

# Institutionen för systemteknik

## Department of Electrical Engineering

Examensarbete

## Indoor Navigation Using Accelerometer and Magnetometer

Examensarbete utfört i Reglerteknik  
vid Tekniska högskolan vid Linköpings universitet  
av

Johnny Merkel, Joel Säll

LiTH-ISY-EX-ET--11/0385--SE

Linköping 2011



**Linköpings universitet**  
**TEKNISKA HÖGSKOLAN**



# Indoor Navigation Using Accelerometer and Magnetometer

Examensarbete utfört i Reglerteknik  
vid Tekniska högskolan i Linköping  
av

**Johnny Merkel, Joel Säll**


LiTH-ISY-EX-ET--11/0385--SE

Handledare: **Jonas Callmer**  
isy, Linköpings universitet  
**Daniel Staf**  
ENEÄ

Examinator: **David Törnqvist**  
isy, Linköpings universitet

Linköping, 3 October, 2011



	<b>Avdelning, Institution</b> Division, Department  Division of Automatic Control Department of Electrical Engineering Linköpings universitet SE-581 83 Linköping, Sweden	<b>Datum</b> Date  2011-10-03
---	---	--

<b>Språk</b> Language  <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English  <input type="checkbox"/> _____	<b>Rapporttyp</b> Report category  <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> vrig rapport <input type="checkbox"/> _____	<b>ISBN</b> _____ <b>ISRN</b> LiTH-ISY-EX-ET--11/0385--SE <b>Serietitel och serienummer ISSN</b> Title of series, numbering _____
--	--	--

<b>URL fr elektronisk version</b>  <a href="http://www.control.isy.liu.se">http://www.control.isy.liu.se</a> <a href="http://www.ep.liu.se">http://www.ep.liu.se</a>	
---	--

<b>Titel</b> Title	Inomhusnavigering med hjälp av accelerometer och magnetometer Indoor Navigation Using Accelerometer and Magnetometer
<b>Frfattare</b> Author	Johnny Merkel, Joel Säll

**Sammanfattning**  
Abstract

This project will create a navigation system based on dead reckoning using an accelerometer and a magnetometer. There have previously been several studies made on navigation with accelerometers, magnetometers (electronic compass) and gyros. With these three components it is possible to do positioning and different kinds of movement analyses. There are several methods for detection of movement and calculation of position. To achieve greater accuracy in these applications, gyros are often used. Compared to magnetometers and accelerometer gyros consumes a lot of power. In an embedded system with limited power supplies from a battery this may be unacceptable.

In this project a positioning system without a gyro have been developed and evaluated. Is this possible to do, and what accuracy is possible to achieve are questions asked.

Algorithms have been developed and tested in MATLAB. The project is based on a device called a BeeBadge, part of the project will be to transfer the developed algorithms from MATLAB to C-code. Optimizations of the code will be performed due to the constraints in the memory and speed of the microcontroller on the BeeBadge.

<b>Nyckelord</b> Keywords	Kalman, Accelerometer, Magnetometer, Embedded System, Navigation, Hard Iron Compensation, Soft Iron Compensation
------------------------------	--



# Abstract

This project will create a navigation system based on dead reckoning using an accelerometer and a magnetometer. There have previously been several studies made on navigation with accelerometers, magnetometers (electronic compass) and gyros. With these three components it is possible to do positioning and different kinds of movement analyses. There are several methods for detection of movement and calculation of position. To achieve greater accuracy in these applications, gyros are often used. Compared to magnetometers and accelerometers gyros consumes a lot of power. In an embedded system with limited power supplies from a battery this may be unacceptable.

In this project a positioning system without a gyro have been developed and evaluated. Is this possible to do, and what accuracy is possible to achieve are questions asked.

Algorithms have been developed and tested in MATLAB. The project is based on a device called a BeeBadge, part of the project will be to transfer the developed algorithms from MATLAB to C-code. Optimizations of the code will be performed due to the constraints in the memory and speed of the microcontroller on the BeeBadge.

# Sammanfattning

I det här projektet så kommer ett navigeringssystem baserat på dödräkning med hjälp av en accelerometer och en magnetometer att skapas och utvärderas. Det har tidigare gjorts flertalet studier inom navigering med accelerometrar, elektroniska kompasser och gyrooskop. Med dessa tre komponenter är det möjligt att utföra positioneringsberäkningar och olika rörelseanalyser. Det finns flertalet olika metoder för rörelseanalys och beräkning av position. För att uppnå högre noggrannhet vid dessa beräkningar så används ofta gyrooskop. I förhållande till accelerometrar och elektroniska kompasser så drar gyrooskop väldigt mycket ström och i ett inbyggt system där strömförsörjningen från ett batteri är begränsad, kan valet att använda ett gyrooskop bli problematiskt.

I projektet har ett navigeringssystem för positionsbestämning utan gyrooskop utvecklats och utvärderats. Målen har varit att ta reda på om det är genomförbart och vilken noggrannhet man kan uppnå.

Initialt har alla algoritmer utvecklats och testas i MATLAB. Projektet är baserat på en enhet som heter BeeBadge och utöver simuleringarna i MATLAB har alla

algoritmer implementerats till hårdvaran i C-kod. Optimering av koden kommer att utföras på grund av hårdvarubegränsningar av BeeBadgens minne och processor.



# Acknowledgments

We would like to thank:

Enea, for making this project possible.

Our supervisor Daniel Staf at Enea who have been a great aid with software and hardware related problems.

Our supervisor Jonas Callmer who have aided us along the project and we would also like to thank our examiner David Törnqvist



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Method and Purpose . . . . .	3
1.2	Outline . . . . .	4
<b>2</b>	<b>Hardware</b>	<b>5</b>
2.1	Components and Sensors . . . . .	5
2.2	Sensor Axes . . . . .	6
<b>3</b>	<b>Step Detection</b>	<b>9</b>
3.1	Filtering . . . . .	9
3.2	Step Threshold . . . . .	10
3.2.1	Peak Average . . . . .	11
3.2.2	Max and Min Average . . . . .	11
3.3	Step Determination . . . . .	12
<b>4</b>	<b>Heading</b>	<b>15</b>
4.1	Earth's Magnetic Field . . . . .	15
4.2	Rotation based tilt-compensation . . . . .	16
4.3	Projection based tilt-compensation . . . . .	19
4.4	Tilt-compensated Heading . . . . .	20
<b>5</b>	<b>Magnetometer Calibration</b>	<b>23</b>
5.1	Hard Iron Offset . . . . .	25
5.1.1	Experimental Hard Iron Compensation . . . . .	26
5.2	Soft Iron Offset . . . . .	26
5.3	Magnetometer Offset . . . . .	29
<b>6</b>	<b>Navigation Filter</b>	<b>33</b>
6.1	State Estimation and Measurement Models . . . . .	33
6.2	Non-linear Model . . . . .	36
6.3	Filter Model and Implementation . . . . .	36

---

<b>7</b>	<b>Implementation in C</b>	<b>39</b>
7.1	Approach . . . . .	39
7.2	Math Functions . . . . .	40
7.3	Optimization . . . . .	41
<b>8</b>	<b>Experimental Results</b>	<b>43</b>
8.1	Problems and Verification . . . . .	43
8.2	Results . . . . .	45
<b>9</b>	<b>Concluding Remarks</b>	<b>51</b>
9.1	Conclusions . . . . .	51
9.2	Future Work . . . . .	51
	<b>Bibliography</b>	<b>53</b>
<b>A</b>	<b>Trigonometric Functions</b>	<b>55</b>

Symbol	Explanation
$\beta$	Smoothing constant used in filtration
$\theta$	Pitch or soft iron ellipse angle
$\sigma$	Soft iron scaling factor
$\phi$	Roll
$\psi$	Yaw (heading)
$\mathbf{a}$	Accelerometer vector
$a_x, a_y, a_z$	Accelerometer axes
$B$	Used to derive Noise covariance $Q_k$
$B_x, B_y, B_z$	Magnetometer's output (before compensation)
$\mathbf{d}$	Projected vector, tilt compensation
$e_1, e_2, e_3$	World axes
$F_k$	Process model
$g$	Gravity
$G_x, G_y, G_z$	Global axes
$\mathbf{m}$	Magnetometer vector
$\hat{\mathbf{m}}$	Tilt-compensated Magnetometer vector (Projection)
$\mathbf{m}^R$	Tilt-compensated Magnetometer vector (Rotation)
$H_k$	Measurement model
$I_{Ox}, I_{Oy}, I_{Oz}$	Magnetometers internal offset
$m_x, m_y, m_z$	Magnetometer axes (all compensations done)
$m_x^h, m_y^h, m_z^h$	Magnetometers axes (hard iron compensated)
$K_k$	Kalman gain
$n_k$	Accelerometer norm, unfiltered
$N_k$	Accelerometer norm, filtered
$O_x, O_y, O_z$	Hard iron offset on magnetometer axes
$Q_k$	Noise covariance of $w_k$
$P_k$	Estimated Error Covariance
$r_k$	Norm of $m_x$ and $m_y$
$R_1, R_2$	Rotation matrices
$R_k$	Noise covariance of $v_k$
$R_x, R_y$	Rotation matrices
$v_k$	Measurement noise
$w_k$	Process noise
$x_k$	State Estimate
$X_{max}$	$m_x$ when $r_k$ reaches its max value
$Y_{max}$	$m_y$ when $r_k$ reaches its max value
$\hat{x}_k^-$	<i>a priori</i> State Estimate
$\hat{x}_k$	<i>a posteriori</i> State Estimate
$z_k$	Kalman measurement ( $\psi$ )



# Chapter 1

## Introduction

In this project a system for navigation has been developed using a magnetometer and a MEMS accelerometer. The navigation has been based on the detection of steps and calculation of heading.

The project is based on a device called BeeBadge, distributed by Enea. The device is the size of a credit card and in the version used in this project it includes one magnetometer, one accelerometer and a microcontroller used for calculations and wireless communications. In Figure 1.1 the BeeBadge can be seen as it was used during the experiments. It is in the figure placed in a cardboard box and mounted on the hip.

### 1.1 Method and Purpose

Using the accelerometer to detect steps and the magnetometer to calculate heading and position will only give a certain degree of accuracy. Partly due to errors in the step length and partly due to inaccuracies in the compass. The magnetometer is affected by ferrous metals and magnetized objects, which leads to results that sometimes are very erroneous. This is even more so indoors with electrical cabinets, beams or girders. The focus during development of the navigation system has therefore been to do measurements outdoor and from that construct a system that also could work indoors.

Step detection is based on accelerometer data. This will together with the calculated heading from the magnetometer data be processed in an extended Kalman filter. Most of the measurements have been done with the BeeBadge strapped to the belt, but some initial tests were done with the unit placed in the pocket and on the ankle as well as in the hand. A transition of the developed method for having the unit strapped in any of these locations is possible.

## 1.2 Outline

The contents of the chapters in this report will follow the same chronology as the calculations are performed. In the first chapter called *Hardware* there will be an introduction to the hardware and sensors used, followed by how measurements are performed and how the data is processed. The next step is described in *Step Detection* where the treatment of accelerometer data and step detection is described. After this the *Heading* chapter follows where the heading calculation is treated, using both magnetometer and accelerometer data. For the computation of heading to be performed correctly calibration of the magnetometer is necessary and a Chapter *Calibration* is therefore dedicated to this. To increase accuracy in positioning an extended Kalman filter is used, how this is structured can be read about in the *Navigation Filter* chapter. As all development was performed in MATLAB while the BeeBadge is programmed in C, the different functions developed for navigation have also been translated to C. The process behind this is described in Chapter *Implementation in C*.



**Figure 1.1.** In the figure the BeeBadge can be seen mounted on the hip. Photo is taken during testing and the BeeBadge is for convenience mounted in a box.



# Chapter 2

## Hardware

This project is built on an early version of a device called BeeBadge and is the size of a credit card. The BeeBadge is designed to be power efficient and mobile. In the version used in this project an accelerometer, a magnetometer and a wireless microcontroller were included. The microcontroller is used for radio communication and calculations.

Motion tracking and positioning is done in many similar devices, very often a gyro is used. The benefit of having a gyro is that angular velocities can be measured instead of only linear acceleration, allowing for calculations of speed and distance traveled more accurately. Gyros are compared to accelerometers very power consuming. Using only an accelerometer without a gyro to compensate it, acceleration will over time create very large errors.

The BeeBadge has during this project been connected to a FTDI-chip allowing communication via USB. Wireless communication have not been used. The BeeBadge have been printing data via UART to a terminal where it was captured as a txt-file. Simulations in MATLAB were later based on these files, allowing for repeated tests on a specific set of data samples.

### 2.1 Components and Sensors

The accelerometer used is a three-axis and has the possibility to store up to 32 samples of X, Y & Z readings, this means that you need not read as often.

The magnetometer also has three axes though no internal buffer except for the output registers. Worth to mention is that on the magnetometer the output registers are numbered from 0x03 - 0x08, two registers for each X, Y and Z output data. When read sequentially they contain the X, Z and Y-values and not X, Y and Z-values, this needs to be considered when retrieving data and doing calculations.

The calculations are done on a small microcontroller. It has a 32-bit CPU than can run up to 32 MHz with built in 2.4 GHz radio. The RAM and ROM are also built in and 128 kB each. The size of memory and speed of the CPU are the biggest restrictions. The BeeBadge has an API in Eclipse with a framework of functions, including communication with the sensors. Additional functions to support navigation were added and some minor modifications in the existing structure were done, such as sample rate of the sensors and the CPU-speed.

For some timing purposes the microcontroller uses a 32 kHz external crystal oscillator as clock reference. This crystal may be a bit off so there is a built-in possibility of calibrating it against the much more exact 32 MHz internal crystal oscillator. This calibration is basically a measurement of how many 32 kHz ticks there are from the external crystal compared to how many there should be according to the more accurate 32 MHz clock. The calibrated interval is used as a reference for a wake timer. When the wake timer is called it returns the number of ticks that has passed since it was started and the time in seconds can be derived from that. This is amongst other things used as a time-stamp to see when data is read from the sensors.

## 2.2 Sensor Axes

Both the accelerometer and the magnetometer have three axes, X, Y and Z. For the navigation to function as intended some parts of the algorithm requires that the X-axis points forwards, the Y-axis to the right and the Z-axis down. On the BeeBadge the sensors are not aligned with each other. The magnetometer's X and Y axes are turned 90° with respect to the accelerometer's X and Y axes. A global positioning of all axes is therefore necessary, the alignment can be seen in Figure 2.1. The new axes will in this document be referred to as  $G_x$ ,  $G_y$  &  $G_z$  respectively.

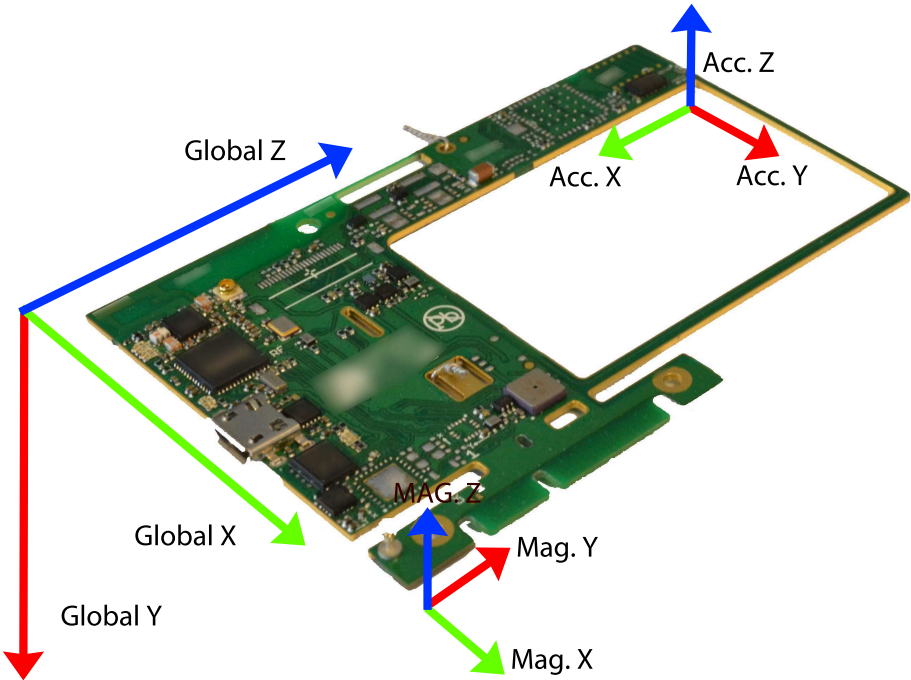


Figure 2.1. Alignment of magnetometer and accelerometer axes in global axes system.



# Chapter 3

## Step Detection

In this project the detection of movement is based on the detection of steps, which are derived from the accelerometer's output. When a step is detected the system will calculate the heading and the new position. In this chapter different methods for detecting steps are explained.

When a step is taken the three axes on the accelerometer will be affected and the output will change. Depending on where the accelerometer is mounted on the person carrying it the output will look different, but there are similar patterns. The methods for step detection described in this chapter have only been thoroughly tested when the unit is carried on the hip, but with some modifications they can also be used when the unit is carried on the ankle, in a pocket or in a hand.

When walking at a constant pace, the acceleration is always greatest at the beginning and at the end of a step [9]. This is due to the cyclic up and down motion the walker has. There will also be a small acceleration in the walking direction and a rotating movement around the center of the walking person. All these motions together affect the accelerometers X, Y and Z output. To simplify the detection of steps the norm of the accelerometer data is calculated. This eliminates the need for the accelerometer to be oriented in a specific way as there is only one instead of three samples of data to look at.

The accelerometer is very sensitive to motions, the different axes have individual biases and the output is not 100% accurate. This makes low pass filtering of data necessary.

### 3.1 Filtering

To be able to calculate accurate results low pass filtering is necessary. During the initial experiments in MATLAB a low pass third order Butterworth filter was used. This filter was later compared to a first order filter and the same accuracy in step detection could be achieved. The low pass Butterworth filter was therefore

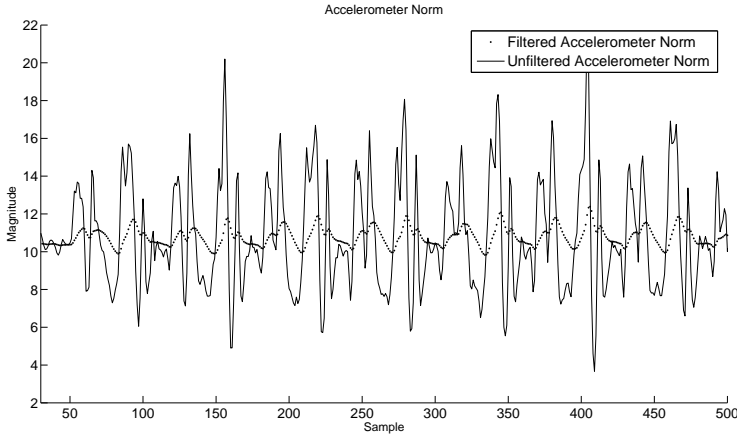
replaced by a simple first order low pass filter, see (3.1a). The new filter does not provide as smooth output because it is only a first order filter, but it proved to produce good results anyway. It is also slightly less computationally heavy than a third order filter.

$$N_k = \beta n_k + (1 - \beta)N_{k-1} \quad (3.1a)$$

$$N_k = N_{k-1} + \beta(n_k - N_{k-1}) \quad (3.1b)$$

Here  $n_k$  is the unfiltered accelerometer norm and  $N_k$  is the filtered norm. The subscript  $k$  is the index in a set of data.  $\beta$  is called the smoothing constant and is a number between 0 and 1. The closer to 1 that  $\beta$  is, the lower is the cut-off frequency. This is more clear in (3.1b).

Filtering the accelerometer axes independently tends to give unreliable results. Filtering of both the accelerometer axes and the norm also gives results that are erroneous. It is therefore sufficient to filter only the calculated norm, saving time and power. In Figure 3.1 both filtered and unfiltered accelerometer norm can be seen when the simple filter is used. In the example used, the unit have been strapped to the belt when walking.



**Figure 3.1.** Filtered and unfiltered norm from accelerometer output. From a walking set when the unit is strapped to the belt.

## 3.2 Step Threshold

In Figure 3.1 the filtered norm of accelerometer data can be seen. From the data norm a threshold is derived to help decide when a step occurs. A step can be said to occur when the data norm crosses this threshold. In this project two simple

ways of calculating a threshold have been tested. Every person has its own style when walking, and the same person will produce different accelerometer norms if walking fast or slow, on grass or on asphalt. During measurements, the person wearing the BeeBadge may be changing speed. The threshold must therefore be dynamic, it must be able to adapt itself after the accelerometers output. To be derived they both require a number of data samples and the algorithms can therefore not be used in real time, but steps are detected immediately after a data set has been collected. The number of samples necessary to calculate a threshold can be chosen arbitrarily. When walking, a set of one second of samples gives a good result, in this project one second corresponds to 50 samples. Fewer samples can be used but gives a more unstable result. A peak in the data norm occurs before or after a foot is put down on the ground. A valley occurs when the foot is in contact with the ground and the leg is vertical above it. The peaks seen in the data norm have different amplitude depending on where the BeeBadge is carried. If it is placed on the right hip the right foot on the same side will give higher peaks in the accelerometer norm.

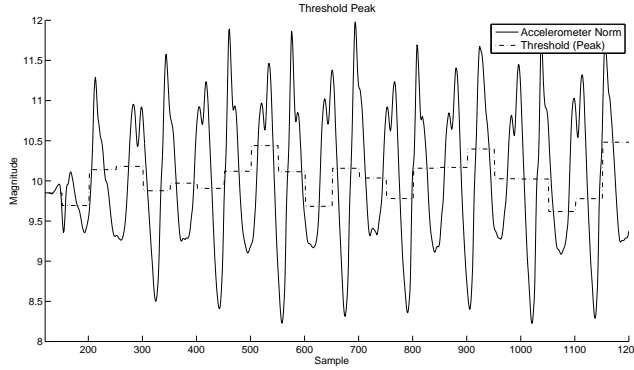
### 3.2.1 Peak Average

This method for calculating a threshold uses the height of the peaks occurring in a set of samples. All the peaks are added together and then divided by the number of peaks that occurred. The result is the threshold for the set. When the different thresholds were tested a low pass Butterworth filter was used, this method then proved to give worse and more unstable result than the Max Min method described below. As the work progressed the Butterworth filter used for filtration was replaced by a much more efficient and simpler low-pass filter. A result of this new filter was that the peak average method gave more stable and reliable results than before. The reason for this might be that the third order Butterworth filter produces a too smooth result. When the new first order filter is used some small peaks between the higher peaks are left. These small peaks might help creating a better average. The peak average method is not implemented in the final version of the navigation algorithm, the method below is used. In Figure 3.2 the threshold when the unit is strapped to the belt can be seen.

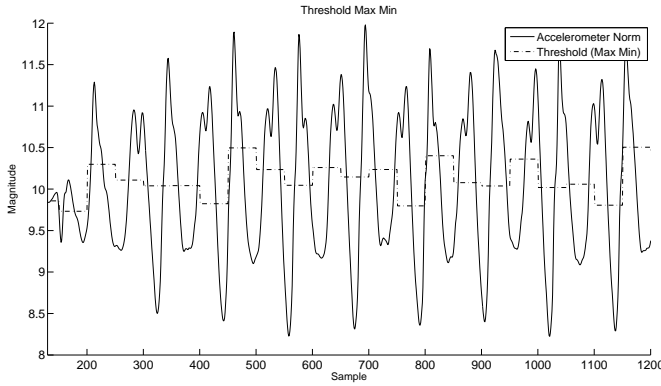
### 3.2.2 Max and Min Average

This method finds the max and min values of the data norm in a set of data and averages them [9]. The result is the threshold for that set. In Figure 3.3 the threshold derived using this method can be seen.

The Max and Min average method gives a more stable threshold and consequently more accurate results. It is therefore the method that is implemented in the final version of the navigation algorithm.



**Figure 3.2.** Threshold of accelerometer norm when walking. Threshold derived with peak average and used for detecting steps.



**Figure 3.3.** Threshold of accelerometer norm when walking. Derived with averaging max and min value in a set of data. Threshold is used for step detection.

### 3.3 Step Determination

Two methods for the step detection have been tested. The first one detects steps when a falling edge of the data norm is equal to, or one sample above the threshold, see Figure 3.4. Even small peaks over the threshold will be considered a valid step, so this method may detect steps also when one is not taken. An example of this is the first circle in Figure 3.4(a).

To avoid this issue another method was used in the final version of the step detection. Here steps are only detected on peaks higher than  $0.5 \text{ m/s}^2$  above the threshold.

A peak is detected by looking at a sample in a data set, say sample  $k$ . If sample  $k$  minus sample  $k - 1$  is positive it means that is the rising edge of the data norm.



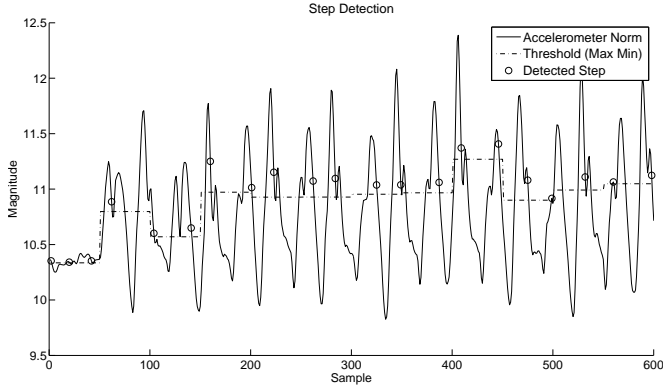
The next step is to subtract sample  $k$  from sample  $k + 1$ , if the result is negative it means that a falling edge of the data norm is detected. If both these two calculations are true, sample  $k$  is a peak, that is, a step. The formula used for peak detection can be seen in (3.2).

*Formula for Detection of Peaks on Norm*

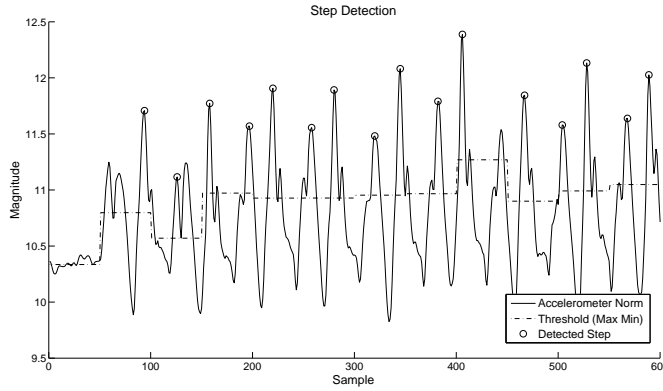
$$if( N_k - N_{k-1} \geq 0 \quad \&\& \quad N_{k+1} - N_k < 0 ) \quad (3.2a)$$

$$then : Step is detected \quad (3.2b)$$

On some occasions there might be small peaks just next to larger peaks. These are not real steps, but it may be difficult to distinguish between them. The fact that they are located close to real peaks means two things. Firstly they are located both before and after real peaks, so statistically they even out. Secondly the fact that they are located close to real peaks also means that two steps can be detected when only one has occurred. During the MATLAB simulations the solution to this was to wait for fifteen data samples to pass before detecting a new step (at 50 Hz corresponding to 0.2 seconds). In the final C-code version of step-detection, the time stamp mentioned in Chapter 2 was used to keep track of when a detected step had occurred, making the idle period more precise and simplifying if the sample rate were to be changed. If the person carrying the BeeBadge is running instead of walking the idle period might be too long, leading to missed steps. This project is focused on walking, but if algorithms are added that can detect whether a person is walking or running the idle time can be adapted after this.



(a) Steps detected on threshold



(b) Steps detected on peaks

**Figure 3.4.** Figure 3.4(a) depicts detected steps when the accelerometer norm crosses the threshold. Figure 3.4(b) depicts steps detected on peaks at least  $0.5 \text{ m/s}^2$  above the threshold. Both methods have a minimum limit of 15 samples idle period between steps.

# Chapter 4

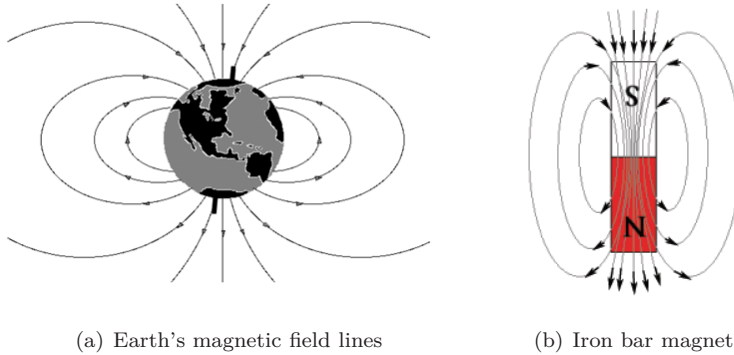
## Heading

The magnetometer measures the strength of Earth's magnetic field in three dimensions. These measurements can be used to determine the magnetometer's relation to Earth's magnetic north. The global axes have each been assigned a direction to represent the BeeBadge's orientation,  $G_x \leftrightarrow \textit{North}$ ,  $G_y \leftrightarrow \textit{East}$  and  $G_z \leftrightarrow \textit{Down}$ .  $h$  is defined as the magnetometer vector and contains data from each of the magnetometer axes. When the magnetometer's  $G_x$  and  $G_y$  axes are aligned with Earth's surface, the heading can be derived as  $\psi = \arctan(m_y/m_x)$ . However, large errors will be introduced if the unit is tilted or disorientated in such a way that  $G_x$  and  $G_y$  no longer are parallel to Earth's surface. This is because Earth's magnetic field is not parallel to the surface which can be seen in Figure 4.1(a). A magnetometer alone can not be used to decide how the device is tilted but together with the accelerometer this is possible. There are different ways to compensate for a tilted unit and two different approaches have been tried in this project.

### 4.1 Earth's Magnetic Field

Earth's magnetic field can be compared to the magnetic field of a regular dipole bar magnet, see Figure 4.1(a) and 4.1(b). The geographic poles lie on Earth's rotation axis where the North Pole is called the true north. However, the magnetic South Pole lies close to the geographic North Pole and the magnetic North Pole close to the geographic South Pole. Magnetic north will be used to refer to the magnetic South Pole. The angle between the magnetic north and the true north is called magnetic declination and it varies all over Earth's surface. It is only possible to compensate for the declination angle if the current position is known. Depending on location, Earth's magnetic field lines will vary in both strength and direction. The magnetic field lines will always point towards magnetic north, on the southern hemisphere they point upward from the surface while on the northern they point downward towards the surface, this is called magnetic inclination. Information of declination and inclination angles for any location on Earth can be found at [4]. In central Sweden the magnetic inclination is approximately 75 degrees downward

towards the surface. Now say that the magnetometer is held parallel to Earth's surface and a heading angle of 45 degrees from north is derived, which indicates that  $G_x$  and  $G_y$  sense the same magnetic strength. If the magnetometer is tilted from its position so that  $G_x$  points slightly downwards while  $G_y$  remains the same,  $G_x$  would sense a greater magnetic field strength than  $G_y$  and the new heading angle would not be 45 degrees. This can make a big difference in the heading calculation and the only solution is to implement some kind of tilt-compensation.

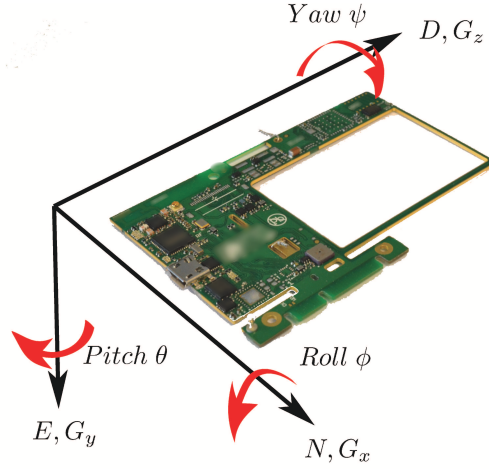


**Figure 4.1.** Earth's magnetic field lines are closely related to the magnetic field of a simple bar magnet. These two pictures illustrates that the magnetic and geographical poles are reversed. 4.1(a) is borrowed from [6]

## 4.2 Rotation based tilt-compensation

If the BeeBadge is tilted in any direction, this can be described as a rotation around one or several of the global axes. Rotations around  $G_x$ ,  $G_y$  and  $G_z$  are called pitch, roll and yaw. The roll  $\theta$  is a rotation around  $G_x$ , the pitch  $\phi$  around  $G_y$  and the yaw  $\psi$  rotations around  $G_z$ . An illustration of the BeeBadge's rotation angles together with its global axes can be seen in Figure 4.2.

Tilting the BeeBadge in a way so that the plane formed by  $G_x$  and  $G_y$  no longer is parallel to Earth's surface will cause erroneous headings. In Figure 4.3(a) an illustration is shown when the BeeBadge has been tilted. The  $e_1$ ,  $e_2$  and  $e_3$  axes forms the coordinate system where the BeeBadge is not tilted.  $e_3$  points straight down towards Earth's surface and  $e_1$  and  $e_2$  forms a horizontal plane. The tilted coordinate system in the figure can be described by two rotations, a roll  $\theta$  around  $G_x$  and a pitch  $\phi$  around  $G_y$ . The separate effects of the roll and the pitch can be seen in Figure 4.3(b) and 4.3(c). If the roll and pitch are known, the global coordinate system could be rotated to align with  $e_d$  and  $e_3$ . The yaw angle is the same thing as heading, if the yaw is compensated the heading will always point towards magnetic north. This is of course not desired and therefore the yaw will not be



**Figure 4.2.** Picture of a BeeBadge illustrating pitch, roll and yaw of the device.  $N, E$  and  $D$  stand for North, East and Down and shows how the BeeBadge is supposed to be positioned.

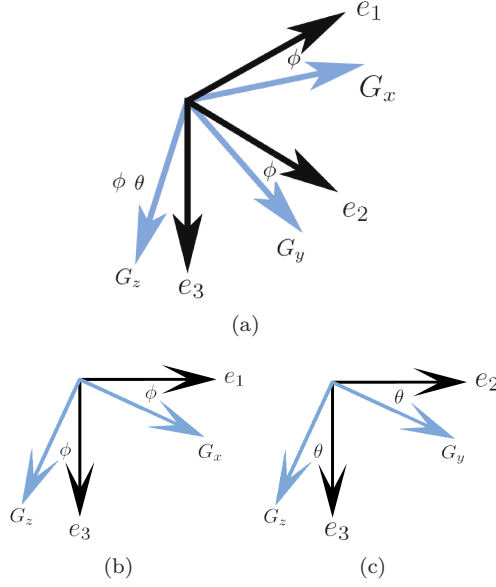
compensated. When the BeeBadge is held completely still, it can be assumed that the only measured acceleration is gravity. The acceleration vector  $\mathbf{a}$  (4.1) would then be aligned with  $e_3$  and point straight down towards Earth's surface. This can be used as a base to derive  $\theta$  and  $\phi$  according to (4.2) and (4.3). In order for this to work both the accelerometer and magnetometer axes must be aligned, that is every accelerometer axis must point in the same direction as the corresponding magnetometer axis. If they are not aligned  $\theta$  and  $\phi$  would not describe the real rotation of the BeeBadge.

$$\mathbf{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \quad (4.1)$$

$$\theta = \text{atan2}(a_x, a_z) \quad (4.2)$$

$$\phi = \text{atan2}(a_y, a_z) \quad (4.3)$$

The actual rotation is done by multiplying the magnetometer vector  $\mathbf{m}$  (4.4) with rotation matrices, one for each axis. The rotation matrices for the  $G_x$  and  $G_y$  axes are called  $R_x$  and  $R_y$ . These are derived based on  $\theta$  and  $\phi$  in (4.5) and (4.6)



**Figure 4.3.**  $e_1$ ,  $e_2$  and  $e_3$  forms the coordinate system where the BeeBadge is not tilted. The top figure shows the BeeBadge in both a tilted and a non-tilted state. The tilt is described by the roll angle  $\theta$  and the pitch angle  $\phi$ . The last two figures shows how roll and pitch effects the other axes.

*Magnetometer vector*

$$\mathbf{m} = \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} \quad (4.4)$$

$$R_y(\phi) = \begin{pmatrix} \cos \phi & 0 & -\sin \phi \\ 0 & 1 & 0 \\ \sin \phi & 0 & \cos \phi \end{pmatrix} \quad (4.5)$$

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{pmatrix} \quad (4.6)$$

The final step is to derive the rotated magnetic vector  $\mathbf{m}^R$  by applying the rotation matrices to  $\mathbf{m}$  (4.7).

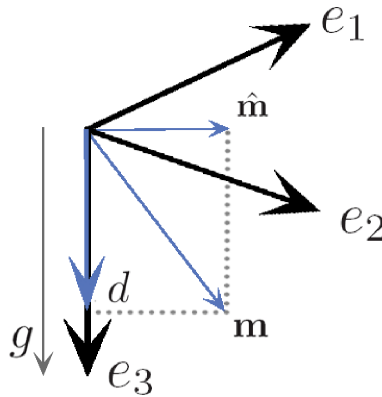
$$\mathbf{m}^R = R_x(\theta)R_y(\phi) \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} \quad (4.7)$$

The tilt-compensated heading can now be derived from the new magnetic vector  $\mathbf{m}^R$  (4.8).

$$\psi = \text{atan2}(m_y, m_x) \quad (4.8)$$

### 4.3 Projection based tilt-compensation

When the BeeBadge is held still, the acceleration vector  $\mathbf{a}$  (4.1) will point straight down towards Earth's surface, regardless of rotation. The magnetic vector  $\mathbf{m}$  (4.7) will point in the same direction as the magnetic field lines. If  $\mathbf{a}$  points straight down, it is possible to project  $\mathbf{m}$  so that  $m_x$  and  $m_y$  form a plane parallel to Earth's surface. In Figure 4.4  $\mathbf{m}$  has been drawn in relation to  $e_1$ ,  $e_2$  and  $e_3$ ; the coordinate system where the BeeBadge is not tilted. If  $\mathbf{m}$  is projected on  $\mathbf{a}$ , a new vector will be received which points in the same direction as  $\mathbf{a}$ . By dividing this vector with the scalar product of  $\mathbf{a}$ , the length of  $\mathbf{m}$  will be represented in  $\mathbf{a}$ . The vector  $\mathbf{d}$  is a projection of  $\mathbf{m}$  on  $\mathbf{a}$  and is derived in (4.9). This can be used to derive how  $\mathbf{m}$  should be projected to put the  $m_x$  and  $m_y$  plane parallel to Earth's surface.



**Figure 4.4.** When the BeeBadge is held still, the acceleration vector  $\mathbf{a}$  will point straight down towards Earth's surface, the same direction as  $e_3$ .  $\mathbf{d}$  is the projected length of  $\mathbf{a}$  on  $\mathbf{m}$ . The projected magnetic vector  $\hat{\mathbf{m}}$  is derived in (4.9)

$$\mathbf{d} = \frac{\mathbf{a}^T \mathbf{m}}{\mathbf{a}^T \mathbf{a}} \mathbf{a} \quad (4.9)$$

As can be seen in Figure 4.4,  $\mathbf{d}$  is aligned with  $e_3$ . The compensated magnetic vector  $\hat{\mathbf{m}}$  is derived by subtracting  $\mathbf{d}$  from  $\mathbf{m}$  (4.10)

$$\hat{\mathbf{m}} = \mathbf{m} - \mathbf{d} \quad (4.10)$$

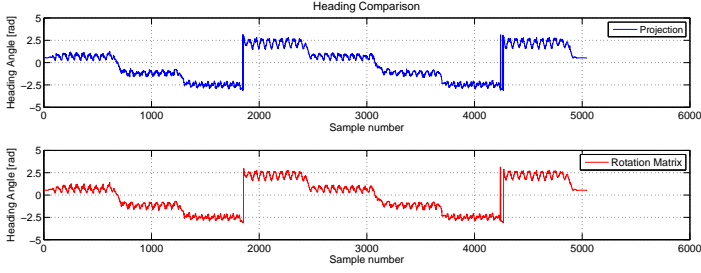
The tilt-compensated magnetic vector should have its  $m_x$  and  $m_y$  plane parallel to Earth's surface and the heading can then be derived in accordance to (4.11).

$$\psi = \text{atan2}(\hat{m}_y, \hat{m}_x) \quad (4.11)$$

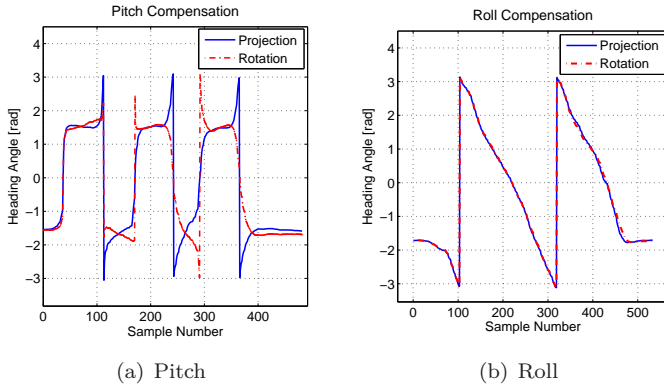
## 4.4 Tilt-compensated Heading

The result of the projection and the rotation is almost exactly the same while wearing the BeeBadge in the belt or in the pocket. In Figure 4.5, both methods were applied on the same measurements. In the figure it can be seen that the two methods perform very equal results. The experiment was made outdoors and with a person walking 160 steps a  $20 \times 20$  steps square pattern. Each side of the rectangle can be seen as a unique heading level in the figure and it can also be seen that the heading returns to the same values in the last 80 steps. Both methods are based on that when the BeeBadge is held still, the acceleration vector points straight down towards Earth's surface. If the BeeBadge is being moved, the acceleration vector will point in different directions. As long as there is not any large, sudden movements, the gravity should still be the largest acceleration measured. To reduce errors introduced by movements, the acceleration vector is low pass filtered quite hard. A pitch and a roll test are shown in Figure 4.6(a) and 4.6(b). In the pitch test the BeeBadge was rotated around the  $G_y$  axis and in the roll test around the  $G_x$ . The roll test in 4.6(b) shows that the two methods gives very similar results, while the pitch test indicates larger differences. It is difficult to say which approach performs best without a real reference what the compensated heading should be. The projection method is less computationally heavy and has therefore been the one implemented in C.





**Figure 4.5.** Picture of heading derived with tilt-compensation, both with projection and matrix rotation. A person has walked 160 steps in a square pattern of  $20 \times 20$  steps. The BeeBadge was worn in the belt.



**Figure 4.6.** Heading comparison of the projection and matrix rotation applied on a pitch and roll test. In the left figure the BeeBadge was rotated around the  $G_y$  axis and in the right around its  $G_x$ .



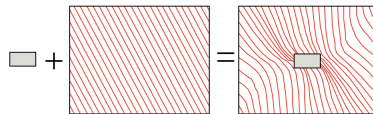
## Chapter 5

# Magnetometer Calibration

The magnetometer measures magnetic fields, when used as a compass you want it to measure the Earth's magnetic field. But it will also be affected by other magnetic fields. These disturbances are usually referred to as hard and soft iron distortions (sometimes effects) and they will affect the accuracy of heading calculations. It is not uncommon that these influences are as large as or larger than the Earth's magnetic field. It is therefore necessary to understand where they come from and how they can be compensated for.

Hard iron distortions are caused by constant magnetic fields around the magnetometer. These magnetic fields come from magnetized metals or magnets. When the magnetometer moves, the hard iron influence on the magnetometer's axes does not change. Hard iron effects cause a constant additive value to the Earth's magnetic field and thereby a constant error to the magnetometer readings. If a speaker containing a magnet is mounted close to the magnetometer it will cause hard iron distortions.

Soft iron distortions are caused by variations in the magnetic field surrounding the magnetometer, see Figure 5.1. It can for example be ferrous metals in the Earth or a car passing by. In most cases the soft iron effects have a smaller contribution to the error than the hard iron effects.



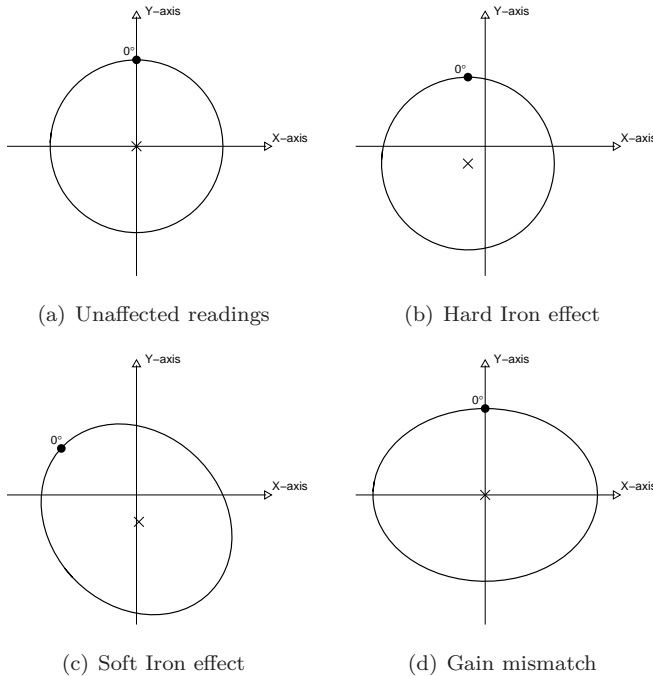
**Figure 5.1.** The effect of soft iron distortions by a ferrous object in a uniform field, figure borrowed from [3]

A third possible cause for errors is the magnetometer itself. The different axes can, due to their inner construction, have different biases (offsets)[1]. This will cause a constant additive error similar to hard iron distortions. The axes can also have

different gain. If the X and Y axes are exposed to one Gauss each, they might produce different outputs. The result of this would be similar to that of soft iron distortions.

Hard and soft iron effects that are in a fixed location with respect to the magnetometer (also if the sensor is moved) can be found during a calibration routine. This is only necessary to do once and after that the offsets can be removed from every data reading. Important to notice is that compensation both is calculated and removed from filtered data. In this project  $\beta$  in Equation 3.1 was set to 0.97.

If a three axis magnetometer were to be turned  $360^\circ$  around its Z axis (which points in the direction of gravity) and if X was plotted with respect to Y, a circle should appear. In the ideal case the circle is centered around zero, but it may be centered around something else due to hard iron effects. The soft iron effects will distort the Earth's magnetic field and give the plot of X with respect to Y an elliptic shape. An example figure of different disturbances can be seen in Figure 5.2.

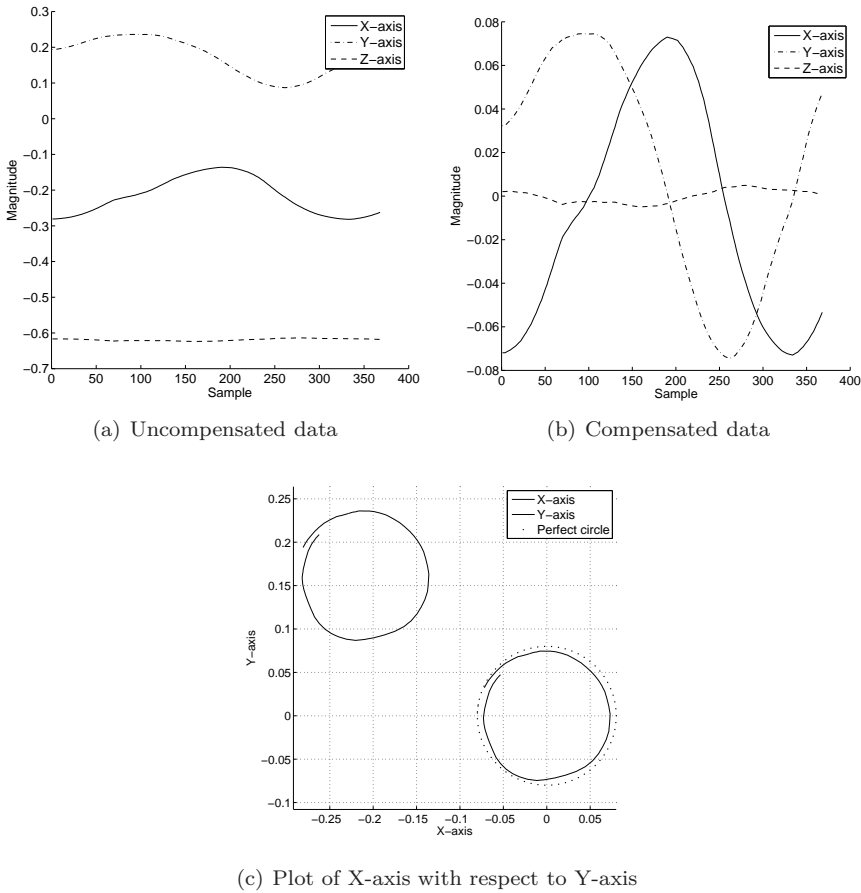


**Figure 5.2.** Example plots of how different offsets can affect the magnetometer data.

## 5.1 Hard Iron Offset

When the magnetometer is measuring Earth's magnetic field there are always distortions. Some of these come from magnetized objects which will cause an additive offset to the Earth's magnetic field. Hard iron effects from objects in a fixed position with respect to the magnetometer can be found during a calibration process and then be removed from every data sample.

There are different methods for calculation of offsets. The method used in this project is simple but gives a good result for the proposed application. The results of the compensation can be seen in Figure 5.3.



**Figure 5.3.** Magnetometer readings when the sensor is turned  $360^\circ$ . In 5.3(a) and (b) the different curves represent X, Y and Z axis output. Figure (a) depicts uncompensated and (b) hard iron compensated magnetometer data. Figure (c) is the X-axis plotted with respect to the Y-axis, both hard iron compensated and uncompensated. All figures are from the same set of data.

To do a calibration, the magnetometer was spun  $360^\circ$  on a flat surface with the X-axis and Y-axis in the horizontal plane and the Z-axis pointing towards the Earth. This was done by hand, which is why there are some small disturbances. To remove the offset the data was slightly low pass filtered to get a smoother set of data to work with. This is the first part of the calibration routine used in this project.

After a complete revolution the magnetometer's X, Y and Z-axes will each have reached a maximum and a minimum value. For each axis the max and min value should be the same but with opposite signs, the average of them should be zero. If it is not, there is an offset present. This offset is divided by two and the result is then removed from the future data samples, see (5.1).

#### *Calculation of Magnetometer Offset*

$$O_x = \frac{\max(B_x) + \min(B_x)}{2} \quad (5.1a)$$

$$O_y = \frac{\max(B_y) + \min(B_y)}{2} \quad (5.1b)$$

$$O_z = \frac{\max(B_z) + \min(B_z)}{2} \quad (5.1c)$$

$$m_x^h = B_x - O_x \quad (5.1d)$$

$$m_y^h = B_y - O_y \quad (5.1e)$$

$$m_z^h = B_z - O_z \quad (5.1f)$$

$B_{x,y,z}$  is the magnetometer's output on the different axes.  $O_{x,y,z}$  is the calculated offset and  $m_{x,y,z}^h$  is the hard iron compensated result, which either also will be soft iron compensated or used in the heading calculations as it is if soft iron compensation is not necessary.

### 5.1.1 Experimental Hard Iron Compensation

During the experiments in MATLAB it was possible to do calculations on complete tests where all data was available. It was then possible to try different kinds of hard iron compensation, for example calculating the mean value of the magnetometer axes. If a test is performed where a person carries the BeeBadge and returns to the starting position after a walk this method can be used. The person walking has to keep a constant pace. The derived mean value can then be used as hard iron compensation.

## 5.2 Soft Iron Offset

The soft iron effect is not caused by materials that are magnetic in themselves, but by materials that affect the magnetic field. The effect on the magnetic field can be quite large. In short, the origin of soft iron effect is because a magnetic

field always wants to take the shortest path. Materials with high permeability are an easier path as they can be said to have a higher conductivity for magnetic fields. When Earth's magnetic field passes in the vicinity of a material with higher permeability than air it changes direction to go through it. This can for example be steel or nickel. The name soft iron distortions comes from that metals with high permeability are called soft when talking in terms of electromagnetism.

So the difference compared to hard iron effects is that it is not caused by materials that are magnetic in them self. It is caused by materials that are affected by a magnetic field and by that affects a magnetic field. The error is therefore not additive to the magnetometers readings as the effect will not increase or decrease the strength of the magnetic field, only change its direction. This makes compensation for soft iron distortion is much more difficult than compensation for hard iron distortion. This is especially true if the distortions are not constantly in a fixed position compared to the magnetometer's axes.

As can be seen in Figure 5.2(c) the soft iron distortion will turn a plotted rotation from a circle to an ellipse. A compensation routine will be described in this document. As with hard iron compensation this routine is only done once to retrieve the compensation values, which then can be used on every data sample. In the routine the ellipse is rotated and scaled to a circle before it is rotated back [8].

The soft iron calibration is based on the same set of data as the hard iron calibration is based on, but it is conducted after hard iron calibration has been performed. If the magnetometer is tilted this must also have been compensated for (see Chapter 4). The first step of the routine is to calculate the norm of the magnetometers X and Y axis, called  $m_x^h$  and  $m_y^h$ . The norm is called  $r_k$ , where  $k$  represents the index in the data set that is used. This norm represents how far from the origin the plot of  $m_x^h$  with respect to  $m_y^h$  would be located. The calculated norm is checked for the points where it reaches its largest value and its smallest value. The ellipse will naturally have two places each where the norm reaches maximum respectively minimum values. If a line is drawn between the two maximum values and another line between the two minimum values, these two lines would then be orthogonal. The line between the maximum values is called *Major axis* and the line between the minimum axis *Minor axis*. Like the names suggest, these lines form a coordinate system for the ellipse. This coordinate system has to be rotated to the global coordinate system so that the ellipse can be transformed into a circle. Therefore the values of  $m_x^h$  and  $m_y^h$  in the maximum point of  $r_k$  are saved in variables called  $X_{max}$  and  $Y_{max}$ , see (5.2). At the same time the maximum value of  $r_k$  is saved in a variable called  $r_{max}$ . The values of  $m_x^h$  and  $m_y^h$  in the minimum point of  $r_k$  does not need to be saved, only the minimum value of  $r_k$  is saved.

*Soft Iron Compensation*

$$r_k = \sqrt{m_x^h[k]^2 + m_y^h[k]^2} \quad (5.2a)$$

$$\text{if } r_k > r_{max} \quad (5.2b)$$

$$r_{max} = r_k \quad (5.2c)$$

$$X_{max} = m_x[k] \quad (5.2d)$$

$$Y_{max} = m_y[k] \quad (5.2e)$$

$$\text{if } r_k < r_{min} \quad (5.2f)$$

$$r_{min} = r_k \quad (5.2g)$$

To rotate *Major axis* and *Minor axis* to the global coordinate system, the angle with which the *Major axis* and  $G_x$  are separated by has to be calculated. The angle is called  $\theta$  and is calculated using (5.3a). For the rotation a rotation matrix called  $R_1$  is used. When the rotation has been done the rotated *Major axis* is scaled by  $\sigma$ , calculated in (5.3b). The rotation is done by multiplying every  $m_x^h$  and  $m_y^h$  reading with  $R_1$ . This is done so that the *Major axis* lies in the same position as the global X axis ( $G_x$ ) and *Minor axis* with global Y axis ( $G_y$ ). The *Major axis* is to be scaled to the same width as the *Minor axis*, this is done by multiplying every rotated  $m_x^h$  value with  $\sigma$ .

*Soft Iron Compensation*

$$\theta = \arctan(Y_{max}, X_{max}) \quad (5.3a)$$

$$\sigma = \frac{r_{min}}{r_{max}} \quad (5.3b)$$

$$R_1 = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

The last step is to rotate everything back to its original position with rotation matrix  $R_2$ , see (5.4c).

$$R_2 = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

*Soft Iron Compensation*

$$\begin{pmatrix} m_x^R \\ m_y^R \end{pmatrix} = R_1 \begin{pmatrix} m_x^h \\ m_y^h \end{pmatrix} \quad (5.4a)$$

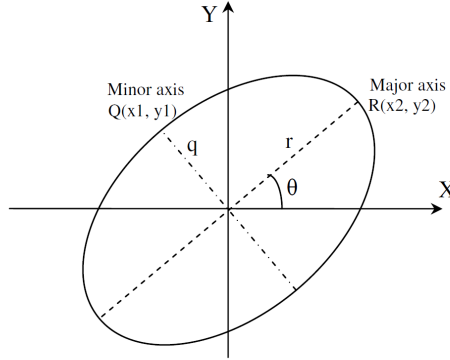
$$m_{x\sigma}^R = \sigma * m_x^R \quad (5.4b)$$

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = R_2 \begin{pmatrix} m_{x\sigma}^R \\ m_y^R \end{pmatrix} \quad (5.4c)$$



What is described above is the derivation and the use of soft iron compensation. The  $\sigma$  and  $\theta$  values are saved. When the compensation is to be carried out during actual measurements only the calculations done in Equations (5.4a-c) are carried out.

An example figure describing  $m_x^h$  plotted with respect to  $m_y^h$  can be seen in Figure 5.4. In this plot the hard iron effects have been removed and tilt compensation has been applied, but the soft iron effects are still present.



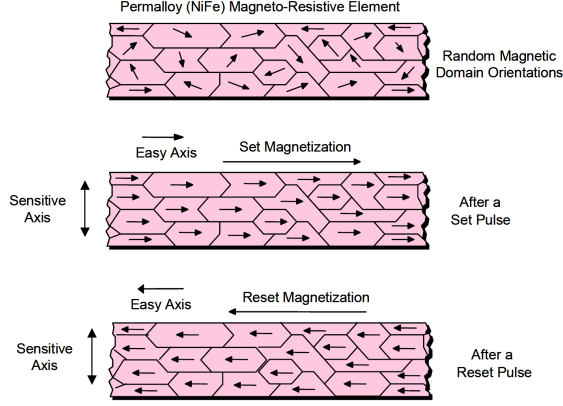
**Figure 5.4.** Magnetometer readings when the sensor is turned 360° and exposed to soft iron effects. The X-axis is plotted with respect to the Y-axis on hard iron compensated data. Centered around [0,0] with rotation  $\theta$ , figure from [8].

## 5.3 Magnetometer Offset

The magnetometer that was used during this project is an anisotropic magnetoresistive (AMR) sensor. Honeywell's AMR sensors use an alloy of nickel and iron in a thin film. When exposed to a magnetic field this film changes resistivity. Just like magnetic recording tapes the magnetic domains of the film can change direction when exposed to a strong magnetic field. If the magnetic domains of the particles in the film are not aligned the accuracy and resolution will be affected. To realign the orientation of the magnetic domains of the film, a magnetic field can be applied to the sensor, see Figure 5.5. This is called a set-reset pulse [1].

Another cause for erroneous readings is that when the temperature changes, the resistance of the film will be affected. If two measurements are made on the same magnetic field, but at different temperatures, the output of the magnetometer will vary. Problems may occur when hard iron distortions are compensated for, if the different axes on the magnetometer do not change with the same amount. The hard iron compensation is a fixed number subtracted from the data measurements so that they are aligned around zero. This value might get too small or big if the

offset from the magnetometer readings vary.



**Figure 5.5.** The alignment of magnetic domains in thin film sensor when affected by strong external magnetic field. Before and after set-reset pulse, figure from [1]

To avoid this problem, the magnetometer has a built in self-test [5]. When this is activated, a bias field with known strength is created. The bias field can be set to be either positive or negative. The field strength is  $\pm 0.64$  Gauss for the magnetometers X and Y axis and  $\pm 0.59$  Gauss for the Z axis. Two measurements are then made, first a set pulse is set by the magnetometer and only the external magnetic field is measured on each axis. Then a reset pulse is set and the external magnetic field plus the activated bias field is measured on each axis. After this the first measurement is subtracted from the second measurement and the result is placed in the output registers of the sensor, see (5.5).

#### *Calculation of Magnetometer Internal Offset*

$$I_{O,x} = B_x - (B_x + 0.64) \quad (5.5a)$$

$$I_{O,y} = B_y - (B_y + 0.64) \quad (5.5b)$$

$$I_{O,z} = B_z - (B_z + 0.59) \quad (5.5c)$$

$$(5.5d)$$

$B_{x,y,z}$  is the magnetometers output on the different axes when only the external magnetic field is measured.  $I_{O,x,y,z}$  is the magnetometers resulting internal offset. The numbers 0.64 and 0.59 are the strength of the applied field in Gauss and can be either positive or negative [5].

The magnetometers output registers does not contain data in Gauss but an unsigned raw format. The numbers can range from  $-2048$  to  $2047$ . To be converted to Gauss, the number retrieved is simply divided by a gain specified when the magnetometer is initialized. If the X axis on a magnetometer does not have any

internal gain then  $I_{O,x}$  in (5.5) will be  $\simeq 655$ , see (5.6) where the gain is 1024.

*Conversion of Magnetometer Internal Offset to Raw Format*

$$1024 \cdot 0.64 \simeq 655 \tag{5.6}$$

If the output register of the magnetometer do not contain this value an offset is present, which should be subtracted from the measurement data. Depending on the application where the magnetometer is implemented this self-test needs to be done at different intervals, from every second and up to just once when the unit is initialized.



# Chapter 6

## Navigation Filter

The objective with this project is to describe trajectories, this is done with coordinates. Coordinates can be derived by the trigonometric sine and cosine functions of the heading angle. In order to improve the precision of the system a more controlled filter environment was desired. The Kalman filter is a powerful recursive method for discrete-data linear filtering problems and offers several features to improve the accuracy of linear systems. It was first introduced by R.E Kalman in 1960 [7]. The filter is a framework for estimating states with covariances based on measurements and models for linear systems. It can be applied in many different applications in different areas and has been especially important for autonomous and navigational systems [2]. In this project an Extended Kalman filter has been used that can handle mildly nonlinear systems.

### 6.1 State Estimation and Measurement Models

The Kalman filter uses knowledge about a system's dynamics together with control signals and measurements to create an estimate of the system's state. The goal is to make a more accurate estimation than an estimation based only on system measurements. The system's calculated state is affected in a negative way by several things, some of these are inaccurate and unaccounted external events that generates noisy measurements. Quantization noise also introduces errors. The system's equations have approximations and limitations. All these things will give rise to offsets from the true sensor data.

The Kalman filter uses state space models which relate inputs, outputs and state variables by first order differential equations. The variables used for inputs, outputs and states are represented by vectors and the equations are represented on matrix form. The Kalman filter is discrete in the time domain and at each discrete time increment a new state is derived based on a linear function and process noise. At different points in the discrete time domain, different knowledge about the system may be available, some sensor inputs may for example be active more rarely than others. The Kalman filter uses all available information in a system at

an instant in time and if there are sensor inputs which are currently not available, they are excluded from the derivation of the new state.

The states are represented by the  $n$  dimension vector  $\mathbf{x}_k$  at time  $k$ . They are derived from the previous states  $\mathbf{x}_{k-1}$ . A process model is needed to derive the state estimate  $\mathbf{x}_k$ , the model is based on the framework of the Kalman filter. This process model is a dynamic model. Usually it is based on basic physical laws where the next state is based on previous state. An example can be a car driving on a road. If the speed of the car is to be calculated it will be last speed measurement plus acceleration multiplied with time since last measurement, see (6.1).

$$\begin{pmatrix} P_{t+1} \\ v_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} P_t \\ v_t \end{pmatrix} + \begin{pmatrix} 0 \\ \omega T \end{pmatrix} \quad (6.1)$$

### State Equation

$$\mathbf{x}_k = F\mathbf{x}_{k-1} + Bu_{k-1} + w_{k-1} \quad (6.2a)$$

$$w_k \sim N(0, Q_k) \quad (6.2b)$$

$F$  is an  $n \times n$  matrix and relates the previous estimate  $\mathbf{x}_{k-1}$  to the current estimate  $\mathbf{x}_k$ . The variable  $u_k$  is an optional  $l$  dimensioned control input and is related to  $\mathbf{x}_k$  by the  $n \times l$  matrix  $B$ . The process noise is described by the variable  $w_k$  and is assumed to be zero mean white noise with a covariance  $Q_k$ .

### Measurement Equation

$$z_k = H_k\mathbf{x}_k + v_k \quad (6.3a)$$

$$v_k \sim N(0, R_k) \quad (6.3b)$$

The  $m$  dimensioned vector  $z_k$  contains the measurement at time  $k$  and the  $m \times n$  matrix  $H$  is a measurement model that relates the current states to the measurement. The measurement noise  $v_k$  is like the process noise considered to be white and has a covariance  $R_k$ . The matrices  $F$  and  $H$  can in different applications be either fixed or updated at each time step.

The process noise covariance and the measurement noise covariance are directly related to how trustworthy a state estimation or a measurement is. Low covariance of  $Q_k$  and  $R_k$  will give rise to high trust, with growing covariances the uncertainty in the equations will also grow. The extreme case  $0 \leftarrow Q_k$  would imply complete trust in the previous state  $\mathbf{x}_{k-1}$  while the case  $Q_k \rightarrow \infty$  implies no trust at all. With an infinitely large  $Q$  the uncertainty would be so large that anything could happen between two updates. The same reasoning goes for the measurement equation, the case  $0 \leftarrow R_k$  will imply that the measurement is exact while the extreme case  $R_k \rightarrow \infty$  would give a measurement with infinitely large uncertainty. A measurement or a previous state with high uncertainty means that it is very unreliable and should be discarded.

The filter process can be divided into two steps, *time* and *measurement update*, these equations can be found in Table 6.1 and 6.2. The filter process alternates between these two steps: predict the future state at the *time update* and adjust the predicted state in the *measurement update*. Equation variables denoted with a minus sign in the right top is *a priori* state variables and are derived in the *time update* equations. The variables derived in the *measurement update* are referred to as *a posteriori* and do not use the minus sign. For example the *a priori* state estimate  $\hat{\mathbf{x}}_k^-$  in (6.4a) is an estimate of the state before the new measurement has taken place and is made unknowingly of the outcome of the measurement.

The state equation (6.5b) derives the final output from the filter at time  $k$  and is a linearization of  $\hat{\mathbf{x}}_k^-$  and a gain scaled difference between the estimated and the real measurement. The gain is described by the  $n \times m$  matrix  $K$  (6.5a). If all indexes in  $K$  are zero the new measurement would be considered so unreliable that the measurement would be completely discarded. The result of equation (6.5b) would then turn out as the *a priori* state estimate  $\hat{\mathbf{x}}_k^-$ . The  $n \times n$  matrix  $P_k$  is the estimated error covariance and describes how much to trust the state estimations.  $P_k$  is updated both in (6.4b) and in (6.5c). The gain matrix  $K$  is derived based on the estimated error covariance.

**Table 6.1.** Kalman filter time update equations

$$\hat{\mathbf{x}}_k^- = F\hat{\mathbf{x}}_{k-1} + B\hat{u}_{k-1} \quad (6.4a)$$

$$P_k^- = FP_{k-1}F^T + Q \quad (6.4b)$$

**Table 6.2.** Kalman filter measurement update equations

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (6.5a)$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + K_k(z_k - H\hat{\mathbf{x}}_k^-) \quad (6.5b)$$

$$P_k = (I - K_k H)P_k^- \quad (6.5c)$$

## 6.2 Non-linear Model

The general approach when it comes to Kalman filters requires a linear system model. This is not the case in this project. The system attains measurements from the accelerometer and the magnetometer with a constant ratio. However, there are parts of the system model which introduce non-linear features. When a person is walking the system tries to detect and determine if a step was taken or not. The time between the steps will end up to be different between each step and this gives rise to non-linearities. The trigonometric functions sine and cosine used to compute the position are of a non-linear nature. The solution is to use an Extended Kalman filter which uses a linearized non-linear model. However, some of the earlier mentioned non-linear aspects will be ignored. Variance in time between different steps is not considered due to the constraint of fixed step length. The non-linear trigonometric sine and cosine functions must be linearized in order to generate trustworthy estimations. For a more detailed explanation and practical examples of both the Kalman filter and the Extended Kalman filter see [2].

The measurement and state models that are used in this project can be seen in Equation (6.6a) and (6.6b) respectively.

$$\hat{\mathbf{x}}_k^- = f(\mathbf{x}_{k-1}, u_{k-1}) + w_{k-1} \quad (6.6a)$$

$$z_k = H\mathbf{x}_k + v_k \quad (6.6b)$$

## 6.3 Filter Model and Implementation

The following equations derived in this section can be found in Table 6.4 and 6.5. The state  $x_k$  describes the current position and heading at sample  $k$ , where  $x$  and  $y$  are coordinates and  $\psi$  heading. Filter sampling is based on the event of a step, where  $k$  is the step number.

$$\mathbf{x}_k = \begin{pmatrix} x_k \\ y_k \\ \psi_k \end{pmatrix}$$

When a step is detected at time  $k$ , the concurrent derived heading is sent as an input to the filter. The first thing that happens is an estimation of the *a priori* state  $\hat{\mathbf{x}}_k^-$  (6.6a). The control signal  $u_{k-1}$  is the step length, set to a constant value in this application. The update of the *a priori* state estimate (6.6a) will be:

$$\hat{\mathbf{x}}_k^- = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \psi_{k-1} \end{pmatrix} + \begin{pmatrix} \cos(\psi_{k-1}) \\ \sin(\psi_{k-1}) \\ 0 \end{pmatrix} u_{k-1} \quad (6.7)$$

What happens next is the calculation of the estimated error covariance  $P_k^-$  (6.9).  $P_k^-$  is a  $3 \times 3$  matrix and all its elements are initially set to zero. For  $P_k^-$  to be calculated the Jacobian matrix  $A_k$  is needed (6.8).  $A_k$  relates the previous estimated error covariance  $P_{k-1}$  to the new estimation  $P_k^-$ .  $A_k$  is updated at each



step and is the derivative of  $f$ .

$$A_k(\hat{\mathbf{x}}_k) = \left. \frac{\delta f}{\delta \mathbf{x}} \right|_{x=\hat{x}_{k-1}} = \begin{pmatrix} 1 & 0 & -\sin(\psi_{k-1})\ell \\ 0 & 1 & \cos(\psi_{k-1})\ell \\ 0 & 0 & 1 \end{pmatrix} \quad (6.8)$$

The process noise  $Q$  is a  $3 \times 3$  matrix and is derived in advance. It is considered to be constant, this is because no consideration is taken to the time between the detected steps. The result of this is that the same process noise will be added to the estimated error covariance at each step, the same portion of uncertainty is added at each step. All variables for an update of  $P_k^-$  are now available.

$$P_k^- = A_k P_{k-1} A_k^T + Q \quad (6.9)$$

The measurement model uses the  $1 \times 3$  matrix  $H_k$  to relate the state to the measurement  $z_k$ . In order to make  $H_k$  relate measurement to the state,  $H_k$  is set to:

$$H_k = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

The filter only has one measurement input, the heading, corresponding to the detected step at time  $k$ .  $z$  holds the new measurement which will be denoted  $\psi_k$ , where  $\psi_k$  is the derived heading at time  $k$ .

$$z_k = \psi_k$$

The measurement noise  $R_k$  is assumed to be constant and is set in advance, just like the process noise. A value of 0.05 has proven to give good results.

$$R_k = 0.05$$

The gain  $K$  is derived in accordance to (6.10). The gain update will look like:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (6.10)$$

The *a posteriori* state estimate  $\hat{\mathbf{x}}_k$  can now be derived accordingly to (6.11).  $\hat{\mathbf{x}}_k$  turn out as the *a priori* state estimate added to the gained difference between the real and the estimated measurement.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + K_k(z_k - H_k \hat{\mathbf{x}}_k^-) \quad (6.11)$$

Finally the estimated error covariance matrix  $P_k$  has to be updated, (6.12), where  $I$  is a  $3 \times 3$  identity matrix.

$$P_k = (I - K_k H_k) P_k^- \quad (6.12)$$

The extended Kalman filter equations are summarized in Table 6.13 and 6.14.

**Table 6.3.** Extended Kalman filter time update equations

$$\hat{\mathbf{x}}_k^- = f(\hat{\mathbf{x}}_{k-1}, \hat{u}_{k-1}^-, 0) \quad (6.13a)$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \quad (6.13b)$$

**Table 6.4.** Extended Kalman filter measurement update equations

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \quad (6.14a)$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + K_k (z_k - h(\hat{\mathbf{x}}_k^-, 0)) \quad (6.14b)$$

$$P_k = (I - K_k H_k) P_k^- \quad (6.14c)$$

$$(6.14d)$$

# Chapter 7

## Implementation in C

The BeeBadge is programmed in C-code. During the development of the different algorithms for navigation, only the raw sensor data were printed and later processed in MATLAB. The algorithms developed in MATLAB were translated to C-code so that they could run on the BeeBadge. The transition was fairly simple with respect to how the code should work. The issues that had to be taken into consideration were that the BeeBadge is an embedded system with limited memory and computational powers.

### 7.1 Approach

On the BeeBadge, a hardware interrupt is invoked every tenth millisecond. Based on this interrupt a 50 Hz interval is derived which is used to activate and execute the code created in this project. The 50 Hz interval is arbitrary, a lower rate probably works as well. When a predefined number of samples (called a data set) has been collected, the calculations are started on this data set. A set length of 50 samples have shown to give good results, the need for such a long set of samples to be gathered is due to the step threshold, see Chapter 3.

First the step detection is run and if one or more steps are detected in the data set the information of when they have been detected is saved in an array. Heading is only calculated if one or more steps were detected, one separate heading is calculated for every step. The calculated heading will be input to the Kalman filter.

The result is printed to a serial port on the BeeBadge. This port is connected to a FTDI-chip on the OC8-H that allows communication via USB. Over USB the data is then sent and printed in a terminal on a PC. If it should be desirable to send the result wireless instead this could easily be done.

In MATLAB there is built in security for data, indexing outside an array is for example not possible. In C there is nothing that prevents one from indexing outside an array and read from or write to random places in memory, resulting in

lost data or a crashed program. During the translation to C-code, extra precautions therefore were taken in finding and making sure such errors do not occur. Multiplication of matrices in MATLAB is also easily done and the amount of data one can store is no problem on a PC. These last two examples are not so much limitations in C as in the fact that the C-code in this implementation is run on an embedded system with limited computational power and memory.

Another difference between the C-code implementation and the MATLAB implementation is that in the latter all data is always available because it was read from files saved on a PC and not directly from the BeeBadge. The reason for this structure is that it is desirable to be able to run several experiments on the same set of data. For MATLAB, reading from a file is also the easiest way of retrieving data. A consequence of that data was saved to files was that the code in MATLAB did not need to be adapted to run on sets of data, as is the case on the BeeBadge. The only part in the MATLAB-code where loops around sets of data were necessary was when the step threshold was calculated, see Chapter 3. All the other functions were written and executed as one sequential run.

A drawback when doing calculations on the BeeBadge is that information about the last sample of data from the magnetometer and the accelerometer must be saved between data sets. Consequently step detection must for example have an array for storing the norm that is one index larger than the length of a data set. The reason for this is that both previous and next sample must be checked when detecting a step, see Section 3.3. A subsequent aftereffect is that the syntax of step detection becomes a little bit harder to follow. Of course this also affects the heading calculations which also have to store every data set's last sample to be used in next data set.

## 7.2 Math Functions

During the early implementation in C-code there were problems getting the linking of the math-library correct. This prohibited use of trigonometric functions, and therefore `sin`, `cos`, square root and `atan2` functions were written.

All functions contain a loop in which the calculation is performed, the loop is run a specified number of times. The more times the loop is run the higher precision can be achieved. An iteration of seven laps usually gives a result with enough precision, though this depends on how large the input values are.

These functions were expected to be faster than those from the native C library, the library eventually got linked correctly and the only function that actually was faster was `atan2`. The gain was fairly small and in the new functions no implementations for extreme inputs had yet been made (for example  $Y \rightarrow \infty$ ). The new functions have therefore been put aside in favor of the math library.

Functions for matrix operations were also written, these are more straightforward and self-explanatory than the trigonometric functions. All mathematical functions written can be seen in Appendix A. A look-up table could be used for sine, cos and atan2, this may speed up things but such an implementation has not been prioritized.

## 7.3 Optimization

As the coding proceeded it became clear that calculations took unacceptably long time. In the print-function there is the possibility to also print the time that has passed since the unit was started, see Section 2.1 for more info. This creates an easy way of getting an indication of where time can be saved.

When the sensors are read, the output is interpreted as an unsigned int. If 16 bits are used this means only non-negative numbers between 0 and 65536 can be represented. The output data has to be transformed in to a signed integer so that the sensor's actual output values can be seen, in the case of the magnetometer this is in the range -2048 to 2047. These numbers are naturally not in Gauss but have to be divided by the gain that is set on the magnetometer. The same applies for the accelerometer. This scaling has no real effect on the calculations for heading and Kalman filtration. The only part where the result is affected if scaling is not performed is the threshold in step detection. The scaling has in the conversion between MATLAB and C-code therefore been removed in all other parts of the code.

The Kalman filter is mathematically quite heavy with its matrix operations, see Section 6. To multiply two  $3 \times 3$  matrices requires 18 additions and 27 multiplications, plus the loops that are used for the iteration through the vectors. The matrix calculations were therefore merged to see what elements of the matrices that were needed and if some of the calculations could be skipped. With the implementation of the Kalman filter used in this project it proved that the calculations could be reduced quite a bit. As an example the calculations from Equation (6.4a) ( $P_k^- = AP_{k-1}A^T + Q$ ) in the time update can be used. In this equation there are two multiplications and one addition, all matrices are  $3 \times 3$  elements in size. If they were to be calculated as matrix operations it would result in 54 multiplications and 45 additions. Also, the transpose of the  $A$  matrix must be derived. The only values that are actually needed in  $P_k^-$  matrix lies in the third row, therefore only these are calculated. The result is that only two multiplications and five additions are necessary.

Another great gain in time was that filtration as well as calculation of the norm were moved from the step detection and heading functions to the read function. This does not reduce the total amount of time needed, it only distributes it more evenly so that not all calculations are done at once. The read function is called to read data from the sensors at a ratio of 50 Hz. When a predefined amount of data

has been collected (normally 30 to 50 samples, called a data set) the step detection, heading and Kalman functions are called. Step detection needs the norm to be low pass filtered and the heading function needs both the magnetometer and accelerometer data to be low pass filtered.

The greatest time saver was by far the movement of the filtration and calculation of the norm. After the optimizations had been done the time consumption for running step detection, heading and Kalman filtration once was reduced from approximately 0.1 seconds to 0.025 seconds on a data set of 50 samples. At a sample rate of 50 Hz one sample is taken every 0.02 seconds, if the calculations take 0.025 seconds this means that one sample is lost every time the calculations are executed. This is not an issue as 50 Hz is more than sufficient to detect steps.

# Chapter 8

## Experimental Results

### 8.1 Problems and Verification

The biggest concern with the system is the compensation for hard iron offsets. All magnetometers may have different hard iron offset, but the offset for a single magnetometer should be constant. This means that a derived compensation for a specific magnetometer calibration can always be used. In this project, different tests made at different times, required different hard iron compensation. This goes for all magnetometer axes. In some tests one axis required adjustments with a positive value, and the same axis could in another test need to be compensated with a negative value. The heading angle is very sensitive to bias changes on the magnetometer axes and therefore all axes must be compensated for hard iron offsets. Even small changes can lead to large errors in the heading. With seemingly random variations on the magnetometer axes it is impossible to derive a correct heading. It is however possible to calculate the offset in MATLAB after a data sequence has been collected, but it is not possible to derive it in real time on the BeeBadge. Without reliable hard iron compensation the system will not work as intended. All navigation plots in this chapter are therefore derived in MATLAB from raw data where it was possible to derive and compensate for the offsets for every individual test set. An issue occurs here because using the min and max value for hard iron compensation has some limitations. When using it on long walking experiments instead of on calibration tests it is very likely that the magnetic field at some point will be affected by ferrous or magnetic objects. The compensations that are discussed in this project only compensate for disturbances in a fixed position with respect to the magnetometer. What can happen is that one or more of the magnetometer's axes may be affected so that it reaches a very high max or very low min value, even after filtration has been applied. If these values are averaged, the offset will either be too large or too small. To still be able to hard iron compensate, the mean value can be calculated and used for compensation. Restrictions with this method is that during the experiment a constant pace has to be kept by the person carrying the BeeBadge and the test has to start and end in the same position. A big problem by performing the compensation like this,

is that it also compensates for random noise which otherwise should have been present in the result. This makes the results look much better than they would have if the noise had not been compensated for. The error covariance in the plots therefor gives a more truthful view of the position.

During the calibration testing the soft iron distortions were not considered to give large errors. In Figure 5.3 a hard iron compensated measurement when the magnetometer is turned  $360^\circ$  can be seen compared to a perfect circle. The magnetometer's plot gives almost no elliptic shape, as it would if soft iron distortions were present. This compensation is therefore not used in the final implementation in C, though it can easily be activated if it is deemed necessary.

The final implementation on the BeeBadge has too long internal delay caused by calculations and transmission of data from the BeeBadge. Writing data through a serial port turned out to be very time consuming, in fact it takes much longer to transmit data from the BeeBadge than to run the application. When the internal delay is too big it is not possible to read new data from the sensors and the system gets stalled. This ends up as time gaps between different data sets where samples are lost. Even though after the implementation had been verified and a lot of previously printed information removed, it still takes a bit too long time to write. Time gaps in the sensor readings still occur between data sets. The time it takes to execute all the calculations has been greatly reduced by optimizations of the code, but another way to transmit the result from the BeeBadge is needed in order not to risk missing data.

During verification of the C code the three major parts of the system, step detection, heading and the Kalman filter, needed to be confirmed and compared to results of the corresponding calculations in MATLAB. For each part of the system different variables was stored on the BeeBadge and printed. They were then compared with the same values calculated in MATLAB to make sure everything worked as intended. This of course made the BeeBadge process very time consuming, all the transmission of data introduced big time gaps between the sensor readings. This was specially noted in the step detection where it could take very long time between two detected steps. For the verification there was actually not a problem to have time gaps and missing sensor data. As long as both the MATLAB and BeeBadge calculation gives the same results the functionality is assured. Every read sample from the sensors was printed together with the data that was calculated on the BeeBadge. To be able to make precise verifications, every time a step was detected on the BeeBadge, the total sample count was saved. This was later compared to when steps were detected in MATLAB. This made a great base for verifying all the parts of the system, as both the heading and the Kalman functions are based on the event of a step. Because of the problems with the hard iron compensation of the magnetometer this part had to be removed from both the BeeBadge and MATLAB.



## 8.2 Results

The goal was to make a system which could keep track of a person by analyzing movements and the relation to Earth's magnetic field. A lot of different tests have been made both indoors and outdoors to verify the functionality of the system. Step detection has been a very important part for the system to actually work and has proved to be quite precise. Table 8.1 contains statistics from different tests where the BeeBadge has been equipped on the right hip. The reference number which the derived step count is compared to has been counted by two persons while one of the persons has been wearing the device. The counted step numbers are believed to be fairly precise but might of course contain some small errors.

**Table 8.1.** Step detection Statistics

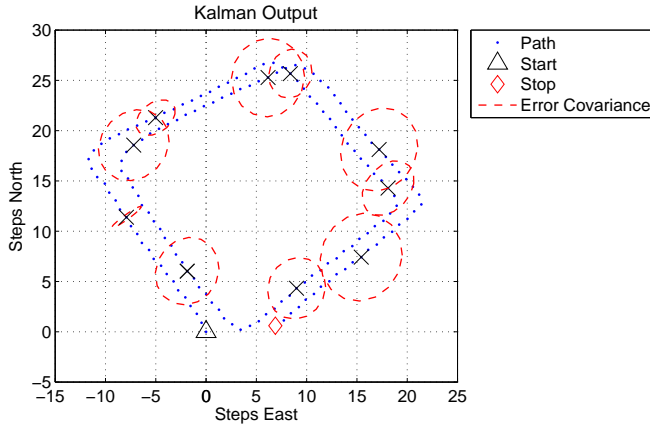
Step Count	BeeBadge Steps Count	Step Difference	Difference %
1195	1209	14	1.17
174	169	-5	-2.87
163	160	-3	-1.84
163	160	-3	-1.84
159	161	2	1.25
158	154	-4	-2.25

### *Parameter values*

Threshold:           Max and Min Average  
Sample Rate:        50 Hz  
Filter value ( $\beta$ ):   0.96

The final output of the filter ( $\hat{x}$ ) contains the derived coordinates of a detected step. Figure 8.1 shows an example of an output sequence. The test used in this figure is the same as the one used to illustrate tilt-compensated heading in Figure 4.5. A person wearing the BeeBadge in the belt has walked 160 steps in a  $20 \times 20$  steps square pattern. Beginning at the coordinate (0,0), the figure shows all detected steps on their respective coordinate. Every 15th step the error covariance from the Kalman filter has been plotted as an ellipse, a cross has also been plotted to clarify which step every circle belongs to. It can be seen that the uncertainty (ellipse size) of the current position grows with every new step, which is expected.

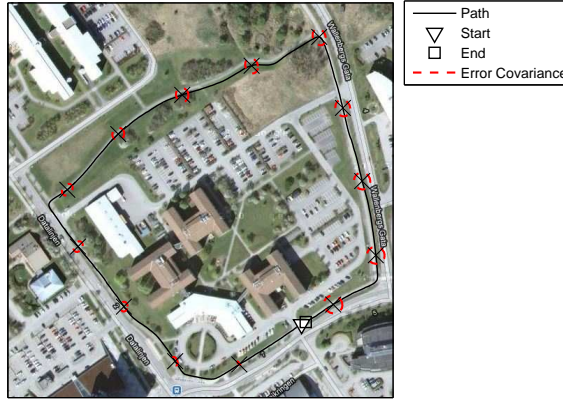
Figure 8.2(a) and 8.3 shows two different test sets which have been taken outdoors. In the figures the result of the tests are plotted on maps of the location where the tests were performed. During the two test sets two different persons have been wearing the equipment. This was done to gain some confidence in how the system works under different conditions, for example different walking styles. In both tests a person have started off from a certain location and afterwards returned to the exact same location. This was done in order to see how much the



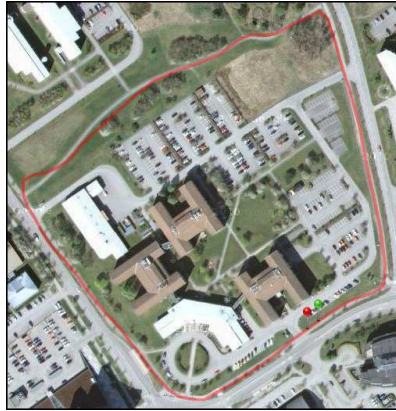
**Figure 8.1.** Picture of the output coordinates from the Kalman filter. The error covariance is plotted every 15th step, marked with a cross. A person has walked 160 steps in a square  $20 \times 20$  steps big. The BeeBadge was worn in the belt.

system would drift. The plots shows that the algorithms for positioning works, the solid line represents the walked path, which quite well follows the walked path. The start positions are marked by a triangle and end positions by a rectangle. As can be seen in both pictures the drift is quite small, the start and end coordinates are located fairly close to each other in both tests. The fact that the step length is fixed reduces the reliability of the calculated coordinates. In Figure 8.2(a) the calculations were done in MATLAB. When the resulting coordinates were plotted on the map, the trajectory had the correct shape, but not the correct size. Of course this also has to do with that the map is in an unknown scale. For the trajectory to fit the roads on the map the coordinates were scaled. Problems may occur if a person who carries the BeeBadge changes between long and short steps during a measurement. The persons performing the tests have been walking with a quite even step length and this makes the result better than it would have been if the step length would have varied more. The plots have been scaled to give a realistic feeling how it would look on a map. The test in Figure 8.2(a) was also accompanied by a GPS on a LG Optimus X2 and the route collected by the GPS can be seen in Figure 8.2(b). The test run in Figure 8.2(a) counted 1195 steps and the distance measured by the GPS was 910 meters. In the test set roads and cycle paths were followed. When comparing 8.2(a) and the GPS route in Figure 8.2(b), it can be seen that the GPS route is very similar to the Kalman filter output. There are some differences and for example in the second right turn the GPS shows to perform worse and overshoots the turn. This might be because some GPS-systems uses a method that alters the coordinates if a road is close and in this scenario a main road was followed just before the second right turn.

The developed system has shown to perform well when used outdoors. The tests that have been made indoors do not give as good results and this is because indoor



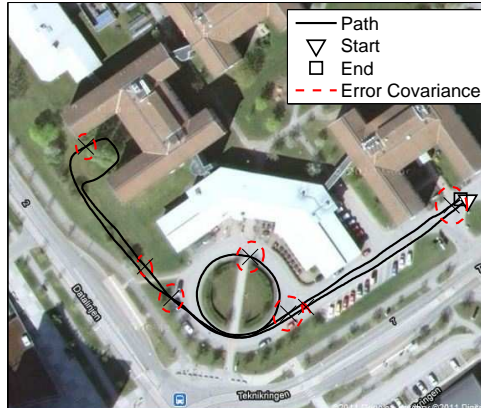
(a) BeeBadge's trajectory



(b) GPS trajectory

**Figure 8.2.** The upper figure shows the result from the Kalman filter and the lower figure shows the GPS route from the same test. The backgrounds are taken from Google maps.

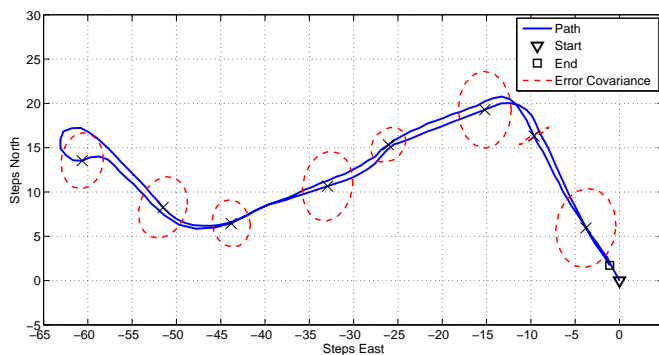
environments often contain more magnetic disturbances. Tests have shown that the possibilities to use this implementation for indoor navigation are more slim than for outdoor navigation. Figure 8.4(a) and 8.4(b) show two plots of more successful tests which have been done in an indoor environment. In Figure 8.4(a) a person has been walking straight forward, then making a  $90^\circ$  left turn, heading on straight forward to make a  $90^\circ$  right turn and then walk back in exactly same path. This particular test was actually quite successful and the start and end position ends up almost at the same location. This set contained 174 steps which is a bit shorter than the earlier described outdoor examples. The path walked in second indoor test was walked in the shape of a rectangle with  $90^\circ$  angles with a total step count of 108 steps, Figure 8.4(b). Even though the start and end position end up close to each other it can clearly be seen that the path is quite



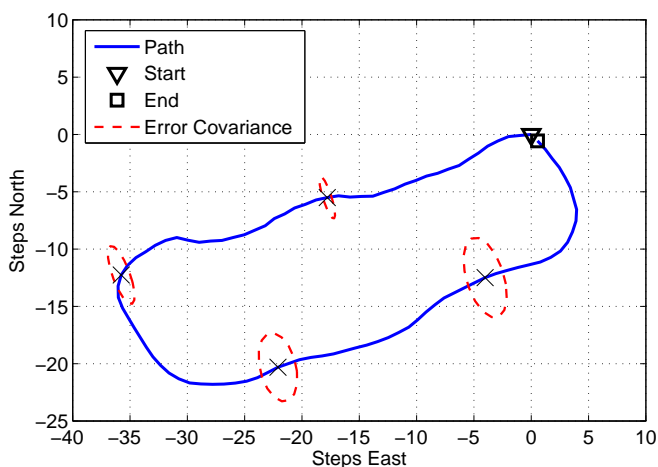
**Figure 8.3.** This figure illustrates the result from the Kalman filter from an outdoor test with a length of 640 steps. The background is taken from Google maps.

unstable. If the system begins to drift and loses its path there is nothing but luck that can put it back on track again. Further implementations could be made to improve the results for both outdoor and indoor usage. A backup GPS could in outdoor environments function as a reference and would not need to be updated very often to get good knowledge of the current position. Information about the current location could also be further improved with time of flight (ToF). ToF uses the signal strength or the time it takes for a radio signal to travel from sender to receiver in a wireless network. With this the distance can be calculated. ToF would be a much better choice than GPS for indoor environments.

As the Kalman filter is implemented now it does not have very much to work with. The only input used is currently the derived heading angle. It can be discussed if the result from the filter actually adds any improvements to the system compared to not using it at all. It is possible to get good results with a low pass filter and this with less calculative heavy operations. If there were to be more information about the current state of the system, the Kalman filter would have more to work with and this could possibly improve the result greatly. The filter can easily be extended to handle more inputs and if for example ToF or a GPS were to be added, much more reliable estimations and corrections would be available. The noise parameters have shown to sometimes generate bad results in some tests, while the same parameters have generated more impressive results in other tests. Low pass filtering has proved to be much more stable in that matter.



(a) Indoor test 1



(b) Indoor test 2

**Figure 8.4.** The derived trajectories from the Kalman filter of two different test sets made indoors. The upper figure contained 174 steps, all turns made in the test where  $90^\circ$ . The second figure contained 108 steps and was walked in the pattern of a square.



# Chapter 9

## Concluding Remarks

### 9.1 Conclusions

The whole project was implemented and simulated in MATLAB on a PC. This offered more than enough computer power for calculations and memory for storing variables and information. Working with MATLAB also meant simple file structures, big libraries of mathematical functions and handy debugging tools. The hardware implementation in C required some extra thought because of limited debugging tools, small memory and low processor power. Compared to the real implementation in C, MATLAB was very fast and easy to work with. The approach to first implement and simulate the functionality of the system in MATLAB made the C implementation tremendously easier.

One of the key problems that has to be solved to be able to create a navigation system is the calibration of the magnetometer. Without a good calibration it is impossible to get correct headings. During the work on this project no correct calibration could be achieved. When calibrations were done on whole sets of data good trajectories could be plotted. The algorithms created can therefore be said to be working, see Chapter 8.

The coordinates calculated on the BeeBadge were printed to a serial port, this takes a bit too long time, see Chapter 8. If the output from the BeeBadge were sent by the built in radio to another BeeBadge for example and printed there, this problem might be avoided.

### 9.2 Future Work

During the simulation a feature was added that calculates the time while a person is considered not to be moving around. It starts to count if no step is detected within one second. When the next step is detected the timer stops. This can be used to determine how long a person has been standing still. A more sophisticated

implementation could for example also keep track of certain areas where a person have been and for how long.

There are possibilities to add support for different kinds of motion tracking. The system could be extended to sense how its wearer is moving, for example if the wearer is walking, running, riding a bike or driving a car. In this area there are a lot of possibilities to work with.

The navigation could be improved greatly with support from additional sensors. Implementation and integration of time of flight and GPS would together with the Kalman filter be a very interesting system.

The hardware provides quite limited possibilities for heavy calculations, more efficient and less time consuming algorithms and methods could be implemented to improve the execution time. A system like this runs on a limited battery and it is possible that accuracy could be sacrificed to improve both speed and battery time while still maintaining acceptable results. In this project the sampling rate has been 50 Hz, it can probably be reduced quite a bit while still producing good results. Lookup tables could probably be used in many of the calculations to speed up the process. It is possible to explore how much time that can be saved by reducing the number of possible values an angle can be represented with. Can the system work with an angle interval of five degrees without losing precision?



# Bibliography

- [1] Set/reset function for magnetic sensors. Technical report, Solid State Electronics Center, 2008.
- [2] G. Bishop and G. Welch. An introduction to the kalman filter. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 2006.
- [3] M. J. Caruso and L. S. Withanawasam. Vehicle detection and compass applications using amr magnetic sensors. Technical report, Honeywell SSEC, 1999.
- [4] National Geophysical Data Center. Declination calculator, 05 2011.
- [5] Honeywell International Inc. *3-Axis Digital Compass IC HMC5883*, form 900405 rev a edition, 2010.
- [6] Honeywell International Inc. *Compass Heading Using Magnetometers*, 7/95 rev. a edition.
- [7] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME - Journal of Basic Engineering*, 82(1):35–45, 1960.
- [8] C. Konvalin. Compensating for tilt, hard iron and soft iron effects. Technical report, MEMSense, 2008.
- [9] N. Zhao. Fill-featured pedometer design realized with 3-axis digital accelerometer. *Analog Dialogue*, 44(06), 2010.



# Appendix A

## Trigonometric Functions

```
float sqrtx(float x, int Precision) {
    double sqrtx = x/2;
    int i = 0;
    for(i = 0 ; i < Precision ; i ++) {
        sqrtx = (sqrtx + x/sqrtx)/2;
    }
    return sqrtx;
}
```

```
float sinx(float x, const int N) {
    if( x > 2*PI) {
        while (x > 2*PI) {
            x -= 2*PI;
        }
    }
    else if (x < -2*PI){
        while(x < -2*PI) {
            x += 2*PI;
        }
    }

    int Sign = 1;
    double Numerator = x;
    double Denominator = 1;
    double sinx = 0;
    int i = 1;

    for (i = 1; i <= N ; i++) {
        sinx = sinx + Sign*(Numerator/Denominator);
```

```

        Numerator = Numerator*x*x;
        Denominator = Denominator*(2*i)*(2*i+1);
        Sign *= -1;
    }
    return sinx;
}

float cosx(float x, const int N){
    if( x > 2*PI) {
        while (x > 2*PI) {
            x -= 2*PI;
        }
    }
    else if (x < -2*PI){
        while(x < -2*PI){
            x += 2*PI;
        }
    }

    int Sign = 1;
    double Numerator = 1;
    double Denominator = 1;
    double cosx = 0;
    int i = 1;

    for (i = 1; i <= N ; i++) {
        cosx = cosx + Sign*(Numerator/Denominator);
        Numerator = Numerator*x*x;
        Denominator = Denominator*(2*i)*(2*i-1);
        Sign *= -1;
    }
    return cosx;
}

float atan2x(float y, float x, const int N){
    double Angle = 0;
    if (x < y) {
        x = -x;
        y = -y;
        Angle = PI;
    }

```

---

```

    if (x < -y){
        double tmp = x;
        x = -y;
        y = tmp;
    }

    double z = y/x;
    int Sign = 1;
    double Numerator = z;
    double Denominator = 1;
    int i = 1;

    for(i = 1 ; i <= N ; i++) {
        Angle += Sign*(Numerator/Denominator);
        Numerator = Numerator*z*z;
        Denominator = (2*i+1);
    }
    return Angle;
}

```

```

/*Make transpose of matrix b, result in a*/
void matrix_transp(float a[3][3], float b[3][3]) {
    int i,j;

    for(i=0;i<=2;i++) {
        for(j=i;j<=2;j++) {
            a[i][j] = b[j][i];
            a[j][i] = b[i][j];
        }
    }
}

```

```

/*Copy content of matrix b in to matrix a*/
void matrix_cpy(float a[3][3], float b[3][3]) {
    int i, j;
    for(i=0; i<=2; i++) {
        for(j=0; i<=2; i++) {
            a[i][j] = b[i][j];
        }
    }
}

```

```
/*Add matrix b & c, answer in a*/
void matrix_add(float a[3][3], float b[3][3], float c[3][3]) {
    int i,j;
    for(i=0; i<=2; i++) {
        for(j=0; j<=2; j++) {
            a[i][j] = b[i][j] + c[i][j];
        }
    }
}
```