Department of Mathematics and Computer Science

University of Heidelberg

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Master thesis

in Applied Computer Science

submitted by

Stephanie Kunkel

born in Naumburg/Saale

April 2016

Study on IMU-based Indoor Positioning

With UWB-wireless Recalibration

This Master thesis has been carried out by Stephanie Kunkel

at the

Institute of Computer Engineering

under the supervision of

Prof. Dr. Peter Fischer and

Dr. Andreas Kugel

Studie zur inertialen Lokalisierung in geschlossenen Räumen mit UWB-basierter drahtloser Rekalibration:

Die Lokalisierung von Objekten in geschloßenen Räumen ist ein viel erforschtes Gebiet. Da GPS in solchen Umgebungen nicht genutzt werden kann, kommen andere Technologien zum Einsatz. Eine davon sind *Inertialsensoren*, welche die Winkelgeschwindigkeit und Beschleunigung von Objekten bestimmen. Da sie einer sogenannten *Drift* unterliegen, sollten sie mittels absoluter Lokalisierung, beispielsweise mit Hilfe von *Ultrabreitband (UWB)*, rekalibiert werden.
In der vorliegenden Masterarbeit ermöglichen beide Technologien die Positionierung eines *Roboterarms*. Das System, welches um einen Zynq-7000 SoC errichtet wurde, ermöglicht die Kommunikation mit den Sensoren, sowie die Steuerung des Roboterarms. Die Ausrichtung der Inertialeinheit kann mit Hilfe des entwickelten *IMU Viewers* in Echtzeit am PC nachverfolgt werden. Alternativ werden die Sensordaten am Ende der Datensammlung zum PC übertragen.
Die *Kalibration* der Inertialeinheit ist statisch oder dynamisch möglich. Zur *Glättung* der Rohdaten werden der gleitende Durchschnitt, FIR Filter und 1D Kalman Filter benutzt. Ein *Komplementärfilter* mit einstellbarem *Beschleunigungsbereich* wurde für die Fusion von Gyroskop und Beschleunigungssensor implementiert. Zum gleichen Zweck wird auch ein *Kalman Filter* getestet. Beide Filter werden zudem für die Fusion von den Intertialsensoren und UWB verwendet.
Alle Methoden werden in zwei Experimenten mit und ohne *Bewegung* des Roboterarms getestet. Nach der Auswertung der gesammelten Daten, werden entsprechende Empfehlungen gegeben.

Study on IMU-based Indoor Positioning With UWB-wireless Recalibration:

Indoor Positioning is a widely researched field. As GPS is not applicable indoors, other technologies must be used. One of them are *Inertial Measurement Units (IMUs)*, which yield the object's angular velocity and acceleration. However, IMUs suffer from drift. Therefore, they may be re-calibrated using absolute positioning techniques, e.g. *Ultra-Wideband (UWB)*.
In this work, both technologies enable the indoor localization of a *robot arm*. The system, built around a Zynq-7000 SoC, facilitates the communication with the different sensors and the control of the robot arm. The IMU's orientation can be viewed in real-time at the PC using the developed *IMU Viewer*. Furthermore, the sensor data may be transferred to the PC, as soon as the data collection has finished.
To *calibrate* the IMU, there are two options available: static and dynamic. Different *smoothing* methods, like the moving average, FIR filter and a 1D-Kalman filter, are applied. Furthermore, a *complementary filter* with an *accelerometer range* was implemented to facilitate the *sensor fusion* of gyroscope and accelerometer in order to reduce orientation drift. A *Kalman filter* is also tested for this purpose. Both kinds of filters are also facilitated to fuse the position obtained from IMU and UWB technology.
All methods are tested in two experiments, one *without motion* and one with 30° *stepwise movements*. The obtained data is evaluated and consilient recommendations are given.

Acknowledgements

It is a pleasure to thank those, who made this thesis possible:

First of all, I would like to acknowledge Dr. Andreas Kugel, for the idea of this thesis and his support in all different areas of my work. Without him, I would not have had the chance to work on this topic, which was interesting and challenging at the same time. I always appreciated his valuable suggestions and ideas, while letting me work autonomously.

Moreover, I would like to show my gratitude to Prof. Dr. Peter Fischer, for supervising my work and taking the time to discuss it in more detail.

Furthermore, I would like to thank my husband, Ingo Kunkel, for his patience, beneficial ideas and his proof-reading, which challenged my work and made it possible to accomplish the comprehensibility of this final version.

Lastly, I would like to acknowledge all the individuals who are coding for all the open-source projects for free. Without LaTeX, this thesis would not have such a professional typeset, supported by all the meaningful figures and graphs, which I created with Octave, Dia and GIMP.

Declaration

I herewith declare that I have produced this thesis without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other German or foreign examination board.

The thesis work was conducted from November 1, 2015 to April 30, 2016 under the supervision of Prof. Dr. Peter Fischer and Dr. Andreas Kugel at the University of Heidelberg.

Mannheim, April 21, 2016 .....

Abbreviations

ACK Acknowledge

ADC Analog-to-Digital Converter

ASCII American Standard Code for Information Interchange

BSP Board Support Package

CFT Continuous Fourier Transform

CRC Cyclic Redundancy Check

DFT Discrete Fourier Transform

DLPF Digital Low Pass Filter

DMP Digital Motion Processor

DSP Digital Signal Processing

ETSI European Telecommunications Standards Institute

FCC Federal Communication Commission

FFT Finite Fourier Transformation

FIFO First-In-First-Out

FIR Finite Impulse Response

FSR Full-Scale Range

GPIO General-Purpose Input/Output

GPS Global Positioning System

IIR Infinite Impulse Response

IMU Inertial Measurement Unit

INS Inertial Navigation System

I/O Input/Output

IP Intellectual Property

IPS Indoor Positioning System

LSB Least Significant Bit

MEMS Microelectromechanical System

MIO Multiplexed Input/Output

NACK Non-Acknowledge

OSG Open Scene Graph

PHY Physical Layer

PLL Phase-Locked Loop

PL Programmable Logic

PMOD Peripheral Module

PSD Power Spectral Density

PS Processing System

PWM Pulse Width Modulation

RFID Radio-Frequency Identification

RF Radio-Frequency

RMSE Root Mean Square Error

RSE Root Square Error

SAW Surface Acoustic Wave

SCL Serial Clock

SDA Serial Data

SoC System on a Chip

SPI Serial Peripheral Interface

TDOA Time Difference of Arrival

TOF Time of Flight

UART Universal Asynchronous Receiver/Transmitter

UWB Ultra-Wideband

VHDL Very High Speed Integrated Circuits (VHSIC) Hardware Description Language

WLAN Wireless Local Area Network

XSDK Xilinx Software Development Kit

ZLO Zero-Level Offset

Contents

List of Figures

List of Tables

1 Introduction and Overview

1.1 Problem Area: Indoor Localization

The desire to know the exact position of people and objects has been present for a while. More than twenty years ago, in 1993, the U.S. Air Force launched the 24^{th} Navstar satellite into the orbit, which completed the satellite network called Global Positioning System (GPS). Using GPS, the location of a receiver could be determined within a few hundred feet.

However, due to walls and roofs blocking the already weak signals coming from the satellites, GPS does not work well within buildings where its accuracy drops significantly. Therefore, indoors, a different approach is required.

Similar to GPS, an *Indoor Positioning System (IPS)* determines the location of people or objects within buildings. It relies on beacons mounted on the walls or ceiling, on sensors attached on the object itself, or a combination of both. Additionally, IPS may be able to determine the direction in which the target is traveling.

Dempsey [1] defines an IPS as a system that "continuously and in real-time can determine the position of something or someone from a distance within a physical space". *Continous* operation means that the system works at all times, if it is not turned off on purpose (e.g. for privacy reasons of individuals carrying cell phones). *Real-time* stands for the frequent update of the target's position. *Distance* implies the ability to locate and track the target within a physical space without the premise that the target passes a portal (e.g. an entrance to a building or room).

If the target object or person carries a mobile device, the position and direction information can be determined by the device using the data of the IPS (*self-positioning*) or the device sends information to the IPS which may or may not make the location information available for location-based services on the mobile device (*infrastructure positioning* or *self-oriented, infrastructure-assisted*).

This can be applied to different *areas*. In large public indoor areas, IPS may be used for position indication, e.g. for tourists in museums or big shopping malls. Another application is the location of lost objects. For example, a lost cell phone may be located using a Laptop. Furthermore, a vehicle could unlock itself when the owner comes within range. There are several applications in health-care as well. In hospitals, IPS may prevent stolen equipment or guide patients. The same idea may be used for an alarm system which recognizes infants or Alzheimer patients trying to

leave an area that they are supposed to stay within. In general, the idea of tracking objects is interesting for all kinds of companies working with assets, inventory, tools, pallets and components, as well as animals — in health-care, manufacturing and assembly, warehousing and logistics, as well as farming.

In general, indoor settings are more *complex* than outdoor environments. As stated above, obstacles such as walls, equipment, or human beings influence the propagation of electromagnetic waves, which lead to so-called multi-path and environmental effects. *Multi-path* propagation stands for the delay of part of the signal that has taken a different, longer path because it previously bounced off obstacles. In addition, there are interference and multi-path propagation from wired or wireless networks which reduce positioning accuracy.

There are different kinds of location information. Using *absolute* position information, the target can be located on a map of the surrounding area while *relative* position information show the motion of the object relative to a reference object. For example, the angle between the body and the arm of a person may be determined, or whether a door is closed or not (relative to the wall). *Proximity* gives a rough estimate of the target's position, e.g. in which (part of the) room the target is.

The main problem in indoor localization is *accuracy* which does not only depend on the technology used, but also on the application. Is it required to know the exact position of the target or is it enough to know in which room it is? Is it necessary to know an absolute location or is a relative position good enough? Is a 3D location required or is the position on a 2D map enough?

Other issues are the *range* of the system (Is it enough to just cover one room or is the position within a large building required?) and its *robustness*, i.e. how much the performance is influenced by the system's environment like weather, light, noise, magnets, obstacles, etc. [1, 2, 3, 4, 5, 6, 7, 8, 9]

1.2 Goal and Tasks of this Thesis

The *goal* is to set up an IPS by combining an Inertial Measurement Unit (IMU) and Ultra-Wideband (UWB) technology to be able to control a robot arm to pick up objects. In the future, it is desired to place the robot onto a chassis and let it move around freely.

Therefore, the following *equipment* is available:

- one *MPU-9150* IMU
- four *DWM1000* UWB modules
- one robot arm

- one ZedBoard Zynq-7000 ARM/FPGA SoC Development Board

The idea is to use the data of the UWB modules to re-calibrate IMU measurements. Therefore, no additional information, e.g. the commands to the robot arm, are supposed to be used.

The overall project is separated into the following *tasks*:

1. Set up the *hardware* and develop the corresponding *software*, including
   - the communication with the IMU, UWB, robot arm and PC
   - the integration and use of the freely-available *Motion Driver* for the IMU
   - the integration and use of the already existing driver and application software for the UWB modules
   - the readout of sensor data
   - the integration and use of the already existing hardware module for controlling the robot's servo motors
   - controlling the robot
   - the communication with the PC to visualize the orientation and position of the IMU in real-time or analyze measured data later on
2. *Experiments* with the system regarding to the achievable accuracy by
   - calibration
   - enhancement of the raw data by smoothing and sensor fusion

1.3 Chapter Overview

In the next chapter, the used technologies for indoor localization, as well as the required methods for rotation and position estimation, are described. Chapter 3 explains the chosen hardware and the final system setup. Chapter 4 contains a detailed description of the software, developed in the course of this thesis. The calibration of the different sensors and actors is described in chapter 5. In chapter 6, the performed experiments and their results are explained. It follows an evaluation of the achieved results in chapter 7. The thesis ends with a summary and conclusion in chapter 8.

2 Indoor Positioning

There are different approaches to locate an object or person. If the starting position is known, one approach is to simply track movements using inertial sensors to determine the new position. If the initial position is unknown, there are other technologies that are capable of absolute positioning, including Infrared, Radio-Frequency Identification (RFID), Wireless Local Area Networks (WLANs), Bluetooth, Ultra-Wideband (UWB), magnetic signals, vision analysis, audible sounds and ultrasound.

All technologies have different advantages and disadvantages. The technology of choice is always a trade-off between accuracy and range, performance and power consumption, as well as complexity and cost. There are some very detailed surveys on existing indoor positioning technologies available, including Gu et al. [5] and Harle [10].

As different sensor types have different strengths and weaknesses, the combination of them can compensate for their weaknesses. For example, inertial sensors do not depend on a pre-installed infrastructure and line-of-sight transmission. However, they are only stable in the short term. Therefore, a combination with a long-term stable technology is desired to improve overall precision. This is called *sensor fusion* and is further explained in section 2.3.

For example, the short-term accurate inertial sensors can be combined with the long-term stable UWB technology as stated by Zwirello et al. [11], Herrera et al. [12], De Angelis et al. [13] and Ascher et al. [14]. This will be the approach for this thesis.

In the following sections, Inertial Sensors and Ultra-Wideband (UWB) will be introduced in sections 2.1 and 2.2, as well as the process of position estimation using each of these technologies. There is also a section about the process of improving results through smoothing and filtering in section 2.3. [5, 9, 11]

2.1 Inertial Sensors

Inertial-based positioning systems are not able to determine the absolute location of an object. However, they use acceleration and rotational velocity measurements

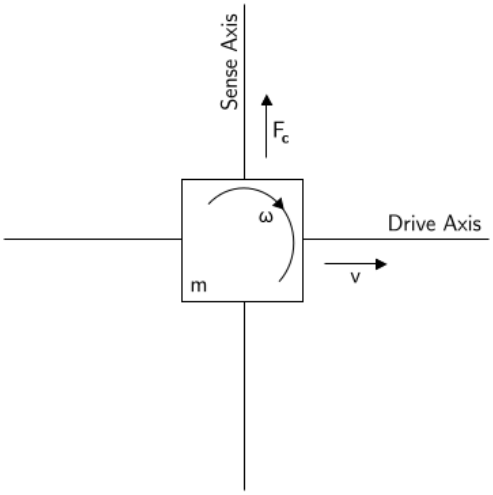


Figure 2.1: A Vibrating Mass Gyroscope [16]

from accelerometers and gyroscopes to determine *motion* and *rotation*. As mentioned before, this method is not stable in the long term due to drift problems, resulting from high noise and different uncertainties like biases and scale factors.

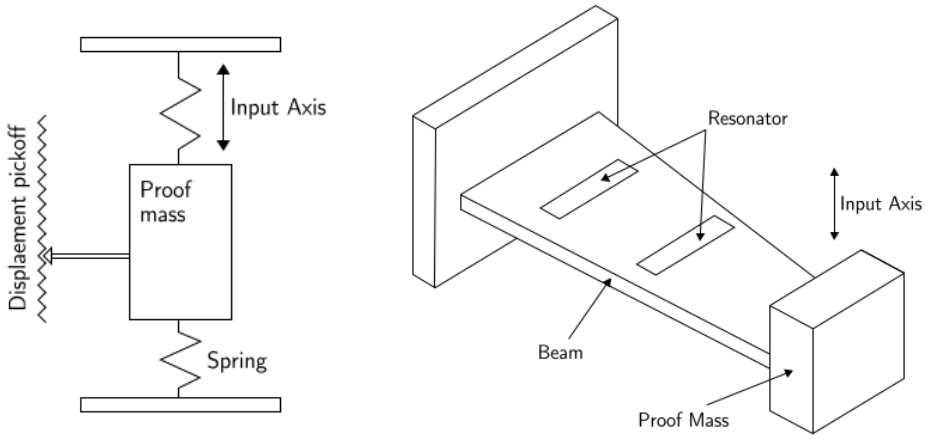
Therefore, gyroscopes and accelerometer are combined with magnetic sensors. This constellation is called *Inertial Measurement Unit (IMU)* or Inertial Navigation System (INS). Usually, there are three orthogonal-rate gyroscopes, three orthogonal accelerometers and one magnetometer in one IMU. [15, 16]

Gyroscopes

IMUs contain Microelectromechanical System (MEMS) gyroscopes, which use the *Coriolis effect* to determine angular velocity. This effect states that a mass m, moving with a velocity v, in a frame of reference, rotating with an angular velocity ω , experiences a force F_c such that

$$F_c = -2m/\omega \times v \tag{2.1}$$

To measure the Coriolis force, MEMS gyroscopes contain vibrating elements, such as vibrating wheels or tuning forks. The simplest geometry is a single mass driven to vibrate along a drive axis, as shown in figure 2.1. Any rotation of the device causes a secondary vibration along the perpendicular sense axis due to the Coriolis force. This secondary rotation is measured to determine angular velocity. [16]



(a) Mechanical (b) Surface Acoustic Wave (SAW)

Figure 2.2: Types of Accelerometers [16]

Accelerometers

Just like gyroscopes, an IMU contains MEMS accelerometers. There are two different kinds of MEMS accelerometers, which are illustrated in figure 2.2. One kind works similar to mechanical accelerometers, which measure the displacement of a mass suspended by springs, as shown in figure 2.2a. The other kind, the SAW accelerometer (figure 2.2b), consists of a cantilever beam resonating at a particular frequency. At that end of the beam, which is free to move, a mass is attached. The other end of the beam is attached to the case. When there is an acceleration applied to the input axis, the beam bends, which causes the frequency of the SAW to change proportionally to the applied strain. The amount of acceleration is determined by measuring the change in frequency. [16]

Frames of Reference

There are different *frames of reference* when talking about indoor positioning. The frame of reference of the local system, e.g. the IMU, is also called *body frame* and the frame of reference, in which it is located, is known as *global frame*. The IMUs position is anywhere within the global frame and its rotation may be different from the global frame, as illustrated in figure 2.3. The same applies to the different sensors within the IMU, which all have a local reference frame within the global reference frame of the IMU.

Woodman [16] states two categories of IMUs that differ in the reference frame of their gyroscopes and accelerometers. In the first category, *stable platform systems*, the IMU is mounted on a platform isolated from any external rotation. Thus, the

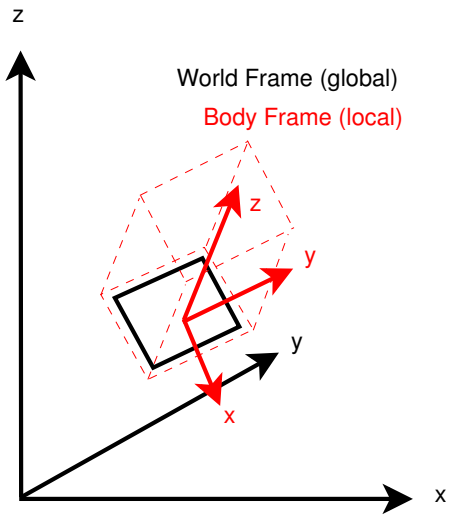


Figure 2.3: Different Frames of Reference

platform is aligned with the global frame. This is done by mounting the platform with gimbals (frames), giving the platform freedom to rotate in all three axes. If the gyroscopes detect any platform rotation, these signals are sent to the torque motors which rotate the gimbals in to opposite direction, in order to cancel out any rotation and keep the platform aligned with the global frame.

The other category is called *strapdown system*. In this kind of system, the inertial sensors are fixed on the device and measure the rotations of the body frame instead of the global frame. As the underlying principles are the same, but a strapdown system is mechanically less complex and smaller, it will be used subsequently. [16]

Applications and Related Work

Applications of IMUs are, among others, navigation of aircraft, tactical and strategic missiles, spacecraft, submarines and ships as well as human and animal motion capture [16].

Different indoor positioning projects have been done with IMUs. Maeda and Ando [17] designed an IMU-based system measuring the user's full body motion. Miller et al. [18] used inertial sensors to control the robot arm of a NASA Robonaut, wireless and in real-time. A real-time solution for 3D arm motion tracking for home-based rehabilitation using a combination of inertial and visual sensors is reported by Tao et al. [19]. Fox [20] presents two inertial-sensor-based hybrid tracking technologies, which take advantage of a compass and an ultrasonic system, respectively. A method for reliably comparing IMUs with a Vicon reference system was developed by Sessa et al. [21]. Ojeda and Borenstein [22] elaborated on a navigation system for security staff and first-responders that uses an IMU instead of GPS. Hartmann

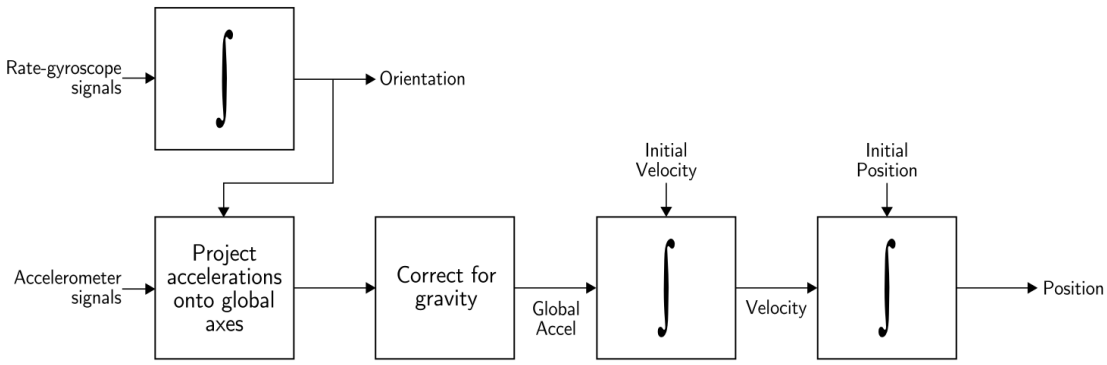


Figure 2.4: Inertial Navigation Algorithm for *Strapdown Systems* [16]

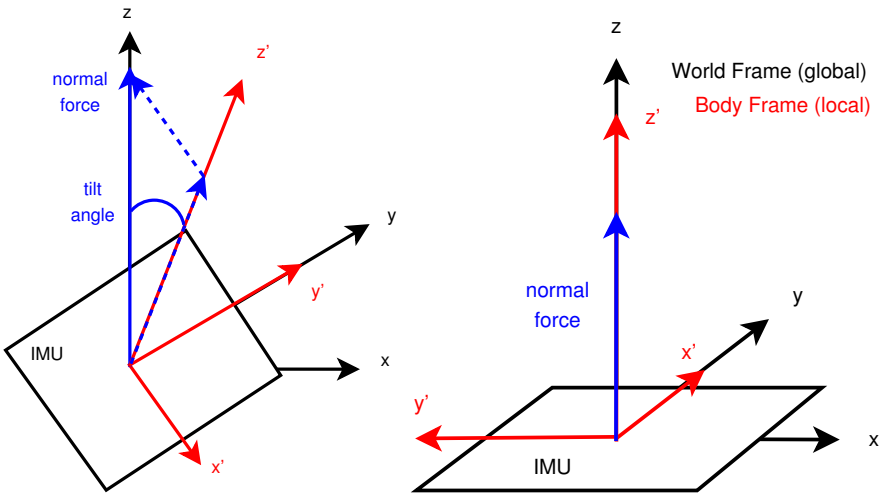
et al. [23] combined an IMU with a marker-based video tracking system with external cameras. A Ph.D. thesis by Luinge [24] goes into detail about inertial sensing of human movement. Recently, Lohse [25] designed a *Sound Glove* using IMUs, which can be used to control a Digital Audio Station.

2.1.1 Rotation Determination

General Approach

The general approach to determine the rotation of a *strapdown system* is shown in figure 2.4. In order to pursue the orientation, the signals coming from the rate gyroscopes have to be integrated over time. For position tracking, the accelerometer measurements have to be projected from local to global coordinates using the orientation known from the integration of the gyroscope signals. Then, the global acceleration signals are corrected for gravity and integrated twice. The first integration yields velocity, the second one position, using the previously computed velocity and position, respectively.

It is also possible to obtain the tilt angle of the IMU, using accelerometer measurements only. The idea is to compute the angle between the global z-axis and the IMU's z-axis, as shown in figure 2.5a. Both axes are equal, if the IMU is perfectly even (figure 2.5b), resulting in a tilt angle of 0°. The normal force, which is opposed to gravity and prevents the IMU from falling down, is used as reference. Note, that this does not work while the IMU is accelerated, because the difference between acceleration and normal force cannot be determined using the accelerometers alone. [16, 26, 27, 28]



(a) Pitted IMU (b) Rotated, but Even IMU

Figure 2.5: Tilt Angle Estimation using Accelerometers

Representation

There are different representations of *orientation*, also called attitude, of an INS relative to the global reference frame. They include, but are not limited to Euler angles, axis-angle representation and quaternions. *Euler angles* describe orientation as rotations about the different axes, called roll (Φ), pitch (Θ) and yaw (Ψ). This is illustrated in figure 2.6. They are intuitive and simple to use. However, they are limited by a phenomenon called *gimbal lock*, where the orientation cannot be determined when the pitch angle approaches $\pm 90°$.

Quaternions do not suffer from this phenomenon. Yet, they are less intuitive and may be harder to use. Quaternions can be computed from the axis-angle representation, which is shown in figure 2.7. [16, 29, 30]

Quaternions A *quaternion* is a four-element vector, containing one real and three complex elements, and it is an extension of the idea of complex numbers. It can be written as

$$q = (w, x, y, z) = w + xi + yj + zk \tag{2.2}$$

where w is the real part and x, y, and z are imaginary. Therefore,

$$i^2 = j^2 = k^2 = -1 \tag{2.3}$$

In other words, w is the amount of rotation around an axis defined by x, y, and z. The idea of rotation around an arbitrary axis is usually described by the axis-angle representation, displayed in figure 2.7. [30, 31, 32]

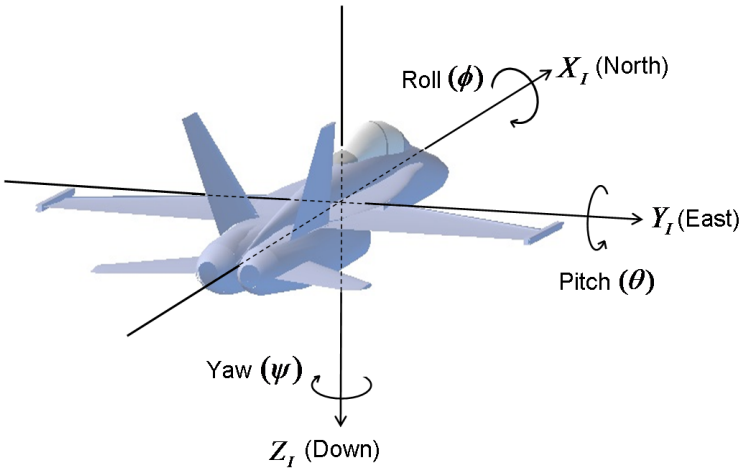


Figure 2.6: Describing Rotation Using Euler Angles [29]

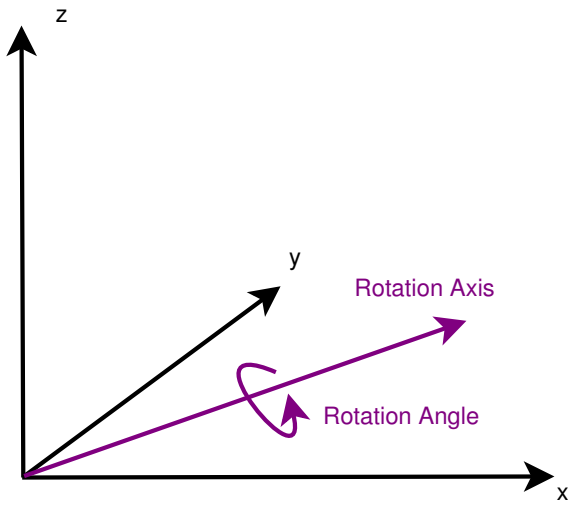


Figure 2.7: Axis-Angle Representation in a Right-handed Coordinate System

Rotation Estimation

Using Gyroscope Data There are a few steps to manipulate rotation described by a quaternion. At first, a temporary quaternion $q_{\Delta t}$, describing the change of rotation (*relative rotation*), has to be constructed by integrating the angular rates of the gyroscope over time. To project the relative rotation to the *absolute rotation* q_{t-1}, the temporary and the permanent quaternion, the latter describing the current (absolute) rotation, have to be multiplied. This leads to a new permanent quaternion q_t describing the new absolute rotation. This process basically determines a rotation in the objects reference frame and projects it to the global reference frame.

The relative attitude described by a quaternion is determined as follows. If the read-out gyroscope data ω is defined as $(\omega_x, \omega_y, \omega_z)$, the rotational vector magnitude $|\omega|$ is computed by

$$|\omega| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \tag{2.4}$$

This is used to compute the unit vector U_ω based on the body rotation vector, which is a normalized representation of the *rotation axis*, which is shown in figure 2.7:

$$U_\omega = \left(\frac{\omega_x}{|\omega|}, \frac{\omega_y}{|\omega|}, \frac{\omega_z}{|\omega|} \right) \tag{2.5}$$

If Δt is the time difference between the current and the last sample, the *rotation angle* Θ is computed by integration over time:

$$\Theta = |\omega| \Delta t \tag{2.6}$$

Using the normalized rotation vector, the quaternion values w, x, y and z are determined using the formulas

$$w_{\Delta t} = \cos\left(\frac{\Theta}{2}\right) \tag{2.7}$$

$$x_{\Delta t} = U_{\omega,x} \sin\left(\frac{\Theta}{2}\right) \tag{2.8}$$

$$y_{\Delta t} = U_{\omega,y} \sin\left(\frac{\Theta}{2}\right) \tag{2.9}$$

$$z_{\Delta t} = U_{\omega,z} \sin\left(\frac{\Theta}{2}\right) \tag{2.10}$$

where $U_{\omega,x}$, $U_{\omega,y}$ and $U_{\omega,z}$ are the x, y and z-parts of U_ω. The resulting quaternion describes the *relative* rotation that has happened between time $t-1$ and t, i.e. the change in absolute rotation of an object during Δt.

Then, the relative rotation quaternion $q_{\Delta t}$ is multiplied by the previous absolute rotation quaternion q_{t-1} resulting in the new *total rotation* quaternion $q_t = q_{\Delta t} q_{t-1}$, where

$$w_t = w_{\Delta t} w_{t-1} - x_{\Delta t} x_{t-1} - y_{\Delta t} y_{t-1} - z_{\Delta t} z_{t-1} \tag{2.11}$$

$$x_t = w_{\Delta t} x_{t-1} + x_{\Delta t} w_{t-1} + y_{\Delta t} z_{t-1} - z_{\Delta t} y_{t-1} \tag{2.12}$$

$$y_t = w_{\Delta t} y_{t-1} - x_{\Delta t} z_{t-1} + y_{\Delta t} w_{t-1} + z_{\Delta t} x_{t-1} \tag{2.13}$$

$$z_t = w_{\Delta t} z_{t-1} + x_{\Delta t} y_{t-1} - y_{\Delta t} x_{t-1} + z_{\Delta t} w_{t-1} \tag{2.14}$$

Now, if the rotation is to be used in form of a *rotation matrix* in a right-handed coordinate system, which is shown in figure 2.7 (p. 11), this is how the matrix R is computed:

$$R = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2xy + 2wz & 2xz + 2wy & 0 \\ 2xy + 2wz & w^2 - x^2 + y^2 - z^2 & 2yz + 2wx & 0 \\ 2xz + 2wy & 2yz - 2wx & w^2 - x^2 - y^2 + z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

Note that in a left-handed coordinate system, the signs of all expressions except the diagonals have to be inverted. [30, 33, 34]

Using Accelerometer Data As stated above, the absolute rotation (i.e. tilt angle) can also be determined using the accelerometers. Therefore, the rotation axis and angle have to be determined. Tilting the IMU corresponds to the rotation around an arbitrary axis. For example, if it is tilted towards the x-axis, there is a rotation around the y-axis. Therefore, the *tilt*, or *rotation*, *angle* is the angle α between the global z-axis $\vec{z}$ and the z-axis of the IMU $\vec{z'}$, which is computed using their scalar (dot) product and magnitudes:

$$\alpha = \cos^{-1}\left(\frac{\vec{z} \cdot \vec{z'}}{|\vec{z}||\vec{z'}|}\right) = \cos^{-1}\left(\frac{z_x z'_x + z_y z'_y + z_z z'_z}{\sqrt{z_x^2 + z_y^2 + z_z^2}\sqrt{z_x'^2 + z_y'^2 + z_z'^2}}\right) \tag{2.16}$$

The rotation axis has to be orthogonal to $\vec{z}$ and $\vec{z'}$, just like the normal $\vec{n}$ of the plane spanned by both vectors. Thus, it is computed as the vector (cross) product of both vectors:

$$\vec{n} = \vec{z} \times \vec{z'} = \begin{pmatrix} z_y z'_z - z_z z'_y \\ z_z z'_x - z_x z'_z \\ z_x z'_y - z_y z'_x \end{pmatrix} \tag{2.17}$$

For example, in figure 2.5a (p. 10), the IMU is tilted towards the x-axis. In this particular case, the rotation axis is equal to the y-axis.

Finally, the computed axis and angle are converted to the *absolute* quaternion q_t similarly to the relative quaternion computed from gyroscope measurements above:

$$w_t = \cos\left(\frac{\alpha}{2}\right) \tag{2.18}$$

$$x_t = n_x \sin\left(\frac{\alpha}{2}\right) \tag{2.19}$$

$$y_t = n_y \sin\left(\frac{\alpha}{2}\right) \tag{2.20}$$

$$z_t = n_z \sin\left(\frac{\alpha}{2}\right) \tag{2.21}$$

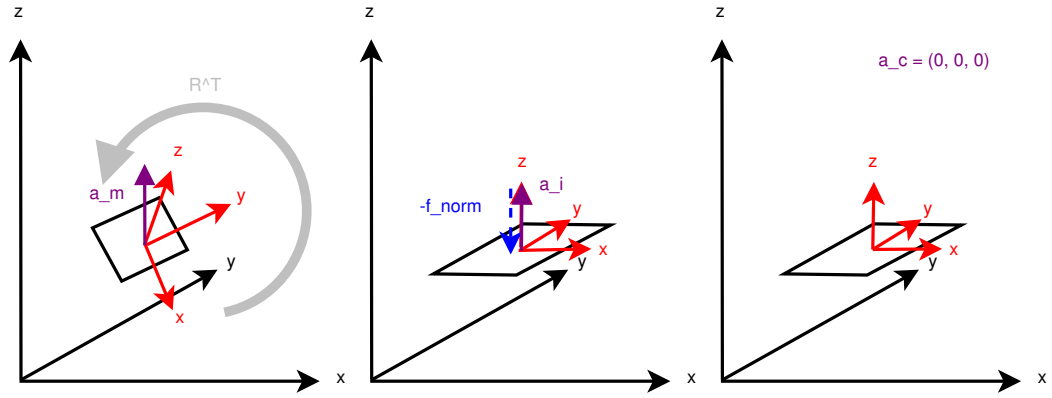
2.1.2 Position Estimation using Inertial Sensors

As shown in figure 2.4 on page 9, the acceleration has to be adjusted before position can be computed. This process is illustrated in figure 2.8, where it is assumed that there is no physical acceleration present. The first step is to project the measured accelerations into the global reference frame (figure 2.8a). If R^T is the transpose of the previously obtained rotation matrix R, and $\vec{a}_m$ is the vector of the measured accelerations, the accelerometer data projected into the global frame ($\vec{a_i}$), is computed as

$$\vec{a}_i = R^T \vec{a}_m \tag{2.22}$$

The next step is to remove gravity (figure 2.8b), more particularly normal force, from the measurements to retrieve the corrected accelerations a_c, also in global reference frame (figure 2.8c). This is computed by

$$\vec{a}_c = \vec{a}_i - \vec{f_n} \tag{2.23}$$



(a) Measured (b) Projected (c) Corrected

Figure 2.8: Preparing Acceleration for Position Computation

where $\vec{f}_n$ is the normal force in global frame. In an ideal system, the magnitude of the normal force vector is $9.81\frac{m}{s^2}$.

Now, the current velocity at time t can be computed by integrating the current acceleration over time:

$$\vec{v}_t = \int \vec{a}_{c,t} \tag{2.24}$$

As in practice, data is measured in discrete time intervals, the current velocity $\vec{v}_t$ is approximated by

$$\vec{v}_t = \vec{v}_{t-1} + \vec{a}_{c,t}\Delta t \tag{2.25}$$

where Δt is the time difference between the current and previous measurements. In addition, $\vec{v}_{t-1}$ is the previous velocity and $\vec{a}_{c,t}$ is the vector of the normal-force-corrected accelerations in global frame from (2.23) at time t.

The final step is to compute the current position $\vec{p}_t$, which is done similarly to calculating the current velocity:

$$\vec{p}_t = \int \vec{v}_t = \iint \vec{a}_{c,t} \tag{2.26}$$

Using the previous position and velocity, the current position p_t is approximated by:

$$\vec{p}_t = \vec{p}_{t-1} + \vec{v}_t\Delta t \tag{2.27}$$

[16, 35]

2.1.3 Errors and Expected Accuracy

Analog-to-Digital Converter

As mentioned before, for the gyroscopes, the vibration caused by the Coriolis force is detected by a capacitive pickoff. For the accelerometers, the displacements of the masses for each axis are differentially measured by capacitive sensors. These analog signals are converted to digital signals by Analog-to-Digital Converters (ADCs), which is one possible error source.

ADCs yield a value within a predefined range. For example, a 10-bit ADC outputs values within a range of 0 to $2^{10} = 1,023$, while 16-bit ADCs output a value between 0 and $2^{16} = 65,536$. The output depends on the ADC's reference voltage. Therefore, continuous voltage values are converted into discrete integer numbers. starlino [36] gives an example of the conversion from analog to digital values for accelerometer data. [36, 37]

Gyroscope Drift

As mentioned in section 2.1, there is a drift over time, especially in MEMS devices. This has several causes. At first, there is a constant *bias* when the gyroscope is not moving, which is the offset from the true value. If integrated, it results in an angular error growing linearly with time.

Furthermore, there is thermo-mechanical *white noise*. It fluctuates at a higher rate than the sampling rate of the sensor. White noise is a sequence of uncorrelated random variables with zero-mean. Gyroscope manufacturers publish the effects of white noise for their sensors as *angle random walk* with the unit $^\circ/\sqrt{h}$, as *Power Spectral Density (PSD)* $((^\circ/h)^2/Hz)$ or as *Finite Fourier Transformation (FFT) noise density* $(^\circ/h/\sqrt{Hz})$. It grows with the square root of time.

Another cause of drift is *flicker noise*, which is noise with a $1/f$ spectrum, often observed at low frequencies in electronic devices. Bias fluctuations caused by flicker noise are usually modelled as random walk, which is computed by dividing bias stability (in $^\circ/h$) by the square root of time. In reality, bias fluctuations are not random, therefore the model of random walk is not a good approximation in the long term. If integrated, this kind of noise results in second-order random walk.

In addition, *temperature* changes in the environment or caused by the sensor heating up lead to bias changes. These effects are not included in bias stability measurements, which are taken under fixed conditions. The relationship between temperature and bias is often non-linear. When integrated, this kind of error grows linearly with time.

Finally, there are *calibration errors* containing errors in scale factors, alignments and linearities of the gyroscopes. These errors can usually be noticed while the device is rotating and result in additional drift in the integrated signal, which is proportional to the rate and duration of motion.

According to Woodman [16], the most important error sources are angle random walk, uncorrected bias errors due to temperature fluctuations and errors in the initial bias estimation. [15, 16]

Accelerometer Measurement Errors

In accelerometers, there are different error sources, which are equivalent to those of gyroscopes. The main difference is the double integration instead of a single integration, which increases the error even more. At first, there is a constant *bias*, which is the difference between the real and the measured signal in g, or converted to m/s^2. When integrated twice to obtain position, it causes a position error growing quadratically with time.

Furthermore, there is signal perturbation by *white noise*, which, when integrated, results in a random walk whose standard deviation grows proportionally to $\sqrt{t}$. Consequently, there is a velocity random walk, specified in $m/s/\sqrt{h}$, which in turn produces a second order random walk in position.

Flicker noise results in a changing bias over time. It is modelled as bias random walk, just like in gyroscopes. The result is a second-order random walk in velocity and a third-order random walk in position.

The effect of *temperature* fluctuations causes bias changes. The relationship between both depends on the device and is often non-linear. It causes an error in position increasing quadratically over time.

Calibration errors show up as bias errors and can be observed while the accelerometer is in motion. However, due to gravitational acceleration, this bias may even be observed while the device is stationary. [16]

Systematic Errors due to the Algorithm

Similar to rotation estimation, there is an increasing error in position estimation which is caused by the algorithm used. The measured acceleration vector is rotated opposite to the rotation measured by the gyroscopes and/or accelerometers. Therefore, the adjusted accelerations point in a slightly different direction.

In addition, the normal force is removed from the rotated acceleration vector. If there has already been an error in rotation, the acceleration vector is not rotated correctly. As a result, normal force is removed in a slightly incorrect angle, which

Error in				
Angle(°)	**Acceleration** ($\frac{m}{s^2}$)	**Velocity** ($\frac{m}{s}$) after $10s$	**Position** (m) after $10s$	**Position** (m) after $1min$
0.1	0.0017	0.17	1.7	61.2
0.5	0.086	0.86	8.6	309.6
1.0	0.017	1.7	17	612
2.0	0.342	3.42	34.2	1231.2
3.0	0.513	5.13	51.3	1846.8
5.0	0.854	8.54	85.4	3074.4

Table 2.1: Velocity and Position Errors Caused by Rotation Estimation Error [35]

leads to some parts of it being considered as physical acceleration. This increases the error even more. The expected acceleration, velocity and position errors caused by an incorrect rotation, assuming ideal accelerometer measurements, are shown in table 2.1 (p. 18). For example, if the rotation error is only 0.1°, the estimated position after one minute is already more than $60m$ off. Note that, if the orientation estimation were perfect, the only error source would be the accelerometer inaccuracy itself.

If there is noise in accelerometer measurements, an erroneous acceleration measured at time k may not be cancelled out by an opposite acceleration at time $k + 1$, which causes the velocity in this direction to stay the same. Consequently, the IMU keeps "moving" in this direction, even though it was just noise and no physical acceleration.

There has already been an error in rotation estimation due to gyroscope drift and single integration. The position error is even greater because of the double integration. It increases over time, as the previously calculated velocity and position are input for the computation of the current velocity and position. [15, 16, 35]

Expected Accuracy

The expected accuracy in literature ranges from 0.05° to 4° for rotation and $0.4cm$ to $150m$ for position. It strongly depends on the methods used to improve the results and the time between starting and finishing the measurement. Regarding the orientation, Sessa et al. [21] state an error of $0.9 - 2.8°$. Kowalczuk and Merta [38] achieved an angle precision of 0.05° using accelerometer measurements only. An inclination error of 2° was determined by Luinge [24], while Lohse [25] reports an orientation error of 4°. Axelsson and Norrlöf [39] report a rotation error of $1 - 2°$. Regarding the position, they state an average position error of $0.4 - 0.9cm$. An error

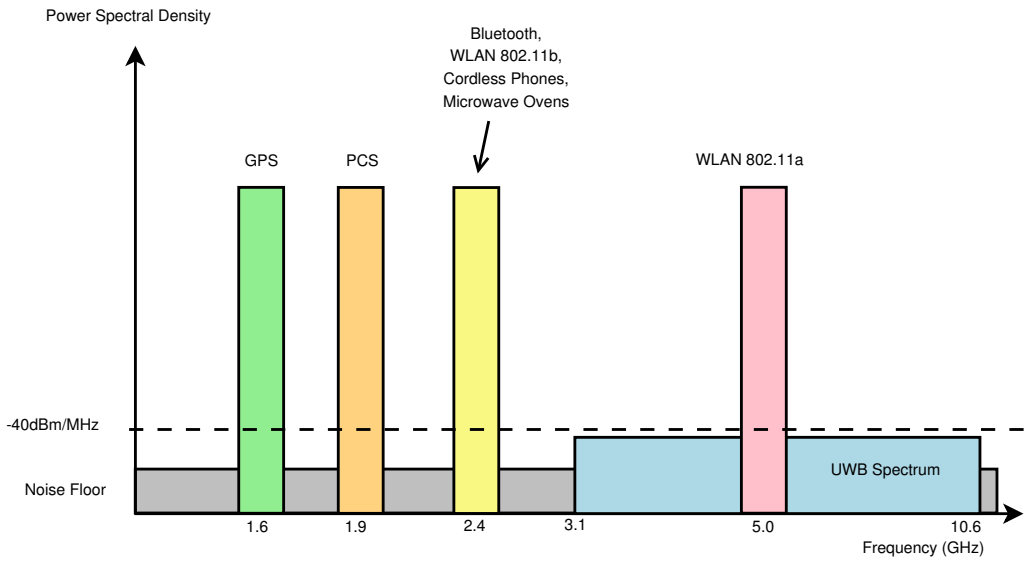


Figure 2.9: The UWB Spectrum [adjusted from 41]

of $1-2cm$ was achieved by Neto et al. [15]. Woodman [16] reduced an position error of more than $150m$ to $5m$ after $60s$ of operation.

The expected accuracy seems to be way too low to locate the robot arm. Therefore, other technologies should be considered for recalibration.

2.2 Ultra-Wideband (UWB)

Radio-Frequency Technology UWB is a Radio-Frequency (RF) technology. The term RF signal refers to wireless electromagnetic signals with frequencies between $3Hz$ and $300GHz$ used for communication. They propagate at speed of light and do not depend on a medium like air. Other RF technologies include RFID, WLAN and Bluetooth.

As RF waves can travel through walls and human bodies easily, they are able to cover larger areas with less hardware than when using other absolute positioning technologies. It is also possible to reuse existing RF technology systems, for example already existing access points in WLAN technology. [5, 40]

UWB According to the US-agency Federal Communication Commission (FCC), which regulates UWB, the allowed frequency band for UWB ranges from $1.6GHz$ to $10.6GHz$. UWB uses frequencies from $3.1GHz$ to $10.6GHz$, which is a broader band than the one of the other technologies. This fact is illustrated by figure 2.9, which also shows that the UWB frequency band interferes with the WLAN frequency

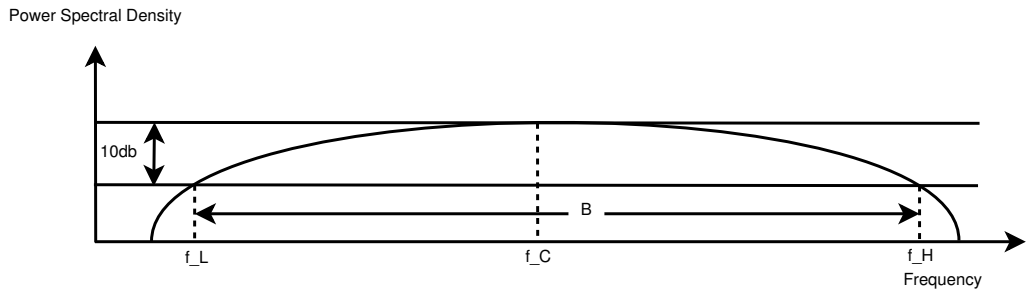


Figure 2.10: Bandwidth and Frequencies of an UWB System [adjusted from 42]

of $5.0GHz$. However, WLAN has a much higher PSD compared to UWB. The PSD is computed by squaring the Discrete Fourier Transform (DFT) and scaling it to N and F_s, where N is the amount of rectangles used to approximate the integral of the Continuous Fourier Transform (CFT) and F_s is the sampling frequency. Furthermore, UWB's large frequency band results in a lower power consumption per antenna, which is close to the noise floor.

UWB has an absolute bandwidth B larger than $500MHz$ or a fractional bandwidth B_{frac} larger than $0.2 - 0.25$. The fractional bandwidth is determined by

$$B_{frac} = \frac{B}{f_c} = \frac{f_H - f_L}{f_c} \tag{2.28}$$

The higher it is, the wider is the band of the antenna.

In the equation above, f_H and f_L are the highest and lowest frequency that the antenna is operating at. In UWB, the central frequency f_c, where the system's maximum power density is located, is smaller than $2.5GHz$. Furthermore, the frequencies f_H and f_L determine where the Power Spectral Density is $10dB$ below the one at f_c as shown in Figure 2.10.

The UWB Physical Layer (PHY) is specified in the *IEEE 802.15.14a* standard. It contains sixteen channels within three frequency bands:

- sub-gigahertz: $250 - 750MHz$, single channel
- low-band: $3,244 - 4,742MHz$, channels 1 to 4
- high-band: $5,944 - 10,234MHz$, channels 5 to 15

UWB devices in each band operate independently of the other bands. Possible data rates for the three bands are $110kb/s$, $851kb/s$ (nominal), $6.81Mb/s$ and $27.24Mb/s$. The range is between ten and one-hundred metres.

Information are transmitted via pulses, which are generated analogically. Therefore, no transmit gain is required. Signals are differentiated in the antennas. The modulation of pulses allows to encode information. Modulation techniques include

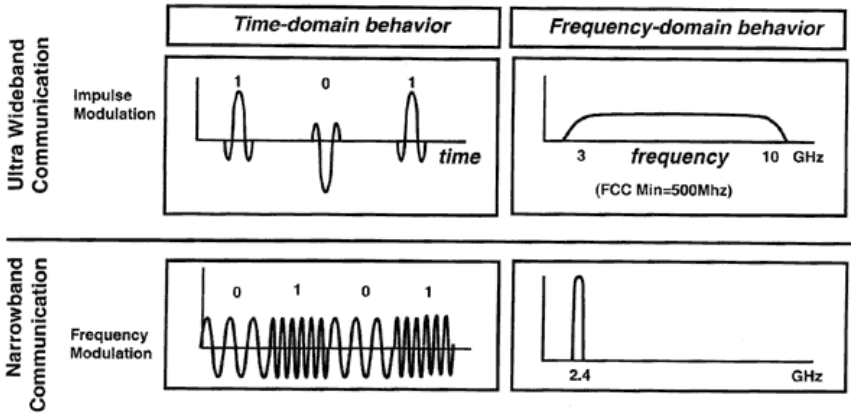


Figure 2.11: Information Encoding in UWB and Narrowband [43]

on/off keying, pulse amplitude modulation and pulse position modulation. Pulses are arranged in frames which determine the minimum time difference between two pulses. This differentiates UWB from WLAN, etc., where information are encoded using frequency modulation, as shown in figure 2.11.

UWB pulses are of a very short duration of less then $1ns$. They also have a short lifetime due to the inverse relationship between time and frequency. The high time resolution is the result of the high frequency. The high bandwidth makes UWB suitable for high-speed communication. Furthermore, it allows to occupy lower frequencies, so signals can more easily pass through walls and obstacles. Therefore, line-of-sight transmission is not necessarily required. Filters separate reflected signals from the original one, which increases accuracy. Additionally, UWB is able to cover a quite large area (i.e. more than one room).

Commercial UWB positioning devices are offered by Ubisense, Time Domain, decaWave, Zebra and nanotron. Furthermore, Kok et al. [44] present an IPS combining inertial and UWB measurements. De Angelis et al. [13] proposed indoor localization using UWB and an INS. A tightly couples UWB/INS system for pedestrian indoor scenarios is presented by Ascher et al. [14]. Finall, Zwirello et al. [11] did a study on techniques for UWB/INS integration. [5, 6, 41, 42, 45, 46, 47, 48, 49]

2.2.1 Position Estimation using UWB

Position can be defined differently depending on the specific use case. For example, it may mean that a target is close to a reference. Another aspect is how different targets are located relative to each other. Finally, an absolute location may be required. [6]

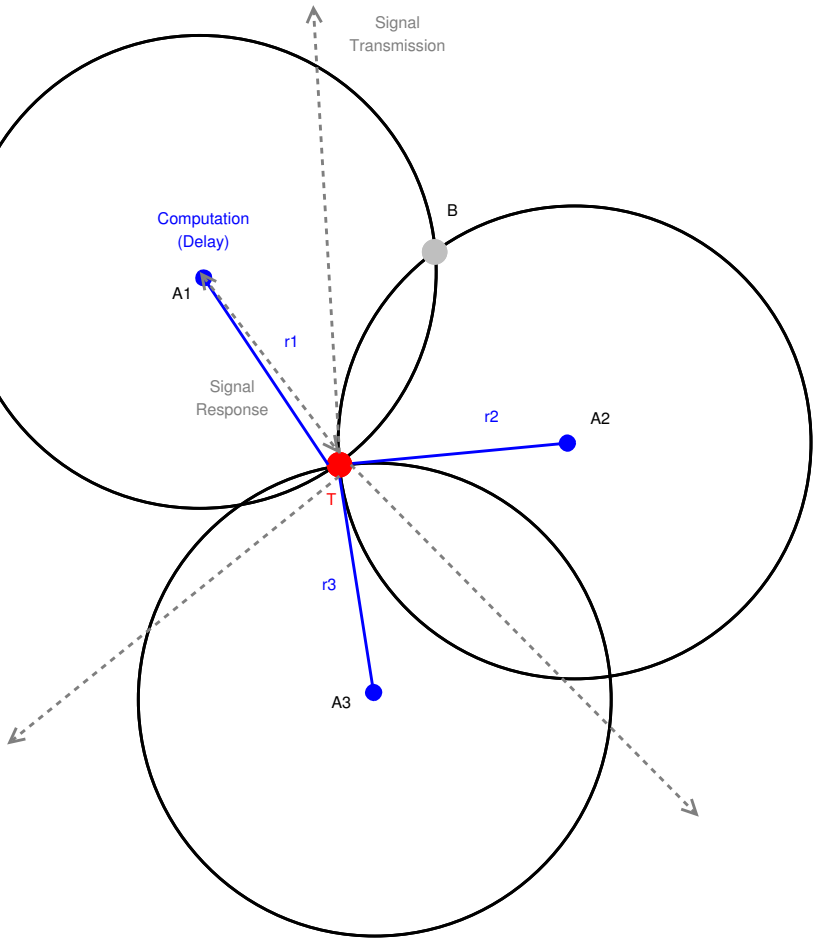


Figure 2.12: Time-based Trilateration

In context of the task of this thesis, only absolute location information are relevant, as the position of the robot within a room has to be determined.

Time-based Trilateration

The concept used in this work is called *time-based trilateration*. The idea of *trilateration* is shown in figure 2.12. The system consists of the target T and three anchors A_1, A_2, and A_3, whose distances to the target are known. If considering only one anchor A_1, the target has to be on a circle with radius r_1 around the anchor. The same applies to the other two anchors, A_2, and A_3, with a radius of r_2 and r_3, respectively. Finally, the target's position has to be at the intersection of the three circles around the anchors. Note that the two anchors A_1 and A_2 are not enough to locate the target, as there would be two possible positions at the two intersections (T and B) of the circles around each anchor.

But how are the distances between the anchors and the target determined? This is the *time-based* part of this method, where the signal's propagation time from one anchor to the target is used to compute the distance between them. In the applied *Time of Flight (TOF)* approach, *two-way ranging*, the sender sends a poll frame to the receiver, who replies with the timestamps of receiving the senders message and sending its reply. The sender then uses this data and the timestamps of sending the poll frame, receiving the response, its own computation and chip delay, as well as the speed of light to compute the distance. The anchors' and target's clocks do not have to be perfectly synchronized is this case, if target and anchors are accurately using their local clocks to compute their local delay.

Let $t_{s,T}$ be the time T sends the poll frame, $t_{r,A}$ the time one anchor receives the signal, $t_{s,A}$ the time the anchor sends the response and $t_{r,T}$ the time the target receives the response. With $t_{s,T} < t_{r,A} < t_{s,A} < t_{r,T}$, if T measures

$$t_T = t_{r,T} - t_{s,T} \tag{2.29}$$

and A measures

$$t_A = t_{s,A} - t_{r,A} \tag{2.30}$$

the TOF is estimated by

$$t_{flight} = \frac{t_T - t_A}{2} \tag{2.31}$$

As signals travel with the speed of light c, the distance d is computed by

$$d = ct_{flight} \tag{2.32}$$

with $c = 299,792,458\frac{m}{s}$. [6, 50]

2.2.2 Errors and Expected Accuracy

There are different sources of error. As mentioned before, clocks are an issue. Even a small divergence in *clock* frequency offset results in clock drift errors. Another source of error is *noise*. Furthermore, *sampling* decreases accuracy as the estimation space is divided into range bins. Continuous tracking, averaging or filtering may improve the results. Finally, *multi-path* channel effects add to the overall error. They occur due to signals being reflected by objects in the environment which causes the signal to arrive at the receiver through many different paths. Techniques to improve multi-path effects include increasing the bandwidth, estimating the channel impulse response and multi-path bias reduction. [50]

The predicted *accuracy* in literature varies between centimeters and meters. Official data sheets, e.g. by DecaWave Ltd. [51], state an accuracy of $10cm$. Lanzisera et al.

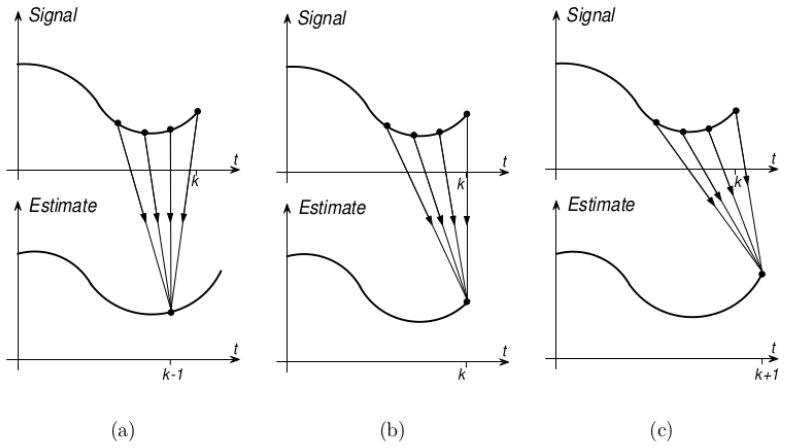


Figure 2.13: Smoothing (a), Filtering (b) and Prediction (c) [54]

[50] achieved an accuracy of $1 - 3m$, while Ingram et al. [52] and Bellusci [53] state that a sub-meter and even centimeter-level accuracy can be reached.

In fact, the centimeter-level accuracy does not seem to be enough to locate the robot arm. Therefore, it needs to be enhanced.

2.3 Enhancement of Results

Usually, sensor measurements are noisy and, in digital systems, taken at discrete points of time. Therefore, they may be smoothed, filtered and/or fused with measurements of other sensors to reconstruct the actual parameter. If the observation vector z_k is obtained at time k, and the process state vector x_{k+m} is supposed to be estimated, there are three different cases:

- *Smoothing* ($m < 0$): The process parameter shall be reconstructed *after* a series of measurements have been taken. For each point of time, measurements from before, now and after are taken to estimate the process variable value. This usually happens offline, after the measurements have been taken in real time.
- *Filtering* ($m = 0$): The actual state of the process is supposed to be determined using information from the previous and the actual measurements. It may be applied in real time.
- *Prediction* ($m > 0$): The current process state shall be guessed using previous measurements only. A good system model is required to obtain a meaningful estimation. Prediction is also applied in real time.

The different methods are illustrated in figure 2.13. [54]

2.3.1 Smoothing and Filtering

Smoothing is the process of transforming noisy and edgy data into an even and regular curve. Methods for smoothing include the moving average and different filters, which block some frequencies and let others pass. Both require a window of size w, which determines how many samples are used in the smoothing or filtering process. [54]

Moving Average

The moving average x_k at time k is determined by calculating the average of all samples z_i between time $k - w - 1$ and time k:

$$x_k = \frac{\sum_{i=0}^{w-1} z_{k-i}}{w - 1} \tag{2.33}$$

For example, let $w = 20$. The first value, which can be computed, is x_{20}. It is derived by adding the values of samples z_1 to z_{20} and dividing their sum by 20. Note, that there is a delay between signal input and output. Its length is equal to the window size w. [55, keyword *filter*]

Finite Impulse Response (FIR) Filter

Finite Impulse Response (FIR) filters are one type of Digital Signal Processing (DSP) filters. Their impulse response is finite, as there is no feedback. Therefore, the FIR filter is stable. For example, let the input of the filter be a single 1, followed by many zeroes. After the 1 went through the filter, there will only be zeros coming out of it. The opposite kind of filter is the Infinite Impulse Response (IIR), where there is feedback. Its output may ring indefinitely after the input of an impulse.

The specification of a low-pass filter is shown in figure 2.14. *Gain* defines the ratio of the output amplitudes to the input amplitudes of the filter. The low-pass filter's bands are set, so that frequencies lower than the *cutoff frequency* f_c can pass and frequencies higher than f_c are attenuated (reduced in magnitude). Due to the filter not being ideal, a tolerance in the different bands has to be specified (grey area in figure 2.14). For example, the passband's width determines how quickly the filter transitions from pass to stop.

Low-pass filters are able to reduce noise and thus smooth data because they block high frequencies, i.e. quick and big changes in the value, and let low frequencies pass (small changes in the value over a longer period of time). [49, 56, 57, 58]

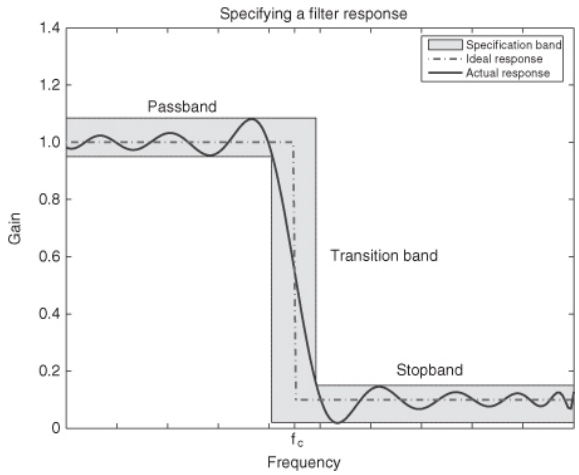


Figure 2.14: Low-pass Filter Response Specification [56]

2.3.2 Sensor Fusion

In order to improve noisy data coming from a single sensor, it can be combined with the data derived from other sensors. The process of combining sensor data or data derived from sensor data, to enhance it in form of an internal representation of the process environment, is called *sensor fusion*. It tries to achieve robustness of the measurement, extended spacial and temporal coverage, a higher confidence in the derived data, less ambiguity and uncertainty, as well as an improved resolution. Criticism on sensor fusion is that it is impossible to create good fused data out of bad raw sensor data.

Two examples of sensor fusion are shown in figure 2.15. The first fusion takes place within the IMU, where sensor readings from gyroscope, accelerometer and magnetometer are combined to retrieve information about rotation, velocity and position. These parameters are input for another sensor fusion process, where they are fused with data from UWB modules to retrieve an absolute position.

There are three categories of sensor fusion depending on the fusion level. *Low-level fusion* combines raw data of several sources, *intermediate-level fusion* combines features, such as edges, corners and lines, into a feature map, and *high-level fusion* combines decisions from different experts. All fusion examples, shown in figure 2.15, are low-level fusions.

Another classification distinguishes sensor fusion depending on the sensor configuration. *Complementary* sensor configuration describes the situation where the sensors do not depend on each other, but may be combined to give a more comprehensive image of the observed phenomenon. In a *competitive* configuration, different sensors measure the same property independently and redundantly. Finally, *coorperative*

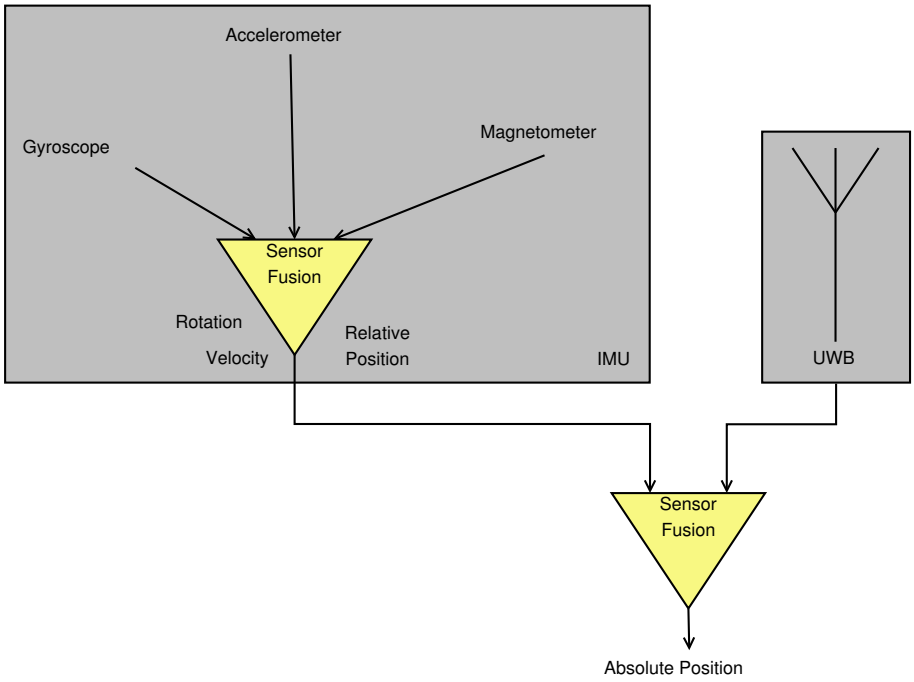


Figure 2.15: Sensor Fusion of IMU and UWB

sensor networks combine the information of independent sensors to derive information about the observation that would not be available if one of the sensors was missing.

There are different filters available for data fusion. They include the *complementary filter*, the *Kalman filter* and the *Mahony filter*. OlliW [59] gives a good overview on how they work, as well as their advantages and disadvantages. [54, 59]

Complementary Filter

As mentioned before, gyroscopes are reliable and accurate in the short-term, while accelerometer data is stable in the long run. However, if moved, the later also measures acceleration other than gravity alone. The idea of a complementary filter is to combine the best of both.

The process of fusing gyroscope and accelerometer data is shown in figure 2.16. The basic idea is to keep the high frequencies of the gyroscope and remove any drift resulting from low frequencies. Therefore, a *high-pass filter* is applied before integrating the angular rate to get the angle. For the long-term stable accelerometer, a *low-pass filter* is applied to remove any short-term (i.e. high-frequent) fluctuations. The rotation obtained by both sensors is then combined to determine the fused rotation.

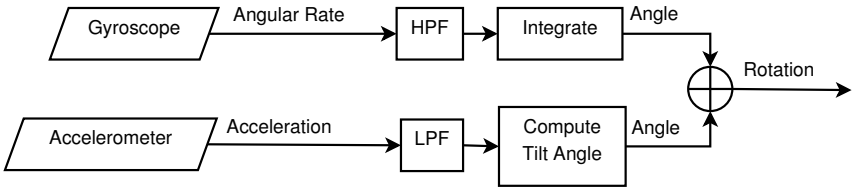


Figure 2.16: Complementary Filter: Gyroscope and Accelerometer Fusion [derived from 60]

The fusion of the rotation derived from both sensors is done by simple giving both values a weight and adding them:

$$rotation = weight_{gyro} * rotation_{gyro} + (1 - weight_{gyro}) * rotation_{accel} \quad (2.34)$$

Van de Maele [61] suggests a value of 0.98 for $weight_{gyro}$. As advantages of the complementary filter, he states the ease of use and the low computation power required. [60, 61]

Kalman Filter

The Kalman filter, which was developed in 1960, is a stochastic filter that makes use of a mathematical model. It can be applied to signals with statistical and systematic errors. The filter may be used for smoothing by estimating the most likely next value for a set of measurement values. However, its main application is the data fusion of different sensors into an internal state vector. By using a time-discrete algorithm, it removes noise from sensor signals.

The idea for the subsequent explanation is to fuse data from control inputs, e.g. the throttle and/or break in a car, with sensor measurements, i.e. accelerometers, to obtain the desired system state (which may be the velocity and/or position in this case). Imagine that there is no speedometer in the car to measure velocity directly. This is congruent to the *hidden Markov model*, which describes a hidden system state y_k, that can only be observed through the observation model z_k.

The Kalman filter consists of two phases, which are displayed in figure 2.17 and will be explained subsequently.

Prediction (Time Update) This phase tries to predict the current state and the error covariance matrix. The predicted state y_k^- at time k is determined by

$$y_k^- = Ay_{k-1} + Bu_k \quad (2.35)$$

where A is the model that predicts the new state from the previous state y_{k-1}, and b is the model that predicts what changes using the current command input u_k. Some

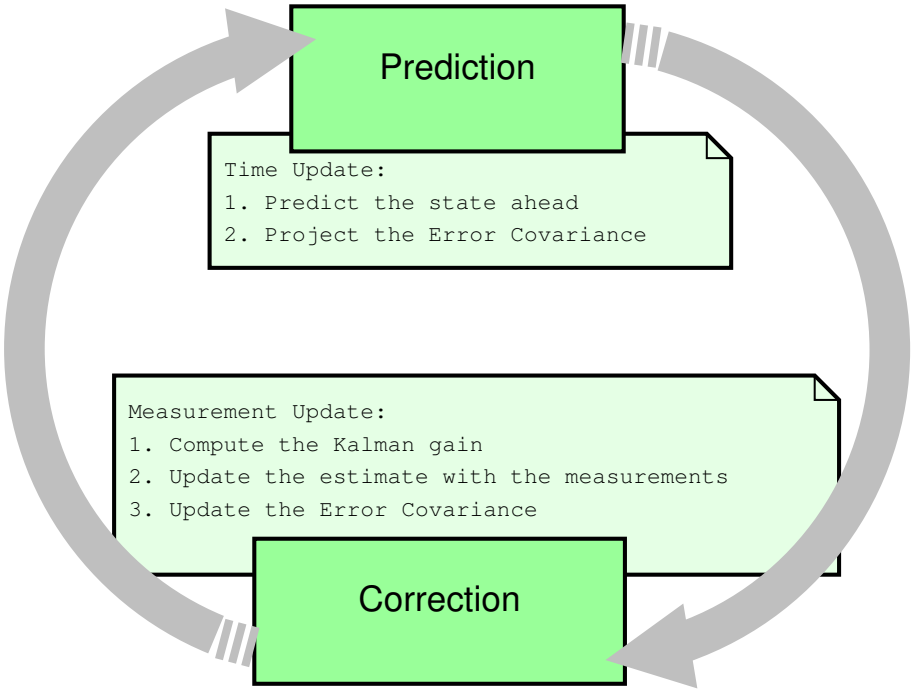


Figure 2.17: Phases of the Kalman Filter [adapted from 62]

sources also add a random variable for Gaussian white system noise w with a zero mean.

The second step of *prediction* projects the error covariance P_k^- at time k:

$$P_k^- = AP_{k-1}A^T + Q \tag{2.36}$$

where A is the model of motion, P_{k-1} is the previous error covariance at time $k-1$, A^T is the transposed model, and Q is the covariance of error noise.

Correction (Measurement Update) In the second phase, the Kalman gain is computed, and there is an update of the estimate with the measurement, as well as an update of the error covariance. At first, the Kalman gain K is computed by

$$K = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.37}$$

with P_k^- being the projected error covariance from (2.36), H being the model of how the sensor readings reflect the system state, H^T being the transposed model, and R being the sensor measurement noise.

The second step is to update the estimate y_k^- from (2.35) with measurements from the sensors z_k by computing

$$y_k = y_k^- + K(z_k - Hy_k^-) \tag{2.38}$$

where y_k is the system state at time k, K is the Kalman gain from (2.37) and H is the model of how the measurement reflects the state.

The final step is to update the error covariance P_k by computing

$$P_k = (I - KH)P_k^- \tag{2.39}$$

with I being the identity matrix, K being the Kalman gain from (2.37), H being the measurement model, and P_k^- being the error noise, previously estimated by (2.36).

The Kalman filter determines the best estimate by adjusting the error covariance depending on the difference between the measurement z_k and the predicted state y_k^- computed in (2.38). The bigger the measurement noise, the larger the value of P. The larger P is, the larger is K and the less the measurement is trusted. In this case, the filter trusts the prediction more. Overall, the filter itself corrects the covariance matrix depending on how much the estimate has been corrected in (2.38).

Nowadays, the Kalman filter seems to be the state of the art for sensor fusion. However, there is one important limitation. Due to linear models, the filter does not always yield satisfying results. In the case of non-linear systems, a modification, the *extended Kalman filter* has to be used. [54, 62, 63]

2.3.3 Expected Accuracy for Combining IMU and UWB

In literature, the expected accuracy for sensor fusion of IMU and UWB varies. Zwirello et al. [11], who used a Kalman filter, mention that 30% of their measurements were more than $10cm$ off. Herrera et al. [12] also used a Kalman filter and reported standard deviations of $4 - 6cm$. Borràs Sillero [64] used a complementary Kalman filter and did not find much improvement compared to using UWB data only. De Angelis et al. [13] used an extended Kalman filter and reported an accuracy of $5cm$. Kok et al. [44] achieved a Root Mean Square Error (RMSE) of $3cm$. A seamless indoor/outdoor navigation system fusing inertial, UWB and GPS was designed by Tanigawa et al. [65], who report an accuracy of less than $20cm$.

Even the best reported accuracy is not going to be enough for the roboter arm to be localized in this work. A better accuracy has to be achieved for it to be able to pick up objects.

3 Hardware Selection and System Setup

The previous chapters described the requirements of an IPS and some technologies for indoor localization, namely IMUs and UWB. In this chapter, the hardware selection process is explained and the final system setup is shown. As mentioned in section 1.2, there was no budget to buy new hardware. Therefore, existing hardware had to be used.

3.1 Robot Arm

The robot arm already existed from former research projects. It was designed and assembled by Dr. Andreas Kugel. The outline is shown in figure 3.1a and a screenshot of the CAD design is shown in figure 3.1b (p. 32). The solid parts, except for the grippers, were printed using a 3D printer. They are $12cm$ long, which is known from Dr. Kugels draft. The grippers are $2cm$ long. Due to these lengths, the robot arm is a little more than $40cm$ high, if all joints are stretched out and the robot is straight.

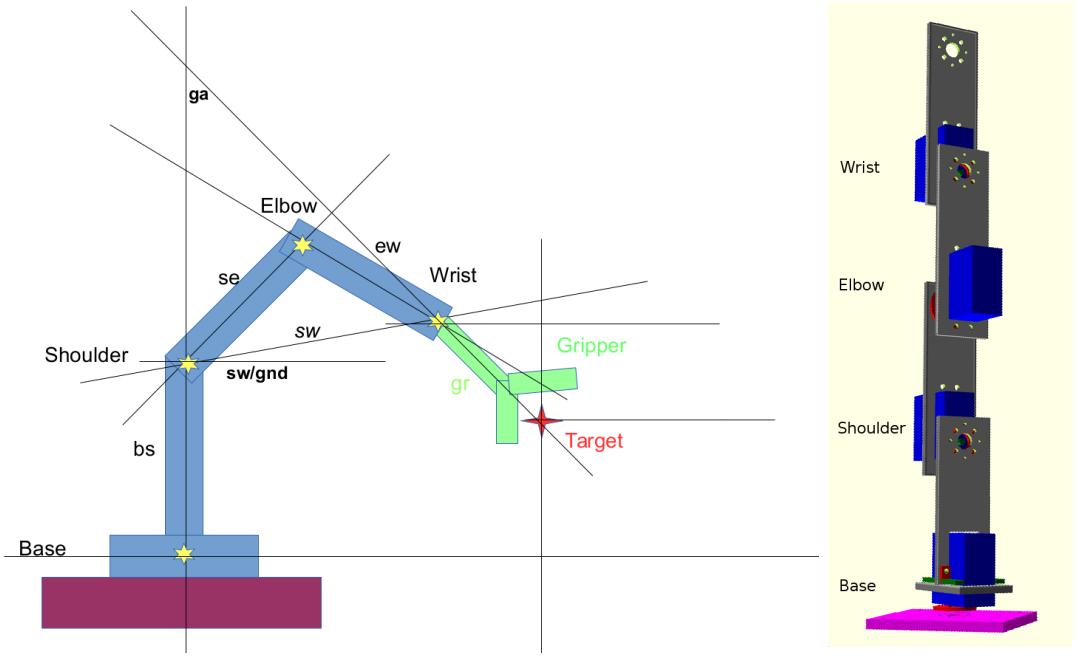
3.1.1 Joints and Degrees of Freedom

As shown in figure 3.1a, the robot arm has five joints, each offering one degree of freedom. Firstly, the *base* joint allows the robot arm to rotate around its own axis to face the target. Then, *shoulder*, *elbow*, and *wrist* joints are moved into the correct angles using inverse kinematics to align the grippers with the target. The required computations are explained in section 4.1. Finally, the *grippers* (*thumb* and *finger*) are closed to grab the target.

3.1.2 Motor Control using Pulse Width Modulation

Pulse Width Modulation

The joint movements are performed by servo motors. They are operated using Pulse Width Modulation (PWM), which involves sending a modulated wave-pulse. A servo expects continuous pulses to hold its position or move. The *frequency* determines how often a positive pulse is fed during a unit period. For example, if the frequency



(a) Outline [adapted from 66]

(b) CAD Design

Figure 3.1: Robot Arm

is $50Hz$, there will be a positive pulse 50 times per second. As time is the inverse of frequency, one pulse has to be sent to the servo every $20ms$ to stay in the same angular position.

The width of the positive pulse (or square wave) is called *duty cycle* and determines the servo's angular position. For example, if the servo has a range of 180°, the center position is at 90°. The pulse length of the center position is obtained from the data sheet of the particular servo. If, for example, the center position is at a pulse length of $1.5ms$ and the minimum and maximum positions are at a pulse length of $1ms$ and $2ms$, every pulse between $1ms$ and $1.5ms$ moves the servo in one direction and each pulse between $1.5ms$ and $2ms$ in the other direction. This example is shown in figure 3.2.

Servo Motors

In this project, the *base* and *shoulder* joints are "DF Metal Geared $15kg$ Standard Servos (DSS-M15)" with an operating frequency of $50 - 300Hz$. Their pulse width range is $500 - 2500\mu s$ with the neutral position at $1500\mu s$. If the pulse width is below $1500\mu s$, the servo rotates clockwise.

The *elbow* and *wrist* joints are plastic-geared "Hitec Deluxe Standard Servos (HS-422)" with a pulse cycle of $20ms$ and a pulse width of $600 - 2400\mu s$ (according to

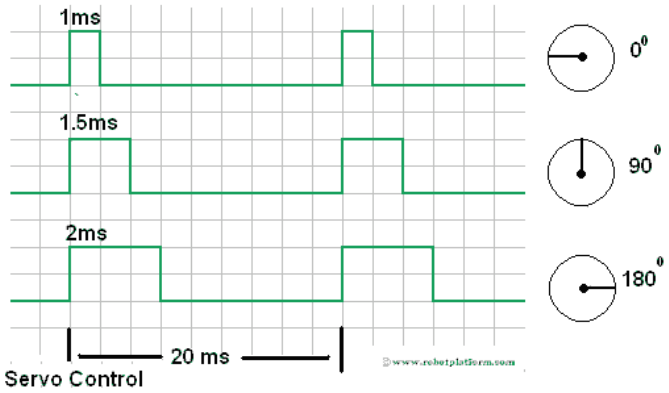


Figure 3.2: Example of a Servo Control [67]

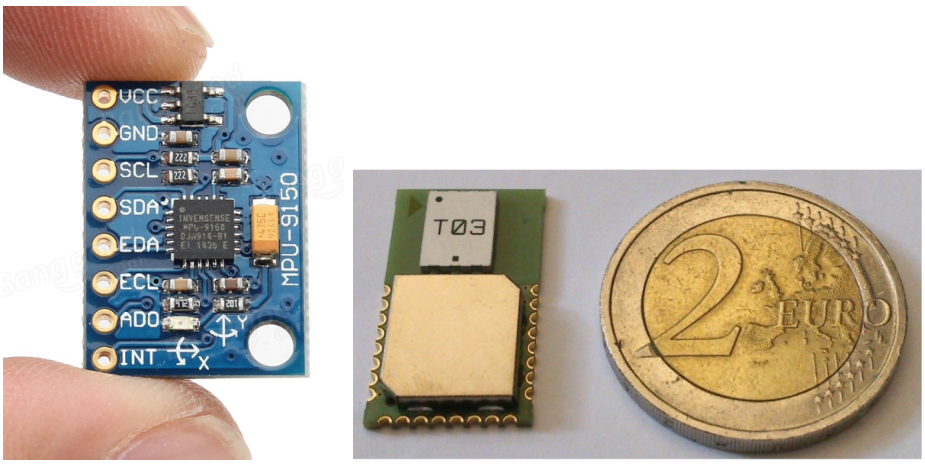
Joint	Center Pulse	Clockwise	Counter-clockwise
Base	$1500\mu s$	$< 1500\mu s$	$> 1500\mu s$
Shoulder			
Elbow			
Wrist		$> 1500\mu s$	$< 1500\mu s$
Thumb		$< 1500\mu s$	$> 1500\mu s$
Finger		$> 1500\mu s$	$< 1500\mu s$

Table 3.1: Servo Center Pulse and Movement Direction

the data sheet). If the pulse width is above $1500\mu s$, the servo moves clockwise. However, as the elbow servo is fixed upside down, it has to be rotated the other way around to move the gripper towards the target.

The *gripper* joints are "Goteck micro metal gear servos ($2.5kg$)". The neutral position is at a pulse length of $1500\mu s$ and the servo moves counter-clockwise at pulse widths higher than $1500\mu s$. Note that one gripper (finger) servo is mounted upside down, so it has to be rotated into the opposite direction of the other gripper in order to close the grip.

Considering the fixture of the servos, the pulse widths relative to the center pulse width of the *wrist* and *finger* servos have to be moved in the opposite to the other joints in order to move the grippers closer to the target. The movement direction in relation to the pulse width for each servo is summarized in table 3.1. [67, 68, 69, 70, 71, 72, 73]



(a) MPU-9150 [74] (b) DWM1000 [51]

Figure 3.3: Sensors used in the Project

3.2 IMU

Just like the robot arm, the chosen IMU was available from past research projects. It is a *MPU-9150*, a 9-axis motion tracking device by InvenSense Inc., combining a 3-axis gyroscope, a 3-axis accelerometer and a 3-axis magnetometer. It also contains an on-chip temperature sensor and a Digital Motion Processor (DMP) to accelerate hardware processing or take load off the host processor. The *MPU-9150* is shown in figure 3.3a.

Different kinds of motion-based applications are featured by the DMP. In addition, data can be read in bursts, using the on-chip 1024-byte First-In-First-Out (FIFO) buffer, and the host processor can enter a low-power state between reads. Furthermore, there are user-programmable interrupts available for gesture recognition, tap detection, and others. One helpful feature is the output of quaternions by the sensor fusion data of gyroscope and accelerometer.

The *MPU-9150* allows to connect other non-inertial digital sensors like pressure sensors. Furthermore, in contains three 16-bit ADCs for digitalization of each gyroscope and accelerometer output, as well as one 13-bit ADC for each magnetometer output. To precisely track fast and slow motion, the MPU-9150 features user-programmable Full-Scale Ranges (FSRs) for all thee kinds of sensors. The FSR determines the minimum and maximum measurement values, that can be mapped onto the minimum and maximum output values of the ADC.

All communication with the IMU is done via I^2C with a maximum clock frequency of $400kHz$. Therefore, the `SDA` and `SCL` pins have to be connected, as well as `VDD` and `GND` for power supply. The level on the `ADO` pin determines Least Significant

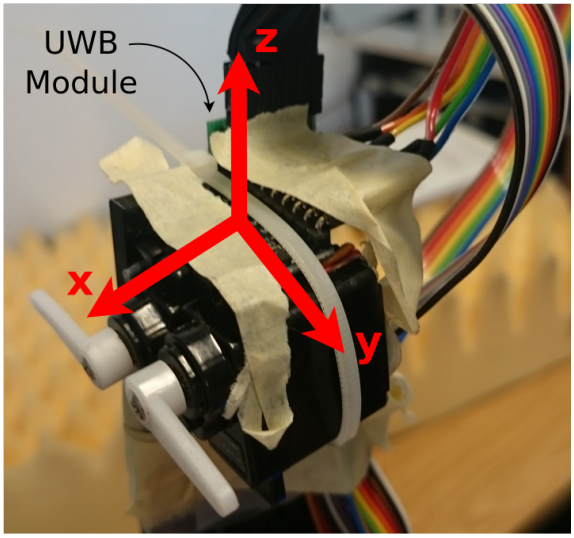


Figure 3.4: Coordinate System of the IMU attached to the Robot Arm

Bit (LSB) of the IMU's I^2C slave address. Finally, the `INT` pin may be used to receive interrupts from the MPU-9150. [37, 75]

In the final system setup, the IMU is attached to the robot arm, close to the grippers, in order to get accurate location results. Its x-axis faces towards the grippers, while its z-axis points upwards, as shown in figure 3.4.

3.3 UWB Module

For this project, there were four *DWM1000* modules by decaWave available. Each of them integrates an antenna, all required RF circuitry, power management and clock circuitry. According to the manufacturer, the *DWM1000* modules may be used for two-way ranging or in Time Difference of Arrival (TDOA) location systems to locate the target with a precision of $10cm$ within ranges of up to $300m$. The *DWM1000* is shown in figure 3.3b.

The UWB module is complient to *IEEE 802.15.4-2011*, as well as FCC and European Telecommunications Standards Institute (ETSI) UWB spectral masks. It supports four RF bands from $3.5GHz$ to $6.5GHz$. The transmit power density is programmable from $-35dBm/MHz$ to $-62dBm/MHz$ and the module supports data rates of up to $6.8Mb/s$. The communication with the host processor is done via Serial Peripheral Interface (SPI) bus with a maximum frequency of $20Mhz$. Furthermore, there is a configurable Always-On memory, which may be used to store the configuration data during the low power operational states, when the on-chip regulators are disabled. The data up- and download is done automatically. [51, 76]

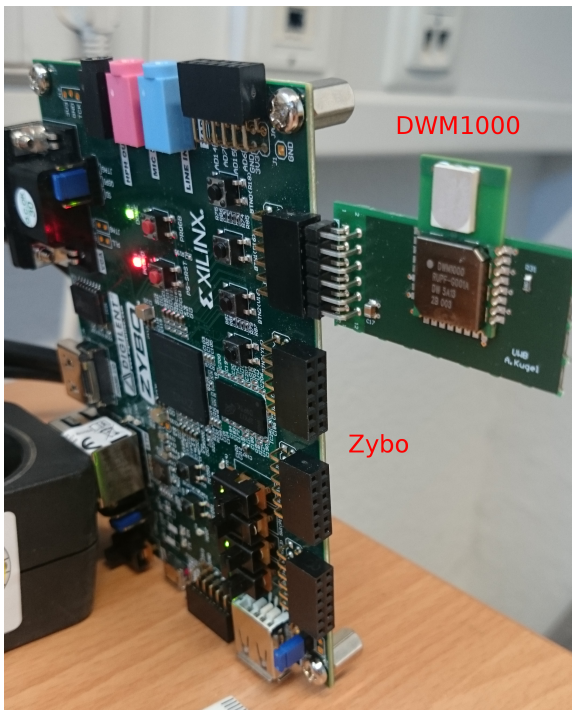


Figure 3.5: UWB Anchor Connected to the Zybo

At the end, two of the four UWB modules are used. One of them, which is used as the anchor, is connected to a *Zybo Zynq-7000 ARM/FPGA SoC*, which has been configured by Dr. Kugel. Therefore, it simply has to be powered on for the UWB module to work. This setup is shown in figure 3.5. The other module is attached to the robot arm close to the grippers, as shown in figure 3.4. Due to the arrangement of the anchor and the robot, only the grippers' location on *one* axis (the IMU's x-axis) can be determined using UWB technology.

3.4 ZedBoard

The goal of this thesis was to localize and control a robot arm. Therefore, a control instance was required. As mentioned in section 3.1, the servo motors of the robot are moved using PWM. As there was a complete PWM module written in VHDL already available, the board of choice was the ZedBoard Zynq-7000 ARM/FPGA SoC Development Board by Digilent, which is a System on a Chip (SoC) containing a Zynq-7000 ARM processor and 7-series Programmable Logic (PL) by Xilinx. One of its main application, besides software acceleration, is motor control. Additionally, the ZedBoard offers LEDs, switches and buttons for user interaction, as well as user I/O, which is used to connect to the servos of robot arm, the IMU and the UWB module. [77, 78]

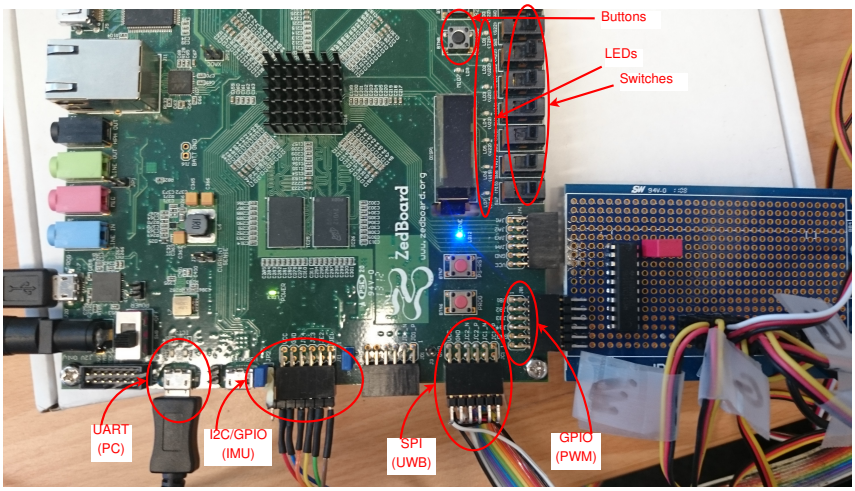


Figure 3.6: ZedBoard and Connected Wires

The different components on the ZedBoard are shown in figure 3.6. The ZedBoard's internal configuration and the interfaces to the different sensors and the PWM module will be explained subsequently.

Block Design

The *block design* of the ZedBoard is shown in figure 3.8. It contains a Zynq-7000 Processing System (PS) and an A*XI Interconnect* to connect it to the different peripherals (red). The *Processor System Reset* (red) is responsible for the processor resets. The two General-Purpose Input/Outputs (GPIOs) `axi_gpio_0` and `axi_gpio_1` are the interfaces to the available LEDs, switches and buttons (purple). The AXI Timer generates a timer interrupt for the PS and can be configured in software (orange). The PWM module `pwm_0` is an Intellectual Property (IP) that packages the PWM functionality (brown). The two constants `pwm_dir_1` and `pwm_oe_0`, with value 1 and 0, are required to configure the PWM modules' direction and *output enable* (brown). On the right bottom part of the diagram, there are all the blocks required for communicating with the UWB module (green). This includes an *AXI Quad SPI*, an *AXI GPIO*, some I/O buffers created by Dr. Kugel, as well as a couple of slices, constants, vectors logics and a concat. This part was adopted as it stands from a design by Dr. Kugel. The Concatenation on the left side combines the interrupts of different interrupt sources and forwards them to the PS (blue).

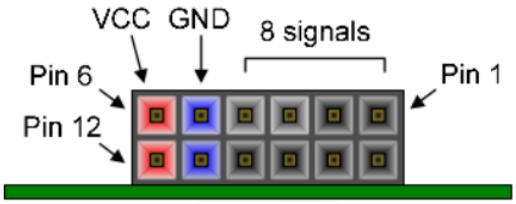


Figure 3.7: ZedBoard's User I/O Pins [adapted from 79]

PWM Signal	ZedBoard Pin
pwmPulse[0]	JB1
pwmPulse[1]	JB2
pwmPulse[2]	JB9
pwmPulse[3]	JB3
pwmPulse[4]	JB10
pwmPulse[5]	JB4
pwmDir	JB7
pwmOe	JB8

Table 3.2: Signal Mapping: PWM Module – ZedBoard Pin

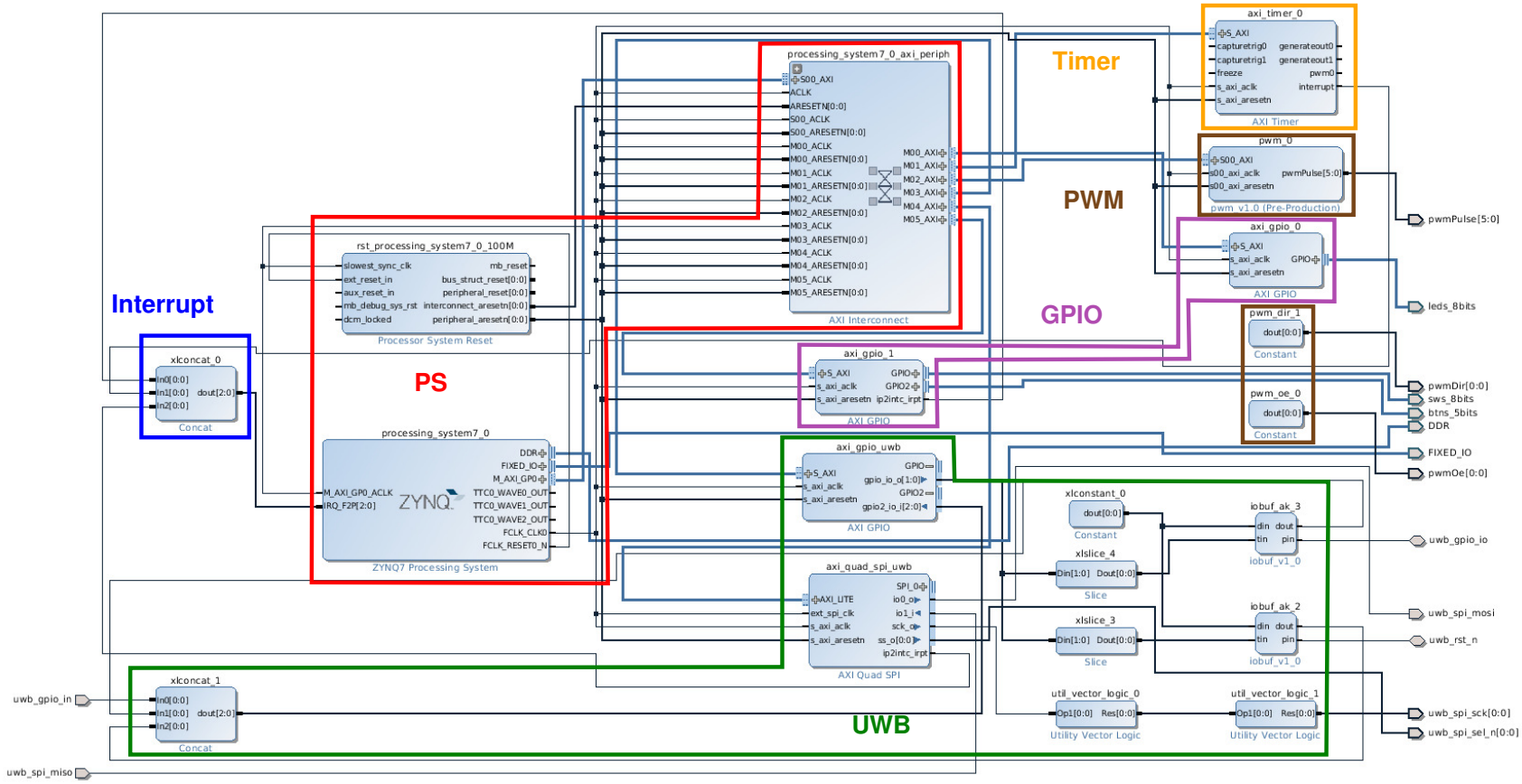


Figure 3.8: ZedBoard Block Diagram

The PWM Module

The PWM IP is configurable, such that the amount of required PWM channels can be defined by the user. In this project, there are six channels required – one for each servo. For the software application, there are twelve registers available, that determine the width of the positive pulse and the time interval for each servo. The default values are set to be $1.5ms$ pulse width and a time interval of $20ms$ between two pulses. The PWM component takes the values from the registers and outputs the corresponding pulses to the equivalent output port in `pwmPulse`, which is mapped to the user I/O pins `JB1..4` and `JB9..10`. The structure of the ZedBoard's user I/O pins is shown in figure 3.7. The constant signals `pwmDir` and `pwmOe` are mapped to `JB7` and `JB8`. The overall mapping is shown in table 3.2.

UWB

The DWM1000 module's pins have to be connected to the ZedBoard to facilitate power supply, communication via SPI and control. The `JC` Peripheral Module (PMOD) has been chosen at the ZedBoard. One of the UWB modules' GPIO pins is connected to the ZedBoard, but it is not used for the application. Table 3.4 summarizes the connection. [76]

PS and I²C Setup

The configuration of the Zynq-7000 PS is shown in figure 3.10. All used components are marked green. Used I/O peripherals are I^2C to communicate with the IMU, Universal Asynchronous Receiver/Transmitter (UART) to send information to the PC, as well as GPIO.

For I^2C communication, the signals `SDA` and `SCL` are mapped to the Multiplexed Input/Output (MIO) pins 10 and 11. `MIO12` and `MIO13` are also reserved for the communication with the IMU. `MIO12` is connected to `AD0` and `MIO13` to `INT`. The communication with both is facilitated by the GPIO of the PS. The mapping of the IMU's signals to the ZedBoard's pins `JE1` to `JE4` and their corresponding MIO numbers are shown in table 3.3.

3.5 System Setup

The final system setup is displayed in figure 3.9. Note that due to the alignment with the UWB anchor and the robot arm, only its displacement along one axis (the IMU's x-axis) can be computed.

***MPU-9150* Pin**	**ZedBoard Pin**	**MIO Number**
VCC	VCC	-
GND	GND	-
AD0	JE4	12
SDA	JE3	11
SCL	JE2	10
INT	JE1	13

Table 3.3: Signal Mapping: MPU-9150 – ZedBoard – Zynq-7000

***DWM1000* Pin**	**ZedBoard Pin**
VDD3V3	VCC
GND	GND
RSTn	JC4P
SPICLK	JC2N
SPIMISO	JC2P
SPIMOSI	JC1N
SPICSn	JC1P

Table 3.4: Signal Mapping: DWM1000 – ZedBoard

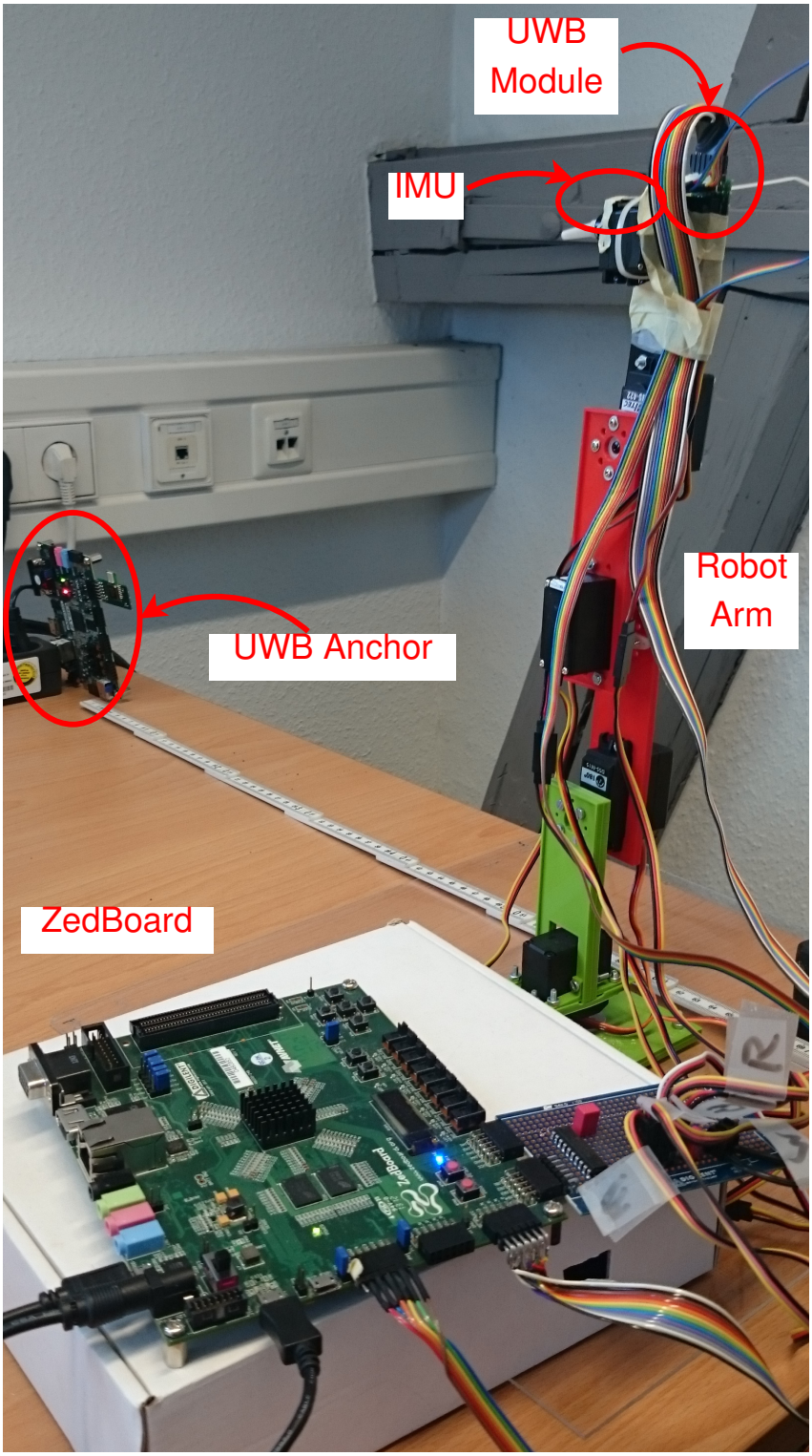


Figure 3.9: Final System Setup

The corresponding block diagram is shown in figure 3.11. The IMU is configured and data is read out and processed by the ZedBoard's PS via I^2C using MIO. The communication with the UWB module is facilitated by an SPI module in the PL. For SPI and I^2C transfers, there are wrapper functions provided by the corresponding Board Support Package (BSP). The PWM module for controlling the robot arm's servos is located in the PL and contains registers, in which the software application can write values, using the generated wrappers. The main loop is in control of all software functionality and has access to the buttons, switches and LEDs by using GPIO modules. The connection between the PS and PL is facilitated by the *AMBA AXI4 Interconnect*. The communication with the PC is done via UART.

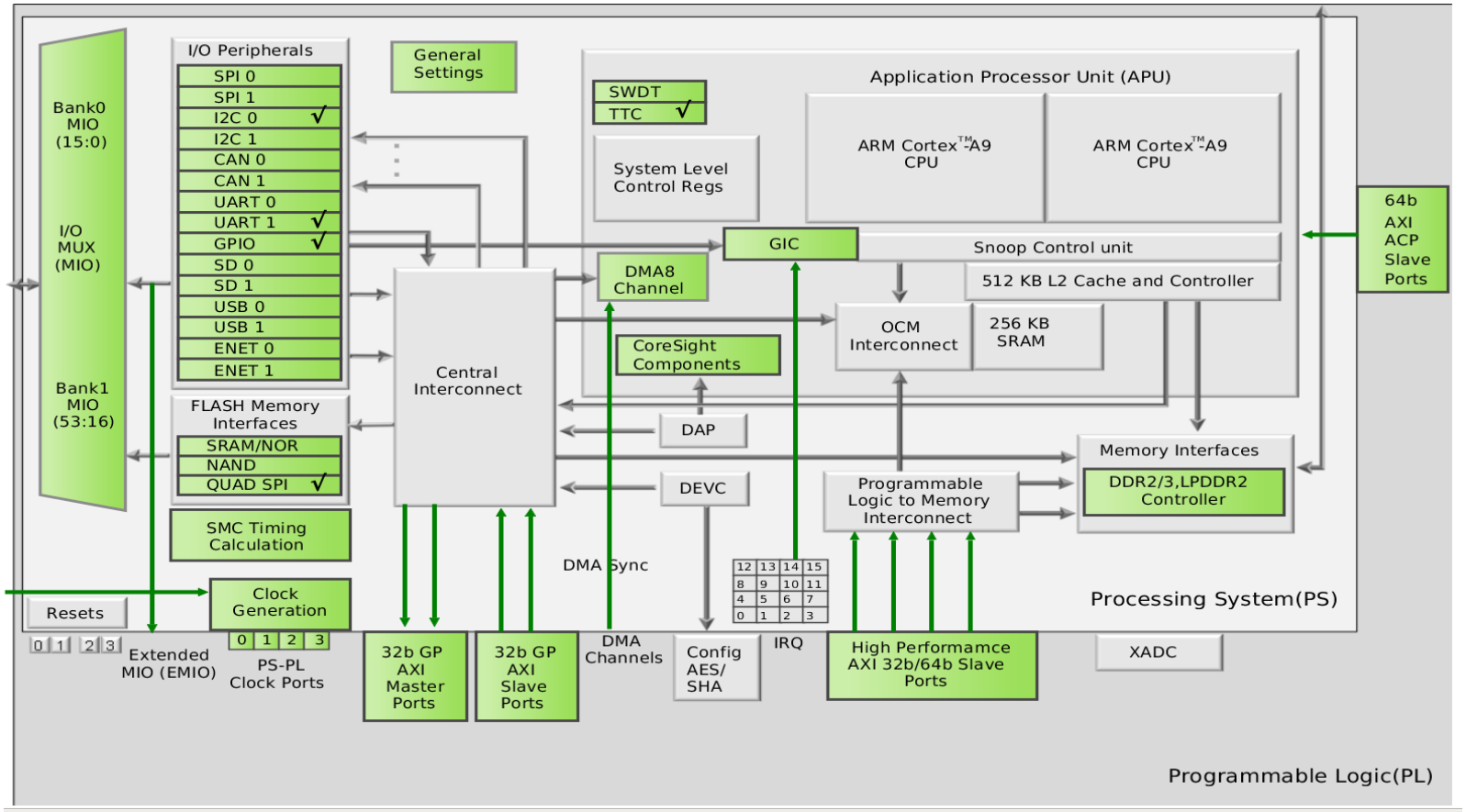


Figure 3.10: ZedBoard Zynq-7000 PS Configuration

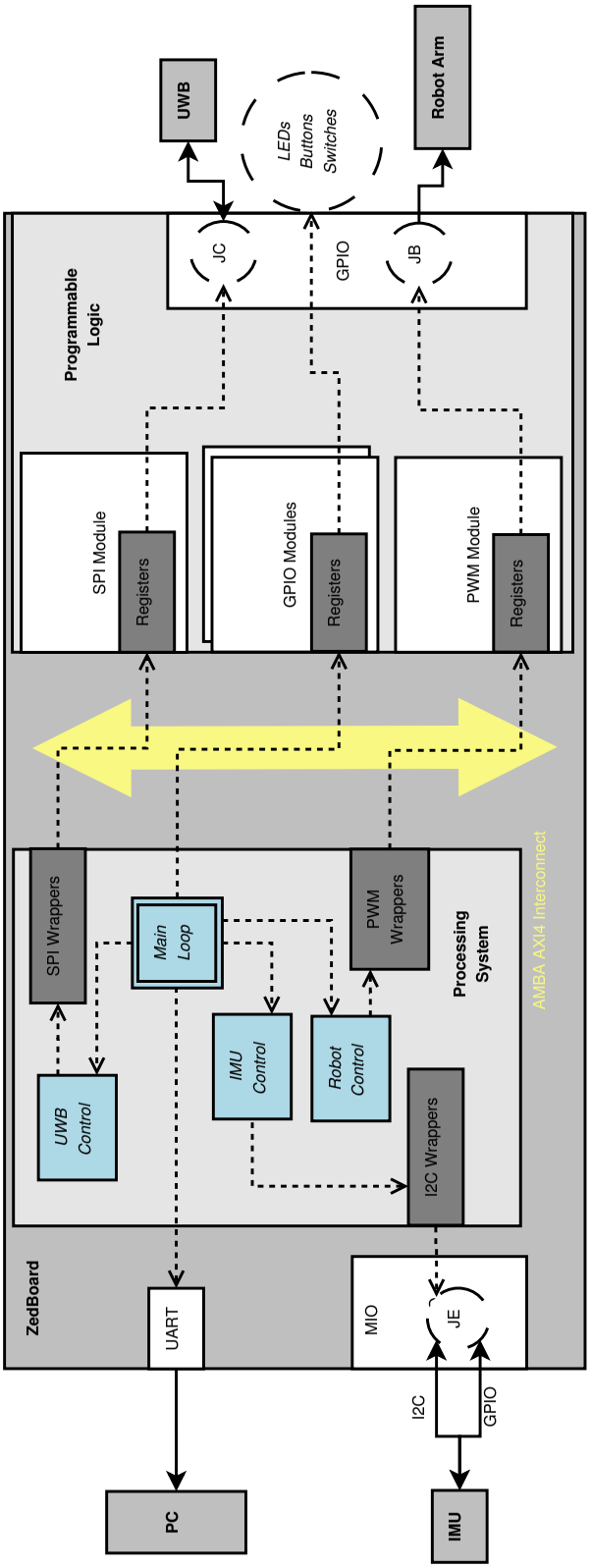


Figure 3.11: System Block Diagram

4 Software Architecture

The software contains different components, whose structure and interaction is shown in figure 4.1. At first, there is the general *Utilities* component, which can be accessed from all other modules. It includes the configuration and use of I^2C and *GPIO*, as well as *Time* and *Timer Interrupt* utilities.

For the communication with the PC, there are the *Print* and *XModem* modules. The latter transmits data, which has been stored in the *Data Buffer* before.

In addition, there are *IMU* and *UWB Control* components, whose *Application* modules access the device-specific drivers. Each driver uses device-specific *Utilities* to access the hardware. Sometimes, the hardware is directly accessed by the *Application*, skipping the *Driver* instance.

The *IMU control* also contains an *Interrupt Control* module, which is fed by either the interrupt coming from the IMU itself or by the *Timer Interrupt* within the ZedBoard.

Another component contains everything related to *Robot Control* and PWM. The *Main Loop* is in control of all components. The functionality of the different components will be explained subsequently.

4.1 Robot Control

4.1.1 PWM Control

As mentioned in section 3.1, the robot's joints are moved using PWM. The PWM module inside the ZedBoard's PL features twelve registers for the software to configure the time interval between PWM pulses and the actual pulse length for each of the six outputs. The robot control software provides the following functionality:

- register read and write operations using low-level functions provided by the GPIO driver,
- resetting the registers to a default value of $20ms$ for the time interval and $1.5ms$ for the pulse width using the functions above,
- conversion of an angle in degrees to the corresponding pulse width and vice versa, and

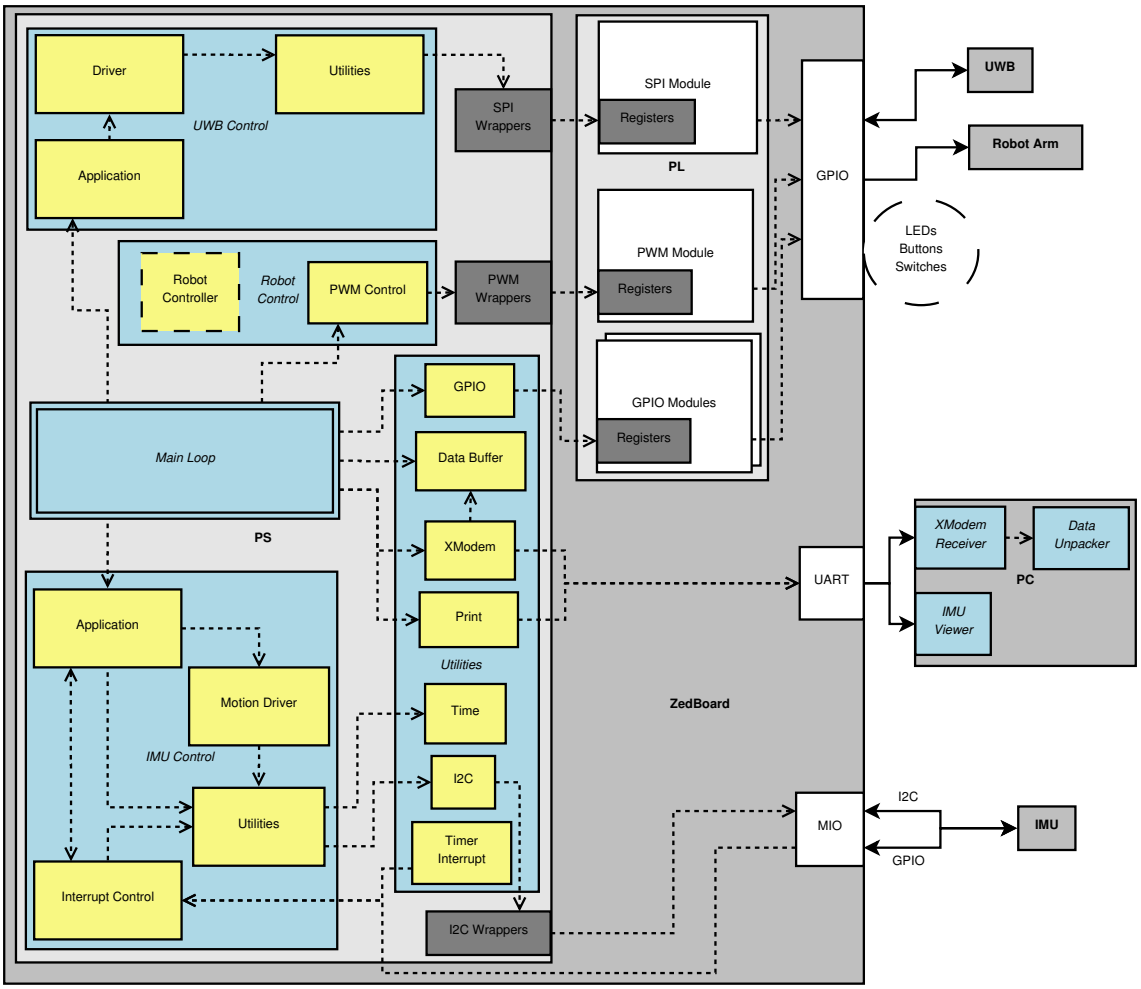


Figure 4.1: Software Architecture

- moving a specified joint servo to a given angle.

Conversion between Pulse Width and Joint Angle

It is assumed that, when the robot arm is pointing upwards (i.e. all joints are streched out and the grippers are open), all angles are 0°. The conversions between the joints' angles (in °) and the corresponding PWM pulse (in μs), as displayed in figure 4.2 and 4.3, are no smooth linear functions. There is a different slope between $-90°$ and 90°. Furthermore, the y-intersection of the functions between $-180°$ and $-90°$, as well as 90° and 180°, differ.

Additionally, the functions for the *wrist* and *finger* joints are opposite to the others, due to the way in which they are attached. The slope and y-intersections of the conversion functions from angle to pulse width for the different joints are summarized in table 4.1. The functions for converting pulse widths to angles are shown in table 4.2.

In the implemented software, before the conversions are processed, there is a check whether the input angle or pulse length is within the allowed range.

Servo Movement

In order to move a servo to a specified angle, the angle has to be converted to a PWM pulse first. After a check, whether the specified value is within the allowed range, it is written into the corresponding register. As a result, the servo moves towards the angle in a sudden, quick motion. In order to ensure safe servo operation, a smooth and not too fast servo rotation needs to be implemented in the future.

4.1.2 Robot Controller

In the future, a robot controller may be implemented in order to use the sensor data to compute the required angles and control the servos to grab an object autonomously.

To determine the joint angles, such that the grippers are able to grab the target, inverse kinematics can be applied. In the course of this thesis, there was not enough time to implement this. However, all the required equations have already been derived. They, as well as their deduction, can be found in the appendix in section A.2 (p. 124) and B (p. 129).

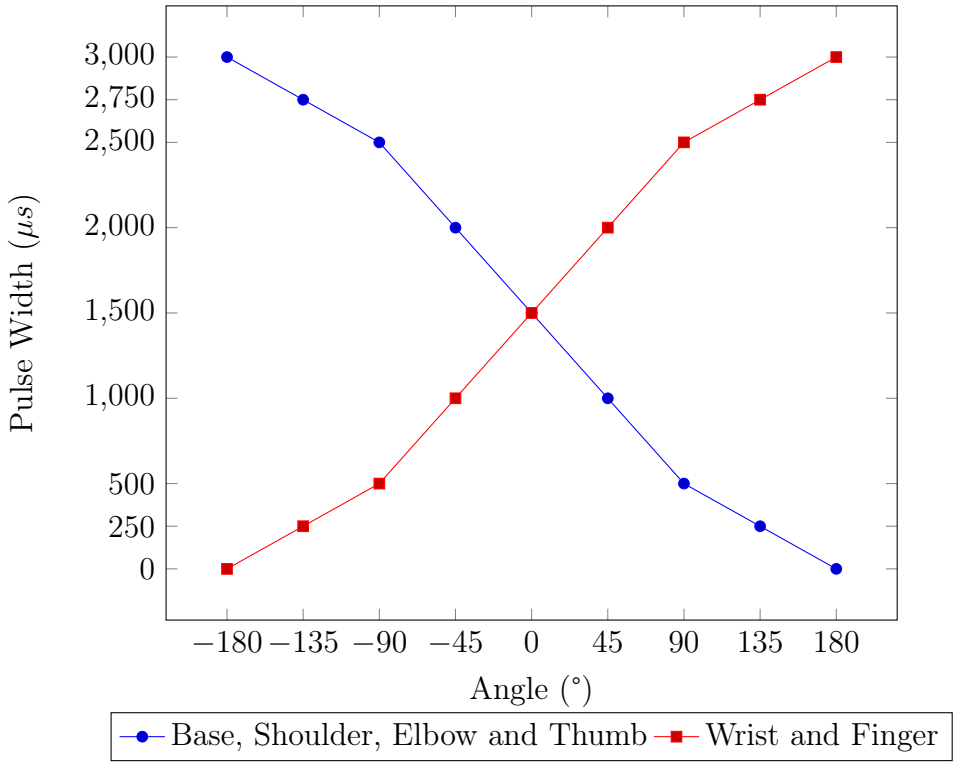


Figure 4.2: Angle vs. Pulse Length

Joint	$-180° \leq \alpha < -90°$	$-90° \leq \alpha \leq 90°$	$90° < \alpha \leq 180°$
Base Shoulder Elbow Thumb	$p = -\frac{50}{9}\alpha + 2000$	$p = -\frac{100}{9}\alpha + 1500$	$p = -\frac{50}{9}\alpha + 1000$
Wrist Finger	$p = \frac{50}{9}\alpha + 1000$	$p = \frac{100}{9}\alpha + 1500$	$p = \frac{50}{9}\alpha + 2000$

Table 4.1: Conversion from Angle to Pulse Width

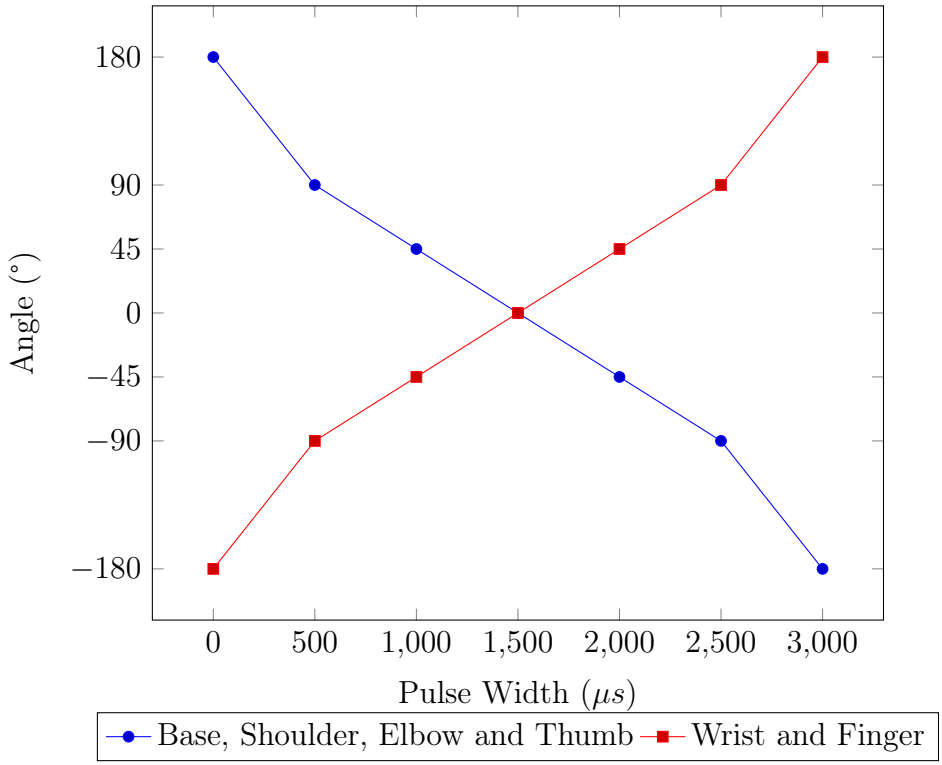


Figure 4.3: Pulse Length vs. Angle

Joint	$0\mu s \leq p < 500\mu s$	$500\mu s \leq p \leq 1500\mu s$	$1500\mu s < p \leq 3000\mu s$
Base Shoulder Elbow Thumb	$\alpha = -\frac{9}{50}p + 180$	$\alpha = -\frac{9}{100}p + 135$	$\alpha = -\frac{9}{50}p + 360$
Wrist Finger	$\alpha = \frac{9}{50}p - 180$	$\alpha = \frac{9}{100}p - 135$	$\alpha = \frac{9}{50}p - 360$

Table 4.2: Conversion from Pulse Width to Angle

4.2 IMU Control

4.2.1 Motion Driver

To simplify the use of the MPU-9150 IMU, its manufacturer InvenSense offers a so-called *Motion Driver* without additional charge. It is "an embedded software stack of the sensor driver layer that easily configures and leverages many of the features of the InvenSense motion tracking solutions" [75]. Furthermore, the Motion Driver contains defines for register addresses and bit maps, as well as functions for the

- initialization and configuration of the IMU,
- configuration of the sensors,
- readout of all sensor values,
- initialization and configuration of the Digital Motion Processor (DMP) and its features, and
- configuration of the FIFO used to read out sensor data and quaternions from the DMP.

4.2.2 Utilities

In order to use any feature of the *Motion Driver*, the user has to implement the following six functions

- `i2c_write` — an I^2C burst write
- `i2c_read` — a simple I^2C read
- `delay_ms` — lets the processor sleep for the specified amount of ms
- `get_ms` — gets the current runtime in ms
- `log_i` — logging of information messages
- `log_e` — logging of error messages

and define them in the *Motion Driver* files specified in InvenSense [80]. [75, 80]

I²C Read and Write

The ZedBoard's PS is used to communicate with the IMU using I²C, where the PS is the master and the IMU acts as the slave. There are low-level functions (*Wrappers*) for polled I²C send and receive as master device available in the BSP, which has been generated for the specified hardware description by the Xilinx software. These read and write methods are called in the read and write functions which are described be subsequently.

I²C Read The I²C read function requires a pointer to the I²C driver instance, the slave's address, the register, a pointer to the data and the number of bytes as input. Then, the following steps are executed:

1. Wait until the bus is idle
2. Set the *Repeated Start* option
3. Send the register address
4. Receive the data of the specified length
5. Clear the *Repeated Start* option

The repeated start option is used to send a command and read back the answer immediately without another master interrupting the atomic operation in a multi-master system. [81]

I²C Burst Write The write function is actually a *burst write*, where the destination address is sent first, followed by all the data (i.e. more than just one byte). The following tasks have to be fulfilled:

1. Create a temporary buffer containing the register address and all data
2. Wait until the bus is idle
3. Send the buffer

The difference to a standard write operation is that rather than sending two bytes of data (i.e. register address and one byte of data) at a time, the register address is sent, followed by all the data without sending the address again.

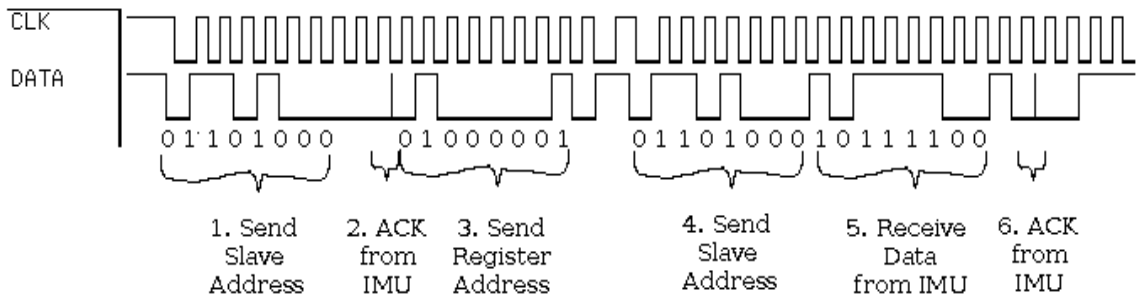


Figure 4.4: Temperature Low-Byte Register Readout Using I²C

Example Transfer Figure 4.4 shows an example I²C read, which has been recorded during real operation. It displays the Serial Clock (SCL) and Serial Data (SDA) signal lines during the readout of the register, which contains the lower byte of the current temperature value, from the IMU.

At first, the processor sends the slave address (`0x68` or `01101000`) over the bus *(1.)* and receives an Acknowledge (ACK) from the IMU *(2.)*. Next, it sends the register address it wants to read from *(3.)*. In this case, it is the address of the lower byte of the temperature value (`0x42` or `01000001`), which is read before the high byte. The next step is to send the slave address again *(4.)*. Then, the content of the required register is transferred over the bus (`10111100`) *(5.)*. The transfer ends with an ACK from the IMU *(6.)*.

The reason for the one longer clock cycle before sending the slave address is doubtful. One cause may be the processing time for operations, which are executed between the two different I²C operations.

Time Functionality

Xilinx uses their *typedef* `XTime` to represent timestamps. In order to convert it to μs and s, this value has to be divided by the Xilinx constants `COUNTS_PER_USECOND` and `COUNTS_PER_SECOND`. The conversion, as well as the computation of the expired runtime of the program, have been implemented.

Delay There is a `usleep` function available in the BSP, but the `delay_ms` function requires the value in ms as input. Therefore, the time in ms is multiplied by 1000 before using it as the input parameter of the `usleep` function.

Timestamp In order to compute the timestamp, i.e. the runtime of the program in ms, the functionality mentioned above is used. The returned runtime in μs is multiplied by 1000 in order to convert it to ms before returning it to the calling

function. In case of long delays, this implementation may cause a variable range overflow. A workaround should be implemented in the future.

Console Output and Logging

Console Output With the aim of preventing unnecessary console output, which increases the application's rumtime, a custom *printf* function, called `myprintf`, has been implemented. In the corresponding header file, there is a define, called `DEBUG`, which is set if all outputs should be enabled. If it is not set, all function calls of `myprintf` are ignored.

Logging Because logging is an unnecessary console output during proper operation, both logging functions `log_i` and `log_e` simply forward the information or error text string to the `myprintf` function, which is printed to the console, depending on the previously mentioned define.

4.2.3 Interrupt Control

Necessity

Using interrupts has been determined to be the only reliable way to communicate with the *MPU-9150*. Simply specifying a FIFO rate and executing a polled read did not work well and lead to various problems including the *Arbitration Lost* phenomenon or a *Non-Acknowledge (NACK)* response from the IMU. The latter also lead to messed-up data reads from the FIFO in the long run.

Both issues are based on the the SDA and SCL signals constantly being pulled high, possibly due to inadequate serial resistors [82]. As a result, the connection to the IMU was lost within a couple of minutes and could not be re-established without resetting both, the *ZedBoard* and the *MPU-9150*.

The resistor configuration in the hardware description was examined and considered to be fine. Even changing the clock frequency or skipping a couple of I^2C reads did not resolve the issue. As this does not allow reliable rotation and position tracking, an interrupt-based communication was implemented.

Implemented Interrupt Modes

There are two different ways to facilitate an interrupt-based communication with the IMU, which are illustrated in figure 4.5. One way is to configure the *IMU*'s interrupt to be triggered, once there is new data available in the FIFO. Another way is to use the interrupt provided by an AXI *Timer* IP.

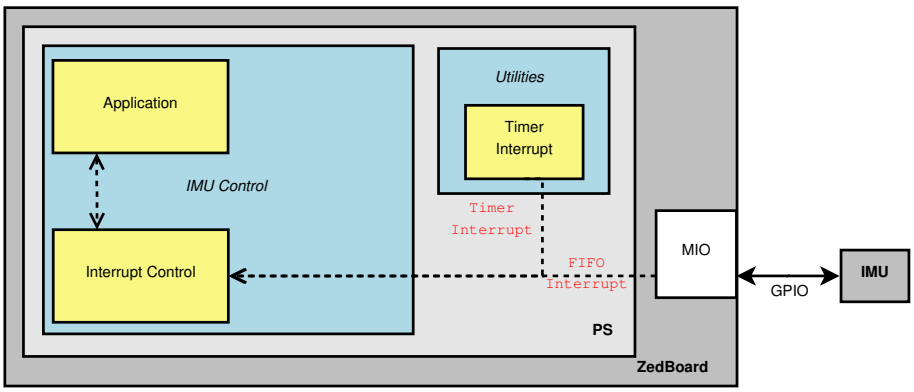


Figure 4.5: Interrupt Setup

The chosen interrupt does not affect the application because both interrupt handler simply set a flag, which is polled by the application. There are different flags for each interrupt, which are set and polled depending on the chosen interrupt source. If the time difference between two samples is equal to the time difference between two timer interrupts, the application works identically.

4.2.4 Application

Use Cases

The implemented application facilitates various use cases, as shown in figure 4.6. There are two different ways to obtain the current rotation. Firstly, the *DMP* can be enabled and used. This keeps load off the processor in the ZedBoard's PS. However, the algorithm is unknown as InvenSense has not published it. Therefore, the user does not know how the quaternions are computed. The user can choose, which sensors are going to be used, but everything else is a black box. Using the orientation determined by the DMP, the position is computed according to the algorithm described in section 2.1.2

The other method is to compute the quaternions from *sensor data* using the algorithm explained in section 2.1.1. The implemented algorithm uses gyroscope data only. The position is computed just as if the DMP were used.

Furthermore, there are two moments when rotation and position can be computed. One of them is during *runtime*, where the computation is done in real time. In this case, the computed quaternions and position are sent to the PC via UART. There are a couple of data sets skipped due to the limited bandwidth of UART. Real-time computation may be used when the application controls the robot arm with the IMU attached to it or when the IMU is detached and moved by hand.

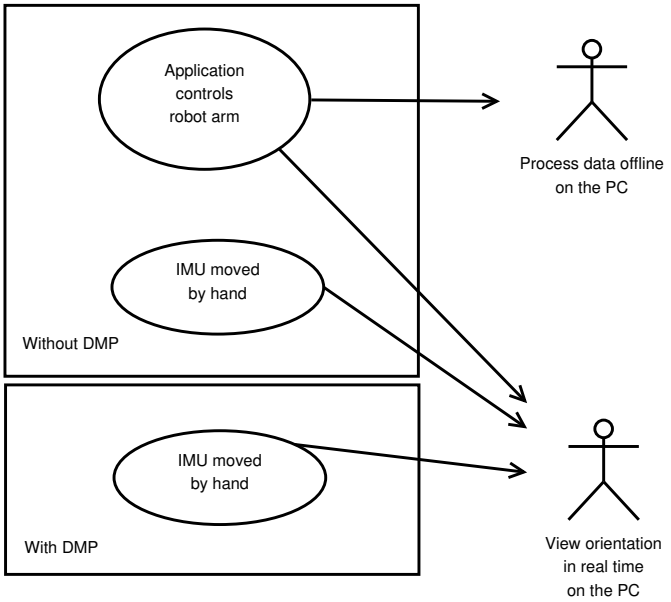


Figure 4.6: Use Cases of the Implemented Software Application

However, for data analysis and the development of filters and data fusion algorithms, it may be required to get *all* sensor data sets. In this case, the collected raw data is stored on the device and transmitted as a whole, using the *XModem* protocol after the data collection has finished. In this way, no data is skipped. However, data transmission takes some time.

The different modes of UART communication will be explained in section 4.4. A guide on how to use to software application for the different use cases can be found in the appendix in section A.1.1 (p. 120).

Program Flow

Initialization Figure 4.7 shows the program flow of the application. At first, different initialization and configuration methods for GPIO (buttons, switches and LEDs), the robot's joint angles, the UWB module and the IMU are processed. There is also an optional IMU sensor calibration. Then, the interrupt (IMU or timer-based) is configured and enabled. Depending on the switch state of the ZedBoard, the UWB antenna delay is calibrated using the push buttons.

The IMU's initialization values, as well as a description of the interrupt setup and handling, can be found in the appendix in section A.3 (p. 127).

Check for interrupt and get PWM data Then, the specified amount of samples is collected. The program continuously checks the flag, which is set by the interrupt

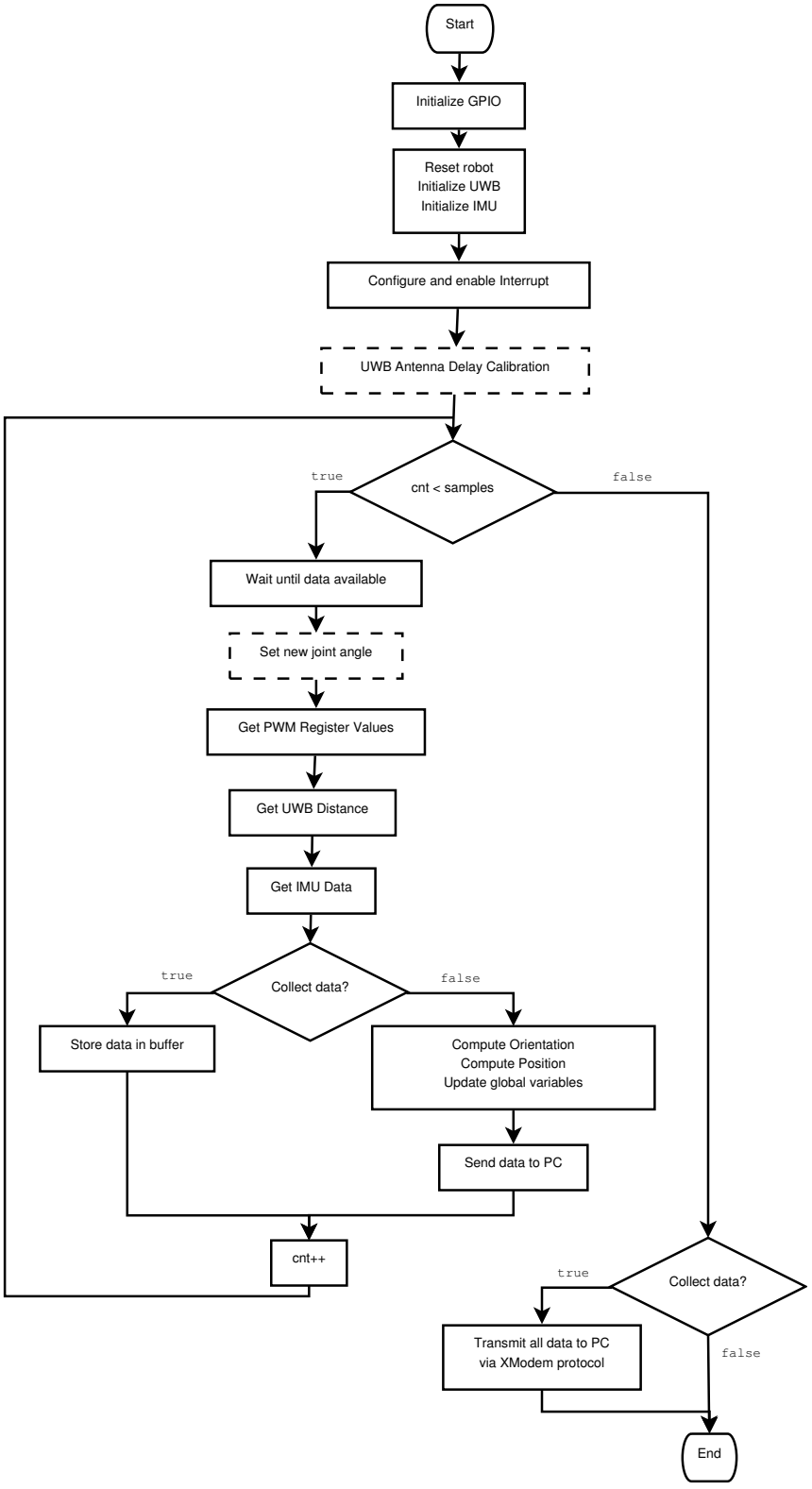


Figure 4.7: Flow Diagram for Collecting Data during Robot Motions

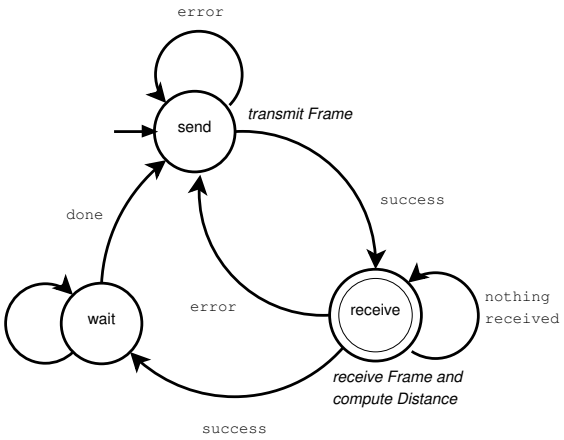


Figure 4.8: State Machine for UWB Distance Determination

handler once an interrupt has been received. There is a time period before the angles of the robot's joints are changed, again due to the high operating frequency. If the corresponding counter has run up to the specified value, a new joint angle is set. Otherwise, the program continues.

Get UWB and IMU data The next step is to get the distance between the UWB module at the robot arm and the anchor. Therefore, the main application triggers a state machine which takes care of this task and returns the distance, if it has successfully been computed. The state machine is explained in section 4.3.

Then, the sensor data from the IMU is collected. This will occur in each iteration, if an interrupt is received.

Finishing up The rest of the program depends on whether the data is processed in real-time or offline. In case of offline computation, the collected data is stored in a buffer and the counter is increased. This process will be explained in section 4.4. Otherwise, the orientation and position are computed using IMU data only. The global variables containing the orientation and position are updated and the next iteration is started.

Data transmission Once the required amount of samples has been collected, the program is done. In case of offline processing, the collected data is transmitted to the PC via the *XModem* protocol before the program is finished.

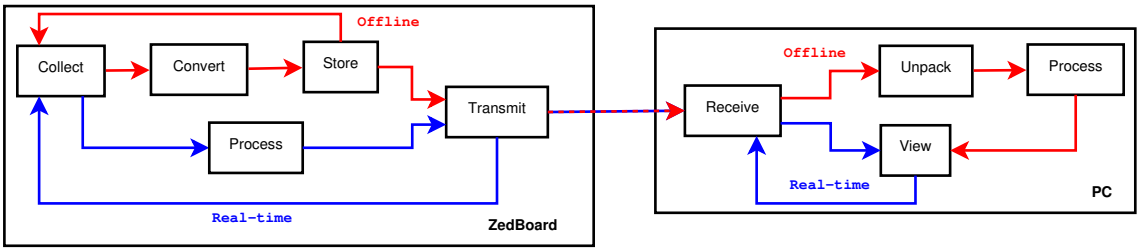


Figure 4.9: Data Flow during Real-time and Offline Processing

4.3 UWB Control

Besides the reset and initialization methods for the *DWM1000* module, the *UWB control* component in figure 4.1 contains a state machine to control the process of computing the distance to the other UWB module. For each call, the state machine checks its current state and calls the corresponding method. The implemented state machine is shown in figure 4.8.

Initially, it is in the *send* state, where it transmits a frame to the anchor UWB module. If the transmission was successful, it changes to the *receive* state.

There, it receives the response frame of the other UWB module and computes the distance as explained in section 2.2.1. In case of an error, it goes back to the *send* state. Otherwise, it makes the computed distance available to the main application. Then, it changes to the *wait* state. If no data has been received, the state machine stays in the *receive* state.

The *wait* state facilitates a wait time between two different distance measurements. This way, the distance is computed with a rate of roughly $20Hz$. If the wait time is over, the state machine switches back to the *send* state, where the whole process starts all over again.

4.4 Communication with the PC

There are two different ways to operate the developed system, as illustrated by figure 4.9. First, the data can be processed on the ZedBoard and transferred to the PC in real time, using the UART interface (*real-time* processing). A corresponding PC application receives the data and displays it for the user. The other way is to collect all data first and transfer it to the PC as a whole, using the *XModem* protocol. There, it is unpacked and further processing may be done (*offline* processing).

Both methods will be described subsequently. Furthermore, a guide on how to use the implemented software can be found in the appendix in section A.1.2 (p. 120).

4.4.1 Real Time Data Transfer

This way to transfer data allows to either view the raw data or the computed orientation and position using an UART terminal program or to view a model of the IMU using the *IMU Viewer*, which will be described in section 4.5. In this mode of operation, the computed data is transferred in the American Standard Code for Information Interchange (ASCII) format.

Depending on the specified sampling rate, not all data sets are sent to the PC, as this causes the I^2C bus stop working. The reason for this may be the time-consuming data transfer being interrupted by the interrupt service routine. Data is transferred to the PC with a frequency of $10Hz$, compared to a sampling frequency which is 20 to 50 times higher.

By experiment, it was observed that the sampling frequency may not be higher than $200Hz$ to keep the I^2C bus running. Using higher frequencies, cause the communication to break down within a few seconds.

4.4.2 XModem Transfer

For a comprehensive data analysis, it is not satisfying accept missing data sets due to the limited transfer bandwidth of UART. Therefore, another means to send exhaustive data to the PC had to be used. This operating mode collects data over a specified period of time and stores it in a data buffer. As soon as the collection phase is over, the data is send to the PC all together, using the *XModem/CRC* protocol.

Data Buffer

During sampling time, all data received from the IMU and UWB module, as well as the PWM register contents, are stored in a data buffer to be transmitted, once the data collection has finished. This buffer is located within the DDR memory of the ZedBoard to make sure there is enough memory available. This may not be the case, if the data is stored in the heap or stack.

Data Structures A data struct for the IMU's sensor values has been developed to store the register data in a well-structured way. It is shown at the bottom of figure 4.10. It contains arrays with three elements, one for each axis, for the different sensors. The temperature value is a data type that requires more space. Thus, its rectangle is displayed larger than the one of the other sensors.

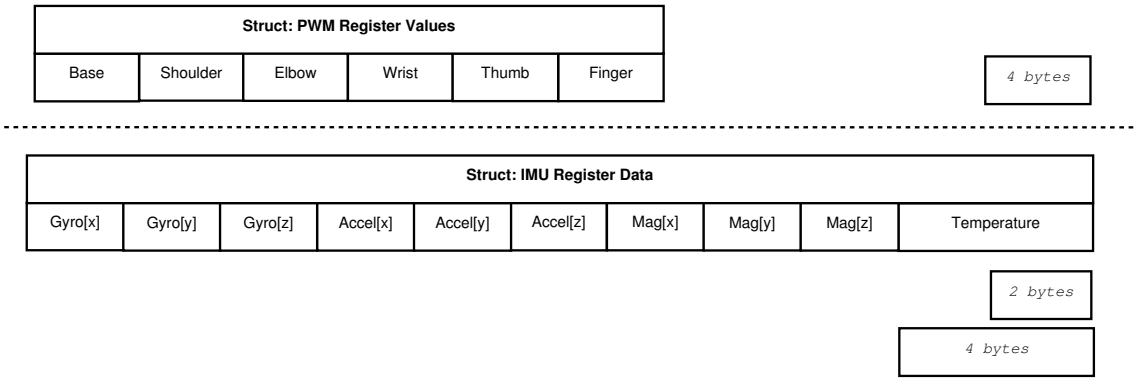


Figure 4.10: Data Structure

Furthermore, there is an array for the values of the PWM registers, which is shown at the top of figure 4.10. Its first element contains the *base* joint's register value, the second one, the *shoulder*'s, etc.

Data Storage As all data to be transmitted over the XModem protocol have to be in binary form, the collected data have to be structured in an arbitrary way, before they can be sent, received and unpacked in the same way. If this process is not predetermined, data unpacking on the receiver's side may yield useless data, which has different reasons. Firstly, it may be caused by different sizes of the used data types. Another reason is the varying endianness of different processor architectures.

Therefore, the order, in which the data, and even the different data types, are stored into the buffer, is predetermined. It is illustrated by figure 4.11. The first four bytes of the buffer are reserved for the amount of samples to be transferred. They are written into the buffer as soon as the data collection has finished.

Once a data set has been collected and temporarily been stored in the data structs in the program stack, they are converted into `char` data and stored in the buffer according to figure 4.11. The lowest (or least significant) byte is stored first, followed by the next byte, and so on, until the most significant byte of this particular value has been stored.

The data type of the number of samples, PWM register data and temperature are four bytes in size while the gyroscope, accelerometer and magnetometer, as well as the distance computed from the UWB data are two bytes in size. Overall, one data set requires 48 bytes in the data buffer.

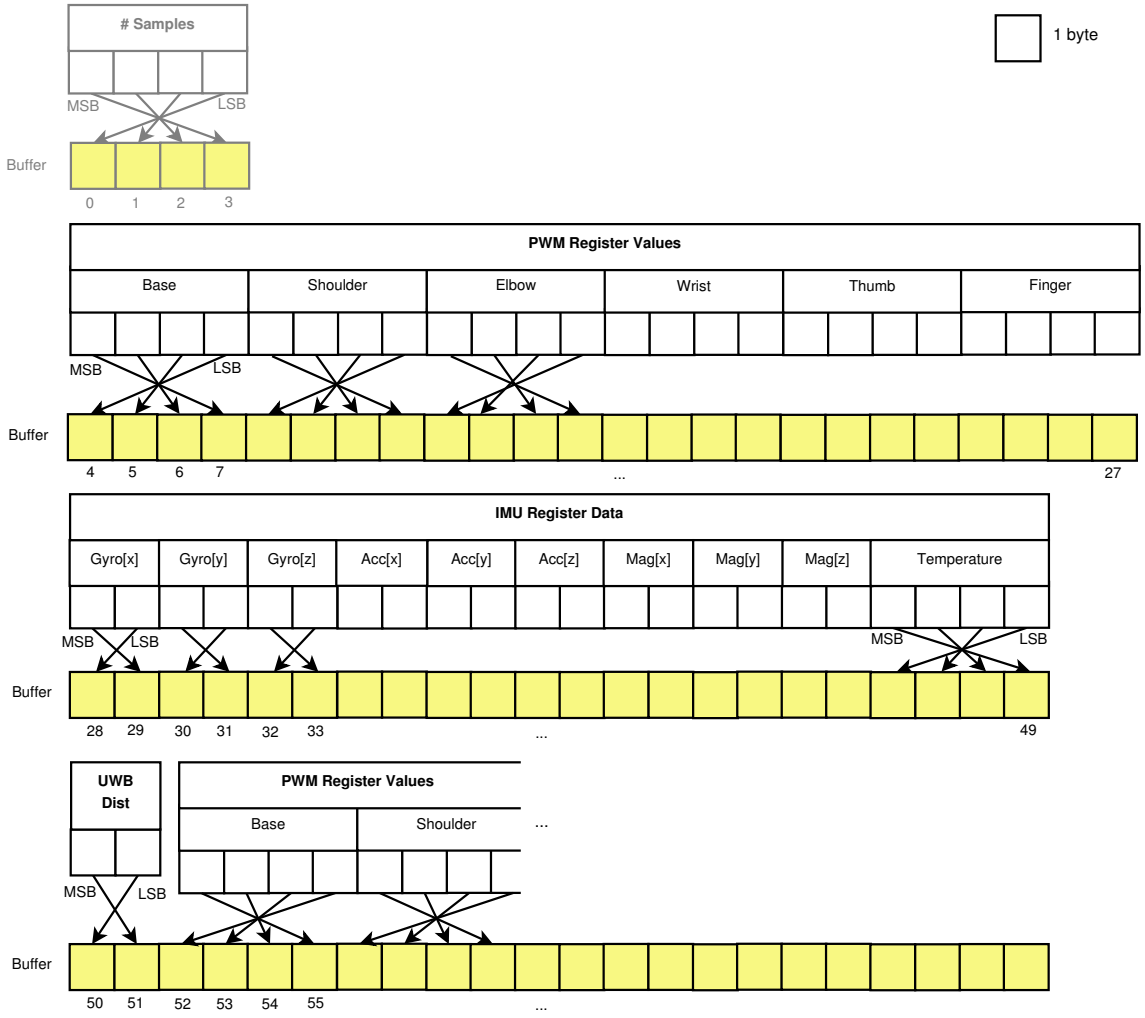


Figure 4.11: Data Storage Sequence

The XModem Protocol

The *XModem* protocol was developed by Ward Christensen in 1977 as one of the oldest file transfer protocols. In this protocol, files, or blocks of data, are divided into blocks of 128 bytes, followed by an 8-bit checksum. Files are transferred one packet at the time, followed by an ACK by the receiver, if the checksum it computed out of the received data was equal to the received checksum. Then, the next packet can be transferred. In the case of non-matching checksums, the receiver sends a NACK, which prompts the sender to re-send the last package. Thus, the XModem protocol is completely receiver-controlled.

As the 8-bit checksum is simple, it is possible to miss errors. Therefore, John Mahr developed the *XModem/CRC* protocol to be able to transmit 16-bit Cyclic Redundancy Check (CRC) sums. To keep the protocol downwards-compatible with the original *XModem* protocol, the NACK character was replaced with the ASCII character 'C'. This way, the receiver control has been preserved.

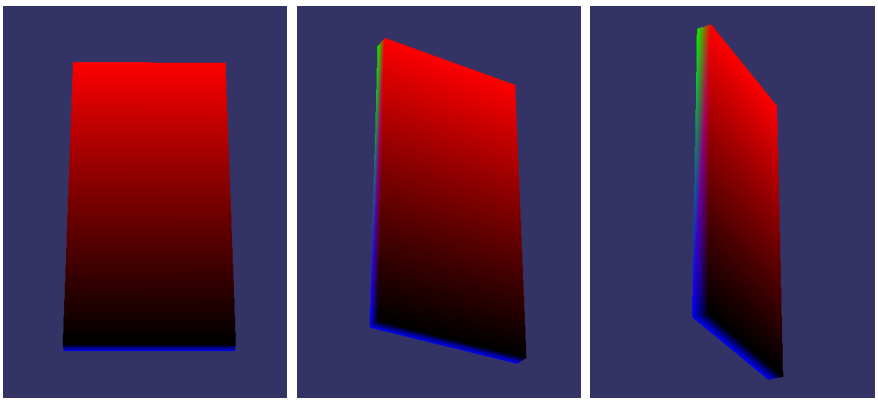
For slower connections, a block size of 128 bit may be enough. However, in order to transfer a larger amount of data, using the higher computing power and communication speed, which is available nowadays, the block size was increased to 1024 bits. Therefore, this modification of the protocol is called *XModem-1k*. It is driven by the sender and assumes, that the receiver can handle it. This is uncommon for the XModem protocol, because the communication will fail, if the receiver is not able handle it.

In order to ensure an undisturbed data transmission, in this implementation, the timer interrupts are disabled during transmission. [83]

Data Unpacking The conversion of the data, received by the PC, back to their original data types is done analogously to the storage process. At first, the amount of samples is retrieved from the binary file. Next, there is an iteration over the number of samples, where the original data types are put back together and exported into an ASCII file, which can be used for data analysis.

4.5 IMU Viewer

To visualize the orientation and position data, received over UART in real time (as described in section 4.4.1), the *IMU Viewer* has been developed. It is a *C++* program using the Open Scene Graph (OSG) framework, which is a high performance open source 3D graphics toolkit, used for visual simulation, games, virtual reality, scientific visualization and modelling. OSG is written in *C++* and *OpenGL*. Therefore, it runs on almost all operating systems.



(a) Initial (b) 30° Rotation (c) 60° Rotation

Figure 4.12: Different Rotations of the IMU Around Its Y-Axis

When the IMU Viewer program is started on the PC, it tries to open and configure an UART connection first. If no connection can be established, data is taken from a local file. This feature facilitates the illustration of data, which has been computed offline. As long as the user does not press the *Esc* key, or the end of the UART transmission or file has been detected, the program views the current (or offline computed) orientation of the IMU. The display of the position is implemented, but it has been disabled due to the inaccurate results of the position computation, which will be explained in chapter 6.

The coloring of the IMU model is supposed to facilitate an easy attitude recognition. A screenshot of the IMU Viewer in different orientations is shown in figure 4.12.

The orientation and position update of the model is implemented as follows. At first, one line is read from either the UART connection or the local file. Lines starting with the number sign (`#`) are ignored in the file mode to facilitate reading in Octave matrix files (`*.mat`). In the case of an unsuccessful read, the program exits.

Then, the line is scanned for the four quaternion values w, x, y, z and three position values, one for each axis. This is the data sequence, that the implemented application uses to send the data during *real-time* processing. The quaternion and position y-values are inverted due to the OSG coordinate system being left-handed, while all data have been computed for a right-handed coordinate system, as explained in section 2.1.1. Then, the orientation of the IMU model is updated. As mentioned before, the position update has been disabled. If it was enabled, the model would drift out of the picture in less than a second, using the data computed in the course of this work.

As soon as the end of transmission or file has been detected, or the user has pressed the `Esc` key, the program exits. [84]

5 Sensor and Actor Calibration

A good sensor is characterized by its *precision* (the ability to produce the same output for the same input) and its *resolution* (the ability to detect small changes in the measured parameter). A sensor's precision is particularly affected by (random) noise. Other important *characteristics* of a good sensor are

- *Linearity*: the output is proportional to the input, and
- *Speed*: the time it takes to read out sensor values, assuming everything else (particularly precision) is the same.

Nowadays, most available sensors work well for non-critical applications. However, for applications that require a high accuracy, they are not good enough. For example, if you use a sensor with a low accuracy on the robot arm to let it pick up small objects, inaccurate sensor readouts will cause the orientation calculation to be erroneous, which makes it impossible to fulfill the task.

The accuracy of sensors depends on the *assembly* of the particular sensor in place, because one sensor model does not only differ between two samples, but there are also differences within the sample. Additionally, the sensor is affected by heat, cold, shock and humidity during assembly, storage and shipping.

Furthermore, sensors vary depending on their *type*. Inertial sensors are highly dependent on their alignment with the overall system, whereas temperature measurements are affected by thermal gradients between the sensor and the measurement point.

Earl [85] states that accuracy is "a combination of precision, resolution and calibration". If a sensor is able to deliver repeatable measurements with a good resolution, accuracy is achieved by calibration.

In order to calibrate a sensor, a standard *reference* is required. This can either be a calibrated sensor or a standard physical reference, e.g. boiling water, which has 100°C at sea level, for temperature sensors or gravity, which is constant at 1G on the surface of the earth, for accelerometers.

Each sensor has a *characteristics curve* defining the response of the sensor to the input. Calibration maps the sensor's response to an ideal linear response. The difference between the two is called *structural error*. Different characteristics affecting the sensor's output are:

- *Offset*: The sensor's output is lower or higher than the ideal output. Particularly, *zero-level offset* describes the phenomenon of the output values not being equal to zero, even though there is no input to the sensor.
- *Sensitivity* or Slope: The output of the sensor changes at a different rate than ideally, i.e. when a known level of input is applied to one of the sensor's axes, and the output is off by a certain factor, this factor must be determined in order to calculate the scaling.
- Linearity (vs. *Drift*): If the output of the sensor is not linear, there is a drift. Sensitivity and zero-level offset change, depending on the sensor's temperature. The temperature values have to be recorded and compared to the initial value in order to apply adjustments.

To calibrate a sensor, it is placed in a condition, where the input stimuli are known. Thus, errors can be determined.

For this work, the IMU must be placed well-aligned on the robot arm. Any deviations will lead to less accurate calibration results. For example, even an orientation error of only 0.5° on one axis leads to a cross-axis sensitivity of 0.87%. If the orientation error is even 1°, the resulting cross-sensitivity is 1.75%.

Furthermore, the robot's servos have to be well-aligned to make sure that an input angle of 90° results in a real rotation of 90°.

Finally, the UWB modules should be aligned, such that the antennas are both facing up. If this is not the case, the antenna delay, and thus the distance, will vary. [37, 85, 86]

5.1 Gyroscope and Accelerometer

5.1.1 Zero-level Offset

Idea

To compute the Zero-Level Offsets (ZLOs) of the *gyroscope*, the IMU has to lay on an even surface without any external rotation affecting it. The initial gyroscope offsets of the different axes are assumed to be zero. Then, the average of a specified amount of samples is used as offset for the gyroscope to remove noise from the measurement. The ideal result is, that all values are equal to zero, except for the accelerometer axis, where the normal force is measured. This measurement should be equal to 1G due to the earth's gravitation.

The ZLOs of the *accelerometer* are determined using a *6-point tumble test*. In this test, the accelerometer is rotated into all possible orientations, such that the

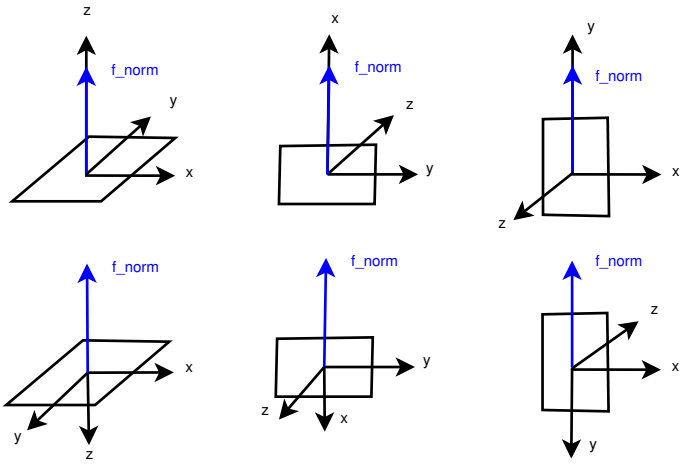


Figure 5.1: Possible Orientations for the 6-Point-Tumble-Test

normal force is either parallel to one of the axes and such that there is no external acceleration applied to the other two axes. The first axis measures the normal force. For each of the six possible orientations, which are displayed in figure 5.1, measurements are taken for the axes, which do *not* experience any external force, similarly to the gyroscope's ZLO above. For example, in the two orientations on the left side of figure 5.1, the x- and y- axis do not experience any acceleration.

In a modification of this test, reported by Lohse [25], testing can be performed without expensive equipment. The IMU is simply attached to an IKEA Kallax shelf, which is rotated into the desired orientations.

In addition, the sensors should be running for five to ten minutes to make sure the temperature of the chip is its typical operation temperature before computing the zero-level offset. [25, 87]

Implementation

Even though the *Motion Driver* contains a function to automatically calibrate gyroscope and accelerometer as stated in InvenSense Inc. [75], the author found that this function did not work. Therefore, an own calibration algorithm was developed. Note that in this case, calibration is limited to zero-level offset compensation.

The implemented zero-level offset compensation routine is shown in figure 5.2. It assumes that there is no motion or rotation affecting the IMU. Firstly, if the DMP is in use, it is disabled. Then, the gyroscope and accelerometer offset values are set to zero in the corresponding registers. Next, a set of samples, as specified by a define, is taken and discarded to make sure the sensors are running well.

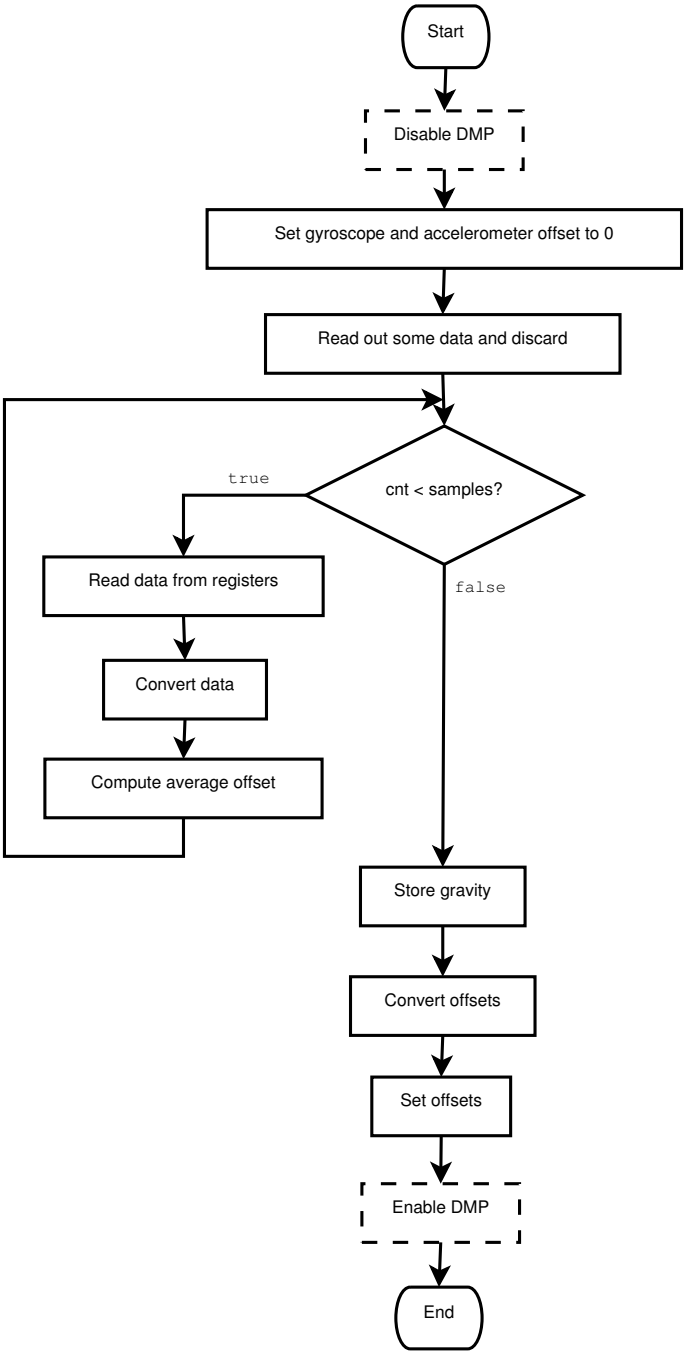


Figure 5.2: Implemented Calibration Routine

Then, the specified amount of samples is taken by reading the raw values from the IMU's registers, converting them from hardware units to $^\circ/s$ for gyroscope data, and to G for accelerometer data, and compute the average. The conversion of the raw data is accomplished using the following formula:

$$x_{conv} = \frac{x_{raw}}{s} \tag{5.1}$$

where s is the *conversion sensitivity* of the sensor, which depends on the selected Full-Scale Range (FSR). The FSR of a sensor determines the range in which values can be measured. The measured value is then scaled to the range of the corresponding data type. This scale factor is the conversion sensitivity.

The average, or arithmetic mean, which is usually calculated by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.2}$$

is computed by a formula derived from (5.2):

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n} \tag{5.3}$$

This prevents the memory consumption of storing all samples before computing the average. Instead, the average is computed using the samples as they are measured. The drawback of this way to compute the average is, that in the case of a large n, the summands are quite small. As the values are computed as floating point numbers, the value of very small numbers may not show up in the sum, due to the way floating point values are stored internally.

As the IMU is assumed to experience no external force but gravity, the measured acceleration is equal to the normal force. Therefore, the measured acceleration values are stored as the normal force vector for the position computation as described in section 2.1.2. Then, the offsets are converted back to hardware units and written into the corresponding registers of the IMU. If the DMP has been disabled, the procedure ends by re-enabling it.

Calibration Features of the DMP

If the *quaternion computation* is handled by the DMP, it is possible to push the previously computed gyroscope and accelerometer zero-level offsets to the DMP, such that it consideres them for the gyroscope integration. The initial gyroscope offsets have to be determined independently and made known to the DMP in particular registers.

Additionally, drifting gyroscope biases at later points of time can be computed by the DMP after eight seconds of no motion. This is the so-called *dynamic gyroscope calibration*. The *Motion Driver* contains a function to enable and disable dynamic gyroscope calibration during runtime.

The accelerometer bias drift is not handled by the DMP. The only case, in which the DMP handles accelerometer biases, is, when they have been determined a-priori and being made available to the DMP. In this case, they will be removed from gyroscope- and accelerometer-fused quaternion computation, if this feature is enabled. [75]

5.1.2 Sensitivity

The *gyroscope*'s sensitivity is computed by taking samples during a predefined motion. For example, a 90° rotation may be performed by the robot arm and the corresponding rotation is determined using the gyroscope measurements. By comparing the expected and the computed values, the gyroscope's sensitivity is derived.

The sensitivity of the *accelerometer* can be determined using measurements of the *6-point tumble test* by taking the acceleration of the axis, which is parallel to the normal force. For example, in the two orientations on the left side of figure 5.1 (p. 69), the z-axis experiences the normal force in the orientation at the top, and the negative normal force in the orientation at the bottom. By comparing the measured acceleration to the expected (i.e. the normal force), the sensitivity is determined.

Similar to computing the ZLO, the average of a sampling set should be taken after an appropriate warm-up time for the sensors.

5.1.3 Temperature-based Drift

The offset and sensitivity drift due to temperature changes have not been considered in the course of the work. An exhaustive and time-consuming offset and sensitivity analysis is necessary to get enough data to be able to compensate the drift.

5.2 Magnetometer

The magnetometer has not been used in the course of this work. Therefore, the magnetometer has not been calibrated.

5.3 Temperature

Usually, the temperature should be calibrated as well. This could be done using a two-point calibration of a laboratory thermometer as described in Earl [85]. In this method, the sensor has to be placed into both boiling and ice water. This requires a calibrated thermometer and adequate protection for the chip. Due to insufficient protection of the chip, it did not seem to be practical to put the IMU into water.

5.4 UWB Antenna Delay

Background In the course of this work, *antenna delay* refers to "the delay between the time an outgoing message is time-stamped inside the DW1000 and the time the signal leaves the antenna" [88]. It depends on the specific hardware design, as well as the components included in the transmit and receive paths between chip and antenna. Therefore, for each design, the specific antenna delay has to be determined and made known to the software.

If the value of the antenna delay is set too large, the communication seems to works, but the Time of Flight (TOF) is not computed. If the antenna delay is set too low, the distance may be too large. Determining the correct value is a process of trial and error, until the average of the computed distances matches the real distance, which can be determined using a measuring tape, etc. [88]

For the localization of the robot arm, the *absolute* value of the distance between the anchors and the robot arm will only be important if the robot arm is set onto a chassis and moves freely within the room. Since there was only one anchor used, when the robot is moved, the difference between the current and the inital distance is enough to determine the grippers location along the x-axis of the IMU. As long as the absolute location of the robot arm is known and only the change in distance to the anchor is important, an exact calibration of the antenna delay is not required. However, the computed distances should not be negative either.

Implementation The antenna delays can be calibrated using the switches and buttons once Zedboard and Zybo are powered up, the Zedboard's PS is configured and its software is running. On the Zybo, the second switch has to be turned on, the other ones have to be switched off. Buttons 1 and 2 de- and increase the antenna delay.

On the ZedBoard, the localization software will not be started as long as the Switch 1 is turned on. The user can de- and increase the antenna delay using the down- and centre button. During the calibration process, the computed distance can be viewed using UART. Once the calibration is done, the switch has to be turned off to start the localization.

5.5 Servo Rotation

Angle The last point of calibration are the servo motors. In order to pick up an object, it is important that the particular joint moves to the specified angle accurately. Therefore, the servos may be calibrated by computing the ZLO and sensivity.

The ZLO could be computed by setting all joint angles to 0° and measuring the real angle. The sensitivity can be determined by letting the particular joint perform a 90°-movement, measuring the real angle and comparing it to the input angle.

Rotation Speed For tweaking the speed of the servo movements, the input voltage can be adjusted. A higher voltage increases the speed while a lower voltage slows down the movements.

Implementation Due to the poor localization results, the robot has never been used to pick up objects. Therefore, the calibration of the servos has not been performed.

6 Experiments

All experiments were performed in the form of *offline processing*. This means, the raw sensor data, collected during movements, were stored on the ZedBoard. Once it was finished, all data were transferred to the PC. After unpacking the data, Octave was used for the computations and plots. An explanation of the created script files can be found in the appendix in section A.1.3 (p. 121).

6.1 Calibration

In the previous section, calibration was explained. In this section, two different IMU calibration methods, namely *static* and *dynamic* calibration, will be examined.

6.1.1 Static Calibration

In order to retrieve data for a future static calibration, the IMU was fixed on the robot arm, such that the z-axis pointed upwards and the x-axis pointed in the same direction as the grippers (as shown in figure 3.4 at page 35). Then, different data sets were collected for each of the different orientations of the IMU in time intervals of two minutes.

As the IMU was mounted to the robot arm already, it was not possible to measure data while the z-axis was pointing towards the earth. To facilitate this, the robot needs to be hanging from the ceiling, pointing down to the floor. However, for the other orientations, samples were collected and the average raw output was computed. These measurements yield the *Zero-Level Offsets (ZLOs)* of the gyroscope for almost all axes, as well as the ZLOs for the axes of the accelerometer, where there was no gravity measured. For the axis, where the normal force was measured, the *sensitivity* of the accelerometer was determined.

In order to determine the *sensitivity* of the gyroscopes, the robot was commanded to perform 90° movements. By rotating the *wrist* joint, the sensitivity of the gyroscope's y-axis could be determined. Therefore, the quaternion value for the particular axis was divided by the value expected for a 90° rotation (0.7071).

The sensitivity of the z-axis was determined by rotating the *base* joint. It was not possible to determine the sensitivity of the gyroscope's x-axis because it points in

Number	Orientation	Movement	Sensor	Parameters	Axis
1	z-axis up	90° (wrist)	G	Sensitivity	y
2	z-axis up	90° (base)	G	Sensitivity	z
3	x-axis down	no	A	Sensitivity ZLO	-x y, z
4	x-axis up	no	A	Sensitivity ZLO	x y, z
5	y-axis down	no	A	Sensitivity ZLO	-y x, z
6	y-axis up	no	A	Sensitivity ZLO	y x, z
7	z-axis up	no	A A G	Sensitivity ZLO ZLO	z x, y x, y, z

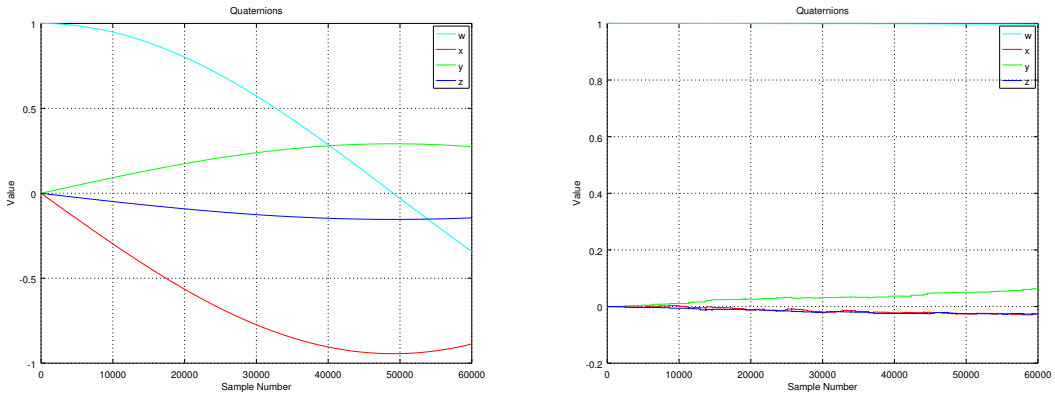
Table 6.1: Sample Sets for IMU Calibration

the direction in which all joints move. There is no configuration such that a rotation around the x-axis could be performed.

The different sample sets and their *experimental setup* are summarized in table 6.1. It also shows which parameters for which axes are determined. Table 6.2 shows the computed calibration values. The ZLOs were determined by computing the average of all samples where there was no external rotation or acceleration affecting the particular axis. Then, the raw sensor measurements *raw* were corrected using the

Sensor	Axis	$\mathbf{ZLO}_{raw}$	$\mathbf{ZLO}_{conv}$	Sensitivity
Gyroscope	x	-56	$-1.7°$	N/A
	y	17	$+0.5°$	1.0422
	z	-9	$-0.3°$	0.9851
Accelerometer	x	180	$+0.021G$	1.0008
	y	-226	$-0.027G$	1.0006
	z	-35	$-0.004G$	0.9865

Table 6.2: Results of the Static IMU Calibration



(a) Raw Data (b) Static Calibration

Figure 6.1: Quaternions (Raw Data and Static Calibration)

following formula

$$cal = \frac{1}{sens}(raw - zlo) \tag{6.1}$$

where *cal* is the corrected (calibrated) sensor value, *sens* is the sensitivity and *zlo* is the ZLO of the sensor, taken from table 6.2. For example, the corrected value for the y-axis of the gyroscope was computed by

$$y_{cal} = \frac{1}{1.0422}(y_{raw} - 17) \tag{6.2}$$

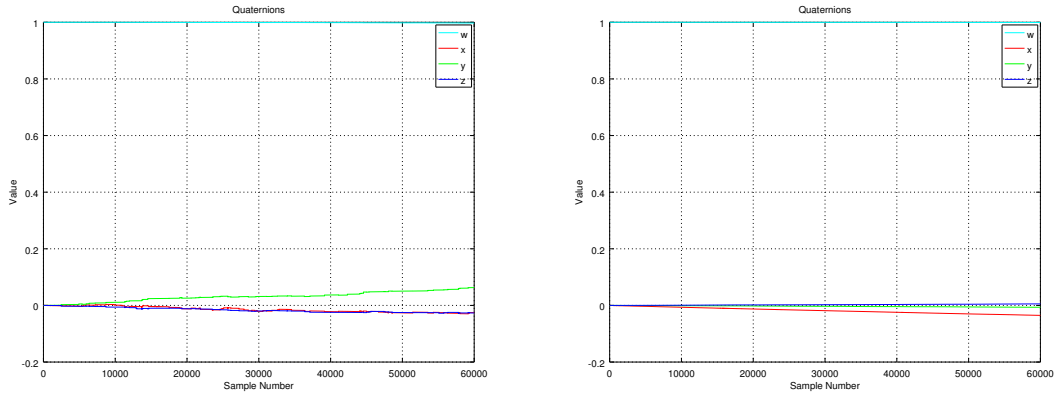
which means that the amplitudes are slightly reduced and the offset of 17 is removed.

As table 6.2 shows, the *raw* values are off quite a bit. Imagine, a position was computed using these values. Within one second, the rotation were off by the offsets displayed in the table. Without considering the incorrect rotation, by simply applying the accelerations, the IMU would have a displacement of $2.1cm$ on the x-axis, $-2.7cm$ on the y-axis and $-0.4cm$ on the z-axis. Using these erroneous measurements, the robot arm would definitely miss any object to be picked up.

Figure 6.1 shows the *effect* of static calibration on the quaternions computed using the measurement values of a new data set. The quaternions are computed from gyroscope data, which are all expected to be $(1, 0, 0, 0)$ due to absent external rotation or motion. However, the quaternions computed from uncalibrated data drift a lot (figure 6.1a). For example, at the point of time, where the w-value is zero, the computed quaternion suggests that the IMU has roughly done a 180°-rotation around the x-axis. The quaternions computed from statically calibrated data are more stable, as they drift less (figure 6.1b).

Sensor	Axis	$\mathbf{ZLO}_{raw}$	$\mathbf{ZLO}_{conv}$
Gyroscope	x	-55.67	$-1.7°$
	y	17.75	$+0.5°$
	z	-9.39	$-0.3°$
Accelerometer	x	329.59	$+0.040G$
	y	-266.53	$-0.032G$
	z	-114.4	$-0.013G$

Table 6.3: Results of the Dynamic IMU Calibration



(a) Static Calibration

(b) Dynamic Calibration

Figure 6.2: Quaternions (Static and Dynamic Calibration)

6.1.2 Dynamic Calibration

Dynamic Calibration is limited to the compensation of the ZLO, which is determined by using the average of the first 100 samples of each data set. The sensitivity of the different sensors is not taken into account. The basic assumption is that the IMU is sitting still for a small period of time before there are any movements. Assuming a sample rate of $500Hz$, this is a time interval of $\frac{1}{5}s = 200ms$.

The *results* of the dynamic calibration are summarized in table 6.3. For the gyroscope, they are equal to the results of the static calibration. For the accelerometer, they are similar, except for the z-axis, where the acceleration was $-0.004G$ compared to $-0.013G$ with dynamic calibration.

Figure 6.2 compares the results of the static and the dynamic calibration. The final quaternion after two minutes for static calibration is $(0.997, -0.025, 0.062, -0.026)$ compared to $(0.999, -0.035, -0.006, 0.005)$ for dynamic calibration. Except for the x-value, the latter seems to yield an orientation, which is closer to the real value

of $(1, 0, 0, 0)$. Other data sets confirmed this observation. Therefore, from now on, dynamic calibration is used.

6.2 Smoothing

In this section, the methods for smoothing data, explained in section 2.3.1 (p. 25), will be compared to a 1D-Kalman filter. There are two experiments:

1. No Motion for two minutes while all joint angles are 0°: The rotation is expected to be constant and represented by a quaternion of $(1, 0, 0, 0)$. The velocity and position should be 0 at all axes. This experiment is supposed to show the long-term stability of the measurements.

2. One set of 30° step-wise shoulder movements up to 90° and back, with a short break between the different movements: The quaternion should change step-wise to $(0.7071, 0, 0.7071, 0)$, which represents a 90°-rotation around the y-axis, and back to $(1, 0, 0, 0)$ for the initial orientation. The position is expected to change step-wise and reach $(0.36m, 0m, -0.36m)$ at an angle of 90°, due to the length of the solid parts ($12cm$ each), which are stated in section 3.1. This experiment is supposed to unveil the short-term accuracy of the measurements during motion.

The sampling frequency was set to $50Hz$, which is why one minute of sampling time is equivalent to $30,000$ samples. Therefore, for the first experiment, there are $60,000$ samples. In the second one, on complete movement was finished after 9750 samples, which is equivalent to approximately $20s$. For all smoothing methods, dynamically calibrated raw data was used as input. To convert it to $^\circ/s$ (gyroscope data) and to G (accelerometer data), the values have to be converted using the following formulas

$$gyr_{conv} = \frac{gyr_{raw}}{32.8} \tag{6.3}$$

$$acc_{conv} = \frac{acc_{raw}}{8,192} \tag{6.4}$$

where the divisors represent the hardware value equivalent to $1^\circ/s$ and $1G$, respectively. They depend on the Full-Scale Range (FSR), configured for each sensor, as mentioned in section 3.2.

6.2.1 Models and Parameters

Moving Average and FIR Filter For the moving average, as explained in section 2.3.1 (p. 25), and the (low-pass) FIR filter, as explained in section 2.3.1 (p. 25), a window size of 20 was used. The FIR filter's cutoff frequency was set to $0.2\frac{F_s}{2}$, where F_s is the sampling frequency of $500Hz$.

Kalman Filter The model for the Kalman filter is based on Nowotny [89], who reduced all the required math to a 1D-problem. This means, that the filtered values are computed for each axis separately. Furthermore, he assumes that there are no external forces. The following equations determine the model for the prediction:

$$y_k^- = y_{k-1} \tag{6.5}$$

$$p_k^- = p_{k-1} + q \tag{6.6}$$

and measurement update:

$$k = \frac{p_k^-}{p_k^- + r} \tag{6.7}$$

$$y_k = y_k^- + k(z_k - y_k^-) \tag{6.8}$$

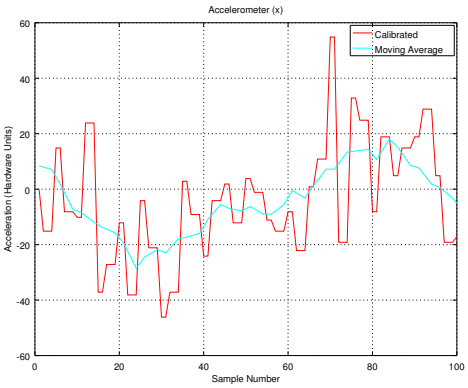
$$p_k = (1 - k)p_k^- \tag{6.9}$$

Compared to the model explained in section 2.3.2 (p. 28), the letters are all lowercase because they are simple values instead of vectors or matrices.

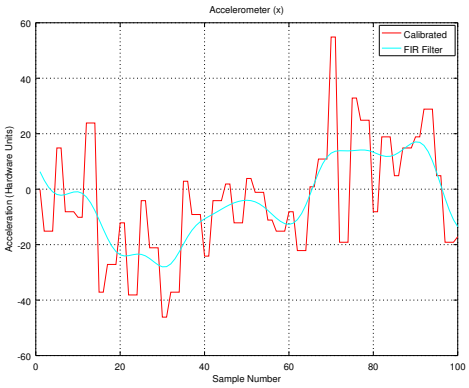
The Kalman filter's q and r values were 0.125 and 4.0 as suggested by Nowotny [89]. The value of the sensor noise q determines how much the filter trusts the sensor. The higher r is, the more stable is the resulting graph. The value of r defines the process noise. If it is set too high, the filter does not make much of a difference, compared to the original value. By decreasing it, the lag behind the measured signal increases, but the graph is less noisy. [89]

6.2.2 Results

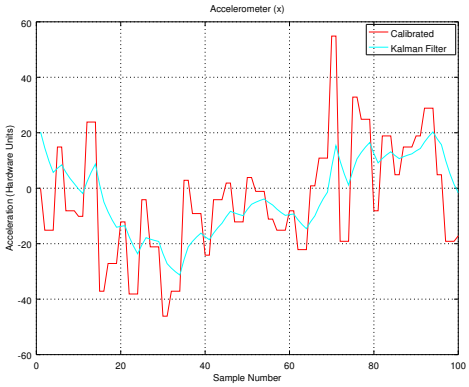
Acceleration



(a) Moving Average



(b) FIR Filter



(c) Kalman Filter

Figure 6.3: Acceleration (x-Axis, No Motion, Smoothing)

Quat.	Dyn. Cal.	Mv. Avg.	FIR	Kalman
w	0.999935	0.999934	0.999934	0.999935
x	0.00175	0.00174	0.00174	0.00177
y	0.01111	0.01115	0.01115	0.01112
z	0.0013	0.0016	0.0016	0.0017

Table 6.4: Final Quaternions With and Without Smoothing

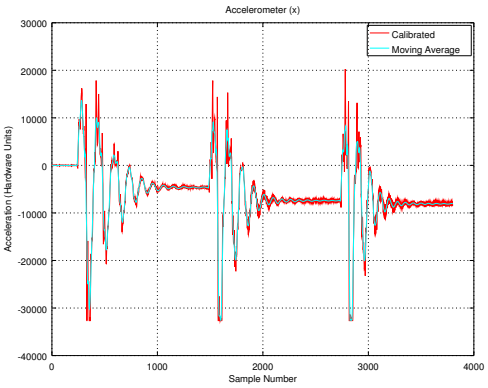
No Motion Figure 6.3 shows the *results* for the x-value of the acceleration using the different filters. The displayed data set was reduced to one-hundred samples. The first twenty samples are not shown due to the filter delay of the moving average method and the FIR filter. The original data is noisy as the values are spread around zero. The moving average (figure 6.3a) creates quite edged data. The FIR filter (figure 6.3b) seems to smooth the data the most. The Kalman filter (figure 6.3c) is a compromise between the two.

However, in terms of computation speed, the author perceived the moving average computation as the quickest, followed by the FIR filter. The calculation of the Kalman filter felt the slowest, even though no measurements of the computation time were taken.

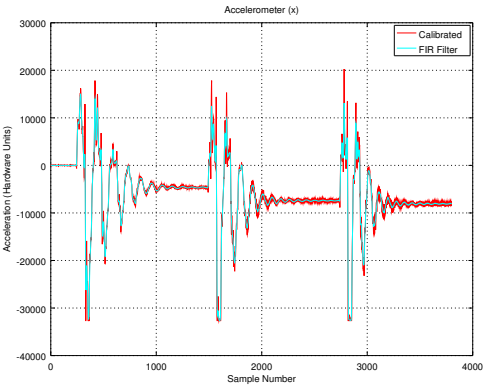
Motion There was also an experiment for motion data, where the robot performed 30° step-wise movements with the shoulder joint. For each step, an acceleration peak should be visible at the x-axis of the accelerometer, followed by a constant acceleration, which represents part of the normal force, depending on the angle.

The *results* of the different filters for a *down*-movement in three steps are shown in figure 6.4. For each step, there is an acceleration towards the positive x-axis, which is oscillating massively until it gets closer to the final value of this step. Depending on the current angle, it is equivalent to a fraction of -8192 hardware units ($-1G$). There is a negative acceleration, corresponding to the normal force, because the x-axis is pointing towards the floor, while the normal points towards the ceiling. The oscillation is due to the assembly of the robot. The solid parts are quite thin. Furthermore, they are only connected to each other by the servo motors. The *shoulder* joint is the second-lowest joint after the *base* joint and it carries a lot of weight. This reinforces the observed oscillations.

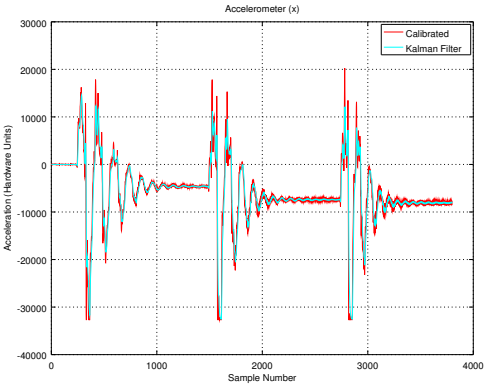
Rotation



(a) Moving Average

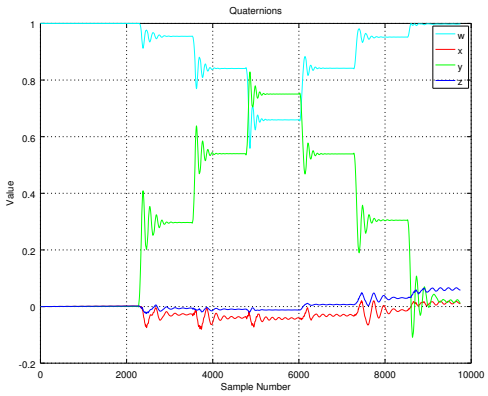


(b) FIR Filter

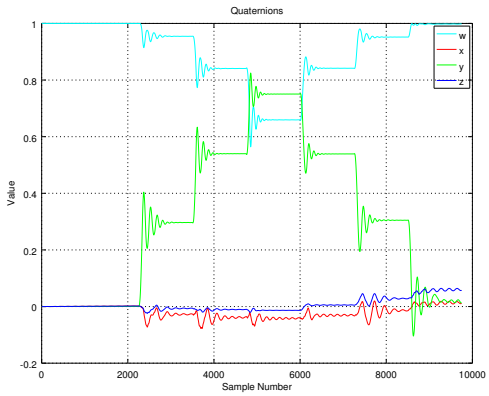


(c) Kalman Filter

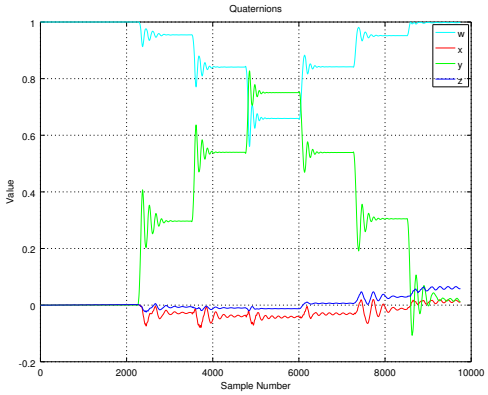
Figure 6.4: Acceleration (x-Axis, 30° Step-wise Motion, Smoothing)



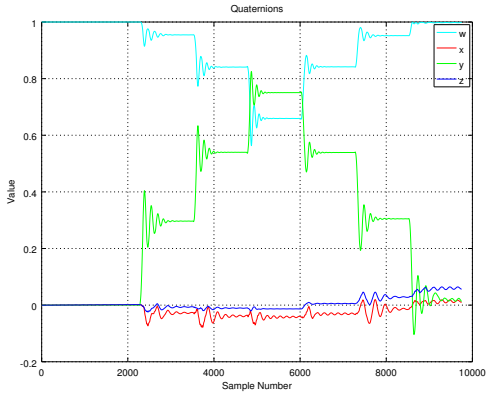
(a) Dynamic Calibration



(b) Moving Average



(c) FIR Filter

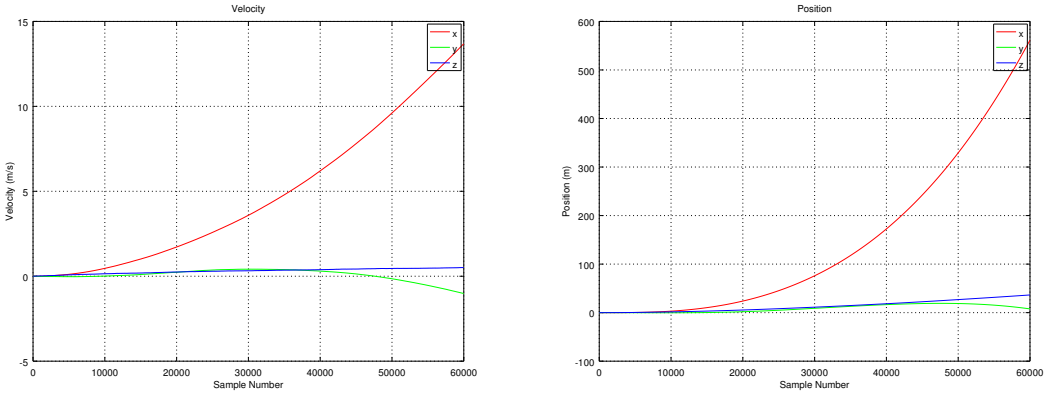


(d) Kalman Filter

Figure 6.5: Quaternions (30° Shoulder Movements, Smoothing)

No Motion When computing the rotation for the experiment without motion, the resulting *quaternions* look similar to the ones without smoothing but with dynamic calibration, as shown in figure 6.1b (p. 77). The final quaternion for each method after two minutes of no motion is shown in table 6.4. They are all pretty similar, but keeping in mind that the real value is $(1, 0, 0, 0)$, the dynamically calibrated value without smoothing is the best, except for the x-value, where the moving average and FIR filter values are slightly better.

Motion If the *quaternions* are computed for 30° step-wise movements, the different smoothing methods do not seem to improve the result. Figure 6.5 shows the quaternions for the different methods. The different angles can be distinguished easily. However, there is a small drift in orientation, which causes the w-value to not return to 1, when moving back to 0°, the x- and z-values not returning to 0. Especially the z-value drifts significantly, which is equivalent to a rotation around



(a) Velocity (b) Position

Figure 6.6: Velocity and Position (No Motion, Dynamic Calibration)

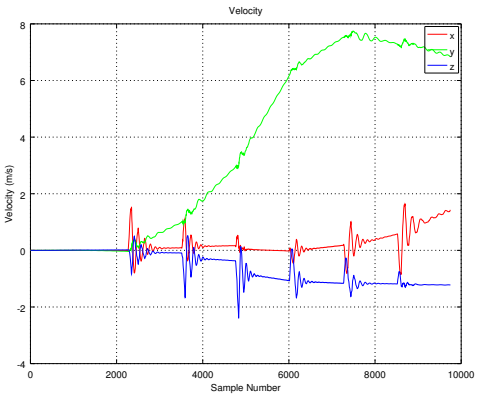
this axis. Furthermore, the w and y values do not exactly equal 0.7071, which is the expected value at the angle of 90°.

Velocity and Position

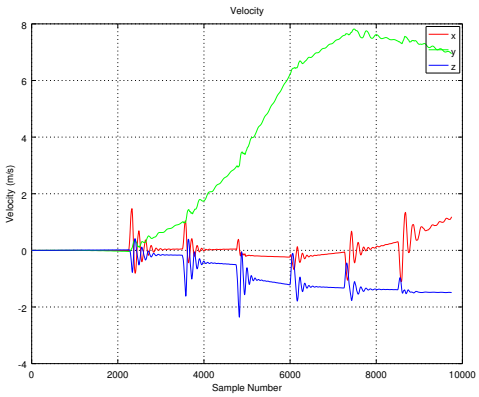
No Motion No matter of the type of smoothing or whether the data has been smoothed at all, the velocity and position are very similar. The resulting plots for the velocity and position using dynamic calibration are shown in figure 6.6. The velocity at the x-axis increases drastically, which is probably caused by the normal force not being removed properly due to rotation drift. Consequently, the final displacement of $562m$ is quite large.

Motion During motion, the observed velocity differs widely from the expected one. For the x-value, there should be a three impulses for the *down*-movement and three for the *up*-movement, corresponding to the 30°-steps. The amplitude of the second impulse is expected to be smaller due to the angle. The last velocity impulse should be the smallest, as the travelled distance is the smallest. The impulses of the z-velocity should be pointing to the opposite direction as the velocity in x-direction, while the pulses are expected to increase in size. On the y-axis there should not be any velocity at all. This expected behaviour is caused by distribution of the overall velocity over all three axes. The amount each axis gets depends on the tilt angle of the IMU.

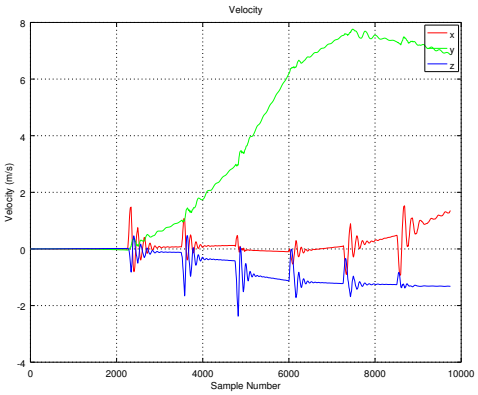
Figure 6.7 shows the computed velocity for the different smoothing methods and with dynamic calibration only. In all graphs, it is possible to observe the expected velocity changes on the x-and z-values. However, there is a large drift observed at



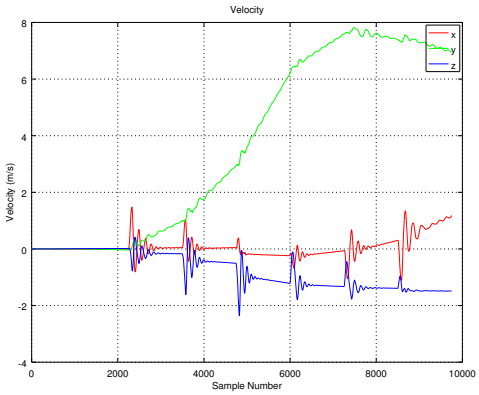
(a) Dynamic Calibration



(b) Moving Average

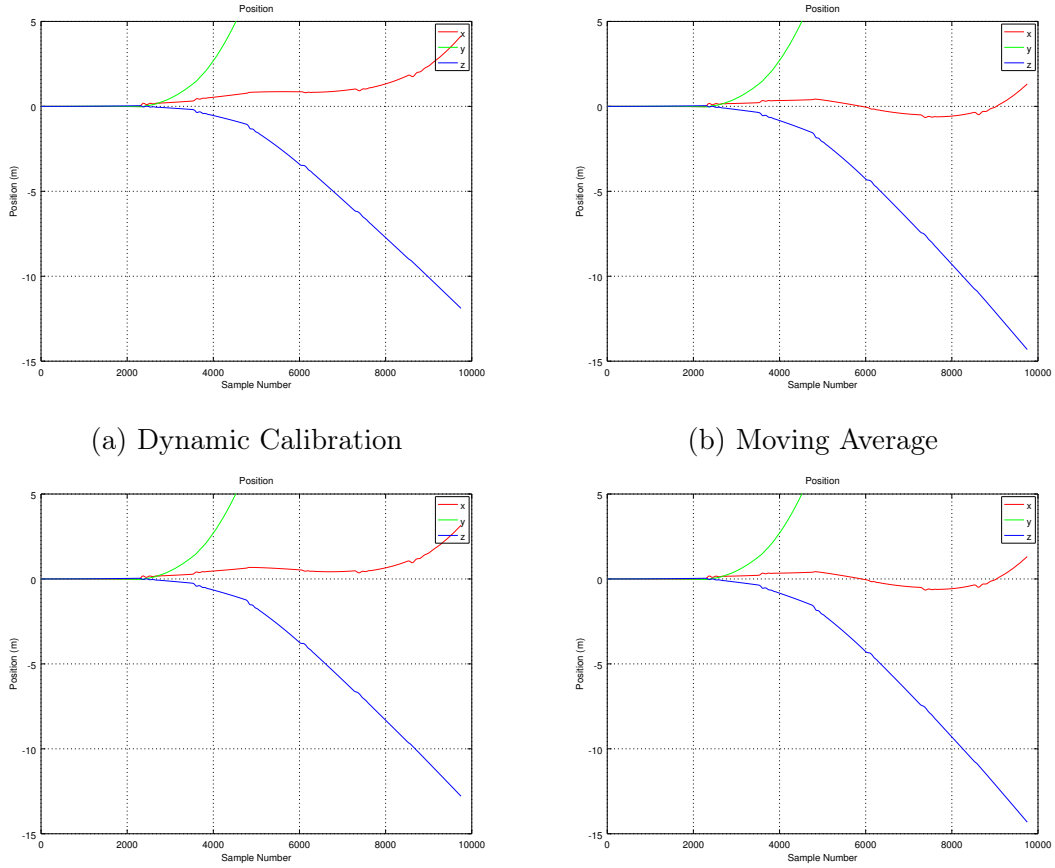


(c) FIR Filter



(d) Kalman Filter

Figure 6.7: Velocity (30° Step-wise Motion, Smoothing)



(a) Dynamic Calibration

(b) Moving Average

(c) FIR Filter

(d) Kalman Filter

Figure 6.8: Position (30° Step-wise Motion, Smoothing)

the y-axis, no matter which filter has been used. One cause for this could be the vibrations of the robot arm.

The resulting position is shown in figure 6.8. The graphs show, that the final position of the dynamically calibrated data (figure 6.8a) and the data retrieved from the FIR filter (figure 6.8c) at the x-axis are closer to the real value of zero, than the values of the other two filters. However, the z-position retrieved from the moving average (figure 6.8b) and the Kalman filter (figure 6.8d) suffer from less drift, than the ones of the other filters. The position at the y-axis drifts most, no matter, which filter has been used. The final displacement after one movement using dynamic calibration was equal to $72m$.

6.3 Sensor Fusion

The previous section showed that there is a drift in orientation, resulting in a drift in velocity and position. To compensate this drift, sensor fusion can be applied.

In this section, the two different sites for sensor fusion, as explained in section 2.3.2 (p. 26), will be evaluated using experiments. At first, within the IMU, gyroscope and accelerometer data is fused to retrieve an orientation without drift. The other fusion is the one of IMU and UWB to remove position drift. For both, complementary and Kalman filters are used.

6.3.1 Gyroscope and Accelerometer (Orientation)

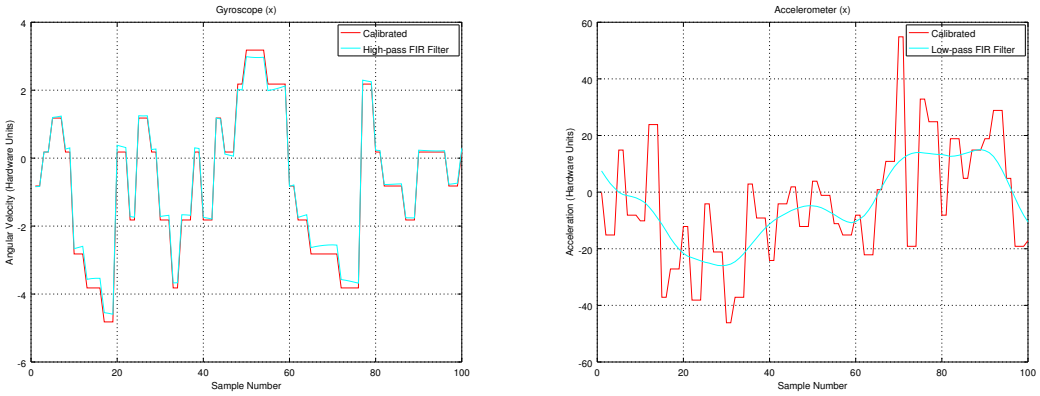
To reduce orientation drift, two different kinds of filters for fusing gyroscope and accelerometer data were used. They will be explained subsequently. The experiments are the same as in the previous section (one sample set without any motion and one with 30° step-wise movements down to 90° and back to 0°).

Complementary Filter

In this section, the application of the complementary filter, as explained in section 2.3.2, is elaborated. Firstly, the required filtering of the raw, dynamically calibrated data before obtaining rotation angles is explained. Then, the filter setup is described and the results are explained. To verify that any changes in the accuracy are due to the filter and not caused by pre-processing, the results of the complementary are shown with and without pre-processing.

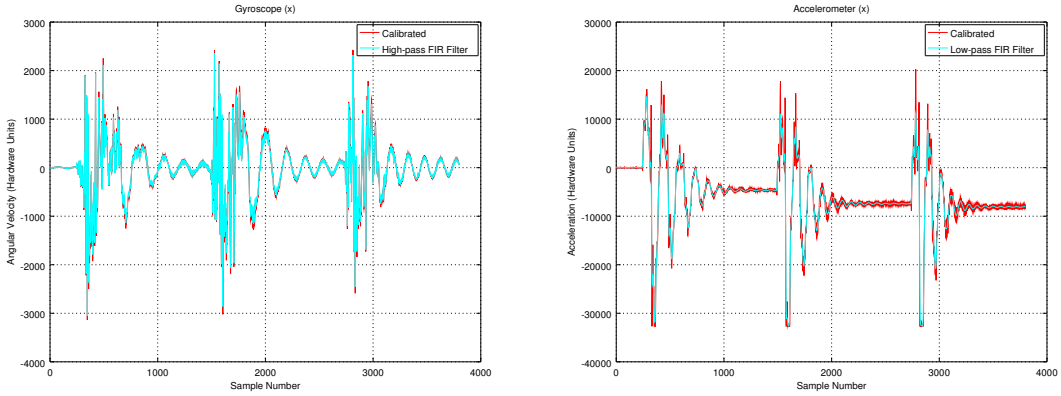
Pre-processing According to the filter specification, stated in section 2.3.2 (p. 27), before computing the angles using gyroscope and accelerometer, the dynamically calibrated data is filtered. A *high-pass filter* applied to the gyroscope data is supposed to remove any drift. To the accelerometer data, a *low-pass filter* is applied to remove sensor noise. A cutoff-frequency of $0.01\frac{F_s}{2}$ was chosen for the high-pass filter, while the cutoff-frequency for the low-pass filter was set to $0.001\frac{F_s}{2}$, with F_s being the sampling frequency of $500Hz$. More details on the filter setup in Octave can be found in section C.1.1 in the appendix (p. 135).

The resulting gyroscope and accelerometer data for the x-axis *without motion* are shown in figure 6.9. The results of filtering data *during motion* are shown in figure 6.10. Due to the low coefficient for the high-pass filter, the gyroscope data does not change much. By experiment, it was observed that a too low cutoff-frequency removed too much rotation information in form of smaller amplitudes. Compared



(a) Gyroscope (High-pass Filter) (b) Accelerometer (Low-pass Filter)

Figure 6.9: Pre-processing (x-Axis, No Motion)



(a) Gyroscope (High-pass Filter) (b) Accelerometer (Low-pass Filter)

Figure 6.10: Pre-processing (x-Axis, 30° Step-wise Motion)

Quat.	Gyr. Only	Acc. Only	Fused (Prep.)	Fused (No Prep.)
w	0.9999351	0.9999999	0.9999987	0.9999994
x	0.0017722	−0.0003046	−0.0007914	0.0000163
y	0.0111173	0.0003218	0.0006181	−0.0000255
z	0.0017120	-3.8×10^{-22}	0.0000224	0.0000189

Table 6.5: Final Quaternions (No Motion, Complementary Filter)

to the gyroscope data, the accelerometer data was filtered much more, which leads to a smoother curve with a smaller amplitude.

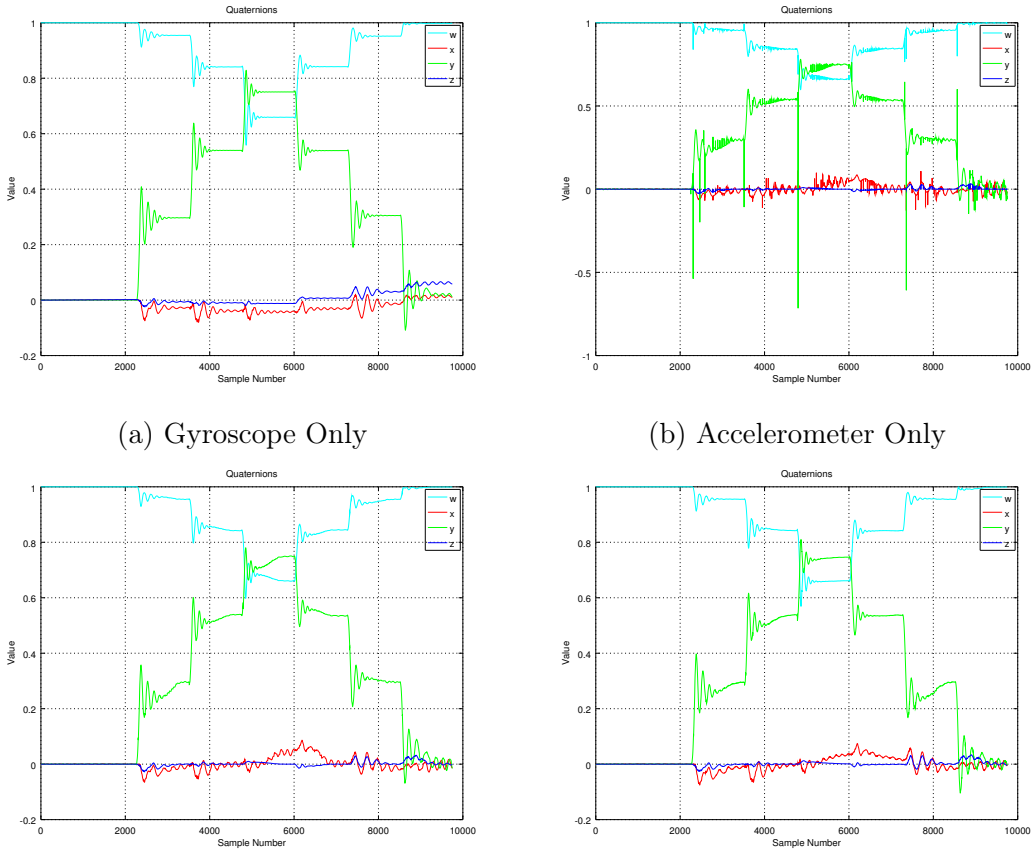
Filter Setup In order to fine-tune the complementary filter, there are two parameters. Firstly, the *gyroscope weight* determines how much of the angle computed from gyroscope data goes into the fused angle. It was set to 0.98. Consequently, the weight of the accelerometer data is 0.02.

The second parameter relates to the issue, that the tilt angle can only be computed reliably, if there is no external force present, except for the normal force. During motion, this is not the case, as external forces cause physical acceleration. Therefore, an *acceleration range* was defined, which represents the range around the normal force, which is taken into account for computing the tilt angle. All acceleration values below or above the lower and higher limit are not used for orientation computation. In such cases, only gyroscope data is used. The acceleration range was set to $0.01G$.

Results

Rotation The effect of the complementary filter on the orientation is summarized in table 6.5. It shows the orientation after two minutes *without motion*, computed from gyroscope and accelerometer data only, as well as from fused data. The final quaternion computed from accelerometer data only creates the best w- and z-value. The fused value is better than the one computed from gyroscope data only. These results make sense, as the accelerometer delivers stable measurements, if there is no physical acceleration applied to the system. The sensor fusion does its job as expected by removing some drift. Unexpectedly, the final quaternion without pre-processing yields slightly better results than with filtering. Again, as the accelerometer alone leads to the most accurate results, and filtering decreases its accuracy, the final accuracy after pre-processing is lower than without it.

The computed quaternions for data *during motion* are shown in figure 6.11. The graph of the quaternions in figure 6.11a, which has been retrieved from gyroscope



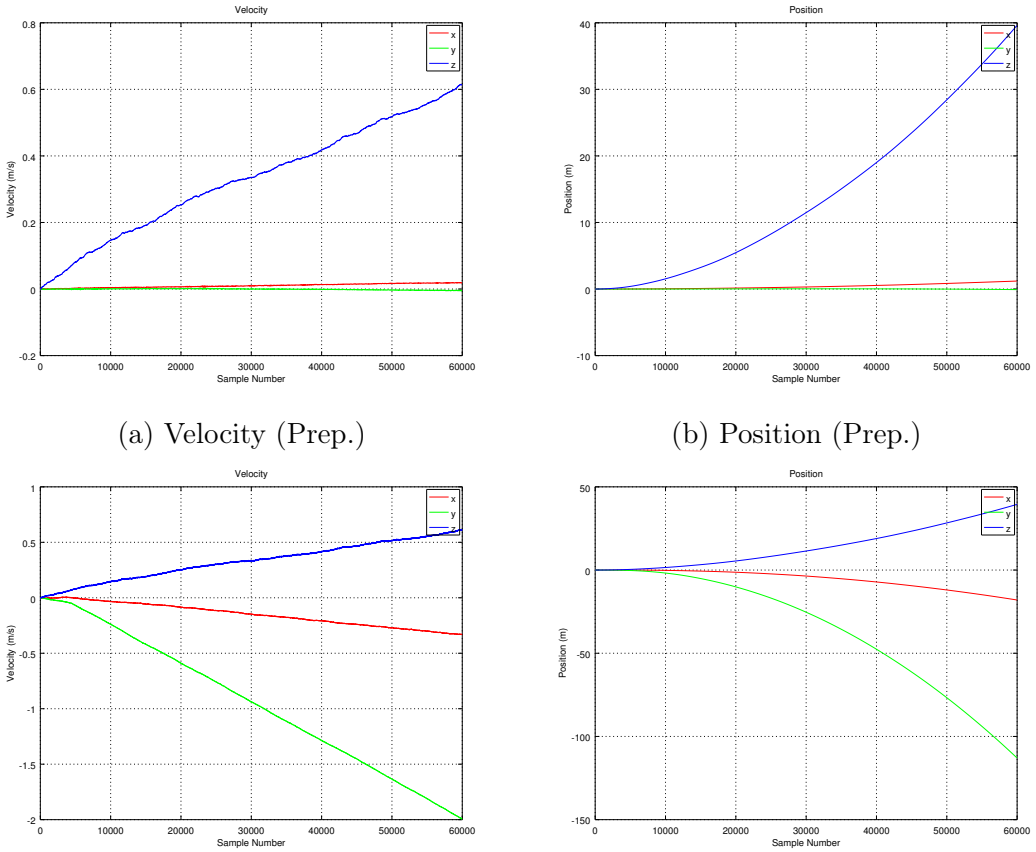
(a) Gyroscope Only

(b) Accelerometer Only

(c) Fused (Prep.)

(d) Fused (No Prep.)

Figure 6.11: Quaternions (30° Step-wise Motion, Complementary Filter)



(a) Velocity (Prep.)

(b) Position (Prep.)

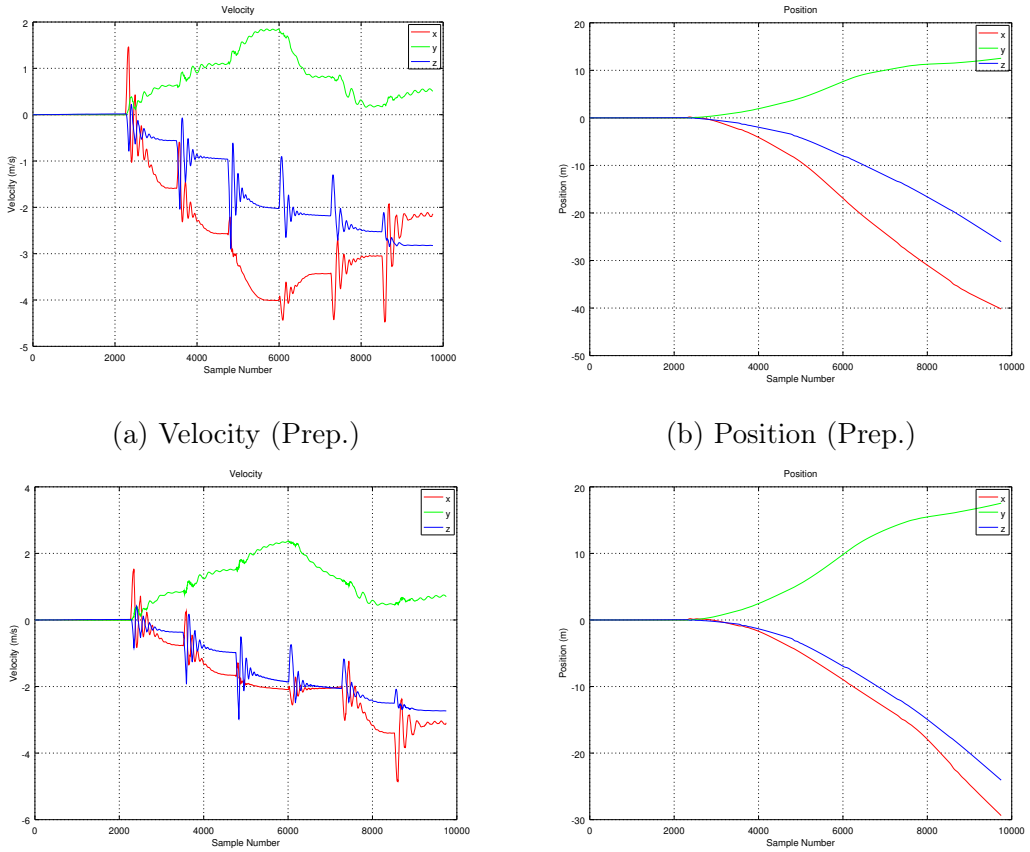
(c) Velocity (No Prep.)

(d) Position (No Prep.)

Figure 6.12: Velocity and Position (No Motion, Complementary Filter)

data only, is the same as shown in figure 6.5. The quaternions computed from accelerometer data, as shown in figure 6.11b, look quite different. There are several peaks in the w- and y-values just before a new movement is initiated. This is due to the acceleration applied to the robot arm when the servo starts moving. Unfortunately, this causes the quaternion computation to be erroneous. The fused quaternions, as shown in figure 6.11c wipe out the faults of both. They do not suffer from drift or large peaks before movements. However, they are a little less edged and do not show as nice step-wise movements as the quaternions computed from gyroscope data only. It is important to remember, that if the overall acceleration is not within the allowed range, only gyroscope data has been used. However, the quaternions retrieved from the filter without pre-processing (figure 6.11d) show more accurate step-wise movements than the ones with pre-processing.

Velocity and Position The resulting velocity and position, computed from fused data *without motion*, with and without pre-processing, are shown in figure 6.12.



(a) Velocity (Prep.)

(b) Position (Prep.)

(c) Velocity (No Prep.)

(d) Position (No Prep.)

Figure 6.13: Velocity and Position (30° Step-wise Motion, Complementary Filter)

Compared to the velocity computed from dynamically calibrated data only (figure 6.6a, p. 85), there is a huge improvement. The velocities measured at the x and y-axis are quite stable. Only the velocity at the z-axis drifts off. However, the overall drift is way less than without sensor fusion.

This also affects the computed position, where the overall displacement from the origin without sensor fusion was approximately $562m$, while the overall displacement with sensor fusion is $39m$.

The resulting velocity and position without pre-processing at the z-axis is similar to the one with pre-processing. The velocity and position at the x- and y-axis drift much more. The overall displacement is $120m$, which is worse than with pre-processing, but better than without sensor fusion.

The velocity and position computed from fused data *during motion* are shown in figure 6.13. Compared to dynamically calibrated data without sensor fusion (figure 6.7, p. 86), the velocity at the y-axis drifts less than the velocity at the other

axes. While the velocity retrieved without the complementary filter was at least partially useful, it seems that using the complementary filter blurs the velocity instead of improving it. However, the position drifts less than without using the filter. The final displacement after one movement is equal to $49m$, compared to a displacement of $72m$ without the filter. However, the data during the movement itself seems to be less useful.

Compared to the data with pre-processing, the graphs for the complementary filter without pre-processing look more stable. The final displacement after one movement is equal to $41m$, which is less than with pre-processing.

Overall, the filter with pre-processing performs better if there is no motion, while the filter without pre-processing performs better during motion. Both filters improve the result compared to dynamic calibration only.

Kalman Filter

Now, the Kalman filter is applied to the same dynamically calibrated data in order to get the *rotation angles* for each axis. After the explanation of the model, the results will be explained. The phases of the Kalman filter, including the required math, are described in section 2.3.2 (p. 28).

Model For this implementation of the Kalman filter, the approach of Lauszus [63] for balancing a robot was taken. He assumes that the system state y of each axis is determined by the current angle Θ and the amount of gyroscope bias (drift) ω_b:

$$y = \begin{bmatrix} \Theta \\ \omega_b \end{bmatrix} \tag{6.10}$$

The model A, predicting the new state from the previous one, is set to

$$A = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \tag{6.11}$$

while the model B, that predicts what changes, is set to

$$B = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \tag{6.12}$$

The negative Δt in A makes sense, because it will reduce the computed angle by the angle bias. If it was positive, the angle drift was added to the angle, additionally to the measured angular rate. The variable u, which is usually the command input, is set to the angular rate ω, measured by the gyroscope:

$$u = \omega \tag{6.13}$$

As suggested by Lauszus [63], the error covariance matrix Q is defined as

$$Q = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \tag{6.14}$$

where σ^2 is the variance, which is computed for the angle and the angular rate over all samples, respectively. The model H of how the sensor readings reflect the system state is set to:

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{6.15}$$

Next, the covariance of error noise R is set to

$$R = \begin{bmatrix} \sigma_\theta & 0 \\ 0 & \sigma_\omega \end{bmatrix} \tag{6.16}$$

with σ being the standard deviation, as suggested by Grovers [62]. The measurement z is simply the current angle θ measured by the accelerometer:

$$z = \theta \tag{6.17}$$

The filter is applied to each axis separately. At the end, the computed angles for the rotation around each axis are converted to a quaternion. This is done by computing three separate quaternions for the rotation around each axis, where each quaternion is computed by the following equations

$$w = \cos\left(\frac{\theta}{2}\right) \tag{6.18}$$

$$x = \vec{u}_x \sin\left(\frac{\theta}{2}\right) \tag{6.19}$$

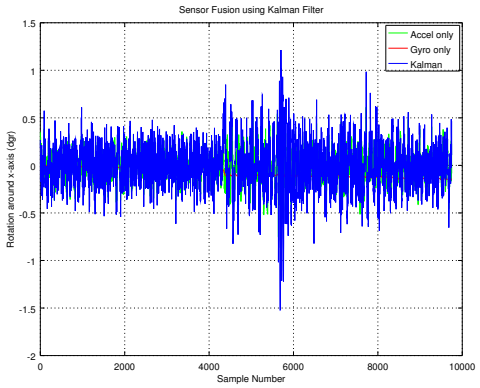
$$y = \vec{u}_y \sin\left(\frac{\theta}{2}\right) \tag{6.20}$$

$$z = \vec{u}_z \sin\left(\frac{\theta}{2}\right) \tag{6.21}$$

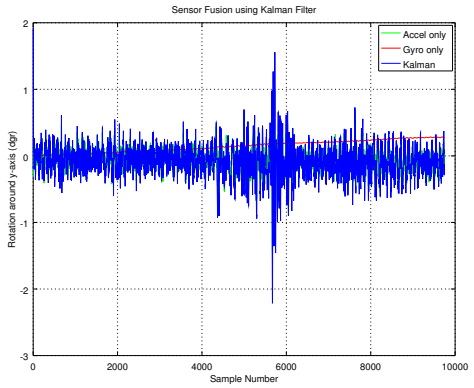
where θ is the previously computed angle around the particular axis in radians, and u_x, u_y and u_z are the x-, y- and z-values of the unit vector of the rotation axis. The absolute rotation is calculated by multiplying the three partial rotation quaternions, such that

$$q_{abs} = q_x q_y q_z \tag{6.22}$$

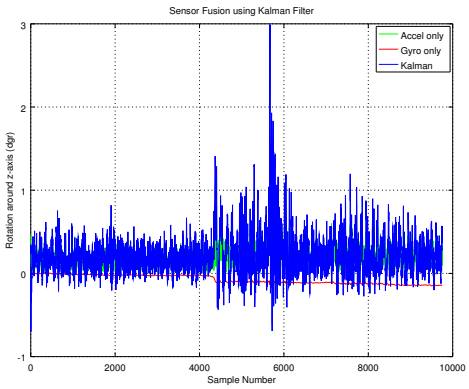
where q_x, q_y and q_z are the quaternions representing the partial rotations around the different axes.



(a) x-Axis

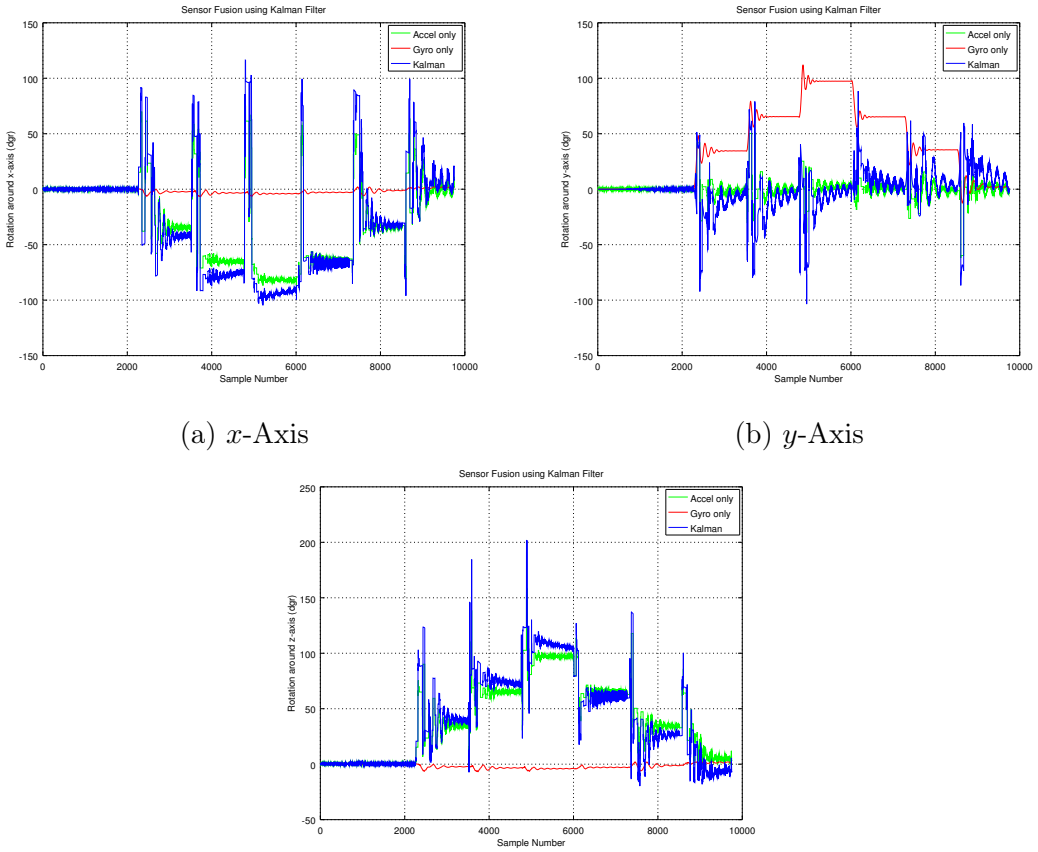


(b) y-Axis



(c) z-Axis

Figure 6.14: Noisy Rotation (No Motion, Kalman Filter)



(a) x-Axis

(b) y-Axis

(c) z-Axis

Figure 6.15: Rotation (30° Step-wise Motion, Kalman Filter)

Results

Rotation Angles The obtained rotation angles around each axis *without movement* are shown in figure 6.14. Overall, the filter takes the accelerometer data and amplifies them. The filter uses mainly accelerometer data, which is confirmed by the final K value. It is greater than 0.5 on all axes. All final P and K values are summarized in section C.1.2 in the appendix (p. 135).

The obtained rotation angles around each axis *during motion* are shown in figure 6.15. Again, the angle retrieved from the Kalman filter behaves pretty much like the angle computed from accelerometer data only. However, in these graphs, the manner of functioning of the filter with the specified model can be observed. The filter reduces the previous angle by the angles bias and adds the new rotation to it. Therefore, the retrieved angle is mostly below the angles computed from accelerometer data when the angle computed from gyroscope data is increasing (first 6000

Quat.	Gyr. Only	Comp. (Prep.)	Comp. (No Prep.)	Kal. Filt.
w	0.9999351	0.9999987	0.9999994	0.9999994
x	0.0017722	−0.0007914	0.0000163	0.0009830
y	0.0111173	0.0006181	−0.0000255	0.0000680
z	0.001712	0.0000224	0.0000680	0.0004733

Table 6.6: Final Quaternions (No Motion, Kalman Filter)

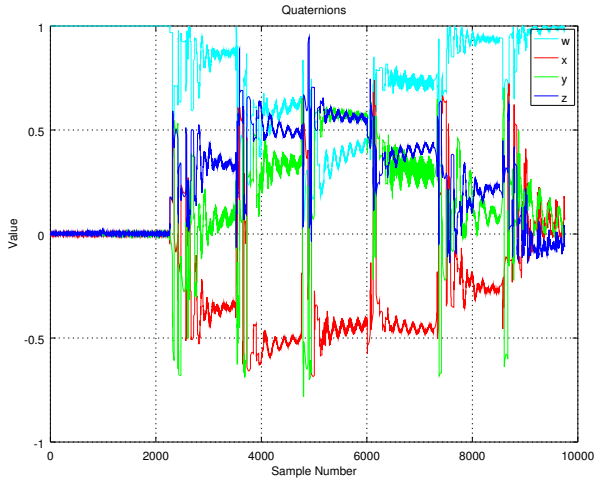


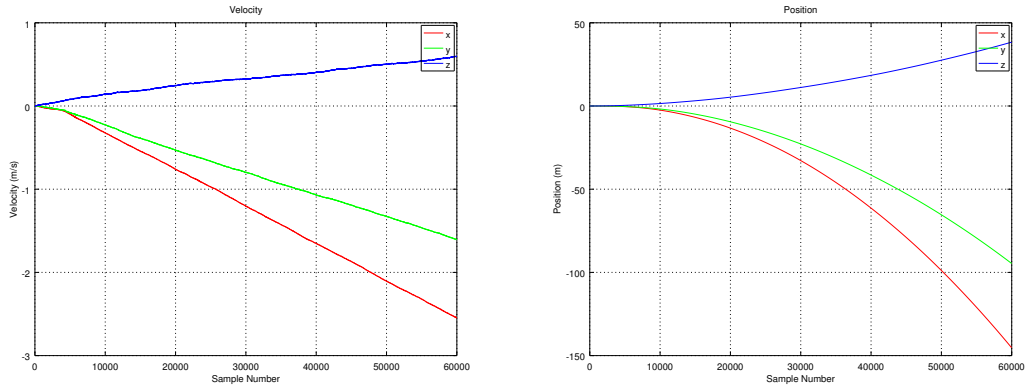
Figure 6.16: Orientation (30° Step-wise Motion, Kalman Filter)

samples in figure 6.15b). It is mostly above, if the angle calculated from gyroscope data is decreasing (last 4000 samples in figure 6.15b). Overall, it is further away from zero than the angle retrieved from accelerometer data.

Orientation Table 6.6 compares the final quaternions of the two filters after two minutes *without motion* to the quaternion computed by gyroscope data only. Compared to the final quaternion retrieved from the complementary filter with pre-processing, the Kalman filter yields better results for the w- and y-value. In comparison to the complementary filter without pre-processing, it yields slightly worse results. Overall, both filters improve the result compared to no sensor fusion.

Figure 6.16 shows the obtained orientation during the 30° step-wise *movements*. As expected, due to the incorrect rotation angles, the resulting quaternions are incorrect. While the w- and y-values are not too bad, the x- and z-values are incorrect. Interestingly, they are opposed to each other.

This behaviour is caused by the inability to distinguish physical acceleration from



(a) Velocity (b) Position

Figure 6.17: Velocity and Position (No Motion, Kalman Filter)

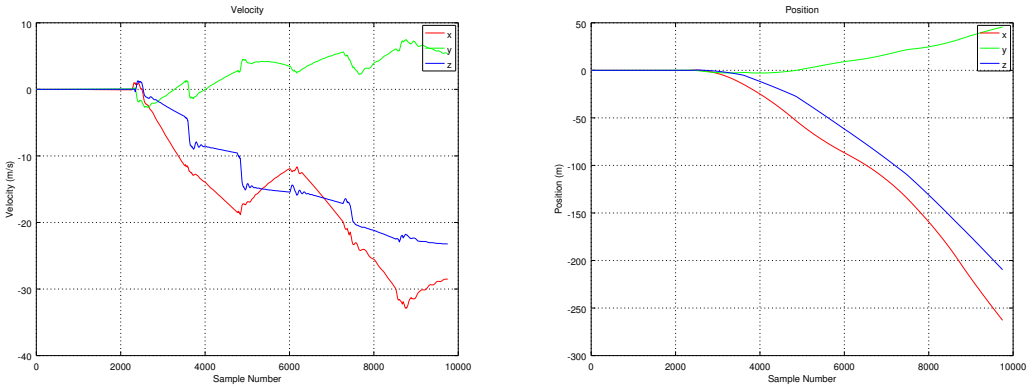
the normal force. The physical acceleration when the robot arm starts to move is much higher than the normal force, which results in these erroneous rotation angles. The Kalman filter does not recognize them as erroneous, which is surprising, as in the literature, this is stated as one of the strengths of the Kalman filter. Even changing the values in the matrices Q and R did not solve the issue.

Velocity and Position The resulting velocity and position *without motion* are shown in figure 6.17. The final velocity values are further away from zero than the ones from the complementary filter (figure 6.12a and 6.12c, p. 92), especially at the x- and z-axis. The final displacement of $177m$ is better than the original displacement of $562m$ without data fusion (figure 6.8a, p. 87), but worse than the displacements of $39m$ and $120m$ using a complementary filter (figure 6.12b and 6.12d, p. 92).

Figure 6.18 shows the velocity and position *during motion*, computed using the orientation from the Kalman filter. Due to the erroneous orientation, it is no surprise that the velocity values are drifting off rapidly. The peaks at the beginning of each motion step are visible, but less significant than the drift. Consequently, after one complete motion, the displacement of $339m$ is far more off than the displacements of $49m/41m$ and $72m$ using the complementary filter with or without pre-processing (figure 6.13b and 6.13d, p. 93) or no filter (figure 6.8a, p. 87).

6.3.2 IMU and UWB (Position)

Due to the use of *two* UWB modules, it is *not* possible to determine the absolute 3D-position of the robot's grippers within the room. However, the distance between the



(a) Velocity (b) Position

Figure 6.18: Velocity and Position (30° Step-wise Motion, Kalman Filter)

UWB module fixed at the robot and the anchor can be computed. Due to the alignment of the robot arm to the UWB anchor, this allows to compute the displacement of the robot arm at the x-axis by subtracting the average of the first one-hundred sampled distances, where the robot arm is not moved, from the measured distance of each sample. As the drift on the x-axis was the smallest for dynamically calibrated IMU data without sensor fusion, it was used as input for the sensor fusion with UWB data.

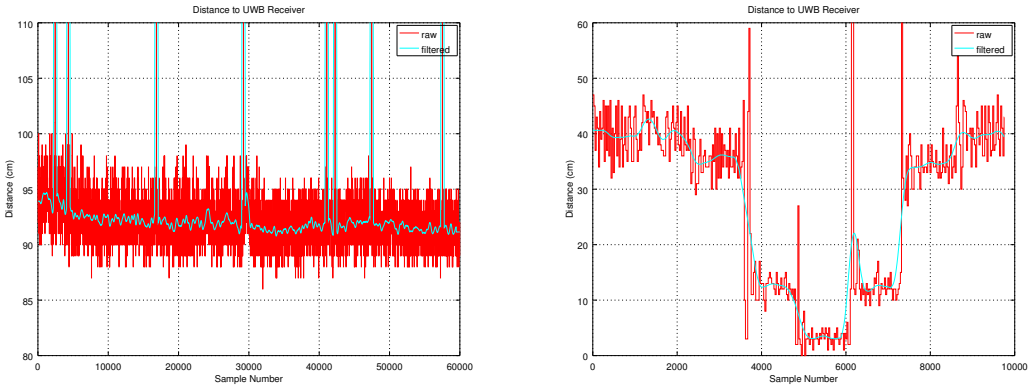
The raw data and time periods are the same than in the sections above.

Data Preparation

Filtering Before sensor fusion is applied, the measured UWB distance is filtered and converted into the displacement of the robot arm at the x-axis. The applied low-pass filter is a FIR filter with a window size of 500 and a cutoff frequency of $0.002\frac{F_s}{2}$. The first and last 250 values were discarded due to the filter setup time. The same samples of the IMU were discarded as well.

The resulting values with and without motion are displayed in figure 6.19. The absolute distance between the two data sets differs due to different antenna delays. The distance measured with a tape was $90cm$ in the experiment without motion and $60cm$ in the experiment during motion. Furthermore, there are some outliers, especially without motion (figure 6.19a). For the experiment with movements (figure 6.19b), they are not as high and thus filtered out well.

Conversion to Position Now, the computed distance has to be converted to a position in the IMU's coordinate system. Therefore, the average of the first one-



(a) No Motion (b) 30° Step-wise Motion

Figure 6.19: UWB Distance (Smoothing)

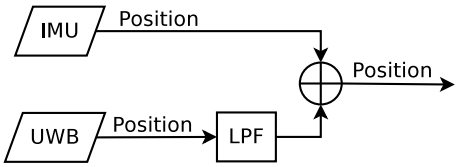


Figure 6.20: Complementary Filter to Fuse IMU and UWB Measurements

hundred distances is used as a baseline, which is subtracted from all measured distances.

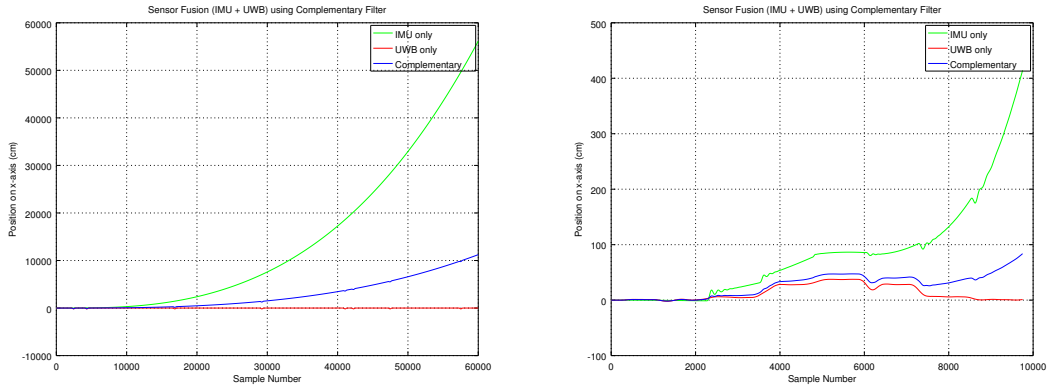
Complementary Filter

The complementary filter fuses the *position* obtained from IMU and UWB measurements. Its setup is shown in figure 6.20.

Filter Setup For the complementary filter, the position obtained from the IMU was given a weight of 0.2, which yielded the best possible results for this experiment. The data was fused by computing

$$position = weight_{IMU} * position_{IMU} + (1 - weight_{IMU}) * position_{UWB} \quad (6.23)$$

There was no high-pass filter applied to the IMU data as the obtained graph was a smooth line. A filter would not make much of a difference.



(a) No Motion (b) 30° Step-wise Motion

Figure 6.21: Position (x-Axis, Complementary Filter)

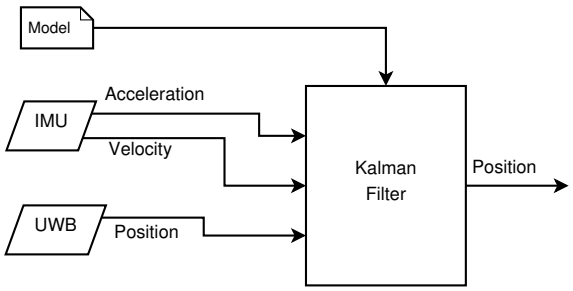


Figure 6.22: Kalman Filter to Fuse IMU and UWB Measurements

Result The resulting positions are shown in figure 6.21. In the experiment without any motion, the complementary filter reduces the position drift of the IMU data, but it is not able to compensate it. Furthermore, the final position still contains the outliers of the UWB measurements. Overall, UWB data alone yields better results than any sensor fusion.

Kalman Filter

As an alternative to the complementary filter, a Kalman filter was applied. It uses the *acceleration* and *velocity* computed from IMU data, as well as the *position* determined by the UWB module. The setup is shown in figure 6.22.

Model At first, the system state y is defined by the current position s, velocity v and acceleration a:

$$y = \begin{bmatrix} s \\ v \\ a \end{bmatrix} \tag{6.24}$$

The model A, predicting the new state from the previous one, is set to

$$A = \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.25}$$

while the model B, that predicts what changes, is set to

$$B = \begin{bmatrix} \Delta t^2 \\ \Delta t \\ 1 \end{bmatrix} \tag{6.26}$$

The variable u, which is usually the command input, is set to the acceleration a, measured by the accelerometer:

$$u = a \tag{6.27}$$

The meaning of the equations (6.25) to (6.27) is that the current position is guessed using the previous position, the previous velocity and the current acceleration, measured by the accelerometer. The guessed velocity is composed of the recent velocity, as well as the current acceleration. Finally, the guessed acceleration is equal to the measured acceleration. In this model, *uniformly accelerated motion* is assumed. Next, the error covariance matrix Q is defined as

$$Q = \begin{bmatrix} \sigma_s^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_a^2 \end{bmatrix} \tag{6.28}$$

where σ^2 is the variance, which is computed for the position, velocity and acceleration over all samples, respectively. The model H of how the sensor readings reflect the system state is set to

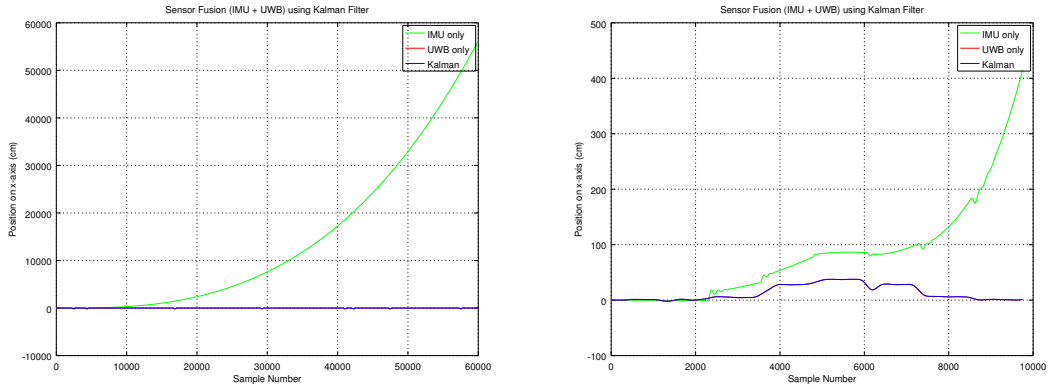
$$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{6.29}$$

Next, the covariance of error noise R is set to

$$R = \sigma_s \tag{6.30}$$

with σ_s being the standard deviation of the position obtained from the UWB module. The measurement z is simply the position s_{UWB} derived from UWB measurements:

$$z = s_{UWB} \tag{6.31}$$



(a) No Motion (b) 30° Step-wise Motion

Figure 6.23: Position (x-Axis, Kalman Filter)

Result The positions, determined by the Kalman filter without and during motion, are shown in figure 6.23. Compared to the position computed by the complementary filter (figure 6.21, p. 102), the Kalman filter realizes the lacking quality of the IMU data and ignores them almost entirely. Consequently, the results of the Kalman filter are just as good as the UWB measurements.

All final P and K values are summarized in section C.1.2 in the appendix (p. 135).

7 Evaluation

In this chapter, the results of the experiments are evaluated. Firstly, the different methods to improve the orientation are examined. In section 7.2, the achieved accuracy of the final position is surveyed.

7.1 Orientation

To compare the quality of the *quaternions*, obtained from different smoothing methods and sensor fusion approaches, the *Root Square Error (RSE)* e for the quaternions at two different angles was computed:

$$e = \sqrt{(w_{meas} - w_{exp})^2 + (x_{meas} - x_{exp})^2 + (y_{meas} - y_{exp})^2 + (z_{meas} - z_{exp})^2} \quad (7.1)$$

It is similar to the Root Mean Square Error (RMSE), but instead of using the difference to the mean, the difference to the expected value is computed for each measurement. The smaller the RSE, the closer is the measurement to the expected value.

The resulting RSEs for the different experiments are shown in table 7.1 and 7.2. In the experiment *without motion*, the final quaternion (sample $59,989$) was used to compute the RSE. The expected quaternion is $(1, 0, 0, 0)$. For the experiment with *step-wise movements*, the first quaternion value is at an angle of 90° (sample number $6,000$), while the second one is at an angle of 0° at the end of one set of 30° step-wise movements (sample number $9,750$). The expected quaternion at 90° is $(0.7071, 0, 0.7071, 0)$, while the expected quaternion at the end of the movement is $(1, 0, 0, 0)$.The last column of table 7.2 shows the sum of the two RSEs.

7.1.1 Smoothing

No Motion For the experiment *without motion*, the smoothing methods are ranked by their RSE as follows:

1. Dynamic Calibration
2. 1D Kalman filter
3. Moving average

Method	**RSE**
Smoothing	
Dyn. Cal.	0.011387
Mov. Avg.	0.011418
FIR Filt.	0.011420
1D Kal. Filt.	0.011395
Sensor Fusion	
Com. Filt. (Prep.)	0.001004
Com. Filt. (No Prep.)	0.000036
Kal. Filt.	0.001093

Table 7.1: Root Square Error of the Final Quaternion Without Motion

Method	**RSE** (90°)	**RSE** (0°)	**Sum**
Smoothing			
Dyn. Cal.	0.0773	0.0612	0.1385
Mov. Avg.	0.0788	0.0596	0.1384
FIR Filt.	0.0780	0.0609	0.1389
1D Kal. Filt.	0.0771	0.0592	0.1363
Sensor Fusion			
Com. Filt. (Prep.)	0.0792	0.0203	0.0995
Com. Filt. (No Prep.)	0.0708	0.0084	0.0792
Kal. Filt.	0.7415	0.1353	0.8768

Table 7.2: Root Square Error of Different Quaternions During Motion

4. FIR Filter

Overall, the smoothing methods remove peaks from the graphs, which may be the reason why they are less accurate than dynamic calibration only. Compared to the FIR filter, the Kalman filter has a feedback loop. Maybe this is why it is more accurate than the FIR filter.

Motion According to the total RSE for the measurements during *motion*, the smoothing methods can be ordered from best to worst as follows:

1. 1D Kalman filter
2. Moving average
3. Dynamic Calibration
4. FIR Filter

Again, the 1D Kalman filter provides a feedback loop, which is probably why it performs best during motion. The moving average and FIR filter achieve better results than dynamic calibration at the end of the motion, but they perform worse at 90°. The reason for this could be the use of a window, which the Kalman filter does not use.

Final Ranking The final ranking for smoothing combines the rankings for both experiments:

1. 1D Kalman filter
2. Dynamic Calibration
3. Moving average
4. FIR Filter

Consequently, the recommendation for this application is to use a Kalman filter to improve the obtained orientation.

7.1.2 Sensor Fusion

No Motion According to the RSE, sensor fusion improves the result if there is *no motion*. In this case, the *complementary* filter, especially without pre-processing, performs slightly better than the *Kalman* filter. All of them aim to reduce orientation drift, which is why they lead to better results than smoothing alone. The reason for the complementary filter performing better *with* pre-processing is that the low-pass filter reduces acceleration peaks. As there is less acceleration applied to the system, there is less displacement.

Motion For the *motion*-experiment, the *complementary* filter *with* pre-processing performs better than dynamic calibration only. It is slightly less accurate during motion, but it removes the drift. *Without* pre-processing, the complementary filter performs even better, which is shown by its lower RSE.

The *Kalman* filter is way off and cannot be used to determine the rotation reliably. The reason for this is, that it fails to distinguish between an actual acceleration and the normal force. Thus, it simply amplifies the angle computed from accelerometer data. However, the expected outcome was a smooth line, similar to the graph of the angles computed from gyroscope data, but with the baseline of the angles computed from accelerometer data.

This is surprising, as it is supposed to be one of the strengths of the Kalman filter to recognize outliers and unreliable sensors and give them less weight. This issue may be resolved by using another model instead of one for balancing a robot. Another chance is the use of the *Extended Kalman filter* to fuse quaternions instead of angles.

Conclusion Consequently, the use of the Kalman filter with the used model is not recommended during motion. The *complementary filter*, especially without pre-processing, yields quite good results. It outperforms all smoothing methods, as well as dynamic calibration alone.

7.2 Position

To compare the final position of the different methods, the overall displacement d has been computed as the length of the final position vector s:

$$d = \sqrt{s_x^2 + s_y^2 + s_z^2} \tag{7.2}$$

For the comparison of the results of sensor fusion of IMU and UWB measurements, only the x-value has been used. Table 7.3 shows the resulting displacements for the different smoothing and sensor fusion methods, as well as the expected value and those from IMU and UWB, respectively. It displays the final displacement after two minutes without motion (sample 59, 989), as well as the displacement at an angle of 90° and at the end of the movement, back at 0° (samples 6000 and 9750).

The displacements of the smoothing methods and sensor fusion for the orientation are compared to the displacement computed from dynamically calibrated data. The displacement of the UWB measurements and the sensor fusion on position are simply the x-values of the corresponding sample. They are compared to the final x-position using dynamically calibrated data.

Displacement	No Motion		Step-wise Movements			
	Final	Vs. Dyn. Cal.	At 90°	Vs. Dyn. Cal.	At End of Mvmt.	Vs. Dyn. Cal.
Expected	0.00m		0.36m		0.00m	
Unprocessed						
Dyn. Cal	562.43m		17.98m		72.56m	
UWB Only	0.03m		0.32m		0.01m	
Smoothing						
Mv. Avg.	561.19m	99.78%	18.30m	101.80%	73.70m	101.57%
FIR Fil.	561.44m	99.82%	18.13m	100.87%	72.96m	100.55%
Kal. Fil. 1D	562.00m	99.92%	18.13m	100.87%	73.50m	101.30%
Sensor Fusion (Orientation)						
Comp. Fil. (Prep.)	39.59m	7.04%	20.20m	112.37%	49.51m	68.23%
Comp. Fil. (No P.)	120.98m	21.51%	15.01m	83.52%	41.86m	57.69%
Kal. Fil.	177.64m	31.58%	106.81m	594.12%	339.42m	467.78%
Sensor Fusion (Position, x-axis only)						
Dyn. Cal.	561.19m		0.86m		4.15m	
Comp. Fil.	112.26m	20.00%	0.43m	49.71%	0.84m	20.23%
Kal. Fil.	0.03m	0.005%	0.32m	37.14%	0.01m	0.29%

Table 7.3: Displacement of the Different Methods With and Without Motion

7.2.1 Raw Values

Even though it has been calibrated, *IMU* measurements alone are not accurate enough to grab an object. It would not even be precise enough to locate a person within a room. The problem is that the velocity, which is computed from the measured acceleration, is rotated using the obtained quaternions. If they drift, velocity drifts as well. Better calibration equipment and methods are probably able to improve the result.

The measurements of the *UWB* module were quite accurate, except for a few outliers. It does not suffer from drift and would be accurate enough to locate a person within a room. However, all final displacements are greater than half the gripper length, which is why they would probably not be precise enough to grab objects.

7.2.2 Smoothing

The impact of smoothing is small. Without motion, smoothing slightly increases the accuracy, but as the final displacement is in the same range as without smoothing, this improvement is not significant. During motion, the obtained positions are insignificantly less accurate than the ones obtained using dynamic calibration only.

Maybe, smoothing would yield better results using a different window size. Assuming a sampling frequency of $500Hz$, the window size of 20 corresponds to a time interval of only $0.04s$. By increasing the window size, the raw values would be smoothed more, which would reduce noise in the obtained orientation and acceleration. Furthermore, for the FIR filter, the cutoff-frequency may be varied. For the Kalman filter, the result may be improved by experimenting with different p and q values.

7.2.3 Sensor Fusion (Orientation)

Complementary Filter The different methods to fuse gyroscope and accelerometer data to obtain an orientation without drift yield very different results. The complementary filter *with* pre-processing reduces the position drift and thus it works well if there is no motion. It improves the dynamically calibrated data by more than 95%. The version *without* pre-processing performs not as good, because it improves the result with dynamic calibration only by not even 80%.

During motion, at an angle of 90°, the displacement of the complementary filter *with* pre-processing is greater than *without* sensor fusion, but back at the initial position, the drift is reduced. The issue during movement is probably caused by a slightly incorrect orientation. One influencing factor for this is the *accelerometer range*, which was set to $0.1G$. It means that all accelerations, which differ less than

$0.1G$ from the normal force, are already taken into account to compute the tilt angle from accelerometer data.

The version *without* pre-processing works well during all phases of motion. At an angle of 90°, it improves the result of dynamically calibrated data by more than 15% and back at 0°, it is more than 50% better than dynamically calibrated data. It is the only filter that improves the results of dynamically calibrated area in both experiments, with and without motion.

Kalman Filter The Kalman filter reduces the drift without motion by almost 60%, but it completely fails during motion. As mentioned before, this is due to the inability to distinguish physical acceleration from the normal force for orientation computation. It is no surprise, that with an erroneous orientation, the position is incorrect as well.

This result may be improved by introducing an *accelerometer range*, like for the complementary filter. However, it was expected that the Kalman filter was able to yield fair results without any constraints and pre-filtering.

7.2.4 Sensor Fusion (Position)

Complementary Filter Due to its manner of functioning, it is no surprise, that the complementary filter reduces the position drift. However, it is not able to compensate for it completely. The final displacement is worse than the one of the complementary filter to fuse gyroscope and accelerometer data. This is because the input of this filter was dynamically calibrated data. Therefore, an idea is to use the output of the previous complementary filter as the input of this complementary filter. The inferior displacement at 90° may be compensated by the UWB measurements.

Kalman Filter Compared to the previous results, this Kalman filter works as expected. It recognizes the inaccurate IMU measurements and gives them less weight. As a result, the final displacements are very close to the ones when using UWB data only. This is confirmed by the final Kalman gain values, stated in section C.1.2, which show, that the Kalman filter uses UWB data almost entirely.

In comparison to the expected values, using UWB data only yields the most accurate results, which are just compounded by fusing it with IMU data.

8 Summary and Conclusion

Summary At the end of the work for this thesis, all components were set up and worked together. The *communication* with the IMU, the UWB module and the PC is facilitated by the I^2C and SPI buses, as well as UART. The communication over SPI works without any issues. To overcome the *I^2C Bus Busy* problem, an interrupt-based readout of the IMU's registers was implemented, using either the interrupt of the IMU or a timer interrupt within the ZedBoard. Consequently, the connection is more stable and reliable, even though the issue still arises from time to time. The readout of the different sensor data works well, as long as there are no issues with the bus.

Furthermore, to control the *IMU*, the freely available *Motion Driver* by InvenSense was integrated into the software architecture, such that the included functions did not have to be implemented again.

The *driver* and application software for the *UWB* modules have been integrated by splitting the required functionality into several methods. An implemented state machine takes care of the execution of the suited function at the appropriate point of time.

The *VHDL module* to control the *robot*'s servos using PWM has been included in the ZedBoard's PL. There are several functions to set the angle of a specified joint. The conversion to the corresponding PWM value has been implemented. Even though this work did not go as far as grabbing an object, the required math to compute the joint angles has been done.

Finally, two different methods to communicate with the PC were implemented. Firstly, the sample just read from the sensors can be transferred to the PC, where the user can view them using a UART Terminal or use the *IMU Viewer* to visualize the IMU's movements in real-time. Due the to massive position drift, the display of the position has been disabled. Nevertheless, displaying the IMU's rotation works very well.

The other way to transfer the measurements from the ZedBoard is to collect them in the ZedBoard's DDR memory and transfer them to the PC all together, using the *XModem* protocol. Then, the data can be analyzed offline, using Octave or other tools.

There were two different methods for calibrating the gyroscope and accelerometer implemented. *Static calibration* uses previously determined offsets and adjusts measurements according to the a-priori sensitivity of the sensors. The other method, *dynamic calibration*, is to compute the offset using the first one-hundred values of the corresponding sample set.

The experiments suggest that, generally, *dynamic calibration* works better. It results in quite stable quaternions without motion. During step-wise movements, the quaternions are usable, but they suffer from drift. After two minutes without any movements, the IMU was displaced by $562m$. After one stepwise movement of the shoulder joint to 90° and back, the displacement was equal to $72m$.

In order to enhance the results, the dynamically calibrated data was smoothed using the *moving average*, *FIR filter* and a *1D Kalman filter*. In general, all smoothing methods did not change the resulting position significantly.

To remove the orientation drift, the gyroscope and accelerometer data were fused using two different types of filters: the *complementary* and the *Kalman filter*. In the experiment without motion, both improved the results of dynamically calibrated data by more than 90% (complementary filter with pre-processing) and 70% (Kalman filter), with a final displacement of $39m$ and $177m$. However, during motion, the Kalman filter was not able to distinguish between physical acceleration and normal force, which is why it impairs the dynamically calibrated data, instead of improving it. Overall, for orientation determination, the complementary filter yielded better results than the smoothing methods.

To find out whether the good results of the complementary filter were due to preprocessing, the same filter was applied to unprocessed data. In this case, the filter performed worse in the experiment without motion, where the final displacement was $120m$. However, during all phases of the experiment with motion, it performed better than any smoothing or sensor fusion method, as it improved the dynamically calibrated data. The displacement after one movement was equal to $41m$.

To re-calibrate the position obtained from IMU data, it was fused with UWB data, using the same filters as above. The *complementary filter* was able to reduce the drift, but it was not able to compensate for it. The final displacement after two minutes without motion was $112m$. The *Kalman filter* detected that the position obtained from IMU data is useless. Therefore, it almost completely used UWB data, which means that it is just as good as using UWB measurements only. The final displacement after two minutes without motion was $0.028m$. During motion, it was a little less accurate than UWB data only.

Future Work The results may be improved by introducing *constaints*. For example, the UWB outliers could be deleted by measuring the minimum and maximum

distance between the UWB module attached to the robot arm and the anchor, using a measuring tape. All smaller and larger values would be ignored.

More constraints may be applied to the *position* by using the length of the solid parts of the robot arm. If it is not set onto a chassis and thus not able to move freely within the room, the maximum displacement for each axis is equal to the sum of the length of all solid parts. Furthermore, the velocity may be constrained by the maximum velocity of the servo motors.

Another idea to improve the result of the *Kalman filter* for orientation is to use the PWM register values as command input u. This allows to determine the angle quite accurately. However, as the aim was to use IMU and UWB data only, this has not been tried for this work. Further experiments with the model, or even the use of an Extended Kalman filter to fuse quaternions instead of angles may also lead to better results.

Finally, the *smoothing* methods may be evaluated in more detail by experimenting with their parameters. Further experiments with the parameters of the complementary filter may improve the results as well.

To include all findings of this thesis into the *software*, the smoothing and sensor fusion algorithms will have to be implemented in the future.

Conclusion Overall, this work did not lead to any significantly different outcome than the already existing literature. The problem of orientation drift remains as the main challenge in this field. The main contribution of this work is the suggestion to determine the orientation using a complementary filter with *acceleration range*, while the position is most accurately computed from UWB data only for this special application.

Appendices

A Software Details

A.1 Source Code and Application Guide

The required sources for the system setup, software application, IMU Viewer and the Data Unpacker, as well as the Octave scripts for data analysis can be accessed at:

https://github.com/stkunkel/indoor-loc

It contains the following folders:

- `imu_viewer_osg`: IMU Viewer PC application
- `indoor-loc`: hardware and software project for the ZedBoard
- `output`: collected data and Ocatve scripts

For this project, *Vivado 2015.03* (64-bit) was used for the PS and PL setup of the ZedBoard, and the corresponding Xilinx Software Development Kit (XSDK) was used for the software development. It is recommended to use the same version of Vivado to ensure proper functionality. The hardware project file `indoor-loc.xpr` is located in the folder `indoor-loc/`.

To import the hardware description, software project and BSP into the XSDK, its *Import* functionality can be used. In the tab *General*, there is an option called *Existing Projects into Workspace*. The root directory of the software project is `indoor-loc/indoor-loc.sdk/`. The following projects have to be imported:

- `IndoorLoc`: software application
- `design_1_wrapper_hw_platform_1`: hardware description
- `IndoorLoc_gpio_timer_pwm_uwb_bsp`: BSP

The source code for the *software application* is located in `indoor-loc/indoor-loc.sdk/IndoorLoc/src/`. It contains different sub-folders for sources related to the IMU, robot arm control, UWB and different ZedBoard utilities. Furthermore, there are some general source files, as well as the main program `run.c` and its parameters in `program_parameters.h`.

A.1.1 Viewing the IMU's Orientation in Real-time

To view the IMU's orientation on the PC in real time, the following steps have to be done:

1. On the PC, open a terminal, change to the `imu_viewer_osg` directory, perform a *Make*, and execute the resulting executable `imuviewer`.
2. Set the required program parameters in the *XSDK*:
   - `USE_DMP` is commented out to use the DMP to compute the quaternions. If it is commented, the quaternions are computed by the application, using sensor data.
   - `DEBUG` must be commented to prevent unnecessary console output.
3. There are two different options available:
   - To move the IMU manually, comment out the function `printGeneric`, set its parameters and comment out the `moveAndCollect` function. Make sure, that the first argument of the print-function is `PRINT_FOR_VIEWER`, and the second one is `SEPARATOR`. They may be varied, if the data is simply viewed using a UART terminal, but to view them with the IMU Viewer, the specified data format must be ensured.
   - To let the robot perform the movements, the different angles have to be specified in the `angle` array and the joint to be moved has to be specified in the `moveAndCollect` function, which is used instead of the `printGeneric` function. Its last parameter has to be set to `BOOL_FALSE`, in order to select real-time data transmission instead of a comprehensive data collection and later transmission.
4. Adjust the gyroscope and accelerometer FSR in `mpu.h` (optional).
5. Program the PL and execute the software.

Alternatively, instead of opening the IMU Viewer, a UART terminal may be used to readout the data. If this is the case, the `DEBUG` define may be commented out to see more debug printouts. Furthermore, instead of `PRINT_FOR_VIEWER` and `SEPARATOR`, different parameters may be used. For the print mask, there are several options defined in the file `mpu.h`, which is located in the folder `imu/`.

A.1.2 Collecting and Receiving Robot Movement Data

To receive and analyze robot movement data on the PC, a program, which is able to receive data using the *XModem* protocol, has to be installed. For this project, Kermit was used. The following steps have to be done to receive the sensor and PWM data:

1. Set the desired program parameters and comment out the function `moveAndCollect`, while commenting the function `printGeneric`. Make sure that its final parameter is set to `BOOL_TRUE`. Furthermore, the movement has to be specified in the `angle` array.

2. In the XSDK, set a breakpoint before the *XModem* transmission starts, e.g. in the function `xmodemTransmit` within the file `xmodem.c` in the `zedboard/` folder. This ensures, that the user has enough time to start the receiver program on the PC and prevents a timeout, which results in the loss of the collected data. Then, the PL has to be configured and the software application has to be executed in debug mode.

3. On the PC, open a terminal and change to the `output` directory. Once the breakpoint is reached, execute the *XModem* receiver program and call the output file `data.bin`. For example, after starting *Kermit*, the command is:

   ```
   receive /as-name:data.bin /protocol:xmodem
   ```

4. Continue the program in the XSDK and wait until the data transfer has finished. This usually takes some time, depending on the sampling time.

5. On the PC, exit the *XModem* receiver program and perform a *Make* in the `output/` directory, where the file `data.bin` must be located. Then, the executable `unpackData` has to be executed. It outputs an unpacked data file, called `data.txt`, which contains the raw IMU sensor data, the UWB distance in *cm*, and the content of the PWM registers.

A.1.3 Analyzing Robot Movement Data

To analyze the received data, there are several Octave scripts available in the `template/` folder in the directory `output/`. The functionality of the different template files is summarized in table A.1. The most important functions are `filterForCompFil.m`, `filterUwb.m`, `fusionImuUwb.m`, `quaternions_simple_cal_filter.m`, `quatVelPos_cal_fil_compFilter.m`, `quatVelPos_cal_fil_kalman.m` and `simple_calibration.m`.

Most of the script files export the computed data as an Octave matrix file (`*.mat`). The output of the `quatVelPos` files can be used as input for the IMU Viewer by renaming it to `quatPos.mat` and copying it into the `imu_viewer_osg` directory.

The functionality of the scripts marked with `(*)` is configured, using parameters at the beginning of the file, which specify the input data (raw, calibrated, filtered, etc.) and the applied smoothing method, filter, etc. They should be adjusted to the required purpose before executing them. Furthermore, the required input file must have been generated beforehand.

File Name	Functionality
cal_getStd.m	Standard deviation of statically calibrated data
calibrate.m	Static calibration
createFilteredData.m	Plots for FIR filtered data
filterForCompFil.m	Pre-processing for complementary filter
filterUwb.m*	Low-pass filter for UWB distance
firFilterAcc.m	Low-pass FIR filter for acceleration
firFilter_cal.m	Low-pass FIR filter for statically calibrated data
firFilterGyro.m	Low-pass FIR filter for angular velocity
firFilter_simple_cal.m	Low-pass FIR filter for statically calibrated data
fusionImuUwb.m	Sensor fusion on position
getMean.m	Mean of acceleration
kalman_cal.m	1D Kalman filter for statically calibrated data
kalman_simple_cal.m	1D Kalman filter for dynamically calibrated data
movingAverage_data_cal.m	Moving average for statically calibrated data
movingAverage_data_simple_cal.m	Moving average for dynamically calibrated data
movingAverage_temp_raw.m	Moving average for raw temperature data
plotAcc.m	Plot raw acceleration and apply FFT
plotFirstDer.m	Plot expected and measured values (test function for dyn. cal. data)
plotGyro.m	Plot raw angular velocity and apply FFT
plotMag.m	Plot raw magnetometer data
plotQuat_compFil.m*	Plot Quaternions from complementary filter
plotQuat.m*	Plot Quat. from raw, calibrated, smoothed or Kalman-filtered data
plotRaw.m	Plot raw data
plotTemp.m	Plot Temperature
plotVelPos.m*	Velocity and position from dyn. cal., smoothed or Kalman-filt. data
printVarStd.m	Standard deviation of dynamically calibrated data
quat_cal_compFilter.m	Quaternions from stat. cal., complementary-filtered data

quaternions_cal.m	Quaternions from statically calibrated data
quaternions.m	Quaternions from raw data
quaternions_simple_cal_filter.m*	Quat. from raw, calibrated or smoothed data
quatFirstPeak.m	First peak of quaternion during motion
quatVelPos_cal_fil_compFilter.m*	Quaternions from Complementary Filter
quatVelPos_cal_fil_kalman.m*	Quaternions from Kalman Filter
simple_calibration.m	Dynamic Calibration
velPos_cal.m	Plot velocity and position from statically calibrated data
velPos.m	Plot velocity and position from raw data

Table A.1: Octave Templates

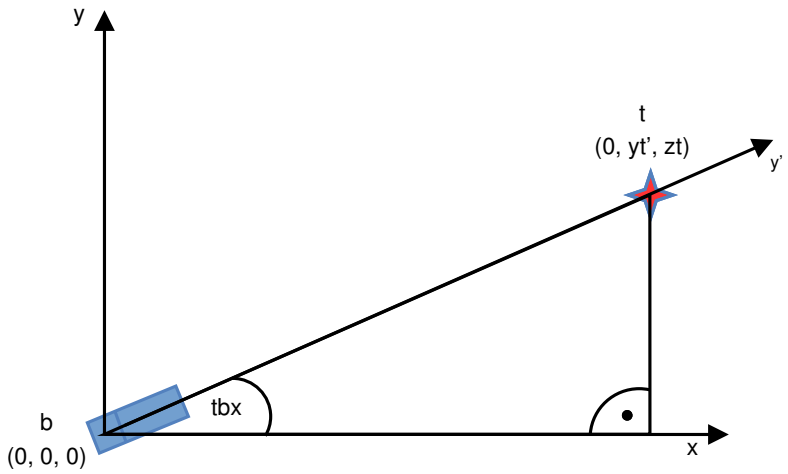


Figure A.1: Transformation to the New Coordinate System

A.2 Robot Control: Inverse Kinematics

If the robot is required to pick up an object, its joints have to be aligned, such that the grippers' position is equal to the position of the target. The different joint angles are determined by *inverse kinematics*.

In the robot arm's coordinate system, the x-axis points to the right, the y-axis away from the reader and the z-axis upwards. As the robot arm offers five degrees of freedom (one for each joint), inverse kinematics is applied by using the *cosine law*, as described below. [66]

Determine the Base Angle and Facing the Target

In order to grab an object at known location (x_t, y_t, z_t), the base first has to be rotated by an angle of

$$tbx = \arctan\left(\frac{y_t}{x_t}\right) \tag{A.1}$$

such that the robot faces the target. This simplifies the problem from a 3D- to a 2D-coordinate system, where $x' = 0$ and $y' = \sqrt{x^2 + y^2}$ as shown in figure A.1.

Defining the Grip Angle

To determine the angles of the joints, it is assumed that the angle ga between the connection $\overline{gr}$ of gripper and wrist in relation to the base-shoulder connection $\overline{bs}$ has already been defined (see figure A.2).

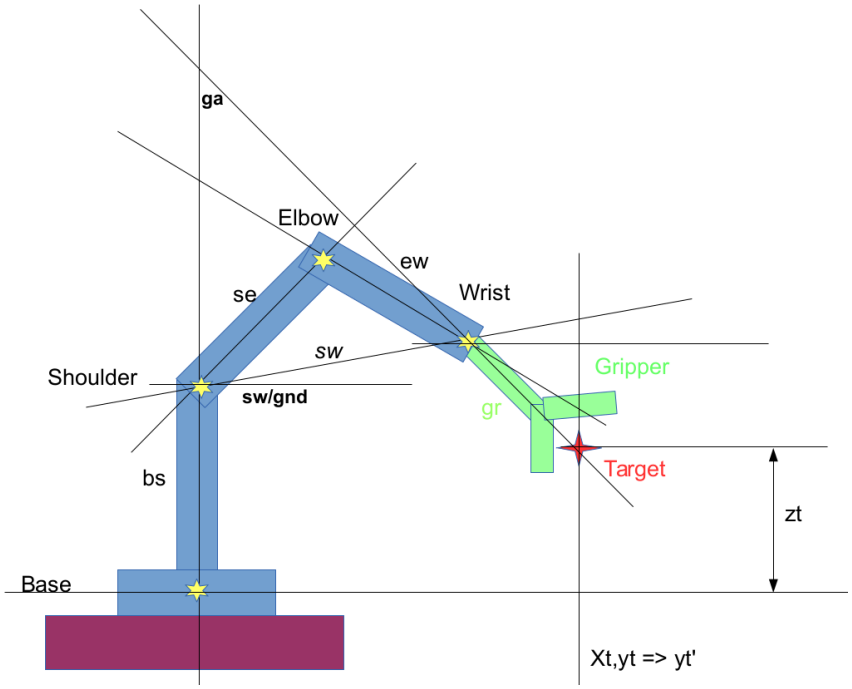


Figure A.2: Robot Arm Angles [66]

Determine the Wrist Position

Now, the wrist position (y_w, z_w) can be computed from (y'_t, z_t) using the angle ga and the length of the gripper $\overline{gr}$:

$$y_w = y'_t - \overline{gr}\sin(ga) \tag{A.2}$$

$$z_w = z_t - \overline{gr}\cos(ga) \tag{A.3}$$

Computing the Distance between Shoulder and Wrist

The next step is to compute the shoulder-wrist distance $\overline{sw}$ from the wrist position, where the shoulder's y-value is zero and $z_s = \overline{bs}$:

$$\overline{sw} = \sqrt{y'^2_w + (z_w - \overline{bs})^2} \tag{A.4}$$

Calculating the Angle between the Shoulder-Wrist-Line and the Ground

The angle of the shoulder-wrist line to the ground is calculated using the following formula:

$$sw/gnd = sg = \arccos\left(\frac{y'_w}{\overline{sw}}\right) \tag{A.5}$$

Determine the Shoulder and Elbow Angles

Then, the *shoulder* and *elbow* angles esw and sew are determined using the cosine law:

$$esw = \arccos\left(\frac{-\overline{ew}^2 + \overline{se}^2 + \overline{sw}^2}{2\overline{es} \times \overline{se}}\right) \tag{A.6}$$

$$sew = \arccos\left(\frac{\overline{ew}^2 + \overline{se}^2 - \overline{sw}^2}{2\overline{ew} \times \overline{se}}\right) \tag{A.7}$$

Transformation back to World Coordinate System

Finally, the equations

$$x = \sqrt{y'^2 - y^2} \tag{A.8}$$

and

$$y = \sqrt{y'^2 - x^2} \tag{A.9}$$

transform all coordinates in the new coordinate system back to the world coordinate system.

Summary of Equations

The equations required to compute the joints' angles are summarized in table A.2. As stated in section 3.1 (p. 31), all connections between the joints ($\overline{bs}$, $\overline{se}$, $\overline{ew}$ and $\overline{gr}$) are $12cm$ long, while the grippers are $2cm$ long. The distance between *shoulder* and *wrist* is computed by equation (A.4). Note again, that a given grip orientation is assumed. However, as a fixed grip angle will not work in all cases (i.e. leading to computed values out of range), a re-computation of the joint angles using a different grip angle may be required. From previous testing, steps of approximately 30° are supposed to be reasonable. [66]

All deviations of all formulas are explained later on in chapter B (p. 129). How the joints are moved to the corresponding angles, has already been described in section 4.1.1.

Joint	Angle
Base	$tbx = \arctan\left(\frac{y_t}{x_t}\right)$
Shoulder	$esw = \arccos\left(\frac{-\overline{ew}^2+\overline{se}^2+\overline{sw}^2}{2\overline{es}\times\overline{se}}\right)$
Elbow	$sew = \arccos\left(\frac{\overline{ew}^2+\overline{se}^2-\overline{sw}^2}{2\overline{ew}\times\overline{se}}\right)$
Wrist	ga has to be given.

Table A.2: Equations to Compute the Joint Angles of the Robot Arm

A.3 Details on IMU Operation

A.3.1 *MPU-9150* Initialization Values

Parameter	Value
Gyroscope FSR	$\pm 1000^{\circ}/s$
Accelerometer FSR	$\pm 4G$
DLPF	$42Hz$
FIFO Rate	$500Hz$
Clock Source	Gyroscope PLL
Enable FIFO	No
Data Ready Interrupt	Disabled
	Active Low
	Unlatched

Table A.3: *MPU-9150* Initialization Values

A.3.2 Interrupts

IMU Interrupt

Interrupt Setup The IMU's interrupt is configured using mostly *Motion Driver* functions:

1. Configuration of the IMU's interrupt pin as *push-pull* with the interrupt being *active high*, until it is cleared by reading the interrupt status register.
2. Setting the interrupt level in the *Motion Driver* to *active high*
3. Enable *latched* interrupt in the *Motion Driver*

4. Setting the interrupt mode as *continuous*, meaning that an interrupt is sent each time one FIFO period is over
5. Configure and enable the interrupt in the PS as described in Taylor [90].

Interrupt Handling As soon as there is an interrupt from the *MPU-9150* detected at the corresponding GPIO pin, the inherent interrupt handler is called. It clears the interrupt status register of the IMU and sets a flag which shows that there is new data available in the IMU's FIFO.

Note, that the GPIO interrupt handler provided by the BSP lacks to check whether the received interrupt has been enabled for this pin beforehand. As a result, all received interrupts, not only the ones triggered by the specified pin, lead to the execution of the interrupt handler.

Consequently, the IMU's FIFO was tried to be read more often than necessary, leading to I^2C errors. In the long term, the connection broke down completely. To resolve this issue, the missing check has been implemented in a custom GPIO interrupt handler.

Timer Interrupt

Interrupt Setup Another way is to configure the timer interrupt in the PL. A Timer counts from zero up to a specified value, where an interrupt is triggered. The timer interrupt is set up by following the sequence:

1. Initialize the Timer Controller
2. Set the interrupt handler for the Timer Controller
3. Reset the Timer value
4. Set the desired options
5. Configure and enable the interrupt in the PS, similar to step 5 above
6. Start the Timer

Interrupt Handling The timer interrupt handler simply stops the Timer Controller, sets a flag and resets the timer, before re-starting the Timer Controller.

B Derivation of Functions for Robot Joint Angle Computation

B.1 Base Angle

Figure B.1 shows the robot in relation to the target viewed from above. The robot arm points along the x-axis. In order to grab the target, the robot arm has to face the target. To compute the angle of the target, relative to the x-axis, and therefore the angle that the robot arm has to turn, the *tangent* function for a right triangle can be used:

$$\tan(tbx) = \frac{y_t}{x_t} \tag{B.1}$$

To compute the angle, the equation above is rewritten as

$$tbx = \arctan\left(\frac{y_t}{x_t}\right) \tag{B.2}$$

B.2 Wrist Position

A sketch of how to compute the wrist position is shown in figure B.2. The point A, where there is a right angle, is located below the wrist. Its coordinates (y'_A, z_A) are determined as follows:

$$y'_A = y'_t - y'_w \tag{B.3}$$

$$z_A = z_t - z_w \tag{B.4}$$

The sine of the angle ga is determined by

$$\sin(ga) = \frac{y'_t - y'_w}{\overline{gr}} \tag{B.5}$$

which can be rewritten as

$$\overline{gr}\sin(ga) = y'_t - y'_w \tag{B.6}$$

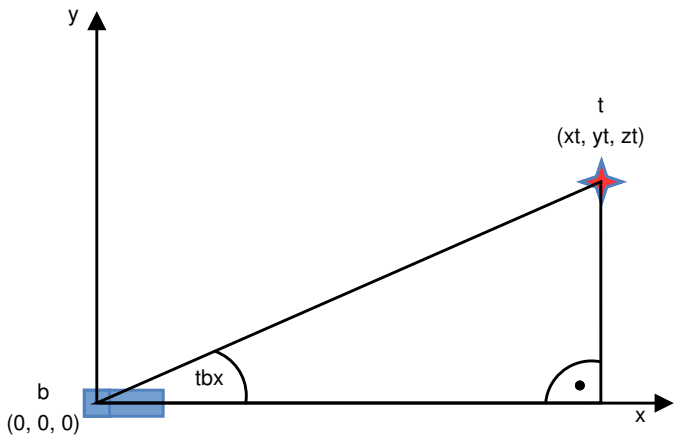


Figure B.1: Base Angle

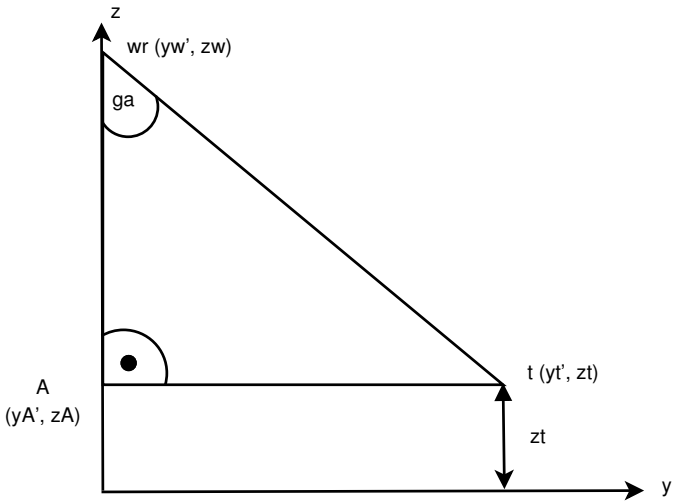


Figure B.2: Wrist Position

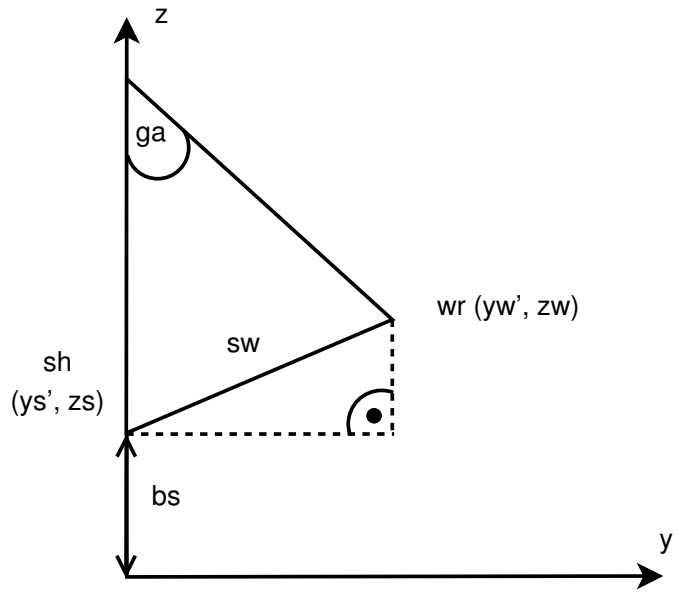


Figure B.3: Distance between Shoulder and Wrist

Finally, y'_w can be computed by

$$y'_w = y'_t - \overline{gr}\sin(ga) \tag{B.7}$$

The cosine of the angle ga is determined by

$$\cos(ga) = \frac{z_t - z_w}{\overline{gr}} \tag{B.8}$$

which can be rewritten as

$$\overline{gr}\cos(ga) = z_t - z_w \tag{B.9}$$

Then, z_w can be calculated by

$$z_w = z_t - \overline{gr}\cos(ga) \tag{B.10}$$

B.3 Distance between Shoulder and Wrist

The distance between shoulder and wrist $\overline{sw}$ is determined by

$$\overline{sw} = \sqrt{(y'_w - y'_s)^2 + (z_w - z_s)^2} \tag{B.11}$$

using the dotted triangle shown in figure B.3. As the shoulder is located above the base with a distance of $\overline{bs}$, which is the distance between them, the shoulder's

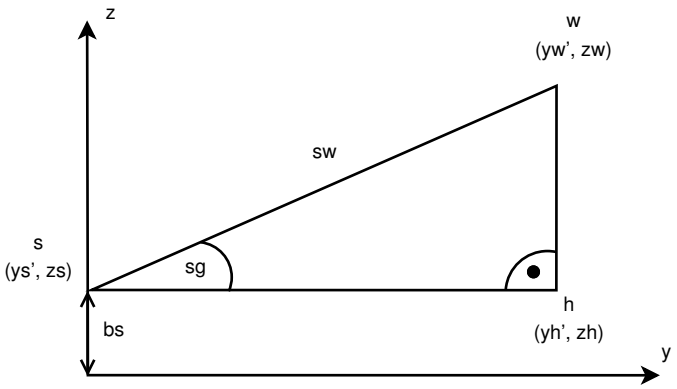


Figure B.4: Angle between Shoulder and Ground

coordinates (y_s, z_s) can be resolved as $(0, \overline{bs})$. By substitution, this leads to the following formula to compute $\overline{sw}$:

$$\overline{sw} = \sqrt{(y'_w - 0)^2 + (z_w - \overline{bs})^2} \tag{B.12}$$

Finally, by solving equation (B.12), $\overline{sw}$ is calculated by:

$$\overline{sw} = \sqrt{y'^2_w + (z_w - \overline{bs})^2} \tag{B.13}$$

B.4 Angle between Shoulder and Ground

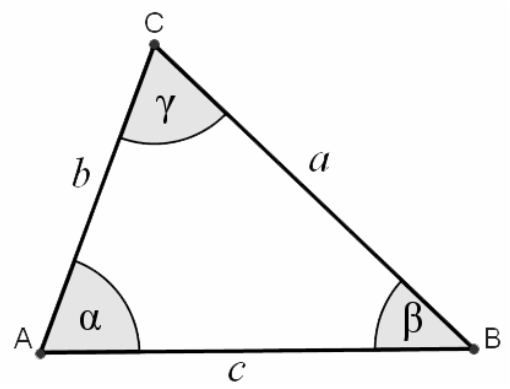
The computation of the angle between the shoulder and the ground is based on figure B.4. The helper point h is as high as the shoulder, but right below the wrist. Therefore, $\overline{sh}$ is parallel to the ground. The coordinates (y'_h, z_h) of the helper point h are determined easily, as the y'-value is the same as the wrist's, and the z-value is the difference between the wrist's and the shoulder's z-values:

$$y'_h = y'_w \tag{B.14}$$

$$z_h = z_w - z_s \tag{B.15}$$

The angle $sg = sw/gnd$ between the shoulder and the ground is computed by the trigonometric function

$$\cos(sg) = \frac{y'_h - y'_s}{\overline{sw}} \tag{B.16}$$



$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$
$$b^2 = a^2 + c^2 - 2ac\cos\beta$$
$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$
$$\cos\alpha = \frac{-a^2 + b^2 + c^2}{2bc}$$
$$\cos\beta = \frac{a^2 - b^2 + c^2}{2ac}$$
$$\cos\gamma = \frac{a^2 + b^2 - c^2}{2ab}$$

Figure B.5: Cosine Law [66, 91]

By using equation (B.14), (B.16) can be rewritten as

$$sg = \arccos\left(\frac{y'_w - y'_s}{\overline{sw}}\right) \tag{B.17}$$

Considering that the shoulder joint is right above the base, it follows that $y'_s = 0$. Therefore,

$$sg = \arccos\left(\frac{y'_w}{\overline{sw}}\right) \tag{B.18}$$

B.5 Shoulder and Elbow Angles

The *cosine law* describes how to determine angles or distances within a triangle using the cosine. All formulas are shown in figure B.5. If replacing A, B and C with *shoulder*, *wrist* and *elbow* and a, b and c with $\overline{ew}$, $\overline{se}$ and $\overline{sw}$, the cosine of the shoulder and elbow angles esw and sew can be determined by

$$\cos(esw) = \frac{-\overline{ew}^2 + \overline{se}^2 + \overline{sw}^2}{2\overline{es} \times \overline{se}} \tag{B.19}$$

$$\cos(sew) = \frac{\overline{ew}^2 + \overline{se}^2 - \overline{sw}^2}{2\overline{ew} \times \overline{se}} \tag{B.20}$$

By resolving (B.19) and (B.20), the angles are computed by

$$esw = \arccos\left(\frac{-\overline{ew}^2 + \overline{se}^2 + \overline{sw}^2}{2\overline{es} \times \overline{se}}\right) \tag{B.21}$$

$$sew = \arccos\left(\frac{\overline{ew}^2 + \overline{se}^2 - \overline{sw}^2}{2\overline{ew} \times \overline{se}}\right) \tag{B.22}$$

C Experiment Details

C.1 Fusion of Gyroscope and Accelerometer

C.1.1 Complementary Filter

The coefficients for the high- and low-pass filters were generated using the `fir1` function with the following parameters:

- Window size: 20
- Normalized cutoff Frequency: 0.01 / 0.001
- Type: `high` / `low`

The obtained coefficients are shown in table C.1.

C.1.2 Kalman Filter

The final matrices of the Kalman filter fusing gyroscope and accelerometer data for the measurements without motion and during motion are stated in table C.2 and C.3.

C.2 Fusing IMU and UWB Measurements

C.2.1 Preprocessing

The setup of the FIR filter to smooth UWB measurements is as follows:

- Window size: $500 = 20\frac{f_{IMU}}{f_{UWB}}$, with $f_{IMU} = 500Hz$ and $f_{UWB} = 20Hz$
- Normalized cutoff frequency: $\frac{1}{f_{IMU}} = 0.002\frac{F_s}{2}$, with $F_s = f_{IMU} = 500Hz$
- Type: `low`

Due to the high amount of coefficients, they are not stated here. They can be retrieved by executing Octave with the command `fir1(500, 0.002, 'low')`. The first and last 250 values were discarded due to the filter setup time. The same measurements of the IMU were discarded as well.

Filter	
High-pass	**Low-pass**
-7.1272×10^{-4}	0.0073529
-9.1564×10^{-4}	0.0094222
-1.5027×10^{-3}	0.0154276
-2.4186×10^{-3}	0.0247811
-3.5752×10^{-3}	0.0365673
-4.8598×10^{-3}	0.0496324
-6.1465×10^{-3}	0.0626974
-7.3088×10^{-3}	0.0744836
-8.2321×10^{-3}	0.0838371
-8.8254×10^{-3}	0.0898425
9.9168×10^{-1}	0.0919118
-8.8254×10^{-3}	0.0898425
-8.2321×10^{-3}	0.0838371
-7.3088×10^{-3}	0.0744836
-6.1465×10^{-3}	0.0626974
-4.8598×10^{-3}	0.0496324
-3.5752×10^{-3}	0.0365673
-2.4186×10^{-3}	0.0247811
-1.5027×10^{-3}	0.0154276
-9.1564×10^{-4}	0.0094222
-7.1272×10^{-4}	0.0073529

Table C.1: Coefficients for the High- and Low-pass Filter

Axis	**P**	**K**
x	$\begin{bmatrix} 0.60484 & -0.49225 \\ -0.49225 & 0.44883 \end{bmatrix}$	$\begin{bmatrix} 0.84955 \\ -0.32763 \end{bmatrix}$
y	$\begin{bmatrix} 0.34538 & -0.26453 \\ -0.26453 & 0.23862 \end{bmatrix}$	$\begin{bmatrix} 0.57663 \\ -0.18480 \end{bmatrix}$
z	$\begin{bmatrix} 0.84167 & -0.74982 \\ -0.74982 & 0.71092 \end{bmatrix}$	$\begin{bmatrix} 0.88491 \\ -0.37471 \end{bmatrix}$

Table C.2: Kalman Filter (Orientation) – Final Matrices (No Motion)

Axis	P	K
x	$\begin{bmatrix} 4.1505 \times 10^5 & -4.1500 \times 10^5 \\ -4.1500 \times 10^5 & 4.1499 \times 10^5 \end{bmatrix}$	$\begin{bmatrix} 1.33582 \\ -0.35765 \end{bmatrix}$
y	$\begin{bmatrix} 7.5140 \times 10^6 & -7.5139 \times 10^6 \\ -7.5139 \times 10^6 & 7.5139 \times 10^6 \end{bmatrix}$	$\begin{bmatrix} 1.98798 \\ -0.98936 \end{bmatrix}$
z	$\begin{bmatrix} 3.7967 \times 10^5 & -3.7963 \times 10^5 \\ -3.7963 \times 10^5 & 3.7961 \times 10^5 \end{bmatrix}$	$\begin{bmatrix} 1.37981 \\ -0.40386 \end{bmatrix}$

Table C.3: Kalman Filter (Orientation) – Final Matrices (Motion)

C.2.2 Kalman Filter

Motion	P	K
No	$\begin{bmatrix} 2.7565 \times 10^1 & 3.8802 \times 10^2 & 0 \\ 3.8802 \times 10^2 & 5.8830 \times 10^6 & 0 \\ 0 & 0 & 4.4200 \times 10^1 \end{bmatrix}$	$\begin{bmatrix} 0.96808 \\ 13.6268 \\ 0 \end{bmatrix}$
Yes	$\begin{bmatrix} 1.2812 \times 10^1 & 3.8226 \times 10^2 & 0 \\ 3.8226 \times 10^2 & 2.9739 \times 10^6 & 0 \\ 0 & 0 & 1.9825 \times 10^5 \end{bmatrix}$	$\begin{bmatrix} 0.93961 \\ 28.0352 \\ 0 \end{bmatrix}$

Table C.4: Kalman Filter (Position) – Final Matrices

The final matrices for the Kalman filter fusing IMU and UWB measurements are summarized in table C.4.

Bibliography

[1] Mike Dempsey. Indoor positioning systems in healthcare - a basic overview of technologies. White paper, Radianse, Inc., MA, USA, June 2003. URL http://web.archive.org/web/20040727133118/http://www.cimit.org/pubs/ips_in_healthcare.pdf.

[2] National Park Service. Global positioning systems - history, 2016. URL http://www.nps.gov/gis/gps/history.html. Viewed 01/02/2016.

[3] SenionLab. Indoor positioning 101, 2015. URL https://senionlab.com/indoor-positioning-101/. Viewed 02/07/2016.

[4] SenionLab. What is indoor positioning systems?, 2015. URL https://senionlab.com/indoor-positioning-system/. Viewed 02/07/2016.

[5] Yanying Gu, A. Lo, and I. Niemegeers. A survey of indoor positioning systems for wireless personal networks. *Communications Surveys Tutorials, IEEE*, 11 (1):13–32, First 2009. ISSN 1553-877X. doi: 10.1109/SURV.2009.090103.

[6] DecaWave Ltd. Real time location systems: An introduction. Technical report, DecaWave Ltd., 2014. URL http://www.decawave.com/support/download/file/nojs/449.

[7] Cisco Engineers. Multipath and diversity, January 2008. URL http://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/27147-multipath.html. Viewed 01/05/2016.

[8] Clarinox Technologies Pty Ltd. Real time location systems. Technical report, Clarinox Technologies Pty Ltd, November 2009. URL http://www.clarinox.com/docs/whitepapers/RealTime_main.pdf.

[9] Brian Gaffney. Considerations and challenges in real time locating systems design. Technical report, DecaWave Ltd., Dublin, 2008. URL http://www.decawave.com/support/download/file/nojs/347.

[10] R. Harle. A survey of indoor inertial positioning systems for pedestrians. *Communications Surveys Tutorials, IEEE*, 15(3):1281–1293, Third 2013. ISSN 1553-877X. doi: 10.1109/SURV.2012.121912.00075.

[11] L. Zwirello, C. Ascher, G.F. Trommer, and T. Zwick. Study on uwb/ins integration techniques. In *Positioning Navigation and Communication (WPNC), 2011 8th Workshop on*, pages 13–17, April 2011. doi: 10.1109/WPNC.2011.5961007.

[12] Edith Pulido Herrera, Ricardo Quirós, and Hannes Kaufmann. Analysis of a kalman approach for a pedestrian positioning system in indoor environments. In Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol, editors, *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 931–940. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74465-8. doi: 10.1007/978-3-540-74466-5_100. URL http://dx.doi.org/10.1007/978-3-540-74466-5_100.

[13] Alessio De Angelis, John Nilsson, Isaac Skog, Peter Händel, and Paolo Carbone. Indoor positioning by ultrawide band radio aided inertial navigation. *Metrology and Measurement Systems*, 17(3):447–460, 2010. doi: 10.2478/v10178-010-0038-0. URL http://dx.doi.org/10.2478/v10178-010-0038-0.

[14] C. Ascher, L. Zwirello, T. Zwick, and G. Trommer. Integrity monitoring for uwb/ins tightly coupled pedestrian indoor scenarios. In *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*, pages 1–6, Sept 2011. doi: 10.1109/IPIN.2011.6071948.

[15] P. Neto, J. Norberto Pires, and A.P. Moreira. 3-d position estimation from inertial sensing: Minimizing the error from the process of double integration of accelerations. In *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, pages 4026–4031, November 2013. doi: 10.1109/IECON.2013.6699780.

[16] Oliver J Woodman. An introduction to inertial navigation. Technical Report UCAM-CL-TR-696, University of Cambridge - Computer Laboratory, Cambridge, UK, August 2007. URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf.

[17] T. Maeda and H. Ando. Wearable robotics as a behavioral interface - the study of the parasitic humanoid. In *Wearable Computers, 2002. (ISWC 2002). Proceedings. Sixth International Symposium on*, pages 145–151, 2002. doi: 10.1109/ISWC.2002.1167236.

[18] N. Miller, O.C. Jenkins, M. Kallmann, and M.J. Mataric. Motion capture from inertial sensing for untethered humanoid teleoperation. In *Humanoid Robots, 2004 4th IEEE/RAS International Conference on*, volume 2, pages 547–565 Vol. 2, Nov 2004. doi: 10.1109/ICHR.2004.1442670.

[19] Yaqin Tao, Huosheng Hu, and Huiyu Zhou. Integration of vision and inertial sensors for 3d arm motion tracking in home-based rehabilitation. *The International Journal of Robotics Research*, 26(6):607–624, 2007. doi: 10.1177/0278364907079278. URL http://ijr.sagepub.com/content/26/6/607.abstract.

[20] *Miniature six-DOF inertial system for tracking HMDs*, volume 3362, 1998. doi: 10.1117/12.317434. URL http://dx.doi.org/10.1117/12.317434.

[21] Salvatore Sessa, Massimiliano Zecca, Zhuohua Lin, Luca Bartolomeo, Hiroyuki Ishii, and Atsuo Takanishi. A methodology for the performance evaluation of inertial measurement units. *Journal of Intelligent & Robotic Systems*, 71 (2):143–157, 2013. ISSN 0921-0296. doi: 10.1007/s10846-012-9772-8. URL http://dx.doi.org/10.1007/s10846-012-9772-8.

[22] Lauro Ojeda and Johann Borenstein. Non-gps navigation for security personnel and first responders. *The Journal of Navigation*, 60:391–407, 9 2007. ISSN 1469-7785. doi: 10.1017/S0373463307004286. URL http://journals.cambridge.org/article_S0373463307004286.

[23] B. Hartmann, N. Link, and G.F. Trommer. Indoor 3d position estimation using low-cost inertial sensors and marker-based video-tracking. In *Position Location and Navigation Symposium (PLANS), 2010 IEEE/ION*, pages 319–326, May 2010. doi: 10.1109/PLANS.2010.5507248.

[24] Hendrik Johannes Luinge. *Inertial sensing of human movement*. PhD thesis, University of Twente, 2002.

[25] Henning Lohse. Sound glove: A wireless hand motion capturing device for audio applications. Master's thesis, University of Heidelberg, Granstedt, August 2015.

[26] STMicroelectronics. Tilt measurement using a low-g 3-axis accelerometer. Application note AN3182, STMicroelectronics, April 2010. URL http://www.st.com/web/en/resource/technical/document/application_note/CD00268887.pdf.

[27] Christopher J. Fisher. Using an accelerometer for inclination sensing. Application Note AN-1057, Analog Devices, 2010. URL http://www.analog.com/media/en/technical-documentation/application-notes/AN-1057.pdf.

[28] Mark Pedley. Tilt sensing using a three-axis accelerometer. Application Note AN3461, Freescale Semiconductor, Inc., March 2013. URL https://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf.

[29] CHRobotics LLC. Understanding euler angles, 2016. URL http://www.chrobotics.com/library/understanding-euler-angles. Viewed 01/11/2016.

[30] CHRobotics LLC. Understanding quaternions, 2016. URL http://www.chrobotics.com/library/understanding-quaternions. Viewed 01/11/2016.

[31] Petelomax. Quaternion type. Wiki Entry, October 2015. URL http://rosettacode.org/wiki/Quaternion_type. Viewed 01/11/2016.

[32] confuted. Using quaternion to perform 3d rotations, 2011. URL http://www.cprogramming.com/tutorial/3d/quaternions.html. Viewed 01/11/2016.

[33] Phillip J. Schmidt. Quaternion integration of 3d rotations. Technical report, Phillip's Techology Corner, January 2014. URL https://www.dropbox.com/s/87pl9lgv4bfubne/Incremental%20Quaternion%20from%203D%20Rotational%20Vector%20Derivation.pdf?dl=0.

[34] Phillip J. Schmidt. Fast quaternion integration for attitude estimation. Technical report, Phillip's Techology Corner, September 2014. URL http://philstech.blogspot.de/2014/09/fast-quaternion-integration-for.html.

[35] CHRobotics LLC. Using accelerometers to estimate position and velocity. Application note, CHRobotics LLC, 2015. URL http://www.chrobotics.com/library/accel-position-velocity.

[36] starlino. A guide to using imu (accelerometer and gyroscope devices) in embedded applications, December 2009. URL http://www.starlino.com/imu_guide.html. Viewed 04/18/2016.

[37] InvenSense Inc. Mpu-9150 product specification revision 4.3. Technical report, InvenSense Inc., September 2013. URL http://43zrtwysvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/MPU-9150-Datasheet.pdf. Viewed 04/18/2016.

[38] Z. Kowalczuk and T. Merta. Modelling an accelerometer for robot position estimation. In *Methods and Models in Automation and Robotics (MMAR), 2014 19th International Conference On*, pages 909–914, September 2014. doi: 10.1109/MMAR.2014.6957478.

[39] Patrik Axelsson and Mikael Norrlöf. Method to estimate the position and orientation of a triaxial accelerometer mounted to an industrial manipulator. In *10th IFAC Symposium on Robot Control, Dubrovnik, Croatia, 5-7 September 2012*, pages 283–288, 2012. doi: 10.3182/20120905-3-HR-2030.00066.

[40] Mouser Electronics, Inc. Rf wireless technology, 2016. URL http://www.mouser.de/applications/rf-wireless-technology/. Viewed 01/03/2016.

[41] Osama Haraz. Why do we need ultra-wideband?, November 2012. URL http://www.vlsiegypt.com/home/?p=518. Viewed 03/31/2016.

[42] Mohammadreza Yavari and Bradford G Nickerson. Ultra wideband wireless positioning systems. Technical Report TR14-230, University of New Brunswick, Fredericton, NB, March 2014. URL http://www.cs.unb.ca/tech-reports/documents/TR14-230.pdf.

[43] Alexander Medvedev. Following the idf: Ultra wide band wireless data transfer technology, October 2002. URL http://ixbtlabs.com/articles2/uwb/.

[44] M. Kok, J.D. Hol, and T.B. Schon. Indoor positioning using ultrawideband and inertial measurements. *Vehicular Technology, IEEE Transactions on*, 64 (4):1293–1303, April 2015. ISSN 0018-9545. doi: 10.1109/TVT.2015.2396640.

[45] Antenna-Theory.com. Fractional bandwidth (fbw), 2014. URL http://www.antenna-theory.com/definitions/fractionalBW.php. Viewed 01/04/2016.

[46] Maria-Gabriella Di Benedetto and Guerino Giancola. *Understanding ultra wide band radio fundamentals*. Safari Books Online. Prentice Hall PTR, Upper Saddle River, N.J., 2004. ISBN 978-0-13-244249-7. URL http://proquest.tech.safaribooksonline.de/9780132442497.

[47] Lutz Freitag. Ultrabreitband uwb. January 2008. URL http://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2008-09_WS/S_19565_Proseminar_Technische_Informatik/freitag09ultra-wideband_slides.ppt.

[48] E. Karapistoli, F. N. Pavlidou, I. Gragopoulos, and I. Tsetsinas. An overview of the ieee 802.15.4a standard. *IEEE Communications Magazine*, 48(1):47–53, January 2010. ISSN 0163-6804. doi: 10.1109/MCOM.2010.5394030.

[49] Anne Angermann, Michael Beuschel, Martin Rau, and Ulrich Wohlfarth. *MATLAB – Simulink – Stateflow*. Oldenbourg, Munich, Germany, 5th edition, 2007.

[50] Steven Lanzisera, David Zats, and Kristofer S. J. Pister. Radio frequency time-of-flight distance measurement for low-cost wireless sensor localization. *IEEE Sensors Journal*, 11(3):837 – 845, January 2011. doi: 10.1109/JSEN.2010.2072496.

[51] DecaWave Ltd. Decawave scensor dwm1000 module. Technical report, DecaWave Ltd., 2013. URL http://www.decawave.com/support/download/file/nojs/330.

[52] S. J. Ingram, D. Harmer, and M. Quinlan. Ultrawideband indoor positioning systems and their use in emergencies. In *Position Location and Navigation Symposium, 2004. PLANS 2004*, pages 706–715, April 2004. doi: 10.1109/PLANS.2004.1309063.

[53] Giovanni Bellusci. *Ultra-Wideband Ranging for Low-Complexity Indoor Positioning Applications*. PhD thesis, Technische Universiteit Delft, January 2011. URL https://www.xsens.com/wp-content/uploads/2014/01/Thesis_PhD_Giovanni_Bellusci.pdf.

[54] Wilfried Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, October 2002. URL https://mobile.aau.at/~welmenre/papers/elmenreich_Dissertation_sensorFusionInTimeTriggeredSystems.pdf.

[55] MathWorks. Documentation, 2016. URL http://de.mathworks.com/help/. Viewed 03/24/2016.

[56] John Leis. *Digital signal processing using MATLAB for students and researchers*. Wiley, Hoboken, NJ, 2011. ISBN 978-0-470-88091-3. URL http://proquest.tech.safaribooksonline.de/9780470880913.

[57] dspGuru. Fir filter faq, 2015. URL http://dspguru.com/dsp/faqs/fir. Viewed 03/24/2016.

[58] MikroElektronika. Digital filter design, 2016. URL http://learn.mikroe.com/ebooks/digitalfilterdesign/chapter/introduction-fir-filter/. Viewed 03/24/2016.

[59] OlliW. Imu data fusing: Complementary, kalman, and mahony filter, September 2013. URL http://www.olliw.eu/2013/imu-data-fusing/. Viewed 04/04/2016.

[60] Jonathan P. How. Inertial sensors - complementary filtering - simple kalman filtering. MIT Course Number 16.333, Lecture 15, Fall 2014. URL http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-333-aircraft-stability-and-control-fall-2004/lecture-notes/lecture_15.pdf.

[61] Pieter-Jan Van de Maele. Reading a imu without kalman: The complementary filter, April 2013. URL http://www.pieter-jan.com/node/11. Viewed 03/01/2016.

[62] Francis X. Grovers. Kalman filter tutorial. Presentation at DPRG Robot Builders Night Out (RBNO). Viewed 03/24/2016., October 2013. URL https://youtu.be/18TKA-YWhX0.

[63] Kristian Sloth Lauszus. A practical approach to kalman filter and how to implement it. Blog Post, September 2012. URL http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/. Viewed 12/19/2015.

[64] José Borràs Sillero. Sensor fusion methods for indoor navigation using uwb radio aided ins/dr. Master's thesis, Polytechnic University of Catalonia, Barcelone, Spain, July 2012. URL http://www.diva-portal.se/smash/get/diva2:600448/FULLTEXT01.pdf.

[65] Makoto Tanigawa, Jeroen D Hol, Fred Dijkstra, Henk Luinge, and Per Slycke. Augmentation of low-cost gps/mems ins with uwb positioning system for seamless outdoor/indoor positioning. Technical report, Xsens Technologies, 2008. URL https://www.xsens.com/images/stories/PDF/FullPaper_Makoto_ION_GNSS_08_Final.pdf.

[66] Andreas Kugel. Robot arm - inverse kinematics. Explanation on Inverse Kinematics with a Robot Arm with five degrees of freedom, December 2014.

[67] Robot Platform. Servo control tutorial, 2016. URL http://www.robotplatform.com/knowledge/servo/servo_control_tutorial.html. Viewed 01/13/2016.

[68] DFRobot. Df metal geared 15kg standard servo 180° (dss-m15), 2016. URL http://www.dfrobot.com/index.php?route=product/product&product_id=120#.VpZath9yvCI. Viewed 01/13/2016.

[69] ServoDatabase.com. Hitec hs-422 - deluxe standard servo, 2016. URL http://www.servodatabase.com/servo/hitec/hs-422. Viewed 01/13/2016.

[70] Hitec. *Announced Specification of HS-422 STandard Deluxe Servo*. Hitec, November 2001. URL http://cdn.sparkfun.com/datasheets/Robotics/hs422-31422S.pdf.

[71] ServoCity.com. Hs-422 super sport, 2015. URL https://www.servocity.com/html/hs-422_super_sport_.html#.VxDivnV97CI. Viewed 04/15/2016.

[72] DFRobot. Goteck micro metal gear servo (2.5kg), 2016. URL http://www.dfrobot.com/index.php?route=product/product&product_id=121. Viewed 01/13/2016.

[73] dmrctoys.tk. Goteck gs9025mg micro servo metal gear motor for aircraft helicopter reviews, September 2015. URL http://dmrctoys.tk/2015/09/06/reviews-p252296/. Viewed 01/13/2016.

[74] Banggood.com. Mpu9150 9 axis accelerometer sensor module for arduino, 2016 . URL http://www.banggood.com/MPU9150-9-Axis-Accelerometer-Sensor-Module-For-Arduino-p-971657.html. Viewed 03/25/2016.

[75] InvenSense Inc. Motion driver 6.12 - features user guide. User Guide AN-EMAPPS-0.0.6, InvenSense Inc., Sunnyvale, CA, USA, May 2015.

[76] DecaWave Ltd. Dwm1000 data sheet. Technical report, DecaWave Ltd., June 2015. URL http://www.decawave.com/support/download/file/nojs/330.

[77] Digilent Inc. Zedboard zynq-7000 arm/fpga soc development board, 2016. URL http://store.digilentinc.com/zedboard-zynq-7000-arm-fpga-soc-development-board/. Viewed 01/13/2016.

[78] Avnet, Inc. Zedboard - zynq evaluation and development. Hardware User Guide Version 1.1, Avnet, Inc., August 2012.

[79] Andreas Kugel. Servo - atlys interface. Explanation on how to connect the robot arm to a Spartan-6 FPGA's user I/O, December 2015.

[80] InvenSense. Motion driver 6.12 - user guide. User Guide AN-EMAPPS-0.0.6, InvenSense, May 2015. Revision: 1.2.

[81] telos Systementwicklung GmbH. Repeated start condition, 2016. URL http://www.i2c-bus.org/repeated-start-condition/. Viewed 01/20/2016.

[82] telos Systementwicklung GmbH. Analysing obscure problems, 2016. URL http://www.i2c-bus.org/i2c-primer/analysing-obscure-problems/. Viewed 01/21/2016.

[83] ADONTEC. Xmodem protocol feature description, September 2012. URL http://www.adontec.com/xmodem_e.htm. Viewed 03/30/2016.

[84] OpenSceneGraph. The openscenegraph project website, 2016. URL http://www.openscenegraph.org/. Viewed 01/22/2016.

[85] Bill Earl. Why calibrate?, June 2015. URL https://learn.adafruit.com/downloads/pdf/calibrating-sensors.pdf.

[86] VectorNav. Importance of industrial grade sensor calibration, 2015. URL http://www.vectornav.com/support/library/calibration. Viewed 11/25/2015.

[87] luisrodenas. How to decide gyro and accelerometer offsett. forum entry, December 2013. URL http://www.i2cdevlib.com/forums/topic/91-how-to-decide-gyro-and-accelerometer-offsett/. Viewed 11/25/2015.

[88] DecaWave Ltd. Frequently asked questions - antenna, 2016. URL http://www.decawave.com/faq-page/71#t71n313. Viewed 04/04/2016.

[89] Marcus Nowotny. Filtering sensor data with a kalman filter, December 2009. URL http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/. Viewed 04/04/2016.

[90] Adam P. Taylor. How to use interrupts on the zynq soc. *Xcell Journal*, 2/2014: 38–43, May 2014. URL http://www.xilinx.com/support/documentation/xcell_articles/how-to-use-interrupts-on-zynqsoc.pdf.

[91] Miloš Petrović. Calculators :: Plane geometry :: Sine and cosine law calculator, 2016. URL http://www.mathportal.org/calculators/plane-geometry-calculators/sine-cosine-law-calculator.php. Viewed 01/19/2016.