Universidad de Granada

Fachhochschule Münster University of Applied Sciences

MÜNSTER UNIVERSITY OF APPLIED SCIENCES
Department of Electrical Engineering and Computer Science

Bachelor Thesis

KALMAN FILTERING APPLIED TO ORIENTATION ESTIMATION IN HUMAN BODY MOTION ANALYSIS

Author:
Robin Weiß

Supervisors:
Prof. Dr.-Ing. Peter Glösekötter
Ph.D. Alberto Olivares Vicente
Prof. Dr. med. Kai Bötzel

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Electrical Engineering

May 2015

STATEMENT OF AUTHORSHIP

I hereby certify that this bachelor thesis has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged. It has not been accepted in any previous application for a degree.

Granada, 15th May 2015

Robin Weiß

PREFACE

This thesis was submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Electrical Engineering. It describes the implementation of a new Kalman filter based orientation algorithm to improve the estimation of orientation angles by means of inertial sensors.

I took part in the joint research project "Human Body Motion Analysis of Patients with Neurodegenerative Diseases by Means of Inertial Sensors" between the Research Centre for Information and Communications Technologies of the University of Granada (CITIC-UGR), Spain, and the Department of Neurology of the Klinikum Großhadern, which is part of the Ludwig Maximilian University of Munich, Germany. The goal of the overall project was to obtain several gait parameters by wearable inertial sensors and validate them against conventional methods, such as cameras in combination with visual markers and force measuring platforms. Physicians and medical researchers are interested in this approach of body motion analysis, as unobtrusive wearable sensors can assist the diagnosis of neurodegenerative diseases, such as, for instance Parkinson's disease. Prior to this thesis I completed a three-months internship at the CITIC-UGR in which I worked on the synchronisation of a force measuring platform with inertial sensors within the above-mentioned project.

ABSTRACT

Along the course of this thesis

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ARW	Angle random walk
CITIC-UGR	Research Centre for Information and Communications Technologies of the University of Granada
EKF	Extended Kalman filter
IMU	Inertial measurement unit
LTSD	Long term spectral detector
MARG sensors	Magnetic, angular rate, and gravity sensors
MEMS	Microelectromechanical systems
MIMU	Magnetic inertial measurement unit
NED	North-east-down
RMSE	Root-mean-square error

NOTATION

a	Amplitude of an oscillating mode
$\mathbf{a}$	Acceleration vector, $\mathbf{a} \in \mathbb{R}^3$
$\mathbf{B}$	Control matrix that relates the control input to the state $\mathbf{x}$
$\mathbf{C}_{bw}$	Transformation matrix transforming a position vector from the body frame to the world frame
$\mathbf{C}_{wb}$	Transformation matrix transforming a position vector from the world frame to the body frame
d	Desired filter response of a linear discrete-time filter
D	Damping coefficient
e	Error signal of a linear discrete-time filter
$\mathbf{E}$	Orthonormal basis $\{x, y, z\} \in \mathbb{R}^3$
$\mathbf{f}$	Force vector, $\mathbf{f} \in \mathbb{R}^3$
F	One-dimensional force
$\mathbf{g}$	Gravity vector
$\|\|\mathbf{g}\|\|$	Magnitude of the gravity vector
$h_0, h_1, h_2, \ldots$	Impulse response of a linear discrete-time filter
$\mathbf{h}$	Functional denoting the *non-linear* measurement matrix function of a discrete dynamical system
$\mathbf{H}$	Measurement sensitivity matrix defining the linear relationship between the state of the dynamical system and the measurements that can be made

$\mathbf{H}^{[1]}$	Linear approximation of the measurement sensitivity matrix
$\mathbf{I}_n$	Identity matrix $\mathbf{I}_n \in \mathbb{R}^n$
k	Discrete time normalised to sampling interval, that is sample number, $k \in \mathbb{N}^0$
k_x	Spring constant along the x-axis
K	Weighting factor
$\mathbf{K}$	Kalman gain matrix
m	Mass
μ	Mean value of a conditional probability density
n	Fixed discrete time, non-dimensional number, or number of iterations applied to recursive algorithms
ω	Scalar angular velocity
$\boldsymbol{\omega}$	Angular velocity, $\boldsymbol{\omega} \in \mathbb{R}^3$
Ω	Resonance frequency
$\boldsymbol{\Omega}_{\mathbf{E}\to\mathbf{E}'}$	Function that transforms a position vector $\mathbf{p}$ in the vector space $\mathbf{E}$ into the vector $\mathbf{p}^{'}$ in the vector space $\mathbf{E}^{'}$
$\mathbf{p}$	Position vector in a three-dimensional vector space
$\mathbf{P}$	Covariance matrix of state estimation uncertainty
ϕ	Roll angle that determines the rotation around the x-axis
$\boldsymbol{\phi}$	Functional denoting the *non-linear* transition matrix function of a discrete dynamical system
$\boldsymbol{\Phi}$	State transition matrix of a discrete linear dynamical system
$\boldsymbol{\Phi}^{[1]}$	Linear approximation of the state transition matrix of a discrete linear dynamical system
ψ	Yaw angle that determines the rotation around the z-axis
$\mathbf{Q}$	Covariance matrix of process noise in the system state dynamics

$\mathbf{R}$	Covariance matrix of measurement noise
σ	Standard deviation of a random variable
σ^2	Variance of a random variable
t	Continuous time
$\mathbf{T}$	Transformation matrix
θ	Pitch angle that determines the rotation around the y-axis
u	Scalar nominal velocity
$\mathbf{v}$	Measurement noise vector, $\mathbf{v} \in \mathbb{R}^m$; or mass velocity, $\mathbf{v} \in \mathbb{R}^3$
w	Scalar noise term
$\mathbf{w}$	Process noise vector
$\mathbf{w} \sim \mathcal{N}(\mu, \sigma^2)$	The random variable $\mathbf{w}$ is distributed normally with mean μ and variance σ^2
x	One-dimensional location
$\hat{x}$	Estimate of x
Δx	One-dimensional displacement in x-direction
x, y, z	Axes of the fixed world frame
X, Y, Z	Axes of the moving body frame
$\mathbf{x}$	State vector of a linear dynamical system
$\mathbf{x}_k$	The kth element of a sequence $\ldots, \mathbf{x}_{k-1}, \mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$ of vectors
$\hat{\mathbf{x}}$	Estimate of the state vector of a linear dynamical system
$\hat{\mathbf{x}}_k^-$	A priori estimate of $\hat{\mathbf{x}}$, conditioned on all prior measurements except the one at time t_k
$z(0), z(1), z(2), \ldots$	Time series that serves as input to a linear discrete-time filter

z^{-1}	Unit-delay
$\mathbf{z}$	Observation or measurement vector of a dynamical system
$\hat{\mathbf{z}}$	Prediction of the measurement vector

1 INTRODUCTION

Monitoring and assessment of human body motion, in particular the analysis of gait, has become an integral part of medical diagnosis, therapy techniques, and rehabilitation [1]. *Gait analysis* involves the measurement and assessment of quantitative parameters that characterise human locomotion. First research in this field was conducted in the late 19th century [1]. The quantitative data enable physicians to diagnose a variety of medical conditions, validate treatment success, set goals in rehabilitation and regularly alter them when necessary. However, standard gait analysis based on multi-camera motion capture systems and force measuring platforms require specialised gait laboratories, expensive equipment, and lengthy setup times. Moreover, the assessments of gait based on measurements performed in clinical settings might not be truly representative [2].

Unobtrusive wearable sensors mitigate the aforementioned limitations. The progressive miniaturisation of inertial and magnetic field sensors has made them more acceptable to patients and has consequently lead to an increasingly pervasive adoption for medical applications [3]. Low cost sensors have been successfully employed in clinical and home environments to constantly monitor the movements of patients [4]. Additionally, wearable inertial and magnetic sensors are used to capture gait kinematics, among others. *Kinematics* is a branch of classical mechanics, which is concerned with the motion of objects without reference to the forces causing the motion. The position, i. e. the orientation, the velocity, and the acceleration of a body are of particular interest in kinematics. All three can be estimated from inertial data.

The orientation of the legs is essential in gait analysis. For the application in health care accurate orientation estimates are crucial. A high degree of precision based on data from miniaturised sensors necessitates adequate signal processing, in order to mitigate the influence of disruptive factors, such as bias instability and noise, among others. The signal processing of inertial and magnetic data encompasses calibration, adaptive filtering, and sensor fusion. The latter two were carried out in an extended Kalman filter in the course of this thesis.

1.1 MOTIVATION

Gait analysis provides a powerful means to derive diagnostic information about the functioning of the musculoskeletal, vestibular, and central and peripheral nervous system [5]. Accurate orientation estimation of the extremities by means of wearable inertial and magnetic field sensors allows objective assessment of human gait without the aforementioned constraints of camera-based motion capture systems. A more reliable and more precise orientation estimation would enable an even more informative gait analysis. Therefore, a multitude of applications in the medical field would profit from a more accurate orientation estimation. [6]. The direct relation to health care and the resulting possibility to improve the quality of life of many patients was the motivation for this thesis.

1.2 GOALS

A system for human body motion analysis based on wearable sensors had been developed earlier in order to gather and process movement data of patients. A detailed description of the so-called GaitWatch system is found in Section 4.1. The goal of this thesis was implementing a new Kalman filter based orientation algorithm proposed by Bennett et al. in [7], in order to improve the estimation of the orientation angles of the human leg. After the adaptation of the proposed mathematical model of the leg and the extended Kalman filter to the existing system, the algorithm should be implemented using the numerical computing environment MATLAB®. Subsequently, the results should be validated against existing algorithms by comparing their respective root-mean-square error. Other than in [7], the filter algorithms should be applied to data of real patients for testing.

1.3 METHODOLOGY

This document presents my work within the overall project in a chronological order. Subsequent to the previous introductory overview of the topic and the definition of the project objectives, this chapter ends with a description of the state of the art. To accomplish the tasks defined in the previous section, I had to acquire knowledge regarding various subjects. Chapters 2 and 3 outline the necessary fundamentals of MARG sensors and orientation estimation, and digital filters, respectively. This enables comprehension of the overall project, even for readers that are not familiar with some of the subjects. Those readers are referred to Chapters 2 and 3 at this point, before reading the state of the art. The actual implementation of the Kalman filter, including a prior theoretical design is given in Chapter 4. This chapter also encompasses the experimental setup, the results and a discussion of the latter. Finally, Chapter 5 covers conclusions and future work.

As additional means to communicate with my supervisor and in order to enable him to follow the progress of my work at any time we used Pivotal Tracker, a tool for agile project management, and GitHub, a repository hosting service based on the distributed version control system Git. This thesis was written in LaTeX.

1.4 STATE OF THE ART

There are several research works in the literature dealing with orientation estimation by means of inertial sensors. Kalman filters have been used successfully to improve the estimation of orientation angles from inertial data. The state of the art at the commencement of the project is described below. Subsequently, applications of wearable inertial sensors in health care are presented.

1.4.1 *Kalman Filtering Applied to Orientation Estimation*

Considering the fact that inertial and magnetic field sensors are used to establish objective body motion parameters that affect medical diagnosis, therapy, and rehabilitation, the necessity of high levels of accuracy becomes obvious. In order to obtain precise orientation estimates from sensor data it is essential to mitigate the effects of measurement noise and to combine the advantages of different sensors through sensor fusion. Therefore, a wide variety of Kalman filter algorithms have been developed in the past few years. It is common practice to fuse accelerometer and gyroscope measurements to mitigate their respective drawbacks and thus obtain more accurate angle estimates.

Luinge et al. [8] alleged that the gravitational component of the acceleration signal has a greater magnitude than the component caused by motion for many human movements. They estimated the tilt angle, which is defined as the angle between the sensor axes and the vertical. The separate estimates from an accelerometer and a gyroscopes were fused with a Kalman filter. To test their method they moved the sensors around by hand for 30 seconds and then put it in a known position. The orientation obtained by integrating the angular rate only served as an additional reference. They concluded that a fusion of accelerometer and gyroscope signals accounts for a considerable improvement of the orientation estimation. This approach lacks of dynamical comparison, since it only compares the errors at specific static positions.

Due to human motion intensity usually being subject to change, Olivares Vicente implemented a *gated Kalman filter* in [9]. They modelled linear acceleration during intense motion as noise and improved the performance of the Kalman filter by dynamically adjusting the variance of both the process and measurement noise, according to the motion intensity. Therefore, they applied a long term spectral detector (LTSD) and set the variance between two predefined values. Then, the gated Kalman filter fused the information from the accelerometer and the gyroscope signals. With this method they improved the adapting capability of the filter and consequently the precision of the orientation estimation.

Bennett et al. demonstrated in [7] that accelerometer angle estimates are inaccurate for typical motions of the leg. They affirmed the need to decouple the acceleration due to motion from the acceleration due to gravity, since the former cannot be neglected during fast motions. Therefore, they deployed a kinematic model of the leg to determine the acceleration that occurs due to motion and corrected the acceleration signal accordingly. An extended Kalman filter fused the corrected acceleration data with measurements of a gyroscope. This method improved upon the raw acceleration method during motion and at rest by an 83% smaller root-mean-square error. Their proposed approach is the foundation of the filter algorithm implemented in Section 4.2.

1.4.2 *Wearable Sensors in Health Care*

Inertial sensors can be found in smart phones, fitness trackers, and other wearable devices, among others. With increasing capability of body sensor networks and wearable computing, they have become prevalent in research environments for estimation and tracking of human body motion [7]. They are used in activity monitoring [10–12], rehabilitation [13, 14], sports training [12, 15], and localisation [16, 17]. Also, emergency falls of elderly people were detected by means of inertial sensors [18–20].

Many neurodegenerative diseases such as, for instance, Parkinson's disease, impair stable stance and gait and reduce the patient's mobility. Thus, they diminish the quality of life significantly. *Parkinson's disease* is a movement disorder that is characterised by marked slow movements, tremors, and unstable posture. Especially in advanced stages of the disease many patients exhibit an episodic, brief inability to step, which delays gait initiation or interrupts ongoing gait. In fact, one of the most reliable diagnostic criterion of the disease is gait [1]. Hence, wearable motion sensors have been used successfully to classify the severity of the disease objectively [21–23].

Stroke patients, who regained their walking ability, need to undergo rehabilitation to recover their independent mobility. Ambulatory gait analysis provides a means to assess the function of the lower extremities of hemiparetic post-stroke patients and follow the progress of rehabilitation [1, 24]. In addition, the presence of neurologic gait abnormalities is used as a significant predictor of the risk of developing dementia [25].

2 ORIENTATION ESTIMATION USING MARG SENSORS

This chapter covers the working principals of MARG sensors as well as the fundamentals of orientation estimation that are necessary for the implementation of the aforementioned system. Subsequently a mathematical construct used to express orientation – Euler angles – are explained. Towards the end of the chapter, different approaches to compute orientation estimates from magnetic and inertial data including their pros and cons are described. Finally sensor fusion as a means to mitigate the drawbacks of each approach is introduced.

2.1 MARG SENSORS

MARG sensors is a collective term for magnetic, angular rate, and gravitational sensors, which encompasses inertial sensors, as well as magnetic field sensors, also referred to as magnetometers. Inertial sensors itself generally fall into two categories: instruments sensing linear inertial displacement, i. e. accelerometers, and rotational inertial rate sensors, that is gyroscopes. They are applied in various contexts to quantify vibration, motion, and shock [26]. Particularly, the development of microelectromechanical systems (MEMS) opened up many medical applications as stated in Section 1.4.2. MEMS sensors have low manufacturing costs, small physical size, and low power consumption [26]. This section compiles the functional principles of different MARG sensors and introduces inertial measurement units (IMUs) as a combination of those. At the end of the section the aforementioned GaitWatch system that was used to gather the movement data for the experiments is described in detail.

2.1.1 *Accelerometers*

Accelerometers measure the acceleration of an object relative to an inertial frame. Since acceleration cannot be sensed directly, the force exerted

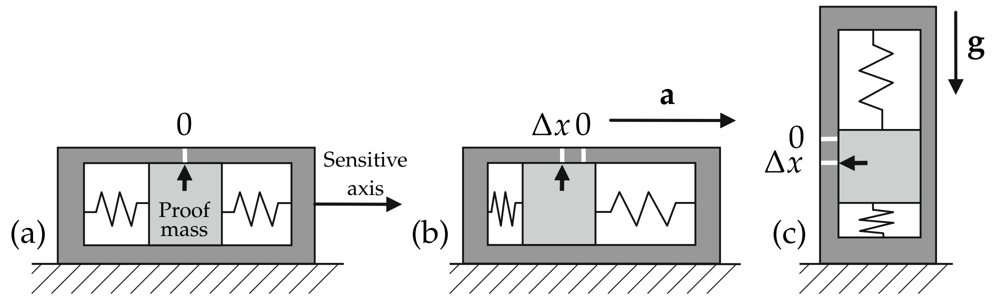

Figure 2.1: A mass-and-spring accelerometer under different conditions: (a) at rest or in uniform motion, (b) accelerating, and (c) at rest, being exposed to the gravity **g**, from [26].

on a reference mass is measured. The resultant acceleration is computed according to Newton's second law $\mathbf{f} = m \cdot \mathbf{a}$, where $\mathbf{f} \in \mathbb{R}^3$ denotes the force vector, m the mass, and $\mathbf{a} \in \mathbb{R}^3$ the acceleration vector. Usually, an accelerometer consists of a small proof mass connected via a spring to the case of the instrument. The proof mass is displaced by Δx with respect to the case, when the instrument experiences a certain acceleration along its sensitive axis. Disregarding drag force, the displacement is directly proportional to the force exerted by the mass and thus to the acceleration. Therefore, by measuring the displacement of the proof mass the acceleration can be obtained. Figure 2.1 shows the displacement Δx of the mass in respect to the case of the instrument for three different conditions: (a) at rest or in uniform motion, (b) accelerating, and (c) at rest, being exposed to the gravity **g**. According to how the mass displacement is sensed, accelerometers can be classified as resistive, capacitive, and piezoelectric. Besides, there are surface acoustic wave, fibre optic, vibrating beam and solid-state MEMS accelerometers. To obtain a three-dimensional accelerometer, three single-axis accelerometers are mounted together. Nowadays most accelerometers are manufactured using MEMS technology, which was developed for the military and aerospace markets in the 1970s [26].

2.1.2 *Gyroscopes*

Gyroscopes are used for measuring and maintaining angular orientation. In essence, based on two different physical principles, namely the Sagnac and Coriolis effect, gyroscopes sense angular velocity, which is why they are also referred to as angular velocity sensors or angular rate sensors. By integrating the angular velocity the rotation angle can obtained. Here we will only elaborate on the working principle of vibrating gyroscopes, since they are utilised in the GaitWatch system. Armenise et al. give a comprehensive overview of current gyroscope technologies in [27].

Coriolis vibratory gyroscopes, or vibrating gyros for short, sense angular velocity based on the effect of Coriolis force on a vibrating mass. The Coriolis force is a fictitious force experienced by a mass m moving in a rotating reference frame. It can be calculated as: $\mathbf{f}_C = -2m(\boldsymbol{\omega} \times \mathbf{v})$, where $\mathbf{v}$ is the mass velocity in the rotating reference frame and $\boldsymbol{\omega}$ is the angular velocity of the reference frame. As seen in this equation the Coriolis force is only present when the mass varies its distance with respect to the spin axis. Otherwise, if $\boldsymbol{\omega}$ and $\mathbf{v}$ are parallel, the cross product becomes zero. The two degree-of-freedom spring-mass-damper system shown in Figure 2.2 serves as a simple model of a vibrating angular rate sensor. The mass m can move along the x and y-axis, respectively. The angular velocity around the z-axis is denoted with ω. The drive or primary oscillating mode, that is, the oscillation along x, is excited by the force F_x directed along the x-axis. The oscillation along y, called sense or secondary oscillating mode, is due to system rotation around the z-axis. D_x and D_y are the damping coefficients and k_x and k_y are the spring constants along the x and y-axis, respectively. Typically, the primary oscillating mode is excited by a sinusoidal force with an angular frequency close to the resonance frequency, so that $\Omega_x \cong \sqrt{k_x/m}$. Its amplitude is kept constant at a_x. As shown in [27],the amplitude of the sense mode is then given by

$$a_y = -\frac{2a_x\Omega_x\omega}{\sqrt{(\Omega_x^2 - \Omega_y^2)^2 + \Omega_x^2\Omega_y^2/Q_y^2}}, \tag{2.1}$$

where $\Omega_y = \sqrt{k_y/m}$ is the resonance frequency of the secondary resonator and $Q_y = \sqrt{mk_y}/D_y$ its quality factor. The amplitude a_y is directly proportional to the angular rate of the two degree-of-freedom spring-mass-damper system. Thus, ω can be estimated by measuring the amplitude of the oscillation along the y-axis.

Usually, vibrating gyroscopes are manufactured using MEMS technology. MEMS gyros are of low to medium accuracy [26], but due to their size they are ideally suited for medical applications.

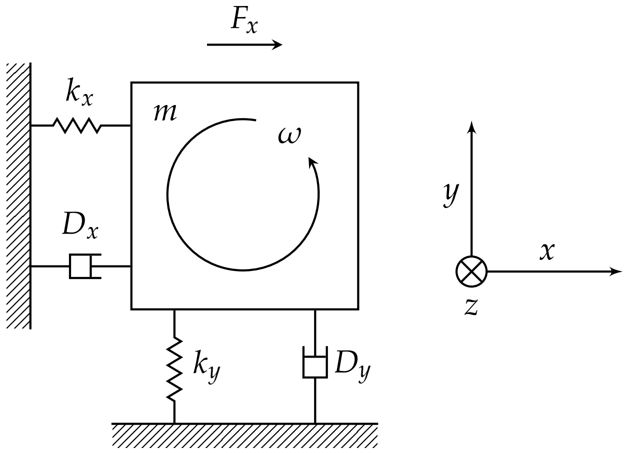

Figure 2.2: A simple model of a Coriolis vibratory gyroscope: A two degree-of-freedom spring-mass-damper system in a rotating reference frame, from [27].

2.1.3 *Magnetometers*

Magnetometers measure the strength and the direction of the magnetic field at a point in space. There are numerous techniques used to produce magnetic field sensors, which exploit a broad range of physical phenomena [28]. Lenz and Edelstein give a complete survey of common technologies used for magnetic field sensing in [28]. Many MEMS magnetometers sense mechanical motion of a MEMS structure due to Lorentz force and estimate the strength of the magnetic field according to the displacement. When an external magnetic field interacts with a current-carrying silicon MEMS structure the Lorentz force causes a displacement of this structure. Piezoresistive, capacitive, or optical sensing can be used to detect the displacement of the MEMS structure. MEMS Lorentz force magnetometers are free from hysteresis, require no specialised materials and can be monolithically integrated with other MEMS inertial sensors [29].

2.2 INERTIAL MEASUREMENT UNITS

Devices using a combination of accelerometers and gyroscopes to measure the orientation of a rigid body with up to six degrees of freedom are referred to as IMUs. If they include additional magnetometers they are termed magnetic inertial measurement units (MIMUs). The number of degrees of freedom states the number of independent motions, with respect to a reference frame, that are allowed to the body in space. MIMUs are portable and relatively inexpensive. They can be easily attached to the body and thus allow non-clinical longterm application. Their drawbacks are complex calibration procedures and drift behaviour over time, depending on intensity and duration of the measurement interval. Hence, in order to maintain a satisfactory degree of precision, periodical recomputation of the calibration parameters is required [9].

2.3 EULER ANGLES

As well as in aircraft navigation, in the motion monitoring field the position of the coordinate frame of the body, that is the *body frame*, with respect to a reference coordinate frame, termed the *world frame*, is known as *attitude*, which is used as a synonym of orientation. *Euler angles* are one of several mathematical ways to describe the attitude of an object in three-dimensional Euclidean space. They represent a sequence of three elemental rotations about the axes of the coordinate system, defined as follows:

- The *roll* angle ϕ determines the rotation around the x-axis.
- The *pitch* angle θ determines the rotation around the y-axis.
- The *yaw* angle ψ determines the rotation around the z-axis.

Figure 2.3 depicts the rotation about the axes z, y', X by ψ, θ, ϕ, respectively, according to the Tait-Bryan convention. The colour blue indicates the world frame which is matching with body frame before the rorations. The colour red indicates the orientation of the body frame after the rotations were carried out. In contrast to *extrinsic rotations*, where the three elemental rotations may occur either about the axes of the original coordinate system, the Tait-Bryan rotations are *intrinsic rotations* that occur about the axes of the rotating coordinate system, which changes its orientation after each rotation.

Euler angles are a simple and intuitive means to represent rotations in three-dimensional space. However, for the above mentioned parameterisation they have singularities at values of $\theta = n\pi$, $n \in \mathbb{Z}$. At these points a rotation about the x-axis and the z-axis constitute the same motion, which results in the loss of one degree of freedom and makes changes in ϕ and ψ indistinguishable. This phenomenon is called *gimbal lock*.

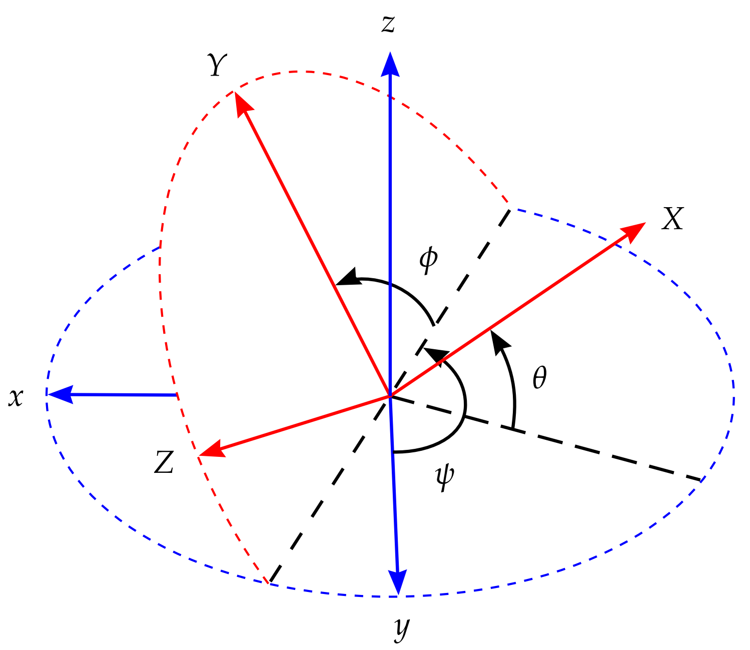

Figure 2.3: Representation of the body frame (red) with respect to the world frame (blue). The body frame was rotated, by the Euler angles ψ, θ, ϕ about the axes z, y', X, respectively, from [30].

2.3.1 *Transformation Matrix*

Coordinates representing a point in one coordinate system can be transformed to another. Such a transformations can be expressed as multiplication of a matrix with the coordinate vector that is to be transformed. Let $\mathbf{E}$ denote the orthonormal basis $\{x, y, z\} \in \mathbb{R}^3$ and let $\mathbf{E}'$ denote the orthonormal basis $\{X, Y, Z\} \in \mathbb{R}^3$. Furthermore, let $\mathbf{p}$ denote the position vector of an arbitrary point in three-dimensional Euclidean space. The coordinate transformation from $\mathbf{E}$ to $\mathbf{E}'$ is denoted $\mathbf{\Omega}_{\mathbf{E}\to\mathbf{E}'} : (p_1, p_2, p_3) \mapsto (p'_1, p'_2, p'_3)$. Then, the linear transformation from $\mathbf{p}$ to $\mathbf{p}'$ is given by

$$\mathbf{p}' = \mathbf{\Omega}_{\mathbf{E}\to\mathbf{E}'}(\mathbf{p}) = \mathbf{T}\mathbf{p}\,, \tag{2.2}$$

where $\mathbf{T}$ is the *transformation matrix*, which is a function of the rotation angles between the two coordinate systems.

In order to transform the coordinate vector from the *world frame* to the *body frame*, according to the common aerospace rotation sequence mentioned above and the north-east-down (NED) coordinate system, the transformation matrix $\mathbf{C}_{wb}$ is given by

$$\begin{aligned}\mathbf{C}_{wb} &= \mathbf{T}_x(\phi)\mathbf{T}_y(\theta)\mathbf{T}_z(\psi)\\ &= \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & \sin\phi\\ 0 & -\sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & -\sin\theta\\ 0 & 1 & 0\\ \sin\theta & 0 & \cos\theta\end{bmatrix}\begin{bmatrix}\cos\psi & \sin\psi & 0\\ -\sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}\cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta\\ \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \sin\phi\cos\theta\\ \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi & \cos\phi\cos\theta\end{bmatrix}\end{aligned} \tag{2.3}$$

Plugged in Equation 2.2 ($\mathbf{T} = \mathbf{C}_{wb}$), the pre-multiplications of the matrices $\mathbf{T}_x(\phi), \mathbf{T}_y(\theta), \mathbf{T}_z(\psi)$ to the vector $\mathbf{p}$ represent the coordinate rotations about the single axes x, y, z, according to the right hand rule, respectively. That is, the function $\mathbf{\Omega}_{\mathbf{E}\to\mathbf{E}'}$ maps the vector $\mathbf{p}$ to its orthogonal projection onto the axes of the coordinate system, which result from the respective two-dimensional rotation of ϕ, θ, ψ about the axes x, y, z. This is illustrated for a single rotation around the z-axis by the angle ψ in Figure 2.4. The matrices $\mathbf{T}_x(\phi), \mathbf{T}_y(\theta)$, and $\mathbf{T}_z(\psi)$ are also known as direction cosine matrices, since their elements are the cosines of the unsigned angles between the body-fixed axes and the axes of the world frame, as shown in [31]. The form stated here is already simplified. The matrix $\mathbf{C}_{bw}$ that transforms a coordinate vector from the body frame to the world frame is given by

$$\mathbf{C}_{bw} = \begin{bmatrix}\cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi\\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi\\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta\end{bmatrix} \tag{2.4}$$

Note that $\mathbf{C}_{bw} = \mathbf{C}_{wb}^T = \mathbf{C}_{wb}^{-1}$. Thus, $\mathbf{C}_{bw}$ and $\mathbf{C}_{wb}$ are orthogonal matrices so that $\mathbf{C}_{bw}\mathbf{C}_{wb} = \mathbf{I}_3$, where $\mathbf{I}_3 \in \mathbb{R}^3$ is the identity matrix.

2.4 PROJECTION OF THE GRAVITY VECTOR

As described in Section 2.1.1, accelerometers measure the linear acceleration they experience. Under static or quasi-static conditions, that is the sensor is in steady, linear motion, or at low acceleration it can be assumed that the measured acceleration is mainly that of gravity. By means of simple trigonometric functions estimates for the pitch and the roll angle can

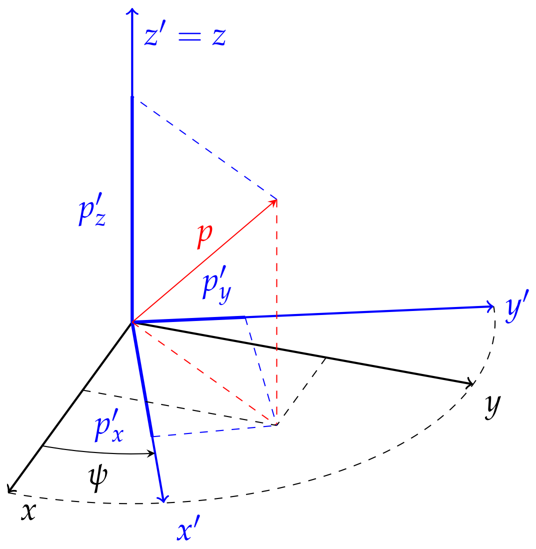

Figure 2.4: An exemplary coordinate rotation about the z-axis by an angle ψ, illustrating the orthogonal projection on the resulting axes x', y', z'.

be obtained. Since the gravity vector is perpendicular to the xy-plane, and thus a rotation around the z-axis will not cause any variation in the sensed acceleration, the yaw angle cannot be obtained by this method. To solve this problem a three-dimensional magnetometer is used, which measures the variation of Earth's magnetic field while rotating around the z-axis.

When the accelerometer is motionless, its measurements will be directly related to the angle of the sensor relative to gravity, as depicted in Figure 2.5 (a). In that case θ with respect to the vertical is given by

$$\theta = \text{atan2}(A_z, A_x)\,, \tag{2.5}$$

where A_x and A_z are the components of the acceleration vector in x and z direction, respectively. However, when the sensor is in motion, in addition to the gravity, there are radial and tangential acceleration components due to motion, as depicted in Figure 2.5 (b). The magnitude of the gravity vector $\mathbf{g}$ is denoted with $\|\mathbf{g}\|$. Ignoring these components will cause incorrect angle estimates.

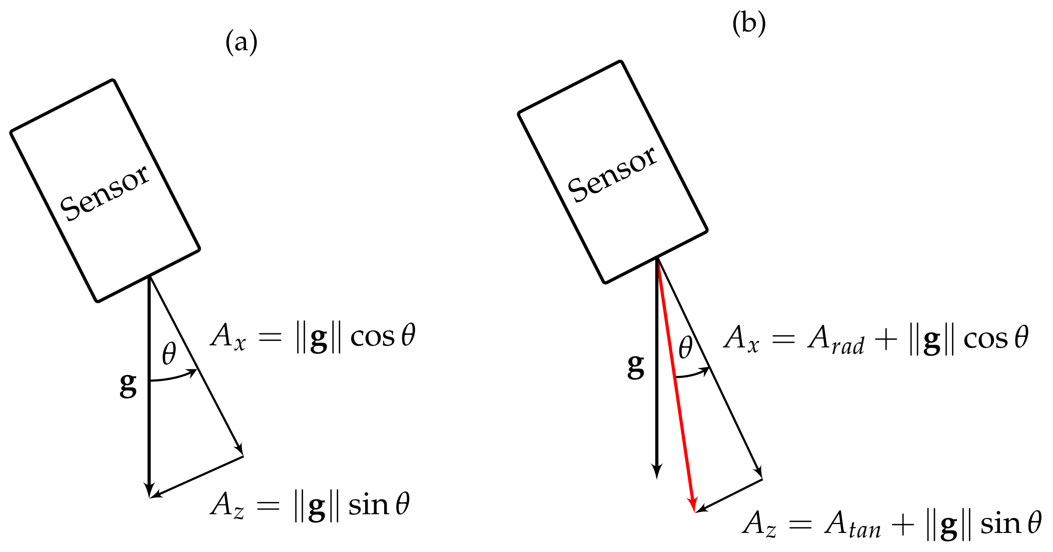
(a)
Sensor
$A_x = \|\mathbf{g}\| \cos\theta$
θ
$\mathbf{g}$
$A_z = \|\mathbf{g}\| \sin\theta$
(b)
Sensor
$A_x = A_{rad} + \|\mathbf{g}\| \cos\theta$
θ
$\mathbf{g}$
$A_z = A_{tan} + \|\mathbf{g}\| \sin\theta$

Figure 2.5: Acceleration seen by the sensor (b) with and (a) without motion, from [7].

2.5 INTEGRATION OF THE ANGULAR RATE

Another way to estimate the attitude of an object is the integration of the angular rate around the x, y and z-axis, respectively. Although this would theoretically lead to very accurate orientation estimates, they are impaired by angle random walk (ARW) and dynamical bias in practice. ARW is an effect caused by the integration of high-frequency, thermo-mechanical noise, which leads to a random additive angle in the orientation signal. An even greater impact than AWR has the gyroscopes dynamic bias, which has its origin in low-frequency flicker noise. Both effects cause a dramatical drift in the angle signal over time.

2.6 SENSOR FUSION

Since the projection of the gravity vector is only valid under static or quasi-static conditions, or at low acceleration, and the integration of the angular rate leads to non-reliable estimates due to ARW and dynamic bias, but is not affected by the intensity of motion, a means to combine the information of both sensors is desirable. The combination of information from multiple sensors to increase the overall precision of the estimation of a certain quantity of interest is termed *sensor fusion*. Raol [32] states the following advantages of sensor fusion:

- Robust functional and operational performance is given, in case of data loss from one sensor, due to redundancy provided by multiple sensors.
- Enhanced confidence in the results inferred from the measurement of one sensor, if they are confirmed by the measurement of another sensor.
- With sensor fusion an arbitrary fine time resolution of measurements is possible, whereas single sensors need a finite time to transmit measurements and so limit the frequency of measurements.
- One sensor might be, to some extent, better in a certain state of the measured process, e. g. low or high motion intensity in attitude estimation, and thus, by fusing multiple sensor signals, a satisfactory accuracy among all states of the process could be attained.

Sensor fusion can be realised by the use of a Kalman filter. This specific digital filter is described in detail in the next Chapter und applied in Chapter 4 to fuse the sensor signals of accelerometers and gyroscopes.

3 DIGITAL FILTERS

Conceived in general terms, a filter is a physical device for removing unwanted components of a mixture. In the technical field a filter is a system designed to extract information from noisy measurements of a process. That is, the filter delivers an estimate of the variables of principal interest, which is why it may also be called an estimator. Filter theory is applied in diverse fields of science and technology, such as communications, radar, sonar, navigation, and biomedical engineering [33].

In contrast to *analogue filters* that consist of electronic circuits to attenuate unwanted frequencies in continuous-time signals and thus extract the useful signal, a *digital filter* is a set of mathematical operations applied to a discrete-time signal in order to extract information about the hidden quantity of interest. A *discrete-time* signal is a sequence of samples at equidistant time instants that represent the continuous-time signal with no loss, provided the sampling theorem is satisfied, according to which the sample frequency has to be greater than twice the highest frequency component of the continuous-time signal.

Digital filters can be classified as *linear* and *non-linear*. If the quantity at the output of the filter is a *linear* function of its input, that is, the filter function satisfies the superposition principle, the filter is said to be *linear*. Otherwise, the filter is *non-linear*.

3.1 THE FILTERING PROBLEM

Consider, as an example involving filter theory, the continuous-time dynamical system depicted in Figure 3.1. The desired state vector of the system, $\mathbf{x}(t)$, is usually hidden and can only be observed by indirect measurements $\mathbf{z}(t)$ that are a function of $\mathbf{x}(t)$ and subject to noise. Equally, the equation describing the evolution of the state $\mathbf{x}(t)$ is usually subject to system errors. These could be caused by, for instance, effects not ac-

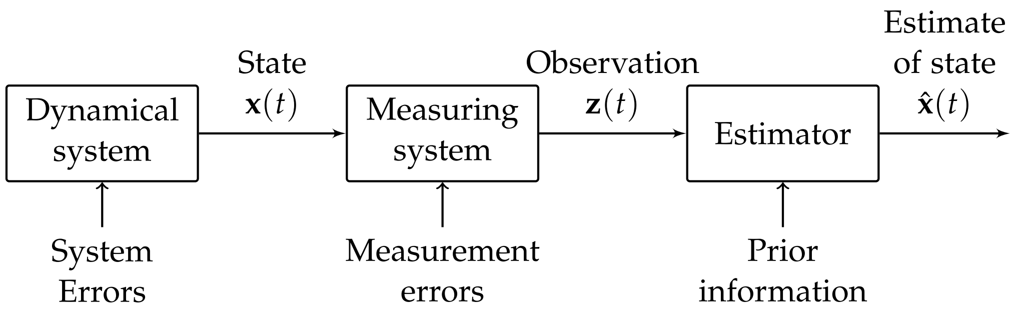

Figure 3.1: Block diagram depicting the components involved in state estimation, from [33].

counted for in the model. The dynamical system may be an aircraft in flight, in which case the elements of the state vector are constituted by its position and velocity. The measuring system may be a tracking radar producing the observation vector $\mathbf{z}(t)$ over an interval $[0, T]$. The requirement of the filter is to deliver a reliable *estimate* $\hat{\mathbf{x}}(t)$ of the actual state, by taking the measurement as well as a prior information into account.

3.2 THE WIENER FILTER

A statistical criterion, according to which the performance of a filter can be measured, is the mean-squared error. Consider the linear dicrete-time filter with the impulse response $h_0, h_1, h_2, \ldots$ depicted in Figure 3.2. At some discrete time k it produces an output designated by $\hat{x}(k)$, which provides an estimate of a desired response denoted by $d(k)$. According to Haykin [33], the essence of the filtering problem and the resulting requirement is summarised with the following statement:

> "Design a linear discrete-time filter whose output $\hat{x}(k)$ provides an estimate of the desired response $d(k)$, given a set of input samples $z(0), z(1), z(2), \ldots$, such that the mean-square value of the estimation error $e(k)$, defined as the difference between the desired response $d(k)$ and the actual response $\hat{x}(k)$, is minimized."

Assume a *stationary* stochastic process with known statistical parameters as the mean and correlation functions of the useful signal and the unwanted additive noise. Then, the solution to this statistical optimisation problem is commonly known as the *Wiener filter*. Yet, since the

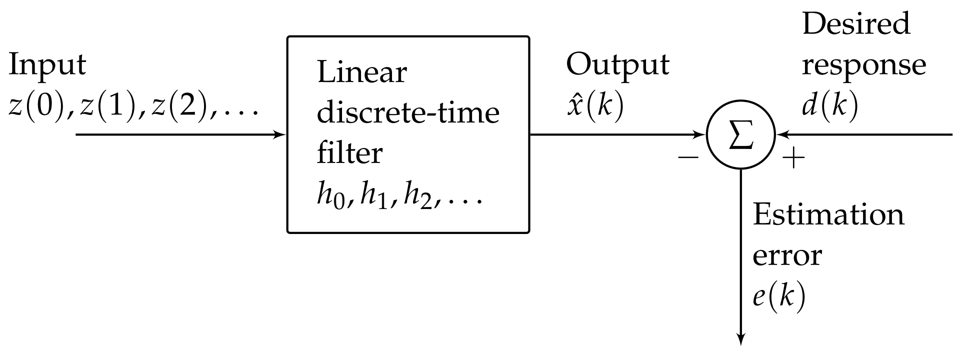

Figure 3.2: Block diagram representation of the statistical filtering problem, from [33].

Wiener filter requires a priori information about the statistics of the data to be processed, it may not be optimum for *non-stationary* processes. For such an environment, in which the statistics are time-varying, it needs a filter that constantly adapts its parameters to optimise its output.

3.3 ADAPTIVE FILTERS

A possible approach to mitigate the limitations of the Wiener filter for non-stationary processes is the 'estimate and plug' procedure. The filter 'estimates' the statistical parameters of the relevant signals and 'plugs' them into a *non-recursive* formula for computing the filter parameters. This procedure requires excessively elaborate and costly hardware for real-time operation [33]. To overcome this disadvantage one may use an *adaptive filter*, which is a self-designing system that relies, in contrast, on a *recursive* algorithm. This allows the filter to perform satisfactorily even if there is no complete knowledge of the relevant signal characteristics. Provided the variations in the statistics of the input data are sufficiently slow, the algorithm can track time variations and is thus suitable for non-stationary environments. The algorithm starts from some predetermined set of initial conditions respecting the knowledge about the system. In a stationary environment it converges to the optimum Wiener solution in some statistical sense after successive iterations. The *Kalman filter* is one such adaptive filter.

Due to the fact that the parameters of an adaptive filter are updated each iteration, they become data dependent. The system does not obey the principles of superposition which therefore makes the adaptive filter in reality a *non-linear* system. However, an adaptive filter is commonly said to be *linear* if its input-output map satisfies the superposition principle, as long as its parameters are held fixed. Otherwise it is said to be *non-linear*.

3.4 THE KALMAN FILTER

The *Kalman filter* is a set of recursive mathematical equations that provide an efficient means to estimate the state of a *linear* dynamic system, perturbed by additive white Gaussian noise, even when the precise nature of the modelled system is unknown. It incorporates knowledge of the system and measurement device dynamics, the statistical description of the system errors and measurement noise, and available information about initial conditions of the variables of interest, in order to produce an estimate of these variables, in a way that the mean of the squared error is minimised [34].

The filter is named after Rudolf E. Kalman who 1960 published his famous paper describing a recursive solution to the discrete-data linear filtering problem [35]. Since that time, the Kalman filter has been the subject of extensive research, due to a large extent to the advances in digital computing [36]. It finds applications in radar tracking, navigation, and orientation estimation, among others. Zarchan and Musoff [37] stated: "With the possible exception of the fast Fourier transform, Kalman filtering is probably the most important algorithmic technique ever devised."

3.4.1 *An Introductory Example*

The following introductory example from Maybeck [34] is an illustrative description of the determination of a one-dimensional position to understand how the Kalman filter works. Suppose you are lost at sea during the night and take a star sighting to determine your approximate position at time t_1 to be z_1. Your location estimate is, due to inherent measurement device inaccuracies and human error, somewhat uncertain, and thus assumed to be associated with a standard deviation σ_{z_1}. The conditional probability of $x(t_1)$, your actual position at time t_1, conditioned on the observed value z_1, is depicted in Figure 3.3. The best estimate of your position, based on this conditional probability density, is

$$\hat{x}(t_1) = z_1 , \tag{3.1}$$

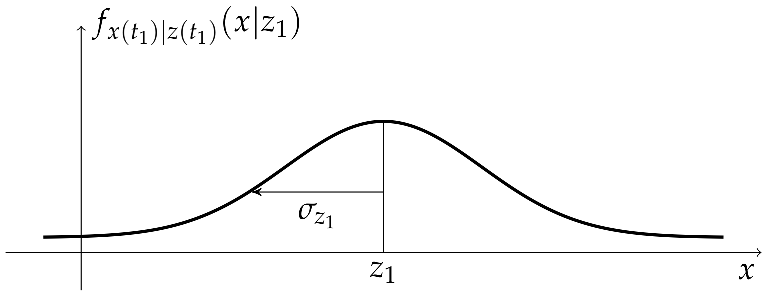

Figure 3.3: Conditional probability density of the position based on measurement value z_1, from [34].

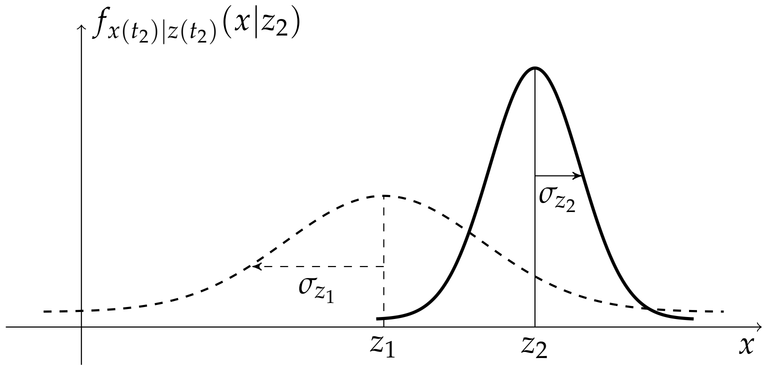

Figure 3.4: Conditional probability density of the position based on measurement value z_2 alone, from [34].

and the variance of the error in the estimate is

$$\sigma_x^2(t_1) = \sigma_{z_1}^2 \,. \tag{3.2}$$

Right after you, say a trained navigator friend takes an independent fix at time $t_2 \cong t_1$, so that the true position has not changes at all. He obtains a measurement z_2 with a variance $\sigma_{z_2}^2$, which is somewhat smaller than yours, since he has a higher skill. Figure 3.4 depicts the conditional probability density of your position at time t_2, based only on the measurement value z_2. Combining these data, your position at time $t_2 \cong t_1$, $x(t_2)$, given both z_1 and z_2, is then a Gaussian density with mean μ and variance σ^2, as indicated in Figure 3.5, with

$$\mu = z_1 \frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} + z_2 \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} \tag{3.3}$$

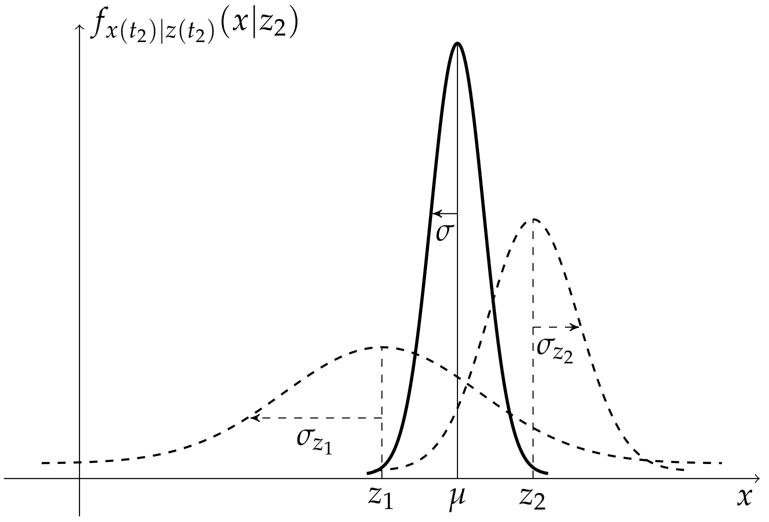

Figure 3.5: Conditional probability density of the position based on data z_1 and z_2, from [34].

and

$$\frac{1}{\sigma^2} = \frac{1}{\sigma_{z_1}^2} + \frac{1}{\sigma_{z_2}^2} \,. \tag{3.4}$$

The uncertainty in your estimate of position has been decreased because σ is less than either $\sigma_{z_1}^2$ or $\sigma_{z_2}^2$. Even if σ_{z_1} was very large, the variance of the estimate is less than σ_{z_2}, which means that even poor quality data increases the precision of the filter output. The best estimate, given this density, is

$$\hat{x}(t_2) = \mu \,, \tag{3.5}$$

with an associated error variance σ^2.

Having a closer look at the form of μ in Equation 3.3, one notices that it makes good sense. If the measurements were of equal precision, meaning $\sigma_{z_1} = \sigma_{z_2}$, the optimal estimate is simply the average of both measurements, as would be expected. If σ_{z_1} is larger than σ_{z_2}, the equation weights z_2 more heavily than z_1.

Equation 3.5 for the filter output can be written as

$$\begin{aligned}\hat{x}(t_2) &= z_1 \frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} + z_2 \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} \\ &= z_1 + \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} [z_2 - z_1] \,,\end{aligned} \tag{3.6}$$

or in a form that is used in Kalman filter implementations, with $\hat{x}(t_1) = z_1$, as

$$\hat{x}(t_2) = \hat{x}(t_1) + K(t_2)[z_2 - \hat{x}(t_1)] \,, \tag{3.7}$$

where

$$K(t_2) = \frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2} \,. \tag{3.8}$$

These equation represent the 'predictor-corrector' structure of the Kalman filter. A prediction of the value that the desired variables and the measurements will have at the next measurement time is made, based on all previous information. Then the difference between the measurement and its predicted value is used to correct the prediction of the desired variables. According to Equation 3.7 the optimal estimate at time t_2, that is $\hat{x}(t_2)$, is equal to $\hat{x}(t_1)$, the best prediction of its value before z_2 is taken, plus a correction term of an optimal weighting value times the difference between z_2 and the best prediction of it before the measurement is actually taken.

To incorporate dynamics into the model, suppose you travel for some time before taking another measurement. The best model you have for your motion may be of the form

$$\frac{dx}{dt} = u + w \,, \tag{3.9}$$

where u is a nominal velocity and w is a noise term, representing the uncertainty in your knowledge of the actual velocity due to disturbances and effects not accounted for in the simple first order equation. It will be modelled as white Gaussian noise with a mean of zero and variance of σ_w^2.

The conditional density of the position at time t_2, given z_1 and z_2, was previously derived. Figure 3.6 shows graphically how the density travels along the x-axis as time progresses. It start at the best estimate

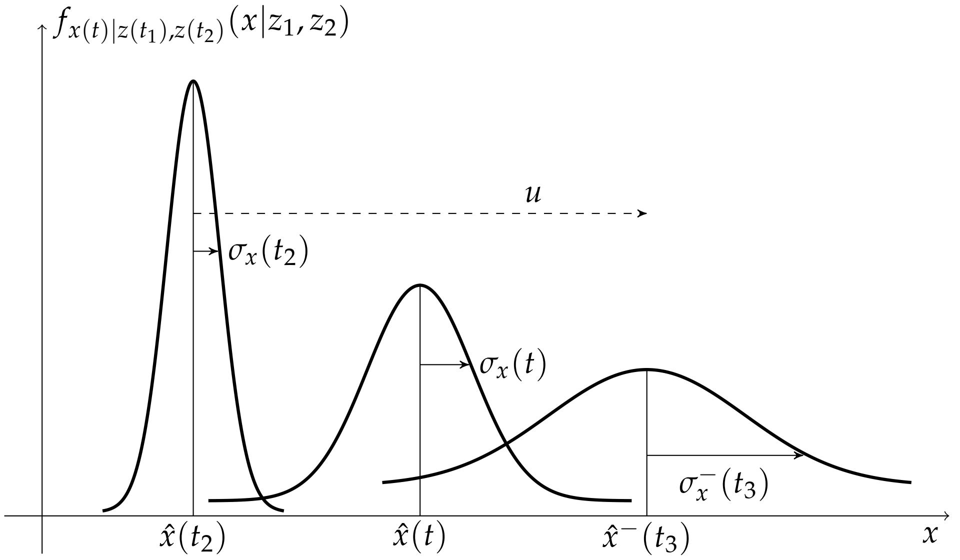

Figure 3.6: Propagation of conditional probability density, from [34].

and moves according to the above mentioned model of dynamics. Due to the constant addition of uncertainty over time it spreads out. Thus, as the variance becomes greater, you become less sure of your position. The Gaussian density $f_{x(t_3)|z(t_1),z(t_2)}(x|z_1,z_2)$ can be expressed mathematically by its mean and variance given by

$$\hat{x}^-(t_3) = \hat{x}(t_2) + u[t_3 - t_2], \tag{3.10}$$

$$\sigma_x^{2-}(t_3) = \sigma_x^2(t_2) + \sigma_w^2[t_3 - t_2], \tag{3.11}$$

where the superscript $-$ denotes the prediction of $\hat{x}$ and σ_x^2, respectively. Before the measurement is taken at time t_3, $\hat{x}^-(t_3)$ is the optimal prediction of the location at t_3, associated with the variance $\sigma_x^{2-}(t_3)$ in this prediction.

Now a measurement z_3 with an assumed variance $\sigma_{z_3}^2$ is taken. As before, its conditional probability density is combined with the density with mean $\hat{x}^-(t_3)$ and variance $\sigma_x^{2-}(t_3)$, to yield a Gaussian density with mean

$$\hat{x}(t_3) = \hat{x}^-(t_3) + K(t_3)[z_3 - \hat{x}^-(t_3)] \tag{3.12}$$

and variance

$$\sigma_x^2(t_3) = \sigma_x^{2-}(t_3) - K(t_3)\sigma_x^{2-}(t_3)\,, \tag{3.13}$$

where the gain $K(t_3)$ is given by

$$K(t_3) = \frac{\sigma_x^{2-}(t_3)}{\sigma_x^{2-}(t_3) + \sigma_{z_3}^2}\,. \tag{3.14}$$

Observing the form of Equation 3.14 the reasonableness of the filter structure becomes obvious. If the variance of the measurement noise $\sigma_{z_3}^2$ is large, then $K(t_3)$ is small, meaning that little confidence is put in a very noisy measurement and that it is weighted lightly. For $\sigma_{z_3}^2 \to \infty$, $K(t_3)$ becomes zero, and $\hat{x}(t_3)$ equals $\hat{x}^-(t_3)$. Thus, an infinitely noisy measurement is totally ignored. Likewise, if the dynamical system noise variance σ_w^2 is large, then according to Equation 3.11, $\sigma_x^{2-}(t_3)$ will be large, and so will be $K(t_3)$. Therefore, the measurement is weighted heavily, in case you are not very certain about the output of the system model within the filter structure. In the limit as $\sigma_w^2 \to \infty$, $\sigma_x^{2-}(t_3) \to \infty$, and $K(t_3) \to 1$, so Equation 3.5 yields

$$\hat{x}(t_3) = \hat{x}(t_3^-) + 1 \cdot [z_3 - \hat{x}(t_3^-)] = z_3\,. \tag{3.15}$$

That means that in the limit of absolutely no confidence in the system model output, solely the new measurement is taken as the optimal estimate. Finally, if you are absolutely sure of your estimate before z_3 becomes available, $\sigma_x^{2-}(t_3)$ would become zero, and so would $K(t_3)$, which means that the measurements would be left disregarded.

Extending Equations 3.10, 3.11, 3.12, 3.13, and 3.14 to the vector case, and allowing time varying parameters in the system and noise description leads to the general Kalman filter equations. A complete mathematical derivation is found in Haykin [33].

3.4.2 *Formulation of the Kalman Filter Equations*

Let $\mathbf{x}_k \in \mathbb{R}^n$ be the state vector of a discrete-time controlled process governed by the *linear* stochastic difference equation

$$\mathbf{x}_k = \mathbf{\Phi}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{3.16}$$

and $\mathbf{z}_k \in \mathbb{R}^m$ the observation or measurement vector of this process, given by

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k\,, \tag{3.17}$$

where the index $k \in \mathbb{N}^0$ denotes discrete time normalised to the sampling interval. The $n \times 1$ vector $\mathbf{w}_k$ and the $m \times 1$ vector $\mathbf{v}_k$ represent the process noise and the measurement noise, respectively, modelled as zero-mean, Gaussian white noise

$$\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)\,, \tag{3.18}$$

$$\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)\,, \tag{3.19}$$

with the process noise covariance matrix $\mathbf{Q}_k$ and the measurement noise covariance matrix $\mathbf{R}_k$. The $n \times n$ state transition matrix $\mathbf{\Phi}_{k-1}$ in Equation 3.16 relates the state at the previous time step $k-1$ to the state at the current step k. The $n \times l$ matrix $\mathbf{B}_{k-1}$ relates the known, optional control input $\mathbf{u}_{k-1} \in \mathbb{R}^l$ to the state $\mathbf{x}_k$. Finally, the $m \times n$ measurement matrix $\mathbf{H}_k$ in Equation 3.17 relates the state $\mathbf{x}_k$ to the measurement $\mathbf{z}_k$. Both noise processes are assumed to be uncorrelated. The process noise might not always have a physical meaning. However, it represents the fact that the model of the real world is not precise. The process and measurement noise covariance matrices are related to the respective noise vectors according to

$$\mathbf{Q}_k = \mathrm{E}[\mathbf{w}_k, \mathbf{w}_k^T]\,, \tag{3.20}$$

$$\mathbf{R}_k = \mathrm{E}[\mathbf{v}_k, \mathbf{v}_k^T]\,, \tag{3.21}$$

where E denotes the expected value.

The Kalman filter solves the problem of estimating the state $\mathbf{x}_k$ of the given linear stochastic system, minimising the weighted mean-squarred error. This problem is called the *linear quadratic Gaussian* estimation problem; the dynamic system is linear, the performance cost function is quadratic, and the random process is Gaussian.

We define the vector $\hat{\mathbf{x}}_k^- \in \mathbb{R}^n$ as the *a priori* state estimate representing knowledge of the process prior to step k and $\hat{\mathbf{x}}_k \in \mathbb{R}^n$ as the *a posteriori* state estimate at step k given the measurement $\mathbf{z}_k$:

$$\hat{\mathbf{x}}_k^- = \mathbf{\Phi}_{k-1} \hat{\mathbf{x}}_{k-1} + \mathbf{B}_{k-1} \mathbf{u}_{k-1}\,, \tag{3.22}$$

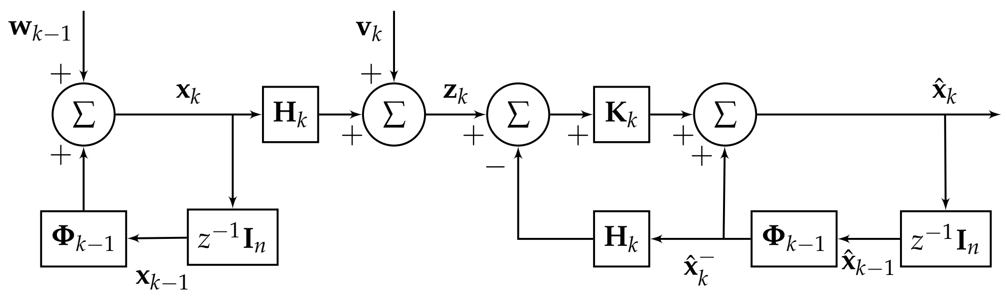

Figure 3.7: Block diagram depicting the relation between a discrete-time dynamical system, its observation, and the Kalman filter.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-]\,. \tag{3.23}$$

The term $[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-]$ is called the measurement *innovation* or *residual.* It reflects the discordance between the predicted measurement $\mathbf{H}_k\hat{\mathbf{x}}_k^-$ and the actual measurement $\mathbf{z}_k$. The $n \times m$ matrix $\mathbf{K}_k$ is termed the Kalman gain and is given by

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T[\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k]^{-1}\,, \tag{3.24}$$

with

$$\mathbf{P}_k^- = \boldsymbol{\Phi}_{k-1}\mathbf{P}_{k-1}\boldsymbol{\Phi}_{k-1}^T + \mathbf{Q}_{k-1} \tag{3.25}$$

and

$$\mathbf{P}_k = [\mathbf{I}_n - \mathbf{K}_k\mathbf{H}_k]\mathbf{P}_k^-\,. \tag{3.26}$$

Figure 3.7 illustrates the relation of the Kalman filter to the discrete-time dynamical system with $\mathbf{u}_k = 0$, for the sake of simplicity, whereby z^{-1} denotes the unit-delay and $\mathbf{I}_n$ the $n \times n$ identity matrix.

The Kalman filter equations can be divided into two groups: *time update* Equations 3.22, 3.25 and *measurement update* Equations 3.23 , 3.24, and 3.26, as seen in Figure 3.8, which shows the 'predict and correct' behaviour of the filter algorithm. After an initialisation of the parameters, the *time update* and *measurement update* steps are repeated recursively every time step.

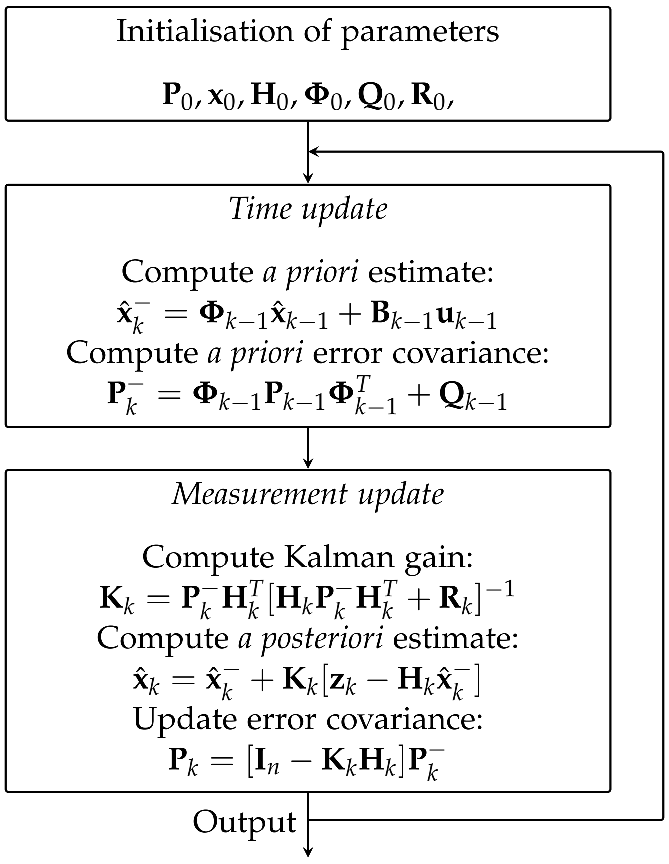
Initialisation of parameters
$\mathbf{P}_0, \mathbf{x}_0, \mathbf{H}_0, \mathbf{\Phi}_0, \mathbf{Q}_0, \mathbf{R}_0,$
Time update
Compute *a priori* estimate:
$\hat{\mathbf{x}}_k^- = \mathbf{\Phi}_{k-1}\hat{\mathbf{x}}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1}$
Compute *a priori* error covariance:
$\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}\mathbf{P}_{k-1}\mathbf{\Phi}_{k-1}^T + \mathbf{Q}_{k-1}$
Measurement update
Compute Kalman gain:
$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T[\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k]^{-1}$
Compute *a posteriori* estimate:
$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-]$
Update error covariance:
$\mathbf{P}_k = [\mathbf{I}_n - \mathbf{K}_k\mathbf{H}_k]\mathbf{P}_k^-$
Output

Figure 3.8: Operation cycle of the Kalman filter algorithm illustrating 'predict and correct' behaviour.

3.4.3 *The Extended Kalman Filter*

Up to this point the Kalman filter has solved the filtering problem for *linear* time-dynamical systems. One may extend the Kalman filter to systems with state dynamics governed by *non-linear* state transformations

$$\mathbf{x}_k = \boldsymbol{\phi}_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1}, \quad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)\,, \tag{3.27}$$

and/or a *non-linear* transformation from state variables to measurement variables

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)\,. \tag{3.28}$$

The *functional* $\boldsymbol{\phi}_{k-1}$ denotes the *non-linear* transition matrix function that may be time varying. It relates the state at the previous time step $k-1$ to the current time step k, depending on the exogenous control input $\mathbf{u}_{k-1}$. The functional $\mathbf{h}_k$ denotes a *non-linear* measurement matrix function that relates the state $\mathbf{x}_k$ to the measurement $\mathbf{z}_k$ and is possibly time varying, too.

Some non-linear problems can be deemed *quasilinear*, which means that the variation of the non-linear functionals $\boldsymbol{\phi}_k$ and $\mathbf{h}_k$ are predominantly linear about the value $\mathbf{x}_0$. That is,

$$\boldsymbol{\phi}_k(\mathbf{x}_0 + d\mathbf{x}, \mathbf{u}) \approx \boldsymbol{\phi}_k(\mathbf{x}_0, \mathbf{u}) + d\mathbf{x}\, \frac{\partial \boldsymbol{\phi}_k(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_0}, \tag{3.29}$$

$$\mathbf{h}_k(\mathbf{x}_0 + d\mathbf{x}) \approx \mathbf{h}_k(\mathbf{x}_0) + d\mathbf{x}\, \frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_0}, \tag{3.30}$$

which requires that $\boldsymbol{\phi}_k$ and $\mathbf{h}_k$ are differentiable at $\mathbf{x}_0$.

Through a *linearisation* of the state-space model of Equations 3.27 and 3.28 at each time instant around the most recent state estimate, the standard Kalman filter equation from Section 3.4.2 can be applied. The filter resulting from a *linear approximation* of the state transitions and the relation of the measurement to the respective state is referred to as the *extended Kalman filter*.

The linearisation of the functionals $\boldsymbol{\phi}_k$ and $\mathbf{h}_k$ is given by their respective Jacobian matrices

$$\boldsymbol{\Phi}_{k-1}^{[1]} = \frac{\partial \boldsymbol{\phi}_{k-1}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}^{-}, \mathbf{u}=\mathbf{u}_{k-1}}, \tag{3.31}$$

and

$$\mathbf{H}_k^{[1]} = \frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}_k^{-}}. \tag{3.32}$$

The ijth entry of $\boldsymbol{\Phi}_{k-1}^{[1]}$ is equal to the partial derivative of the ith component of $\boldsymbol{\phi}_{k-1}(\mathbf{x})$ with respect to the jth component of $\mathbf{x}$. The derivatives are evaluated at $\mathbf{x} = \hat{\mathbf{x}}_{k-1}^{-}, \mathbf{u} = \mathbf{u}_{k-1}$. Likewise, the ijth entry of $\mathbf{H}_k^{[1]}$ is equal to the partial derivative of the ith component of $\mathbf{h}_k(\mathbf{x})$ with respect to the jth component of $\mathbf{x}$. The derivatives are evaluated at $\mathbf{x} = \hat{\mathbf{x}}_k^{-}$. The superscript [1] denotes the *first-order* approximation.

Similar to Equation 3.22, the predicted state estimate is given by

$$\hat{\mathbf{x}}_k^{-} = \boldsymbol{\phi}_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}), \tag{3.33}$$

and the predicted measurement by

$$\hat{\mathbf{z}}_k = \mathbf{h}_k(\hat{\mathbf{x}}_k^-)\,. \tag{3.34}$$

The *a posteriori* estimate is then, conditioned on the actual measurement,

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k[\mathbf{z}_k - \hat{\mathbf{z}}_k]\,. \tag{3.35}$$

The corresponding *a priori* covariance matrix $\mathbf{P}_k^-$, the Kalman gain $\mathbf{K}_k$, and the *a posteriori* covariance matrix $\mathbf{P}_k$ are equal to Equations 3.24, 3.25, and 3.26 in Section 3.4.2. They are reproduced here with the linear approximation of the state transition and measurement matrices for convenience of presentation:

$$\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}^{[1]}\mathbf{P}_{k-1}\mathbf{\Phi}_{k-1}^{[1]T} + \mathbf{Q}_{k-1}\,, \tag{3.36}$$

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^{[1]T}[\mathbf{H}_k^{[1]}\mathbf{P}_k^-\mathbf{H}_k^{[1]T} + \mathbf{R}_k]^{-1}\,, \tag{3.37}$$

$$\mathbf{P}_k = [\mathbf{I}_n - \mathbf{K}_k\mathbf{H}_k^{[1]}]\mathbf{P}_k^-\,. \tag{3.38}$$

Figure 3.9 illustrates the 'predict and correct' behaviour of the extended Kalman filter algorithm. The *time update* and *measurement update* steps are repeated recursively every time step, after an initialisation of the parameters, similar to the classical Kalman filter algorithm. In addition, the Jacobian matrices have to be computed, in order to linearise the state-space model at each time instant around the most recent state estimate.

Extended Kalman filtering is commonly used. In fact, it was the first successful application of the Kalman filter [38]. Unlike its linear counterpart, the extended Kalman filter may not necessarily be an optimal estimator. Owing to its linearisation the EKF may quickly diverge, if the process is modelled incorrectly or the initial state estimate is wrong.

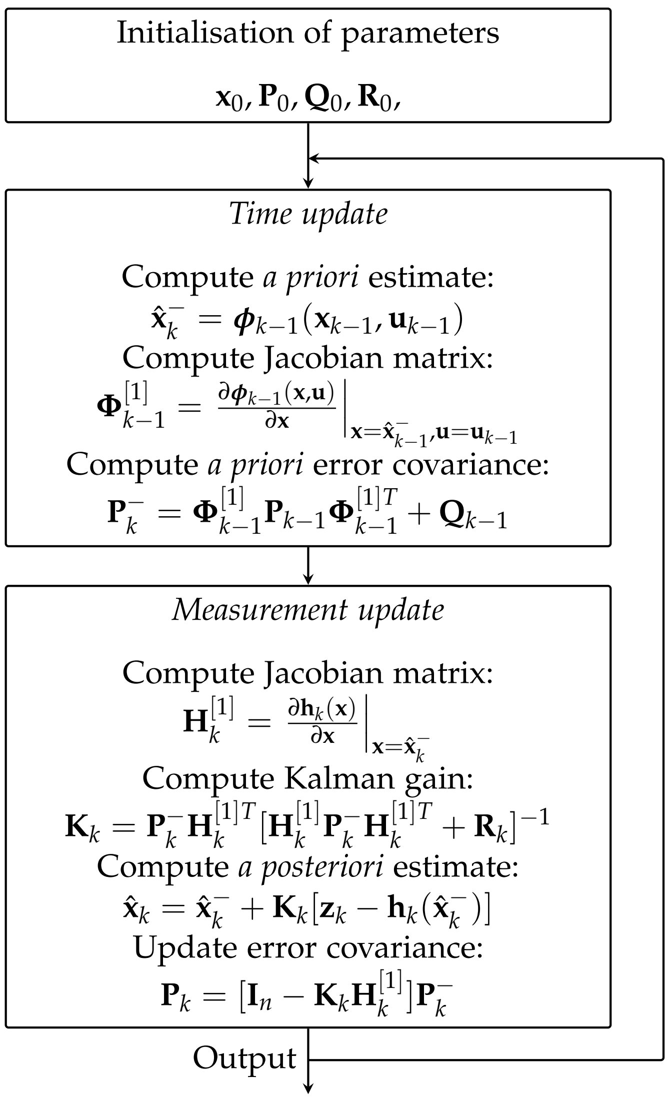
Initialisation of parameters
$\mathbf{x}_0, \mathbf{P}_0, \mathbf{Q}_0, \mathbf{R}_0,$
Time update
Compute *a priori* estimate:
$\hat{\mathbf{x}}_k^- = \boldsymbol{\phi}_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$
Compute Jacobian matrix:
$\boldsymbol{\Phi}_{k-1}^{[1]} = \frac{\partial \boldsymbol{\phi}_{k-1}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}^-, \mathbf{u}=\mathbf{u}_{k-1}}$
Compute *a priori* error covariance:
$\mathbf{P}_k^- = \boldsymbol{\Phi}_{k-1}^{[1]} \mathbf{P}_{k-1} \boldsymbol{\Phi}_{k-1}^{[1]T} + \mathbf{Q}_{k-1}$
Measurement update
Compute Jacobian matrix:
$\mathbf{H}_k^{[1]} = \frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_k^-}$
Compute Kalman gain:
$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^{[1]T} [\mathbf{H}_k^{[1]} \mathbf{P}_k^- \mathbf{H}_k^{[1]T} + \mathbf{R}_k]^{-1}$
Compute *a posteriori* estimate:
$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k[\mathbf{z}_k - \mathbf{h}_k(\hat{\mathbf{x}}_k^-)]$
Update error covariance:
$\mathbf{P}_k = [\mathbf{I}_n - \mathbf{K}_k \mathbf{H}_k^{[1]}]\mathbf{P}_k^-$
Output

Figure 3.9: Operation cycle of the extended Kalman filter algorithm illustrating 'predict and correct' behaviour.

4 IMPLEMENTATION

This chapter describes the implementation of the filter algorithm proposed by Bennett et al. in [7], based on the fundamentals acquired in the previous chapters. Other than in [7], in which the filter algorithm was tested moving a mechanical model of the leg by hand, we used movement data from human subjects. After outlining the initial situation, i. e. describing the GaitWatch system in detail, as well as stating orientation algorithms that were already implemented, the theoretical design of the filter is elaborated in detail. Subsequently, the software implementation and experiments are presented, followed by the results and their discussion.

4.1 INITIAL SITUATION

4.1.1 *The GaitWatch System*

As indicated above, to gather and preprocess movement data from subjects, we used a system called GaitWatch [39], which was designed to monitor the motion of patients by means of inertial sensors attached to the body. It was developed at the Department of Neurology of the Ludwig-Maximilians University in Munich, Germany, in association with the Department of Signal Theory, Telematics and Communications of the University of Granada, Spain.

Hardware

From the hardware perspective, the system is composed of a set of magnetic and inertial sensors wired to a box containing a microcontroller. This microcontroller is in charge of collecting data from the sensors embedded in the box, as well as from external measurement units, and storing them on a memory card. The separate units are placed at the patient's trunk, arms, thighs, and shanks as shown in Figure 4.1. The components of the three different kinds of subunits are listed below:

- TYPE A – THIGHS AND SHANKS:

  IMU Analog Combo Board with 5 Degrees of Freedom [40], containing an IDG500 biaxial gyroscope, from which only y-axis is actually used, with a measurement range of $\pm 500°/\mathrm{s}$ [41] and a $\pm 3\,\mathrm{g}$ triaxial accelerometer, ADXL335 [42].

- TYPE B – ARMS:

  IDG500 biaxial gyroscope with a measurement range of $\pm 500°/\mathrm{s}$ [41].

- TYPE C – TRUNK:

  ADXL345 triaxial accelerometer with a programmable measurement range of $\pm 2/\pm 4/\pm 8/\pm 16\,\mathrm{g}$ [43], IMU3000 triaxial gyroscope with a programmable measurement range of $\pm 250/\pm 500/\pm 1000/\pm 3000°/\mathrm{s}$ [44], Micromag3 triaxial magnetometer with a measurement range of ± 11 Gauss [45], AL-XAVRB board containing an AVR ATxmega processor [46].

Software

In addition to the hardware, there was an existing MATLAB® toolbox consisting of routines for reading the data and carrying out the necessary calibration. Also, several algorithms that determine the motion intensity and compute the orientation of the human body from the movement data were already implemented. There were four algorithms implemented that estimate the pitch angle of the thighs and shanks, as described below:

- PROJECTION OF THE GRAVITY VECTOR: One way to obtain the pitch angle from the inertial data is using the projection of the gravity vector on the axes of the accelerometer, as described in Section 2.4. The first algorithm computed the pitch angle according to Equation 2.5.

- INTEGRATION OF THE ANGULAR RATE: Another way to obtain the pitch angle is the integration of the angular rate, as outlined in Section 2.5. There are many different numerical integration procedures.

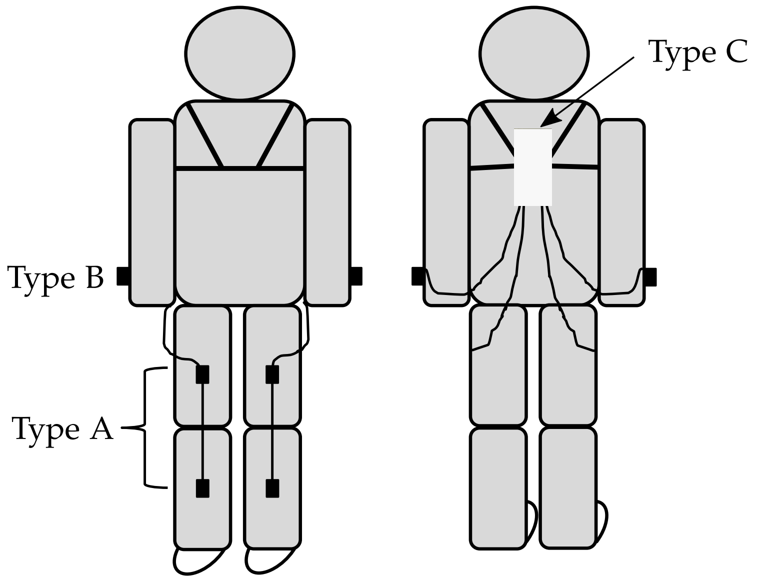

Figure 4.1: Placement of the GaitWatch components at the body, from [39].

The existing algorithm used the approximation of the integral according to the trapezoidal rule

$$\theta(n) = \theta_0 + \int_0^{nT_s} \omega(t)\, dt \approx \theta_0 + \frac{T_s}{2} \sum_{k=1}^{n} \left[\omega_{k-1} + \omega_k\right] , \qquad (4.1)$$

where $n, k \in \mathbb{N}$ denote time normalised to the sample period T_s, ω_k the measured angular rate at instant k, and θ_0 the angle at $k = 0$. The implemented algorithm computed the angle recursively as

$$\theta(n) \approx \theta(n-1) + \frac{T_s}{2} \left[\omega_{k-1} + \omega_k\right] , \quad \theta(0) = \theta_0 \, . \qquad (4.2)$$

The initial value θ_0 can be computed from the projection of the gravity vector, assuming that the patient stands still when the records are started. Figure 4.2 shows exemplary the result of the first two algorithms applied to estimate the shank angle with respect to the x-axis, according to the mechanical model of the leg depicted in Figure 4.5. As can be seen in Figure 4.2, the accelerometer-based approach does not suffer from drift but delivers a very poor angle estimate during

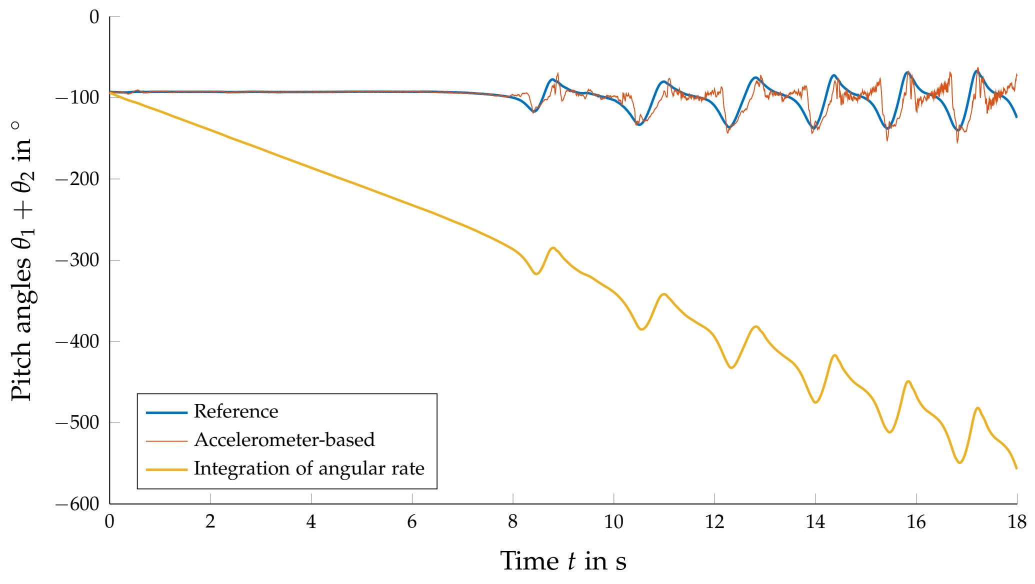

Figure 4.2: Pitch angle of the right shank with respect to the x-axis, obtained by the projection of the gravity vector and by integrating the angular rate, in comparison to the reference.

periods of motion, especially with increasing motion intensity. On the other hand, integrating the angular rate delivers an accurate angle estimate during motion, but suffers from drift over time, which accounts for an error in the estimate of more than 500° in only eighteen seconds for this exemplary signal. The reference signal was obtained with the Qualisys® motion capture system.

- KALMAN FILTER: The third algorithm fused the orientation angle based on the projection of the gravity vector with the orientation angle based on the integration of the angular rate measured with the gyroscope in a classical Kalman filter, without taking the motion intensity into account [39]. The result in comparison with the projection of the gravity vector alone is depicted in Figure 4.3.

In tandem with an orientation estimation based on a Qualisys® motion capture system using high-speed cameras in combination with optical markers, the orientation angles estimated with the above-mentioned algorithms served as a reference. The placement of the markers on the leg are depicted in Figure 4.4. From the recorded motion of the markers

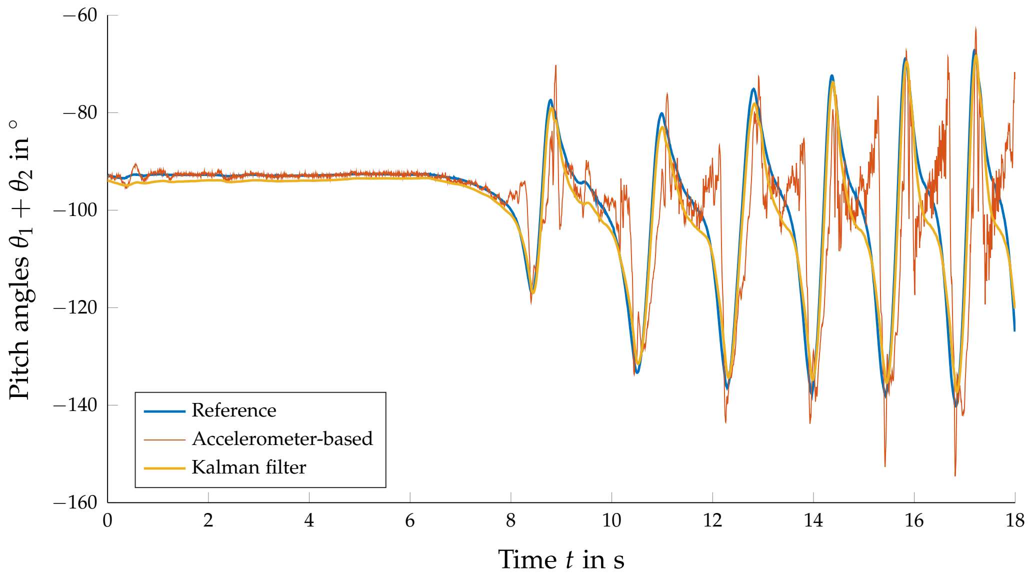

Figure 4.3: Pitch angle of the right shank with respect to the x-axis, obtained by the projection of the gravity vector and by sensor fusion of accelerometer and gyroscope data in a classical Kalman filter, in comparison to the reference.

in space the motion capture system we computed the reference angles of the thighs and shanks, using an existing algorithm.

4.2 THEORETICAL DESIGN

In this section the theoretical design of the system proposed by Bennett et al. in [7] is mapped to the existing GaitWatch system. It states the assigned coordinate frames and the conventions regarding rotations about their axes. Furthermore, it presents the kinematic model used to improve the angle estimates and the extended Kalman filter algorithm with its underlying state-space model in detail.

4.2.1 *Kinematic Model*

The kinematic model relates the respective angles of the thigh and shank about the hip and knee joint to the acceleration seen by the wearable sensors. When walking in a straight line, the human leg can be modelled as a two-link planar revolute robot [7]. Then, thighs and shanks remain in a single plane which is approximately parallel to the direction

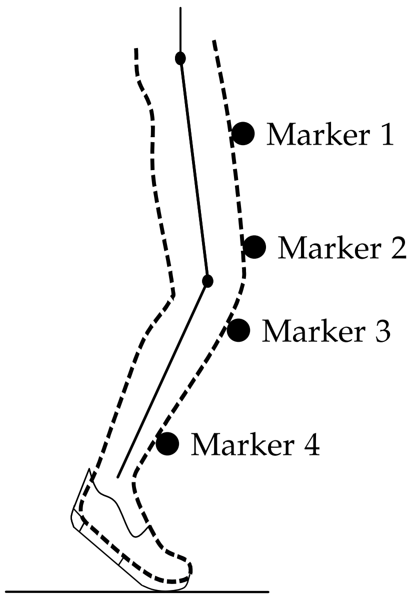

Figure 4.4: Human leg with optical markers, from [1].

of motion. As depicted in Figure 4.5, the revolute joints of the pendulum robot represent the hip und knee joint, and the two links the thigh and shank, respectively. The origin of the inertial world frame is located at the base of link 1, the upper of both links. The x-axis points forward, the y-axis points out from the hip to the right, and the z-axis points down. This configuration follows the right-hand rule, which can also be used to determine the sense of rotation around the axes. The pitch angle θ_1 is measured with respect to the x-axis, and the pitch angle θ_2 of link 2 with respect to link 1.

The IMUs placed on the thighs and shanks measured the angular velocity around the y-axis and the linear acceleration along the x and z-axis, respectively. According to Spong and Hutchinson [47], the x and z displacement and its derivatives in the world frame are as follows:

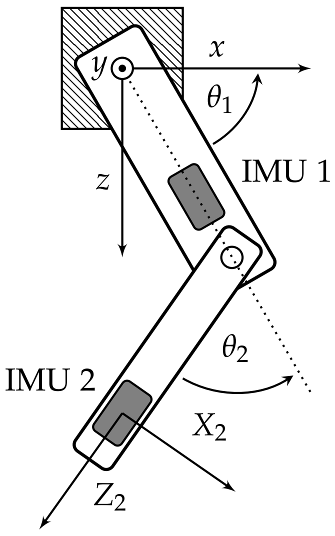

Figure 4.5: Kinematic model of the human leg, from [7].

$$x = +l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \tag{4.3}$$
$$\dot{x} = -l_1 \dot{\theta}_1 \sin(\theta_1) - l_2(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2) \tag{4.4}$$
$$\begin{aligned} \ddot{x} = &-l_1[\dot{\theta}_1^2 \cos(\theta_1) + \ddot{\theta}_1 \sin(\theta_1)] - l_2[(\dot{\theta}_1 + \dot{\theta}_2)^2 \cos(\theta_1 + \theta_2) \\ &+(\ddot{\theta}_1 + \ddot{\theta}_2) \sin(\theta_1 + \theta_2)] \end{aligned} \tag{4.5}$$

$$z = -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) \tag{4.6}$$
$$\dot{z} = -l_1 \dot{\theta}_1 \cos(\theta_1) - l_2(\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2) \tag{4.7}$$
$$\begin{aligned} \ddot{z} = &-l_1[\ddot{\theta}_1 \cos(\theta_1) - \dot{\theta}_1^2 \sin(\theta_1)] - l_2[(\ddot{\theta}_1 + \ddot{\theta}_2) \cos(\theta_1 + \theta_2) \\ &-(\dot{\theta}_1 + \dot{\theta}_2)^2 \sin(\theta_1 + \theta_2)] \end{aligned} \tag{4.8}$$

in which l_1 and l_2 are the lengths of the two links, respectively. Plugging the *a priori* estimates of the angles θ_1 and θ_2 and their derivatives, i. e. the angular rates ω_1 and ω_2, and the angular accelerations α_1 and α_2, obtained with the EKF described in Section 4.2.2, into Equations 4.5 and 4.8, we can estimate the respective motion-based acceleration components a_x and a_z in x and z direction that sensor 2 will see in the world coordinate frame. Written as a function of the state variables of the extended Kalman filter, Equations 4.5 and 4.8 yield

$$a_x = -l_1[\omega_1^2 \cos(\theta_1) + \alpha_1 \sin(\theta_1)] - l_2[(\omega_1 + \omega_2)^2 \cos(\theta_1 + \theta_2) \\ +(\alpha_1 + \alpha_2) \sin(\theta_1 + \theta_2)] \tag{4.9}$$

$$a_z = -l_1[\alpha_1 \cos(\theta_1) - \omega_1^2 \sin(\theta_1)] - l_2[(\alpha_1 + \alpha_2) \cos(\theta_1 + \theta_2) \\ -(\omega_1 + \omega_2)^2 \sin(\theta_1 + \theta_2)] \tag{4.10}$$

The orientation of the sensor frames at rest are different from the world frame and dynamic when the pendulum is in motion. In order to transform the values from the world frame to the dynamic body frame of IMU 2, which is depicted in Figure 4.5, we used the transformation matrix $\mathbf{T}_y(\theta)$ from Equation 2.3. The body frame of sensor two is not aligned with the world frame for $\theta_1 = \theta_2 = 0$. Thus, in order to align both frames, an offset of 90° is required. With $\theta = \theta_1 + \theta_2 + 90°$, this yields

$$\mathbf{T}_y(\theta_1 + \theta_2 + 90°) = \begin{bmatrix} \cos(\theta_1+\theta_2+90°) & 0 & -\sin(\theta_1+\theta_2+90°) \\ 0 & 1 & 0 \\ \sin(\theta_1+\theta_2+90°) & 0 & \cos(\theta_1+\theta_2+90°) \end{bmatrix}. \tag{4.11}$$

According to Equation 2.2, the rotated radial and tangential components of the motion-based acceleration estimates in the body frame of accelerometer 2, A_{X_2} and A_{Z_2}, are found by pre-multiplying the transformation matrix with the acceleration vector in the world frame, constituted of the results of Equations 4.9 and 4.10.

$$\mathbf{a} = \begin{bmatrix} a_{X_2} \\ a_{Y_2} \\ a_{Z_2} \end{bmatrix} = \mathbf{T}_y(\theta_1 + \theta_2 + 90°) \begin{bmatrix} a_x \\ 0 \\ a_z \end{bmatrix} \|\mathbf{g}\|^{-1} \tag{4.12}$$

The axial component A_{Y_2} is zero, since the leg is assumed to be oriented perpendicular to the earth's surface and thus the gravity vector $\mathbf{g}$ is perpendicular to the y-axis. The term $\|\mathbf{g}\|^{-1}$ normalises the rotated motion-based acceleration estimate to gravity, where $\|\mathbf{g}\|$ denotes the magnitude of gravity.

Then, the motion based radial and tangential acceleration components are subtracted from the sensor readings $A_{X_1 m}$ and $A_{Z_1 m}$, which leaves an estimate of the gravity based acceleration $\mathbf{g}$ that acts on the sensor:

$$\mathbf{g} = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} \approx \begin{bmatrix} a_{X_2 m} \\ 0 \\ a_{Z_2 m} \end{bmatrix} - \begin{bmatrix} a_{X_2} \\ 0 \\ a_{Z_2} \end{bmatrix}. \tag{4.13}$$

The angle estimate of the shank with respect to the x-axis, based on the projection of the gravity vector on the axes of the accelerometer, is then

$$\theta_1 + \theta_2 = \text{atan2}(g_z, g_x) - 180^\circ\,. \tag{4.14}$$

This improved angle estimate is then fused with the estimate based on the integration of the angular rate measured with the gyroscope, in order to reduce the estimation error due to gyroscope drift.

4.2.2 *The Extended Kalman Filter*

The State-Space Model

The state-space model of the extended Kalman filter is given by the state vector $\mathbf{x} \in \mathbb{R}^{n=10}$

$$\mathbf{x} = \begin{bmatrix} x, & z, & \theta_1, & \omega_1, & \alpha_1, & \theta_2, & \omega_2, & \alpha_2, & \beta_1, & \beta_2 \end{bmatrix}^T \tag{4.15}$$

where x and y correspond to the horizontal and vertical position of the end of link 2 with respect to the origin of the world frame, i. e. the hip joint. θ_1 is the angle, ω_1 the angular velocity, and α_1 the angular acceleration of the first joint, respectively. The corresponding values for the second link are θ_2, ω_2, and α_2. The biases from the gyroscope on the first and second sensor are β_1 and β_2, respectively. They are assumed to be constant or slowly time varying.

The measurement vector $\mathbf{z} \in \mathbb{R}^{m=3}$ is given by

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} \omega_1 + \beta_1, & \omega_1 + \omega_2 + \beta_1 + \beta_2, & \theta_1 + \theta_2 \end{bmatrix}^T + \mathbf{v}\,. \tag{4.16}$$

where $\mathbf{v}$ is the random measurement noise process, modelled as zero-mean, Gaussian white noise. The element z_1 represents the measurement of the first link angular velocity, which is the sum of the first link rotation and the gyroscope 1 bias. The element z_2 represents the measurement of the second link angular velocity, which is the sum of the first and second link rotation and the bias of gyroscope 1 and gyroscope 2. Finally, the element z_3 is the angle estimate of the second accelerometer, which will see the angular displacement of both links.

According to Rowell [48], the plant dynamics of a system can be expressed as a set of n coupled first-order ordinary differential equations, known as the *state equations*. The modelled system is governed by the *non-linear* first-order ordinary differential equations

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) + \mathbf{w} = \begin{bmatrix} -l_1\omega_1 \sin(\theta_1) - l_2(\omega_1+\omega_2)\sin(\theta_1+\theta_2) \\ -l_1\omega_1 \cos(\theta_1) - l_2(\omega_1+\omega_2)\cos(\theta_1+\theta_2) \\ \omega_1 \\ \alpha_1 \\ 0 \\ \omega_2 \\ \alpha_2 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \mathbf{w}, \qquad (4.17)$$

where $\dot{\mathbf{x}}$ consists of the component-wise time derivatives of the state vector $\mathbf{x}$, expressed in terms of the state variables $x_1(t), \ldots, x_n(t)$. Its elements are given by

$$\dot{x}_i = f_i(x_1(t), \ldots, x_n(t), t) + w_i = \frac{dx_i}{dt} + w_i, \quad i = 1, \ldots, n, \qquad (4.18)$$

The noise term $\mathbf{w}$, modelled as zero-mean, Gaussian white noise again represents the uncertainty in the model. Given this *state-space representation*, the system state at any instant may be interpreted as a point in an n-dimensional state space whose axes are the state variables. The dynamic state response $\mathbf{x}(t)$ can be interpreted as a trajectory traced out in the state space. The system described by Equation 4.17 is *time-invariant*, since it does not depend explicitly on time. Thus, we may leave out the t and write from now on $\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}, t)$.

For a *linear* system in state-space form given by

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x}, \qquad (4.19)$$

with a time-invariant *system dynamics matrix* $\mathbf{F}$ there is a *state transition matrix* $\mathbf{\Phi}(t - t_0)$ that propagates the state of the system forward from any time t_0 to a time t, according to

$$\mathbf{x}(t) = \mathbf{\Phi}(t - t_0)\mathbf{x}(t_0)\,. \qquad (4.20)$$

The solution to the system described by Equation 4.19 is

$$\mathbf{x}(t) = e^{\mathbf{F}(t-t_0)}\mathbf{x}(t_0)\,, \qquad (4.21)$$

where $\mathbf{x}(t_0)$ is an integration constant. As outlined in [37], the state transition matrix can be found by a Taylor-series expansion of $e^{\mathbf{F}(t-t_0)}$,

$$\begin{aligned}\mathbf{\Phi}(t-t_0) = e^{\mathbf{F}(t-t_0)} &= \sum_{k=0}^{\infty} \frac{\mathbf{F}^n[t-t_0]^n}{k!} \\ &= \mathbf{I}_n + \mathbf{F}[t-t_0] \\ &\quad + \frac{\mathbf{F}^2[t-t_0]^2}{2!} + \frac{\mathbf{F}^3[t-t_0]^3}{3!} + \cdots ,\end{aligned} \tag{4.22}$$

where $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix. Truncating the Taylor series after the first order terms yields the *linear approximation* of the fundamental matrix:

$$\mathbf{\Phi}(t-t_0) \approx \mathbf{I}_n + \mathbf{F}[t-t_0] \, . \tag{4.23}$$

The discrete fundamental matrix that propagates the state of the system forward from time step k to $k+1$ can be found by substituting T_s for $t-t_0$, which yields

$$\mathbf{\Phi}_k = \mathbf{\Phi}(T_s) \approx \mathbf{I}_n + \mathbf{F}T_s \, , \tag{4.24}$$

where T_s is the sampling period. In the *linear* case $\mathbf{\Phi}_k$ is constant.

Because the state equations of our system are *non-linear*, a first-order approximation of the system dynamics matrix $\mathbf{F}$ is used, given by the Jacobian of $\mathbf{f}(\mathbf{x})$

$$\begin{aligned}\mathbf{F}_k^{[1]} = \mathbf{J}_\mathbf{f} &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & A & C & 0 & E & G & 0 & 0 & 0 \\ 0 & 0 & B & D & 0 & F & H & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{\mathbf{x}=\hat{\mathbf{x}}_k} ,\end{aligned} \tag{4.25}$$

with

$$\begin{aligned}
A &= -l_1\omega_1\cos(\theta_1) - l_2(\omega_1+\omega_2)\cos(\theta_1+\theta_2)\,, \\
B &= +l_1\omega_1\sin(\theta_1) + l_2(\omega_1+\omega_2)\sin(\theta_1+\theta_2)\,, \\
C &= -l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2)\,, \\
D &= -l_1\cos(\theta_1) - l_2\cos(\theta_1+\theta_2)\,, \\
E &= -l_2(\omega_1+\omega_2)\cos(\theta_1+\theta_2)\,, \\
F &= +l_2(\omega_1+\omega_2)\sin(\theta_1+\theta_2)\,, \\
G &= -l_2\sin(\theta_1+\theta_2)\,, \\
H &= -l_2\cos(\theta_1+\theta_2)\,.
\end{aligned}$$

The partial derivatives are evaluated at the state estimate $\hat{\mathbf{x}}_k$. The discrete state transition matrix must be recomputed every time step. Here the subscript k denotes the state transition matrix that propagates the state at time step k to time step $k+1$. It is given by

$$\mathbf{\Phi}_k^{[1]} \approx \mathbf{I}_n + \mathbf{F}_k^{[1]} T_s\,. \tag{4.26}$$

The Filter Algorithm

The estimate $\hat{\mathbf{x}}_{k-1}$ can be propagated forward to the *a priori* estimate $\hat{\mathbf{x}}_k^-$ by integrating the *non-linear* differential equations at each sampling interval. Applying Euler integration Equation 3.33 yields

$$\begin{aligned}
\hat{\mathbf{x}}_k^- &= \boldsymbol{\phi}_{k-1}(\hat{\mathbf{x}}_{k-1}, 0) \\
&= \hat{\mathbf{x}}_{k-1} + \mathbf{f}(\hat{\mathbf{x}}_{k-1}) T_s\,,
\end{aligned} \tag{4.27}$$

where T_s is the integration interval. The control input $\mathbf{u}_k$ is equal to zero since the system does not have any inputs. A higher-order numerical integration procedure would not improve the *a priori* estimate since the function $\mathbf{f}(\mathbf{x})$ is constant over time.

The relation between the states and the measurements is *linear*, according to Equation 3.17. The measurement matrix $\mathbf{H} \in \mathbb{R}^{3\times 10}$ is given by

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}\,. \tag{4.28}$$

The constant process noise covariance matrix $\mathbf{Q} \in \mathbb{R}^{10\times 10}$ is given by

$$\begin{aligned}
\mathbf{Q} &= \begin{bmatrix} \mathrm{Cov}(w_1,w_1) & \mathrm{Cov}(w_1,w_2) & \cdots & \mathrm{Cov}(w_1,w_n) \\ \mathrm{Cov}(w_2,w_1) & \mathrm{Cov}(w_2,w_2) & \cdots & \mathrm{Cov}(w_2,w_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(w_n,w_1) & \mathrm{Cov}(w_n,w_2) & \cdots & \mathrm{Cov}(w_n,w_n) \end{bmatrix} \\
&= \begin{bmatrix}
\sigma_d^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \sigma_d^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{\sigma_{\theta_1}^{18}}{9} & \frac{\sigma_{\theta_1}^{8}}{4} & \frac{\sigma_{\theta_1}^{10}}{5} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{\sigma_{\theta_1}^{8}}{4} & \frac{\sigma_{\theta_1}^{6}}{3} & \frac{\sigma_{\theta_1}^{4}}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{\sigma_{\theta 1}^{10}}{5} & \frac{\sigma_{\theta_1}^{4}}{2} & \sigma_{\theta_1}^2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{\sigma_{\theta_2}^{18}}{9} & \frac{\sigma_{\theta_2}^{8}}{4} & \frac{\sigma_{\theta_2}^{10}}{5} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{\sigma_{\theta_2}^{8}}{4} & \frac{\sigma_{\theta_2}^{6}}{3} & \frac{\sigma_{\theta_2}^{4}}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{\sigma_{\theta_2}^{10}}{5} & \frac{\sigma_{\theta_2}^{4}}{2} & \sigma_{\theta_2}^2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_\beta^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_\beta^2
\end{bmatrix}.
\end{aligned} \tag{4.29}$$

The diagonal elements represent the respective variances of the elements $w_1, \dots, w_n$ of the process noise vector $\mathbf{w}$, due to the relation $\mathrm{Cov}(w_i, w_i) = \mathrm{Var}(w_i)$. The other elements are the covariances of all possible pairs of the random variables of the process noise vector. The noise processes interfering with the state variables x and y, and β_1 and β_2, respectively, were modelled as independent. In contrast, the covariances of the noise components interfering with the state variables $\theta_i, \omega_i, \alpha_i, i \in 1, 2$, which account for the block-diagonal structure, reflect a random walk process, that is, the integration of a signal perturbed by Gaussian white noise. A detailed derivation of the form of the elements is found in [49]. The parameters $\sigma_d, \sigma_{\theta_1}, \sigma_{\theta_2}$, and σ_β were determined by tuning for the best performance.

The measurement noise covariance matrix $\mathbf{R} \in \mathbb{R}^{3\times 3}$, is given by

$$\mathbf{R} = \begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 \\ 0 & 0 & 0 & \sigma_4^2 \end{bmatrix}, \tag{4.30}$$

The parameters σ_1 and σ_2 are constant. They are determined by computing the sample standard deviation of the measurement data during an initialisation stage of $T_{\text{init}} = 2\,\text{s}$ seconds, while the subject stands still. The standard deviation of a finite data set with n samples $x_1, x_2, \ldots, x_n$ is given by

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(x_k - \mu)^2}, \text{ where } \mu = \frac{1}{n}\sum_{k=1}^{n} x_k\,. \tag{4.31}$$

with

$$n = T_{\text{init}} f_s\,. \tag{4.32}$$

The standard deviations σ_3 and σ_4 of the accelerometer-based angle estimates are set dynamically based on the motion intensity. It toggles between σ_s and σ_f according to slow or fast motion, distinguished by a marker signal m:

$$\sigma_3^2 = \begin{cases} \sigma_{3s}^2 & m_k = 0 \\ \sigma_{3f}^2 & m_k = 1 \end{cases}. \tag{4.33}$$

$$\sigma_4^2 = \begin{cases} \sigma_{4s}^2 & m_k = 0 \\ \sigma_{4f}^2 & m_k = 1 \end{cases}. \tag{4.34}$$

The value of a marker signal toggles between 1 and 0. In order to determine the marker signal we used the long term spectral detector LTSD developed in [39], which distinguishes between fast and slow motion by computing the long term spectral envelope of the signal.

Summary of the Entire Filter Algorithm

The state estimates are computed recursively according to the *time update* Equations 3.36, 4.27 and the *measurement update* Equations 3.23, 3.24, and 3.26. The entire computation steps of the recursive filter algorithm are summarised in Figure 4.6. The filter algorithm was implemented in

MATLAB®. Listing 5.2 in the appendix shows the source code of the filter function.

4.3 EXPERIMENTS

The orientation algorithm was tested with data from real subjects. The movement data was gathered at the Department of Neurology of the Klinikum Großhadern in Munich, while the subjects performed the following trial.

4.3.1 *Data Collection Protocol*

The subject stood on a treadmill wearing the GaitWatch system on its body. Then, the GaitWatch record was started and shortly afterwards the treadmill was turned on. After a variable time depending on the treadmill speed and the walking distance, first the treadmill was switched off, then the GaitWatch records. The experiments were carried out at 2 km/h, 4 km/h, and 6 km/h.

4.3.2 *Initial Conditions*

Each trial began with the subject standing still and it was assumed that the subjects legs are fully stretched. This leads to the following initial state estimates:

$$\begin{aligned} x &= 0\,\mathrm{m}\,, & z &= -(l_1 + l_2)\,\mathrm{m}\,, \\ \theta_1 &= -90^\circ\,, & \theta_2 &= 0\,^\circ\,, \\ \omega_1 &= 0\,\frac{^\circ}{\mathrm{s}}\,, & \omega_2 &= 0\,\frac{^\circ}{\mathrm{s}}\,, & m_0 &= 0 \\ \beta_1 &= \mu_1\,\frac{^\circ}{\mathrm{s}}\,, & \beta_2 &= \mu_2\,\frac{^\circ}{\mathrm{s}}\,, \\ \alpha_1 &= 0\,\frac{^\circ}{\mathrm{s}^2}\,, & \alpha_2 &= 0\,\frac{^\circ}{\mathrm{s}^2}\,, \end{aligned} \tag{4.35}$$

where μ_1 and μ_2 are the mean values of the gyroscope signals collected during the initial $T_{\mathrm{init}} = 2\,\mathrm{s}$ of the rest period before the subject started moving. The mean value is given by

$$\mu = \frac{1}{n}\sum_{k=1}^{n} \omega_k\,, \quad n = T_{\mathrm{init}} f_s\,, \tag{4.36}$$

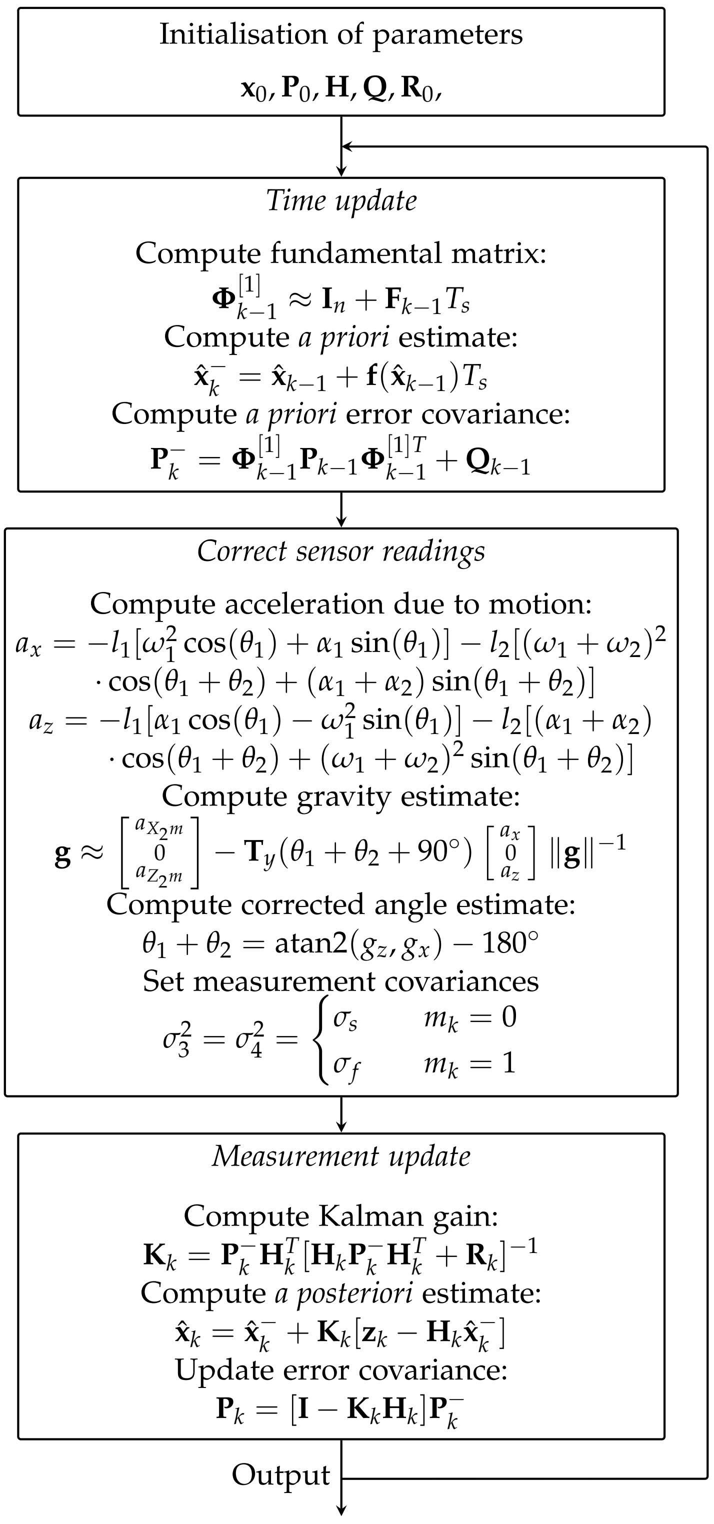
Initialisation of parameters
$\mathbf{x}_0, \mathbf{P}_0, \mathbf{H}, \mathbf{Q}, \mathbf{R}_0,$
Time update
Compute fundamental matrix:
$\mathbf{\Phi}^{[1]}_{k-1} \approx \mathbf{I}_n + \mathbf{F}_{k-1} T_s$
Compute *a priori* estimate:
$\hat{\mathbf{x}}^-_k = \hat{\mathbf{x}}_{k-1} + \mathbf{f}(\hat{\mathbf{x}}_{k-1}) T_s$
Compute *a priori* error covariance:
$\mathbf{P}^-_k = \mathbf{\Phi}^{[1]}_{k-1} \mathbf{P}_{k-1} \mathbf{\Phi}^{[1]T}_{k-1} + \mathbf{Q}_{k-1}$
Correct sensor readings
Compute acceleration due to motion:
$a_x = -l_1[\omega_1^2 \cos(\theta_1) + \alpha_1 \sin(\theta_1)] - l_2[(\omega_1 + \omega_2)^2 \cdot \cos(\theta_1 + \theta_2) + (\alpha_1 + \alpha_2) \sin(\theta_1 + \theta_2)]$
$a_z = -l_1[\alpha_1 \cos(\theta_1) - \omega_1^2 \sin(\theta_1)] - l_2[(\alpha_1 + \alpha_2) \cdot \cos(\theta_1 + \theta_2) + (\omega_1 + \omega_2)^2 \sin(\theta_1 + \theta_2)]$
Compute gravity estimate:
$\mathbf{g} \approx \begin{bmatrix} a_{X_2 m} \\ 0 \\ a_{Z_2 m} \end{bmatrix} - \mathbf{T}_y(\theta_1 + \theta_2 + 90^\circ) \begin{bmatrix} a_x \\ 0 \\ a_z \end{bmatrix} \|\mathbf{g}\|^{-1}$
Compute corrected angle estimate:
$\theta_1 + \theta_2 = \text{atan2}(g_z, g_x) - 180^\circ$
Set measurement covariances
$\sigma_3^2 = \sigma_4^2 = \begin{cases} \sigma_s & m_k = 0 \\ \sigma_f & m_k = 1 \end{cases}$
Measurement update
Compute Kalman gain:
$\mathbf{K}_k = \mathbf{P}^-_k \mathbf{H}^T_k [\mathbf{H}_k \mathbf{P}^-_k \mathbf{H}^T_k + \mathbf{R}_k]^{-1}$
Compute *a posteriori* estimate:
$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}^-_k + \mathbf{K}_k[\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}^-_k]$
Update error covariance:
$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k \mathbf{H}_k] \mathbf{P}^-_k$
Output

Figure 4.6: Entire computation steps of the recursive filter algorithm.

where ω_k is the angular velocity at instant k and f_s is the sampling frequency.

The initial error covariance matrix $\mathbf{P}_0$ was given by

$$\mathbf{P}_0 = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} . \tag{4.37}$$

Its diagonal elements, that is, the variances of the initial state estimates, represent the confidence in the knowledge about the initial state of the system and is crucial for the convergence of the filter. Unlike the initial error covariance matrix of the classical Kalman filter, it may cause the filter to diverge, in case the confidence in the estimates is too small. Equally, too vague initial state estimates can cause the filter to diverge.

4.3.3 *Test Preparation*

After the implementation of the filter algorithm, I implemented a parameter optimiser. Based on the optimal set of parameters for a given signal, we compared the newly implemented filter algorithm against the classical Kalman filter. The parameter optimisation function given by Listing 5.2 called an adaptive Nelder-Mead simplex algorithm, which minimised the error function E. Since we are interested in an accurate angle estimate of both angles and thus need parameters that optimise both angle estimates, we scalarise this multi-objective optimisation problem. Using linear scalarisation yields

$$\begin{aligned} \mathrm{E}(\hat{\theta}_t, \hat{\theta}_s) &= \mathrm{RMSE}_{\hat{\theta}_t} + \mathrm{RMSE}_{\hat{\theta}_s} \\ &= \sqrt{\frac{\sum_{k=1}^{n}(\hat{\theta}_{t,k} - \theta_{t,k})^2}{n}} + \sqrt{\frac{\sum_{k=1}^{n}(\hat{\theta}_{s,k} - \theta_{s,k})^2}{n}} , \end{aligned} \tag{4.38}$$

	2 m/s	4 m/s	6 m/s
σ_d^2	10.0000	10.0000	10.0000
$\sigma_{\theta_1}^2$	0.0431	0.0116	0.4314
$\sigma_{\theta_2}^2$	0.0431	0.0116	0.4314
σ_β^2	0.0011	0.0001	0.0008
σ_1^2	2.1767	3.6758	2.7102
σ_2^2	2.9788	3.2287	1.7409
σ_{3s}^2	3.6014	2.9168	5.2650
σ_{3f}^2	29.4645	125.4298	62.8998
σ_{4s}^2	27.5862	18.0040	72.5493
σ_{4f}^2	312.8627	227.0883	232.8577

Table 4.1: Filter parameters.

where $\hat{\theta}_{t,k}$ and $\hat{\theta}_{s,k}$ denote the thigh and shank angle at instant k, estimated by the extended Kalman filter, and $\theta_{t,k}$ and $\theta_{s,k}$ the reference angles at instant k, respectively. The implementation of the error function is shown in Listing 5.2. The optimiser returned the parameters that minimise the error function.

Parameterisation

The parameters for the three different walking speeds in Table 4.1 were determined by the optimisation routines. In order to improve the legibility, all parameters are normalised to their units. That is, since the variance is the square of the standard deviation, they are normalised to the square of the unit of the corresponding element of the state vector. For instance, the variance σ_d^2 of the displacement is normalised to m^2.

4.3.4 *Test Execution*

4.4 RESULTS

This section presents the results obtained by the experiments. Listing **??** the script used to run the experiments. Figures **??** and **??** show.

	2 m/s		4 m/s		6 m/s	
	KF	EKF	KF	EKF	KF	EKF
$\text{RMSE}_{\hat{\theta}_t}$	1.5324	1.0767	2.3658	1.9455	10.1907	8.2013
$\text{RMSE}_{\hat{\theta}_s}$	4.4460	3.3153	3.3271	2.4269	14.5714	9.3883
SUM	5.9784	4.3920	5.6928	4.3724	24.7621	17.5895

Table 4.2: Root-mean-square errors of the Kalman filter and the extended Kalman filter.

4.5 DISCUSSION

5 CONCLUSION AND FUTURE WORK

5.1 CONCLUSIONS

Gait analysis is a useful tool both in clinical practice and biomechanical research. In this work I implemented an extended Kalman filter along with some auxiliary routines that . and showed that motion-based acceleration correction can improve the angle estimates during faster motion. In order to replace a camera-based motion capture system with low cost wearable MARG sensors, some technical deails need to be improved. One could use a more complex mathematical model of the leg that takes motions outside

Summarising the above, I can say that I have learned a lot in the four month that I spent in Granada. Amongst others I have come to know many new work methods, not only due to being exposed to people from a different culture, but also due to the fact that scientific research differs strongly from the work as a student at university. I gained a deeper understanding of orientation estimation and how Kalman filtering benefits the accuracy. I improved my MATLAB® skills and I am now familiar with tools such as GitHub and Pivotal Tracker which make working in a team much easier and significantly more efficient. While working at the research centre, I obtained a valuable insight into scientific research and could improve my oral and written English skills. Furthermore I now know the fundamentals of LaTeX.

All in all it was a great experience, professionally as well as personally. I truly and unreservedly recommend such a stay to *every* university student.

5.2 FUTURE WORK

BIBLIOGRAPHY

[1] Weijun Tao, Tao Liu, Rencheng Zheng, and Hutian Feng. Gait analysis using wearable sensors. *Sensors*, 12(2):2255–2283, February 2012. doi: 10.3390/s120202255. URL http://www.mdpi.com/1424-8220/12/2/2255.

[2] Paolo Bonato. Advances in wearable technology and applications in physical medicine and rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 2:2–2, 2005. ISSN 1743-0003. doi: 10.1186/1743-0003-2-2. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC552335/.

[3] Wee-Soon Yeoh, I. Pek, Yi-Han Yong, Xiang Chen, and A.B. Waluyo. Ambulatory monitoring of human posture and walking speed using wearable accelerometer sensors. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, pages 5184–5187, Aug 2008. doi: 10.1109/IEMBS.2008.4650382.

[4] A. Godfrey, R. Conway, D. Meagher, and G. ÓLaighin. Direct measurement of human movement by accelerometry. *Medical Engineering & Physics*, 30(10):1364–1386, December 2008. ISSN 1350-4533. doi: 10.1016/j.medengphy.2008.09.005. URL http://www.sciencedirect.com/science/article/pii/S1350453308001653.

[5] Terrell Bennett, Roozbeh Jafari, and Nicholas Raphael Gans. An extended Kalman filter to estimate human gait parameters and walking distance. September 2013. URL http://www.essp.utdallas.edu/uploads/Main/Publications/ACC13.pdf.

[6] Wai Yin Wong, Man Sang Wong, and Kam Ho Lo. Clinical Applications of Sensors for Human Posture and Movement Analysis: A Review. *Prosthetics and Orthotics International*, 31(1):62–75, March

2007. ISSN 0309-3646, 1746-1553. doi: 10.1080/03093640600983949. URL http://poi.sagepub.com/content/31/1/62.

[7] T.R. Bennett, R. Jafari, and N. Gans. Motion Based Acceleration Correction for Improved Sensor Orientation Estimates. In *2014 11th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 109–114, June 2014. doi: 10.1109/BSN.2014.17.

[8] H.J. Luinge, P.H. Veltink, and C.T.M. Baten. Estimation of orientation with gyroscopes and accelerometers. In *[Engineering in Medicine and Biology, 1999. 21st Annual Conference and the 1999 Annual Fall Meetring of the Biomedical Engineering Society] BMES/EMBS Conference, 1999. Proceedings of the First Joint*, volume 2, pages 844 vol.2–, Oct 1999. doi: 10.1109/IEMBS.1999.803999.

[9] Alberto Olivares Vicente. *Signal Processing of Magnetic and Inertial Sensor's Signals applied to Human Body Motion Monitoring*. PhD thesis, University of Granada, Granada, January 2013. URL http://hera.ugr.es/tesisugr/21910947.pdf.

[10] P.H. Veltink, H.B.J. Bussmann, W. de Vries, Wim L.J. Martens, and R.C. van Lummel. Detection of static and dynamic activities using uniaxial accelerometers. *Rehabilitation Engineering, IEEE Transactions on*, 4(4):375–385, Dec 1996. ISSN 1063-6528. doi: 10.1109/86.547939.

[11] B. Najafi, K. Aminian, A. Paraschiv-Ionescu, F. Loew, C.J. Bula, and P. Robert. Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. *Biomedical Engineering, IEEE Transactions on*, 50(6):711–723, June 2003. ISSN 0018-9294. doi: 10.1109/TBME.2003.812189.

[12] M. Ermes, J. Parkka, J. Mantyjarvi, and I. Korhonen. Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions. *Information Technology in Biomedicine, IEEE Transactions on*, 12(1):20–26, Jan 2008.

[13] O. Giggins, D. Kelly, and B. Caulfield. Evaluating rehabilitation exercise performance using a single inertial measurement unit. In *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2013 7th International Conference on*, pages 49–56, May 2013.

[14] J. Lupinski, V. Menon, P. Roh, S. Yuen, A. Valdevit, and J. Andrish. Measuring knee compliance to facilitate post-op ligament rehabilitation. In *Bioengineering Conference (NEBEC), 2011 IEEE 37th Annual Northeast*, pages 1–2, April 2011. doi: 10.1109/NEBC.2011.5778529.

[15] V. Bonnet, C. Mazza, P. Fraisse, and A. Cappozzo. Real-time estimate of body kinematics during a planar squat task using a single inertial measurement unit. *Biomedical Engineering, IEEE Transactions on*, 60 (7):1920–1926, July 2013. ISSN 0018-9294. doi: 10.1109/TBME.2013.2245131.

[16] Rui Zhang, F. Hoflinger, and L. Reindl. Inertial sensor based indoor localization and monitoring system for emergency responders. *Sensors Journal, IEEE*, 13(2):838–848, Feb 2013. ISSN 1530-437X. doi: 10.1109/JSEN.2012.2227593.

[17] T. Bennett, R. Jafari, and N. Gans. An extended kalman filter to estimate human gait parameters and walking distance. In *American Control Conference (ACC), 2013*, pages 752–757, June 2013. doi: 10.1109/ACC.2013.6579926.

[18] A. K. Bourke and G. M. Lyons. A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. *Medical Engineering & Physics*, 30(1):84–90, January 2008. ISSN 1350-4533. doi: 10.1016/j.medengphy.2006.12.001. URL `http://www.sciencedirect.com/science/article/pii/S1350453306002657`.

[19] A.K. Bourke, P. van de Ven, M. Gamble, R. O'Connor, K. Murphy, E. Bogan, E. McQuade, P. Finucane, G. OLaighin, and J. Nelson. Assessment of waist-worn tri-axial accelerometer based fall-detection algorithms using continuous unsupervised activities. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 2782–2785, Aug 2010. doi: 10.1109/IEMBS.2010.5626364.

[20] A.K. Bourke, P.W.J. van de Ven, A. Chaya, G. Olaighin, and J. Nelson. Design and test of a long-term fall detection system incorporated into a custom vest for the elderly. In *Signals and Systems Conference, 208. (ISSC 2008). IET Irish*, pages 307–312, June 2008.

[21] M. Mancini, C. Zampieri, P. Carlson-Kuhta, L. Chiari, and F. B. Horak. Anticipatory postural adjustments prior to step initiation are hypometric in untreated Parkinson's disease: an accelerometer-based approach. *European Journal of Neurology: The Official Journal of the European Federation of Neurological Societies*, 16(9):1028–1034, September 2009. ISSN 1468-1331. doi: 10.1111/j.1468-1331.2009.02641.x.

[22] L. Palmerini, L. Rocchi, S. Mellone, F. Valzania, and L. Chiari. Feature selection for accelerometer-based posture analysis in Parkinson's disease. *IEEE Transactions on Information Technology in Biomedicine*, 15(3):481–490, May 2011. ISSN 1089-7771. doi: 10.1109/TITB.2011.2107916.

[23] D. G. M. de Klerk, J. P. P. van Vugt, J. a. G. Geelen, and T. Heida. A long-term monitor including activity classification for motor assessment in Parkinson's disease patients. In Jos Vander Sloten, Pascal Verdonck, Marc Nyssen, and Jens Haueisen, editors, *4th European Conference of the International Federation for Medical and Biological Engineering*, number 22 in IFMBE Proceedings, pages 1706–1709. Springer Berlin Heidelberg, January 2009. ISBN 978-3-540-89207-6, 978-3-540-89208-3. URL `http://link.springer.com/chapter/10.1007/978-3-540-89208-3_406`.

[24] Kaveh Saremi, Jon Marehbian, Xiaohong Yan, Jean-Philippe Regnaux, Robert Elashoff, Bernard Bussel, and Bruce H. Dobkin. Reliability and Validity of Bilateral Thigh and Foot Accelerometry Measures of Walking in Healthy and Hemiparetic Subjects. *Neurorehabilitation and Neural Repair*, 20(2):297–305, June 2006. ISSN 1545-9683, 1552-6844. doi: 10.1177/1545968306287171. URL `http://nnr.sagepub.com/content/20/2/297`.

[25] Joe Verghese, Richard B. Lipton, Charles B. Hall, Gail Kuslansky, Mindy J. Katz, and Herman Buschke. Abnormality of gait as a predictor of non-alzheimer's dementia. *New England Journal of Medicine*, 347(22):1761–1768, 2002. doi: 10.1056/NEJMoa020441. URL `http://dx.doi.org/10.1056/NEJMoa020441`. PMID: 12456852.

[26] Jussi Collin, Pavel Davidson, Martti Kirkko-Jaakkola, and Helena Leppäkoski. Inertial sensors and their applications. In Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo

Takala, editors, *Handbook of Signal Processing Systems*, pages 69–96. Springer New York, 2013. ISBN 978-1-4614-6858-5. doi: 10.1007/978-1-4614-6859-2_3. URL `http://dx.doi.org/10.1007/978-1-4614-6859-2_3`.

[27] M.N. Armenise, C. Ciminelli, F. Dell'Olio, and V.M.N. Passaro. *Advances in Gyroscope Technologies*. Springer, 2010. ISBN 9783642154942. URL `https://books.google.de/books?id=lJUiyigJRBgC`.

[28] J. Lenz and Alan S. Edelstein. Magnetic sensors and their applications. *Sensors Journal, IEEE*, 6(3):631–649, June 2006. ISSN 1530-437X. doi: 10.1109/JSEN.2006.874493.

[29] M.J. Thompson, M. Li, and D.A. Horsley. Low power 3-axis Lorentz force navigation magnetometer. In *Micro Electro Mechanical Systems (MEMS), 2011 IEEE 24th International Conference on*, pages 593–596, Jan 2011. doi: 10.1109/MEMSYS.2011.5734494.

[30] Juansempere. Wikimedia Commons File: Taitbrianzyx.svg. URL `http://commons.wikimedia.org/wiki/File:Taitbrianzyx.svg`. [Accessed 31 March, 2015].

[31] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. Technical report, Stanford University, 2006.

[32] J.R. Raol. *Multi-Sensor Data Fusion with MATLAB®*. CRC Press, 2009. ISBN 9781439800058. URL `https://books.google.de/books?id=7s8xpR-5r0UC`.

[33] Simon Haykin. *Adaptive filter theory*. Prentice Hall, Upper Saddle River, N.J, 2002. ISBN 9780130484345.

[34] Maybeck. Chapter 1 introduction. In Peter S. Maybeck, editor, *Stochastic Models, Estimation and Control: Volume 1*, Mathematics in Science and Engineering, pages 1 – 24. Elsevier, 1979. doi: 10.1016/S0076-5392(08)62166-9. URL `http://www.sciencedirect.com/science/article/pii/S0076539208621669`.

[35] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[36] Gregory F. Welch. Kalman filter. In Katsushi Ikeuchi, editor, *Computer Vision*, pages 435–437. Springer US, 2014. ISBN 978-0-387-30771-8. doi: 10.1007/978-0-387-31439-6_716. URL `http://dx.doi.org/10.1007/978-0-387-31439-6_716`.

[37] Paul Zarchan and Howard Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. Progress in astronautics and aeronautics. Academic Press, 2009. ISBN 9781600867187. URL `https://books.google.es/books?id=LUDKrQEACAAJ`.

[38] M.S. Grewal and A.P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. Wiley, 2008. ISBN 9780470377802. URL `http://books.google.es/books?id=J_fqMHOCzB8C`.

[39] Alberto Olivares Vicente and Kai Bötzel. *GaitWatch User Manual*.

[40] SparkFun Electronics. *IMU Analog Combo Board 5DOF*, . URL `https://www.sparkfun.com/products/retired/9268`. [Accessed 17 March 2015].

[41] SparkFun Electronics. *Gyro Breakout Board IDG500*, . URL `https://www.sparkfun.com/products/retired/9094`. [Accessed 17 March 2015].

[42] Analog Devices. *ADXL335: Small, Low Power 3-Axis ±3G Accelerometer*, . URL `http://www.analog.com/en/mems-sensors/mems-inertial-sensors/adxl335/products/product.html`. [Accessed 17 March 2015].

[43] Analog Devices. *ADXL345: 3-Axis, ±2g/±4g/±8g/±16g Digital Accelerometer*, . URL `http://www.analog.com/en/mems-sensors/mems-inertial-sensors/adxl345/products/product.html`. [Accessed 17 March 2015].

[44] InvenSense. *IMU-3000 Triple Axis MotionProcessor™Gyroscope*. URL `http://www.invensense.com/mems/gyro/imu3000.html`. [Accessed 17 March 2015].

[45] SparkFun Electronics. *MicroMag 3-Axis Magnetometer*, . URL `https://www.sparkfun.com/products/retired/244`. [Accessed 17 March 2015].

[46] ALVIDI. *AVR ATxmega – Development Module*. URL `http://www.alvidi.de/avr_xmodul_en.html`. [Accessed 17 March 2015].

[47] M.W. Spong and S. Hutchinson. *Robot Modeling and Control*. Wiley, 2005. ISBN 9780471649908. URL `http://books.google.de/books?id=wGapQAAACAAJ`.

[48] Derek Rowell. State-space representation of LTI systems. 2002. URL `http://web.mit.edu/2.14/www/Handouts/StateSpace.pdf`.

[49] Alonzo Kelly. A 3d state space formulation of a navigation kalman filter for autonomous vehicles. Technical Report CMU-RI-TR-94-19, Robotics Institute, Pittsburgh, PA, May 1994.

APPENDIX

MATLAB CODE

Listing 1: MATLAB® code file fusion_EKF.m

```
function [theta1, theta2, theta12_c, a_m, X, par] = ...
            fusion_EKF(gyro_thigh_y, gyro_shank_y, ...
                       acc_thigh_x, acc_thigh_z, ...
                       acc_shank_x, acc_shank_z, ...
                       fs, l1, l2, p)

% FUNCTION fusion_EKF applies an extended Kalman filter
% in order to fuse the accelerometer and gyroscope data
% and thus obtain an accurate orientation estimate of
% the thighs and shanks.
%
% Input arguments:
% |_ 'gyro_thigh_y':  Row vector containing the angular
%                     rate of the thigh about the
%                     y-axis in radians per second.
% |_ 'gyro_shank_y':  Row vector containing the angular
%                     rate of the shank about the
%                     y-axis in radians per second.
% |_ 'acc_thigh_x':   Row vector containing the linear
%                     acceleration of the thigh along
%                     the x-axis in g.
% |_ 'acc_thigh_z':   Row vector containing the linear
%                     acceleration of the thigh along
%                     the z-axis in g.
% |_ 'acc_shank_x':   Row vector containing the linear
%                     acceleration of the shank along
%                     the x-axis in g.
% |_ 'acc_shank_z':   Row vector containing the linear
%                     acceleration of the shank along
%                     the z-axis in g.
% |_ 'fs':            Sampling frecuency in Hertz. Must
%                     be real positive.
% |_ 'l1':            Length of the thigh in m. Must be
%                     real positive.
% |_ 'l2':            Length of the shank in m. Must be
%                     real positive.
% |_ 'p':             Row vector consisting of the
%                     filter parameters sigma_t1,
%                     sigma_t2, sigma_b, sigma_f_1,
%                     sigma_f_2, sigma_s_1, sigma_s_2.
%
% Output:
% |_ 'theta1':        Row vector containing the thigh
%                     angle with respect to the x-axis
%                     of the world frame in radians.
% |_ 'theta2':        Row vector containing the shank
%                     angle with respect to the thigh
%                     in radians.
% |_ 'theta2':        Row vector containing the shank
%                     angle with respect to the thigh
%                     in radians.
% |_ 'a_m':           Row vector containing the
%                     estimate of the acceleration that
%                     sensor 2 will see due to motion.
% |_ 'X':             10 x length(gyro_thigh_y) matrix
%                     containing the internal state
%                     vector at each instant as columns.
%
% IMPORTANT NOTE:     gyro_thigh_y, gyro_shank_y,
%                     acc_thigh_x, acc_thigh_z,
%                     acc_shank_x, and acc_shank_z
%                     must have the same length.
%                     Otherwise, an error will
%                     be returned.
% -----------------------------------------------------
% Authors:            Robin Weiss
% Entity:             University of Applied Sciences
%                     Munster, Munster, Germany
% Last modification:  13/05/2015
% -----------------------------------------------------

% 1) Check input arguments.
if ~isequal(length(gyro_thigh_y), ...
            length(gyro_shank_y), ...
            length(acc_thigh_x), ...
            length(acc_thigh_z), ...
            length(acc_shank_x), ...
            length(acc_shank_z))
    error(['Input␣arguments␣''gyro_thigh_y'',␣', ...
           '''acc_thigh_x'',␣''acc_thigh_z'',␣', ...
           '''acc_shank_x'',␣''acc_shank_z'',␣', ...
           'must␣have␣the␣same␣length.']);
end

if (fs <= 0 || ~isreal(fs))
    error(['Input␣argument␣''fs''␣must␣be␣real', ...
           'positive.']);
end

if (l1 <= 0 || ~isreal(l1))
    error(['Input␣argument␣''a1''␣must␣be␣real', ...
          '␣positive.']);
end

if (l2 <= 0 || ~isreal(l2))
    error(['Input␣parameter␣''a2''␣must␣be␣real', ...
           '␣positive.']);
end

% 2) Import GaitWatch and WaGyroMag functions library.
%    All existing functions have to be called using
%    either 'gw.functionName' or 'wag.functionName'.
gw = gwLibrary;
wag = wagLibrary;

% 3) Compute the sampling period and the length of the
%    signal vectors.
Ts = 1 / fs;
len = length(gyro_thigh_y);
```

```
% 4) Set the magnitude of gravity.
gravity = 9.81;

% 5) Compute correction factor. This factor will be
%    used throughout the entire code in order to
%    convert degrees into radians.
c_w = pi / 180;

% 6) Initialise output vectors.
theta1 = zeros(1, len);
theta2 = zeros(1, len);
theta12_c = zeros(1, len);
a_m = zeros(3, len);
X = zeros(10, len);

% 7) Compute intensity level.
lwin_ltsd = 20;
threshold_ltsd = 4;
shift_ltsd = 19;
input_signal = sqrt(acc_shank_x.^2+acc_shank_z.^2);
[V_fsd, T_fsd] = wag.ltsd(input_signal', lwin_ltsd, ...
                      shift_ltsd, 512, threshold_ltsd);

% 8) Determine marker signal.
[marker, ~] = gw.compEstMark(V_fsd, T_fsd, ...
                          input_signal, lwin_ltsd, ...
                          shift_ltsd);

% INITIALISATION OF PARAMETERS %

% 9) Compute mean of the first two seconds of the
%    gyroscope signals.
mu1 = mean(gyro_thigh_y(1:2*fs));
mu2 = mean(gyro_shank_y(1:2*fs));

% 10) Initialise the state vector.
x = [0, -(l1+l2), -85, 0, 0, -5, 0, 0, mu1, mu2]';

% 11) Initialise the error covariance matrix.
P = diag(ones(1, 10) * 1);
```

```
% 12) Define the measurement matrix.
H = [0 0 0 1 0 0 0 0 1 0; ...
     0 0 0 1 0 0 1 0 1 1; ...
     0 0 1 0 0 0 0 0 0 0; ...
     0 0 1 0 0 1 0 0 0 0];

% 13) Define process noise covariance matrix.
sigma_d_sq = 10;
sigma_t1_sq = p(6);
sigma_t2_sq = p(6);
sigma_b_sq = p(5);
Q = [...
sigma_d_sq 0 0           0              0        0 0 0 0 0; ...
0 sigma_d_sq 0           0              0        0 0 0 0 0; ...
0 0 sigma_t1_sq^9/9 sigma_t1_sq^4/4 sigma_t1_sq^5/5 0 0 0 0 0; ...
0 0 sigma_t1_sq^4/4 sigma_t1_sq^3/3 sigma_t1_sq^2/2 0 0 0 0 0; ...
0 0 sigma_t1_sq^5/5 sigma_t1_sq^2/2   sigma_t1_sq   0 0 0 0 0; ...
0 0 0 0 0 sigma_t2_sq^9/9 sigma_t2_sq^4/4 sigma_t2_sq^5/5 0 0; ...
0 0 0 0 0 sigma_t2_sq^4/4 sigma_t2_sq^3/3 sigma_t2_sq^2/2 0 0; ...
0 0 0 0 0 sigma_t2_sq^5/5 sigma_t2_sq^2/2   sigma_t2_sq   0 0; ...
0 0 0 0 0       0              0          0 sigma_b_sq 0; ...
0 0 0 0 0       0              0          0 0 sigma_b_sq];

% 14) Compute sample variance of the first two
%     seconds of the gyroscope signals.
sigma_1_sq = var(gyro_thigh_y(1:2*fs));
sigma_2_sq = var(gyro_shank_y(1:2*fs));

% 15) Define measurement noise covariance matrix.
sigma_s3_sq = p(1);
sigma_s4_sq = p(2);
sigma_f3_sq = p(3);
sigma_f4_sq = p(4);
R = [sigma_1_sq    0       0       0; ...
        0    sigma_2_sq    0       0; ...
        0       0    sigma_s3_sq   0; ...
        0       0       0   sigma_s4_sq];

% Map parameter vector to output
par = [sigma_d_sq, sigma_t1_sq, sigma_t2_sq, ...
       sigma_b_sq, sigma_1_sq, sigma_2_sq, ...
```

```
        sigma_s3_sq, sigma_f3_sq, sigma_s4_sq, ...
        sigma_f4_sq];

% 16) Define matrix function f.
function f_k = f

         f_k = [- l1 * c_w * x(4) * sind(x(3)) ...
            - l2 * c_w * (x(4) + x(7)) ...
            * sind(x(3) + x(6)); ...
            - l1 * c_w * x(4) * cosd(x(3)) ...
            - l2 * c_w * (x(4) + x(7)) ...
            * cosd(x(3) + x(6)); ...
            x(4);
            x(5);
            0;
            x(7);
            x(8);
            0;
            0;
            0];

end

% 17) Define Jacobian of F.
function F_k = F

    A = - l1 * c_w * x(4) * cosd(x(3)) ...
        - l2 * c_w * (x(4) + x(7)) * cosd(x(3) + x(6));
    B = + l1 * c_w * x(4) * sind(x(3)) ...
        + l2 * c_w * (x(4) + x(7)) * sind(x(3) + x(6));
    C = - l1 * sind(x(3)) - l2 * sind(x(3) + x(6));
    D = - l1 * cosd(x(3)) - l2 * cosd(x(3) + x(6));
    E = - l2 * c_w * (x(4) + x(7)) * cosd(x(3) + x(6));
    F = + l2 * c_w * (x(4) + x(7)) * sind(x(3) + x(6));
    G = - l2 * sind(x(3) + x(6));
    h = - l2 * cosd(x(3) + x(6));

    F_k = [0, 0, A, C, 0, E, G, 0, 0, 0; ...
           0, 0, B, D, 0, F, h, 0, 0, 0; ...
           0, 0, 0, 1, 0, 0, 0, 0, 0, 0; ...
           0, 0, 0, 0, 1, 0, 0, 0, 0, 0; ...
```

```
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0; ...
              0, 0, 0, 0, 0, 0, 1, 0, 0, 0; ...
              0, 0, 0, 0, 0, 0, 0, 1, 0, 0; ...
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0; ...
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0; ...
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

end

% 18) Filter loop.
for i=1:1:len

    % TIME UPDATE %

    % Compute fundamental matrix.
    Phi = eye(10) + F * Ts;

    % Compute a priori state estimate.
    x = x + f * Ts;

    % Compute a priori error covariance matrix.
    P = Phi * P * Phi' + Q;

    % CORRECT SENSOR READINGS %

    % Compute acceleration in the world frame that
    % occurs due to motion, based on a priori state
    % estimate.
    ax = - l1 * ((c_w * x(4))^2 * cosd(x(3)) ...
         + c_w * x(5) * sind(x(3))) ...
         - l2 * ((c_w * x(4) + c_w * x(7))^2 ...
         * cosd(x(3) + x(6)) ...
         + c_w * (x(5) + x(8)) * sind(x(3) + x(6)));
    az = - l1 * (c_w * x(5) * cosd(x(3)) ...
         - (c_w * x(4))^2 * sind(x(3))) ...
         - l2 * (c_w * (x(5) + x(8)) ...
         * cosd(x(3) + x(6)) ...
         - (c_w * x(4) + c_w * x(7))^2 ...
         * sind(x(3) + x(6)));

    % Normalise acceleration to gravity.
    ax_n = ax / gravity;
    az_n = az / gravity;

    % Compute transformation matrix
    Tz = [cosd(x(3) + x(6) + 90), 0, ...
         -sind(x(3) + x(6) + 90); 0, 1, 0; ...
          sind(x(3) + x(6) + 90), 0, ...
          cosd(x(3) + x(6) + 90)];

    % Rotate acceleration to body frame.
    a_mb = Tz * [ax_n; 0; az_n];

    % Compute gravity estimate by subtracting
    % motion-based acceleration from sensor readings.
    g_c = [acc_shank_x(i); 0; acc_shank_z(i)] - a_mb;

    % Constitute the measurement vector from the
    % gyroscope signals, theta_1, and the corrected
    % angle estimate theta_1 + theta_2.
    z = [gyro_thigh_y(i); gyro_shank_y(i); 0; 0];
    z(3) = atan2d(acc_thigh_z(i), acc_thigh_x(i)) - 180;
    z(4) = atan2d(g_c(3), g_c(1)) - 180;

    % Output map.
    theta12_c(i) = z(4);
    a_m(:, i) = a_mb;

    % Set sigma_3 in measurement noise covariance
    % matrix according to motion intensity.
    if marker(i) == 1
       R(3, 3) = sigma_f3_sq;
       R(4, 4) = sigma_f4_sq;
    end

    if marker(i) == 0
       R(3, 3) = sigma_s3_sq;
       R(4, 4) = sigma_s4_sq;
    end

    % MEASUREMENT UPDATE %

    % Compute Kalman gain.
    K = P * H' / (H * P * H' + R);

    % Compute a posteriori estimate.
    x = x + K * (z - H * x);

    % Update error covariance matrix.
    P = (eye(10) - K * H) * P;

    % Map internal states to output vector.
    theta1(i) = x(3);
    theta2(i) = x(6);

    % Map the entire state vector to the output.
    X(:, i) = x;

end

end
```

Listing 2: MATLAB® code file optimise_EKF.m

```
function [x_min, f_min, ct] = optimize_EKF( ...
                        gyro_thigh_y, gyro_shank_y, ...
                        acc_thigh_x, acc_thigh_z, ...
                        acc_shank_x, acc_shank_z, ...
                        fs, l1, l2, ref_angles, p0, ...
                        rmse_off)

% FUNCTION OPTIMIZE_EKF uses an adaptive Nelder-Mead
% simplex ANMS algorithm to find the optimal parameters
% of the Extended Kalman filter.
%
% Input arguments:
% |_ 'gyro_thigh_y':  Row vector containing the angular
%                     rate of the thigh about the
%                     y-axis in radians per second.
% |_ 'gyro_shank_y':  Row vector containing the angular
%                     rate of the shank about the
%                     y-axis in radians per second.
% |_ 'acc_thigh_x':   Row vector containing the linear
%                     acceleration of the thigh along
%                      the x-axis in g.
% |_ 'acc_thigh_z':    Row vector containing the linear
%                      acceleration of the thigh along
%                      the z-axis in g.
% |_ 'acc_shank_x':    Row vector containing the linear
%                      acceleration of the shank along
%                      the x-axis in g.
% |_ 'acc_shank_z':    Row vector containing the linear
%                      acceleration of the shank along
%                      the z-axis in g.
% |_ 'fs':             Sampling frecuency in Hertz. Must
%                      be real positive.
% |_ 'l1':             Length of the thigh in m. Must be
%                      real positive.
% |_ 'l2':             Length of the shank in m. Must be
%                      real positive.
% |_ 'p':              Row vector consisting of the
%                      filter parameters sigma_t1,
%                      sigma_t2, sigma_b, sigma_f_1,
%                      sigma_f_2, sigma_s_1, sigma_s_2.
% |_ 'ref_angle':      Orientation angle reference.
% |_ 'p0':             Initial value of the parameters
%                      to be optimized.
% |_ 'rmse_off':       Offset in the RMSE computation.
%
% % Output:
% |_ 'xmin':           Value of the parameters that
%                      minimize the error function.
% |_ 'fmin':           Minimum value of the error
%                      function.
% |_ 'ct':             Number of algorithm iterations
%                      to find the minimum.

% 1) Set variables.
% -----------------------------------------------------
global gyro_thigh_y_g; gyro_thigh_y_g = gyro_thigh_y;
global gyro_shank_y_g; gyro_shank_y_g = gyro_shank_y;
global acc_thigh_x_g; acc_thigh_x_g = acc_thigh_x;
global acc_thigh_z_g; acc_thigh_z_g = acc_thigh_z;
global acc_shank_x_g; acc_shank_x_g = acc_shank_x;
global acc_shank_z_g; acc_shank_z_g = acc_shank_z;
global fs_g; fs_g = fs;
global l1_g; l1_g = l1;
global l2_g; l2_g = l2;
global true_angles;    true_angles = ref_angles;
global rmse_offset;   rmse_offset = rmse_off;

% 2) Call the minimisation routine.
% ---------------------------------------------------------
disp('Optimising␣parameters␣of␣extended␣Kalman␣Filter...');

% Set tolerance limit between the minimum values found
% in subsequent iterations of the algorithm.
tol = 10^-6;

% Set maximum number of evaluations of the error
% function.
max_feval = 5000;

% Call ANMS algorithm which minimises the error function.
[x_min, f_min, ct] = ANMS(@eofEKF, p0, tol, max_feval);

end
```

Listing 3: MATLAB® code file eofEKF.m

```
function F = eofEKF(p)

% FUNCTION EOFKALMAN is the error function to be
% minimised: That is, the sum of the RMSE between the
% actual angle and the estiamted orientation angle
% computed with the extended Kalman filter.
%
% Input arguments:
% |_ 'p':    Vector of initial value of the parameters
%            to be optimized.
%
% Output:
% |_ 'F':    Value of the error function.

% 1) Set variables.
% ---------------------------------------------------------
global gyro_thigh_y_g;
global gyro_shank_y_g;
global acc_thigh_x_g;
global acc_thigh_z_g;
global acc_shank_x_g;
global acc_shank_z_g;
global fs_g;
global l1_g;
global l2_g;
global true_angles;
global rmse_offset;

% 3) Estimate the orientation angle using the extended
%    Kalman Filter.
[theta1, theta2, ~, ~, ~] = fusion_EKF(...
                   gyro_thigh_y_g, gyro_shank_y_g, ...
                   acc_thigh_x_g, acc_thigh_z_g, ...
                   acc_shank_x_g, acc_shank_z_g, ...
                   fs_g, l1_g, l2_g, p);

thigh_angle_EKF = theta1;
shank_angle_EKF = theta1 + theta2;

% 4) Compute the error function.
F1 = sqrt(mean((true_angles(1, rmse_offset : end) -  ...
     thigh_angle_EKF(rmse_offset : end)) .^ 2));
F2 = sqrt(mean((true_angles(2, rmse_offset : end) -  ...
     shank_angle_EKF(rmse_offset : end)) .^ 2));

F = F1 + F2;
```

Listing 4: MATLAB® code file EKF_experiments_1.m

```
%% 0) Initialisation \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

clear all; close all; clc;

% Generate Tikz-pictures: Yes (1), or no (0).
tikz = 1;

% Load existing signals of the GaitWatch and the
% Qualisys motion capture system.
```

```
load('GaitWatch_data_1.mat');
load('Qualisys_data_1.mat');
load('pmin_1.mat');

% Import GaitWatch and WaGyroMag functions library.
% All existing functions have to be called using
% either 'gw.functionName' or 'wag.functionName'.
gw = gwLibrary;
wag = wagLibrary;

% Initialise number of figure.
n = 4;

% Set value of the magnitude of the gravity vector.
g = 9.81;

% Set the first and the last sample, that is, the
% interval of the signal that is used for the following
% computations.
n1 = 1;
n2 = 24 * f;

%% 1) Optimise filter parameters \\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

% Set RMSE offset.
rmse_offset = 1;

% ---KALMAN FILTER THIGH-------------------------------

% Set initial value of parameters;
p0_KF = [1000 0.001];

% Call the optimization routine.
[xmin, fmin, ct] = gw.optimize_KF( ...
            g_Y_right_thigh_1_C(n1:n2)', ...
            pitch_acc_right_thigh(n1:n2)', ...
            f, var(pitch_acc_right_thigh(n1:n2)), ...
            var(pitch_acc_right_thigh(n1:n2)), ...
            var(g_Y_right_thigh_1_C(n1:n2)), ...
            pitch_acc_right_thigh(1), ...
```

```
              pitch_QS_right_thigh(n1:n2), ...
              p0_KF, rmse_offset);

fprintf('-------------KF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-Alpha:␣%0.4f', ...
        '\n␣-Beta:␣%0.4f\n'], xmin(1), xmin(2))
fprintf('----------------------------------------\n')

% Extract optimal parameters.
opt_alpha_KF = xmin(1);
opt_beta_KF = xmin(2);

% Compute thigh angle.
pitch_KF_right_thigh = gw.fusion_KF( ...
        g_Y_right_thigh_1_C, pitch_acc_right_thigh, ...
        f, var(pitch_acc_right_thigh), ...
        var(pitch_acc_right_thigh),...
        var(g_Y_right_thigh_1_C), opt_alpha_KF, ...
        opt_beta_KF, pitch_acc_right_thigh(1));

% ---KALMAN FILTER SHANK-------------------------------

% Set initial value of parameters;
p0_KF = [1000 0.001];

% Call the optimization routine.
[xmin, fmin, ct] = gw.optimize_KF( ...
                g_Y_right_shank_1_C(n1:n2)', ...
                pitch_acc_right_shank(n1:n2)', f, ...
                var(pitch_acc_right_shank(n1:n2)), ...
                var(pitch_acc_right_shank(n1:n2)),...
                var(g_Y_right_shank_1_C(n1:n2)), ...
                pitch_acc_right_shank(1), ...
                pitch_QS_right_shank(n1:n2), ...
                p0_KF, rmse_offset);

fprintf('-------------KF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-Alpha:␣%0.4f', ...
        '\n␣-Beta:␣%0.4f\n'], xmin(1), xmin(2))
fprintf('----------------------------------------\n')

% Extract optimal parameters.
opt_alpha_KF = xmin(1);
opt_beta_KF = xmin(2);

% Compute shank angle.
pitch_KF_right_shank = gw.fusion_KF( ...
        g_Y_right_shank_1_C, pitch_acc_right_shank, ...
        f, var(pitch_acc_right_shank), ...
        var(pitch_acc_right_shank),...
        var(g_Y_right_shank_1_C), opt_alpha_KF, ...
        opt_beta_KF, pitch_acc_right_shank(1));

% ---EXTENDED KALMAN FILTER----------------------------

% Set initial value of parameters.
p0 = [3.5, 30, 30, 300, 0.001, 0.05];

 % Call the optimization routine.
[pmin, fmin, ct] = gw.optimize_EKF( ...
                    g_Y_right_thigh_1_C(n1:n2)', ...
                    g_Y_right_shank_1_C(n1:n2)', ...
                    a_X_right_thigh_1_C(n1:n2)', ...
                    a_Z_right_thigh_1_C(n1:n2)', ...
                    a_X_right_shank_1_C(n1:n2)', ...
                    a_Z_right_shank_1_C(n1:n2)', ...
                    f, 0.35, 0.25, ...
                    [pitch_QS_right_thigh(n1:n2); ...
                     pitch_QS_right_shank(n1:n2)] - 90, ...
                    p0, rmse_offset);

fprintf('-------------EKF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-sigma_t1:␣', ...
          '%0.4f\n␣-sigma_t2:␣%0.4f\n␣-sigma_b:␣', ...
          '%0.4f\n␣-sigma_s_1:␣%0.4f\n␣-sigma_s_2:', ...
          '␣%0.4f'], pmin);
fprintf('----------------------------------------\n')
%%
% Compute pitch angles with extended Kalman filter.
% Additionally, store the internal state vector at each
% time step in x, the motion based acceleration in a_m,
% and the angle estimate theta_1 + theta_2, based on
% the corrected acceleration signal in theta12_c.
[pitch_EKF_right_thigh, pitch_EKF_right_shank, ...
 theta12_c, a_m, x, par] = fusion_EKF( ...
                            g_Y_right_thigh_1_C', ...
                            g_Y_right_shank_1_C', ...
                            a_X_right_thigh_1_C', ...
                            a_Z_right_thigh_1_C', ...
                            a_X_right_shank_1_C', ...
                            a_Z_right_shank_1_C', ...
                            f, 0.35, 0.25, pmin);

save ('Data_1/pmin_1', 'pmin')
save ('Data_1/parameters_1', 'par')

%% 3) Plot results \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

% Plot: Thigh angle estimate - acceleration-based,
%       KF, and EKF.
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_thigh(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_thigh(n1:n2) - 90);
plot(time(n1:n2), pitch_KF_right_thigh(n1:n2) - 90, ...
     time(n1:n2), pitch_EKF_right_thigh(n1:n2), ...
     'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter', 'latex');
ylabel(['Pitch␣angle␣$\theta_1$␣in␣', ...
        '$^{\circ}$'], 'interpreter','latex');
legend('Reference', 'Accelerometer-based', ...
       'Kalman␣filter', 'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

n = n + 1;

% Plot: Shank angle estimate - acceleration-based,
%       KF, and EKF.
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_shank(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_shank(n1:n2)-90);
plot(time(n1:n2), pitch_KF_right_shank(n1:n2)-90, ...
     time(n1:n2), pitch_EKF_right_thigh(n1:n2) ...
     + pitch_EKF_right_shank(n1:n2), 'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter', 'latex');
ylabel(['Pitch␣angle␣$\theta_1␣+␣\theta_2$␣in␣', ...
        '$^{\circ}$'], 'interpreter','latex');
legend('Reference', 'Accelerometer-based', ...
       'Kalman␣filter', 'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

n = n + 1;

% Compute root-mean-square error.
% -Thigh
RMSE_acc = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
                - pitch_acc_right_thigh(n1:n2)').^2));
RMSE_KF = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
               - pitch_KF_right_thigh(n1:n2)').^2));
```

```
RMSE_EKF = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
             - 90 - pitch_EKF_right_thigh(n1:n2)).^2));
RMSE = [RMSE_acc, RMSE_KF, RMSE_EKF];
% -Shank
RMSE_acc = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
                - pitch_acc_right_shank(n1:n2)').^2));
RMSE_KF = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
               - pitch_KF_right_shank(n1:n2)').^2));
RMSE_EKF = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
             - 90 - pitch_EKF_right_thigh(n1:n2) ...
             - pitch_EKF_right_shank(n1:n2)).^2));
RMSE = [RMSE; RMSE_acc, RMSE_KF, RMSE_EKF;];

% Compute sum of seperate RMSEs.
RMSE = [RMSE; RMSE(1, 1) + RMSE(2, 1), RMSE(1, 2) + ...
        RMSE(2, 2), RMSE(1, 3) + RMSE(2, 3)];

% Plot bar graph and values on top of the bars.
figure(n);
b = bar(RMSE, 0.3);
offset = 0.8;
yb = cat(1, b.YData);
xb = bsxfun(@plus, b(1).XData, [b.XOffset]');
hold on;

for i = 1:3
   for j = 1:3
        text(xb(j, i), yb(j, i) + offset, ...
            ['\scriptsize␣', num2str(RMSE(i, j), ...
            '$%0.2f$')], 'rotation', 0, ...
            'interpreter', 'latex', ...
            'HorizontalAlignment','center');
   end
end

b(1).FaceColor = [0.8500    0.3250    0.0980];
b(2).FaceColor = [0.9290    0.6940    0.1250];
b(3).FaceColor = [0.4940    0.1840    0.5560];

text(1.4, 18, ['\scriptsize␣$\frac{\operatorname', ...
        '{RMSE}_{EKF_{thigh}}␣+␣\operatorname', ...
        '{RMSE}_{EKF_{shank}}}{\operatorname', ...
        '{RMSE}_{KF_{thigh}}␣+␣\operatorname', ...
        '{RMSE}_{KF_{shank}}}␣=␣', ...
        num2str((RMSE(1, 3) + RMSE(2, 3)) / ...
        (RMSE(1, 2) + RMSE(2, 2)), '%0.2f$')], ...
        'interpreter','latex');

ylim([0, max(max(RMSE)) + 2]);
ylabel('Root-mean-square␣error␣in␣$^{\circ}$', ...
       'interpreter','latex');

labels = {'Thigh', 'Shank', 'Thigh␣+␣Shank'};
format_ticks(gca, labels, [], [], [], 0);

legend('Acceleration-based', 'Kalman␣filter', ...
          'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

n = n + 1;

% Plot: Acceleration-based pitch angle shank - corrected.
n1 = 4 * f + 1;
n2 = 20 * f;
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_shank(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_shank(n1:n2) - 90);
plot(time(n1:n2), theta12_c(n1:n2), 'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter','latex');
ylabel(['Pitch␣angle␣$\theta_1␣+␣\theta_2$␣in␣', ...
        '$^{\circ}$'], 'interpreter', 'latex');
legend('Reference', 'Accelerometer-based', ...
       'Accelerometer␣based␣-␣corrected');
```

```
% Compute root-mean-square error.
RMSE_acc = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
                - pitch_acc_right_shank(n1:n2)').^2));
RMSE_acc_corr = sqrt(mean((pitch_QS_right_shank(n1:n2)...
                - 90 - theta12_c(n1:n2)).^2));
RMSE_acc = [RMSE_acc, RMSE_acc_corr];

text(4.5,-130, ['\scriptsize␣$\operatorname{RMSE}', ...
                '_{Acc}␣=␣', num2str(RMSE_acc(1), ...
                '%0.2f$')], 'interpreter','latex');
text(4.5,-140, ['\scriptsize␣$\operatorname{RMSE}', ...
                '_{Acc-corrected}␣=␣', ...
                num2str(RMSE_acc(2), '%0.2f$')], ...
                'interpreter','latex');
text(4.5,-150, ['\scriptsize␣$\frac{\operatorname', ...
                '{RMSE}_{Acc-corrected}}', ...
                '{\operatorname{RMSE}_{Acc}}␣=␣', ...
                num2str(RMSE_acc(2) / RMSE_acc(1), ...
                '%0.2f$')], 'interpreter','latex');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

% Display the results.
fprintf('-----------------Results---------------\n');
fprintf(['AB:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 1));
fprintf(['KF:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 2));
fprintf(['EKF:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 3));
fprintf(['EKF␣parameters:␣\n␣-sigma_d_sq:␣␣', ...
         '%0.4f\n␣-sigma_t1_sq:␣', ...
         '%0.4f\n␣-sigma_t2_sq:␣', ...
         '%0.4f\n␣-sigma_b_sq:␣␣', ...
         '%0.4f\n␣-sigma_1_sq:␣␣', ...
         '%0.4f\n␣-sigma_2_sq:␣␣', ...
         '%0.4f\n␣-sigma_s3_sq:␣', ...
         '%0.4f\n␣-sigma_f3_sq:␣', ...
         '%0.4f\n␣-sigma_s4_sq:␣', ...
         '%0.4f\n␣-sigma_f4_sq:␣', ...
         '%0.4f\n'], par);
fprintf('----------------------------------------\n');
```

Listing 5: MATLAB® code file EKF_experiments_2.m

```
%% 0) Initialisation \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

clear all; close all; clc;

% Generate Tikz-pictures: Yes (1), or no (0).
tikz = 1;

% Load existing signals of the GaitWatch and the
% Qualisys motion capture system.
load('GaitWatch_data_2.mat');
load('Qualisys_data_2.mat');
load('pmin_2.mat');

% Import GaitWatch and WaGyroMag functions library.
% All existing functions have to be called using
% either 'gw.functionName' or 'wag.functionName'.
gw = gwLibrary;
wag = wagLibrary;

% Initialise number of figure.
n = 8;

% Set value of the magnitude of the gravity vector.
g = 9.81;

% Set the first and the last sample, that is, the
% interval of the signal that is used for the following
% computations.
n1 = 1;
n2 = 24 * f;

%% 1) Optimise filter parameters \\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

% Set RMSE offset.
rmse_offset = 1;

% ---KALMAN FILTER THIGH-------------------------------

% Set initial value of parameters;
p0_KF = [1000 0.001];

% Call the optimization routine.
[xmin, fmin, ct] = gw.optimize_KF( ...
            g_Y_right_thigh_1_C(n1:n2)', ...
            pitch_acc_right_thigh(n1:n2)', ...
            f, var(pitch_acc_right_thigh(n1:n2)), ...
            var(pitch_acc_right_thigh(n1:n2)), ...
            var(g_Y_right_thigh_1_C(n1:n2)), ...
            pitch_acc_right_thigh(1), ...
            pitch_QS_right_thigh(n1:n2), ...
            p0_KF, rmse_offset);

fprintf('-------------KF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-Alpha:␣%0.4f', ...
        '\n␣-Beta:␣%0.4f\n'], xmin(1), xmin(2))
fprintf('----------------------------------------\n')

% Extract optimal parameters.
opt_alpha_KF = xmin(1);
opt_beta_KF = xmin(2);

% Compute thigh angle.
pitch_KF_right_thigh = gw.fusion_KF( ...
        g_Y_right_thigh_1_C, pitch_acc_right_thigh, ...
        f, var(pitch_acc_right_thigh), ...
        var(pitch_acc_right_thigh),...
        var(g_Y_right_thigh_1_C), opt_alpha_KF, ...
        opt_beta_KF, pitch_acc_right_thigh(1));

% ---KALMAN FILTER SHANK-------------------------------

% Set initial value of parameters;
p0_KF = [1000 0.001];

% Call the optimization routine.
[xmin, fmin, ct] = gw.optimize_KF( ...
                g_Y_right_shank_1_C(n1:n2)', ...
                pitch_acc_right_shank(n1:n2)', f, ...
                var(pitch_acc_right_shank(n1:n2)), ...
                var(pitch_acc_right_shank(n1:n2)),...
                var(g_Y_right_shank_1_C(n1:n2)), ...
                pitch_acc_right_shank(1), ...
                pitch_QS_right_shank(n1:n2), ...
                p0_KF, rmse_offset);

fprintf('-------------KF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
          'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-Alpha:␣%0.4f', ...
         '\n␣-Beta:␣%0.4f\n'], xmin(1), xmin(2))
fprintf('---------------------------------------\n')

% Extract optimal parameters.
opt_alpha_KF = xmin(1);
opt_beta_KF = xmin(2);

% Compute shank angle.
pitch_KF_right_shank = gw.fusion_KF( ...
        g_Y_right_shank_1_C, pitch_acc_right_shank, ...
        f, var(pitch_acc_right_shank), ...
        var(pitch_acc_right_shank),...
        var(g_Y_right_shank_1_C), opt_alpha_KF, ...
        opt_beta_KF, pitch_acc_right_shank(1));

% ---EXTENDED KALMAN FILTER----------------------------

% % Set initial value of parameters.
% p0 = [3.5, 30, 30, 300, 0.001, 0.05];
%
%  % Call the optimization routine.
% [pmin, fmin, ct] = gw.optimize_EKF( ...
%                      g_Y_right_thigh_1_C(n1:n2)', ...
%                      g_Y_right_shank_1_C(n1:n2)', ...
%                      a_X_right_thigh_1_C(n1:n2)', ...
%                      a_Z_right_thigh_1_C(n1:n2)', ...
%                      a_X_right_shank_1_C(n1:n2)', ...
%                      a_Z_right_shank_1_C(n1:n2)', ...
%                      f, 0.35, 0.25, ...
%                      [pitch_QS_right_thigh(n1:n2); ...
%                       pitch_QS_right_shank(n1:n2)] - 90, ...
%                      p0, rmse_offset);
%
% fprintf('-------------EKF OPTIMISATION------------\n');
% fprintf(['The optimisation process finished in %d ', ...
%          'iterations.\n'], ct);
% fprintf('The minimum RMSE found is: %0.4f\n', fmin);
% fprintf(['Optimal parameters are: \n -sigma_t1: ', ...
%          '%0.4f\n -sigma_t2: %0.4f\n -sigma_b: ', ...
%          '%0.4f\n -sigma_s_1: %0.4f\n -sigma_s_2:', ...
%          ' %0.4f'], pmin);
% fprintf('----------------------------------------\n')
load('pmin_2.mat')
%%
% Compute pitch angles with extended Kalman filter.
% Additionally, store the internal state vector at each
% time step in x, the motion based acceleration in a_m,
% and the angle estimate theta_1 + theta_2, based on
% the corrected acceleration signal in theta12_c.
[pitch_EKF_right_thigh, pitch_EKF_right_shank, ...
 theta12_c, a_m, x, par] = fusion_EKF( ...
                           g_Y_right_thigh_1_C', ...
                           g_Y_right_shank_1_C', ...
                           a_X_right_thigh_1_C', ...
                           a_Z_right_thigh_1_C', ...
                           a_X_right_shank_1_C', ...
                           a_Z_right_shank_1_C', ...
                           f, 0.35, 0.25, pmin);

save ('Data_2/pmin_2', 'pmin')
save ('Data_2/parameters_2', 'par')

```

```
%% 3) Plot results \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

% Plot: Thigh angle estimate - acceleration-based,
%       KF, and EKF.
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_thigh(n1:n2) - 90, ...
     ’linewidth’, 1);
plot(time(n1:n2), pitch_acc_right_thigh(n1:n2) - 90);
plot(time(n1:n2), pitch_KF_right_thigh(n1:n2) - 90, ...
     time(n1:n2), pitch_EKF_right_thigh(n1:n2), ...
     ’linewidth’, 1);

xlabel(’Time␣$t$␣in␣s’, ’interpreter’, ’latex’);
ylabel([’Pitch␣angle␣$\theta_1$␣in␣’, ...
        ’$^{\circ}$’], ’interpreter’,’latex’);
legend(’Reference’, ’Accelerometer-based’, ...
       ’Kalman␣filter’, ’Extended␣Kalman␣filter’);

if tikz
matlab2tikz([’../tikz/experiment_’, num2str(n), ...
             ’.tikz’], ’height’, ’\figureheight’, ...
             ’width’, ’\figurewidth’);
end

n = n + 1;

% Plot: Shank angle estimate - acceleration-based,
%       KF, and EKF.
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_shank(n1:n2) - 90, ...
     ’linewidth’, 1);
plot(time(n1:n2), pitch_acc_right_shank(n1:n2)-90);
plot(time(n1:n2), pitch_KF_right_shank(n1:n2)-90, ...
     time(n1:n2), pitch_EKF_right_thigh(n1:n2) ...
     + pitch_EKF_right_shank(n1:n2), ’linewidth’, 1);

xlabel(’Time␣$t$␣in␣s’, ’interpreter’, ’latex’);
ylabel([’Pitch␣angle␣$\theta_1␣+␣\theta_2$␣in␣’, ...
        '$^{\circ}$'], 'interpreter','latex');
legend('Reference', 'Accelerometer-based', ...
       'Kalman␣filter', 'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

n = n + 1;

% Compute root-mean-square error.
% -Thigh
RMSE_acc = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
               - pitch_acc_right_thigh(n1:n2)').^2));
RMSE_KF = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
              - pitch_KF_right_thigh(n1:n2)').^2));
RMSE_EKF = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
            - 90 - pitch_EKF_right_thigh(n1:n2)).^2));
RMSE = [RMSE_acc, RMSE_KF, RMSE_EKF];
% -Shank
RMSE_acc = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
               - pitch_acc_right_shank(n1:n2)').^2));
RMSE_KF = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
              - pitch_KF_right_shank(n1:n2)').^2));
RMSE_EKF = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
            - 90 - pitch_EKF_right_thigh(n1:n2) ...
            - pitch_EKF_right_shank(n1:n2)).^2));
RMSE = [RMSE; RMSE_acc, RMSE_KF, RMSE_EKF;];

% Compute sum of seperate RMSEs.
RMSE = [RMSE; RMSE(1, 1) + RMSE(2, 1), RMSE(1, 2) + ...
        RMSE(2, 2), RMSE(1, 3) + RMSE(2, 3)];

% Plot bar graph and values on top of the bars.
figure(n);
b = bar(RMSE, 0.3);
offset = 0.8;
yb = cat(1, b.YData);
xb = bsxfun(@plus, b(1).XData, [b.XOffset]');
```

```
hold on;

for i = 1:3
   for j = 1:3
        text(xb(j, i), yb(j, i) + offset, ...
            ['\scriptsize␣', num2str(RMSE(i, j), ...
            '$%0.2f$')], 'rotation', 0, ...
            'interpreter', 'latex', ...
            'HorizontalAlignment','center');
   end
end

b(1).FaceColor = [0.8500    0.3250    0.0980];
b(2).FaceColor = [0.9290    0.6940    0.1250];
b(3).FaceColor = [0.4940    0.1840    0.5560];

text(1.4, 26, ['\scriptsize␣$\frac{\operatorname', ...
        '{RMSE}_{EKF_{thigh}}␣+␣\operatorname', ...
        '{RMSE}_{EKF_{shank}}}{\operatorname', ...
        '{RMSE}_{KF_{thigh}}␣+␣\operatorname', ...
        '{RMSE}_{KF_{shank}}}␣=␣', ...
        num2str((RMSE(1, 3) + RMSE(2, 3)) / ...
        (RMSE(1, 2) + RMSE(2, 2)), '%0.2f$')], ...
        'interpreter','latex');

ylim([0, max(max(RMSE)) + 2]);
ylabel('Root-mean-square␣error␣in␣$^{\circ}$', ...
       'interpreter','latex');

labels = {'Thigh', 'Shank', 'Thigh␣+␣Shank'};
format_ticks(gca, labels, [], [], [], 0);

legend('Acceleration-based', 'Kalman␣filter', ...
          'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
            '.tikz'], 'height', '\figureheight', ...
            'width', '\figurewidth');
end
```

```
n = n + 1;

% Plot: Acceleration-based pitch angle shank - corrected.
n1 = 4 * f + 1;
n2 = 20 * f;
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_shank(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_shank(n1:n2) - 90);
plot(time(n1:n2), theta12_c(n1:n2), 'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter','latex');
ylabel(['Pitch␣angle␣$\theta_1␣+␣\theta_2$␣in␣', ...
        '$^{\circ}$'], 'interpreter', 'latex');
legend('Reference', 'Accelerometer-based', ...
       'Accelerometer␣based␣-␣corrected');

% Compute root-mean-square error.
RMSE_acc = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
                - pitch_acc_right_shank(n1:n2)').^2));
RMSE_acc_corr = sqrt(mean((pitch_QS_right_shank(n1:n2)...
                - 90 - theta12_c(n1:n2)).^2));
RMSE_acc = [RMSE_acc, RMSE_acc_corr];

text(6,-45, ['\scriptsize␣$\operatorname{RMSE}', ...
              '_{Acc}␣=␣', num2str(RMSE_acc(1), ...
              '%0.2f$')], 'interpreter','latex');
text(6,-55, ['\scriptsize␣$\operatorname{RMSE}', ...
              '_{Acc-corrected}␣=␣', ...
              num2str(RMSE_acc(2), ...
              '%0.2f$')], 'interpreter','latex');
text(6,-65, ['\scriptsize␣$\frac{\operatorname',...
             '{RMSE}_{Acc-corrected}}', ...
             '{\operatorname{RMSE}_{Acc}}␣=␣', ...
             num2str(RMSE_acc(2) / RMSE_acc(1), ...
             '%0.2f$')], 'interpreter','latex');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

% Display the results.
fprintf('-----------------Results---------------\n');
fprintf(['AB:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 1));
fprintf(['KF:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 2));
fprintf(['EKF:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 3));
fprintf(['EKF␣parameters:␣\n␣-sigma_d_sq:␣␣', ...
         '%0.4f\n␣-sigma_t1_sq:␣', ...
         '%0.4f\n␣-sigma_t2_sq:␣', ...
         '%0.4f\n␣-sigma_b_sq:␣␣', ...
         '%0.4f\n␣-sigma_1_sq:␣␣', ...
         '%0.4f\n␣-sigma_2_sq:␣␣', ...
         '%0.4f\n␣-sigma_s3_sq:␣', ...
         '%0.4f\n␣-sigma_f3_sq:␣', ...
         '%0.4f\n␣-sigma_s4_sq:␣', ...
         '%0.4f\n␣-sigma_f4_sq:␣', ...
         '%0.4f\n'], par);
fprintf('---------------------------------------\n');
```

Listing 6: MATLAB® code file EKF_experiments_3.m

```
%% 0) Initialisation \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
% -----------------------------------------------------

clear all; close all; clc;

% Generate Tikz-pictures: Yes (1), or no (0).
tikz = 1;

% Load existing signals of the GaitWatch and the
% Qualisys motion capture system.
load('GaitWatch_data_3.mat');
load('Qualisys_data_3.mat');
load('pmin_3.mat');

% Import GaitWatch and WaGyroMag functions library.
% All existing functions have to be called using
% either 'gw.functionName' or 'wag.functionName'.
gw = gwLibrary;
wag = wagLibrary;

% Initialise number of figure.
n = 12;

% Set value of the magnitude of the gravity vector.
g = 9.81;

% Set the first and the last sample, that is, the
% interval of the signal that is used for the following
% computations.
n1 = 1;
n2 = 24 * f;

%% 1) Optimise filter parameters \\\\\\\\\\\\\\\\\\\\\\
% ----------------------------------------------------------

% Set RMSE offset.
rmse_offset = 1;

% ---KALMAN FILTER THIGH-------------------------------

% Set initial value of parameters;
p0_KF = [1000 0.001];

% Call the optimization routine.
[xmin, fmin, ct] = gw.optimize_KF( ...
            g_Y_right_thigh_1_C(n1:n2)', ...
            pitch_acc_right_thigh(n1:n2)', ...
            f, var(pitch_acc_right_thigh(n1:n2)), ...
            var(pitch_acc_right_thigh(n1:n2)), ...
            var(g_Y_right_thigh_1_C(n1:n2)), ...
            pitch_acc_right_thigh(1), ...
            pitch_QS_right_thigh(n1:n2), ...
            p0_KF, rmse_offset);

fprintf('-------------KF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
```

```
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-Alpha:␣%0.4f', ...
        '\n␣-Beta:␣%0.4f\n'], xmin(1), xmin(2))
fprintf('----------------------------------------\n')

% Extract optimal parameters.
opt_alpha_KF = xmin(1);
opt_beta_KF = xmin(2);

% Compute thigh angle.
pitch_KF_right_thigh = gw.fusion_KF( ...
        g_Y_right_thigh_1_C, pitch_acc_right_thigh, ...
        f, var(pitch_acc_right_thigh), ...
        var(pitch_acc_right_thigh),...
        var(g_Y_right_thigh_1_C), opt_alpha_KF, ...
        opt_beta_KF, pitch_acc_right_thigh(1));

% ---KALMAN FILTER SHANK-------------------------------

% Set initial value of parameters;
p0_KF = [1000 0.001];

% Call the optimization routine.
[xmin, fmin, ct] = gw.optimize_KF( ...
                g_Y_right_shank_1_C(n1:n2)', ...
                pitch_acc_right_shank(n1:n2)', f, ...
                var(pitch_acc_right_shank(n1:n2)), ...
                var(pitch_acc_right_shank(n1:n2)),...
                var(g_Y_right_shank_1_C(n1:n2)), ...
                pitch_acc_right_shank(1), ...
                pitch_QS_right_shank(n1:n2), ...
                p0_KF, rmse_offset);

fprintf('-------------KF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-Alpha:␣%0.4f', ...
        '\n␣-Beta:␣%0.4f\n'], xmin(1), xmin(2))
fprintf('----------------------------------------\n')
```

```
% Extract optimal parameters.
opt_alpha_KF = xmin(1);
opt_beta_KF = xmin(2);

% Compute shank angle.
pitch_KF_right_shank = gw.fusion_KF( ...
        g_Y_right_shank_1_C, pitch_acc_right_shank, ...
        f, var(pitch_acc_right_shank), ...
        var(pitch_acc_right_shank),...
        var(g_Y_right_shank_1_C), opt_alpha_KF, ...
        opt_beta_KF, pitch_acc_right_shank(1));

% ---EXTENDED KALMAN FILTER----------------------------

% Set initial value of parameters.
p0 = [3.5, 30, 30, 300, 0.001, 0.05];

 % Call the optimization routine.
[pmin, fmin, ct] = gw.optimize_EKF( ...
                    g_Y_right_thigh_1_C(n1:n2)', ...
                    g_Y_right_shank_1_C(n1:n2)', ...
                    a_X_right_thigh_1_C(n1:n2)', ...
                    a_Z_right_thigh_1_C(n1:n2)', ...
                    a_X_right_shank_1_C(n1:n2)', ...
                    a_Z_right_shank_1_C(n1:n2)', ...
                    f, 0.35, 0.25, ...
                    [pitch_QS_right_thigh(n1:n2); ...
                     pitch_QS_right_shank(n1:n2)] - 90, ...
                    p0, rmse_offset);

fprintf('-------------EKF␣OPTIMISATION------------\n');
fprintf(['The␣optimisation␣process␣finished␣in␣%d␣', ...
         'iterations.\n'], ct);
fprintf('The␣minimum␣RMSE␣found␣is:␣%0.4f\n', fmin);
fprintf(['Optimal␣parameters␣are:␣\n␣-sigma_t1:␣', ...
         '%0.4f\n␣-sigma_t2:␣%0.4f\n␣-sigma_b:␣', ...
         '%0.4f\n␣-sigma_s_1:␣%0.4f\n␣-sigma_s_2:', ...
         '␣%0.4f'], pmin);
fprintf('----------------------------------------\n')
%%
% Compute pitch angles with extended Kalman filter.
% Additionally, store the internal state vector at each
% time step in x, the motion based acceleration in a_m,
% and the angle estimate theta_1 + theta_2, based on
% the corrected acceleration signal in theta12_c.
[pitch_EKF_right_thigh, pitch_EKF_right_shank, ...
 theta12_c, a_m, x, par] = fusion_EKF( ...
                            g_Y_right_thigh_1_C', ...
                            g_Y_right_shank_1_C', ...
                            a_X_right_thigh_1_C', ...
                            a_Z_right_thigh_1_C', ...
                            a_X_right_shank_1_C', ...
                            a_Z_right_shank_1_C', ...
                            f, 0.35, 0.25, pmin);

save ('Data_3/pmin_3', 'pmin')
save ('Data_3/parameters_3', 'par')

%% 3) Plot results \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
% -------------------------------------------------------

% Plot: Thigh angle estimate - acceleration-based,
%       KF, and EKF.
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_thigh(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_thigh(n1:n2) - 90);
plot(time(n1:n2), pitch_KF_right_thigh(n1:n2) - 90, ...
     time(n1:n2), pitch_EKF_right_thigh(n1:n2), ...
     'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter', 'latex');
ylabel(['Pitch␣angle␣$\theta_1$␣in␣', ...
        '$^{\circ}$'], 'interpreter','latex');
legend('Reference', 'Accelerometer-based', ...
       'Kalman␣filter', 'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
```

```
end

n = n + 1;

% Plot: Shank angle estimate - acceleration-based,
%       KF, and EKF.
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_shank(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_shank(n1:n2)-90);
plot(time(n1:n2), pitch_KF_right_shank(n1:n2)-90, ...
     time(n1:n2), pitch_EKF_right_thigh(n1:n2) ...
     + pitch_EKF_right_shank(n1:n2), 'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter', 'latex');
ylabel(['Pitch␣angle␣$\theta_1␣+␣\theta_2$␣in␣', ...
        '$^{\circ}$'], 'interpreter','latex');
legend('Reference', 'Accelerometer-based', ...
       'Kalman␣filter', 'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

n = n + 1;

% Compute root-mean-square error.
% -Thigh
RMSE_acc = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
                - pitch_acc_right_thigh(n1:n2)').^2));
RMSE_KF = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
               - pitch_KF_right_thigh(n1:n2)').^2));
RMSE_EKF = sqrt(mean((pitch_QS_right_thigh(n1:n2) ...
             - 90 - pitch_EKF_right_thigh(n1:n2)).^2));
RMSE = [RMSE_acc, RMSE_KF, RMSE_EKF];
% -Shank
RMSE_acc = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
                - pitch_acc_right_shank(n1:n2)').^2));
```

```
RMSE_KF = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
               - pitch_KF_right_shank(n1:n2)').^2));
RMSE_EKF = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
             - 90 - pitch_EKF_right_thigh(n1:n2) ...
             - pitch_EKF_right_shank(n1:n2)).^2));
RMSE = [RMSE; RMSE_acc, RMSE_KF, RMSE_EKF;];

% Compute sum of seperate RMSEs.
RMSE = [RMSE; RMSE(1, 1) + RMSE(2, 1), RMSE(1, 2) + ...
        RMSE(2, 2), RMSE(1, 3) + RMSE(2, 3)];

% Plot bar graph and values on top of the bars.
figure(n);
b = bar(RMSE, 0.3);
offset = 0.8;
yb = cat(1, b.YData);
xb = bsxfun(@plus, b(1).XData, [b.XOffset]');
hold on;

for i = 1:3
   for j = 1:3
        text(xb(j, i), yb(j, i) + offset, ...
            ['\scriptsize␣', num2str(RMSE(i, j), ...
            '$%0.2f$')], 'rotation', 0, ...
            'interpreter', 'latex', ...
            'HorizontalAlignment','center');
   end
end

b(1).FaceColor = [0.8500    0.3250    0.0980];
b(2).FaceColor = [0.9290    0.6940    0.1250];
b(3).FaceColor = [0.4940    0.1840    0.5560];

text(1.4, 50, ['\scriptsize␣$\frac{\operatorname', ...
        '{RMSE}_{EKF_{thigh}}␣+␣\operatorname', ...
        '{RMSE}_{EKF_{shank}}}{\operatorname', ...
        '{RMSE}_{KF_{thigh}}␣+␣\operatorname', ...
        '{RMSE}_{KF_{shank}}}␣=␣', ...
        num2str((RMSE(1, 3) + RMSE(2, 3)) / ...
        (RMSE(1, 2) + RMSE(2, 2)), '%0.2f$')], ...
        'interpreter','latex');
```

```

ylim([0, max(max(RMSE)) + 2]);
ylabel('Root-mean-square␣error␣in␣$^{\circ}$', ...
       'interpreter','latex');

labels = {'Thigh', 'Shank', 'Thigh␣+␣Shank'};
format_ticks(gca, labels, [], [], [], 0);

legend('Acceleration-based', 'Kalman␣filter', ...
          'Extended␣Kalman␣filter');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

n = n + 1;

% Plot: Acceleration-based pitch angle shank - corrected.
n1 = 4 * f + 1;
n2 = 20 * f;
figure(n);
hold on;
plot(time(n1:n2), pitch_QS_right_shank(n1:n2) - 90, ...
     'linewidth', 1);
plot(time(n1:n2), pitch_acc_right_shank(n1:n2) - 90);
plot(time(n1:n2), theta12_c(n1:n2), 'linewidth', 1);

xlabel('Time␣$t$␣in␣s', 'interpreter','latex');
ylabel(['Pitch␣angle␣$\theta_1␣+␣\theta_2$␣in␣', ...
        '$^{\circ}$'], 'interpreter', 'latex');
legend('Reference', 'Accelerometer-based', ...
       'Accelerometer␣based␣-␣corrected');

% Compute root-mean-square error.
RMSE_acc = sqrt(mean((pitch_QS_right_shank(n1:n2) ...
                - pitch_acc_right_shank(n1:n2)').^2));
RMSE_acc_corr = sqrt(mean((pitch_QS_right_shank(n1:n2)...
                - 90 - theta12_c(n1:n2)).^2));
RMSE_acc = [RMSE_acc, RMSE_acc_corr];
```

```

text(6,-30, ['\scriptsize␣$\operatorname{RMSE}', ...
             '_{Acc}␣=␣', num2str(RMSE_acc(1), ...
             '%0.2f$')], 'interpreter','latex');
text(6,-40, ['\scriptsize␣$\operatorname{RMSE}', ...
             '_{Acc-corrected}␣=␣', ...
             num2str(RMSE_acc(2), ...
             '%0.2f$')], 'interpreter','latex');
text(6,-50, ['\scriptsize␣$\frac{\operatorname', ...
             '{RMSE}_{Acc-corrected}}', ...
             '{\operatorname{RMSE}_{Acc}}␣=␣', ...
             num2str(RMSE_acc(2) / RMSE_acc(1), ...
             '%0.2f$')], 'interpreter','latex');

if tikz
matlab2tikz(['../tikz/experiment_', num2str(n), ...
             '.tikz'], 'height', '\figureheight', ...
             'width', '\figurewidth');
end

% Display the results.
fprintf('-----------------Results---------------\n');
fprintf(['AB:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 1));
fprintf(['KF:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 2));
fprintf(['EKF:\n␣RMSE_thigh:␣%0.4f\n␣RMSE_shank:␣', ...
         '%0.4f\n␣RMSE_sum:␣␣␣%0.4f\n\n'], RMSE(:, 3));
fprintf(['EKF␣parameters:␣\n␣-sigma_d_sq:␣␣', ...
         '%0.4f\n␣-sigma_t1_sq:␣', ...
         '%0.4f\n␣-sigma_t2_sq:␣', ...
         '%0.4f\n␣-sigma_b_sq:␣␣', ...
         '%0.4f\n␣-sigma_1_sq:␣␣', ...
         '%0.4f\n␣-sigma_2_sq:␣␣', ...
         '%0.4f\n␣-sigma_s3_sq:␣', ...
         '%0.4f\n␣-sigma_f3_sq:␣', ...
         '%0.4f\n␣-sigma_s4_sq:␣', ...
         '%0.4f\n␣-sigma_f4_sq:␣', ...
         '%0.4f\n'], par);
fprintf('----------------------------------------\n');
```