Universität Rostock

Traditio et Innovatio

Sensor-in-the-Loop

Eine neuartige Debugging-Architektur für intelligente Sensoren

Dissertation

Vorgelegt von:

Daniel Gis,

geboren am 25.06.1984 in Pasewalk

zur Erlangung des akademischen Grades eines

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

Eingereicht am:

06.02.2023

https://doi.org/10.18453/rosdok_id00004384

UNIVERSITÄT ROSTOCK | FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK

Institut für Angewandte Mikroelektronik und Datentechnik

Albert-Einstein-Str. 26, 18059 Rostock | Tel: +49 381 498-7251

Gutachter

- Prof. Dr.-Ing. habil. Christian Haubelt
  Lehrstuhl “Eingebettete Systeme”
  Institut für Angewandte Mikroelektronik und Datentechnik
  Universität Rostock

- Prof. Dr.-Ing. Martin Radetzki
  Institut für Technische Informatik
  Universität Stuttgart

Datum der Abgabe: 06.02.2023
Datum der Verteidigung: 06.07.2023

Danksagung

Die vorliegende Dissertation entstand während meiner Arbeit als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mikroelektronik und Datentechnik der Universität Rostock.

Ein ganz besonderer Dank gilt meinem Doktorvater Herrn Prof. Dr.-Ing. habil. Christian Haubelt, der mir die Promotion am Lehrstuhl für Eingebettete Systeme ermöglicht hat. Bei der Forschung und Ausarbeitung zu meinem Promotionsvorhaben unterstützte er mich stets in besonderem Maße.

Weiterhin danke ich meinem Zweitgutachter Herrn Prof. Dr.-Ing. Martin Radetzki von der Universität Stuttgart für die Begutachtung meiner Arbeit und die Teilnahme an meiner Disputation.

Für die stets angenehme Arbeitsatmosphäre bedanke ich mich bei den Mitarbeitern des Instituts für Angewandt Mikroelektronik und Datentechnik. Einen großen Einfluss auf mein Denken und Handeln während der Zeit am Institut hatten vor allem Herr Dr. Kai Neubauer und Herr Dr. Florian Grützmacher. Mein besonderer Dank gilt hier Herrn Dr. Nils Büscher für die hervorragende und inspirierende Zusammenarbeit über die letzten Jahre.

Besonders möchte ich mich auch bei meiner Frau Jana Gis für die Unterstützung und die stets aufmunternden Worte in der Zeit der Promotion bedanken.

Zusammenfassung

Die Digitalisierung und Automatisierung unterschiedlichster Prozesse ist ein Kernthema unserer modernen Gesellschaft. Dies betrifft sowohl unser alltägliches Leben mit automatisierten Gebäuden und intelligenten Assistenten als auch das industrielle Umfeld mit intelligenten Fabriken und autonomen Systemen. All diese Anwendungen haben eine Gemeinsamkeit; um mit der physikalischen Welt zu interagieren, benötigen sie eine Vielzahl von Sensoren. Aufgrund der immer höheren Integration von Mikroprozessoren hat sich der Markt der Sensoren gewandelt. Immer mehr Sensoren werden als sogenannte intelligente Sensoren gefertigt. Das bedeutet, das die Sensoren neben ihren Sensorelementen integrierte Recheneinheiten und Speicherelemente besitzen. Dies ermöglicht es u.a. durch Firmware die ursprünglichen Sensordaten bereits auf dem intelligenten Sensor weiterzuverarbeiten. Die Erstellung von Firmware für diese Systeme stellt Entwickler allerdings vor besondere Herausforderungen.

Diese Arbeit stellt Konzepte und Implementierungen vor, welche die Erstellung von Firmware für intelligente Sensoren komfortabler und effizienter gestalten. Aufgrund der Interaktion mit der physikalischen Welt ist es besonders herausfordernd, Firmware und Algorithmen, welche abhängig von den Sensordaten sind, zu validieren und zu testen. Die vorgestellte Sensor-in-the-Loop (SiL)-Architektur adressiert dieses Problem. Ziel ist es, die Vorteile von State-of-the-Art Entwicklungsmethoden für intelligente Sensoren zu kombinieren, um einen effizienten und komfortablen Entwicklungsprozess bereitzustellen. Das vorgestellte System ermöglicht es, wiederholbare und reproduzierbare Tests direkt auf der jeweiligen Sensorhardware auszuführen. Die Verwendung des SiL-Ansatzes ermöglicht dabei neben der Untersuchung der funktionalen Parameter auch eine Betrachtung von extrafunktionalen Parametern des Sensorsystems.

Das vorgestellte Konzept wurde unter realen Bedingungen getestet und eingesetzt, um verschiedene auf Sensordaten beruhende Algorithmen zu untersuchen und miteinander zu vergleichen. Besonders die Untersuchung der extrafunktionalen Eigenschaften wurde erst durch die Verwendung des vorgestellten Konzeptes ermöglicht. Weiterhin werden Ideen und Konzepte präsentiert, welche die SiL-Architektur erweitern. Durch die vorgestellten Erweiterungen wird die Effizienz und der Komfort des Systems weiter gesteigert. So wird zum einen die Erweiterung und Manipulation von Sensordaten ermöglicht, um spezielle Testfälle für Sensoralgorithmen zu generieren. Des Weiteren wird eine Erweiterung vorgestellt, welche die Untersuchung und Berücksichtigung des Energieverbrauchs erlaubt.

Abstract

The digitalization and automation of a wide variety of processes is a core topic of our modern society. This affects both our everyday lives with automated buildings and intelligent assistants as well as the industrial environment with smart factories and autonomous systems. All these applications have one thing in common; in order to interact with the physical world they need a variety of sensors. Due to the ever-increasing integration of microprocessors, the market of sensors has changed. More and more sensors are being manufactured as so-called smart sensors. This means that the sensors have integrated computing units and memory elements in addition to their sensing elements. This makes it possible, among other things, to further process the original sensor data already on the intelligent sensor by means of firmware. The creation of firmware for these systems poses special challenges to developers.

This thesis presents concepts and implementations that make the creation of firmware for smart sensors more convenient and efficient. Due to the interaction with the physical world, it is particularly challenging to validate and test firmware and algorithms that depend on sensor data. The presented Sensor-in-the-Loop architecture addresses this problem. The goal is to combine the advantages of state-of-the-art development methods for smart sensors to provide an efficient and comfortable development process. The presented system allows to perform repeatable and reproducible tests directly on the respective sensor hardware. The use of the SiL approach allows not only the investigation of functional parameters but also the consideration of extra-functional parameters of the sensor system.

The presented concept was tested and used under real conditions to investigate and compare different algorithms based on sensor data. Especially the investigation of the extra-functional properties was only made possible by using the presented concept. Furthermore, ideas and concepts are presented that extend the SiL architecture. The extensions presented further increase the efficiency and convenience of the system. On the one hand, the augmentation and manipulation of sensor data is enabled to generate special test cases for sensor algorithms. Furthermore, an extension is presented that allows the investigation and consideration of the extra-functional property of energy consumption.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADC . . . . . Analog to Digital Converter

AHRS . . . . . Attitude Heading Reference System

ASIP . . . anwendungsspezifischen Instruktionssatz Prozessoren (*Application-specific instruction-set processor*)

CI . . . . . Kommunikationsinterface (*Communication Interface*)

CSV . . . . . Comma Separated Values

DoF . . . . . Degrees of Freedom (*Degrees of Freedom*)

DSP . . . . . digitaler Signalprozessor (*Digital Signal Processor*)

FB . . . . . Funktionsblock

FIFO . . . . . First In First Out

FPGA . . . . . Field Programmable Gate Array

Gage R&R . . . . . Gage Repeatability and Reproducibility

GPIO . . . . . General Purpose Input Output

GPT . . . . . General Purpose Timer

GUI . . . . . Grafische Benutzeroberfläche (*Graphical User Interface*)

HiL . . . . . Hardware-in-the-Loop

I²C . . . . . Inter-Integrated Circuit

IDE . . . integrierte Entwicklungsumgebung (*Integrated Developement Environment*)

IMU . . . . . Inertiale Messeinheit (*Inertial Measurement Unit*)

Jcc . . . . . Cycle-to-Cycle-Jitter

Jper . . . . . Periodic-Jitter

Jpp . . . . . peak-to-peak Cycle-to-Cycle-Jitter

JTAG . . . . . Joint Test Action Group

KI . . . künstliche Intelligenz (*artificial intelligence*)
MARG . . . Magnetic, Angular Rate and Gravity
MEMS . . . mikro-elektro-mechanisches System (*Micro-Electro-Mechanical System*)
ODR . . . Ausgangsdatenrate (*Output Data Rate*)
PCB . . . Printed Circuit Board
PM . . . Energiemodel (*Power Model*)
RAM . . . Random-Access Memory
RTT . . . Real Time Transfer
SiL . . . Sensor-in-the-Loop
SiP . . . System in Package
SoC . . . System On Chip
SoM . . . System on Module
SPI . . . Serial Peripheral Interface
SPU . . . Sensor-Prozessor-Einheit (*Sensor Processing Unit*)
SWD . . . Serial Wire Debug Interface
UART . . . Universal Asynchronous Receiver Transmitter
UAV . . . Unmanned Aerial Vehicle
VP . . . virtueller Prototyp (*virtual prototype*)
µC . . . Mikrocontroller (*Microcontroller*)

Publikationen

Author

[1] D. Gis, C. Haubelt und N. Büscher, „Method for checking, evaluation, and/or error diagnosis of a sensor system, sensor system, and system Method for checking, evaluation, and/or error diagnosis of a sensor system, sensor system, and system“, US-Pat. US20220095022A1, März 2022,

Abstract: A method for checking, evaluation, and/or error diagnosis of a sensor system. The sensor system includes at least one sensor unit and an internal data processing unit. The sensor unit outputs sensor data as a function of a physical stimulus and the internal data processing unit is configured to process the sensor data output by the sensor unit to form output data. The sensor data output by the sensor unit is read out from the sensor system and recorded by an external processor unit in a recording step. The recorded sensor data are fed by the external processor unit into the internal data processing unit in an injection step. The fed-in sensor data are processed by the internal data processing unit in a processing step. A sensor system, and a system made up of a sensor system and a processor unit are also described.

[2] D. Gis, C. Haubelt und N. Büscher, „Verfahren zur Überprüfung, Evaluation und/oder Fehlerdiagnose eines Sensorsystems, Sensorsystem und System“, dt. Pat. DE102021200244A1, März 2022,

Abstract: Es wird ein Verfahren zur Überprüfung, Evaluation und/oder Fehlerdiagnose eines Sensorsystems (1), wobei das Sensorsystem (1) mindestens eine Sensoreinheit (2) und eine interne Datenverarbeitungseinheit (3) aufweist, wobei die Sensoreinheit (2) in Abhängigkeit eines physikalischen Stimulus Sensordaten ausgibt und die interne Datenverarbeitungseinheit (3) dazu konfiguriert ist, die von der Sensoreinheit (2) ausgegebenen Sensordaten zu Ausgangsdaten zu verarbeiten, wobei die von der Sensoreinheit (2) ausgegebenen Sensordaten in einem Aufzeichnungsschritt durch eine externe Rechnereinrichtung (5) aus dem Sensorsystem (1) ausgelesen und aufgezeichnet werden, die aufgezeichneten Sensordaten in einem Injektionsschritt durch die externe Rechnereinrichtung (5) in die interne Datenverarbeitungseinheit (3) eingespeist werden, und die eingespeisten Sensordaten in einem Verarbeitungsschritt von der internen Datenverarbeitungseinheit (3) verarbeitet werden.Weiterhin wird ein Sensorsystem (1) und eine System aus einem Sensorsystem (1) und einer Rechnereinrichtung (5) vorgeschlagen.

[3] D. Gis, N. Büscher und C. Haubelt, „Real-Time Power Analysis of Smart Sensors Using Advanced Debugging Methods“, *Micromachines*, Jg. 12, Nr. 11, 2021, ISSN: 2072-666X. DOI: 10.3390/mi12111276. Adresse: https://www.mdpi.com/2072-666X/12/11/1276,

Abstract: To achieve a good estimate of the power consumption of an embedded system, including its firmware, is a crucial step in the development of systems with a severely constrained power supply. This is especially true for cases where the device is powered by a small battery or through energy harvesting. The state-of-the-art approaches to measure or estimate the power consumption are formal methods, using power debugging tools with the real hardware or simulation based estimations. In the work at hand, a novel method to estimate the power consumption is proposed, it utilizes the sensor-in-the-loop architecture and enhancing it with a power estimation functionality. The proposed method combines the benefits of former methods, allowing for run-time analysis of the power-consumption in a reproducible way using recorded data without the need for power debugging hardware. In the experiments it is shown that, once set up, the proposed method is able to estimate the power consumption with an error of less than 1 % compared to a power debugging hardware. Thus, the proposed method provides a reliable and fast way to estimate the systems power consumption.

[4] D. Gis, N. Büscher und C. Haubelt, „Investigation of Timing Behavior and Jitter in a Smart Inertial Sensor Debugging Architecture“, *Sensors*, Jg. 21, Nr. 14, S. 4675, Juli 2021. DOI: 10.3390/s21144675,

Abstract: Due to upcoming higher integration levels of microprocessors, the market of inertial sensors has changed in the last few years. Smart inertial sensors are becoming more and more important. This type of sensor offers the benefit of implementing sensor-processing tasks directly on the sensor hardware. The software development on such sensors is quite challenging. In this article, we propose an approach for using prerecorded sensor data during the development process to test and evaluate the functionality and timing of the sensor firmware in a repeatable and reproducible way on the actual hardware. Our proposed Sensor-in-the-Loop architecture enables the developer to inject sensor data during the debugging process directly into the sensor hardware in real time. As the timing becomes more critical in future smart sensor applications, we investigate the timing behavior of our approach with respect to timing and jitter. The implemented approach can inject data of three 3-DOF sensors at 1.6 kHz. Furthermore, the jitter shown in our proposed sampling method is at least three times lower than using real sensor data. To prove the statistical significance of our experiments, we use a Gage R&R analysis, extended by the assessment of confidence intervals of our data.

[5] D. Gis, N. Büscher und C. Haubelt, „Advanced Debugging Architecture for Smart Inertial Sensors using Sensor-in-the-Loop“, in *2020 International Workshop on Rapid System Prototyping (RSP)*, IEEE, Sep. 2020. DOI: 10.1109/rsp51120.2020.9244851,

Abstract: Smart inertial sensors have emerged in the last years enabling developers to implement sensor fusion or tasks like gesture detection directly in the sensor hardware and thus reducing the processing latency and energy consumption. The development of software for

smart inertial sensors however faces difficulties from the limited hardware capabilities of the µC hardware and the lack of options to conduct reproducible tests directly on the hardware. We propose a Sensor-in-the-Loop architecture that allows a developer to record data from a smart sensor and inject the recorded data back into the sensor at a later time to evaluate the performance of the software in a reproducible way. With the proposed architecture it is possible to examine the performance and computational load on real hardware with recorded data thus allowing developers to optimize and improve the software in a more targeted way. In our implementation, the memory overhead for the code instrumentalization is just 0.63 %. The used RTT interface is at least four times faster than the regular SPI sensor interface. In experiments, we show that our proposed architecture is able to record and inject data of up to three 3-DOF sensors with 1.6 kHz sampling frequency in real time.

[6] D. Gis, A. Fink und H. Beikirch, „Bluetooth Enabled MARG Platform for Real-Time Orientation Sensing“, in *6th IEEE Int. Conf. on Indoor Positioning and Indoor Navigation (IPIN)*, 2015, S. 1–2,

Abstract: The knowledge of the exact orientation is the basis of any navigation system based on inertial measurements. The online processing of the magnetic angular rate and gravity (MARG) sensor as a hybrid IMU, especially the data fusion part, are challenging with limited processing power. We present an adaption of a PC-based offline fusion approach to a Cortex-M microcontroller platform with real-time orientation sensing with quaternion representation. The gradient descent method is used to fuse the data of accelerometer and magnetometer. The output is used to correct the predicted orientation given by a tri-axis gyroscope in a Kalman filter structure. By using code optimization and the IEEE 754-compliant hardware floating point unit (FPU) of the Cortex M4, an orientation accuracy around 1°/h with an update rate of 1 kHz has been achieved. The orientation data are easily accessible through a Bluetooth interface.

Co-Author

[7] N. Büscher, D. Gis, J. P. Wolff und C. Haubelt, „Data Augmentation Framework for Smart Sensor System Development Using the Sensor-in-the-Loop Prototyping Platform“, in *2021 IEEE International Workshop on Rapid System Prototyping (RSP)*, IEEE, Okt. 2021. DOI: 10.1109/RSP53691.2021.9806209,
Abstract: Sensor sub-systems are becoming more complex as they increasingly take on various tasks, from signal processing to pattern recognition. This off-loading of continuous processes can decrease power consumption of the entire system, as the application processor of e.g. a smartwatch can spend more time in low-power modes. The design and validation of these sensor subsystems are thus becoming more difficult and time consuming. This development process relies heavily on the availability of suitable sensor signals for testing. We propose a modular component-based framework to generate a multitude of tests using either prerecorded sensor signals or artificial signals to allow for a both faster and more thorough development process and prototyping. Using the recently proposed Sensor-in-the-Loop architecture, the work at hand demonstrates not only the ability to test and develop the software offline in a simulation, but to also use the augmented sensor signal data directly on the hardware prototype at run-time. The framework can be used in various testing- and development setups to simulate sensor characteristics, processing steps and possible errors. We show, based on three comprehensive examples, that the proposed framework is able to simulate the common errors found in inertial MEMS sensors, generate signal traces to develop, train, and evaluate gesture recognition algorithms, and simulate pre-processing steps in order to evaluate their feasibility before they are implemented in the sensor firmware.

[8] R. Dorsch, C. Haubelt, S. Stieber, D. Gis und N. Büscher, „Verfahren zum Verarbeiten der Sensorsignale einer Drehratensensoranordnung, Vorrichtung und Verfahren zum Betreiben der Vorrichtung“, DE102020205943B3, Mai 2021,
Abstract: Die Erfindung betrifft ein Verfahren zum Verarbeiten der Sensorsignale einer Drehratensensoranordnung mit mindestens drei Drehratensensorkomponenten, die Sensorsignale g_x, g_y, g_z für drei linear unabhängige Raumrichtungen in einem Sensorbezugssystem liefern, bei dem ein Ausgangssignal mit zwei Signalkomponenten g'_y, g'_z für zwei linear unabhängige Raumrichtungen eines absoluten Bezugssystems erzeugt wird, indem zumindest die Sensorsignale g_y, g_z für zwei der drei linear unabhängigen Raumrichtungen des Sensorbezugssystems in das absolute Bezugssystem transformiert werden, wodurch eine Verdrehung ϕ der Drehratensensoranordnung um die dritte linear unabhängige Raumrichtung des Sensorbezugssystems zumindest teilweise kompensiert wird. Der Kompensation der Verdrehung ϕ der Drehratensensoranordnung um die dritte linear unabhängige Raumrichtung des Sensorbezugssystems wird bei der Transformation der zumindest zwei Sensorsignale g_y, g_z zumindest das dritte Sensorsignal g_x für die dritte linear unabhängige Raumrichtung des Sensorbezugssystems zugrunde gelegt.

[9] N. Büscher, D. Gis, V. Kühn und C. Haubelt, „On the Functional and Extra-Functional Properties of IMU Fusion Algorithms for Body-Worn Smart Sensors“, *Sensors*, Jg. 21, Nr. 8, S. 2747, Apr. 2021. DOI: 10.3390/s21082747,

Abstract: In this work, four sensor fusion algorithms for inertial measurement unit data to determine the orientation of a device are assessed regarding their usability in a hardware restricted environment such as body-worn sensor nodes. The assessment is done for both the functional and the extra-functional properties in the context of human operated devices. The four algorithms are implemented in three data formats: 32-bit floating-point, 32-bit fixed-point and 16-bit fixed-point and compared regarding code size, computational effort, and fusion quality. Code size and computational effort are evaluated on an ARM Cortex M0+. For the assessment of the functional properties, the sensor fusion output is compared to a camera generated reference and analyzed in an extensive statistical analysis to determine how data format, algorithm, and human interaction influence the quality of the sensor fusion. Our experiments show that using fixed-point arithmetic can significantly decrease the computational complexity while still maintaining a high fusion quality and all four algorithms are applicable for applications with human interaction.

[10] N. Büscher, D. Gis, S. Stieber und C. Haubelt, „Multi-aspect Evaluation Method for Digital Pointing Devices“, in *Proceedings of the 9th International Conference on Pervasive and Embedded Computing and Communication Systems*, SCITEPRESS - Science und Technology Publications, 2019. DOI: 10.5220/0008355701280135,

Abstract: For decades the computer mouse has been used as the most common input device for laptops and computersalike. However for speeches a presentation remote with a laser pointer was used because they allowed the pre-senter more freedom. With the emergence of small and lightweight inertial sensors, a new type of presentationremotes becomes popular. These remotes use inertial sensors to move a digital pointer allowing presenters toshow things on more than one screen or use enhancement methods like highlighting a region. Using inertialsensors however proves to be a difficult task and can lead to problems with the usability of such devices.When developing such systems, the designer faces the problem that no method for quantifying the usability ofpointing devices based on inertial sensors is available. In the paper at hand, we propose an evaluation methodconsisting of three different tests to assess the manageability, speed and precision of digital pointing devicesfor a measurable comparison. Additionally, we conducted an evaluation to show that our tests reflect the sub-jective assessment from the users. Our quantitative test results showed a strong correlation to the qualitativesubjective assessment from the users.

[11] J. Rudolf, D. Gis, S. Stieber, C. Haubelt und R. Dorsch, „SystemC Power Profiling for IoT Device Firmware using Runtime Configurable Models“, in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, Juni 2019. DOI: 10.1109/meco.2019.8759994,

Abstract: In IoT domain energy aware firmware development is critical for applications that run on mobile or battery constrained devices. Virtual system prototypes (VSP) empower

developers to assess the application power consumption behavior before actual hardware prototypes become available. The SystemC modeling language has become the widely adopted industry standard for the implementation of such VSPs. In this paper, we present a novel approach for extending SystemC based VSPs with pluggable, pre-compiled power models that can be configured during runtime to generate accurate power profiles for the simulated firmware. The necessary modifications to the VSP are kept minimal. We demonstrate the application of our approach by instrumenting a pre-existing SystemC model for a state-of-the-art MEMS-based inertial sensor with a power model and show that the generated power profile estimation matches closely the energy consumption measured from its hardware prototype. As an additional advantage of our proposed precompiled approach, manufactures can ship their power models to costumers without disclosing implementation IP.

[12] J.-P. Wolff, S. Stieber, F. Holzke, D. Gis, C. Haubelt, P. Lepidis und J. Meier, „Improving Pedestrian Dead Reckoning using Likely and Unlikely Paths“, in *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, IEEE, März 2018. DOI: 10.1109/upinlbs.2018.8559709,

Abstract: Pedestrian Dead Reckoning is a method estimating a persons path from a known starting point based on length and direction of all performed steps. Measuring these parameters, e.g. using inertial sensors, introduces small errors that accumulate quickly into large distance errors. Knowledge of a buildings geography may reduce these errors as it can be used to keep the estimated position from moving through walls and onto likely paths. In this paper, we use building maps to improve localization based on a single foot-mounted inertial sensor. We show how correction algorithms using likely and unlikely paths can rectify errors intrinsic to pedestrian dead reckoning tasks and discuss restrictions and disadvantages of these algorithms. Our quantitative results show an endpoint accuracy improvement of up to 60% when using likely paths and 23% when using unlikely paths. However, both approaches can also decrease accuracy in certain scenarios. We identify those scenarios and offer further ideas for improving Pedestrian Dead Reckoning methods.

1 Einleitung

In unserer modernen Welt spielen Sensoren in vielen Bereichen eine wichtige Rolle. Sie bestimmen die Temperatur und messen die Luftqualität in unseren Wohnräumen, überwachen Vitalparameter und detektieren Bewegungen. Auch im Straßenverkehr und der Industrie sind Sensoren eine wichtige Komponente, um die Effizienz zu steigern und die Sicherheit zu gewährleisten. Durch die fortschreitende technische Miniaturisierung ist es möglich, Sensoren in einer großen Anzahl von Prozessen des privaten und wirtschaftlichen Lebens zu integrieren. Häufig sind diese Sensoren so klein und gut integriert, dass sie von uns gar nicht mehr wahrgenommen werden.

Vor allem im Bereich der mikro-elektro-mechanischen Systeme (Micro-Electro-Mechanical Systems, MEMSs) ist diese fortschreitende Miniaturisierung und die dadurch mögliche hohe Integration in technische Systeme sehr deutlich. Dies ermöglicht es, eine große Anzahl an Sensoren in Geräten und Systemen zur Messung verschiedenster physikalischer Größen einzusetzen. Zugleich werden die einzelnen Sensoren immer komplexer. Die Fortschritte in der Entwicklung von hochintegrierten und platzsparenden Recheneinheiten ermöglichen eine neue Art von Sensoren, die sogenannten *intelligenten Sensoren* oft auch als »Smart Sensors« bezeichnet. Bei dieser Art von Sensoren sind neben der eigentlichen Sensoreinheit zusätzlich noch eine Recheneinheit und Speicherelemente in den Sensor integriert. Dies ermöglicht dem Entwickler der Sensoren Berechnungen wie etwa Filteralgorithmen, direkt auf dem Sensor durchzuführen.

Die intelligenten Sensoren bieten Sensorentwicklern und auch Entwicklern von Sensorsystemen entscheidende Vorteile gegenüber den klassischen Sensorelementen. So ist es beispielsweise möglich, die Rohdaten des Sensorelementes vorzuverarbeiten bevor Sie an eine weitere Komponente des Systems übertragen werden. Dies führt unter Umständen zu einer Datenkompression vor der Übertragung, welche den Kommunikationsaufwand im System minimiert. Weiterhin kann durch die direkte Verarbeitung oftmals auch die Latenz für die Erkennung von bestimmten Ereignissen in den Sensordaten reduziert werden. Durch die hohe Effizienz der integrierten Recheneinheiten kann regelmäßig auch der Energiebedarf des Sensorsystems signifikant reduziert werden.

Durch die steigende Komplexität der intelligenten Sensoren stehen Entwickler dieser Systeme aber auch vor neuen Herausforderungen bei der Umsetzung und Entwicklung. Die Entwicklung von Software für die internen Recheneinheiten, die oft auch als »Firmware« bezeichnet wird, stellt besondere Anforderungen an die Entwickler.

1.1 Motivation und Zielsetzung

Die größte Herausforderung bei der Entwicklung von Firmware für intelligente Sensoren besteht im Testen und Verifizieren der implementierten Funktionalität. Dabei ist es oft notwendig das gesamte Sensorsystem in den Tests zu berücksichtigen. Diese Tests können in unterschiedlichen Abstraktions- und Detailgraden durchgeführt werden. So ist es für manche Anwendungsfälle ausreichend ausschließlich die Funktionalität des erstellten Algorithmus zu überprüfen. Für andere Anwendungsfälle kann es auch notwendig sein, zusätzlich extrafunktionale Eigenschaften zu berücksichtigen. Besonders bei sicherheitskritischen Anwendungen oder stark ressourcenbeschränkten Systemen kann dies von großem Interesse sein. Wichtige extrafunktionale Eigenschaften sind hier zum Beispiel die Laufzeit von Programmfunktionen und Algorithmen, die Größe der erstellten Firmware im Speicher oder auch der Energiebedarf des Sensorsystems während der Ausführung. Diese Betrachtungen können bis hin zur Auswertung der physikalischen Eigenschaften des Sensorsystems reichen.

Um das Testen der Sensorfirmware durchzuführen, existieren im Wesentlichen drei Arten von Testmethoden. Die erste Methode überprüft lediglich den erstellten Algorithmus auf die korrekte Ausführung im Sinne der Funktionalität. Dies geschieht typischerweise nicht in der implementierten Firmware selbst, sondern oftmals in einer Modellierungs- oder Spezifikationssprache z.B. *Matlab*- oder Python-Scripten. Hierbei wird ausschließlich die Funktionsweise des Algorithmus überprüft, es kann aber keine Aussage über die das Verhalten der Firmware auf der später verwendeten Architektur getroffen werden. Die zweite Methode, das sogenannten *Virtual Prototyping*, benutzt Simulationsmodelle der Sensorhardware, um die erstellte Firmware zu testen. Diese Methode kann eine bessere Aussage über das spätere Verhalten auf der echten Hardware treffen. Dabei kann bei entsprechender Bit-genauen Modellierung der Hardware sogar die Ausführung der finalen Firmware auf der simulierten Hardware emuliert werden. Leider ist es nicht möglich, den Sensor insbesondere in seinen physikalischen Eigenschaften bis in jedes Detail zu modellieren, da der zeitliche Aufwand für eine Simulation drastisch steigt je mehr Details im Modell abgebildet sind. Diese Methode stellt somit einen guten Kompromiss zwischen Simulationsgeschwindigkeit und -genauigkeit dar, liefert aber nicht in jedem Fall eine ausreichende Genauigkeit. Bei der dritten Methode wird die Firmware direkt auf einem Hardwareprototypen ausgeführt und getestet. Dabei können sowohl funktionale als auch extrafunktionale Eigenschaften überprüft werden. Allerdings können diese Eigenschaften auch stark von den Eingabedaten abhängen. Bei Sensoren werden diese Daten von dem entsprechenden Sensorelement geliefert.

Hierin liegt genau das Problem bei den Hardwareprototypen, da bei diesen die exakte Wiederholung der Eingabewerte oft mit großem Aufwand einhergeht. Oftmals ist es gar nicht möglich, exakt die gleichen physikalischen Gegebenheiten wieder herzustellen.

Somit ist es bei dieser Methode extrem schwierig oder sogar unmöglich, wiederholbare oder gar reproduzierbare Tests mit exakt denselben Sensordaten durchzuführen.

Jede dieser Methoden hat Ihre Vorteile aber auch einige entscheidende Nachteile beim Testen und Verifizieren von Sensorfirmware. Das Ziel dieser Arbeit ist das Erstellen einer Testarchitektur und Methodik, welche die entscheidenden Vorteile der oben genannten Methoden bietet aber weniger Limitierungen aufweist. Dafür wurde ein Ansatz entwickelt, welcher das Testen der Sensorfirmware auf der echten Hardware durchführt. Zusätzlich existiert die Möglichkeit aufgezeichnete oder generierte Sensordaten bei jedem Testlauf in den Sensorprototypen zu injizieren. Diese Methode schafft die Möglichkeit, wiederholbare und reproduzierbare Tests auf der echten Sensorhardware, und somit mit entsprechend hoher Genauigkeit, durchzuführen. Durch die vorgestellte Methodik wird es sogar möglich, in Realzeit funktionale und extrafunktionale Eigenschaften des Sensorsystems auf wiederholbare und reproduzierbare Weise zu überprüfen.

Der in dieser Arbeit vorgestellte Ansatz mit dem Namen „Sensor-in-the-Loop “ wurde anhand moderner MEMS-basierter intelligenter Inertialsensoren konzipiert und entwickelt. Er lässt sich aber auch auf andere Typen intelligenter Sensoren übertragen. Der Vorteil des hier vorgestellten Ansatzes ist bei Inertialsensoren aber besonders hoch, da es typischerweise sehr schwierig ist, ein und dieselbe Bewegung exakt gleich auszuführen. Dies gilt umso mehr, da viele der hier diskutieren Fallstudien sich mit der Bewegungserkennung mittels körpergetragener Sensorik beschäftigen.

1.2 Beiträge

Diese Arbeit wurde auf der Grundlage von vorausgegangen wissenschaftlichen Beiträgen erstellt. Diese Beiträge umfassen Veröffentlichungen auf Fachkonferenzen, Artikel in Fachzeitschriften und eingereichten sowie erteilten Patentschriften. Alle im Weiteren beschrieben Beiträge, beschäftigen sich mit der Entwicklung von Konzepten, Methoden und Systemen auf Basis von Sensoren. Dabei werden im speziellen vordergründig Inertialsensoren wie z.B. Beschleunigungssensoren oder Drehratensensoren benutzt. In den meisten dieser Beiträge kommen sogenannte intelligente Sensoren zum Einsatz. Diese Art von Sensoren besitzt neben der eigentlichen Sensoreinheit zusätzlich eine oder mehrere integrierte Recheneinheiten. Des Weiteren werden auch Anwendungen und Erweiterungen der zuvor erstellten Konzepte und Systeme untersucht und bewertet.

Die untersuchten Sensoren werden vorwiegend im Bereich der Unterhaltungselektronik eingesetzt. Bei dieser Art von Elektronik stehen vor allem Funktionalität und Benutzererfahrung im Vordergrund. Der Anwendungsbereich umfasst zum Beispiel Smartphones, Smartwatches, intelligente Bedienelemente oder ähnliches. Daher wurden auch die meisten Veröffentlichungen in diesem Kontext erstellt, die vorgestellten Konzepte und die gewonnenen Erkenntnisse sind allerdings nicht darauf beschränkt. So ist zum Beispiel

auch eine Nutzung im Automobilsektor oder der Avionik denkbar. Die vorgestellten Arbeiten werden in drei Themenbereiche unterteilt, diese Bereiche werden in Abbildung 1.1 dargestellt. In der Grafik ist sowohl eine zeitliche als auch eine thematische Einteilung der einzelnen Beiträge vorgenommen.

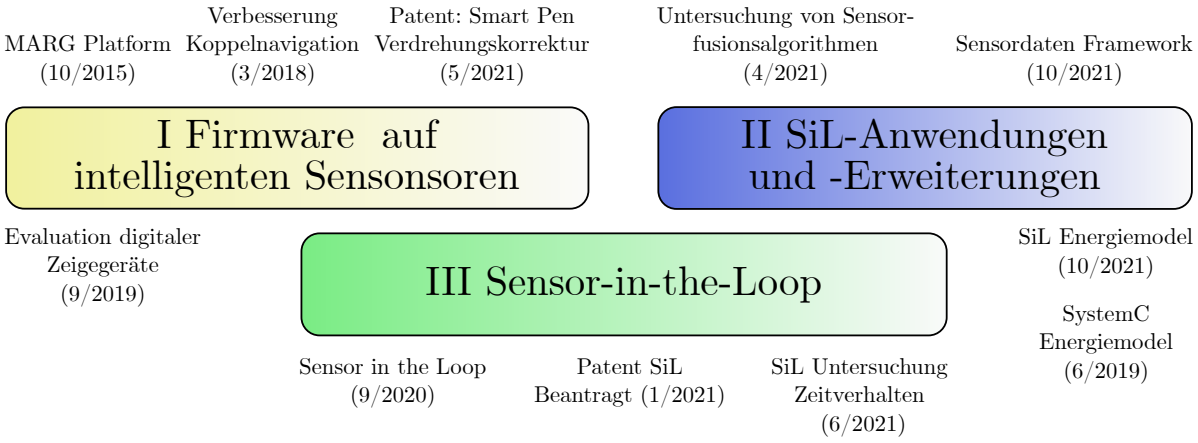


Abbildung 1.1: Übersicht der wissenschaftlichen Beiträge

Die drei Themenbereiche bilden eine chronologische Reihenfolge wobei zeitliche Überschneidungen in den Bereichen vorkommen. Den inhaltlich größten und wichtigsten Anteil an dieser Dissertation hat der Themenbereich II „Sensor-in-the-Loop (SiL)“. Direkt gefolgt und ergänzt durch den dritten Themenbereich „SiL Anwendungen und Erweiterungen“.

Der Themenbereich I „Firmware auf intelligenten Sensoren“ befasst sich mit der Entwicklung von Konzepten und Methoden basierend auf Sensorhardware oder deren Daten. Dabei ist es eine Sammlung unterschiedlicher Beiträge von der Entwicklung eigener Sensorhardware bis zur Software zur Bewertung entwickelter Handgestenalgorithmen. Während der Arbeit in diesem Bereich wurden allgemeine Konzepte und Probleme bei der Arbeit mit intelligenten Sensoren erkannt und gesammelt. Dieses Wissen führte zu Ideen, um die Entwicklung auf dieser Art von Sensoren effizienter und einfacher zugänglich zu gestalten.

Im Bereich II „Sensor-in-the-Loop (SiL)“ sind die Beiträge gesammelt, welche sich mit dem eigentlichen Entwicklungsprozess von Firmware auf intelligenten Sensoren beschäftigen. Dabei wurde versucht, die im ersten Themenbereich gewonnen Erkenntnisse umzusetzen und die erkannten Probleme zu bewältigen. Dieser und auch Teile des nächsten Bereiches umfassen den eigentlichen Kern dieser Abhandlung. Die hier vorgestellten Beiträge beschreiben konzeptionell die entwickelte Debugging-Architektur für intelligente Sensoren. Weiterhin wird eine Umsetzung des Konzepts auf ihre Eignung systematisch anhand unterschiedlicher Parameter hin untersucht.

Der Themenbereich III „SiL-Anwendungen und -Erweiterungen“ beinhaltet Beiträge, welche sich an den zweiten Themenbereich anschließen und diesen erweitern. Diese basieren auf Veröffentlichungen, die den entwickelten Sensor-in-the-Loop (SiL) Ansatz benutzen, um verschieden Sensordaten basierte Algorithmen reproduzierbar auf Ihre Eigenschaften hin zu untersuchen. Weiterhin werden Beiträge vorgestellt, welche die Erweiterungen der entwickelten Architektur vorschlagen und umsetzen.

Im ersten Beitrag der Kategorie I [B1] wurde eine Sensor-Plattform entwickelt, welche neben den eigentlichen Inertialsensoren auch eine dedizierte Recheneinheit und eine Bluetooth Kommunikationsschnittstelle besitzt. Weiterhin wurde eine Orientierungsschätzung auf Grundlage der drei vorhanden Sensoren, Drehratensensor, Beschleunigungssensor, und Magnetometer durchgeführt. Die Stabilität der Orientierungsschätzung ist mit 1°/h vergleichbar mit kommerziell verfügbaren, sehr viel kostenintensiveren Systemen vergleichbar. Die Arbeit wurde im Oktober 2015 auf der *6th IEEE Int. Conf. on Indoor Positioning and Indoor Navigation (IPIN)* vorgestellt. Diese Veröffentlichung wurde als federführender Autor verfasst.

Die zweite Veröffentlichung beschäftigt sich mit der Verbesserung der Koppelnavigation von Fußgängern in Gebäuden [B2]. Dabei werden zusätzlich zur Nutzung der Inertialsensoren, Kartendaten des Gebäudes genutzt, um die erhalten Position der Person und somit die Navigation zu verbessern. Die Arbeit wurde auf der *Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPIN-LBS)* Konferenz im März 2018 vorgestellt. Der Beitrag des Autors zu dieser Publikation bestand zum einen in der Unterstützung bei der Entwicklung von Sensor-Firmware und der Auswertung der Ergebnisse. Zum anderen wurde an der Erstellung der schriftlichen Abhandlung mitgewirkt.

Der nächste Beitrag schlägt eine Methode zur Untersuchung und Bewertung von digitalen Zeigegeräten vor [B3]. Dabei liegt der Fokus auf Zeigegeräten, die zur Umsetzung der Funktionalität Inertialsensordaten verwenden, wobei die Methodik nicht auf solche beschränkt ist. Die Ergebnisse der Metriken zur quantitativen Bewertung der Ergebnisse wurden mit subjektiven qualitativen Bewertungen von Versuchspersonen verglichen. Die Arbeit wurde auf der *Pervasive and Embedded Computing and Communication Systems (PECCS)* Konferenz im September 2019 veröffentlicht. Der Beitrag des Autors an dieser Arbeit lag in der gemeinsamen Erstellung der Konzepte, der Implementierung des Bewertungsframeworks und der Erstellung der schriftlichen Veröffentlichung.

Im Rahmen der Arbeit mit digitalen Zeigegeräten ist auch der letzte Beitrag in dieser Kategorie entstanden. Dabei handelt es sich um ein erteiltes Patent zur effizienten und energiesparenden Verdrehungskorrektur bei digitalen Zeigegeräten basierend auf Inertialsensorik [B4]. Das Patent wurde im Mai 2020 eingereicht und im Mai 2021 erteilt. Der Autor ist einer drei Haupterfinder dieses Patentes.

Bereich II „Sensor-in-the-Loop “ umfasst im Wesentlichen drei Veröffentlichungen und stellt den Hauptteil dieser Abhandlung dar. Der erste Beitrag beschreibt die konzeptio-

nelle Erstellung der SiL-Architektur [B5]. Es wird beschrieben, welche Voraussetzungen erfüllt sein müssen, um diesen Ansatz zu nutzen. Weiterhin werden die einzelnen Teile der Architektur, ihre Vorteile und Beschränkungen aufgezeigt. In dieser Veröffentlichung wird auch die erstellte Implementierung des Ansatzes beschrieben. Um die Implementierung auf Ihre Eignung zur Validierung von Sensorfirmware mit reproduzierbaren Daten zu untersuchen, wurden erste Untersuchungen des Zeitverhaltens angestellt. Der Beitrag wurde im September 2020 auf dem *2020 International Workshop on Rapid System Prototyping (RSP), IEEE* im Rahmen der *Embedded Systems Week* vorgestellt. Erstautor dieses zentralen Beitrags ist der Autor dieser Dissertation. Die Beiträge dieser Workshop-Veröffentlichung sind überwiegend diesem Autor zuzuordnen.

Um die in [B5] vorgestellten Konzepte detaillierter zu beschreiben und tiefer gehende Untersuchungen durchzuführen, wurde ein Fachzeitschriftenartikel erstellt [B6]. Dieser Artikel stellt noch einmal kurz die entwickelte SiL-Architektur und die erstellte Implementierung dieser vor. Der Hauptfokus des Artikels liegt in der detaillierten Betrachtung des Zeitverhaltens der erstellten Umsetzung. Es werden hierbei sowohl die eigentliche Ausführungszeit der Einspeisung als auch der zeitliche Jitter der SiL-Schnittstelle untersucht. Die detaillierte zeitliche Untersuchung des Ansatzes soll sicherstellen, dass eine wiederholbare und vielmehr reproduzierbare Untersuchung von Sensorfirmware mit den vorgestellten Ansatz möglich ist. Der Artikel wurde im Juni 2021 im *MDPI Sensors Journal* veröffentlicht. Erstautor und wesentlicher Beitragender zu dieser Arbeit ist wiederum der Autor dieser Dissertation.

Zusätzlich zu den wissenschaftlichen Veröffentlichungen des Sensor-in-the-Loop-Konzepts wurde eine Patentschrift erstellt. Dieses Patent umfasst das Konzept des erstellten Systems. Zusätzlich wird auch das gesamte Verfahren der reproduzierbar Einspeisung von Sensordaten während der gleichzeitigen Beobachtung der resultierenden Ergebnisse beansprucht. Das Verfahren beinhaltet weiterhin die Gewinnung von funktionalen und extrafunktionalen Eigenschaften des Systems während des Prozesses. Das Patent [B7] wurde im Januar 2021 in Deutschland beantragt. Zusätzlich wurde im September 2021 ein US-Patent beantragt [B8]. Die Patentschriften wurden im März 2022 in Deutschland, den USA und China veröffentlicht. Der Autor ist federführender Erfinder dieses Patents.

Im letzten Bereich „SiL-Anwendungen und -Erweiterungen “ wurden folgenden Veröffentlichungen erfolgreich vorgenommen, um die Anwendbarkeit des Ansatzes zu zeigen und diesen zu erweitern. Die Erste Veröffentlichung in diesem Bereich untersucht die funktionalen und extrafunktionalen Eigenschaften von verschieden Algorithmen zur Orientierungsschätzung aus Inertialsensordaten [B9]. Zur reproduzierbaren Untersuchung der Algorithmen direkt auf der Sensorhardware wurde hierbei das Sensor-in-the-Loop System benutzt. Die Arbeit wurde im April 2021 im *MDPI Sensors Jounal* veröffentlicht. Der Beitrag des Autors an dieser Veröffentlichung bestand in der gemeinsamen konzeptionellen Ausarbeitung der Tests, der Erstellung der Sensorfirmware als Basis für

die Untersuchung der Algorithmen auf der echten Hardware. Weiterhin wurden die Versuche zur Untersuchung der extrafunktionalen Eigenschaft wie Speicherverbrauch und Zeitverhalten mittels der SiL durchgeführt. Es wurde bei der Auswertung der Tests und der Erstellung des Manuskripts unterstützt.

Der nächste Beitrag erweiter die SiL-Implementierung um ein Framework zur künstlichen Erzeugung von Sensordaten und Stimuli [B10]. Weiterhin können aufgenommenen Sensorrohdaten vor einer Einspeisung in den Sensor durch Filterung oder Überlagerung bearbeitet werden. Der Beitrag wurde im Oktober 2021 auf dem *2021 International Workshop on Rapid System Prototyping (RSP), IEEE* im Rahmen der *Embedded Systems Week* vorgestellt. Der Beitrag des Autors an dieser Veröffentlichung ist die gemeinsame Konzepterstellung, und Auswahl der Versuchsreihen. Es wurde bei der Anbindung an das SiL-Framework unterstützt und Versuche mit diesem durchgeführt. Weiterhin ist eine Unterstützung bei der Auswertung der Ergebnisse und dem Erstellen der schriftlichen Ausarbeitung erfolgt.

Der letzte Beitrag erweitert das Sensor-in-the-Loop-Framework um ein Model zur Abschätzung des Energieverbrauches [B11]. Dabei baut es auf einer vorhergehenden Veröffentlichung auf [B12]. Diese nutzt ein Energiemodel, um die Energieverbrauchsschätzung für Sensoren während der Simulation über ein SystemC-Model des Sensors zu schätzen. Der Beitrag an diese Veröffentlichung war die gemeinsame konzeptionelle Erarbeitung, das Erstellen des experimentellen Aufbaus und die Auswertung der Experimente. Weiterhin wurde bei der Erstellung der schriftlichen und grafischen Ausarbeitung der Veröffentlichung unterstützt. Im Gegensatz zu der vorhergehenden Veröffentlichung, nutzt die aktuelle Veröffentlichung, die tatsächliche Sensorhardware in Kombination mit dem SiL-Ansatz. Dadurch wird eine Energieabschätzung während der Laufzeit, auf der echten Hardware, mit reproduzierbaren Ergebnissen erreicht. Die Funktionsweise und die Genauigkeit des geschätzten Energieverbrauchs wurde anhand eines komplexen Beispielszenarios untersucht und überprüft. Dieser Beitrag wurde im Oktober 2021 im *MDPI Micromashines Journal* veröffentlicht. Dieser Artikel wurde als federführender Autor verfasst.

1.3 Gliederung der Arbeit

Der Inhalt der vorliegenden Arbeit ist wie folgt gegliedert:

In Kapitel 2 werden die technischen und wissenschaftlichen Grundlagen erläutert. Diese Grundlagen dienen dem allgemein besseren Verständnis für die Thematik dieser Abhandlung.

Kapitel 3 beschäftigt sich mit dem aktuellen Stand der Technik und wissenschaftlichem Kontext dieser Abhandlung. Dabei wird der aktuelle Stand der Technik zur Entwicklung von Sensorfirmware auf intelligenten Sensoren beschrieben mit dieser Arbeit in Verbindung gebracht. Weiterhin wird die vorliegende Arbeit mit anderen wissenschaftlichen Veröffentlichungen in Kontext gesetzt. Es werden Ansätze und Ergebnisse aus aktuellen Publikationen vorgestellt. Diese werden mit den Ansätzen und Ergebnissen der vorliegen Arbeit verglichen. Dadurch wird der geschaffene wissenschaftliche Mehrwert der vorgestellten Entwicklung herausgestellt.

Im Kapitel 4 das allgemeine Vorgehen und der Prozess der Firmwareentwicklung auf intelligenten Sensoren erläutertet. Es wird die konventionelle best-practice Methodik vorgestellte und Ihre Nachteile beschrieben. Es folgt eine Beschreibung der Entwicklung unter Einfluss des Sensor-in-the-Loop-Konzeptes und deren Vorteile.

Das Kapitel 5 stellt die entwickelte Sensor-in-the-Loop-Architektur vor. Diese umfasst den Hauptbeitrag dieser Abhandlung welche die Entwicklung und wissenschaftliche Untersuchung der SiL-Architektur darstellt. Das Kapitel beschreibt zuerst den grundlegenden Aufbau und die Ideen zu SiL. Anschließend wird eine erstellte Implementierung des Konzeptes systematisch auf verschiedene Eigenschaften hin untersucht.

Kapitel 6 fasst verschiedene Anwendungen und Erweiterungen der vorgestellten SiL-Architektur zusammen. Es wird gezeigt, wie SiL benutzt werden kann, um funktionale und extrafunktionale Eigenschaften von Sensorfirmware reproduzierbar, auf echter Sensorhardware zu untersuchen. Weiterhin werden Erweiterungen vorgestellt, um den Mehrwert des entwickelten Konzeptes für Entwickler von Sensor-Firmware weiter zu steigern.

Abschließend werden in Kapitel 7 die vorgestellten Konzepte und Ergebnisse dieser Arbeit noch einmal zusammengefasst. Es wird eine Bewertung der erstellten Konzepte, Implementierungen und Ergebnisse durchgeführt. Dabei werden die Ergebnisse vor allem im wissenschaftlichen Kontext und im Vergleich mit dem bisherigen Stand der Technik bewertet. Das Kapitel schließt mit einem Ausblick auf weiter Forschungen und Entwicklungen, welche im Kontext dieser Arbeit möglich wären.

2 Grundlagen

Dieses Kapitel erläutert die notwendigen technischen und wissenschaftlichen Grundlagen. Diese Dissertation baut auf den hier dargestellten Grundlagen auf.

Diese Dissertation beschäftigt sich mit Möglichkeiten die Entwicklung von Firmware für intelligente Sensoren zu verbessern. Deshalb werden in den Grundlagen zu Beginn Sensoren im allgemeinen Sinne behandelt. Hauptsächlich wurden für die beispielhafte Implementierung und die Experimente Inertialsensoren wie z.B. Beschleunigungssensoren oder Gyroskope benutzt. Diese Art von Sensoren wird deshalb extra in den Grundlagen beschrieben. Um überhaupt eine Firmware-Entwicklung auf den Sensoren vornehmen zu können, müssen diese entsprechend mit einer Recheneinheit und Speicher bestückt sein. Diese Art von Sensoren werden auch als intelligente Sensoren bezeichnet. Computersysteme, welche in einen technischen Kontext eingebettet sind und mit der physikalischen Welt in Interaktion stehen, werden auch *eingebettete Systeme* genannt. Intelligente Sensoren können auch als solche Systeme angesehen werden, deshalb wird der Begriff vorher in den Grundlagen eingeführt. Grundlagen, die nur in einzelnen Kapiteln von Interesse sind, werden nicht hier, sondern direkt an entsprechender Stelle erläutert.

2.1 Sensoren

Elektronische Systeme finden in vielen Bereichen unseres täglichen Lebens Anwendung, um uns das Leben zu erleichtern oder bestimmte technische Anwendungen überhaupt erst zu ermöglichen. Dabei steuern sie Abläufe in der Industrie, regeln den Verkehr oder überwachen das Wetter. Diese mannigfaltigen Anwendungen der elektronischen Systeme werden erst durch die Interaktion dieser Systeme mit der Umwelt ermöglicht. Dabei spielen Sensoren eine entscheidende Rolle. Sie sind sozusagen die Sinnesorgane der elektronischen Systeme.

Man unterscheidet Sensoren in der Regel an der zu messenden physikalischen Größe, für welche der jeweilige Sensor geeignet ist. So werden zum Beispiel akustische Sensoren zur Messung von Tonhöhe und Lautstärke von optischen Sensoren unterschieden, welche etwa Lichtintensität oder die Farbe von Gegenständen erfassen. Für die meisten existierenden physikalischen Größen gibt es spezielle Sensoren, um diese zu erfassen.

Weitere typische Sensorarten sind Temperatursensoren, Drucksensoren, Konzentrationssensoren für bestimmte Gase oder auch Sensoren zu Messung der Beschleunigung oder der Drehrate.

2.1.1 MEMS-Sensoren

In vielen Bereichen, in denen eine große Anzahl von Sensoren benötigt wird und Faktoren wie Kosten und Energieeffizienz eine wichtige Rolle spielen, werden häufig MEMS-Sensoren eingesetzt. Bei dieser Technologie wird das Sensorelement ähnlich wie bei der Herstellung anderer Mikrochips über ein Lithographieverfahren direkt auf einem Siliziumelement gefertigt [A1]. Dies ermöglicht Sensoren mit sehr feinen Strukturen in großer Stückzahl herzustellen.

Durch die mögliche Miniaturisierung bei dieser Herstellungsform, können besonders energieeffiziente Sensoren produziert werden. Dies liegt nicht zuletzt daran, dass die zu bewegenden Elemente in den Sensoren mikroskopisch klein sind und dem entsprechend wenig Kraft für die Bewegung aufgewendet werden muss. Da das Fertigungsverfahren standardisiert ist und auf sogenannten Wafern von 200 mm bis 300 mm arbeitet, ist es möglich mit einem Durchgang mehrere hundert MEMS-Sensoren zu fertigen. Aufgrund der geringen Größe dieser Sensoren sind sie fast überall einzusetzen und das häufig in großer Stückzahl, um möglichst viele unterschiedliche Messgrößen an einem Ort aufzunehmen.

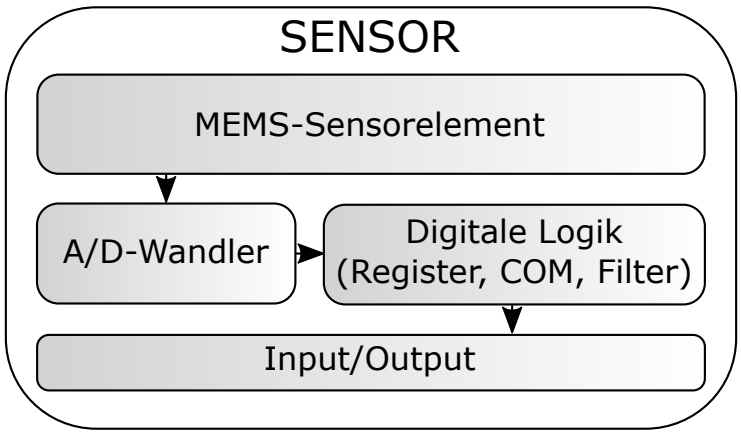


Abbildung 2.1: Schematische Darstellung eines MEMS-Sensors

Der schematische Aufbau eines typischen MEMS-Sensors ist in Abbildung 2.1 dargestellt. Dieser Sensor besteht aus dem eigentlichen Sensorelement, welches die Transformation der zu messenden physikalischen Größe in ein elektrisches Signal übernimmt. Um diese elektrische Größe, meistens eine Spannung von wenigen Millivolt, in digitalen Systemen

verwenden zu können wird diese von einem Analog-Digital-Wandler in ein digitales Signal (zeit- und wertdiskret) umgesetzt. Häufig sind in modernen Sensoren zusätzliche digitale Logikbausteine vorhanden. Diese ermöglichen es, die digitalen Messwerte zu speichern oder einfache fest vorgegebene Filteroperationen auf diesen anzuwenden. Des Weiteren sind diese Komponenten für die Bereitstellung einer Kommunikationsschnittstelle zuständig. Darüber lassen sich die Sensorwerte abrufen oder auch Sensor-Parameter wie etwa die Abtastzeit einstellen.

2.1.2 Inertialsensoren

Die Umsetzung des in dieser Dissertation vorgestellten Konzeptes und die experimentellen Untersuchungen des Ansatzes wurden weitestgehend an Inertialsensoren durchgeführt. Diese eignen sich sehr gut dafür, den implementierten Ansatz zu demonstrieren, da eine reproduzierbare Generierung von Sensordaten auf konventionellem Wege nahezu unmöglich ist. Der Ansatz lässt sich aber auch auf andere Sensortypen übertragen und ist nicht auf diese Art von Sensoren beschränkt.

Inertialsensoren beruhen auf dem Trägheitsprinzip von seismischen Massen. Dabei wird die Auslenkung einer Masse aufgrund einer von Außen auf das Gesamtsystem einwirkenden Kraft bestimmt. Diese Kraft kann zum Beispiel durch die Erdbeschleunigung hergerufen werden. Über die gemessene Auslenkung der seismischen Masse lässt sich dann die Erdbeschleunigung bestimmen. Eine weitere auf diesem Prinzip beruhende Sensorart bilden die Drehratensensoren oder auch Gyroskope. Häufig werden beide dieser Sensoren in einem gemeinsamen Sensorgehäuse untergebracht und in Kombination betrieben. Bei diesen Sensoren spricht man dann von sogenannten inertialen Messeinheiten oder häufiger von (Inertial Measurement Units) IMUs. Diese Kombination bietet den Vorteil, das sowohl translatorische als auch rotatorische Bewegungen des Sensors gemessen werden können. Des Weiteren kann dabei ein gemeinsamer Bezugspunkt für den Beschleunigungssensor und das Gyroskop angenommen werden, da sich beide räumlich sehr nahe beieinander befinden. Bei Bestimmungen von Drehungen über das verwendete Gyroskop, kann der zusätzliche Beschleunigungssensor zur Korrektur der Gyroskopdrift auf den horizontalen Achsen genutzt werden.

In vielen modernen Sensorsystemen wird zusätzlich zu den eingangs erwähnten Sensoren ein Magnetometer verwendet. Daher wird es auch oft zu den Inertialsensoren gezählt. Diese Klassifikation ist streng genommen nicht richtig, da die Messung der magnetischen Feldstärke in diesen Sensoren nicht auf dem Prinzip der Masseträgheit basiert. Bei einer Kombination aus all diesen drei Sensoren wird oft von einem Magnetic, Angular Rate and Gravity (MARG) Sensor oder System gesprochen. Eine zusätzliche Verwendung des Magnetometers ermöglicht die absolute Richtungsbestimmung des Systems in Bezug auf das Weltkoordinatensystem. Deshalb werden diese Systeme auch als Attitude Heading

Reference Systems (AHRSs) bezeichnet. Ein weiterer Vorteil ist die mögliche Korrektur des Gyroskopdrifts in allen drei Achsen. Durch die gleichzeitige Messung von drei Sensoren in jeweils drei Achsen ergibt sich ein neun dimensionaler Messvektor. Diese Dimensionalität des Messwertvektors führt zu der weiterhin gebräuchlichen Bezeichnung 9-DoF-Sensor. Nachfolgend werden die einzelnen in einem solchen System enthaltenen Sensoren einzeln vorgestellt.

Beschleunigungssensor

Ein Beschleunigungssensor oder auch Accelerometer, misst die durch eine Beschleunigung a resultierende Kraft F auf eine Masse m [A2]. Als Einheit für die Messwerte wird m/s^2 verwendet, es ist auch gebräuchlich die Einheit g zu verwenden welche ein Vielfaches der mit $9,080665\,m/s^2$ mittleren Erdbeschleunigung entspricht.

Die ausgeübte Kraft steht über das zweite Newton'sche Axiom mit der beschleunigten Masse in einer direkten Proportionalität.

$$F = m \cdot a \tag{2.1}$$

Bei den üblichen Beschleunigungssensoren wird die Beschleunigung über die ausgeübte Kraft auf eine seismische Masse bestimmt. Diese Masse ist federnd gelagert mit dem festen Teil (Gehäuse) des Sensors verbunden, dieses Prinzip findet auch Anwendung in den üblichen MEMS-basierten Accelerometern. Aufgrund dieses Aufbaus nach dem Feder-Masse-Prinzip kann über das Hookesche-Gesetz in Gleichung (2.2) die ausgeübte Kraft F aus der Auslenkung der Masse z bestimmt werden.

$$F = k \cdot z \tag{2.2}$$

Die Federkonstante k ist materialabhängig und über das System definiert. Unter Verwendung von Gleichung (2.1) und Gleichung (2.2) kann nun ein Eindeutig die Beschleunigung a über die Auslenkung der Masse z bestimmt werden. Die Konstanten k und m in Gleichung (2.3) sind über den Sensor definiert.

$$a = \frac{k}{m} \cdot z \tag{2.3}$$

Die schematische Darstellung eines Beschleunigungssensors in Abbildung 2.2 zeigt konzeptionell den möglichen Aufbau eines solchen Sensors. Dabei ist die seismische Masse federnd gelagert über die Anker mit dem festen Teil des Sensors verbunden. Ein solcher Aufbau lässt sich sehr gut mit MEMS-Fertigungsverfahren realisieren. Die Verschiebung

der Masse kann über die Kapazitätsänderung welche zwischen den festen und beweglichen Kondensatorelektroden entstehen bestimmt werden.

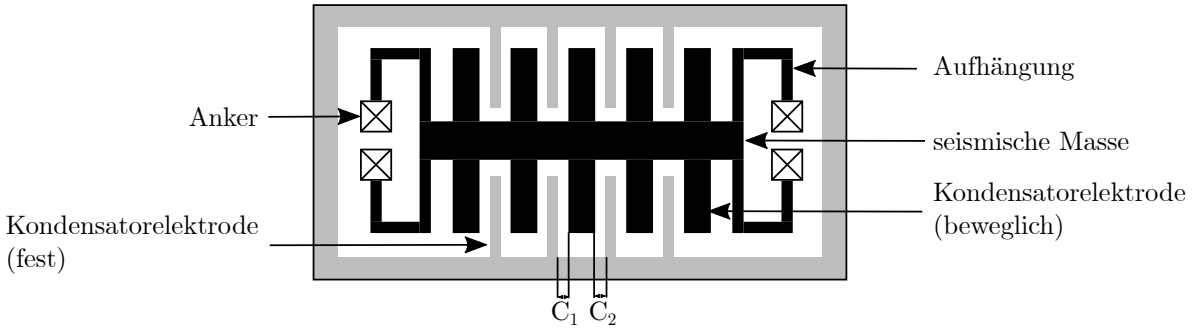


Abbildung 2.2: Kapazitiver Beschleunigungssensor nach [A3]

Drehratensensoren

Die Drehratensensoren, häufig auch als Gyroskope bezeichnet, messen die Rotationsgeschwindigkeit Ω um eine oder mehrere definierte Achsen in $^\circ/s$ [A2]. Eine alternativ in der Literatur zu findende Bezeichnung ist Winkelgeschwindigkeit, diese wird dann mit dem Formelzeichen ω angegeben. Im Wesentlichen existieren zwei Verfahren welche zur Bestimmung der Winkelgeschwindigkeit gebräuchlich sind.

Das erste Verfahren bestimmt die Winkelgeschwindigkeit mit optischen Methoden. Dabei wird der Sagnac-Effekt genutzt. Bei dieser Methode werden Lichtstrahlen über einen Spiegel oder Glasfaseroptiken in einer Kreisbahn geführt. Je nach Größe der anliegenden Rotationsgeschwindigkeit legen die Lichtstrahlen einen längeren oder kürzeren Weg zurück. Systeme, die nach diesem Prinzip arbeiten, sind hochgenau und unabhängig von einer auf sie ausgeübten linearen Beschleunigung. Daher finden sie häufig Anwendung in der Luft- und Raumfahrt oder im militärtechnischen Bereichen. Für den Konsumer- und Multimediabereich sind diese Systeme aufgrund ihrer Größe und Kostenintensität nicht geeignet.

Die zweite Möglichkeit die Winkelgeschwindigkeit zu messen nutzt die Corioliskraft. Dieses Prinzip wird auch bei MEMS-Gyroskopen eingesetzt.

In Abbildung 2.3 ist der schematische Aufbau eines Stimmgabelgyroskopes dargestellt. Die Darstellung zeigt aus Gründen der besseren Übersicht nur eine Achse, in realen Sensoren werden üblicherweise drei orthogonal zueinander angeordnete Achsen eingesetzt. Aktuelle in MEMS-Technologie gefertigt Drehratensensoren nutzen prinzipiell die gleichen Ansätze zum Messen der Winkelgeschwindigkeit. Die beiden seismischen Masse m werden gegensinnig radial zueinander in Schwingung versetzt. Wird das Gesamtsystem nun rotiert, resultiert daraus eine zusätzliche Schwingung. Diese schwingt tangential und

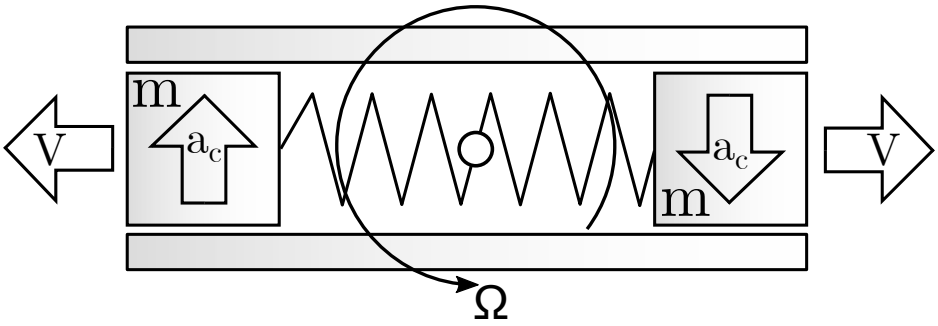


Abbildung 2.3: Schematische Darstellung einer Achse eines Drehratensensors nach [A2]

senkrecht zu den bisherigen Schwingungen. Die Beschleunigung dieser Schwingung wird durch die Corioliskraft verursacht. Zur Messung dieser Beschleunigung werden Akzelerometer verwendet.

$$\vec{a_c} = \frac{\vec{F_c}}{m} = 2\vec{v_{rel}} \times \vec{\Omega} \tag{2.4}$$

Die über die Beschleunigungssensoren gemessene Beschleunigung $\vec{a_c}$ kann über die Gleichung (2.4) in Relation gesetzt werden. Die Erregerschwingung bewegt die seismischen Massen mit einer relativen Geschwindigkeit von $\vec{v_{rel}}$. Diese Schwingung kann über eine entsprechende Dimensionierung der Federn von der signalbildenden Schwingung, verursacht durch die Coriolisbeschleunigung $\vec{a_c}$, entkoppelt werden. Die Einflussnahmen von über die Anregungsaktoren eingekoppelten Störquellen wird über dieses Verfahren reduziert. Dabei wird eine höhere Genauigkeit als bei anderen Systemen erzielt, bei denen Sensor- und Aktormode gekoppelt sind [A2].

Magnetometer

Magnetfeldsensoren sind keine Inertialsensoren im eigentlichen Sinn, da sie nicht auf dem Trägheitsprinzip von seismischen Massen beruhen. Diese Sensoren sind auch nicht Bestandteil von typischen IMUs. Häufig werden diese Sensoren in Kombination mit Beschleunigungssensoren und Gyroskopen eingesetzt. Diese Kombination wird dann als MARG oder 9-DoF Sensor bezeichnet. Diese Kombination dient oft der genauen Raumlageerkennung. Durch den Magnetfeldsensor kann der Fehler in der Lageerkennung verursacht durch die Gyroskopdrift minimiert werden. Besonders in der z-Achse was einer Drehung um den Yaw-Winkel Ψ entspricht, ist eine signifikante Verbesserung zu erwarten. Durch diese häufige gemeinsame Verwendung wird das Magnetometer oft zusammen mit den Inertialsensoren genannt.

Magnetfeldsensoren existieren in vielen unterschiedlichen Ausführungen und Typen. Dabei werden ganz unterschiedliche physikalische Effekte zu Messung des Magnetfeldes genutzt. Eine Gruppe der Magnetfeldsensoren stellen die Hall-Sensoren dar. Diese messen das Magnetfeld unter Nutzung des Hall-Effekts. Diese Sensoren werden an dieser Stelle nicht weiter erläutert, da sie in MEMS basierten integrierten IMUs selten Anwendung finden. Eine Beschreibung über das Funktionsprinzip dieser Sensoren finden sich beispielsweise in [A3] und in [A2].

Die magnetoresistiven Sensoren, bilden eine weitere Gruppe der Magnetfeldsensoren. Sie beruhen auf der Änderung des elektrischen Widerstandes in ferromagnetischen Materialien bei der Durchdringung magnetischer Felder. Diese Gruppe ist weiter unterteilbar nach unterschiedlichen Messverfahren zum Beispiel nach dem Giant Magneto Resistive (GMR) oder dem Anisotropic Magneto Resistive (AMR) Sensorprinzipien. In typischen MEMS-Sensoren finden häufig die AMR-Sensoren Anwendung, weshalb ihr Wirkprinzip anschließen beschrieben wird.

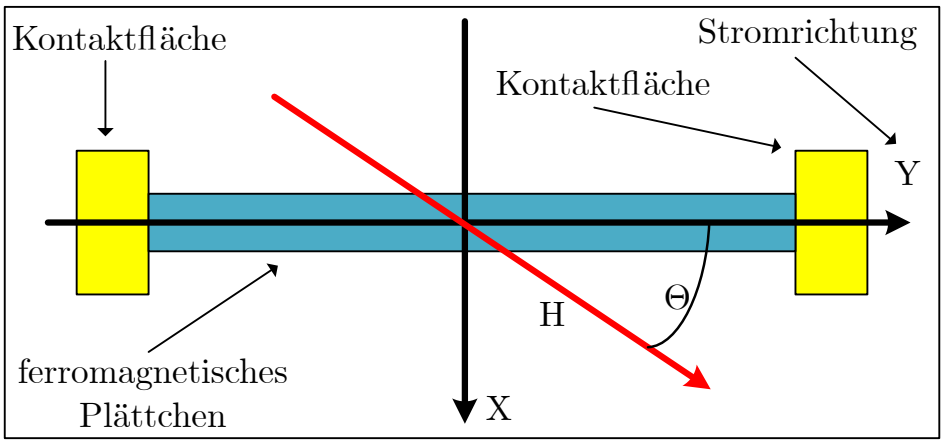


Abbildung 2.4: AMR Sensor schematische Darstellung nach [A4]

In Abbildung 2.4 ist eine Achse eines AMR-Sensors schematisch dargestellt. In der Mitte befindet sich das ferromagnetische Plättchen, welches typischerweise aus Permaloy hergestellt wird. Der elektrische Widerstand ändert sich stark mit der magnetischen Feldstärke. Der Zusammenhang zwischen der Feldstärke H ist direkt proportional zur Widerstandsänderung ΔR.

Die zweite direkte Abhängigkeit welche, von AMR-Sensoren genutzt wird, ist die Änderung des Widerstandes mit der Richtung des Feldes. In der Abbildung 2.5 ist dies Proportionalität grafisch dargestellt.

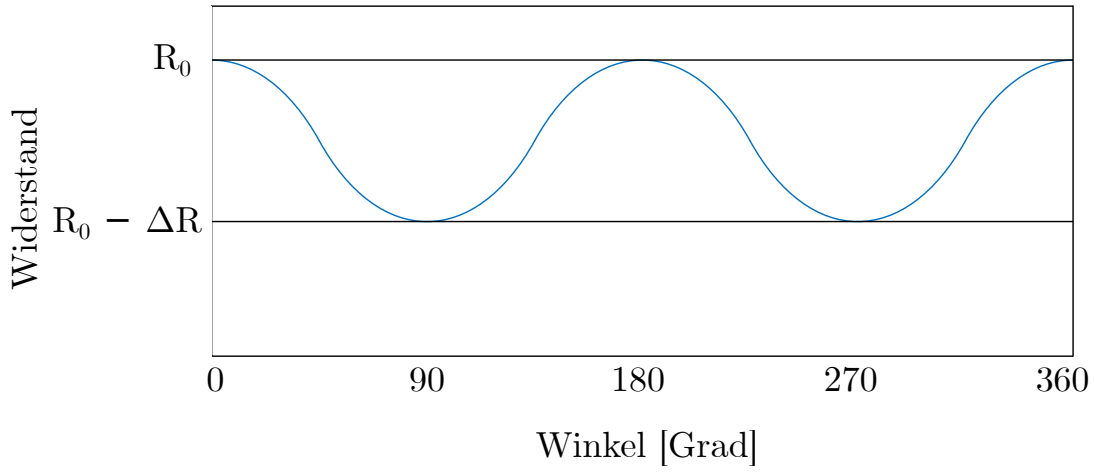


Abbildung 2.5: Zusammenhang Magnetfeldrichtung und Widerstand nach [A4]

Dieser Zusammenhang wird auch durch Gleichung (2.5) beschrieben. Es zeigt sich bei einem vertikal zur Stromrichtung stehendem Feld ($\theta = 90°, 270°$) ist die Änderung des Widerstandes ΔR am geringsten. Steht das magnetische Feld horizontal zu dem Permaloyleiter ($\theta = 0°, 180°$), so ist die Widerstandsänderung am stärksten.

$$R = R_0 - \Delta R \sin^2 \theta \tag{2.5}$$

2.2 Eingebettete Systeme

Sensoren werden häufig in sogenannten eingebetteten Systemen eingesetzt. Von eingebetteten Systemen spricht man, wenn ein Computersystem in einen technischen Kontext eingebettet ist. Das jeweilige System wird oft speziell für den jeweiligen Kontext entwickelt und optimiert. Diese Computersysteme besitzen immer irgend eine Art von Peripherie, um mit Ihrer Umgebung zu interagieren. Die Peripherieelemente können Sensoren sein, um Prozessparameter zu bestimmen, oder auch Aktoren, um bestimmte Steueraufgaben zu übernehmen. Typische Beispiele für solche Systeme sind etwa Gebäudesteuerung oder auch deren einzelnen Komponenten wie etwa die Heizungs- oder Rollladensteuerung. Weitere eingebettete Systeme finden sich auch im Haushalt, so arbeiten auch in Waschmaschinen und viel moderne Kühlschränken eingebettete Systeme.

In Abbildung 2.6 ist ein eingebettetes System und seine typischen Komponenten dargestellt. Als Recheneinheit kommen für solche Systeme oft sehr spezialisierte Lösungen

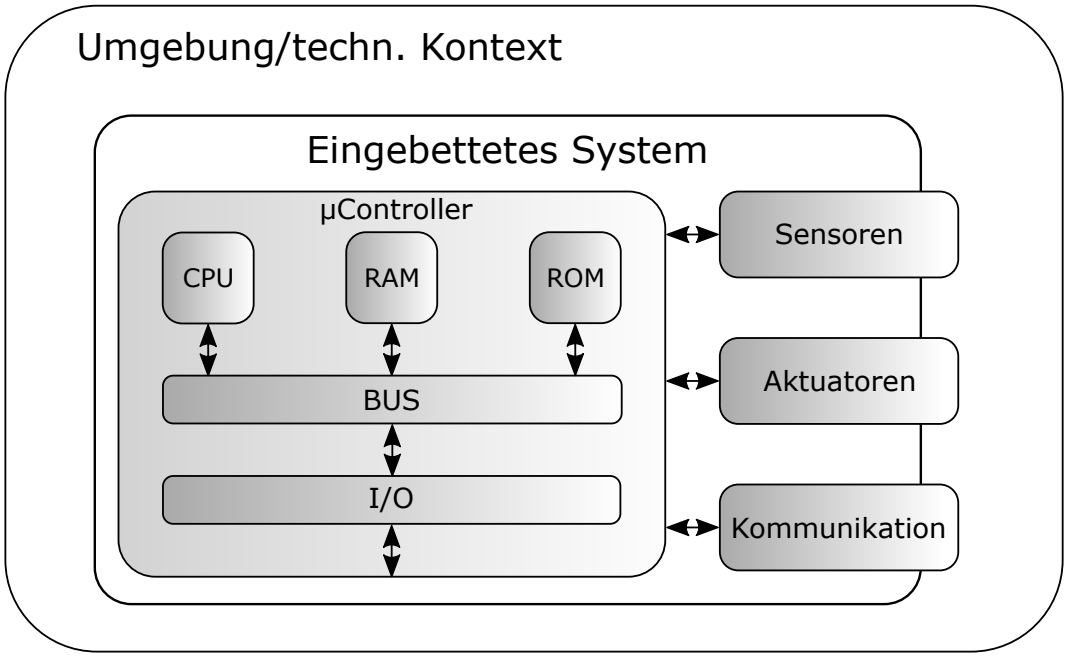


Abbildung 2.6: Exemplarische Darstellung eines eingebetteten Systems

zum Einsatz. Häufig werden auch Mikrocontroller verwendet, diese besitzen zusätzlich zur eigentlichen Recheneinheit bereits integrierte Peripherieelemente um direkt Sensoren oder Kommunikationselemente anzubinden. Bei der Auswahl gibt es in der Regel Beschränkungen hinsichtlich Parametern wie Energieeffizienz der Systeme, Größe der zu wählenden Komponenten oder auch besondere Anforderungen an die Kosteneffizienz. Die Grafik zeigt ebenfalls die Peripherieelemente des eingebetteten Systems, welche mit der Umgebung in Verbindung stehen. Die Sensoren erfassen dabei Prozess- oder Umgebungsparameter. Diese Daten können dann in der integrierten Recheneinheit ausgewertet werden. Aufgrund der Auswertung kann das System dann bestimmte Aktoren nutzen, um den jeweiligen Prozess zu steuern. Die Kommunikationsschnittstellen dienen der Vernetzung von mehreren eingebetteten Systemen untereinander, um kollaborativ Prozesse zu steuern oder zu überwachen. Weiterhin können die Schnittstellen genutzt werden um das System mit anderen Systemen wie Webservern und Datenbanken zu verbinden.

Häufig werden auch moderne Smartphones als eingebettete Systeme klassifiziert. Sie erfüllen einige der für diese Systeme typischen Anforderungen und Beschränkungen. So besitzen diese oft Sensoren wie Mikrofone, Beschleunigungssensoren, Gyroskope und Magnetometer um Umgebungsparameter zu erfassen. Weiterhin sind sie Resourcenbeschränkt, da sie eine immer noch recht limitierte Akkukapazität aufweisen. Einige Merkmale von eingebetteten Systemen werden von Smartphones allerdings nicht erfüllt. So sind die in Ihnen verbauten Recheneinheiten ähnlich wie in PCs in der Lage sehr viele

unterschiedlichste Programme auszuführen und Aufgaben zu bearbeiten. So kann ein Smartphone sowohl für Multimedia Anwendungen als auch für Navigation oder einfach zum Telefonieren genutzt werden. Die Resourcenbeschränktheit trifft auch nur teilweise zu, da moderne Smartphones ähnlich leistungsfähige Rechen- und Speichereinheiten besitzen wie mobile PCs. Daher ist die Einordnung von Smartphones zu den eingebetteten Systeme kontextabhängig.

2.3 Intelligente Sensoren

Neben den leistungsfähigeren und multimedial ausgestatteten Systemen wie den Smartphones können auch bereits Sensoren selbst als eingebettetes Systeme klassifiziert werden. Dabei handelt es sich um sogenannte intelligente Sensoren, im Englischen als *Smart Sensors* bezeichnet. In intelligenten Sensoren wird das eigentliche Sensorelement häufig in MEMS-Technologie gefertigt. Neben dem Sensorelement enthält solch ein Sensor eine Reihe weitere Komponenten, die ihn als intelligenten Sensor auszeichnen.

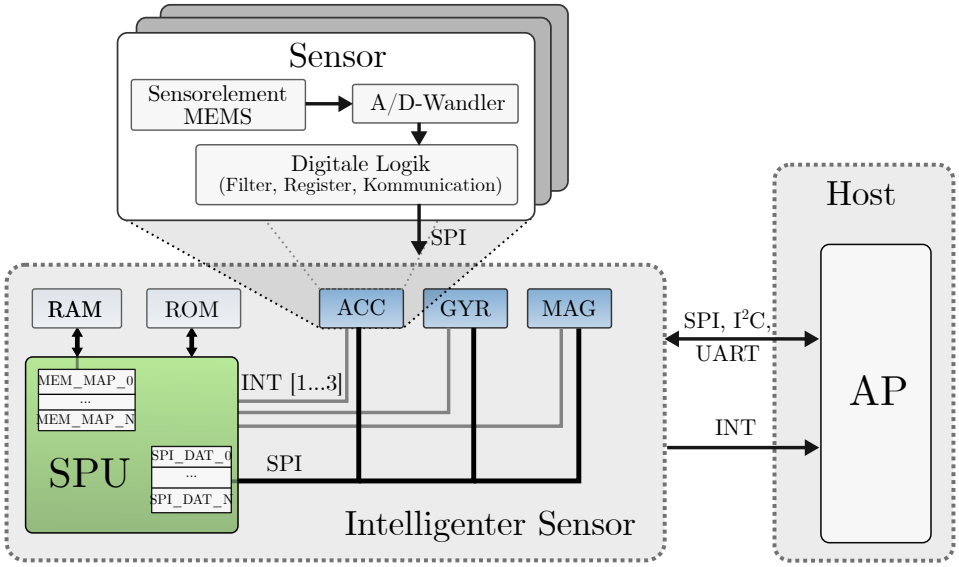


Abbildung 2.7: Aufbau eines intelligenten Sensors

Ein typischer intelligenter Sensor ist in Abbildung 2.7 dargestellt, er besteht hauptsächlich aus drei Komponenten, wobei der Integrationsgrad einzelner Komponenten je nach Typ variieren kann.

Die erste Komponente ist der eigentliche Sensor. Dies kann ein Inertialsensor wie Beschleunigungsmesser oder Gyroskop sein. Andere Sensortypen, wie z.B. Temperatur-

oder Drucksensoren, werden ebenfalls häufig verwendet. Wie in Abbildung 2.7 dargestellt, können diese Komponenten in der Regel auch in drei weitere Unterkomponenten unterteilt werden. Die erste Unterkomponente ist das eigentliche Sensorelement zur Erfassung der zu messenden physikalischen Größe. Der Analog-Digital-Wandler (ADC) transformiert das elektrische Signal vom Sensorelement, welches zeit- und wertkontinuierlich ist, in ein digitales Signal, welches im digitalen Logikteil des Sensors verwendet werden kann. In der digitalen Logik kann das digitale Signal durch Filter vorverarbeitet werden, z. B. durch Tiefpassfilter oder Mittelwertbildung über eine konfigurierbare Anzahl von Abtastwerten. Nach der Vorverarbeitung werden die Daten in Registern gespeichert und können von der Recheneinheit des intelligenten Sensors über die Kommunikationsschnittstelle abgefragt werden.

Die zweite Komponente eines intelligenten Sensors wird durch Kommunikationselemente gebildet. Es gibt eine Reihe von Kommunikationsschnittstellen die innerhalb von Sensorsystemen ihren Einsatz finden. Am häufigsten werden Serial Peripheral Interface (SPI) oder Inter-Integrated Circuit (I^2C) verwendet. Zusätzlich zur Kommunikationsschnittstelle verbinden einige Sensoren auch eine oder mehrere Interrupt-Leitungen mit der Recheneinheit um eine Benachrichtigung über Ereignisse mit geringer Latenz, wie z. B. neue Sensordaten, zu ermöglichen.

Die Recheneinheit bildet die dritte Komponente des intelligenten Sensors. Erst durch die Verwendung einer solchen im nachfolgenden auch als Sensor-Prozessor-Einheit (Sensor Processing Unit, SPU) bezeichnet, kann dieser als intelligenter Sensor bezeichnet werden. Häufig finden besonders energieeffiziente Prozessoren oder Mikrocontroller Anwendung. Dabei können standardisiert Einheiten wie etwa ARM Cortex M basierte Prozessoren oder auch anwendungsspezifische proprietäre Recheneinheiten verwendet werden. Die SPU kann durch zusätzlichen Speicher erweitert werden, der über eine Standardkommunikationsschnittstelle oder eine dedizierte Speicherschnittstelle verbunden wird. Die Verwendung einer SPU in einem intelligenten Sensor ermöglicht es dem Anwendungsentwickler, alle vorhandenen Sensorelemente direkt auf der Sensorhardware abzufragen und eine Vorverarbeitung der erhaltenen Daten vorzunehmen. Diese so schon vorverarbeiteten Sensordaten können über zusätzliche Kommunikationsschnittstellen an eine externe Recheninstanz, den sogenannten Host, übertragen werden. Die Möglichkeiten, die abgetasteten Daten auf der SPU zu manipulieren und zu verarbeiten sind mannigfaltig. Beispielsweise können die gesammelten Daten komprimiert oder gefiltert werden. Zusätzlich kann der Entwickler sogenannte virtuelle Sensoren generieren, z.B. Schritterkennung, absolute Orientierung oder Gestenerkennung. Diese virtuellen Sensoren kombinieren oft die Sensordaten unterschiedlicher Sensoren, um höherwertige Informationen zu generieren.

3 Stand der Technik und Forschung

Der Fokus dieser Arbeit liegt auf der Erweiterung und Bereitstellung von Lösungen für die Entwicklung von Firmware auf intelligenten Sensoren. Im Speziellen wird eine Möglichkeit geschaffen, die entwickelte Firmware direkt auf dem Sensor reproduzierbar zu testen und zu verifizieren. Dabei werden bereits vorhandenen Möglichkeiten der Firmwareentwicklung genutzt und an entsprechender Stelle erweitert. Das Konzept wurde für State-of-the-Art intelligente Sensoren entwickelt und auf diesen implementiert und getestet. Dabei wird hauptsächlich auf Inertialsensorik zurückgegriffen. Diese eignet sich aufgrund der anspruchsvollen und oft nicht erreichbaren Reproduzierbarkeit der Daten besonders gut für Untersuchungen und Demonstrationen der entwickelten Ansätze. Das vorgestellte Konzept ist darüber hinaus auch für andere Sensoren ohne Limitierung übertragbar und nutzbar.

Im Folgenden wird zunächst der Stand der Technik beschrieben. Dabei werden die vorhanden Sensortypen und die Möglichkeiten der Entwicklung von Sensorfirmware dargestellt. Weiterhin sind Möglichkeiten zum Testen und Debuggen der Firmware Bestandteil dieses Abschnitts. Im darauf folgenden Abschnitt wird die wissenschaftliche Problemstellung dargelegt. Hierbei wird vor allem auf die Möglichkeiten zur Verifikation und zum Testen von intelligenten Sensoren und deren Firmware eingegangen. Im letzten Abschnitt findet eine Einordnung dieser Arbeit in den zuvor beschriebenen Stand der Technik und den wissenschaftlichen Kontext statt.

3.1 Stand der Technik

Das in dieser Arbeit vorgestellte Sensor-in-the-Loop-Konzept erweitert bisher vorhandene Möglichkeiten zur Firmwareentwicklung auf intelligenten Sensoren. Es bedient sich dabei bisher vorhandener Hardware- und Softwarekomponenten zur effizienten Softwareentwicklung auf modernen intelligenten Sensoren. In diesem Abschnitt werden State-of-the-Art programmierbare Sensoren vorgestellt. Weiterhin werden heutzutage häufig genutzte Firmwareentwicklungswerkzeuge betrachtet.

Intelligente Sensoren besitzen neben dem eigentlichen Sensorelement immer eine programmierbare Recheneinheit und dazugehörige Speicherelemente. Erst durch diese zusätzlichen Komponenten kann ein Sensor als intelligenter Sensor definiert werden. Das

eigentliche Sensorelement ist dabei nicht ausschlaggebend. Es existieren auf dem Markt eine Vielzahl von Sensoren, um eine breite Palette an Messaufgaben abzudecken. Mögliche Sensorelemente sind beispielsweise Beschleunigungssensoren welche üblicherweise in drei Achsen die Beschleunigung des Sensors in m/s^2 messen [A5], [A6]. Weitere oft in MEMS-Technologie ausgeführte Sensorelemente sind Gyroskope [A7], [A8] oder auch Magnetfeldsensoren [A9], welche häufig auch als e-Kompass bezeichnet werden [A10]. Oft werden zwei oder auch alle drei dieser sogenannten Inertialsensoren in einem einzigen Sensorgehäuse vereint, diese Sensoren werden dann als Inertiale Messeinheit (Inertial Measurement Unit, IMU) bezeichnet [A11], [A12]. Viele Hersteller von MEMS-Sensoren bieten neben den Inertialsensoren eine Vielzahl anderer Sensoren an. Darunter sind beispielsweise Luftdrucksensoren [A13] oder auch Abstandssensoren [A14].

Um diese Sensorelemente zu einem intelligenten Sensor zu erweitern, werden von den Sensorherstellern verschiedene Recheneinheiten eingesetzt. Diese Mikroprozessoren können je nach Anwendungsfall und Sensortyp sehr unterschiedlich ausfallen. Dabei kann der jeweilige Prozessor für bestimmte Anforderungen optimiert ausgewählt oder sogar speziell entwickelt und gefertigt sein. Bei Anwendungen, die wenig Kontrollflussoperationen aber dafür viele arithmetische Operationen benötigen wie in [A15], können digitale Signalprozessoren (Digital Signal Processors, DSPs) eingesetzt werden. Diese sind für die digitale Signalverarbeitung optimiert und können diese besonders effizient berechnen. Häufig werden aber noch weitaus spezialisiertere Prozessoren als Recheneinheit in intelligenten Sensoren benötigt und eingesetzt. Diese auch ASIP genannten Prozessoren werden speziell für einen bestimmten Anwendungsfall entwickelt und haben häufig angepasste spezielle Instruktionen [A16]. Dadurch sind sie für den speziellen Fall besonders effizient aber können durch ihre Programmierbarkeit bis zu einem gewissen Grad leicht angepasst und optimiert werden. Mit der stetigen Weiterentwicklung im Bereich des maschinellen Lernens und der Nutzung von Algorithmen basierend auf künstlicher Intelligenz, finden auch spezielle auf KI-Algorithmen optimierte Prozessoren Anwendungen in intelligenten Sensoren. Diese Prozessoren sind auf die Ausführung von KI-Algorithmen wie Beispielsweise die Berechnung von neuronale Netzen hin optimiert [A17]. In vielen Fällen werden aber auch standardisierte Mikroprozessoren oder Mikrocontroller verwendet. Dies hat den Vorteil, dass der Entwickler des jeweiligen Sensors ein breites Spektrum an Anwendungen abdecken kann. Weiterhin können oft Standard-Anwendungen, Methoden und Werkzeuge für die Entwicklung eingesetzt werden. Die verwendeten Mikrocontroller können Eigenentwicklungen der Sensorhersteller oder aber auch lizenzierte Standard-Komponenten wie beispielsweise der *ARM Cortex-M* sein. So wird beispielsweise in der BMF055 IMU von *Bosch Sensortec* ein *ARM Cortex-M0+* eingesetzt [A18]. Dieser Sensor findet in dieser Abhandlung für die Implementierung und die Untersuchung des Sensor-in-the-Loop (SiL)-Ansatzes Anwendung.

Die Firmware für intelligente Sensoren wird aktuell in vielen Fällen in Hochsprachen wie *C* oder auch *C++* geschrieben. In Spezialfällen wie zum Beispiel besonders sicherheitskritischen Anwendungen wird auch häufig noch auf die direkte *Assembler*-Programmierung

zurückgegriffen. In jedem Fall werden zum Schreiben der Firmware immer häufiger standardisierte Entwicklungsumgebungen, sogenannte integrierte Entwicklungsumgebungen (Integrated Developement Environments, IDEs), verwendet. Diese Softwareprodukte werden auch für die Programmierung von klassischen Computerprogrammen oder Mikrocontrollern eingesetzt. Die IDEs bieten dem Entwickler neben dem eigentlichen Schreiben und Compilieren des Programmcodes einige zusätzliche Funktionalität, die das Entwickeln komfortabler und effizienter gestalten. Funktionalitäten wie automatische Code-Vervollständigung, Formatierung des Quellcodes oder das Hervorheben von Fehlern direkt im Quellcode sind Standard und in den meisten IDEs verfügbar. Viele Hersteller von Mikrocontrollern und Sensoren, bieten ihre eignen spezialisierten Entwicklungsumgebungen an [A19], [A20]. Häufig basieren diese speziellen IDEs auf Standard IDEs wie zum Beispiel *Eclipse* [A21] oder *VisualStudio* [A22].

Zum Übertragen der erstellten Firmware dienen häufig genutzte Schnittstellen wie Serial Peripheral Interface (SPI) oder Universal Asynchronous Receiver Transmitter (UART). Diese ermöglichen lediglich die Übertragung des Firmware-Images vom Entwicklungsrechner auf den Sensor. Neben diesen einfachen Möglichkeiten existieren Schnittstellen, die es zusätzlich erlauben die erstellte Firmware während der Laufzeit zu beeinflussen. Diese Debugging-Schnittstellen ermöglichen das Anhalten des Programmablaufes oder auch das Instruktionsweise, *Schritt für Schritt* Ausführen der Firmware. Zusätzlich können Zustände der Firmware oder einzelne Speichereinheiten beobachtet oder auch modifiziert werden. In Kombination mit einer IDE bieten diese Schnittstellen wie zum Beispiel JTAG [A23] oder SWD [A24] eine sehr gute Möglichkeit der effizienten Firmwareentwicklung.

3.2 Wissenschaftliche Problemstellung

Die wissenschaftliche Problemstellung beschäftigt sich hauptsächlich mit der Beobachtbarkeit und Verifizierbarkeit von Sensorsystemen. Dabei werden die funktionalen Eigenschaften eines Sensorsystems betrachtet, welche die korrekte Ausführung der Sensoralgorithmen und darauf aufbauende Sensorfirmware umfassen. Neben diesen funktionalen Eigenschaften können weiterhin extrafunktionale Eigenschaften eines Sensorsystems untersucht werden. Diese Eigenschaften wie zum Beispiel Laufzeit, Speicherbedarf oder Energieverbrauch können einen entscheidenden Einfluss auf die Bewertung des Sensorsystems haben [A25]. Häufig ist es schwierig wiederholbare oder reproduzierbare Untersuchungen an Sensorsystemen durchzuführen, besonders wenn neben den funktionalen auch extrafunktionale Eigenschaften betrachtet werden sollen [A26], [A27].

In manchen Fällen ist es ausreichend nur die funktionalen Eigenschaften eines bestimmten, Sensordaten verarbeitenden Algorithmus zu testen und zu verifizieren. Für diese Aufgabe eignen sich besonders Mathematik-Softwaretools wie *Matlab* [A28] oder *GNU*

Octave [A29]. Weiterhin finden auch Skriptsprachen wie *Python* Anwendung in diesem Bereich, häufig werden diese durch zusätzlich Softwarepakete wie *NumPy* [A30] oder *SciPy* [A31] erweitert. Dies ermöglicht es zeit- und kosteneffizient die funktionalen Eigenschaften von Sensoralgorithmen zu überprüfen.

Speziell für die Simulation von sensordatenbasierten Algorithmen entwickelte Tools und Toolboxen bieten zusätzliche Funktionalität, um die Entwicklung komfortabler und effizienter zu gestalten. Ein Beispiel dafür ist *dSPACE*, welches einen Hardware-in-the-Loop *Sensor Simulation PC* für die Sensorsimulation anbietet. Er ermöglicht eine deterministische Simulation der physikalischen Modelle eines Sensors in Echtzeit [A32]. Die entwickelten Algorithmen können basierend auf den Sensordaten reproduzierbar ausgewertet werden. Der *Sensor Simulation PC* ist in erster Linie auf das autonome Fahren ausgerichtet. Daher bietet er Simulationsdaten für Kamerasensoren, LIDAR und RADAR. Die *Sensor Fusion and Tracking Toolbox* von *MathWorks* [A33] oder die GNSS-INS-SIM [A34] sind weitere Toolboxen, welche sich eignen um sensordatenbasierte Algorithmen zu entwickeln und zu validieren. Beide Toolboxen bieten ein High-Level-Softwarepaket und Werkzeuge, welche speziell für die Arbeit mit Sensordaten entwickelt wurden. Die *Sensor Fusion and Tracking Toolbox* von *MathWorks* enthält bereits mehrere verschiedene vordefinierte Algorithmen zur Objektdetektion, Positionsverfolgung oder Situationserkennung. Dabei kann man entweder aufgezeichnete oder komplett synthetische Daten verwenden. Beide Toolboxen sind vollständig simulationsbasiert und arbeiten auf einer hohen Abstraktionsebene. Dies macht es unmöglich, das Verhalten auf realer Hardware oder extrafunktionale Eigenschaften wie Stromverbrauch und Speicherbedarf zu bestimmen. Weiterhin kann auch keine Aussage über Ausführungsgeschwindigkeit des Algorithmus auf dem zu verwenden Sensor getroffen werden.

Ein moderner Ansatz zur Bewertung eines Inertialsensorsystems ist die Verwendung eines Simulationsmodells mit einem sogenannten virtuellen Prototypen (VP). Diese VPs simulieren nicht nur das funktionale Verhalten des Systems, sondern auch seine extrafunktionalen Eigenschaften wie Timing, Stromverbrauch oder Speicherbedarf. Um dies zu erreichen wird die verwendete Hardware des Sensors selbst simuliert, welche bis zu einem gewissen Abstraktionsgrad modelliert wurde. In [A35] stellen die Autoren zum Beispiel ein RISC-V-Prozessormodell vor. Dieses enthält neben den Standard-Modulen auch ein Sensor-Submodul. Es gibt viele weitere frei verfügbare virtuelle Prototypen für eine breite Palette von Architekturen [A36]. Ein virtueller Prototyp benötigt nicht die tatsächliche Hardware, um die entwickelten Fusionsalgorithmen zu evaluieren, ist aber dennoch recht genau, was das Verhalten der Hardware angeht. Die genaue Simulation komplexer Hardwaresysteme in jedem Detail kann jedoch sehr rechenintensiv werden, und es ist oft nicht möglich, die Hardware in Echtzeit zu simulieren. Außerdem ist die Genauigkeit der simulierten Hardware auf das jeweilige Abstraktionslevel begrenzt. Es ist in der Regel nicht möglich, alle physikalischen Details der Hardware zu simulieren, da sie entweder nicht dokumentiert oder unbekannt sind. Ein Beispiel dafür ist der Jitter, dieser kann entweder vom Messverfahren des Sensors oder von der Kommunikation

zwischen den Elementen beeinflusst werden.

Eine Sonderform der Simulationsmodels stellt die Implementierung des Systems auf einem Field Programmable Gate Array (FPGA) dar. Dabei wird die Hardwarebeschreibung des Systems synthetisiert und durch programmierbare Logik-Gatter nachgebildet. Solche Systeme können eingesetzt werden um den zu entwickelten Sensor schon vor der Fertigstellung der eigentlichen Hardware eingehender zu testen. Besonders die höhere Geschwindigkeit gegenüber des VP und die Möglichkeit der Verwendung eines Hardware-in-the-Loop (HiL) Ansatzes, sind die Vorteile dieser Methode. In [A37] wird ein solcher Ansatz beschrieben. Die Verwendung von FPGAs wird häufig auch mit dem Rapid Prototyping assoziiert [A38]. Genau wie bei den VP basierten Ansätzen ist die komplexe und aufwendige Systembeschreibung ein Nachteil dieser Methode. Weiterhin können die extrafunktionalen Eigenschaften des Sensorsystems nicht untersucht werden.

Die Verwendung eines Prototyp- oder Applikationsboards ist eine weitere moderne Methode zur Bewertung der Inertialsensoren [A39], [A40]. Diese Prototypen beinhalten die realen Sensoren häufig schon in Kombination mit anderen, später in der Anwendung verwendeten, Hardwarekomponenten. Dies ermöglicht eine Bewertung der tatsächlichen Hardware in Verbindung mit der zu entwickelnden Firmware. Alle Effekte, die von den physikalischen Eigenschaften der Hardware abhängen, können mit diesem Ansatz beobachtet werden. Allerdings ist es mit dieser Methode schwierig bis unmöglich, reproduzierbare Sensordaten für verschiedene Testläufe zu erzeugen. Weiterhin kann es vorkommen, dass sich der verwendetet Prototyp leicht von der finalen Implementierung unterscheidet. Dies führt unter Umständen zu weiterem Testaufwand nach der Fertigstellung des Produktes. Der Einsatz von Robotern ist ein Weg, um Messungen wiederholbar zu machen. Die programmierbare Ausführung von Bewegungen kann für einige Anwendungsfälle eine Möglichkeit darstellen Inertialsensordaten teilweise reproduzierbar zu erzeugen. Aufgrund von Hardwarebeschränkungen, speziell im Bereich der Gelenkverbindungen, sind Roboter jedoch häufig nicht in der Lage, echte menschliche Bewegungen oder sehr komplexe Bewegungen im Detail zu simulieren [A41].

3.3 Abhandlung im wissenschaftlichen Kontext

In den beiden vorhergehenden Abschnitten wurden der aktuelle Stand der Technik und die wissenschaftliche Problemstellung bei der Entwicklung und Analyse von Firmware auf intelligenten Sensoren vorgestellt. Dieser Abschnitt dient der Einordnung des entwickelten Sensor-in-the-Loop-Ansatzes sowohl in den Stand der Technik als auch in die in Abschnitt 3.2 vorgestellten Methoden. Die in dieser Arbeit vorgestellte Methodik zum Testen und zur Verifikation von Firmware auf intelligenten Sensoren wurde so konzipiert, dass Sie mit in Abschnitt 3.1 vorgestellten Tools und Sensoren kompatibel ist. Dabei soll es möglich sein, das SiL-Konzept mit modernen IDEs zu nutzen und in State-of-the-Art

intelligente Sensoren zu integrieren. Die SiL-Architektur wurde entwickelt, um die in Abschnitt 3.2 vorgestellten Methoden zu erweitern. Es wurde dabei darauf abgezielt, die Vorteile der einzelnen Methoden in einem einzigen Verfahren zur Verfügung zu stellen, ohne die jeweiligen Limitierungen zu übernehmen.

Formale Methoden der Untersuchung von Sensoralgorithmen wie sie in [A28] - [A31] verwendet werden bieten genau, wie die simulativen Methoden aus [A32] - [A36] und das FPGA basierte Prototyping [A37], [A38], die Möglichkeit der reproduzierbaren Testausführung. Ziel des Sensor-in-the-Loop-Konzeptes ist es, diese Reproduzierbarkeit auf realer Sensorhardware nutzbar zu machen. Dieses Konzept wurde zuerst in der Publikation [B5] vorgestellt. Später wurde der Ansatz weiter verfeinert und das entstandene System eingehend auf seine Qualität und Eignung untersucht. Diese Untersuchungen sind im MDPI Sensors Journal [B6] publiziert worden. Für dieses System und das Verfahren der reproduzierbaren Validierung von Sensorfirmware auf echter Hardware mittels Sensordateninjektion wurde ein deutsches und ein US-Patent beantragt [B7], [B8]. Dabei können die Vorteile der Methode basierend auf Hardwareprototypen [A39], [A40] genutzt werden. In der Journal-Publikation [B9] wurde das SiL-System verwendet, um die funktionalen und extrafunktionalen Eigenschaften von Sensorfusionsalgorithmen zu überprüfen und zu bewerten. Speziell mit der extrafunktionalen Eigenschaft des Energieverbrauches beschäftigen sich die Konferenzveröffentlichung [B12] und die Journal-Publikation [B11].

Tabelle 3.1: Sensor-in-the-Loop im Vergleich zu State-of-the-Art Ansätzen

Ansatz	Abstraktion	Ausführung	Reproduzierbar	Systemeigenschaften
Sensor-in-the-Loop	Komplettes System	schnell	Ja	Ja
Funktionale Simulation	Nur Daten	schnell	Ja	Nein
Virtueller Prototype	Teile des Systems	langsam	Ja	teilweise
FPGA Prototype	Teile des Systems	schnell	Ja	teilweise
Hardware Prototype	Komplettes System	schnell	Nein	Ja

Die Tabelle 3.1 zeigt den Sensor-in-the-Loop-Ansatz im direkten Vergleich zu den in Abschnitt 3.2 vorgestellten Methoden zum Testen und Verifizieren von Sensorfirmware. Dabei werden zum Vergleich der einzelnen Methoden vier Kriterien gegenübergestellt. Das erste Kriterium beschreibt wie detailliert das jeweilige Sensorsystem betrachtet werden kann. Dieses reicht von der ausschließlichen Betrachtung der Algorithmen und Daten, bei der Simulation, bis hin zur Möglichkeit, dass komplette physikalische Sensorsystem im Kontext des jeweiligen Testlaufes zu berücksichtigen. Bei dem zweiten Kriterium wird die Ausführungszeit eines Testlaufes betrachte. Diese ist natürlich immer abhängig von der jeweiligen Ausgestaltung der einzelnen Methode. Bei simulativen Methoden, wozu auch der VP gehört, hängt diese natürlich von der Hardware ab, auf welcher die Simulation gerechnet wird. Zusätzlich hat die Komplexität des jeweiligen Simulationsmodels

einen großen Einfluss auf die Ausführungsgeschwindigkeit. Das Kriterium der Reproduzierbarkeit beschreibt die Möglichkeit unterschiedliche Testläufe des Sensorsystems mit exakt denselben Sensordaten zu wiederholen. Hiermit ist zusätzlich zur Wiederholbarkeit des Tests von einem Entwickler auf einem Setup, auch eine Möglichkeit der Wiederholbarkeit des Tests von unterschiedlichen Testern auf unterschiedlichen Systemen gemeint. Die Betrachtung der extrafunktionalen Eigenschaften des Sensorsystems ist das letzte Bewertungskriterium. Diese Eigenschaften wie die Laufzeit von Algorithmen, Speicherbedarf oder Energieverbrauch spielen für die Bewertung des Sensorsystems eine entscheidende Rolle. Nach der korrekten Funktionalität, sind es diese Eigenschaften, welche über die Auswahl für oder gegen ein bestimmtes Sensorsystem betrachtet werden. Deshalb ist es wichtig, bereits während der Entwicklung diese extrafunktionalen Eigenschaften betrachten und optimieren zu können. Eine detaillierte Untersuchung aller extrafunktionalen Eigenschaften ist nur möglich, wenn die jeweilige Testmethode die reale Sensorhardware verwendet.

Der in dieser Arbeit vorgestellte SiL-Ansatz benutzt die reale Sensorhardware, daher wird wie bei Hardwareprototypen-Konzepten das gesamte Sensorsystem beim Testen und validieren der Firmware betrachtet. Die Ausführung der Tests kann mit der realen Geschwindigkeit des jeweiligen Sensors durchgeführt werden. Dadurch ist es auch möglich die extrafunktionale Eigenschaft der Laufzeit zu untersuchen und gegebenenfalls zu optimieren. Funktionale Simulationen können unter Umständen z.B. durch einfache Parallelisierung um ein vielfaches schneller sein als die reale Laufzeit des Sensors. Allerdings ist es hier auch schwieriger die reale Laufzeit auf dem Zielsystem abzuschätzen. Im Gegensatz zu Standard-Hardware-Prototyping-Ansätzen bietet der Sensor-in-the-Loop-Ansatz die Möglichkeit der Verwendung derselben Sensordaten bei unterschiedlichen Testläufen. Dies ermöglicht ein wiederholbares und reproduzierbares Validieren der Firmware. Durch die Verwendung des realen intelligenten Sensors, ist es möglich neben der funktionalen Überprüfung, parallel auch die extrafunktionalen Eigenschaften zu untersuchen. So wird die Firmware direkt für die Zielhardware kompiliert, was eine Bestimmung und Optimierung des Speicherbedarfes ermöglicht. Weiterhin sind Energiemessungen parallel zum Testen der Sensorfirmware möglich.

4 Firmwareentwicklung auf intelligenten Sensoren

Die Entwicklung von Firmware für intelligente Sensoren ist häufig ein komplexer und langwieriger Prozess. Die Algorithmen, welche auf den Sensordaten arbeiten, werden oft in speziellen Mathematik- oder Algorithmen-Entwicklungsumgebungen erstellt. Erst später werden die fertigen Algorithmen in die eigentliche Firmware des intelligenten Sensors eingefügt. Dieses ist nicht immer optimal und kann den Entwicklungsprozess verlangsamen. Dieses Kapitel beschreibt beispielhaft den Entwicklungsprozess der Firmware von intelligenten Sensoren. Im Allgemeinen werden unterschiedliche Entwicklungsprozesse je nach Anforderung und Branche durchgeführt. Im nachfolgenden wird exemplarisch ein häufig verwendeter Prozess beschrieben. Anschließend wird dieser Prozess unter Verwendung der neu entwickelten Sensor-in-the-Loop-Architektur erneut betrachtet. Dabei werden die Veränderungen im Prozessablauf genannt und bewertet. Das Kapitel schließt mit einem Zwischenfazit für diese Abhandlung.

4.1 Klassische Entwicklung von Sensorfirmware: Ein Fallbeispiel

In der Softwareentwicklung existieren unterschiedliche Modelle für die Planung und Umsetzung eines Softwareproduktes. Grundsätzlich kann man diese in klassische oder auch formelle Modelle und die agilen oder auch informellen Methoden unterteilen. Zu den klassischen Modellen gehören etwa das *Wasserfallmodel*, das *V-Model* oder auch das Spiralmodel. Einige Beispiele für die agilen Methoden sind *SCRUM*, *XP* oder *Kanban* [A42].

Da die Entwicklung von Firmware für intelligente Sensoren im Grunde auch ein Softwareprodukt ist, können diese Methoden auch hier angewandt werden. Abhängig von der Größe des Projektes und der Firmenphilosophie des entwickelnden Unternehmens werden entweder eher formale oder agilere Methoden gewählt. Natürlich sind in der Realität eher Mischformen oder angepasste Modelle üblich. Unabhängig von der Art des Models, welches für die Firmwareentwicklung priorisiert wird, ist das Testen und Validieren ein entscheidender Schritt im Erstellungsprozess der Firmware.

Um die Anschaulichkeit des Beispiels zu erhöhen, wird für die nachfolgende Beschreibung ein an das *Wasserfallmodel* angelehntes Entwicklungsmodel gewählt.

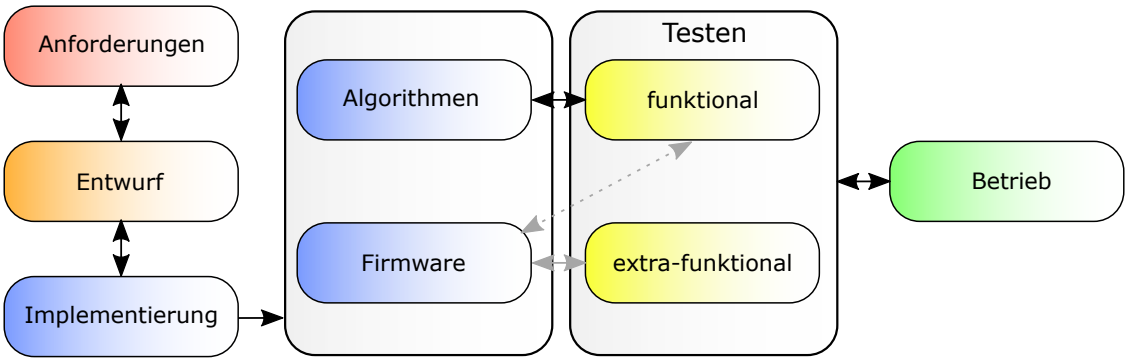


Abbildung 4.1: Firmwareentwicklungsprozess auf intelligenten Sensoren

Abbildung 4.1 zeigt exemplarisch die Phasen der Firmwareentwicklung angelehnt an das klassische Wasserfallmodel. Um die Übersichtlichkeit in der Grafik zu verbessern sind die Phasen nicht wie klassischerweise treppenartig angeordnet. Wie im klassischen Model ist der Verlauf der einzelnen Phasen linear, und die vorhergehende Phase bedingt die darauffolgende. Dieser sequenzielle Verlauf ist typisch für das *Wasserfallmodel*.

In der ersten Phase werden die Anforderungen an die zu erstellende Firmware definiert und ausgearbeitet. Diese Phase endet klassischerweise mit der Erstellung eines Lastenheftes. Anschließend wird in der Entwurfsphase ein konzeptioneller Entwurf der Firmware erstellt. Bei diesem Entwurf werden die einzelnen Komponenten und funktionalen Einheiten der zu erstellenden Firmware identifiziert und geplant. Auch die Komposition dieser einzelnen Einheiten und ihre Kommunikation wird in dieser Phase geplant.

Die Phase der Implementierung ist in Firmwareprojekten auf intelligenten Sensoren häufig in zwei Teile unterteilt. Zum einen werden die Algorithmen, welche auf den Sensordaten arbeiten, in extra Programmen für Algorithmenentwicklung oder auch Mathematiksoftware entwickelt. Dies ist oft notwendig, da die zu implementierenden Algorithmen immer komplexer werden und dadurch häufig von Spezialisten aus dem Signalverarbeitungsbereich erstellt werden. Nachdem die Algorithmenentwicklung abgeschlossen ist, werden diese dann in die Sensorfirmware integriert. Die eigentliche Firmware auf dem Sensor wird davon unabhängig in einer anderen Entwicklungsumgebung und häufig auch in einer anderen Programmiersprache erstellt. Dies ist unter anderem notwendig, um den Ressourcenbeschränkungen der intelligenten Sensoren gerecht zu werden. Zusätzlich arbeiten auch hier oft Entwickler, welche auf die Programmierung von Firmware spezialisiert sind. Bei der Entwicklung der Firmware wird dann oft speziell auf die extrafunktionalen Eigenschaften hin optimiert. Diese starke Trennung in der Entwicklung schafft besondere Herausforderungen. Bei der Integrierung der Algorithmen in die Firmware kann es vorkommen, das diese auf dem Zielsystem nicht wie erwartet funktionieren. Dies

hat vor allem mit der Ressourcenbeschränktheit der Sensorsysteme zu tun. Vor allem die Verwendung von unterschiedlichen Datentypen und der unter Umständen unterschiedlichen Implementierung von arithmetischen Operationen in der Algorithmensoftware und auf dem Sensor können zu unerwarteten Fehlern führen.

Das Testen der Algorithmen und der eigentlichen Firmware sind daher auch oft voneinander unabhängige Prozesse. In der Entwicklung der Algorithmen ist ausschließlich ein funktionales Testen möglich, dies kann auf dem PC reproduzierbar mit aufgezeichneten Sensordaten erfolgen. Darüber hinaus können allerdings keine extrafunktionalen Eigenschaften wie Laufzeit, Speicherbedarf oder Energieverbrauch untersucht werden. In der eigentlichen Firmware auf dem Zielsystem ist es dann möglich diese Eigenschaften zu untersuchen. Eine funktionale Überprüfung ist allerdings nur bedingt möglich, da das Reproduzieren der Sensordaten schwer bis unmöglich ist. Dies führt auch dazu, dass die extrafunktionalen Eigenschaften nicht systematisch und reproduzierbar getestet werden können. In der Abbildung 4.1 ist dieser Mangel der Reproduzierbarkeit durch die grauen Pfeile verdeutlicht. Sollte in der Firmware ein Fehler auftreten, müssen die Algorithmen unter Umständen mit anderen Sensordaten erneut isoliert getestet werden.

Die letzte Phase beschreibt den eigentlichen Betrieb der Firmware nach der Auslieferung. In einigen Quellen wird diese Phase auch als Wartungsphase bezeichnet, da wie bei jeder Software auch in Sensorfirmware noch unerwartete Fehler während des Betriebes auftreten können.

4.2 Entwicklungsprozess unter Einfluss von Sensor-in-the-Loop

Die Zweiteilung der Implementierungs- und Testphase bei der Entwicklung von Firmware für intelligente Sensoren bringt einige Nachteile mit sich. Die Zeit bis zur Fertigstellung der Firmware kann sich durch diesen Prozess unnötig verlängern. Durch die Entkopplung der Algorithmenentwicklung von der Firmwareentwicklung, werden funktionale Fehler des Algorithmus oder der Firmware nicht oder zu spät erkannt. Vor allem Fehler, die aus der Kombination dieser beiden Komponenten entstehen, sind schwer zu finden und zu beheben.

An dieser Stelle setzt die Sensor-in-the-Loop-Architektur ein, sie ermöglicht reproduzierbare Tests der Sensorfirmware mit aufgezeichneten oder künstlich erzeugten Sensordaten auf der eigentlichen Sensorhardware. Dadurch wird es möglich die entwickelten Algorithmen direkt in der Firmware zu entwickeln und zu testen. Eine Zweiteilung der Implementierungs- und Testphase kann vermieden werden. Häufig werden die Entwicklung der Algorithmen und der Firmware trotzdem getrennt voneinander durchgeführt.

Allerdings wird die Einbettung der Algorithmen in die eigentliche Firmware durch das Sensor-in-the-Loop (SiL)-System optimiert und vereinfacht.

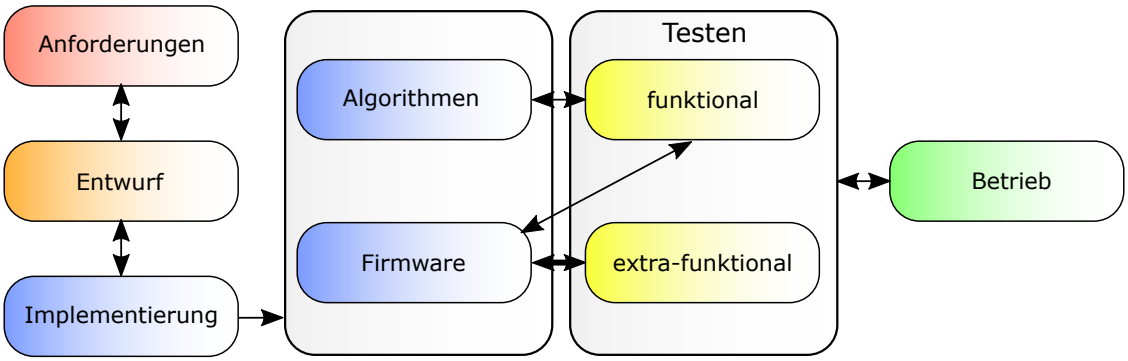


Abbildung 4.2: Firmwareentwicklungsprozess auf intelligenten Sensoren mit SiL

Abbildung 4.2 stellt den Entwicklungsprozess der Sensorfirmware unter der Verwendung des Sensor-in-the-Loop-Konzeptes dar. Bei den ersten beiden Phasen, der Anforderungs- und der Entwurfsphase ändert sich nichts im Vergleich zu dem in Abschnitt 4.1 vorgestellten Prozess. Die Implementierungs- und die Testphase werden optimiert, da die Algorithmen jetzt direkt in der Sensorfirmware implementiert und getestet werden können. Die nun schwarz dargestellten Pfeile verdeutlichen, dass es nun möglich ist neben den extrafunktionalen Eigenschaften auch die Funktion direkt in der Firmware auf dem Zielsystem zu testen. Das Untersuchen sowohl der funktionalen als auch der extrafunktionalen Eigenschaft ist nun in systematischer und reproduzierbarer Weise möglich. Bei der Betriebs- bzw. Wartungsphase gibt es keine Änderung zum vorhergehenden Entwicklungsprozess. Sollte in der Wartungsphase Änderungen an den Algorithmen nötig sein, sind die Aussagen für die Implementierungs- und Testphase natürlich auch hier gültig.

Das vorgestellte Sensor-in-the-Loop-Konzept vereinfacht den Entwicklungsablauf durch das Zusammenführen der Algorithmen- und Firmwareentwicklung. Zusätzlich ist es möglich, Fehler in den Algorithmen direkt in der entwickelten Firmware zu erkennen und zu beheben. Vor allem Fehler, welche aus dem Zusammenspiel der Algorithmen mit der Firmware des Sensors entstehen, können komfortabel erkannt und behoben werden. Die Möglichkeit zuvor in anderen Tools entwickelte Algorithmen nachträglich in die Sensorfirmware zu integrieren bleibt hierbei natürlich erhalten. Auch hierbei kann die SiL-Architektur helfen, die funktionale Äquivalenz der Algorithmen reproduzierbar zu verifizieren da die Testergebnisse direkt mit denen aus anderen Tools verglichen werden können.

4.3 Zwischenfazit

Die Entwicklung von Firmware für intelligente Sensoren stellt den Entwickler vor ganz besondere Herausforderungen. In der Regel werden Algorithmen benötigt, die auf den Sensordaten operieren und ihrer Ergebnisse an die Firmware übermitteln. Bei der Entwicklung dieser Algorithmen werden in unterschiedlichen Testläufen idealerweise dieselben Sensordaten benötigt, um reproduzierbare Tests zu ermöglichen. Dies macht es notwendig, die Algorithmen getrennt von der Firmware in spezieller Entwicklungssoftware mit integrierten Verifikationsmöglichkeiten zu entwickeln. Ein Testen der gesamten Firmware mit reproduzierbaren Sensordaten ist bisher schwer durchführbar, vor allem wenn zu den funktionalen auch noch extrafunktionale Eigenschaften untersucht werden sollen. Die in Abschnitt 3.2 vorgestellten Methoden sind gerade in Hinblick auf die extrafunktionalen Eigenschaften keine idealen Lösungen. In den Publikationen [B1], [B2], [B4] wurde die verwendete Sensorfirmware nach dem bisherigen Stand der Technik entwickelt und getestet.

Um diese Limitierungen bei der Entwicklung von Sensorfirmware aufzulösen, wird eine Möglichkeit benötigt, Sensordaten reproduzierbar direkt in der Sensorfirmware zur Verfügung zu stellen. Die in dieser Abhandlung vorgestellte Sensor-in-the-Loop-Architektur adressiert dieses Problem. Sie bietet die Möglichkeit neben der Funktionalität der Algorithmen auch, die gesamte Firmware reproduzierbar zu untersuchen. Die Wiederholbarkeit der Tests bringt einen großen Mehrwert für die Entwicklung der Firmware und die Optimierung der extrafunktionalen Eigenschaften. Dabei ist diese so konzipiert, dass die SiL-Architektur in Kombination mit modernen Softwareentwicklungsumgebungen funktioniert. Die Integration in fortschrittliche Firmwareentwicklungstools wie Debuggingadapter schafft einen zusätzlichen Vorteil. Um die genannte Funktionalität zu ermöglichen, muss allerdings die Firmware des Sensors angepasst werden, dies führt unter Umständen zu bestimmten Einschränkungen. So kann die Anpassung das Laufzeitverhalten der Firmware verändern, was evtl. die Genauigkeit der Funktionstest beeinflusst. Die ausgeführten Experimente zeigen allerdings, dass diese Änderungen verhältnismäßig gering und somit zu vernachlässigen sind.

Im folgenden Teil der Arbeit wird das Sensor-in-the-Loop-Konzept genauer erläutert. Weiterhin werden Untersuchung an einer möglichen Implementierung des Konzeptes durchgeführt. Diese Untersuchungen konzentrieren sich auf die Eignung des Systems reproduzierbare Sensordaten während der Laufzeit des Sensors zur Verfügung zu stellen. Nach erfolgreicher Implementierung wird das System für die Untersuchung von Sensorfusionsalgorithmen verwendet. Anschließend wird die Sensor-in-the-Loop-Implementierung noch um wichtige Funktionen, wie beispielsweise ein Sensorenergiemodel erweitert.

5 Sensor-in-the-Loop

In diesem Kapitel wird die neuartige Debugging-Architektur für intelligente Sensoren vorgestellt, dieses Konzept wurde auch unter dem Namen Sensor-in-the-Loop (SiL) publiziert. Dieses Kapitel baut auf den zuvor beschriebene technischen Grundlagen auf und bildet gemeinsam mit dem darauf folgenden Kapitel den Kern dieser Abhandlung. Die allgemeine Einordnung in den wissenschaftlichen Kontext und den Stand der Technik fanden bereits in Kapitel 3 statt. In Kapitel 4 wurde die klassische Herangehensweise bei der Entwicklung von Firmware für intelligente Sensoren beschrieben. Weiterhin ist ein Vergleich mit der Arbeitsweise unter Zuhilfenahme des SiL Ansatzes durchgeführt worden. Diese diente als Motivation für die Verwendung der hier vorgestellten Methodiken und Architekturen.

Das Kapitel beginnt mit einer Beschreibung von bestehenden Ansätzen in der Firmwareentwicklung von intelligenten Sensoren. Anschließend wird das Konzept des Sensor-in-the-Loop Ansatzes erläutert. Nach der konzeptionellen Vorstellung wird die Beispielimplementierung basierend auf dem BMF055 [A18] der Firma *Bosch Sensortec GmbH* beschrieben. Diese Implementierung wird auf Ihre Eignung als Verifikationsinstrument für funktionale und extrafunktionale Eigenschaften von Sensorfirmware untersucht. Dabei wird ein besonders Augenmerk auf das zeitliche Verhalten der SiL-Architektur im Vergleich zu real abgetasteten Sensordaten gelegt. Das Kapitel schließt mit einer Untersuchung der gesammelten und ausgewerteten Daten auf ihre statistische Signifikanz.

5.1 Einleitung

In den letzten Jahren haben sich Inertialsensoren für eine Vielzahl von Anwendungen durchgesetzt, welche auf den unterschiedlichsten Geräten implementiert werden. Die Anwendungen reichen von der Gestenerkennung [A43], [A44] über die Schritterkennung [A45], [A46] bis hin zum autonomen Fahren oder dem Fliegen sogenannter Unmanned Aerial Vehicles (UAVs)[A47], [A48].

Moderne Sensoren sind oft als sogenannte intelligente Sensoren ausgeführt. Diese Art von Sensoren besteht aus der eigentlichen Sensoreinheit und einer zusätzlichen Verarbeitungseinheit. Mithilfe dieser Einheit sind intelligente Sensoren in der Lage, die erfassten

Daten direkt auf dem Sensor vorzuverarbeiten. Erst danach werden die vorverarbeiteten oder komprimierten Daten an den Anwendungsprozessor des Systems gesendet.

Bei einer Vielzahl von Anwendungen, insbesondere bei sicherheitskritischen Systemen, ist es von entscheidender Bedeutung, dass das gesamte System, einschließlich der Inertialsensoren, unter allen möglichen Aspekten bewertet werden kann. Die Bewertung von Systemen im Hinblick auf ihre physikalischen Aspekte kann je nach der verwendeten Testumgebung eine große Herausforderung darstellen [A49]. Um aussagekräftige Ergebnisse zu erhalten ist es notwendig, das gesamte System einschließlich der Vorverarbeitung oder Filterung der Daten innerhalb der Inertialsensoren zu untersuchen. Die gängigen Methoden zu Untersuchung von Algorithmen für die Verarbeitung von Sensordaten oder Sensorsysteme wurden in Abschnitt 3.2 erläutert. Für die Bewertung des gesamten Sensorsystems werden nach dem aktuellen Stand der Technik hauptsächlich zwei Methoden verwendet.

Die erste Methode ist die Verwendung von Simulationsmodellen für die Systembewertung. Es gibt eine Vielzahl von Implementierungen, die diesen Simulationsansatz verwenden. Bei solchen Systemen ist es üblich, nur bestimmte Teile des gesamten Systems zu simulieren, z. B. nur den Datenpfad oder den Befehlssatz. Die Simulation eines ganzen Systems erfordert viel Simulationszeit und Rechenleistung, weshalb sie für Echtzeitsimulationen ganzer Sensorsysteme nicht geeignet ist.

Die zweite Methode verwendet Hardware-Prototypen der Sensoren, die häufig auf einer Prototyp-Leiterplatte einem sogenannten Printed Circuit Board (PCB) montiert sind. Dies erleichtert den Aufbau eines schnellen Prototyps zum Testen intelligenter Sensoren in einer vollständigen Sensorsystemumgebung. Die Verwendung des Hardware-Prototyps ermöglicht es dem Entwickler, alle Aspekte des intelligenten Sensors zu testen. Dazu gehören auch die extrafunktionalen Eigenschaften des Systems wie Stromverbrauch oder Wärmeverteilung. Der große Nachteil dieser Methode ist die fehlende Reproduzierbarkeit der Tests. Bei den derzeitigen Ansätzen, welche Hardware-Prototypen verwenden, ist es nicht möglich einen Test mit exakt denselben Sensordaten zu wiederholen.

Die vorgestellten klassischen Methoden der Bewertung von Sensorsystemen haben ihre Vorteile. Allerdings weisen auch beide Ansätze entscheidende Limitierungen auf. Um diese Nachteile zu überwinden, wurde der Sensor-in-the-Loop-Ansatz entwickelt, welcher auf einem Hardware-in-the-Loop (HiL)-Ansatz wie zum Beispiel in [A50] verglichen werden kann. Der SiL-Ansatz zielt darauf ab, alle positiven Aspekte der zuvor beschriebenen Methoden zu vereinen. Dabei verwendet er echte Hardware in einem Prototyp oder auf einem Evaluierungsboard. Die vorgestellte Methodik verwendet zuvor aufgezeichnete Sensordaten und nutzt diese für wiederholte Testläufe als Sensoreingangsdaten. Durch das erneute Einspeisen zuvor erfasster Sensordaten können reproduzierbare Systemtestergebnisse erzeugt werden, da der Sensor die eingespeisten Daten wie echte Sensordaten verarbeitet. Daher unterstützt der entwickelte Ansatz die Entwicklung, das Debugging und das Profiling von Firmware auf intelligenten Sensoren. Dies ermöglicht

wiederholbare und sogar reproduzierbare Testläufe der Sensorfirmware. Der Entwickler kann funktionale und extrafunktionale Systemeigenschaften wie Systemressourcen, Laufzeit oder Stromverbrauch in Echtzeit auf dem laufenden Zielsensor beobachten.

Die in diesem Kapitel vorgestellte SiL-Architektur wurde vom Autor dieser Arbeit zuerst im Rahmen der *Embedded Systems Week* auf dem *Rapid Prototyping Workshop* vorgestellt [B5]. Anschließend wurde das Verfahren und das entwickelte System in Deutschland [B7] und den USA [B8] zum Patent angemeldet. Weiterhin wurde eine Analyse des Zeitverhaltens des implementierten Systems durchgeführt. Diese Analyse wurde im *Sensors* Journal veröffentlicht [B6]. Dieses Kapitel beschreibt im Detail die Inhalte dieser Veröffentlichungen.

5.2 Konzept

Das Ziel des hier vorgestellten Ansatzes ist es, dem Entwickler von Sensor-Firmware eine Möglichkeit zu bieten, diese reproduzierbar zu testen und zu debuggen. Dies wird ermöglicht durch die Aufzeichnung von realen Sensordaten des zu verwendenden Sensors und die spätere Wiederverwendung dieser Daten zum Testen und Validieren der Firmware.

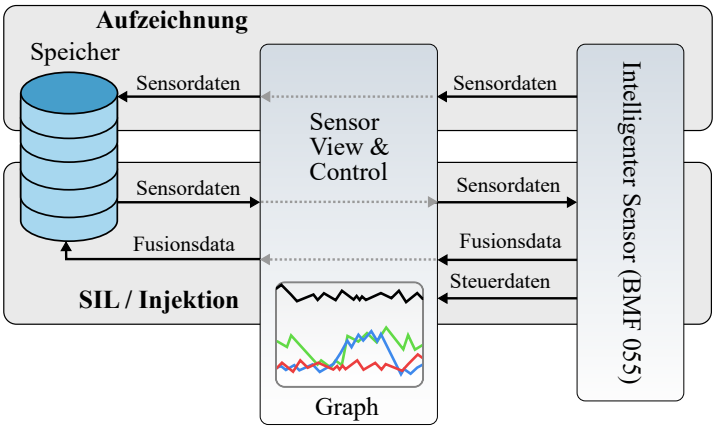


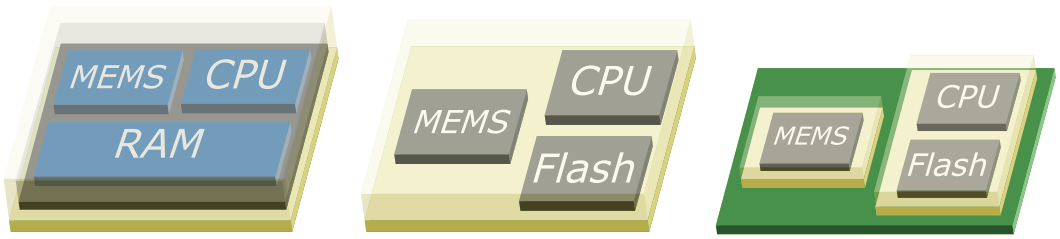
Abbildung 5.1: Konzept Workflow des SiL

Die Abbildung 5.1 zeigt die konzeptionelle Funktionsweise des SiL-Ansatzes. Dabei wird die Benutzung in zwei Phasen unterteilt. In Aufzeichnungsphase werden die Daten vom Sensor an eine Anwendung auf dem Entwicklungsrechner gesendet. Diese wird hier als *Sensor View & Control* bezeichnet. Die Anwendung visualisiert die Sensordaten und speichert sie für die spätere Verwendung. In der *Injektions-* oder *Replay*-Phase werden die zuvor aufgenommenen Daten zurück in die laufenden Sensor-Firmware übertragen. In dieser Zeit werden die Daten des eigentlichen Sensorelements ignoriert/verworfen.

Während dieser Phase wird die zu entwickelnde Sensor-Firmware mit reproduzierbaren Sensordaten getestet. Ergebnisse von Sensor-Algorithmen oder andere Kontrolldaten können gleichzeitig an den Entwicklungsrechner gesendet werden. Diese Daten helfen dem Entwickler den aktuellen Stand und die Funktionsweise der Firmware zu bewerten.

Das Sensor-in-the-Loop-Konzept besteht aus drei Teilen, die über eine gemeinsame Schnittstelle verbunden sind, was sie separat wartbar und austauschbar macht. In diesem Kapitel werden diese drei Teile von der Firmware innerhalb des Sensors über die Debugger-Schnittstelle bis hin zur Steuer- und Visualisierungssoftware auf dem PC des Entwicklers.

Der erste Teil befindet sich im Sensor selbst. Um mit dem entwickelten SiL-Konzept arbeiten zu können ist es notwendig, dass der Sensor ein sogenannter intelligenter Sensor ist. Diese Art von Sensoren enthält neben dem eigentlichen MEMS-Sensorelement eine interne Recheneinheit und Speicherelemente. Eine ausführliche Definition von intelligenten Sensoren ist in Abschnitt 2.3 zu finden.



(a) SoC **(b)** SiP **(c)** SoM

Abbildung 5.2: Integrationsstufen für intelligente Sensoren

Intelligente Sensoren lassen sich in drei verschiedene Integrationsstufen ausführen, die in Abbildung 5.2 dargestellt sind.

Die Version mit dem höchsten Integrationsgrad wird als System On Chip (SoC) bezeichnet. Eine grafische Darstellung findet sich in Abbildung 5.2(a). Bei diesem Integrationsgrad befinden sich alle Komponenten des Sensors auf einem einzigen Siliziumelement. Im Falle eines intelligenten Sensors gibt es mindestens ein MEMS-Element, eine Recheneinheit und etwas Speicher, welcher oft als Random-Access Memory (RAM) ausgeführt ist.

Abbildung 5.2(b) zeigt die zweite Integrationsstufe, welche als System in Package (SiP) bezeichnet wird. Bei diesem Integrationsgrad befinden sich die einzelnen Komponenten nebeneinander im selben Gehäuse und sind über Bonddrähte miteinander verbunden. Das MEMS-Element, die Recheneinheit und der Flash-Speicher befinden sich im selben Gehäuse aber auf dedizierten Siliziumelementen welche auch als *Dies* bezeichnet

werden. Diese Integrationsform ist häufiger anzutreffen als die SoC-Variante, da für die Herstellung des MEMS-Elements und der Recheneineinheit mit Speicher typischerweise unterschiedliche Technologien zum Einsatz kommen.

Die niedrigste Integrationsebene bietet die System on Module (SoM)-Implementierung eines intelligenten Sensors, die in der Abbildung 5.2(c) dargestellt ist. In dieser Ausführung hat jede Systemkomponente ihr eigenes Gehäuse. Alle Komponenten werden auf einer gemeinsamen Leiterplatte oder auch PCB platziert und über Leiterbahnen miteinander verbunden.

Neben diesen drei Grundformen sind auch Kombinationen dieser Integrationsgrade möglich. Der in dieser Abhandlung vorgestellt Sensor-in-the-Loop-Ansatz ist dabei nicht auf einen Implementierungsform beschränkt. Er eignet sich für die meisten intelligenten Sensoren unabhängig vom Integrationsgrad und spezifischen Implementierungsdetails.

5.2.1 Sensor Firmware

Um den vorgeschlagenen SiL-Ansatz zu realisieren, muss die auf der Sensor-Prozessor-Einheit (Sensor Processing Unit, SPU) oder im Mikrokontroller (Microcontroller, µC) implementierte Firmware angepasst werden. Dies könnte durch die Implementierung einer Bibliothek oder einer Sammlung von Programmfunktionen realisiert werden. Diese vorgefertigten Elemente steuern die Aufzeichnung und Einspeisung der Sensordaten. Im besten Fall ähneln diese Funktionen den ursprünglichen Funktionen der Sensor-API. Dabei müssen die ursprünglichen Funktionen zum Senden und Empfangen von Sensordaten ersetzt werden. Diese bedienen sich üblicherweise Standard-Kommunikationsschnittstellen wie Serial Peripheral Interface (SPI) oder Inter-Integrated Circuit (I^2C). Die Sensor-Firmware muss die Schnittstelle zwischen dem eigentlichen Sensor, der SPU und der Grafische Benutzeroberfläche (Graphical User Interface, GUI) implementieren. Entscheidend ist jedoch, dass die modifizierten Firmware-Funktionen das zeitliche Verhalten der ursprünglichen Funktionen so nahe wie möglich abbilden.

5.2.2 Kommunikationsschnittstelle

Der zweite Teil des SiL-Konzeptes ist die Kommunikationsschnittstelle zwischen der Recheneinheit des Sensors und der Kontrollsoftware (View & Control) auf dem Entwicklungsrechner. Es ist möglich, eine große Anzahl von physikalischen Schnittstellen und Kommunikationsprotokollen zu verwenden. Je nach gewünschter Anzahl verschiedener Sensortypen und Abtastraten jedes einzelnen Sensors sind folgende Kriterien zu berücksichtigen:

- Reihenfolge der Daten
- Zuverlässigkeit
- Durchsatz
- Latenz

Es ist unbedingt erforderlich, dass die Reihenfolge der Datenpakete (Sensordaten) während der Übertragung gleich bleibt. Sollte die Reihenfolge nicht automatisch durch die verwendete Kommunikationsschnittstelle sichergestellt werden, muss dies die jeweilige SiL-Implementierung gewährleisten. Eine einfache Möglichkeit stellt hier zum Beispiel die Verwendung von fortlaufenden IDs oder Zeitstempeln für die einzelnen Sensorsamples dar. Durch eine Nichteinhaltung der Reihenfolge verliert der SiL-Ansatz seinen Anspruch auf Reproduzierbarkeit und die Sensor-Firmware kann nicht zuverlässig validiert werden.

Die Zuverlässigkeit der Schnittstelle stellt sicher, dass während der Übertragung keine Daten verloren gehen. So wird sichergestellt, dass jeder Durchlauf des Sensors mit den aufgezeichneten oder künstlichen Daten genau die gleichen Daten in der gleichen Reihenfolge verwendet. Auch bei dieser Anforderung kann die Verwendung von Sample-IDs oder Zeitstempeln eine mögliche Lösung darstellen.

Der Durchsatz der Schnittstelle sollte größer sein als der eigentliche Sensordatendurchsatz. Dies ist notwendig da in der Regel zusätzliche Informationen wie Kontrolldaten oder Berechnungsergebnisse über die Schnittstellen gesendet werden müssen:

$$R_{min} = \sum_{n=1}^{N} (fs_n \cdot (bw_n \cdot ac_n + P_{OH} + ACK_{OH})) + CI_{OH} \cdot f_T \tag{5.1}$$

Der minimale Durchsatz R_{min} kann mit Gleichung (5.1) geschätzt werden. Der Summenteil berücksichtigt den Einfluss der sensorabhängigen Daten an der Kommunikationsmenge. Dabei fließen die individuelle Frequenz jedes Sensors fs_n, die Bitbreite bw_n und die Anzahl der Signale pro Sensor ac_n (z.B. Sensorachsen) in die Berechnung ein. Hinzu kommen der Protokoll-Overhead P_{OH} und der Quittierungs-Overhead ACK_{OH} pro Sensor. Diese Overheads entstehen durch das Sicherstellen der Kriterien *Reihenfolge der Daten* und *Zuverlässigkeit* in der jeweiligen SiL-Implementierung. Neben den Sensordaten abhängigen Faktoren trägt zum Gesamtdurchsatz noch der Kommunikationsschnittstellen-Overhead CI_{OH} bei. Dieser hängt nur von der gewählten Schnittstelle ab und variiert von nicht vorhanden bis zu einem Vielfachen der tatsächlichen Sensordaten. Der Overhead der Kommunikationsschnittstelle wird mit der Frequenz des Datenpakettransfers f_T multipliziert.

Ein Beispiel für die Durchsatzabschätzung

Der Sensor besteht aus einem 3-dimensionalen 16 Bit-Beschleunigungsmesser und einem 16 Bit-Temperatursensor. Die Sensoren werden mit $2\,kHz$ bzw. $1\,Hz$ abgetastet. Zusätzlich zu den Sensordaten beträgt der Paket-Overhead P_{OH} $5\,Byte$, dies Datenfeld enthält die ID des jeweiligen Sensorsamples. Das Acknowledgment-Paket für jede Abtastung hat eine Länge von $6\,Byte$. Wenn die Kommunikationsschnittstelle *UDP* verwendet, kommt ein zusätzlicher Overhead CI_{OH} von $20\,Byte$ hinzu. Dieser wird mit der Frequenz der einzelnen Datenpakete multipliziert, f_T wird hier mit $100\,Hz$ angenommen. Die Frequenz der Datenpakete kann kleiner als die Abtastfrequenz der jeweiligen Sensoren sein, wenn eine entsprechende Pufferstruktur implementiert wird. Nach Einsetzen der Werte kann der minimale Durchsatz der Schnittstelle für dieses Beispiel mit Gleichung (5.2) geschätzt werden.

$$\begin{aligned} R_{min} &= (2000\,Hz \cdot (2\,B \cdot 3 + 5\,B + 6\,B)) \\ &\quad + (1\,Hz \cdot (2\,B \cdot 1 + 5\,B + 6\,B)) \\ &\quad + 20\,B \cdot 100\,Hz \\ &= 36013, B/s \end{aligned} \tag{5.2}$$

Die Latenzzeit muss gering genug sein, um die Sensordaten für die vorgesehene Abtastrate in Echtzeit zu übertragen. Dieses Kriterium kann vernachlässigt werden, wenn das Validieren der Sensor-Firmware nicht in Echtzeit erfolgen soll. In diesem Fall muss aber unbedingt eine ausreichend große Pufferstruktur für die Kommunikation zur Verfügung stehen. Dieses Puffern der Sensordaten verhindert das Sensorsamples verloren gehen.

Im Allgemeinen sollte es immer noch möglich sein, eine Schnittstelle zu verwenden, die nicht alle diese Anforderungen erfüllt. In diesem Fall ist es allerdings notwendig das jeweilige Kriterium in der SiL-Implementierung auf höheren Protokollebene manuell sicherzustellen.

5.2.3 Visualisierungs- und Steuerungsapplikation

Der letzte Teil des SiL-Konzepts ist die grafische Benutzeroberfläche. Diese Oberfläche wird direkt auf dem jeweiligen Entwicklungsrechner ausgeführt. Dabei wird sie für das jeweilige präferierte Betriebssystem entwickelt. Im besten Fall wird diese Visualisierungs- und Steuerapplikation direkt in die jeweilige Sensor-Firmware-Entwicklungsumgebung integriert. Es bieten sich weit verbreitete integrierte Entwicklungsumgebungen (Integrated Developement Environments, IDEs) wie etwa *Eclipse* oder *VSCode* an. Diese Integration bietet den Vorteil, dass der Entwickler in seiner vertrauten IDE weiterarbeiten kann und das für unterschiedliche Betriebssysteme häufig nur eine Implementierung nötig ist.

Die grafische Benutzeroberfläche ermöglicht die Aufzeichnung der Sensor-Rohdaten des entsprechenden Sensors in der gewünschten Datenrate. Dabei sollte es ermöglicht werden, dass der Entwickler notwendige Konfigurationen wie etwa Anzahl und Art der Sensoren, Sampleraten oder Datenformate direkt in der Oberfläche konfigurieren kann. Weiterhin wäre ein Speichern und Laden der jeweiligen Konfiguration in einem üblichen Datenformat wie zum Beispiel *XML* oder *JSON* wünschenswert. Nach der Aufzeichnung der Daten können diese modifiziert oder direkt zur Injektion verwendet werden. Dies ermöglicht es verschiedene Varianten der Sensor-Firmware mit reproduzierbaren Datenproben zu testen. Die Visualisierung der aufgezeichneten Daten und der entsprechenden Berechnungsergebnisse sollte ebenfalls über die GUI möglich sein. Unter Verwendung des SiL-Konzeptes kann ein Entwickler so auf einfache und gezielte Weise Firmware mit reproduzierbaren Sensordaten aus dem Inertialsensor entwickeln, verbessern oder evaluieren.

5.3 Implementierung

Um die Anwendbarkeit des vorgeschlagenen SiL-Konzepts zu überprüfen und die Vorteile dieses Systems zu untersuchen, wurden die in Abschnitt 5.2 vorgestellten Konzepte in einem Prototypen implementiert. Die Implementierung des Prototyps wird in der gleichen Reihenfolge wie in Abschnitt 5.2 beschrieben. Als erster Schritt wurde die angepasste Firmware auf einem geeigneten intelligenten Sensor implementiert. Im zweiten Abschnitt wird die benutzte Kommunikationsschnittstelle für den Transport der Sensorsamples zwischen Sensor und Entwicklungsrechner beschrieben. Der dritte Teil beschreibt die Implementierung der grafischen Benutzeroberfläche zum Steuern und Beobachten des SiL-Prozesses.

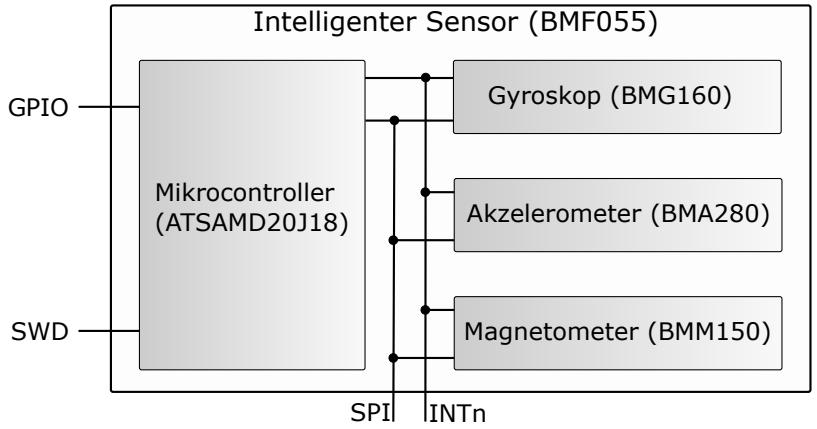


Abbildung 5.3: Schematische Darstellung des BMF055

Um den ersten Teil des SiL-Konzeptes zu implementieren, wurde der *BMF055* verwendet. Der *BMF055* ist ein intelligenter programmierbarer Sensor der Firma *Bosch Sensortec GmbH* [A18]. Dieser Sensor ist als SiP ausgeführt, wie es konzeptionell in Abbildung 5.2(b) dargestellt ist. Der *BMF055* kombiniert einen triaxialen 14-Bit-Beschleunigungsmesser [A51], ein triaxiales 16-Bit-Gyroskop [A52] und einen triaxialen geomagnetischen Sensor [A53] mit einem 32-Bit Cortex M0+ Mikrocontroller [A54]. Abbildung 5.3 zeigt die Architektur des BMF055.

5.3.1 Sensor-Firmware Implementierung

Um die SiL-Beispielfirmware auf dem intelligenten Sensor zu implementieren, wurde auf die *DataStream*-Firmware [A55] von *Bosch Sensortec* aufgebaut. Die *DataStream*-Beispielfirmware ist eine übersichtliche Implementierung, welche die Möglichkeiten des Sensors demonstrieren soll. Die Firmware konfiguriert die im *BMF055* enthaltenen Sensoren. Nach der Konfigurationsphase tastet die Firmware die einzelnen Sensoren mit einer konfigurierbaren Datenrate ab und gibt die erfassten Daten über die Universal Asynchronous Receiver Transmitter (UART) Schnittstelle des µC aus. Abbildung 5.4 visualisiert den Programmfluss der *DataStream*-Beispielfirmware. Die Beispielimplementierung wurde durch die SiL-Bibliothek erweitert. Diese enthält die notwendigen C-Funktionen, um die SiL-Funktionalität zu realisieren. Die erstellte C-Bibliothek bietet Funktionen, um die Sensordaten über die SiL-Schnittstelle an die Entwicklungsrechner zu übertragen. Dort können diese dann aufgezeichnet, angezeigt und nachbearbeitet werden. Darüber hinaus werden Funktionalitäten zur Verfügung gestellt, um die zuvor aufgezeichneten oder künstlich erzeugten Daten wieder in die laufende SPU zu übertragen.

Nachdem die Beispielfirmware auf dem intelligenten Sensor implementiert wurde, kann eine Ermittlung des Speicherbedarfs der SiL-Funktionen durchgeführt werden. Zu diesem Zweck wurden sechs verschiedene Versionen der Sensor-Firmware erstellt. Die erste Version enthält keiner SiL-Funktionalität. In der zweiten Version wurde ausschließlich die Funktion zum Aufzeichnen von Sensordaten verwendet. Eine weitere Version der Firmware stellt den vollständigen SiL-Funktionsumfang zur Verfügung. Diese Firmware wird auch für die späteren Experimente in Abschnitt 5.4 verwendet. Alle diese Firmware-Versionen wurden in zwei Compiler-Optimierungsstufen übersetzt. Zum einen ohne jegliche Optimierungen *O0* und zum anderen mit der maximalen Optimierungsstufe *O3*. Die Firmware-Versionen wurden mit dem *GNU ARM GCC Version 9.2.1* Compiler [A56] erstellt. Um den Speicherbedarf für jede Firmware zu ermitteln, wurde das *Size-Tool* der *GNU ARM Embedded Toolchain* in Version 2.33 verwendet. Der Speicherplatzbedarf für die verschiedenen Versionen ist in Tabelle 5.1 aufgelistet. In dieser Beispielimplementierung verursacht das SiL-Feature einen Overhead von 4.6 $KByte$ für die optimierte Variante und 5 $KByte$ für die nicht optimierte Firmware-Version. Für die Implementierung wurde 1 $KByte$ Puffer-Speicher für jeden der drei Sensoren verwendet, dieser

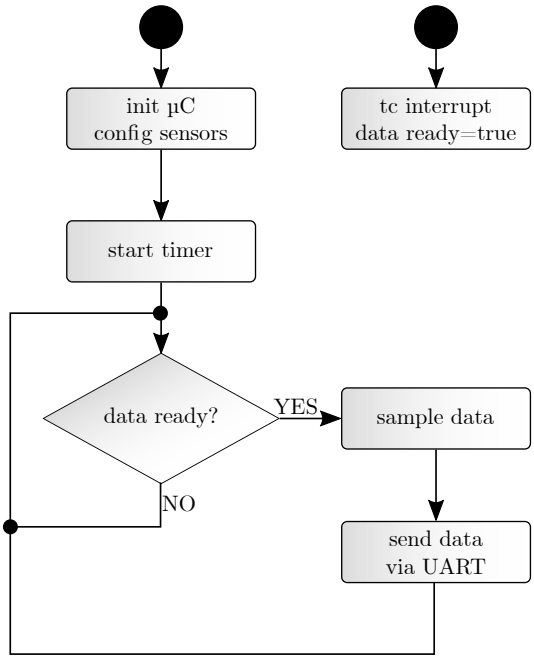


Abbildung 5.4: Programmfluss der entwickelten Beispielfirmware

Puffer-Speicher ist abhängig von der Datenrate und kann in den meisten Anwendungsfällen wesentlich kleiner ausfallen. Ohne die Datenpuffer verwendet die SiL-Funktionalität etwa $0,63\,\%$ des internen Flash-Speichers.

Tabelle 5.1: Flash-Speicherbelegung für die Sensor-Firmware in Byte

Firmware	O0 [Hex]	O0 [Dez]	O3 [Hex]	03 [Dez]
ohne SiL	8f60	36,704	6a1c	27,164
SiL Aufnahme	9ed4	40,660	77e4	30,692
SiL gesamt	a2f4	41,716	7c3c	31,804

5.3.2 J-Link RTT Interface

Der Debug-Process auf dem Entwicklungsrechner steuert den Datenfluss vom Host zum Sensor und visualisiert die entsprechenden Sensordaten. Für die Kommunikation zwischen diesem Prozess und der Sensor-Firmware wird die Real Time Transfer (RTT)-Schnittstellen der Firma *Segger* verwendet. Diese Echtzeit-Kommunikationsschnittstelle bietet eine Vielzahl von Funktionen, welche für die Implementierung des SiL-Interface nützlich sind. Das Übertragungsprotokoll ist in den *J-Link Hardware-Debugger* integriert. Dieser dient dem Firmware-Entwickler zum Übertragen und Debuggen der Sensor-

Firmware. Das bedeutet, dass für die Implementierung der SiL-Funktionen keine zusätzliche Hardware benötigt wird. Weitere Informationen über die verwendete RTT-Schnittstelle finden sich auf der Seite des Herstellers [A57]. Für die Implementierung in dieser Abhandlung wurde der *Segger J-Link Base* verwendet. Dies ist die Lite-Version des Debuggers, welche eine über eine geringere Übertragungsgeschwindigkeit zwischen PC und dem Hardware-Target verfügt. Wie die späteren Experimente zeigen, ist diese Version allerdings für viele Anwendungsfälle ausreichend. Wenn die SiL-Anwendung höhere Datenraten erfordert, wäre die Verwendung des *Segger J-Link Pro, PLUS oder ULTRA+* möglich. Der Debug-Adapter wird über die USB-Schnittstelle mit dem Host-PC verbunden.

Zur Aufnahme des eigentlichen Sensors wird das *BMF055 Shuttle Board* verwendet. Dieses sitzt dann auf einem Breakout-Board von *Bosch Sensortec*. Das Breakout-Board, welches auch als *Application-Board* bezeichnet wird, stellt die Spannungsversorgung für den Sensor bereit. Weiterhin bietet es die Möglichkeit zusätzliche Komponenten mit dem Sensor zu verbinden. Über den 20-poligen Standardstecker auf dem Breakout-Board wird der J-Link unter Verwendung der Serial Wire Debug Interface (SWD)-Schnittstelle mit dem Sensor verbunden. Eine Übersicht über die physikalische Verbindung ist in Abbildung 5.5 dargestellt.

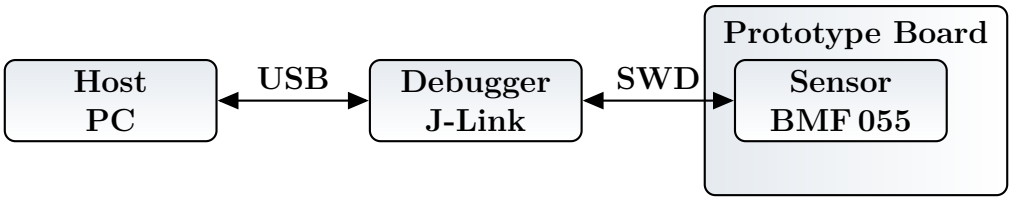


Abbildung 5.5: Physikalische Schnittstellen im SiL Setup

5.3.3 Sensor-View Plugin

Um den gesamten SiL-Prozess während des Debuggens des Sensors zu kontrollieren, bedarf es einer zusätzlichen Anwendung auf dem Entwicklungsrechner. Diese Anwendung wurde für die Implementierung dieser Arbeit als Eclipse-Plugin ausgeführt. Die Verwendung eines Eclipse-Plugins ermöglicht es, alle SiL-Funktionen für den Firmware-Entwicklungsprozess aus einer gängigen Open-Source-IDE heraus zu nutzen. Für die Kommunikation mit dem Target bietet der verwendete J-Link einen TCP-Socket. Die Verwendung der Standard-Socket-Implementierung in Java ermöglicht es der Anwendung, den Datenfluss der SiL-Pakete für die Sensoranwendung zu kontrollieren. Nachdem der Debugger die Daten vom Socket empfangen hat, verwendet er die RTT-Schnittstelle, um mit dem Sensor selbst zu kommunizieren. Die verwendeten Verbindungsprotokolle sind in Abbildung 5.6 schematisch dargestellt.

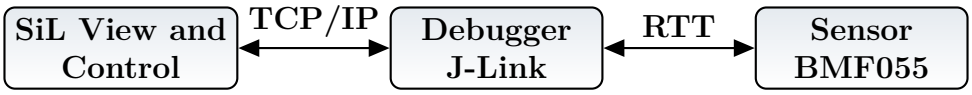


Abbildung 5.6: Kommunikationsprotokolle im SiL Setup

In der Basisimplementierung bietet das Eclipse-Plugin drei Funktionen. Nach dem Start des Debug-Prozesses ist es mit der Standard-Debug-Target-Option von *Eclipse* möglich, das SiL-Plugin mit dem laufenden Sensor innerhalb der Sensoransicht zu verbinden. Nachdem die Verbindung erfolgreich hergestellt wurde, können die Sensordaten direkt vom Sensor abgerufen werden. Die empfangenen Daten werden live in der Sensoransicht visualisiert. Dabei spielt es keine Rolle, ob das Target ohne Unterbrechung läuft oder ob der Firmware-Entwickler den Firmware-Code schrittweise ausführt. In jedem Fall sind die Sensordaten in der Darstellung immer synchron mit denen der Sensor-Firmware. Sobald die Daten abgerufen und in der Sensoransicht visualisiert wurden, kann der komplette Datensatz oder eine Teilmenge der Daten in einer CSV-Datei gespeichert werden. Diese Daten können in späteren Sitzungen verwendet oder mit Kollegen geteilt werden. Dies ermöglicht es dem Firmware-Entwickler Sensordaten während seines Entwicklungsprozesses für reproduzierbare Tests zu verwenden.

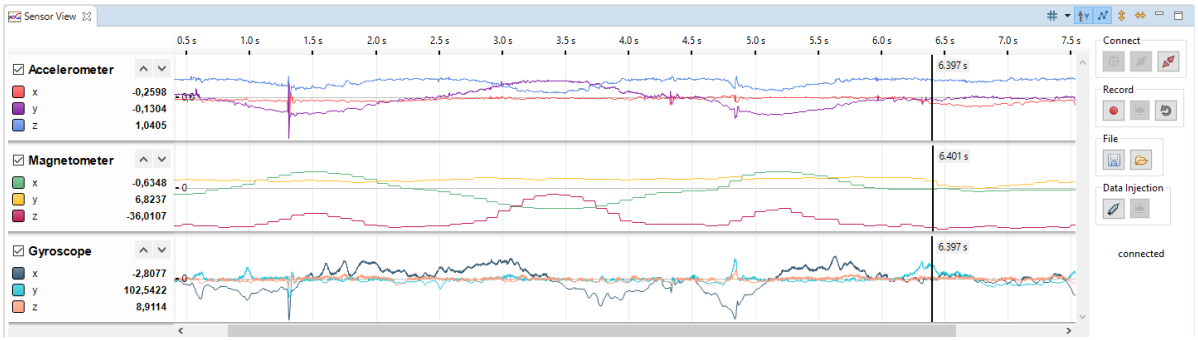


Abbildung 5.7: Darstellung in der PC-Applikation

Abbildung 5.7 zeigt die Sensoransicht nach der Aufzeichnung von Sensordaten. Auf der x-Achse ist der Zeitfortschritt zu sehen, auf der y-Achse die Amplituden der Sensorsignale. Es wurden die Daten von einem jeweils dreidimensionalen Beschleunigungssensor, Magnetometers und Gyroskops aufgezeichnet. Die Ansicht kann Signale mit unterschiedlichen Abtastraten verarbeiten. In der Abbildung wird der Beschleunigungsmesser beispielsweise mit 800 Hz, das Magnetometer mit 100 Hz und das Gyroskop mit 1,6 kHz abgetastet.

Alle Bedienelemente befinden sich auf der rechten Seite der Ansicht. Diese Elemente sind in vier Gruppen gegliedert, die den Schritten der Verwendung der SiL-Funktion folgen. Die erste Gruppe behandelt den Konfigurations- und Verbindungsprozess. Hier

können Darstellungsparameter und Sensorparameter wie die Abtastrate, Kommunikationsinterface usw. eingestellte werden. Weiterhin steuert sie den Verbindungsprozess mit dem Sensor. Die folgende Gruppe ermöglicht es dem Entwickler, die Aufzeichnung der konfigurierten Signale zu steuern. Die dritte Gruppe ist für die Dateiverwaltung gedacht, so kann man aufgezeichnete Sensorsamples als CSV-Datei speichern oder zuvor gespeicherte Dateien laden. Existieren bereits aufgezeichnete Daten, so ist es mit der letzten Gruppe möglich, diese Daten an den angeschlossenen Sensor zu übertragen. Über die Auswahlbox links neben den einzelnen Signalen kann ausgewählt werden welches Sensorsignal im nächsten Lauf übertragen wird. Natürlich ist es auch möglich, während der Injektionsphase Datenaktualisierungen zu erhalten. Darüber können aus den injizierten Daten abgeleitet Daten visualisiert werden. Ein Beispiel hierfür wäre die Quaternion-Darstellung der Sensorausrichtung, die anhand der Winkelgeschwindigkeit und Beschleunigungsdaten berechnet wurde.

5.3.4 Datenpuffer Strukturen

Das SiL-*Eclipse*-Plugin soll mit verschiedenen Sensoren und verschiedenen Verbindungsschnittstellen zur Sensorhardware nutzbar sein. Bei verschiedenen Kommunikationsschnittstellen können die Bandbreite und die Latenzzeit stark voneinander abweichen. Um mit dieser möglicherweise höheren Latenz umzugehen, wurde eine zusätzliche Pufferstruktur in das Eclipse-Plugin implementiert.

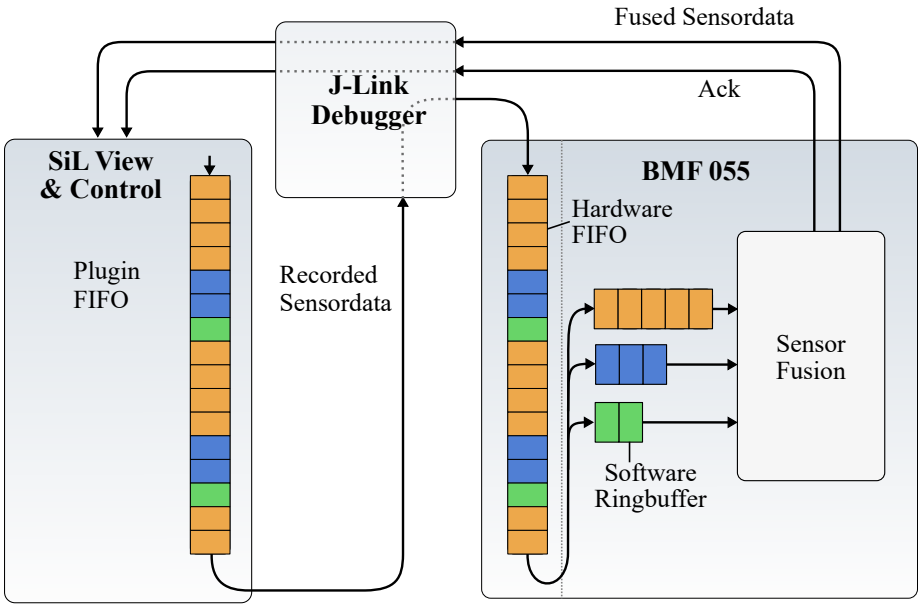


Abbildung 5.8: Aufbau der Datenpufferstruktur im SiL

Abbildung 5.8 stellt diese Pufferimplementierung grafisch dar. Die Implementierung ist in drei separate Puffer aufgeteilt. Diese sind in verschiedenen Teilen der Toolchain implementiert sind. Der erste Puffer ist als klassischer First In First Out (FIFO)-Puffer aufgebaut, der für jeden Sensor eine unterschiedliche Anzahl von Sensorproben speichert. Die Länge dieses FIFOs ist konfigurierbar und hängt von der Latenz der verwendeten Schnittstelle ab. Normalerweise haben verschiedene Sensortypen eines Sensor-Hubs unterschiedliche Abtastraten. Um diese unterschiedlichen Raten bewältigen zu können, müssen häufig abgetastete Sensoren mehr Platz in diesem Puffer erhalten als Sensoren, welche mit einer niedrigeren Frequenz abgetastet werden. Der Platz im Speicher für jeden einzelnen Sensor wird mit Gleichung (5.3) berechnet. In diese Gleichung gehen die Frequenz f_i und die Summe der Frequenzen aller Sensoren f_{ges} ein. Außerdem muss die Größe des Puffers $B_{SIZE_{MAX}}$ und die Anzahl der Sensoren festgelegt werden. Ohne Berücksichtigung der tatsächlichen Abtastung erhält jeder Sensor Platz für mindestens 2 Abtastungen im FIFO.

$$B_{SIZE_N} = \max(2, \frac{f_i}{f_{ges}} \cdot B_{SIZE_{MAX}} - ((N-1) \cdot 2)) \tag{5.3}$$

$$f_{ges} = \sum_{n=1}^{N} f_n \tag{5.4}$$

Auf der Seite der Sensor-Firmware werden die eingehenden Daten zu Beginn in eine Hardware-FIFO und später in einzelne Puffer für jeden Sensor aufgeteilt. Diese Puffer sind als Ringpuffer implementiert, um einen zuverlässigen und effizienten Datenzugriff auf der Sensorseite zu gewährleisten. Die übertragenen Daten in Richtung Sensor zu GUI werden nicht extra durch eine Softwarestruktur gepuffert. Diese Richtung ist nicht sonderlich zeitkritisch, da sie hauptsächlich der Aufzeichnung und Visualisierung von Daten dient. Allerdings haben alle Datenpakete einen individuellen Zeitstempel, um das zeitliche Verhalten nach der Übertragung an das SiL-View- & Control-Plugin zu rekonstruieren.

5.4 Timinganalyse

Nachdem die SiL-Funktionalität wie in Abschnitt 5.3 beschrieben implementiert wurde, werden in diesem Kapitel Testverfahren benutzt, um das zeitliche Verhalten der Implementierung zu untersuchen. Das zeitliche Verhalten der Implementierung ist entscheidend, um das korrekte Verhalten der Sensor-Firmware mit den über das SiL-Interface eingespielten Sensordaten nicht zu verfälschen [A58]. Zum Testen des Konzepts wurde die Firmware-Implementierung des in Abbildung 5.4 gezeigten Datenstrom-Beispiels

verwendet. Mit dieser Firmware wurde die Zeit für die Abtastung von Sensordaten über die SiL-Schnittstelle im Vergleich zur Abtastung realer Sensordaten gemessen.

Es existieren verschiedene Methoden der Zeitmessung in Bare-Metal-C-Programmen, die auf einem µC oder auch einem intelligenten Sensor verwendet werden. Ein möglicher Weg ist die Verwendung eines Timers wie beispielsweise eines Systemtimers oder eines sogenannten General-Purpose-Timers. Mit diesem Timer könnte man die Dauer der Datenabtastung messen und diese als Anzahl der Taktzyklen angeben. Da die Taktfrequenz des jeweiligen Timers bekannt ist, lässt sich aus den gezählten Taktzyklen die tatsächliche Zeit berechnet. Diese Methode kann ohne zusätzliche Hardware verwendet werden und führt je nach verwendetem Timer möglicherweise auch zu einer genauen Messung. Um eine gültige und genaue Zeit mit dieser Methode zu erhalten, muss sichergestellt werden, dass der Zeitgeber während des gesamten Experiments einen bekannten und vor allem auch einen konstanten Timer-Wert bzw. Periode hat.

Eine andere Methode der Zeitmessung ist die Verwendung externer Messgeräte wie Oszilloskope oder Logikanalysatoren engl. Logic-Analyzer. Hierfür muss die Firmware ein externes Signal erzeugen welches dann über ein solches Gerät gemessen werden kann. Dies kann unter Verwendung eines General Purpose Input Output (GPIO)-Pins geschehen. Jedes Mal zu Beginn der Messperiode wird der Pin auf Logisch-Eins oder auch High-Pegel gesetzt und am Ende wieder auf Logisch-Null oder auch Low-Pegel. Die Dauer des geänderten Spannungspegels wird dann mit einem Logikanalysator oder Oszilloskop gemessen. Bei dieser Methode ist die Genauigkeit des Ergebnisses unabhängig von der Firmware des intelligenten Sensors und den integrierten Timern. Die zeitliche Auflösung und Genauigkeit wird nur durch die verwendete Messhardware bestimmt. Aus diesem Grund wurde für die alle zeitlichen Messungen in dieser Abhandlung diese zweite Methode verwendet. Für die Messungen wurde der *Saleae Logic Pro 8 USB Logic Analyzer* verwendet [A59]. Dieses Gerät ist ein 8-Kanal-500-MSPS-Logik-Analyzer. Die zeitliche Auflösung des Gerätes liegt bei 2 ps. Da die Sensorereignisse im SiL-Kontext im µs Bereich liegen, ist die Auflösung des verwendeten Messgerätes für diese Anwendung ausreichend.

5.4.1 Ausführungszeit

In diesen Experimenten wird die benötigte Zeit für die üblichen Sensorabtastung des realen Sensors und die Zeit für die über die SiL-Schnittstelle abgetasteten Sensordaten ermittelt. Die Zeiten für die Abtastung der SiL-Schnittstelle sollten im Idealfall gleich oder kleiner sein als über die tatsächliche physikalische Sensorschnittstelle. Im ersten Versuch erfolgt die Kommunikation mit den physikalischen Sensoren über die SPI-Schnittstelle, die mit einem Bustakt von $10, MHz$ betrieben wird. Die Sensordaten

wurden für den zweiten Test mit dem SiL-Interface aufgezeichnet und in Comma Separated Values (CSV)-Dateien gespeichert, um sie später wiederzuverwenden. Der Aufzeichnungsteil ist in Abbildung 5.1 in Abschnitt 5.2 dargestellt. Diese Abbildung zeigt die gesamte konzeptionelle Struktur der SiL-Architektur welche in die Aufzeichnungs- und Speicherfunktionalität und den Injektionsbereich unterteilt ist. Während der Aufzeichnung der Sensordaten wurde die Referenzzeit für die Abtastung der realen Sensordaten gemessen. Wie in Tabelle 5.2 und Tabelle 5.3 dargestellt sind für dieses Experiment drei verschiedene Szenarien gemessen worden.

Für alle Messungen wurde ein Datensatz von jeweils $10\,s$ Länge verwendet. Beim ersten Test wurden nur die Daten des Beschleunigungssensors mit einer Abtastrate von $1600\,Hz$ abgetastet. In den beiden anderen Tests wurden alle drei verfügbaren physikalischen Sensoren verwendet. Im ersten dieser Fälle wurden alle drei Sensoren mit der gleichen Abtastrate von $1600\,Hz$ betrieben. Der letzte Test verwendet unterschiedliche Abtastraten für jeden der drei Sensoren, um eine unausgewogene Füllung der Ringspeicher zu untersuchen. In diesem Fall wurde der Beschleunigungssensor mit $800\,Hz$, das Gyroskop mit $1600\,Hz$ und das Magnetometer mit $100\,Hz$ abgetastet.

Wie eingangs beschrieben, wird ein Logikanalysator verwendet, um die Zeiten zwischen zwei Zustandswechseln eines GPIO zu messen. Die Anweisungen zum Umschalten dieses GPIO und auch der Pegelwechsel selbst benötigen Zeit. Diese Zeit muss hier als Overhead berücksichtigt werden. Um diesen zeitlichen Overhead zu bestimmen, wurde die Zeit zwischen den Pegelwechseln eines GPIO gemessen.

Die Zeit für die Aufzeichnung der Sensordaten über die SiL-Schnittstelle wurde zusätzlich bestimmt. Diese Zeit ist mit keiner Funktion des Sensors ohne SiL-Schnittstelle vergleichbar. Es ist trotzdem eine relevante Messung, da sie zeigt wie sehr die Aufzeichnungsfunktion das Zeitverhalten der Firmware beeinflusst.

5.4.2 Ergebnisse: Ausführungszeit

Die zuvor beschriebenen Experimente konnten mit der SiL-Implementierung erfolgreich durchgeführt werden. Es sind weder bei der Aufzeichnung der Sensordaten noch bei der Injektion in den Sensor Sensordaten in Ihrer Reihenfolge vertauscht worden oder verloren gegangen. Weiterhin war die Injektion schnell genug, um die Re-Injektionsversuche im zeitlich korrekten Verlauf durchzuführen.

Die Abbildung 5.9 zeigt die Mittelwerte der Ausführungszeit für die einzelnen Testfälle (Szenarien) im direkten Vergleich. Dabei sind Szenario eins und zwei miteinander vergleichbar, welche die Werte des Beschleunigungssensors auslesen. In den Szenarien drei und vier werden die Werte aller drei Sensoren ausgelesen. Szenario fünf liest auch die Werte aller drei Sensoren aus, allerdings sind die Samplingraten der einzelnen Sensoren verschieden. In Szenario sechs sind die Zeiten für das Umschalten des GPIO gemessen

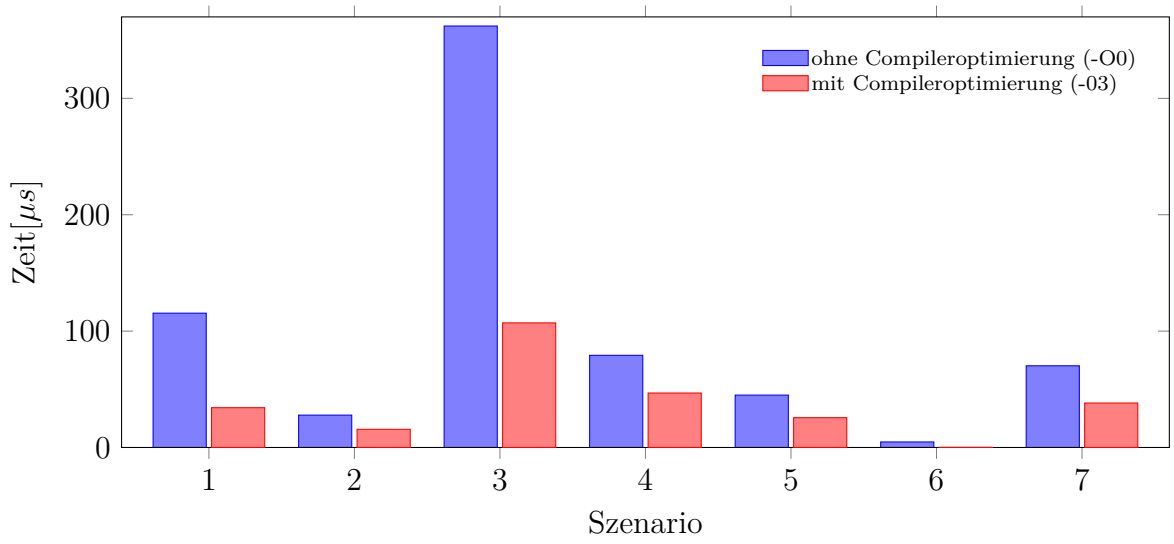


Abbildung 5.9: Ausführungszeiten der einzelnen Versuche mit und ohne Compileroptimierung (Szenario ID in Tabelle 5.2 oder Tabelle 5.3)

worden. Das letzte Szenario sieben, gibt die Zeiten für die Übertragung von Daten vom Sensor hin zum Entwicklungsrechner an.

In Tabelle 5.2 und Tabelle 5.3 sind die gemessenen Zeiten für alle drei Experimente tabellarisch dargestellt.

Die Abtastung der echten Sensordaten über die SPI-Schnittstelle (ID 1) dauerte $1,15 \cdot 10^{-4}\,s$, im Vergleich dazu dauerte die Abtastung mit dem SiL-Interface (ID 2) $2,78 \cdot 10^{-5}\,s$. Die Übertragung mit dem SiL-Interface ist viermal schneller als das Auslesen des echten Sensors über die SPI-Schnittstelle. Dies ermöglicht es den implementierten Ansatz für ein reproduzierbares Echtzeit-Debugging zu nutzen.

Tabelle 5.2: Zeitverhalten für unoptimierte Firmware (-O0)

ID	Name	Max[s]	Min[s]	Mittel[s]	Std[s]
1	get_acc_spi_api_1600	$1.17 \cdot 10^{-4}$	$1.15 \cdot 10^{-4}$	$1.15 \cdot 10^{-4}$	$1.34 \cdot 10^{-7}$
2	get_acc_RTT_1600	$2.93 \cdot 10^{-5}$	$2.70 \cdot 10^{-5}$	$2.78 \cdot 10^{-5}$	$3.21 \cdot 10^{-7}$
3	get_all_spi_api_1600	$3.64 \cdot 10^{-4}$	$3.62 \cdot 10^{-4}$	$3.62 \cdot 10^{-4}$	$2.51 \cdot 10^{-7}$
4	get_all_RTT_1600	$9.47 \cdot 10^{-5}$	$6.36 \cdot 10^{-5}$	$7.92 \cdot 10^{-5}$	$1.86 \cdot 10^{-6}$
5	get_all_RTT_800_1600_100	$8.64 \cdot 10^{-5}$	$2.85 \cdot 10^{-5}$	$4.50 \cdot 10^{-5}$	$1.20 \cdot 10^{-5}$
6	pin_toggle_reference	$6.00 \cdot 10^{-6}$	$4.73 \cdot 10^{-6}$	$4.74 \cdot 10^{-6}$	$1.98 \cdot 10^{-8}$
7	record_Acc_1600	$7.30 \cdot 10^{-5}$	$6.95 \cdot 10^{-5}$	$7.02 \cdot 10^{-5}$	$2.78 \cdot 10^{-7}$

Für n Sensoren benötigt die SPI-Abtastung die n-fache Zeit, da das Auslesen für jeden Sensor gleich implementiert ist. Es werden lediglich unterschiedliche Adressen verwendet.

So ergibt sich für den Test mit der ID 3 eine durchschnittliche Zeit von $3,62 \cdot 10^{-4}\,s$. In Abbildung 5.8 in Abschnitt 5.3 ist die verwendete Pufferstruktur dargestellt. Aufgrund dieser Struktur werden die Daten manchmal aus einem Ringpuffer und manchmal aus dem Hardware-FIFO unter Verwendung der RTT-Schnittstelle ausgelesen. Dieses Verhalten führt zu einem etwas anderen Timing als bei der SPI-Kommunikation. Deshalb ist die Zeit für das Auslesen von drei Sensoren nicht genau 3-mal länger. Mit $7,92 \cdot 10^{-5}\,s$ ist die Messung für das Sampling aller drei Sensoren über die SiL-Schnittstelle noch mehr als 4-mal schneller als die Abtastung mit SPI.

Die implementierte Pufferstruktur wirkt sich stärker auf das Timing aus, wenn die Daten für verschiedene Sensoren unterschiedliche Abtastraten haben. Dies führt zu einem unausgewogenen Füllstand der Ringspeicher. Die durchschnittliche Zeit für diesen Testfall (ID 5) war mit $4,5 \cdot 10^{-5}\,s$ geringer als die Zeit für die Verwendung derselben Abtastrate aber die Standardabweichung nimmt in diesem Fall zu. Der Anstieg der Standardabweichung ist darauf zurückzuführen, dass bei der Abtastung häufiger zwischen Ringpuffer und Hardware-FIFO gewechselt wird.

Die Zeit für das reine Umschalten des GPIO in dem Experiment mit der ID 6 beträgt $4.74 \cdot 10^{-6}\,s$ diese Zeit ist in allen anderen Messungen als Overhead enthalten. Dieser Wert muss von den anderen Werten subtrahiert werden, um die tatsächlichen Zeiten für die jeweiligen Operationen zu erhalten.

Die Datenübertragung vom Sensor zur SiL-Hostschnittstelle dauert $7,02 \cdot 10^{-5}\,s$. Aufgrund der schnellen Übertragung ist sie für die Aufzeichnung und Beobachtung von Sensordaten in einer laufenden Firmware in Echtzeit nutzbar.

Tabelle 5.3: Zeitverhalten für optimierte Firmware (-O3)

ID	Name	Max[s]	Min[s]	Mittel[s]	Std[s]
1	get_acc_spi_api_1600	$3.52 \cdot 10^{-5}$	$3.41 \cdot 10^{-5}$	$3.42 \cdot 10^{-5}$	$5.03 \cdot 10^{-8}$
2	get_acc_RTT_1600	$1.67 \cdot 10^{-5}$	$1.49 \cdot 10^{-5}$	$1.56 \cdot 10^{-5}$	$2.47 \cdot 10^{-7}$
3	get_all_spi_api_1600	$1.08 \cdot 10^{-4}$	$1.07 \cdot 10^{-4}$	$1.07 \cdot 10^{-4}$	$9.40 \cdot 10^{-8}$
4	get_all_RTT_1600	$5.70 \cdot 10^{-5}$	$3.75 \cdot 10^{-5}$	$4.68 \cdot 10^{-5}$	$1.06 \cdot 10^{-6}$
5	get_all_RTT_800_1600_100	$5.12 \cdot 10^{-5}$	$1.51 \cdot 10^{-5}$	$2.56 \cdot 10^{-5}$	$7.51 \cdot 10^{-6}$
6	pin_toggle_reference	$8.00 \cdot 10^{-8}$	$7.80 \cdot 10^{-8}$	$7.87 \cdot 10^{-8}$	$9.43 \cdot 10^{-10}$
7	record_Acc_1600	$4.00 \cdot 10^{-5}$	$3.70 \cdot 10^{-5}$	$3.82 \cdot 10^{-5}$	$4.20 \cdot 10^{-7}$

In Tabelle 5.3 sind die Messwerte für die mit (-O3) optimierten Firmwareversionen aufgelistet. Die Verwendung von (-O3) als Optimierungsstufe für die Kompilierung beschleunigt die Vorgänge sowohl für die Abtastung über SPI als auch für die Abtastung über die SiL-Schnittstelle. Die Optimierung scheint einen größeren Einfluss auf das Sampling über das SPI-Interface zu haben. Daher ist die Verbesserung in der Geschwindigkeit bei SPI größer als bei dem SiL-Interface. Das Sampling über das SiL ist auch bei dieser Optimierung noch doppelt so schnell wie das Auslesen der realen Sensoren. Über dieses

Verhalten hinaus ist keine Besonderheit bei der Verwendung der Optimierungsstufe (-O3) festzustellen. Die Werte werden hier nicht noch einmal im Detail gegenübergestellt, da in vielen Sensoranwendungen ausschließlich nicht automatisch optimierte Firmware zum Einsatz kommt. Besonders während der Entwicklung beim sogenannten Debuggen von Firmware wird in der Regel keine optimierte Firmware verwendet. Da das SiL-Konzept der Entwicklung und dem Debuggen von Sensor-Firmware dient sind die zuerst vorgestellten Werte für (-O0) relevanter.

5.4.3 Auswertung: Ausführungszeit

Das Zeitverhalten der implementierten SiL-Schnittstelle vor allem die Ausführungszeit ist entscheidend für den erfolgreichen Einsatz des vorgeschlagenen Ansatzes. Dabei muss die implementierte SiL-Schnittstelle mindesten genauso schnell sein wie die Schnittstelle zu den realen Sensorelementen. Idealerweise ist diese Schnittstelle wesentlich schneller, dann kann nachträglich das korrekte zeitliche Verhalten abgebildet werden. Dieses ist dann nachträglich über Firmware-Implementierungen zum Beispiel über Timer oder Warteroutinen zu realisieren.

Die Messergebnisse für die Beispielimplementierung erfüllen diese Anforderungen. Für die nicht Compiler optimierte Firmware ist das Abfragen der Sensordaten über die SiL-Schnittstelle viermal schneller als über die SPI-Schnittstellen. Auch für die optimierte Version zeigt sich ein ähnliches Bild. Hier zeigte sich allerdings nur die doppelte Geschwindigkeit im Gegensatz zum Abtasten der realen Sensordaten. Für viele Anwendungen im Bereich realer Sensoren wird keine automatische Compiler Optimierung verwendet. Vor allem in der Entwicklungsphase und während des Debuggens ist es nicht üblich eine Compileroptimierung zu verwenden. Daher sind die Werte ohne Optimierung (-O0) als relevanter anzusehen.

5.4.4 Jitter

Für viele Anwendungen im Bereich der intelligenten Sensoren ist es ausreichend das Zeitverhalten wie im vorhergehenden Abschnitt zu untersuchen und zu reproduzieren. Allerdings kann es auch notwendig sein, das Zeitverhalten noch genauer abzubilden. Um das Timing-Verhalten des Sensor-in-the-Loop-Ansatzes genauer zu untersuchen, wurden unterschiedliche Arten von Jitter innerhalb verschiedener Methoden zur Abtastung von Sensordaten untersucht.

Untersuchungen zeigen, dass Jitter einen großen Einfluss auf das Gesamtergebnis von sensorbasierten Berechnungen haben kann [A58]. Viele Publikationen beschäftigen sich mit dem Einfluss des Jitters auf Regelsysteme und wie dieser zu handhaben ist. In [A60] wird der Einfluss von Sampling-Jitter auf einen Servomotorregler gezeigt und wie dieser

möglicherweise zu minimieren ist. Die Publikation [A61] zeigt den Effekt von Sampling-Jitter auf einen PID-Regler, welcher auf einem µC implementiert ist. Die Autoren schreiben, dass dieser Jitter zu einer Instabilität des gesamten Systems führen kann. Auch in [A62] kommen die Autoren zu dem Schluss, dass Jitter einen signifikanten Einfluss auf Regelsysteme haben kann. Diese Beispiele zeigen die Relevanz des Zeitverhaltens, insbesondere des Jitters für sensorbasierte Systeme.

Dieses Kapitel gibt zuerst einen kurzen Überblick über die verschiedenen Arten von Jitter. Dabei werden die in dem experimentellen Aufbau untersuchten Jitter genannt und genauer beschrieben.

Im aktuellen Stand der Literatur wird Jitter als eine zeitliche Abweichung von der tatsächlichen Taktflanke oder Periodizität in einem periodischen Signal beschrieben [A63]–[A65]. Normalerweise setzt sich der Gesamtjitter aus zwei Kategorien zusammen, dem zufälligen Jitter und dem deterministischen Jitter [A65]. Abbildung 5.10 zeigt einen Überblick über die verschiedenen Arten von Jitter und ihre Beziehungen zueinander.

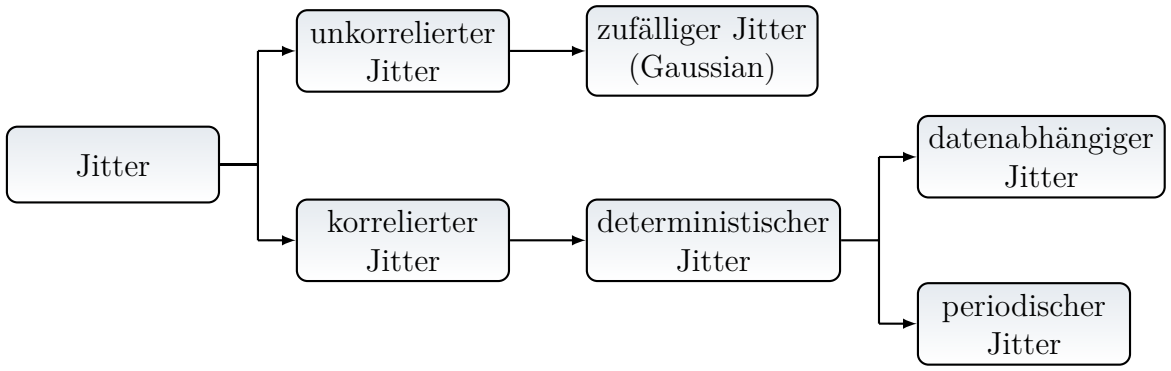


Abbildung 5.10: Arten von Jitter

Zufälliger Jitter ist in jedem System vorhanden und ist ein unbegrenzter breitbandiger stochastischer Gauß-Prozess. Manchmal wird er auch als intrinsisches Rauschen bezeichnet. Zufälliger Jitter kann einen signifikanten Beitrag zum Gesamtjitter in einem System leisten. Diese Art von Rauschen hat immer eine Gaußsche Verteilung und breitet sich unbegrenzt aus. [A64].

Deterministischer Jitter hat immer eine bestimmte Ursache, er kann periodisch sein und ist oft schmalbandig. Die Periodizität zeigt sich in einer oder mehreren Frequenzen des Signals [A63]. Es gibt hauptsächlich zwei Arten von deterministischem Jitter. Die erste Art wird als periodischer Jitter bezeichnet und kann durch ein Schaltnetzteil, ein nahes Taktsignal oder sogar durch die Implementierung des Systemtakts selbst verursacht werden. Die zweite Art von deterministischem Jitter ist der datenabhängige Jitter. Eine Erkennung dieser Art von Jitter ist nicht so einfach, da er vom System, der implementierten Funktionalität und anderen Faktoren abhängt [A63]. Dies kann zu einer

Fehlinterpretation von datenabhängigem Jitter als unkorreliertem oder zufälligem Jitter führen.

Darüber hinaus kann Jitter in korrelierten und unkorrelierten Jitter unterteilt werden. Korrelierter Jitter ist immer deterministisch und mit einer bestimmten Quelle korreliert. Er kann periodisch oder aperiodisch sein. Unkorrelierter Jitter, z. B. zufälliger Jitter, ist statistisch nicht mit einer bestimmten Quelle verbunden. Wenn sich verschiedene korrelierte Jitter überlagern, können sie ohne genauere Analyse als unkorreliert erscheinen.

In der Literatur sind verschiedene Jitter-Messungen beschrieben [A63]–[A65], die dem aktuellen Stand der Technik entsprechen. Für die später in dieser Abhandlung dargestellten Ergebnisse wurden drei Arten von Jitter-Messungen verwendet.

5.4.4.1 Cycle-to-Cycle-Jitter

Cycle-to-Cycle-Jitter (Jcc) kann als die Zeitdifferenz zwischen zwei benachbarten und aufeinanderfolgenden Taktzyklen beschrieben werden. In Abbildung 5.11 werden mehrere Perioden eines Signals dargestellt. Alle dargestellten Perioden P_1 bis P_n sind nicht ideal. Das heißt, ihre Länge ist zeitlich unterschiedlich im Vergleich zur gewünschten Länge einer für dieses Signal definierten Periode.

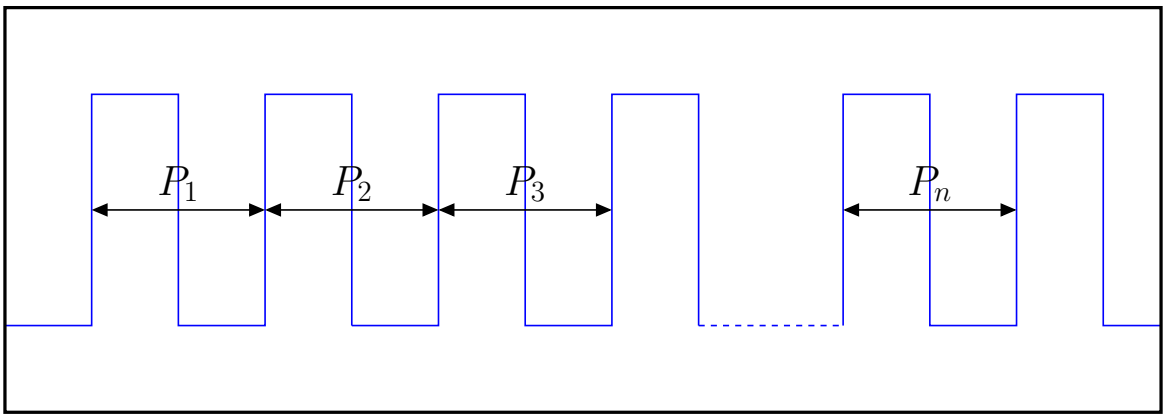


Abbildung 5.11: Cycle-to-Cycle-Jitter

Unter Verwendung von Gleichung (5.5) kann der Cycle-to-Cycle-Jitter für das gemessene Signal berechnet werden. Jcc ist dabei eine Liste von Werten, die die einzelnen Jcc-Werte für jede Periode des Signals enthält. Die Symbole P_1 bis P_n stehen für die einzelnen Perioden des betrachteten Signals.

$$Jcc = |P_2 - P_1|, |P_3 - P_2|, ..., |P_n - P_{n-1}| \tag{5.5}$$

5.4.4.2 Peak-to-Peak Cycle-to-Cycle-Jitter

Um die größte Differenz zwischen den Perioden in einem gemessenen Signal zu bestimmen, wird peak-to-peak Cycle-to-Cycle-Jitter (Jpp) verwendet. Wie in Gleichung (5.6) modelliert, stellt er die Differenz zwischen der kürzesten und der längsten Periode aller Abtastwerte einer Messreihe dar.

$$Jpp = max(Jcc) - min(Jcc) \tag{5.6}$$

Diese Perioden können aufeinanderfolgen oder aneinander angrenzen. In diesem Fall entspricht der Jpp dem Höchstwert des Jcc. Es ist aber auch möglich, dass die längste und die kürzeste Periode an einer beliebigen Stelle in der Messreihe auftauchen.

5.4.4.3 Periodic-Jitter

Der Periodic-Jitter (Jper) ist die Abweichung zwischen der individuellen Periode und der idealen Periode. In manchen Fällen gibt es keine ideale Referenz, mit der die einzelnen Perioden verglichen werden können. In diesem Fall wird üblicherweise das arithmetische Mittel über alle abgetasteten Perioden als Referenz verwendet.

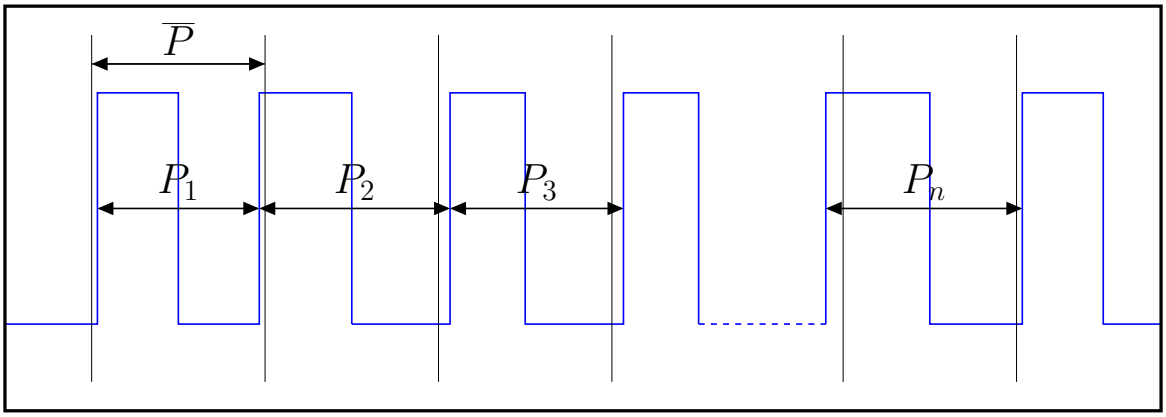


Abbildung 5.12: Periodic-Jitter

Das Signal in Abbildung 5.12 wird durch seine Perioden P_1 bis P_n beschrieben. Um den Jper für jede Periode zu berechnen, wird der Mittelwert aller Perioden $\overline{P}$ verwendet.

$$Jper = |\overline{P} - P_1|, |\overline{P} - P_2|, ..., |\overline{P} - P_n| \tag{5.7}$$

Der Periodic-Jitter wird wie in Gleichung (5.7) dargestellt berechnet. *Jper* ist eine numerische Liste, die den Jper-Wert für jede einzelne Periode innerhalb des gemessenen Signals enthält. Das Symbol $\overline{P}$ ist der Mittelwert über alle Einzelperioden P_1 bis P_n.

5.4.5 Jitter Experimente

Alle im vorigen Abschnitt beschriebenen Formen von Jitter wurden gemessen und bestimmt, um das zeitliche Verhalten des implementierten Sensor-in-the-Loop-Ansatzes weiter zu untersuchen. Zu diesem Zweck wurden die in Unterabschnitt 5.4.1 gezeigten Ausführungszeit-Experimente durch die Jitter-Experimente in diesem Abschnitt umfassend erweitert.

5.4.5.1 Messungen

Um den Jitter des implementierten SiL-Ansatzes zu vergleichen, wurde Zeit für verschiedene Sensorabtastmethoden im zeitlichen Verlauf gemessen. Bei diesen Experimenten wurde nur die Injekionsphase also das Abfragen von Sensordaten der SiL-Schnittstelle berücksichtigt. Das Zeitverhalten dieser Phase ist wichtig, weil es der Abtastung der realen oder Live-Sensordaten so ähnlich wie möglich sein muss. Das Zeitverhalten der Aufzeichnungsphase ist unkritisch da die aufgezeichneten Sensordaten einen Zeitstempel haben und zuverlässig rekonstruiert werden können.

Tabelle 5.4: Übersicht der Jitter-Experimente

Szenario ID	Szenario
1	int_acc_int_spi_sample
2	int_gyr_int_spi_sample
3	timer_int
4	timer_int_9dof_SiL_sample
5	timer_int_9dof_spi_sample
6	timer_int_acc_SiL_sample
7	timer_int_acc_spi_sample

In Tabelle 5.4 werden alle ausgewerteten firmwareabhängigen Testszenarien der Jitter-Messung aufgelistet. Alle diese Szenarien wurden sowohl in einer compileroptimierten Version als auch in einer nicht optimierten Version der Sensor-Firmware gemessen. Für die optimierte Firmware wurde die Optimierungsstufe (-O3) und für die nicht optimierte Version die Stufe (-O0) verwendet. Neben diesen firmwareabhängigen Szenarien wurde auch das Zeitverhalten des *Data Ready Interrupts* der im Smart Sensor verwendeten Hardware-Sensoren gemessen. Diese Messungen werden in den späteren Ergebnissen nicht mit den anderen Werten verglichen, da sie nicht von der auf dem intelligenten Sensor laufenden Firmware abhängen. Der Jitter der einzelnen Sensorelemente findet trotzdem Beachtung, da er in den Messungen von Szenario eins und zwei enthalten ist.

Szenario eins misst die Zeit zwischen zwei Sensorabtastungen (fallende Flanke zu fallender Flanke) unter Verwendung des *Data Ready Interrupts* des Beschleunigungssensors. Die steigende Flanke des zu messenden Signals wird durch den ersten Befehl in der Interrupt-Routine des Mikrokontroller erzeugt. Die fallende Flanke wird durch den nächsten Befehl erzeugt, nachdem die Sensordaten erfolgreich abgetastet wurden.

Im zweiten Szenario werden die Sensordaten vom Gyroskop abgefragt. Das Verfahren ist ähnlich wie beim ersten Szenario nur, dass der Interrupt vom Gyroskop ausgelöst wird und die Daten von der entsprechenden Adresse gelesen werden.

Das dritte Szenario zeigt die Zeit zwischen zwei Timer-Interrupts des General Purpose Timer (GPT) in der SPU. Diese Messung ist wichtig, um zu sehen, wie groß der Einfluss des GPT-Jitter auf die folgenden Szenarien ist. Alle Testszenarien ab diesem Punkt verwenden den GPT als Interrupt-Quelle, um das Sampling der Sensordaten zu starten.

In den Szenarien vier und fünf werden die Sensordaten von allen drei im verwendeten intelligenten Sensor verfügbaren Sensoren abgetastet. Die steigende Flanke zur Messung wird durch den ersten Befehl in der Interrupt-Routine des GPT erzeugt. Die fallende Flanke wird erzeugt, nachdem die Daten von allen drei Sensoren erfolgreich ausgelesen wurden. In Szenario vier wird die Sensor-in-the-Loop-Schnittstelle verwendet, um die Daten aller Sensoren abzutasten. In diesem Fall handelt es sich bei den Sensordaten um voraufgezeichnete Daten, die durch den vorgestellten Sensor-in-the-Loop-Ansatz injiziert werden. Alle drei Sensoren werden sequenziell abgefragt. Es wird mit dem Beschleunigungssensor begonnen, gefolgt vom Gyroskop und dem Magnetometer. Szenario fünf ähnelt Szenario vier. In diesem Test wird die SPI-Schnittstelle verwendet, um die Daten von den einzelnen Sensorelementen zu erhalten.

Im sechsten und dem siebten Szenario wird nur die Abtastung des Beschleunigungssensors berücksichtigt. Wie in Szenario vier wird auch im sechsten Szenario die SiL-Schnittstelle verwendet, um die aufgezeichneten Sensordaten vom Beschleunigungssensor zu erhalten. Im letzten Szenario werden die Beschleunigungssensordaten über die SPI-Schnittstelle abgefragt.

Für alle vom Sensorinterrupt abhängigen Szenarien wurden die Sensorgeräte so konfiguriert, dass sie einen *Data Ready Interrupt* mit einer Frequenz von 1000 Hz erzeugen. Der General Purpose Timer des Mikrokontroller wurde außerdem so konfiguriert, dass er jeweils nach 1 ms einen Timer-Interrupt auslöst. Dies führt zu einer vergleichbaren Abtastfrequenz von 1000 Hz für alle untersuchten Szenarien.

5.4.5.2 Ergebnisse

Um ein besseres Verständnis des Timing-Verhaltens der vorgestellten Sensor-in-the-Loop-Implementierung zu erhalten, wurden die zuvor beschriebenen weiteren Untersu-

chungen durchgeführt. Diese Messungen konzentrieren sich auf den Jitter, also die Zeitdifferenz zwischen den einzelnen Sensorabtastungen. Alle zuvor beschriebenen Experimente wurde erfolgreich durchgeführt. Die Ergebnisse dieser Messungen werden nachfolgend vorgestellt. Die Beschreibung der verschiedenen Experimente oder auch Szenarien ist in Unterabschnitt 5.4.5.1 zu finden.

Tabelle 5.5 zeigt die Ergebnisse der Jitter-Messungen für den Cycle-to-Cycle-Jitter. Die Messungen wurden mit einer Firmware durchgeführt, die ohne Compiler-Optimierungen kompiliert wurde. In dieser Tabelle ist $Jcc[s]$ der maximale Wert in der Liste aller Jcc-Werte für das gemessene Signal. Der größte Jitter ist in den Testszenarien der Sensorabtastung zu sehen, welche den *Data Ready Interrupt* der Sensorelemente verwendet. Bei diesen Messungen ist der Jitter des Gyroskops mit $9,70 \cdot 10^{-5}\,s$ fast dreimal so groß wie der Jitter des Beschleunigungssensors. Diese Abweichung zwischen zwei aufeinanderfolgenden Abtastzeitpunkten kann bei dem vorgestellten Anwendungsfall mit einer Abtastrate von $1000\,Hz$ fast $10\,\%$ der Abtastperiode betragen. Der geringste Jitter ist bei den Szenarien mit der Sensor-in-the-Loop-Schnittstelle zu beobachten. Wird ausschließlich ein Sensor verwendet, dann beträgt der Jcc $2,80 \cdot 10^{-6}\,s$. Der Jitter verursacht durch die SPI-Schnittstelle beträgt $2,91 \cdot 10^{-6}\,s$ für einen Sensor und $3,24 \cdot 10^{-6}\,s$ für alle drei Sensoren. Verglichen mit dem Jitter der Sensorelemente fällt der Jitter der SiL-Schnittstelle gering aus.

Die Mittelwerte für den Jcc sind mit etwa zwei Größenordnungen wesentlich kleiner. Wenn von Zeit zu Zeit eine größere Differenz zwischen zwei benachbarten Abtastzeiträumen für einen Anwendungsfall tolerierbar ist, können auch die Mittelwerte für eine Bewertung berücksichtigt werden. Der Jpp beschreibt die Differenz zwischen der längsten und der kürzesten Periode im gesamten Signal. Damit zeigt er, ob es im Signal signifikante zeitliche Unterschiede über die gesamte Laufzeit der Messung existieren. In diesem Fall wäre der Jpp deutlich größer als der maximale Jcc-Wert. Bei den durchgeführten Messungen unterscheiden sich die Jpp-Werte nicht wesentlich von den Jcc-Werten. Daher ist davon auszugehen, dass der zeitliche Versatz der Sampling-Perioden sich nicht über die Laufzeit ändert.

Tabelle 5.5: Cycle-to-Cycle-Jitter ohne Compileroptimierung (-O0)

Name	Jcc[s]	Jcc on Signal[%]	Mittel Jcc[s]	Jpp	Jpp on signal [%]
int_acc_int_spi_sample	$3.31 \cdot 10^{-5}$	3.35	$3.43 \cdot 10^{-7}$	$3.33 \cdot 10^{-5}$	3.37
int_gyr_int_spi_sample	$9.70 \cdot 10^{-5}$	9.70	$3.77 \cdot 10^{-5}$	$9.71 \cdot 10^{-5}$	9.70
timer_int	$1.01 \cdot 10^{-6}$	0.10	$1.90 \cdot 10^{-7}$	$1.70 \cdot 10^{-6}$	0.17
timer_int_9dof_SiL_sample	$3.08 \cdot 10^{-5}$	3.04	$6.02 \cdot 10^{-7}$	$3.14 \cdot 10^{-5}$	3.10
timer_int_9dof_spi_sample	$3.24 \cdot 10^{-6}$	0.32	$3.57 \cdot 10^{-7}$	$3.70 \cdot 10^{-6}$	0.36
timer_int_acc_SiL_sample	$2.80 \cdot 10^{-6}$	0.28	$1.64 \cdot 10^{-7}$	$3.09 \cdot 10^{-6}$	0.30
timer_int_acc_spi_sample	$2.91 \cdot 10^{-6}$	0.29	$1.92 \cdot 10^{-7}$	$3.21 \cdot 10^{-6}$	0.32

Die Compiler-Optimierung des Firmware-Codes kann einen erheblichen Einfluss auf den

Jitter haben. Die Tabelle Tabelle 5.6 zeigt die Jcc- und Jpp-Werte für die mit (-O3) kompilierten Testfälle. Den größten Einfluss hat die Optimierung auf den Jitter für die Sensor-in-the-Loop-abhängigen Sampling-Szenarien und die Implementierung der Timer-Interrupt-Routine. Dies hat einen Einfluss auf alle Timer-abhängigen Abtastszenarien. Der geringste Einfluss zeigt sich bei den Szenarien eins und zwei. Der Jitter dieser beiden Abtastszenarien hängt hauptsächlich vom Jitter der Hardwaresensoren ab und wird somit durch die Compileroptimierung nicht beeinflusst.

Tabelle 5.6: Cycle-to-Cycle-Jitter mit Compileroptimierung (-O3)

Name	Jcc[s]	Jcc on signal [%]	Mittel Jcc[s]	Jpp[s]	Jpp on signal [%]
int_acc_int_spi_sample	$3.32 \cdot 10^{-5}$	3.36	$2.69 \cdot 10^{-7}$	$3.33 \cdot 10^{-5}$	3.37
int_gyr_int_spi_sample	$9.63 \cdot 10^{-5}$	9.63	$3.77 \cdot 10^{-5}$	$9.64 \cdot 10^{-5}$	9.64
timer_int	$4.01 \cdot 10^{-7}$	0.04	$7.11 \cdot 10^{-8}$	$1.04 \cdot 10^{-6}$	0.10
timer_int_9dof_SiL_sample	$1.89 \cdot 10^{-5}$	1.87	$3.44 \cdot 10^{-7}$	$2.00 \cdot 10^{-5}$	1.98
timer_int_9dof_spi_sample	$2.23 \cdot 10^{-6}$	0.22	$1.29 \cdot 10^{-7}$	$2.60 \cdot 10^{-6}$	0.26
timer_int_acc_SiL_sample	$2.20 \cdot 10^{-6}$	0.22	$1.34 \cdot 10^{-7}$	$2.56 \cdot 10^{-6}$	0.25
timer_int_acc_spi_sample	$2.35 \cdot 10^{-6}$	0.23	$1.31 \cdot 10^{-7}$	$2.64 \cdot 10^{-6}$	0.26

Eine dritte Art von Jitter, die aus den Messungen berechnet werden kann, ist der Periodic-Jitter. Die Definition und Berechnungsvorschrift für diese und alle anderen verwendeten Arten von Jitter finden sich zu Beginn des Unterabschnitt 5.4.4. Dieser Typ beschreibt die Abweichung der aktuellen Periode von der idealen Periode oder wie im beschriebenen Fall von der mittleren Periode. Bei den in diesem Artikel durchgeführten Experimenten ist der Jper kleiner als die Werte für den Jcc. In einigen Fällen, zum Beispiel bei den Szenarien eins und zwei, beträgt der Periodic-Jitter nur die Hälfte des Cycle-to-Cycle-Jitter. Dies lässt sich erklären, wenn die tatsächlichen Deltas in diesen Signalen genauer betrachtet werden. In Abbildung 5.13 sind die Periodenlängen der *Data Ready Interrupts* des Gyroskops über der Zeit aufgetragen. In diesem Signal folgt auf jede lange Periode eine kurze, dieses Verhalten führt zu einem größeren Wert für Jcc.

Tabelle 5.7: Periodic-Jitter ohne Compileroptimierung (-O0)

Name	Std	Jper[s]	Mittel Jper[s]	Jper on Signal [%]
int_acc_int_spi_sample	$5.42 \cdot 10^{-7}$	$1.68 \cdot 10^{-5}$	$2.24 \cdot 10^{-7}$	1.70
int_gyr_int_spi_sample	$2.97 \cdot 10^{-5}$	$4.86 \cdot 10^{-5}$	$1.89 \cdot 10^{-5}$	4.86
timer_int	$2.96 \cdot 10^{-7}$	$1.12 \cdot 10^{-6}$	$1.96 \cdot 10^{-7}$	0.11
timer_int_9dof_SiL_sample	$1.54 \cdot 10^{-6}$	$1.59 \cdot 10^{-5}$	$3.70 \cdot 10^{-7}$	1.57
timer_int_9dof_spi_sample	$3.90 \cdot 10^{-7}$	$1.90 \cdot 10^{-6}$	$2.40 \cdot 10^{-7}$	0.19
timer_int_acc_SiL_sample	$2.05 \cdot 10^{-7}$	$1.60 \cdot 10^{-6}$	$1.44 \cdot 10^{-7}$	0.16
timer_int_acc_spi_sample	$2.46 \cdot 10^{-7}$	$1.64 \cdot 10^{-6}$	$1.57 \cdot 10^{-7}$	0.16

Die Tabellen Tabelle 5.7 und Tabelle 5.8 enthalten auch die Standardabweichung der Periodenlänge. Es zeigt sich, dass dieser Wert in einigen Fällen mit dem Jitter korreliert.

Bei genauerer Betrachtung zeigt sich jedoch, dass dies nicht in allen Fällen zutrifft. Für Szenario eins beispielsweise ist die Standardabweichung mit $5,54 \cdot 10^{-7}\,s$ zwei Größenordnungen kleiner als für Szenario zwei mit $2,97 \cdot 10^{-5}\,s$. Die Periodic-Jitters liegen mit $1,68 \cdot 10^{-5}\,s$ und $4,82 \cdot 10^{-5}\,s$ in der gleichen Größenordnung. Die Standardabweichung zeigt also einen Trend in der Periodenabweichung, reicht aber nicht aus um die zeitliche Qualität des Signals zu beurteilen.

Tabelle 5.8: Periodic-Jitter mit Compileroptimierung (-O3)

Name	Std	Jper[s]	Mittel Jper[s]	Jper on Signal [%]
int_acc_int_spi_sample	$4.89 \cdot 10^{-7}$	$1.68 \cdot 10^{-5}$	$1.88 \cdot 10^{-7}$	1.70
int_gyr_int_spi_sample	$2.97 \cdot 10^{-5}$	$4.82 \cdot 10^{-5}$	$1.89 \cdot 10^{-5}$	4.82
timer_int	$1.40 \cdot 10^{-7}$	$5.60 \cdot 10^{-7}$	$1.13 \cdot 10^{-7}$	0.06
timer_int_9dof_SiL_sample	$8.98 \cdot 10^{-7}$	$1.03 \cdot 10^{-5}$	$2.47 \cdot 10^{-7}$	1.01
timer_int_9dof_spi_sample	$1.96 \cdot 10^{-7}$	$1.34 \cdot 10^{-6}$	$1.35 \cdot 10^{-7}$	0.13
timer_int_acc_SiL_sample	$1.75 \cdot 10^{-7}$	$1.35 \cdot 10^{-6}$	$1.31 \cdot 10^{-7}$	0.13
timer_int_acc_spi_sample	$1.75 \cdot 10^{-7}$	$1.38 \cdot 10^{-6}$	$1.31 \cdot 10^{-7}$	0.14

Der Einfluss der Compiler-Optimierung auf die Jper-Jitter-Werte ist ähnlich wie bei den Cycle-to-Cycle-Jitter-Werten. Auch hier hat die Optimierung den größten Einfluss auf die Szenarien, welche abhängig vom Timer-Interrupt sind.

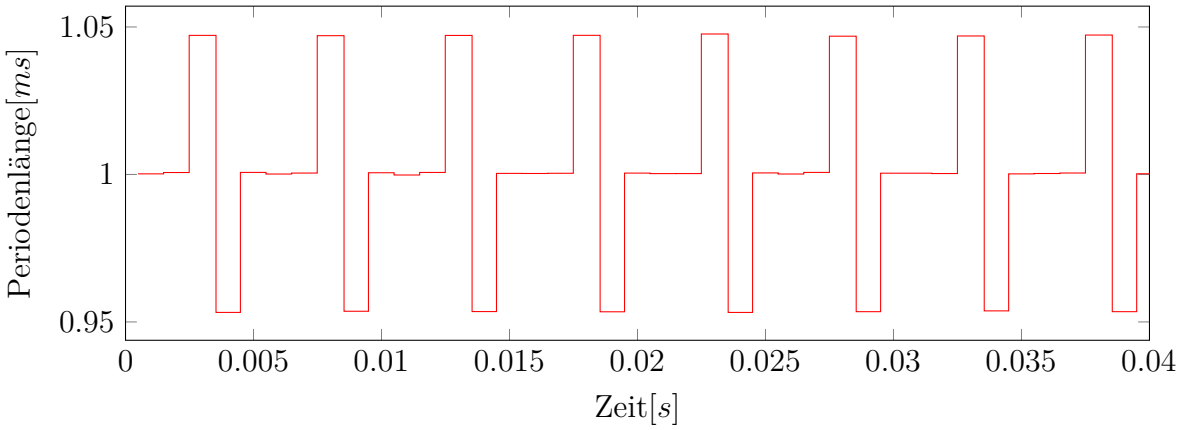


Abbildung 5.13: Periodenlängen für die Bereitstellung von Gyroskop Sensordaten

In Abbildung 5.13 werden die einzelnen Periodenlängen im Zeitverlauf dargestellt. Das abgebildete Verhalten erklärt, warum die Jcc-Werte doppelt so groß sind wie die Jper-Werte. Hierbei folgt auf eine sehr kurze Periode immer eine sehr lange Periode. Das Diagramm zeigt auch, dass der Jitter des Gyroskop-Interrupts als deterministischer periodischer Jitter klassifiziert werden kann.

In Abbildung 5.14 ist der Periodic-Jitter für alle sieben gemessenen Szenarien aufgetragen. Die Testszenarien eins und zwei, die den Jitter der tatsächlichen Hardwaresensoren

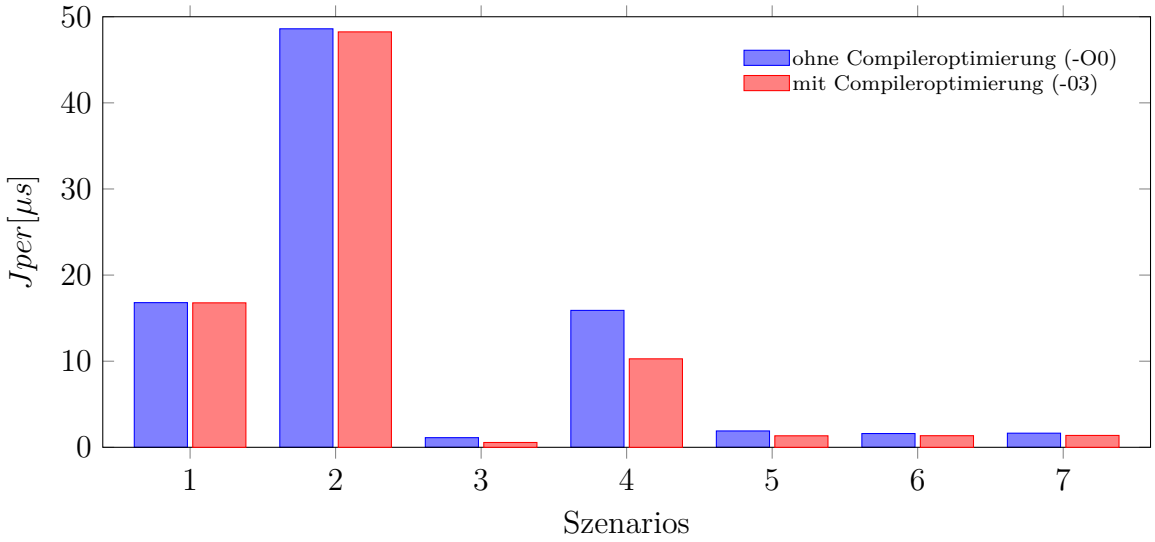


Abbildung 5.14: Periodic-Jitter mit und ohne Optimierung der Sensor-Firmware (Szenario IDs wie in Tabelle 5.4)

beinhalten, zeigen einen deutlich größeren Jitter als alle anderen Szenarien. Der Jitter in den Szenarien, in denen die Sensordaten über die Sensor-in-the-Loop-Schnittstelle abgetastet werden, ist vergleichbar mit dem Lesen über die SPI-Schnittstelle. Außer in Szenario vier beim Lesen von allen drei Sensoren über das SiL-Interface hier ist der Jitter größer als beim Auslesen über SPI. Allerdings ist der Jitter immer noch geringer als der durch den Beschleunigungssensor oder das Gyroskop verursachte Jitter. Wenn man die Balken für optimierte und nicht optimierte Sensor-Firmware nebeneinander aufträgt, zeigt sich, dass der Einfluss der Code-Optimierung nicht signifikant ist. Dies gilt besonders, wenn die Abtastung von einem Hardware-Sensor abhängt.

Da alle untersuchten Jitter für die Szenarien mit dem SiL-Ansatz geringer sind, hat die Verwendung von wiedergegebenen Sensordaten wenig bis keinen Einfluss auf die getestete Firmware oder die implementierten Algorithmen. Alle gemessenen Jitter sind kleiner als fünf Prozent der Abtastfrequenz, d.h. ein Einfluss des Jitters auf Algorithmen, zum Beispiel in Regelungsanwendungen, ist vernachlässigbar [A66].

5.4.6 Statistische Signifikanzanalyse

Nachdem die Ergebnisse der Untersuchung zu Ausführungszeit und Jitter ausgewertet wurden, wird in diesem Abschnitt die statistische Bedeutung dieser Ergebnisse untersucht. Um den Einfluss der verschiedenen Parameter wie unterschiedliche Sensoren oder Wiederholungen auf die Ergebnisse zu analysieren, wurde eine statistische Analyse der

Ergebnisse durchgeführt. Diese Analyse ist mittels zweier nachfolgen vorgestellten Methoden erfolgt.

5.4.6.1 Methoden statistischer Analyse

Die Analyse der statistischen Signifikanz der Ergebnisse wurde mithilfe von zwei Methoden durchgeführt, die im Folgenden beschrieben werden.

Gage Repeatability and Reproducibility (Gage R&R)

Die erste verwendete Methode ist die Messsystemanalyse oder auch Gage Repeatability and Reproducibility (Gage R&R)-Analyse [A67]. Üblicherweise wird sie für diese Analyse zur Bewertung der Wiederholbarkeit und Reproduzierbarkeit eines Messsystems verwendet [A68], [A69]. Diese Methode wendet eine Varianzanalyse an, um den Einfluss von drei unabhängigen Faktoren auf die Gesamtergebnisse der Messungen zu schätzen [A70]. Die traditionelle Gage R&R-Analyse wird mit den drei Faktoren *Testgerät*, *Testperson* und *Testwiederholungen* [A71] durchgeführt. Im Falle der vorliegenden Arbeit ermittelt die Analyse den Einfluss des *Testgerätes* oder auch Sensors, des *Szenarios* oder auch Methode und der *Wiederholung* der Messung auf die abhängige Variable. Die abhängige Variable ist in den späteren Untersuchungen der *Jitter*.

Für die Berechnung werden die Varianz, der Ergebnisbereich und der Mittelwert gruppiert nach jeder der unabhängigen Faktoren berechnet. Die Summe der Fehlerquadrate wird innerhalb jeder Gruppe im Vergleich zum Gruppenmittelwert und zwischen allen Gruppen im Vergleich zum Mittelwert aller Ergebnisse berechnet. Diese Werte werden dann ins Verhältnis gesetzt, um den Einfluss der einzelnen Gruppen auf den Mittelwert und die Varianz des Gesamtergebnisses zu bestimmen. Die Gleichungen und Berechnungsregeln finden sich beispielhaft in [A72].

Konfidenzintervall

Die zweite Messmethode ist das Konfidenzintervall [A73]. Das Konfidenzintervall beschreibt einen Bereich in dem der wahre Wert eines unbekannten Parameters aus einer Reihe von Messwerten mit einer bestimmten Wahrscheinlichkeit liegt. Die Wahrscheinlichkeit wird vorher festgelegt. Ein häufig gewählter Wahrscheinlichkeitswert für das Intervall ist 95%. Die Berechnung des Konfidenzintervalls ist in Gleichung (5.8) dargestellt.

$$CI = \overline{X} \pm Z^* \cdot \frac{\sigma}{\sqrt{n}} \tag{5.8}$$

In der Formel ist $\overline{X}$ der Mittelwert der Messwerte, σ ist die Standardabweichung der Messwerte und n ist die Anzahl der Stichproben. Der Wert für Z^* kann für eine gegebene Anzahl von Stichproben und die erforderliche Wahrscheinlichkeit einer Tabelle entnommen werden. Es ist wichtig zu beachten, dass bei einem Stichprobenumfang von mehr als 100 die Tabelle für eine Standardnormalverteilung (oder Z-Verteilung) verwendet werden kann. Bei einem Stichprobenumfang von weniger als 100 ist eine Tabelle mit der studentischen t-Verteilung erforderlich [A74]. Das ± zeigt an, dass die gleiche Gleichung zur Berechnung der oberen und unteren Grenzen des Konfidenzintervalls verwendet wird. Um zu zeigen, dass mehrere Gruppen von Messungen (z. B. im Fall der vorliegenden Arbeit nach Szenario) unterschieden werden können, sollte das Konfidenzintervall für jede Gruppe im Vergleich zum Wertebereich eng sein. Bei statistisch abhängigen Variablen dürfen sich die Intervalle nicht überschneiden.

5.4.6.2 Ergebnisse der Signifikanzanlysen

In diesem Abschnitt werden die Ergebnisse der beiden statistischen Analysen für das Cycle-to-Cycle-Jitter vorgestellt.

Die Gage R&R-Analyse wurde für beide Testfälle getrennt durchgeführt. Eine Analyse für das ohne Optimierung kompilierte Firmware-Image und eine separate Analyse für die vom Compiler optimierte Version. Um ein statistisch signifikantes Ergebnis zu erhalten und dies durch einen solchen Signifikanztest nachzuweisen, wurde eine gemeinsame Menge an Testdaten generiert. Zu diesem Zweck wurden die Tests mit drei verschiedenen Sensoren durchgeführt. Auf diesen drei Sensoren wurden die sieben verschiedenen Szenarien (Methoden) in der optimierten und nicht optimierten Version getestet. Jeder dieser Tests wurde dreimal durchgeführt. Die Datenrate für diese Testfälle betrug $1000\,Hz$ und jeder Durchlauf wurde für $10\,s$ durchgeführt. Dies führte zu einer Gesamtmenge von 1.260.000 Datenpunkten für beide Gage R&R-Analysen.

Tabelle 5.9: Gage R&R Signifikanzanalyse für Jcc auf unoptimierter Firmware (-O0)

	Sensor	Methode	Wiederholung	TOTAL Variation
Varianz (VarComp)	$1.01 \cdot 10^{-14}$	$1.22 \cdot 10^{-9}$	$7.04 \cdot 10^{-14}$	$1.22 \cdot 10^{-9}$
Beitrag [%]	0.001	99.980	0.010	100.000
Standard Abweichung	$1.00 \cdot 10^{-7}$	$3.49 \cdot 10^{-5}$	$2.65 \cdot 10^{-7}$	$3.49 \cdot 10^{-5}$
Studienvariation [%]	0.290	99.990	0.760	100.000
P-Wert	0.302	0.000		

Die Ergebnisse der statistischen Analyse für die Testfälle ohne Compileroptimierung sind in Tabelle 5.9 aufgelistet.

Die Zeile *Beitrag [%]* zeigt, dass die Methode, d. h. das verwendete Szenario aus Tabelle 5.4 mit $1,22{\cdot}10^{-9}$, bei Weitem den größten Einfluss auf die Gesamtvariation hat. Wobei die beiden anderen Parameter nur geringfügig zum Gesamtergebnis beitragen. Der Einfluss des Testobjektes oder der verschiedenen verwendeten Sensoren ist mit $1,01 \cdot 10^{-14}$ sehr gering. Auch die Wiederholungen des Versuchs haben $7,04 \cdot 10^{-14}$ nahezu keinen Einfluss auf die Gesamtvariation von $1,22 \cdot 10^{-9}$. Bei dem Messanalyseverfahren wurde auch der P-Wert für die beiden unabhängigen Faktoren Sensor und Methode bestimmt. Einem P-Wert kleiner 0,05 bedeutet einen signifikanten Einfluss des entsprechenden Parameters. Der Sensor hat mit 0,302 keinen signifikanten Einfluss auf das Gesamtergebnis. Hingegen ist der Einfluss der Methode mit einem P-Wert von 0,000 für das Ergebnis der Untersuchung signifikant.

Tabelle 5.10: Gage R&R Signifikanzanalyse für Jcc auf optimierter Firmware (-O3)

	Sensor	Methode	Wiederholung	TOTAL Variation
Varianz (VarComp)	$0.00 \cdot 10^{0}$	$1.22 \cdot 10^{-9}$	$7.68 \cdot 10^{-14}$	$1.22 \cdot 10^{-9}$
Beitrag [%]	0.000	99.990	0.010	100.000
Standard Abweichung	$0.00 \cdot 10^{0}$	$3.49 \cdot 10^{-5}$	$2.77 \cdot 10^{-7}$	$3.49 \cdot 10^{-5}$
Studienvariation [%]	0.000	99.990	0.790	100.000
P-Wert	0.613	0.000		

Die gleichen Experimente wurden auch mit einer compileroptimierten Firmware-Version durchgeführt. In Tabelle 5.10 sind die Ergebnisse für die optimierte Firmware dargestellt. Die statistische Analyse zeigt ähnliche Ergebnisse wie bei der nicht optimierten Version. Auch in diesem Fall hat die Methode einen Einfluss von über 99 Prozent auf das Gesamtergebnis. Der P-Wert von 0,000 bestätigt diese Aussage. Die Werte für den Sensor und die Wiederholungen liegen nahe null. Auch hier liegt der P-Wert für den Sensor mit 0,613 weit über 0,05, somit ist der Einfluss des Sensors auch in diesen Untersuchungen nicht signifikant.

Beide Messsystemanalysen zeigten im Speziellen, dass die Ergebnisse für die Cycle-to-Cycle-Jitter-Werte statistisch signifikant sind. Die Ergebnisse der ausgeführten Analyse zeigen die statistische Signifikanz der gesamten Experimente zum Zeitverhalten der SiL-Implementierung. Weiterhin kann aufgrund des geringen Einflusses der Wiederholbarkeit auf das Gesamtergebnis darauf geschlossen werden, dass das Timing und die ermittelten Jitter nicht datenabhängig sind.

Um einen zweiten Anhaltspunkt für die statistische Signifikanz der Timing- und Jitter-Ergebnisse zu erhalten, wurden die Konfidenzintervalle für die Jcc-Ergebnisse für alle gemessenen Szenarien berechnet. Diese Konfidenzintervalle sind in Abbildung 5.15 dargestellt. Für alle Szenarien sind die Konfidenzintervalle im Vergleich zum berechneten

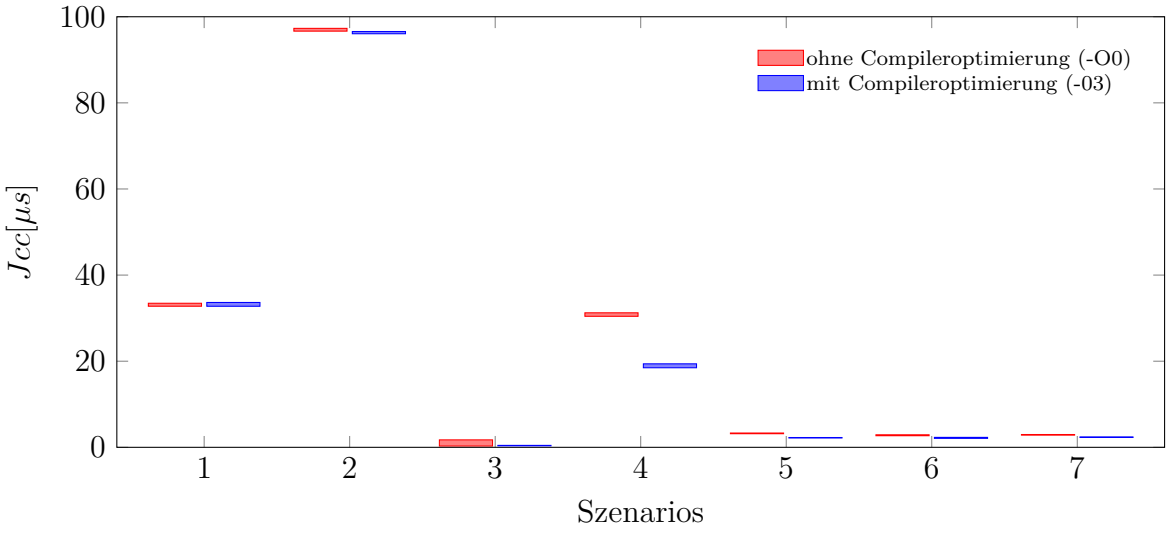


Abbildung 5.15: Konfidenzintervalle für Jcc mit und ohne Compileroptimierung (Szenario IDs wie in Tabelle 5.4)

Mittelwert der Jcc-Werte relativ klein. Außerdem gibt es keine Überschneidungen der einzelnen Konfidenzintervalle der einzelnen Szenarien.

Tabelle 5.11 zeigt die in der Abbildung 5.15 dargestellten Werte für eine genauere Auswertung in tabellarischer Form. Dabei werden neben den Werten der einzelnen Konfidenzintervalle auch die Mittelwerte der jeweiligen Jcc-Messung aufgelistet.

Tabelle 5.11: Konfidenzintervalle und Mittelwerte für die Jcc Messungen

Szenario ID	Mittel Jcc -O0 [s]	CI -O0 [s]	Mittel Jcc -O3 [s]	CI -O3 [s]
1	$3.311 \cdot 10^{-5}$	$3.468 \cdot 10^{-7}$	$3.320 \cdot 10^{-5}$	$4.416 \cdot 10^{-7}$
2	$9.698 \cdot 10^{-5}$	$3.233 \cdot 10^{-7}$	$9.631 \cdot 10^{-5}$	$2.591 \cdot 10^{-7}$
3	$1.014 \cdot 10^{-6}$	$7.158 \cdot 10^{-7}$	$4.009 \cdot 10^{-7}$	$4.793 \cdot 10^{-8}$
4	$3.083 \cdot 10^{-5}$	$4.067 \cdot 10^{-7}$	$1.893 \cdot 10^{-5}$	$4.564 \cdot 10^{-7}$
5	$3.240 \cdot 10^{-6}$	$8.053 \cdot 10^{-8}$	$2.227 \cdot 10^{-6}$	$4.907 \cdot 10^{-8}$
6	$2.801 \cdot 10^{-6}$	$1.083 \cdot 10^{-7}$	$2.197 \cdot 10^{-6}$	$1.134 \cdot 10^{-7}$
7	$2.907 \cdot 10^{-6}$	$6.567 \cdot 10^{-8}$	$2.348 \cdot 10^{-6}$	$4.241 \cdot 10^{-8}$

Die Analysen der Konfidenzintervalle untermauern die Ergebnisse der Gage R&R-Analysen. Auch aus ihnen lässt sich schließen, dass die Timing- und Jitter-Experimente statistisch verwertbare Ergebnisse liefern. Die Ergebnisse für die in diesem Abschnitt verwendeten Jcc-Werte, lässt sich auch auf alle anderen Jitter-Werte und auch auf die Ausführungszeiten übertragen. Die Übertragung der Ergebnisse ist möglich, da diese anderen Ergebnisse die gleiche Datenbasis haben und statistisch voneinander abhängen.

5.5 Zusammenfassung und Bewertung der Ergebnisse

In diesem Kapitel wurde die entwickelte SiL-Architektur vorgestellt, es stellt somit gemeinsam mit dem folgenden Kapitel den Kern dieser Abhandlung dar. Teile dieses Kapitels wurden im Vorfeld bereits in den peer-reviewten Publikationen [B5], [B6] veröffentlicht. Weiterhin wurde das hier vorgestellte Konzept in Deutschland und den USA zum Patent angemeldet [B7], [B8]. Das vorgestellte SiL-Konzept unterstützt Entwickler von Firmware für intelligente Sensoren bei der Entwicklung und Verifikation dieser Firmware. Dabei ermöglicht es das wiederholbare und reproduzierbare Testen der Firmware auf echter Sensorhardware.

Zu Beginn dieses Kapitels wurden die Voraussetzungen und die konzeptionellen Grundlagen des Ansatzes erläutert. Auf Grundlage dieses Konzeptes wurde eine Beispielimplementierung auf einem State-of-the-Art intelligenten Sensor durchgeführt. Diese Implementierung umfasst die Instrumentalisierung der Firmware auf dem BMF055 [A18]. Diese dient dazu, die Sensordaten nicht nur von den eigentlichen Sensorelementen, sondern auch von der erstellten SiL-Schnittstelle auszulesen. Weiterhin wurde eine Möglichkeit geschaffen, die Sensordaten für spätere reproduzierbare Test aufzuzeichnen. Die Implementierung umfasst zusätzlich die Erstellung einer Erweiterung in der eigentlichen Entwicklungssoftware des intelligenten Sensors auf dem PC. Diese Erweiterung dient der Visualisierung und der Steuerung aller Komponenten und Abläufe des SiL-Prozesses.

Um die Möglichkeiten des Ansatzes im allgemeinen und der Implementierung im Speziellen zu untersuchen, wurden verschieden Experimente durchgeführt. Diese konzentrierten sich vor allem auf das Zeitverhalten der implementierten SiL-Funktionalität. Zunächst wurde die Gesamtleistung der Implementierung über die Ausführungszeit untersucht. Dazu sind verschiedene Sensorkonfigurationen aufgezeichnet worden, um sie später in Echtzeit mit wiederholbaren Ergebnissen zu injizieren. Alle Tests wurden mit maximal drei Sensoren durchgeführt, wobei jeder der Sensoren aus drei Sensorachsen besteht. Die Ergebnisse zeigten, dass es möglich ist, die Sensordaten mit einer Abtastrate von $1600\,Hz$ zu injizieren und mit dem vorgeschlagenen Ansatz wiederholbare und reproduzierbare Ergebnisse in Echtzeit zu erhalten. Die zweiten Experimente konzentrierten sich auf den Jitter im Abtastprozess. Es wurden verschiedene Arten von Jitter bei der Abtastung der tatsächlichen Hardwaresensoren und bei der Verwendung der Sensor-in-the-Loop-Architektur gemessen und verglichen. Darüber ist der Jitter für verschiedene Hardware-Konfigurationen gemessen worden, z. B. Timer-basiertes Sampling oder Interrupt-basiertes Sampling. Die Ergebnisse zeigen, dass der Jitter bei der Abtastung tatsächlicher Hardwaresensoren und der Abtastung mit unserem SiL-Ansatz unterschiedlich ist. Das bedeutet, dass die Implementierung das Timing-Verhalten nicht perfekt reproduzieren kann. In Anbetracht dessen hat der Jitter für die SiL-Implementierung die gleiche Größenordnung, bleibt aber kleiner als der Jitter für die tatsächlichen Sensorgeräte. Der Jitter, welcher durch die eigentlichen Sensorelemente verursacht wird, ist

allerdings wesentlich größer. Daher ist der Schluss zulässig, dass diese Implementierung das Timing-Verhalten gut genug reproduziert, um sie für die meisten Anwendungsfälle zu verwenden. Laut [A66] liegen die gemessenen Jitter in einem vernachlässigbaren Bereich. Es ist also davon auszugehen, dass der hier gemessene Jitter keinen Einfluss auf die implementierten Algorithmen hat, unabhängig davon, ob reale oder künstliche Sensordaten verwendet werden. Durch den geringeren Jitter in den durch das SiL-System übertragenen Daten wäre es evtl. möglich, den Jitter der realen Sensoren nachzubilden. Diese Nachbildung des Jitters könnte über verschieden Timer Implementierungen realisiert werden.

Um die statistische Signifikanz der Experimente nachzuweisen sind zwei verschiedene statistische Methoden verwendet worden. Die zuerst verwendete Gage R&R-Analyse zeigt den Einfluss der verschiedenen Parameter auf die Ergebnisse. Darüber konnte gezeigt werden, dass die Ergebnisse datenunabhängig sind und nicht durch die Verwendung verschiedener Sensoren der gleichen Art beeinflusst werden. Darüber hinaus beträgt der Einfluss des tatsächlich gemessene Szenario auf das Ergebnis mehr als 99 Prozent. Die geringe Varianz in den Analyseergebnissen zeigt die statistische Signifikanz der Testdaten. Um die Gage R&R-Analyse zu untermauern wurde auch das Konfidenzintervall der Testdaten berechnet. Die geringe Weite des Konfidenzintervalls im Vergleich zum berechneten Mittelwert spricht ebenfalls für statistisch signifikante Daten. Alle statistischen Tests wurden exemplarisch für die Cycle-to-Cycle-Jitter-Daten durchgeführt da alle anderen Jitter- und Timing-Berechnungen statistisch miteinander verbunden sind.

Die durchgeführten Experimente zeigen, dass sich die erstellte Implementierung der SiL-Architektur eignet, um Funktionalität und Tauglichkeit des vorgestellten Ansatzes zu demonstrieren. Somit kann die erhaltene Implementierung in echten Szenarien der Entwicklung von Firmware und Algorithmen für intelligente Sensoren eingesetzt werden. Durch verschiedene Erweiterungen könnte der Mehrwert des Ansatzes vor allem im Bereich der extrafunktionalen Eigenschaften noch weiter gesteigert werden.

6 Anwendungen und Erweiterungen des SiL-Konzeptes

Im vorhergehenden Kapitel wurde das Konzept und eine beispielhafte Implementierung der Sensor-in-the-Loop (SiL)-Architektur vorgestellt und erläutert. Weiterhin wurde durch Experimente gezeigt, dass sich das Konzept und die erstellte Implementierung grundsätzlich für die reproduzierbare Validierung von Sensor-Firmware auf echter Sensorhardware eignet. Für die Experimente wurden Beispielimplementierungen von Firmware für intelligente Sensoren erstellt, die geeignet sind die gewünschten Parameter des Systems zuverlässig und transparent zu testen. Diese Testfirmware-Implementierungen spiegeln allerdings keinen echten Anwendungsfall für das erstellte System wider.

Diese Kapitel dient als Erweiterung für das Kapitel 5. Es soll die Anwendung des erstellten SiL-Konzeptes im Rahmen eines realen Untersuchungsszenarios von Algorithmen und Firmware für intelligente Sensoren zeigen. Weiterhin beschreibt es Erweiterungen des Systems, um den Nutzen und die Praktikabilität innerhalb der Entwicklung von Firmware für intelligente Sensoren weiter zu erhöhen.

Das Kapitel teilt sich in drei Abschnitte welche die Anwendung des SiL-Ansatzes oder eine entsprechende Erweiterung beschreiben. Der erste Abschnitt beschreibt die Anwendung des SiL-Konzeptes für die Untersuchung verschiedener Algorithmen zu Schätzung der Orientierung. Diese Algorithmen basieren auf den Daten der Inertialsensoren. Weiterhin werden die untersuchten Algorithmen direkt auf einem intelligenten Sensor ausgeführt. Dabei wird die in Kapitel 5 vorgestellte Implementierung verwendet, um die funktionalen und extrafunktionalen Eigenschaften der Algorithmen zu untersuchen und miteinander zu vergleichen.

Der zweite Abschnitt stellt eine Erweiterung des SiL-Frameworks vor. Bisher ist es möglich Sensordaten aufzunehmen und diese für weitere Testläufe reproduzierbar in den zu testenden Sensor einzuspeisen. Die entwickelte Erweiterung ermöglicht es, die aufgezeichneten Sensordaten vor der Wiederverwendung auf komfortable Art zu manipulieren. Dabei ist etwa ein Überlagern der Daten mit Rauschen oder Sensorfehlern möglich. Auch die Erstellung künstlich erzeugter Signale für die Injektion in den Sensor ist über die Verwendung der Erweiterung realisierbar.

Speziell mit der extrafunktionalen Eigenschaft des Energieverbrauches beschäftigt sich der dritte Abschnitt dieses Kapitels. Er beschreibt die Erstellung eines online Energiemodels für intelligente Sensoren. Dieses Model liefert während der Laufzeit der Sensor errechnete Werte für den aktuellen Energieverbrauch. In Kombination mit dem SiL-Ansatz können diese Werte zusammen mit den Sensordaten visualisiert werden. Diese Kombination aus Sensordaten, Ergebnissen von Sensoralgorithmen und Energieverbrauch ermöglichen eine energiebewusste Entwicklung der Firmware für intelligente Sensoren.

6.1 Validierung und Klassifizierung von Sensorfusionsalgorithmen

In den vorhergehenden Kapiteln wurde das SiL-Konzept vorgestellt. Weiterhin ist eine Beispielimplementierung dieses Konzeptes auf ihre Eigenschaften hin untersucht worden. Diese Untersuchung war theoretischer Natur und betrachtete quantitative Eigenschaften der konzeptionellen Implementierung.

Nachfolgend wird die erstellte SiL-Implementierung eingesetzt, um Fusionsalgorithmen basierend auf Daten von Inertialsensoren zu untersuchen und miteinander zu vergleichen. Dabei werden zum einen die funktionalen Eigenschaften der Algorithmen betrachtet. Aufgrund der Verwendung des SiL-Konzeptes können zum anderen auch besonders komfortabel extrafunktionale Eigenschaften wie Laufzeit und Speicherauslastung untersucht werden.

6.1.1 Einleitung

Die bisher in der Arbeit vorgestellten und verwenden mikro-elektro-mechanisches System (Micro-Electro-Mechanical System, MEMS) basierende Inertialsensoren sind zum De-facto-Standard für Inertiale Messeinheits (Inertial Measurement Units, IMUs) in der Unterhaltungselektronik [A75] geworden. Darüber hinaus sind sie aufgrund ihrer Leistungsfähigkeit und Energieeffizienz auch prädestiniert für Gesten- und Aktivitätserkennung. Weiterhin werden sie in der Gesundheitsüberwachung im Sport [A76], in intelligente Kleidung oder ferngesteuerte Geräte, welche oft durch Energy Harvesting betrieben werden, eingesetzt.

Für die genannten Anwendungen werden häufig intelligente Sensoren eingesetzt, die neben den Trägheitssensoren auch einen Mikrocontroller und Schnittstellen zur Vorverarbeitung, Zusammenführung und Analyse der erfassten Daten enthalten [A77]. Die Implementierung des vorgestellten SiL-Ansatzes beruht wie bereits beschrieben auf dieser Art von Sensoren. Die Integration aller Funktionen in ein sogenannten System in Package (SiP) bringt viele Vorteile mit sich. Die Vorverarbeitung und die Sensorfusion

können direkt auf dem intelligenten Sensor durchgeführt werden, was zu einem geringeren Kommunikationsaufwand führt. Darüber hinaus ist je nach Anwendung eine Senkung des Stromverbrauchs möglich.

Nachfolgend wird die Eignung von vier weit verbreiteten Sensorfusionsalgorithmen zur Bestimmung der Orientierung eines am Körper getragenen Geräts in einer hardwarebeschränkten Umgebung untersucht. Unabhängig von der Systemarchitektur sind die verwendeten Recheneinheiten jedoch häufig hinsichtlich ihrer Rechenleistung und Speichergröße begrenzt. So benötigt beispielsweise ein Cortex M0+ bis zu 5,9 mA [A54], während ein leistungsfähigerer Cortex M3 bis zu 27 mA benötigt [A78]. Besonders komplexe Sensorfusionsalgorithmen mit einer hohen Ausgangsdatenrate (Output Data Rate, ODR) können leicht die Kapazitäten der kleineren Recheneinheiten erreichen oder überschreiten. Die Algorithmen auf einem typischen intelligenten Sensor implementiert und untersucht. Für die Entwicklung der Algorithmen auf dem Sensor und vor allem für die Untersuchung der extrafunktionalen Eigenschaften wurde das vorgestellte SiL-Konzept benutzt. Die Inhalte dieses Unterkapitels wurden zuerst im *Sensors* Journal veröffentlicht [B9][1].

6.1.2 Verwandte Arbeiten

Um die Ausrichtung eines Geräts zu bestimmen wurden vier Algorithmen zur Sensorfusion verwendet. Diese Arten von Methoden werden häufig als Algorithmen für Lage- und Kursreferenzsysteme oder auch als Attitude Heading Reference System (AHRS) bezeichnet. Diese werden häufig in Verbindung mit Luftfahrzeugen verwendet [A79], [A80]. Die Bewertung wurde über ein erweitertes Kalman-Filter, ein Madgwick-Filter [A81], [A82], ein Mahony-Filter [A83] und einem komplementären Filter [A84] durchgeführt.

In der Vergangenheit hat es bereits mehrere Studien gegeben, die die berechneten Fusionsergebnisse dieser Algorithmen quantitativ verglichen haben. Zum Beispiel in [A85] kommt ein Vergleich zwischen Kalman-Filter, Mahony-Filter und Madgwick-Filter zu ähnlichen Ergebnis wie nachfolgend vorgestellt. Die Auswertungen werden dabei sowohl in einer Simulation als auch mit real aufgezeichneten Daten durchgeführt.

Weitere Arbeiten, die die Brauchbarkeit der Sensorfusion auf der Grundlage des Komplementärfilters im Vergleich zu einem Kalman-Filter bestätigen, sind in [A86] und [A87] veröffentlicht. Beide Arbeiten vergleichen einen Kalman-Filter und einen komplementären Filter und kommen zu dem Schluss, dass beide Arten von Filtern brauchbar und vergleichbar sind. In [A86] wird festgestellt, dass das komplementäre Filter einfacher zu kalibrieren ist, da er weniger Parameter hat. Beide Arbeiten führen einen direkten

[1] Diese Veröffentlichung entstand in enger Kooperation mit Herrn Nils Büscher. Eine Abgrenzung der einzelnen Beiträge befindet sich in Abschnitt 1.2. Aus Gründen der Lesbarkeit wird an dieser Stelle der komplette Ansatz vorgestellt.

Vergleich von zwei oder mehr Sensorfusionsalgorithmen für die Orientierung durch und kommen übereinstimmend zu dem Ergebnis, dass die komplementären filterbasierten Ansätze den Kalman-Filtern nicht unterlegen sind.

Im Gegensatz zu den vorgenannten Arbeiten kommt [A88] zu dem Schluss, dass das Kalman-Filter im Vergleich zum Madgwick- oder Mahony-Filter überlegen ist. Der Vergleich in [A88] wird für mehrere Bewegungsgeschwindigkeiten unter Verwendung eines Roboterarms als Referenz durchgeführt. Obwohl man zu dem Schluss kommt, dass das Kalman-Filter besser ist, sind die Unterschiede bei den Messungen recht gering. Der Fehler des Madgwick-Filters und des Mahony-Filters ist im Durchschnitt nur um 0,4 Grad höher als der Fehler des Kalman-Filters, was einen durchschnittlichen Gesamtfehler von 3,4 Grad ergibt. Die Unterschiede sind höchstwahrscheinlich auf die Verwendung eines anderen Messsystems mit unterschiedlichen Annahmen über die durchgeführten Bewegungsgeschwindigkeiten und Szenarien zurückzuführen. Darüber hinaus lässt die Verwendung eines Roboterarms mit sehr abrupten aber linearen Bewegungen keine direkten Rückschlüsse auf das Filterverhalten zu, wenn die Bewegung von einem Menschen ausgeführt wird. Durch die Verwendung des SiL-Ansatzes können vom Menschen ausgeführte Bewegungen benutzt werden, um die Algorithmen reproduzierbar zu testen. Daher sind die Bewegungen fließender und vielfältiger als bei einem Roboter.

In den oben genannten Arbeiten werden keine unterschiedlichen Datenformate untersucht. Außerdem führen sie keine statistische Analyse der Filterqualität durch, was bei der Untersuchung natürlicher Bewegungen von besonderer Bedeutung ist. Schließlich analysieren sie auch nicht direkt extrafunktionale Eigenschaften wie den Rechenaufwand und die Codegröße.

Andere Arbeiten untersuchen die Verwendbarkeit von Ansätzen auf der Grundlage von Komplementärfiltern für am Körper getragene sensorische Anwendungen und Anwendungen im Gesundheitswesen. Dies unterstreicht den Bedarf an Algorithmen mit geringen Hardware-Anforderungen und die Nutzbarkeit komplementärer filterbasierter Ansätze. In [A89] wird ein neuartiger Ansatz für ein komplementäres Filter vorgestellt, der speziell auf die Verwendung in am Körper getragenen Sensoren mit begrenzten Hardwarekapazitäten zugeschnitten ist. Eine Evaluierung wurde mit einem Bewegungserfassungssystem durchgeführt, um die Präzision des vorgeschlagenen Algorithmus zu bestimmen. Abschließend wird festgestellt, dass der vorgeschlagene komplementäre Filter für den Einsatz in am Körper getragenen Sensoren geeignet ist, was die Ergebnisse der vorliegenden Arbeit bestätigt. Darüber hinaus wurde in [A90] ein kopfgetragenes System zur Sturzerkennung vorgeschlagen, das den Madgwick-Filter zur Berechnung der Orientierung des kopfgetragenen Geräts verwendet. Die Autoren geben an, dass das Madgwick-Filter wegen seines geringeren Rechenaufwands und der Tatsache, dass er mit niedrigen Aktualisierungsfrequenzen gut funktioniert und keine Initialisierungsphase benötigt, verwendet wird. Auch diese beiden Ansätzen untersuchen nicht systematisch extrafunktionale Eigenschaften.

Im Allgemeinen werden bei den meisten Ansätzen die Untersuchungen nicht direkt auf einem intelligenten Sensor durchgeführt. Mangels der Möglichkeit, direkt auf dem Sensor reproduzierbar zu testen und zu validieren, werden die Untersuchungen oft in Simulationen oder nur funktional auf leistungsfähigeren Rechnern durchgeführt. Unter Verwendung von SiL kann die Untersuchung direkt auf dem Sensor erfolgen, was auch die Bestimmung der extrafunktionalen Eigenschaften erlaubt.

6.1.3 Theoretische Grundlagen

Bevor die Bewertung der funktionalen und extra-funktionalen Eigenschaften der Sensorfusionsalgorithmen in den Abschnitten 6.1.4 und 6.1.5 beschrieben werden, werden in diesem Abschnitt allgemeine Informationen über die verwendeten Sensorfusionsalgorithmen, Datenformate, Hardware und die Implementierung gegeben. Diese Informationen sind wichtig, um die Ergebnisse und Interpretationen in den richtigen Kontext zu stellen. Darüber hinaus werden in diesem Abschnitt die Methoden beschrieben, die zur Bewertung der extrafunktionalen Eigenschaften und der funktionalen Eigenschaften der Fusionsalgorithmen verwendet wurden.

Sensorfusionsalgorithmen

Für die Untersuchung der AHRS-Sensorfusionsalgorithmen wurden die vier am häufigsten verwendeten Algorithmen zur Bestimmung der Orientierung eines Geräts ausgewählt, nämlich das Madgwick-Filter, das Mahony-Filter, ein erweitertes Kalman-Filter und ein komplementäres Filter. Die Implementierungen des erweiterten Kalman-Filters sowie des komplementären Filters verwenden die in [A91] beschriebene Methode zur Bestimmung der Orientierung aus Beschleunigungs- und Magnetometerdaten, um die Verwendung von rechenintensiven trigonometrischen Funktionen zu vermeiden.

Das **Komplementärfilter** ist das grundlegendste der verwendeten Filter. Es macht sich die Tatsache zunutze, dass die Daten des Gyroskops bei höheren Frequenzen und die Daten des Beschleunigungsmessers bei niedrigeren Frequenzen präziser sind. Das Komplementärfilter wendet einen Tiefpassfilter auf die aus den Beschleunigungsmesserdaten berechnete Orientierung und einen Hochpassfilter auf die aus den Gyroskopdaten berechnete Orientierung an [A84]. Beide Orientierungen werden dann über eine gewichtete Addition gemäß Gleichung (6.1) kombiniert.

$$q_t = (1 - \alpha) \cdot q_{acc} - \alpha \cdot q_{gyr} \tag{6.1}$$

Die Orientierungsquaternion q_{acc} wird aus den Daten des Beschleunigungssensors nach der in [A91] beschriebenen Methode extrahiert. Die Orientierung q_{gyr} die Orientierung

aus dem vorherigen Schritt q_{t-1}, die durch die aktuellen Winkelraten des Gyroskops aktualisiert wird. Dieses Filter verwendet einen einzigen Parameter α, mit dem die Gewichtung zwischen den Beschleunigungsmesserdaten und den Gyroskopdaten festgelegt wird.

Das **Mahony-Filter** ist eine Verbesserung des *Komplementärfilters*, indem ein Proportional-Integral-Regler (PI-Regler) auf die Fehlerfunktion zwischen der Orientierung aus den Beschleunigungsmesserdaten und der Orientierung aus den Gyroskopdaten angewendet wird. Daher liefert das Mahony-Filter gute Ergebnisse und ist dennoch rechnerisch kostengünstig. Der integrale Teil des Reglers ist in der Lage, einen konstanten Offset, der durch verzerrte Beschleunigungsmesser- oder Gyroskopdaten verursacht wird, zuverlässig zu entfernen. Dieses Filter hat zwei Parameter zur Abstimmung der Ergebnisse, den Faktor für den proportionalen Teil K_p und den Faktor für den integralen Teil K_i. Die proportionalen und integrierten Fehler werden für jede Achse der Gyroskopdaten berechnet. Die Korrektur des PI-Reglers wird auch auf die Eingangsdaten des Gyroskops angewendet. Der Korrekturschritt ist in Gleichung (6.2) zu sehen, die die Korrektur der Winkelrate für eine der drei Achsen zeigt.

$$anglerate_{corrected} = anglerate + error_p \cdot K_p + error_i \tag{6.2}$$

$$error_i = error_i + error_p \cdot K_i \cdot dt \tag{6.3}$$

Der proportionale Fehler $error_p$ wird als Summe des Kreuzprodukts zwischen dem durch die Orientierung geschätzten und durch die Gyroskopdaten aktualisierten Schwerevektor und dem durch den Beschleunigungsmesser gemessenen Schwerevektor berechnet. Der integrale Fehler $error_i$ wird mithilfe von Gleichung (6.3) durch Integration des proportionalen Fehlers $error_p$ multipliziert mit K_i und der Abtastperiode dt berechnet.

Die korrigierte Winkelrate $anglerate_{corrected}$ wird dann zur Aktualisierung der aktuellen Orientierung verwendet.

Das **Madgwick-Filter** basiert ebenfalls lose auf den Konzepten des *Komplementärfilters*, verwendet jedoch einen Gradientenabstiegsalgorithmus, um den Orientierungsfehler zu berechnen und die Orientierung aus den Beschleunigungsmesserdaten und den Gyroskopdaten zu fusionieren. Dieses Filter verwendet einen einzigen β-Parameter, der für die Steilheit des Gradientenabstiegsalgorithmus verwendet wird. Die Korrektur der Orientierung unter Verwendung des β-Parameters ist in Gleichung (6.4) dargestellt.

$$q_t = q_{t-1} + (update_{gyr} - \beta \cdot S) \cdot dt \tag{6.4}$$

Der Wert $update_{gyr}$ ist die aus den Gyroskopdaten berechnete Änderungsquaternion. Die Korrekturschritt-Quaternion S wird aus der aktuellen Orientierung und den Daten des Beschleunigungsmessers mithilfe eines Gradientenabstiegsalgorithmus berechnet. Der korrigierte Aktualisierungsschritt für die Orientierung wird mit der Abtastperiode dt multipliziert, bevor er zur letzten Orientierung q_{t-1} hinzugefügt wird.

Das **Kalman-Filter** ist das rechenaufwendigste Filter, der in dieser Untersuchung verwendet wird. Theoretisch ist das Kalman-Filter ein optimaler Fehlerschätzer für lineare Probleme mit gaußschem Rauschen [A92], [A93]. Zu seiner Berechnung verwendet das Kalman-Filter mehrere Matrizen und Vektoren. Der Vektor $\mathbf{x}$ enthält den Zustand des Systems, während die Matrix P die Kovarianzmatrix für die Zustandsvariablen enthält. Die Matrizen A und H beschreiben die Dynamik des Systems. Die Matrix A ist die Zustandsübergangsmatrix zur Vorhersage des nächsten Schrittes. Im Falle eines erweiterten Kalman-Filters muss diese Matrix bei jedem Aktualisierungsschritt mithilfe einer Linearisierungsmethode berechnet werden. Die Matrix H beschreibt den Sensoreingang. Die Matrix R beschreibt das Messrauschen und die Matrix Q beschreibt das Rauschen des Systems. Sowohl R als auch Q sind die wichtigsten Matrizen, die zur Kalibrierung des Kalman-Filters verwendet werden.

Das Kalman-Filter ist ein rekursives Filter, das in zwei Schritten arbeitet. Im ersten Schritt wird der aktuelle Zustand unter Verwendung der vorherigen Zustandsvariablen und ihrer Unsicherheiten vorhergesagt. Der Vorhersageschritt ist in den Gleichungen Gleichung (6.5) und Gleichung (6.6) dargestellt.

$$\mathbf{x}_t^- = \mathbf{x}_{t-1} \cdot A \tag{6.5}$$

$$P_t^- = A \cdot P_{t-1} \cdot A^T + Q \tag{6.6}$$

Der Zustandsvektor $\mathbf{x}_t^-$ enthält die Vorhersage des nächsten Zustands. Die Matrix P_t^- ist die vorhergesagte Kovarianz des nächsten Zustands.

Auf die Vorhersagephase folgt die in Gleichung (6.7), Gleichung (6.8) und Gleichung (6.9) beschriebene Korrekturphase.

$$K_t = P_t^- \cdot H \cdot (H \cdot P_t^- \cdot H^T + R)^{-1} \tag{6.7}$$

$$\mathbf{x}_t = \mathbf{x}_t^- + K \cdot (\mathbf{z}_t - H \cdot \mathbf{x}_t^-) \tag{6.8}$$

$$P_t = (I - K_t \cdot H) \cdot P_t^- \tag{6.9}$$

In K_t ist die Kalman-Verstärkung des aktuellen Schritts. Die in Gleichung (6.7) berechnete Kalman-Verstärkung K_t wird in Gleichung (6.8) verwendet, um den nächsten Zustand unter Verwendung des vorhergesagten Zustands $\mathbf{x}_t^-$ und des Sensoreingangsvektors $\mathbf{z}_t$ zu berechnen. Die Kovarianz P_t wird in Gleichung (6.9) unter Verwendung

der vorhergesagten Kovarianzmatrix P_t^- und der Kalman-Verstärkung K_t berechnet. Die Matrix I ist eine Einheitsmatrix.

6.1.3.1 Quaternion Repräsentation

Alle in Abschnitt 6.1.3 beschriebenen Algorithmen schätzen die Orientierung des Inertialsensorsystems anhand der Quaternionendarstellung. Quaternionen werden häufig in der Sensorfusion, Computergrafik und Navigation verwendet. Andere häufig verwendete Darstellungen sind Euler-Winkel, Drehmatrizen oder Achsenwinkel.

Im Vergleich zu Rotationsmatrizen benötigt die Quaternionendarstellung weniger Werte, um eine Drehung darzustellen. Bei der Sensorfusion besteht ein wesentlicher Vorteil der Quaternionen darin, dass es Methoden zur reibungslosen Interpolation zwischen zwei Orientierungen durch lineare Interpolation und die präzisere sphärische lineare Interpolation gibt [A94].

Bei Euler-Winkeln besteht das Problem der kardanischen Verriegelung oder der Mehrdeutigkeit [A95]. Dieses Problem tritt bei Quaternionen nicht auf. Außerdem benötigen die Quaternionen im Gegensatz zu den Euler-Winkeln keine trigonometrische Funktionen für ihre Berechnung, was sie für den Einsatz auf kleinen µCs oder intelligenten Sensoren geeignet macht.

Ein weiterer wichtiger Vorteil von Quaternionen ist die Tatsache, dass sie auf eine Länge von 1 normiert sind, um eine Orientierung darzustellen. Wenn man den Wertebereich der Eingabedaten kennt, kann man die erforderliche Position des Radixpunktes bestimmen, um Überläufe zu vermeiden und die höchstmögliche Genauigkeit bei der Verwendung von Festkomma-Arithmetik beizubehalten.

Mathematisch gesehen können Quaternionen als ein vier-dimensionaler Vektor $q = (w, x, y, z)$ betrachtet werden, wobei der skalare Teil w und die Werte x, y und z die Drehung darstellen. Für die spätere Verwendung in dieser Arbeit wird q_w als das w-Element des Quaternions q definiert.

6.1.3.2 Datenformate

Je nach Zielhardware kann das verwendete Datenformat einen erheblichen Einfluss auf die Berechnungskomplexität eines Algorithmus haben. Dies gilt insbesondere für die Verwendung von Fließkommadaten auf einer Hardware welche ohne weiteres keine Fließkommaoperationen unterstützt. Vor allem in intelligenten Sensoren fehlt oft eine Einheit für die Berechnung von Fließkommazahlen. Auch der in dieser Arbeit verwendete Sensor besitzt, wie in Unterabschnitt 6.1.3.3 beschrieben, keine solche Einheit. Im Folgenden beschreiben wir die in unserer Studie verwendeten Datenformate.

Fließkommazahlen mit einfacher Genauigkeit

Standardmäßig werden die meisten Sensorfusionsalgorithmen mit einer Fließkommazahlendarstellung implementiert. Dies ist auch bei den vier untersuchten Filtern der Fall. Fließkommazahlen haben den Vorteil, dass sie sowohl sehr große als auch sehr kleine Zahlen speichern können. Sie nutzen eine Kombination aus einem Exponenten und Mantisse zur Darstellung der Zahl. Daher können Fließkommazahlen wesentlich größere Wertebereiche abdecken als Festkommazahlen derselben Bitlänge. Fließkommazahlen mit einfacher Genauigkeit haben einen Exponenten von 8 Bit und eine Mantisse von 23 Bit, was die Grenze für ihre Genauigkeit darstellt. Der Nachteil von Fließkommazahlen ist, dass sie bei allen Rechenoperationen mehr Aufwand erfordern. Selbst bei einer einfachen Addition müssen beide Werte zunächst so umgewandelt werden, dass sie denselben Exponenten haben, bevor die Addition durchgeführt werden kann [A96]. Anschließend muss das Ergebnis der Addition normalisiert werden, da sonst die Genauigkeit der Daten mit jeder Rechenoperation abnimmt. Diese erhöhte Komplexität erschwert die Verwendung von Fließkommazahlen in einer hardwarebeschränkten Umgebung.

Festkommazahl

Wie der Name schon sagt, ist bei Festkommazahlen der Radixpunkt an einer bestimmten Bitposition festgelegt. Dies ist unabhängig von dem Wert, der in der Festkommazahl gespeichert ist. Ihr größter Vorteil gegenüber den Fließkommazahlen ist der geringere Rechenaufwand, der für mathematische Operationen benötigt wird. Insbesondere Additionen und Subtraktionen sind sehr effizient. Multiplikationen erfordern zusätzliche Verschiebe- und optionale Rundungsoperationen, sind aber immer noch effizienter als Fließkommamultiplikationen [A97]. Der Nachteil der Festkomma-Arithmetik besteht darin, dass der Datenbereich und die Genauigkeit der Daten reziprok zueinander sind. Ein größerer Bereich der gespeicherten Zahlen führt zu einer geringeren Genauigkeit und umgekehrt. Die Verwendung von Festkommazahlen ist der bevorzugte Weg für eine hardwarebeschränkte Umgebung, wie z. B. bei intelligenten Sensoren. Aufgrund des Zusammenspiels von Bereich und Genauigkeit muss geprüft werden, ob Festkommazahlen für das angestrebte Szenario geeignet sind.

6.1.3.3 Sensorhardware

Bei der für die Messung verwendeten Hardware handelt es sich um einen BMF055, einen intelligenten Sensor von *Bosch Sensortec* [A18]. Der BMF055 verwendet den SAM D20-Mikrocontroller, einen ARM Cortex M0+ Mikrocontroller von *Microchip*, der mit einer Frequenz von 48 MHz betrieben wird [A54]. Er verfügt über eine 32 Bit Architektur mit zwei Pipelinestufen und ist für einen geringen Stromverbrauch optimiert. Wichtig für die Bewertung der extrafunktionalen Eigenschaften der Algorithmen sind zwei Schlüsseleigenschaften des Sensors:

- Der SAM D20 enthält einen Ein-Zyklus-Hardware-Multiplikator für 32 Bit-Ganzzahlen, daher benötigen Addition und Multiplikation die gleiche Zeit.
- Der SAM D20 hat keine Hardware-Unterstützung für Fließkommazahlen. Alle Fließkommaoperationen müssen in Software emuliert werden, was zu einer höheren Ausführungszeit und einem höheren Stromverbrauch führt.

Das Gyroskop des BMF055 entspricht dem des BMI055 mit einem Bereich von bis zu ±2000 °/s und einer Datenbreite von 16 Bit [A18], [A98]. Der Beschleunigungsmesser entspricht dem BMA280 mit einem maximalen Bereich von ±16g und einer Datenbreite von 14 Bit [A18], [A51]. Das Magnetometer entspricht dem BMM150 mit einer Auflösung von 0,3μT und einer Datenbreite von 13 Bits [A18], [A53].

Der μC benötigt einen Versorgungsstrom von etwa 1 mA im Leerlauf und bis zu 5,9 mA unter Volllast. Der gesamte Versorgungsstrom des BMF055 kann bis zu 13,7 mA betragen, wenn alle Sensoren mit einer ODR von 100 Hz im Normalmodus verwendet werden. Im Energiesparmodus und werden durchschnittlich 2,6 mA benötigt.

6.1.3.4 Analyse extrafunktionaler Eigenschaften

Die Analyse der extrafunktionalen Eigenschaften wurde mit der in Unterabschnitt 6.1.3.3 beschriebenen realen Hardware durchgeführt. Die Speicherbelegung unterteilt in RAM- und ROM-Speicher ist die erste untersuchte Eigenschaft. Weiterhin wurde der Rechenaufwand der Algorithmen über die Ausführungszeit für die Bestimmung der Orientierung untersucht.

Codegröße

Für die Bewertung der extrafunktionalen Eigenschaften wurden die Codegröße der Algorithmen und die durchschnittliche Rechenzeit, die für die Aktualisierung der Orientierung anhand neuer IMU-Daten erforderlich ist, für den Cortex M0+ μC analysiert. Die Kompilierung wurde mit dem GNU C Compiler für jeden der Algorithmen mit der Optimierungsoption -O0 und dem Release-Modus mit -O3 durchgeführt, um einen fairen Vergleich zu ermöglichen.

Um zu untersuchen, wie viel Speicherplatz die vier Fusionsalgorithmen im Speicher der Sensor-Prozessor-Einheit (Sensor Processing Unit, SPU) benötigen, wurden die einzelnen Kompilate mit dem Tool *size tool* aus der GNU ARM Embedded Toolchain auf die Größe des Binärcodes untersucht. Dabei wurde zwischen dem Platzbedarf im Flash-Speicher (ROM) und dem Platz für die Daten im SRAM unterschieden.

Rechenaufwand

Für den rechnerischen Aufwand wurden alle Algorithmen auf dem realen intelligenten Sensor in Kapitel 5 vorgestellten Sensor-in-the-Loop-Architektur ausgeführt und

bewertet. Um vergleichbare Ergebnisse sicherzustellen, wurde ein Satz zuvor aufgezeichneter menschlicher Bewegungsdaten verwendet. Dann wurde für jeden Testlauf die SiL-Funktionalität benutzt, um diese Daten reproduzierbar in den Sensor zu injizieren. Diese Methode ermöglicht es, für alle Algorithmen und Datenformate exakt die gleichen Daten zu verwenden und gewährleistet so eine hohe Vergleichbarkeit der Messungen, während die Software weiterhin direkt auf der Zielhardware ausgeführt wird. Der Aufbau für die Messung ist in Abbildung 6.1 zu sehen.

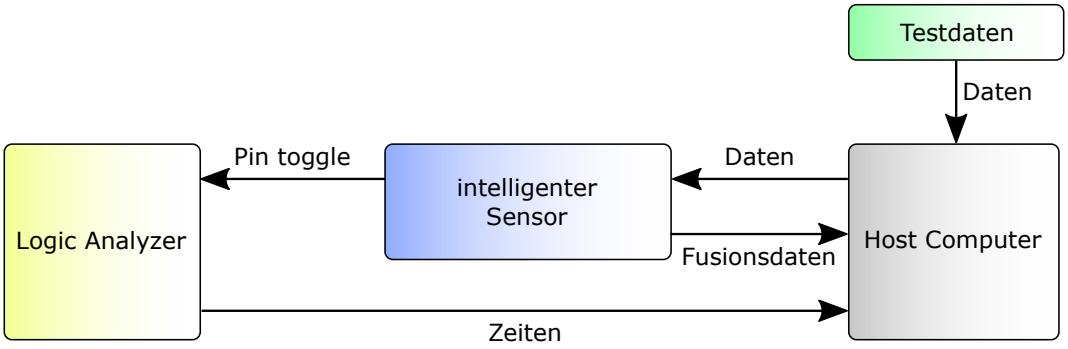


Abbildung 6.1: Messaufbau zum Ermitteln der Rechenzeiten der einzelnen Fusionsalgorithmen

Um den Rechenaufwand der Algorithmen zu messen, sendet der Host-Computer die Sensordaten an den Sensor. Der intelligente Sensor setzt einen Ausgangspin des µC auf *high*, wenn die Daten ankommen und die Ausführung des Algorithmus beginnt. Sobald der Algorithmus beendet ist, wird der Pin wieder auf *low* zurückgesetzt. Der besagte Pin wird mit einem Logic Analyzer überwacht, welcher den Zustand des Pins aufzeichnet und die Daten zur späteren Analyse an den Host-Computer sendet. Die Zeit für das Umschalten des Pin, wurde zuvor gemessen und bei der Analyse berücksichtigt. Mit dieser Methode kann die Zeit gemessen werden, die der Fusionsalgorithmus benötigt, um die Orientierung für jedes neue Sensordatum zu aktualisieren. Für die Analyse wurden sowohl die durchschnittliche Zeit zur Aktualisierung der Orientierung als auch die Varianz der benötigten Zeit untersucht.

6.1.3.5 Analyse funktionaler Eigenschaften

Die Bewertung der Qualität der Ergebnisse der vier Orientierungsfilter erfolgte anhand einer externen Referenz, mit der das Ergebnis der Sensorfusion verglichen wurde. Die externe Referenz wurde mit der in [A99], [A100] beschriebenen Methode ermittelt. Diese Methode nimmt ein Muster über eine Kamera auf und verwendet die aufgenommenen Bilder, um die relative Ausrichtung zwischen Kamera und Muster über eine Bildanalyse unter Verwendung der OpenCV-Bibliothek [A101] zu berechnen. Ein Smartphone mit einer Benutzeroberfläche wurde verwendet, um den Benutzer über die gewünschte

Bewegung zu instruieren und das Muster ähnlich wie bei [A100] aufzuzeichnen. Für die Untersuchung wurde der intelligente Sensor fest mit dem Muster verbunden und dann vor der statischen Kamera bewegt. Es ist wichtig zu beachten, dass die Bewegungen des Sensors von einer Person ausgeführt werden, die einer Anleitung auf dem Bildschirm folgt. Auf diese Weise wurde jedes Experiment mit ähnlichen, aber nicht identischen Bewegungen durchgeführt. Dies ist der Grund, warum in Abschnitt *statistische Analyse* eine statistische Analyse zur Auswertung durchgeführt wird. Somit gelten die Experimente auf diese Weise auch für Bewegungen, die von am Körper getragenen Sensoren erfasst werden können. Abbildung 6.2 zeigt die Schritte um die Sensordaten und Referenzdaten zu erhalten und zu vergleichen.

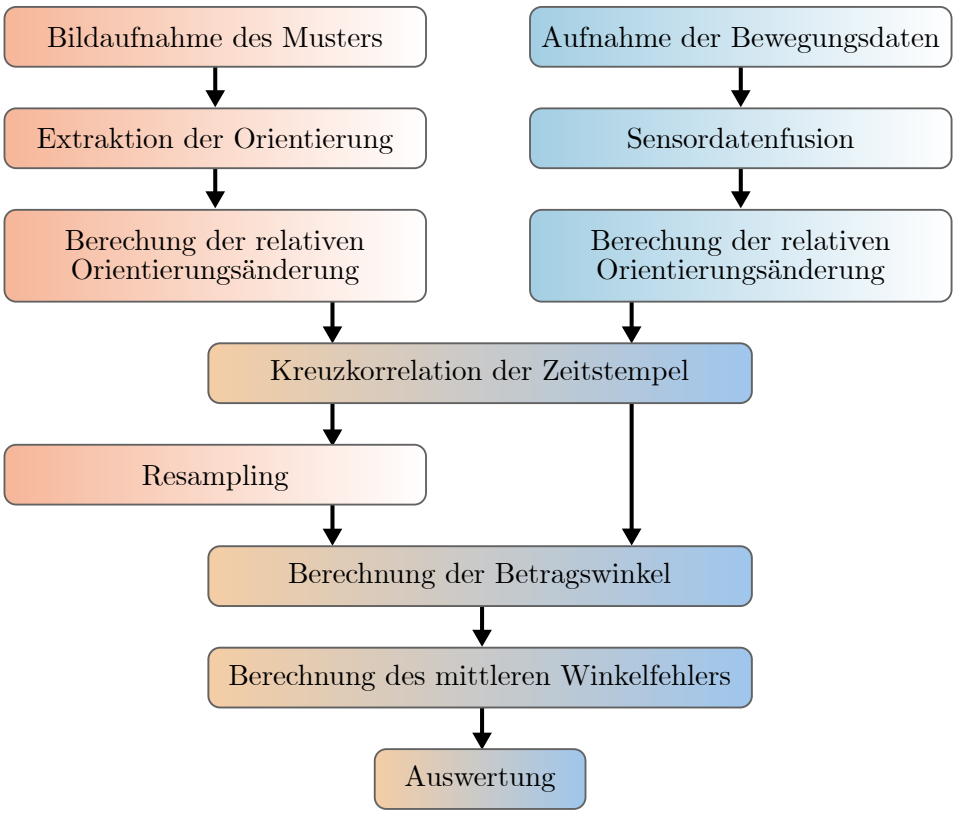


Abbildung 6.2: Datenfluss der Messsetups zur funktionalen Analyse der Fusionsalgorithmen

In einem ersten Schritt wurden die Daten aus beiden Quellen gleichzeitig erfasst und für die weitere Untersuchung gespeichert. Für beide Datensätze wurde die relative Orientierungsänderung zum Beginn der Messung berechnet. Die Verwendung der relativen Orientierung anstelle der absoluten Orientierung ermöglichte es einen direkten Vergleich zwischen den Orientierungen durchzuführen, ohne beide Orientierungen in ein gemeinsames Weltkoordinatensystem zu transformieren. Der nächste Schritt der Auswertung ist eine Kreuzkorrelation der Zeitstempel der Daten beider Quellen. Dieser Schritt ist

notwendig, da Sensor- und Kamerasystem unabhängig voneinander laufen und möglicherweise unterschiedliche Zeitbasen für ihre Zeitstempel verwenden und eine Taktdrift aufweisen.

Schließlich müssen die Referenzdaten der Kamera neu abgetastet werden, um einen direkten Vergleich mit den Daten aus der Sensorfusion zu ermöglichen.

Der Fehler zwischen der Ausgangsorientierung der Filter und der Referenzorientierung wird als minimaler Winkel berechnet. Dies ist die Drehung um eine beliebige Achse, um die Ausgangsorientierung in die Referenzorientierung zu transformieren. Die Formel zur Berechnung des Fehlerwinkels ist in Gleichung (6.11) dargestellt. Das Ergebnis $qDiff$ ist die Differenz-Quaternion zwischen der Referenz und dem Filterausgang. Wie in Gleichung (6.10) gezeigt, wird $qDiff$ als Ergebnis der Quaternionenmultiplikation zwischen $qRef$ und der $conjugate$ Quaternion des $qFilter$ berechnet.

$$qDiff = qRef \cdot conjugate(qFilter) \tag{6.10}$$

$$error = 2 \cdot atan(qDiff_w) \cdot \frac{180}{\pi} \tag{6.11}$$

Zur Berechnung des durchschnittlichen Fehlers zwischen Referenz- und Sensor-Fusionsergebnis wurde der mittlere quadratische Fehler verwendet.

Filterparameter und Messkonfiguration

Die Messung der Sensordaten wurde mit einer Abtastfrequenz von 200 Hz durchgeführt. Die Bilddaten für die Referenz wurden mit 30 Hz aufgezeichnet. Das Gyroskop wurde auf eine Empfindlichkeit von ±2000°/s maximaler Winkelrate konfiguriert, was zu einer Auflösung von 0,061°führt. Der Beschleunigungsmesser wurde auf eine maximale Beschleunigung von ±16 g eingestellt, was eine Auflösung von 0,0048 m/s^2 ergibt.

Alle Filter wurden so konfiguriert, dass sie mit zuvor durchgeführten Kalibrierungsmessungen optimal funktionieren. Die unten gewählten Parameter werden in Abschnitt Abschnitt 6.1.3 beschrieben. Für das Komplementärfilter wurde ein α-Wert von *0,97* verwendet. Dieser Parameter liegt nahe an einem α-Wert von *0,98*, der häufig für ein komplementäres Filter verwendet wird. Das Mahony-Filter verwendet einen Wert von *1,05* für den proportionalen Teil K_p und einen Wert von *0,2* für den integralen Teil K_i des PI-Reglers. Das Madgwick-Filter verwendet einen β-Wert von *0,115*. Sowohl für das Mahony-Filter als auch für das Madgwick-Filter liegen die gewählten Werte nahe an den von Madgwick verwendeten Originalwerten [A82]. Beim Kalman-Filter wird für die Prozessrauschmatrix Q ein Wert von *0,1* für die Einträge in der Hauptdiagonale verwendet. Die Messrauschmatrix R verwendet für die Einträge in der Hauptdiagonalen einen Wert von *50,0*. Der *0,1* für die R-Matrix wurde anhand des typischen Rauschens

ermittelt, das im Datenblatt des Gyroskops [A98] angegeben ist. Die Q-Werte wurden empirisch ermittelt, da sie von der erwarteten Stärke der vom Benutzer ausgeführten Bewegung abhängen.

Statistische Analyse

Für eine eingehendere Analyse des Einflusses der verschiedenen Sensorfusionsalgorithmen und Datenformate wurde eine statistische Analyse der Ergebnisse durchgeführt. Zu diesem Zweck wurden die statistischen Methoden einer Gage-R&R-Analyse verwendet, um abzuschätzen, wie stark die Unterschiede zwischen den Messungen durch verschiedene Faktoren beeinflusst werden. Bei einer traditionellen Gage-R&R-Analyse sind diese Faktoren das geprüfte Teil, die Person, die das Teil prüft, und die Wiederholung der Messung [A67], [A102]. In dieser Arbeit wurden diese Methoden in der statistischen Analyse angepasst, um den Einfluss des verwendeten Algorithmus, des Datenformats bzw. der Bewegungsgeschwindigkeit zu bewerten.

6.1.4 Extrafunktionale Eigenschaften

Ein wesentlicher Aspekt der in der vorliegenden Arbeit untersuchten Sensorfusionsalgorithmen sind ihre extrafunktionalen Eigenschaften. Zu diesen Eigenschaften gehören der Rechenaufwand, die Codegröße und die Konfigurierbarkeit der verwendeten Sensorfusionsalgorithmen. Insbesondere im Kontext von ressourcenbeschränkter Hardware, wie z.B. intelligenten Sensoren, entscheiden der Rechenaufwand und die Codegröße darüber, ob ein Algorithmus nur mit eingeschränkter Funktionalität oder überhaupt nicht verwendet werden kann. In diesem Abschnitt werden die Codegröße und der Rechenaufwand in Bezug auf die Reaktionszeit und den Durchsatz der vier Sensorfusionsalgorithmen diskutiert. Diese Algorithmen wurden jeweils mit Verwendung drei unterschiedlicher Datenformaten implementiert. Bevor die Analyse der extrafunktionalen Eigenschaften durchgeführt wurde, wurde sichergestellt, dass alle Algorithmen mit allen drei Datenformaten funktionsfähig sind. Eine eingehende Analyse der funktionalen Eigenschaften, d.h. der Qualität der Sensorfusion, wird später in Unterabschnitt 6.1.5 behandelt.

6.1.4.1 Codegröße

Die Codegröße wurde für jede Kombination von Fusionsalgorithmus und Datenformat bestimmt. Die Ergebnisse werden in diesem Abschnitt dargestellt und diskutiert. Grundsätzlich wurde in allen hier gezeigten Ergebnissen keine Compileroptimierung angewendet, die generierte Firmware wurde also mit (-O0) kompiliert. Da keiner der verwendeten Fusionsalgorithmen aktive Speicherzuweisung oder rekursive Funktionen verwendet, kann die Ausgabe des Compilers direkt die benötigte Speichergröße der Algorithmen

anzeigen. Die Größen der Binärdateien im Flash-Speicher sind in Abbildung 6.3 dargestellt.

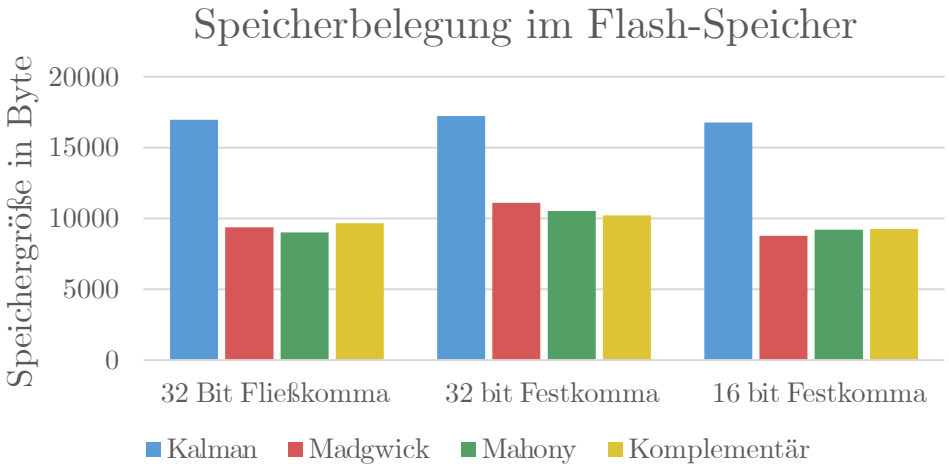


Abbildung 6.3: Flash-Speicherbelegung der untersuchten Algorithmen

Es ist zu erkennen, dass es keinen großen Unterschied zwischen den Datenformaten gibt. Die Fließkomma-Implementierung und die 16 Bit Festkomma-Implementierung sind ungefähr gleich groß, die 32 Bit Festkomma-Implementierung ist etwas größer. Zwischen dem Kalman-Filter und den anderen Filtern gibt es jedoch einen erheblichen Größenunterschied. Dies ist hauptsächlich auf die für das Kalman-Filter erforderlichen Matrixoperationen zurückzuführen, die sowohl mehr Code als auch etwas mehr SRAM zum Speichern der Werte oder Zwischenwerte während der Berechnung erfordern. Das Madgwick-, das Mahony- und das Komplementärfilter benötigen für ihre Berechnungen keine Matrixoperationen und sind daher kleiner. Die Matrixoperationen für das Kalman-Filter stammen aus der *Eigenmatrix-Bibliothek* [A103]. Für die besonders komplexe Berechnung der inversen Matrix wird eine spezielle Methode für 4x4-Matrizen verwendet, die aus der *MESA*-Implementierung der *OpenGL Utility Libraries* [A104], [A105] stammt.

Abbildung 6.4 zeigt die Größe, die die Algorithmen im dynamischen Speicher (SRAM) des μC benötigen um Werte, Zwischenergebnisse und den Funktionenstack zu speichern.

Interessanterweise benötigt das Kalman-Filter nicht viel mehr dynamischen Speicher, um die Daten der Matrizen und Zwischenergebnisse zu speichern. Dies lässt sich dadurch erklären, dass das Madgwick-Filter, das Mahony-Filter und das Komplementärfilter so optimiert sind, dass sie viele der Zwischenergebnisse in temporären Variablen speichern. Dies vermeidet, dass derselbe Wert bei mehrfacher Verwendung wiederholt berechnet wird. Es ist zu erkennen, dass die Verwendung des 16 Bit Festkomma-Datentyps die Menge des benötigten SRAM im Vergleich zur 32 Bit Fließkomma-Implementierung reduziert. Der 32 Bit Festkomma-Datentyp benötigt etwas mehr SRAM.

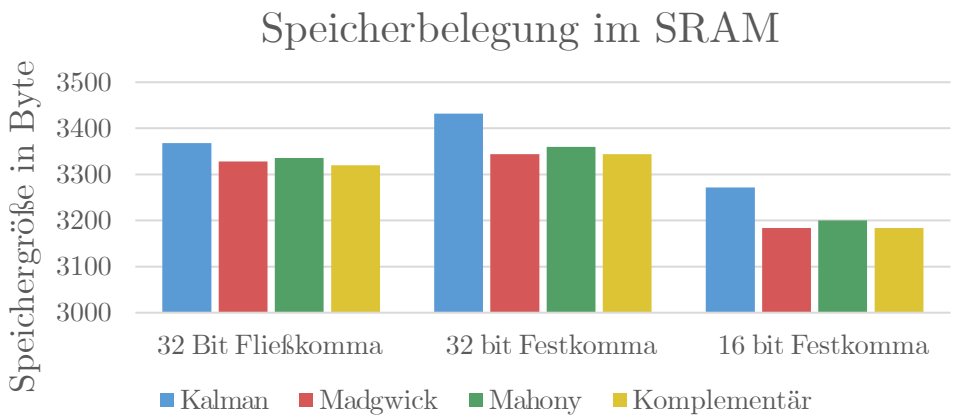


Abbildung 6.4: SRAM-Speicherbelegung der untersuchten Algorithmen

Erläuterung der Codegrößendifferenzen

Da wie in Unterabschnitt 6.1.3.3 beschrieben, aufgrund der fehlenden Hardware-Unterstützung eine SoftFloat-Bibliothek erforderlich ist, wäre zu erwarten, dass die Fließkomma-Implementierung der Algorithmen mehr Speicher benötigt als die Festkomma-Implementierungen. Dies ist jedoch nicht der Fall, da auch die Festkomma-Implementierungen zusätzlichen Code benötigen, um zu funktionieren. Einerseits benötigen die Multiplikationen zusätzliche Verschiebeoperationen für den resultierenden Wert und eine Rundung des Ergebnisses. Zum anderen wird eine Implementierung der Funktion $\frac{1}{\sqrt{x}}$ benötigt, um die Eingangs- und Ausgangswerte zu normalisieren. Diese Funktionen können sehr effizient in einer Fließkomma-Implementierung [A106], [A107] implementiert werden. Die Festkomma-Variante benötigt eine Lookup-Tabelle und damit mehr Speicherplatz. Zusätzlich zu den oben genannten Punkten gibt es noch eine weitere Besonderheit der Festkommaarithmetik, die sowohl die Größe des Codes als auch die Ausführungszeit beeinflusst: Die Multiplikationen müssen mit der doppelten Bittiefe der verwendeten Festkommazahlen durchgeführt werden, um einen Überlauf des Zwischenergebnisses zu verhindern. Bei 16 Bit Festkommazahlen müssen die Multiplikationen in 32 Bit durchgeführt werden, bei 32 Bit Festkommazahlen in 64 Bit. Die für die 32 Bit Festkomma-Implementierungen erforderlichen 64 Bit-Multiplikationen erklären auch die größere Größe der 32 Bit-Implementierung sowohl für den Flash-Speicher als auch für den SRAM. Dies ist der Fall, weil die 64 Bit-Operationen mindestens doppelt so viele Operationen für die mathematischen Berechnungen und doppelt so viele Bytes für die Speicherung von 64 Bit-Zwischenergebnissen erfordern.

6.1.4.2 Ausführungszeit

Die wichtigste extrafunktionale Eigenschaft der bewerteten Fusionsalgorithmen ist die Ausführungszeit bzw. der Rechenaufwand den jeder Algorithmus benötigt, um die Daten von den Inertialsensoren zu verarbeiten. Der Rechenaufwand beeinflusst sowohl die maximale Frequenz, mit der die Daten verarbeitet werden können, als auch den Stromverbrauch des μC.

Die Ergebnisse der Messungen für die Ausführungszeit der Algorithmen sind in Abbildung 6.5 dargestellt.

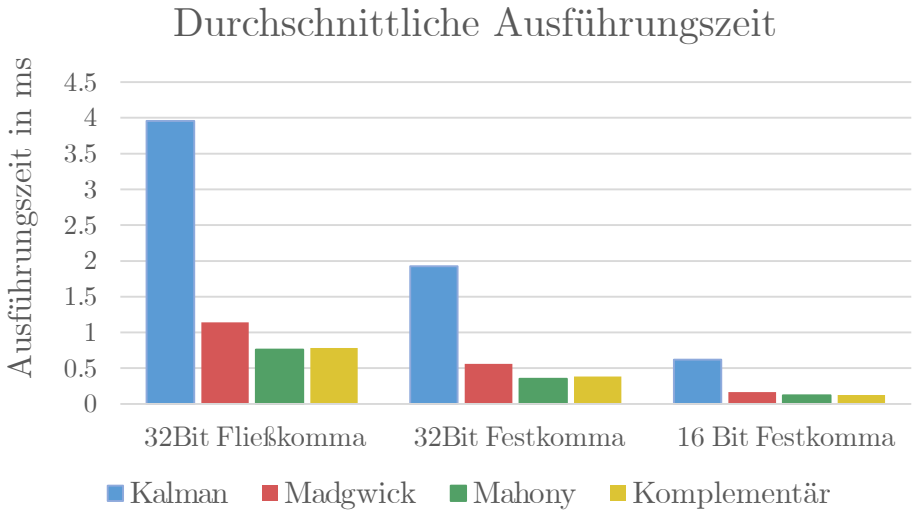


Abbildung 6.5: Ausführungszeit der Algorithmen nach Datenformaten

Es ist deutlich zu erkennen, dass das Kalman-Filter für alle drei Datenformate die höchste Ausführungszeit benötigt. Im Vergleich zu den anderen Filtern benötigt das Kalman-Filter mehr als 3,4-mal so viel Zeit für die Berechnung des Fusionsergebnisses. Bei der 32 Bit Fließkomma-Implementierung benötigt er 3,96 ms für jeden Fusionsschritt, was die maximale Aktualisierungsfrequenz auf etwa 250 Hz begrenzt. In Anbetracht der Tatsache, dass der µC auch andere Berechnungen und Steuerungsaufgaben durchführen muss, dürfte diese Frequenz letztendlich deutlich niedriger sein.

Die zweite Erkenntnis ist, dass die Festkomma-Implementierungen nur einen Bruchteil der Rechenzeit der 32 Bit Fließkomma-Implementierung benötigt. Die 32 Bit Festkomma-Implementierung benötigt für alle vier Algorithmen etwa 50 % weniger Zeit als die 32 Bit Fließkomma-Pendants. Außerdem benötigen die 16 Bit Festkomma-Implementierungen nur ein Drittel der Rechenzeit der 32 Bit Festkomma-Implementierungen.

Die genauen Werte der Ausführungszeiten sind in Tabelle 6.1 aufgeführt.

Datenformat	Kalman [ms]	Madgwick [ms]	Mahony [ms]	Complementary [ms]
32 bit Fließkomma	3.963	1.142	0.758	0.782
32 bit Festkomma	1.923	0.560	0.350	0.382
16 bit Festkomma	0.621	0.166	0.121	0.123

Tabelle 6.1: durchschnittliche Ausführungszeit pro Sensorsample

Um zu beurteilen, wie konsistent sich die Algorithmen verhalten wurde neben der durchschnittlichen Ausführungszeit auch die Varianz der einzelnen Ausführungszeiten ausgewertet. Es wurde eine geringe Varianz erwartet, da die verwendeten Algorithmen nicht viele Verzweigungen enthalten. Daher sollte die Anzahl der Operationen pro Sensordatenaktualisierung nahezu konstant bleiben. Für die Festkomma-Implementierungen hat sich dieses Verhalten bestätigt. Bei der Fließkomma-Implementierung war die Varianz der Ausführungszeiten jedoch höher. Insbesondere die Varianz beim Kalman-Filter war deutlich höher. Die Varianzen der einzelnen Algorithmen sind in Abbildung 6.6 dargestellt.

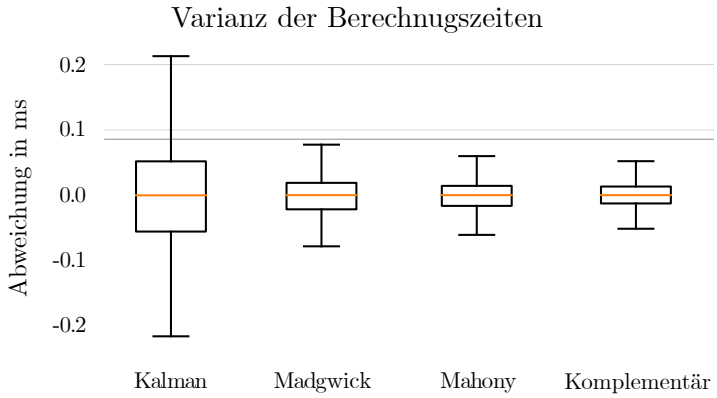


Abbildung 6.6: Box-Plot der Ausführungszeiten für die 32 Bit Fließkomma-Implementierung in ms.

Die höhere Varianz der Ausführungszeiten für die Fließkomma-Varianten hat den gleichen Grund wie die erhöhte Ausführungszeit im Allgemeinen: Die mathematischen Operationen werden innerhalb einer SoftFloat-Bibliothek ausgeführt. Bei Additionen und Multiplikationen ist die Anzahl der Zyklen für jede Operation nicht festgelegt, sondern hängt von den Werten des verwendeten Operanden ab [A96]. Daher variiert die Ausführungszeit je nach dem aktuellen Zustand und der Eingabe des Filters. Besonders das Kalman-Filter hat hier einen Nachteil, da die Anzahl der benötigten Operationen sowie der Bereich der verwendeten Werte im Vergleich zu den anderen Filtern höher ist. Dies führt zu mehr Operationen für die Normalisierung der Daten.

6.1.4.3 Zusammenfassung: Extrafunktionale Eigenschaften

Insgesamt lässt sich feststellen, dass der Rechenaufwand der verwendeten Algorithmen durch die Verwendung einer Festkomma-Implementierung erheblich reduziert werden kann. Insbesondere der Rechenaufwand der 16 Bit Festkomma-Implementierungen ist etwa sechsmal geringer als der ihres 32 Bit Fließkomma-Pendants. Betrachtet man allein den Rechenaufwand, sollten Festkomma-Implementierungen gegenüber Fließkomma-Implementierungen bevorzugt werden. Außerdem sollten das Komplementärfilter, das Madgwick-Filter und das Mahony-Filter gegenüber dem Kalman-Filter bevorzugt werden.

Der geringere Rechenaufwand allein lässt jedoch keine eindeutige Aussage darüber zu, welcher Fusionsalgorithmus und welches Datenformat in einer hardwarebeschränkten Umgebung vorzuziehen ist. Im folgenden Unterabschnitt 6.1.5 wird untersucht, inwieweit die Qualität der verwendeten Sensorfusionsalgorithmen durch die Verwendung von Festkomma-Arithmetik beeinflusst wird und wie die Qualität der Algorithmen im Vergleich aussieht.

6.1.5 Funktionale Eigenschaften

Um die Nutzbarkeit der verwendeten Sensorfusionsalgorithmen und Datenformate zu beurteilen, ist es nicht nur wichtig, deren Codegröße und Rechenaufwand zu untersuchen. Es liegt auf der Hand, dass auch die Qualität der Ausgabe der Sensorfusionsalgorithmen für die Nutzbarkeit entscheidend ist. So kann beispielsweise die Aktivitäts- oder Gestenerkennung beeinträchtigt werden, wenn die Qualität der Sensorfusion durch die Verwendung eines anderen Datenformats zu stark leidet. In diesem Abschnitt werden zwei Untersuchungen zur Qualität der Sensorfusion behandelt.

6.1.5.1 Gegenüberstellung der Fusionsergebnisse

Um einen allgemeinen Überblick über den Einfluss sowohl des verwendeten Fusionsalgorithmus als auch des verwendeten Datenformats zu erhalten, wurden mehrere Messungen für einen direkten Vergleich der Fusionsergebnisse durchgeführt. Dabei wurden jeder der vier Algorithmen mit jeweils drei Datentypen untersucht. Jede dieser Messungen wurde in drei unterschiedlichen Bewegungsgeschwindigkeiten untersucht. Alle der daraus resultierenden 36 Tests wurden 10 Mal wiederholt, in Summe wurden 360 Einzelmessungen durchgeführt. Zu Beginn der Untersuchung wurde beschlossen, die Ergebnisse der Fusionsalgorithmen nur mit den Daten des Beschleunigungsmessers und des Gyroskops zu vergleichen und die Daten des Magnetometers nicht zu verwenden. Erste Messungen

zeigten, dass die Daten für die z-Achse trotz vorheriger Kalibrierung des Magnetometers zu große Fehler aufwiesen, um eine aussagekräftige Bewertung der Qualität der Sensorfusionsalgorithmen zu ermöglichen.

Abbildung 6.7 zeigt die Ergebnisse der Messungen unter ausschließlicher Verwendung der Daten des Gyroskops und des Beschleunigungsmessers.

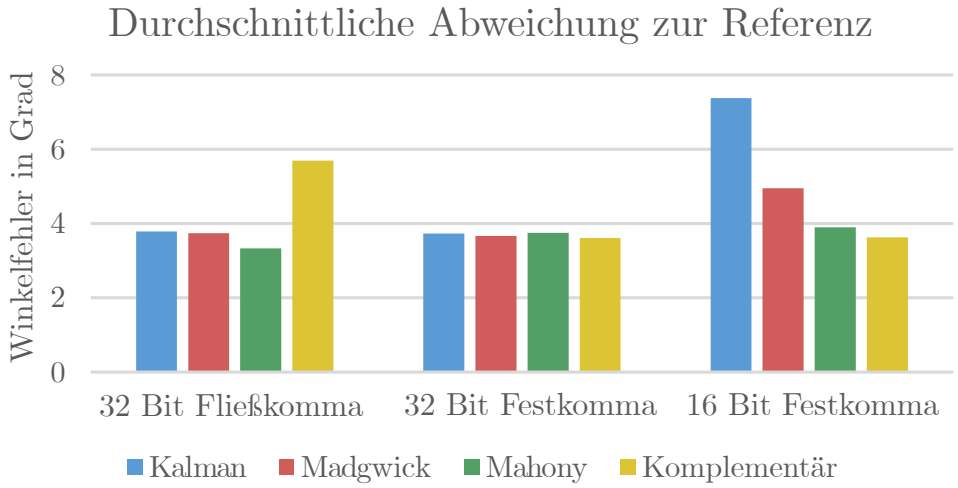


Abbildung 6.7: Fehler inklusive z-Achse

Es ist zu erkennen, dass die Fließkommaformate und die 32 Bit Festkommaformate etwa den gleichen Fehler aufweisen. Das Komplementärfilter weist bei der Fließkomma-Implementierung einen etwas höheren Fehler auf. Bei der 16 Bit Festkomma-Implementierung weist das Kalman-Filter einen deutlich höheren Fehler auf als die anderen verwendeten Algorithmen. Der Grund für diesen Fehler wird später unter Quantisierungsfehler beschrieben. Insgesamt zeigt sich, dass die Fehler für alle Konfigurationen relativ hoch sind mit einem durchschnittlichen Fehler von 4,14° für die Fließkommazahlen, 3,69° für die 32 Bit Festkommazahlen und 4,90° für die 16 Bit Festkommazahlen.

Der Grund für die relativ hohen Fehler ist, dass das Weglassen der Daten des Magnetometers und damit die Möglichkeit die Daten der z-Achse zu korrigieren. Dies führt zu einer Drift der z-Achse welche den Gesamtfehler drastisch erhöht. Abbildung 6.8 zeigt die Drehung für die Achsen x, y und z für eine Messung. Bei diesem Test wurde jede Achse nacheinander um etwa 20°hin und her gedreht wurde.

Die durchgezogenen Linien zeigen die vom Referenzsystem gemessene Rotation. Die gestrichelten Linien zeigen die Schätzung durch den Madgwick-Filter. Es ist zu erkennen, dass die z-Achse der geschätzten Orientierung mit der Zeit von der Referenz weg driftet. Diese Drift ist höher als die durchschnittliche Differenz, die für die x- und y-Achse gefunden wurde, und überlagert den zu messenden Fehler. Daher wurde beschlossen, eine

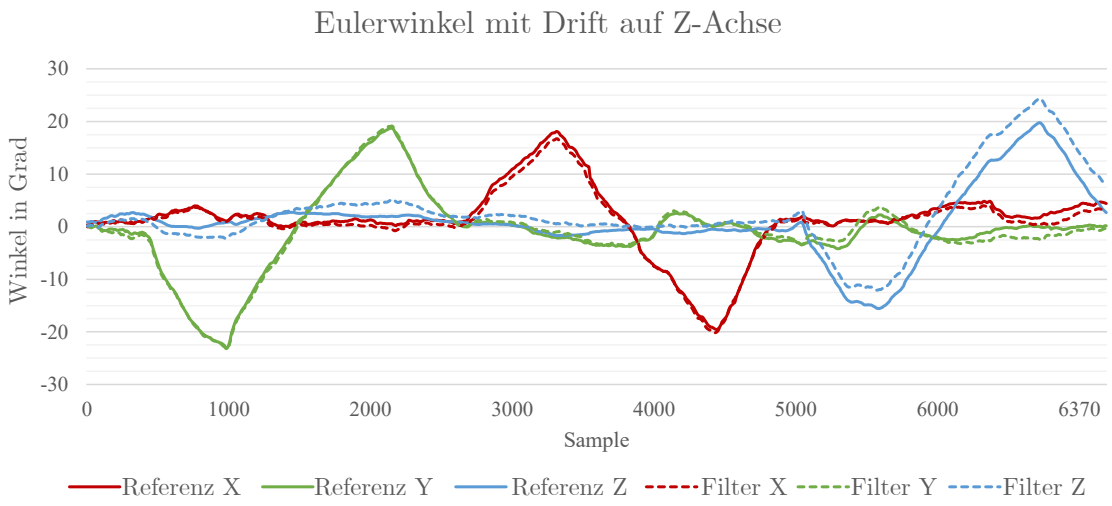


Abbildung 6.8: Eulerwinkel als Ergebnis des Madgwickfilters mit Drift auf der z-Achse

Korrektur für die z-Achse vorzunehmen, um deren Fehler auszuschließen. Die Ergebnisse der Auswertung mit einer korrigierten z-Achse sind in Abbildung 6.9 dargestellt.

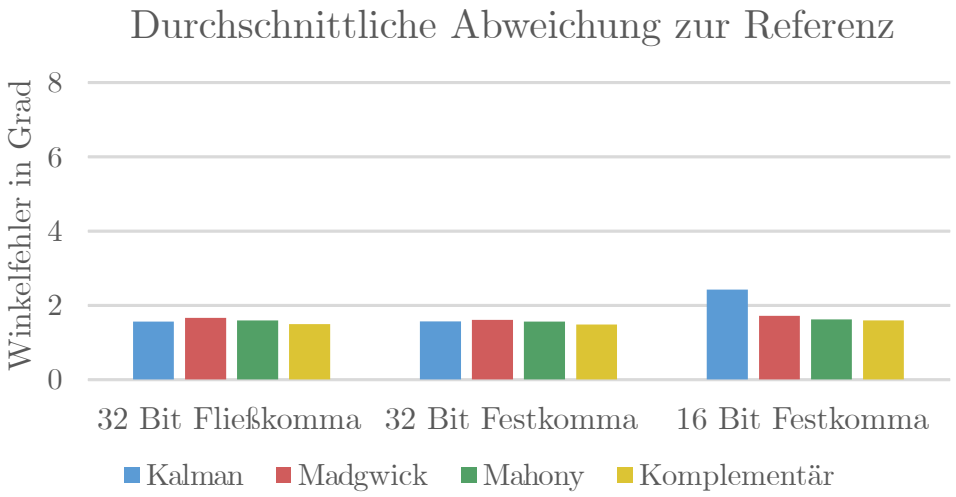


Abbildung 6.9: Fehler ohne Berücksichtigung der z-Achse

Der erste offensichtliche Unterschied zum vorherigen Bewertungsansatz ist ein deutlich geringerer Gesamtfehler für alle ersetzten Konfigurationen. Der Fehler liegt mit Ausnahme der 16 Bit Festkomma-Implementierung des Kalman-Filters deutlich unter 2 Grad. Tabelle 6.2 zeigt den durchschnittlichen Fehler für alle Fehler aus 15 Messungen.

Die in Tabelle 6.2 dargestellten Ergebnisse zeigen, dass es keinen bedeutenden Unterschied zwischen den verwendeten Sensorfusionsalgorithmen und Datenformaten gibt. Interessanterweise zeigen die Ergebnisse der 32 Bit Festkomma-Implementierung sogar

Datenformat	Kalman [°]	Madgwick [°]	Mahony [°]	Komplementär [°]
32 bit Fließkomma	1.557	1.657	1.588	1.489
32 bit Festkomma	1.562	1.603	1.557	1.478
16 bit Festkomma	2.429	1.722	1.626	1.539

Tabelle 6.2: Durchschnittlicher Fehler in Grad

einen etwas geringeren Fehler im Vergleich zur Fließkomma-Implementierung. Auch die 16 Bit Festkomma-Variante weist außer beim Kalman-Filter keine signifikant höheren Fehler auf.

Quantisierungsfehler

Die Ergebnisse der allgemeinen Bewertung der Unterschiede zwischen den verwendeten Filtern und Datenformaten zeigen zwei interessante Ergebnisse, die in diesem Abschnitt diskutiert werden.

Die erste Erkenntnis ist, dass die Ergebnisse der Sensorfusion unter Verwendung von 32 Bit Festkommadatentypen einen etwas geringeren Fehler aufwiesen als ihr Fließkomma-Pendant. Dieses Artefakt lässt sich durch das Zusammenspiel zweier Faktoren erklären. Die Daten innerhalb eines Quaternions werden immer auf eine Länge von eins normiert, was den Bereich der möglichen Werte einschränkt. Bei den untersuchten Filtern konnte der Radixpunkt der Daten für den ganzzahligen Teil auf 7 Bits und für den gebrochenen Teil auf 24 Bits gesetzt werden. Verglichen mit der Standard-Fließkomma-Implementierung, die 23 Bit für die Mantisse bereithält, hat die 32 Bit Festkomma-Implementierung eine um 1 Bit höhere Genauigkeit.

Die zweite Erkenntnis ist die offensichtlich geringere Qualität des Ergebnisses wann das Kalman-Filter die 16 Bit Festkomma-Implementierung verwendet. Aufgrund der Art und Weise, wie das Kalman-Filter intern arbeitet, benötigt es im Vergleich zu den anderen drei Filtern einen größeren Bereich von Werten. Für das Madgwick-Filter, das Mahony-Filter und das Komplementärfilter reichten 3 Bit für den ganzzahligen Teil und 12 Bit für Nachkommateil aus um den Bereich der möglichen Werte abzudecken. Das Kalman-Filter benötigt jedoch die 7 Bit für den ganzzahligen Teil, die bereits in der 32 Bit Festkomma-Implementierung verwendet werden. Daher bleiben nur 8 Bit für den Nachkommateil übrig. Das Ergebnis dieser reduzierten Genauigkeit ist in Abbildung 6.10 zu sehen.

A und **B** zeigen das Kalman-Filter mit einer 32 Bit Fließkomma- bzw. einer 32 Bit Festkomma-Implementierung. Es gibt keinen sichtbaren Unterschied zwischen den beiden Ergebnissen. Bei den in **C** gezeigten 16 Bit Festkommadaten sind jedoch deutliche Quantisierungsfehler zu erkennen. Diese sind auf die geringere Genauigkeit zurückzuführen und führen zu einem höheren Gesamtfehler. Zum Vergleich zeigt **D** die Ergebnisse für das Mahony-Filter mit 16 Bit Festkommadaten, bei denen keine Quantisierungsfehler

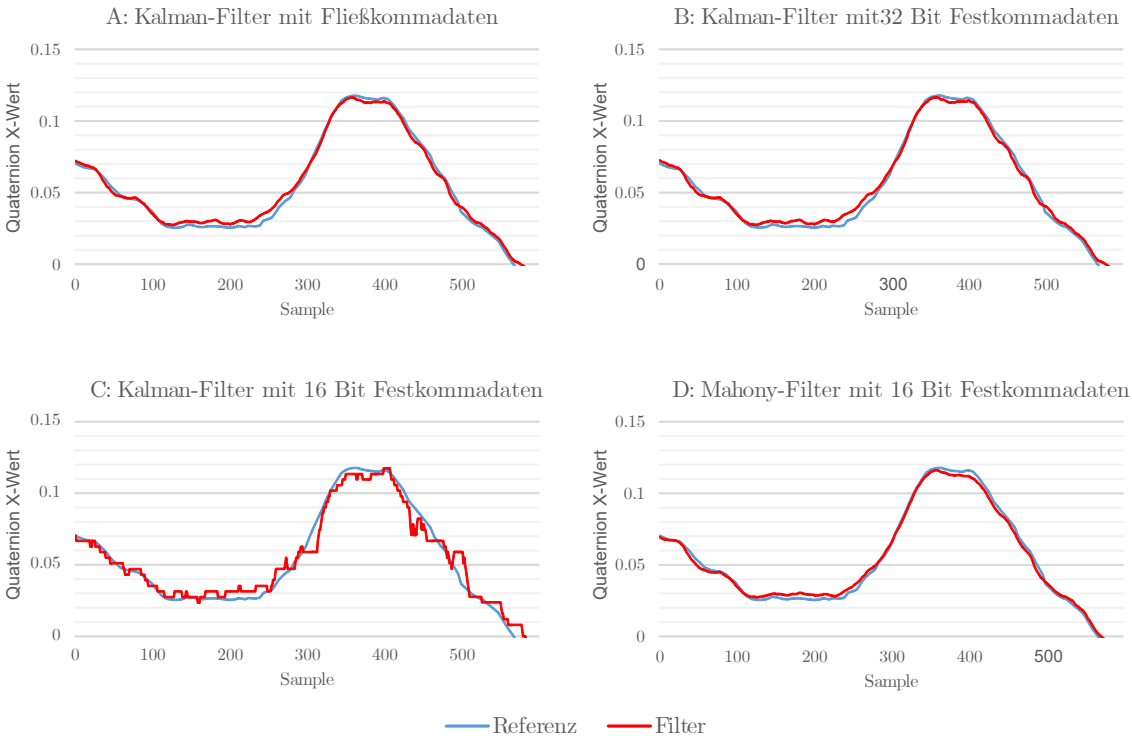


Abbildung 6.10: Vergleich der Fusion durch das Kalman-Filter für die drei Datenformate. **A:** Ausgabe des Kalman-Filters mit 32 Bit Fließkommadaten. **B:** Ausgabe des Kalman-Filters mit 32 Bit Festkommadaten. **C:** Ausgabe des Kalman-Filters mit 16 Bit Festkommadaten. **D:** Ausgabe des Mahony-Filters mit 16 Bit Festkommadaten.

sichtbar sind. Die Quantisierungsfehler sind auch der Grund dafür, dass das Kalman-Filter vor der Korrektur der z-Achse einen deutlich höheren Fehler aufweist. Ohne die Magnetometerdaten wird die z-Achse nur durch die Daten des Gyroskops bestimmt. Aufgrund der geringen Genauigkeit werden alle Winkelraten die kleiner als 90°/s sind auf 0 gerundet. Bei den durchgeführten Messungen überstieg die Winkelgeschwindigkeit diesen Wert nur selten so, dass die z-Achse keine Rotationsänderung zeigte.

6.1.5.2 Statistische Analyse

Um die statistische Signifikanz der Ergebnisse zu zeigen und den Einfluss der einzelnen Parameter auf die Ergebnisse zu untersuchen, wurde eine statistische Analyse durchgeführt. Diese Analyse bedient sich ähnlich wie in Unterabschnitt 5.4.6 der Gage R&R-Analyse und der Bestimmung des Konfidenzintervalls. Da diese Untersuchungen sich ausschließlich auf die funktionalen Eigenschaften beziehen und diese nicht so stark vom

SiL-Konzept beeinflusst werden, sind sie an dieser Stelle nicht näher ausgeführt. In der Veröffentlichung [B9] können diese Untersuchungen nachvollzogen werden.

6.1.6 Zusammenfassung und Bewertung der Ergebnisse

In diesem Abschnitt der Arbeit wurden verschieden Sensorfusionsalgorithmen auf ihre funktionalen und extrafunktionalen Eigenschaften untersucht. Dabei wurde zusätzlich der Einfluss des verwendeten Datenformates für die Implementierung betrachtet. Es wurde gezeigt, dass die Verwendung von Festkomma-Arithmetik den Rechenaufwand der untersuchten Algorithmen um etwa 50 % für 32 Bit Festkommadaten und um etwa 80 % für 16 Bit Festkommadaten im Vergleich zu einer Standardimplementierung mit 32 Bit Fließkommadaten reduzieren kann.

Die Verwendung einer 32 Bit Festkomma-Implementierung hat keine negativen Auswirkungen auf das Ergebnis der Sensorfusionsalgorithmen. Selbst die 16 Bit Festkomma-Implementierungen lieferten brauchbare Ergebnisse und können ebenfalls verwendet werden, wenn auch mit einer geringfügigen Verringerung der Genauigkeit.

Eine Ausnahme bildet das Kalman-Filter, welches unter Nutzung der 16 Bit Festkomma-Implementierung schlechtere Ergebnisse lieferte und durchweg etwa die dreifache Berechnungszeit benötigt. Daher ist dieses Filter für eine hardwarebeschränkte Umgebung nicht zu empfehlen. Wie auch in anderen Arbeiten [A85]–[A87] lassen die Ergebnisse den Schluss zu, dass das Madgwick-Filter und das Mahony-Filter eine ähnliche Qualität der Ergebnisse liefern.

Die gerade die extrafunktionalen Eigenschaften wie die Speicherauslastung und die Rechenzeit zeigen, dass in Ressourcen-beschränkten Systemen das Madgwick-Filter oder das Mahony-Filter die bessere Wahl sind. Diese Untersuchungen der extrafunktionalen Eigenschaften konnten direkt auf dem zu untersuchenden System durchgeführt werden. Mithilfe der SiL-Implementierung konnten die Algorithmen direkt auf dem intelligenten Sensor implementiert und untersucht werden. Aufgrund der Eigenschaften dieses Systems ist trotzdem ein reproduzierbares Testen der Algorithmen mit den gleichen Testdaten für jede einzelne Implementierung möglich. Dies ermöglicht einen besseren Vergleich der Algorithmen, speziell in Bezug auf Ihre Eignung in Ressourcen-beschränkten Systemen wie intelligenten Sensoren.

6.2 Verwenden künstlich erzeugter oder überlagerter Stimuli

Im letzten Unterkapitel wurden Sensorfusionsalgorithmen auf ihre funktionalen und extrafunktionalen Eigenschaften hin untersucht. Dabei wurde das Sensor-in-the-Loop-Konzept in einem realen Anwendungsfall eingesetzt, um diese Untersuchung vergleichbar und reproduzierbar durchzuführen. Besonders die Untersuchung der extrafunktionalen Eigenschaften dieser Algorithmen wurde durch die Verwendung vom SiL-Ansatz ermöglicht.

Um den Einsatzbereich des vorgestellten Ansatzes zu erweitern wurden zusätzliche Funktionen entwickelt und in das bestehende System integriert. Diese Erweiterungen erhöhen den Nutzen des Systems für den Anwender und erweitern den Einsatzbereich der Anwendung.

Dieses Unterkapitel stellt die erste dieser Erweiterung vor. Bisher wurden für das SiL-Konzept Sensordaten aufgezeichnet, um sie in späteren Testläufen wieder in den zu testenden Sensor einzuspeisen. Die nachfolgend vorgestellte Erweiterung ermöglicht es diese aufgezeichneten Sensordaten auf komfortable Weise zu manipulieren. Darüber hinaus können auch künstliche Sensordaten und Stimuli für die Injektion während verschiedener Testläufe erzeugt werden.

6.2.1 Einleitung

Durch die zunehmende Leistungsfähigkeit von eingebetteten Systemen und insbesondere von Inertialsensor-Subsystemen ist es möglich, viele Aufgaben wie Vorverarbeitung, Sensordatenfusion oder Gestenerkennung auf die Sensorhardware zu verlagern und damit den Stromverbrauch des Gesamtsystems zu senken. Insbesondere intelligente Sensoren mit einem programmierbaren Mikroprozessor sind für solche Aufgaben prädestiniert. Allerdings wird auch die Entwicklung und der Test der Firmware auf einem solchen Sensor-Subsystem immer komplexer und zeitaufwändiger. Für die Aufgabe, die Software für diese Sensoren zu entwerfen und zu evaluieren, wurde in dieser Arbeit die SiL-Architektur vorgeschlagen. Sie ermöglicht es einem Entwickler, zuvor aufgezeichnete Sensordaten zurück in den intelligenten Sensor zu injizieren, um seine Firmware auf der realen Hardware zur Laufzeit zu testen und zu bewerten. Dieser Ansatz kombiniert die Vorteile des Testens auf echter Hardware mit wiederholbaren und reproduzierbaren Ergebnissen.

Ein wichtiger Aspekt für den Entwicklungsprozess ist die Verfügbarkeit und Vielfalt von Sensordatenaufzeichnungen. Besonders für sicherheitskritische Szenarien, wie sie in ISO13849 [A108] und IEC61508 [A109] beschrieben sind, ist es wichtig, die Firmware für

möglichst viele verschiedene Szenarien und Fehlerfälle zu testen. Insbesondere bei der Arbeit mit Inertialsensoren ist es jedoch nicht möglich, Daten für alle denkbaren Szenarien zu erfassen. Daher wird in diesem Abschnitt ein modulares Framework zur Datenerweiterung vorgestellt. Dieses ermöglicht es eine Vielzahl von Testdaten aus aufgezeichneten oder künstlichen Daten zu erzeugen. Es können Messungen kombiniert, die Sensordaten auf Grundlage eines Fehlermodells künstlich verschlechtert oder Vorverarbeitungsschritte an den Sensordaten durchgeführt werden. In Kombination mit der SiL-Architektur schafft das vorgeschlagene Framework ein leistungsfähiges Werkzeug, um Sensorfusionsalgorithmen zu implementieren, Gestenerkennungsalgorithmen zu entwerfen oder ein korrektes Verhalten des Systems in Fehlerfällen sicherzustellen. Die Implementierung des Frameworks ermöglicht es Sensordaten zu generieren, zu manipulieren und zu inspizieren. Dabei ist ein Neukompilieren von Code auf dem Host oder der Sensor-Firmware nicht erforderlich.

Um die Fähigkeiten des vorgeschlagenen Frameworks zu demonstrieren, werden in diesem Unterkapitel drei umfassende Beispiele vorgestellt. Diese decken verschiedene Anwendungsfälle ab. Es wird die Simulation gängiger MEMS-Sensorfehler, die Vorverarbeitung von Sensordaten oder die Generierung von Daten zum Entwurf und Test eines Gestenerkennungsalgorithmus erläutert. Inhalte dieses Kapitels wurden zuerst im Rahmen der *Embedded Systems Week* auf dem *Rapid System Prototyping* Workshop vorgestellt [B10][2].

6.2.2 Verwandte Arbeiten

Ein häufiges Problem in der Prototyping- und Entwicklungsphase eines Produkts besteht darin, dass für die Entwicklung und den Test eines Systems eine Vielzahl von Testdaten unterschiedlichster Sensoren erzeugt werden müssen. Oft ist die benötigte Ausrüstung sehr kostenintensiv oder nicht ohne weiteres verfügbar. Im Falle der Gestenerkennung wird möglicherweise eine sehr große Menge an Daten benötigt, um die Algorithmen zu trainieren und zu testen [A110].

Die Arbeiten [A111] und [A112] beschreiben, wie man Daten von Kameras und Inertialsensoren erweitert. Sie zeigen den Nutzen der Datenanreicherung für die Entwicklung und das Training von Gestenerkennungsalgorithmen. Beide Arbeiten zeigen wie die Daten augmentiert werden, sie schlagen aber kein Framework für eine einfache und dynamische Generierung augmentierter Daten vor.

Zwei etablierte und bekannte Toolboxen zur Generierung von Simulationsdaten für Inertialsensor- und Positionierungssysteme sind die GNSS-INS-SIM von *Aceinna* [A113]

[2] Diese Veröffentlichung entstand in enger Kooperation mit Herrn Nils Büscher. Eine Abgrenzung der einzelnen Beiträge befindet sich in Abschnitt 1.2. Aus Gründen der Lesbarkeit wird an dieser Stelle der komplette Ansatz vorgestellt.

und die Sensor Fusion and Tracking Toolbox für *MatLab* [A114]. Das Hauptziel dieser Toolboxen ist die Simulation von Szenarien mit dem Fokus auf Navigation in Kombination mit GPS- oder Automotive-Systemen. Beide konzentrieren sich auf die Generierung von künstlichen Daten für Inertialsensoren aus vordefinierten Trajektorien und Sensormodellen und nicht auf die Schaffung von Mitteln zur Datenerweiterung. Der Zweck des vorgeschlagenen Frameworks ist es, die Erweiterung und Verarbeitung vorhandener Sensordaten zu ermöglichen. Die so veränderten Daten sollen für die Auswertung direkt auf der Sensorhardware unter Verwendung der SiL-Implementierung verwenden werden können.

Da beide Frameworks auf die Erzeugung künstlicher Sensordaten abzielen kann es sinnvoll sein, sie mit dem vorgeschlagenen Framework zu kombinieren, um diese um die Möglichkeiten zur schnellen Erzeugung von Testdaten zu erweitern. Eine Kombination mit anderen Arbeiten wie den in [A115] oder [A116] vorgestellten Frameworks ist ebenfalls möglich. Dabei werden die allgemeinen Sensordaten mit den genannten Ansätzen generiert um schnell über verschiedene Szenarien und Alternativen der Daten mit dem vorgeschlagenen Framework zu iterieren.

6.2.3 Konzept

Bei der Datenerweiterung besteht das primäre Ziel des Frameworks darin, Testfälle für eine breite Palette von Szenarien und Konfigurationen auf der Grundlage zuvor aufgezeichneter oder generierter Sensordaten zu generieren. Die Durchführung der Datenerweiterung auf dem intelligenten Sensor selbst würde nicht nur die möglichen Manipulationsschritte aufgrund der begrenzten Ressourcen der SPU einschränken, sondern auch das untersuchte Verhalten des intelligenten Sensors beeinflussen oder verzerren. Dies gilt insbesondere für die Berechnungszeit, den Speicherbedarf und den Stromverbrauch. Daher sollte das Framework auf einem Host-Computer laufen und zwei Anforderungen erfüllen:

1. Das Framework sollte in der Lage sein, Testfalldaten aus zuvor aufgezeichneten Sensordaten oder künstlichen Signalen zu generieren.
2. Die Generierung der Testfälle muss mit gestreamten Live-Daten nutzbar sein d.h. das Framework sollte in der Lage sein, dass Ergebnis direkt aus neuen Datenproben zu generieren, ohne das ein Zugriff auf den gesamten Datensatz erforderlich ist oder eine erhebliche Verzögerung bei der Verarbeitung der Daten entsteht.

Daher wurde ein flussbasierter Ansatz ähnlich einem Pipeline- und Filterentwurfsmuster gewählt, der beide Anwendungsfälle ermöglicht. Das Framework ist so konzipiert, dass es aus einer Vielzahl von Komponenten besteht. Diese werden als Funktionsblöcke (FBs) bezeichnet und implementieren jeweils einen bestimmten Verarbeitungsschritt der

Sensordaten. Diese Funktionsblöcke sind miteinander verbunden, um eine Pipeline von Manipulationsschritten zu bilden. Jeder FB kann einen oder mehrere Eingänge und Ausgänge haben. Darüber hinaus sollte jeder Ausgang mit mehreren Eingängen von nachfolgenden Funktionsblöcken verbunden werden können. Um unnötige Berechnungen und Overhead zu vermeiden, werden neue Daten zu einem bestimmten Zeitpunkt von den Sensorausgängen des Frameworks angefordert. Diese Kommunikation wird im Detail im Unterabschnitt 6.2.4 beschrieben.

6.2.3.1 Szenarios

Durch die Erweiterung der Sensor-in-the-Loop-Architektur mit diesem Framework sind eine Vielzahl von Szenarien für die Auswertung der Datenverarbeitung auf dem intelligenten Sensor möglich. Eine Kombination mit dem SiL-Ansatz ist jedoch nicht zwingend erforderlich. Das vorgeschlagene Framework sollte auch mit anderen State-of-the-Art-Ansätzen wie Hardware-in-the-Loop (HiL) oder simulationsbasierten Ansätzen verwendet werden können. Es sind allerdings nur in Kombination mit dem vorgestellten SiL-Konzept alle beschriebenen Szenarien realisierbar und somit das volle Potenzial des Frameworks nutzbar.

Abbildung 6.11 zeigt verschiedene Szenarien, in denen das vorgeschlagene Framework in Kombination mit der SiL-Plattform verwendet werden kann.

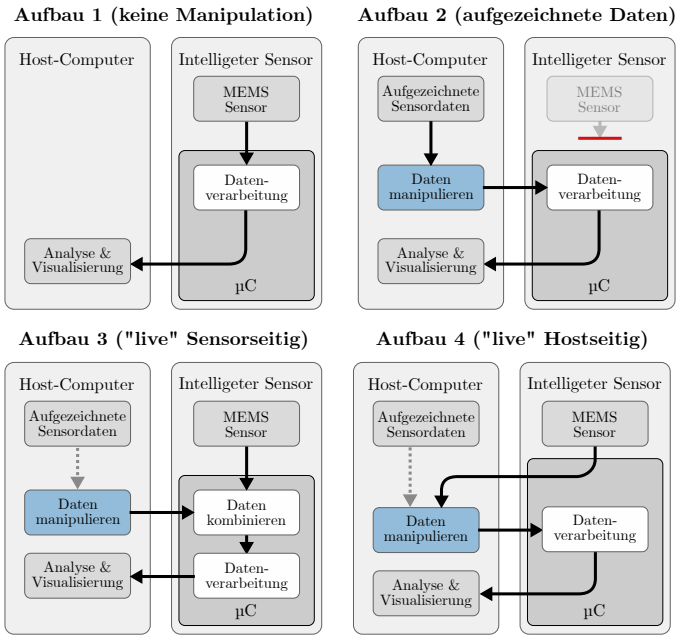


Abbildung 6.11: Mögliche Szenarien für die Evaluation von Sensor-Firmware.

Aufbau 1 stellt den Standardfall ohne jegliche Manipulation der Sensordaten dar. Es wird eine Auswertung auf dem Host-Computer vorgenommen, bei der die Daten entweder empirisch oder anhand anderer Messungen, mit einer Referenz verglichen werden. Dieses Setup entspricht der normalen Aufzeichnungsphase des vorgestellten SiL-Ansatzes.

In *Aufbau 2* wurden die Daten vom Sensor aufgezeichnet. Anschließend werden sie mit dem Framework auf der Host-Seite manipuliert. Die manipulierten Daten werden wieder in den Sensor eingespeist. Dort werden die Daten wie Rohdaten direkt vom Inertialsensor behandelt. Die verarbeiteten Daten werden dann an den Host zurückgesendet. Die Verwendung der aufgezeichneten Daten ermöglicht es, dieselben Daten mit unterschiedlichen Fehlern oder Merkmalen in vorhersehbarer und reproduzierbarer Weise erneut auszuwerten. Dieser Aufbau mit Umgehung des Sensorelementes entspricht der Wiederholungs- oder Injektionsphase des vorgestellten SiL-Ansatzes.

In *Aufbau 3* werden Live-Daten vom Sensor in Verbindung mit einem vom Framework künstlich erzeugten Signal verwendet. Die Daten aus dem Framework werden in den Sensor eingespeist, wo sie von der SPU mit den direkt vom Sensor kommenden Daten kombiniert werden. Die resultierenden Daten werden zur Analyse an den Host zurückgesendet.

Aufbau 4 arbeitet ähnlich wie Aufbau 3 mit Live-Daten vom Sensor, sendet aber die vom Sensor erfassten Daten an den Host. Der Host verarbeitet sie und leitet sie zur weiteren Verwendung zurück an den Sensor. Auch hier werden die verarbeiteten Daten zur Analyse an den Host zurückgeschickt.

Sowohl *Aufbau 3* als auch *Aufbau 4* ermöglichen die Manipulation und Analyse von Live-Sensordaten. Es ist jedoch wichtig zu beachten, dass beide Setups wahrscheinlich nicht echtzeitfähig im engeren Sinne sind. Vor allem die Kommunikationsschnittstelle und das Scheduling auf dem Host-Computer können zu einer erheblichen Verzögerung führen. Die Analyse der Ergebnisse wird jedoch nicht durch die zusätzliche Latenzzeit beeinflusst, wenn Zeitstempel für jedes Sensorsample verwendet werden.

6.2.3.2 Architektur

Die Architektur des Frameworks ist so konzipiert, dass es verwendet werden kann, ohne dass Code auf dem Host-Computer oder dem zu testenden intelligenten Inertialsensor kompiliert werden muss. Daher besteht das Framework aus einer Sammlung vordefinierter Funktionsblöcke. Diese werden zur Laufzeit in einer Pipeline miteinander verbunden und können eine Vielzahl von Manipulationen der Sensordaten ermöglichen.

Das Konzept der Verbindung einzelner FBs ist in Abbildung 6.12 dargestellt. Nach der Instanziierung der Funktionsblöcke kann ein Ausgang eines FBs mit einem Eingang eines anderen FBs verbunden werden, indem er als dessen Datenquelle registriert wird. Ein

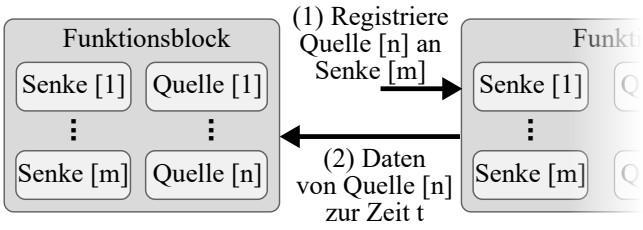


Abbildung 6.12: Verbindung der einzelnen Funktionsblöcke in einer Pipelinestruktur

FB kann mehrere Eingänge haben, aber jeder Eingang kann nur einer Datenquelle zugeordnet sein. Ein Ausgang eines FBs kann jedoch als Datenquelle für mehrere Eingänge dienen.

6.2.3.3 Funktionalitäten

Um eine breite Palette von Möglichkeiten zur Augmentierung der Sensordaten und zur Generierung von Testfällen zu bieten, wurde eine Vielzahl von Funktionsblöcken implementiert. Ihre Funktionalität reicht von einfachen mathematischen Operationen über Speicher bis hin zu logischen Operationen und digitalen Filtern. Die folgenden Typen von Funktionsblöcken sind verfügbar:

- *Eingangs-Blöcke* für Sensordaten, Konstanten, Vektoren, Matrizen und künstliche Signale (Rauschen, Sinuskurve, Rechteckfunktion, etc.)
- *Mathematik-Blöcke* zur Durchführung von arithmetischen Operationen, sowie zum Integrieren, Differenzieren und Runden von Werten.
- *Logik-Blöcke*, um Werte zu vergleichen oder Daten von mehreren Eingängen in Abhängigkeit von einem Eingangswert auszuwählen (Multiplexer).
- *Manipulations-Blöcke* zur Aufteilung von Vektoren in Einzelwerte, Kombination von Einzelwerten zu einem Vektor, Verzögerung von Signaldaten, Sättigung eines Wertes oder Wiederholung von Sensordaten.
- *Spezialisierte Blöcke* für allgemeine Operationen bei der Arbeit mit Sensordaten. Dazu gehören FIR- und IIR-Filter, Quaternion-Operationen, Sensorfusion (Madgwick- und Mahony-Filter), Vektordrehung oder Fehlausrichtung von Sensorachsen.
- *Ausgabe-Blöcke*, um die geänderten Sensordaten zurückzugeben.

Die oben erwähnten Typen von FBs erlauben eine breite Palette von Möglichkeiten, die Sensordaten zu ergänzen und zu kombinieren.

6.2.4 Implementierung

Das vorgeschlagene Framework zur Datenerweiterung wurde in Java als Add-on für die Eclipse IDE implementiert. So wird eine nahtlose Integration in die bisherige Sensor-in-the-Loop-Implementierung ermöglicht. Die Implementierung ist in zwei Einheiten unterteilt. Eine Einheit implementiert die Gesamtfunktionalität und den Datenfluss des vorgeschlagenen Frameworks. Die zweite Einheit ist eine grafische Benutzeroberfläche zur Erstellung und Überprüfung der erweiterten Daten zur Laufzeit.

6.2.4.1 Kontrollfluss der Funktionsblöcke

Die Funktionsblöcke sind so konzipiert, dass sie aus einem gegebenen Satz von Eingabedaten eine Ausgabe für eine bestimmte Operation erzeugen. Wenn eine neue Datenprobe angefordert wird, prüft der FB, ob das Ergebnis für diesen Zeitstempel bereits berechnet wurde. Wenn dies der Fall ist, wird das Ergebnis zurückgegeben. Andernfalls ist der FB selbst an der Reihe. Er fordert die Daten für die Berechnung von seinen Eingängen an und gibt die Ergebnisse zurück, sobald die Berechnung abgeschlossen ist. Der gesamte Datenfluss ist in Abbildung 6.13 dargestellt.

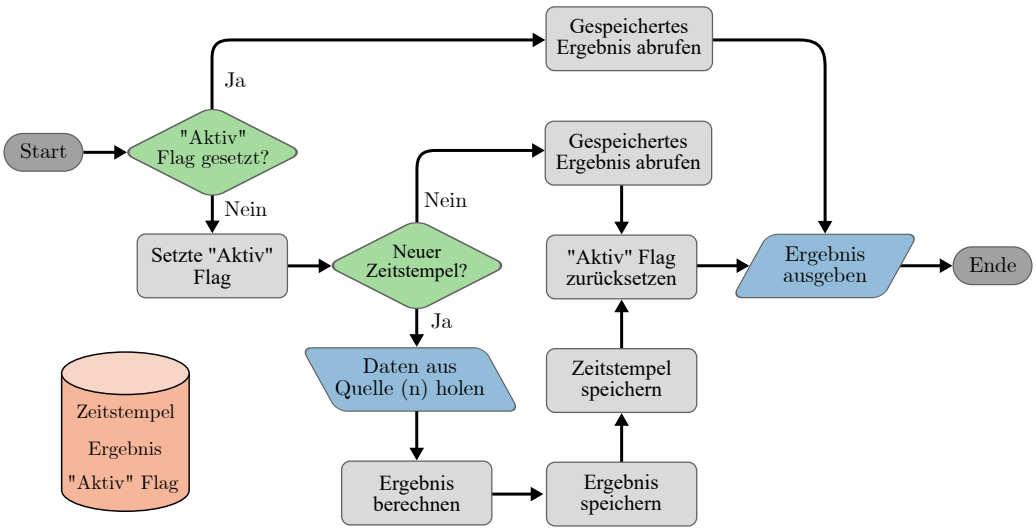


Abbildung 6.13: Kontrollfluss für das Aktualisieren eines FBs

Da beim Verbinden der FBs auch Schleifen gebildet werden können, wurde ein Save-Guard implementiert der Endlosschleifen verhindert. Wenn Daten vom FB angefordert werden, während er selbst auf angeforderte Daten wartet, gibt er direkt das Ergebnis der vorherigen Iteration zurück. Durch die Bildung von Schleifen wird also implizit ein Äquivalent zu einem temporären Speicher geschaffen. Dies ermöglicht eine noch größere

Bandbreite an möglichen Datenmanipulationen z. B. die Implementierung eines rekursiven Tiefpassfilters wie in Gleichung 6.12 dargestellt.

$$xOut_{(t)} = (1 - \alpha) \cdot xIn_{(t)} + \alpha \cdot xOut_{(t-1)} \tag{6.12}$$

6.2.4.2 Datenverteilung

Wie im Konzept kurz beschrieben, verwendet das vorgeschlagene Framework einen flussbasierten Ansatz mit funktionalen Komponenten und einer anforderungsbasierten Datenverteilung. Jedes Programm, welches das Framework zur Datenerweiterung nutzen möchte, muss die folgenden Schritte implementieren, um mit ihm zu interagieren:

1. Empfangen neuer Sensordaten vom Sensor.
2. Wenn der Sensor verwendet wird, müssen die empfangenen Sensordaten als Eingang des entsprechenden Sensors in die Manipulationseinrichtung eingefügt werden.
3. Abfrage der manipulierten Sensordaten vom Ausgangssensor der Manipulationseinrichtung, falls der Sensor verwendet wird.
4. Wenn das Manipulations-Setup einen Sensor als Ausgang hat, werden die manipulierten Daten verwendet, ansonsten die Originaldaten.

Die Verwaltung der Weitergabe der neuen Daten durch das Framework wird dezentral von den Funktionsblöcken übernommen. Als Beispiel wird in Abbildung 6.14 eine einfache Manipulation von Gyroskopdaten gezeigt.

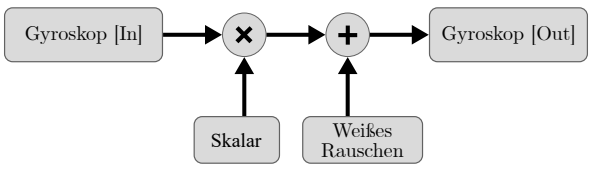


Abbildung 6.14: Beispiel Gyroskopdatenmanipulation

Die sich daraus ergebende Kommunikation zwischen den Funktionsblöcken ist für diesen Aufbau in Abbildung 6.15 dargestellt.

Die Anwendung fordert die manipulierten Daten von der Komponente *Gyroskop [Out]* an. Die Komponente *Gyroskop [Out]* hat die Komponente *Addieren* als Eingang und fordert daher die Daten von ihr an. Besagte Komponente *Addieren* hat zwei Eingänge registriert: eine Komponente *Multiplizieren* und einen Funktionsblock, der weißes Rauschen erzeugt. Daher fordert die Komponente *Addieren* die Ergebnisse von beiden

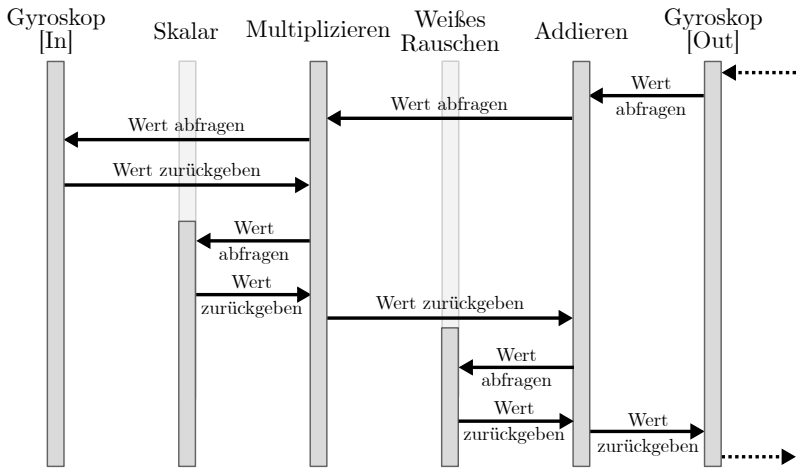


Abbildung 6.15: Datenfluss für das Beispiel aus Abbildung 6.14

Komponenten an, um die Summe zu berechnen. Diese Anforderungen werden sukzessiv weitergegeben bis sie eine Komponente erreichen, welche keinen Eingang hat.

6.2.4.3 Grafische Benutzeroberfläche

Damit ein Benutzer des Frameworks Testfälle für den intelligenten Sensor oder das Sensor-Subsystem erstellen kann, wurde eine grafische Benutzeroberfläche implementiert. Diese ermöglicht die Erstellung von Testfällen in Form eines *Node-Editors*. In diesem ist jeder Funktionsblock durch einen Knoten dargestellt. Es können Verbindungen zwischen kompatiblen Ein- und Ausgängen erstellt werden. Wie bei den Funktionsknoten vorgesehen, kann jeder Ausgang mit mehreren Eingängen verbunden werden. Allerdings kann jeder Eingang immer nur eine Datenquelle akzeptieren. Ein Beispiel für einen Testaufbau im *Node-Editor* ist in Abbildung 6.16 zu sehen.

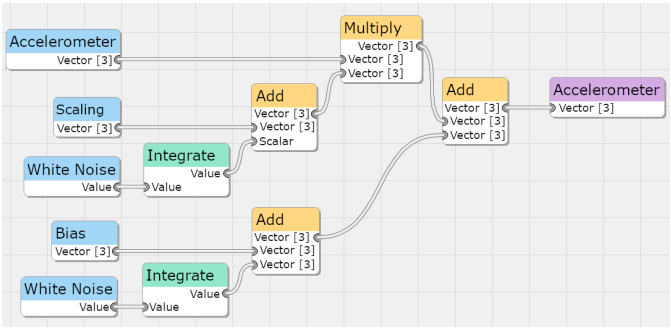


Abbildung 6.16: Beispiel Beschreibung für die Generierung von Offset und Skalierungsfehlern mit zusätzlicher Temperaturdrift, wie in Unterabschnitt 6.2.5.2 beschrieben.

Wenn die Einrichtung der Knoten und Verbindungen abgeschlossen ist, können die Funktionsblöcke und Verbindungen erzeugt und an einen *Listener* übermittelt werden. Der *Listener* kann dann das generierte Setup verwenden, um Sensordaten einzufügen und die modifizierten Sensordaten als Ergebnis zu erhalten. Der *Node-Editor* wurde dem SiL-Software-Framework als Plugin hinzugefügt. Die SiL-Plattform wurde so erweitert, dass sie die Einstellungen für die Bearbeitung von Sensordaten aus dem *Node-Editor* empfängt. So kann direkt visualisiert werden, wie das vorgeschlagene Framework einen zuvor aufgezeichneten Satz von Sensordaten manipuliert. Außerdem ist es möglich, die manipulierten Sensordaten wieder in den Sensor einzuspeisen, um eine umfassendere Bewertung der intelligenten Sensoren und der Sensor-Firmware zu ermöglichen. Dieser Schritt entspricht dem in Abbildung 6.11 gezeigten Aufbau 2.

In Abbildung 6.17 ist die Visualisierung der Originalsensordaten und der manipulierten Daten im Framework dargestellt.

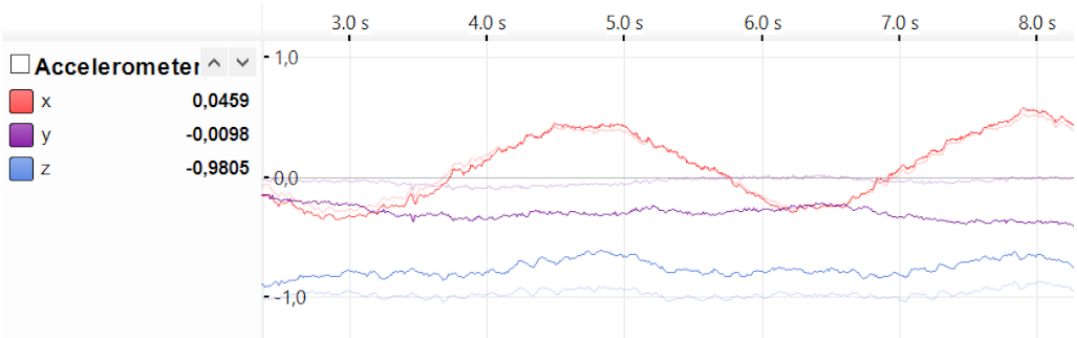


Abbildung 6.17: Screenshot aus dem Sensorframework zeigt, die Originalsensordaten (blass) und manipulierten Daten (kräftig)

Je nach Einstellung, ob die ursprünglichen oder die geänderten Daten in den Sensor eingespeist werden, werden diese unterschiedlich dargestellt. Dabei werden die nicht verwendeten Daten in blassen Farben angezeigt. Die zur Injektion ausgewählten Daten werden hingegen in kräftigen Farben dargestellt.

6.2.5 Beispiel Anwendungsfälle

In diesem Abschnitt werden mehrere Anwendungsfälle vorgestellt, die mit dem vorgeschlagenen Framework möglich sind. Das erste Beispiel befasst sich mit Rauschen, welches einen wichtigen Faktor bei der Aufzeichnung von Sensordaten darstellt. Die folgenden drei Beispiele zeigen gängige und mögliche Anwendungsfälle für ein breites Spektrum von Anwendungen.

6.2.5.1 Verschieden Arten von Rauschen

Ein wichtiger Faktor, der zu den Fehlern von Inertialsensoren insbesondere MEMS-Sensoren beiträgt, ist das Rauschen. Aufgrund der physikalischen Eigenschaften von MEMS-Sensoren ist dieses Rauschen oft ein rosa $(1/f)$ oder rotes $(1/f^2)$ Rauschen. Vor allem das rote Rauschen ist stark verbreitet. Es wird durch die brownsche Bewegung verursacht, z.B. im Zusammenhang mit der Temperatur (thermisches Rauschen). Das Framework ermöglicht die einfache Erzeugung von rosa und rotem Rauschen aus weißem Rauschen. Das rote Rauschen kann durch Integration von weißem Rauschen erzeugt werden [A117]. Rosa Rauschen kann durch Filterung des weißen Rauschens mit einem IIR-Filter [A118] ausreichend angenähert werden. Obwohl es sich nur um eine Annäherung handelt, wird in [A119] gezeigt, dass der Frequenzgang des IIR-Filters ausreicht, um das rosa Rauschen zu modellieren.

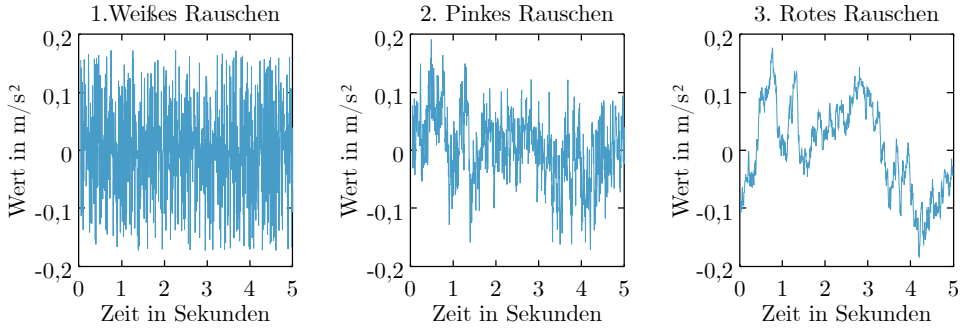


Abbildung 6.18: Verschiedene Arten des Rauschens generiert durch die FBs. (1) Weißes Rauschen, (2) Pinkes Rauschen über IIR Filter, (3) Rotes Rauschen über Integration

Wie in Abbildung 6.18 dargestellt, können die Funktionsblöcke des Frameworks alle drei vorgeschlagenen Arten von Rauschen generieren. Da sowohl ein IIR-Filter als auch die Integration von Werten als Funktionsblock verfügbar sind, kann die Erzeugung von $1/f$-Rauschen und $1/f^2$-Rauschen mit nur zwei FBs erreicht werden.

6.2.5.2 Fehlermodel für MEMS-basierte inertial Sensoren

Ein wichtiger Punkt für die Bewertung von Sensorfusionsalgorithmen, ist die Fähigkeit verschiedene Fehlereigenschaften des Inertialsensors zu modellieren. Es ermöglicht zu beurteilen, wie ein Algorithmus mit dem möglichen Fehlerspektrum umgeht. Die Verwendung realer Sensoren in Hardware-Prototypen erfordert eine sorgfältige Auswahl von Sensoren aus einer großen Auswahl, um das gesamte Spektrum möglicher Fehler abzudecken. Oft ist dies aus zeitlichen Gründen oder mangels geeigneter Sensorexemplare nicht möglich. Abgesehen vom Rauschen, das in Unterabschnitt 6.2.5.1 beschrieben wird, sind die häufigsten in Datenblättern aufgeführten Fehler [A120]:

- Bias
- Skalierungsfehler
- Temperaturdrift
- Nichtlinearität
- Ausrichtungsfehler
- Übersprechen von Achsen

Die *Bias* und der *Skalierungsfehler* können durch Addition eines konstanten Wertes zu den Sensordaten (Bias) oder durch Multiplikation mit einem konstanten Wert (Skalierungsfehler) modelliert werden. Die *Temperaturdrift* wirkt sich sowohl auf den *Bias* als auch auf den *Skalierungsfehler* aus. Sie kann durch Hinzufügen von rotem Rauschen zu beiden Skalaren modelliert werden. Abbildung 6.19 zeigt, wie diese beiden Fehler einschließlich der Temperaturdrift modelliert werden können.

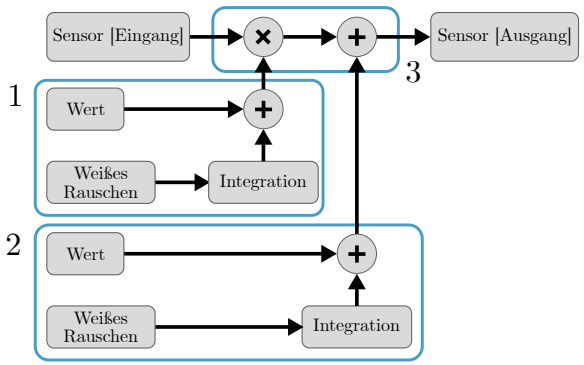


Abbildung 6.19: Funktionsblöcke zur Generierung von Bias und Skalierungsfehlern inklusive Temperaturdrift

Die Abschnitte **1** und **2** enthalten die Blöcke für die Erzeugung der Werte für Bias und Skalierung aus einem konstanten Vektor und einem integrierten weißen Rauschen. In **3** werden die Sensordaten manipuliert, indem sie zunächst mit dem Ergebnis aus **1** skaliert werden und anschließend der Bias aus **2** hinzugefügt wird.

Nichtlinearität kann den Sensordaten entweder durch Verwendung der vorhandenen mathematischen Funktionen oder durch Verwendung eines speziellen Funktionsblocks hinzugefügt werden. Dieser bildet einen Eingangswert auf einen Ausgangswert unter Verwendung einer kubischen Kurve ab. Ein Beispiel für eine nicht lineare Abbildung ist in Abbildung 6.20 dargestellt.

Die x-Achse in Abbildung 6.20 (1) ist der Eingangswert des Sensorsignals relativ zu seiner maximalen Amplitude. Wobei die y-Achse den relativen Ausgangswert beschreibt.

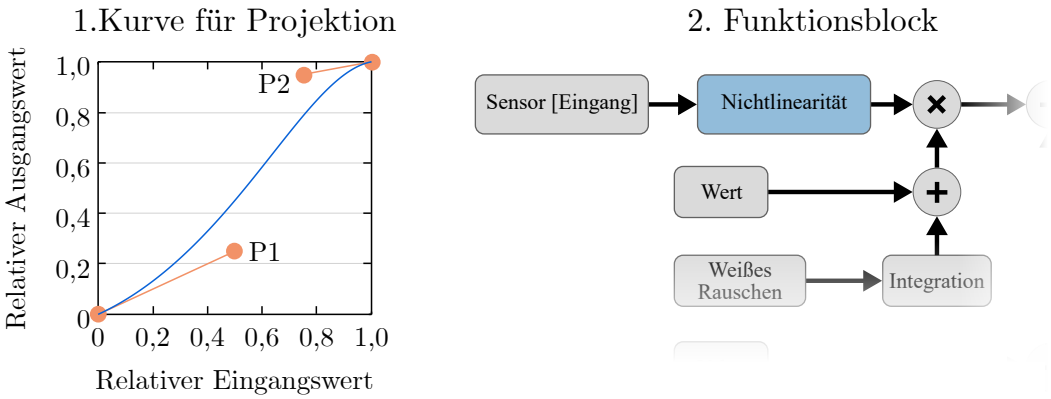


Abbildung 6.20: (1) Abbildungsfunktion für eine Nichtlinearität in Sensordaten. (2) Funktionsblock zum Hinzufügen einer Nichtlinearität zum Model aus Abbildung 6.19

Die Krümmung wird durch die beiden Kontrollpunkte *P1* bei $[0.5, 0.25]$ und *P2* bei $[0.75, 0.95]$ definiert

Der *Ausrichtungsfehler* kann in zwei Typen unterteilt werden. Der erste Typ ist der Gehäuseausrichtungsfehler und beschreibt die Fehlausrichtung zwischen Sensorachsen und Gehäuse. Die Abweichung vom idealen 90 °-Winkel zwischen zwei oder drei orthogonalen Achsen wird vom orthogonalen Ausrichtungsfehler beschrieben. Die erste Art kann durch Anwendung einer Vektordrehung hinzugefügt werden. Bei der zweiten Art muss jede Achse einzeln gedreht werden. Dies kann wiederum durch einen spezialisierten Funktionsblock erreicht werden. In Abbildung 6.21 wird der Ausrichtungsfehler des Gehäuses zum Sensorfehlermodell hinzugefügt.

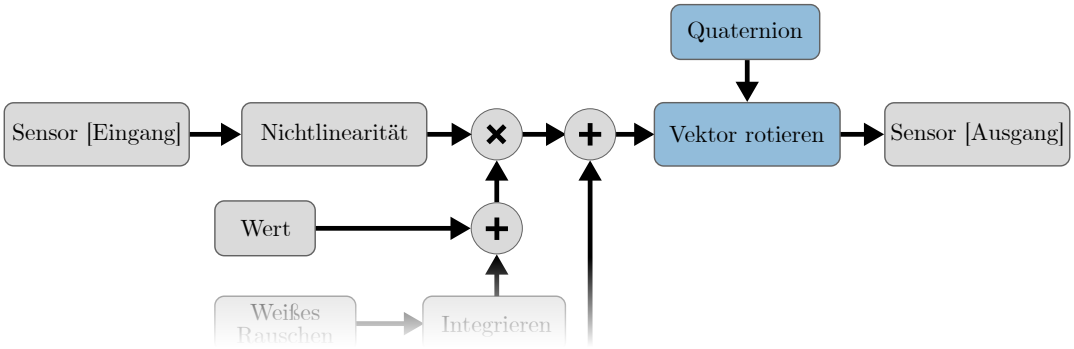


Abbildung 6.21: Hinzufügen eines Ausrichtungsfehlers zum Model aus Abbildung 6.20

Das *Übersprechen von Achsen* zeigt an, wie viel einer Beschleunigung von einem Sensor gemessen werden kann, der senkrecht zur Achse der besagten Beschleunigung steht.

Dieser Fehler ist eine Kombination aus mehreren Fehlern, einschließlich des Ausrichtungsfehlers [A121]. Die Achsenfehlausrichtung ist jedoch nicht der einzige Faktor, der zur Empfindlichkeit der Querachse beiträgt. Andere Ursachen können auch Ungenauigkeiten bei der Fertigung oder ein Übersprechen der Schaltkreise sein. Die nicht durch die Achsfehlstellung verursachten Fehler können durch Addition oder Subtraktion eines Bruchteils einer Achse von einer senkrechten Achse modelliert werden

Abbildung 6.22 stellt die Ergebnisse eines partiellen Fehlermodells für eine konstante und driftende Verzerrung in den Sensordaten dar. Die grüne Linie sind die ursprünglichen Sensordaten für die x-Achse des Beschleunigungssensors. Die blaue Linie zeigt die Sensordaten mit einer konstanten Verzerrung. In violette sind die Sensordaten mit einer zusätzlichen Temperaturdrift dargestellt. Die rote Linie bildet die Sensordaten bei einer Kombination der beiden vorgenannten Fehler vorhanden ab.

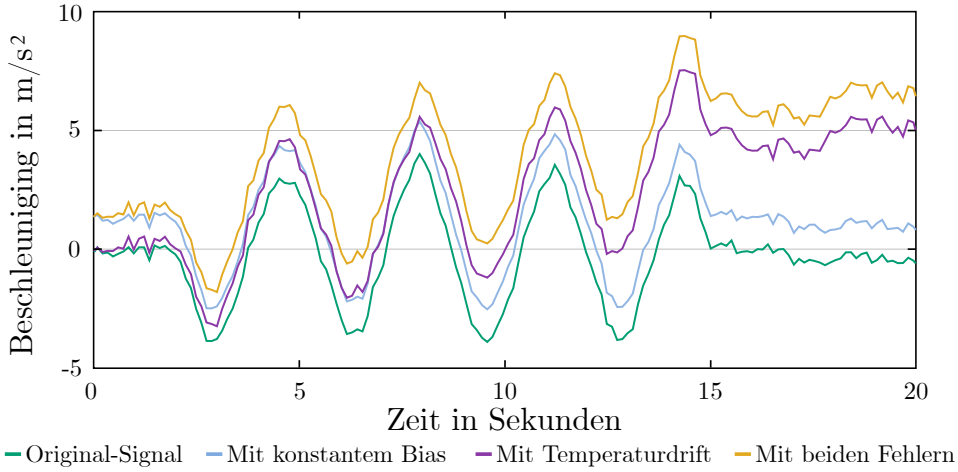


Abbildung 6.22: Sensordaten des Beschleunigungssensors für Bias, Temperaturdrift und kombinierten Fehler

Es ist zu erkennen wie die Fehler das Sensorsignal verschlechtern. Außerdem ist zu erkennen, dass der Fehler durch die Temperaturdrift mit der Zeit zunimmt und sich fortwährend ändert. Hier ist anzumerken, dass die Sensordaten bereits die Fehler des Sensors selbst enthalten und durch das Framework weiter verschlechtert werden. Die wahren Fehler für Bias und Temperaturdrift sind in der Realität wesentlich kleiner, in dem gezeigten Beispiel sind sie zur besseren Veranschaulichung höher gewählt.

6.2.6 Überlagerung von Gesten

Intelligente Sensoren werden auch häufig für Anwendungen verwendet, die Gesten als eine Form der Interaktion mit einem Benutzer nutzen. Häufig erfolgt die Gestenerkennung oder eine entsprechende Vorverarbeitung der Sensordaten direkt auf dem intelligenten

Sensor. Die Gestenerkennung muss mit einer großen Anzahl von Gestenaufzeichnungen getestet werden, um ihre Zuverlässigkeit zu gewährleisten. Außerdem wird bei der Verwendung von maschinellem Lernen eine große Anzahl von Gestenproben für das Training benötigt.

Da die Gesten bestenfalls unabhängig von der Situation, in der sich der Benutzer gerade befindet, erkannt werden sollen, ist es erforderlich die Gesten in einer Vielzahl von Szenarien aufzuzeichnen. Als Beispielszenarien wären hier Laufen, Fahrradfahren oder im Zug sitzen zu nennen. Um den Prozess der Aufzeichnung von Gesten für mehrere Szenarien zu beschleunigen werden oft Methoden zur Datenaugmetation verwendet, um mehr Daten zu generieren. Das Framework kann zu diesem Zweck verwendet werden, um Inertialsensordaten aus einem Szenario mit den Daten einer Geste zu überlagern. Diese sogenannte Superposition erlaubt es, unabhängig voneinander aufgezeichnete Gesten mit Szenarien zu kombinieren, um einen größeren Datensatz zum Trainieren und Testen der Gestenerkennung zu generieren. Die überlagerten Datensätze können anschließend direkt über das SiL-Interface des Sensors zum Trainieren oder Testen der Algorithmen genutzt werden. Ein allgemeiner Überblick über die Überlagerungsmethodik ist in Abbildung 6.23 abgebildet.

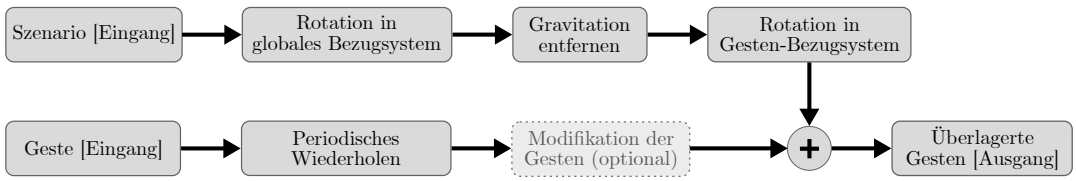


Abbildung 6.23: Funktionsblöcke für die Überlagerung der Sensordaten inklusive des Gravitationsvektors aus dem Szenario

Um die Gestendaten mit einem Szenario zu überlagern, müssen zwei wichtige Schritte durchgeführt werden. Normalerweise ist das aufgezeichnete Szenario viel länger als die Aufzeichnung einer Reihe von Gesten. Daher muss zunächst die aufgezeichnete Geste mehrfach wiederholt werden, um sie mit dem Szenario zu vermischen. Die Wiederholung der Geste ermöglicht es, sie in verschiedenen Situationen innerhalb des verwendeten Szenarios zu bewerten. Zweitens muss die Aufnahme des Szenarios so gedreht werden, dass ihr Koordinatenbezugssystem mit dem der aufgenommenen Geste übereinstimmt. Bei Messungen mit einem Beschleunigungssensor muss die Schwerkraft aus den Szenariodaten extrahiert werden. Dies stellt sicher, dass die Schwerkraft in den überlagerten Daten nicht doppelt vorhanden ist.

Die Wiederholung der Gesten erfolgt mit einer optionalen Verzögerung zwischen den einzelnen Iterationen. Die Leerzeit zwischen den einzelnen Iterationen kann entweder mit Nullen aufgefüllt werden, eine Interpolation zwischen Start und Ende oder eine Wiederholung der Werte der letzten Gestenprobe sein.

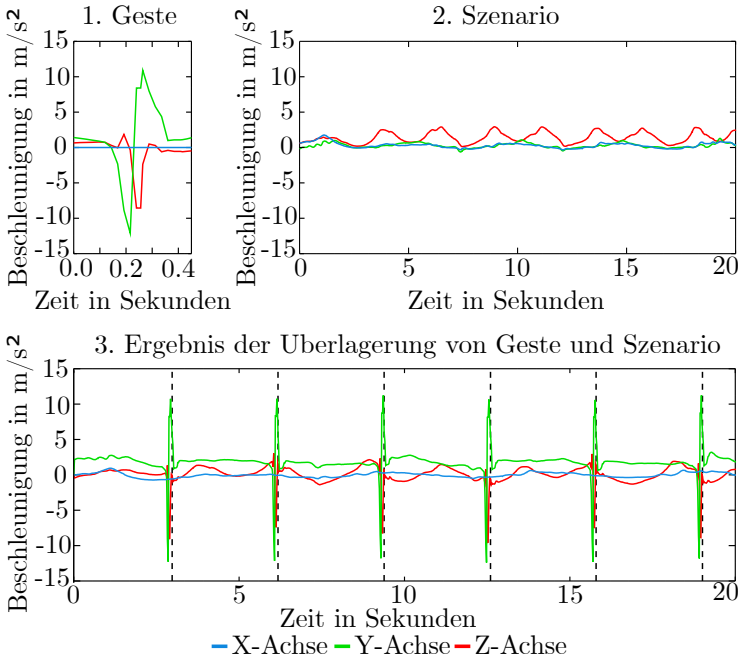


Abbildung 6.24: (1) Aufgenommene Geste, (2) Szenario (Gehen), (3) Ergebnis aus der Überlagerung

Abbildung 6.24 stellt die verwendeten Gestendaten, Szenariodaten und das resultierende Signal dar. Die verwendete Geste ist eine Wischbewegung mit einem Gerät. Alle Gestendaten werden nach 120 Abtastungen wiederholt. Für das Szenario wurde Sensordaten einer gehenden Person aufgezeichnet. Es ist zu erkennen, dass der Gestenerkennungsalgorithmus alle Gesten innerhalb des Szenarios korrekt erkennt. Nach der funktionalen Prüfung außerhalb des Sensors können dieselben Daten unter Verwendung des SiL-Ansatzes in den physikalischen Sensor injiziert werden. Dies ermöglicht das Testen der Gestenerkennung auf weitere funktionale und vor allem extrafunktionale Parameter. Die in diesem Beispiel verwendeten Methoden eignen sich daher hervorragend zum Trainieren und Testen von Gestenerkennungsalgorithmen.

6.2.6.1 Vorverarbeitung von Daten

Dieses letzte Beispiel zeigt wie das Framework verwendet werden kann, um die Verarbeitung auf ihre Tauglichkeit zu testen, bevor sie in die Sensor-Firmware implementiert wird. Gestenerkennungsalgorithmen verwenden zum Beispiel häufig Merkmale, die aus den rohen Sensordaten extrahiert werden, um daraus eine Geste zu erkennen [A122]. Zwei mögliche Methoden für eine solche Merkmalsextraktion sind in Abbildung 6.25 dargestellt, beide extrahieren das gleiche Merkmal.

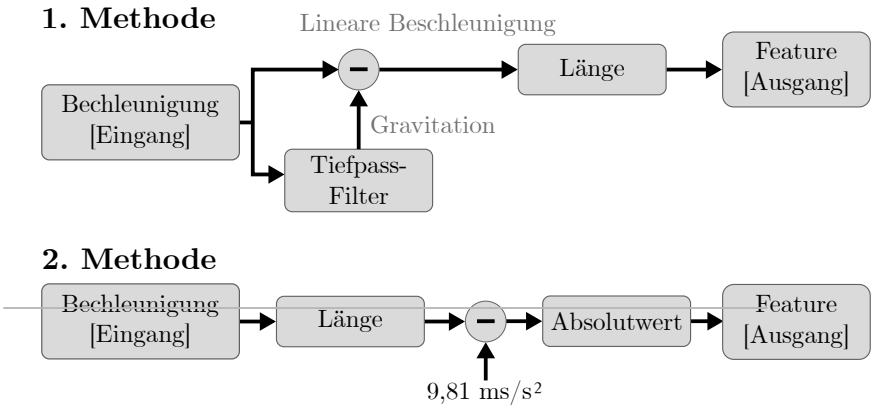


Abbildung 6.25: Setup zur Extraktion des Betrages der linearen Beschleunigung aus den Beschleunigungsdaten

Methode 1 aus Abbildung 6.25 filtert die Beschleunigungsmesserdaten mit einem Tiefpassfilter, um die Schwerkraft zu erhalten. Anschließend wird diese von den Beschleunigungsmesserdaten subtrahiert, um die lineare Beschleunigung zu bestimmen. Danach wird der Betrag des Beschleunigungsvektors berechnet, der als Merkmal für die Erkennung einer Schüttelgeste verwendet wird. *Methode 2* wird zunächst der Betrag des Vektors berechnet und dann ein konstanter Faktor für die Schwerkraft abgezogen. Der absolute Wert des Ergebnisses ist der Betrag der linearen Beschleunigung.

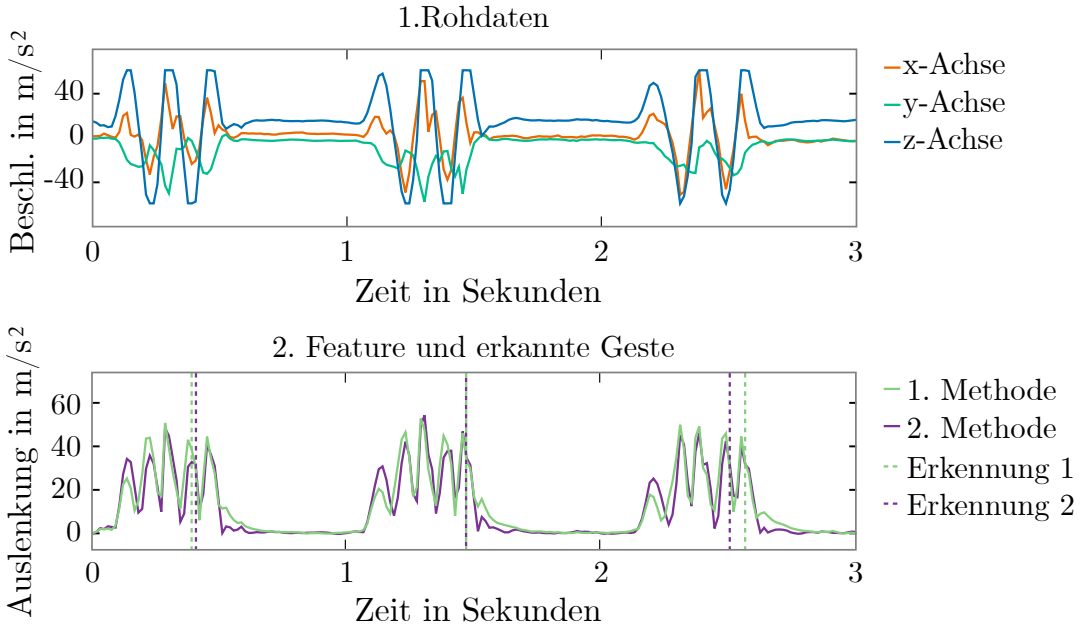


Abbildung 6.26: (1) Rohe Beschleunigungssensordaten eine Schüttelgeste (2) Vorverarbeitete Daten und detektierte Gesten mit beiden Methoden aus Abbildung 6.25

Abbildung 6.26 zeigt die aufgezeichneten Rohdaten des Beschleunigungsmessers. Die sich daraus ergebende Größe der linearen Beschleunigung, die mit beiden Methoden extrahiert wurde, ist im zweiten Diagramm dargestellt. Die vertikalen gestrichelten Linien zeigen an, wann der Algorithmus zur Erkennung der Schüttelgeste diese erkannt hat. Es ist zu erkennen, dass z. B. bei der ersten Methode die Erkennung des dritten Schüttelns später ausgelöst wird. Die Ergebnisse zeigen, wie verschiedene Verarbeitungsschritte sehr schnell evaluiert und verglichen werden können, bevor sie in die Sensor-Firmware implementiert werden.

6.2.7 Zusammenfassung und Bewertung

In diesem Unterkapitel wurde ein neuartige Frameworkerweiterung der SiL-Plattform zur dynamischen Erweiterung und Manipulation von Daten aus MEMS-Inertialsensoren vorgestellt. Sie unterstützt den Entwickler bei der Entwicklung und dem Testen der Firmware komplexer intelligenter Inertialsensorsysteme und hilft diese Vorgänge zu beschleunigen. Das Framework ist in der Lage die Daten für die Sensoren zur Laufzeit zu generieren und zu manipulieren ohne, dass der Code kompiliert werden muss. Als eine Erweiterung der Sensor-in-the-Loop-Prototyping-Plattform ist es ein leistungsfähiges Werkzeug zum Testen, Bewerten und Implementieren einer Vielzahl von Sensoraufgaben. Es erweitert dabei die vorgestellte SiL-Plattform um eine nützliche und sinnvolle Komponente. Durch die Kombination von komfortabel manipulierbaren Sensordaten mit dem SiL-Ansatz können noch schneller und reproduzierbar bestimmte Testszenarien auf der realen Sensorhardware durchgeführt. Die vorgestellten Beispiele zeigen, dass das Framework in der Lage ist ein breites Spektrum von Anwendungsfällen abzudecken. Dieses Spektrum reicht von der einfachen Sensorfehlersimulation über die Generierung von Daten für die Gestenerkennung bis hin zum Prototyping von Verarbeitungsschritten.

6.3 Energiebewusste Entwicklung von Sensorfirmware

Im letzten Unterkapitel wurde das SiL-Konzept um eine Komponente erweitert, die es ermöglicht, die zu untersuchenden Sensordaten zu manipulieren. Durch die Manipulation der Sensordaten ergibt sich die Möglichkeit mittels SiL bestimmte Testfälle zu generieren und zu testen, welche sonst nur schwer zu produzieren wären. Vor allem das Generieren von Grenzfälle und bestimmte Sensorfehlern erleichtern das Testen und machen es wesentlich effizienter.

Um den Komfort und die Effizienz beim Entwickeln und Testen von Firmware für intelligente Sensoren weiter zu steigern, wurde eine zusätzliche Erweiterung des SiL-Ansatzes entwickelt. Diese Erweiterung zielt auf die extrafunktionale Eigenschaft des Energieverbrauchs von intelligenten Sensoren ab. Gerade in mobilen Systemen, die mit Akkus oder Kondensatoren betrieben werden, spielt der Energieverbrauch eine entscheidende Rolle. Die in diesem Abschnitt vorgestellte Erweiterung unterstützt den Entwickler der Firmware bereits während der Entwicklung den Energieverbrauch zu berücksichtigen. Dies wird ermöglicht durch ein Energiemodell, welches bereits während des Testens und Debuggens Energieverbrauchswerte des Sensors relativ zu den Sensordaten zur Verfügung stellt. Diese Werte werden komfortabel im bereits bestehenden Framework visualisiert und mit den anderen Daten in Bezug gesetzt.

6.3.1 Einleitung

Bei der Arbeit mit eingebetteten Systemen ist sehr oft der maximale Energieverbrauch des verwendeten Systems begrenzt oder soll so gering wie möglich sein, um die Laufzeit des Systems zu maximieren [A123], [A124]. Der Gesamtverbrauch des Systems hängt nicht nur von der verwendeten Hardware ab, sondern auch von der Software, die auf dem eingebetteten System läuft [A125]. Wenn das System zum Beispiel regelmäßig aufwacht und komplexe Berechnungen durchführt, ist der Stromverbrauch relativ hoch. Daher ist es wichtig, über Mittel zu verfügen, mit denen der Stromverbrauch eines Systems leicht gemessen werden kann. Bei der Arbeit mit eingebetteten Systemen und insbesondere bei der Arbeit mit intelligenten Sensoren kann die Erstellung von Energieverbrauchsprofilen jedoch äußerst kompliziert sein. Wenn sich beispielsweise der Zustand des Systems in Abhängigkeit von Unterbrechungen durch die Sensoren ändern kann, führt diese Datenabhängig zu einem schwer vorhersagbaren Energieverbrauch. Weiterhin hängt oft die Komplexität der Berechnung von den Daten der Sensoren ab. Dies kann die Erstellung von Profilen zum Energieverbrauch sehr komplex und zeitaufwändig machen. In dieser Arbeit wurde die Debugging-Plattform SiL entwickelt, die es einem Entwickler ermöglicht zuvor aufgezeichnete Sensordaten wieder in den Sensor zu injizieren, um eine wiederholbare und reproduzierbare Möglichkeit zur Bewertung der Firmware auf dem intelligenten Sensor zu schaffen. In diesem Kapitel wird eine Erweiterung der SiL-Plattform vorgestellt,

welche es ihr ermöglicht über die Schätzung des Energieverbrauchs eine Leistungsanalyse des Systems durchzuführen. Mit dieser vorgeschlagenen Erweiterung ermöglicht es die SiL-Plattform einem Entwickler, den Stromverbrauch der intelligenten Sensoren für ein bestimmtes Szenario anhand der zuvor aufgezeichneten und erneut eingespeisten Sensordaten einfach und vergleichbar abzuschätzen. Darüber hinaus ermöglicht die zusätzliche Fähigkeit zur Erstellung von Energieverbrauchsprofilen eine schnelle und gezielte Optimierung der Sensor-Firmware im Hinblick auf einen geringeren Stromverbrauch. Durch das erneute Einspeisen von zuvor aufgezeichneten Sensordaten können energieintensive Teile der Firmware identifiziert und optimiert werden. Mithilfe der zuvor aufgezeichneten Daten wird sichergestellt, dass ein veränderter Stromverbrauch durch Änderungen im Code und nicht durch Unterschiede in den Sensordaten verursacht wird. Darüber hinaus ist die vorgeschlagene Erweiterung der SiL-Architektur auch in der Lage, den Stromverbrauch während der Aufzeichnung von Live-Sensordaten zu schätzen.

Dieses Unterkapitel stellt die SiL-Erweiterung inklusive des dafür entwickelten Energiemodells vor. Dafür wird zunächst das Konzept und im Anschluss die darauf aufbauende Implementierung präsentiert. Anschließend werden Experimente mit einem realen intelligenten Sensor durchgeführt, um das erstellte Sensormodell zu konfigurieren. Weitere Experimente zeigen die Funktionsweise und die Genauigkeit des entwickelten Ansatzes an einem realen Sensor-Szenario. Das Unterkapitel schließt mit den Ergebnissen und der Bewertung des vorgestellten Ansatzes. Die Inhalte dieses Unterkapitels wurden erstmals im Oktober 2021 im *Micromachines* Journal vorgestellt [B11].

6.3.2 Verwandte Arbeiten

Derzeit existieren drei moderne Ansätze zur Durchführung einer Energieverbrauchsanalyse für ein System oder einen Prototyp [A124]. Alle drei Ansätze haben ihre eigenen Vorteile und Grenzen. In diesem Abschnitt wird ein neuartiger Ansatz vorgestellt, der darauf abzielt die Vorteile dieser drei Methoden zu kombiniert.

Die erste State-of-the-Art Methode besteht darin, einen an den Prototyp angeschlossenen Power-Debugger zu verwenden, um dessen Stromverbrauch zur Laufzeit zu messen [A126], [A127]. Der große Vorteil dieses Ansatzes besteht darin, dass der Stromverbrauch für die reale Hardware, inklusive der Ausführung der unveränderten Software gemessen wird und alle Auswirkungen einschließlich Änderungen des Stromverbrauchs durch Taktdrift, Temperaturänderungen oder andere Umwelteinflüsse angezeigt werden. Die Verwendung eines Power-Debuggers hat jedoch den Nachteil, dass die Messungen bei der Arbeit mit Live-Sensordaten nicht vollständig reproduzierbar sind. Außerdem kann der Power-Debugger den gemessenen Energieverbrauch beeinflussen, da er einen Overhead für die erforderliche Kommunikation z. B. über Joint Test Action Group (JTAG) hinzufügt und möglicherweise bestimmte Stromsparzustände der SPU aufgrund der Kommunikation verhindert.

Der zweite Ansatz ist die Verwendung einer Simulation für die Analyse des Energieverbrauchs [B12], [A128], [A129]. Mit diesem Ansatz kann die Analyse für alle Betriebszustände und in reproduzierbarer Weise durchgeführt werden. Für die Simulation kann die Analyse des Energieverbrauchs jedoch nicht in Echtzeit durchgeführt werden. Das Ergebnis hängt stark von der Genauigkeit des Modells ab, so existieren z. B. zyklenakkurate Modelle [A128], anweisungsakkurate Modelle [B12] oder komponentenbasiert Modelle, welche auf Systemebene arbeiten [A129]. Die Erstellung eines Simulationsmodells für den Energieverbrauch kann je nach Abstraktionsgrad des Modells sehr zeitaufwändig sein.

Der dritte Ansatz besteht darin, eine formale Analyse des Energieverbrauchs der kompilierten Software für den Mikrocontroller durchzuführen. In [A130] wurde beispielsweise der Stromverbrauch für jede Compiler-Anweisung bestimmt, um den Gesamtstromverbrauch zu ermitteln. Dieser Ansatz berücksichtigt auch die Auswirkungen von Caches, Cache-Misses oder Stalls. In [A131] wurde eine formale Analyse für einen 8-Bit-Mikrocontroller vorgeschlagen, die auch den Stromverbrauch der Peripheriegeräte und nicht nur den des Mikrocontrollers selbst berücksichtigt. Beide Methoden können besonders stromintensive Softwareteile aufdecken und den groben Gesamtstromverbrauch abschätzen. Die Analyse ist jedoch nicht datenabhängig, sodass sich diese Ansätze nicht für Systeme eignen, deren Zustand von den Eingangsdaten abhängt. Ein formaler Ansatz, der die Datenabhängigkeit einbezieht, wurde in [A132] vorgestellt, der den Worst-Case-Stromverbrauch auf Anweisungsebene schätzt. Oft ist jedoch der durchschnittliche Stromverbrauch relevanter als der Worst-Case-Stromverbrauch. Außerdem stellen die Autoren in [A132] fest, dass eine formale Analyse ganzer komplexer Programme sehr zeitaufwändig sein kann. Keine der oben genannten formalen Methoden ist in der Lage, auch den Stromverbrauch von zusätzlicher angeschlossener Hardware wie Inertialsensoren zu analysieren.

Häufig stellen die Hersteller von Mikrocontrollern, die in eingebetteten Systemen mit geringem Stromverbrauch eingesetzt werden, Tools oder Plugins für ihre Entwicklung zur Verfügung. Diese ermöglichen es den Entwicklern sich schon früh im Entwicklungsprozess einen Überblick über den Stromverbrauch zu verschaffen [A133], [A134]. Diese Tools verwenden in der Regel eine der oben genannten Methoden oder eine Kombination aus ihnen für die Abschätzung des Energieverbrauchs.

Der in der vorliegenden Arbeit vorgestellte Ansatz zielt darauf ab, die Vorteile der oben genannten Methoden zu kombinieren. Es wird eine Analyse in Echtzeit auf der realen Hardware mit einem zuvor erstellten Energiemodell der genannten Hardware ermöglicht. Dafür muss die Software nur minimal angepasst werden. Durch die Verwendung der SiL-Architektur werden so die Vorteile einer Echtzeit-Energieverbrauchsabschätzung auf der realen Hardware mit den Vorteilen von simulationsbasierten Ansätzen kombiniert, um reproduzierbare Ergebnisse zu ermöglichen. Der Nachteil des vorgeschlagenen Ansatzes ist, dass er die Erstellung eines Energiemodells erfordert. Die Erstellung ei-

nes Energiemodells ist allerdings weniger aufwendig als die Erstellung eines kompletten Simulationsmodells. Zusammenfassend gilt somit für den hier vorgeschlagenen Ansatz, dass die Genauigkeit des erstellten Energiemodells die Gesamtgenauigkeit beschränkt. Auch sorgt die leicht modifizierte Firmware für Abweichungen von den realen Werten. Diese Nachteile sind aber im Vergleich zu den Vorteilen eher marginal.

6.3.3 Energiemodell für intelligente Sensoren

In diesem Abschnitt wird ein Energiemodell für intelligente Sensoren vorgestellt, welches die vorgeschlagene SiL-Entwicklungs- und Debugging-Umgebung um eine zusätzliche Komponente erweitert. Diese Komponente ist in der Lage eine Abschätzung des Energieverbrauchs für das gesamte System zu erstellt. Dies ermöglicht es dem Entwickler eines solchen Systems, seinen Entwicklungs- und Testprozess mit Fokus auf den Energieverbrauch zu gestalten.

Die Abschätzung wird durch zwei Kernkomponenten erreicht, welche die notwendige Funktionalität bieten. Die erste Komponente ermöglicht eine Kommunikation mit dem untersuchten intelligenten Sensor, um folgende zwei Ziele zu erreichen: Erstens sollte es möglich sein, den internen Zustand zu beobachten und Daten vom intelligenten Sensor während der Laufzeit oder beim Debuggen des Sensors zu empfangen. Zweitens sollte die Kommunikationskomponente es ermöglichen, Daten vom Host an die Sensoren zu senden, sodass der Sensor zuvor aufgezeichnete Daten verarbeiten kann. Unter Einhaltung dieser Voraussetzungen ist eine wiederholbare und sogar reproduzierbare Untersuchung des Sensorsystems und der zu entwickelnden Firmware möglich. Und damit auch eine reproduzierbare Möglichkeit, den Stromverbrauch für bestimmte Situationen zu schätzen. Mit der in dieser Arbeit vorgestellten Sensor-in-the-Loop-Debugging-Architektur ist diese Funktionalität bereits verfügbar.

Die zweite Kernkomponente ist ein spezielles Energiemodell für den intelligenten Sensor, für den die Firmware entwickelt wird. Dieses Energiemodel (Power Model, PM) erfüllt die eigentliche Aufgabe der Schätzung des Stromverbrauchs des verwendeten intelligenten Sensors. Es muss in der Lage sein, den Energiezustand des Systems zu beobachten. Dies kann automatisch durch Beobachtung spezieller System-Register, welche den aktuellen Zustand anzeigen, geschehen. Durch die Beobachtung des Stromversorgungszustands des Systems kann das Energiemodell den aktuellen Energieverbrauch des Systems unter Verwendung zuvor definierter Energieverbrauchsraten in Kombination mit dem Zustand melden. Bei komplexen intelligenten Sensoren, bei denen nicht alle energiebedarfsabhängigen Register jederzeit zugänglich sind, kann dies jedoch eine schwierige Aufgabe sein. In diesem Fall ist es praktischer, das PM selbst zur Kontrolle des Energiezustands des Systems zu verwenden.

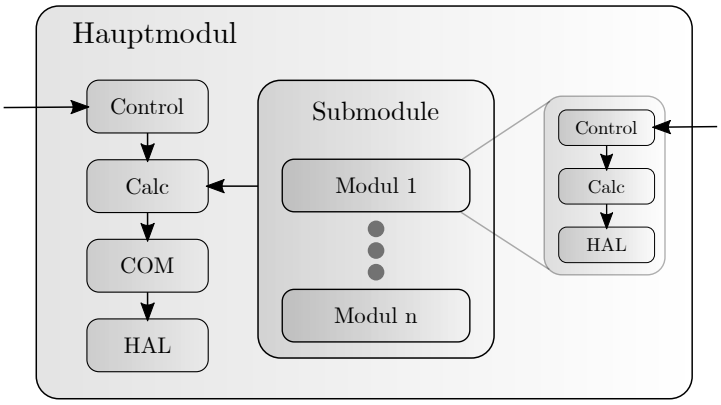


Abbildung 6.27: Allgemeine Struktur eines Energiemodells für intelligente Sensoren

In diesem Abschnitt wird ein solches Energiemodell vorgeschlagen, das in der Lage ist, die Energiezustände der einzelnen Systemkomponenten des intelligenten Sensors zu überwachen und den Energieverbrauch an den Host zurückzumelden. Ein solches Energiemodell könnte wie in Abbildung 6.27 dargestellt aufgebaut sein. Dieses PM hat eine hierarchische Struktur, um moderne, komplexe intelligente Inertialsensoren, welche aus mehreren Systemkomponenten bestehen, zu handhaben. Normalerweise hat ein intelligenter Sensor mindestens eine Verarbeitungseinheit, die mit einer Vielzahl von Peripheriegeräten wie General Purpose Input Outputs (GPIOs), Analog to Digital Converters (ADCs) oder Kommunikationsinterfaces (Communication Interfaces, CIs) ausgestattet ist. Neben dieser sogenannten SPU gibt es einen oder mehrere Sensorelemente, z.B. Beschleunigungsmesser, Gyroskop, Magnetometer oder einen Temperatursensor. Jede dieser Komponenten sollte als separates Modul im Energiemodell modelliert werden. Die Komponenten besitzen jeweils eigene Zustände, Energieverbrauchsschätzungen und Steuerlogiken.

Das PM des in Abbildung 6.27 dargestellten intelligenten Sensors besteht aus einem Hauptmodul, das die Submodule für die verschiedenen Komponenten des Systems enthält. Innerhalb des Hauptmoduls befinden sich vier Funktionsblöcke, die verschiedene Aufgaben übernehmen. Der Block *Control* übernimmt die Kommunikation zwischen dem Benutzer und dem Energiemodell. Mithilfe dieses Blocks ist der Firmware-Entwickler in der Lage, Energiezustände des Moduls zu schalten und zusätzliche Informationen des Modells zu erhalten. Wenn der *Control*-Block einen Zustandswechsel auslöst, übernimmt der Berechnungsblock *Calc* die Aufgabe und berechnet den geschätzten Energieverbrauch für den neuen Zustand. Dazu fragt er den aktuellen Energieverbrauch jedes Teilmoduls ab. Diese aktualisierte Energieverbrauchsschätzung wird vom Kommunikationsblock *COM* des Modells an das Framework auf dem Entwicklungsrechner zurückgemeldet. Der letzte Funktionsblock konfiguriert den ausgewählten Energiemodus für die aktuelle Hardwarekomponente. Zur Konfiguration des tatsächlichen Hardware-Energiezustands wird der Hardware-Abstraktionsschichtblock *HAL* verwendet. Er besteht aus Funktionen oder Mechanismen zur Rekonfiguration des Energiezustands des entsprechenden Hardware-

elements. Jedes Untermodul des PM funktioniert als unabhängige Komponente und kann vom Entwickler direkt konfiguriert und von außen zugänglich gemacht werden. Die Submodule können nicht durch das Hauptmodul konfiguriert werden, um sicherzustellen, dass es keine unerwarteten Änderungen im Energiezustand der Komponenten des Systems gibt. Die Funktionalität der Submodule ist in drei Funktionsblöcke unterteilt, die dem Hauptmodul sehr ähnlich sind. Der Steuerblock (*Control*) verwaltet den Zustand des Untermoduls und ermöglicht die Kommunikation. Er kommuniziert auch mit dem Hauptmodul, um die aktualisierte Energieabschätzung für das Modul zu melden. Das Funktionsverhalten des *Calc*-Blocks und des *HAL*-Blocks ist dem des Hauptmoduls ähnlich. Wenn ein Untermodul seinen Energiestatus ändert und die neue Energieabschätzung berechnet, meldet es diese Änderung an das übergeordnete Modul. Das Topmodul berechnet dann den Stromverbrauch des gesamten Systems neu und meldet die Änderungen an den Entwickler zurück. Der Block *COM* fehlt in den Submodulen, da die Meldung der Energieverbrauchsschätzung vom Top-Level-Modul übernommen wird.

Die Umsetzung dieses Konzepts kombiniert die Vorteile der State-of-the-Art Methoden zur Energieabschätzung, welche mit intelligenten Sensoren verwendet werden. Es bietet eine Wiederholbarkeit und Reproduzierbarkeit der Untersuchung, die derzeit nur mit simulationsbasierten Ansätzen möglich ist. Außerdem nutzt diese Methode die reale Hardware, um die Sensor-Firmware zu testen und zu verifizieren. Dies ermöglicht es dem Entwickler, Informationen über die funktionalen Eigenschaften und darüber hinaus, die extrafunktionalen Eigenschaften des zu testenden Systems zu gewinnen. Das extrafunktionale Merkmal Energieverbrauch kann neben verschiedenen anderen Merkmalen wie Laufzeit und Speichernutzung beobachtet werden. Im Vergleich zu Methoden, die den Stromverbrauch mit einem Power-Debugger messen, liefert diese Methode die Werte, die für einen laufenden Sensor ohne angeschlossenen Debugger gelten würden.

6.3.4 Beispiel Implementierung

Das vorgeschlagene Energiemodell wurde auf dem BMF055 [A18], einem aktuellen intelligenten Inertialsensor, implementiert. Dieser Sensor wurde verwendet, weil er bereits über eine Implementierung der Sensor-in-the-Loop-Schnittstelle verfügt. Dies hat den Vorteil, dass der geschätzte Energieverbrauch leicht von der Firmware an die integrierte Entwicklungsumgebung (Integrated Developement Environment, IDE) übertragen werden kann. Darüber hinaus ermöglicht es, vorab aufgezeichnete Sensordaten für reproduzierbare Tests des implementierten Energiemodells zu verwenden.

Das Energiemodell wurde nach dem in Unterabschnitt 6.3.3 beschriebenen und in Abbildung 6.27 dargestellten Konzept implementiert. Die Programmiersprache C wurde verwendet, um das entwickelte PM auf der aktuellen Sensor-Firmware zu implementieren. Das Hauptmodul wird durch die SPU repräsentiert, genauer gesagt durch den

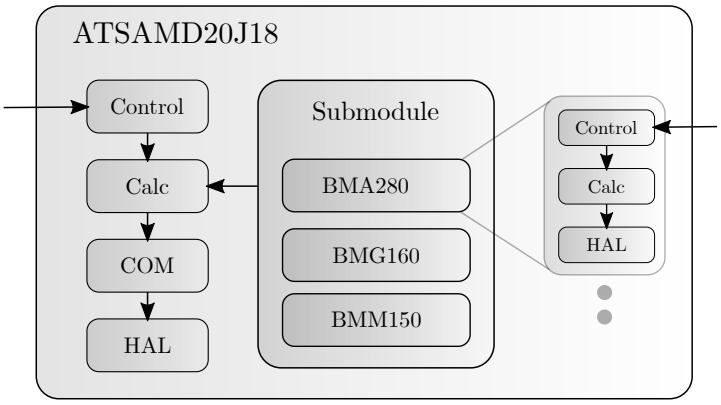


Abbildung 6.28: Spezielle Struktur des Energiemodells für den BMF055

ATSAMD20J18 µC. Alle Blöcke wurden wie in Unterabschnitt 6.3.3 Konzept beschrieben implementiert. Für den *COM*-Block wird die SiL-Schnittstelle zur Übermittlung des geschätzten Energieverbrauchs verwendet. Der *HAL*-Block enthält Funktionen, die von *Microchip* als Hersteller des verwendeten µC bereitgestellt werden.

Das implementierte Energiemodell hat drei zusätzliche Submodule, eines für jedes Sensorelement des intelligenten Sensors BMF055. Der Sensor ist mit einem Beschleunigungsmesser (BMA280)[A51], einem Gyroskop (BMG160)[A52] und einem Magnetometer (BMM150)[A53] ausgestattet. Für jeden dieser Sensoren wurde ein Untermodul erstellt. Der *HAL*-Block dieser Module wird mithilfe der von *Bosch Sensortec* bereitgestellten Hardware-Abstraktionsbibliothek [A135] implementiert. In Abbildung 6.28 kann man die spezifische Implementierung des Energiemodells auf dem intelligenten Sensor BMF055 sehen.

Tabelle 6.3: Verbrauchszahlen aus dem Datenblättern der Komponenten

ATSAMD20J18 in µA @3.3V					
While1	NORMAL	IDLE0	IDLE1	IDLE2	Standby
2330	4030	1350	950	780	4
BMA280 in µA @2.4V					
Normal	Suspend	Deep Suspend	LowPower1	LowPower2	Standby
130	2.1	1	6.5	66	62
BMG160 µA @2.4V					
Normal	FastPowerUp	Suspend	DeepSuspend		
5000	2500	25	5		
BMM150 in µA @ 2.4V					
Normal	Normal @10Hz	LowPower @ 10Hz	High acc @20Hz	Suspend	
800	500	170	4900	3	

Die Energieverbrauchswerte für die einzelnen Zustände des Energiemodells wurden den

Datenblättern der einzelnen Komponenten des intelligenten Sensors entnommen. In Tabelle 6.3 sind diese Werte für jede Komponente einzeln aufgeführt. Der Strom für die einzelnen Komponenten wurde bei verschiedenen Spannungspegeln gemessen. Das Datenblatt der Sensoren liefert Messungen bei 2,4 V und das Datenblatt des Mikrocontrollers bei 3,3 V. Die Sensoren haben einen internen linearen Spannungsregler, sodass der Strom nahezu unabhängig von der Spannung ist, solange die Spannung im zulässigen Bereich liegt. Der Strom des Mikrocontrollers hängt von der verwendeten Spannung ab, sodass in den Experimenten die 3,3 V für das gesamte System verwenden werden. Die Verwendung derselben Versorgungsspannung für alle Komponenten gewährleistet vergleichbare Ergebnisse. Alle Komponenten sind mit ihren einzelnen Energiezuständen und der entsprechenden Stromaufnahme aufgeführt. Wie in Unterabschnitt 6.3.6 erörtert wird, sind diese Schätzungen nicht für alle Anwendungsfälle sehr zuverlässig und müssen kalibriert werden, um zufriedenstellende Ergebnisse zu erzielen.

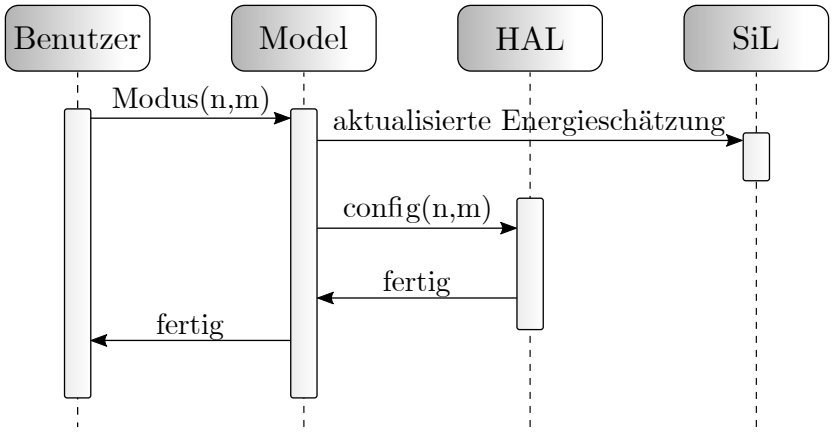


Abbildung 6.29: Sequenzdiagramm für die Interaktion mit dem Modell

Abbildung 6.29 zeigt das Sequenzdiagramm für die Umschaltung in einen anderen Betriebszustand. Der Benutzer konfiguriert einen neuen Betriebszustand mithilfe des *Control*-Blocks. Das Modul berechnet die neue Energieabschätzung und teilt sie über die SiL-Schnittstelle mit. Danach wird der *HAL*-Block vom Modell aufgerufen, um den aktuellen Betriebszustand der Sensorkomponente umzuschalten.

Abbildung 6.30 zeigt, wie der durch das Energiemodell geschätzte Stromverbrauch durch das Sensor-in-the-Loop-Framework visualisiert wird. Diese Abbildung zeigt die Daten für ein komplexes reales Sensor-Szenario mit Zustandsänderungen und unterschiedlichen Abtastraten der Sensoren. Eine detailliertere Beschreibung dieses Beispiels ist im Unterabschnitt 6.3.5 zu finden. Im Unterabschnitt 6.3.6 sind detailliertere Ansichten des vom Modell gelieferten Stromverbrauchs dargestellt. Das Framework visualisiert den Stromfluss in das System, der tatsächliche Energieverbrauch hängt von der Spannung ab, mit der das System betrieben wird. Für die Experimente wurde eine Spannung von

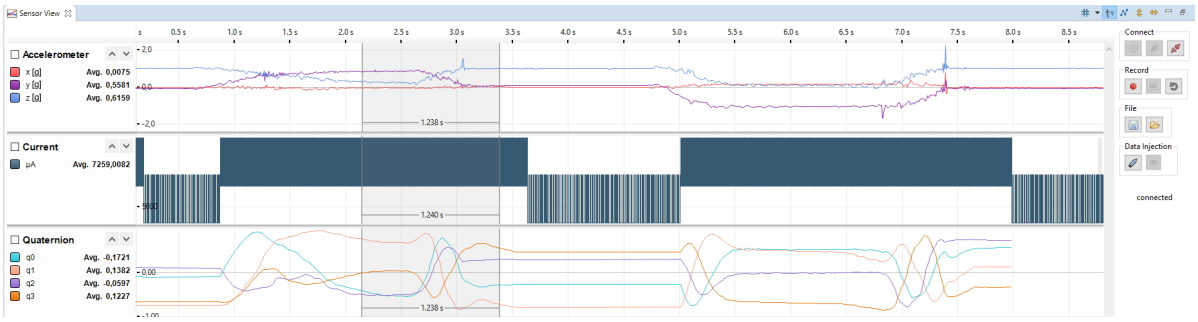


Abbildung 6.30: Sensor View in der Eclipse Entwicklungsumgebung

3,3 Volt verwendet, aber die gewählte Spannung kann in verschiedenen Szenarien variieren. Dies muss bei der Konfiguration des Modells berücksichtigt und entsprechend konfiguriert werden. Eine Änderung der Spannung einzelner Komponenten während der Laufzeit wird von der aktuellen Implementierung jedoch nicht unterstützt. Zusätzlich zu den Energieverbrauchsschätzungen kann der Entwickler die Rohdaten der einzelnen Sensoren einsehen. Darüber hinaus ist es möglich, interne Systemzustände oder Ergebnisse von Sensoralgorithmen wie die Quaternion-Darstellung der räumlichen Lage des Sensors anzuzeigen. Auf diese Weise können alle beobachtbaren Daten in Beziehung zum Energieverbrauch des Systems gesetzt werden. Diese ermöglicht dem Entwickler eine energiebewusste Firmwareentwicklung. Die Abbildung 6.30 zeigt eine Sequenz von ca. 8,5 s, um Details des aktuellen Signals zu sehen, muss der Benutzer in das Signal hineinzoomen. Eine detailliertere Ansicht des aktuellen Signals ist in Unterabschnitt 6.3.6 z.B. in Abbildung 6.41 dargestellt.

6.3.5 Experimente

Nach der Implementierung des Energiemodells auf dem intelligenten Sensor wurden Experimente zum Energieverbrauch des Systems durchgeführt. Diese Experimente wurden in zwei Messreihen unterteilt:

1. In der ersten Serie wurde die Stromaufnahme jeder einzelnen Komponente des Sensors gemessen und mit dem entsprechenden Energiemodell verglichen. Somit kann in dieser Serie überprüft werden, wie gut das Energiemodell mit dem tatsächlichen Verbrauch der Hardware übereinstimmt. Außerdem können diese Messungen zur Kalibrierung des Energiemodells für alle Komponenten verwendet werden, um eine genauere Energieabschätzung des gesamten Systems zu gewährleisten. Diese Experimente können auch aufzeigen, wo mehr Aufwand erforderlich ist, um das Energiemodell so genau wie möglich zu gestalten.

2. Der zweite Teil der Experimente wurde für die Energieabschätzung des Gesamtsystems, bestehend aus den einzelnen Komponenten, durchgeführt. Dieses Experiment sollte zeigen, wie gut die Gesamtenergieabschätzung mit der vorgeschlagenen Methode durchgeführt werden kann.

Zur Messung des Stromverbrauchs des intelligenten Sensors im Versuchsaufbau wurde ein Current-Waveform-Analyzer verwendet [A136]. Die Stromversorgung für den Sensor wurde auf einen festen Spannungspegel von 3,3 V eingestellt. Diese 3,3 V werden verwendet, um vergleichbare Ergebnisse zu den Stromwerten aus den Datenblättern zu erhalten. Die Werte für den Mikrocontroller sind mit einer Spannung von 3,3 V angegeben und hängen von dieser Spannung ab. Die Stromwerte für die Sensoren sind hinsichtlich ihres internen Spannungsreglers nicht spannungsabhängig.

Alle Messungen wurden für verschiedene Betriebszustände durchgeführt, z. B. aktiv, Ruhezustand und Standby. Während der Experimente werden die Betriebszustände der getesteten Komponente oder des Systems iterativ durchlaufen. Nach jeder Änderung des Betriebszustandes einer Komponente des intelligenten Sensors berechnet das interne Energiemodell den aktuellen Stromverbrauch des Sensors. Dieser neue Wert wird über die Sensor-in-the-Loop-Schnittstelle an die IDE gesendet. In der IDE können diese Daten neben den Sensorrohdaten und dem Ergebnis der Orientierungsberechnung visualisiert werden. Für die Experimente wurden diese Daten mit dem tatsächlichen vom Current-Waveform-Analyzer gemessenen Stromverbrauch verglichen. In Kombination mit den Informationen aus den Datenblättern können die Energiemodelle der Einzelkomponenten für genauere Ergebnisse kalibriert werden. Für die Vergleichsmessung des Systems wird eine Firmware verwendet, welche lediglich die Aufzeichnung der Sensordaten enthält, alle sonstigen SiL-Funktionalitäten und das PM sind nicht enthalten. Dies stellt sicher, dass die Messung des Energieverbrauchs so nahe wie möglich am tatsächlichen Energiebedarf der getesteten Firmware liegt.

6.3.5.1 Messungen der Einzelkomponenten

Für die Messungen der einzelnen Komponenten wurden alle enthaltenen Komponenten und Sensorelemente des intelligenten Sensorsystems auf ihre Betriebsmodi mit der geringsten Energieaufnahme konfiguriert. Dadurch wird sichergestellt, dass der Einfluss der einzelnen Komponenten aufeinander so gering wie möglich gehalten wird. Die Messungen für alle getesteten Komponenten wurden mit der gleichen Methodik durchgeführt. Die getestete Komponente startet in ihrem Standard-Betriebszustand. In einem Intervall von zwei Sekunden schaltet das getestete Bauteil zwischen all seinen möglichen Energiezuständen um. Die Zeit jedes Stromversorgungszustands wird von einem Timer gesteuert. Dieser Timer löst einen Interrupt aus, um den Prozessor aufzuwecken. Nach dem Aufwachen wird der nächste Betriebszustand konfiguriert und der Timer erneut gestartet. Für jede der nachfolgenden Messungen wurde ein extra Firmware erstellt, welche die

zeitgesteuerte Umschaltung der Energiezustände übernimmt. Bei einer Abtastrate des Current-Waveform-Analyzers von 1 MSamples pro Sekunde liefert das Zwei-Sekunden-Intervall ausreichend Messpunkte, um aussagekräftige Ergebnisse für jeden Zustand zu erhalten.

Für den ersten Test der einzelnen Komponenten wurde die SPU in allen möglichen Betriebszuständen gemessen. Die in dem intelligenten Sensor verwendete SPU ist ein ATSAMD20J18 Mikrocontroller von Microchip [A54]. Das Flussdiagramm in Abbildung 6.31 zeigt den Programmablauf der Firmware während der Messungen.

Abbildung 6.31: Flussdiagramm der SPU Betriebsmodi

Nach der Messung der SPU wurde der Stromverbrauch der einzelnen Sensorelemente untersucht. Daher wurde die SPU in den Standby-Modus versetzt, um ihren Einfluss zu minimieren.

Zunächst wurde der Stromverbrauch des Gyroskops gemessen. Wie in Abbildung 6.32 gezeigt, wurden alle möglichen Betriebsmodi dieses Sensorgeräts konfiguriert.

Abbildung 6.32: Flussdiagramm für den Test der Gyroskop-Betriebsmodi

Als Zweites wurde der Energieverbrauch des Beschleunigungssensors untersucht. Abbildung 6.33 zeigt den Kontrollfluss für die Beschleunigungssensor-Messungen

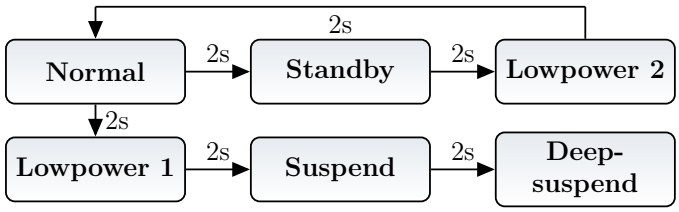


Abbildung 6.33: Flussdiagramm für die Energieverbrauchstests des Beschleunigungssensors

Mit dem Beschleunigungssensor ist es nicht möglich, direkt zwischen allen Betriebsmodi zu wechseln. Dies ist nicht möglich, da es keinen gültigen Zustandsübergang zwischen dem Modus *Lowpower 2* und dem Modus *Lowpower 1* gibt. Daher muss vor der Verwendung des Modus *Lowpower 1* wieder in den Modus *Normal* gewechselt werden. Abgesehen davon wird der Test wie beim Gyroskop durchgeführt.

Der letzte gemessene Sensor war das Magnetometer. Es verfügt über die meisten Betriebsarten aller im intelligenten Sensor verwendeten Sensorelemente. Die Abtastmodi sind in vier Modi unterteilt, welche von *Normal* bis *Lowpower* reichen. Die Messungen wurden ähnlich wie bei den beiden vorherigen Sensoren durchgeführt, der Kontrollfluss kann in Abbildung 6.34 nachvollzogen werden.

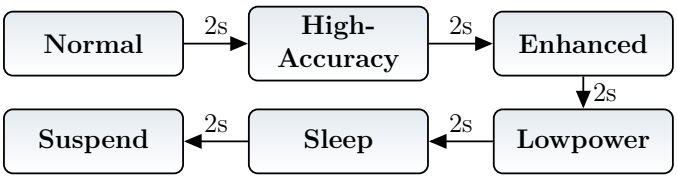


Abbildung 6.34: Flussdiagramm für den Energieverbrauchstest für das Magnetometer

Nachdem die Experimente für die isolierten Betriebsarten jeder Komponente des intelligenten Sensors durchgeführt wurden, können die gemessenen Werte zum Vergleich mit den Werten in den Datenblättern verwendet werden. Außerdem werden die Messergebnisse für die Kalibrierung des Energiemodells der einzelnen Komponenten verwendet, um genauere Ergebnisse zu erhalten. Dieser Schritt ist in Unterkapitel Ergebnisse näher erläutert.

6.3.5.2 Messungen für das Gesamtsystem

Nach den Messungen und der Kalibrierung für die einzelnen Komponenten des Systems wurde ein Experiment für das gesamte System durchgeführt. Damit sollte überprüft werden, wie gut die vorgeschlagene Methodik den Energieverbrauch anhand der Modelle für die einzelnen Komponenten modellieren kann. Um den Energieverbrauch des gesamten Systems mit den vom Energiemodell gelieferten Verbrauchswerten zu vergleichen, wurde ein komplexer Testfall konstruiert. Dieser Testfall stellt eine häufig verwendete Anwendung für intelligente Inertialsensoren dar. Das Flussdiagramm in Abbildung 6.35 beschreibt den Programmfluss der Firmware des intelligenten Sensors.

Das Programm ist hauptsächlich in drei Phasen unterteilt. Die Firmware beginnt mit der Initialisierungsphase, in der die SPU und alle Peripheriegeräte wie GPIOs, Kommunikationsschnittstellen und Timer konfiguriert werden. Zur Abtastung der Gyroskop- und Beschleunigungsmessdaten wird ein Timer konfiguriert, der einen Interrupt mit einer Frequenz von 200 Hz auslöst. Dieser Interrupt wird als *Timer-Counter* Interrupt oder kurz *tc interrupt* bezeichnet.

Der Ausgangszustand der Firmware ist *S1*, nach jedem Interrupt werden die Sensordaten abgetastet und ein *No-Motion*-Algorithmus prüft anhand der gemessenen Beschleunigung, ob sich der Sensor bewegt. Wenn sich der Sensor bewegt, wird die Ausrichtung des Sensors mithilfe des Madgwick-Fusionsalgorithmus [A81] berechnet. Dieser Algorithmus

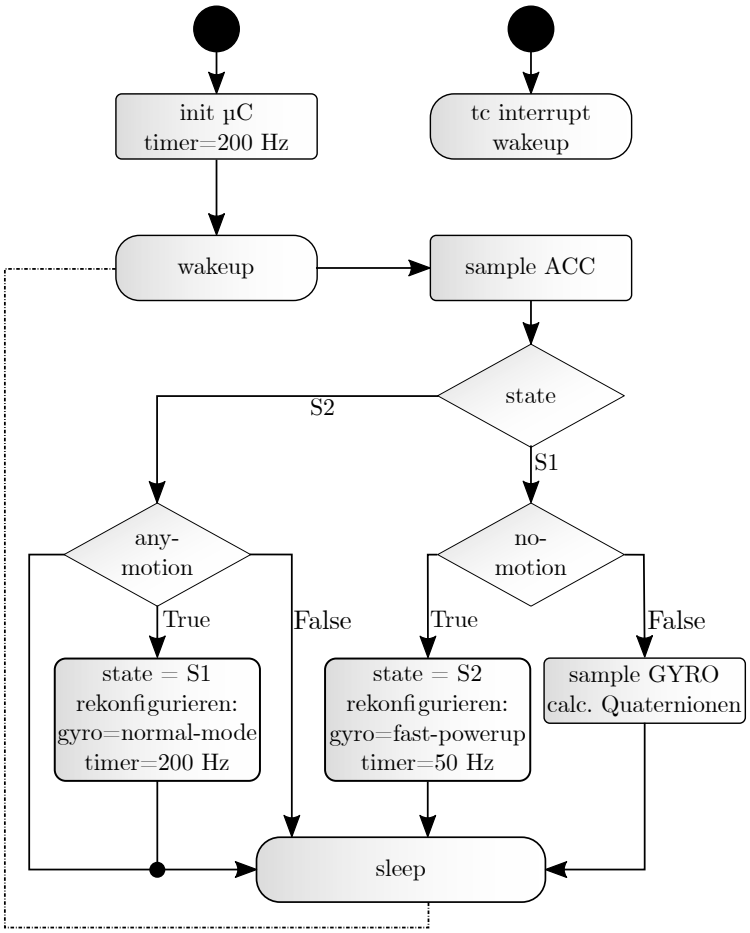


Abbildung 6.35: Flussdiagramm des komplexen Anwendungsbeispiels

berechnet die Ausrichtung des Sensors in Quaternion-Darstellung unter Verwendung der Drehraten und der Beschleunigungsdaten. Der Sensor geht nach der Bestimmung der Orientierung in den Schlafmodus, bis der nächste Timer-Interrupt auftritt. Wenn der *No-Motion*-Algorithmus in *S1* feststellt, dass sich der Sensor nicht mehr bewegt, wird der Zustand auf *S2* umgeschaltet und die SPU geht in den Ruhezustand. Außerdem wird das Gyroskop auf den Schlafmodus *Fast-Powerup* konfiguriert, da seine Daten in *S2* nicht benötigt werden. Der Timer für die Abtastrate wird auf 50 Hz umkonfiguriert.

In *S2* erkennt ein *Any-Motion*-Algorithmus, ob sich der Sensor wieder bewegt. Dazu verwendet der Algorithmus die 50 Hz-Beschleunigungssensordaten. Das Gyroskop befindet sich in einem *Fast-Powerup*-Schlafmodus, in diesem Zustand werden keine Gyroskopdaten abgetastet. Wenn der *Any-Motion*-Algorithmus eine Bewegung des Sensors feststellt, wird der Zustand zurück zu (S1) gewechselt. Außerdem wird das Gyroskop in den *Normal*-Modus geschaltet und die Aktualisierungsrate des Timers wird auf eine Abtastfrequenz von 200 Hz umkonfiguriert.

6.3.6 Ergebnisse

Im vorangegangenen Abschnitt wurde die verschiedenen Experimente erläutert welche für das Modell und die Testanordnung durchgeführt wurden. In diesem Abschnitt werden die Ergebnisse der Experimente gezeigt und jedes Ergebnis im Detail diskutiert. Wie im Unterabschnitt 6.3.5 beschrieben, wurden die Experimente in zwei Serien unterteilt. In der ersten Serie wurde der Stromverbrauch für jede einzelne Komponente gemessen. Diese Messungen dienten dazu, die grundsätzliche Durchführbarkeit und Genauigkeit der vorgeschlagenen Methodik zur Schätzung des Energieverbrauchs zu überprüfen. Außerdem wurden die Messungen zur Kalibrierung der Energiemodelle der einzelnen Komponenten verwendet.

Die zweite Serie von Experimenten wurde nach der Kalibrierung durchgeführt, um die Genauigkeit der vorgeschlagenen Methodik für das gesamte System zu messen.

6.3.6.1 Messungen der Einzelkomponenten

Im ersten Experiment für die einzelnen Komponenten wurde der Energieverbrauch für jeden Betriebsmodus der SPU gemessen. Die Ergebnisse der Messungen für alle fünf Betriebsmodi werden in Abbildung 6.36 angezeigt.

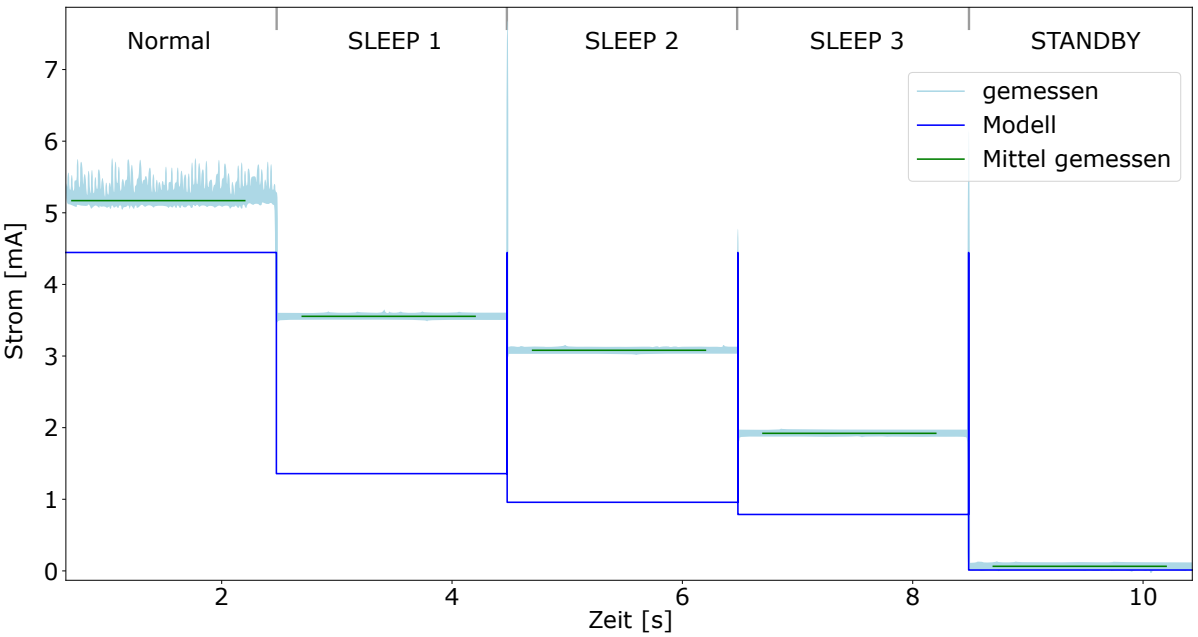


Abbildung 6.36: Gemessener und modellierte Stromfluss der SPU-Betriebsmodi

Der tatsächlich gemessene Strom ist in hellblau dargestellt. Um einen vergleichbaren Stromwert zu erhalten, wurde der Mittelwert der Stromwerte über 1,5 s für jeden Betriebszustand berechnet. Dieser Mittelwert ist in Grün über den gemessenen Stromwerten eingezeichnet. Die berechneten Werte des Energiemodells sind in Blau dargestellt, diese Werte repräsentieren den mittleren Strom für das gesamte Sensorsystem zu einem bestimmten Zeitpunkt. Die hohen Spitzenwerte zwischen den verschiedenen Zuständen werden durch die Aktivierung der SPU zur Rekonfiguration des Betriebszustandes verursacht. Während dieser kurzen Zeitspanne ist der Stromverbrauch viel höher als im Durchschnitt, da die Kondensatoren des Systems nach dem Einschalten aufgeladen werden müssen [A137], [A138].

Es ist zu erkennen, dass die Ausgabe des Energiemodells und die tatsächlich gemessenen Werte in Bezug auf den Strom nicht sehr gut übereinstimmen. Dies hat hauptsächlich zwei Gründe:

1. Die Werte im Datenblatt werden unter sehr spezifischen Bedingungen hinsichtlich der Konfiguration des tatsächlichen Mikrocontrollers und seiner Peripheriegeräte gemessen. Der Stromverbrauch einer SPU oder eines anderen Mikrocontrollers hängt von einer Vielzahl von Faktoren ab, die das Setup bestimmen. Er hängt in hohem Maße von der Takteinstellung ab. Beispielsweise davon, welche Komponente als Taktgeber verwendet wird (interner Taktoszillator oder externer Oszillator). Auch die verwendete Taktfrequenz für die separaten Taktdomänen, die für den CPU-Kern und die Peripheriegeräte verwendet werden, ist ein Faktor. Die verwendeten Peripheriegeräte wie Timer, GPIOs oder Kommunikationsschnittstellen haben zusätzlich einen signifikanten Einfluss auf den Stromverbrauch des Systems.
2. Die Werte im Datenblatt beziehen sich auf einen einzelnen isolierten Mikrocontroller. Bei einem intelligenten Sensor kann die SPU geringfügig anders sein, da der Hersteller des Sensors ihn an seinen Bedarf angepasst hat.

Diese große Bandbreite an Parametern, welche den Stromverbrauch beeinflussen, macht eine Kalibrierung des Energiemodells erforderlich. Nur so können zuverlässige Ergebnisse durch das Modell generiert werden.

Betrachtet man das Timing der Messungen und die durch das Energiemodell berechneten Daten, so zeigt Abbildung 6.36 eine sehr gute Übereinstimmung. Dies macht den Ansatz auch für das Auffinden und Debuggen von Fehlern in Programmablauf oder im zeitlichen Verhalten der Firmware geeignet.

Um repräsentative Ergebnisse zu erhalten, wurde das Energiemodell für die SPU mit den Werten aus den ersten Experimenten kalibriert. Die berechneten Mittelwerte aus den Messungen, die im Diagramm als grüne Linien dargestellt sind, können in Tabelle 6.4 eingesehen werden. In dieser Tabelle sind auch die Messwerte für alle anderen Komponenten aufgeführt.

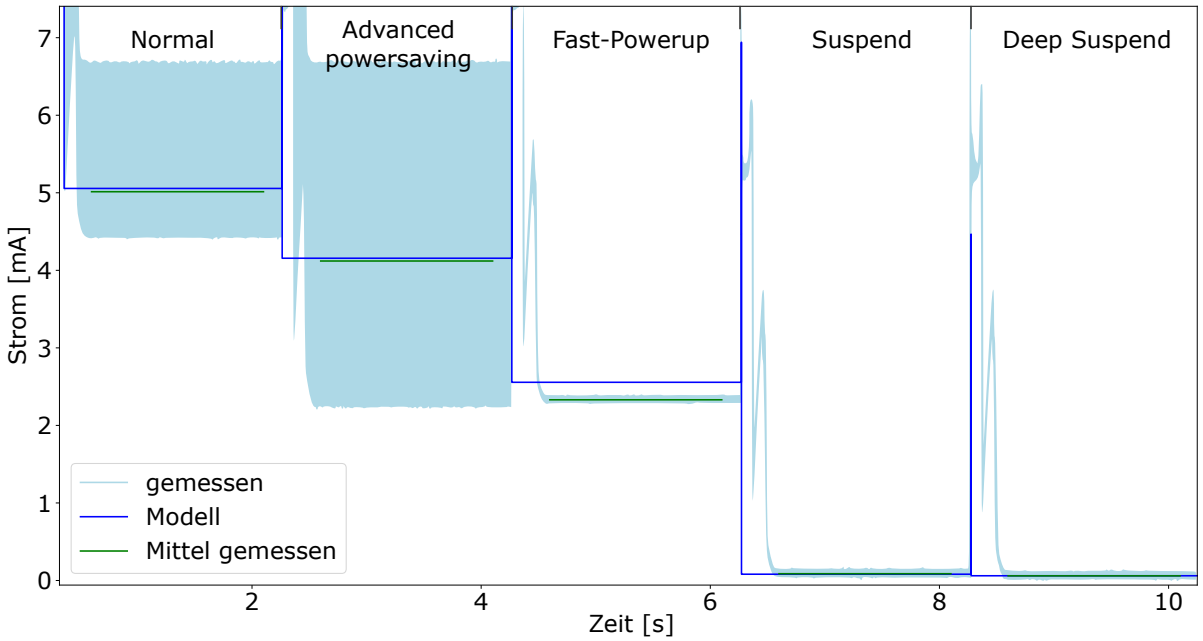


Abbildung 6.37: Gemessener und modellierte Stromfluss der Gyroskope-Betriebsmodi

Die Ergebnisse für die Betriebsmodi des Gyroskops sind in Abbildung 6.37 dargestellt. Wie bei allen anderen Komponententests wurde jeder Energiezustand für zwei Sekunden aktiviert, bevor die SPU aufwacht und das Gyroskop neu konfiguriert. Die SPU befand sich während der Messungen des Stromverbrauchs des Gyroskops im Standby-Modus. Die gemessenen Werte für jeden Modus sind in Tabelle 6.4 zu finden. Um das in Abbildung 6.37 gezeigte Diagramm zu erstellen, wird der Standby-Strom der Verarbeitungseinheit aus den Daten als Offset entfernt.

Die theoretischen Werte aus dem Datenblatt stimmen in den meisten Betriebszuständen mit den gemessenen Werten überein. Nur der Modus *Fast-Powerup* weicht von den Datenblattwerten ab. Für das Gyroskop wäre es möglich, mit den Verbrauchswerten aus dem Datenblatt zu arbeiten und vernünftige Ergebnisse aus dem Energiemodell zu erhalten. Für den Test des gesamten intelligenten Sensors wurden jedoch die gemessenen Daten zur Anpassung des Energiemodells verwendet. Dadurch wird sichergestellt, dass die Verbrauchsabschätzungen des Modells der tatsächlichen Stromaufnahme des Systems so nahe wie möglich kommen.

Die Abbildung 6.38 zeigt den gemessenen Strom für alle möglichen Betriebszustände des Beschleunigungssensors. Die gemessenen Werte entsprechen für die meisten Zustände den Werten aus dem Datenblatt. Der dritte (Lowpower 1) und der fünfte (Lowpower 2) Modus weichen von den erwarteten Werten ab. Dies ist auf den größeren Parametersatz für diese Zustände zurückzuführen. Der Entwickler kann diese Modi mit einer Vielzahl

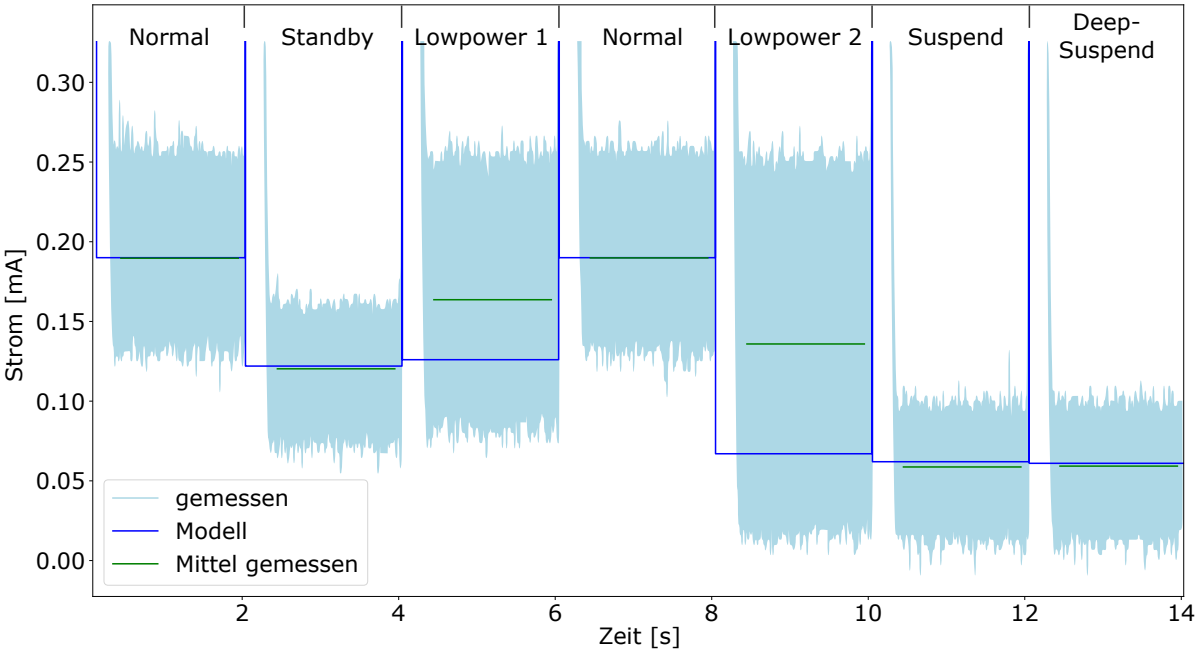


Abbildung 6.38: Gemessener und modellierte Stromfluss aller Betriebsmodi des Beschleunigungssensors

von Parametern konfigurieren, um den gewünschten Zweck zu erfüllen. Die Werte im Datenblatt sind der niedrigste mögliche Stromverbrauch für diese Modi. Für die Experimente wurden für jeden Modus die Standardkonfiguration verwendet. Dies führt zu einem etwas höheren Stromverbrauch.

Die Messungen für die Betriebsmodi des Magnetometers sind in Abbildung 6.39 dargestellt. Die Mittelwerte für alle gemessenen Zustände sind in Tabelle 6.4 aufgeführt. Wie bei allen anderen Experimenten wurde jeder Zustand für 2 s gemessen. Der Mittelwert des gemessenen Stroms passt zu den vom Energiemodell gelieferten Daten. Die hohen Spitzen des gemessenen Signals werden durch die aktuelle Abtastung des Magnetometers verursacht. Jeder Peak repräsentiert also eine Abtastung. Dieses Verhalten wird nicht durch das PM abgebildet. Man könnte hier aber das Modell so verfeinern, dass es den mittleren Stromverbrauch und den Spitzenstrom berechnet, falls dies notwendig ist. Im *High-Accuracy*-Modus weichen die Mess- und die Modelldaten am stärksten voneinander ab. Wie bei allen anderen Sensoren auch sind die Daten für den modellierten Stromverbrauch dem Datenblatt entnommen. Diese Werte werden unter perfekten Bedingungen gemessen und stellen immer den geringstmöglichen Stromverbrauch dar.

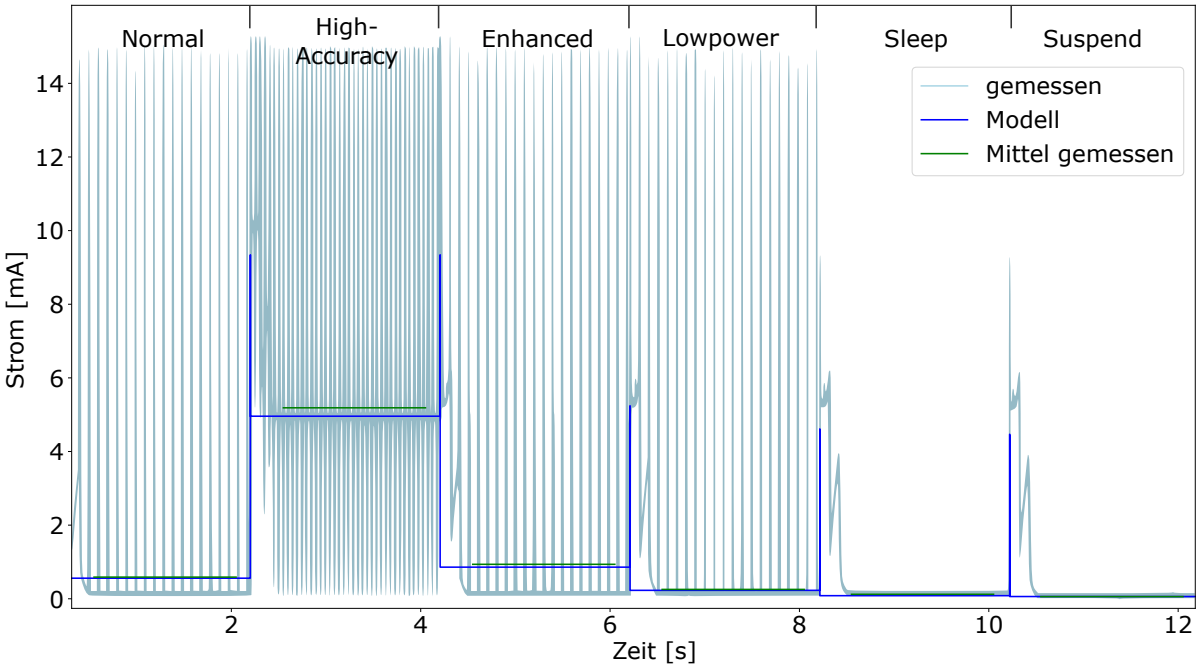


Abbildung 6.39: Gemessener und modellierte Stromfluss aller Betriebsmodi des Magnetometers

Die Messwerte für jede gemessene Komponente des intelligenten Sensors sind in Tabelle 6.4 aufgeführt. Diese Tabelle enthält die Werte für jeden einzelnen Betriebszustand der jeweiligen Komponente. Diese Daten wurden verwendet, um das Energiemodell zu kalibrieren. Nach der Kalibrierung wurde das endgültige Experiment für das Gesamtsystem durchgeführt.

Tabelle 6.4: Messwerte für den Stromverbrauch der einzelnen Sensorkomponenten

ATSAMD20J18 in µA @3.3V					
NORMAL	IDLE0	IDLE1	IDLE2	Standby	
5170	3554	3081	1921	64	
BMA280 in µA @3.3V					
Normal	Suspend	Deep Suspend	LowPower1	LowPower2	Standby
190	59	59	136	164	120
BMG160 µA @3.3V					
Normal	AdvPowSave	FastPowerUp	Suspend	DeepSuspend	
5014	4120	2330	87	60	
BMM150 in µA @ 3.3V					
Enhanced @10Hz	Normal @10Hz	LowPower @ 10Hz	High acc @20Hz	Sleep	Suspend
936	588	254	5187	122	59

6.3.6.2 Messungen für das Gesamtsystem

Nach der Kalibrierung der Energiemodelle der einzelnen Komponenten wurde das Experiment für den komplexen Testfall wie in Unterabschnitt 6.3.5 beschrieben durchgeführt.

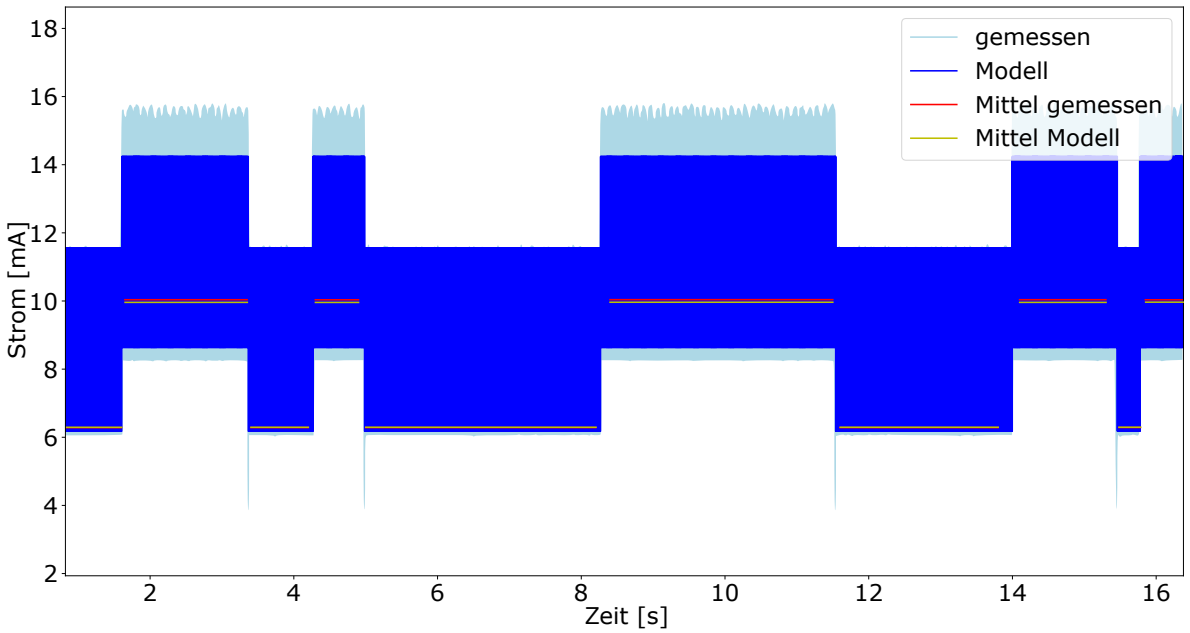


Abbildung 6.40: Stromfluss während des komplexen Testszenarios

Die tatsächlich gemessenen Werte sind in hellblau dargestellt, die vom Energiemodell gelieferten Daten in Blau. Um beide Daten besser vergleichen zu können, wurde der Mittelwert für jedes Segment berechnet. Wie in Abbildung 6.40 zu sehen, gibt es eindeutig abgrenzbare Zustände über die Laufzeit des Experiments. In diesem Experiment gibt es zehn Zustände von unterschiedlicher Länge. Die Zustände mit dem insgesamt geringsten Stromverbrauch sind die *No-Motion*-Zustände, bei denen die Sensorplattform nicht bewegt wird. In diesem Zustand läuft nur der Beschleunigungsmesser mit einer Abtastrate von 50 Hz. Die SPU wacht mit derselben Frequenz auf und führt den *Any-Motion*-Algorithmus auf der Grundlage der Beschleunigungsmesserdaten aus. In den Phasen mit dem höheren Stromverbrauch führt die SPU eine Orientierungsschätzung auf der Grundlage der Daten des Gyroskops und des Beschleunigungsmessers durch. Diese Berechnung wird mit einer Frequenz von 200 Hz durchgeführt.

Abbildung 6.41 zeigt eine vergrößerte Ansicht des Experiments für das gesamte in Abbildung 6.40 dargestellte System. Ein Wechsel zwischen zwei Zuständen ist in dieser Abbildung im Detail zu sehen. Sie zeigt auch die zeitliche Genauigkeit zwischen den gemessenen Daten und den durch das Modell geschätzten Energieverbrauch. Dieses genaue

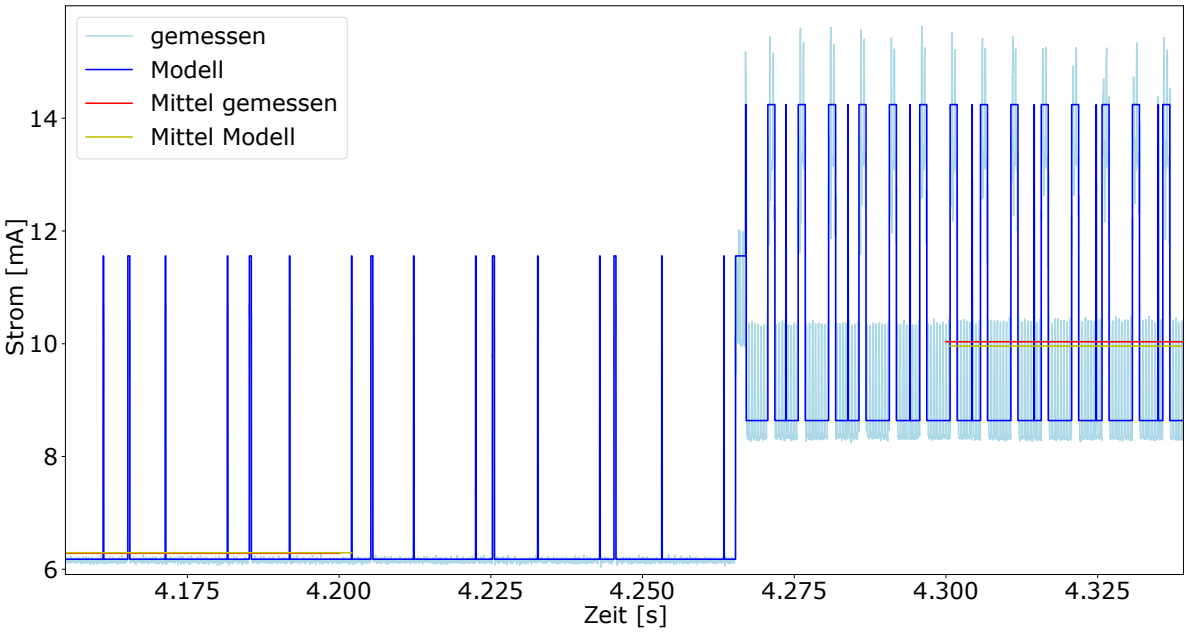


Abbildung 6.41: Vergrößerte Ansicht des Beispiels in Abbildung 6.40

Timing ermöglicht es dem Entwickler der intelligenten Sensor-Firmware Fehlverhalten im Programmablauf leicht zu finden. Aus der Darstellung in Abbildung 6.41 können mehrere Abschnitte der laufenden Firmware identifiziert werden. In der ersten Hälfte des Signals sind die breiteren Spitzen des Modellsignals das 50-Hz-Wake-up; in diesen Perioden wird der Beschleunigungsmesser abgetastet und der *Any-Motion*-Algorithmus ausgeführt. Die kurzen Spitzen des Stroms sind Unterbrechungen des System-Timers, der mit einer Frequenz von 100 Hz ausgelöst werden. Der zweite Teil der Abbildung zeigt das System im *Motion*-Zustand. Hier sind die breiten Spitzen die Abtastung aller Sensoren und die Berechnung des Algorithmus zur Orientierungsschätzung, gefolgt von dem *No-Motion*-Algorithmus. In dieser Phase treten die hohen Spitzenwerte mit einer Frequenz von 200 Hz auf. Der System-Timer-Interrupt ändert seine Frequenz während der gesamten Laufzeit der Firmware nicht.

Die mittleren Stromwerte für alle zehn Segmente sind in Tabelle 6.5 zu finden. Die Tabelle zeigt die Mittelwerte für das gemessene Signal im Vergleich zu den Modelldaten. Der Fehler ist relativ zu den gemessenen Daten in Prozent für jedes einzelne Signal angegeben. Die maximale Abweichung beträgt -0,79 %, das Minus steht hierbei für einen zu gering geschätzten Energieverbrauchverbrauch. In Segment neun ist die geringste Abweichung mit 0,16 %, die mittlere Abweichung über die gesamte Laufzeit beträgt -0,31 %. Das Szenario wurde so gewählt, dass eine möglichst große Anzahl an unterschiedlichen Energiezuständen benutzt wird und auch ein häufiger Wechsel zwischen diesen stattfindet. Daher ist auch bei anderen Szenarien, mit dem gleichen Sensor, eine ähnliche

Genauigkeit der Schätzung zu erwarten.

Tabelle 6.5: Komplexes Testszenario Messwerte und Fehler

Segment	gemessen [mA]	Model [mA]	Fehler [%]
1	6.282	6.293	0.17
2	10.037	9.959	−0.77
3	6.284	6.293	0.15
4	10.035	9.958	−0.77
5	6.284	6.294	0.16
6	10.039	9.963	−0.75
7	6.283	6.294	0.19
8	10.038	9.959	−0.78
9	6.286	6.295	0.14
10	10.035	9.956	−0.79

Es ist zu erkennen, dass die vom Energiemodell gelieferten Daten in allen Phasen mit den gemessenen Daten übereinstimmen. Dies gilt sowohl für den Zeitpunkt als auch für den Mittelwert der Stromamplituden. Innerhalb der zeitlichen Auflösung des Modells von einer Mikrosekunde, wird eine hohe Übereinstimmung zwischen Modell und realen Messwerten erreicht. Allerdings weichen die Modelldaten in kurzen Zeitabschnitten ab, in denen der real gemessene Strom hohe Spitzen aufweist. Dieses Verhalten bei Übergängen wird durch das Modell nicht beschrieben. Dies ist jedoch nicht notwendig, um eine gute Schätzung des mittleren Energieverbrauchs in einer brauchbarer zeitlichen Auflösung zu erhalten.

In den *No-Motion*-Abschnitten mit niedrigerer Abtastrate und geringerer Aktivität ist der modellbasierte geschätzte Strom etwas höher als der gemessene. In den *Motion*-Abschnitten mit höherer Abtastrate und mehr SPU-Aktivität ist der gemessene Strom jedoch höher als der modellierte, was zu einem negativen Fehler führt. Betrachtet man alle Abschnitte dieses Experiments, so liegt die Gesamtabweichung zwischen den gemessenen Daten und den modellierten Daten bei weniger als 1 %.

Das verwendete Szenario spiegelt eine häufig auftretenden Anwendungsfall für einen entsprechenden Sensor wider. Dabei wurde es so gewählt, dass ein häufiger Wechsel zwischen möglichst vielen Energiezuständen stattfindet. Dies soll sicherstellen, dass die verwendeten Ergebnisse des Beispiels möglichst repräsentativ auch für anderer Szenarien sind.

6.3.7 Zusammenfassung und Bewertung

In diesem Abschnitt der Arbeit wurde eine Erweiterung des SiL-Konzepts vorgestellt, die es dem Entwickler von intelligenten Sensorsystemen ermöglicht, den Stromverbrauch während des Entwicklungsprozesses zu beobachten und zu bewerten. Durch die Möglichkeit, den Energieverbrauchverbrauch live zu beobachten, ist es einfach diesen in Beziehung zu bestimmten Abschnitten oder Zuständen der Firmware zu setzen. Außerdem können die Energieverbrauchsschätzungen in Bezug zu bestimmte Sensordaten gesetzt werden. In Kombination mit dem Sensor-in-the-Loop-Ansatz ist es möglich, den Energieverbrauch von intelligenten Sensorsystemen auf eine wiederholbare und reproduzierbare Weise zu untersuchen.

Um die Anwendbarkeit des Ansatzes zu zeigen wurde eine Beispielimplementierung auf einem BMF055 erstellt. Es wurde ein zustandsbasiertes Energiemodell entwickelt, das zur Laufzeit kontinuierlich Verbrauchsabschätzungen liefert. Die Verbrauchsschätzungen des Modells werden aus dem Datenblatt des entsprechenden Sensors abgeleitet. Um die Genauigkeit des Energiemodells zu verbessern, ist es notwendig, das Modell mit zuvor gemessenem Stromverbrauch zu kalibrieren. Diese Kalibrierung ist in der Regel nur einmal pro Sensormodell nötig und kann danach mit Anwenden des gleichen Modells geteilt werden.

Die Ergebnisse zeigen, dass der Ansatz in der Lage ist, während des Entwicklungsprozesses von Firmware für intelligente Sensoren aktuelle Energieverbrauchsschätzungen zu liefern. Darüber hinaus kann gezeigt werden, dass diese Schätzungen während mehrerer Durchläufe der Firmware wiederholbar und reproduzierbar sind. Dies ermöglicht dem Entwickler solcher Firmware eine energiebewusste Entwickelung und Debugging des Sensorsystems. Der implementierte Ansatz zeigt eine durchschnittliche Genauigkeit von 99 % bei der Schätzung des Stromverbrauchs für das komplexe Anwendungsbeispiel.

7 Zusammenfassung

Die Entwicklung von Firmware für intelligente Sensoren stellt Entwickler vor besondere Herausforderungen. Diese Sensorsysteme verfügen über Recheneinheiten, die teilweise stark ressourcen-beschränkt sind. Diese Beschränkungen spiegeln sich zum einen in der Rechenleistung und dem verfügbaren Speicher wider, zum anderen verlangen die Anwendungen oft eine besondere Energieeffizienz. Die Entwicklung und das Testen werden zusätzlich durch die Interaktion mit der physikalischen Welt erschwert. Dadurch ist es eine große Herausforderung die entwickelte Firmware mit immer wieder den gleichen Sensordaten zu testen. Diese Abhandlung adressiert diese Probleme bei der Entwicklung von Algorithmen und Firmware für intelligente Sensoren und Sensorsystemen. Sie schlägt eine Methode und ein System zur wiederholbaren und reproduzierbaren Validierung und Debuggen solcher Systeme vor. Dabei werden nicht nur die funktionalen Parameter der Algorithmen und der entwickelten Firmware betrachtet, sondern auch extra-funktionale Parameter wie Laufzeit, Speicherauslastung und Energieverbrauch.

Im ersten Hauptkapitel dieser Arbeit wird das entwickelte Konzept zur wiederholbaren und reproduzierbaren Entwicklung von intelligenten Sensoren vorgestellt. Der entwickelte Ansatz wird als Sensor-in-the-Loop (SiL) bezeichnet, da er ähnlich wie Hardware-in-the-Loop (HiL)-Ansätze den zu entwickelten Sensor mit definierten Sensordaten speist und dabei die Parameter des Systems überwacht. Diese Parameter sind, wie bei klassischen HiL-Ansätzen, die Ausgaben der Firmware in Bezug auf die Stimuli. Zusätzlich werden weitere extra-funktionale Parameter wie Laufzeit und Energieverbrauch basierend auf den eingegebenen Sensordaten beobachtet. Das Kapitel beginnt mit der konzeptionellen Vorstellung des entwickelten Ansatzes. Dieses Konzept kann auf die meisten State-of-the-Art intelligenten Sensoren angewendet werden. Dabei ist die Art der eigentlichen Sensorelemente nicht entscheidend. So ist der Ansatz sowohl für Inertialsensoren als auch für andere Sensoren, wie zum Beispiel optische oder akustische Sensoren geeignet. Das vorgestellte Konzept wurde anschließend auf einem State-of-the-Art intelligenten Sensor implementiert. Dabei handelt es sich um einen kombinierten 9-DoF Inertialsensor, den BMF055 von *Bosch Sensortec*. Die Implementierung umfasst sowohl eine Anpassung der verwendeten Sensor-Firmware als auch die Wahl und die Instrumentalisierung einer geeigneten Schnittstelle zur Sensor-Prozessor-Einheit (Sensor Processing Unit, SPU). Weiterhin wurde zur Steuerung und Visualisierung des SiL-Prozesses eine Erweiterung für eine häufig verwendetet IDE geschaffen. Dies ermöglicht dem Entwickler die Benutzung des Konzeptes aus seiner gewohnten Entwicklungsumgebung. Die erstellte

Beispielimplementierung wurde anschließend systematisch auf ihre Funktionsweise und ihre Qualität untersucht. Bei der Untersuchung wurde neben dem Speicheroverhead des Ansatzes vor allem das Zeitverhalten eingehend untersucht. Zuletzt wurden die statistische Signifikanz der Ergebnisse überprüft.

Das zweite Hauptkapitel beschäftigt sich mit der Anwendung der entwickelten SiL-Architektur. Weiterhin stellt es Erweiterungen vor, welche den Ansatz komfortabler und nützlicher für Entwicklung von Firmware für intelligente Sensoren machen. Der erste Teil des Kapitels stellt die Untersuchung von Algorithmen zur Orientierungsschätzung auf intelligenten Sensoren vor. Dabei werden die auf Inertialsensordaten arbeitenden Algorithmen mithilfe des SiL-Systems, direkt auf dem intelligenten Sensor überprüft und verglichen. Die Untersuchung umfasst die funktionalen Eigenschaften der Algorithmen und zusätzlich extra-funktionale Eigenschaften wie Laufzeit und Speicherauslastung. Der zweite Teil des Kapitels stellt eine Erweiterung vor, welche es ermöglicht die Sensordaten direkt in der Entwicklungsumgebung zu manipulieren und zu erweitern. Dies ermöglicht es dem Entwickler verschiedene Testfälle für die Sensor-Firmware künstlich zu generieren. Es werden unterschiedliche Beispiele vorgestellt, um die Möglichkeiten und den Funktionsumfang des entwickelten Frameworks zu demonstrieren. In Kombination mit der SiL-Schnittstelle können so komfortabel und reproduzierbar eine Vielzahl von Testfällen auf dem realen Sensor überprüft werden. Im letzten Teil dieses Kapitels wird eine Erweiterung vorgestellt, die sich der wichtigen extra-funktionalen Eigenschaft des Energieverbrauches widmet. Sie ermöglicht es den Energieverbrauch direkt während der Entwicklung der Firmware zu berücksichtigen und so eine energieeffizientere Sensor-Firmware zu erstellen. Dies wird durch die Erstellung eines Energiemodels erreicht, welches zur Laufzeit des Sensors eine Schätzung des aktuellen Energieverbrauchs vornimmt. Diese Verbrauchsdaten können dann über die SiL-Schnittstelle kommuniziert und in der Entwicklungsumgebung visualisiert werden. Die Visualisierung in Bezug zu den eingespeisten Sensordaten ermöglicht ein reproduzierbares Testen und Optimieren des Energieverbrauchs. Der Ansatz wurde auf dem intelligenten Inertialsensor BMF055 implementiert und getestet. Dabei wurde das erstellte Model über Messungen des tatsächlichen Energieverbrauchs der einzelnen Sensorelemente konfiguriert. An einem komplexen Anwendungsszenario wurden die vom Model geschätzten Verbrauchswerte mit gemessenen Werten überprüft.

7.1 Bewertung der vorgestellten Ergebnisse

Die in dieser Arbeit vorgestellte Sensor-in-the-Loop-Architektur unterstützt den Entwickler von intelligenten Systemen bei einer effektiveren und konsistenten Firmwareentwicklung. Sie ermöglicht ein wiederholbares und reproduzierbares Validieren und Testen von Sensorfirmware. Im Gegensatz zu simulationsbasierten Ansätzen ermöglicht der SiL-Ansatz ein Testen der Firmware direkt auf der Zielhardware. Dies ermöglicht es das gesamte Sensorsystem während des Testens zu beobachten und den Einsatz der Firmware im späteren Produktivsystem zu untersuchen. Der Einsatz der realen Sensorhardware bietet den entscheidenden Vorteil, dass neben der eigentlichen Funktion auch extrafunktionale Eigenschaften überprüft und optimiert werden können. Um das vorgeschlagene Systeme zu benutzen, muss die zu verwendende Sensorhardware bereits während der Entwicklungsphase zur Verfügung stehen. Leider ist dies nicht in jedem Fall gegeben und stellt eine Limitierung des vorgeschlagenen Ansatzes dar. Das System wurde in eine Standard-Entwicklungsumgebung integriert, so ist es möglich in frühen Phasen der Entwicklung die Software zu entwickeln und auf einem Simulator zu testen. Nach Fertigstellung der Hardware kann dann nahtlos auf die SiL-Architektur umgestellt werden, um weitergehende Tests durchzuführen.

Das vorgestellte Konzept wurde auf einem State-of-the-Art intelligenten Sensor beispielhaft implementiert und getestet. Dabei wurden die Speicherauslastung und vor allem das Zeitverhalten der Implementierung untersucht. Der zusätzlich benötigte Speicher für das Instrumentalisierung der Sensor-Firmware liegt bei der vorliegenden Implementierung bei etwa 0,63 % des verfügbaren Gesamtspeichers. Wenn die eigentliche Firmware den kompletten Speicher benötigt, könnte dieser Overhead natürlich ein Problem darstellen. In viele Fälle wird aber gerade in der Entwicklung nicht der gesamte Speicher des Systems benötigt, daher sollte der geringe Overhead tolerierbar sein. Das Zeitverhalten der Implementierung wurde auf unterschiedliche Parameter hin geprüft. Als Erstes wurde die eigentliche Ausführungsgeschwindigkeit beim Lesen der Sensordaten von den eigentlichen Sensoreinheiten und von der SiL-Schnittstelle untersucht und verglichen. Es sind die Daten von drei unterschiedlichen Sensoren mit jeweils drei Sensorachsen mit einer maximalen Samplerate von 1600 Hz gelesen worden. Diese Untersuchungen haben ergeben, dass das Lesen über die gewählte SiL-Schnittstelle etwa viermal schneller ist als das Lesen der Sensordaten über die standard SPI-Schnittstelle. Weiterhin wurde die zeitliche Abweichung beim Lesen der Sensordaten untersucht. Diese Untersuchung wurde in die Bestimmung von unterschiedlichen Arten von Jitter unterteilt. Bei allen Arten von Jitter fällt der ermittelte Wert für die SiL-Implementierung geringer aus als beim Abtasten der Hardwaresensoren. Diese Unterschiede in der Messung der zeitlichen Parameter zeigen, dass die SiL-Implementierung das Verhalten des realen Sensorelementes nicht zu 100 Prozent abbilden kann. Die Abweichungen im Bereich des Jitters sind aber so gering, dass sie auf die Untersuchung der meisten Algorithmen keinen Einfluss haben sollten.

Die vorhandene Implementierung wurde benutzt, um verschiedene Algorithmen zur Orientierungsschätzung basierend auf Inertialsensordaten zu untersuchen. Dabei wurden sowohl die Funktion der Algorithmen als auch ihre extrafunktionalen Eigenschaften untersucht. Das verwendenden der SiL-Implementierung erleichterte das reproduzierbare testen der Algorithmen auf der Zielhardware. Erst durch den Einsatz dieses Systems wurde ein Vergleichen der extrafunktionalen Eigenschaften auf der Sensorhardware möglich.

Die Erweiterung des Systems um eine Komponente zur Manipulation und Erweiterung von Sensordaten schafft einen zusätzlichen Nutzen für den Anwender. Denn darüber ist es möglich das zu untersuchende System mit einer Vielzahl von Stimuli zu testen. Dabei ist es nicht erforderlich zusätzliche Testfälle real aufzunehmen. Aus den bestehenden Aufzeichnungen können unter Verwendung des Frameworks durch Manipulation und Kombination eine Vielzahl von Testfällen generiert werden. All diese Testfälle können dann reproduzierbar direkt auf der Zielhardware untersucht werden.

Durch die Erweiterung des Systems um einen Ansatz zur Schätzung des Energieverbrauchs ist ein Entwickeln von Sensor-Firmware mit Fokus auf den Energieverbrauch des Systems möglich. Die erhaltenen Schätzungen zum Energieverbrauch werden direkt neben den eigentlichen Sensordaten und den daraus resultierenden Ergebnissen visualisiert. Dies ermöglicht es den Energieverbrauch zeitlich in einen Kontext mit den Sensordaten und den bearbeitenden Algorithmen zu bringen. So kann die zu entwickelnde Firmware schon in frühen Phasen der Entwicklung auf Energieeffizienz hin optimiert werden. Natürlich ist die Schätzung des Energieverbrauchs mit einem gewissen Fehler behaftet. In dem untersuchten Beispiel lag diese Fehler allerdings unter 1 %, was für die meisten fälle hinreichen genau sein sollte. Um diese Genauigkeit der Schätzung zu erlangen, ist allerdings eine genaue Konfiguration des Energiemodells notwendig.

Zusammenfassend stellt die in dieser Arbeit vorgestellte Sensor-in-the-Loop-Architektur eine gute Möglichkeit dar, die Entwicklung von Firmware für intelligente Sensoren effizienter und komfortabler zu gestalten. Dabei verbindet sie die Möglichkeit der reproduzierbaren Untersuchung eines Simulationsmodels mit der Beobachtbarkeit von extrafunktionalen Eigenschaften, wie sie nur unter Verwendung eines Hardwareprototyps möglich ist. Die Verwendung des Systems unter realen Bedingung bestätigte den theoretischen Nutzen für die Untersuchung und Entwicklung von Sensor-Firmware. Die verschiedenen Erweiterungen des Systems steigern den Komfort und die praktische Anwendbarkeit für den Entwickler. Weiterhin zeigen sie, dass die vorgestellte SiL-Architektur noch Potenzial zur weiteren Optimierung des Entwicklungsprozesses bietet.

7.2 Ausblick

Die in dieser Abhandlung vorgestellte Sensor-in-the-Loop-Architektur unterstützt Entwickler von intelligenten Sensoren bei der Validierung und Fehleranalyse der Sensor-Firmware. Diese Analyse kann direkt auf dem realen Sensor erfolgen. Die Verwendung und spätere Injektion von Sensordaten macht die Untersuchungen wiederholbar und darüber hinaus können auch reproduzierbare Ergebnisse erzielt werden. Um das Konzept zu verbessern oder zu erweitern ist jedoch weitere Forschungs- und Entwicklungsarbeit notwendig.

Das Steuer- und Visualisierungs-Framework könnte um zusätzliche Komponenten erweitert werden. Erweiterungen welche die Untersuchung zusätzlicher extrafunktionaler Eigenschaften wie etwa das Zeitverhalten erleichtern, könnten den Nutzen steigern. Auch eine Portierung in eine andere IDE wie *Visual Studio Code* wäre denkbar.

In bisherigen Implementierungen und Untersuchungen des Ansatzes kamen ausschließlich Inertialsensoren zum Einsatz. Eine Erweiterung auf andere Sensortypen wie etwa optische oder akustische Systeme wäre denkbar. Es ist zu untersuchen, ob die Entwickler dieser Systeme genau so von dem Konzept profitieren wie es sich in der Arbeit mit Inertialsensoren gezeigt hat.

Generell ist eine Ausweitung der Implementierung auf unterschiedliche intelligente Sensoren vorteilhaft. Besonders in Bezug auf die eingesetzte SPU besteht hier noch viel Potenzial. So können neben der bereits verwendeten Cortex-M Architektur auch andere Architekturen wie beispielsweise RISC-V oder proprietäre SPUs von Sensorherstellern unterstützt werden. In Zukunft werden sehr wahrscheinlich auch Multicore-Systeme in Sensoren eingesetzt werden. Eine Nutzung und Untersuchung des Ansatzes auf solchen Sensorsystemen stellt eine herausfordernde und spannende Aufgabe dar.

Aktuell wird als SiL-Schnittstelle das RTT-Interface den verwendeten Debugger benutzt. Dies erfordert umfangreiche Instrumentalisierung der Sensor-Firmware. Andere Ansätze wie das direkte Einspeisen der Sensordaten über die eigentliche Schnittstelle zwischen SPU und Sensorelement könnten ohne Firmwareanpassung gute Ergebnisse erzielen. Weiterhin könnte eine direkte Schnittstelle in die Sensorregister seitens der Hersteller dieser Sensor-Systeme geschaffen werden.

Bei der Verwendung von intelligenten Sensoren in Anwendungen die besonders zeitkritische sind oder zum Beispiel harte Echtzeit-Anforderungen stellen, ist eine weitere Untersuchung des Zeitverhaltens notwendig. So bildet das aktuelle System das Zeitverhalten nicht vollständig ab. Sollte in der Anwendung bestimmter Sensoren eine wirklich exakte Abbildung des Zeitverhaltens notwendig sein, muss diese extra modelliert werden. Hierbei könnten auch andere Ansätze wie ein direktes Einspeisen über die Sensorschnittstelle oder die Sensorregister zielführend sein.

A Literaturverzeichnis

[A1] H. J. Levinson, *Principles of lithography* (SPIE PM 198), 3. ed. Bellingham, Wash: SPIE Press, 2010, XIV, 504, Includes bibliographical references and index, ISBN: 9780819483249.

[A2] H.-R. Tränkler, Hrsg., *Sensortechnik, Handbuch für Praxis und Wissenschaft* (Technik), 2., völlig neu bearb. Aufl. Berlin [u.a.]: Springer Vieweg, 2014, XIV, 1596, Literaturangaben, ISBN: 9783642299414.

[A3] M. Glück, *MEMS in der Mikrosystemtechnik, Aufbau, Wirkprinzipien, Herstellung und Praxiseinsatz mikroelektromechanischer Schaltungen und Sensorsysteme ; mit 6 Tabellen*, 1. Auflage. Wiesbaden: Teubner, 2005, 212 Seiten, Literaturverzeichnis: Seite [201]-205, ISBN: 3519005204.

[A4] Ltd. KOHDEN Co. „Principle of the AMR sensor“. (2011), Adresse: `https://www.hkd.co.jp/english/amr_tec_amr/` (besucht am 03. 02. 2022).

[A5] Bosch Sensortec. „Accelerometers“. (2022), Adresse: `https://www.bosch-sensortec.com/products/motion-sensors/accelerometers/` (besucht am 22. 02. 2022).

[A6] STMicroelectronics. „Accelerometers“. (2022), Adresse: `https://www.st.com/en/mems-and-sensors/accelerometers.html` (besucht am 22. 02. 2022).

[A7] STMicroelectronics. „Gyroscopes“. (2022), Adresse: `https://www.st.com/en/mems-and-sensors/gyroscopes.html` (besucht am 22. 02. 2022).

[A8] Bosch Sensortec. „BMG250 Gyroscope“. (2022), Adresse: `https://www.bosch-sensortec.com/products/motion-sensors/gyroscopes-bmg250/` (besucht am 22. 02. 2022).

[A9] Bosch Sensortec. „BMM150 Magnetometer“. (2022), Adresse: `https://www.bosch-sensortec.com/products/motion-sensors/magnetometers-bmm150/` (besucht am 22. 02. 2022).

[A10] STMicroelectronics. „e-Compasses“. (2022), Adresse: `https://www.st.com/en/mems-and-sensors/e-compasses.html` (besucht am 22. 02. 2022).

[A11] Bosch Sensortec. „A powerful combination: IMUs“. (2022), Adresse: `https://www.bosch-sensortec.com/products/motion-sensors/imus/` (besucht am 22. 02. 2022).

[A12] STMicroelectronics. „iNEMO-Inertial Modules“. (2022), Adresse: https://www.st.com/en/mems-and-sensors/inemo-inertial-modules.html (besucht am 22. 02. 2022).

[A13] Bosch Sensortec. „Barometric pressure sensors“. (2022), Adresse: https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/ (besucht am 22. 02. 2022).

[A14] STMicroelectronics. „Proximity Sensors“. (2022), Adresse: https://www.st.com/en/mems-and-sensors/proximity-sensors.html (besucht am 22. 02. 2022).

[A15] P. Chlebis, J. Pumr und P. Simonik, „Current sensor with the DSP“, in *2010 International Conference on Applied Electronics*, 2010, S. 1–4.

[A16] M. D. Nil, L. Yseboodt, F. Bouwens, J. Hulzink, M. Berekovic, J. Huisken und J. van Meerbergen, „Ultra Low Power ASIP Design for Wireless Sensor Nodes“, in *2007 14th IEEE International Conference on Electronics, Circuits and Systems*, IEEE, Dez. 2007. DOI: 10.1109/icecs.2007.4511249.

[A17] Jean-Luc Aufranc (CNXSoft). „STMicro Intelligent Sensor Processing Unit (ISPU) combines MEMS sensor with DSP for AI “in the edge”“. (2022), Adresse: https://www.cnx-software.com/2022/02/21/stmicro-intelligent-sensor-processing-unit-ispu-combines-mems-sensor-with-dsp-for-ai-in-the-edge/ (besucht am 21. 02. 2022).

[A18] Bosch Sensortec, *Data sheet BMF055 Custom programmable 9-axis motion sensor*, 2020. Adresse: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmf055-ds000.pdf (besucht am 16. 03. 2021).

[A19] STMicroelectronics. „STM32CubeIDE Integrated Development Environment for STM32“. (2022), Adresse: https://www.st.com/en/development-tools/stm32cubeide.html (besucht am 04. 03. 2022).

[A20] Microchip Technology Inc. „Microchip Studio for AVR® and SAM Devices“. (2022), Adresse: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio (besucht am 04. 03. 2022).

[A21] Eclipse Foundation. „Eclipse IDE The Leading Open Platform for Professional Developers“. (2022), Adresse: https://www.eclipse.org/eclipseide/ (besucht am 04. 03. 2022).

[A22] Microsoft. „Code schnellerintelligenter Arbeiten Gestalten Sie die Zukunft mit Visual Studio 2022“. (2022), Adresse: https://visualstudio.microsoft.com/de/vs/ (besucht am 04. 03. 2022).

[A23] JTAG Technologies. „What is JTAG boundary-scan?“ (2022), Adresse: https://www.jtag.com/what-is-jtag-boundary-scan/ (besucht am 04. 03. 2022).

[A24] Nicolas Oberli. „SWD – ARM's alternative to JTAG“. (2019), Adresse: https://research.kudelskisecurity.com/2019/05/16/swd-arms-alternative-to-jtag/ (besucht am 16. 05. 2019).

[A25] P. Banerjee, N. Debnath und A. Sarkar, „Extra-Functional Properties Driven Component Selection for Component based System“, in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, IEEE, Juli 2018. DOI: 10.1109/indin.2018.8472067.

[A26] S. A. Asadollah, R. Inam und H. Hansson, „A Survey on Testing for Cyber Physical System“, in *Testing Software and Systems*, Springer International Publishing, 2015, S. 194–207. DOI: 10.1007/978-3-319-25945-1_12.

[A27] X. Zhou, X. Gou, T. Huang und S. Yang, „Review on Testing of Cyber Physical Systems: Methods and Testbeds“, *IEEE Access*, Jg. 6, S. 52 179–52 194, 2018. DOI: 10.1109/access.2018.2869834.

[A28] The MathWorks, Inc. „MATLAB Mathematik. Grafiken. Programmierung.“ (2022), Adresse: https://de.mathworks.com/products/matlab.html (besucht am 10. 03. 2022).

[A29] John W. Eaton. „GNUOctave ScientificProgrammingLanguage“. (2022), Adresse: https://www.gnu.org/software/octave/index (besucht am 10. 03. 2022).

[A30] NumPy. „NumPy The fundamental package for scientific computing with Python“. (2022), Adresse: https://numpy.org/ (besucht am 10. 03. 2022).

[A31] SciPy. „SciPy Fundamental algorithms for scientific computing in Python“. (2022), Adresse: https://scipy.org/ (besucht am 10. 03. 2022).

[A32] dSPACE, *Sensor Simulation PC*, https://www.dspace.com/en/pub/home/products/hw/simulator_hardware/sensorsim_pc.cfm, [Online; accessed 04-November-2019], 2019.

[A33] MatLab, *Sensor Fusion and Tracking Toolbox*, https://de.mathworks.com/products/sensor-fusion-and-tracking.html, [Online; accessed 04-November-2019], 2019.

[A34] Aceinna, *GNSS-INS-SIM*, https://github.com/Aceinna/gnss-ins-sim, [Online; accessed 23-October-2019], 2019.

[A35] V. Herdt, D. Große, H. M. Le und R. Drechsler, „Extensible and Configurable RISC-V based Virtual Prototype“, in *Forum on Specification and Design Languages*, 2018, S. 5–16.

[A36] OVP. „Open Virtual Platforms“. (2021), Adresse: https://www.ovpworld.org (besucht am 21. 04. 2021).

[A37] X. Dai, C. Ke, Q. Quan und K.-Y. Cai, „Simulation Credibility Assessment Methodology With FPGA-based Hardware-in-the-Loop Platform“, *IEEE Transactions on Industrial Electronics*, Jg. 68, Nr. 4, S. 3282–3291, Apr. 2021. DOI: 10.1109/tie.2020.2982122.

[A38] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere und P.-E. Gaillardon, „OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs“, in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, Sep. 2019. DOI: 10.1109/fpl.2019.00065.

[A39] Bosch Sensortec, *Application Boards*, https://www.bosch-sensortec.com/bst/support_tools/application_boards/overview_application_boards, [Online; accessed 05-November-2019], 2019.

[A40] Analog Devices, *Inertial MEMS Sensor Evaluation Tools*, https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-platforms/inertial-mems-sensor-evaluation-tools.html, [Online; accessed 05-November-2019], 2019.

[A41] M. Mousavi, A. Somà und F. Pescarmona, „Effects of substituting anthropometric joints with revolute joints in humanoid robots and robotic hands: a case study“, *Robotica*, Jg. 32, Nr. 4, S. 501–513, Aug. 2013. DOI: 10.1017/s0263574713000817.

[A42] B. S. ScienceSoft USA Corporation. „Vorgehensmodelle der Softwareentwicklung“. (2019), Adresse: https://www.scnsoft.de/blog/vorgehensmodelle-der-softwareentwicklung (besucht am 17. 03. 2022).

[A43] P. Koch, M. Dreier, M. Maass, M. Böhme, H. Phan und A. Mertins, „A Recurrent Neural Network for Hand Gesture Recognition based on Accelerometer Data“, in *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, 2019, S. 5088–5091.

[A44] B. Fang, Q. Lv, J. Shan, F. Sun, H. Liu, D. Guo und Y. Zhao, „Dynamic Gesture Recognition Using Inertial Sensors-based Data Gloves“, in *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, Juli 2019. DOI: 10.1109/icarm.2019.8834314.

[A45] K. Wen, K. Yu, Y. Li, S. Zhang und W. Zhang, „A New Quaternion Kalman Filter Based Foot-Mounted IMU and UWB Tightly-Coupled Method for Indoor Pedestrian Navigation“, *IEEE Transactions on Vehicular Technology*, Jg. 69, Nr. 4, S. 4340–4352, 2020.

[A46] X. Liu, N. Li, G. Xu und Y. Zhang, „A Novel Robust Step Detection Algorithm for Foot-Mounted IMU“, *IEEE Sensors Journal*, Jg. 21, Nr. 4, S. 5331–5339, Feb. 2021. DOI: 10.1109/jsen.2020.3030771.

[A47] N. Koksal, M. Jalalmaab und B. Fidan, „Adaptive linear quadratic attitude tracking control of a quadrotor UAV based on IMU sensor data fusion“, *Sensors*, Jg. 19, Nr. 1, S. 46, 2019.

[A48] L. Bigazzi, S. Gherardini, G. Innocenti und M. Basso, „Development of Non Expensive Technologies for Precise Maneuvering of Completely Autonomous Unmanned Aerial Vehicles“, *Sensors*, Jg. 21, Nr. 2, S. 391, Jan. 2021. DOI: 10.3390/s21020391.

[A49] A. Solanki, K. Prasad, R. Oreilly und Y. Singhal, „Inertial MEMS Test Challenges“, in *2011 IEEE 17th International Mixed-Signals, Sensors and Systems Test Workshop*, IEEE, Mai 2011. DOI: 10.1109/ims3tw.2011.12.

[A50] B. A. Krishna und A. S. Pillai, „Digital sensor simulation frame work for hardware-in-the-loop testing“, in *International Conference on Intelligent Computing, Instrumentation and Control Technologies*, IEEE, 2017.

[A51] Bosch Sensortec. „BMA280 Digital, triaxial acceleration sensor“. (2020), Adresse: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmi055-ds000.pdf (besucht am 15. 10. 2021).

[A52] Bosch Sensortec. „BMG160 Digital, triaxial gyroscope sensor“. (2020), Adresse: https://media.digikey.com/pdf/Data%20Sheets/Bosch/BMG160.pdf (besucht am 15. 10. 2021).

[A53] Bosch Sensortec. „BMM150 Geomagnetic Sensor“. (2020), Adresse: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmm150-ds001.pdf (besucht am 15. 10. 2021).

[A54] Microchip Technology. „ATSAMD20J18 - Arm Cortex-M0+ MCU“. (2021), Adresse: https://www.microchip.com/wwwproducts/en/ATSAMD20J18 (besucht am 03. 08. 2021).

[A55] Bosch Sensortec. „BMF055 Example Project – Data Stream“. (2015), Adresse: https://www.bosch-sensortec.com/media/boschsensortec/downloads/application_notes_1/bst-bmf055-ex001-01.pdf (besucht am 22. 04. 2022).

[A56] Arm Limited. „arm Developer GNU Toolchain“. (2022), Adresse: https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain (besucht am 29. 06. 2022).

[A57] SEGGER, *Real Time Transfer*, https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/, [Online; accessed 05-November-2019], 2019.

[A58] E. Peguero, M. Labrador und B. Cook, „Assessing Jitter in Sensor Time Series from Android Mobile Devices“, in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, Mai 2016. DOI: 10.1109/smartcomp.2016.7501679.

[A59] Saleae. „Saleae Tech Specs“. (2022), Adresse: https://www.saleae.com/#section-tech-specs (besucht am 27. 04. 2022).

[A60] Y. Kobayashi, T. Kimura und H. Fujioka, „A servo motor control with sampling jitters“, in *2010 11th IEEE International Workshop on Advanced Motion Control (AMC)*, IEEE, März 2010. DOI: 10.1109/amc.2010.5464023.

[A61] D. Niculae, C. Plaisanu und D. Bistriceanu, „Sampling jitter compensation for numeric PID controllers“, in *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, IEEE, 2008. DOI: 10.1109/aqtr.2008.4588802.

[A62] P. Marti, J. Fuertes, G. Fohler und K. Ramamritham, „Jitter compensation for real-time control systems“, in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, IEEE Comput. Soc, 2001. DOI: 10.1109/real.2001.990594.

[A63] Tektronix. „Understanding and Characterizing Timing Jitter Primer“. (2012), Adresse: https://download.tek.com/document/55W_16146_5_MR_Letter.pdf (besucht am 17. 03. 2021).

[A64] Silicon Laboratories. „AN687 - A PRIMER ON JITTER, JITTER MEASUREMENT AND PHASE-LOCKED LOOPS“, Silicon Laboratories. (2012), Adresse: https://www.silabs.com/documents/public/application-notes/AN687.pdf (besucht am 17. 03. 2021).

[A65] H. Mitchell. „Timing JitterTutorial & Measurement Guide“. (2019), Adresse: https://www.silabs.com/documents/public/white-papers/timing-jitter-tutorial-and-measurement-guide-ebook.pdf (besucht am 17. 03. 2021).

[A66] K. Smeds und X. Lu, „Effect of sampling jitter and control jitter on positioning error in motion control systems“, *Precision Engineering*, Jg. 36, Nr. 2, S. 175–192, Apr. 2012. DOI: 10.1016/j.precisioneng.2011.09.002.

[A67] K. W. D. und J. D. Raffaldi., „An introduction to gage R&R: gage R&R studies can help operators from making costly measurement errors.“, *Quality*, Jg. 44, Nr. 13, S. 24+, 2005.

[A68] J.-N. Pan, „Evaluating the Gauge Repeatability and Reproducibility for Different Industries“, *Quality and Quantity*, Jg. 40, Nr. 4, S. 499–518, Aug. 2006. DOI: 10.1007/s11135-005-1100-y.

[A69] A. Osma, „An assessment of the robustness of gauge repeatability and reproducibility analysis in automotive components“, *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, Jg. 225, Nr. 7, S. 895–912, Mai 2011. DOI: 10.1177/0954407011401504.

[A70] R. K. Burdick, C. M. Borror und D. C. Montgomery, „A Review of Methods for Measurement Systems Capability Analysis“, *Journal of Quality Technology*, Jg. 35, Nr. 4, S. 342–354, Okt. 2003. DOI: 10.1080/00224065.2003.11980232.

[A71] R. PV. „Gage Repeatability and Reproducibility (R&R)“. (2014), Adresse: https://www.qualitymag.com/articles/83529-quality-101-an-introduction-to-gage-r-r (besucht am 04. 08. 2021).

[A72] Minitab. „Methods and formulas for ANOVA table in Crossed Gage R&R Study“. (2020), Adresse: https://support.minitab.com/en-us/minitab/19/help-and-how-to/quality-and-process-improvement/measurement-system-analysis/how-to/gage-study/crossed-gage-r-r-study/methods-and-formulas/anova-table/ (besucht am 26. 06. 2021).

[A73] R. Bevans. „Confidence intervals explained“. (2020), Adresse: https://www.scribbr.com/statistics/confidence-interval/ (besucht am 29. 06. 2021).

[A74] TU Dortmund. „Tabelle der t-Verteilung“. (2022), Adresse: https://www.statistik.tu-dortmund.de/fileadmin/user_upload/Lehrstuehle/Oekonometrie/Lehre/WiSoOekoSS17/tabelletV.pdf (besucht am 06. 05. 2022).

[A75] G. Lammel, „The future of MEMS sensors in our connected world“, in *2015 28th IEEE International Conference on Micro Electro Mechanical Systems (MEMS)*, IEEE, Jan. 2015. DOI: 10.1109/memsys.2015.7050886.

[A76] J. M. Jr., M. Vieira, M. Pires und S. S. Jr., „Sensor Fusion and Smart Sensor in Sports and Biomedical Applications“, *Sensors*, Jg. 16, Nr. 10, S. 1569, Sep. 2016. DOI: 10.3390/s16101569.

[A77] G. W. Hunter, J. R. Stetter, P. Hesketh und C.-C. Liu, „Smart Sensor Systems“, *The Electrochemical Society Interface*, Jg. 19, Nr. 4, S. 29–34, 2010. DOI: 10.1149/2.f03104if.

[A78] STMicroelectronics. „STM32F103CB - 32-bit SAM Microcontrollers“. (2022), Adresse: https://www.st.com/en/microcontrollers-microprocessors/stm32f103cb.html (besucht am 22. 03. 2022).

[A79] Vectornav. „Attitude & Heading Reference System (AHRS)“. (2022), Adresse: https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-ahrs (besucht am 22. 03. 2022).

[A80] Helicoptermaintenancemagazine. „A Layman's Guide to Attitude Heading Reference Systems (AHRS)“. (2022), Adresse: https://helicoptermaintenancemagazine.com/article/layman%E2%80%99s-guide-attitude-heading-reference-systems-ahrs (besucht am 22. 03. 2022).

[A81] S. O. H. Madgwick, A. J. L. Harrison und R. Vaidyanathan, „Estimation of IMU and MARG orientation using a gradient descent algorithm“, in *2011 IEEE International Conference on Rehabilitation Robotics*, IEEE, Juni 2011. DOI: 10.1109/icorr.2011.5975346.

[A82] S. O. Madgwick. „An efficient orientation filter for inertial andinertial/magnetic sensor arrays“. (2010), Adresse: https://www.samba.org/tridge/UAV/madgwick_internal_report.pdf (besucht am 22. 03. 2022).

[A83] R. Mahony, T. Hamel und J.-M. Pflimlin, „Nonlinear Complementary Filters on the Special Orthogonal Group“, *IEEE Transactions on Automatic Control*, Jg. 53, Nr. 5, S. 1203–1218, Juni 2008. DOI: 10.1109/tac.2008.923738.

[A84] G. Baldwin, R. Mahony, J. Trumpf, T. Hamel und T. Cheviron, „Complementary filter design on the Special Euclidean group SE(3)“, in *2007 European Control Conference (ECC)*, IEEE, Juli 2007. DOI: 10.23919/ecc.2007.7068746.

[A85] H. J. Fakhri Alam Zhou ZhaiHe, „A Comparative Analysis of Orientation Estimation Filters using MEMS based IMU“, in *2nd International Conference on Research in Science, Engineering and Technology (ICRSET'2014), March 21-22, 2014 Dubai (UAE)*, International Institute of Engineers, März 2014. DOI: 10.15242/iie.e0314552.

[A86] P. Gui, L. Tang und S. Mukhopadhyay, „MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion“, in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, IEEE, Juni 2015. DOI: 10.1109/iciea.2015.7334442.

[A87] H. Teague, „Comparison of attitude estimation techniques for low-cost unmanned aerial vehicles“, *arXiv preprint arXiv:1602.07733*, 2016.

[A88] A. Cavallo, A. Cirillo, P. Cirillo, G. D. Maria, P. Falco, C. Natale und S. Pirozzi, „Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation“, *IFAC Proceedings Volumes*, Jg. 47, Nr. 3, S. 7585–7591, 2014. DOI: 10.3182/20140824-6-za-1003.01173.

[A89] R. S. McGinnis, S. M. Cain, S. P. Davidson, R. V. Vitali, S. G. McLean und N. C. Perkins, „Validation of Complementary Filter Based IMU Data Fusion for Tracking Torso Angle and Rifle Orientation“, in *Volume 3: Biomedical and Biotechnology Engineering*, American Society of Mechanical Engineers, Nov. 2014. DOI: 10.1115/imece2014-36909.

[A90] C.-L. Lin, W.-C. Chiu, T.-C. Chu, Y.-H. Ho, F.-H. Chen, C.-C. Hsu, P.-H. Hsieh, C.-H. Chen, C.-C. K. Lin, P.-S. Sung und P.-T. Chen, „Innovative Head-Mounted System Based on Inertial Sensors and Magnetometer for Detecting Falling Movements“, *Sensors*, Jg. 20, Nr. 20, S. 5774, Okt. 2020. DOI: 10.3390/s20205774.

[A91] R. G. Valenti, I. Dryanovski und J. Xiao, „Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs“, *Sensors*, Jg. 15, Nr. 8, S. 19 302–19 330, 2015, ISSN: 1424-8220. DOI: 10.3390/s150819302. Adresse: https://www.mdpi.com/1424-8220/15/8/19302.

[A92] M. I. Ribeiro, „Kalman and Extended Kalman Filters: Concept“, *Derivation and Properties*, Jg. 31, S. 41, 2004.

[A93] E. Wan, „Sigma-Point Filters: An Overview with Applications to Integrated Navigation and Vision Assisted Control“, in *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, 2006, S. 201–202. DOI: 10.1109/NSSPW.2006.4378854.

[A94] E. B. Dam, M. Koch und M. Lillholm, *Quaternions, interpolation and animation* (Rapport / DIKU, Datalogisk Institut, Københavns Universitet 98,5). København: DIKU, 1998, IV, 98.

[A95] E. G. Hemingway und O. M. O'Reilly, „Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments“, *Multibody System Dynamics*, Jg. 44, Nr. 1, S. 31–56, 2018.

[A96] D. Goldberg, „What Every Computer Scientist Should Know about Floating-Point Arithmetic“, *ACM Comput. Surv.*, Jg. 23, Nr. 1, S. 5–48, März 1991, ISSN: 0360-0300. DOI: 10.1145/103162.103163. Adresse: https://doi.org/10.1145/103162.103163.

[A97] E. L. Oberstar, „Fixed-point representation & fractional math“, *Oberstar Consulting*, Jg. 9, 2007.

[A98] Bosch Sensortec. „BMI055 Small, versatile 6DoF sensor module“. (2021), Adresse: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmi055-ds000.pdf (besucht am 01. 11. 2021).

[A99] L. Middendorf, R. Dorsch, R. Bichler, C. Strohrmann und C. Haubelt, „A Mobile Camera-Based Evaluation Method of Inertial Measurement Units on Smartphones“, in *International Internet of Things Summit*, Springer, 2015, S. 362–372.

[A100] N. Büscher, L. Middendorf, C. Haubelt, R. Dorsch und F. Wegelin, „Statistical analysis and improvement of the repeatability and reproducibility of an evaluation method for IMUs on a smartphone“, in *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM, Juni 2016. DOI: 10.1145/2933242.2933255.

[A101] OpenCV team. „OpenCV“. (2022), Adresse: https://opencv.org/ (besucht am 25. 03. 2022).

[A102] T. Hessing. „Gage Repeatability and Reproducibility (R&R)“. (2019), Adresse: https://www.qualitymag.com/articles/83529-quality-101-an-introduction-to-gage-r-r (besucht am 13. 04. 2022).

[A103] Eigen. „Eigen“. (2022), Adresse: http://eigen.tuxfamily.org/index.php?title=Main_Page (besucht am 13. 04. 2022).

[A104] Hewlett Packard Enterprise. „OpenGL“. (2022), Adresse: https://www.opengl.org/ (besucht am 13. 04. 2022).

[A105] Mesa. „The Mesa 3D Graphics Library“. (2022), Adresse: https://www.mesa3d.org/ (besucht am 13. 04. 2022).

[A106] J. Blinn, „Floating-point tricks“, *IEEE Computer Graphics and Applications*, Jg. 17, Nr. 4, S. 80–84, 1997. DOI: 10.1109/38.595279.

[A107] IDSoftware. „Quake-III-Arena Source Code“. (2012), Adresse: https://github.com/id-Software/Quake-III-Arena/blob/master/code/game/q_math.c#L552 (besucht am 14. 04. 2022).

[A108] *Sicherheit von Maschinen, Sicherheitsbezogene Teile von Steuerungen* (dt. Normen DIN EN ISO 13849). Berlin: Beuth, 2013, Bd. Teil 2, 98 S., Ersatz für DIN EN ISO 13849-2:2008-09 und DIN EN ISO 13849-2 Berichtigung 1:2009-01.

[A109] *IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems*, International Electrotechnical Commision, 2010.

[A110] J. Brownlee. „How Much Training Data is Required for Machine Learning?“ (2017), Adresse: https://machinelearningmastery.com/much-training-data-required-machine-learning/ (besucht am 01. 07. 2021).

[A111] N. Dawar, S. Ostadabbas und N. Kehtarnavaz, „Data Augmentation in Deep Learning-Based Fusion of Depth and Inertial Sensing for Action Recognition“, *IEEE Sensors Letters*, Jg. 3, Nr. 1, S. 1–4, 2019. DOI: 10.1109/LSENS.2018.2878572.

[A112] L. Tran und D. Choi, „Data Augmentation for Inertial Sensor-Based Gait Deep Neural Network“, *IEEE Access*, Jg. 8, S. 12 364–12 378, 2020. DOI: 10.1109/ACCESS.2020.2966142.

[A113] . Aceinna. „Open-source GNSS + inertial navigation, sensor fusion simulator“. (2021), Adresse: https://github.com/Aceinna/gnss-ins-sim (besucht am 02. 07. 2021).

[A114] . Mathworks. „Sensor Fusion and Tracking Toolbox“. (2021), Adresse: https://de.mathworks.com/products/sensor-fusion-and-tracking.html (besucht am 02. 07. 2021).

[A115] G. Ligorio und A. M. Sabatini, „A Simulation Environment for Benchmarking Sensor Fusion-Based Pose Estimators“, *Sensors*, Jg. 15, Nr. 12, S. 32 031–32 044, 2015. DOI: 10.3390/s151229903.

[A116] A. D. Young, M. J. Ling und D. K. Arvind, „IMUSim: A simulation environment for inertial sensing algorithm design and evaluation“, in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2011, S. 199–210.

[A117] P. Bourke. „Generating noise with different power spectra laws“. (1998), Adresse: http://paulbourke.net/fractals/noise/ (besucht am 16. 06. 2021).

[A118] J. O. S. III. „Example: Synthesis of 1/F Noise (Pink Noise)“. (2020), Adresse: https://ccrma.stanford.edu/~jos/sasp/Example_Synthesis_1_F_Noise.html (besucht am 16. 06. 2021).

[A119] J. O. S. III. „Example: Pink Noise Analysis“. (2020), Adresse: https://ccrma.stanford.edu/~jos/sasp/Example_Pink_Noise_Analysis.html (besucht am 16. 06. 2021).

[A120] A. G. Quinchia, G. Falco, E. Falletti, F. Dovis und C. Ferrer, „A Comparison between Different Error Modeling of MEMS Applied to GPS/INS Integrated Systems“, *Sensors*, Jg. 13, Nr. 8, S. 9549–9588, 2013. DOI: 10.3390/s130809549.

[A121] Analog Devices. „Accelerometer Specifications - Quick Definitions“. (2021), Adresse: https://www.analog.com/en/products/landing-pages/001/accelerometer-specificatios-definitions.html (besucht am 17. 06. 2021).

[A122] M. Ghose und A. Pradhan, „A Hand Gesture Recognition using Feature Extraction“, *International Journal of Current Engineering and Technology (IJCET)*, Jg. 2, S. 323–327, Jan. 2012.

[A123] V. Raghunathan und P. H. Chou, „Design and Power Management of Energy Harvesting Embedded Systems“, in *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, Ser. ISLPED '06, Tegernsee, Bavaria, Germany: Association for Computing Machinery, 2006, S. 369–374, ISBN: 1595934626. DOI: 10.1145/1165573.1165663.

[A124] C. Guo, S. Ci, Y. Zhou und Y. Yang, „A Survey of Energy Consumption Measurement in Embedded Systems“, *IEEE Access*, Jg. 9, S. 60 516–60 530, 2021. DOI: 10.1109/access.2021.3074070.

[A125] P. Ciancarini, S. Ergasheva, Z. Kholmatova, A. Kruglov, G. Succi, X. Vasquez und E. Zuev, „Analysis of Energy Consumption of Software Development Process Entities“, *Electronics*, Jg. 9, Nr. 10, S. 1678, Okt. 2020. DOI: 10.3390/electronics9101678.

[A126] Microchip, *Power Debugger*, https://www.microchip.com/DevelopmentTools/ProductDetails/ATPOWERDEBUGGER, [Online; accessed 26-Juli-2021], 2019.

[A127] L.-B. Chen, Y.-L. Chen und I.-J. Huang, „A Real-Time Power Analysis Platform for Power-Aware Embedded System Development“, *Journal of Information Science and Engineering*, Jg. 27, S. 1165–1182, Jan. 2011.

[A128] T. Simunic, L. Benini und G. De Micheli, „Cycle-accurate simulation of energy consumption in embedded systems“, in *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, 1999, S. 867–872. DOI: 10.1109/DAC.1999.782199.

[A129] K. Grüttner, P. A. Hartmann, T. Fandrey, K. Hylla, D. Lorenz, S. Stattelmann, B. Sander, O. Bringmann, W. Nebel und W. Rosenstiel, „An ESL timing amp; power estimation and simulation framework for heterogeneous socs“, in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, 2014, S. 181–190. DOI: 10.1109/SAMOS.2014.6893210.

[A130] V. Tiwari und M. T.-C. Lee, „Power Analysis of a 32-Bit Embedded Microcontroller“, in *Proceedings of the 1995 Asia and South Pacific Design Automation Conference*, Ser. ASP-DAC '95, Makuhari, Massa, Chiba, Japan: Association for Computing Machinery, 1995, 23–es, ISBN: 0897917669. DOI: 10.1145/224818.224890. Adresse: https://doi.org/10.1145/224818.224890.

[A131] R. Dochia, D. Bogdan und C. Burileanu, „Model for software power estimation of an 8-bit microcontroller“, in *CAS 2011 Proceedings (2011 International Semiconductor Conference)*, Bd. 2, 2011, S. 443–446. DOI: 10.1109/SMICND.2011.6095842.

[A132] J. Pallister, S. Kerrison, J. Morse und K. Eder, „Data Dependent Energy Modeling for Worst Case Energy Consumption Analysis“, in *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*, ACM, Juni 2017. DOI: 10.1145/3078659.3078666.

[A133] NXP Semiconductors. „KINETIS-PET: Kinetis PowerEstimationTool“. (2021), Adresse: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/kv-series-cortex-m4-m0-plus-m7/kinetis-power-estimation-tool:KINETIS-PET (besucht am 01. 09. 2021).

[A134] Microchip Technology Inc. „Power Debugging Module“. (2021), Adresse: https://onlinedocs.microchip.com/pr/GUID-F897CF19-8EAC-457A-BE11-86BDAC9B59CF-en-US-10/index.html?GUID-590491AB-3C05-4C73-B4D4-DD739B095630 (besucht am 01. 09. 2021).

[A135] Bosch Sensortec. „Bosch Sensortec Sensor Drivers“. (2021), Adresse: https://www.bosch-sensortec.com/software-tools/software/drivers/ (besucht am 12. 10. 2021).

[A136] datatec, *CX3300A Series Device Current Waveform Analyzer*, https://www.datatec.de/media/pdf/4c/74/02/Keysight_CX3300A.pdf, "[Online; accessed 03-June-2021]", 2019. Adresse: https://www.datatec.de/media/pdf/4c/74/02/Keysight_CX3300A.pdf (besucht am 03. 06. 2021).

[A137] STMicroelectronics, *Optimizing power and performance with STM32L4 and STM32L4+ Series microcontrollers AN4746*, https://www.st.com/content/ccc/resource/technical/document/application_note/5c/cb/90/97/

4b/84/4e/81/DM00216518.pdf/files/DM00216518.pdf/jcr:content/translations/en.DM00216518.pdf, [Online; accessed 01-09-2021], 2021.

[A138] NXP Semiconductors, *MKW40Z Power Consumption Analysis*, https://www.nxp.com/docs/en/application-note/AN5272.pdf, [Online; accessed 01-09-2021], 2016.

B Liste der Veröffentlichungen und Fachvorträge auf Tagungen

[B1] D. Gis, A. Fink und H. Beikirch, „Bluetooth Enabled MARG Platform for Real-Time Orientation Sensing“, in *6th IEEE Int. Conf. on Indoor Positioning and Indoor Navigation (IPIN)*, 2015, S. 1–2.

[B2] J.-P. Wolff, S. Stieber, F. Holzke, D. Gis, C. Haubelt, P. Lepidis und J. Meier, „Improving Pedestrian Dead Reckoning using Likely and Unlikely Paths“, in *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, IEEE, März 2018. DOI: 10.1109/upinlbs.2018.8559709.

[B3] N. Büscher, D. Gis, S. Stieber und C. Haubelt, „Multi-aspect Evaluation Method for Digital Pointing Devices“, in *Proceedings of the 9th International Conference on Pervasive and Embedded Computing and Communication Systems*, SCITEPRESS - Science und Technology Publications, 2019. DOI: 10.5220/0008355701280135.

[B4] R. Dorsch, C. Haubelt, S. Stieber, D. Gis und N. Büscher, „Verfahren zum Verarbeiten der Sensorsignale einer Drehratensensoranordnung, Vorrichtung und Verfahren zum Betreiben der Vorrichtung“, DE102020205943B3, Mai 2021.

[B5] D. Gis, N. Büscher und C. Haubelt, „Advanced Debugging Architecture for Smart Inertial Sensors using Sensor-in-the-Loop“, in *2020 International Workshop on Rapid System Prototyping (RSP)*, IEEE, Sep. 2020. DOI: 10.1109/rsp51120.2020.9244851.

[B6] D. Gis, N. Büscher und C. Haubelt, „Investigation of Timing Behavior and Jitter in a Smart Inertial Sensor Debugging Architecture“, *Sensors*, Jg. 21, Nr. 14, S. 4675, Juli 2021. DOI: 10.3390/s21144675.

[B7] D. Gis, C. Haubelt und N. Büscher, „Verfahren zur Überprüfung, Evaluation und/oder Fehlerdiagnose eines Sensorsystems, Sensorsystem und System“, dt. Pat. DE102021200244A1, März 2022.

[B8] D. Gis, C. Haubelt und N. Büscher, „Method for checking, evaluation, and/or error diagnosis of a sensor system, sensor system, and system Method for checking, evaluation, and/or error diagnosis of a sensor system, sensor system, and system“, US-Pat. US20220095022A1, März 2022.

[B9] N. Büscher, D. Gis, V. Kühn und C. Haubelt, „On the Functional and Extra-Functional Properties of IMU Fusion Algorithms for Body-Worn Smart Sensors“, *Sensors*, Jg. 21, Nr. 8, S. 2747, Apr. 2021. DOI: 10.3390/s21082747.

[B10] N. Büscher, D. Gis, J. P. Wolff und C. Haubelt, „Data Augmentation Framework for Smart Sensor System Development Using the Sensor-in-the-Loop Prototyping Platform“, in *2021 IEEE International Workshop on Rapid System Prototyping (RSP)*, IEEE, Okt. 2021. DOI: 10.1109/RSP53691.2021.9806209.

[B11] D. Gis, N. Büscher und C. Haubelt, „Real-Time Power Analysis of Smart Sensors Using Advanced Debugging Methods“, *Micromachines*, Jg. 12, Nr. 11, 2021, ISSN: 2072-666X. DOI: 10.3390/mi12111276. Adresse: https://www.mdpi.com/2072-666X/12/11/1276.

[B12] J. Rudolf, D. Gis, S. Stieber, C. Haubelt und R. Dorsch, „SystemC Power Profiling for IoT Device Firmware using Runtime Configurable Models“, in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, Juni 2019. DOI: 10.1109/meco.2019.8759994.