

FAKULTÄT INGENIEURWISSENSCHAFTEN

5420 - EMBEDDED SYSTEMS I

---

**STM32 Positionsbestimmung  
mit MEM - Sensoren**

---

*Autor* Leo Kilian  
*Prüfer* Prof. Pretschner

17. März 2024

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>3</b>
<b>Tabellenverzeichnis</b>	<b>4</b>
<b>1. Aufgabenstellung</b>	<b>1</b>
<b>2. Grundlagen</b>	<b>2</b>
2.1. Komponenten . . . . .	2
2.1.1. Entwicklungsboard STM32 Nucleo-144 . . . . .	4
2.1.2. Magnetometer LIS2MDL . . . . .	4
2.1.3. Beschleunigungssensor LIS2DW12 . . . . .	5
2.2. I2C Kommunikation . . . . .	6
<b>3. Planung der Umsetzung</b>	<b>8</b>
3.1. Konfiguration des ARM®Coretx®-M3 Prozessor . . . . .	8
3.2. Installation und Implementation der MEM - Pakete . . . . .	11
3.3. Integration . . . . .	13
3.3.1. Hardware Abstraction Layer . . . . .	13
3.3.2. MEM-Paket . . . . .	14
<b>4. Aufnahme _ physikalischer</b>	<b>15</b>
4.1. Programmablauf . . . . .	15
4.2. Initialisierung, Implementierung, Instanzienierung . . . . .	16
4.3. Setup / Config . . . . .	17
4.4. Loop . . . . .	18
<b>5. Positionsbestimmung mit MEM Sensoren</b>	<b>19</b>
5.1. Grundlagen . . . . .	19
5.1.1. Navigation . . . . .	19
5.1.2. Nick-, Roll- und Gierwinkel . . . . .	20
5.2. Kalman - Filter . . . . .	21
5.3. Linearer Kalman - Filter Algorithmus . . . . .	23

---

<b>6. Zusammenfassung und Ausblick</b>	<b>25</b>
<b>Literaturverzeichnis</b>	<b>26</b>
Appendix . . . . .	27
A.     Workstation . . . . .	27
B.     Interface Funktionen . . . . .	28
C.     Konfigurationsregister der Sensoren . . . . .	29
D.     Low-Level Funktion zum konfigurieren der Register . . . . .	31

# Abbildungsverzeichnis

2.1.	Schaltplan . . . . .	3
2.2.	Schema I2C-Bus . . . . .	6
2.3.	I2C-Register-Map . . . . .	6
3.1.	CubeMX . . . . .	8
3.2.	CubeIDE Ordner Struktur . . . . .	10
3.3.	Navigation zum Softwarepaket . . . . .	11
3.4.	Liste der verfügbaren Software-Pakete . . . . .	12
3.5.	Implementierung der Software-Pakete . . . . .	12
4.1.	Programm zur Aufnahme der physikalischen Sensorwerte pt.1 . . . . .	15
4.2.	Programm zur Aufnahme der physikalischen Sensorwerte pt.2 . . . . .	16
4.3.	Blockschaltbild: Aufnahme und Verarbeitung der Sensordaten . . . . .	18
5.1.	Schematische Darstellung der Sensorfusion . . . . .	19
5.2.	Schematische Darstellung des linearen Kalman - Algorithmus . . . . .	23
1.	Workstation . . . . .	27
2.	Register Mapping LIS2MDL . . . . .	29
3.	Register Mapping LIS2DW12 . . . . .	30

# Tabellenverzeichnis

3.1. I2C1_Schnittstelle . . . . .	9
3.2. USART3_Schnittstelle . . . . .	9
3.3. Navigationsleiste . . . . .	11
5.1. Liste der Symboldefinitionen . . . . .	24

# 1. Aufgabenstellung

Bewegung ist unverzichtbar für das Leben und Arbeiten auf der Erde. Noch unerlässlicher ist die Information über das „Wo und Wie?“. Sowohl der Mensch, als auch Maschinen, Roboter und viele andere technische Anwendungen sind daran interessiert zu wissen, wo sie sich befinden und in welche Richtung sie sich bewegen. In Analogie dazu steht die Überwachung und Analyse physikalischer Parameter durch hochmoderne Technologie im Mittelpunkt des hier vorgestellten Projekts. Ziel des Projektes ist es, die Basis für ein System zur relativen Positionsbestimmung zu schaffen, das auf der Integration von MEMS-Sensoren basiert. Diese hoch integrierten Bauelemente kombinieren mechanische und elektronische Komponenten auf Mikroebene, um physikalische Größen wie Beschleunigung, Druck und Temperatur mit herausragender Präzision zu messen. Zur Erfassung von relativen Positionen werden zum einen Beschleunigungswerte aufgenommen und zu anderen das Magnetfeld der Erde genutzt, um die Himmelsausrichtung zu messen. Die Implementierung umfasst die Verdrahtung der MEMS-Sensoren, die Erstellung eines STM32Cube-Projekts unter Einsatz des MEMS-Softwarepaketes, die Erfassung der physikalischen Sensorwerte über eine I2C-Schnittstelle und die Implementierung eines Algorithmus für die Positionsbestimmung, der auf der magnetischen Ausrichtung und der räumlichen Bewegung basiert. Im Laufe des Projekts sind eine Reihe von Herausforderungen aufgetreten, die nicht nur meine technischen Fähigkeiten, sondern auch die Fähigkeit, kritisch zu denken und innovative Lösungen zu finden, auf die Probe gestellt hat. Zwei dieser Herausforderungen wurden besonders hervorgehoben: einen hartnäckigen Softwarefehler und einen tückischen Hardwarefehler, die in Kapitel 3 behandelt wird. Ob das formulierte Ziel umsetzbar ist und zu welchen Ergebnissen die Bearbeitung des Projekts ergeben hat, wird im folgenden beschrieben.

# 2. Grundlagen

## 2.1. Komponenten

- STM32 Nucleo 144
- LIS2MDL (Magnetsensor)
- LIS2DW12 (Beschleunigungssensor)
- Pull-Up Widerstände (4.7k)
- Steckbrett + Jumperkabel

Abbildung 2.1 zeigt schematisch die Projektstruktur. Das in diesem Projekt verwendete Entwicklungsboard STM32 Nucleo 144 ist in der Abbildung in verkleinerter Form dargestellt. Die Dokumentation konzentriert sich auf die Channel, die in diesem Projekt verwendet wurden. Die einzelnen Pins eines Mikrocontrollers können verschiedene Funktionen ausführen, wie aus der Legende links neben den Schaltplänen hervorgeht. Welche Signalquelle einem Pin zugeordnet ist, hängt von der Softwarekonfiguration ab. Intern werden Multiplexer verwendet, um die verschiedenen Signalquellen zu schalten. Im Kapitel 3 wird auf die Funktionalität und die genaue Auswahl der Pins mit Hilfe des Datenblattes eingegangen.

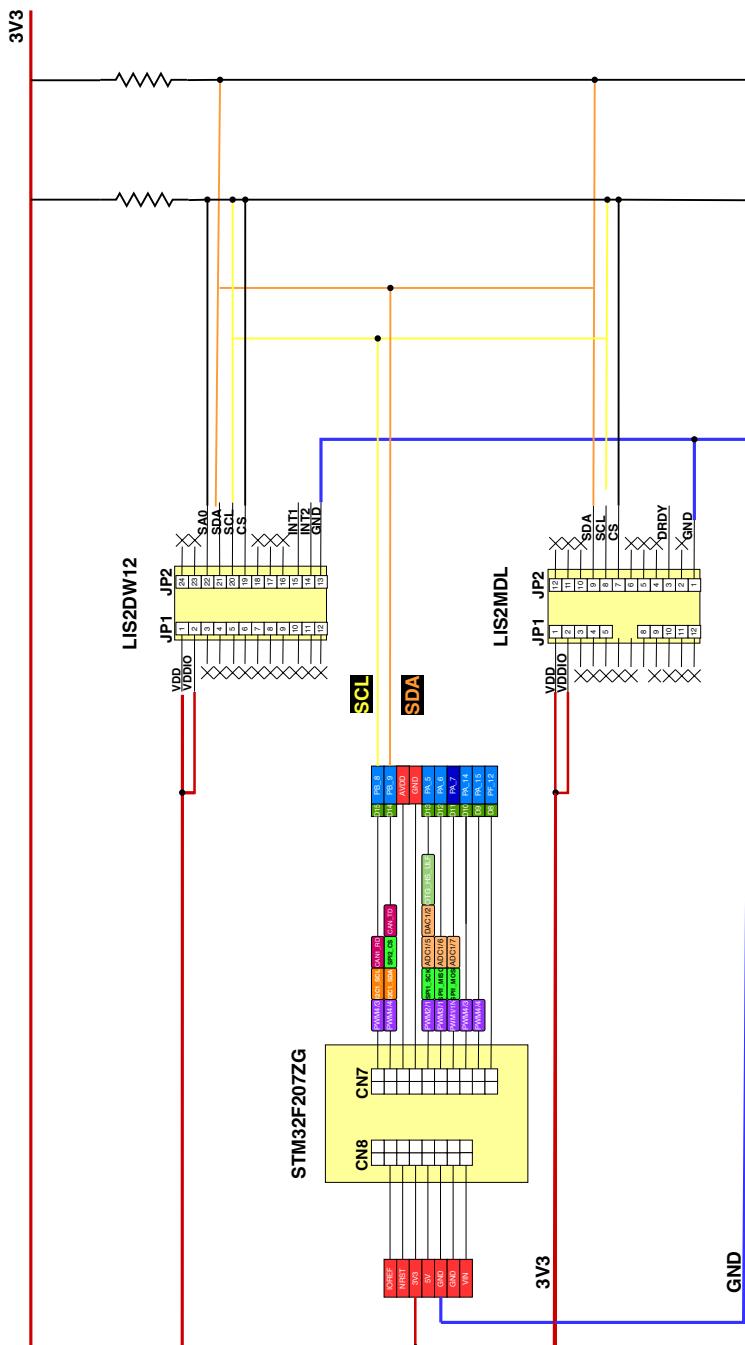


Abbildung 2.1.: Schaltplan

---

### **2.1.1. Entwicklungsboard STM32 Nucleo-144**

Neben Arduino und Raspberry Pi sind STM32-Boards eine hervorragende Wahl für Echtzeitanwendungen, insbesondere für das Auslesen und Verarbeiten von Sensordaten. Das Nucleo-Entwicklungsboard ist mit einem ARM Cortex Mikroprozessor, mehreren Komponenten (z.B. LEDs) und einer umfangreichen Peripherie ausgestattet. Dies eröffnet einen großen Spielraum für das Design von Prototypen, was für dieses Projekt essentiell ist. Die Vorgehensweise wird sein, jeweils einzelne Sensoren anzusteuern und am Ende beide Projekte zusammenzuführen. Die STM-eigenen Programme CubeIDE und CubeMX bieten eine benutzerfreundliche Entwicklungsumgebung, die es Entwickler:innen ermöglicht, schnell und einfach mit der Programmierung zu beginnen. Durch die einfache Integration von Sensoren über CubeMX können Entwickler:innen Zeit sparen und sich auf die eigentliche Entwicklung der Anwendung konzentrieren. CubeMX stellt Bibliotheken zur Verfügung, die eine einfache Integration von Sensoren ermöglichen. Diese Bibliotheken umfassen sowohl OOP- als auch Low-Level-Ansätze, um den Anforderungen unterschiedlicher Entwicklungsansätze gerecht zu werden. Mehr zum Umgang mit der Software finden Sie im Kapitel Kapitel 3. Für mich persönlich hat die Verwendung des STM den Hintergrund, dass ich in der Vergangenheit bereits Erfahrungen mit Eclipse-Software sammeln konnte. Die auf C basierende Syntax erleichtert mir die Entwicklung. Insgesamt bietet das STM32 Nucleo 144 eine optimale Kombination aus leistungsfähiger Hardware, benutzerfreundlicher Entwicklungsumgebung und umfangreicher Peripherieunterstützung, was es zu einer hervorragenden Wahl für die Echtzeiterfassung und -verarbeitung von Sensordaten macht.

### **2.1.2. Magnetometer LIS2MDL**

Der LIS2MDL ist ein hochpräziser digitaler 3-Achsen-Magnetometer. Der Sensor basiert auf dem Hall-Effekt, einem physikalischen Phänomen, bei dem sich der elektrische Widerstand eines Materials ändert, wenn es einem Magnetfeld ausgesetzt wird. Die Funktionsweise des Sensors beruht auf seiner Fähigkeit, Magnetfelder in drei orthogonalen Achsen zu erfassen. Durch Platzieren eines magnetoresistiven Elements in einem empfindlichen Bereich des Chips kann der Sensor Änderungen des Magnetfelds entlang dieser Achsen erkennen. Die STM-Softwarepaket enthält Bibliotheken, die speziell für die Integration verschiedener STM-Sensoren, einschließlich des LIS2MDL, entwickelt wurden. Diese Pakete erleichtern die Implementierung und Nutzung des Sensors, indem sie eine klar strukturierte API und Beispielcodes für eine Vielzahl von Anwendungen zur Verfügung stellen. LIS2MDL unterstützt die

---

Kommunikation über das I2C-Protokoll, das weit verbreitet und einfach zu implementieren ist. Dadurch kann der Sensor leicht mit verschiedenen Mikrocontrollern und anderen Geräten verbunden werden, was Flexibilität und Kompatibilität in meinem Projekt gewährleistet.

### **2.1.3. Beschleunigungssensor LIS2DW12**

Der LIS2DW12 ist ein hochpräziser digitaler 3-Achsen-Beschleunigungssensor, der für die Erfassung von Bewegungen und Neigungen in Echtzeit entwickelt wurde. Auf dem Chip des LIS2DW12 befinden sich winzige mikromechanische Strukturen, die als Massenträger dienen. Diese Strukturen sind in der Regel zwischen zwei elektrischen Leitern angeordnet und bilden so eine kondensatorähnliche Struktur. Wird der Sensor einer Beschleunigung oder Schwerkraft ausgesetzt, verschieben sich die Massen in den MEMS-Strukturen entsprechend. Diese Bewegung führt zu einer Änderung des Abstands zwischen den elektrischen Leitern, wodurch sich die Kapazität des Kondensators ändert. Die Kapazitätsänderung wird als Änderung des elektrischen Signals gemessen und in digitale Werte umgewandelt, die die gemessene Beschleunigung repräsentieren. Analog zum Magnetometer (LIS2MDL) werden auch beim LIS2DW12 sehr gute Dokumentationen und Beispielprojekte von STM zur Verfügung gestellt. Wie der Sensor die Daten aufnimmt und weiterverarbeitet, wird in Kapitel 4 näher beschrieben.

## 2.2. I2C Kommunikation

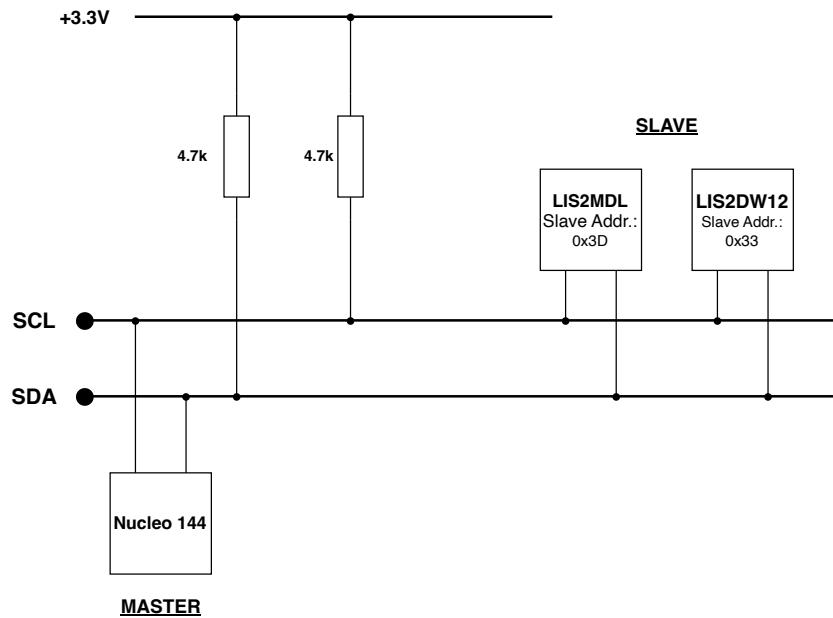


Abbildung 2.2.: I2C-Bus

Die einfache Implementierung ergibt sich aus der Tatsache, dass nur zwei Drähte für die Kommunikation zwischen den Geräten erforderlich sind. Jedes Gerät hat eine eindeutige Geräteadresse, so dass der Master auswählen kann, mit welchem Gerät er kommunizieren möchte. Die eindeutigen Adressen für die beiden Sensoren können dem Datenblatt entnommen werden. Die I2C-Adresse des LIS2MDL ist 0x3d und die I2C-Adresse des LIS2DW12 ist 0x33. Die beiden Leitungen werden als Serial Clock (SCL) und Serial Data (SDA) bezeichnet. Die SCL-Leitung ist das Taktsignal, das die Datenübertragung zwischen den Geräten synchronisiert. Die andere Leitung ist die SDA-Leitung, über die die Daten übertragen werden. Beide Leitungen sind „open-drain“, was bedeutet, dass Pull-up Widerstände angeschlossen werden müssen. Im I2C-Protokoll ist die Datenleitung (SDA) aktiv, wenn sie auf einem niedrigen Spannungspegel liegt.

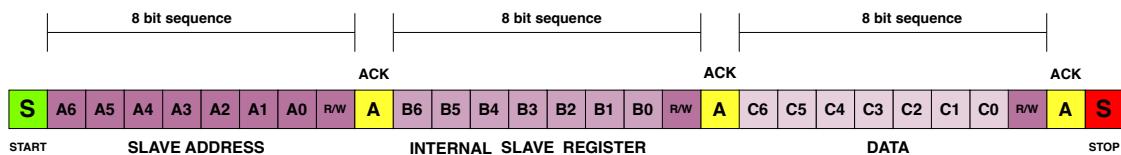


Abbildung 2.3.: I2C-Register-Map

Die Abbildung 2.3 zeigt das I2C-Protokoll anhand der übertragenen Bits. Auf der seriellen Datenleitung werden immer Pakete in Form von 8-Bit-Sequenzen gesen-

---

det. Nach jeder 8-Bit-Sequenz wird vom Slave ein ACK erwartet. Die Übertragung beginnt mit einem bestimmten Startsignal. Danach wird die Slaveadresse des anzu sprechenden Slaves übertragen. Bei einem positiven ACK-Signal wird die Adresse des zu schreibenden/lesenden Registers mit der Information, ob das Register gelesen oder geschrieben wird, gesendet. Anschließend werden die Daten in das Register des ausgewählten Slaves geschrieben. Die Kommunikation wird durch ein Stop-Signal beendet.

### 3. Planung der Umsetzung

### 3.1. Konfiguration des ARM® Cortex®-M3 Prozessor

Neben einer breiten Auswahl an Microcontrollern, Entwicklungsboards und Sensoren bietet STMicroelectronics Softwarelösungen an, um den Workflow von Entwickler:innen zu verbessern. Die beiden Hauptprogramme die STMicroelectronics zur Verfügung stellt und für dieses Projekt genutzt wurden, findet man unter der Bezeichnung CubeMX und CubeIDE. CubeMX ermöglicht die grafische Konfiguration von STM32 Microcontrollern und -prozessoren. Beim Starten des Programms öffnet sich der Reiter „Board-Selector“. Wie in Abbildung 3.1 zu sehen, habe ich mein Board ausgewählt, die Bezeichnung des Chips auf dem Entwicklungsboard ist STM32F207ZGTx.

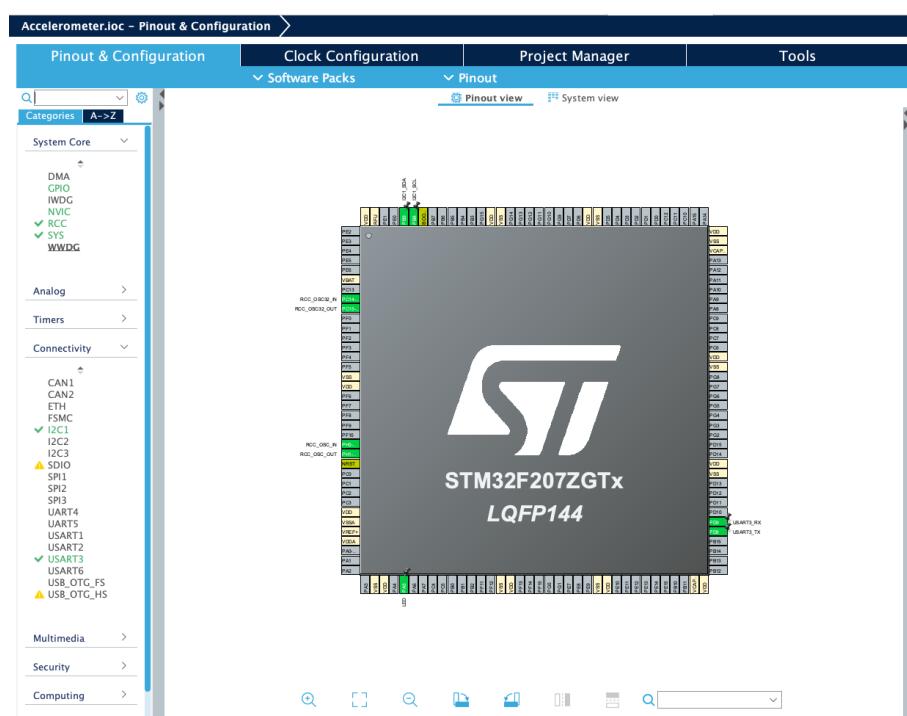


Abbildung 3.1.: CubeMX

---

Im Anschluss wird die Peripherie nach den eigenen Anforderungen konfiguriert. Über die Schaltflächen, an den Seiten des Chips, können den Pins manuell Funktionen zugeordnet werden. Für die Kommunikation über den I2C-Bus muss die I2C-Schnittstelle initialisiert werden. Zusätzlich wurde eine USART-Schnittstelle konfiguriert, um die Ergebnisse des Programms im Terminal ausgeben zu lassen. Die Clock des Chips ist auf HSE (high-speed external) eingestellt, was für dieses Projekt keine weitere Rolle spielt. Auf der linken Seite des Programms befindet sich eine Kategorisierung aller Funktionen des Chips. Im Abschnitt *Connectivity* muss der I2C-Bus aktiviert werden. Es ist darauf zu achten, dass die STM-Chips mehrere Hardware-I2C-Schnittstellen besitzen und diese den entsprechenden Pins zugeordnet sind. Ich verwende das I2C1 Interface und laut Datenblatt sind die beiden benötigten Pins PB8 und PB9.

Pin	Pin name	Signal name	STM32 pin	Function
9	A4	ADC	PB9	I2C1_SDA
11	A5	ADC	PB8	I2C1_SCL

Tabelle 3.1.: I2C1\_Schnittstelle

Dieses Vorgehen muss analog auf die USART-Schnittstelle angewandt werden. Laut Datenblatt wird das USART3-Interface auf den Pins PD8 und PD9 zur Verfügung gestellt.

Pin name	Function	Virtual COM port	ST morpho connection
PD8	USART3 TX	SB5 ON and SB7 OFF	SB5 OFF and SB7 ON
PD9	USART3 RX	SB6 ON and SB4 OFF	SB6 OFF and SB4 ON

Tabelle 3.2.: USART3\_Schnittstelle

CubeMX gibt den Entwickler:innen eine grafische Rückmeldung, ob die Konfiguration erfolgreich war (Schaltfläche wird grün) oder fehlerhaft (Schaltfläche rot) bzw. ob die Konfigurationen nicht vollständig sind (Schaltfläche orange). Ist die Konfiguration des Chips in CubeMX abgeschlossen, generiert die Software einen C-Code, der alle vorgenommenen Konfigurationen berücksichtigt und somit die Grundlage für das Programm bildet. Wenn im *Project Manager* keine spezielle Entwicklungsumgebung hinterlegt ist, öffnet sich beim Generieren der Konfigurationsdatei die STM32 CubeIDE. In dieser Umgebung kann der Code generiert und kompiliert werden, außerdem bietet die IDE einen sehr hilfreichen Debug-Modus. Die IDE basiert auf einem Eclipse Framework, so dass der Benutzer jedes erdenklichen Eclipse Plugin integrieren kann. Auf der linken Seite der Benutzeroberfläche wird die Ordnerstruktur des ausgewählten Workspace angezeigt. Hier befinden sich die Projektordner mit dem Programmcode. Abbildung 3.2 zeigt beispielhaft den Aufbau des Projektordners.

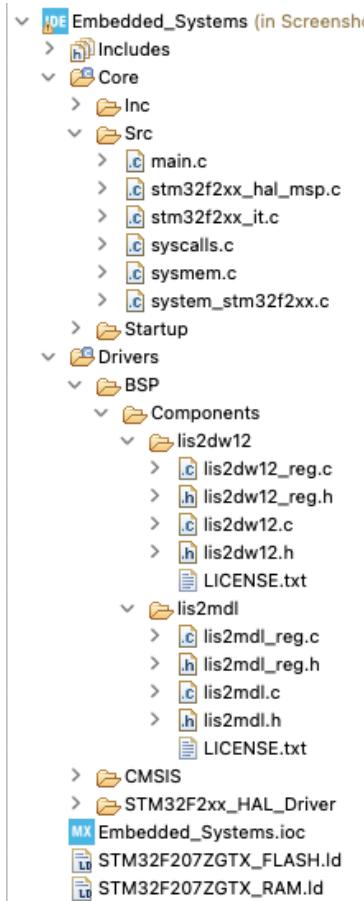


Abbildung 3.2.: CubeIDE Ordner Struktur

Der Inc-Ordner enthält Header-Dateien mit Funktionen und Makros, die im gesamten Projekt verwendet werden. Der Ordner Core enthält typischerweise den Kern des Anwendercodes sowie wichtige Konfigurationsdateien. Von besonderem Interesse ist der Ordner Drivers, in dem sich Treiber und Low-Level-Code befinden, die spezifisch für die Hardwarekomponenten des verwendeten Mikrocontrollers sind. Wie diese Treiber installiert und eingebunden werden, wird im folgenden Abschnitt gezeigt. Die genauere Beschreibung und Verwendung der wichtigsten Bibliotheken findet sich in Abschnitt 3.3 .

## 3.2. Installation und Implementation der MEM - Pakete

Zusätzlich zur Konfiguration des Chips bietet STMicroelectronics bzw. CubeMX Software Packs für spezifische Hardware zum Download an. Die Software Packs enthalten Bibliotheken für beliebige Hardwareanwendungen. Im oberen Bereich der Programmoberfläche befindet sich die Navigationsleiste, siehe Tabelle 3.1. Wählen „Software Packs“ → „Komponenten auswählen“. Anschließend muss das Paket STMicro-electronics.X-CUBE-MEMS1 ausgewählt und installiert werden. Das Paket enthält sowohl Bibliotheken für LIS2MDL als auch für LIS2DW12, die für die Entwicklung des Projekts sehr hilfreich sind. Die Vorgehensweise ist in den folgenden Abbildungen visualisiert; Abbildung 3.3, Abbildung 3.4. Der letzte Schritt, bevor der C-Code von CubeMX generiert werden kann, ist das Einbinden der Software Pakete in das Projekt. Dazu wählt man im linken Reiter den Punkt *Middleware und Software Packs* aus, navigiert zum Software Pack STMicroelectronics.X-CUBE-MEMS1 und aktiviert die beiden Sensoren wie in Abbildung 3.5 beschrieben.

Pinout & Configuration	Clock Configuration	Project Manager	Tools
	↓ Software Packs	↓ Pinout	

Tabelle 3.3.: Navigationsleiste

Packs	Pack / Bundle / Component	Status	Version	Selection
> ITTIA_DB-I-CUBE-ITTIADB		8.8.0	⊕	▲ Install
> Infineon.AIROC-Wi-Fi-Bluetooth-STM32		1.5.1	⊕	▲ Install
> SEGGER.I-CUBE-embOS		1.3.1	⊕	▲ Install
> STMicroelectronics.FP-ATR-ASTRA1	⊕	2.0.0	⊕	▲ Install
> STMicroelectronics.FP-ATR-SIGFOX1	⊕	3.2.0	⊕	▲ Install
> STMicroelectronics.FP-SNS-FLIGHT1	⊕	5.0.2	⊕	▲ Install
> STMicroelectronics.FP-SNS-MOTENV1		4.3.2	⊕	▲ Install
> STMicroelectronics.FP-SNS-MOTENVB1		1.3.0	⊕	▲ Install
> STMicroelectronics.FP-SNS-MOTENVB2		1.2.0	⊕	▲ Install
> STMicroelectronics.FP-SNS-SMARTAG2		8.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AI	⊕	1.3.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-ALCOBUILD		1.0.1	⊕	▲ Install
> STMicroelectronics.X-CUBE-ALS	⊕	1.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-F4	⊕	1.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-F7	⊕	1.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-G0	⊕	1.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-G4	⊕	2.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-H7	⊕	3.2.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-L4	⊕	2.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-L5	⊕	2.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-WB	⊕	2.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-AZRTOS-WL	⊕	2.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-BLE1		7.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-BLE2		3.3.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-BLEMR		3.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-DISPLAY	⊕	3.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-EPRMA1		4.2.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-FREERTOS	⊕	1.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-GNSS1		6.0.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-ISPU		1.1.0	⊕	▲ Install
> STMicroelectronics.X-CUBE-MEMS1	⊕	10.0.0	⊕	▲ Install

Abbildung 3.3.: Navigation zum Software Paket STMicroelectronics.X-CUBE-MEMS1

Pack / Bundle / Component	Status	Version	Selection
STMicroelectronics.X-CUBE-MEMS1	Green	10.0.0	-
Exposed APIs	Green		
Device MEMS1_Applications	Green	10.0.0	
Board Part AccGyr	Green	5.5.0	
Board Part AccMag	Green	5.6.0	
Board Part Acc	Green	1.4.0	
LIS2DW12	Green	1.4.0	I2C
LIS2DH12	Green		Not select...
IIS2LPC	Green		Not select...
AIS2DW12	Green		Not select...
AIS32RDQ	Green		Not select...
AIS3624DQ	Green		Not select...
H3LS331DL	Green		Not select...
IIS2CLX	Green		Not select...
AIS2IH	Green		Not select...
LIS2DU12	Green		Not select...
IIS2WB	Green		Not select...
LIS2DU12	Green		Not select...
Board Part AccTemp / LIS2DTW12	Green		Not select...
Board Part Mag	Green	5.4.0	
LIS3MDL	Green	1.6.0	Not select...
LIS2MDL	Green		I2C
IIS2MDC	Green		Not select...
Board Part HumTemp	Green	5.6.0	
Board Part PressTemp	Green	5.6.0	
Board Part Temp	Green	1.4.0	
Board Part Gyr / A3G4250D	Green		Not select...
Board Part PressTempQvar	Green	1.2.0	
Board Part AccGyrQvar	Green	1.4.0	
Board Part AccQvar / LIS2DUNS12	Green		Not select...

Abbildung 3.4.: Liste der verfügbaren Software-Pakete

The screenshot shows the STM32CubeMX software interface. The left sidebar lists various software packages under 'Categories' (e.g., System Core, Analog, Timers, Connectivity, Multimedia, Security, Computing, Middleware and So...). The main area has two tabs: 'Pinout & Configuration' (selected) and 'Clock Configuration'. In the 'Pinout & Configuration' tab, a search bar and a 'Categories' dropdown are at the top. Below is a tree view of selected packages: 'Board Part Core' (checked), 'Board Part Acc' (checked), and 'Board Part Mag' (checked). The 'Clock Configuration' tab shows the 'STM32CubeMX.X-CUBE-MEMS1.10.0.0 Mode and Configuration' mode. The 'Mode' section lists checked options for 'Board Part Acc' and 'Board Part Mag'. The 'Configuration' section includes a 'Reset Configuration' button and tabs for 'Parameter Settings' and 'Platform Settings'. Under 'BSP', there is a table for 'Platform proposal' with two rows:

Name	IPs or Components	Found Solutions	BSP API
LIS2MDL BUS IO driver	I2C:I2C	Undefined	BSP_BUS_DRIVI
LIS2DW12 BUS IO driver	I2C:I2C	Undefined	BSP_BUS_DRIVI

Abbildung 3.5.: Implementierung der Software-Pakete

---

### **3.3. Integration**

In der aktuellen Projektphase wurden die Schlüsselkomponenten vorgestellt, darunter das STM-Entwicklungsboard und die beiden Sensoren. Außerdem wurde die Wahl des Kommunikationsprotokolls begründet und die Funktionsweise von I2C am aktuellen Beispiel kurz erläutert. Die Bedeutung von STM32 CubeIDE und CubeMX als optimale Entwicklungswerkzeuge für die Arbeit mit STM32 Mikrocontrollern wurde hervorgehoben. Ein detailliertes "Kochrezept" für die Installation und Integration von Softwarepaketen wurde erstellt. Bevor mit der Programmierung des Mikrocontrollers begonnen werden kann, müssen zwei wesentliche Themen behandelt werden: der physikalische Aufbau der Schaltung und die Verwendung der installierten Softwarepakete und des Hardware Abstraction Layer (HAL). Diese wesentlichen Punkte werden in diesem Kapitel ausführlich behandelt.

#### **3.3.1. Hardware Abstraction Layer**

Die Hardware Abstraction Layer (HAL) ermöglicht es höheren Programmschichten, hardwarenahe Operationen auf einer abstrakten Ebene durchzuführen. Dadurch werden die Operationen unabhängig von den spezifischen Details der Hardware, was die Programmierung vereinfacht. In diesem Projekt wurde die Hardware Abstraction Layer mit dem STM eigenen Programm CubeMX erstellt. Das Projekt verwendet die Hardware Abstraction Layer (HAL) Bibliothek, speziell für die STM32F2 Serie, zur Initialisierung und Kommunikation mit den beiden Sensoren über I2C. Die HAL wird verwendet, um grundlegende MCU-Konfigurationen wie Systemtakt und Peripherie (I2C, USART) zu initialisieren, Fehler zu behandeln und eine Verzögerungsfunktion zu implementieren. Speziell für die Sensoren werden die HAL-Funktionen HAL\_I2C\_Mem\_Write() und HAL\_I2C\_Mem\_Read() verwendet, um Konfigurationsregister der Sensoren zu schreiben und zu lesen. Die Implementierung der HAL und die Nutzung der beiden Funktionen ist notwendig für die Initialisierung der Sensoren, das Auslesen der Sensordaten und die Konfiguration der Sensoren für kontinuierliche Messungen. Außerdem wird HAL\_USART\_Transmit verwendet, um Sensorwerte über USART zu übertragen, was zeigt, dass die HAL für die Kommunikation zwischen dem Mikrocontroller und dem Sensor sowie für die Ausgabe von Sensorwerten an einem Computer verwendet wird.

---

### 3.3.2. MEM-Paket

Das Softwarepaket X-CUBE-MEMS1 von STMicroelectronics ist eine Erweiterung für STM32CubeIDE, die speziell entwickelt wurde, um die Entwicklung und Implementierung von Anwendungen mit MEMS-Sensoren zu vereinfachen. Das Paket bietet umfassende Unterstützung für die Integration und Verwendung verschiedener Sensortypen. Um das X-CUBE-MEMS1 Softwarepaket nutzen zu können, muss für jeden Sensor eine genau definierte Schnittstelle initialisiert werden. Der Treiber/Interface, der im C-Programm als *struct* formuliert ist, hat insgesamt vier Member und spielt eine zentrale Rolle. Die Member regeln die Steuerung und den fehlerfreien Datenaustausch zwischen MCU und Sensor. Die ersten beiden Member, `read_reg` und `write_reg`, sind besonders kritisch, da ihnen Funktionen zugewiesen werden. Bei der Zuweisung der Funktionen an die Member wurde auf die Hardware Abstraction Layer zurückgegriffen. Die verwendeten Funktionen sind im vorherigen Abschnitt aufgelistet. Die Member ermöglichen es dem Treiber, Daten vom Sensor zu lesen und Einstellungen oder Kommandos an den Sensor zu schreiben. Da die Kommunikation über das I2C-Protokoll erfolgt, muss vor dem Aufruf des Treibers die 7-Bit Geräteadresse im Datenblatt nach links verschoben werden. Diese Bitmanipulation stellte für mich eine Herausforderung dar. Denn eine falsche Manipulation führt schnell zu einer Fehlerkette, die schwer zu überblicken war. Statt das letzte Bit zu verschieben, habe ich zu Beginn des Projektes immer das erste Bit verschoben. Dies hatte zur Folge, dass einerseits nur ungerade Registeradressen ausgelesen werden konnten und andererseits nie die richtige Adresse angesprochen wurde, was durch mehrmalige I2C-Flankenanalyse validiert werden konnte. Zusammenfassend kann gesagt werden, dass die ersten beiden Member des Treibers eine Kombination aus HAL und Bitmanipulation sind. Das dritte Member, `*handle`, ist ein Pointer, dem das I2C-Handle zugewiesen wird, um die Kommunikationsschnittstelle festzuhalten.

# 4. Aufnahme\_physikalischer

## 4.1. Programmablauf

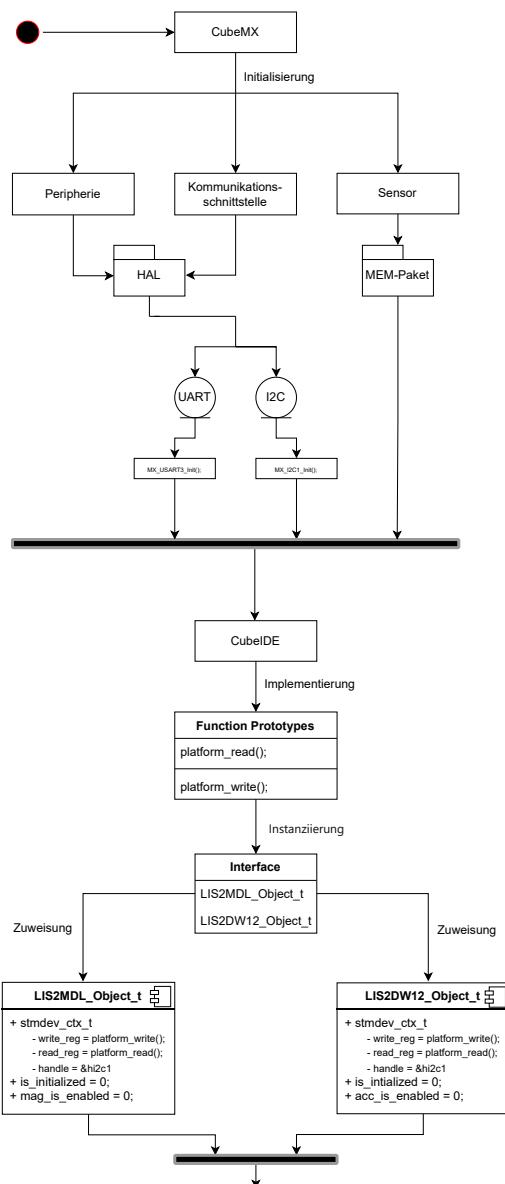


Abbildung 4.1.: Programm zur Aufnahme der physikalischen Sensorwerte

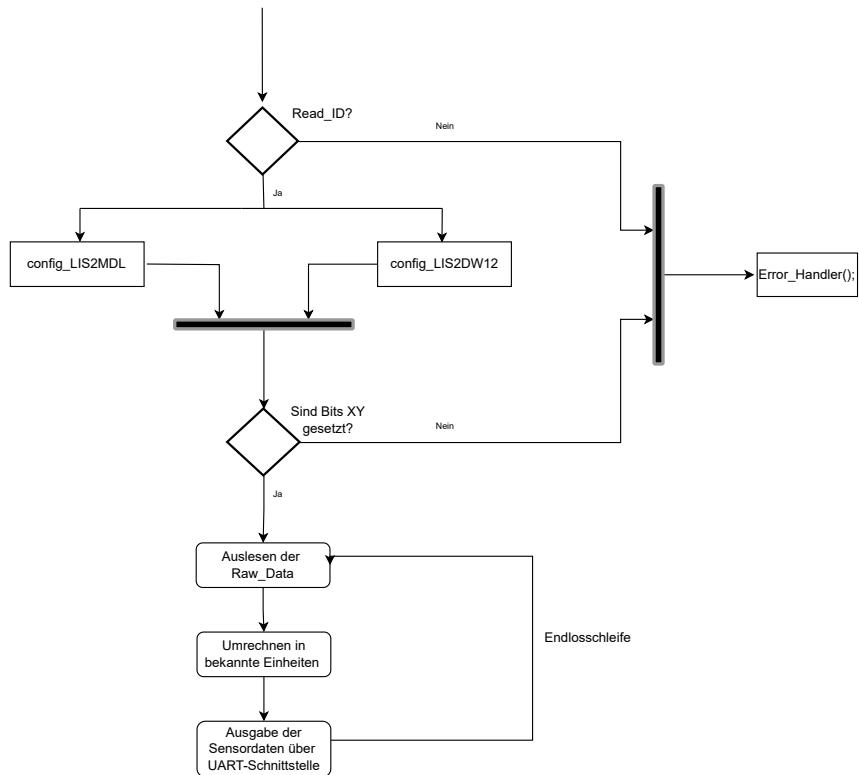


Abbildung 4.2.: Programm zur Aufnahme der physikalischen Sensorwerte

## 4.2. Initialisierung, Implementierung, Instanziierung

Durch Initialisierung, Implementierung und Instanziierung werden die notwendigen Schritte unternommen, um die Hardware zu konfigurieren, Schnittstellenfunktionen bereitzustellen und schließlich ein Objekt zu erzeugen, das die Interaktion mit dem Sensor ermöglicht.

Die Hardware- und Softwarekomponenten werden durch CubeMX initialisiert. Zu den Aufgaben gehört die Initialisierung der I2C-Schnittstelle, die Initialisierung der UART-Schnittstelle für die serielle Kommunikation, sowie die Initialisierung der GPIO-Pins. Diese Schritte sind notwendig, um die Hardware-Peripheriegeräte für die Interaktion mit dem MEMS-Sensor zu konfigurieren und bereitzustellen. Des Weiteren sind eine Reihe von Variablen initialisiert wurden, um Ergebnisse zwischenspeichern, Ergebnisse zu vergleichen oder um Bedingungen in Verzweigungen zu setzen.

Die Implementierung umfasst die Definition von Funktionen, die spezifizieren, wie bestimmte Aufgaben ausgeführt werden sollen. Beispiele hierfür sind die Funktionen `platform_write()` und `platform_read()`. Diese Funktionen abstrahieren die low-level

---

Kommunikation mit dem Sensor über I2C und sind entscheidend für das Schreiben und Lesen von Registern des Sensors. Weiterhin wurden Funktionen implementiert, die spezifische Konfigurationen der Sensor Register vornehmen.

Die Instanziierung tritt auf, wenn ein Objekt aus einer Struktur oder Klasse erzeugt wird. In diesem Projekt wird dies durch die Erstellung der Sensor Objekte vom Typ LIS2MDL\_Object\_t und LIS2DW12\_Object\_t demonstriert. Anschließend wird dieses Objekt mit der notwendigen Kontextinformation (dev\_ctx) initialisiert. Diese enthält die Schnittstellenfunktionen für Schreib- und Leseoperationen sowie den Hardwarekontext. Durch die Instanziierung der Sensor-Objekte und die Zuweisung der initialisierten Strukturen dev\_ctx kann nun über die definierten Schnittstellenfunktionen interagiert werden.

### 4.3. Setup / Config

Nach Bewertung der verfügbaren Optionen und Einschätzung der eigenen Programmierfähigkeiten wurde für dieses Projekt entschieden, die Sensoren im Betriebsmodus Polling zu betreiben. Die Entscheidung basiert auf der Einfachheit und Vorhersehbarkeit, die Polling-Algorithmen bieten, sowie der vollständigen Kontrolle, die sie uns über das Timing der Datenabfrage geben. Priorität hat die Entwicklung einer stabilen und zuverlässigen Basis für die Datenerfassung. Die Implementierung von Polling ist einfach und erleichtert das Testing. Gleichzeitig ermöglicht die klare Kontrolle über das Abfrageintervall eine präzise Optimierung der Systemleistung. Im Vergleich zu interruptbasierten Algorithmen ist zu berücksichtigen, dass der Betriebsmodus Polling potenzielle Nachteile aufweist. Ein interruptbasierter Algorithmus kann in späteren Entwicklungsphasen erhebliche Vorteile bieten, insbesondere hinsichtlich der Effizienz des Energieverbrauchs und der Fähigkeit, Ereignisse mit minimaler Verzögerung zu erfassen.

Im Anhang sind Abbildung zum Register Mapping der Sensoren aufgeführt. Die Register können kategorisch in drei Kategorien eingeteilt werden. Zu diesen zählen Konfigurations-, Output- und Statusregister. Statusregister sind von zentraler Bedeutung für die Überwachung des aktuellen Zustands des Sensors. Sie liefern wichtige Informationen über Fehlermeldungen, den Bereitschaftsstatus, Überlaufwarnungen und weitere Zustands- oder Fehlerindikatoren. Output-Register enthalten die eigentlichen Messergebnisse, wie beispielsweise Magnetfelder oder Beschleunigungswerte. Konfigurationsregister dienen dazu, die Betriebsmodi des Sensors einzustellen, wie beispielsweise Messrate und Auflösung. Es ist wichtig, dass die Beschreibung der Konfigurationsregister klar und präzise ist, um Missverständnisse zu vermeiden. In

Datenblättern wird oft beschrieben, wie Konfigurationsregister modifiziert werden müssen, um den Sensor für einen bestimmten Anwendungsfall vorzubereiten.

## 4.4. Loop

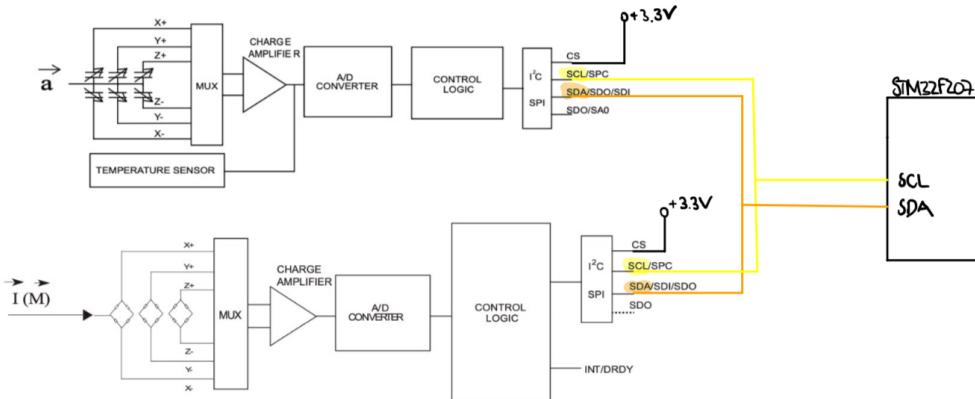


Abbildung 4.3.: Blockschaltbild: Aufnahme und Verarbeitung der Sensordaten

Dieses Blockschaltbild zeigt die Verarbeitungskette von der Erfassung der Sensordaten über die Verstärkung und Digitalisierung bis hin zur Kommunikation mit dem Mikrocontroller. Programmiertechnisch wurde eine Endlosschleife verwendet, um die Sensoren kontinuierlich nach neuen Werten abzufragen. Mit Hilfe des Softwarepaketes X-CUBE-MEMS1 konnten die aufgezeichneten Rohdaten direkt in lesbare physikalische Größen umgewandelt werden. Die Aufzeichnung der Magnetfeld- und Beschleunigungsdaten für alle drei Achsen ist das Ergebnis der Endlosschleife. Damit ist die Grundlage für die Positionsbestimmung eines Objektes mit MEM-Sensoren geschaffen. Die technische Realisierung des Ziels nach Aufnahme und Umrechnung der Sensordaten wird im folgenden Kapitel 5 diskutiert.

# 5. Positionsbestimmung mit MEM Sensoren

## 5.1. Grundlagen

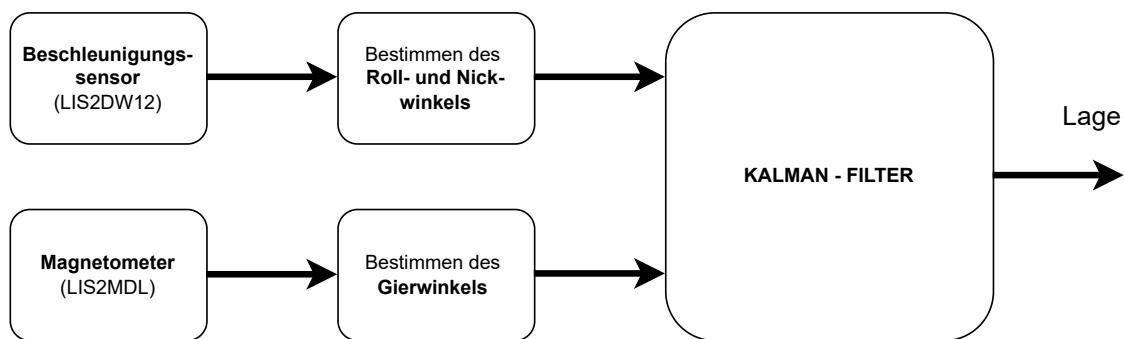


Abbildung 5.1.: Schematische Darstellung der Sensorfusion

Im Folgenden wird auf die Grundlagen der Positionsbestimmung durch Sensorfusion mit Hilfe von Kalman-Filtern eingegangen. Der Schwerpunkt liegt auf der inertialen Navigation, die auf der kontinuierlichen Bestimmung von Bewegungsrichtung, Geschwindigkeit und verstrichener Zeit zur Bestimmung der aktuellen Position basiert. Insbesondere werden die Inertialsensoreinheit, Koordinatensysteme und die Bestimmung von Nick-, Roll- und Gierwinkel mittels Eulerwinkel erläutert.

### 5.1.1. Navigation

Die Transformation von Koordinatensystemen, insbesondere die Transformation von Messdaten, die im körperfesten Koordinatensystem (b-frame) aufgenommen wurden, in das Navigationskoordinatensystem (n-frame), ist ein zentraler Schritt bei der inertialen Navigation und der Verarbeitung von Sensordaten zur Positionsbestimmung. Dieser Prozess ermöglicht es, die im inertialen System gemessenen Beschleunigungen und magnetischen Ausrichtungen in ein globales Bezugssystem zu transformieren,

---

das für Navigationszwecke nützlich ist. Die Achsen des körperfesten Koordinatensystems sind fest mit dem Objekt verbunden. Bewegungen und Orientierungen werden relativ zu dieser Konfiguration gemessen. Das Navigationskoordinatensystem ist ein globales Bezugssystem, das häufig mit der X-Achse nach Norden, der Y-Achse nach Osten und der Z-Achse senkrecht zum Boden (nach unten) ausgerichtet ist. Die Orientierung und Transformation zwischen dem b-Rahmen und dem n-Rahmen kann durch drei Eulerwinkel beschrieben werden: Nickwinkel (Pitch,  $\theta$ ), Rollwinkel (Roll,  $\varphi$ ) und Gierwinkel (Yaw,  $\psi$ ). Diese Winkel geben an, wie sich das Objekt um die jeweiligen Achsen dreht. Diese Winkel können direkt aus den Messdaten der Sensoren abgeleitet werden. Zur Transformation der Sensordaten vom b-Bild in das n-Bild wird die Direction Cosine Matrix (DCM) oder eine äquivalente Rotationsmatrix verwendet. Diese Matrix stellt die Rotation des Objekts im Raum dar und ist eine Funktion der Eulerwinkel. Die Matrix ermöglicht die Transformation der gemessenen Vektoren (z.B. Beschleunigung, magnetische Orientierung) in das globale Bezugssystem, indem die im b-Frame gemessenen Vektoren mit der DCM multipliziert werden. Dies überführt die lokalen Messwerte in das globale Koordinatensystem und ermöglicht die Bestimmung von Position, Geschwindigkeit und Orientierung des Objekts relativ zur Erde.

### 5.1.2. Nick-, Roll- und Gierwinkel

Der Gierwinkel ( $\psi$ ) beschreibt die Drehung um die vertikale Achse und gibt die Ausrichtung des Objekts relativ zur geographischen Nordrichtung an. Das Magnetometer misst die X-, Y- und Z-Komponente des Magnetfeldes im Koordinatensystem des Objektes. Um den Einfluss der Neigung (Roll- und Nickwinkel) auf die Magnetometermessungen zu kompensieren, können die gemessenen Magnetfeldkomponenten korrigiert werden. Die korrigierten Werte für das horizontale Magnetfeld  $X_h$  und  $Y_h$  werden wie folgt berechnet, wobei  $X_m$ ,  $Y_m$  und  $Z_m$  die gemessenen Magnetfeldkomponenten, der Rollwinkel und der Nickwinkel sind.

$$X_h = X_m \cdot \cos(\theta) + Z_m \cdot \sin(\theta) \quad (5.1)$$

$$Y_h = \frac{X_m \cdot \sin(\phi) \cdot \sin(\theta) + Y_m \cdot \cos(\phi) - Z_m \cdot \sin(\phi)}{\cos(\theta)} \quad (5.2)$$

Die Berechnung des Gierwinkelns kann aus den kompensierten horizontalen Magnetfeldkomponenten  $X_h$  und  $Y_h$  erfolgen.

---


$$\psi = \arctan \left( \frac{Y_h}{X_h} \right) \quad (5.3)$$

Der Rollwinkel (Roll,  $\varphi$ ) beschreibt die Drehung um die Längsachse des Objekts. Er kann direkt aus den Beschleunigungsmessungen ( $a_x$ ) und ( $a_z$ ) unter Verwendung der Querbeschleunigung ( $a_y$ ) und der Vertikalbeschleunigungskomponente berechnet werden.

$$\varphi = \arctan \left( \frac{a_y}{\sqrt{a_x^2 + a_z^2}} \right) \quad (5.4)$$

Der Nickwinkel (Pitch,  $\theta$ ) beschreibt die Neigung um die Querachse des Objekts. Er kann aus den Beschleunigungsmessungen wie folgt berechnet werden:

$$\theta = \arctan \left( \frac{-a_x}{\sqrt{a_y^2 + a_z^2}} \right) \quad (5.5)$$

Diese Berechnungen basieren auf der Annahme, dass die einzige auf das Objekt wirkende Beschleunigung die Gravitationsbeschleunigung ist und dass keine dynamischen Einflüsse (z.B. durch Bewegungen) die Messungen beeinflussen.

## 5.2. Kalman - Filter

Der Kalman-Filter ist ein leistungsfähiges mathematisches Verfahren, das in der Signalverarbeitung weit verbreitet ist, um den Zustand eines dynamischen Systems aus einer Reihe von unvollständigen und verrauschten Messwerten zu schätzen. Der Kalman-Filter schätzt die Zustände des Systems auf der Grundlage eines Systemmodells und der Sensordaten des vorangegangenen Zeitintervalls (a-proiri). Das Kalman-Filter erhält zusätzlich zur Schätzung eine Zustandsmessung als Eingabe und optimiert damit die Vorhersage (a-posterior). Die a-posteriori-Schätzung wird im nächsten Zeitintervall zur neuen a-priori-Schätzung. Der Kalman-Filter hat Ein- und Ausgänge. Die Eingänge sind verrauschte und eher ungenaue Messungen. Die Ausgänge sind weniger verrauscht und teilweise genauere Schätzungen. Die Herausforderung in diesem Projekt besteht darin, die Schwächen der Sensoren zu kompensieren. Das Grundprinzip aller Sensoren besteht darin, die Wirkung einer physikalischen Größe auf ein gedämpftes Feder-Masse-System zu messen. Die Verfahrensweise besteht darin, dass sich eine Masse, welche an das Gehäuse gekoppelt

---

ist, bei einer Beschleunigung des Gehäuses, dem Gehäuse gegenüber verschiebt. So- wohl Beschleunigungssensoren und Magnetometer weisen solch eine Struktur auf. Ihre Schwäche liegt in der Erfassung von Daten bei hohen Dynamiken. Die innere Struktur weist eine hohe Trägheit auf und ist daher anfällig auf äußere plötzlich eintretende Veränderungen. Deshalb werden oft Gyroscope ergänzt, die besser für die Erfassung von dynamischen Bewegungen geeignet sind, um ein umfassendes Bild der Bewegung und Orientierung zu bieten. Aufgrund unterschiedlicher Anforderungen gibt es verschiedene Varianten des Kalman-Filters. Für dieses Projekt soll der lineare Kalman-Filter vorgestellt werden.

### 5.3. Linearer Kalman - Filter Algorithmus

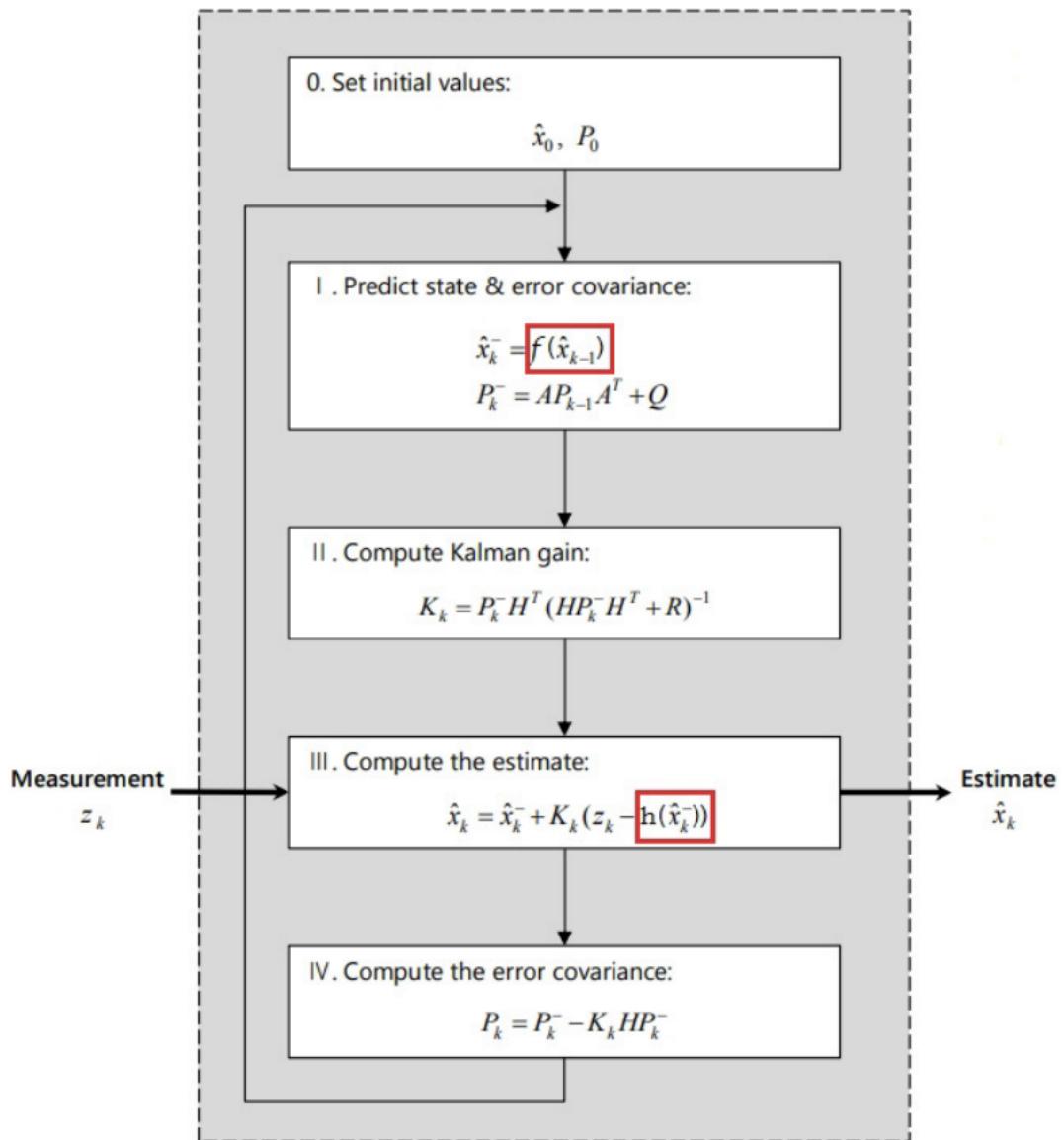


Abbildung 5.2.: Algorithmus des extended Kalman Filters (EKF) (vgl. Kim (2011), S. 146)

---

<b>x</b>	state variable	n x 1 column vector	output
<b>z</b>	measurement	m x 1 column vector	input
P	state covariance matrix	n x n matrix	output
A	state transition matrix	n x n matrix	System Model
H	state-to-measurement matrix	m x n matrix	System Model
R	measurement covariance matrix	m x m matrix	input
Q	process noise covariance matrix	n x n matrix	System Model
K	Kalman Gain	n x m	internal

Tabelle 5.1.: Liste der Symboldefinitionen

Laut [3] unterteilt man das Vorgehen des Filters in drei Prozesse auf: Im Initialisierungsschritt werden zu Beginn die Anfangswerte für  $\hat{x}_0$  und  $P_0$  festgelegt.  $\hat{x}_0$  wird als erste Schätzung des Kalman-Filters interpretiert. Diese Variable wird mit null oder einem schon bekannten Wert für die erste Schatzüng initialisiert. Die Fehlerkovarianz  $P_0$  berechnet die Differenz zwischen dem vom Kalman-Filter geschätzten Wert und dem wahren Wert, d.h. die Kovarianz gibt den Grad der Genauigkeit an. Im Vorhersageschritt wird sowohl die aktuelle Schätzung  $\dot{\hat{x}}_k$  als auch die aktuelle Fehlerkovarianz  $\dot{P}_k$  berechnet. Grundlage für die Berechnung ist zum einen die vorangegangene Zustandsverteilung  $N(\hat{x}_{k-1}, P_{k-1})$  und zum anderen die Systemmodelle A und Q. Der letzte Prozess des Algorithmus umfasst die Schritte II, III und IV aus der Abbildung 13. Schritt II beinhaltet die Bestimmung der Kalman-Verstärkung. Diese Verstärkung wichtet die Messwerte  $z_k$  gegenüber den geschätzten Werten des Kalman - Filters. In Schritt III wird der verbesserte Schätzwert  $\hat{x}_k$  berechnet. Dazu wird die Differenz aus den gemessenen Werten und den vorhergesagten Schätzwerten gebildet, das Ergebnis mit der Kalman-Verstärkung gewichtet und mit dem vorhergesagten Schätzwert addiert. Schritt IV bestimmt die neue Fehlerkovarianz  $P_k$  und bildet damit den Ausgangspunkt für den nächsten Durchlauf des Algorithmus.

# 6. Zusammenfassung und Ausblick

In dieser Projektarbeit wurde die Grundlage für die Positionsbestimmung mit MEM-Sensoren gelegt. Die zur Navigation benötigten Daten wurden mit Hilfe von einem Magnetometer und einem Beschleunigungssensor aufgenommen. Für die Verarbeitung und Kommunikation mit den Sensoren wurde ein STM32 Entwicklungsboard verwendet, welches über ein I2C Bus die Daten der Sensoren erhält. Den Aufbau und die Konfiguration der Schaltung erfolgte über die firmeninternen Softwareprogramme von STMicroelectronics. Zur Realisierung einer guten Positionsbestimmung werden die Daten miteinander fusioniert. Für die Fusionierung der Daten bietet sich der Kalman-Filter Algorithmus an. Die Definition, die Grundlagen und der Umgang mit dem Algorithmus wurden behandelt. Auf eine genaue mathematische Beschreibung der System Modelle A und H sowie auf die Kovarianz Q und R wurde verzichtet, ist aber für die Implementierung des Kalman-Filters von Vorteil.

In Zukunft soll das Projekt weiterentwickelt werden und praktische Anwendung finden. Ein Ansatz hierfür wäre die Fehlereigenschaften der Sensoren zu kompensieren, indem man sowohl von multiplen Beschleunigungssensoren als auch von mehreren Magnetfeldsensoren Daten aufnimmt und diese Daten anhand eines speziellen Messaufbaus kalibriert. Ein geeigneter Messaufbau wäre beispielsweise ein Drehteller. Im Anschluss könnte ein Hardware-Prototyp entwickelt werden, der für praktische Anwendungen einfach zu handhaben ist. An dieser Stelle muss die Theorie aus Kapitel 5 in die Praxis umgesetzt werden. Das bedeutet, dass Testreihen für die zu bestimmenden Winkel aufgenommen werden müssen und diese Ergebnisse dann dem Kalman-Algorithmus zugeführt werden.

# Literaturverzeichnis

- [1] L. M. Tränkler, H.-R.; Reindl. *Sensortechnik. Handbuch für Praxis und Wissenschaft.* 2. Auflage. Springer-Verlag, 2014.
- [2] J. Wendel. *Integrierte Navigationssysteme. Sensordatenfusion, GPS und Inertiale Navigation.* 2. Auflage. Wissenschaftsverlag GmbH, 2011.
- [3] P. Kim. *Kalman Filter for Beginners with Matlab Examples.* A-JIN Publishing company, 2011.

---

# Appendix

## A. Workstation

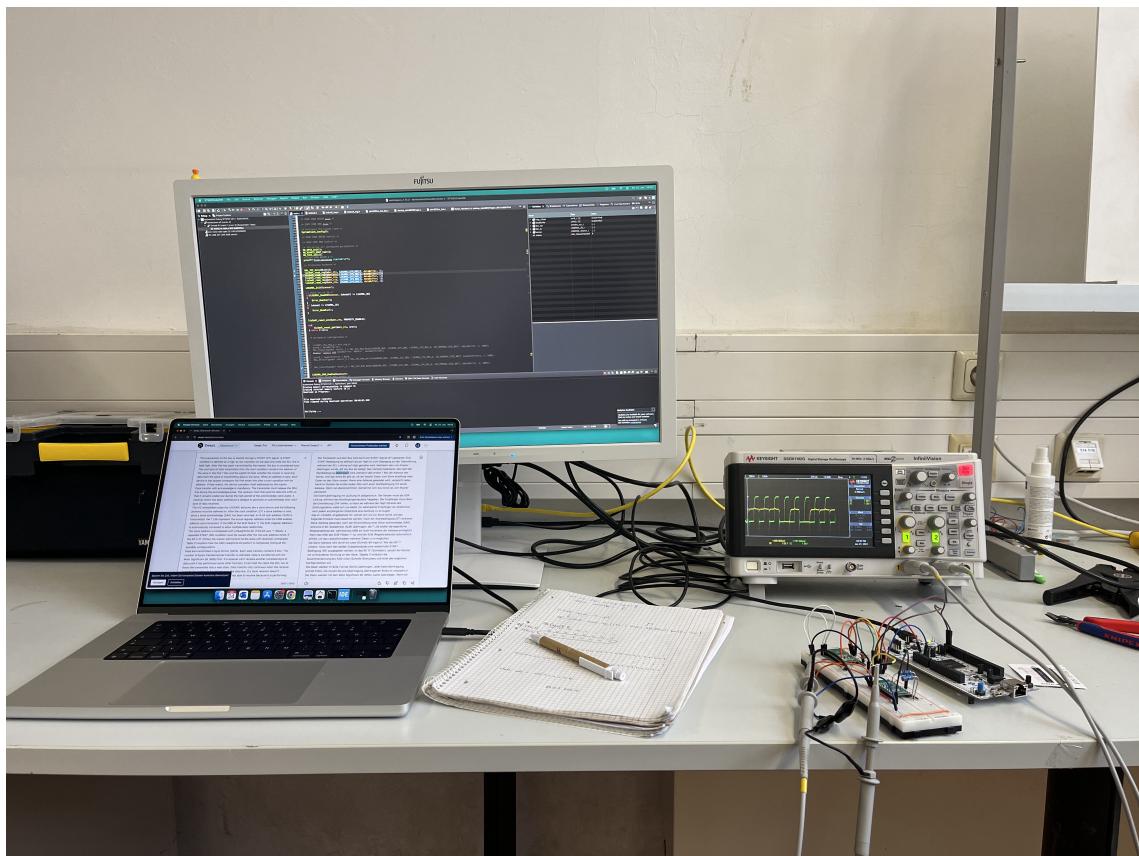


Abbildung 1.: Workstation

---

## B. Interface Funktionen

```
static int32_t platform_write(void *handle, uint8_t reg, const
→  uint8_t *bufp,
                               uint16_t len)
{
    printf("platform_write: %d %02X\n", reg, reg);
    reg |= 0x80U;
    printf("platform_write: %d %02X\n", reg, reg);
    return HAL_I2C_Mem_Write(handle, LIS2MDL_I2C_ADD, reg,
→   I2C_MEMADD_SIZE_8BIT, (uint8_t*) bufp, len, 1000);

}

static int32_t platform_read(void *handle, uint8_t reg, uint8_t
→  *bufp,
                               uint16_t len)
{
    printf("platform_read: %d %02X\n", reg, reg);
    reg |= 0x80U;
    printf("platform_read: %d %02X\n", reg, reg);
    return HAL_I2C_Mem_Read(handle, LIS2MDL_I2C_ADD, reg,
→   I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);

}
```

## C. Konfigurationsregister der Sensoren

Name	Type <sup>(1)</sup>	Register address		Default	Comment
		Hex	Binary		
Reserved		00 - 44			Reserved
OFFSET_X_REG_L	R/W	45	01000101	00000000	Hard-iron registers
OFFSET_X_REG_H	R/W	46	01000110	00000000	
OFFSET_Y_REG_L	R/W	47	01000111	00000000	
OFFSET_Y_REG_H	R/W	48	01001000	00000000	
OFFSET_Z_REG_L	R/W	49	01001001	00000000	
OFFSET_Z_REG_H	R/W	4A	01001010	00000000	
RESERVED		4B-4C			Reserved
WHO_AM_I	R	4F	01001111	01000000	
RESERVED		50-5F			Reserved
CFG_REG_A	R/W	60	01100000	00000011	Configuration registers
CFG_REG_B	R/W	61	01100001	00000000	
CFG_REG_C	R/W	62	01100010	00000000	
INT_CRTL_REG	R/W	63	01100011	11100000	Interrupt configuration registers
INT_SOURCE_REG	R	64	01100100		
INT_THS_L_REG	R/W	65	01100101	00000000	
INT_THS_H_REG	R/W	66	01100110	00000000	
STATUS_REG	R	67	01100111		
OUTX_L_REG	R	68	01101000	output	Output registers
OUTX_H_REG	R	69	01101001	output	
OUTY_L_REG	R	6A	01101010	output	
OUTY_H_REG	R	6B	01101010	output	
OUTZ_L_REG	R	6C	01101100	output	
OUTZ_H_REG	R	6D	01101101	output	
TEMP_OUT_L_REG	R	6E	01101110	output	Temperature sensor registers
TEMP_OUT_H_REG	R	6F	01101111	output	

1. R = read-only register, R/W = readable/writable register

Abbildung 2.: Register Mapping LIS2MDL

Name	Type <sup>(1)</sup>	Register address		Default	Comment
		Hex	Binary		
OUT_T_L	R	0D	00001101	00000000	
OUT_T_H	R	0E	00001110	00000000	Temp sensor output
WHO_AM_I	R	0F	00001111	01000100	Who am I ID
RESERVED	-	10-1F		-	RESERVED
CTRL1	R/W	20	00100000	00000000	
CTRL2	R/W	21	00100001	00000100	
CTRL3	R/W	22	00100010	00000000	
CTRL4_INT1_PAD_CTRL	R/W	23	00100011	00000000	
CTRL5_INT2_PAD_CTRL	R/W	24	00100100	00000000	
CTRL6	R/W	25	00100101	00000000	
OUT_T	R	26	00100110	00000000	Temp sensor output
STATUS	R	27	00100111	00000000	Status data register
OUT_X_L	R	28	00101000	00000000	
OUT_X_H	R	29	00101001	00000000	
OUT_Y_L	R	2A	00101010	00000000	
OUT_Y_H	R	2B	00101011	00000000	
OUT_Z_L	R	2C	00101100	00000000	
OUT_Z_H	R	2D	00101101	00000000	
FIFO_CTRL	R/W	2E	00101110	00000000	FIFO control register
FIFO_SAMPLES	R	2F	00101111	00000000	Unread samples stored in FIFO
TAP_THS_X	R/W	30	00110000	00000000	
TAP_THS_Y	R/W	31	00110001	00000000	
TAP_THS_Z	R/W	32	00110010	00000000	
INT_DUR	R/W	33	00110011	00000000	Interrupt duration
WAKE_UP_THS	R/W	34	00110100	00000000	Tap/double-tap selection, inactivity enable, wakeup threshold
WAKE_UP_DUR	R/W	35	00110101	00000000	Wakeup duration
FREE_FALL	R/W	36	00110110	00000000	Free-fall configuration
STATUS_DUP	R	37	00110111	00000000	Status register
WAKE_UP_SRC	R	38	00111000	00000000	Wakeup source
TAP_SRC	R	39	00111001	00000000	Tap source
SIXD_SRC	R	3A	00111010	00000000	6D source
ALL_INT_SRC	R	3B	00111011	00000000	
X_OFST_USR	R/W	3C	00111100	00000000	
Y_OFST_USR	R/W	3D	00111110	00000000	
Z_OFST_USR	R/W	3E	00000100	00000000	
CTRL7	R/W	3F	00000100	00000000	

1. R = read-only register, R/W = readable/writable register

Abbildung 3.: Register Mapping LIS2DW12

---

## D. Low-Level Funktion zum konfigurieren der Register

```
int32_t configureRegister_C(stmdev_ctx_t *ctx, lis2mdl_cfg_reg_c_t
→ *cfg_reg_c,
                           uint8_t *bitMask)
{
    lis2mdl_cfg_reg_c_t reg;
    lis2mdl_cfg_reg_c_t reg_readback;
    int32_t ret;
    uint8_t dataBuffer[1];

    // Setzen Sie hier die gewünschten Bits in reg.
    reg._4wspi = 0;
    reg.bdu = 0;
    reg.ble = 0;
    reg.drdy_on_pin = 1;
    reg.i2c_dis = 0;
    reg.not_used_02 = 0;
    reg.self_test = 0;
    reg.int_on_pin = 0;

    // Konvertieren Sie reg in einen Byte-Wert, um ihn ins Register
    → zu schreiben.
    memcpy(dataBuffer, &reg, sizeof(reg));
    ret = lis2mdl_write_reg(ctx, LIS2MDL_CFG_REG_C, dataBuffer, 1);
    if (ret != 0) {
        printf("Fehler beim Schreiben in Register C!\n");
        return ret;
    }

    // Warten Sie kurz, um dem Sensor Zeit zu geben, die
    → Konfiguration zu übernehmen.
    // (abhängig von den spezifischen Timing-Anforderungen des
    → Sensors)

    // Lesen Sie das Register zurück, um den aktuellen Zustand zu
    → überprüfen.
    ret = lis2mdl_read_reg(ctx, LIS2MDL_CFG_REG_C, dataBuffer, 1);
```

---

```
if (ret != 0) {
    printf("Fehler beim Lesen von Register C!\n");
    return ret;
}

// Konvertieren Sie das gelesene Byte zurück in die Struktur.
memcpy(&reg_readback, dataBuffer, sizeof(reg_readback));

// Vergleichen Sie die Struktur mit dem erwarteten Wert.
if (memcmp(&reg, &reg_readback, sizeof(reg)) == 0) {
    // Erfolgreich
    printf("Erfolgreich Register C konfiguriert!\n");
    // Hier könnten Sie cfg_reg_c und bitMask setzen, falls
    ↳ nötig.
    memcpy(bitMask, &reg, sizeof(reg)); // Optional, je nach
    ↳ Bedarf.
    *cfg_reg_c = reg;                // Optional, je nach
    ↳ Bedarf.
} else {
    // Fehlgeschlagen
    printf("Fehlerhaft Register C konfiguriert!\n");
}

return ret;
}

# Kommentar
Dargestellt ist eine Beispielfunktion zum konfigurieren des Registers
↳ C vom LIS2MDL.
Die Form der Funktion ist immer identisch, lediglich die Adressen der
↳ Register
ändern sich.
```