Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 318

Entwurf und Implementierung eines drahtlosen Orientierungssensors für ein verteiltes Regelungssystem

Benjamin Schirle (2799717)

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer/in:	Dr. rer. nat. Frank Dürr
Beginn am:	11. April 2016
Beendet am:	11. Oktober 2016
CR-Nummer:	C.2.4, C.3

Kurzfassung

Verteilte Regelungssysteme haben in der Zukunft eine große Bedeutung für die sogenannten *Cyber-Physical Systems*, die sich durch die Verknüpfung von realen Objekten mit virtuellen Objekten auszeichnen. Ziel dieser Bachelorarbeit ist es, ein verteiltes Regelungssystem zu entwickeln, welches Informationen über die Lage eines Pendels liefert.

Das System soll im Allgemeinen das klassische Problem des invertierten Pendels lösen. Zum Einsatz kommen dabei das *Cross-Domain Development Kit* (XDK110) der Firma Bosch Connected Devices and Solutions GmbH, ein Raspberry Pi 2 Model B und ein umgebauter Drucker mit Schrittmotor, der eine Halterung mit Pendel an einer Schienenvorricht besitzt. Die Hauptaufgabe des Systems ist es, die vorhandenen Sensoren auf dem XDK110 zu verwenden, um eine präzise Angabe in Echtzeit über die aktuelle Lage des Pendels machen zu können. Um dies zu realisieren, wurden für das verteilte System Filteralgorithmen implementiert, die auf Basis der Sensordaten die Lage des Pendels berechnen. Zusätzlich wurden verschiedene Vorgehensweisen implementiert, um ein Ergebnis zu erhalten. Die Filterberechnungen können sowohl lokal auf dem XDK110 als auch auf dem Raspberry Pi ausgeführt werden.

Das entworfene System wurde in verschiedensten Versuchen getestet und die implementierten Filteralgorithmen, sowie die unterschiedlichen Vorgehensweisen der Berechnungen, miteinander verglichen. Das System lieferte in allen Versuchen verwertbare Aussagen über die aktuelle Lage des Pendels.

Abstract

Networked control systems are steadily gaining importance for the so-called cyber-physical systems, which are characterized by the linking of real objects with virtual objects. The goal of this Bachelor thesis is to develop a system that provides information about the position of a pendulum.

The system is generally designed to solve the classical problem of the inverted pendulum. The Bosch Cross-domain Development Kit (XDK110), the Raspberry Pi 2 Model B and the converted printer with a stepper motor which has a pendulum holder on a railing mechanism are used. The main task of the system is to use the existing sensors of the XDK110 and make a real time precise information about the position of the pendulum. To make this possible, filter algorithms were implemented for the distributed system which can calculate the position of the pendulum based on the data of the sensors. In addition various procedures were implemented to obtain a result. The filter calculations can be executed locally on the XDK110 as well as on the Raspberry Pi.

The designed system has been tested in various tests and the implemented filter algorithms, as well as the different approaches of the calculations, were compared with each other. The system provided usable information about the current position of the pendulum in all experiments.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CPS Cyber-Physical System

IMU Inertial Mesasurement Unit

IP Internet-Protokoll

IPv4 Internet Protocol Version 4

NDIS Network Driver Interface Specification

NIC Network Information Center

PNG Portable Network Graphics

SSID Service Set Identifier

TCP Transmission Control Protocol

UDP User Datagram Protocol

XDK110 Bosch Cross-Domain Development Kit

Verzeichnis der Algorithmen

1 Einleitung

1.1 Motivation

Verteilte Regelungssysteme gewinnen zunehmend an Bedeutung für die Realisierung von sogenannten Cyber-Physical Systems, bei denen physikalische Abläufe von Computern gesteuert werden. Solche Systeme beinhalten Sensoren, Aktoren und digitale Regler, die sich an unterschiedlichen Orten befinden. Da diese Objekte getrennt voneinander sind, müssen Messungen und Steuersignale über ein Kommunikationsnetzwerk übertragen werden.

Das Skynet Forschungsprojekt der Universität Stuttgart entwirft Konzepte und Methoden für verteilte Regelungssysteme, die über IP-Netzwerke betrieben werden.

Um die Leistungsfähigkeit der zu entworfenen Methoden und Systeme vergleichen zu können, muss ein Demonstrationssystem entwickelt werden, welches die Lage eines invertierten Pendels messen kann. Mittels der gewonnenen Daten ist es möglich die Informationen an einen Aktor zu senden, welcher durch geeignete mechanische Bewegungen das Pendel in einer aufrechten Position halten kann.

1.1.1 Problem des invertierten Pendels

Das allgemeine Problem des invertierten Pendels besteht aus einem beweglichen Stab mit einem Freiheitsgrad, der auf einem Wagen angebracht ist. Der darunter liegende Wagen kann sich nur in horizontaler Richtung bewegen (siehe Abbildung 1.1). Das Ziel besteht darin, den Stab in einer aufrechten Position zu halten, indem sich der Wagen nur in eine horizontale Richtung bewegt [**Sch00**].

1.2 Problemstellung

Ziel dieser Arbeit ist es, ein verteiltes System mit dem Bosch Cross-Domain Development Kit (XDK110) zu entwerfen und zu implementieren, welches ein invertiertes Pendel kontrolliert. Das System soll in der Lage sein die Lage des Pendels zu messen, um daraufhin Vorhersagen treffen zu können, in welche Richtung das Pendel fallen wird. Um die Lage eines Pendels messen zu können, benötigt ein System verschiedene Sensoren. Das XDK110 besitzt einen

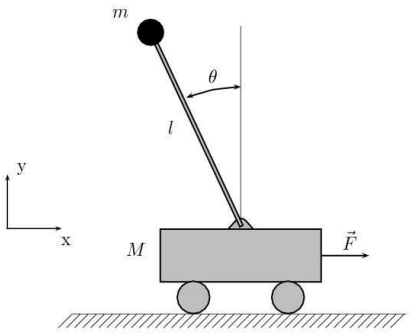


Abbildung 1.1: Das Schema eines invertierten Pendels auf einem Wagen, welcher sich nur in horizontaler Richtung bewegen kann [**inPen**].

Beschleunigungssensor (BMA280) und ein Gyroskop (BMG160), sowie eine inertiale Messeinheit (BMI160), welche einen Beschleunigungssensor und ein Gyroskop besitzt. Diese Sensoren eignen sich sehr gut für die Messung der Lage des Pendels.

Um die genaue Lage des Pendels messen zu können, soll das System alle verfügbaren Sensoren auf dem XDK110 verwenden und diese miteinander fusionieren. Dabei sollen die verschiedensten Methoden der Sensorfusion implementiert, gemessen und miteinander verglichen werden. Die Berechnungen der Sensorfusion können zum einen lokal auf dem XDK110, welcher einen ARM Cortex-M3 Mikrocontroller besitzt, oder auf einer entfernten Maschine (Raspberry Pi 2 Model B), welcher die gemessenen Rohdaten vom XDK110 mittels WiFi gesendet bekommt, erfolgen.

Weitere Details dazu befinden sich im Kapitel 3. Im Detail liefert diese Arbeit Antwort für die folgenden Aufgaben:

- Untersuchung der technischen Hintergrundinformationen, welche die Sensorfusion, die Algorithmen, die XDK Dokumentation und die Konzepte eines Echtzeitbetriebssystems (FreeRTOS, Linux) beinhaltet.
- Entwurf von Sensorfusionsalgorithmen, um die Lage des Pendels mittels verschiedenster Sensor-Rohdaten zu berechnen.
- Implementierung der entworfenen Algorithmen zum einen auf dem XDK110 (FreeRTOS), für die lokale Berechnung und zum anderen auf dem Raspberry Pi (Linux), für die Fernverarbeitung.
- Vergleich der implementierten Algorithmen.
- Vergleich der Leistungsfähigkeit der Filterberechnung, welche lokal auf dem XDK110 stattfindet und der Filterberechnung, die auf dem Raspberry Pi stattfindet.

Gliederung

Diese Bachelorarbeit ist in folgender Weise gegliedert:

Kapitel 2 – Stand der Technik: Dieses Kapitel gibt einen Überblick über den aktuellen Stand der Technik und den vorhandenen Lösungen. Es beinhaltet die verwendete Hardware, die eingesetzten Technologien, sowie die Filteralgorithmen.

Kapitel 3 – Systemmodell: Das Kapitel Systemmodell liefert einen Überblick über das gesamte System. Die vorhandenen Komponenten, ihre Beziehungen untereinander und ihre Aufgaben werden dabei vorgestellt.

Kapitel 4 – Technische Umsetzung In diesem Kapitel wird das erstellte System, welches die Lage des Pendels messen kann, gezeigt. Dabei werden die Themen Sensoren, Algorithmen, Datentransfer und die verschiedenen Ablaufsequenzen genauer betrachtet.

Kapitel 5 – Evaluierung Im Kapitel Evaluation werden die verschiedenen Lösungsmöglichkeiten in den unterschiedlichsten Versuchen analysiert und miteinander verglichen. Die Ergebnisse der einzelnen Versuche sind dort zu finden.

Kapitel 6 – Fazit und Ausblick Im letzten Kapitel befindet sich das Fazit und der Ausblick dieser Arbeit.

2 Stand der Technik

Dieses Kapitel gibt einen Überblick über den aktuellen Stand der Technik und den vorhandenen Lösungen, für die in dieser Arbeit erläuterten Ziele. Zusätzlich dazu werden die Grundlagen der verwendeten Technologien und Verfahren, um Sensoren miteinander fusionieren zu können, beschrieben.

2.1 Cyber Physical System

Cyber-Physical System (CPS) zeichnen sich dadurch aus, dass sie reale Objekte und Prozesse mit virtuellen Objekten und Prozessen über offene und jederzeit miteinander verbundenen Informationsnetzen, verknüpfen. Die Informationsnetze sind in diesem Fall dafür zuständig, die Vernetzung von physischen mit virtuellen Objekten und Prozessen über das Internet zu ermöglichen.

CPSs bieten Einsatzmöglichkeiten in den Bereichen von Smart Grids, der Vernetzung von Fahrzeugen, in der Gesundheitsbranche (E-Health), sowie in Produkt- und Produktionssystemen.

Zu den Vorteilen von Cyber Physical Systems gehören:

- Die Gewinnung und Verarbeitung von Daten (inklusive ihrer Qualität und Verfügbarkeit)
- Die Möglichkeit neue Algorithmen und Dienste, die Daten miteinander in Beziehung auswerten, zu entwerfen.
- Durchgängiger Informationsaustausch
- Schnellere Anpassung der Dienste an veränderte Märkte

CPSs müssen zuverlässig funktionieren und immer verfügbar sein. Außerdem sind sie sehr komplex und besonders das Thema IT-Sicherheit spielt eine wichtige Rolle [**VDI13**].

Abbildung 2.1: Auf der Abbildung ist das XDK110 der Firma Bosch zu sehen [**Bos15**].

2.2 Bosch XDK110

Das XDK110 ist ein Cross Domain Development Kit und ermöglicht es Prototypen für das *Internet der Dinge* einfach und schnell zu entwerfen. Der Benutzer kann mit verschiedenen Ideen experimentieren und die eingebauten Sensoren im XDK110 in Kombination mit verschiedenen Kommunikationsmöglichkeiten verwenden. In der Abbildung 2.1 ist das XDK110 zu sehen [**Bos15**].

2.2.1 Hardware

Das XDK110 besitzt einen Cortex M3 MCU 32-Bit Mikrocontroller der Silicon Labs EFM32-Familie. Dieser Mikrocontroller ist extrem energieeffizient und besonders für den Einsatz in Anwendungen geeignet, die wenig Energie verbrauchen dürfen. Die Entwickler haben Zugriff auf Trägheitssensoren und Umweltsensoren, welche im XDK110 integriert sind:

- Trägheitssensoren:
  - Beschleunigungssensor: BMA280
  - Gyroskop: BMG160
  - Magnetometer: BMM150
  - IMU: BMI160
- Umweltsensoren:
  - Luftfeuchtigkeit/Temperatur/Luftdruck Sensor: BME280
  - Umgebungslicht Sensor: MAX44009

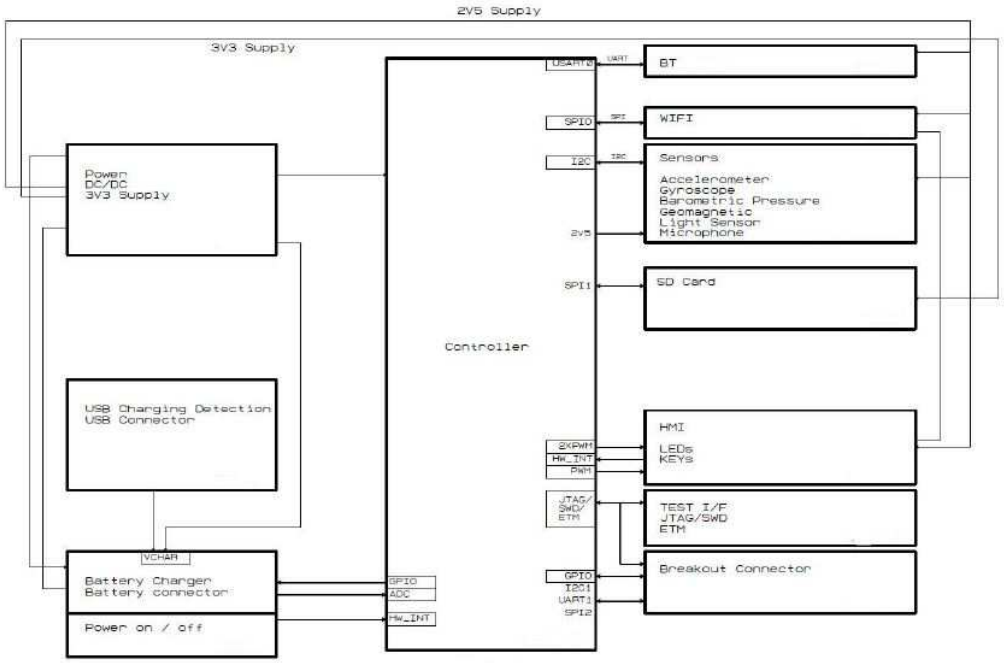


Abbildung 2.2: Das Block-Diagramm des XDK110 [**Bos15**].

– Mikrofon: AKU340

Außerdem besitzt das XDK110 einen internen Li-Ion-Akku mit einer Kapazität von 560 mAh und integrierten Antennen. Die Benutzerschnittstelle besitzt drei programmierbare Status-LEDs, zwei programmierbare Tasten, eine Micro-SD-Karte und eine J-Link-Debug Schnittstelle [**Bos15**].

Die Kommunikation des XDK110s kann kabelgebunden, über USB 2.0, oder kabellos, über Bluetooth und Wireless LAN, erfolgen. Eine detaillierte Ansicht der Hardware und ihren Verbindungen ist in der Abbildung 2.2 zu sehen.

Zusätzlich zum XDK110 selbst gibt es ein 26-poliges Extension-Board. Bei der Version 1.0 des Extension-Boards *XDK Gateway* ist der Bosch Connected Devices and Solutions GmbH ein Fehler in der Pinbelegung unterlaufen. Die korrekte Belegung ist der Abbildung 2.3 zu entnehmen [**Bosch**].

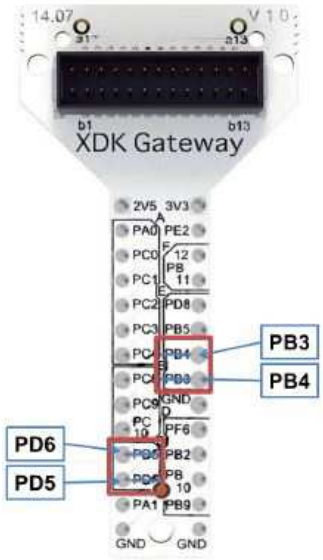


Abbildung 2.3: Das Gateway des XDK110 [**Bosch**].

2.2.2 Sensoren

Um die Lage eines Pendels berechnen zu können, werden der Beschleunigungssensor und das Gyroskop verwendet. Aus diesem Grund werden diese Sensoren im Detail vorgestellt.

Drei-Achsen-Beschleunigungssensor BMA280

Der BMA280 ist ein fortschrittlicher, ultrakleiner, dreiaxialer, Nieder-G-Beschleunigungssensor mit einer digitalen Schnittstelle, der für Anwendungen ausgelegt ist, die wenig Energie verbrauchen dürfen. Mit einer 14-Bit digitalen Auflösung erlaubt der BMA280, mit wenig Rauschen, Messungen der Beschleunigung durchzuführen. Dieser Sensor spürt Neigungen, Bewegungen, Erschütterungen und Vibrationen und kommt in Mobiltelefonen, Mensch-Maschinen Schnittstellen und Spielecontrollern zum Einsatz [**Bos15**]. Die Hauptmerkmale des BMA280 sind der Abbildung 2.4 zu entnehmen.

Drei-Achsen-Winkelgeschwindigkeitssensor BMG160

Der BMG160 ist ein ultrakleiner, digitaler Drei-Achsen-Winkelgeschwindigkeitssensor mit einem Messbereich von bis zu 2000°/s und einer digitalen Auflösung von 16 Bit. Der Sensor ermöglicht eine Messung der Winkelgeschwindigkeit mit wenig Rauschen und kommt, wie der BMA280, in Mobiltelefonen, Mensch-Maschinen Schnittstellen und Spielecontrollern zum Einsatz [**Bos15**]. Die Hauptmerkmale des BMA280 sind der Abbildung 2.5 zu entnehmen.

Specification	Value
Digital Resolution	14 bit
Resolution (in ±2 g range)	0.244 mg
Measurement Ranges (programmable)	±2 g, ±4 g, ±8 g, ±16 g
Sensitivity (calibrated)	±2 g: 4096 LSB/g ±4 g: 2048 LSB/g ±8 g: 1024 LSB/g ±16 g: 512 LSB/g
Zero-g Offset (typical, over life-time)	±50 mg
Noise Density (typical)	120 µg/√Hz
Bandwidths (programmable)	500 Hz ... 8 Hz
Current Consumption (full operation)	130 µA (@2 kHz data rate)

Abbildung 2.4: Die Hauptmerkmale des BMA280 in einer Tabelle angeordnert [**Bos15**].

Specification	Value
Digital Resolution	16 bit
Measurement Ranges (programmable)	± 125 °/s, ± 250 °/s, ± 500 °/s, ± 1000 °/s, ± 2000 °/s
Sensitivity (calibrated)	± 125 °/s: 262.4 LSB/ °/s ± 250 °/s: 131.2 LSB/ °/s ± 500 °/s: 65.5 LSB/ °/s ± 1000 °/s: 32.8 LSB/ °/s ± 2000 °/s: 16.4 LSB/ °/s
Zero-Rate Offset (typical)	± 1 °/s
Zero-Rate Offset over Temperature	0.015 °/s/K
Noise Density (typical)	0.014 °/s/√Hz
Low-Pass Filter Bandwidths (programmable)	230, 116, 64, 47, 32, 23, 12 Hz
Date Rates (programmable)	2000, 1000, 400, 200, 100 Hz
Current Consumption (full operation)	5.0 mA
Current Consumption (fast power-up)	2.5 mA

Abbildung 2.5: Die Hauptmerkmale des BMG160 in einer Tabelle angeordnert [**Bos15**].

Specification	Value
Digital Resolution	Accelerometer (A): 16 bit Gyroscope (G): 16 bit
Measurement Ranges (programmable)	(A): ± 2 g, ± 4 g, ± 8 g, ± 16 g (G): ± 125 °/s, ± 250 °/s, ± 500 °/s, ± 1000 °/s, ± 2000 °/s
Sensitivity (calibrated)	(A): ±2g: 16384 LSB/g ±4g: 8192 LSB/g ±8g: 4096 LSB/g ±16g: 2048 LSB/g (G): ±125 °/s: 262.4 LSB/ °/s ±250°/s: 131.2 LSB/ °/s ±500 °/s: 65.6 LSB/ °/s ±1000 °/s: 32.8 LSB/ °/s ±2000 °/s: 16.4 LSB/ °/s
Zero-Point Offset	(A): ±40mg (G): ± 10 °/s
Noise Density (typical)	(A): 180 µg/√Hz (G): 0.008 °/s/√Hz
Bandwidths (programmable)	1600 Hz ... 25/32 Hz

Abbildung 2.6: Die Hauptmerkmale der BMI160 in einer Tabelle angeordnert [**Bos15**].

Inertiale Messeinheit BMI160

Die BMI160 ist eine kleine 16-Bit inertiale Messeinheit, die wenig Rauschen bei den Messungen produziert und in Anwendungen wie *Augmented Reality* oder bei der Navigation in Gebäuden eingesetzt wird. Die BMI160 enthält einen 16-Bit Beschleunigungssensor, sowie ein 16-Bit Gyroskop. Besonders bei Anwendungen, die genaue Messwerte benötigen, kommt die BMI160 zum Einsatz. Die Hauptmerkmale des BMI160 sind der Abbildung 2.6 zu entnehmen.

Das XDK110 besitzt somit zwei unterschiedliche Gyroskope. Ein alleinstehendes Gyroskop BMG160 und ein Gyroskop, welches mit einem Beschleunigungssensor in der Inertial Mesasurement Unit (IMU) BMI160 kombiniert ist. Die BMI160 ist ein *Open-Loop* Gyroskop mit sehr geringem Rauschen und einem Stromverbrauch von ungefähr 1 mA [**Bos15**].

2.2.3 Software

Die XDK110 Software basiert auf dem Echtzeitbetriebssystem für eingebettete Systeme namens FreeRTOS und kann vollständig über USB betrieben werden. Das bedeutet, das XDK110 kann über USB geflasht und geladen werden. Die XDK-Workbench kompiliert immer eine komplette Anwendung, danach wird das erstellte Image über USB auf das XDK110 geflasht. Außerdem kann über USB eine Datenübertragung vom XDK110 zum Rechner, mit dem Befehl *Printf*, stattfinden.

Ein Logging-Framework ermöglicht es in bestehende Module zu schauen [**Bos15**].

In dieser Arbeit kam die XDK-Workbench mit folgenden Details zum Einsatz:

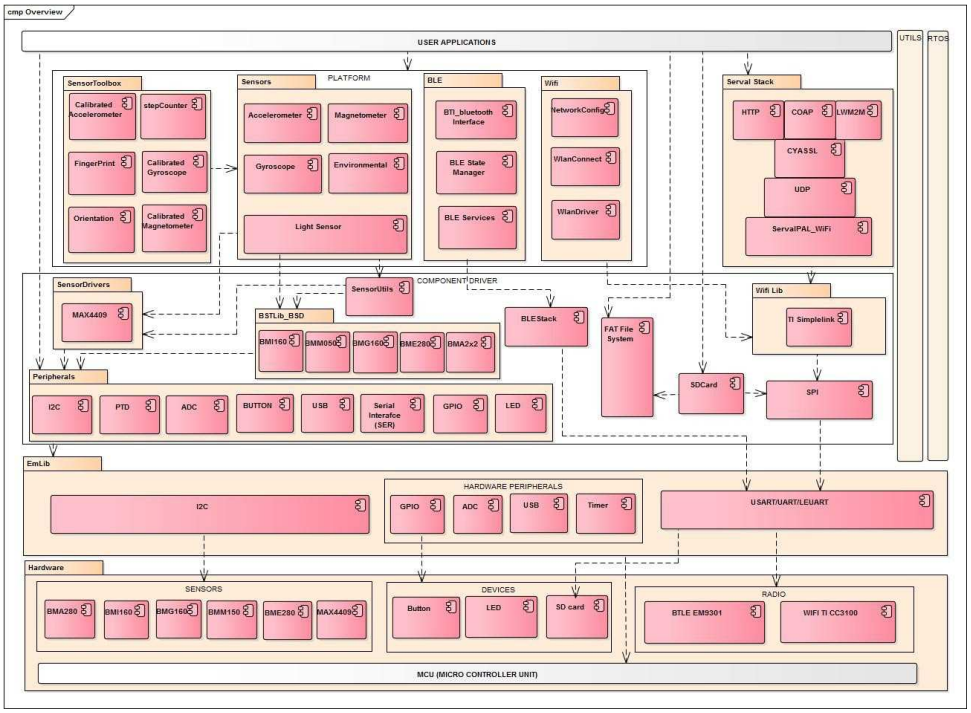


Abbildung 2.7: Zu sehen in der Abbildung ist die Architektur des XDK110 [**XDKWo**].

- Version: 1.5.1-SNAPSHOT
- Build date: 2016-03-01T17:4330Z
- Build ID: B-552

Während der vergangenen sechs Monate war es möglich, ein Update für die XDK-Workbench durchzuführen. Nachdem das Update für die XDK-Workbench durchgeführt wurde, traten beim Kompilieren der Code-Fragmente, welche von der Bosch Connected Devices and Solutions GmbH 2016 stammen, Probleme auf, die nicht behoben werden konnten. Aus diesem Grund wurde wieder die oben beschriebene Version der XDK-Workbench eingesetzt. Die einzelnen XDK-Workbench Module, sowie die Beziehungen der Module untereinander, sind in der Abbildung 2.7 dargestellt.

2.3 Raspberry Pi

Um die Filterberechnungen auf einem entfernten Rechner durchführen zu können und um die Ergebnisse der Filterberechnungen vom XDK110 empfangen zu können, kommt in dieser Arbeit der Raspberry Pi 2 Model B zum Einsatz. Der eingesetzte Raspberry Pi besitzt folgende technische Details:

- 900MHz Quad-Core ARM Cortex-A7 CPU
- 1GB RAM
- 4USB Ports
- 40 GPIO Pins
- HDMI Port
- Ethernet Port
- Mikro-SD-Karten-Aufnahme [**RaspP**]

Der Raspberry Pi nimmt in dieser Arbeit die Rolle des UDP-Servers ein und ist mit einem Ethernet-Kabel mit dem Netzwerk verbunden. Dadurch kann er UDP-Pakete von UDP-Clients empfangen und diese weiter verarbeiten.

2.4 Arduino und Schrittmotor

Damit das Pendel mit dem XDK110 in horizontaler Richtung bewegt werden kann, kommt in dieser Arbeit ein Schrittmotor, der von einem Arduino DUE gesteuert wird, zum Einsatz. Das Arduino DUE ist ein Arduino-Board, welches auf einen 32-Bit-ARM-Core-Mikrocontroller basiert. Mit seinen 54 digitalen Ein-/Ausgängen und 12 analogen Eingängen ist dieses Arduino-Board auch für größere Systeme, die mehr Leistung benötigen, ausgelegt [**ArDUE**].

Der Schrittmotor, der die gesamte Plattform in Bewegung bringt, stammt von einem Drucker. Dabei wird nicht nur der Schrittmotor des Druckers genutzt, sondern die gesamte Schienenvorrichtung. Das hat den Vorteil, dass der Schrittmotor das Pendel mit dem XDK110, perfekt in einer horizontalen Ebene bewegen kann (siehe Abbildung 5.3)

2.5 Datenübertragung

Die einzelnen Hardware-Komponenten des Systems wurden im oberen Teil dieses Kapitels beschrieben. Nun folgen die Technologien, die in diesem System verwendet werden.

Um die kabellose Datenübertragung vom XDK110 zum Raspberry Pi zu ermöglichen, wird ein Transport-Protokoll benötigt. Zur Auswahl stehen das verbindungslose Transport-Protokoll User Datagram Protocol (UDP) und das verbindungsorientierte Transport-Protokoll Transmission Control Protocol (TCP). Diese beiden Protokolle befinden sich im 5-Schichtenmodell in der *Transport-Schicht*. Das 5-Schichtenmodell ist in der Abbildung 2.8 zu sehen.

Damit die Daten versendet werden können, verpackt TCP und UDP den Datenstrom in Pakete, welche an den Empfänger, mittels des Internet-Protokoll (IP)s, an die richtige Adresse gesendet

Schicht		Funktion
5	**Anwendung** *Application*	Anwendungsbezogene Dienste (z.B. http, ftp, telnet, dns, sip...)
4	**Transport** *Transport*	Interprozesskommunikation (TCP, UDP)
3	**Vermittlung** *Network*	Kommunikation zwischen Endsystemen (z.B. IP)
2	**Sicherung** *Data Link*	Kommunikation zwischen benachbarten Knoten (z.B. Point-to-Point Protocol)
1	**Bitübertragung** *Physical*	Kommunikation von Bitströmen über physische Kanäle

Abbildung 2.8: Das 5 Schichten-Modell [**SKPrk**].

werden. Der Treiber der Netzwerkkarte bekommt vom Network Driver Interface Specification (NDIS) den Bitstrom der Pakete weitergeleitet. Vom Netzwerkkarten-Treiber werden die Daten an das Network Information Center (NIC) geschickt. Ab diesem Zeitpunkt übernimmt das Netzwerk die weiteren Aufgaben und ist dafür zuständig, dass die Daten an die richtige Adresse gesendet werden. Erhält der Empfänger die Daten des Senders, so gehen die ankommenden Daten den umgekehrten Weg, wie oben beschrieben, zum Empfänger [**OSIko**].

UDP arbeitet verbindungslos und deswegen unsicher. Der Sender bekommt, wenn er Daten versendet hat, keine Rückmeldung, dass seine Daten beim Empfänger angekommen sind. Im Gegensatz dazu bekommt der Sender bei TCP eine Empfangsbestätigung. Der Vorteil von UDP gegenüber TCP ist, dass durch das fehlen der Empfangsbestätigung der Paket-Header viel kleiner ist und die Bestätigung nicht wieder zurück an den Sender übertragen werden muss.

Im Allgemeinen hat UDP die selben Aufgaben wie TCP, nur ohne Kontrollfunktionen und ist dadurch schlanker und einfacher zu verwenden.

UDP kommt in Anwendungen und Diensten zum Einsatz, die mit Paketverlusten umgehen können bzw. für Anwendungen die nicht nur zusammenhängende, einzelne Datenpakete transportieren müssen. Außerdem kommt es zum Einsatz, wenn die Anwendungen sich selbst um das Verbindungsmanagement kümmern.

Die Nummerierung der einzelnen Datenpakete wird ebenso weggelassen, wie die Empfangsbestätigung. Deshalb ist es mit UDP nicht möglich die Datenpakete in der richtigen Reihenfolge zusammenzusetzen. Die einzelnen Datenpakete werden direkt an die Anwendung weitergeleitet.

Eine Gemeinsamkeit von UDP und TCP ist die Strukturierung der Ports. Diese Port-Strukturierung ermöglicht es Anwendungen, gleichzeitig mehrere Verbindungen über das Netzwerk zu besitzen.

Für Echtzeitanwendungen ist UDP besser geeignet, da eine kontinuierliche Datenübertragung mit TCP nur möglich ist, wenn Bestätigungspakete an den Sender zurückkommen und, weil es mit UDP kein begrenztes Verbindungsmanagement gibt [**UDPko**].

2.5.1 IPv4 - Internet Protocol Version 4

Das Internet Protokoll (IP) ist dafür zuständig, dass die Datenpakete an die richtige Adresse vermittelt werden. Es befindet sich auf der dritten Schicht des 5-Schichtenmodells (siehe Abbildung 2.8). Damit das IP die Datenpakete in einem dezentralen, verbindungslosen und paketorientierten Netzwerk richtig adressieren kann, besitzen alle Netzwerkteilnehmer eine eigene IP-Adresse.

In dieser Arbeit kommt das Internet Protocol Version 4 (IPv4) zum Einsatz. Zu den Aufgaben von IPv4 gehören:

- Logische Adressierung (IPv4-Adresse)
- IPv4-Konfiguration
- IPv4-Header
- IP-Routing
- Namensauflösung [**IP4ko**]

2.5.2 User Datagram Protocol

Da in dem entworfenen System dieser Arbeit UDP als Transportprotokoll zum Einsatz kommt, wird es in diesem Unterkapitel noch einmal im Detail vorgestellt.

UDP ist ein verbindungsloses Transportprotokoll und besitzt:

- keine zuverlässige und keine geordnete Kommunikation
- keine Flusskontrolle
- eine Adressierung von Ports
- eine Sicherung hinsichtlich Bitfehler mittels Prüfsumme [**SKPrk**]

Die einzelnen UDP-Pakete setzen sich aus einem Header- und Datenbereich zusammen. Im Header sind alle Informationen enthalten, die notwendig sind, um eine Datenübertragung zu ermöglichen [**UDPko**]. Ein UDP-Nachrichtenformat ist in der Abbildung 2.9 dargestellt. Die einzelnen Bereiche des Nachrichtenformats werden in der Abbildung erklärt.

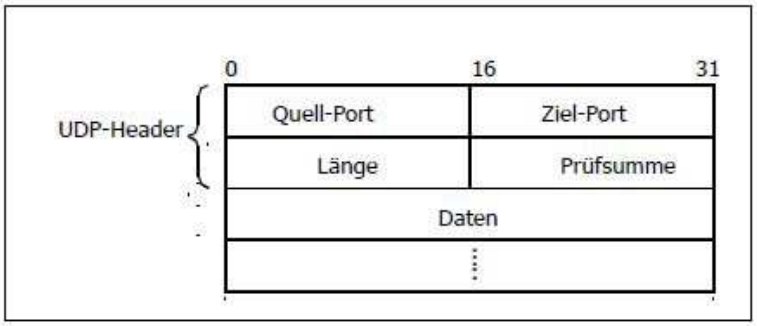


Abbildung 2.9: Dies ist die Darstellung eines UDP-Nachrichtenformats. Beschreibung der einzelnen Bereiche: *Quell-Port*: Identifikation des Senders; *Ziel-Port*: Identifikation des Empfängers; *Länge*: der Nachricht in Byte (einschließlich Header); *Prüfsumme*: über UDP-Header und UDP-Daten (Verwendung ist optional) [**SKPrk**]

2.6 Sensorfusion

Um die Orientierung eines Objektes mittels Sensoren zu bestimmen, können die unterschiedlichsten Sensoren genutzt werden. Zu den Sensoren, die sich für diese Messungen eignen, gehören:

- **Beschleunigungssensoren**, die lineare Beschleunigungen und Erdbeschleunigungen erfassen
- **Gyroskope** zur Messung von Winkelgeschwindigkeiten
- **Magnetometer** zur Messung der Kursbestimmung
- **Drucksensoren**, die den atmosphärischen Druck messen und daraus die Höhe, in der sich das Objekt befindet, ermitteln

Einfache Anwendungen, wie zum Beispiel die Freifallerkennung oder Schrittzähler, benötigen lediglich einen Beschleunigungssensor. Komplexere Anwendungen, wie etwa das Balancieren eines Roboters oder die Navigation von Flugzeugen, benötigen Daten aus mehreren Sensoren. Um die Genauigkeit, Auflösung, Stabilität und Reaktionsgeschwindigkeit zu verbessern, müssen die Rohdaten der Sensoren miteinander fusioniert werden.

Bei der Verknüpfung von Sensoren müssen die jeweiligen Vor- und Nachteile von Beschleunigungssensoren, Gyroskopen, Magnetometern und Drucksensoren, berücksichtigt werden.

Verwendet ein System ein Gyroskop in Kombination mit einem anderen Sensor, dann ist dieses System auf eine erneute Kalibrierung angewiesen, da das Gyroskop durch den Drift, mit der Zeit, seine absolute Richtungsinformation verliert.

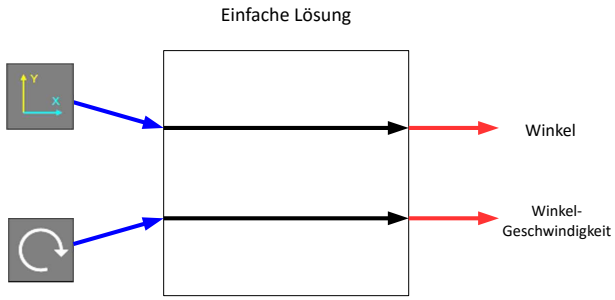


Abbildung 2.10: Zu sehen ist die einfache Lösung der Sensorfusion [**Col07**].

Ein System, welches einen Magnetometer und einen Beschleunigungssensor zur Messung der Orientierung des Objekts verwendet, kann falsche Ergebnisse liefern, da die Messungen der Sensoren, durch ferromagnetische Materialien in ihrer Umgebung, verfälscht werden können.

Werden drei verschiedene Sensoren (Beschleunigungssensor, Magnetometer und Gyroskop) in einem System miteinander fusioniert, eliminiert dies die Drift-Effekte des Gyroskops, das System kann aber immer noch durch magnetische Interferenzen beeinflusst werden.

Damit die Sensorfusion mit den gemessenen Rohdaten verwertbare Ergebnisse produziert, kompensieren Filteralgorithmen die auftretenden Probleme und liefern präzisere Ergebnisse. Muss ein Roboter das Gleichgewicht halten, so ist das System, welches sich um diese Aufgabe kümmert, mit einem Beschleunigungssensor und einem Gyroskop ausgestattet. Befindet sich das System in Ruhe, so kann der Beschleunigungssensor präzise Informationen über den Neigungswinkel liefern. Sobald sich der Roboter bewegt, kann der Beschleunigungssensor den schnellen Bewegungen nicht mehr folgen und benötigt die Hilfe des Gyroskops. Das Gyroskop ist in der Lage bei Rotations- oder Translationsbewegungen, dynamische Winkelgeschwindigkeitsinformationen zu liefern. Eine Winkelverschiebung bzw. der Neigungswinkel lassen sich durch einfache Integration der Daten über die Zeit berechnen. Dennoch sind die Ergebnisse mit Vorsicht zu genießen, da der Fehler dieser Winkelinformationen wegen des Driftes des Gyroskops mit der Zeit weiter zu nimmt [**ENX11**].

2.6.1 Einfache Lösung

Die erste Lösungsmöglichkeit, die Sensorfusion durchzuführen, verwendet einen Beschleunigungssensor und ein Gyroskop. Beide Sensordaten, die von den Sensoren geliefert werden,

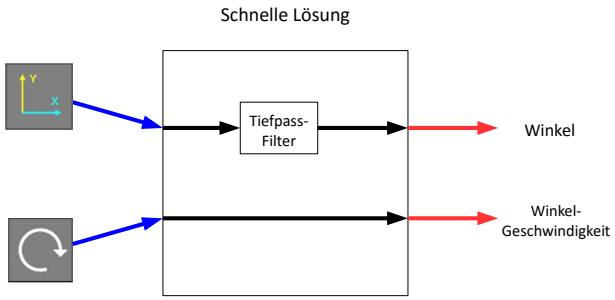


Abbildung 2.11: Die schnelle Lösung der Sensorfusion [**Col07**].

Algorithmus 2.1 Tiefpass-Filter für die schnelle Lösung der Sensorfusion

```
angle = (0.75) * (angle) + (0.25) * (xAcc);
```

werden ohne Filterberechnungen direkt für die Winkelberechnung und die Winkelgeschwindigkeit genutzt, wie in Abbildung 2.10 zu sehen ist. Dieses Vorgehen hat den Vorteil, dass es intuitiv und einfach umzusetzen ist. Ein weiterer Vorteil dieser Lösung ist, dass das Gyroskop eine genaue Winkelgeschwindigkeitsmessung ermöglicht.

Ein Nachteil dieser Lösung ist, dass es rauscht und der Beschleunigungssensor jede horizontale Beschleunigung misst. Das heißt, wenn das System horizontal ausgerichtet ist und die Motoren eine kontinuierliche Vorwärts-Bewegung erzeugen, dann wird der Beschleunigungssensor diese Beschleunigung nicht von der Schwerkraft unterscheiden können [**Col07**].

2.6.2 Schnelle Lösung

Eine weitere einfache Lösung verwendet ebenfalls einen Beschleunigungssensor und ein Gyroskop, siehe Abbildung 2.11. Die Berechnung der Winkelgeschwindigkeit erfolgt wie im vorherigen Beispiel; die Berechnung für den Winkel jedoch, erhält einen Tiefpassfilter.

Die Konstanten 0.75 und 0.25 sind Beispielwerte. Diese können, je nach Zeitkonstanten des Filters, verändert werden.

Auch diese Lösung ist, wie die *Einfache Lösung*, intuitiv und einfach umzusetzen. Ein Vorteil gegenüber der anderen Lösung ist, dass in der einfachen Lösung ein Filter zum Einsatz kommt, der kurzzeitige horizontale Beschleunigungen herausfiltern kann. Langzeit-Beschleunigungen, wie die Schwerkraft, passieren den Filter jedoch.

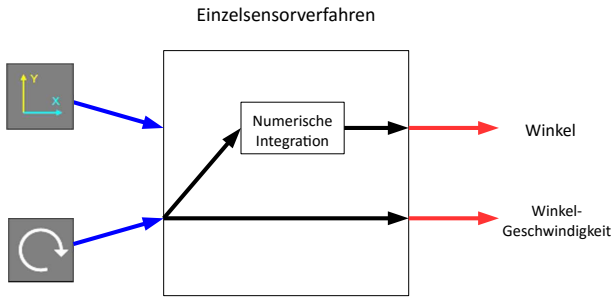


Abbildung 2.12: Das Einzelsensorverfahren der Sensorfusion [**Col07**].

Algorithmus 2.2 Numerische Integration für das Einzelsensorverfahren

```
angle = angle + gyro * dt;
```

Der Nachteil dieser Lösung ist, dass die Winkelmessung sich durch die Mittelung verzögert und Verzögerungen sind im Allgemeinen schlecht für Systeme, die nicht kippen dürfen [**Col07**].

2.6.3 Einzelsensorverfahren

Das Einzelsensorverfahren benutzt im Gegensatz zu den vorherigen beiden Lösungen lediglich ein Gyroskop um den Winkel und die Winkelgeschwindigkeit zu berechnen, siehe Abbildung 2.12. Der Beschleunigungssensor wird in dieser Lösung nicht genutzt. Ein Algorithmus, der dieses Verfahren umsetzt, ist beim Algorithmus 2.2 zu sehen. Zu beachten ist dabei, dass der Algorithmus voraussetzt, dass das Zeitintervall der Updates bekannt ist.

Der Vorteil dieser Lösung ist, dass nur ein Sensor benötigt wird, um den Winkel und die Winkelgeschwindigkeit zu berechnen. Außerdem ist diese Lösung, im Vergleich zu den vorherigen Lösungen schneller, da Verzögerungen keine Probleme bereiten. Zusätzlich dazu unterliegt dieses Verfahren nicht der horizontalen Beschleunigung und ist einfach zu implementieren.

Ein Nachteil ist, dass das Gyroskop driftet. Wenn das Gyroskop im unbewegten Zustand nicht perfekt Null misst, dann wird dieser Messfehler in der Berechnung dazu addiert bis das Ergebnis irgendwann so weit weg vom Ist-Zustand ist, sodass die Ergebnisse nicht mehr zu gebrauchen sind. Das Problem dabei ist, dass Gyroskope im unbewegten Zustand nie genau Null messen [**Col07**].

Algorithmus 2.3 Komplementär-Filter (Prinzip)

```
angle = (0.98) * (angle + gyro * dt) + (0.02) * (xAcc);
```

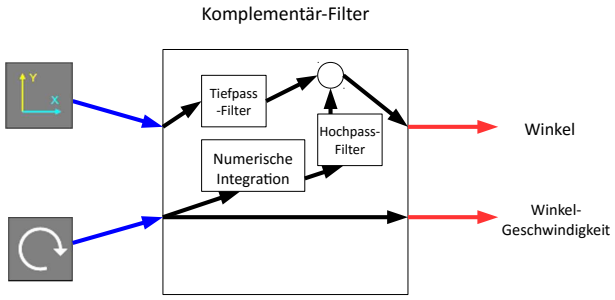


Abbildung 2.13: Das Verfahren des Komplementär-Filters [**Col07**].

Die Filteralgorithmen (Komplementär-Filter und Kalman-Filter), die im entworfenen System zum Einsatz kamen, werden nun im Folgenden im Detail beschrieben.

2.6.4 Komplementär-Filter

Der Komplementär-Filter verwendet, um den Winkel und die Winkelgeschwindigkeit zu berechnen, einen Beschleunigungssensor und ein Gyroskop. Zusätzlich dazu kommen ein Tiefpass-Filter, ein Hochpass-Filter und die numerische Integration zum Einsatz. Der Komplementär-Filter mit allen anderen integrierten Filtern ist in der Abbildung 2.13 zusehen.

Der Programmcode der den Winkel berechnet ist dem Algorithmus 2.3 zu entnehmen.

Der Nachteil dieses Filters ist, dass er mehr Theorie voraussetzt als die vorherigen Lösungen. Im Vergleich zum Kalman-Filter ist er dennoch einfacher zu verstehen [**Col07**].

Die einzelnen Filter, die im Komplementär-Filter verwendet werden, werden in den folgenden Unterkapiteln beschrieben.

Tiefpass-Filter:

Das Ziel des Tiefpass-Filters ist es, dass nur die Langzeit-Änderungen den Filter passieren. Die kurzzeitigen Schwankungen dagegen sollen herausgefiltert werden.

Algorithmus 2.4 Tiefpass-Filter des Komplementär-Filters

```
angle = (0.98) * (angle) + (0.02) * (xAcc);
```

Eine Möglichkeit dieses Ergebnis zu erzielen ist, nach und nach in der Programmschleife die Änderungen zu erzwingen [**Col07**]. Bezogen auf einen Programmcode sieht dies wie beim Algorithmus 2.4 aus.

Hochpass-Filter:

Die Theorie hinter dem Hochpass-Filter ist komplizierter als beim Tiefpass-Filter, doch konzeptionell macht der Hochpass-Filter genau das Gegenteil vom Tiefpass-Filter.

Der Hochpass-Filter filtert die Signale heraus, die konstant über die Zeit sind, während die Kurzzeit-Änderungen den Filter passieren. Dies kann genutzt werden, um den Drift aufzuheben [**Col07**].

Komplementär-Filter:

Im Grunde ist der Komplementär-Filter die Kombination aus verschiedensten Filterberechnungen und ermöglicht dadurch eine präzise und lineare Schätzung.

Wird der Programmcode des Komplementär-Filters genauer betrachtet, so ergeben sich drei Aufteilungen:

1. *((0.02) * (xAcc))* Abschnitt für den Tiefpass-Filter, welcher mit dem Beschleunigungssensor arbeitet.
2. *(angle + gyro * dt)* Abschnitt, welcher die Integration darstellt.
3. ((0.98) * (angle + gyro * dt)) Abschnitt für den Hochpass-Filter, der mit der integrierten Gyro-Winkelschätzung arbeitet. Dieser Abschnitt hat ungefähr die gleiche Zeitkonstante wie der Tiefpass-Filter.

Wenn dieser Filter in einer Schleife, 100 mal pro Sekunde, ausgeführt wird, dann wäre die Zeitkonstante sowohl für den Tiefpass- als auch Hochpass-Filter die Folgende:

$$Zeitkonstante = \frac{a \cdot dt}{1 - a} = \frac{0.98 \cdot 0.01sec}{0.02} = 0.49sec$$

Für Zeitperioden, die kürzer als 0.5sec sind, hat das Integrieren des Gyroskops den Vorrang und die rauschenden horizontalen Beschleunigungen werden herausgefiltert. Dauert die Zeitperiode länger als 0.5sec wird der Beschleunigungssensor mehr gewichtet als das Gyroskop und fängt ab diesem Punkt an zu driften.

Algorithmus 2.5 Komplementär-Filter 2 (Prinzip)

```
// Quelle: robottini . altervista .org/tag/kalman-filter
y1 = ((xAcc - angle) * 20) + y1;
angle = (y1 + (xAcc - angle) * 20 + xGyr) + (angle);
```

Der Komplementär-Filter:

- reagiert bei Winkelberechnungen schnell, ist präzise und unempfindlich gegenüber horizontalen Beschleunigungen oder einem Gyroskop-Drift. Dadurch reduziert sich das Rauschen, sowie der Drift.
- ist Mikroprozessor freundlich, d.h. er benötigt wenig Rechenleistung.
- berechnet den Winkel mit einer geringeren Verzögerung als ein Tiefpass-Filter alleine und ist dadurch schneller.
- erfordert eine kleine Anzahl von Gleitkommazahl-Operationen.
- kann ohne Probleme in Schleifen von über 100Hz ausgeführt werden.
- ist intuitiv und im Vergleich zum Kalman-Filter einfacher zu implementieren und zu verstehen. Gegenüber den vorherigen Lösungen benötigt der Komplementär-Filter ein tieferes Verständnis der Theorie [**Col07**].

Für den Komplementär-Filter existiert eine zweite Variante. Diese Alternativlösung des Komplementär-Filters ist im Algorithmus 2.5 zu sehen.

2.6.5 Kalman-Filter

Der Kalman-Filter wurde in den 1950er Jahren von Dr. Rudolf E. Kalman erfunden und kam seit seiner Einführung 1960 in zahlreichen Anwendungen zum Einsatz. Der Kalman-Filter wurde in den unterschiedlichsten Anwendungen eingesetzt. Er eignet sich zur Positionsbestimmung von Objekten und neben den normalen GPS-Empfängern wurde der Filter auch in Anwendungen verwendet, die GPS- und Trägheitsnavigation kombinieren und somit ein inertialen Navigationssystem bilden [**ENX11**].

Der Kalman-Filter ist vermutlich der ideale Filter für die Kombination von Sensoren, bei der das Ergebnis präzise sein muss, obwohl die kombinierten Sensoren rauschen und driften und dadurch das Ergebnis verfälschen können. Der Nachteil des Kalman-Filters ist, dass er im Vergleich zu den anderen Filter-Methoden sehr komplex und rechenintensiv ist [**Col07**].

Um den Kalman-Filter umzusetzen, wird ein Modell des Systems erstellt. Dieses Modell beinhaltet einen internen Zustand, welcher beobachtet werden kann und durch einen gemessenen Zustand des Systems korrigiert wird. Es wird also eine Vorhersage über den nächsten Zustand des Systems erstellt, die dann mit der Messung des Systems kombiniert wird. Dabei werden je

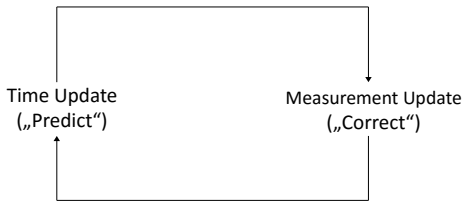


Abbildung 2.14: In der Abbildung ist der Kreislauf des Kalman-Filters zu sehen [**Bes12**].

nach System- und Messrauschen die einzelnen Gewichtungen variiert. Dies hat den Effekt, dass die Bestimmung des Zustandes des Systems präziser ist, als bei einer reinen Messung.

Um den Winkel und die Winkelgeschwindigkeit eines Systems berechnen zu können, verwendet auch der Kalman-Filter einen Beschleunigungssensor sowie ein Gyroskop. Im folgenden wird die allgemeine Funktionsweise des diskreten Kalman-Filters beschrieben, die eigentliche Implementierung befindet sich in Kapitel 4.

Der Kalman-Filter kann auch als Vorhersage-Korrektur Algorithmus bezeichnet werden, siehe Abbildung 2.14. Wie oben beschrieben, gibt es eine Vorhersage (a-priori) für das Verhalten des Systems zum nächsten Zeitpunkt. Diese Vorhersage wird dann durch eine Messung korrigiert (a-postpriori). Im nächsten Zeitschritt wird die a-postpriori Schätzung die neue a-priori Schätzung.

Das System besitzt zu einer Zeit k einen:

- beobachteten Zustand y_k
- internen Zustand x_k

Beide Systemzustände y_k und x_k sind Zufallsvariablen im $\mathbb{R}^n$ (n ist die Anzahl der Zustände und k ist der Zeitindex, der den Zeitpunkt einer Messung oder eines Zustandes angibt).

Als erstes wird die Zustandsgleichung des Kalman-Filters betrachtet:

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{2.1}$$

A = beschreibt den Übergang vom Zeitschritt k zu $k+1$ und ist eine $n \times n$-Matrix
B = beschreibt die Relation zwischen dem Zustand x_k und dem Systeminput u_k und ist eine $n \times l$-Matrix
x_k = ist der aktuelle Zustand, welcher alle Informationen darüber enthält
u_k = Systeminput
w_k = Systemrauschen (Da das Systemmodell nicht perfekt ist wird das Systemrauschen w_k dazu addiert)

Als zweites wird die Output-Gleichung betrachtet:

(2.2) $y_k = Cx_k + z_k$

mit:

y_k = ist die Messung, die man von den Sensoren erhält und somit der gemessene Zustand des Systems
C = setzt die Messung mit dem Zustand des Systems in Relation und ist eine $m \times l$-Matrix
z_k = ist das Messrauschen, welches ebenfalls wie x_k dazu addiert wird, da die Information, welche die Messung liefert, nicht korrekt ist

Im Allgemeinen ist es so, dass die Variablen des additiven Rauschens w_k und z_k zu jedem Zeitpunkt k unabhängige Zufallsvariablen sind. Deshalb kann man die zugehörigen Prozessrauschen-Kovarianzmatrix S_w und Messrauschen-Kovarianzmatrix S_z, mit Erwartungswert E folgendermaßen definieren:

(2.3) $S_w = E(w_k w_k^T)$

(2.4) $S_z = E(z_k z_k^T)$

Die Fehlerkovarianz Matrizen setzen sich aus einer a-priori Zustandsschätzung (ohne Messung) und einer a-posteriori Zustandsschätzung zum Zeitpunkt k zusammen. Die Differenz zwischen dem wahren und dem geschätzten Zustand ist der Fehler der jeweiligen Abschätzung $\hat{x}_k^*$. Siehe für die a-priori Abschätzung die Gleichung 2.5 und für die a-posteriori Abschätzung die Gleichung 2.6.

(2.5) $e_z^* = x_k - \hat{x}_k^*$

(2.6) $e_k = x_k - \hat{x_k}$

Die Varianz der Zustandsverteilung wird durch die a-posteriori Fehlerkovarianz 2.7 beschrieben:

(2.7) $p(x_k|z_k) = N(E[x_k], E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]) = N(\hat{x}_k, P_k)$

Der Messfehler ist gaussverteilt mit Erwartungswert 0 und das Prozessrauschen beschreibt die Abweichung des modellierten Systems vom realen Systemzustand.

Der Kalman-Filter kann mit den folgenden drei Gleichungen (2.8, 2.9, 2.10) beschrieben werden:

(2.8) $K_k = AP_kC^T(CP_kC^T + S_z)^{-1}$

(2.9) $\hat{x}_{k+1} = (A\hat{x}_k + Bu_k) + K_k(y_{k+1} - C\hat{x}_k)$

(2.10) $P_{k+1} = AP_kA^T + S_w - AP_kC^TS_z^{-1}CP_kA^T$

mit:

K_k	= ist der Kalman-Gain (die Summe der Varianzen des Fehlers soll minimal sein)
$(y_{k+1} - C\hat{x}_k)$	= ist die Diskrepanz zwischen Vorhersage und wirklicher Messung.
P	= ist die Fehlerkovarianzmatrix, die die Varianz des Systemzustands und die Kovarianz zwischen den Elementen des Zustandsvektors beschreibt.

Der Kalman-Gain K_k und der gemessene Output y_kk wird der Beschreibung des Systemzustandes $\hat{x_k}$ hinzugefügt, siehe 2.8. Der erwartete Wert ist gleich dem wahren Wert des Zustandes, wobei die Zustandsschätzung so wenig wie möglich vom wahren Zustand variieren soll. Wird der Kalman-Gain K_k klein, so ist die Kovarianzmatrix S_z, das Messrauschen groß, und y_{k+1} wird in der Zustandsschätzung klein, siehe 2.9. Wird der Kalman-Gain dagegen größer, so ist das Messrauschen klein. In der Gleichung 2.10 beschreibt P_{k+1} die geschätzte Fehlerkovarianzmatrix des nächsten Zeitschritts.

Problematisch bei den Gleichungen 2.8 und 2.10 ist die Rechenzeit bei der Berechnung des Filters. Bei einem System steigt die Rechenzeit ab drei Zuständen proportional zu n^3 mit einer Matrixgröße von n [**Bes12**].

3 Systemmodell

Im Folgenden wird das Systemmodell dieser Arbeit vorgestellt. Das Systemmodell gibt einen allgemeinen Überblick über das gesamte System, über die enthaltenen Komponenten und ihre Beziehungen untereinander und über die unterschiedlichen Varianten des Systems.

Das Basis-System besteht aus den folgenden Komponenten:

- UDP-Server (Raspberry Pi)
- UDP-Client (XDK110)
- Datenübertragung (UDP - WiFi)

Die Basis-Komponenten des Systems sind in der Abbildung 3.1 zu sehen. Das Netzwerk, welches dafür zuständig ist, dass ein UDP-Client Daten an einen UDP-Server senden kann, wird dabei von einem Wireless Router, welcher im Unterkapitel 3.1 genauer beschrieben ist, bereitgestellt.

Das Grundprinzip des Systems besteht darin, mit den Komponenten ein verteiltes System zu erstellen, welches Informationen über die Lage des Pendels liefern kann. Da dieses System in Echtzeit arbeiten soll, muss zu jeder Zeit bekannt sein, wie die Lage des Pendels ist. Die gewonnenen Informationen über das Pendel müssen am Ende dem UDP-Server zur Verfügung stehen, damit dieser weitere Schritte einleiten kann, um eventuell einem Schrittmotor mitteilen zu können, dass dieser die Halterung bewegen muss.

Um die Lage eines Pendels messen bzw. berechnen zu können, muss das System Informationen über das Pendel erhalten. Diese Informationen bekommt das System durch das XDK110 geliefert, welches an dem Pendel angebracht ist. Das XDK110 hat Zugriff auf verschiedenste Sensoren und verwendet einen Beschleunigungssensors und ein Gyroskop, um die Lage des Pendels berechnen zu können. Zu den Aufgaben des XDK110s gehören nicht nur das Auslesen der Sensor-Rohdaten, sondern auch das Senden von Daten an einen UDP-Server. Das XDK110 übernimmt deshalb zusätzlich zu der Hauptaufgabe noch die Aufgabe eines UDP-Clients, welcher alle notwendigen Informationen an einen UDP-Server senden muss. Der UDP-Server hat im Allgemeinen die Aufgabe, die gesendeten Daten vom UDP-Client zu empfangen und in einer CSV-Datei aufzuzeichnen.

Damit die Informationen über die Lage des Pendels präzise sind, können die Sensor-Rohdaten nicht direkt verwendet werden. Deshalb findet eine Filterberechnung statt, die auf Basis der Sensor-Rohdaten, präzise Informationen über die aktuelle Lage des Pendels liefert. Ist

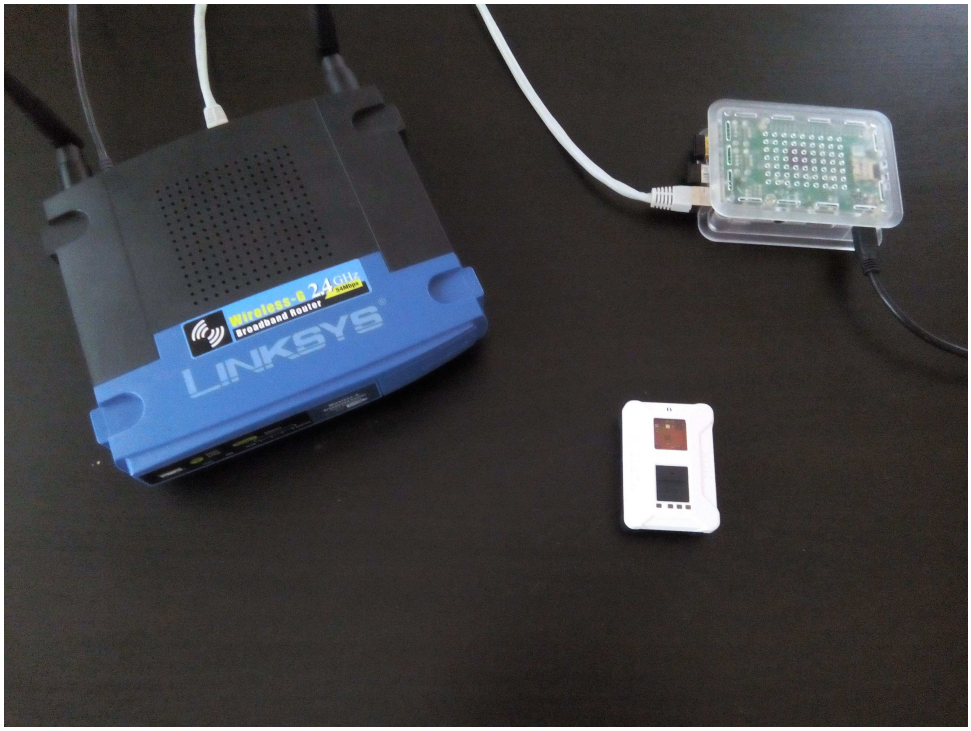


Abbildung 3.1: Die Basis-Komponenten des Systems.

das System an diesem Punkt angelangt, gibt es zwei Möglichkeiten die Filterberechnungen ausführen zu lassen. Die erste Möglichkeit ist, die Berechnungen lokal auf dem XDK110 durchführen zu lassen, welches dann die Ergebnisse an den Raspberry Pi sendet. Bei der zweiten Möglichkeit empfängt der Raspberry Pi die Sensor-Rohdaten vom XDK110 und lässt dann die Filterberechnungen durchführen.

Zur Auswahl stehen zwei Szenarien, die die gleiche Aufgabe mit unterschiedlichen Vorgehensweisen lösen. Damit die Unterschiede der beiden Vorgehensweisen deutlich werden, folgen nun die einzelnen Schritte der beiden Abläufe.

Die erste Variante, welches die Filterberechnungen lokal auf dem XDK110 durchführt, durchläuft die folgenden Schritte:

- Auslesen der Sensor-Rohdaten (XDK110)
- Filterberechnungen (XDK110)
- Senden der Ergebnisse an den UDP-Server (XDK110)
- Empfang der Ergebnisse (Raspberry Pi)
- Aufzeichnen der Daten in einer CSV-Datei (Raspberry Pi)

Die zweite Variante, welches die Filterberechnungen auf dem Raspberry Pi durchführt, durchläuft die folgenden Schritte:

- Auslesen der Sensor-Rohdaten (XDK110)
- Senden der Daten an den UDP-Server (XDK110)
- Empfang der Daten (Raspberry Pi)
- Filterberechnung (Raspberry Pi)
- Aufzeichnen der Daten in einer CSV-Datei (Raspberry Pi)

Da es viele verschiedene Filterberechnungen gibt, die jeweils Vor- und Nachteile besitzen, werden im System unterschiedliche Filterberechnungen umgesetzt, damit es möglich ist, diese miteinander vergleichen zu können.

Um die einzelnen Versuche, die im später folgenden Kapitel 5 vorgestellt werden, vollständig durchführen zu können, werden die Basis-Komponenten von einem umgebauten Drucker ergänzt. Dieser Drucker besitzt einen Schrittmotor und eine Halterung mit einem Pendel, an welches das XDK110 angeschraubt werden kann.

Die speziellen Details des XDK110s und die integrierten Sensoren wurden im Kapitel 2 ausführlich beschrieben. Ebenso wurden die technischen Details des Raspberry Pis und die Theorie, die für die Filterberechnungen notwendig sind, im Kapitel 2 beschrieben.

3.1 Linksys

Damit UDP-Clients an einen UDP-Server Daten senden können, benötigt das gesamte System ein zuverlässiges Netzwerk. Dieses Netzwerk wird durch einen Wireless-G Broadband-Router von der Firma Linksys (A Division of Cisco Systems, Inc.) mit der Modell-Nummer WRT54GL bereitgestellt.

Dieser Linksys Router ist ein kompakter Internet-Sharing-Router und besitzt einen Switch mit vier Ports, sowie einen Wireless-G (802.11g) Access Point mit 54 Mbit/s. Im Allgemeinen ist dieser Router für die gemeinsame Nutzung einer Internetverbindung und anderer Ressourcen mit verdrahteten Ethernet-, Wireless-G- und Wireless-B-Geräten zuständig. In dieser Arbeit ist er dafür zuständig, dass das XDK110 eine kabellose Verbindung mit dem Raspberry Pi aufnehmen kann.

Weitere Funktionen des Linksys-Routers sind:

- TKIP und AES Verschlüsselung
- Wireless-MAC-Adressfilterung
- SPI-Firewall
- SecureEasySetup-Funktion (für eine einfache und sichere Wireless-Konfiguration auf Tastendruck)

4 Technische Umsetzung

Ziel dieser Arbeit ist es ein System zu entwerfen, welches das Problem des invertierten Pendels löst. Dabei wird das XDK110 an ein Pendel angebracht, welches an einer Halterung befestigt ist. Die Halterung selbst kann über einen Schrittmotor in horizontaler Richtung bewegt werden. Die komplette Montage von XDK110, Pendel, Halterung und Schrittmotor ist in der Abbildung 5.3 zu sehen. Anzumerken ist hierbei, dass die Halterung des XDK110s frei schwingen kann. Um solch ein System erstellen zu können, sind bereits im vorherigen Kapitel 2 alle dafür benötigten Informationen genannt worden.

Am Ende dieser Arbeit soll es ein System mit zwei unterschiedlichen Varianten geben. Eines berechnet die Lage des Pendels auf einem entfernten Rechner (Raspberry Pi). Das anderen berechnet die Lage des Pendels auf dem XDK110 lokal und sendet die Ergebnisse an den entfernten Rechner.

In diesem Kapitel wird die eigentliche Umsetzung der Systeme vorgestellt. Es wird dabei auf die wichtigsten Aspekte der Implementierungen der einzelnen Filteralgorithmen eingegangen, sowie auf die Datenübertragung, die eine wichtige Rolle in einem Echtzeitsystem spielt. Am Ende dieses Kapitels werden die verschiedenen Abläufe der Systeme vorgestellt. Ein allgemeiner Überblick über das gesamte System wurde im vorherigen Kapitel 3 vorgestellt.

4.1 Sensoren

Um die Lage des XDK110s berechnen zu können, werden im vorgestellten System Beschleunigungssensoren und Gyroskope zum Einsatz kommen. Dabei gibt es zwei unterschiedliche Möglichkeiten die Rohdaten der Sensoren zu erhalten. Die erste Möglichkeit verwendet, um die Rohdaten zu erhalten, den Beschleunigungssensor BMA280 und das Gyroskop BMG160. Die zweite Variante verwendet die inertiale Messeinehit BMI160, welche einen Beschleunigungssensor und ein Gyroskop integriert hat.

Zusätzlich dazu bietet das XDK110 noch eine interne Lösung an, um die Lage des XDK110s auslesen zu können. Um einen Vergleich der verschiedenen Lösungsmöglichkeiten für die Berechnungen zu haben, wird die interne Lösung ebenfalls im System verwendet. Die Analyse und Vergleich der einzelnen Lösungsmöglichkeiten befindet sich im Kapitel 5.

Algorithmus 4.1 Initialisierung der Sensoren

```
Retcode_T returnValue = RETCODE_FAILURE;

/* initialize calibrated accelerator (BMA280)*/
returnValue = RETCODE_FAILURE;
returnValue = CalibratedAccel_init (xdkCalibratedAccelerometer_Handle);
if ( RETCODE_OK != returnValue) {
        printf ("Calibrated Accelerator initialization failed");
}

/* initialize accelerator BMI160*/
returnValue = RETCODE_FAILURE;
returnValue = Accelerometer_init (xdkAccelerometers_BMI160_Handle);
if ( RETCODE_OK != returnValue) {
        printf ("BMI160 Accelerator initialization failed");
}

/* initialize calibrated gyroscope (BMG160)*/
returnValue = RETCODE_FAILURE;
returnValue = CalibratedGyro_init (xdkCalibratedGyroscope_Handle);
if ( RETCODE_OK != returnValue) {
        printf ("Calibrated Gyroscope initialization failed");
}

/* initialize gyroscope BMI160*/
returnValue = RETCODE_FAILURE;
returnValue = Gyroscope_init(xdkGyroscope_BMI160_Handle);
if ( RETCODE_OK != returnValue) {
        printf ("BMI160 Gyroscope initialization failed");
}

/* initialize orientation sensor */
returnValue = RETCODE_FAILURE;
returnValue = Orientation_init (xdkOrientationSensor_Handle);
if ( RETCODE_OK != returnValue) {
        printf ("Orientation initialization failed");
}
```

4.1.1 Initialisierung der Sensoren

Der Zugriff auf die einzelnen Sensoren findet über den *XdkSensorHandle* statt. Dieser „Handler“ ermöglicht es, direkt die jeweiligen Sensoren, mit einem spezifischen Aufruf für den jeweiligen Sensor, zu initialisieren und die Sensordaten abzufragen. Nach der Initialisierung der einzelnen Sensoren auf dem XDK110 liefert der *XdkSensorHandle* als Rückgabe einen Wert, der den Fehler-Code repräsentiert. Mit diesem Wert ist es möglich nach der Initialisierung des Sensors zu überprüfen, ob diese funktioniert hat.

Im Programmabschnitt 4.1 ist die Initialisierung aller verwendeten Sensoren implementiert.

Algorithmus 4.2 Auslesen der Sensordaten des BMG160

```
Retcode_T returnValue = RETCODE_FAILURE;

// Accelerometer G_Value:
CalibratedAccel_XyzGData_T calibAccelGData;
returnValue = CalibratedAccel_readXyzGValue(&calibAccelGData);

float accXAxisDataGValue = 0;
float accYAxisDataGValue = 0;
float accZAxisDataGValue = 0;

if (RETCODE_OK == returnValue) {
        accXAxisDataGValue = (float) calibAccelGData.xAxisData;
        accYAxisDataGValue = (float) calibAccelGData.yAxisData;
        accZAxisDataGValue = (float) calibAccelGData.zAxisData;
}
```

4.1.2 Auslesen der Sensoren

Nach der erfolgreichen Initialisierung der Sensoren können diese, um Sensordaten zu erhalten, angesprochen werden. Die Sensoren liefern je nach Rückgabetyp unterschiedliche Werte.

Der BMA280 liefert drei verschiedene Einheiten als Rückgabewerte:

- LSB-XYZ-Einheit
- Mps-XYZ-Einheit
- G-XYZ-Werte

Das Gyroskop BMG160 liefert ebenfalls drei verschiedene Einheiten als Rückgabewerte:

- LSB-XYZ-Einheit
- Radiant/Sekunden
- Grad/Sekunden

Die inertiale Messeineheit liefert für den Beschleunigungssensor die Werte als G-XYZ-Werte, das Gyroskop liefert die Werte in der Grad-Einheit zurück.

Das beispielhafte Auslesen der Sensordaten des Gyroskops BMG160 ist im Algorithmus 4.2 zu sehen. Die Sensordaten des BMG160 werden über die Struktur *CalibratedAccel_XyzGData_T* abgefragt und in einer entsprechenden Variablen abgespeichert. Für die anderen Sensoren müssen dementsprechend die Schlüsselwörter und Strukturen angepasst werden.

Dies sind die Schlüsselwörter für die anderen Sensoren:

- BMA280: CalibratedAccel_readXyzMps2Value(&calibAccelMps2Data);

- BMA280: CalibratedAccel_readXyz GValue(&calibAccelGData);
- BMI160 Beschleunigungssensor: Accelerometer_readXyz GValue(xdkAccelerometers_BMI160_Handle, &getaccelData);
- BMG160: CalibratedGyro_readXyzRpsValue(&calibGyroRpsData);
- BMG160: CalibratedGyro_readXyzDpsValue(&calibGyroDpsData);
- BMI160 Gyroskop: Gyroscope_readXyz DegreeValue(xdkGyroscope_BMI160_Handle,&getConvData);
- XDK110 Orientation: Orientation_readEuler DegreeVal(&orientationEulerData);

4.1.3 Kalibrierung

Die Kalibrierung der Sensoren kann enormen Einfluss auf die Richtigkeit der gelieferten Sensordaten haben. Deshalb sollten vor jedem Start des System alle verwendeten Sensoren kalibriert werden.

Die Kalibrierung des Beschleunigungssensors BMA280 und BMI160 erfolgt durch die Drehung des XDK110s auf +/- 90 Grad in allen Achsen. Um die Beschleunigungssensoren erfolgreich zu kalibrieren, benötigt es einige Sekunden.

Die Kalibrierung der Gyroskope BMG160 und BMI160 wird erledigt, indem man das XDK110 für ein paar Sekunden in einer flachen Position liegen lässt.

Bei der Kalibrierung des Orientierungs-Sensors des XDK110s müssen mehrere Vorgänge durchgeführt werden. Als erstes wird die Kalibrierung für das Gyroskop durchgeführt und anschließend die Kalibrierung für den Magnetometer. Das Gyroskop wird wie oben beschrieben kalibriert, indem das XDK110 in einer flachen Position liegen gelassen wird. Die Kalibrierung des Magnetometers erfolgt durch eine ’8’-Figur-Bewegung. Diese Bewegung sollte einige Male wiederholt werden, um eine genau Kalibrierung zu erreichen.

4.2 Filteralgorithmen

Das wichtigste Thema diese Arbeit sind die Filteralgorithmen. Die Theorie, die hinter den Filteralgorithmen steckt, wurde im Kapitel 2 beschrieben. Aus diesem Grund wird in diesem Kapitel die eigentliche Implementierung der Algorithmen vorgestellt.

Von den genannten Filteralgorithmen des vorherigen Kapitels und den Möglichkeiten eine Sensorfusion durchführen zu können, wurden der Komplementär-Filter 1 und 2, sowie der

Algorithmus 4.3 Komplementär-Filter 1

```
// Komplementaer-Filter 1; Quelle: [Col07]
double val1 = gyroRateX * ((double) ((OS_timeGetSystemTimeMs() + 1) - timer) / 1000);
double angle = 0.98 * (angle + val1) + 0.02 * (accRoll);
```

Algorithmus 4.4 Komplementär-Filter 2

```
// Komplementaer-Filter 2; Quelle: robottini . altervista .org/tag/kalman-filter
k = 10;
dt = ((double) ((OS_timeGetSystemTimeMs() + 1) - timer) / 1000);
x1 = (accRoll - angle) * k * k;
y1 = dt * x1 + y1;
x2 = y1 + (accRoll - angle) * 2 * k + gyroRateX;

angle = dtc2 * x2 + angle;
```

Kalman-Filter implementiert. Die Entscheidung fiel auf diese drei Filteralgorithmen, da diese im Allgemeinen in Systemen zum Einsatz kommen, die etwas Balancieren müssen.

In den endgültigen Systemen kommen die Filteralgorithmen sowohl auf dem XDK110, als auch auf dem Raspberry Pi zum Einsatz. In diesem Kapitel wird lediglich die Implementierung der Algorithmen auf dem XDK110 in der Programmiersprache C++ vorgestellt. Die Implementierung der Algorithmen auf dem Raspberry Pi, mit der Programmiersprache Java, unterscheidet sich gegenüber der Implementierung auf dem XDK110 kaum. Ausschließlich die sprachspezifischen Unterschiede zwischen Java und C++ sind vorhanden.

4.2.1 Komplementär-Filter

Die Algorithmen Komplementär-Filter 1 (4.3) und Komplementär-Filter 2 (4.4) sind im Vergleich zum Kalman-Filter schneller und einfacher zu implementieren.

Es wurden diese zwei Varianten des Komplementär-Filters implementiert, um in den folgenden Versuchen einen Vergleich zu erhalten. Kleine Änderungen an den Konstanten und Tiefpass-Filter bzw. Hochpass-Filtern können zu anderen Ergebnissen führen.

4.2.2 Kalman-Filter

Der Kalman-Filter ist im Gegensatz zum Komplementär-Filter komplizierter umzusetzen und zu implementieren. Dies zeigt sich ebenfalls in der Länge des Programmcodes im Algorithmus 4.5. Dieser Algorithmus berechnet das Ergebnis des Kalman-Filters in verschiedenen Schritten. Die Schritte sind im Algorithmus durch Kommentare im Programmcode erklärt. Auch bei diesem Filteralgorithmus ist die eigentliche Theorie im Kapitel 2 zu finden.

Algorithmus 4.5 Kalman-Filter

```
// Quelle: github.com/TKJElectronics/KalmanFilter
// Kalman-Filter: time update equations (" Predict ")
/* Schritt 1 */
rate = (newRate - bias);
angle = (angle + (dt * rate));

// Update estimation error covariance - Project the error covariance ahead
/* Schritt 2 */
P[0][0] += dt * (dt * P[1][1] - P[0][1] - P[1][0] + Q_angle);
P[0][1] -= dt * P[1][1];
P[1][0] -= dt * P[1][1];
P[1][1] += Q_bias * dt;

// Kalman-Filter: measurement update equations (" Correct ")
// Calculate Kalman gain - Compute the Kalman gain
/* Schritt 4 */
float S = P[0][0] + R_measure; // Estimate error

/* Schritt 5 */
float K[2] = { 0, 0 }; // Kalman gain
K[0] = P[0][0] / S;
K[1] = P[1][0] / S;

// Calculate angle and bias
/* Schritt 3 */
float y = newAngle - angle; // Angle difference

/* Schritt 6 */
angle = (angle + (K[0] * y));
bias = (bias + (K[1] * y));

// Calculate estimation error covariance - Update the error covariance
/* Schritt 7 */
float P00_temp = P[0][0];
float P01_temp = P[0][1];

P[0][0] -= K[0] * P00_temp;
P[0][1] -= K[0] * P01_temp;
P[1][0] -= K[1] * P00_temp;
P[1][1] -= K[1] * P01_temp;
```

4.3 Datenübertragung

Die Datenübertragung in einem System, welches in Echtzeit arbeiten muss, hat eine enorme Bedeutung. Die entworfenen Filteralgorithmen können noch so performant sein, wenn die Sensordaten vom XDK110 zu einem anderen Rechner zu viel Zeit benötigen, gibt es keine Möglichkeit das Pendel in einer vertikalen Position zu halten. Aus diesem Grund bekommt das Thema Datenübertragung ein eigenes Unterkapitel.

Die XDK Workbench bietet eine einfache und schnelle Möglichkeit die Datenübertragung via UDP zu realisieren, indem die Datei „SendDataOverUdp.h“ in das Projekt eingebunden wird. Dadurch gibt es die Möglichkeit die Funktionen zu implementieren, mit denen das XDK110 sich mit einem Netzwerk verbindet und eine IP-Adresse erhält. Anschließend ist es möglich in regelmäßigen Zeitabschnitten Daten als UDP-Pakete über WiFi an den Empfänger zu senden.

Im entworfenen System werden alle 10ms die Werte an den Empfänger gesendet. Die Konfiguration des UDP Clients im Netzwerk erfolgt durch „Macros“, welche die WPA/WPA2 Einstellungen ändern. Anzumerken ist, dass die IP-Adresse im hexadezimalen Format angegeben werden muss.

4.3.1 XDK110 mit Netzwerk verbinden

Um das XDK110 erfolgreich mit einem Netzwerk verbinden zu können, müssen Informationen über das Netzwerk selbst und den UDP-Server, also dem Empfänger, bekannt sein. Ohne die Informationen über den Empfänger ist es dem XDK110 nicht möglich UDP-Pakete an diesen zu versenden.

Folgende Informationen müssen bekannt sein, um das XDK110 erfolgreich mit einem Netzwerk verbinden zu können und UDP-Pakete an einen Empfänger zu senden:

- Service Set Identifier (SSID)
- Passwort des Netzwerks
- Server-IP
- Server-Port

Der Algorithmus 4.6 zeigt wie das XDK110 mit einem Netzwerk verbunden wird. Als erstes werden im Programmcode die spezifischen Einstellungen für das Netzwerk benötigt und abgefragt. Dazu gehören z.B. die IP-Adresse, die SSID und der WPA-Pass. Ist die Verbindung des XDK110s mit dem Netzwerk erfolgreich gewesen, startet der *WiFi-Sending-Timer*. Ab diesem Zeitpunkt ist es möglich, Daten über das Netzwerk an einen Empfänger zu senden.

Algorithmus 4.6 XDK110 mit einem Netzwerk verbinden

```
NCI_ipSettings_t myIpSettings;
memset(&myIpSettings, (uint32_t) 0, sizeof(myIpSettings));
int8_t ipAddress[PAL_IP_ADDRESS_SIZE] = { 0 };
Ip_Address_T* IpaddressHex = Ip_getMyIpAddr();
WLI_connectSSID_t connectSSID;
WLI_connectPassPhrase_t connectPassPhrase;

connectSSID = (WLI_connectSSID_t) WLAN_CONNECT_WPA_SSID;
connectPassPhrase = (WLI_connectPassPhrase_t) WLAN_CONNECT_WPA_PASS;

if (WLI_SUCCESS == WLI_connectWPA(connectSSID, connectPassPhrase, NULL)) {
        NCI_getIpSettings(&myIpSettings);
        *IpaddressHex = Basics_htonl(myIpSettings.ipV4);
        (void) Ip_convertAddrToString(IpaddressHex, (char *) &ipAddress);
} else {
        printf("'Error occurred connecting"', WLAN\_CONNECT\_WPA\_SSID);
        return;
}

/* After connection start the wifi sending timer*/
OS_timerStart(wifiSendTimerHandle, TIMERBLOCKTIME_UDP);
```

4.3.2 Daten an Empfänger senden

Nach dem erfolgreichen Verbinden mit einem Netzwerk ist es möglich die Sensordaten des XDK110s an einen Empfänger zu senden, siehe Algorithmus 4.7. Die Daten, die an den Empfänger gesendet werden sollen, werden nicht einzeln versendet, sondern in einem Puffer. Sind alle Daten im Puffer angelegt, kann dieser Puffer über das Netzwerk an den Empfänger gesendet werden. Dazu wird als erstes eine *SocketID* erstellt, die für das anschließende Senden des Puffers notwendig ist. Nach dem Senden der Dateien, wird die Verbindung zum Empfänger, mittels der *SocketID*, wieder geschlossen.

4.3.3 Daten von XDK110 empfangen

Damit die gesendeten Daten des XDK110s erfolgreich empfangen werden können ist der Raspberry Pi für das Empfangen der Daten zuständig. Das Empfangen der Daten mittels Programmcode wird im Algorithmus 4.8 gezeigt.

Das Programm des Raspberry Pis, erstellt dazu ein *DatagramSocket* und eine *InetSocketAddress*, mit einer IP-Adresse und einem Port. Anschließend wird in einer Endlossschleife nach ankommenden Datenpaketen gelauscht. Sind Daten über die Verbindung angekommen müssen diese noch entpackt werden. Dies geschieht durch die *wrap*-Funktion. Ohne dieses „Entpacken“ wäre es nicht möglich, die Daten weiterzuverarbeiten. Auf die einzelnen Daten kann nach dem Entpacken zugegriffen werden.

Algorithmus 4.7 Sensordaten an Empfänger senden

```
// Write all data to the buffer
bsdBuffer_mau[0] = accXAxisData;
bsdBuffer_mau[1] = accYAxisData;
bsdBuffer_mau[2] = accZAxisData;

Addr.sin_family = SL_AF_INET;
Addr.sin_port = sl_Htons((uint16_t) port);
Addr.sin_addr.s_addr = sl_Htonl(SERVER_IP);
AddrSize = sizeof(SlSockAddrIn_t);

/* The return value is a positive number if successful */
SockID = sl_Socket(SL_AF_INET, SL_SOCK_DGRAM, IPPROTO_UDP);
if (SockID < (int16_t) ZERO) {
        /* error case */
        return (SOCKET\_ERROR);
}

Status = sl_SendTo(SockID, bsdBuffer_mau, BUFFER_SIZE * sizeof(uint16_t), (uint32_t) ZERO,
    (SlSockAddr_t *) &Addr, AddrSize);
/* Check if 0 transmitted bytes sent or error condition */
if (Status <= (int16_t) ZERO) {
        sl_Close(SockID);
        return (SEND_ERROR);
}

sl_Close(SockID);
```

4.4 Ablaufbeschreibungen

Dieses Unterkapitel beinhaltet den Programmablauf von Anfang bis Ende. Da das System aus einer einfachen Client/Server Architektur besteht, gibt es in den Abläufen zwei unterschiedliche Programmstarts. Im allgemeinen wird der UDP-Server des Raspberry Pis als erstes gestartet. Im Anschluss danach wird das Programm des UDP-Clients gestartet. Der UDP-Client lässt sich starten, indem das XDK110 angeschaltet wird. Das XDK110 benötigt nach dem Anschalten, einige Sekunden um das Programm zu starten. Ist das Programm des XDK110s erfolgreich gestartet so beginnt es mit dem Senden der jeweiligen Daten.

Da in dieser Arbeit zwei unterschiedliche Programmabläufe vorhanden sind, werden diese beide einzeln mit einem Sequenzdiagramm beschrieben. Das Sequenzdiagramm, welches den Ablauf beschreibt, in dem die Filterberechnungen auf dem UDP-Server (Raspberry Pi) stattfinden, befindet sich in der Abbildung 4.1. Das Sequenzdiagramm, welches dagegen die Filterberechnungen lokal auf dem XDK110 berechnet, befindet sich in der Abbildung 4.2.

Algorithmus 4.8 Daten von XDK110 empfangen

```
byte[] receiveData = new byte[36];

float gyrX;
float gyrY;
float gyrZ;

try {
  @SuppressWarnings("resource")
  DatagramSocket serverSocket = new DatagramSocket(null);
  InetSocketAddress address = new InetSocketAddress(ipAdress, port);
  serverSocket.bind(address);
  while (true) {
    DatagramPacket receivePacket = new DatagramPacket(receiveData,
        receiveData.length);
    serverSocket.receive(receivePacket);

    ByteBuffer buf =
        ByteBuffer.wrap(receivePacket.getData()).order(ByteOrder.LITTLE_ENDIAN);

    gyrX = buf.getFloat(0);
    gyrY = buf.getFloat(4);
    gyrZ = buf.getFloat(8);
  }
}
```

4.4.1 Ablaufbeschreibung: Filterberechnung auf dem Raspberry Pi

Das Sequenzdiagramm 4.1, welches die Filterberechnungen auf dem Raspberry Pi durchführt, beginnt mit dem Start des UDP-Clients und des UDP-Servers. Das XDK110 beginnt nach dem Start mit der Ausführung der *main()*-Methode und bereitet anschließend das System zum starten vor. Bevor das XDK110 Daten als UDP-Pakete verpackt und an den UDP-Server senden kann, werden alle notwendigen Initialisierungen durchgeführt und eine Verbindung mit dem Netzwerk aufgebaut. Sind alle wichtigen Komponenten (Sensoren und Netzwerk-Verbindung) erfolgreich initialisiert, so führt das XDK110 alle 10ms eine Schleife aus. In dieser Schleife werden alle Sensorwerte abgefragt und an den UDP-Server gesendet.

Bevor der UDP-Server aber die Daten des XDK110s empfangen kann, startet dieser seine *main()*-Methode, liest die Konfigurationsdatei ein und ändert daraufhin die jeweiligen Einstellungen. Anschließend erstellt der UDP-Server noch eine CSV-Datei, damit die berechneten Daten festgehalten werden können. Waren diese Schritte erfolgreich, dann ist der UDP-Server bereit die Daten vom UDP-Client zu empfangen. Die ersten 500 Datenpakete werden nicht sofort für die eigentliche Filterberechnung verwendet, sondern sind dazu da, die Verzerrung der

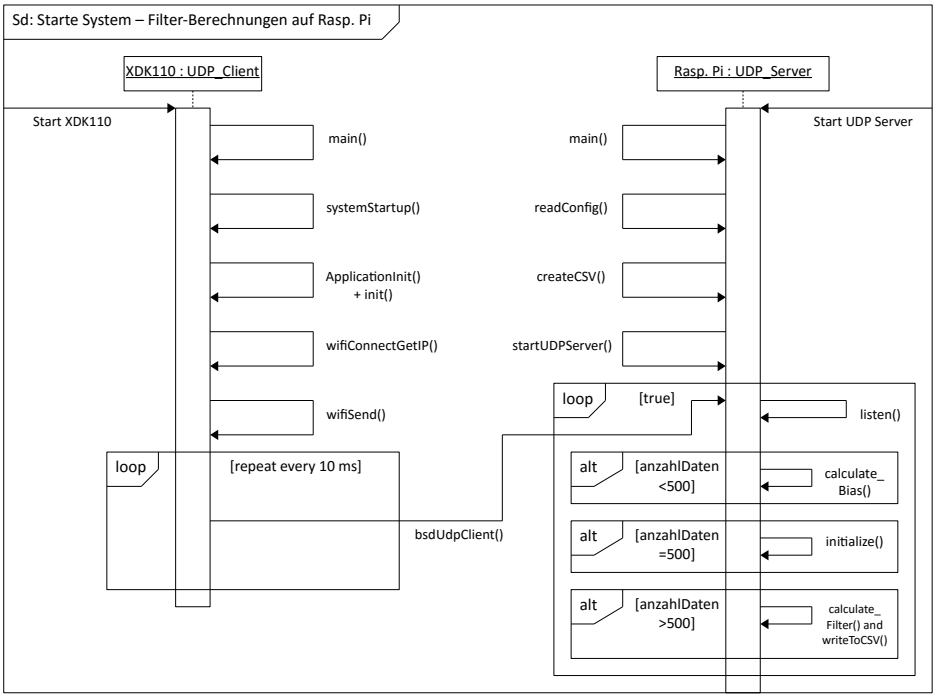


Abbildung 4.1: Das Sequenzdiagramm, welches den Programmablauf zeigt, bei dem die Filterberechnungen auf dem Raspberry Pi durchgeführt werden.

Sensoren zu berechnen und eine einmalige Initialisierung durchzuführen. Die nun folgenden Datenpakete des UDP-Clients werden für die Filterberechnungen genutzt. Wurde auf Basis der Sensordaten die entsprechende Lage des Pendels berechnet, werden die Ergebnisse mit einem entsprechendem Zeitstempel in der CSV-Datei abgespeichert.

4.4.2 Ablaufbeschreibung: Filterberechnung auf dem XDK110

Finden die Filterberechnungen lokal auf dem XDK110 statt und nicht auf dem Raspberry Pi, so ändert sich im Sequenzdiagramm der Bereich der Berechnungen. Die Berechnung der Verzerrung findet ebenfalls auf dem XDK110 statt, da diese Berechnung vor der eigentlichen Filterberechnung durchgeführt werden muss, um ein präziseres Ergebnis zu erhalten. Anstatt dass das XDK110 in einer Schleife alle 10ms nur die Sensorwerte abfragt, fragt das XDK110 in diesem Fall die Sensorwerte ab und berechnet anschließend alle Filterberechnungen, die daraufhin an den UDP-Server gesendet werden.

Der UDP-Server ist in diesem Fall nur dazu da, um die Ergebnisse des XDK110 zu empfangen und in einer CSV-Datei mit einem entsprechenden Zeitstempel festzuhalten. Der Rest des Sequenzdiagramms 4.2 besitzt keine Unterschiede im Vergleich zum Sequenzdiagramm 4.1, dass die Filterberechnungen auf dem Raspberry Pi zeigt.

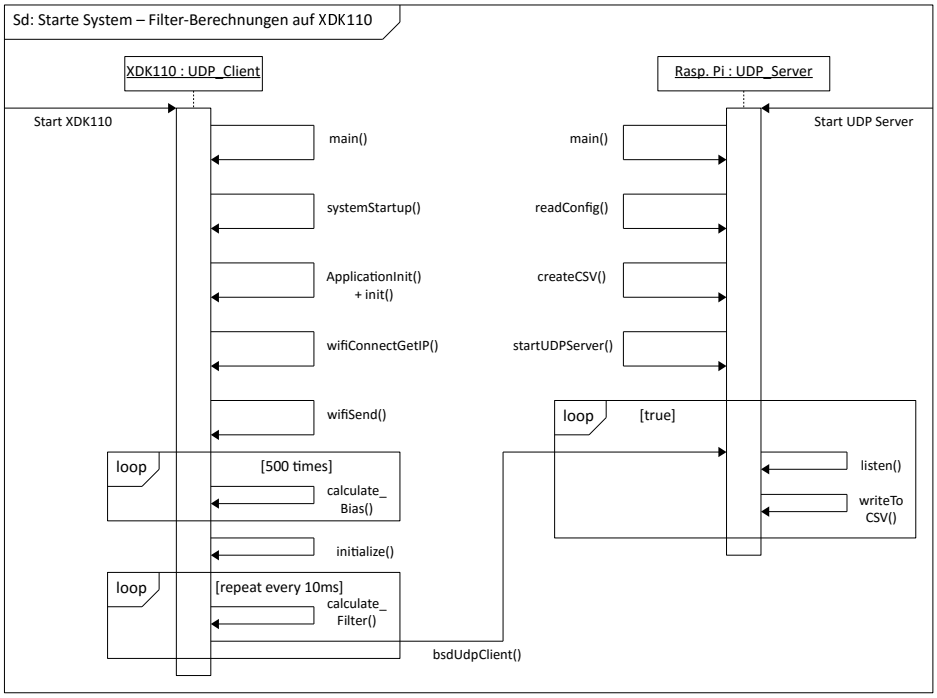


Abbildung 4.2: Das Sequenzdiagramm, welches den Programmablauf zeigt, bei dem die Filterberechnungen auf dem XDK110 durchgeführt werden.

5 Evaluierung

Dieses Kapitel erläutert die Ergebnisse dieser Arbeit. Dabei werden die einzelnen Versuche vorgestellt und analysiert. Außerdem werden die verschiedenen Filteralgorithmen miteinander verglichen.

5.1 Versuchsaufbau

Die Basis aller Versuche basiert auf den Komponenten:

- XDK110 (UDP-Client)
- Raspberry Pi (UDP-Server)
- Pendel mit Halterung am Drucker

Zu sehen ist diese Basis in der Abbildung 5.3 und 3.1. Das XDK110 ist dabei an die Halterung, welche sich an der Drucker-Schiene befindet, angebracht. Werden die einzelnen Versuche gestartet, verbinden sich das XDK110 und der Raspberry Pi mit dem Netzwerk, um Daten vom XDK110 zum Raspberry Pi senden zu können.

5.1.1 Datenaufzeichnung

Damit eine anschließende Analyse der einzelnen Versuche stattfinden kann, benötigt ein System eine zuverlässige Möglichkeit die anfallenden Daten aufzuzeichnen. In dieser Arbeit wurden die gemessenen Daten bzw. die berechneten Daten in einer CSV-Datei festgehalten, welche anschließend analysiert und visuell aufbereitet wurden.

Die festzuhaltenden Daten wurden, ab dem Zeitpunkt an dem sie zur Verfügung standen, in die CSV-Datei mit einem entsprechenden Zeitstempel geschrieben.

Um die CSV-Dateien visuell ansprechend für die Ausarbeitung dieser Arbeit aufzubereiten, wurde das Online-Tool *Plotly* (https://plot.ly) eingesetzt. Plotly ist ein Analyse- und Datenvisualisierungstool, mit dem CSV-Dateien nach dem Hochladen aufbereitet werden können. Anschließend können die entworfenen Diagramme und Grafiken als Portable Network Graphics (PNG)-Datei heruntergeladen werden. Die Basisversion von Plotly ist kostenlos verfügbar,

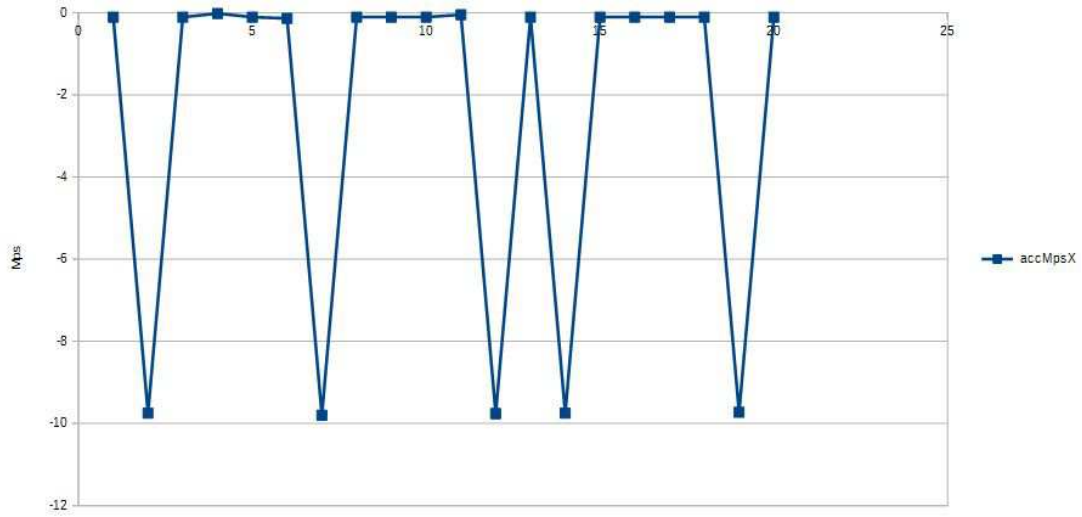


Abbildung 5.1: Zu sehen sind die Rohdaten der X-Achse des BMA280, während das XDK110 in Ruhe auf einem Tisch lag.

zusätzliche Analyse- und Exportmöglichkeiten sind nur in der kostenpflichtigen Version nutzbar.

5.2 Analyse der Sensor-Rohdaten

Bevor die durchgeführten Versuche im Detail vorgestellt und die Ergebnisse präsentiert werden, widmet sich dieses Unterkapitel den Sensor-Rohdaten.

Liefern die eingebauten Sensoren des XDK110s keine verwertbaren Sensordaten, so kann eine korrekte Berechnung durch die Filter, basierend auf den Sensordaten, nicht gewährleistet werden. Im Folgenden werden die eingesetzten Sensoren auf eventuelle Probleme untersucht und die Ergebnisse vorgestellt.

In diesem System wurden die Sensoren BMA280 und BMG160 verwendet, da diese im Vergleich zu der inertialen Messeinheit, welche einen Beschleunigungssensor und ein Gyroskop besitzt, Daten in den Einheiten *Mps* und *Rps* liefern. Die inertiale Messeinheit liefert im Gegensatz dazu die Daten nur in der Einheit *G-Value* und *Dps*.

Die Sensor-Rohdaten wurde bei einem kleinen Versuch, bei dem das XDK110 in Ruhe auf einem Tisch lag, festgehalten und anschließend analysiert. Die restlichen Achsen der BMA280 und BMG160 Sensoren lieferten im Ruhezustand korrekte Daten und werden deshalb in diesem Kapitel nicht genauer betrachtet.

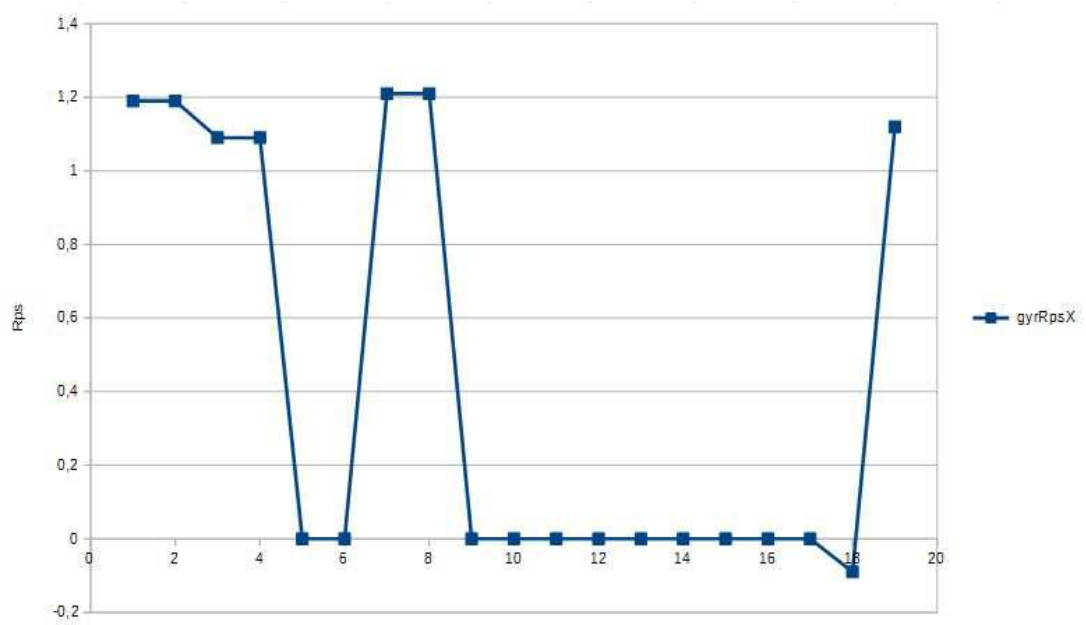


Abbildung 5.2: Zu sehen sind die Rohdaten der X-Achse des BMG160, während das XDK110 in Ruhe auf einem Tisch lag.

5.2.1 Beschleunigungssensor BMA280

Der Beschleunigungssensor BMA280 liefert, wie der Abbildung 5.1 zu entnehmen ist, häufig falsche Daten. Zu sehen sind in dieser Abbildung starke Ausschläge der X-Achse des BMA280.

Obwohl das XDK110 in Ruhe auf einem Tisch lag, gibt der Sensor für die X-Achse Beschleunigungen an, die die späteren Filterberechnungen negativ beeinflussen. Da diese falschen Ausschläge nicht minimal sind, sondern zum Teil um bis zu zehn Werte erhöht sind, haben diese falsche Daten einen enormen Einfluss in den späteren Filterberechnungen.

5.2.2 Gyroskop BMG160

Auch das Gyroskop BMG160 liefert für die X-Achse falsche Daten, obwohl das XDK110 in Ruhe auf einem Tisch lag (siehe Abbildung 5.2). Auch diese Daten haben einen negativen Einfluss auf die Filterberechnungen und verfälschen somit das Ergebnis. Die Filterberechnungen können diese falsche Daten nicht herausfiltern, da sie nicht wissen können, wann das XDK110 sich in Wirklichkeit bewegt und wann nicht. Im Vergleich zum BMA280 Sensor sind die Ausschläge des BMG160 nicht so deutlich und beeinflussen das Ergebnis der späteren Filterberechnungen nicht in der Stärke wie der BMA280.

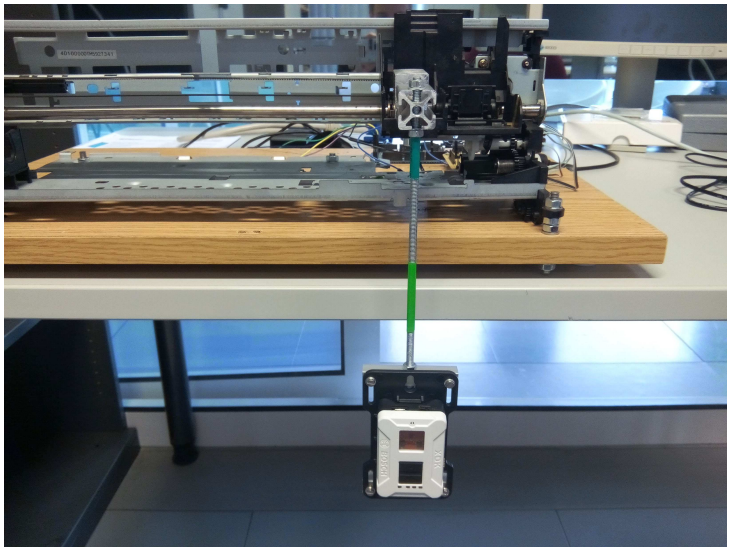


Abbildung 5.3: Zu sehen ist der Versuchsaufbau des Versuches *Ruhezustand*. Das XDK110 hängt dabei in ruhe nach unten.

5.3 Versuche

In diesem Kapitel werden die durchgeführten Versuche vorgestellt und die dabei gewonnenen Ergebnisse präsentiert. Wie die einzelnen Versuche im Detail abgelaufen sind, wird in den jeweiligen Unterkapiteln erklärt.

Obwohl der Komplementär-Filter 2 (4.4), um die Lage des Pendels zu berechnen anders vorgeht als der Komplementär-Filter 1 (4.3), unterscheiden sich die Ergebnisse beider Filteralgorithmen nicht. Aus diesem Grund wird der Komplementär-Filter 2 in den Diagrammen nicht berücksichtigt und nur der Komplementär-Filter 1 gezeigt.

Ein weiterer wichtiger Punkt ist, dass es praktisch keinen Unterschied gibt, ob die Filterberechnungen lokal auf dem XDK110 stattfinden oder die Sensor-Rohdaten zum Raspberry Pi übermittelt werden und dort die Filterberechnungen durchgeführt werden. Die Ergebnisse sind in beiden unterschiedlichen Systemen dieselben. Das XDK110 hat genügend Rechenleistung, um auch den Kalman-Filter in kürzester Zeit berechnen zu können. Aus diesem Grund wird in den folgenden Versuchen dieses Thema nicht weiter berücksichtigt, sondern das Augenmerk auf die Unterschiede der jeweiligen Filteralgorithmen gelegt.

5.3.1 Ruhezustand

Der erste Versuch der in dieser Arbeit durchgeführt wurde, lässt das XDK110 in Ruhe am Pendel nach unten hängen. Das XDK110 ist an die Halterung des Pendels angeschraubt und das Pendel bewegt sich nicht. Den Versuchsaufbau für diesen Versuch sieht man in der Abbildung

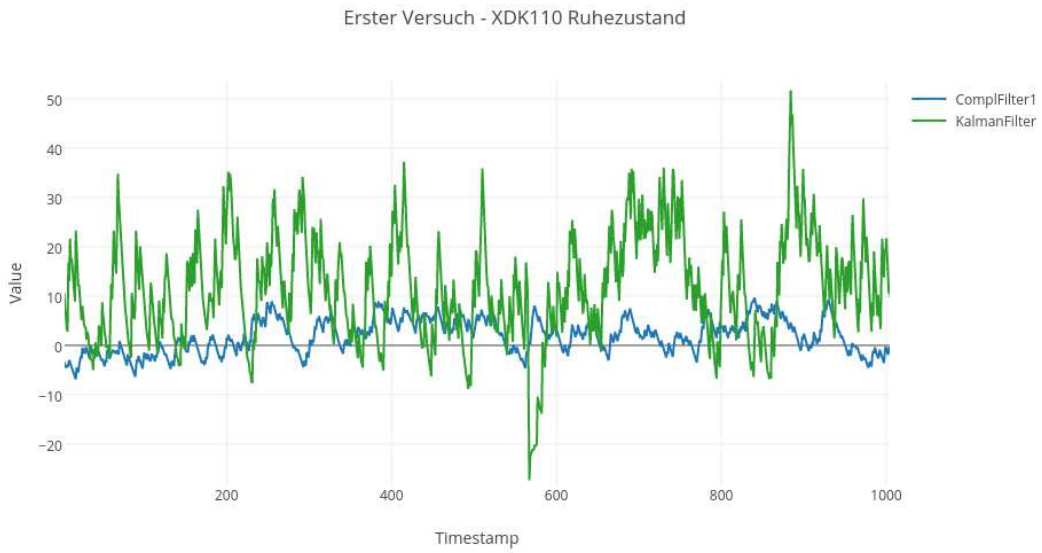


Abbildung 5.4: Das Diagramm zeigt die Ergebnisse der Filterberechnungen des Ruhezustand-Versuchs.

5.3. Der Versuch beginnt, sobald der UDP-Server und der UDP-Client gestartet wurden und bereit sind. Das bedeutet, als erstes wird der UDP-Server auf dem Raspberry Pi gestartert und anschließend das XDK110 aktiviert. Wichtig ist, dass das Pendel während des Versuches nicht bewegt wird und keine Erschütterungen am Tisch das gesamte System beeinflussen. Der Versuch wird beendet, indem das XDK110 ausgeschaltet wird und der UDP-Server auf dem Raspberry Pi beendet wird. Die entstandenen Daten wurden während des Versuches in einer CSV-Datei festgehalten und stehen nach dem Beenden des UDP-Servers zur Verfügung.

Wie der Abbildung 5.4 entnommen werden kann, ist der Komplementär-Filter im Vergleich zum Kalman-Filter „ruhiger“. Das bedeutet, dass die Ergebnisse die die Filterberechnungen des Kalman-Filters liefern, stark schwanken und deutlich größere Abweichungen haben, als die des Komplementär-Filters. Der Komplementär-Filter schwankt nicht so stark und hat keine starken „Ausschläge“ im Ergebnis.

Das Ergebnis des Komplementär-Filters liegt in einem akzeptablen Bereich von -5 und +9. Dagegen sind die Ergebnisse des Kalman-Filters nicht akzeptabel. Obwohl sich das XDK110 in einem Ruhezustand befindet, teilt uns der Kalman-Filter an einigen Zeitpunkten mit, dass sich das XDK110 bewegt und sich somit die Lage des Pendels verändert. Würden diese Ergebnisse an den Schrittmotor gesendet werden, so würde dieser versuchen, diese Bewegungen auszugleichen, obwohl sich das Pendel nicht bewegt. Dies hätte zur Folge, dass sich das Pendel aufgrund des Kalman-Filters und des Schrittmotors bewegen würde.

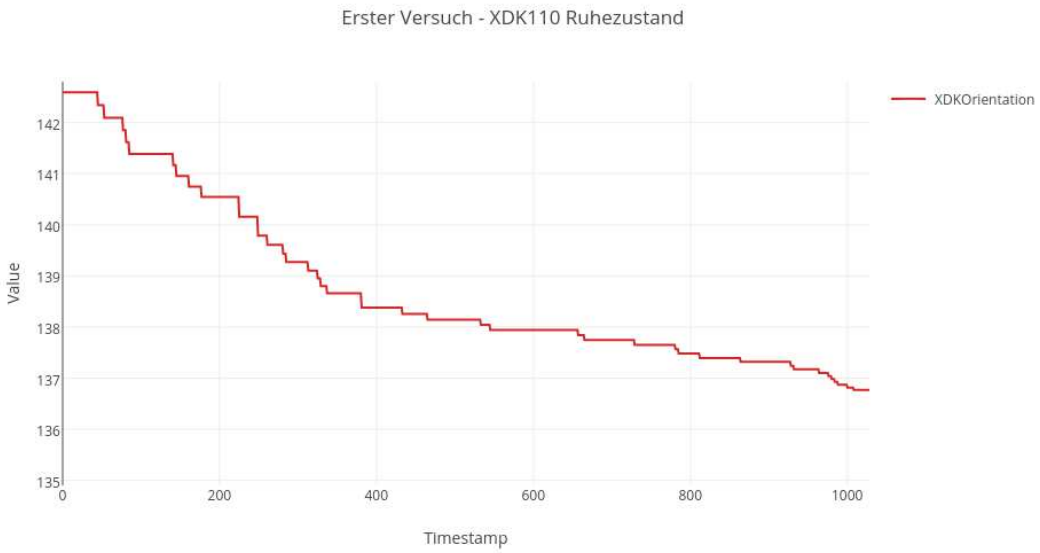
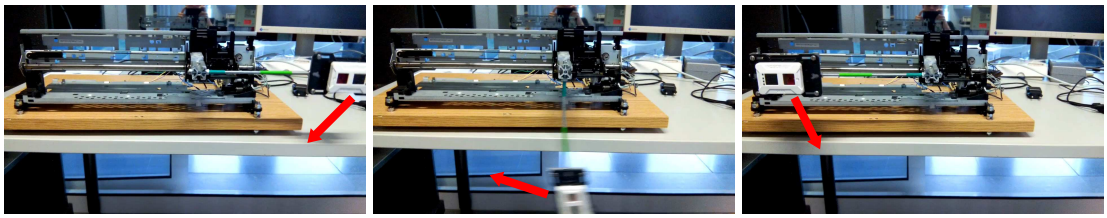


Abbildung 5.5: Dieses Diagramm zeigt die Ergebnisse des *Ruhezustandes* der internen Orientierungs-Lösung des XDK110s an.

Die Schwankungen, die im Diagramm 5.4 zu sehen sind, könnten durch die Daten der X-Achse des Beschleunigungssensors und des Gyroskops entstehen. Da die Daten der X-Achse des BMA280 und des BMG160 häufig falsche Daten liefern (siehe Unterkapitel 5.2) besteht die Möglichkeit, dass diese Daten zu den Schwankungen führen. In der Wirklichkeit bewegt sich das XDK110 nicht, doch die von den Sensoren gelieferten Daten, führen dazu, dass die Filterberechnungen Ergebnisse liefern, die von einer Bewegung des XDK110 stammen könnten.

Die implementierten Filteralgorithmen sind nicht in der Lage, diese falschen Daten herauszufiltern, da die Algorithmen die falschen Daten nicht von den richtigen Daten unterscheiden können.

Um einen Vergleich zu haben, welche Ergebnisse die Berechnungen der internen Lösung des XDK110s liefert, kann die Abbildung 5.5 betrachtet werden. Hier ist zu sehen, dass die Lage des Pendels nicht korrekt berechnet wurde. Das Ergebnis verändert sich deutlich, von anfangs 143 auf 137, obwohl sich das XDK110 nicht bewegt. Die Ergebnisse, die durch die Orientierung des XDK110s zustande kommen, können nicht verwendet werden um eine Aussage über die Lage des Pendels zu machen.



(a) (b) (c)

Abbildung 5.6: Zu sehen ist der Versuchsaufbau der *Pendelbewegung*. Der Start beginnt, sobald sich das XDK110 90° auf der rechten Seite befindet: 5.6a. Der Mittelteil zeigt das XDK110 in der untersten Position: 5.6b. Das Ende der Pendelbewegung ist erreicht, sobald das XDK110 die 90° auf der linken Seite erreicht hat: 5.6c.

5.3.2 Pendelbewegung

Wie in der Abbildung 5.6 zu sehen ist, wurde im zweiten Versuch dieser Arbeit eine klassische Pendelbewegung mit dem XDK110 durchgeführt. Der Versuch wird ebenso wie der vorherige Versuch (*Ruhezustand*) begonnen, indem der UDP-Server und der UDP-Client gestartet werden. Nachdem beide Komponenten bereit sind, wird das Pendel mit dem angeschraubten XDK110 in eine Position gebracht, die in der Abbildung 5.6a zu sehen ist. Anschließend wird das XDK110 losgelassen und das Pendel beginnt zu schwingen. Sind genügend Daten durch den UDP-Server aufgezeichnet, kann der UDP-Server und der UDP-Client beendet werden. Die Pendelbewegung bzw. die Geschwindigkeit der Pendelbewegung des XDK110s wird durch die Vorrichtung nicht verringert, was den Vorteil hat, dass die einzelnen Pendelbewegungen nahezu identisch sind und sich dadurch sehr gut für eine anschließende Analyse der CSV-Datei eignen.

Im Diagramm 5.7 ist zusehen, dass der Kalman-Filter schneller und sensibler als der Komplementär-Filter auf Änderungen der Lage des Pendels reagiert. Dadurch kann der Kalman-Filter auch schnelle Änderungen des Pendels registrieren und könnte dem Schrittmotor so schneller mitteilen, dass er eine entsprechende Bewegung durchführen muss, um das Pendeln vor einem Fallen zu bewahren.

5.3.3 Schrittmotor

Der letzte und dritte Versuch in dieser Arbeit verwendet zusätzlich zu den vorherigen Komponenten einen Schrittmotor, um das Pendel bewegen zu können. Dieser Schrittmotor lässt das gesamte Pendel von links nach rechts fahren und anschließend wieder zurück. Diese Bewegung wird in der Abbildung 5.9 dargestellt. Das Pendel bleibt während des Versuchs nach unten gerichtet und das XDK110 wird nicht bewegt. Lediglich durch die Beschleunigung des Schrittmotors beginnt das XDK110 zu schwingen.

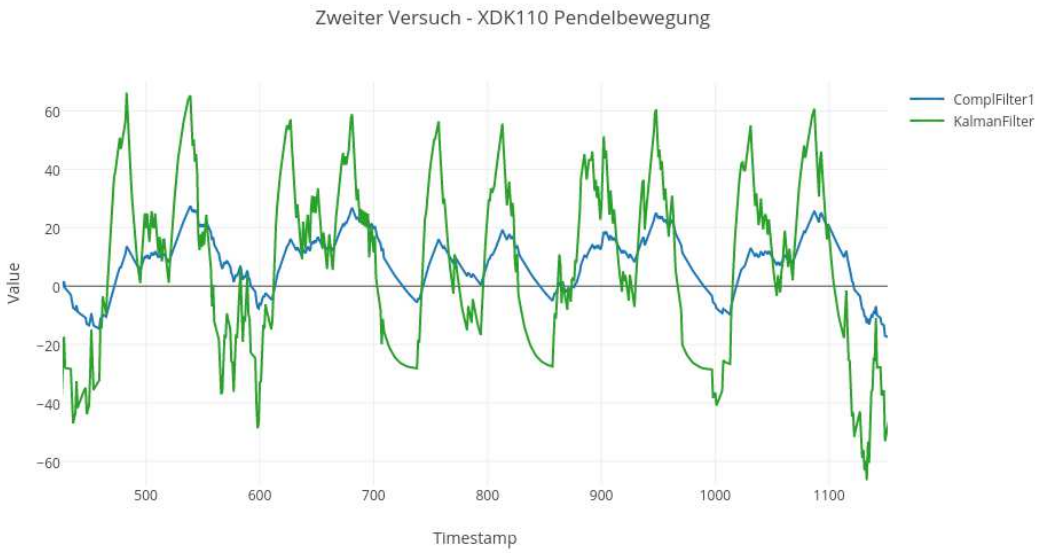


Abbildung 5.7: Das Diagramm zeigt die Ergebnisse der Filterberechnungen der Pendelbewegungen.

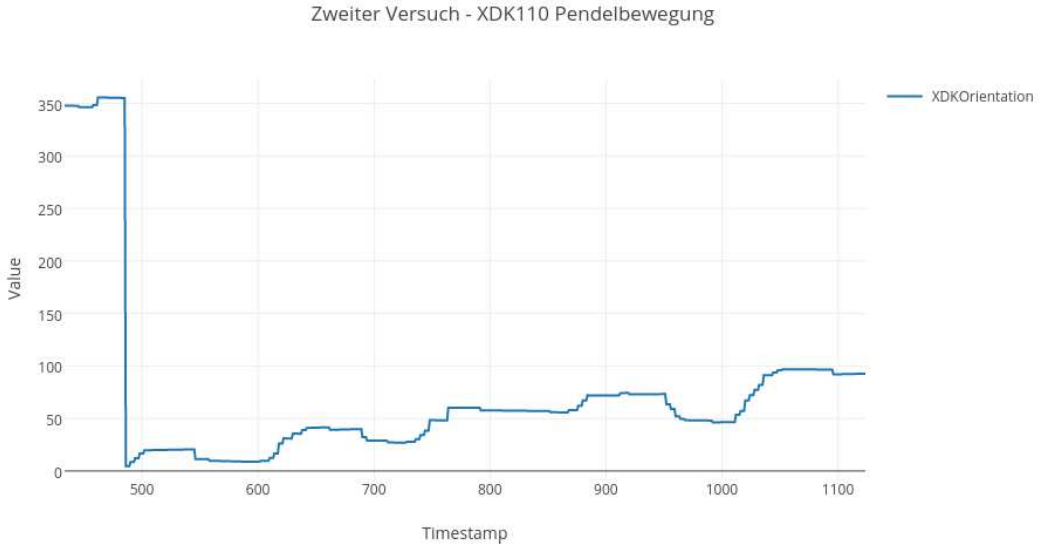
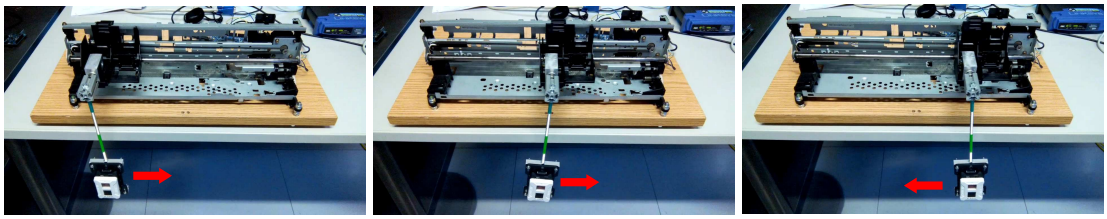


Abbildung 5.8: Dieses Diagramm zeigt die Ergebnisse der internen Orientierungs-Lösung des XDK110s für die Pendelbewegung an.



(a) (b) (c)

Abbildung 5.9: Zu sehen ist der Versuchsaufbau, bei dem der Schrittmotor zum Einsatz kam. Der Start des Versuchs: 5.9a. Der Mittelteil des Versuchs: 5.9b. Das Ende des Versuchs: 5.9c.

Auch in diesem Versuch werden der UDP-Server und der UDP-Client gestartet. Sind beide Komponenten erfolgreich gestartet, muss in diesem Fall noch der Arduino, der den Schrittmotor steuert, gestartet werden. Der Arduino enthält ein passendes Programm, welches den Schrittmotor mit einer definierten Geschwindigkeit laufen lässt. Dieser Versuch gliedert sich in zwei verschiedene Versuche auf.

Beim ersten Versuch wird das gesamte Pendel mit einer normalen und gleichmäßigen Geschwindigkeit bewegt. Die Ergebnisse, die dabei entstanden sind, sind im Diagramm 5.10 zu sehen. Die Filteralgorithmen erkennen die durch die Bewegung ausgelösten Schwingungen und zeigen diese im Diagramm an. Während eines gewissen Zeitabschnittes sind die Ergebnisse auf der Y-Achse nach unten verschoben. Nach einer gewissen Zeit werden die Ergebnisse der Berechnungen wieder normal und die angezeigten Pendelbewegungen verlaufen im Bereich der X-Achse. Vermutlich wird die Verschiebung auf der Y-Achse nach unten durch die Beschleunigung des Pendels ausgelöst.

Im zweiten Versuch wird das gesamte Pendel mit einer speziellen Geschwindigkeit bewegt, welche dafür sorgt, dass es starke Vibrationen an der gesamten Halterung und speziell am XDK110 gibt. Das Diagramm 5.11 zeigt die Ergebnisse der Filteralgorithmen bei diesem Versuch. Diese starken Vibrationen wurden mit Absicht hervorgerufen, um zu testen wie sich die Sensoren und das gesamte System verhalten, wenn es zu starken Vibrationen kommt.

Wie im Diagramm 5.11 zu sehen ist, unterscheiden sich die Ergebnisse der Filteralgorithmen gegenüber der anderen Versuche (5.4 und 5.10). Die Ergebnisse der Filterberechnungen werden durch die Vibrationen am gesamten System auf der Y-Achse nach unten verschoben und kehren nach einer gewissen Zeit nicht wieder in den normalen Bereich um die X-Achse zurück. Die Filteralgorithmen können diese Störungen, die durch die Vibrationen ausgelöst wurden, nicht herausfiltern und liefern dadurch ein falsches Ergebnis. Würde man das gesamte Ergebnis auf der Y-Achse nach oben verschieben, so hätte man ein richtiges Ergebnis.

Der Grund warum in diesem Diagramm eine Zeitspanne von bis zu 1800 Zeitstempel angezeigt wird, ist der, dass die Geschwindigkeit des Schrittmotors im Vergleich zu der normalen

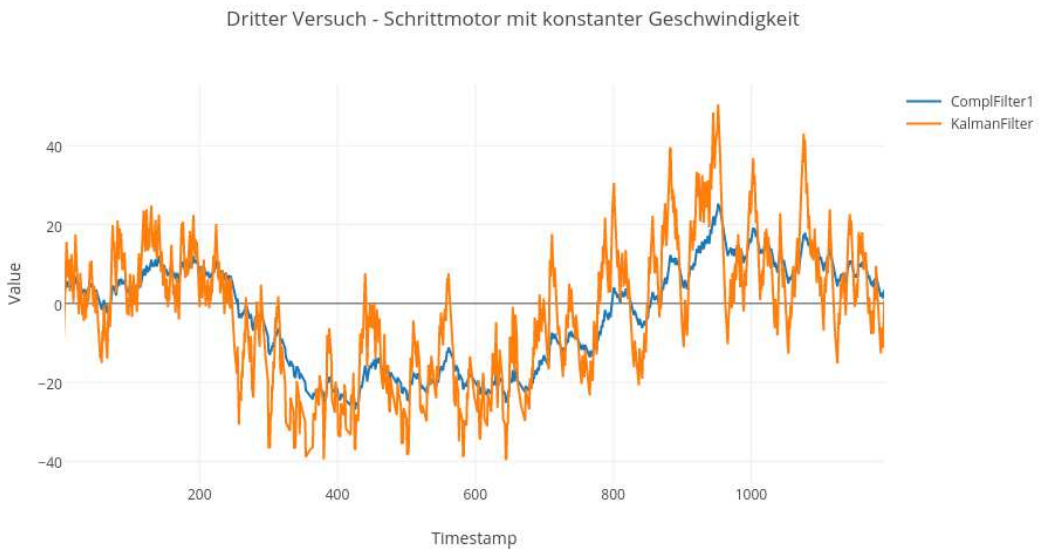


Abbildung 5.10: Das Diagramm zeigt die Ergebnisse der Filterberechnungen, während das XDK110 mit einer normalen Geschwindigkeit auf der Drucker-Schiene bewegt wurde.

Geschwindigkeit, langsamer ist und das Pendel somit mehr Zeit dafür benötigt, um von links nach rechts und wieder zurück zu fahren.

Im Allgemeinen sind die Ergebnisse der Filteralgorithmen für diesen Versuch korrekt, da sie leichte Schwingungen des Pendels erkennen. Lediglich die Bewegung des Pendels auf der Drucker-Schiene führt dazu, dass die Ergebnisse auf der Y-Achse verschoben sind. Die Bewegung des Pendels, die in diesem Versuch durchgeführt wurde, würde in einem allgemeinen System, welches versucht ein Pendel in einer aufrechten Position zu halten, so nicht auftreten. Der Schrittmotor würde in solch einem System nur durch schnelle und kurze Beschleunigungen bzw. Bewegungen das Pendel in einer aufrechten Position halten und nicht durch lang andauernde Bewegungen die Position des Pendels verändern. Trotzdem ist dieser Versuch ein guter Test, um zu sehen wie das System auf Extremsituationen reagiert und hat deshalb eine Daseinsberechtigung.

5.3.4 Verbesserungsmöglichkeit

Damit die Ergebnisse der Filterberechnungen nicht in dieser Weise zittern, wie das in den vorherigen Diagrammen zu sehen ist, gibt es die Möglichkeit, den Mittelwert aus zwei berechneten Ergebnissen zu bildet. Dabei wird das erste Ergebnis der Berechnung zwischengespeichert

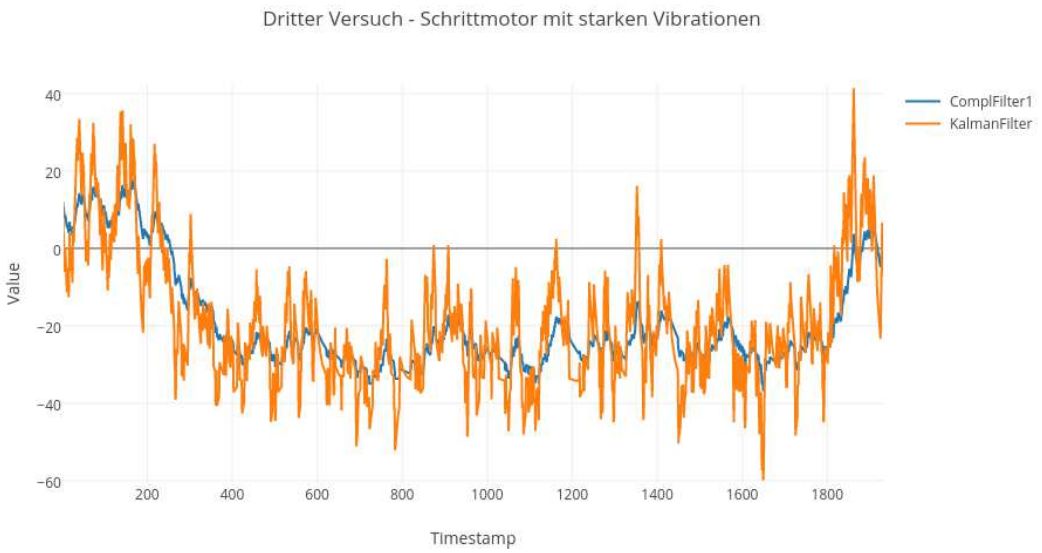


Abbildung 5.11: Das Diagramm zeigt die Ergebnisse der Filterberechnungen wenn das XDK110 mit einer Geschwindigkeit auf der Drucker-Schiene bewegt wird, die starke Vibrationen am ganzen System auslöst.

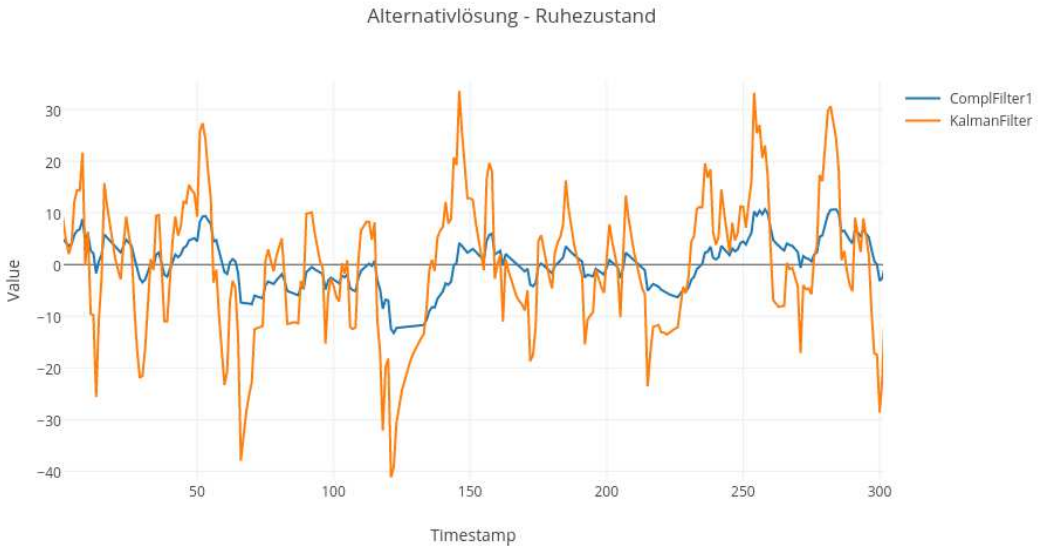


Abbildung 5.12: Diese Abbildung zeigt die Ergebnisse der alternativen Lösung, die beim Versuch *Ruhezustand* erzielt wurden.

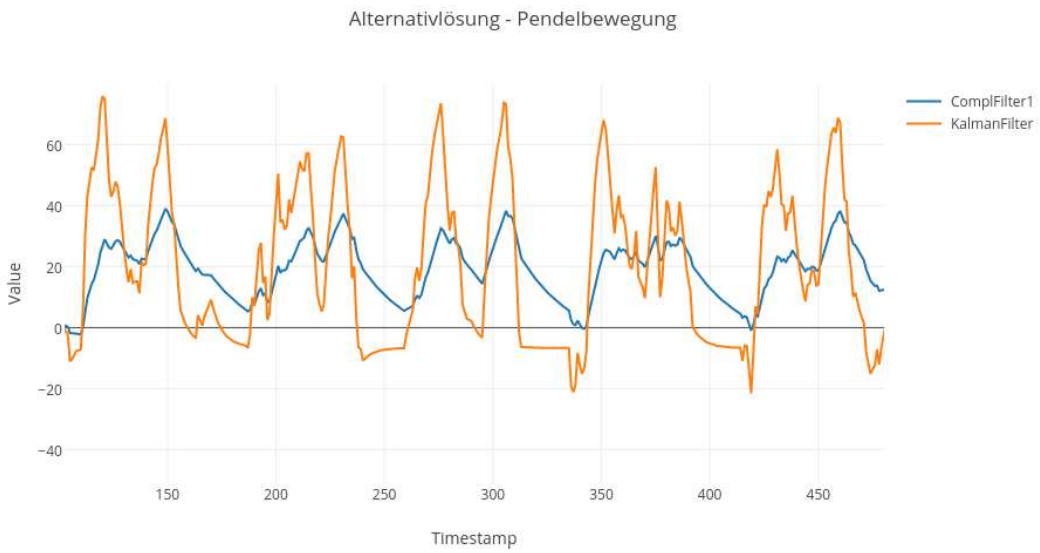


Abbildung 5.13: Das Diagramm zeigt die Ergebnisse der Filterberechnungen die bei den Pendelbewegungen entstanden sind.

und mit dem darauffolgendem Ergebnis der Mittelwert gebildet, welches nach weiteren 10ms vorhanden ist. Zu sehen sind die Ergebnisse in den Diagrammen 5.12 und 5.13.

Ein Nachteil dieser Lösung ist, dass ein verwertbares Ergebnis erst nach 20ms vorhanden ist. Das bedeutet, dass anstatt wie zuvor in einer Sekunde 100 mal die Lage des Pendels berechnet, in dieser Variante nur 50 mal in einer Sekunde die Lage berechnet wird. Für Systeme, die mit dieser Zeitspanne nicht umgehen können (d.h. die Ergebnisse mindestens alle 10ms vorhanden sein müssen, um auf Änderungen besser reagieren zu können) ist diese Alternativlösung nicht geeignet. Für Systeme, die nur alle 20ms ein Ergebnis benötigen und trotzdem auf Änderungen reagieren können, ist die Alternativlösung eine Verbesserung gegenüber der Standard-Lösung, die alle 10ms Ergebnisse liefert. Die einzelnen Ausschläge der Filteralgorithmen treten bei der Alternativlösung nicht so stark auf, was zur Folge hat, dass die Kurven in den Diagrammen flüssiger und ruhiger werden.

6 Fazit und Ausblick

Das in dieser Arbeit entstandene System ist in fähig mittels unterschiedlicher Filteralgorithmen die Lage eines Pendels in Echtzeit berechnen zu können. Dabei wurde als erstes der allgemeine Stand der Technik und die vorhandenen Komponenten vorgestellt. Anschließend wurde ein Systemmodell entwickelt, welches die vorgestellten Komponenten zu einem gesamtem System vereint. Die einzelnen Komponenten bekamen spezielle Aufgaben zugeordnet und standen mit anderen Komponenten in enger Beziehung.

Aufbauend auf diesem Modell wurden die unterschiedlichen Abfolgen des Systems entworfen und umgesetzt. Dabei gibt es die Möglichkeit, dass das System die Filterberechnungen lokal auf dem XDK110 oder auf dem entfernten Raspberry Pi ausführt. Die durchgeführten Versuche haben gezeigt, dass das XDK110 mit der Rechenleistung des Raspberry Pi mithalten kann und es somit keinen Unterschied macht, ob die Filterberechnungen auf dem XDK110 oder auf dem Raspberry Pi durchgeführt werden.

Das XDK110 ist fähig alle 10ms die Sensor-Rohdaten als UDP-Datenpakete an den Raspberry Pi zu senden. Diese kurze Zeitspanne ermöglicht es dem System die Lage des Pendels pro Sekunde 100 mal zu berechnen. Dieses häufige Senden der Daten vom XDK110 zum Raspberry Pi wurde durch den Linksys Wireless-Router ermöglicht, der dafür gesorgt hat, dass sich das XDK110 ohne Probleme und Störungen im Netzwerk mit dem Raspberry Pi verbinden kann.

Die unterschiedlichen Versuche, die mit dem System durchgeführt wurden, sind gezielt ausgesucht worden, um das System und sein Verhalten bei speziellen Fällen beobachten zu können.

Der erste Versuch zielte auf das Problem des Ruhezustandes ab. Dies betrifft vor allem die Sensoren, die im XDK110 eingebaut sind, da diese im Ruhezustand keine falschen Daten liefern dürfen. Dabei kam heraus, dass die Sensoren Bewegungen und Änderungen messen, obwohl sich das XDK110 nicht bewegt hat. Dies hat zur Folge, dass die Filteralgorithmen die tatsächliche Lage des Pendels falsch berechnen. Die einzelnen Filteralgorithmen haben gezeigt, dass der Kalman-Filter gegenüber dem Komplementär-Filter unruhiger ist und häufig große Abweichungen in den Ergebnissen produziert.

Der zweite Versuch sollte zeigen, ob das System eine klassische Pendelbewegung erkennen kann. Die Ergebnisse aus diesem Versuch haben gezeigt, dass die Filteralgorithmen diese Bewegung sehr gut erkennen bzw. berechnen können. Bei diesem Versuch hat der Kalman-Filter bessere Ergebnisse erzielt als der Komplementär-Filter. Der Kalman-Filter reagiert schneller auf Änderungen der Lage des Pendels und wäre somit imstande, diese Änderungen schneller an einen Schrittmotor zu senden, der daraufhin eine entsprechende Bewegung durchführen kann

um die Lage des Pendels zu verändern. Der Komplementär-Filter verhält sich bei schnellen Änderungen träge und könnte aus diesem Grund nur verzögert die Ergebnisse einem Schrittmotor mitteilen.

Der letzte Versuch in dieser Arbeit hatte das Ziel, das System an die Grenzen zu bringen. Dabei kam zum ersten Mal der Schrittmotor zum Einsatz. Der Schrittmotor hat in diesem Versuch das Pendel von links nach rechts und wieder zurück bewegt. Dabei entstanden natürliche Schwingbewegungen des Pendels. Die Filteralgorithmen haben in diesem Versuch verwertbare Ergebnisse geliefert, aber auch dieses mal reagierte der Kalman-Filter schneller auf Änderungen der Lage des Pendels. Beim zweiten Durchlauf dieses Versuches, wurde der Schrittmotor so eingestellt, dass er starke Vibrationen bei der Bewegung des Pendels am ganzen System auslöst. Dies ist sozusagen ein Worst-Case-Fall bei dem man sehen wollte, ob die Sensoren falsche Daten liefern und wie das System auf diese starken Vibrationen reagiert. Das Ergebnis dieses Versuchs ist, dass die starken Vibrationen Einfluss auf die Sensoren und dadurch auf die Filteralgorithmen haben. Die Ergebnisse der Filterberechnungen sind im Allgemeinen in Ordnung, allerdings wurden sie leicht nach unten verschoben.

6.1 Ausblick

Da das System nur einen Schrittmotor zur Verwendung hatte, welcher nicht ausreicht um ein Pendel in einer aufrechten Position zu halten, wäre es sehr interessant, das gesamte System mit einem Motor zu testen, der in der Lage ist, ein Pendel, durch schnelle Bewegungen in horizontaler Ebene, in einer aufrechten Position zu halten. Bei solch einem Versuch ist es wichtig, dass die berechneten Ergebnisse der Filteralgorithmen in kürzester Zeit an den Arduino weitergeleitet werden, damit dieser durch eine passende Ansteuerung des Schrittmotors das Pendel vor dem kippen bewahrt. Um das aktuelle System zu verbessern, müsste eine Lösung gefunden werden, die die gelieferten Daten der Sensoren für richtig und falsch einordnen kann. Es müsste sozusagen einen Filter geben, der erkennt, ob die gerade gesendeten Daten der Sensoren in der aktuellen Lage des XDK110s sinnvoll sind. Diese Lösung würde aber nur das Problem der Sensoren im Ruhezustand lösen, wenn sich das XDK110 nicht bewegt. Sobald sich das XDK110 bewegt und sich dadurch die Lage des Pendels verändert, liefern die aktuell verwendeten Filteralgorithmen gute Ergebnisse.

Zusätzlich dazu wäre es für das System sinnvoll einen Filter einzubauen, der starke Vibrationen herausfiltern kann. Da es immer Vibrationen gibt, die durch den Schrittmotor ausgelöst werden, würde dieser Filter eine enorme Verbesserung für das System bedeuten.

Literaturverzeichnis

[Bes12] J. Beschorner. „Seminar Automobile Systeme in der Automatisierung - Kalman Filter“. 2012. URL: https://www.uni-koblenz-landau.de/de/koblenz/fb4/ist/AGZoebel/Lehre/ss2012/seminar/jBeschorner. (zitiert auf S.34, 36).

[Bosch] *XDK 110 Cross Domain Development Kit.* URL: http://xdk.bosch-connectivity.com/documents/37728/87798/XDK_Getting_Started.pdf. (zitiert auf S.19, 20)

[Col07] S.Colton. „A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform“. 2007. URL: http://portal.ts-muenchen.de/Dateien/filter.pdf. (zitiert auf S.28, 29, 30, 31, 32, 33, 45).

[VDI13] *Thesen und Handlungsfelder Cyber-Physical Systems : Chancen und Nutzen aus Sicht der Automation*. 2013. URL: https://www.vdi.de/uploads/media/Stellungnahme_Cyber-Physical_Systems.pdf. (zitiert auf S.17).

[Bos15] *Cross-Domain Development Kit XDK110 Platform for Application Development.* 2015. URL: *http://xdk.bosch-connectivity.com/documents/37728/87798/XDK_User_Guide.pdf.* (zitiert auf S.18, 19, 20, 21, 22).

[inPen] Bild: *Invertiertes Pendel*. URL: https://de.wikipedia.org/wiki/Inverses_Pendel#/media/File:Cart-pendulum.png. (zitiert auf S.14).

[Sch00] U. Schneider. „*Das Invertierte Pendel*“. 2000. URL: http://www2.htw-dresden.de/~iwe/Belege/SchneiderPendel/pendel.html. (zitiert auf S.13).

[XDKWo] Interne XDK110 Workbench Dokumentation. (zitiert auf S.23).

[SKPrk] K. Rothermel, B. Koldehofe. „Systemkonzepte und -programmierung – Grundlagen der Rechnernetze“. WS 14/15. Vorlesung. (zitiert auf S.25, 26, 27).

[OSIko] *OSI-Schichtenmodell in der Netzwerktechnik*. URL: http://www.elektronik-kompendium.de/sites/net/0706101.htm. (zitiert auf S.25).

[UDPko] *UDP – User Datagram Protocol*. URL: http://www.elektronik-kompendium.de/sites/net/0812281.htm. (zitiert auf S.).

[IP4ko] IPv4 – Internet Protocol Version 4. URL: http://www.elektronik-kompendium.de/sites/net/0811271.htm. (zitiert auf S.26).

[ENX11] J. Esfandyari, R. Nuccio, G. Xu. „Beschleunigungsaufnehmer“. 2011. URL: http://www.all-electronics.de/beschleunigungsaufnehmer/. (zitiert auf S.28, 33).

[RaspP] *Raspberry Pi 2 Model B*. URL: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. (zitiert auf S.24).

[ArDUE] *Arduino Board Due*. URL: https://www.arduino.cc/en/Main/ArduinoBoardDue. (zitiert auf S.24).

Alle URLs wurden zuletzt am 09.10.2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift