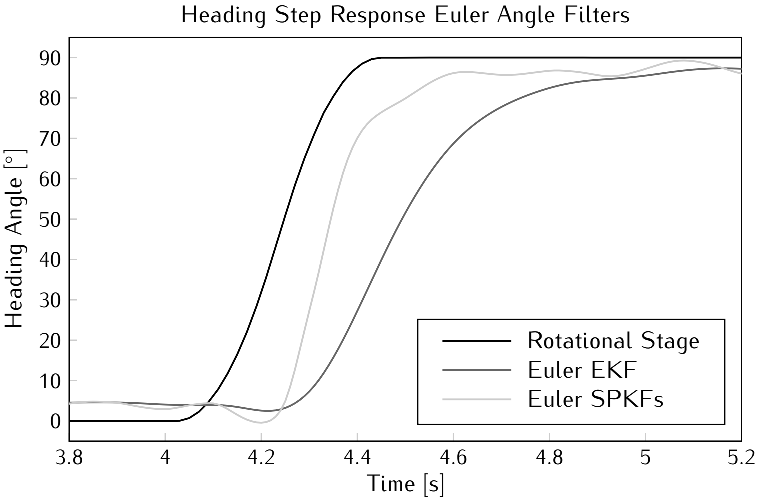


*Figure 5.9: Heading step response of the Euler type filters.*

From multiple block sequences, an average 10 % settling time of 480 ms was obtained for the SPKFs and 767 ms for the EKF, the standard deviation to these values was 35.7 and 26.7 ms respectively. It must be noted that the difference in performance between both filters is clearly smaller than was the case in simulation. However, the step input here is not instant and the main difference between both filters is the steepness of the step response. Furthermore, the delay of the EKF noted here is essentially noticeable and even troublesome when it is used in realtime. Taking into account the transient time of the rotational stage, the SPKFs reach within 10 % of the final value after about 80 ms, while the EKF takes 360 ms. This indicates that the delay of the extended version is about four to five times the delay of the SPKFs.

The sigma point filters' response is clearly faster, yet the steepness seems to decrease when the rotational stage decelerates before stopping at the 90° angular position. Given the fact that the sensor values are no longer changing when the rotational stage comes to a full stop, the error between the predicted and the actual measurement will decrease. The result is that the Kalman gain will start to decrease and the response of the filter slows down. This effect is however not seen in the EKF response, since the sensitivity of this filter is much lower and the error remains large when the rotational stage reaches 90°. The output is hence much smoother, but also less steep.

As was the case in simulation, peaks are found on the tilt angle output of the filter. These peaks find their origin in the fact that the rotation takes place around the gravity vector, which means all estimations are based on the magnetometer's output signals. The high sensitivity of the filter however, causes changes on the other angles as well. In this case the peaks are about five times smaller compared to the simulation. The reason lies in the slower transition from a zero angle to the final position, which results in a more gradual change in output angles.

### Quaternion

The 10 % mean settling times for the quaternion filters and their standard deviations are given in table 5.1. It is clear that all filters have a similar step response and perform as well as the sigma point versions of the Euler angle type filter.

| | EKF | SPKFs |
|---|---|---|
| Mean [ms] | 478 | 484 |
| Standard Deviation [ms] | 24.9 | 29.9 |

*Table 5.1: Quaternion type filters settling time to a heading step.*

A comparison of both Euler and quaternion type filters is given in figure 5.10. The output of the quaternion filters has been converted to Euler angles for the comparison to make sense. As can be seen, the differences between the filters are very small. The response of the Euler SPKFs and quaternion EKF almost coincide, while the quaternion SPKFs exhibit a slightly steeper response. However, this advantage does not lead to a lower delay as the other filters seem to respond better when the stage reaches its final destination. Since all filters display similar responses, it is also clear that the sigma point version of the quaternion filter yields an increase in complexity, while offering very little advantage over the extended filter. This validates the choice to embed the quaternion EKF in the firmware of the sensor nodes.

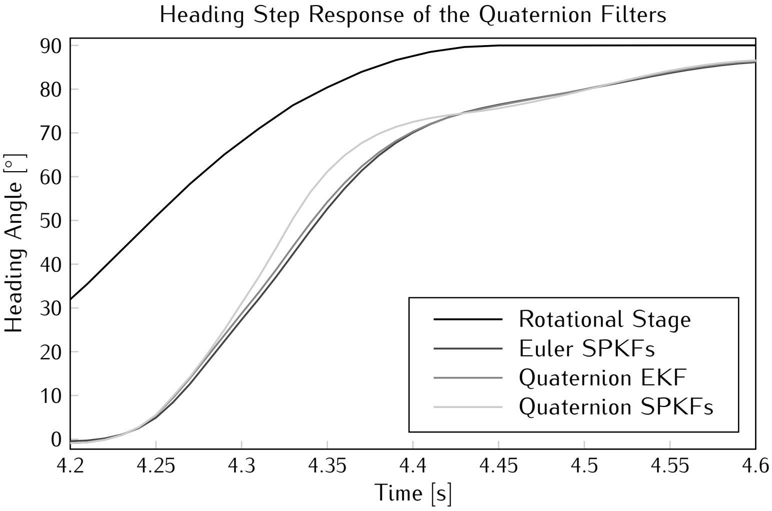


*Figure 5.10: Heading step response of the Euler and Quaternion filters.*

### 5.2.2.2 Tilt

The 10 % mean settling times of the filter output response to a tilt step and their standard deviations are given in table 5.2. The trend of an underperforming Euler type EKF is again obvious. However, contrast to the expectations, only the Euler EKF performs significantely better for a tilt step compared to a heading step. The standard deviations are comparable, but for the Euler EKF it is much lower, hinting to a more uniform response due to the larger magnitude of the sensor signals.

| | Euler | | Quaternion | |
|---|---|---|---|---|
| | EKF | SPKFs | EKF | SPKFs |
| Mean [ms] | 606 | 469 | 462 | 454 |
| Standard Deviation [ms] | 7.8 | 29.4 | 27.3 | 31.8 |

*Table 5.2: Filter settling time to a tilt step.*

Figure 5.11 displays a time graph of the tilt step response of each of the filters. Comparing this result to the graphs in figures 5.9 and 5.10 explains why the settling time is almost equal. In the heading case, the filter needs more time to detect which angle should be changed. This can be seen by looking at the point where the filter response starts increasing, in the heading case this only occurs at timestamp 4.2 s, while in the tilt case the angle is already increased to 10° at this point. From this point on to approximately timestamp 4.4 s, the heading response reaches a higher steepness than is the case for the tilt step and both end at about 65° when the rotational stage stops moving. The main reason for the difference in steepness lies in the fact that the step input is not instant, but rather gradual since it is executed by the rotational stage. In the tilt case, the response is equally steep as the input, while in the heading case the filter seems to catch up. This higher steepness is the merit of the feedback parameter. After the 4.4 s mark, the tilt response maintains a higher rate of change since the Kalman gain is larger due to the larger magnitude of the sensor signals and the fact that both sensors give information about the error in the orientation estimate.

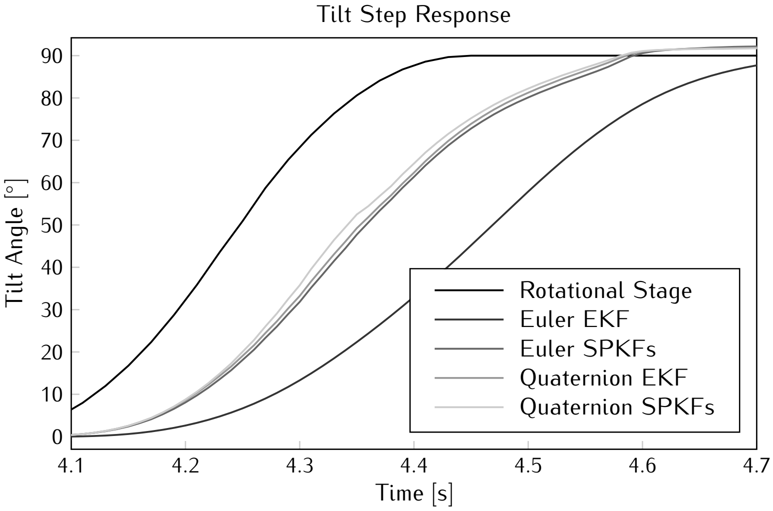


*Figure 5.11: Tilt step response of the Euler and Quaternion filters.*

### 5.2.3 Angular Speed and Delay

The jogging test consists of a fixed number of revolutions performed at a certain constant speed. From this test, the maximal angular speed that can be tracked correctly is determined, as well as the delay between the filter output and the stage orientation during motion. Although the ability to track faster speeds is always more appealing, much of the common human movement remains below a single revolution per second.

Each jogging test consists of five full circle rotations. The angular speed starts at 10 °/s and increases in steps of 10 °/s with each test, until the highest attainable speed is reached, i.e. 720 °/s.

Before analysing the output sequences of the filters, an offset was applied in order to correct for the north orientation in the heading case and misalignment of the sensor node to the rotational stage in the tilt case. This offset was obtained from the stepping test described earlier in this section.

#### 5.2.3.1 Heading

The maximum angular speed that the filters were able to track are given in table 5.3. These values are determined by looking at the final heading output value after five subsequent full circle rotations and checking if this value exceeds 1620°, which corresponds to four and a half revolutions. If this was not reached, it is assumed that at least one revolution has been missed by the filter.

| Euler | | Quaternion | |
|---|---|---|---|
| EKF | SPKFs | EKF | SPKFs |
| 290 | 600 | 640 | 670 |

*Table 5.3: Maximum traceable angular speed [°/s] for heading rotations.*

For the Euler case, the results once again show the slow response time of the EKF compared to the SPKFs. The filters with quaternion representation seem to perform equally well. However, there is a bit of difference between the Euler SPKFs and the quaternion filters that is not expected considering that their step response given in figure 5.10 is very similar. The reason probably lies in the way rotations are described by each representation and how the measurement model is built. The non-linearity of the sines and cosines in the Euler model are very hard to overcome. While the EKF only manages to generate slow linearisation and tracking, the SPKFs fulfill this task better. In the quaternion case, the model non-linearity is not so extreme and an extended approximation even seems to be up to the task. This difference in linearity is now reflected in the maximum speed that the filters can track.

Figure 5.12 displays the delay between the rotational stage output and the filter estimations versus the angular speed of the jog test. Note that the discrete nature of the graphs is due to the fact that the delay is initially calculated as a discrete number of samples. All filters exhibit an increasing delay with increasing angular speed. The delay of the Euler EKF lies between two to three times higher than the other filters and remains above 200 ms. Up until about one revolution per minute, the delay of the other filters stays

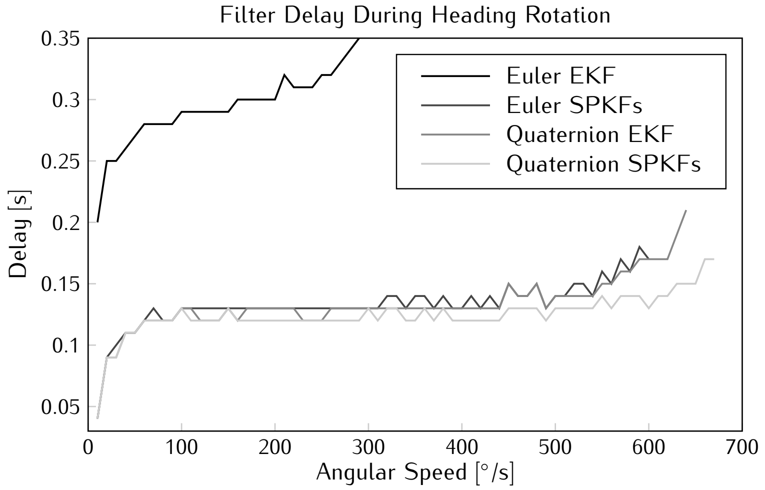


*Figure 5.12: Filter output delay for heading rotations.*

around 125 ms. The quaternion SPKFs also seem to perform slightly better, but then again, these filters have also shown the steepest step response.

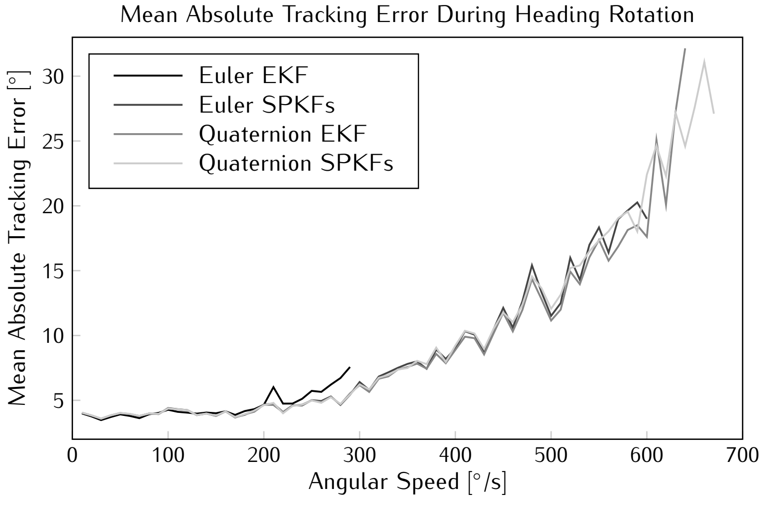


*Figure 5.13: Mean absolute error on the filter output for heading rotations.*

The mean absolute error between the filters and the rotational stage is shown in figure 5.13. This error has been calculated between the delayed version of the filter output and the angular position of the rotational stage.

From the graph, it is easily understood that all filters exhibit the same amount of error when the same angular speed is applied. The mean error also seems to increase in a quadratic way with increasing angular speed. On average, the error remains below 8° when the angular speed is restricted to a single revolution per second.

Finally, figure 5.14 shows the standard deviation on the error shown in figure 5.13. The increase indicates that by increasing the angular speed, the error is less constant, meaning that the response is less linear. Indeed, this effect is validated by the graphs in figure 5.15, where the filter output has been plotted versus the rotational stage output. At 400 °/s, the graph exhibits sinusoidal variations that are not visible at 100 °/s. The cause is mainly found in the calibration of the sensors as was already shown in section 5.2.1.1.

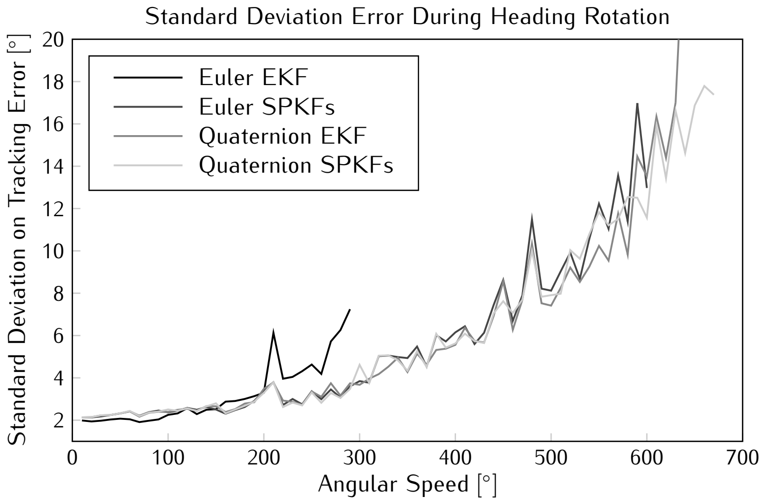


*Figure 5.14: Standard deviation on the absolute error of the filter output for for heading rotations.*

### 5.2.3.2 Tilt

In the tilt case, all of the filters, except the Euler EKF, manage to track rotations with angular speeds up to 720 °/s. The maximum traceable speed can hence not be determined since the rotational stage has reached its limit. This result follows from the fact that for tilt rotations, both the accelerometer and the magnetometer contribute actively to the output of the filter and that the amplitude of the signal on the sensor readings is also larger.

The tracking delay versus the angular speed is given by the graphs in figure 5.16. The Euler EKF again displays a much higher delay which is

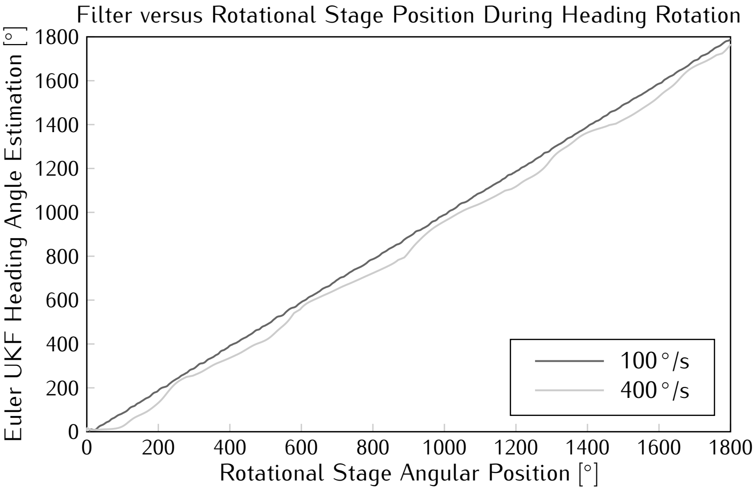


*Figure 5.15: Euler UKF output versus rotational stage angular position for heading rotations.*

mostly above 200 ms over the entire tracking range. From an angular speed of 550 °/s on, the delay on this filter decreases. However, as will be shown by the following figures, the tracking performance of this filter quickly deteriorates at these angular speeds and the resulting delay may no longer be accurate. For the other filters, the delay is slightly lower in the tilt case compared to tracking heading rotations. Over the entire range, the delay on all these filters remains below 130 ms.

The mean absolute error between the rotational stage's angular position and the filter output is shown in figure 5.17 and the associated standard deviation in figure 5.18. The error on the EKF slowly increases until an angular speed of about 550 °/s, here a sudden increase is seen due to the fact that the filter is having a hard time keeping up with the rotational stage's output. As a result, the linearity in the response is greatly reduced as can be seen by the increase in the standard deviation graph. All other filters show different behaviour where the absolute mean error rises until the 500 °/s point, after which the mean error seems to fluctuate highly, unlike the heading case where the error increased. Since the filters can actually track higher angular speeds, the mean absolute error is now a function of sensor noise and calibration errors and therefore varies with each test. In general however, it seems that the tracking error of all filters except the Euler EKF remains below 10° for any rotation speed.

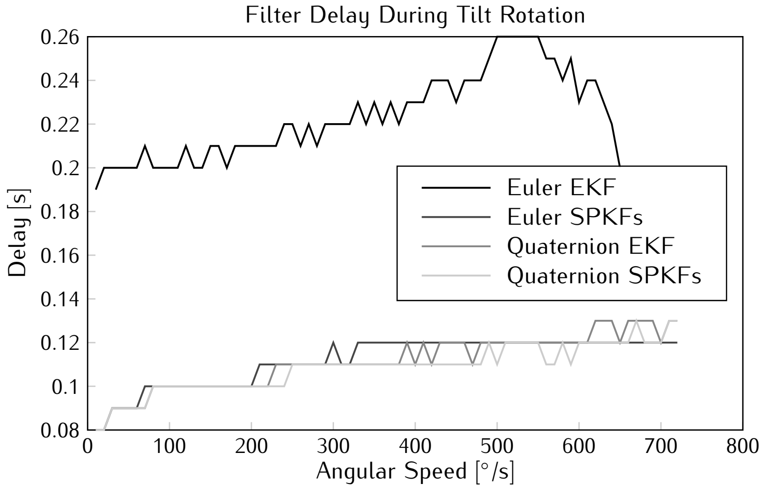


*Figure 5.16: Filter output delay for tilt rotations.*

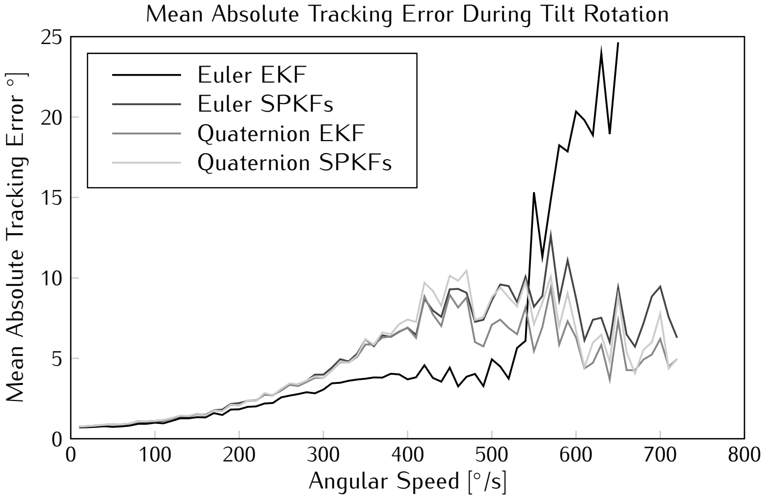


*Figure 5.17: Mean absolute error on the filter output for tilt rotations.*

### 5.2.4 Embedded Filter

In this section, the performance of the embedded version of the tracking filter proposed in section 3.4 is analysed. The linearity will not be determined, since it has been proven that this characteristic entirely depends on the quality of the calibration. The filter characteristics are compared to the performance of

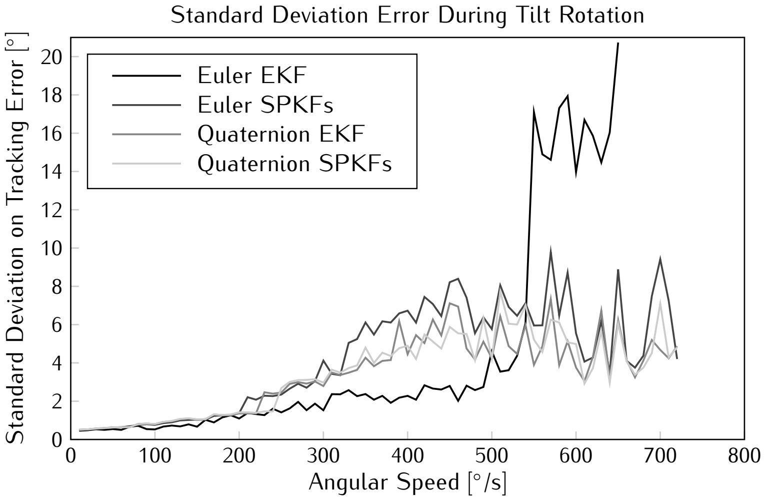


*Figure 5.18: Standard deviation on the absolute error of the filter output for tilt rotations.*

the quaternion type EKF, since the embedded filter is of the same type.

Note that since the embedded filter runs on the microcontroller, this filter is not influenced by package drops inherent to the wireless communication interface. The software filter is only run on the samples that are actually received by the computer. If a package is lost while rotations are taking place, the filter will need to catch up later on.

Also note that the results in this section were obtained by repeating the tests described in the previous sections. Since nodes were programmed to either transmit sensor data or orientation estimations, a single test did not suffice. This can explain small differences between both results.

### 5.2.4.1 Step Response

Table 5.4 lists the mean of the 10 % settling time of the filters as well as the standard devation. The embedded filter seems to have more difficulty in tracking heading steps as the delay is about 100 ms more compared to the software implementation. In the tilt case however, it seems to perform slightly better. The standard deviation on the embedded filter is also significantely lower indicating a more uniform response. The reason for these differences lies in the fact that the measurement covariance matrix consists of larger values in order to avoid overflow of the fixed point representation format. This filter is hence more conservative and is less sensitive than the software version resulting in a more uniform response.

| | Software | | Embedded | |
|---|---|---|---|---|
| | Heading | Tilt | Heading | Tilt |
| Mean [ms] | 478 | 462 | 594 | 448 |
| Standard Deviation [ms] | 24.9 | 37.5 | 13.2 | 12.6 |

*Table 5.4: Embedded and software filter settling time.*

The heading and tilt step response of the embedded and software filter are shown in figures 5.19 and 5.20 respectively. In the heading case, both responses are very similar. The main difference is found in the fact that the embedded filter's response is less steep during the entire sequence. Also, since the embedded filter has only reached 60° when the rotational stage stops moving, the final part where the gain is smaller takes longer to complete.

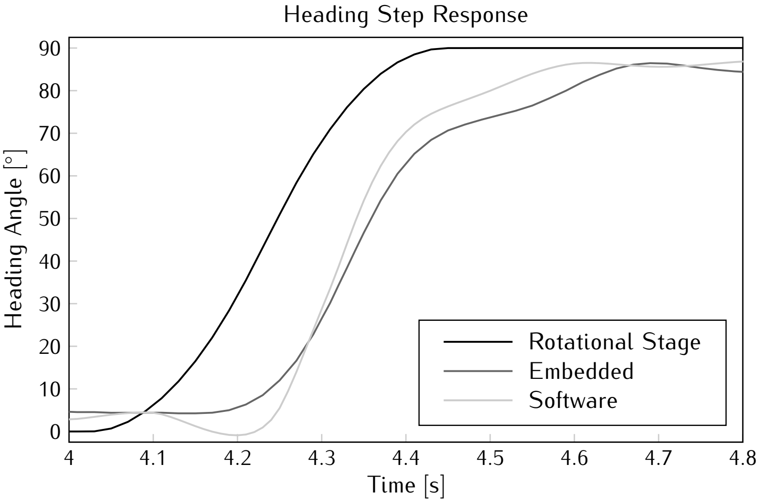


*Figure 5.19: Response of the embedded and software filter to a heading step.*

For the tilt case, both responses display a slightly different form. While the software filter slows down at the end to crawl towards the final value, the embedded filter overshoots before settling. An explanation for this behaviour may again be found in the difference in parameter values. The embedded filter is programmed with higher measurement noise, meaning that this filter relies more on its prediction compared to the software filter. Since the prediction dictates that the quaternion rate of change is approximately maintained, the filter output keeps increasing while the rotational stage is already decelerating. At a certain point, the filter notices that the sensor values correspond better to the predicted measurements and the correction dictates that the rate of change in the response should be tempered. This also explains why the response of the embedded filter seems less steep in the beginning, it takes some time before the required rate of change builds up. Once the it is established, the feedback keeps it alive. However, this also means that it takes some time before this rate is reduced again.

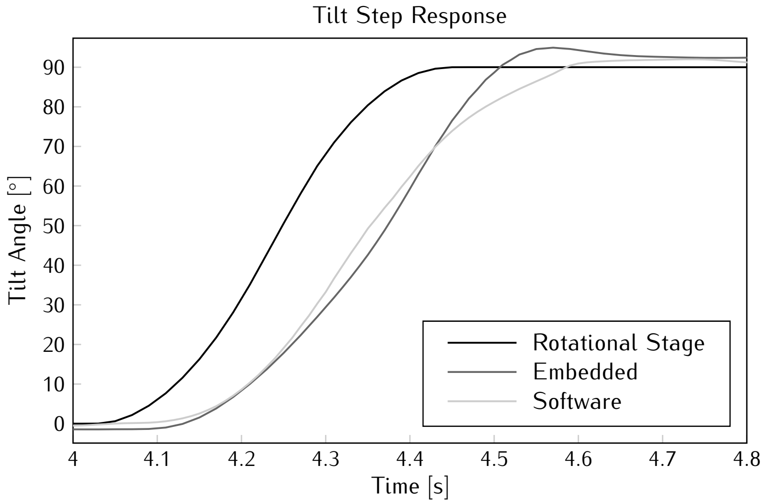


*Figure 5.20: Response of the embedded and software filter to a tilt step.*

### 5.2.4.2 Angular Speed and Delay

The maximum angular speed that the filters are able to track during both heading and tilt rotations are listed in table 5.5. Both filters manage to track tilt revolutions with angular speeds all the way up to 720 °/s. In the heading case, the embedded filter is clearly outperformed by its software counterpart.

| Software | | Embedded | |
|---|---|---|---|
| Heading | Tilt | Heading | Tilt |
| 640 | 720 | 460 | 720 |

*Table 5.5: Maximum traceable angular speed [°/s] of the embedded and software filter.*

Figure 5.21 shows the delays that each of the filters exhibit during constant heading and tilt rotations. In the tilt case, both are very tight together, yet for heading rotations, the underperformance of the embedded filter is confirmed. Clearly, the higher measurement covariance is again to blame.

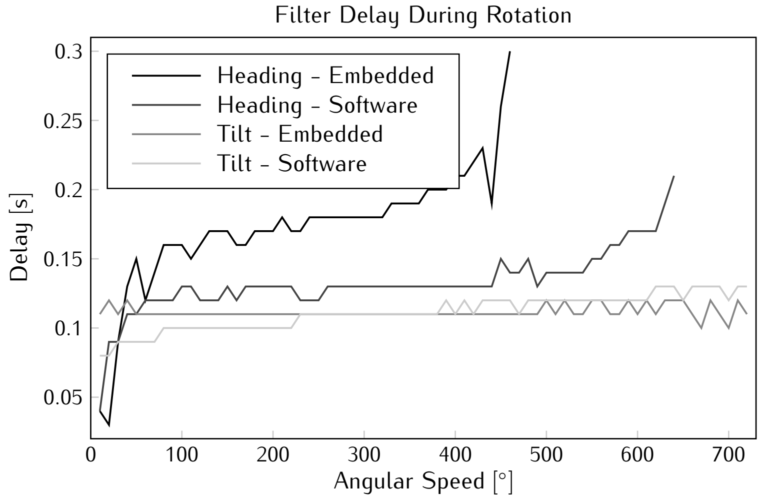


*Figure 5.21: Embedded and software filter output delay during rotation.*

The mean absolute error and the corresponding standard deviation are displayed in figures 5.22 and 5.23 respectively. For the heading case, the form of both graphs is very similar. As was the case for the step response given in figure 5.19. For tilt rotations, the error graphs coincide up to approximately 450 °/s, at which point the error on the software filter output starts varying wildly, while for the embedded filter the error increases further before stabilising around 10°. This trend difference is not seen in the standard deviation, here, both graphs almost coincide.

### 5.2.5 MARG Filter

The performance of the designed filter must still be compared to the MARG filter performance. Given the fact that the gyroscope only supplies information when movements are taking place, the linearity will not be analysed. Obtaining the angular rate as an extra input is only expected to improve the dynamic response of the filter, while the static behaviour is dictated by the accelerometer and magnetometer combination. Given the fact that the MARG filter used is an adaptive, unscented Kalman filter using Euler angle representation, its characteristics are compared to the Euler MFG UKF.

Note that the results obtained in this section use the same data sequences that were used for calculating the characteristics of the MFG software filters. This way, both filters experience the same data loss due to environmental disturbances and the same noise is present on the sensor readings.

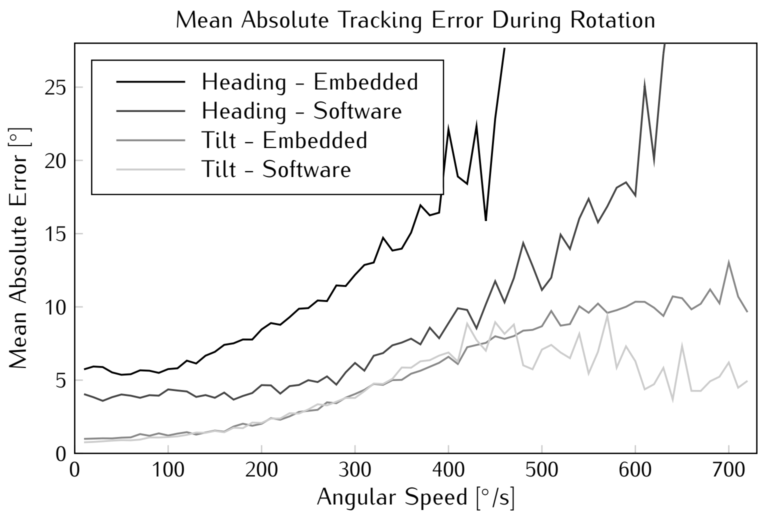


*Figure 5.22: Mean absolute error on the embedded and software filter output during rotation.*

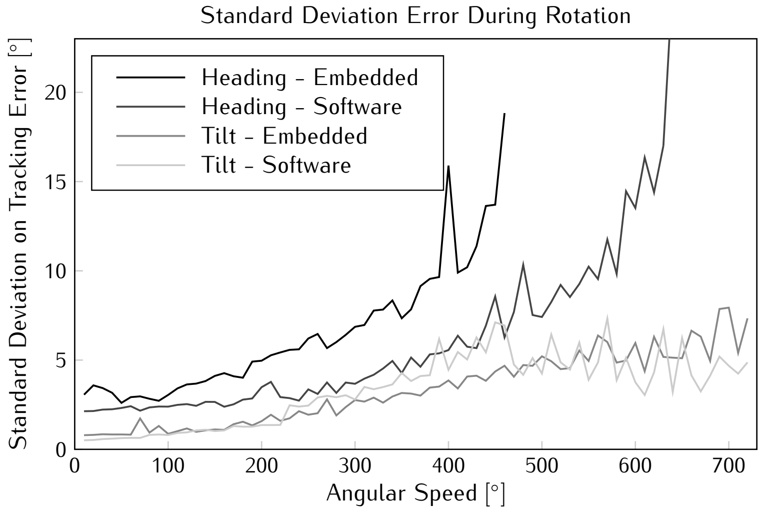


*Figure 5.23: Standard deviation on the absolute error of the embedded and software filter output during rotation.*

### 5.2.5.1 Step Response

The mean 10 % settling times on the heading and tilt step response of the MFG and MARG filter are given in table 5.6, together with the respective standard deviations. The MARG filter is between 30 and 70 ms faster than the gyroless tracker. Also, the standard deviation is a lot smaller. Both of these results are thanks to the better prediction consisting of the integration of gyroscope signals.

| | MFG | | MARG | |
|---|---|---|---|---|
| | Heading | Tilt | Heading | Tilt |
| Mean [ms] | 480 | 469 | 414 | 434 |
| Standard Deviation [ms] | 26.7 | 29.4 | 7.2 | 6.9 |

*Table 5.6: MFG and MARG filter settling time.*

A time graph of the heading and tilt step response of both filters is given in figures 5.24 and 5.25 respectively. It is easily understood that the response of the MARG filter is much smoother and a little faster in both situations. In the heading case, the MARG filter response starts rising as soon as the gyroscope readings indicate a rotation. Since this signal is not available for the MFG filter and all estimations for heading rotations are based on small variations on the magnetometer signal, it is already lagging behind from the start. At the end of the step, an overshoot occurs in the MARG graph. The reason lies in the calibration of the gyroscope. It seems that either the offset is too high or the gain is too large, resulting in a positive integration error. This error is then slowly corrected by the magnetometer readings and evolves towards the MFG filter output final settling value.

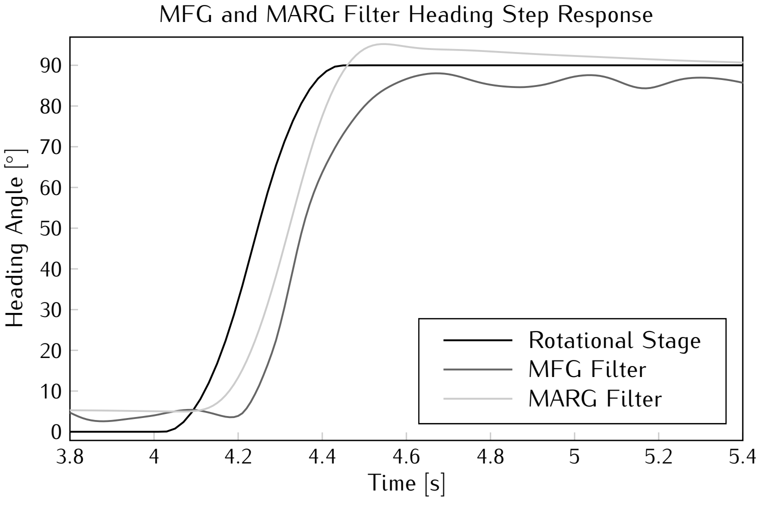


*Figure 5.24: Response of the MFG and MARG filter to a heading step.*

For the tilt step, a slightly different scenario is seen. Both filters start responding approximately at the same time and no overshoot is seen on the MARG response. The explanation to these observations is found in both a better calibration of this sensitivity axis for the gyroscope and the fact that all three sensors offer information about this rotation with signals of a bigger magnitude. Therefore, both the prediction in the MARG filter and the correction in either filter is of a better quality.

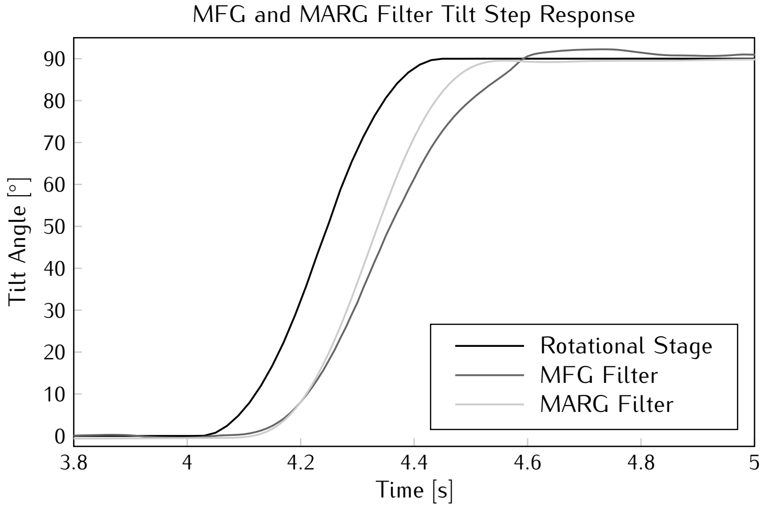


*Figure 5.25: Response of the MFG and MARG filter to a tilt step.*

Note that the overshoot and steeper response in the heading case due to small calibration errors explain the fact that the settling time result stated in table 5.6 is higher for the tilt step. However, this also means that if the tracking error would be considered, it would be found to be higher in the heading case.

#### 5.2.5.2 Angular Speed and Delay

The MARG filter manages to track all revolutions in jog tests with angular speeds up to 720 °/s in both the tilt and heading direction. The delay that is measured with these tests is given by the graphs in figure 5.26. In the heading case, a negative delay is found due to the error in the calibration of the gyroscope. The rotational stage angular position is lagging behind the filter output because the integration results in higher values than expected. However, the form of the graph is similar to the delay seen in the MFG output. For tilt revolutions, the MARG filter seems to consistently exhibit a

30 ms lower delay, except for very low rotational speeds where the correction still has a major influence on the final estimate since the predicted change is fairly small.

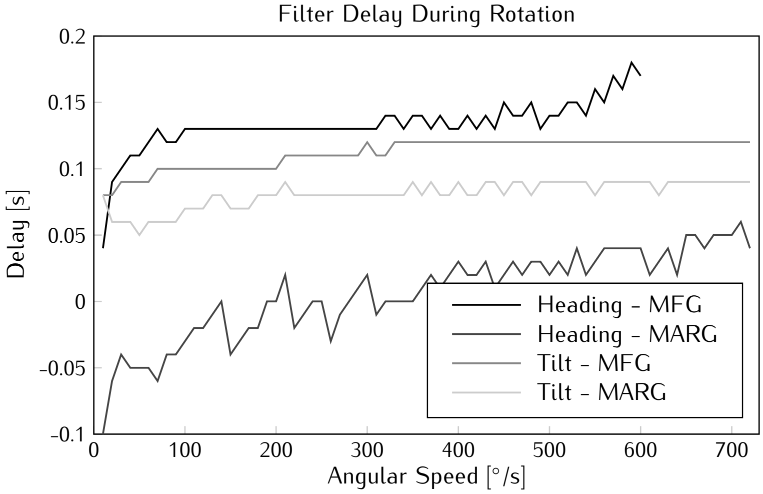


*Figure 5.26: MFG and MARG filter output delay during rotation.*

The mean absolute error on the filter outputs during rotation is plotted versus the angular speed in figure 5.27. In general, the MARG filter displays a lower error than the MFG filter over the entire range and for higher angular speeds, the MARG filter graphs for tilt and heading rotations indicate an equivalent error. Since the error on the output at high speeds is mainly dominated by the gyroscope readings, this is no surprise. Apart from some exceptions, the absolute error on the MARG output remains below 6°.

For the heading case, the MARG response displays a single peak at 210 °/s. This is due to the loss of several packages in a short timeframe. The prediction is therefore less accurate since the discrete integration is no longer correct, and a relatively big error is seen on the output. The magnetometer signal corrects this error, yet since the prediction is trusted more in the MARG type of filter, this takes more time than is the case in the MFG filter where the loss of samples is comparable to a higher angular speed. At low angular speed, the MARG error decreases with increasing speed. Since the MARG output here is still affected by the readings of the magnetometer, the error is similar to the error on the MFG output. At higher speeds, the prediction becomes more important and the graphs diverge.

Below 200 °/s, both filters' error graphs for tilt revolutions coincide. Afterwards, the error on the MFG output increases further as the MARG error

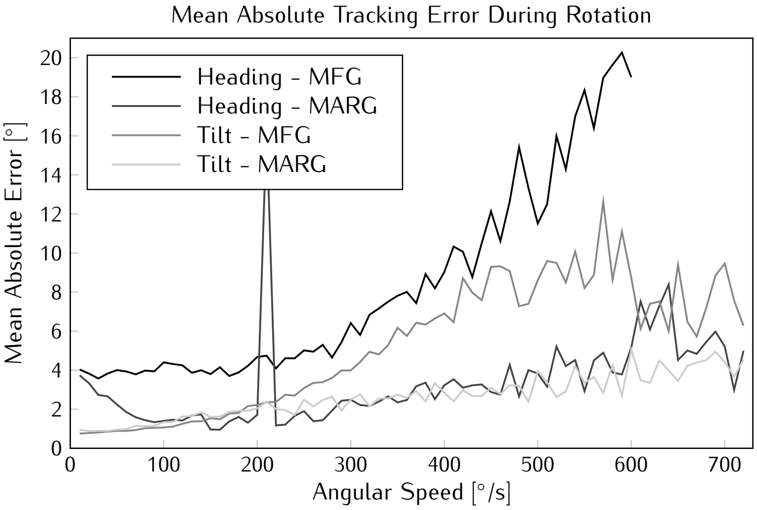


*Figure 5.27: Mean absolute error on the MFG and MARG filter output during rotation.*

stabilises. It seems that in general the error on the MARG filter output increases approximately linear with the angular speed, while for the MFG filter a quadratic relation is seen.

The standard deviation is not given, since these graphs show the same trends and the exact same conclusions may be drawn from them.

## 5.3 Full Body Tracking

Contrast to the previous section, sensor data is gathered from nodes that are subjected to complex motions where the axis of rotation is not constant and not necessarily coincides with one of the sensitivity axes. To this extend, they are placed on the body of a test subject who is to perform certain actions. This person is then simultaneously tracked with an optical system in order to allow a comparison to be made.

The inertial system has been used on three occasions to perform full body tracking. The first time was in December 2008 when a coworker has been tracked during a dance choreography. A second experiment took place in April 2009 and consisted of tracking a person during the execution of various movements. And finally, in January 2011, a subject was asked to perform walking and running exercises on a treadmill while being equipped with sensor nodes.

### 5.3.1 Dance Performance

The dance demonstration was organised in conjunction with the Institute for Psychoacoustics and Electronic Music (IPEM), since they have an optical tracking system available and also have a background in motion analysis of music and dance related performances [5–7]. Nadine Carchon, a coworker at CMST, accepted the challenge to develop a dance choreography that was suited for the initial testing of the system. This choreography should start with slow movements of individual limbs in the beginning to more complex full body and faster motion at the end.

The test subject was equipped with the second generation system, as this was the newest version available at that time. In total, ten nodes were applied to the body, two on each limb and two on the torso. At the same time, several reflective spheres were also placed to allow tracking by the optical system. Furthermore, a camera was also used to record the entire performance from the front.

Although an optical system has been used for tracking, numerical data describing the orientation of the limbs has never been extracted. Therefore, the comparison that is made here consists of screen captures from video footage and both the optical and inertial system visualisations by means of a stickman.

Figure 5.28 displays eight snapshots of the recorded sequence along with the corresponding inertial tracking estimation. The captures span a four second interval, so the time in between two of them is approximately half a second. Note that the stickman visualisation and filter architecture date from an older software version. Hence, no world model including a floor is present and offsets due to misalignment have not been subtracted. Also, no corrections according to the human model are applied since this was not yet available at the time of capture.

The general posture of the stickman seems to correspond well to the pose of the test subject in the video footage. The heading might in some cases not be exact. In the first picture e.g. the left upper leg seems to point rather forward, where in the stickman reconstruction it points more towards the left. This exact same error can also be seen in the third snapshot. Some slight mistakes in tilt can be observed in the final capture, where the torso and right arm should normally overlap according to the video footage.

Snapshots from a different part of the dance choreography are given in figure 5.29. The timespan covered here lasts approximately twelve seconds. The groundwork in the first four figures is nicely tracked aside from some offset errors. The second part where the dancer gets up can be recognised when looking at the stickman, yet the visualisation is less smooth and transients are visible since this happens quite fast.

From this dance experiment, some of the limitations of the optical system

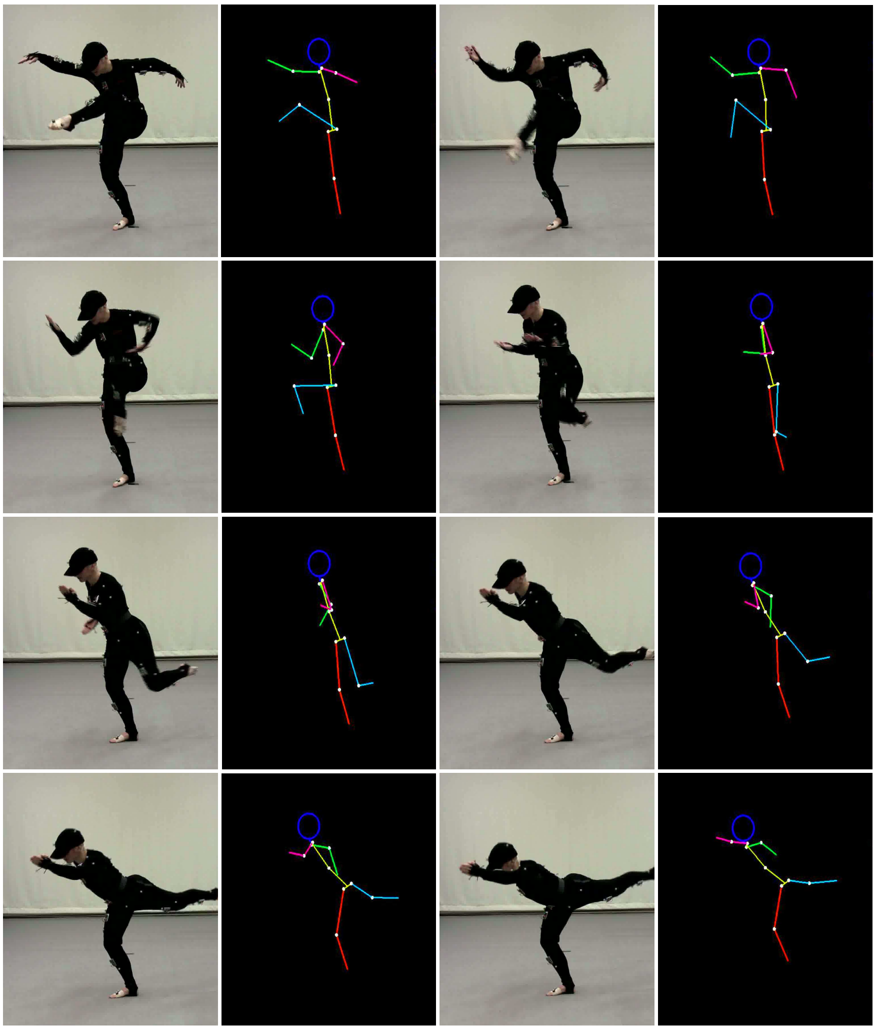


*Figure 5.28: Eight snapshots spanning a four second period in the dance choreography where the left leg swings from front to back. Video capture and corresponding stickman visualisation from inertial estimations are given.*

were very clear. First, the choreography had to be limited in space, since the tracking area is approximately restricted to a circle with a radius of one meter. For the inertial system, only the range of the wireless interface restricts the working volume. Second, it took some time to set up the markers on the test subject's body and to calibrate the camera system, while the inertial system was ready for capturing after merely five to ten minutes. Finally, occlusion

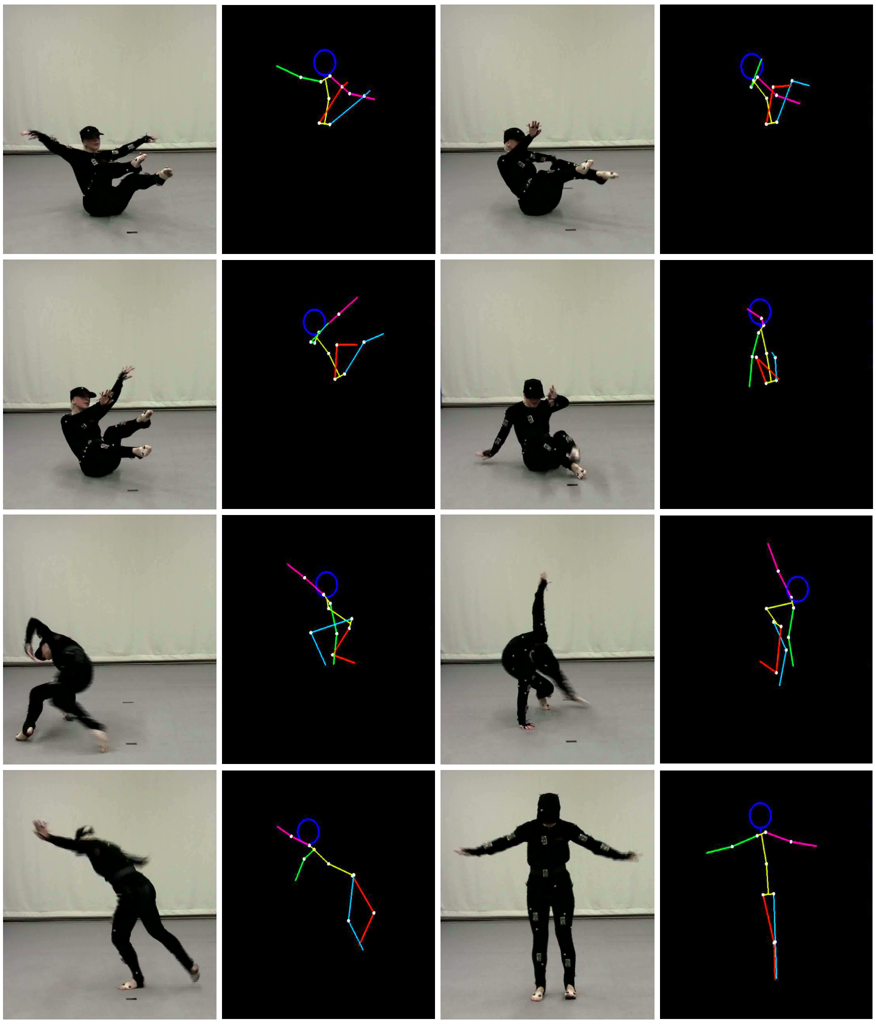


*Figure 5.29: Eight snapshots spanning a twelve second period in the dance choreography including groundwork and getting back up in spiral motion. Video capture and corresponding stickman visualisation from inertial estimations are given.*

of some of the markers leads to wrongful reconstruction of the posture. This problem is shown by the stickman visualisations given in figure 5.30. The eight snapshots in this figure display the posture estimations of the optical system to the video captures given in figure 5.29. Each snapshot of the stickman visualisation gives a three dimensional view in the left upper corner, a front view in the upper right, a side view in the lower left corner and a top view in the lower right. In some of the captures, one of the limbs of the stickman is coloured red, indicating that occlusion is taking place for the markers on this particular bodypart. The right lower leg suffers from this problem in the first five snapshots and temporarily assumes incorrect orientations.

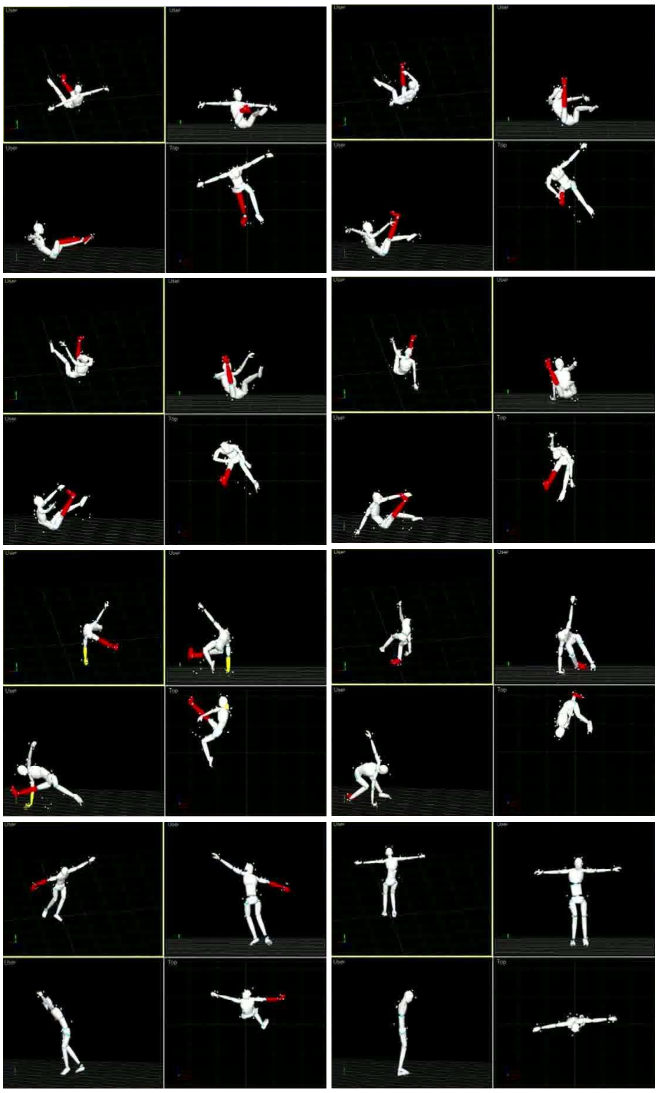


*Figure 5.30: Optical system stickman visualisation*

### 5.3.2 Various Movements

The other two full body tracking experiments have been executed in cooperation with the department of movement and sport sciences who have been active for many years in human gait analysis [8, 9] based on measurements performed with an optical tracking system they also own. In this experiment, a person was tracked using both the inertial and optical systems while performing a set of different movements. Table 5.7 lists the different tests that were executed.

| Test | Description |
|---|---|
| 1 | Arm and leg lifting |
| 2 | Walking |
| 3 | Running |
| 4 | Turning around |
| 5 | Arm and leg twisting |
| 6 | Lower body movements |
| 7 | Upper body movements |
| 8 | Fast knee flexion and extension |

*Table 5.7: List of the movements in the second full body tracking experiment consisting of various movements.*

Once again, a total of ten nodes was used for inertial tracking in each test. Two are placed on each limb and two on the torso. Contrast to the dance demonstration however, the data received from the optical system during the experiment was processed to obtain the orientation of the same bodyparts that were tracked by the inertial system. Both outputs can be compared in order to analyse the accuracy of the system in a more quantitative way. Important to mention is that the delay of tracking between both systems cannot be determined, since the data was received on two different computers and no synchronisation was present aside from a human approach where both programs were started after a countdown.

Table 5.8 lists the mean absolute error between the Euler unscented filter estimations and the optical system output in degrees. These errors have been determined after applying offset correction on the filter angles and attempting to remove the time delay between both sequences. The offset results from alignment errors between the reference frame of the sensor nodes and that of the optical system, while the delay is due to the manual synchronisation and the delay inherent to the filter.

In the table, several cells have been given a grey background. The error mentioned in these cells should not be considered too seriously since visual inspection of the output indicated that the optical system output is rather untrustworthy due to data loss by blocked LoS from the markers to one or more cameras resulting in clipping or occlusion of the bodypart. Furthermore, three column headers have also been coloured. The reason here is found in the calibration. The nodes associated to the bodyparts listed in these columns exhibited problems in the calibration of the magnetometer. Hence the mean absolute error on the yaw angle result for these bodyparts are rather high compared to the other nodes.

Important to note is that the test was conducted in a room with metal walls. Therefore, the earth magnetic field was shielded and weakened inside the room. The result is that the calibration of the magnetometer in general is no longer entirely valid and the performance of the filter is not optimal. Also, the results listed in table 5.8 were obtained using the same parameters as listed in the beginning of section 5.2, except for the measurement noise covariance. These values were chosen higher as the filter seemed way too sensitive to changes and the effect of motion disturbance had been underestimated. Finally, $r_{acc}$ was set to $2 \times 10^{-3}$ and $r_{mag}$ to $1 \times 10^{-3}$, or 100 times larger than was determined in section 2.5.2.1.

The comparison of the data sets is not straightforward and requires some tricky preprocessing. Since Euler angles are used, both reference frames should be lined up first. The optical frame will be used in the sequel where the Z-axis points upwards, the Y-axis points forward out of the bodypart and the X-axis is chosen to form a right-handed base, namely pointing to the right side of the subject. The zero or neutral position where each angle equals zero corresponds to the standard pose as defined in section 4.3.2 and visualised in figure 4.5. The subject will however not be facing North, yet to the front of the room, which explains the offset. Since multiple Euler angle sets can represent the same orientation as has been outlined in section 2.2.1, the angles were first converted to a unique set where $\phi \in [-180°, 180°[$, $\theta \in [-90°, 90°[$ and $\psi \in [0°, 360°[$. The optical data was obtained in a time driven way, so the sample rate is linear. The inertial data however, is captured when it is received by the computer. As wireless communication is involved, samples were lost in the process. These samples were reconstructed by linear interpolation where the package counter indicates that data is missing. This step is required in order to keep both sequences aligned in time. Finally the mean absolute error is calculated modulo 360° for roll and yaw and modulo 180° for pitch after applying the delay for synchronisation.

Since the mean absolute error only supplies general information about the tracking precision, time graphs of orientation originating from both systems will be compared in the following. Figure 5.31 shows graphs of the orientation output of the filter and the optical system for the right upper leg during the first test. Leg orientation is given since the arms were out of the camera scope when lifted. The inertial system seems to track the changes in orientation pretty well, as could already be determined from the values in table 5.8. However, there clearly is a lot more noise present on the inertial signals, distinctly

*Table 5.8: Mean absolute error between filter estimations and optical system output angles in degrees.*

| Test no. | Euler angle | Left upper leg | Left lower leg | Right upper arm | Right lower arm | Left upper arm | Left lower arm | Right upper leg | Right lower leg | Upper torso | Lower torso |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Roll | 0.89 | 2.82 | 6.83 | 2.80 | 20.15 | 3.01 | 1.14 | 1.94 | 1.44 | 2.29 |
| | Pitch | 1.64 | 1.17 | 6.18 | 2.37 | 19.09 | 2.40 | 0.80 | 0.74 | 1.52 | 1.53 |
| | Yaw | 2.13 | 1.59 | 11.74 | 3.30 | 30.11 | 3.71 | 2.30 | 3.68 | 2.39 | 2.43 |
| 2 | Roll | 6.74 | 13.63 | 4.03 | 11.79 | 3.87 | 8.21 | 11.11 | 11.27 | 2.75 | 3.53 |
| | Pitch | 4.44 | 6.60 | 3.92 | 4.73 | 3.61 | 4.42 | 5.77 | 5.50 | 3.53 | 3.24 |
| | Yaw | 12.33 | 15.24 | 10.42 | 31.35 | 13.59 | 14.81 | 39.76 | 14.54 | 12.54 | 24.55 |
| 3 | Roll | 16.60 | 19.11 | 15.00 | 30.93 | 10.30 | 21.59 | 18.46 | 26.88 | 5.29 | 3.86 |
| | Pitch | 8.47 | 7.74 | 9.68 | 17.72 | 7.17 | 18.36 | 7.91 | 15.87 | 4.34 | 4.08 |
| | Yaw | 29.57 | 21.28 | 27.26 | 84.36 | 19.43 | 48.29 | 51.93 | 35.13 | 16.44 | 24.03 |
| 4 | Roll | 7.53 | 7.84 | 5.84 | 9.26 | 13.96 | 7.90 | 8.46 | 7.26 | 6.38 | 7.83 |
| | Pitch | 6.96 | 7.23 | 7.76 | 6.39 | 8.47 | 8.53 | 7.36 | 6.09 | 4.98 | 5.40 |
| | Yaw | 19.77 | 21.74 | 19.35 | 27.46 | 22.27 | 26.79 | 32.39 | 23.06 | 19.81 | 26.39 |
| 5 | Roll | 3.29 | 13.55 | 5.91 | 11.60 | 8.40 | 9.11 | 4.87 | 6.44 | 1.79 | 4.62 |
| | Pitch | 7.02 | 5.50 | 4.11 | 9.37 | 7.02 | 6.86 | 5.75 | 4.52 | 2.96 | 3.07 |
| | Yaw | 9.30 | 34.00 | 12.62 | 18.86 | 10.97 | 16.45 | 11.91 | 12.79 | 7.76 | 6.80 |
| 6 | Roll | 2.72 | 9.53 | 1.77 | 2.98 | 3.85 | 2.03 | 4.87 | 9.11 | 1.36 | 3.97 |
| | Pitch | 5.04 | 7.14 | 1.53 | 2.26 | 2.42 | 2.34 | 4.67 | 4.58 | 1.79 | 2.81 |
| | Yaw | 9.47 | 22.81 | 4.45 | 5.86 | 3.73 | 5.15 | 11.45 | 18.31 | 4.36 | 5.84 |
| 7 | Roll | 1.05 | 2.27 | 10.95 | 9.38 | 8.61 | 8.62 | 1.32 | 1.00 | 2.69 | 3.58 |
| | Pitch | 2.05 | 1.71 | 7.85 | 8.70 | 8.24 | 7.69 | 1.36 | 0.93 | 3.29 | 4.04 |
| | Yaw | 3.66 | 5.92 | 25.14 | 26.39 | 15.39 | 17.45 | 3.51 | 2.03 | 6.23 | 4.96 |
| 8 | Roll | 11.08 | 23.97 | 6.22 | 8.10 | 8.79 | 6.61 | 4.64 | 4.64 | 3.92 | 8.55 |
| | Pitch | 10.88 | 9.60 | 4.12 | 8.73 | 9.42 | 9.38 | 4.42 | 5.64 | 3.77 | 5.37 |
| | Yaw | 22.67 | 55.98 | 18.66 | 26.69 | 22.51 | 19.91 | 28.80 | 17.36 | 19.32 | 26.58 |

on the pitch and yaw angles. This is mainly due to the weak magnetic field and the fact that errors on the yaw angle are corrected through changing the pitch angle in an attempt to match up the magnetic readings. An additional post-filter can reduce the noise, but an extra amount of delay must in that case be tolerated.

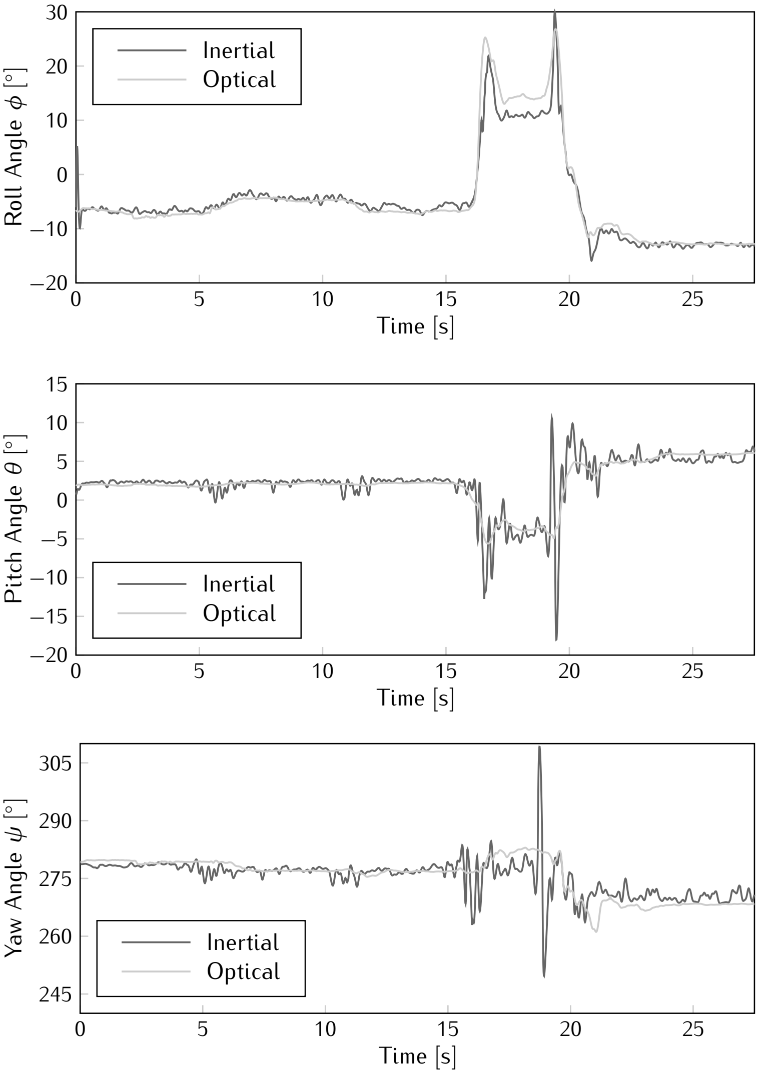


*Figure 5.31: Right upper leg orientation angles during test 1.*

The orientation of the left upper leg during the second test is displayed in figure 5.32. The subject was asked to walk from left to right and back again several times. Clearly, the limitations of the camera system surface in this test since no orientation data is available at the turning points due to the fact that the person was out of the tracking space where not enough cameras could see the markers correctly. The turning points are obvious in the yaw graphs however, as the output switches between two steady states at around 270° and 90°, which are separated by 180°. In the roll graph, the inertial system is close to the optical reference output, in the other graphs, errors seem to emerge. The cause of these errors is found in the disturbance of the accelerometer signal each time the subject's foot hits the ground and a shock is measured. Thanks to the dynamic adaptation of the accelerometer measurement noise, this error is somewhat suppressed. When $\zeta$ is set to zero, the mean absolute error on the left upper leg output is found to be 14.11°, 5.98° and 16.36° on roll, pitch and yaw respectively. A similar trend is seen on the other bodyparts, yet more important is the fact that the instant error grows even larger.

Figure 5.33 shows time graphs of the orientation of the lower torso during the running test. The filter is clearly having a hard time keeping up the estimates and relatively big errors are seen. The yaw angle is able to keep up the turning points however and the roll angle is still pretty much tracking right. However, as can be seen in table 5.8, the results for the limbs is much worse. Also note that the calibration error on the magnetometer of the sensor node attached to the lower torso, which was already indicated in the table, is pretty clear from the yaw graph.

The orientation estimates for the right upper arm during the fourth test have been plotted in figure 5.34. Several of the full circle rotations are visible on the yaw graph, however, the filter is not able to keep up with all of them. Cycle slips are visible around timestamps 6, 12 and 20 s. The filter keeps up at first, but finally turns its estimates around the other direction to meet back with the optical output values. With each cycle slip, errors on the other two angles can also be distinguished, roll exhibits negative peaks and pitch positive ones. These temporary transients are an attempt of the filter to line up the sensor measurements to the expected values with a wrongly estimated yaw.

A time graph of the lower left leg orientation during test number 6 is shown in figure 5.35. Although the filter is able to keep up with the knee flexion and leg lifting actions as indicated by the respective downward and upward peaks in the roll graph, these movements are not tracked entirely as the amplitude is smaller than the optical output indicates. As was the case for the the fourth test, the fact that the filter is unable to reach the final value introduces errors on the other two angle estimates.

Figure 5.36 depicts the orientation of the left upper arm during upper body movements including arm lifting, elbow bending and upper torso movements.

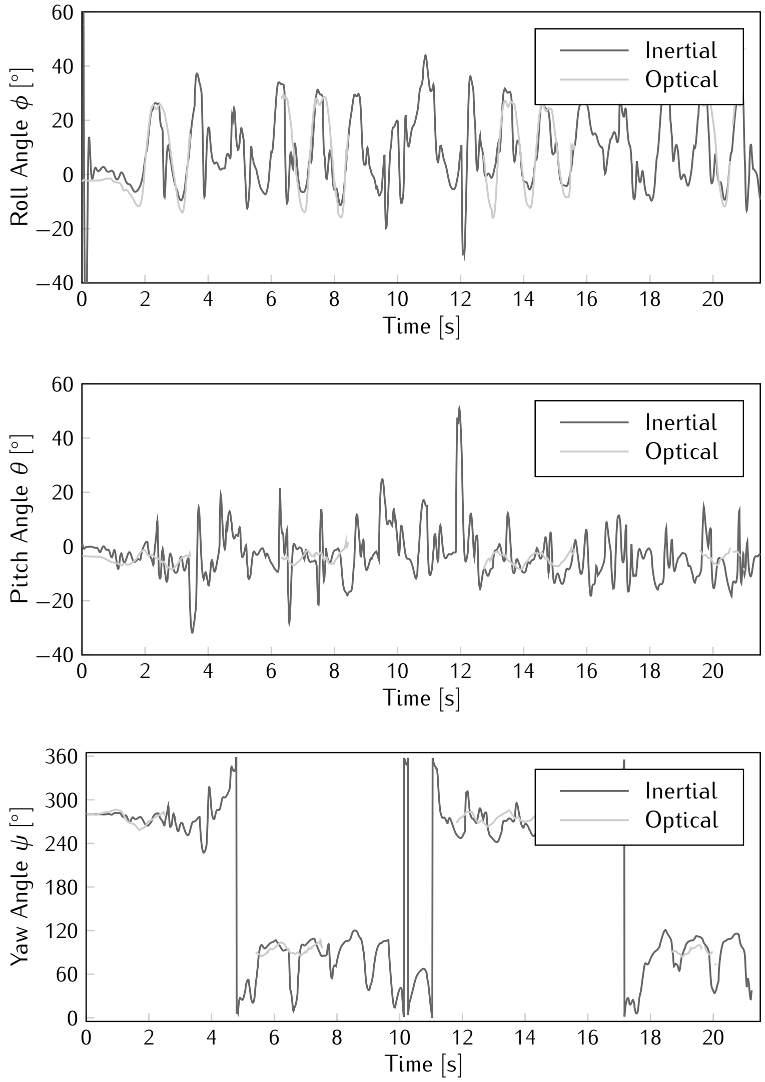


*Figure 5.32: Left upper leg orientation angles during test 2.*

The graphs of all three angles seem to correspond pretty well, although the mean absolute error listed in table 5.8 indicates errors around 8° on roll and pitch and 15° on yaw. Note that the jumps of 180° at the end of the sequence in both roll and yaw are due to the fact that the angles are reduced to a unique set. Since pitch is limited to $[-90°, 90°[$, values outside this range trigger these jumps.

The left lower leg roll angle during fast knee flexion has been plotted in

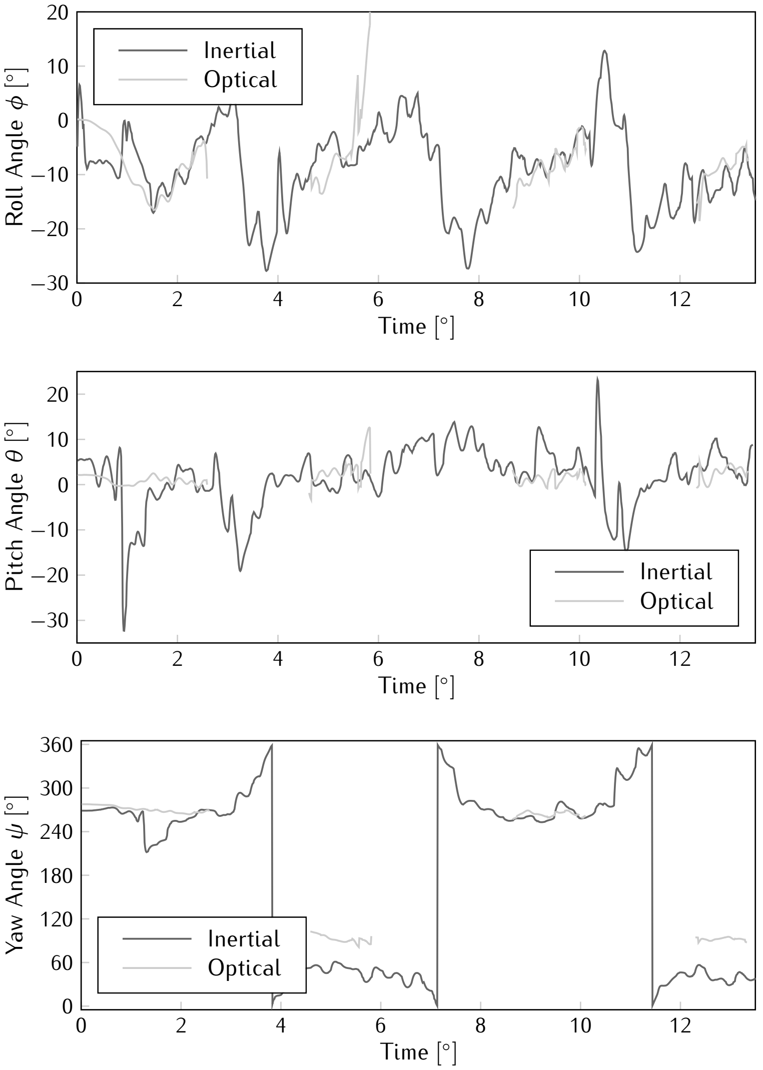


*Figure 5.33: Lower torso orientation angles during test 3.*

figure 5.37. While the filter was able to keep up with the knee flexion in test 6, this is not the case for test 8 where two flexions per second can be seen in the graphs. The roll angle seems to change somewhat, but the entire amplitude is never reached.

Since the filter is not up to the task of tracking high speed motions, as was seen by the graphs for running and also fast knee flexion, it might be more appropriate to tune the filter for better tracking of slower movements. Reducing

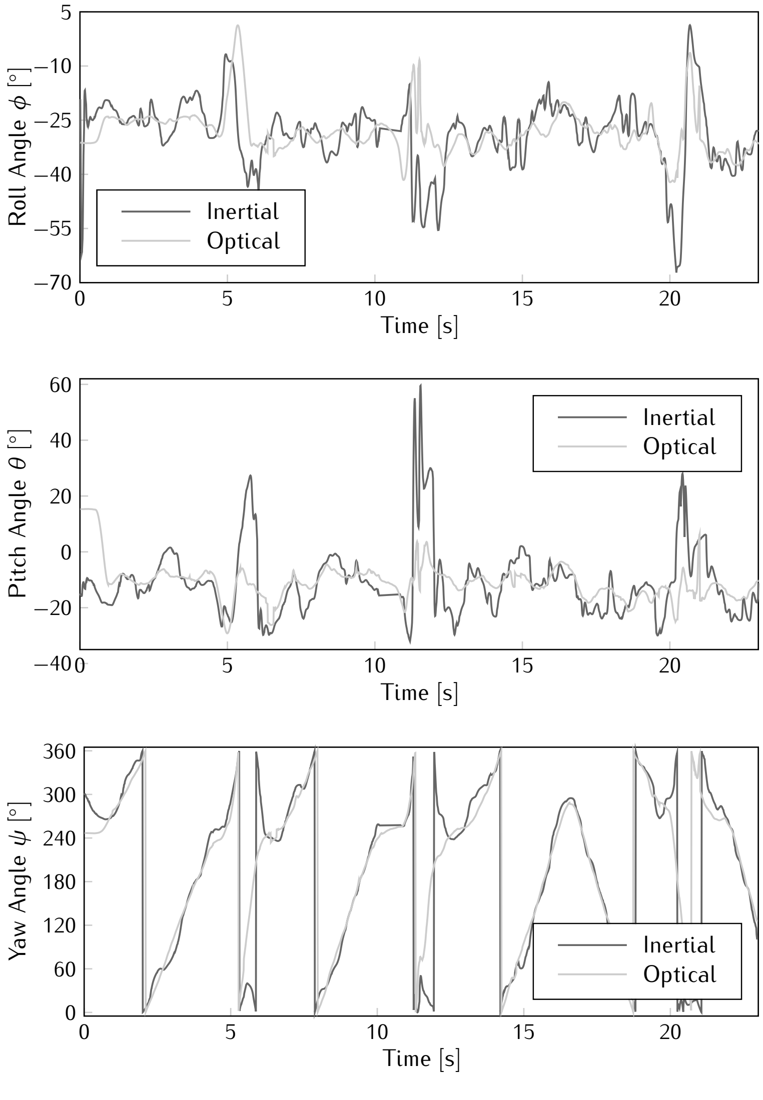


*Figure 5.34: Right upper arm orientation angles during test 4.*

the feedback parameter $\tau$ results in better tracking for several tests, but the error on fast movements on the other hand increases. With the decrease of the feedback, the measurement noise may also be reduced, meaning that motion disturbance might again have a larger influence.

Another way to improve the performance of the filter can be obtained by adjusting the system model for the type of movement that is executed. The impact of the foot hitting the ground during walking or running for example

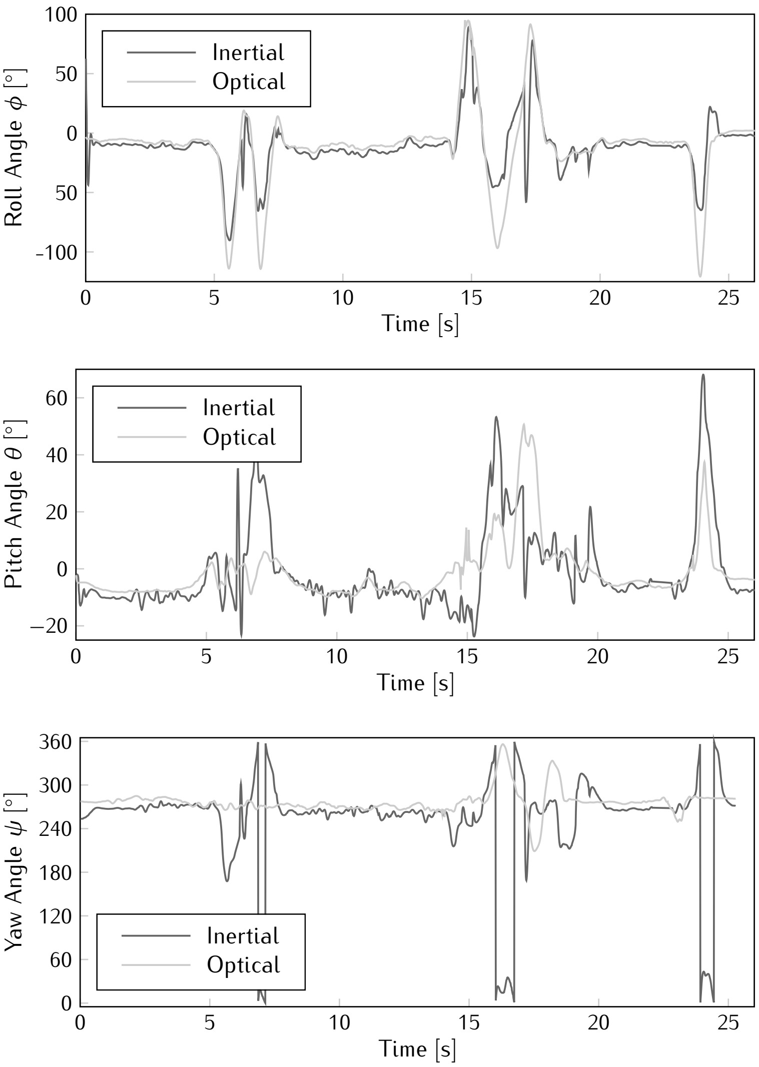


*Figure 5.35: Left lower leg orientation angles during test 6.*

is a very repetitive and maybe even slightly predictable error source. In this case however, classification of movements forms a new challenge.

As a final remark it should be pointed out that the error on the orientation output of nodes assigned to the torso are generally lower than the errors found on the limb orientation. The reason is logically found in the fact that these nodes are not subjected to large accelerations as the limb nodes are. This observation justifies the choice made in chapter 4 to designate one of these nodes as the root node for drawing the stickman. Since the orientation of the

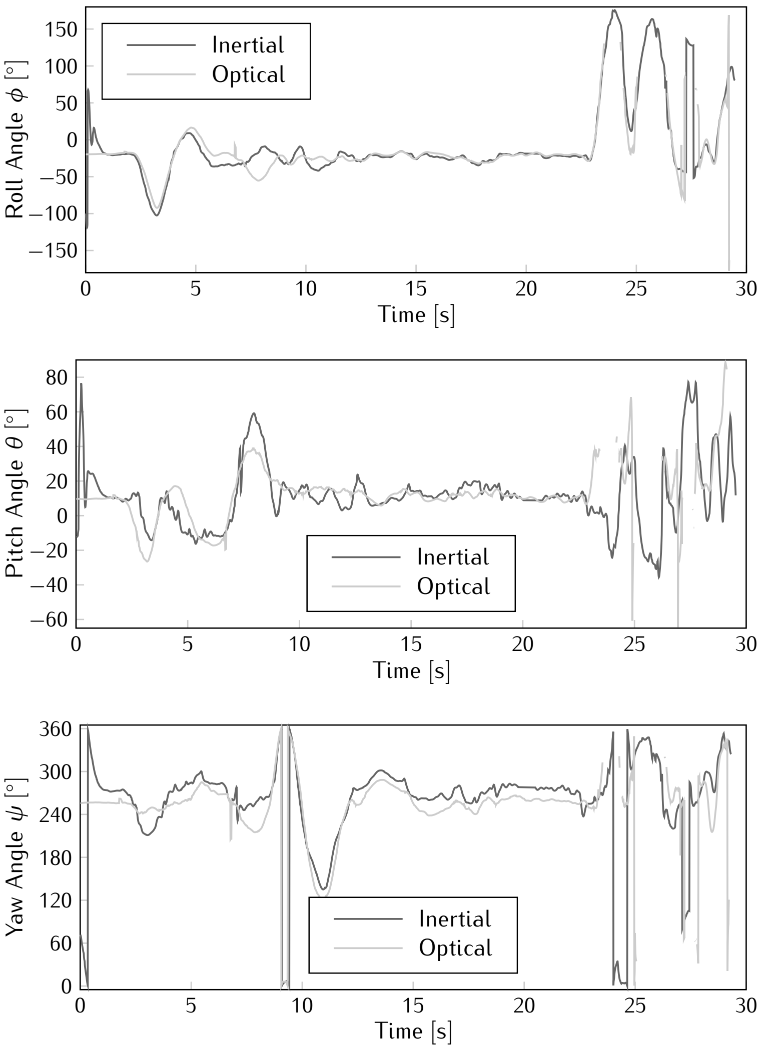


*Figure 5.36: Left upper arm orientation angles during test 7.*

lower torso node is simply not corrected in the human model, the error is at least lower than when a lower leg node would have been chosen. Note that this information could also be used to determine better models for each of the limbs separately in stead of using the same system model for all of them.

### 5.3.3 Treadmill Exercises

Since the optical system was unable to track the entire motion during the walking and running experiments due to the fact that the test subject was

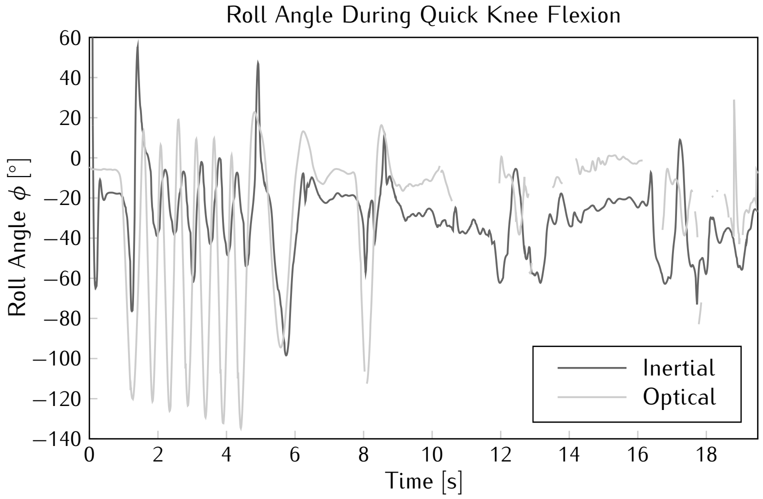


*Figure 5.37: Roll angle on the lower left leg during test 8.*

out of the tracking space, additional experiments have been conducted where walking and running exercises are performed on a treadmill. This time, third generation MARG nodes were used for data gathering such that angular rate measurements are also available. This allows to apply both the MFG and MARG filters to the sequences and compare the output to each other, and to the optical system. Ten nodes were applied to the test subject's body, as there were only this many available. However, both torso nodes did not hold through the entire test. Therefore their output will not be considered here.

Although the experiment was executed on a different floor, it was still conducted in the same building as the previous test, meaning that the metal walls were also present. In fact, a calibration of the magnetic sensors has been executed at this time, within the test room, which indicated that the earth magnetic field magnitude is reduced to two thirds compared to our own test lab. Hence, small errors on the signals automatically deteriorate the filter performance, certainly for the heading output.

An attempt was made to synchronise the data captures by using a trigger and starting both programs simultaneously, but in the end, this did not seem to work. Therefore it was necessary to resort to human synchronisation after a countdown using a clock present in the room. This also means that the delay between the systems may again not be determined from the resulting output sequences.

The experiment that is considered here is a ramped cycle that is used to determine the transition point between walking and running. The treadmill starts moving slow and gradually increases the speed until the subject is required to run, this state is maintained for a while and afterwards the speed is again decreased. The capture programs were started together with the treadmill. This was however not beneficial for the inertial tracking system. Since the initial estimates for the Kalman filter generally do not correspond to the actual state, a transient period is expected when captures are started. As the test subject was already in motion, this transient takes place during dynamic changes and results in suboptimal performance. In order to address this issue, the initial state of the filter is set to the offset values determined from the calibration procedure of the stickman. As explained in section 4.3.2, offset are needed to remove the influence of misalignment between the sensor node and the bone. In the reference stance, these offsets can be determined and it is expected that these values are closer to the actual orientation when the person is already in motion.

Figure 5.38 shows the inertial filter estimates versus the optical system output for the lower left leg. This bodypart has been chosen as it is moving extensively during walking and running exercises and is expected to be very hard to track using inertial sensors, which is clearly reflected by the graphs. The walk-to-run transition is approximately found at the 12 s time mark, while the inverse transition takes place at about the 25 s mark.

From the graphs, it is easily understood that the yaw and pitch angles are poorly estimated compared to the golden standard. Generally, the amplitude of both signals is a lot larger. The roll angle estimates are better and seem to generate a signal with approximately the same amplitude and frequency. During the running part, the inertial system however looses track and exhibits a slightly smaller amplitude, though the frequency remains correct.

Although the inertial system seems to have a very hard time in supplying qualitative orientation estimates, it is more important to also analyse the differences between the MFG and the MARG tracker. In general, the MARG curves seem smoother, which is also expected given the availability of angular rate information. The biggest difference however, is found in the running part of the experiment. Here, the MFG filter starts experiencing drift on its yaw estimate, resulting in bigger errors on the pitch estimate as well. The MARG filter on the other hand does not seem to be bothered with the transition except for a slight increase in the instant pitch error.

The poor performance of the inertial estimators can mainly be attributed to motion disturbance on the accelerometer signal. Figure 5.39 displays a time graph of the magnitude of the sensor readings. Clearly, the mean magnitude of the acceleration lies above the expected unity value. This indicates that influence of motion disturbance on the filter output has been underestimated. During the running operation, the magnitudes seems to be rather close to 1.75 g. Note that the average magnitude of the magnetic field seems to have

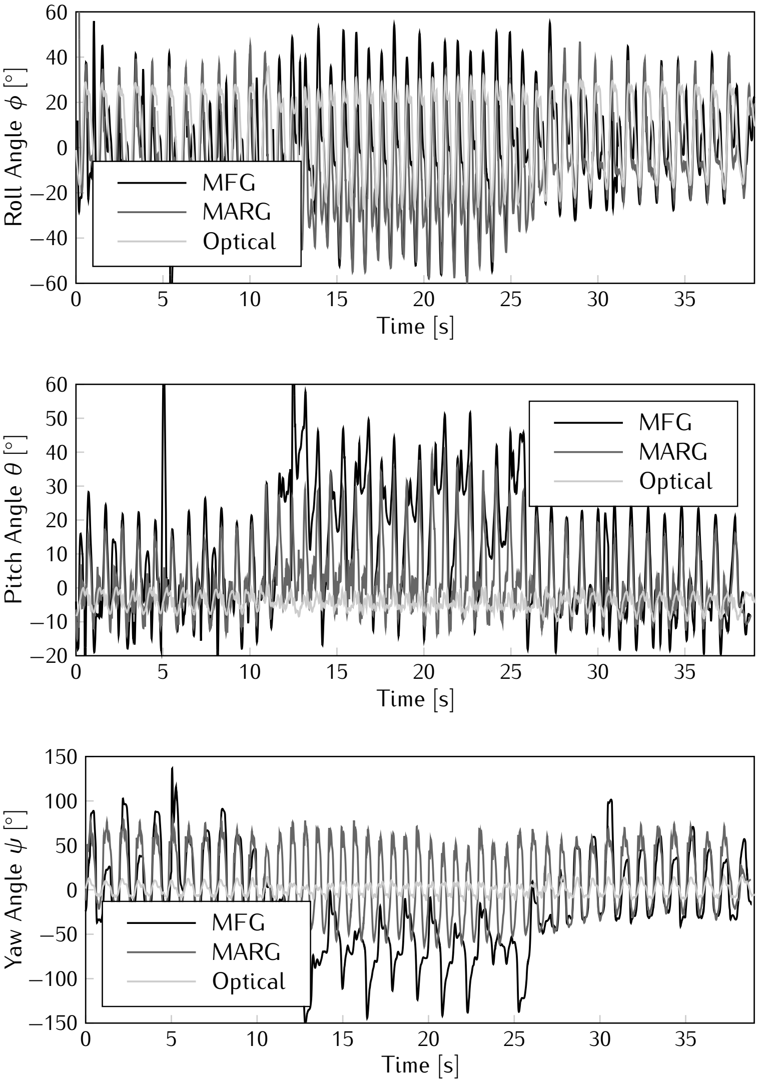


*Figure 5.38: Left lower leg orientation angles.*

been overestimated as well, although a calibration was executed in the very same room. This indicates that the uniformity of the magnetic field within the room is clearly not very good. Also, the metal contained in the treadmill structure might have an influence here.

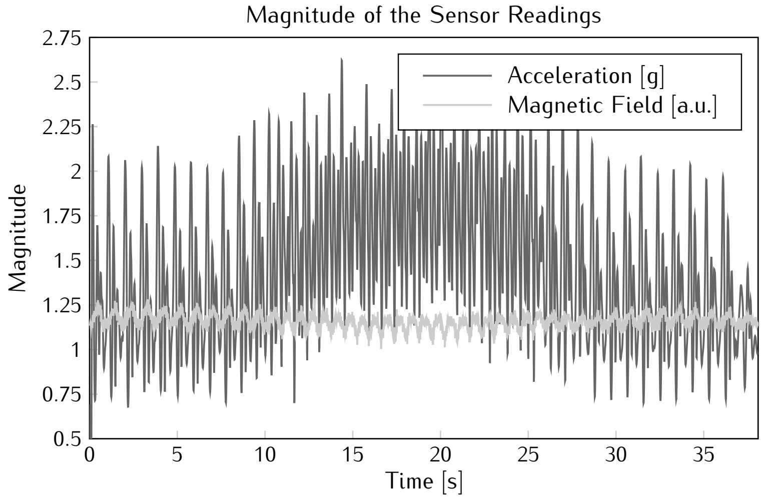


*Figure 5.39: Magnitude of the sensor readings of the sensor node associated to the lower left leg.*

## 5.4 Conclusion

A quantitative analysis of the performance of the tracking algorithm is performed by placing the nodes on a rotational stage and comparing the estimations with the angular position. Three types of tests are used: a step test to determine the linearity, a block test for the step response and a jogging test to obtain the maximal traceable angular speed and the delay of the filter. A distinction was also made between heading and tilt estimation, since there is clearly a difference between both.

The linearity of the filter was greatly affected by the quality of the calibration, especially for the heading case since the magnetic field of the earth is very weak. Poor calibration quickly leads to systematic linearity errors with a sinusoidal shape. The step response of the filters again point out that the Euler EKF underperforms quite a bit. The other filters all display a 10 % settling time of approximately 480 ms for a heading step and 460 ms for tilt. Keeping in mind the 400 ms travel time of the rotational stage, this leads to step delays below 100 ms. The maximum angular speed for tilt rotations that can be traced by all filters except for the Euler EKF is at least higher than the maximum speed of the rotational stage, i.e. 720 °/s. In the heading case, the Euler SPKFs manage up until 600 °/s, while the quaternion filters reach as high as 650 °/s. The delay during motion is around 125 ms for each of the well performing filters and for angular speeds that are tracked with a mean absolute error below 15°.

The same tests have also been used to determine the performance of the embedded quaternion EKF implementation and the MARG filter. In the step response, the embedded filter suffers from a higher settling time for heading rotations, but for tilt the filter is in fact even slightly faster due to minor overshoot. The maximum traceable speed for heading also turns out to be lower with a delay that amounts to 180 ms. The MARG filter displays a smoother and faster step response than the MFG filter due to the presence of angular rate information. However, small calibration errors cause the step to be over- or underestimated resulting in a slow evolution to the actual final value. For both heading and tilt rotations, the MARG filter is also able to keep track of an angular speed up to at least 720 °/s.

A more qualitative analysis of the functionality of the entire system was obtained by performing full body experiments. A test subject was equipped with both the inertial sensor nodes and the reflective markers of an optical tracking system serving as a reference. The system operation was first validated in a dance demonstration. Screen captures of both the stickman visualisation and a video recording confirm that posture can be reconstructed. In a second test consisting of various movement exercises, the orientation of body segments is extracted from the captured data streams of the optical system and a comparison can be made between the optical and inertial estimates. From the mean absolute error, it can be concluded that the yaw angle generally exhibits larger errors. Naturally, this is a direct consequence of the coincidence of gravity with the Z-axis of the reference frame. Comparing the time graphs of the Euler angles further demonstrates that the system is able to keep up with the optical system as long as the speed of the movements is limited. Fast knee flexion and quickly turning around e.g. are easily missed. In a final running and walking experiment on a treadmill, the main source of errors is observed. Motion disturbance causing the accelerometer to measure much more than just gravity causes the filter estimates to deteriorate. With the absence of angular rate data, the MFG filter suffers from this effect during the running parts of the experiment, while the MARG filter response is more uniform.

# References

[1] Newport. *URB100CC, High-Speed Belt-Driven Rotation Stage*. Irvine, CA, USA, 2010. http://assets.newport.com/webDocuments-EN/images/14964.pdf.

[2] Newport. *SMC100CC & SMC100PP, Single-Axis Motion Controller/Driver for DC or Stepper Motor*. Irvine, CA, USA, 2005. http://assets.newport.com/webDocuments-EN/images/14444.pdf.

[3] D. Grundgeiger. *Programming Visual Basic .NET*. O'Reilly Media, May 2002.

[4] L. Taerwe. *Waarschijnlijkheidsrekening en Statistiek*. Lecture Notes, 2003.

[5] M. Demey, M. Leman, L. De Bruyn, F. Bossuyt, and J. Vanfleteren. *The Musical Synchrotron: using Wireless Motion Sensors to Study how Social Interaction Affects Synchronization with Musical Tempo*. In Proceedings of the International Conference on New Interfaces for Musical Expression, Genova, Italy, 2008.

[6] P.-J. Maes, M. Leman, M. Lesaffre, and M. Demey. *From Expressive Gesture to Sound: The Development of an Embodied Mapping Trajectory Inside a Musical Interface*. Journal on Multimodal User Interfaces, pages 67–78, 2010.

[7] M. Grachten, M. Demey, D. Moelants, and M. Leman. *Analysis and Automatic Annotation of Singer's Postures During Concert and Rehearsal*. In Proceedings of the Sound and Music Computing Conference, pages 191–198, 2010.

[8] V. Segers, M. Lenoir, P. Aerts, and D. De Clercq. *Kinematics of the Transition Between Walking and Running when Gradually Changing Speed*. Gait & Posture, 26(3):349–361, 2007.

[9] I. Van Caekenberghe, K. De Smet, V. Segers, and D. De Clercq. *Overground Versus Treadmill Walk-to-Run Transition*. Gait & Posture, 31(4):420–428, 2010.

# 6 Conclusion & Outlook

The final chapter discusses the main accomplishments of this work and possible improvements that may be added in the future.

## 6.1 General Discussion

In this work, the design of a gyroless inertial tracking system is presented. By comparing accelerometer and magnetometer information to respectively earth's gravity and magnetic field, driftless absolute orientation is obtained. While traditionally angular rate measured by gyroscopes is integrated offering a first estimate, this sensor is deliberately omitted in order to drastically reduce the current consumption. Furthermore, this approach allows the use of smaller, single board sensor nodes with a restricted height and enables to increase the unobtrusiveness by applying advanced packaging techniques. Both of these aspects clearly enhance the mobile nature of the tracking system.

The system comprises of several parts that contribute to the functionality. Sensor data originates from commercial MEMS devices included in the hardware design of a sensor node that also contains a processing unit and a wireless interface. Embedded software running on the microcontroller obtains the data from the sensors and implements a communication protocol that allows the data to be transferred reliably to a base station, which in turn passes it through to a backend computer. Finally, software is used to calculate an orientation estimate from the received data and a visualisation is built on the monitor. Each of these parts needed to be developed and properly tested.

The Kalman filter was selected to serve as a sensor fusion algorithm for obtaining an orientation estimate. Several flavours of the filter architecture were applied to the problem at hand using both Euler angle and quaternion representation. Additional features, such as feedback and adaptive noise covariance were added to improve the performance in the absence of angular rate information. All parameters of the filters were estimated using real life motion captures from an optical tracking system in order to ensure an adequate modeling of the problem. Simulations of the estimation algorithm were executed to test its characteristics and determine an estimate for the adaptive parameters in the system. Step response simulations revealed that the filter behaves differently for tilt and heading steps as was expected due to the coincidence of the Z-axis and the gravity vector. Settling times ranged from 30 ms for tilt to 60 ms for heading changes. Noise response simulations with both simulated and real noise showed that the variance on the output signals was ten times higher with the latter due to poor whiteness. However, pre-filtering the sensor output reduced this variance back to approximately the same level as the variance of the simulated noise response. Finally, motion disturbance simulations, where a disturbing sinusoidal signal is added to one of the accelerometer outputs, confirmed the efficiency of the adaptive filtering approach. A value of 1000 for the adaptive filtering parameter $\zeta$ significantly reduced the mean square error on the output of both the Euler and quaternion type filters.

The sensor node hardware design first focused on creating microsystems that consume very little power. Later, user comfort was increased by using flexible board technology and chip thinning. Each of the nodes includes a microcontroller connected to both three dimensional, fully integrated sensors and an RF transceiver. The functionality was carefully implemented in embedded software to further minimise power consumption. To this extend, a fully plug and play, wireless ad hoc network protocol was introduced based on a master and slave hierarchy and a TDMA like network scheme. The role of a node and the timeslot it occupies is determined at runtime according to availability. Synchronisation in the network is provided by the package transmissions of the master. Therefore, its presence is carefully monitored by the slaves and upon failure, one of them will take on the master role. Additional mechanisms ensure that only one master is active and a timeslot is only used by a single slave. In an attempt to reduce the computational stress on the backend computer, the orientation estimator was also implemented in firmware using a fixed point number format. The microcontroller's hardware multiplier and Newton's approximation method allowed to implement both multiplication and square root operations efficiently. Matrix symmetry further reduced the number of instructions required for each iteration. Finally, third generation sensor nodes were able to calculate an orientation estimate within the available 10 ms timeframe and a single base station could accommodate a maximum of 19 nodes while each of them consumes an average current of 6.5 mA.

Aside from implementing the estimation algorithm when this is not yet completed by the microcontroller, the computer software allows visualisation by means of a graph, a rotating cube or an animated stickman. For the latter, a recursive tree structure was built from bone class objects that are interconnected by joints. Using the swing-twist parameterisation, joint limits were defined in order to correct anatomically impossible postures to feasible ones. Bones were then categorised according to the amount of freedom allowed by the joint connecting them to their parent. Free bones simply copy the orientation given by their associated sensor node, while fixed bones follow the orientation of their parent. Single plane constraint bones are connected through a hinge type joint allowing the swing to position the bone into a half-plane. A ball-and-socket joint leads to a multiple constraint bone where the swing restrictions are modeled by a cone of forbidden directions. The twist of the corrected bones is bound separately to fixed values. Smooth visualisations were obtained by tolerating small deviations on each of the restrictions. The limits were chosen fairly liberal and screenshots have shown the functionality of the model.

Using a uniaxial rotational stage, the tracking performance for both tilt and heading rotations of a single node was obtained. The linearity of the filters was determined to depend on the accuracy of the calibration since the error was systematic, especially for the heading. The step response showed a delay of 80 ms for heading changes and 60 ms for tilt. A jogging test finally revealed that tilt rotations up to at least 720 °/s could be tracked with a delay around 120 ms and a mean absolute error below 10°. For the heading case, cycle slips occurred when the angular speed exceeded 600 °/s. A qualitative assessment of the entire system functionality was obtained by performing full body tracking experiments. From a dance demonstration, video captures and stickman visualisations validated the system operation. In other experiments, an optical system served as golden standard given its proven track record. A comparison of the output data showed that the inertial system was capable of tracking many of the movements of the test subject, as long as the influence of motion disturbance was restricted. Clearly, the absence of the gyroscopes means that only relatively slow motion can be tracked accurately, meaning that the system could be interesting for revalidation applications. Even with faster movement however, the type of action may still be determined from the output information. As a result, the gyroless trackers could certainly be used in gaming applications where high level classification of motion is of intrest. Furthermore, if very long capture sessions are required, the low current consumption of the MFG nodes is a clear asset.

## 6.2 Future Work

Throughout the entire world, one important constant exists within research: the work is never done, there is always room for improvement. Therefore, this final section is devoted to some of these aspects.

The filters that have been developed within this work use adaptive techniques to address the issue of disturbance on the sensor signals. However, full body experiments have clearly shown that motion disturbance has a bigger influence than initially expected. One way of improving the behaviour of the filter is to attempt to estimate the acceleration due to motion. However, this would require an additional part in the state vector containing this acceleration component which increases the complexity of the filter significantly. Furthermore, a model would be needed in order to estimate the acceleration which might not be easy to obtain given the absence of gyroscopes. A different approach is to take the estimation one level higher. Using knowledge about the location of a sensor node on the body and combining this with a more elaborate human model allows to estimate the errors and motion disturbance more precisely and individually for each node. One could attempt to detect the general motion of the person and adapt the parameters of the filter according to that specific movement. During a walking exercise for example, shocks on the accelerometer values are expected when the user's foot hits the ground. In this case, different models would need to be developed and additional mechanisms are needed to detect which model is appropriate at that specific time.

Concerning the hardware, the first steps towards the use of flexible board technology and component integration have already been taken. Logically, further work in this area includes the actual production of this type of nodes. However, aside from the integration of the microcontroller and the RF transceiver, the sensors might also be considered for UTCP. Given the fact that these devices mostly consist of both a sensor MEMS chip and a silicon controller implementing the interface, this still remains to be a challenge. On the base station side, additional work includes the network of networks implementation where motion capturing of multiple persons could be enabled by having each of the individuals carry around their own intermediate access point. All of these access point could then be networked separately from the sensor networks to communicate to the backend. This would allow that the sensor nodes themselves use less power for wireless transmission since the distance to be covered is clearly lower. The range of the system then depends on the quality of the link from the access points to the backend. However, the downside is that these access points are more power hungry and would probably not be extremely unobtrusive. Another approach that does not suffer from this is to place multiple base stations distributed in the environment.

The communication to the backend could in this case be wired if feasible or wireless if required.

When considering different network architectures for the base stations, the sensor network protocol could also require adaptations. The efficiency of the protocol as it has been described in chapter 3 lies in the fact that data is only transferred from the nodes to the base station. However, this feature is at the same time also its greatest weakness. Small adaptations cannot be completed without fully reprogramming the nodes. When nodes need to be calibrated e.g. this requires manual work from the user since they are programmed to continuously send data to the base station. Any change to this way of working will however require to completely redefine the entire protocol. Two-way communication would be needed and the base station will need to actively participate in the network. This enables that the base station starts controlling the network and keeps track of its health. Base stations might also direct nodes to transmit their information to a different base station in order to maintain a better link quality. However, the price to pay is some loss in efficiency of the data transfer which is bound to decrease due to the fact that valuable time is lost.

Finally, some improvements to the human model might also be considered. As described in chapter 4, the model used at the moment corrects the orientation of a bone by considering the orientation of the parent as correct. Furthermore, one of the bones, denoted as the root, always remains uncorrected since otherwise certain poses would generally remain impossible. A different approach is to consider the entire posture that results from the direct application of the estimated orientation from the nodes and to determine the most likely corrections that are needed to obtain an anatomically feasible posture. Furthermore, the posture that was determined in the previous update might also be considered. Note that some of the bones are still more likely to be corrected more than others, since they are prone to bigger errors due to motion disturbance. Another possible improvement consists of using the human model information to provide feedback to the orientation estimator as was mentioned in the beginning of this section. Major corrections on the orientation of a certain bodypart e.g. might indicate that the node associated with this bodypart is exhibiting motion disturbance. Shocks during walking can also be predicted as they occur when the anchorpoint of the stickman with the floor is transferred from the endpoint of one leg to the other.

# A
# Quaternion Decomposition

## A.1 Swing-Twist Decomposition

In this appendix, the mathematical derivation for the decomposition of a quaternion into two separate quaternions is given. One component is called the swing as it describes the change of one direction to the next, while the other component is referred to as the twist as it corresponds to a rotation about the direction vector. Both components describe rotations that are essentially executed around mutually perpendicular axes.

In the first section, it is assumed that the twist axis corresponds to the Z-axis, while the swing axis lies in the XY-plane. The second section describes the decomposition in the more general case where the twist axis is known but corresponds to an arbitrary axis.

## A.2 Z-Axis Twist

A rotation quaternion **q** will be decomposed into two subquaternions, a swing quaternion $\mathbf{q}_s$ and a twist quaternion $\mathbf{q}_t$:

$$\mathbf{q}_s \otimes \mathbf{q}_t = \mathbf{q} \tag{A.1}$$

Right multiplying both sides by the inverse of **q** and left multiplying be the inverse of $\mathbf{q}_s$ results in:

$$\mathbf{q}_t \otimes \mathbf{q}^{-1} = \mathbf{q}_s^{-1} \tag{A.2}$$

Since all quaternions represent a rotation, the inverse can also be replaced by the conjugate:

$$\mathbf{q}_t \otimes \mathbf{q}^{*} = \mathbf{q}_s^{*} \tag{A.3}$$

The full rotation quaternion $\mathbf{q}$ represents a random rotation in 3D space. This quaternion is to be decomposed into a twist quaternion $\mathbf{q}_t$, which only contains a rotation around the Z-axis and a swing quaternion $\mathbf{q}_s$, containing the direction component. Each of the quaternions can be written as a sum of coefficients:

$$\mathbf{q} = w + x\,i + y\,j + z\,k \tag{A.4a}$$
$$\mathbf{q}_t = w_t + z_t\,k \tag{A.4b}$$
$$\mathbf{q}_s = w_s + x_s\,i + y_s\,j \tag{A.4c}$$

Rewriting (A.3) by means of the coefficients results in:

$$w_s = w_t\,w + z_t\,z \tag{A.5a}$$
$$-x_s = -w_t\,x + z_t\,y \tag{A.5b}$$
$$-y_s = -w_t\,y - z_t\,x \tag{A.5c}$$
$$0 = -w_t\,z + z_t\,w \tag{A.5d}$$

The norm of each of the quaternions must also equal unity, as the quaternions represent a rotation:

$$|\mathbf{q}|^2 = w^2 + x^2 + y^2 + z^2 = 1 \tag{A.6a}$$
$$|\mathbf{q}_s|^2 = w_s^2 + x_s^2 + y_s^2 = 1 \tag{A.6b}$$
$$|\mathbf{q}_t|^2 = w_t^2 + z_t^2 = 1 \tag{A.6c}$$

Substituting (A.5a) – (A.5c) in (A.6b) results in:

$$(w_t\,w + z_t\,z)^2 + (w_t\,x - z_t\,y)^2 + (w_t\,y + z_t\,x)^2 = 1 \tag{A.7}$$

Expanding the squares yields:

$$\begin{aligned} w_t^2\, w^2 \;+\; 2\, w\, z\, w_t\, z_t \;+\; z_t^2\, z^2 & \\ +\; w_t^2\, x^2 \;-\; 2\, x\, y\, w_t\, z_t \;+\; z_t^2\, y^2 & \\ +\; w_t^2\, y^2 \;+\; 2\, x\, y\, w_t\, z_t \;+\; z_t^2\, x^2 \;=\; 1 & \end{aligned} \tag{A.8}$$

Scrapping and combining terms:

$$\left(w^2 \;+\; x^2 \;+\; y^2\right) w_t^2 \;+\; \left(x^2 \;+\; y^2 \;+\; z^2\right) z_t^2 \;+\; 2\, w\, z\, w_t\, z_t \;=\; 1 \tag{A.9}$$

Using (A.6a) and (A.5d) further simplifies the equation to:

$$\left(1 \;-\; z^2\right) w_t^2 \;+\; \left(1 \;-\; w^2\right) z_t^2 \;+\; 2\, z_t^2\, w^2 \;=\; 1 \tag{A.10}$$

Using (A.6c) leads to:

$$\left(1 \;-\; z^2\right) \left(1 \;-\; z_t^2\right) \;+\; \left(1 \;+\; w^2\right) z_t^2 \;=\; 1 \tag{A.11}$$

Further calculus reveals:

$$z_t^2 \;=\; \frac{z^2}{z^2 \;+\; w^2} \tag{A.12}$$

Combining (A.12) and (A.5d) finally results in:

$$w_t \;=\; \frac{\pm\, w}{\sqrt{w^2 \;+\; z^2}} \tag{A.13a}$$

$$z_t \;=\; \frac{\pm\, z}{\sqrt{w^2 \;+\; z^2}} \tag{A.13b}$$

Where the duality in sign clearly reflects the fact that both a unit quaternion and its negative represent the same rotation.

Finally, the swing quaternion can also be calculated:

$$\mathbf{q}_s \;=\; \mathbf{q} \;\otimes\; \mathbf{q}_t^* \tag{A.14}$$

Note however that the decomposition introduces a singularity. Equations (A.13a) and (A.13b) result in an exception when both $z$ and $w$ are zero. This can be rephrased using the angle-axis formalism for quaternions:

$$w \;=\; cos\left(\frac{\phi}{2}\right) \;=\; 0 \tag{A.15a}$$

$$z \;=\; v_z\, sin\left(\frac{\phi}{2}\right) \;=\; 0 \tag{A.15b}$$

From these equations, the problem clearly arises when the following conditions are met:

$$\phi = 180^\circ \tag{A.16a}$$

$$v_z = 0 \tag{A.16b}$$

A quaternion fulfilling these conditions corresponds to a 180° rotation around an axis in the XY-plane, or a 180° swing rotation. In this case, the twist angle is free to choose, a different choice will only lead to a different swing quaternion. This can easily be understood with a simple example. Suppose $\mathbf{q}$ represents a 180° rotation around the X-axis. A simple decomposition of this quaternion would be to have zero twist and $\mathbf{q}_s$ equal $\mathbf{q}$. Yet, another valid decomposition is obtained by choosing the twist angle to be 180° and assigning a 180° rotation around the Y-axis to $\mathbf{q}_s$:

$$\begin{aligned} \mathbf{q}_s \otimes \mathbf{q}_t &= [0,\ 0,\ 1,\ 0] \otimes [0,\ 0,\ 0,\ 1] && \text{(A.17)} \\ &= [0,\ 1,\ 0,\ 0] && \text{(A.18)} \\ &= \mathbf{q} && \text{(A.19)} \end{aligned}$$

An infinite number of other decompositions exist resulting in the same combined rotation. Testing for this singularity and simply choosing a twist angle before calculating the swing quaternion is a common solution.

## A.3 Arbitrary Twist

In this section, no limitations are put forward for the direction of the twist axis but it is assumed that the axis is known. Using the axis-angle formalism, both the quaternions can be rewritten as follows:

$$\mathbf{q}_s = [w_s,\ \boldsymbol{v}_s] = \left[\cos\frac{\sigma}{2},\ \boldsymbol{u}_s \sin\frac{\sigma}{2}\right] \tag{A.20}$$

$$\mathbf{q}_t = [w_t,\ \boldsymbol{v}_t] = \left[\cos\frac{\tau}{2},\ \boldsymbol{u}_t \sin\frac{\tau}{2}\right] \tag{A.21}$$

Where $\boldsymbol{u}_t$ and $\boldsymbol{u}_s$ respectively represent a unit vector along the twist and swing axes. Substituting these expressions in (A.1) yields:

$$\begin{aligned} \mathbf{q} &= \mathbf{q}_s \otimes \mathbf{q}_t && \text{(A.22)} \\ &= [w_s w_t - \boldsymbol{v}_t \cdot \boldsymbol{v}_s,\ w_s \boldsymbol{v}_t + w_t \boldsymbol{v}_s + \boldsymbol{v}_s \times \boldsymbol{v}_t] && \text{(A.23)} \\ &= [w_s w_t,\ w_s \boldsymbol{v}_t + w_t \boldsymbol{v}_s + \boldsymbol{v}_s \times \boldsymbol{v}_t] && \text{(A.24)} \end{aligned}$$

In the calculations, the fact is used that the rotation axes of the twist and swing quaternion are mutually perpendicular and that the scalar product of two perpendicular vectors is zero. A new quaternion $\mathbf{q}_p$ is defined as the projected version of the initial quaternion $\mathbf{q}$ on the twist axis:

$$\begin{aligned}
\mathbf{q}_p &= [w,\ (\boldsymbol{v} \cdot \boldsymbol{u}_t)\ \boldsymbol{u}_t] && \text{(A.25)}\\
&= [w_s\, w_t,\ (w_s\, \boldsymbol{v}_t \cdot \boldsymbol{u}_t\ +\ w_t\, \boldsymbol{v}_s \cdot \boldsymbol{u}_t\ +\ (\boldsymbol{v}_s \times \boldsymbol{v}_t) \cdot \boldsymbol{u}_t)\ \boldsymbol{u}_t] && \text{(A.26)}\\
&= \left[w_s\, w_t,\ w_s\ |\boldsymbol{u}_t|^2\ \sin\frac{\tau}{2}\ \boldsymbol{u}_t\right] && \text{(A.27)}\\
&= [w_s\, w_t,\ w_s\, \boldsymbol{v}_t] && \text{(A.28)}
\end{aligned}$$

Here, the fact is used that the vector product of two vectors yields a new vector that is perpendicular to both arguments, thus its scalar product with a vector parallel to one of the arguments is always zero. By normalising this quaternion, the following result is obtained:

$$\begin{aligned}
\frac{\mathbf{q}_p}{|\mathbf{q}_p|} &= \frac{[w_s\, w_t,\ w_s\, \boldsymbol{v}_t]}{\sqrt{w_s^2\, w_t^2\ +\ w_s^2\ |\boldsymbol{v}_t|^2}} && \text{(A.29)}\\
&= \frac{w_s\ [w_t,\ \boldsymbol{v}_t]}{w_s \sqrt{w_t^2\ +\ |\boldsymbol{v}_t|^2}} && \text{(A.30)}\\
&= \frac{[w_t,\ \boldsymbol{v}_t]}{|\mathbf{q}_t|} && \text{(A.31)}\\
&= [w_t,\ \boldsymbol{v}_t] && \text{(A.32)}\\
&= \mathbf{q}_t && \text{(A.33)}
\end{aligned}$$

This shows that the normalised version of the projected quaternion equals the twist quaternion. Since $\mathbf{q}_p$ can be obtained using only the original quaternion and the axis of twist rotation, $\mathbf{q}_t$ can thus be calculated for any arbitrary twist axis. The swing quaternion can still be calculated using (A.14).

As was the case for the decomposition with a twist around the Z-axis, a singularity arises when the quaternion consists of a pure swing over 180°. This can best be understood by keeping in mind that the swing quaternion of such a decomposition will always have a zero scalar part, from which immediately follows that the projected quaternion equals zero and the normalisation is actually a division by zero. Detecting the singularity can hence also be executed by checking if the projected quaternion is not to close to the zero quaternion, in which case the twist quaternion can be arbitrarily chosen.

# B

# Second Generation Firmware

This appendix contains the firmware of the second generation motion tracking sensor nodes.

## B.1 Hardware

*Code Segment B.1: Hardware.h*

```
#define F_CPU 7372800UL

#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>

#define low(port, pin) (port &= ~_BV(pin))
#define high(port, pin) (port |= _BV(pin))

//#define FIRST

//CYWM pins
#define CYWM_SCK      PB5    // Output
#define CYWM_MISO     PB4    // Input
#define CYWM_MOSI     PB3    // Output
#define CYWM_nSS      PD7    // Output

#define CYWM_nPD      PC1    // Output
#define CYWM_nRESET     PD3    // Output
#define CYWM_IRQ      PD2    // Input
```

```
#define CYWM_PACTL      PB0    // Input
#define CYWM_RXPA    PB2    // Input
#define CYWM_TXPA    PD4    // Input

#define CYWM_nSS_PORT PORTD


//LED Pin
#define LED_PIN       PD1    // Output
#define LED_PORT      PORTD
#define LED_DDR       DDRD

//Magnetometer Reset pin
#define MAGRESET      PD0 //reset pin magnetometer

void initLEDPort (void);
void setLED (void);
void clearLED (void);
void toggleLED(void);
void Master_Timer_Init (void);
void Slave_Timer_Init (unsigned char TimeSlot);
void Timer_Start_Up(unsigned char Number);
void Timer_Master_Check(void);
void Timer_Slave_Scan(void);
void Reset_Timer(void);
void Init_Pin_interrupt(void);
void Port_Init(void);
void ADC_Init(void);
```

*Code Segment B.2: Hardware.c*

```
#include "hardware.h"
#include <stdlib.h>

void initLEDPort (void)
{
  clearLED();

  // set PB1 as output
  LED_DDR |= (1<<LED_PIN);
}

// switch the LED on
void setLED(void)
{
  // set to inactive / LED is ON
  LED_PORT &= ~(1<<LED_PIN);
}

// switch the LED off
void clearLED (void)
{
  // set to active / LED is off
  LED_PORT |= (1<<LED_PIN);
}

void toggleLED(void) {
  LED_PORT ^= (1<<LED_PIN);
}

void Master_Timer_Init(void)
{
  //Initialize 16 bit counter1:
  TCCR1A = 0x00;

  //clk setting: clk_I/O/8 and CTC mode: up to ICR
  TCCR1B = 0x1A;

  //interrupt on count to 9216 = 10ms
  ICR1H = 0x24;
  ICR1L = 0x00;

  //Enable ICR compare interrupt interrupt
  TIMSK1 = 0x20;
}

void Slave_Timer_Init(unsigned char TimeSlot)
{

  //Initialize 16 bit counter1:
  TCCR1A = 0x00;

  //clk setting: clk_I/O/8 and CTC mode: up to ICR
  TCCR1B = 0x1A;

  //interrupt on count to 9216 = 10ms
  ICR1H = 0x24;
  ICR1L = 0x00;

  //interrupt depends on timeslot
  OCR1AH = 0x03 * TimeSlot;
  OCR1AL = 0x00;

  //interrupt on count to 7680 ~= 8.33ms
  OCR1BH = 0x1E;
  OCR1BL = 0x00;
  TIMSK1 = 0x00;
}

void Timer_Start_Up(unsigned char Number)
{
  //Initialize 16 bit counter1:
  TCCR1A = 0x00;

  //clk setting: clk_I/O/64 and CTC mode: up to ICR
  TCCR1B = 0x1B;
  ICR1H = 0x40;
  ICR1L = 0x00;
  TIMSK1 = 0x20;
}

void Timer_Master_Check(void)
{
  //Initialize 16 bit counter1:
```

```
  TCCR1A = 0x00;

  //clk setting: clk_I/O/64 and CTC mode: up to ICR
  TCCR1B = 0x1B;

  //interrupt on count +- 100 ms
  ICR1H = 0x30;
  ICR1L = 0x00;
  TIMSK1 = 0x20;
}

void Timer_Slave_Scan(void)
{
  //Initialize 16 bit counter1:
  TCCR1A = 0x00;

  //clk setting: clk_I/O/64 and CTC mode: up to ICR
  TCCR1B = 0x1B;

  //interrupt on count +- 100 ms
  ICR1H = 0x30;
  ICR1L = 0x00;
  TIMSK1 = 0x20;
}

void Reset_Timer(void)
{
  TCCR1B = 0x00;
  TCNT1 = 0;
}

void Init_Pin_interrupt(void)
{
  //Allow interrupt (rising edge) on INT0, IRQ from RF chip
  EICRA = 0x03;
  EIMSK = 0x00;
}

void Port_Init(void)
{
  //port I/O directions
  DDRB = _BV(CYWM_SCK) | _BV(CYWM_MOSI)|
              _BV(CYWM_RXPA)| _BV(CYWM_PACTL);
  DDRC = _BV(CYWM_nPD);
  DDRD |= _BV(CYWM_nSS) | _BV(CYWM_nRESET)|
              _BV(CYWM_TXPA) | _BV(MAGRESET) | _BV(LED_PIN);
  clearLED();
}
```

## B.2 I$^2$C

*Code Segment B.3: I2C.h*

```
void I2CStart(void);
void I2CSendData(unsigned char data);
unsigned char I2CReceive(void);
unsigned char I2CReceiveNMAK(void);
void I2CStop(void);
```

*Code Segment B.4: I2C.c*

```
#include "I2C.h"
#include "hardware.h"

void I2CStart(void)
{
  // send start condition
  TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
  // wait for complete
  loop_until_bit_is_set(TWCR, TWINT);
}

void I2CSendData(unsigned char data)
{
  TWDR = data;
  TWCR = (1<<TWINT) | (1<<TWEN);
  // wait for complete
  loop_until_bit_is_set(TWCR, TWINT);
}

unsigned char I2CReceive(void)
{
  TWCR = (1<<TWINT) |  (1<<TWEN)| (1<<TWEA) ;
  // wait for complete
  loop_until_bit_is_set(TWCR, TWINT);
  return (TWDR);
}

unsigned char I2CReceiveNMAK(void)
{
  TWCR = (1<<TWINT) |  (1<<TWEN)| (0<<TWEA) ;
  // wait for complete
  loop_until_bit_is_set(TWCR, TWINT);
  return (TWDR);
}

void I2CStop(void)
{
  //Send stop condition
  TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN); //TWBR = 50;
}
```

## B.3 SPI

*Code Segment B.5: SPI.h*

```
void SPI_Write(unsigned char byte);
void Spi_Init(void);
```

*Code Segment B.6: SPI.c*

```
#include "SPI.h"
#include "hardware.h"

void SPI_Write(unsigned char byte)
{
  SPDR = byte;
  // Wait for SPI transmission complete
  while(!(SPSR & (1<<SPIF)));
}

void Spi_Init(void)
{
  // Setup SPI
  SPCR = (1<<SPE) | (1<<MSTR);
  SPSR |= (1<<SPI2X);
  high(CYWM_nSS_PORT, CYWM_nSS);
  return;
}
```

## B.4 Fixed Point

*Code Segment B.7: FixedPoint.h*

```
short fpMult(short a, short b);
```

*Code Segment B.8: FixedPoint.c*

```
#include "fixedPoint.h"

short fpMult(short a, short b)
{
  //Declare return variable
  short c;

  //part 1: a * integer part b
  c = a * (b >> 10);

  //part 2: a * decimal part b
  for(unsigned char i = 0 ; i < 10 ; i++) {
    if( b & (1 << i)) {
      c += (a >> (10-i));
    }
  }

  //Return result
  return c;
}
```

## B.5 Accelerometer

*Code Segment B.9: Acc.h*

```
void init_Acc(void);
void readAcc(void);
```

*Code Segment B.10: Acc.c*

```
#include "acc.h"
#include "CYWUSB693x.h"
#include "hardware.h"
#include "I2C.h"


void init_Acc(void)
{

  // set bitrate register to 5, i.e. ~2.7usec SCK cycle time
  TWBR = 50;

  // send start condition
  I2CStart();

  // Send i2c address with r/w bit one
  I2CSendData(0x3A); // set data byte 00111010

  // send sub
  I2CSendData(0x20);
  I2CSendData(0x97);

  //Send stop condition
  I2CStop();
}

void readAcc(void)
{
  //Init variables
  unsigned char outx_L,outx_H,outy_L,outy_H,outz_L,outz_H;

  //Communicate via I2C
  I2CStart();
  I2CSendData(0x3A);
  I2CSendData(0xA8);
  I2CStart();
  I2CSendData(0x3B);
  outx_L = I2CReceive();
  outx_H = I2CReceive();
  outy_L = I2CReceive();
  outy_H = I2CReceive();
  outz_L = I2CReceive();
  outz_H = I2CReceiveNMAK();
  I2CStop();

  //Write values to sendbuffer
  CYWM_AddToTXBuffer((((outx_L>>4) & 0x0F) | ((outx_H<<4) & 0xF0)));
  CYWM_AddToTXBuffer((((outy_L>>4) & 0x0F) | ((outy_H<<4) & 0xF0)));
  CYWM_AddToTXBuffer((((outz_L>>4) & 0x0F) | ((outz_H<<4) & 0xF0)));
}
```

## B.6 Magnetometer

*Code Segment B.11: Mag.h*

```
#define I2CMAGREAD      0x5D
#define I2CMAGWRITE    0x5C

void initMag(unsigned char SENSOR);
void readMag(void);
```

*Code Segment B.12: Mag.c*

```
#include "mag.h"
#include "CYWUSB693x.h"
#include "I2C.h"
#include "fixedPoint.h"
#include "hardware.h"
#include "calibration.h"

unsigned char CAL0= 0;  //Cal bytes
unsigned char CAL1= 0;  //Cal bytes
unsigned char CAL2= 0;  //Cal bytes
unsigned char CAL3= 0;  //Cal bytes
unsigned char CAL4= 0;  //Cal bytes
unsigned char CAL5= 0;  //Cal bytes
unsigned char CAL6= 0;  //Cal bytes
unsigned char CAL7= 0;  //Cal bytes
unsigned char CAL8= 0;  //Cal bytes

unsigned char roughx= 0;  //Cal bytes
unsigned char roughy1= 0;  //Cal bytes
unsigned char roughy2= 0;  //Cal bytes

float a[9];
short b[9];

void initMag(unsigned char SENSOR)
{

  //Init variables
  unsigned char rough0= 0;
  unsigned char rough1= 0;
  unsigned char rough2= 0;
  unsigned char rough3= 0;
  unsigned char rough4= 0;
  unsigned char rough5= 0;

  //Initialize registers: write zeros
  I2CStart();
  I2CSendData(I2CMAGWRITE);
  I2CSendData(0b10000000);
  I2CStart();
  I2CSendData(I2CMAGWRITE);
  I2CSendData(0b11000000);

  //Activate initialization coils
  for(unsigned char i = 0;i<8;i++) {
    I2CStart();
    I2CSendData(I2CMAGWRITE);
    I2CSendData(0b10010000 | i);

    I2CStart();
    I2CSendData(I2CMAGWRITE);
    I2CSendData(0b10000000 | ((i+1)&0b00000111));
  }

  //Read factory calibration
  I2CStart();
  I2CSendData(I2CMAGWRITE);
  I2CSendData(0b11001000); //subadres

  I2CStart();
  I2CSendData(I2CMAGREAD);
  CAL0 = I2CReceive();
  CAL1 = I2CReceive();
  CAL2 = I2CReceive();
  CAL3 = I2CReceive();
  CAL4 = I2CReceive();
  CAL5 = I2CReceive();
  CAL6 = I2CReceive();
  CAL7 = I2CReceive();
  CAL8 = I2CReceiveNMAK();
  I2CStop();

  //Perform rough offset measurement
  I2CStart();
  I2CSendData(I2CMAGWRITE);
  I2CSendData(0b11000000); //subadres
  _delay_ms(3);
  I2CStart();
  I2CSendData(I2CMAGWRITE);
  I2CSendData(0b00000001); //subadres
  _delay_ms(3);
  I2CStart();
  I2CSendData(I2CMAGREAD);
  rough5= I2CReceive(); //rough values
  rough4= I2CReceive();
  rough3= I2CReceive();
  rough2= I2CReceive();
  rough1= I2CReceive();
  rough0= I2CReceiveNMAK();

  //Calculate rough offsets
  #ifdef FIRST
  roughx = 15;
  roughy1 = 26;
```

```
roughy2 = 11;

//EEPROM Write Magnetometer Rough offsets
SetRoughx(roughx);
SetRoughy1(roughy1);
SetRoughy2(roughy2);

#else

//Retrieve roughs from EEPROM
roughx = RetrieveRoughx();
roughy1 = RetrieveRoughy1();
roughy2 = RetrieveRoughy2();
#endif

//Fixing calibration issues?
if (roughx != (rough0 & 0b00011111))
{
  for(;;)
  {
    for(int i = 0; i < 10000 ; i++);
  }
}
if (roughy1 != (rough2 & 0b00011111))
{
  for(;;)
  {
    for(int i = 0; i < 10000 ; i++);
  }
}
if (roughy2 != (rough4 & 0b00011111))
{
  for(;;)
  {
    for(int i = 0; i < 10000 ; i++);
  }
}

//Write the rough offsets to the registers
I2CStart();
I2CSendData(I2CMAGWRITE);
I2CSendData(0b00100000 | (roughx-5));
I2CStart();
I2CSendData(I2CMAGWRITE);
I2CSendData(0b01000000 | (roughy1-5));
I2CStart();
I2CSendData(I2CMAGWRITE);
I2CSendData(0b01100000 | (roughy2-5));

//Order mag to perform first measurement
I2CStart();
I2CSendData(I2CMAGWRITE);
I2CSendData(0b00000000);
I2CStop();

//Calculate calibration values as float
a[0] = 1;
```

```
  a[1] = ((CAL0 & 0xFC)>>(2)) - 32;
  a[1] /= 100;
  a[2] = ((CAL0 & 0x03)<<(2)) + ((CAL1 & 0xC0)>>(6)) - 8;
  a[2] /= 100;
  a[3] = (CAL1 & 0x3F) - 32;
  a[3] /= 100;
  a[4] = ((CAL2 & 0xFC)>>(2)) - 32;
  a[4] = a[4] / 100 + 0.7;
  a[5] = ((CAL2 & 0x03)<<(4)) + ((CAL3 & 0xF0)>>(4)) - 32;
  a[5] /= 100;
  a[6] = ((CAL3 & 0x0F)<<(2)) + ((CAL4 & 0xC0)>>(6)) - 32;
  a[6] /= 100;
  a[7] = (CAL4 & 0x3F) - 32;
  a[7] /= 100;
  a[8] = ((CAL5 & 0xFE)>>1) - 64;
  a[8] = a[8] / 100 + 1.3;

  //Calculate fixed point representation
  for(uint8_t i = 0;i<9;i++) {
    if (a[i]<0){
      a[i] = -a[i];
      b[i] = 0x8000;
    }
  }
  float test = 16;
  for(uint8_t i=1;i<16;i++){
    for(uint8_t j=0;j<9;j++){
      if (a[j] >= test){
        a[j] -= test;
        b[j] |= 1<<(15-i);
      }
    }
    test /= 2;
  }
  for(uint8_t i = 0;i<9;i++) {
    if (b[i] >> 15){
      b[i] -= 0x8001;
      b[i] ^= 0xFFFF;
    }
  }
}

void readMag(void)
{
  //Init variables
  unsigned char waarde0 = 0;
  unsigned char waarde1 = 0;
  unsigned char waarde2 = 0;
  unsigned char waarde3 = 0;
  unsigned char waarde4 = 0;
  unsigned char waarde5 = 0;

  //Communicate via I2C
  I2CStart();
  I2CSendData(I2CMAGREAD);
  waarde5 = I2CReceive();
  waarde4 = I2CReceive();
  waarde3 = I2CReceive();
  waarde2 = I2CReceive();
  waarde1 = I2CReceive();
  waarde0 = I2CReceiveNMAK();
  I2CStop();

  //Command mag to perform new measurement for next read
  I2CStart();
  I2CSendData(I2CMAGWRITE); //adres is 2e en schrijven
  I2CSendData(0b00000000);
  I2CStop();

  //Declare fixed point variables, short = 16 bit variable.
  short x,y,z;

  //Calculate fixed point representation and add rough offset value
  x = ((((roughx - 5) + (waarde1 & 0x07)) << 10)
          | (waarde0 << 2)) - (15 << 10);
  y = ((((roughy1 - 5) + (waarde3 & 0x07)) << 10)
          | (waarde2 << 2)) - (15 << 10);
  z = ((((roughy2 - 5) + (waarde5 & 0x07)) << 10)
          | (waarde4 << 2)) - (15 << 10);

  //Convert the axis values into the standard coordinate system
  x = -x;
  short temp = z - y;
  z = y + z;
  y = temp;

  //Apply factory calibration
  short MData[3];
  MData[0] = x              + fpMult(b[1], y) + fpMult(b[2], z);
  MData[1] = fpMult(b[3], x) + fpMult(b[4], y) + fpMult(b[5], z);
  MData[2] = fpMult(b[6], x) + fpMult(b[7], y) + fpMult(b[8], z);

  //Add values to sendbuffer
  CYWM_AddToTXBuffer(MData[0]);
  CYWM_AddToTXBuffer(MData[0]>>8);
  CYWM_AddToTXBuffer(MData[1]);
  CYWM_AddToTXBuffer(MData[1]>>8);
  CYWM_AddToTXBuffer(MData[2]);
  CYWM_AddToTXBuffer(MData[2]>>8);
}
```

## B.7 Gyroscope

*Code Segment B.13: Gyr.h*

```
void ADC_Init(void);
void readGyr(void);
```

*Code Segment B.14: Gyr.c*

```
#include "gyr.h"
#include "I2C.h"
#include "CYWUSB693x.h"
#include "hardware.h"

void ADC_Init(void) {
  //ADC ucontroller
  DIDR0 = 0x0c;
  ADMUX = (0<<REFS1) | (1<<REFS0) |  (1<<ADLAR) |0x03 ;

  //Gyro-pins
  DDRC  &= ~(1<<PIN0);
  DDRC  &= ~(1<<PIN1);
  DDRC  &= ~(1<<PIN2);
  DDRB  |= (1<<PIN1);

  //ADC extern
  I2CStart();
  I2CSendData(0x42);
  I2CSendData(0x02);
  I2CSendData(0x70);   //channel 1,2,3 enable en filter enable
  I2CStop();
}

uint8_t ADC_Value(void) {
  ADMUX = (1<<REFS1) | (1<<REFS0) |   (1<<ADLAR) | 3; // 1.1Vref
  ADCSRA = (1<<ADEN) | (1<<ADSC) | (3<<ADPS0); // ADC3
  while (ADCSRA & (1<<ADSC));
  return ADCH;
}

void readGyr(void)
{
  //Read gyro output via internal ADC
  //Gyro x: ADC2
  ADMUX = (0<<REFS1) | (1<<REFS0) |  (1<<ADLAR) |0x02 ;
  ADCSRA = (1<<ADEN) | (1<<ADSC) | (3<<ADPS0);
  while (ADCSRA & (1<<ADSC));
  CYWM_AddToTXBuffer(ADCH-118);

  //Gyro y: ADC1
  ADMUX = (0<<REFS1) | (1<<REFS0) |  (1<<ADLAR) |0x01 ;
  ADCSRA = (1<<ADEN) | (1<<ADSC) | (3<<ADPS0);
  while (ADCSRA & (1<<ADSC));
  CYWM_AddToTXBuffer(ADCH-101);

  //Gyro z: ADC0
  ADMUX = (0<<REFS1) | (1<<REFS0) |  (1<<ADLAR) |0x00 ;
  ADCSRA = (1<<ADEN) | (1<<ADSC) | (3<<ADPS0);
  while (ADCSRA & (1<<ADSC));
  CYWM_AddToTXBuffer(ADCH-111);
}
```

## B.8 Calibration

*Code Segment B.15: Calibration.h*

```
#define NODENUMBER     (unsigned char*)0x10

#define ROUGHx       (unsigned char*)0x20
#define ROUGHy1      (unsigned char*)0x21
#define ROUGHy2      (unsigned char*)0x22


unsigned char RetrieveSensorNumber(void);
void SetSensorNumber(unsigned char SENSOR);
unsigned char RetrieveRoughx(void);
void SetRoughx(unsigned char rough);
unsigned char RetrieveRoughy1(void);
void SetRoughy1(unsigned char rough);
unsigned char RetrieveRoughy2(void);
void SetRoughy2(unsigned char rough);
```

*Code Segment B.16: Calibration.c*

```
#include "calibration.h"
#include "hardware.h"

unsigned char RetrieveSensorNumber(void)
{
  eeprom_busy_wait();
  return eeprom_read_byte(NODENUMBER);
}

void SetSensorNumber(unsigned char SENSOR)
{
  eeprom_busy_wait();
  eeprom_write_byte(NODENUMBER, SENSOR);
}

unsigned char RetrieveRoughx(void)
{
  eeprom_busy_wait();
  return eeprom_read_byte(ROUGHx);
}

void SetRoughx(unsigned char rough)
{
  eeprom_busy_wait();
  eeprom_write_byte(ROUGHx, rough);
}

unsigned char RetrieveRoughy1(void)
{
  eeprom_busy_wait();
  return eeprom_read_byte(ROUGHy1);
}

void SetRoughy1(unsigned char rough)
```

```
{
  eeprom_busy_wait();
  eeprom_write_byte(ROUGHy1, rough);
}

unsigned char RetrieveRoughy2(void)
{
  eeprom_busy_wait();
  return eeprom_read_byte(ROUGHy2);
}

void SetRoughy2(unsigned char rough)
{
  eeprom_busy_wait();
  eeprom_write_byte(ROUGHy2, rough);
}
```

## B.9 RF Transceiver

*Code Segment B.17: CYWUSB693x.h*

```
#define REG_WRITE      0x80
// ------------------------------
// Channel register
// ------------------------------
#define CHANNEL_ADR                                             0x00
#define CHANNEL_RST                                             0x48
#define CHANNEL_MSK                                             0x7F

#define CHANNEL_MAX                                             0x62
#define CHANNEL_MIN                                             0x00
#define CHANNEL_2P498_GHZ                                       0x62
#define CHANNEL_2P4_GHZ                                         0x00

// ------------------------------
// TX Length register
// ------------------------------
#define TX_LENGTH_ADR                                           0x01
#define TX_LENGTH_RST                                           0x00
#define TX_LENGTH_MSK                                           0xFF

// ------------------------------
// TX Control register
// ------------------------------
#define TX_CTRL_ADR                                             0x02
#define TX_CTRL_RST                                             0x03

// TX_CTRL bit masks
#define TX_GO                                                   0x80
#define TX_CLR                                                  0x40

// ------------------------------
// TX Configuration register
// ------------------------------
```

```
#define TX_CFG_ADR                                          0x03
#define TX_CFG_RST                                          0x07

// separate bit field masks
#define TX_DATCODE_LEN_MSK                                  0x20
#define TX_DATMODE_MSK                                      0x18
#define PA_VAL_MSK                                          0x07

// DATCODE_LEN register masks
#define DATCODE_LEN_64                                      0x20
#define DATCODE_LEN_32                                      0x00

// DATMODE register masks
#define DATMODE_1MBPS                                       0x00
#define DATMODE_8DR                                         0x08
#define DATMODE_DDR                                         0x10
#define DATMODE_SDR                                         0x18

// PA_SET register masks
#define PA_N30_DBM                                          0x00
#define PA_N25_DBM                                          0x01
#define PA_N20_DBM                                          0x02
#define PA_N15_DBM                                          0x03
#define PA_N10_DBM                                          0x04
#define PA_N5_DBM                                           0x05
#define PA_0_DBM                                            0x06
#define PA_4_DBM                                            0x07

// ------------------------------
// TX IRQ Status register
// ------------------------------
#define TX_IRQ_STATUS_ADR                                   0x04

// TX_IRQ bit masks
#define XS_IRQ                                              0x80
#define LV_IRQ                                              0x40
#define TXB15_IRQ                                           0x20
#define TXB8_IRQ                                            0x10
#define TXB0_IRQ                                            0x08
#define TXBERR_IRQ                                          0x04
#define TXC_IRQ                                             0x02
#define TXE_IRQ                                             0x01

// ------------------------------
// RX Control register
// ------------------------------
#define RX_CTRL_ADR                                         0x05
#define RX_CTRL_RST                                         0x07

// RX_CTRL bit masks
#define RX_GO                                               0x80

// ------------------------------
// RX Configuration register
// ------------------------------
#define RX_CFG_ADR                                          0x06
#define RX_CFG_RST                                          0x92
```

```
#define AUTO_AGC_EN                                        0x80
#define LNA_EN                                             0x40
#define ATT_EN                                             0x20
#define HI                                                 0x10
#define LO                                                 0x00
#define FASTTURN_EN                                        0x08
#define RXOW_EN                                            0x02
#define VLD_EN                                             0x01

// ------------------------------
// RX IRQ register
// ------------------------------
#define RX_IRQ_STATUS_ADR                                  0x07

// RX_IRQ bit masks
#define RXOW_IRQ                                           0x80
#define SOFDET_IRQ                                         0x40
#define RXB16_IRQ                                          0x20
#define RXB8_IRQ                                           0x10
#define RXB1_IRQ                                           0x08
#define RXBERR_IRQ                                         0x04
#define RXC_IRQ                                            0x02
#define RXE_IRQ                                            0x01

// ------------------------------
// RX Status register
// ------------------------------
#define RX_STATUS_ADR                                      0x08

// single flag bits & multi-bit-field masks
#define RX_ACK                                             0x80
#define RX_PKTERR                                          0x40
#define RX_EOPERR                                          0x20
#define RX_CRC0                                            0x10
#define RX_BAD_CRC                                         0x08
#define RX_DATCODE_LEN                                     0x04
#define RX_DATMODE_MSK                                     0x03

// ------------------------------
// RX Count register
// ------------------------------
#define RX_COUNT_ADR                                       0x09
#define RX_COUNT_RST                                       0x00
#define RX_COUNT_MSK                                       0xFF

// ------------------------------
// RX Length Field register
// ------------------------------
#define RX_LENGTH_ADR                                      0x0A
#define RX_LENGTH_RST                                      0x00
#define RX_LENGTH_MSK                                      0xFF

// ------------------------------
// Power Control register
// ------------------------------
#define PWR_CTRL_ADR                                       0x0B
```

```
#define PWR_CTRL_RST                            0xA0

// single flag bits & multi-bit-field masks
#define PMU_EN                                  0x80
#define LV_IRQ_EN                               0x40
#define PMU_SEN                                 0x20
#define PFET_OFF                                0x10
#define LV_IRQ_TH_MSK                           0x0C
#define PMU_OUTV_MSK                            0x03

// LV_IRQ_TH values
#define LV_IRQ_TH_1P8_V                         0x0C
#define LV_IRQ_TH_2P0_V                         0x08
#define LV_IRQ_TH_2P2_V                         0x04
#define LV_IRQ_TH_PMU_OUTV                      0x00

// PMU_OUTV values
#define PMU_OUTV_2P4                            0x03
#define PMU_OUTV_2P5                            0x02
#define PMU_OUTV_2P6                            0x01
#define PMU_OUTV_2P7                            0x00

// ------------------------------
// Crystal Control register
// ------------------------------
#define XTAL_CTRL_ADR                           0x0C
#define XTAL_CTRL_RST                           0x04

// single flag bits & multi-bit-field masks
#define XOUT_FNC_MSK                            0xC0
#define XS_IRQ_EN                               0x20
#define XOUT_FREQ_MSK                           0x07

// XOUT_FNC values
#define XOUT_FNC_XOUT_FREQ                      0x00
#define XOUT_FNC_PA_N                           0x40
#define XOUT_FNC_RAD_STREAM                     0x80
#define XOUT_FNC_GPIO                           0xC0

// XOUT_FREQ values
#define XOUT_FREQ_12MHZ                         0x00
#define XOUT_FREQ_6MHZ                          0x01
#define XOUT_FREQ_3MHZ                          0x02
#define XOUT_FREQ_1P5MHZ                        0x03
#define XOUT_FREQ_P75MHZ                        0x04

// ------------------------------
// I/O Configuration register
// ------------------------------
#define IO_CFG_ADR                              0x0D
#define IO_CFG_RST                              0x00
#define IO_CFG_MSK                              0xFF

// single flag bits & multi-bit-field masks
#define IRQ_OD                                  0x80
#define IRQ_POL                                 0x40
#define MISO_OD                                 0x20
```

```
#define XOUT_OD                                       0x10
#define PACTL_OD                                      0x08
#define PACTL_GPIO                                    0x04
#define SPI_3_PIN                                     0x02
#define IRQ_GPIO                                      0x01

// ------------------------------
// GPIO Control register
// ------------------------------
#define GPIO_CTRL_ADR                                 0x0E
#define GPIO_CTRL_RST                                 0x00
#define GPIO_CTRL_MSK                                 0xF0

// single flag bits & multi-bit-field masks
#define XOUT_OP                                       0x80
#define MISO_OP                                       0x40
#define PACTL_OP                                      0x20
#define IRQ_OP                                        0x10
#define XOUT_IP                                       0x08
#define MISO_IP                                       0x04
#define PACTL_IP                                      0x02
#define IRQ_IP                                        0x01

// ------------------------------
// Transaction Configuration register
// ------------------------------
#define XACT_CFG_ADR                                  0x0F
#define XACT_CFG_RST                                  0x80

// single flag bits & multi-bit-field masks
#define ACK_EN                                        0x80
#define FRC_END_STATE                                 0x20
#define END_STATE_MSK                                 0x1C
#define ACK_TO_MSK                                    0x03

// END_STATE field values
#define END_STATE_SLEEP                               0x00
#define END_STATE_IDLE                                0x04
#define END_STATE_TXSYNTH                             0x08
#define END_STATE_RXSYNTH                             0x0C
#define END_STATE_RX                                  0x10

// ACK_TO field values
#define ACK_TO_4X                                     0x00
#define ACK_TO_8X                                     0x01
#define ACK_TO_12X                                    0x02
#define ACK_TO_15X                                    0x03

// ------------------------------
// Framing Configuration register
// ------------------------------
#define FRAMING_CFG_ADR                               0x10
#define FRAMING_CFG_RST                               0xA5

// single flag bits & multi-bit-field masks
#define SOP_EN                                        0x80
#define SOP_LEN                                       0x40
```

```
#define LEN_EN                                          0x20
#define SOP_THRESH_MSK                                  0x1F

// ------------------------------
// Data Threshold 32 register
// ------------------------------
#define DATA32_THOLD_ADR                                0x11
#define DAT32_THRESH_RST                                0x04
#define DAT32_THRESH_MSK                                0x0F

// ------------------------------
// Data Threshold 64 register
// ------------------------------
#define DATA64_THOLD_ADR                                0x12
#define DAT64_THRESH_RST                                0x0A
#define DAT64_THRESH_MSK                                0x1F

// ------------------------------
// RSSI register
// ------------------------------
#define RSSI_ADR                                        0x13
#define RSSI_RST                                        0x20

// single flag bits & multi-bit-field masks
#define SOP_RSSI                                        0x80
#define LNA_STATE                                       0x20
#define RSSI_LVL_MSK                                    0x1F

// ------------------------------
// EOP Control register
// ------------------------------
#define EOP_CTRL_ADR                                    0x14
#define EOP_CTRL_RST                                    0xA4

// single flag bits & multi-bit-field masks
#define HINT_EN                                         0x80
#define HINT_EOP_MSK                                    0x70
#define EOP_MSK                                         0x0F

// ------------------------------
// CRC Seed registers
// ------------------------------
#define CRC_SEED_LSB_ADR                                0x15
#define CRC_SEED_MSB_ADR                                0x16
#define CRC_SEED_LSB_RST                                0x00
#define CRC_SEED_MSB_RST                                0x00

// CRC related values
// USB CRC-16
#define CRC_POLY_MSB                                    0x80
#define CRC_POLY_LSB                                    0x05
#define CRC_RESI_MSB                                    0x80
#define CRC_RESI_LSB                                    0x0D

// ------------------------------
// TX CRC Calculated registers
// ------------------------------
```

```
#define TX_CRC_LSB_ADR                                    0x17
#define TX_CRC_MSB_ADR                                    0x18

// ------------------------------
// RX CRC Field registers
// ------------------------------
#define RX_CRC_LSB_ADR                                    0x19
#define RX_CRC_MSB_ADR                                    0x1A
#define RX_CRC_LSB_RST                                    0xFF
#define RX_CRC_MSB_RST                                    0xFF

// ------------------------------
// Synth Offset registers
// ------------------------------
#define TX_OFFSET_LSB_ADR                                 0x1B
#define TX_OFFSET_MSB_ADR                                 0x1C
#define TX_OFFSET_LSB_RST                                 0x00
#define TX_OFFSET_MSB_RST                                 0x00

// single flag bits & multi-bit-field masks
#define STRIM_MSB_MSK                                     0x0F
#define STRIM_LSB_MSK                                     0xFF

// ------------------------------
// Mode Override register
// ------------------------------
#define MODE_OVERRIDE_ADR                                 0x1D
#define MODE_OVERRIDE_RST                                 0x00

#define FRC_AWAKE                                         0x03
#define FRC_AWAKE_OFF_1                                   0x01
#define FRC_AWAKE_OFF_2                                   0x00

// single flag bits & multi-bit-field masks
#define DIS_AUTO_SEN                                      0x80
#define SEN_TXRXB                                         0x40
#define FRC_SEN                                           0x20
#define FRC_AWAKE_MSK                                     0x18
#define MODE_OVRD_FRC_AWAKE                               0x18
#define MODE_OVRD_FRC_AWAKE_OFF_1                         0x08
#define MODE_OVRD_FRC_AWAKE_OFF_2                         0x00
#define RST                                               0x01
#define FRC_PA                                            0x02

// ------------------------------
// RX Override register
// ------------------------------
#define RX_OVERRIDE_ADR                                   0x1E
#define RX_OVERRIDE_RST                                   0x00

// single flag bits & multi-bit-field masks
#define ACK_RX                                            0x80
#define EXTEND_RX_TX                                      0x40
#define MAN_RXACK                                         0x20
#define FRC_RXDR                                          0x10
#define DIS_CRC0                                          0x08
#define DIS_RXCRC                                         0x04
```

```
#define ACE                                   0x02

// ------------------------------
// TX Override register
// ------------------------------
#define TX_OVERRIDE_ADR                       0x1F
#define TX_OVERRIDE_RST                       0x00

// single flag bits & multi-bit-field masks
#define ACK_TX_SEN                            0x80
#define FRC_PREAMBLE                          0x40
#define DIS_TX_RETRANS                        0x20
#define MAN_TXACK                             0x10
#define OVRRD_ACK                             0x08
#define DIS_TXCRC                             0x04
#define CO                                    0x02
#define TXINV                                 0x01

//-------------------------------------------
//      File Function Detail
//-------------------------------------------

// ------------------------------
// TX Buffer - 16 bytes
// ------------------------------
#define TX_BUFFER_ADR                         0x20

// ------------------------------
// RX Buffer - 16 bytes
// ------------------------------
#define RX_BUFFER_ADR                         0x21

// ------------------------------
// Framing Code - 8 bytes
// ------------------------------
#define SOP_CODE_ADR                          0x22

// CODESTORE_REG_SOF_RST
#define CODESTORE_BYTE7_SOF_RST               0x17
#define CODESTORE_BYTE6_SOF_RST               0xFF
#define CODESTORE_BYTE5_SOF_RST               0x9E
#define CODESTORE_BYTE4_SOF_RST               0x21
#define CODESTORE_BYTE3_SOF_RST               0x36
#define CODESTORE_BYTE2_SOF_RST               0x90
#define CODESTORE_BYTE1_SOF_RST               0xC7
#define CODESTORE_BYTE0_SOF_RST               0x82

// ------------------------------
// Data Code - 16 bytes
// ------------------------------
#define DATA_CODE_ADR                         0x23

// CODESTORE_REG_DCODE0_RST
#define CODESTORE_BYTE7_DCODE0_RST            0x01
#define CODESTORE_BYTE6_DCODE0_RST            0x2B
#define CODESTORE_BYTE5_DCODE0_RST            0xF1
#define CODESTORE_BYTE4_DCODE0_RST            0xDB
```

```
#define CODESTORE_BYTE3_DCODE0_RST                              0x01
#define CODESTORE_BYTE2_DCODE0_RST                              0x32
#define CODESTORE_BYTE1_DCODE0_RST                              0xBE
#define CODESTORE_BYTE0_DCODE0_RST                              0x6F

// CODESTORE_REG_DCODE1_RST
#define CODESTORE_BYTE7_DCODE1_RST                              0x02
#define CODESTORE_BYTE6_DCODE1_RST                              0xF9
#define CODESTORE_BYTE5_DCODE1_RST                              0x93
#define CODESTORE_BYTE4_DCODE1_RST                              0x97
#define CODESTORE_BYTE3_DCODE1_RST                              0x02
#define CODESTORE_BYTE2_DCODE1_RST                              0xFA
#define CODESTORE_BYTE1_DCODE1_RST                              0x5C
#define CODESTORE_BYTE0_DCODE1_RST                              0xE3

// ------------------------------
// Preamble - 3 bytes
// ------------------------------
#define PREAMBLE_ADR                                            0x24

#define PREAMBLE_CODE_MSB_RST                                   0x33
#define PREAMBLE_CODE_LSB_RST                                   0x33
#define PREAMBLE_LEN_RST                                        0x02

// ------------------------------
// Laser Fuses - 8 bytes (2 hidden)
// ------------------------------
#define MFG_ID_ADR                                              0x25

// ------------------------------
// XTAL Startup Delay
// ------------------------------
#define XTAL_CFG_ADR                                            0x26
#define XTAL_CFG_RST                                            0x00

// ------------------------------
// Clock Override
// ------------------------------
#define CLK_OVERRIDE_ADR                                        0x27
#define CLK_OVERRIDE_RST                                        0x00

#define RXF                                                     0x02

// ------------------------------
// Clock Enable
// ------------------------------
#define CLK_EN_ADR                                              0x28
#define CLK_EN_RST                                              0x00

#define RXF                                                     0x02

// ------------------------------
// Receiver Abort
// ------------------------------
#define RX_ABORT_ADR                                            0x29
#define RX_ABORT_RST                                            0x00
```

```
#define ABORT_EN                                            0x20

// ------------------------------
// Auto Calibration Time
// ------------------------------
#define AUTO_CAL_TIME_ADR                                   0x32
#define AUTO_CAL_TIME_RST                                   0x0C

#define AUTO_CAL_TIME_MAX                                   0x3C

// ------------------------------
// Auto Calibration Offset
// ------------------------------
#define AUTO_CAL_OFFSET_ADR                                 0x35
#define AUTO_CAL_OFFSET_RST                                 0x00

#define AUTO_CAL_OFFSET_MINUS_4                             0x14

//-------------------------------
// Channels
//-------------------------------
#define CHANNEL0                    0
#define CHANNEL1                    6
#define CHANNEL2                    12
#define CHANNEL3                    18
#define CHANNEL4                    24
#define CHANNEL5                    30
#define CHANNEL6                    36
#define CHANNEL7                    42
#define CHANNEL8                    48
#define CHANNEL9                    54

#define CMST
//#define IPEM

#ifdef IPEM
#define MASTERCHANNEL    CHANNEL1
#define SLAVECHANNEL     CHANNEL2
#endif
#ifdef CMST
#define MASTERCHANNEL    CHANNEL5
#define SLAVECHANNEL     CHANNEL6
#endif

//Number of bytes to send per package
#define PACKETSIZE    11

//Functions in CYWUSB693x.c
void CYWM_WriteReg(unsigned char which, unsigned char data);
unsigned char CYWM_ReadReg(unsigned char which);
void CYWM_Init(void);
void CYWM_SetPacketSizeSlave(void);
void CYWM_SetPacketSizeMaster(void);
void CYWM_SetChannel(unsigned char channel);
void CYWM_AddToTXBuffer(unsigned char data);
void CYWM_ClearTXBuffer(void);
void CYWM_SendPacket(void);
void CYWM_Receive(void);
void CYWM_EndReceive(void);
void send(void);
```

*Code Segment B.18: CYWUSB693x.c*

```
#include "CYWUSB693x.h"
#include "hardware.h"
#include "SPI.h"

void CYWM_WriteReg(unsigned char which, unsigned char data)
{
  low(CYWM_nSS_PORT, CYWM_nSS);
  SPI_Write(REG_WRITE | which);
  SPI_Write(data);
  high(CYWM_nSS_PORT, CYWM_nSS);
}

unsigned char CYWM_ReadReg(unsigned char which)
{
  low(CYWM_nSS_PORT, CYWM_nSS);
  SPI_Write(which);
  SPI_Write(which);
  high(CYWM_nSS_PORT, CYWM_nSS);
  return SPDR;
}

void CYWM_Init(void)
{
  //Soft reset
  CYWM_WriteReg( MODE_OVERRIDE_ADR, RST);

  //init of CYWUSB

  //Necessary writes for good operation,
  //set AUTO calibration registers to std values
  CYWM_WriteReg( AUTO_CAL_TIME_ADR, AUTO_CAL_TIME_MAX);
  CYWM_WriteReg( AUTO_CAL_OFFSET_ADR, AUTO_CAL_OFFSET_MINUS_4);

  //Disable Power Management Unit
  CYWM_WriteReg(PWR_CTRL_ADR, PMU_SEN);

  //Start up the crystal
  CYWM_WriteReg(XTAL_CFG_ADR, 0x04);
  CYWM_WriteReg(XTAL_CTRL_ADR, XOUT_FNC_GPIO);

  //Set IRQ pin to active HIGH
  CYWM_WriteReg(IO_CFG_ADR, IRQ_POL);

  //Enable LNA, FastTurn and disable RX overwrite and Auto AGC
  CYWM_WriteReg(RX_CFG_ADR,((RX_CFG_RST | FASTTURN_EN | LNA_EN)
                            & ~( HI | AUTO_AGC_EN | RXOW_EN)));

  //Set DATA Mode
  CYWM_WriteReg(TX_CFG_ADR, TX_CFG_RST | DATMODE_8DR);

  //Enable SOP and set threshhold
  CYWM_WriteReg(FRAMING_CFG_ADR, SOP_EN | 0x05);

  //length of transmit buffer is PACKETSIZE byte
  CYWM_WriteReg(TX_LENGTH_ADR, PACKETSIZE);

  //Set preamble address
  low(CYWM_nSS_PORT, CYWM_nSS);
  SPI_Write(REG_WRITE | PREAMBLE_ADR);
  SPI_Write(0x04);
  SPI_Write(0x33);
  SPI_Write(0x33);
  high(CYWM_nSS_PORT, CYWM_nSS);

  //Set end state to idle
  CYWM_WriteReg(XACT_CFG_ADR, END_STATE_IDLE);

  //Synthesiser offset
  CYWM_WriteReg(TX_OFFSET_LSB_ADR, 0x55);
  CYWM_WriteReg(TX_OFFSET_MSB_ADR, 0x05);
  return;
}

void CYWM_SetChannel(unsigned char channel)
{
  CYWM_WriteReg(CHANNEL_ADR, channel);
}

void CYWM_AddToTXBuffer(unsigned char data)
{
  CYWM_WriteReg(TX_BUFFER_ADR, data);
}

void CYWM_ClearTXBuffer(void)
{
  CYWM_WriteReg(TX_CTRL_ADR, TX_CLR);
}

void CYWM_SendPacket(void)
{
  // Start transmit
  CYWM_WriteReg(TX_CTRL_ADR, TX_CTRL_RST | TX_GO);
  while (!(CYWM_ReadReg(TX_IRQ_STATUS_ADR) & TXC_IRQ)) {}
  CYWM_ClearTXBuffer();
}

void CYWM_Receive(void)
{
  CYWM_WriteReg(RX_CTRL_ADR, (RX_GO | RXC_IRQ));
}

void CYWM_EndReceive(void)
{
  CYWM_WriteReg(XACT_CFG_ADR,
            CYWM_ReadReg(XACT_CFG_ADR) | FRC_END_STATE);
  while(CYWM_ReadReg( XACT_CFG_ADR) &  FRC_END_STATE) {};
}
```

```
//Send data to RF chip
void send(void)
{
  CYWM_SendPacket();
}
```

## B.10 Main

*Code Segment B.19: Auto.c*

```
#include "CYWUSB693x.h"
#include "hardware.h"
#include "SPI.h"
#include "I2C.h"
#include "mag.h"
#include "gyr.h"
#include "acc.h"
#include "fixedPoint.h"
#include "calibration.h"

#define wdr() __asm__ __volatile__ ("wdr" ::)

unsigned char TIMESLOT;
unsigned char SENSOR;

unsigned char Slaves[10];

unsigned char counter;
unsigned char LEDcounter;
unsigned short master_counter;

unsigned char waitToSend;
unsigned char waitToReceive;
unsigned char waitForMaster;

unsigned char wakeup;

unsigned char startUp;
unsigned char isMaster;
unsigned char checkingMaster;
unsigned char scanningSlaves;


void StartUpDelay(void)
{
  //Set boolean
  startUp = 1;

  //Set up timer
  Timer_Start_Up(SENSOR);

  //Wait for timer interrupt
  set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  sleep_mode();

  //Reset Timer
  Reset_Timer();
}

void CheckMaster(void)
{

  //Set boolean
  checkingMaster = 1;

  //Set up timer
  Timer_Master_Check();

  //Enable pin interrupt
  Init_Pin_interrupt();
  EIMSK = 0x01;

  //Set up RF
  CYWM_SetChannel(MASTERCHANNEL);
  CYWM_Receive();

  //Wait for completion
  set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  sleep_mode();

  //Reset Timer
  Reset_Timer();
}

void SlaveScan(void)
{
  //Clear slavestable
  for (unsigned char i=0; i<10; i++)
  {
    Slaves[i] = 0;
  }

  //Set boolean
  scanningSlaves = 1;

  //Set up timer
  Timer_Slave_Scan();

  //Enable pin interrupt
  Init_Pin_interrupt();
  EIMSK = 0x01;

  //Set up RF
  CYWM_SetChannel(SLAVECHANNEL);

  //Wait for completion
  do
  {
    CYWM_Receive();
    set_sleep_mode(SLEEP_MODE_IDLE);
    sleep_enable();
    sleep_mode();
  }while (scanningSlaves == 1);

  //Assign timeslot
  for (unsigned char i=0; i<10; i++)
  {
    if (Slaves[i] == 0){
      TIMESLOT = i+1;
      i = 10;
    }
  }

  //Reset to master channel
  CYWM_SetChannel(MASTERCHANNEL);

  //Reset Timer
  Reset_Timer();
}

void MasterConflictCheck(void)
{
  //Set boolean
  checkingMaster = 1;

  //Enable interrupts
  EIMSK = 0x01;
  TIMSK1 = 0x20;

  //Set up RF
  CYWM_Receive();

  //Wait for completion
  set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  sleep_mode();

  // Initialize node as slave if another master is detected
  if(isMaster==0)
    {
      wdr();
    SlaveScan();
    Slave_Timer_Init(TIMESLOT);
    }
  else
  {
    TIMSK1 = 0x20;
  }
}

int main()
{

  //Setup Watchdog timer
  //Reset
  wdr();
```

```
    /* Start timed equence */
    WDTCSR |= (1<<WDCE) | (1<<WDE);
    /* Set new prescaler(time-out) value = 16K cycles (~125 ms) */
    WDTCSR = (1<<WDE) | (1<<WDP1) | (1<<WDP0);

    // Initialize LedPort
    initLEDPort();

    // Disable interrupts
    cli();

    #ifdef FIRST
    //EEPROM Write Sensor number
    SetSensorNumber(8);
    #endif

    //EEPROM Read Sensor number
    SENSOR = RetrieveSensorNumber();
    TIMESLOT = 1;

    // Initialize ports
    Port_Init();

    // Set magnetometer to active
    high(PORTD, MAGRESET);

    // Reset RF chip
    high(PORTD, CYWM_nRESET);
    low(PORTD, CYWM_nRESET);

    // Initialize SPI interface
    Spi_Init();

    // Place interrupt vector at beginning of
    // the Boot Loader section of the Flash
    MCUCR = _BV(ISC01) | _BV(ISC00);

    // Enable interrupts
    sei();

    // Initialize Cypress RF Chip
    CYWM_Init();

    // Initialize sensors and ADC for gyros
    init_Acc();
    //ADC_Init();
    initMag(SENSOR);

    // Set the RF channel and clear the sendbuffer
    CYWM_SetChannel(MASTERCHANNEL);
    CYWM_WriteReg(CRC_SEED_LSB_ADR,0x02);
    CYWM_ClearTXBuffer(); //Clear  buffer

    // Init booleans
    wakeup = 0;
    waitToSend = 0;
    waitToReceive = 0;
```

```
waitForMaster = 0;
isMaster = 0;
checkingMaster = 0;
scanningSlaves = 0;
startUp = 0;
counter = 0;
LEDcounter = 0;
master_counter = 0;

//Perform startup delay number of times equal to SENSOR number
for (unsigned char i=0; i<=SENSOR; i++){
  wdr();
  StartUpDelay();
}

//Check for master
wdr();
CheckMaster();

// Initialize node as master or slave
if (isMaster == 1){
  Master_Timer_Init();
}
else{
  wdr();
  SlaveScan();
  Slave_Timer_Init(TIMESLOT);
}

//Start infinite loop
for(;;)
{

  //Reset watchdogtimer
  wdr();

  if (TIMESLOT < 5){

    if (isMaster == 1)
    {
      //Add counter to sendbuffer
      CYWM_AddToTXBuffer(counter);
    }
    else
    {
      //Add timeslot to sendbuffer
      CYWM_AddToTXBuffer(TIMESLOT);
    }

    //Add sensornumber to sendbuffer
    CYWM_AddToTXBuffer(SENSOR);

    //Read sensor outputs
    //readGyr();
    readAcc();
    readMag();
  }
```

```
if (isMaster == 0)
{
  //Set boolean to wait for master receive
  waitToReceive = 1;
  do{
    if (waitToReceive == 1){
      TIMSK1 = 0x04;
    }
    if (waitForMaster == 1){
      CYWM_Receive();
      EIMSK = 0x01;
    }
    set_sleep_mode(SLEEP_MODE_IDLE);
    sleep_enable();
    sleep_mode();
  } while (wakeup==0);

  //Clear boolean
  wakeup = 0;

  if (TIMESLOT > 4){

    //Add timeslot to sendbuffer
    CYWM_AddToTXBuffer(TIMESLOT);

    //Add sensornumber to sendbuffer
    CYWM_AddToTXBuffer(SENSOR);

    //Read sensor outputs
    //readGyr();
    readAcc();
    readMag();
  }
}

//Wait for timeslot
set_sleep_mode(SLEEP_MODE_IDLE);
sleep_enable();
sleep_mode();

//Send data over RF link
send();

//Increment Counter
counter++;

//If node is master
if (isMaster == 1)
{

  //Blink LED
      if (!((counter & 0x10) == 0))
      {
        toggleLED();
      }
      //Decrement master conflict check counter
      master_counter++;

      //If zero is reached: check for conflicts and reset counter
      if (master_counter == 1000+SENSOR)
      {
        master_counter = 0;
        MasterConflictCheck();
      }
    }

    //If node is slave
    else
    {
      //Blink LED number of times equal to TIMESLOT
      if (((counter >> 4) < TIMESLOT) & ((counter & 0x08) == 0x00))
      {
        setLED();
      }
      else
      {
        clearLED();
      }
    }
  }
  return 0;
}

ISR(TIMER1_CAPT_vect){
  sleep_disable();
  if (startUp == 1){
    startUp = 0;
    TIMSK1 = 0x00;
    TCNT1 = 0;
  }
  if (checkingMaster == 1){
    checkingMaster = 0;
    CYWM_EndReceive();
    isMaster = 1;
    TIMESLOT = 0;
    EIMSK = 0x00;
    TIMSK1 = 0x00;
    TCNT1 = 0;
  }
  if (scanningSlaves == 1){
    scanningSlaves = 0;
    CYWM_EndReceive();
    EIMSK = 0x00;
    TIMSK1 = 0x00;
    TCNT1 = 0;
  }
}

ISR(TIMER1_COMPA_vect){
  if (TCNT1 >= OCR1A)
  {
    sleep_disable();
    waitToSend = 0;
    TIMSK1 = 0x00;
  }
}

ISR(TIMER1_COMPB_vect){
  if (TCNT1 >= OCR1B)
  {
    sleep_disable();
    waitToReceive = 0;
    waitForMaster = 1;
    TIMSK1 = 0x00;
    CYWM_SetChannel(MASTERCHANNEL);
  }
}

ISR(INT0_vect){
  if (CYWM_ReadReg(RX_IRQ_STATUS_ADR) & RXC_IRQ)
  {
    sleep_disable();
    if (checkingMaster == 1){
      checkingMaster = 0;
      CYWM_EndReceive();
      TIMSK1 = 0x00;
      isMaster = 0;
      for (unsigned char i=0; i<PACKETSIZE; i++){
        CYWM_ReadReg(RX_BUFFER_ADR);
      }
    }
    if (scanningSlaves == 1){
      unsigned char temp = CYWM_ReadReg(RX_BUFFER_ADR);
      if (temp < 10)
      {
        Slaves[temp-1] = 1;
      }
      for (unsigned char i=1; i<PACKETSIZE; i++){
        CYWM_ReadReg(RX_BUFFER_ADR);
      }
    }
    if (waitForMaster == 1)
    {
      CYWM_EndReceive();
      waitForMaster = 0;
      waitToSend = 1;
      EIMSK = 0x00;
      TCNT1 = 0;
      TIMSK1 = 0x02;
      CYWM_SetChannel(SLAVECHANNEL);
      wakeup = 1;
    }
  }
  return;
}
```

# C
# Third Generation Firmware

This appendix contains the firmware of the third generation motion tracking sensor nodes.

## C.1 Hardware

*Code Segment C.1: Hardware.h*

```
#ifndef HARDWARE_H
#define HARDWARE_H

#include "msp430x24x.h"

// measurement interval in clockpulses on watch crystal
#define MEASUREINTERVAL 327

// waiting time before turning on receiver
#define receiveWait MEASUREINTERVAL-30

// waiting time for master detection
#define MASTERCHECKTIME 1635

// scanning time for slave detection
#define SLAVESCANTIME    2000

// time in between slots
#define SLOTSIZE   28

// Number of slaves per channel
#define SLAVES   9
```

```
// Booleans for interrupts
extern unsigned char waitToSend;
extern unsigned char waitToReceive;

//Sensor node info
extern unsigned char timeSlot;
extern unsigned char Sensor;

// Wait times
extern int sendWait;

//enum P1_PINS {
//              };

//enum P2_PINS {
//              };

enum P3_PINS {I2C_SDA    = 1,  // USCI B0 = I2C Acc and Mag
              I2C_SCL    = 2,  // USCI B0 = I2C Acc and Mag
              NRF_SIMO   = 6,  // USCI A1 = SPI nRF
              NRF_SOMI   = 7   // USCI A1 = SPI nRF
              };

enum P4_PINS {P4_CLK1    = 0,  // Shorted to NRF CLK1 for routing
              NRF_DR1 = 5,     // nRF Data Ready 1
              NRF_CS  = 6,     // nRF Chip Select
              };

enum P5_PINS {NRF_CLK   = 0,   // USCI A1 = SPI nRF
              NRF_SIMO2= 1,    // USCI B1 = SPI2 nRF
              NRF_SOMI2= 2,    // USCI B1 = SPI2 nRF
              NRF_CLK2 = 3,    // USCI B1 = SPI2 nRFe
              NRF_DR2   = 4,   // nRF Data Ready 2
              NRF_PWRUP= 5,    // nRF Power Up pin
              NRF_CE    = 6,   // nRF Chip Enable
              LED       = 7    // LED output
              };

enum P6_PINS {MPIO      = 0    // Multipurpose input output pin
              };

#define LED_ON      P5OUT |=  (1<<LED)
#define LED_OFF     P5OUT &= ~(1<<LED)
#define LED_TOGGLE P5OUT ^=  (1<<LED)

//RF-chip interfaces

//SPI interfaces RF-chip

#define SPIIFG     UC1IFG
#define SPIRXIFG   UCA1RXIFG
#define SPITXIFG   UCA1TXIFG
#define SPIRXBUF   UCA1RXBUF
#define SPITXBUF   UCA1TXBUF

#define SPI2IFG    UC1IFG
```

```
#define SPI2RXIFG UCB1RXIFG
#define SPI2TXIFG UCB1TXIFG
#define SPI2RXBUF UCB1RXBUF
#define SPI2TXBUF UCB1TXBUF

#define SPI_PSEL          P3SEL
#define SPI_PDIR          P3DIR

//Control pins

#define CS_POUT     P4OUT
#define PWRUP_POUT  P5OUT
#define CE_POUT     P5OUT

// CPU clock frequency in MHz (used for delays)
#define CLOCKFREQUENCY 16

// macro for delay in microseconds
// use with constants only (evaluated at compile time)
#define DELAYU(us){
  int t = (us*CLOCKFREQUENCY)/15;
  delay(t);
}
#define DELAYM(ms){
  int mscnt;
  for (mscnt=0; mscnt<ms; mscnt++)
    DELAYU(1000);
}

//Functions in hardware.c
void Clock_init(void);
void Port_init(void);
void Com_init(void);
void Calculate_sendWait(unsigned char channel);
void Setup_Counter_Startup(void);
void Setup_Counter_Master_Check(void);
void Setup_Counter_Slave_Scan(void);
void Setup_Counter_Slave_Check(void);
void Setup_Counter_Master(void);
void Setup_Counter_Slave(void);
void delay(unsigned int d);

#endif
```

*Code Segment C.2: Hardware.c*

```
#include "hardware.h"

// Booleans for interrupts
unsigned char waitToSend = 0;
unsigned char waitToReceive = 0;

// Node info
unsigned char timeSlot = 1;
unsigned char Sensor;

// Wait times
```

```
int sendWait = 0;

void Clock_init(void)
{
  #if CLOCKFREQUENCY == 1

    //Select 1 MHz clock
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;
  #endif
  #if CLOCKFREQUENCY == 8

    //Select 8 MHz clock
    BCSCTL1 = CALBC1_8MHZ;
    DCOCTL = CALDCO_8MHZ;
  #endif
  #if CLOCKFREQUENCY == 12

    //Select 8 MHz clock
    BCSCTL1 = CALBC1_12MHZ;
    DCOCTL = CALDCO_12MHZ;
  #endif
  #if CLOCKFREQUENCY == 16

    //Select 8 MHz clock
    BCSCTL1 = CALBC1_16MHZ;
    DCOCTL = CALDCO_16MHZ;
  #endif
}

void Port_init(void)
{
  // All not connected I/O's should be set as
  // OUTPUT for minimal current consumption

  //Port1: Not connected
  P1OUT = 0x00;
  P1DIR = 0xFF;
  P1REN = 0xFF;
  P1SEL = 0x00;

  //Port2: Not connected
  P2OUT = 0x00;
  P2DIR = 0xFF;
  P2REN = 0xFF;
  P2SEL = 0x00;

  //Port3: Digital comm: I2C and part SPI NRF interface 1
  P3OUT = (1<<I2C_SDA) | (1<<I2C_SCL);
  P3DIR = (1<<I2C_SDA) | (1<<I2C_SCL) | (1<<NRF_SOMI);
  P3DIR = ~P3DIR;
  P3REN = (1<<I2C_SDA) | (1<<I2C_SCL)
                | (1<<NRF_SIMO) | (1<<NRF_SOMI);
  P3REN = ~P3REN;
  P3SEL = (1<<I2C_SDA) | (1<<I2C_SCL)
                | (1<<NRF_SIMO) | (1<<NRF_SOMI);
  //Port4: NRF control signals
  P4OUT = 0x00;
  P4DIR = ~((1<<P4_CLK1) | (1<<NRF_DR1));
  P4REN = ~((1<<P4_CLK1) | (1<<NRF_DR1) | (1<<NRF_CS));
  P4SEL = 1<<NRF_DR1;

  //Port5: Digital communication second part SPI NRF interface 1
  //and SPI NRF interface 2 + LED and control signals
  P5OUT = 0x00;
  P5DIR = ~((1<<NRF_DR2) | (1<<NRF_SOMI2));
  P5REN = 0x00;
  P5SEL = (1<<NRF_CLK) | (1<<NRF_SOMI2) | (1<<NRF_CLK2);

  //Port6: MPIO
  P6OUT = 0x00;
  P6DIR = 0xFF;
  P6REN = 0xFF;
  P6SEL = 0x00;
}

void Com_init(void)
{
  //SPI interface 1: USCI A1
  UCA1CTL1 |= UCSWRST;
  UCA1CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC;
  UCA1CTL1 |= UCSWRST + UCSSEL_2;

  #if CLOCKFREQUENCY == 1

    // Clock devider = 1 => 1MHz
    UCA1BR0 = 1;
  #endif
  #if CLOCKFREQUENCY == 8

    // Clock devider = 8 => 1MHz
   UCA1BR0 = 8;
  #endif
  #if CLOCKFREQUENCY == 12

    // Clock devider = 12 => 1MHz
   UCA1BR0 = 12;
  #endif
  #if CLOCKFREQUENCY == 16

    // Clock devider = 16 => 1MHz
   UCA1BR0 = 16;
  #endif
  UCA1BR1 = 0;

  // **Initialize USCI state machine**
  UCA1CTL1 &= ~UCSWRST;

  //SPI interface 2: USCI B1
  UCB1CTL1 |= UCSWRST;
  UCB1CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC;
  UCB1CTL1 |= UCSWRST + UCSSEL_2;
  #if CLOCKFREQUENCY == 1

    // Clock devider = 1 => 1MHz
    UCB1BR0 = 1;
  #endif
  #if CLOCKFREQUENCY == 8

    // Clock devider = 8 => 1MHz
    UCB1BR0 = 8;
  #endif
  #if CLOCKFREQUENCY == 12

    // Clock devider = 12 => 1MHz
    UCB1BR0 = 12;
  #endif
  #if CLOCKFREQUENCY == 16

    // Clock devider = 16 => 1MHz
    UCB1BR0 = 16;
  #endif
  UCB1BR1 = 0;

  // **Initialize USCI state machine**
  UCB1CTL1 &= ~UCSWRST;

  //I2C: USCI B0
  UCB0CTL1 |= UCSWRST;
  UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;
  UCB0CTL1 = UCSSEL_2 + UCSWRST;

  #if CLOCKFREQUENCY == 1

    // fSCL = SMCLK/3 = ~333kHz
   UCB0BR0 = 3;
  #endif
  #if CLOCKFREQUENCY == 8

   // fSCL = SMCLK/18 = ~400kHz
   UCB0BR0 = 18;
  #endif
  #if CLOCKFREQUENCY == 12

   // fSCL = SMCLK/27 = ~400kHz
   UCB0BR0 = 27;
  #endif
  #if CLOCKFREQUENCY == 16

   // fSCL = SMCLK/36 = ~400kHz
   UCB0BR0 = 36;
  #endif

  UCB0BR1 = 0;
  UCB0CTL1 &= ~UCSWRST;          //Clear SW reset, resume operation
  IE2 |= UCB0TXIE | UCB0RXIE;   //Enable TX/RX interrupt
}

void Calculate_sendWait(unsigned char channel)
```

```
{
  if(channel == 1){
    sendWait = (timeSlot-1)*SLOTSIZE;
  }
  else
  {
    sendWait = (timeSlot-(SLAVES+1))*SLOTSIZE;
  }
}

void Setup_Counter_Startup()
{

  //Timer A: interrupt depends on node number
  TACCTL0 = CCIE;                // TA0CCR0 interrupt enable
  TACCR0 = 3000;                 // interrupt for startup delay
  TACTL = TASSEL_1 + MC_1;       // ACLK, up mode
}

void Setup_Counter_Master_Check(void)
{

  //Timer A: fixed interrupt
  TACCTL0 = CCIE;                 // TA0CCR0 interrupt enable
  TACCR0 = MASTERCHECKTIME;       // interrupt time
  TACTL = TASSEL_1 + MC_1;        // ACLK, up mode

  //Timer B: DR1 Interrupt generation
  TBCCTL5 = CM_1 + CCIS_0 + CAP + CCIE;
  TBCTL = TBSSEL_2 + MC_2;
}

void Setup_Counter_Slave_Scan(void)
{

  //Timer A: fixed interrupt
  TACCTL0 = CCIE;                 // TA0CCR0 interrupt enable
  TACCR0 = SLAVESCANTIME;         // interrupt time
  TACCTL1 = 0;
  TACCTL2 = 0;
  TACTL = TASSEL_1 + MC_1;        // ACLK, up mode

  //Timer B: DR1 Interrupt generation
  TBCCTL5 = CM_1 + CCIS_0 + CAP + CCIE;
  TBCTL = TBSSEL_2 + MC_2;
}

void Setup_Counter_Slave_Check(void)
{

  // The function assumes the timer has been setup
  // as in Setup_Counter_Slave. It only sets the
  // sendWait value higher to allow its slot to pass.
  TACCTL1 = 0;
  TACCR1 = sendWait + 2*SLOTSIZE;
}
```

```
void Setup_Counter_Master(void)
{

  //Timer A: 100 Hz reset based on watch crystal:
  TACCTL0 = CCIE;                  // TA0CCR0 interrupt enable
  TACCR0 = MEASUREINTERVAL;
  TACCTL1 = 0;                     // TA0CCR1 interrupt disable
  TACCR1 = sendWait;
  TACCTL2 = 0;                     // TA0CCR2 interrupt disable
  TACCR2 = receiveWait;
  TACTL = TASSEL_1 + MC_1;         // ACLK, up mode

  //Timer B: DR1 Interrupt generation
  TBCCTL5 = CM_1 + CCIS_0 + CAP + CCIE;
  TBCTL = TBSSEL_2 + MC_2;
}

void Setup_Counter_Slave(void)
{

  //Timer A: 100 Hz reset based on watch crystal:
  TACCTL0 = 0;                     // TA0CCR0 interrupt disable
  TACCR0 = MEASUREINTERVAL;
  TACCTL1 = CCIE;                  // TA0CCR1 interrupt enable
  TACCR1 = sendWait;
  TACCTL2 = CCIE;                  // TA0CCR2 interrupt enable
  TACCR2 = receiveWait;
  TACTL = TASSEL_1 + MC_1;         // ACLK, up mode

  //Timer B: DR1 Interrupt generation
  TBCCTL5 = CM_1 + CCIS_0 + CAP + CCIE;
  TBCTL = TBSSEL_2 + MC_2;
}

//Delay function.
void delay(unsigned int d) {
  unsigned i;
  for (i = 0; i<d; i++) {
          __no_operation();
          __no_operation();
  }
}
```

## C.2 I$^2$C

*Code Segment C.3: I2C.h*

```
//Variables

extern volatile unsigned char RxBuffer[10];

//Functions

void Set_I2CAddress(unsigned char Address);
void Send_I2C(unsigned char TxData[], unsigned char BytesToSend);
void Receive_I2C(unsigned char BytesToReceive);
```

*Code Segment C.4: I2C.c*

```
#include "hardware.h"
#include "I2C.h"

// I2C TX variables
unsigned char *PTxData;
unsigned char TXByteCtr;

// I2C RX variables
unsigned char *PRxData;
unsigned char RXByteCtr;
volatile unsigned char RxBuffer[10];

void Set_I2CAddress(unsigned char Address)
{
  UCB0CTL1 |= UCSWRST;              // Enable SW reset
  UCB0I2CSA = Address;              // Slave Address
  UCB0CTL1 &= ~UCSWRST;             // Clear reset, resume operation
  IE2 |= UCB0TXIE | UCB0RXIE;       // Enable TX/RX interrupt
}

void Send_I2C(unsigned char TxData[], unsigned char BytesToSend)
{
  PTxData = (unsigned char *)TxData;  // TX array start address
  TXByteCtr = BytesToSend;            // Load TX byte counter
  while (UCB0CTL1 & UCTXSTP);         // Ensure stop condition
  UCB0CTL1 |= UCTR | UCTXSTT;         // TX and start condition
  __low_power_mode_0();               // Enter LPM0 w/ interrupts
                                      // Remain in LPM0 until
                                      // all data is TX'd
}

void Receive_I2C(unsigned char BytesToReceive)
{
  PRxData = (unsigned char *)RxBuffer; // Start of RX buffer
  RXByteCtr = BytesToReceive;          // Load RX byte counter
  while (UCB0CTL1 & UCTXSTP);          // Ensure stop condition
  UCB0CTL1 &= ~UCTR;                   // Set to receiver mode
  UCB0CTL1 |= UCTXSTT;                 // I2C start condition
  __low_power_mode_0();                // Enter LPM0 w/ interrupts
                                       // Remain in LPM0 until
                                       // all data is RX'd
}

// Transmitting and receiving I2C data
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
  if (UCB0CTL1 & UCTR)
  {
    if (TXByteCtr)                     // Check TX byte counter
    {
      UCB0TXBUF = *PTxData++;          // Load TX buffer
      TXByteCtr--;                          // Decrement TX byte counter
    }
    else
    {
      UCB0CTL1 |= UCTXSTP;                  // I2C stop condition
      IFG2 &= ~UCB0TXIFG;                   // Clear USCI_B0 TX int flag
      __low_power_mode_off_on_exit();       // Exit LPM0
    }
  }
  else
  {
    RXByteCtr--;                            // Decrement RX byte counter
    if (RXByteCtr)
    {
      *PRxData++ = UCB0RXBUF;               // Move RX data to PRxData
      if (RXByteCtr == 1)                   // Only one byte left?
        UCB0CTL1 |= UCTXSTP;                // Generate stop condition
    }
    else
    {
      *PRxData = UCB0RXBUF;                 // Move RX data to PRxData
      __low_power_mode_off_on_exit();       // Exit LPM0
    }
  }
}
```

## C.3 SPI

*Code Segment C.5: SPI.h*

```
//Functions

void Send_SPI(unsigned char data);
unsigned char Receive_SPI(void);
unsigned char Receive_SPI2(void);
```

*Code Segment C.6: SPI.c*

```
#include "hardware.h"

void Send_SPI(unsigned char data)
{
  //Wait for previous character to be fully transmitted
  while ((SPIIFG & SPITXIFG) == 0);

  //Put the new data in the buffer
  SPITXBUF = data;
  #if CLOCKFREQUENCY == 8
    DELAYU(10);
  #endif
  #if CLOCKFREQUENCY == 12
    DELAYU(13);
  #endif
  #if CLOCKFREQUENCY == 16
    DELAYU(15);
  #endif
}

unsigned char Receive_SPI(void)
{
  SPIIFG &= ~SPIRXIFG;
  Send_SPI('x');

  //wait for complete reception
  while (!(SPIIFG & SPIRXIFG));
  return SPIRXBUF;
}

unsigned char Receive_SPI2(void)
{
  SPI2IFG &= ~SPI2RXIFG;
  Send_SPI('x');
  while(!(SPI2IFG & SPI2RXIFG));
  return SPI2RXBUF;
}
```

## C.4 Fixed Point

*Code Segment C.7: FixedPoint.h*

```
//Functions

short ShortfpMult(short a, short b);
```

*Code Segment C.8: FixedPoint.c*

```
#include "fixedPoint.h"
#include "hardware.h"

short ShortfpMult(short a, short b)
{
  //Declare return variable
  short c;

  //Use Hardware multiplier for 16bit by 16bit multiplication
  MPYS = a;
  OP2 = b;

  //Shift result into fixed point form
  c = ((RESHI & 0x03FF) << 6) + ((RESLO & 0xFC00) >> 10);

  //Return result
  return c;
}
```

## C.5 Dynamic Protocol

*Code Segment C.9: Dynamic.h*

```
#ifndef DYNAMIC_H
#define DYNAMIC_H

// Booleans for interrupts
extern unsigned char checkingMaster;
extern unsigned char scanning_slaves;
extern unsigned char waitForMaster;

// Timeslot & Node role info
extern unsigned char is_master;
extern unsigned char slave_channel;
extern unsigned char timeslot_table[2*SLAVES];
extern unsigned char slaves_count;

// Wait times
extern unsigned int delayMasterDown;

//Functions in dynamic.c
void Startup_Delay(void);
void Check_Master(void);
void Slave_Scan(void);
void Init_Slave(void);
void Master_Conflict_Check(void);
void Slave_Conflict_Check(void);

#endif
```

*Code Segment C.10: Dynamic.c*

```
#include "hardware.h"
#include "nrf2401.h"

// Booleans for interrupts
unsigned char checkingMaster = 0;
unsigned char scanning_slaves = 0;
unsigned char waitForMaster = 0;

// Timeslot & Node role info
unsigned char is_master = 0;
unsigned char slave_channel = 1;
unsigned char timeslot_table[2*SLAVES];
unsigned char slaves_count = 0;

// Wait times
unsigned int delayMasterDown;

void Startup_Delay(void)
{

  // Setup counter for correct operation
  Setup_Counter_Startup();
  for (int i = 0; i<(Sensor & 0x1F); i++)
  {
    // Wait for delay in LPM
    __low_power_mode_3();
  }

  // Stop timer
  TACTL = TASSEL_1 + MC_0;
}

void Check_Master(void)
{
  // Config RF for receival on master channel
  Config_Slave_Receive();

  // Set boolean
  checkingMaster = 1;

  // Setup counter for correct operation
  Setup_Counter_Master_Check();

  // Activate receiver
  Start_Receive();

  // Wait in LPM for process to complete
  __low_power_mode_3();
}

void Slave_Scan(void)
{

  // Set boolean
  scanning_slaves = 1;

  // Activate reciever
  Start_Receive();

  // Wait in LPM for process to complete
  __low_power_mode_3();
}

void Init_Slave(void)
{
  // Clear timeslot table to indicate all channels are available
  for(int i=0; i<2*SLAVES; i++){
    timeslot_table[i] = i+1;
  }

  // Clear the slave count
  slaves_count = 0;

  // Setup counter for correct operation
  Setup_Counter_Slave_Scan();

  // Config RF for receival on first slave channel
  Config_Slave_Scan_Channel1();
  // Scan slaves on the channel
  Slave_Scan();

  // If channel is full, repeat the operation for
  // the second channel and set the slave channel
  if(slaves_count > SLAVES-1)
  {
    Config_Slave_Scan_Channel2();
    Slave_Scan();
    slave_channel = 2;
  }
  else
  {
    slave_channel = 1;
  }

  // Go through the timeslot table to find an empty slot
  for(int i=0;i<2*SLAVES;i++)
  {
    if(timeslot_table[i]<timeSlot)
    {
      timeSlot = timeslot_table[i];
      i = 2*SLAVES;
    }
  }

  // If an available timeslot is found
  if(timeSlot<255)
  {

    // Waiting time untill a slave becomes master:
    // 52 ms + 20 ms * timeslot
    delayMasterDown = 1700 + timeSlot * 654;

    // Calculate waiting time untill timeslot is reached
    Calculate_sendWait(slave_channel);

    // Configure the counters for slave mode
    Setup_Counter_Slave();

    // Configure RF for receival in the master channel
    Config_Slave_Receive();
  }
}

void Master_Conflict_Check(void)
{
  // Config RF chip for receival
  Config_Slave_Receive();

  // Set boolean
  checkingMaster = 1;

  // Start receiving data
  Start_Receive();

  // Wait in LPM for receival or timeout
  __low_power_mode_3();

  // Initialize node as slave if another master is detected,
  // otherwise reconfig RF chip for transmit
  if(!is_master)
  {
    Init_Slave();
    rf_txdata[0] = timeSlot;
    waitToReceive = 1;
  }else{
    Config_Master_Transmit();
  }
}

void Slave_Conflict_Check(void)
{
  // Clear timeslot table to indicate all channels are available
  for(int i=0; i<2*SLAVES; i++){
    timeslot_table[i] = i+1;
  }

  // Clear the slave count
  slaves_count = 0;

  // Config RF chip for receival
  if (slave_channel == 1)
  {
    Config_Slave_Scan_Channel1();
  }
  else
  {
    Config_Slave_Scan_Channel2();
  }

  // Set boolean
  scanning_slaves = 1;

  // Setup counter for correct operation
  Setup_Counter_Slave_Check();

  // Scan slaves on the channel
  Slave_Scan();

  // Check if the sensors own slot is still available
  if (timeslot_table[timeSlot - 1] == 255)
  {

    //Reinitialize sensor node for other slave timeslot
    timeSlot = 255;
    Init_Slave();
    rf_txdata[0] = timeSlot;
    waitToReceive = 1;
  }
  else
  {
    // Reset timer settings
    Setup_Counter_Slave();
    // Config for transmission
    if(slave_channel == 1)
    {
      Config_Slave_Transmit_Channel1();
    }else
    {
      Config_Slave_Transmit_Channel2();
    }
  }
}
```

## C.6 Accelerometer

*Code Segment C.11: AccIO.h*

```
/**********************************************************************
*
* Standard register and bit definitions for the ST Microelectronics
* LIS302DL Accelerometer.
*
**********************************************************************/

#ifndef __lis302dl
#define __lis302dl
#endif

/***********************************************************************
* I2C ADDRESS
***********************************************************************/

#define AccAddress 0x1C

/***********************************************************************
* REGISTER ADDRESSES
***********************************************************************/

#define WHO_AM_I              (0x0F)     //Dummy Register
#define CTRL_REG1             (0x20)
#define CTRL_REG2             (0x21)
#define CTRL_REG3             (0x22)
#define HP_FILTER_RESET       (0x23)     //Dummy Register
#define STATUS_REG            (0x27)
#define OUTX                  (0x29)     //X output register
#define OUTY                  (0x2B)     //Y output register
#define OUTZ                  (0x2D)     //Z output register
#define FF_WU_CFG_1           (0x30)
#define FF_WU_SRC_1           (0x31)
#define FF_WU_THS_1           (0x32)
#define FF_WU_DURATION_1      (0x33)
#define FF_WU_CFG_2           (0x34)
#define FF_WU_SRC_2           (0x35)
#define FF_WU_THS_2           (0x36)
#define FF_WU_DURATION_2      (0x37)
```

```
#define CLICK_CFG           (0x38)
#define CLICK_SRC           (0x39)
#define CLICK_THSY_X        (0x3B)
#define CLICK_THSZ          (0x3C)
#define CLICK_TIMELIMIT     (0x3D)
#define CLICK_LATENCY       (0x3E)
#define CLICK_WINDOW        (0x3F)

#define Multiread           (0x80)    //Add to read multiple bytes


/**********************************************************************
* REGISTER BITS
**********************************************************************/

//CTRL_REG1

#define DR    (0x80)    //Data rate selection.
#define PD    (0x40)    //Power Down Control.
#define FS    (0x20)    //Full Scale selection.
#define STP   (0x10)    //Self Test Enable.
#define STM   (0x08)    //Self Test Enable.
#define ZEN   (0x04)    //Z axis enable.
#define YEN   (0x02)    //Y axis enable.
#define XEN   (0x01)    //X axis enable.


//CTRL_REG2

#define SIM        (0x80)    //SPI Serial Interface Mode selection.
#define BOOT       (0x40)    //Reboot memory content.
#define FDS        (0x10)    //Filtered Data Selection.
#define HPFF_WU2   (0x08)    //HPF enabled for FreeFall/WakeUp # 2.
#define HPFF_WU1   (0x04)    //HPF enabled for FreeFall/WakeUp # 1.
#define HP_COEFF2  (0x02)    //HPF cut-off frequency configuration
#define HP_COEFF1  (0x01)    //HPF cut-off frequency configuration


//CTRL_REG3

#define IHL        (0x80)     //Interrupt active high, low.
#define PP_OD      (0x40)     //Push-pull/Open Drain on interrupt pad.
#define I2CFG2     (0x20)     //Data Signal on Int2 pad control bits.
#define I2CFG1     (0x10)     //Data Signal on Int2 pad control bits.
#define I2CFG0     (0x08)     //Data Signal on Int2 pad control bits.
#define I1CFG2     (0x04)     //Data Signal on Int1 pad control bits.
#define I1CFG1     (0x02)     //Data Signal on Int1 pad control bits.
#define I1CFG0     (0x01)     //Data Signal on Int1 pad control bits.


//STATUS_REG

#define ZYXOR    (0x80)    //X, Y and Z axis data overrun.
#define ZOR      (0x40)    //Z axis data overrun.
#define YOR      (0x20)    //Y axis data overrun.
#define XOR      (0x10)    //X axis data overrun.
#define ZYXDA    (0x08)    //X, Y and Z axis new data available.
```

```
#define ZDA     (0x04)    //Z axis new data available.
#define YDA     (0x02)    //Y axis new data available.
#define XDA     (0x01)    //X axis new data available.
```

Code Segment C.12: Acc.h

```
//Variables

extern unsigned char adata[];
extern long acc[];

//Functions

void Acc_init(void);
void readAcc(void);
void convertAcc(void);
void FilterAcc(void);
```

Code Segment C.13: Acc.c

```
#include "acc.h"
#include "I2C.h"
#include "accio.h"
#include "calibration.h"
#include "fixedPoint.h"
#include "Kalman.h"

// Acceleration variables
unsigned char adata[3];
fixed8_24 acc[3];
fixed8_24 filt_nom[3] = {-2.3728*BASE24,
                              1.929*BASE24,
                                  -0.5309*BASE24};
fixed8_24 filt_den[4] = {0.02565*BASE24, -0.01299*BASE24,
                         -0.01299*BASE24, 0.02565*BASE24};
fixed8_24 output[3][3] = {{0,0,0},{0,0,0},{BASE24,BASE24,BASE24}};
fixed8_24 input[3][3] = {{0,0,0},{0,0,0},{BASE24,BASE24,BASE24}};

void Acc_init(void)
{
  unsigned char TxData[2];

  //Set Slave address to accelerometer
  Set_I2CAddress(AccAddress);

  //Write control bytes
  TxData[0] = CTRL_REG1;
  TxData[1] = PD | XEN | YEN | ZEN;
  Send_I2C(TxData, 2);
}

void readAcc(void)
{
  unsigned char TxData[1];

  //Set slave address to accelerometer
  Set_I2CAddress(AccAddress);

  //Perform read
  TxData[0] = OUTX | Multiread;
  Send_I2C(TxData, 1);
  Receive_I2C(5);

  //Assign values
  adata[0] = -RxBuffer[2];
  adata[1] = RxBuffer[0];
  adata[2] = RxBuffer[4];
}

void convertAcc(void)
{
  short temp[3];

  //Convert to long for Kalman filter
  temp[0] = (signed char)adata[0];
  temp[1] = (signed char)adata[1];
  temp[2] = (signed char)adata[2];
  temp[0] = ShortfpMult(temp[0] << 3, 0x0A00);
  temp[1] = ShortfpMult(temp[1] << 3, 0x0A00);
  temp[2] = ShortfpMult(temp[2] << 3, 0x0A00);
  acc[0] = temp[0];
  acc[1] = temp[1];
  acc[2] = temp[2];
  acc[0] = (acc[0] << 14) + offacc[0];
  acc[1] = (acc[1] << 14) + offacc[1];
  acc[2] = (acc[2] << 14) + offacc[2];

  FilterAcc();
}

void FilterAcc(void){

  //apply digital filter
  for(int i=0;i<3;i++){
    fixed8_24 intermediate = Mult8_24(filt_den[0], acc[i]);
    for(int j=0;j<3;j++){
      intermediate = intermediate
              + Mult8_24(filt_den[j+1], input[i][j])
                    - Mult8_24(filt_nom[j], output[i][j]);
    }
    for(int j=2;j>0;j--){
      input[i][j] = input[i][j-1];
      output[i][j] = output[i][j-1];
    }
    input[i][0] = acc[i];
    output[i][0] = intermediate;
    acc[i] = intermediate;
  }
}
```

## C.7 Magnetometer

*Code Segment C.14: MagIO.h*

```
/*********************************************************************
*
* Standard register and bit definitions for the Yamaha
* YAS529 Magnetometer.
*
*********************************************************************/

#ifndef __yas529
#define __yas529
#endif

/**********************************************************************
* I2C ADDRESS
**********************************************************************/

#define MagAddress 0x2E

/**********************************************************************
* REGISTER ADDRESSES
**********************************************************************/

#define CMDR          (0x00)     //Measurement command register
#define XOFFSETR      (0x20)     //Rough offset X
#define Y1OFFSETR     (0x40)     //Rough offset Y1
#define Y2OFFSETR     (0x60)     //Rough offset Y2
#define ICOILR        (0x80)     //Initialization coil register
#define CONFR         (0xC0)     //Configuration register
#define DOUTR         (0xE0)     //Digital output


/**********************************************************************
* REGISTER BITS
**********************************************************************/

//CMDR

#define NORMAL     (0x00)     //Normal magnetic field measurement
#define ROUGH      (0x01)     //Rough offset measurement
#define NORMTEMP   (0x02)     //Normal + temperature measurement
#define EXTINP     (0x03)     //External input pin measurement
#define COILPLUS   (0x04)     //Normal with test coil on (+)
#define COILMIN    (0x0C)     //Normal with test coil on (-)

//ICOILR

#define COILE      (0x10)     //Coil enable
#define COILSEL0   (0x01)     //Coil select bit 0
#define COILSEL1   (0x02)     //Coil select bit 1
#define COILSEL2   (0x04)     //Coil select bit 2

#define COILSEL_0 (0x00)      //Coil select 0
#define COILSEL_1 (0x01)      //Coil select 1
```

```
#define COILSEL_2 (0x02)     //Coil select 2
#define COILSEL_3 (0x03)     //Coil select 3
#define COILSEL_4 (0x04)     //Coil select 4
#define COILSEL_5 (0x05)     //Coil select 5
#define COILSEL_6 (0x06)     //Coil select 6
#define COILSEL_7 (0x07)     //Coil select 7

//CONFR

#define RDSEL0       (0x08)     //Read select bit 0
#define RDSEL1       (0x10)     //Read select bit 1

#define RDSEL_NORM   (0x00)     //Select measurements results
#define RDSEL_CAL    (0x08)     //Select calibration for read
#define RDSEL_ICOIL  (0x10)     //Select init coil register

//DOUTR

#define Res      (0x14)
#define GPOR     (0x01)     //General purpose output pin
```

*Code Segment C.15: Mag.h*

```
//Variables

extern short mdata[3];
extern long mag[3];

//Functions

void Mag_init(void);
void readMag(void);
void convertMag(void);
void FilterMag(void);
```

*Code Segment C.16: Mag.c*

```
#include "mag.h"
#include "mag.h"
#include "I2C.h"
#include "magio.h"
#include "calibration.h"
#include "hardware.h"
#include "fixedPoint.h"
#include "Kalman.h"

// Avoid Roughs from being read from flash
//#define FIRST

// Writing roughs to flash
//#define SECOND

// Calibration variables
unsigned char CAL[9];
short b[9];
```

```
// Magnetic field variables
short mdata[3];
fixed8_24 mag[3];
fixed8_24 Magfilt_nom[3] = {-2.3728*BASE24,
                                1.929*BASE24,
                                   -0.5309*BASE24};
fixed8_24 Magfilt_den[4] = {0.02565*BASE24, -0.01299*BASE24,
                             -0.01299*BASE24, 0.02565*BASE24};
fixed8_24 Magoutput[3][3] = {{0,0,0},{0,0,0},
                                       {BASE24,BASE24,BASE24}};
fixed8_24 Maginput[3][3] = {{0,0,0},{0,0,0},
                                       {BASE24,BASE24,BASE24}};

void Mag_init(void)
{
  unsigned char TxData[1];

  //Set Slave address to magnetometer
  Set_I2CAddress(MagAddress);

  //Initialize registers: write zeros
  TxData[0] = ICOILR;
  Send_I2C(TxData, 1);
  TxData[0] = CONFR;
  Send_I2C(TxData, 1);

  //Activate initialization coils
  for(unsigned char i = 0; i<8 ;i++) {
    TxData[0] = ICOILR | COILE | i;
    Send_I2C(TxData, 1);

    TxData[0] = ICOILR | ((i+1) & 0x07);
    Send_I2C(TxData, 1);
  }

  //Read factory calibration
  TxData[0] = CONFR | RDSEL_CAL;
  Send_I2C(TxData, 1);
  Receive_I2C(9);
  CAL[0] = RxBuffer[0];
  CAL[1] = RxBuffer[1];
  CAL[2] = RxBuffer[2];
  CAL[3] = RxBuffer[3];
  CAL[4] = RxBuffer[4];
  CAL[5] = RxBuffer[5];
  CAL[6] = RxBuffer[6];
  CAL[7] = RxBuffer[7];
  CAL[8] = RxBuffer[8];

  //Perform rough offset measurement
  TxData[0] = CONFR | RDSEL_NORM;
  Send_I2C(TxData, 1);
  TxData[0] = CMDR | ROUGH;
  Send_I2C(TxData, 1);
  DELAYM(3);
  Receive_I2C(6);
  //Calculate rough offsets
  rough[0] = (RxBuffer[5] & 0x1F);
  rough[1] = (RxBuffer[3] & 0x1F);
  rough[2] = (RxBuffer[1] & 0x1F);

  //Write roughs to flash
  #ifdef SECOND
    WriteMagRough2Flash();
  #endif

  //Retrieve roughs from Flash (not the first time)
  #ifndef FIRST
    RetrieveRoughs();
  #endif

  //Write the rough offsets to the registers
  TxData[0] = XOFFSETR | (rough[0]-5);
  Send_I2C(TxData, 1);
  TxData[0] = Y1OFFSETR | (rough[1]-5);
  Send_I2C(TxData, 1);
  TxData[0] = Y2OFFSETR | (rough[2]-5);
  Send_I2C(TxData, 1);

  //Order mag to perform first measurement
  TxData[0] = CMDR | NORMAL;
  Send_I2C(TxData, 1);

  //Calculate fixed point representation
  CalcFixedPoint(CAL, b);
}

void readMag(void)
{
  unsigned char TxData[1];

  //Set slave address to magnetometer
  Set_I2CAddress(MagAddress);

  //Communicate via I2C
  Receive_I2C(6);

  //Command mag to perform new measurement for next read
  TxData[0] = CMDR | NORMAL;
  Send_I2C(TxData, 1);

  //Declare fixed point variables, short = 16 bit variable.
  short x,y,z;

  //Calculate fixed point representation and add rough offset
  x = ((((rough[0] - 5) + (RxBuffer[4] & 0x07)) << 10)
                             | (RxBuffer[5] << 2)) - (15 << 10);
  y = ((((rough[1] - 5) + (RxBuffer[2] & 0x07)) << 10)
                             | (RxBuffer[3] << 2)) - (15 << 10);
  z = ((((rough[2] - 5) + (RxBuffer[0] & 0x07)) << 10)
                             | (RxBuffer[1] << 2)) - (15 << 10);

  //Convert the axis values into the standard coordinate system
  x = -x;
  short temp = z - y;
  z = y + z;
  y = temp;

  //Apply factory calibration
  mdata[0] = x
                 + ShortfpMult(b[1], y)
                       + ShortfpMult(b[2], z);
  mdata[1] = ShortfpMult(b[3], x)
                 + ShortfpMult(b[4], y)
                       + ShortfpMult(b[5], z);
  mdata[2] = ShortfpMult(b[6], x)
                 + ShortfpMult(b[7], y)
                       + ShortfpMult(b[8], z);

  //Assign values
  mdata[0] = -mdata[0];
  mdata[1] = -mdata[1];
  mdata[2] =  mdata[2];
}

void convertMag(void)
{

  //Convert to long for Kalman filter
  mag[0] = mdata[0];
  mag[1] = mdata[1];
  mag[2] = mdata[2];
  mag[0] = (mag[0] << 14) + offmag[0];
  mag[1] = (mag[1] << 14) + offmag[1];
  mag[2] = (mag[2] << 14) + offmag[2];

  FilterMag();
}

void FilterMag(void){

  //apply digital filter
  for(int i=0;i<3;i++){
    fixed8_24 intermediate = Mult8_24(Magfilt_den[0], mag[i]);
    for(int j=0;j<3;j++){
      intermediate = intermediate
          + Mult8_24(Magfilt_den[j+1], Maginput[i][j])
              - Mult8_24(Magfilt_nom[j], Magoutput[i][j]);
    }
    for(int j=2;j>0;j--){
      Maginput[i][j] = Maginput[i][j-1];
      Magoutput[i][j] = Magoutput[i][j-1];
    }
    Maginput[i][0] = mag[i];
    Magoutput[i][0] = intermediate;
    mag[i] = intermediate;
  }
}
```

## C.8 Calibration

*Code Segment C.17: Calibration.h*

```
//Variables

extern unsigned char rough[];
extern long offacc[];
extern long offmag[];
extern long gainacc[];
extern long gainmag[];

//Functions

void WriteSensorNumber2Flash(int number);
void WriteAccCal2Flash(void);
void WriteMagRough2Flash(void);
void WriteMagCal2Flash(void);
char RetrieveSensorNumber(void);
void RetrieveRoughs(void);
void RetrieveAccCal(void);
void RetrieveMagCal(void);
void CalcFixedPoint(unsigned char CAL[], short* b);
```

*Code Segment C.18: Calibration.c*

```
#include "hardware.h"
#include "calibration.h"

// Offset variable
#define BASE24 16777216

// Flash memory locations
#define SENSOR 0x1000
#define ACCOFF 0x1010
#define ACCGAI 0x1020
#define MAGROU 0x1030
#define MAGOFF 0x1040
#define MAGGAI 0x1050

// Flash pointer
char *Flash_ptr;

//Calibration variables
unsigned char rough[] = {0, 0, 0};
long offacc[3] = {-0.07 * BASE24, -0.05 * BASE24, 0.09 * BASE24};
long offmag[3] = {-0.91 * BASE24, -8.12 * BASE24, -6.77  * BASE24};
long gainacc[3] = {0.89 * BASE24, 0.94 * BASE24, 0.91 * BASE24};
long gainmag[3] = {1.03 * BASE24, 1.04 * BASE24, 1.06 * BASE24};

void WriteSensorNumber2Flash(int number){
  FCTL2 = FWKEY + FSSEL0 + FN1;
  FCTL3 = FWKEY;
  FCTL1 = FWKEY + WRT;

  // Point to Sensor node number
  Flash_ptr = (char *)SENSOR;

  // Write to Flash seg
  *Flash_ptr = number;
}

void WriteAccCal2Flash(void){
  FCTL2 = FWKEY + FSSEL0 + FN1;
  FCTL3 = FWKEY;
  FCTL1 = FWKEY + WRT;

  // Point to start of Acc calibration offset values
  Flash_ptr = (char *)ACCOFF;

  // Write offsets serially byte by byte
  for(int i=0;i<3;i++){
    for(int j=3;j>=0;j--){
      *Flash_ptr++ = (char)(offacc[i] >> (8*j));
    }
  }

  // Point to start of Acc calibration gain values
  Flash_ptr = (char *)ACCGAI;

  // Write gains serially byte by byte
  for(int i=0;i<3;i++){
    for(int j=3;j>=0;j--){
      *Flash_ptr++ = (char)(gainacc[i] >> (8*j));
    }
  }
}

void WriteMagRough2Flash(void){
  FCTL2 = FWKEY + FSSEL0 + FN1;
  FCTL3 = FWKEY;
  FCTL1 = FWKEY + WRT;

  // Point to start of Mag rough offset calibration values
  Flash_ptr = (char *)(MAGROU);

  // Write rough offsets
  for(int i=0;i<3;i++){
    *Flash_ptr++ = rough[i];
  }
}

void WriteMagCal2Flash(void){
  FCTL2 = FWKEY + FSSEL0 + FN1;
  FCTL3 = FWKEY;
  FCTL1 = FWKEY + WRT;

  // Point to start of Mag calibration offset values
  Flash_ptr = (char *)(MAGOFF);

  // Write offsets serially byte by byte
  for(int i=0;i<3;i++){
    for(int j=3;j>=0;j--){
      *Flash_ptr++ = (char)(offmag[i] >> (8*j));
    }
  }

  // Point to start of Mag calibration gain values
  Flash_ptr = (char *)(MAGGAI);

  // Write offsets serially byte by byte
  for(int i=0;i<3;i++){
    for(int j=3;j>=0;j--){
      *Flash_ptr++ = (char)(gainmag[i] >> (8*j));
    }
  }
}

char RetrieveSensorNumber(void){
  Flash_ptr = (char *)SENSOR;
  return *Flash_ptr;
}

void RetrieveRoughs(void){
  Flash_ptr = (char *)MAGROU;
  for(int i=0;i<3;i++){
    rough[i] = *Flash_ptr++;
  }
}

void RetrieveAccCal(void){
  Flash_ptr = (char *)ACCOFF;
  for(int i=0;i<3;i++){
    offacc[i] = 0;
    for(int j=3;j>=0;j--){
      offacc[i] <<= 8;
      offacc[i] += *Flash_ptr++;
    }
  }
  Flash_ptr = (char *)ACCGAI;
  for(int i=0;i<3;i++){
    gainacc[i] = 0;
    for(int j=3;j>=0;j--){
      gainacc[i] <<= 8;
      gainacc[i] += *Flash_ptr++;
    }
  }
}

void RetrieveMagCal(void){
  Flash_ptr = (char *)MAGOFF;
  for(int i=0;i<3;i++){
    offmag[i] = 0;
    for(int j=3;j>=0;j--){
      offmag[i] <<= 8;
      offmag[i] += *Flash_ptr++;
    }
  }
  Flash_ptr = (char *)MAGGAI;
  for(int i=0;i<3;i++){
    gainmag[i] = 0;
    for(int j=3;j>=0;j--){
      gainmag[i] <<= 8;
      gainmag[i] += *Flash_ptr++;
    }
  }
}

void CalcFixedPoint(unsigned char CAL[], short* b){

  //Temp float array
  float a[9];

  //Calculate calibration values as float
  a[0] = 1;
  a[1] = ((CAL[0] & 0xFC)>>(2)) - 32;
  a[1] /= 100;
  a[2] = ((CAL[0] & 0x03)<<(2)) + ((CAL[1] & 0xC0)>>(6)) - 8;
  a[2] /= 100;
  a[3] = (CAL[1] & 0x3F) - 32;
  a[3] /= 100;
  a[4] = ((CAL[2] & 0xFC)>>(2)) - 32;
  a[4] = a[4] / 100 + 0.7;
  a[5] = ((CAL[2] & 0x03)<<(4)) + ((CAL[3] & 0xF0)>>(4)) - 32;
  a[5] /= 100;
  a[6] = ((CAL[3] & 0x0F)<<(2)) + ((CAL[4] & 0xC0)>>(6)) - 32;
  a[6] /= 100;
  a[7] = (CAL[4] & 0x3F) - 32;
  a[7] /= 100;
  a[8] = ((CAL[5] & 0xFE)>>1) - 64;
  a[8] = a[8] / 100 + 1.3;

  //Calculate fixed point representation
  for(unsigned char i = 0;i<9;i++) {
    if (a[i]<0){
      a[i] = -a[i];
      b[i] = 0x8000;
    }
  }
  float test = 16;
  for(unsigned char i=1;i<16;i++){
    for(unsigned char j=0;j<9;j++){
      if (a[j] >= test){
        a[j] -= test;
        b[j] |= 1<<(15-i);
      }
    }
    test /= 2;
  }
  for(unsigned char i = 0;i<9;i++) {
    if (b[i] >> 15){
      b[i] -= 0x8001;
      b[i] ^= 0xFFFF;
    }
  }
}
```

## C.9 RF Transceiver

*Code Segment C.19: nRF2401IO.h*

```
/**************************************************************
*
* Standard register and bit definitions for the Nordic
* nRF2401 Transceiver.
*
**************************************************************/

#ifndef __nrf2401
#define __nrf2401
#endif

// Nordic nRF2401 register configuration

#define RFTXMODE   0x00
#define RFRXMODE   0x01

// config bits 15..8
#define RFTWOCHANNELRX   0x80
#define RFONECHANNELRX   0x00

#define RFSHOCKBURST   0x40
#define RFDIRECT       0x00

#define RF1MBIT    0x20
#define RF250KBIT  0x00

#define RFXTAL4MHZ   0x00
#define RFXTAL8MHZ   0x04
#define RFXTAL12MHZ  0x08
#define RFXTAL16MHZ  0x0C
#define RFXTAL20MHZ  0x10

#define RFPOWERM20DBM  0x00
#define RFPOWERM10DBM  0x01
#define RFPOWERM5DBM   0x02
#define RFPOWER0DBM    0x03

#define RFCRC16BIT   0x02
#define RFCRC8BIT    0x00

#define RFCRCENABLE    0x01
#define RFCRCDISABLE   0x00

// config bits 63..24: address for channel 1,
// 40 bits (5 bytes), unused can be set to 0
#define RFADDR1_1 0xBB
#define RFADDR1_2 0xEE
#define RFADDR1_3 0x55
#define RFADDR1_4 0xFF
#define RFADDR1_5 0x00

// config bits 103..64 address for channel 2,
// 40 bits (5 bytes), unused can be set to 0
#define RFADDR2_1 RFADDR1_1
#define RFADDR2_2 RFADDR1_2
#define RFADDR2_3 RFADDR1_3
#define RFADDR2_4 RFADDR1_4
#define RFADDR2_5 RFADDR1_5
```

*Code Segment C.20: nRF2401.h*

```
// RF channels

// 0..127 = 2400..2527MHz
#define RFMASTERCHANNEL 100
#define RFSLAVECHANNEL1 101

// Channel 2 must be 8 channels apart in multi-channel operation
#define RFSLAVECHANNEL2 109

// (bits 23..18) Addresswidth (max 40 = 5 bytes)
#define RFADDRWIDTH 16

// config bits 111..104
// RF data packets:
// 1 byte = counter
// 1 byte = node ID
// 1 byte = Accelerometer X-Value
// 1 byte = Accelerometer Y-Value
// 1 byte = Accelerometer Z-Value
// 2 bytes = Magnetometer X-Value + Rough Offset
// 2 bytes = Magnetometer Y1-Value + Rough Offset
// 2 bytes = Magnetometer Y2-Value + Rough Offset
// TOTAL: 11 bytes = 88 bits
#define RFDATAWIDTH1 88
// config bits 119..112
#define RFDATAWIDTH2 RFDATAWIDTH1

#define RFFRAMESIZE (RFDATAWIDTH1/8)

extern unsigned char rf_txdata[RFFRAMESIZE];
extern unsigned char rf_rxdata[RFFRAMESIZE];
extern unsigned short rf_rxtime;
extern unsigned short rf_txtime;
extern volatile unsigned char rf_rxtimeoutflag;

// Functions
void config_Nordic(void);
void powerup_Nordic(void);
void powerdown_Nordic(void);
void transmit_Nordic(void);
void Receive_Nordic(void);
void Config_Master_Transmit(void);
void Config_Slave_Receive(void);
void Config_Slave_Scan_Channel1(void);
void Config_Slave_Transmit_Channel1(void);
void Config_Slave_Scan_Channel2(void);
void Config_Slave_Transmit_Channel2(void);
void Start_Receive(void);
void Stop_Receive(void);
```

Code Segment C.21: nRF2401.c

```
/*
 * RF code for Nordic nRF2401
 *
 * MSP430F2132 microcontroller I/O allocation see hardware.h
 */

#include "hardware.h"
#include "nrf2401.h"
#include "nrf2401io.h"
#include "SPI.h"

//default=TRANSMIT, 250 kbit, 16 bit CRC
const unsigned char nordic_config[15]
  ={(RFMASTERCHANNEL<<1)|RFTXMODE,
    RFONECHANNELRX | RFSHOCKBURST | RF250KBIT
          | RFXTAL16MHZ | RFPOWER0DBM,
    RFADDR1_1, RFADDR1_2, RFADDR1_3, RFADDR1_4, RFADDR1_5,
    RFADDR2_1, RFADDR2_2, RFADDR2_3, RFADDR2_4, RFADDR2_5,
    RFDATAWIDTH1,
    RFDATAWIDTH2};

const unsigned char master_transmit
                        = (RFMASTERCHANNEL<<1) | RFTXMODE;
const unsigned char slave_receive
                        = (RFMASTERCHANNEL<<1) | RFRXMODE;
const unsigned char slave_scan_channel1
                        = (RFSLAVECHANNEL1<<1) | RFRXMODE;
const unsigned char slave_transmit_channel1
                        = (RFSLAVECHANNEL1<<1) | RFTXMODE;
const unsigned char slave_scan_channel2
                        = (RFSLAVECHANNEL2<<1) | RFRXMODE;
const unsigned char slave_transmit_channel2
                        = (RFSLAVECHANNEL2<<1) | RFTXMODE;

unsigned char rf_txdata[RFFRAMESIZE];
unsigned char rf_rxdata[RFFRAMESIZE];

void config_Nordic(void)
{
  int i;

  CS_POUT |= (1<<NRF_CS);
  for(i=14;i>=0;i--)           // MSB first
  {
    Send_SPI(nordic_config[i]);
  }
  DELAYU(5);      //THIS DELAY NEEDED!!!!!!
  CS_POUT &= ~(1<<NRF_CS);
}

void powerup_Nordic(void)
{
  //power up RF
  PWRUP_POUT |= (1<<NRF_PWRUP);

  // startup delay (needed)
  DELAYM(3);
}

void powerdown_Nordic(void)
{
  //power down RF
  PWRUP_POUT &= ~(1<<NRF_PWRUP);
}

void transmit_Nordic(void)
{
  int i;

  CE_POUT |= (1<<NRF_CE);
  DELAYU(0);

  // address -- MSB first
  #if RFADDRWIDTH>=40
    Send_SPI(RFADDR1_5);
  #endif
  #if RFADDRWIDTH>=32
    Send_SPI(RFADDR1_4);
  #endif
  #if RFADDRWIDTH>=24
    Send_SPI(RFADDR1_3);
  #endif
  #if RFADDRWIDTH>=16
    Send_SPI(RFADDR1_2);
  #endif
  Send_SPI(RFADDR1_1);

  // data
  for (i=0; i<RFFRAMESIZE; i++)
  {
    Send_SPI(rf_txdata[i]);
  }

  DELAYU(5);
  CE_POUT &= ~(1<<NRF_CE);
}

void Receive_Nordic(void)
{
  // unassign SIMO pin
  SPI_PSEL &= ~(1<<NRF_SIMO);
  SPI_PDIR &= ~((1<<NRF_SIMO)|(1<<NRF_SOMI));

  //Clock out data
  for (int i=0; i<RFFRAMESIZE; i++)
  {
    rf_rxdata[i] = Receive_SPI();
  }

  // reassign SIMO pin
  SPI_PSEL |= (1<<NRF_SIMO);
  SPI_PDIR |= ((1<<NRF_SIMO)|(1<<NRF_SOMI));
}

void Config_Master_Transmit(void)
{
  CS_POUT |= (1<<NRF_CS);
  Send_SPI(master_transmit);
  DELAYU(5);
  CS_POUT &= ~(1<<NRF_CS);
}

void Config_Slave_Receive(void)
{
  CS_POUT |= (1<<NRF_CS);
  Send_SPI(slave_receive);
  DELAYU(5);
  CS_POUT &= ~(1<<NRF_CS);
}

void Config_Slave_Scan_Channel1(void)
{
  CS_POUT |= (1<<NRF_CS);
  Send_SPI(slave_scan_channel1);
  DELAYU(5);
  CS_POUT &= ~(1<<NRF_CS);
}

void Config_Slave_Transmit_Channel1(void)
{
  CS_POUT |= (1<<NRF_CS);
  Send_SPI(slave_transmit_channel1);
  DELAYU(5);
  CS_POUT &= ~(1<<NRF_CS);
}

void Config_Slave_Scan_Channel2(void)
{
  CS_POUT |= (1<<NRF_CS);
  Send_SPI(slave_scan_channel2);
  DELAYU(5);
  CS_POUT &= ~(1<<NRF_CS);
}

void Config_Slave_Transmit_Channel2(void)
{
  CS_POUT |= (1<<NRF_CS);
  Send_SPI(slave_transmit_channel2);
  DELAYU(5);
  CS_POUT &= ~(1<<NRF_CS);
}

void Start_Receive(void)
{
  // enable Nordic receiver
  CE_POUT |= (1<<NRF_CE);
}
```

```
void Stop_Receive(void)
{
  // disable Nordic receiver
  CE_POUT &= ~(1<<NRF_CE);
}
```

## C.10 Kalman

*Code Segment C.22: Kalman.h*

```
#define BASE24 16777216
typedef long fixed8_24;

//Variables

extern long acc[];
extern long mag[];

// function declarations Kalman filter

fixed8_24* filterupdate(fixed8_24* acceleration,
                        fixed8_24* magnetic);
void extendedCorrection(fixed8_24* acceleration,
                        fixed8_24* magnetic);
void Normalize(fixed8_24* acc, fixed8_24* mag);
void HUpdate(void);
void KCalculate(void);
void inverse(void);
void StateUpdate(fixed8_24* acc, fixed8_24* mag);
void CovarianceUpdate(void);
void NormalizeQuaterion(void);
fixed8_24 Mult8_24(fixed8_24 a, fixed8_24 b);
fixed8_24 InvSqrt8_24(fixed8_24 x);
```

*Code Segment C.23: Kalman.c*

```
#include "Kalman.h"
#include "hardware.h"

// Kalman filter implementation on MSP430

//Initialize indices, state- and sensorvector size
int i, j, k;
#define n 4
#define m 6

//Magnetic field constant
const fixed8_24 Gy = (fixed8_24)(0.403073309488957 * BASE24);
const fixed8_24 Gz = (fixed8_24)(-0.91516769347350735 * BASE24);

//Declare filter parameters
fixed8_24 Q[n] = {(fixed8_24)(0.0001 * BASE24),
                    (fixed8_24)(0.0001 * BASE24),
                      (fixed8_24)(0.0001 * BASE24),
                        (fixed8_24)(0.0001 * BASE24)};
fixed8_24 R[m] = {(fixed8_24)(0.01 * BASE24),
                    (fixed8_24)(0.01 * BASE24),
                      (fixed8_24)(0.01 * BASE24),
                        (fixed8_24)(0.02 * BASE24),
                          (fixed8_24)(0.02 * BASE24),
                            (fixed8_24)(0.02 * BASE24)};
fixed8_24 xhat[n] = {1 * BASE24, 0, 0, 0};
fixed8_24 xhatPrev[n] = {1 * BASE24, 0, 0, 0};
fixed8_24 P[n][n] = {{1 * BASE24, 0, 0, 0},
                     {0, 1 * BASE24, 0, 0},
                     {0, 0, 1 * BASE24, 0},
                     {0, 0, 0, 1 * BASE24}};
fixed8_24 H[m][n] = {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0},
                     {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}};
fixed8_24 K[n][m] = {{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}};
fixed8_24 PHT[n][m] = {{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                       {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}};
fixed8_24 temp[m][m] = {{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}};
fixed8_24 L[m][m] = {{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                     {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}};
fixed8_24 Linv[m][m] = {{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}};
fixed8_24 sum = 0;
fixed8_24 t[m] = {0, 0, 0, 0, 0, 0};
fixed8_24 delta[n] = {0, 0, 0, 0};
fixed8_24 tau = 0.8 * BASE24;
fixed8_24 speed[n] = {0, 0, 0, 0};

fixed8_24* filterupdate(fixed8_24* acceleration,
                        fixed8_24* magnetic){

  //Normalize sensor data
  Normalize(acceleration, magnetic);

  //DEBUG
  //acceleration[0] = BASE24;
  //acceleration[1] = 0;
  //acceleration[2] = 0;
  //magnetic[0] = Gz;
  //magnetic[1] = Gy;
  //magnetic[2] = 0;

  //Perform update
  //A: Prediction with unitary matrix

  for(i=0; i<n; i++){
    xhat[i] = xhatPrev[i] + Mult8_24(tau,speed[i]);
    P[i][i] = P[i][i] + Q[i];
  }
  //Calculate norm
  fixed8_24 qNorm = 0;
  for(i=0; i<n; i++){
    qNorm = qNorm + Mult8_24(xhat[i], xhat[i]);
  }

  //Square root
  qNorm = InvSqrt8_24(qNorm);

  //Divide
  for(i=0; i<n; i++){
    xhat[i] = Mult8_24(xhat[i], qNorm);
  }

  //B: Correction
  extendedCorrection(acceleration, magnetic);

  //Normalize result
  NormalizeQuaterion();

  //Save speed
  for(i=0; i<n; i++){
    speed[i] = xhat[i] - xhatPrev[i];
    xhatPrev[i] = xhat[i];
  }

  //return solution
  return xhat;
}

void extendedCorrection(fixed8_24* acceleration,
                        fixed8_24* magnetic){

  //Determine H-matrix with partial derivatives
  HUpdate();

  //Calculate Kalman-Gain
  KCalculate();

  //Correct
  StateUpdate(acceleration, magnetic);

  //Covariance
  CovarianceUpdate();
}

void Normalize(fixed8_24* acc, fixed8_24* mag){

  //Calculate norms
  fixed8_24 AccNorm = 0;
  fixed8_24 MagNorm = 0;
  for(i=0; i<3; i++){
    AccNorm = AccNorm + Mult8_24(acc[i], acc[i]);
    MagNorm = MagNorm + Mult8_24(mag[i], mag[i]);
  }
```

```
   //Square root
   AccNorm = InvSqrt8_24(AccNorm);
   MagNorm = InvSqrt8_24(MagNorm);

   //Normalize
   acc[0] = Mult8_24(acc[0], AccNorm);
   acc[1] = Mult8_24(acc[1], AccNorm);
   acc[2] = Mult8_24(acc[2], AccNorm);
   mag[0] = Mult8_24(mag[0], MagNorm);
   mag[1] = Mult8_24(mag[1], MagNorm);
   mag[2] = Mult8_24(mag[2], MagNorm);
}

void HUpdate(void){

   //Precalculate some multipliations
   fixed8_24 Gy0 = Mult8_24(Gy, xhat[0]) << 1;
   fixed8_24 Gy1 = Mult8_24(Gy, xhat[1]) << 1;
   fixed8_24 Gy2 = Mult8_24(Gy, xhat[2]) << 1;
   fixed8_24 Gy3 = Mult8_24(Gy, xhat[3]) << 1;
   fixed8_24 Gz0 = Mult8_24(Gz, xhat[0]) << 1;
   fixed8_24 Gz1 = Mult8_24(Gz, xhat[1]) << 1;
   fixed8_24 Gz2 = Mult8_24(Gz, xhat[2]) << 1;
   fixed8_24 Gz3 = Mult8_24(Gz, xhat[3]) << 1;

   //Linearization for magnetic field part
   H[0][0] =  Gy3 - Gz2;
   H[0][1] =  Gy2 + Gz3;
   H[0][2] =  Gy1 - Gz0;
   H[0][3] =  Gy0 + Gz1;
   H[1][0] =  Gy0 + Gz1;
   H[1][1] = -Gy1 + Gz0;
   H[1][2] =  Gy2 + Gz3;
   H[1][3] = -Gy3 + Gz2;
   H[2][0] = -Gy1 + Gz0;
   H[2][1] = -Gy0 - Gz1;
   H[2][2] =  Gy3 - Gz2;
   H[2][3] =  Gy2 + Gz3;

   //Linearization for acceleration part
   H[3][0] = -(xhat[2] << 1);
   H[3][1] =  (xhat[3] << 1);
   H[3][2] = -(xhat[0] << 1);
   H[3][3] =  (xhat[1] << 1);
   H[4][0] =  (xhat[1] << 1);
   H[4][1] =  (xhat[0] << 1);
   H[4][2] =  (xhat[3] << 1);
   H[4][3] =  (xhat[2] << 1);
   H[5][0] =  (xhat[0] << 1);
   H[5][1] = -(xhat[1] << 1);
   H[5][2] = -(xhat[2] << 1);
   H[5][3] =  (xhat[3] << 1);
}

void KCalculate(void){

   //Formula: K = P*transpose(H)*inverse(H*P*transpose(H)+R)
```

```
  //PHT = P * Transpose(H)
  for(i=0; i<n; i++){
    for(j=0; j<m; j++){
      PHT[i][j] = Mult8_24(P[i][0], H[j][0]);
      //for(k=1; k<n; k++){
      //  PHT[i][j] = PHT[i][j] + Mult8_24(P[i][k], H[j][k]);
      //}
      for(k=1; k<i+1; k++){
        PHT[i][j] = PHT[i][j] + Mult8_24(P[i][k], H[j][k]);
      }
      for(k=i+1; k<n; k++){
        PHT[i][j] = PHT[i][j] + Mult8_24(P[k][i], H[j][k]);
      }
    }
  }

  //temp = H * PHT = H * P * transpose(H)
  //[ONLY LOWER TRIANGLE => SYMMETRICAL MATRIX!!]
  for(i=0; i<m; i++){
    for(j=0; j<i+1; j++){
      temp[i][j] = Mult8_24(H[i][0], PHT[0][j]);
      for(k=1; k<n; k++){
        temp[i][j] = temp[i][j] + Mult8_24(H[i][k], PHT[k][j]);
      }
    }
  }

  //temp = temp + R = H * P * transpose(H) + R
  for(i=0; i<m; i++){
    temp[i][i] = temp[i][i] + R[i];
  }

  //temp = inverse(temp)
  inverse();

  //K = PHT*temp = P*transpose(H)*inverse(H*P*transpose(H)+R)
  for(i=0; i<n; i++){
    for(j=0; j<m; j++){
      //Symmetric temp: [i][j] with i=0..m-1 and j=0..i
      K[i][j] = Mult8_24(PHT[i][0], temp[j][0]);
      for(k=1; k<j+1; k++){
        K[i][j] = K[i][j] + Mult8_24(PHT[i][k], temp[j][k]);
      }
      for(k=j+1; k<m; k++){
        K[i][j] = K[i][j] + Mult8_24(PHT[i][k], temp[k][j]);
      }
    }
  }
}

void inverse(void){

  float x = 0;

  //Inversion of a symmetric matrix:
  //  Cholesky Decomposition: temp = L * transpose(L),
  //  L = lower triangular matrix
  //  inverse(L) by backsubstitution
  //  inverse(temp) = transpose(inverse(L)) * inverse(L)

  //Cholesky decomposition
  for (i=0; i<m; i++){
    for (j=0; j<i; j++){
      sum = 0;
      for (k=0; k<j; k++){
        sum = sum + Mult8_24(L[i][k], L[j][k]);
      }
      L[i][j] = Mult8_24((temp[i][j] - sum), t[j]);
    }
    sum = 0;
    for (k=0; k<i; k++){
      sum = sum + Mult8_24(L[i][k], L[i][k]);
    }
    t[i] = InvSqrt8_24(temp[i][i] - sum);
    x = t[i];
    x = BASE24 / x;
    x = x * BASE24;
    L[i][i] = (fixed8_24)x;
  }

  //Inversion of L
  for (i=0; i<m; i++){

    //Linv[i][i] = 1 / L[i][i]
    x = L[i][i];
    x = BASE24 / x;
    x = x * BASE24;
    Linv[i][i] = (fixed8_24)x;

    //Linv[i][j] = Linv[i][i]*sum(L[i][k]*Linv[k][j], k=j..i-1)
    for (j=0; j<i; j++){
      sum = 0;
      for (k=j; k<i; k++){
        sum = sum - Mult8_24(L[i][k], Linv[k][j]);
      }
      Linv[i][j] = Mult8_24(Linv[i][i], sum);
    }
  }

  //inverse(temp) = transpose(inverse(L)) * inverse(L)
  //[ONLY LOWER TRIANGLE => SYMMETRICAL MATRIX!!]
  for (i=0; i<m; i++){
    for (j=0; j<i+1; j++){
      temp[i][j] = Mult8_24(Linv[i][i], Linv[i][j]);
      for (k=i+1; k<m; k++){
        temp[i][j] = temp[i][j] + Mult8_24(Linv[k][i], Linv[k][j]);
      }
    }
  }
}

void StateUpdate(fixed8_24* acc, fixed8_24* mag){
  //Formula: xhat = xhat + K * (z - h(xhat))

  //temp = h(xhat)
  fixed8_24 t2 = Mult8_24(xhat[0], xhat[0]);
  fixed8_24 u2 = Mult8_24(xhat[1], xhat[1]);
  fixed8_24 v2 = Mult8_24(xhat[2], xhat[2]);
  fixed8_24 w2 = Mult8_24(xhat[3], xhat[3]);

  fixed8_24 tu = Mult8_24(xhat[0], xhat[1]);
  fixed8_24 tv = Mult8_24(xhat[0], xhat[2]);
  fixed8_24 tw = Mult8_24(xhat[0], xhat[3]);
  fixed8_24 uv = Mult8_24(xhat[1], xhat[2]);
  fixed8_24 uw = Mult8_24(xhat[1], xhat[3]);
  fixed8_24 vw = Mult8_24(xhat[2], xhat[3]);

  t[0] = (Mult8_24((tw + uv), Gy) + Mult8_24((uw - tv), Gz)) << 1;
  t[1] = Mult8_24((t2 - u2 + v2 - w2), Gy)
              + (Mult8_24((tu + vw), Gz) << 1);
  t[2] = Mult8_24((t2 - u2 - v2 + w2), Gz)
              + (Mult8_24((vw - tu), Gy) << 1);

  t[3] = (uw - tv) << 1;
  t[4] = (tu + vw) << 1;
  t[5] = t2 - u2 - v2 + w2;

  //temp = z - temp = z - h(xhat)
  for (i=0; i<m/2; i++){
    t[i]   = mag[i] - t[i];
    t[i+m/2] = acc[i] - t[i+m/2];
  }

  //delta = K * temp = K * (z - h(xhat))
  for (i=0; i<n; i++){
    delta[i] = Mult8_24(t[0], K[i][0]);
    for (j=1; j<m; j++){
      delta[i] = delta[i] + Mult8_24(t[j], K[i][j]);
    }
  }

  //xhat = xhat + delta = xhat + K * (z - h(xhat))
  for (i=0; i<n; i++){
    xhat[i] = xhat[i] + delta[i];
  }
}

void CovarianceUpdate(void){

  //Formula: P = P - K * H * P

  //temp = - K * H * P = - K * transpose(PHT)
  for (i=0; i<n; i++){
    for (j=0; j<i+1; j++){
      temp[i][j] = -Mult8_24(PHT[j][0], K[i][0]);
      for (k=1; k<m; k++){
        temp[i][j] = temp[i][j] - Mult8_24(PHT[j][k], K[i][k]);
      }
    }
  }

  //P = P - temp = P - K * H * P ()
  for (i=0; i<n; i++){
    for (j=0; j<i+1; j++){
      P[i][j] = P[i][j] + temp[i][j];
    }
  }
}

void NormalizeQuaterion(void){

  //Calculate norm
  fixed8_24 qNorm = 0;
  for(i=0; i<n; i++){
    qNorm = qNorm + Mult8_24(xhat[i], xhat[i]);
  }

  //Square root
  qNorm = InvSqrt8_24(qNorm);

  //Divide
  for(i=0; i<n; i++){
    xhat[i] = Mult8_24(xhat[i], qNorm);
  }
}


fixed8_24 Mult8_24(fixed8_24 a, fixed8_24 b){

  //Declare return variable
  fixed8_24 RES_LSB = 0;
  fixed8_24 RES_MSB = 0;
  char negative = 0;

  //Sign trouble
  if (a < 0){
    if (b < 0){
      a = -a;
      b = -b;
    }
    else{
      negative = 1;
      a = -a;
    }
  }
  else{
    if (b < 0){
      negative = 1;
      b = -b;
    }
  }

  //Use Hardware multiplier for 4 16bit by 16bit multiplications
  MPYS = a >> 16;
  OP2 = b >> 16;
  RES_MSB = RESHI << 8;
  RES_MSB += RESLO >> 8;
  RES_LSB = RESLO << 8;
  MPY = b >> 16;
  OP2 = a;
  MAC = a >> 16;
  OP2 = b;
  RES_LSB += RESHI << 8;
  RES_LSB += RESLO >> 8;
  RES_MSB += RESHI >> 8;

  //Shift result into fixed point form
  RES_MSB = RES_MSB << 16;
  RES_MSB += RES_LSB;

  //Sign trouble
  if (negative == 1){
    RES_MSB = -RES_MSB;
  }

  //Return result
  return RES_MSB;
}

fixed8_24 InvSqrt8_24(fixed8_24 x){
    fixed8_24 xhalf = x >> 1;
    float t = x;

    // store floating-point bits in integer
    long i = *(long*)&t;

    // initial guess for Newton's method
    i = 0x5f3759d5 - (i >> 1);
    x = (fixed8_24)(*(float*)&i * BASE24 * 4096);

    // One round of Newton's method
    x = Mult8_24(x, (0x01800000 - Mult8_24(xhalf, Mult8_24(x, x))));
    return x;
}
```

## C.11 Main

*Code Segment C.24: Auto.c*

```
#include "hardware.h"
#include "dynamic.h"
#include "nrf2401.h"
#include "calibration.h"
#include "acc.h"
#include "mag.h"
#include <stdlib.h>

//#define FIRST
#define NUMBER 112
```

```
//Counters
unsigned char counter;
unsigned int conflict_counter;
unsigned int conflict;

void main( void ){

  // Stop WatchDogTimer
  WDTCTL = WDTPW | WDTHOLD;

  // Delay program start
  DELAYM(5);

  #ifdef FIRST
    WriteSensorNumber2Flash(NUMBER);
  #endif

  // Initialize clock registers
  Clock_init();

  //Setup digital communiction
  Com_init();

  // Setup Pin I/O
  Port_init();

  //Turn LED on
  LED_ON;

  //Powerup and config nrf
  powerup_Nordic();
  config_Nordic();

  // enable interrupts (=Set GIE bit)
  __enable_interrupt();

  //Initialize sensors
  Acc_init();
  DELAYU(100);
  Mag_init();

  //Initialize counter
  counter = 0;

  //Initialize timeslot
  timeSlot = 255;

  //Retrieve sensor number
  Sensor = RetrieveSensorNumber();

  //Initialize conflict counter
  conflict_counter = 0;
  srand(Sensor);
  conflict = (rand() & (0x02FF)) + 512;

  //Variable startup delay to avoid multiple masters
  Startup_Delay();
```

```
//Turn LED off
LED_OFF;

//Check if a master is active
Check_Master();

//Initialize counters and wireless interface
if(is_master)
{
  //Setup timers
  Setup_Counter_Master();

  //Set nordic in receive mode
  Config_Master_Transmit();
}else
{
  //Initialize slave (determine timeslot)
  Init_Slave();
  waitToReceive = 1;
}

//Add sensor ID to packet
rf_txdata[1] = Sensor;

while(1){

  //Read new sensor data
  readMag();
  readAcc();

  // RF data packets:
  // 1 byte = MASTER: modulo 256 package counter
  //          SLAVE: timeslot value
  // 1 byte = sensor ID
  // 1 byte = Accelerometer X-Value
  // 1 byte = Accelerometer Y-Value
  // 1 byte = Accelerometer Z-Value
  // 2 bytes = Magnetometer X-Value
  // 2 bytes = Magnetometer Y-Value
  // 2 bytes = Magnetometer Z-Value
  // TOTAL: 11 bytes = 88 bits

  if(is_master)
  {

    //Add counter to packet
    rf_txdata[0] = counter;
  }
  else
  {

    //Add timeslot to packet
    rf_txdata[0] = timeSlot;
  }

  //Add sensor data to buffer
```

```
rf_txdata[2] = adata[0];
rf_txdata[3] = adata[1];
rf_txdata[4] = adata[2];
rf_txdata[5] = mdata[0];
rf_txdata[6] = mdata[0]>>8;
rf_txdata[7] = mdata[1];
rf_txdata[8] = mdata[1]>>8;
rf_txdata[9] = mdata[2];
rf_txdata[10]= mdata[2]>>8;

 // Enter LPM3 with interrupt
__low_power_mode_3();

//Transmit datapackage
transmit_Nordic();

//Increment counter
counter++;
conflict_counter++;

//Conflict avoidance
if(conflict_counter == conflict)
{
    conflict_counter = 0;
    conflict = (rand() & (0x02FF)) + 512;
    if(is_master)
      Master_Conflict_Check();
    else
      Slave_Conflict_Check();
}

if(is_master)
{

  //Blink LED
  if (!((counter & 0x10) == 0))
    LED_TOGGLE;
}
else
{

  //Blink LED
  if(slave_channel == 1)
  {
    //Blink LED number of times equal to TIMESLOT
    if(((counter >> 4) < timeSlot) &
                            ((counter & 0x08) == 0x00))
      LED_ON;
    else
      LED_OFF;
  }
  else
  {
    //Occult LED number of times equal to TIMESLOT
    if(((counter >> 4) < (timeSlot - SLAVES)) &
                            ((counter & 0x08) == 0x00))
      LED_OFF;
```

```
        else
          LED_ON;
      }

      //If counter == 0, resync with master
      if (counter == 0)
      {

        //Config Nordic for receive and wait for turnon
        Config_Slave_Receive();
        waitToReceive = 1;
      }
      else
      {

        //Wait for timeslot
        waitToSend = 1;
      }
    }
  }
}

// Timer0_A0 interrupt service routine
#pragma vector = TIMERA0_VECTOR
__interrupt void Timer0_A0(void)
{
  if(checkingMaster)
  {
    // Unset boolean
    checkingMaster = 0;

    // Turn of receiver
    Stop_Receive();

    // Set boolean to become master
    is_master = 1;

  }else if(scanning_slaves)
  {
    // Unset boolean
    scanning_slaves = 0;

    // Turn off receiver
    Stop_Receive();

  }else if(waitForMaster)
  {
    // Unset boolean
    waitForMaster = 0;

    // Turn off receiver
    Stop_Receive();

    // Stop & reset timer
    TACTL = TASSEL_1 + MC_0;        // ACLK, stop mode
    TACTL |= TACLR;
    // Set boolean to become master
    is_master = 1;

    // Reset counter and LED
    counter = 0;
    LED_OFF;

    // Setup node for master operation
    Setup_Counter_Master();
    Config_Master_Transmit();
  }

  // Wake from sleep
  __low_power_mode_off_on_exit();
}

// Timer0_A1 interrupt service routine
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer0_A1(void)
{
  // Efficient switch-implementation
  switch (TAIV)
  {
    case  2:                      // TACCR1
      if (waitToSend)
      {
        //Unset boolean
        waitToSend = 0;

        //Return to main
        __low_power_mode_off_on_exit();
      }
      else if (scanning_slaves)
      {
        //Unset boolean
        scanning_slaves = 0;

        // Turn off receiver
        Stop_Receive();

        //Return to main
        __low_power_mode_off_on_exit();
      }
      break;
    case  4:                      // TACCR2
      if (waitToReceive)
      {
        //Unset boolean
        waitToReceive = 0;

        //Set new boolean
        waitForMaster = 1;

        //Setup timer to detect if master is down
        TACCR0 = delayMasterDown;
        TACCTL0 = CCIE;
        //Start receiving
        Start_Receive();
      }
      else if (scanning_slaves)
      {
        //Setup timer to interrupt
        TACCTL1 = CCIE;
      }
      break;
    case 10: break;              // Overflow not used
  }
}

#pragma vector = TIMERB1_VECTOR
__interrupt void TIMERB1(void)
{
  switch (TBIV)
  {
  case 10:
    if(checkingMaster)
    {
      // Unset boolean
      checkingMaster = 0;

      // Turn off receiver
      Stop_Receive();

      // Unset master boolean
      is_master = 0;

      // Wake from sleep
      __low_power_mode_off_on_exit();
    }
    else if(scanning_slaves)
    {

      // Receive data
      Receive_Nordic();

      // Indicate timeslot is in use
      if(timeslot_table[rf_rxdata[0]-1] != 255)
      {
        timeslot_table[rf_rxdata[0]-1] = 255;
        slaves_count++;
      }
    }
    else if(waitForMaster)
    {
      //unset boolean
      waitForMaster = 0;

      // Turn off receiver
      Stop_Receive();

      // Receive counter
      Receive_Nordic();
      counter = rf_rxdata[0];
      // Config for transmission
      if(slave_channel == 1)
      {
        Config_Slave_Transmit_Channel1();
      }else
      {
        Config_Slave_Transmit_Channel2();
      }

      DELAYU(100);

      //Wait for timeslot
      waitToSend = 1;

      //Reset timer
      TACCTL0 = 0;
      TACCR0 = MEASUREINTERVAL;
      TAR = MEASUREINTERVAL;
    }
    break;
  default: break;
  }
}
```

*Code Segment C.25: KalmanAuto.c*

```
#include "hardware.h"
#include "dynamic.h"
#include "nrf2401.h"
#include "calibration.h"
#include "acc.h"
#include "mag.h"
#include "Kalman.h"
#include <stdlib.h>

//#define FIRST
#define NUMBER  100
//#define SECOND

// Orientation
long* quaternion;

// counters
unsigned char counter;
unsigned int conflict_counter;
unsigned int conflict;

void main( void ){

   // Stop WatchDogTimer
  WDTCTL = WDTPW | WDTHOLD;

  // Delay program
  DELAYM(5);

  #ifdef FIRST
  WriteSensorNumber2Flash(NUMBER);
#endif

// Initialize clock registers
Clock_init();

//Setup digital communiction
Com_init();

// Setup Pin I/O
Port_init();

//Turn LED on
LED_ON;

//Powerup and config nrf
powerup_Nordic();
config_Nordic();

// enable interrupts (=Set GIE bit)
__enable_interrupt();

//Initialize sensors
Acc_init();
DELAYU(100);
Mag_init();

#ifdef SECOND
  WriteAccCal2Flash();
  WriteMagCal2Flash();
#endif

//Retrieve calibration
RetrieveAccCal();
RetrieveMagCal();

//Initialize counter
counter = 0;

//Initialize timeslot
timeSlot = 255;

//Retrieve sensor number
Sensor = RetrieveSensorNumber();

//Initialize conflict counter
conflict_counter = 0;
srand(Sensor);
conflict = (rand() & (0x02FF)) + 512;

//Variable startup delay to avoid multiple masters
Startup_Delay();

//Turn LED off
LED_OFF;

//Check if a master is active
```

```
Check_Master();

//Initialize counters and wireless interface
if(is_master)
{
  //Setup timers
  Setup_Counter_Master();

  //Set nordic in receive mode
  Config_Master_Transmit();
}else
{
  //Initialize slave (determine timeslot)
  Init_Slave();
  waitToReceive = 1;
}

//Add sensor ID to packet
rf_txdata[1] = Sensor;

while(1){

  //Read new sensor data
  readMag();
  readAcc();

  //Convert sensor data to long
  convertAcc();
  convertMag();

  // RF data packets:
  // 1 byte = MASTER: modulo 256 package counter
  //          SLAVE: timeslot value
  // 1 byte = sensor ID
  // 1 byte = Accelerometer X-Value
  // 1 byte = Accelerometer Y-Value
  // 1 byte = Accelerometer Z-Value
  // 2 bytes = Magnetometer X-Value
  // 2 bytes = Magnetometer Y-Value
  // 2 bytes = Magnetometer Z-Value
  // TOTAL: 11 bytes = 88 bits

  if(is_master)
  {

    //Add counter to packet
    rf_txdata[0] = counter;
  }
  else
  {

    //Add timeslot to packet
    rf_txdata[0] = timeSlot;
  }

  //Calculate Kalman update
  quaternion = filterupdate(acc, mag);
```

```
//Add to sendbuffer
//rf_txdata[2] = quaternion[0] >> 18;
//rf_txdata[3] = quaternion[0] >> 10;
//rf_txdata[4] = quaternion[1] >> 18;
//rf_txdata[5] = quaternion[1] >> 10;
//rf_txdata[6] = quaternion[2] >> 18;
//rf_txdata[7] = quaternion[2] >> 10;
//rf_txdata[8] = quaternion[3] >> 18;
//rf_txdata[9] = quaternion[3] >> 10;
long a,b,c;
if (quaternion[0]>0){
  a = quaternion[1];
  b = quaternion[2];
  c = quaternion[3];
}
else{
  a = -quaternion[1];
  b = -quaternion[2];
  c = -quaternion[3];
}
rf_txdata[2] = a >> 18;
rf_txdata[3] = a >> 10;
rf_txdata[4] = a >> 2;
rf_txdata[5] = b >> 18;
rf_txdata[6] = b >> 10;
rf_txdata[7] = b >> 2;
rf_txdata[8] = c >> 18;
rf_txdata[9] = c >> 10;
rf_txdata[10]= c >> 2;

// Enter LPM3 with interrupt
__low_power_mode_3();

//Transmit datapackage
transmit_Nordic();

//Increment counter
counter++;
conflict_counter++;

//Conflict avoidance
if(conflict_counter == conflict)
{
    conflict_counter = 0;
    conflict = (rand() & (0x02FF)) + 512;
    if(is_master)
    {
      Master_Conflict_Check();
    }
    else
    {
      Slave_Conflict_Check();
    }
}

if(is_master)
```

```
    {

      //Blink LED
      if (!((counter & 0x10) == 0))
      {
        LED_TOGGLE;
      }
    }
    else
    {

      //Blink LED
      if(slave_channel == 1)
      {
        //Blink LED number of times equal to TIMESLOT
        if(((counter >> 4) < timeSlot) &
                                ((counter & 0x08) == 0x00))
          LED_ON;
        else
          LED_OFF;
      }
      else
      {
        //Occult LED number of times equal to TIMESLOT
        if(((counter >> 4) < (timeSlot - SLAVES)) &
                                ((counter & 0x08) == 0x00))
          LED_OFF;
        else
          LED_ON;
      }

      //If counter == 0, resync with master
      if (counter == 0)
      {

        //Config Nordic for receive and wait for turnon
        Config_Slave_Receive();
        waitToReceive = 1;
      }
      else
      {

        //Wait for timeslot
        waitToSend = 1;
      }
    }
  }
}

// Timer0_A0 interrupt service routine
#pragma vector = TIMERA0_VECTOR
__interrupt void Timer0_A0(void)
{
  if(checkingMaster)
  {
    // Unset boolean
    checkingMaster = 0;
    // Turn of receiver
    Stop_Receive();

    // Set boolean to become master
    is_master = 1;

  }else if(scanning_slaves)
  {
    // Unset boolean
    scanning_slaves = 0;

    // Turn off receiver
    Stop_Receive();

  }else if(waitForMaster)
  {
    // Unset boolean
    waitForMaster = 0;

    // Turn off receiver
    Stop_Receive();

    // Stop & reset timer
    TACTL = TASSEL_1 + MC_0;          // ACLK, stop mode
    TACTL |= TACLR;

    // Set boolean to become master
    is_master = 1;

    // Reset counter and LED
    counter = 0;
    LED_OFF;

    // Setup node for master operation
    Setup_Counter_Master();
    Config_Master_Transmit();
  }

  // Wake from sleep
  __low_power_mode_off_on_exit();
}

// Timer0_A1 interrupt service routine
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer0_A1(void)
{
  // Efficient switch-implementation
  switch (TAIV)
  {
    case  2:                          // TACCR1
      if (waitToSend)
      {
        //Unset boolean
        waitToSend = 0;

        //Return to main
        __low_power_mode_off_on_exit();
      }
      else if (scanning_slaves)
      {
        //Unset boolean
        scanning_slaves = 0;

        // Turn off receiver
        Stop_Receive();

        //Return to main
        __low_power_mode_off_on_exit();
      }
      break;
    case  4:                           // TACCR2
      if (waitToReceive)
      {
        //Unset boolean
        waitToReceive = 0;

        //Set new boolean
        waitForMaster = 1;

        //Setup timer to detect if master is down
        TACCR0 = delayMasterDown;
        TACCTL0 = CCIE;

        //Start receiving
        Start_Receive();
      }
      break;
    case 10: break;                // Overflow not used
  }
}

#pragma vector = TIMERB1_VECTOR
__interrupt void TIMERB1(void)
{
  switch (TBIV)
  {
  case 10:
    if(checkingMaster)
    {
      // Unset boolean
      checkingMaster = 0;

      // Turn off receiver
      Stop_Receive();

      // Unset master boolean
      is_master = 0;

      // Wake from sleep
      __low_power_mode_off_on_exit();
    }
    else if(scanning_slaves)
    {
      // Receive data
      Receive_Nordic();

      // Indicate timeslot is in use
      if(timeslot_table[rf_rxdata[0]-1] != 255)
      {
        timeslot_table[rf_rxdata[0]-1] = 255;
        slaves_count++;
      }
    }
    else if(waitForMaster)
    {
      //unset boolean
      waitForMaster = 0;

      // Turn off receiver
      Stop_Receive();

      // Receive counter
      Receive_Nordic();
      counter = rf_rxdata[0];

      // Config for transmission
      if(slave_channel == 1)
      {
        Config_Slave_Transmit_Channel1();
      }else
      {
        Config_Slave_Transmit_Channel2();
      }

      DELAYU(100);

      //Wait for timeslot
      waitToSend = 1;

      //Reset timer
      TACCTL0 = 0;
      TACCR0 = MEASUREINTERVAL;
      TAR = MEASUREINTERVAL;
    }
    break;
  default: break;
  }
}
```