Semesterprojekt: Entwicklung eines Drehinstruments mit einem STM32-Microcontroller

Leon Sengün, 3526112

1 Aufgabenstellung

In diesem Bericht wird die Entwicklung eines Drehinstruments beschrieben, welches kontinuierlich seine Ausrichtung im Erdmagnetfeld und seine Drehgeschwindigkeit bestimmen soll. Basierend auf den gemessenen Sensordaten sollen unterschiedliche Outputs erfolgen: Bei Drehung im Uhrzeigersinn soll die grüne LED (LD3) Leuchten, bei Drehung Entgegen des Uhrzeigersinns die blaue LED (LD4). Die Helligkeit der LEDs soll folgendermaßen angepasst werden: Je höher der Betrag der Drehgeschwindigkeit, desto heller soll die jeweilige LED leuchten. Durch Drücken des blauen „USER"-Buttons soll für den Ausrichtungswinkel ein Nullwert eingestellt werden können. Wenn sich der Sensor innerhalb von ±10° des Nullwerts befindet, soll das System die jeweilige Abweichungsrichtung mittels Blinkens einer der LEDs anzeigen. Dabei soll bei Abweichung im Uhrzeigersinn (positiver Winkel) die grüne LED blinken, und bei Abweichung gegen den Uhrzeigersinn die blaue LED. Die Programmlogik soll Interrupt-gesteuert sein.

Das System besteht aus einem STM32F0DISCOVERY Entwicklungsboard, auf dem der STM32F051R8T6 Cortex-M0 Mikrocontroller mit 48MHz Taktfrequenz verbaut ist. Zur Lage- und Drehratenbestimmung kommen zwei Sensoren des Herstellers NXP zum Einsatz: Ein Gyroskop des Modells FXAS21002 und eine Kombination aus Beschleunigungssensor und Magnetometer, Modell: FXOS8700. Alle Sensoren sind auf einer gemeinsamen Platine montiert, und werden mit den jeweiligen Sensoradressen angesprochen.

2 Peripherie

Im Folgenden werden die genutzten Hardware-Schnittstellen, Timer und Interrupts erläutert. Die Konfiguration des Mikrocontrollers wurde dabei mit dem Device Configuration Tool innerhalb der STM32-CubeIDE durchgeführt.

2.1 Bus-Schnittstelle

Der Mikrocontroller kommuniziert mit den Sensoren via I^2C-Bus. Hierfür wird die Schnittstelle I2C1 im Standard Mode (100KHz) genutzt, da dieser für die vorliegende Anwendung hinreichend schnell ist.

Die genutzten Pins bzw. Anschlüsse sind in Tabelle 1 aufgelistet und in Abbildung 1 dargestellt. Zur einfachen Bedienung der Schnittstelle werden die Funktionen HAL_I2C_Mem_Read() und HAL_I2C_Mem_Write() genutzt. Um sicherzustellen, dass ein Lese- oder Schreibzugriff abgeschlossen ist, wenn der jeweilige Funktionsaufruf beendet wurde, findet die Kommunikation hierbei im Blocking Mode statt.

Beschreibung	Pin Entwicklungsboard	Pin Sensorboard
Versorgungsspannung	5V	VIN
Masse	GND	GND
SCL für I^2C	PB6	SCL
SDA für I^2C	PB7	SDA

Tabelle 1: Pin-Belegung

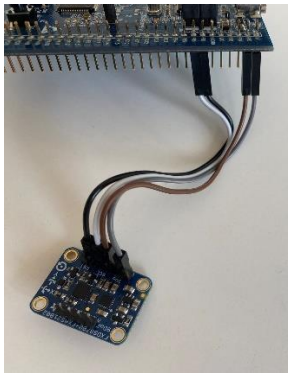

Abbildung 1: Anschluss des Sensorboards an das Entwicklungsboard

2.2 Sensor-Konfiguration

Die Sensoren werden mittels Beschreibens der Control-Register konfiguriert. Die Register des Gyroskops (FXAS21002) werden wie in Tabelle 2 aufgelistet eingestellt. CTRL_REG0 bestimmt dabei die Bandwidth und die Full-Scale Range und ist so gewählt, dass diese ±250 dps (Grad pro Sekunde) beträgt, was zu einer Sensitivität von 0.0078125 dps/LSB führt. CTRL_REG1 legt die Output Data Rate (ODR) auf 50 Hz fest und setzt das ACTIVE Bit auf 1, sodass der Sensor betrieben werden kann.

Register-Name	Adresse	Wert
CTRL_REG0	0x0D	0b00000011
CTRL_REG1	0x13	0b00010010

Tabelle 2: Control-Register für FXAS21002

Die Register der Magnetometer-Beschleunigungssensor-Kombination (FXOS8700) wurden wie folgt gewählt. Mittels CTRL_REG1 wird hier die ODR auf 50 Hz festgelegt wird, der low-noise-mode aktiviert, und das ACTIVE Bit auf 1 gesetzt. Mittels M_CTRL_REG1 wird die Auto-Kalibrierung aktiviert, die Oversampling Ratio (OSR) auf den Maximalwert 7 gesetzt, um Rauschen zu minimieren, Und der Beschleunigungssensor deaktiviert, da dieser für die vorliegende Anwendung nicht benötigt wird.

Register-Name	Adresse	Wert
CTRL_REG1	0x2A	0b00100101
M_CTRL_REG1	0x5B	0b10011111

Tabelle 3: Control-Register für FXOS8700

2.3 Timer

Insgesamt kommen bei diesem System drei Timer zur Verwendung. Timer TIM3 ist für die Helligkeitssteuerung der LED verantwortlich. Hierzu werden die beiden Channel CH3 und CH4 von TIM3 an den Pin der blauen (PC8) bzw. grünen (PC9) LED angeschossen, in den Modus „PWM Generation“ versetzt, und der Timer selbst auf eine Frequenz von 100 Hz eingestellt.

Beim Start des Systems werden zudem zwei Timer aktiviert, die jeweils bei jedem Ablauf einen Interrupt auslösen: Timer TIM2 wird im global interrupt mode betrieben und dessen Channel 1 ist in den Modus „Output Compare no output“ versetzt. Er ist so eingestellt, dass er mit der Frequenz von 2 Hz einen Interrupt auslöst, was der gewünschten Blinkfrequenz der LEDs entspricht. Timer TIM14 wird

ebenfalls im global interrupt mode betrieben und löst mit einer Frequenz von 50 Hz, die der ODR der beiden Sensoren entspricht, einen Interrupt aus, um von diesen Sensoren zyklisch die Daten auszulesen. Die nähere Konfiguration der Timer (Prescaler PSC, AutoReloadRegister ARR) ist in Tabelle 4 dargestellt.

Name	Zweck	PSC	ARR	Resultierende Frequenz
TIM2	Steuerung der Blinkfunktion	48000	500	2 Hz
TIM3	PWM-Signalgeber	480	1000	100 Hz
TIM14	Auslesen der Sensordaten	48000	20	50 Hz

Tabelle 4: Timer-Konfiguration

2.4 General-Purpose Input/Output (GPIO)

In dieser Peripherie wird lediglich von Pin PA0 Gebrauch gemacht, dessen Modus auf „External Interrupt Mode with Rising edge trigger detection“ gesetzt ist. Dieser Interrupt dient dazu, den Nullwert des Magnetometers beim Drücken des blauen Knopfes auf den zuletzt gemessenen Wert zu setzen. Dieser Wert gilt nun bis zum erneuten Betätigen des Knopfes als Referenzwert für die Blinkfunktion.

3 Programmlogik

Die Software ist strukturiert in eine Datei Sensor_comm.c, die die Kommunikation mit dem Sensor übernimmt, eine Datei stm32f0xx_it.c die einzelnen Interrupt-Service-Routinen definiert, sowie eine Datei main.c, in der die einzelnen Unterfunktionen aufgerufen werden.

3.1 Sensor_comm.c

Diese Datei besteht aus drei Funktionen, die zu verschiedenen Zeitpunkten im Programmablauf aufgerufen werden. Die Funktion initialize_sensors() prüft zunächst ob das WHO_AM_I Register den erwarteten wert zurückgibt und beschreibt dann die verschiedenen, in Abschnitt 2.2 erläuterten Control-Register beider Sensoren mit den entsprechend definierten Werten.

Die Funktion read_Sensor_data() liest jeweils 6 aufeinanderfolgende Register eines Sensors aus, beginnend mit dem Start_Register. Die 6 ausgegebenen Bytes stellen jeweils die Werte für die 3 gemessenen Achsen jedes der Sensoren dar, bestehend aus jeweils einem 16-Bit wert der in ein Least Significant Byte und ein Most Significant Byte aufgeteilt ist. Daher müssen anschließend MSB and LSB zusammengefügt werden, indem das zuerst ausgelesene Byte (MSB) um 8 Bits nach links verschoben und anschließend mit dem nicht verschobenen LSB verodert wird. Dies entspricht der Little-Endian Codierung.

Die Funktion convert_Sensor_data() errechnet mittels eines Skalierungsfaktors aus den Rohdaten eines Sensors den eigentlichen ausgabewert in der geeigneten Einheit (dps für das Gyroskop, µT für das Magnetometer).

3.2 stm32f0xx_it.c

In dieser Datei werden die einzelnen Interrupt-Service-Routinen der Timer-Interrupts definiert. TIM2 schaltet in seinem Interrupt-Handler TIM2_IRQHandler() zunächst beide LEDs, indem TIM3 auf beiden Channels gestoppt wird und somit kein PWM Signal für die LEDs existiert. Zudem schaltet der Handler eine Flag um, die signalisiert, ob die LED innerhalb des nächsten Timer-Zyklus eingeschaltet werden darf. Jede Halbsekunde springt diese flag also von 0 auf 1 bzw. umgekehrt, was den Takt für das Blinken der LED vorgibt. Der Interrupt-Handler von TIM14 ruft zyklisch die Funktionen sensor_routine() und sensor_average() auf, die weiter unten in Abschnitt 3.3 erläutert werden.

3.3 main.c

Dies ist die Hauptdatei der Programmstruktur, in der alle Aufrufe zum initialisieren der Peripherien, das starten der Timer sowie die zyklischen Funktionsaufrufe stattfinden.

Die Funktion LED_di m() prüft, welche LED leuchten muss und steuert zudem die Helligkeit der jeweiligen LED basierend auf der gemessenen Drehgeschwindigkeit, indem es den DutyCycle, also den Zeitanteil eines jeden PWM-Zyklus, abhängig vom Betrag der Drehgeschwindigkeit festlegt, und anschließend TIM3 für den jeweiligen channel startet, und mittels der Funktion __HAL_TIM_SET_COMPARE() den DutyCycle übergibt.

Die Funktion LED_flash() lässt abhängig von dem gemessenen Winkel bezüglich des Nullwerts eine der beiden LEDs mit einem festgelegten DutyCycle von 20% leuchten, bis diese wieder durch TIM2 ausgeschaltet wird (siehe Abschnitt 3.2).

Die Funktion HAL_GPIO_EXTI_Callback() wird bei Betätigung des blauen „USER“-Knopfes ausgelöst und setzt einen Nullwert φ_0 für den gemessenen Winkel in der x-y Ebene bezüglich des Erdmagnetfelds, gemäß Abschnitt 2.4. Der Winkel wird dabei mittels folgender Formel berechnet [1]:

$$\varphi_0 = \tan^{-1}\left(\frac{y}{x}\right) \cdot \frac{180°}{\pi}$$

Die notwendige Fallunterscheidung, in welchem Quadranten y und x liegen, wird dabei durch das verwenden der Funktion atan2(y, x) aus der math Library übernommen.

Die Funktion sensor_routine() ruft die Funktionen read_Sensor_data() und convert_Sensor_data() (siehe Abschnitt 3.1) für beide Sensoren auf und berechnet zudem aus den x- und y- Werten des Magnetometers den aktuellen Winkel φ in der x-y Ebene bezüglich des Erdmagnetfelds und speichert das Ergebnis in den nullten Eintrag des angle_array. Hierzu kommt die gleiche Formel wie in HAL_GPIO_EXTI_Callback() zum Einsatz.

Die Funktion sensor_average() berechnet aus den letzten 10 gemessenen Werten des Winkels im Erdmagnetfeld einen Durchnschnittswert schiebt die alten werte as angle_array je einen index weiter, sodass der älteste Wert aus dem Array herausfällt. Dies dient dazu, das starke Rauschen des Magnetsensors zu kompensieren, hat jedoch den Nachteil, dass es einen gewissen verzögerungseffekt bezüglich des aktuell gemessenen Sensorwerts erzeugt.

Die main()-funktion ist für den übergeordneten Programmablauf verantwortlich. In ihr werden die Peripherien initialisiert, die Timer für die interrupts gestartet und eine Endlosschleife ausgeführt. Innerhalb der Endlosschleife wird ständig der Winkel bezüglich des Nullwinkels

$$\varphi_N = \varphi - \varphi_0$$

berechnet, und basierend auf dem Ergebnis entweder die LED mittels LED_dim() zum Leuchten oder mittels LED_flash() zum Blinken gebracht. LED_flash wird dabei nur aktiviert, wenn die entsprechende Flag auf 1 gesetzt ist, und $|\varphi_N| > 0{,}2°$ (um Hin- und Herflackern zwischen grüner und blauer LED zu vermeiden) und $|\varphi_N| < 5°$ (gemäß Aufgabenstellung) ist. Ist der Betrag des $|\varphi_N| > 5°$, so wird LED_dim() aufgerufen. Eine Übersicht des Programmablaufs ist in Abbildung 2 als Flowchart dargestellt.

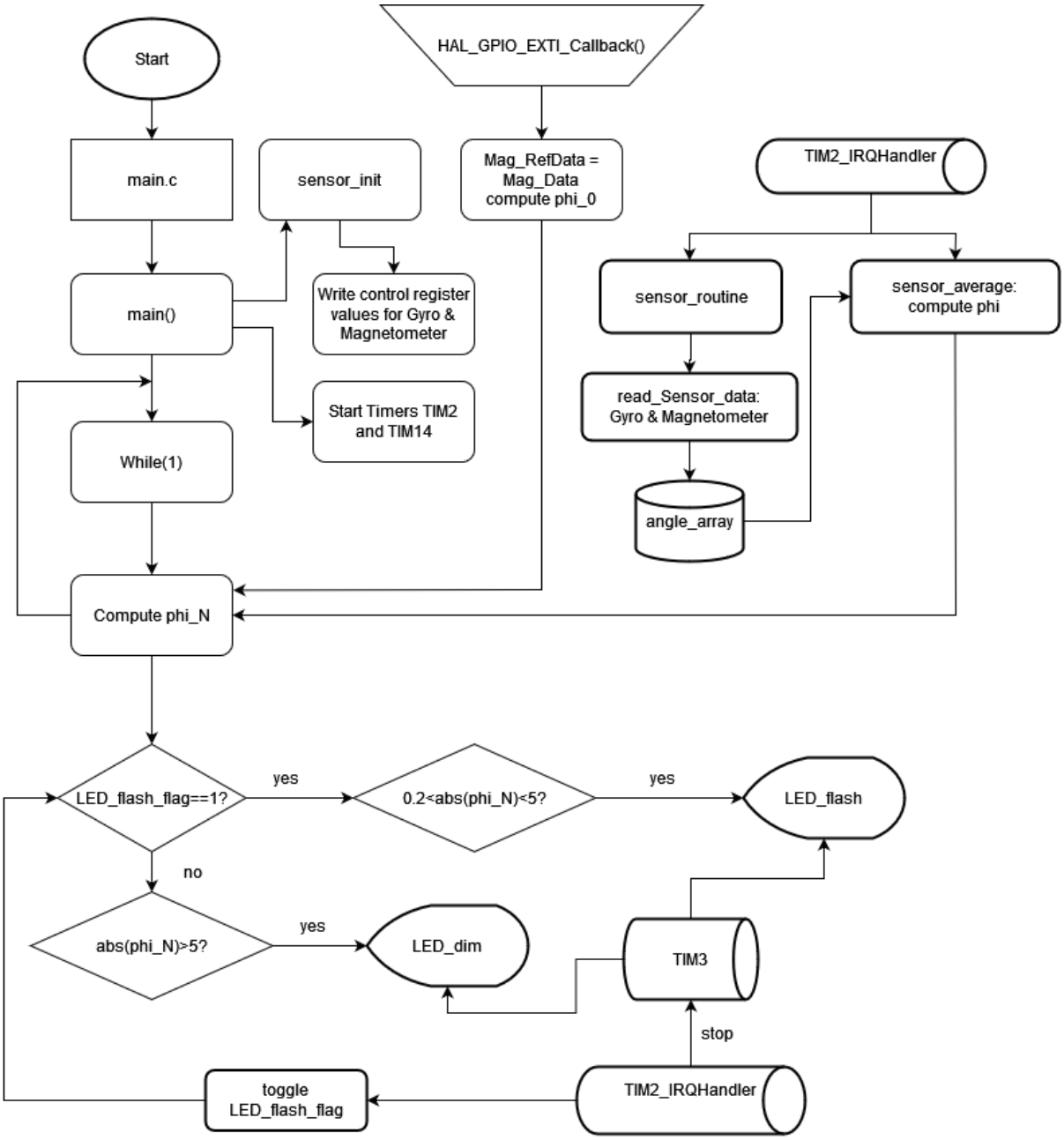

Abbildung 2: Flowchart des Programmablaufs

4 Gebrauchsanweisung

Zur Benutzung des Drehinstruments sind zunächst die Pins des Entwicklungsboards mit denen des Sensorboards, gemäß Tabelle 1 bzw. Abbildung 1 mittels der vorhandenen Kabel zu verbinden. Dann ist das Sensorboard mittels Mini-USB Kabel an eine geeignete Spannungsquelle, bspw. Den USB-Anschluss eines PCs zu verbinden. Sollte das gerät zunächst inaktiv sein, muss der schwarze „RESET“-Button gedrückt werden, um das Programm neu zu starten. Das Sensorboard ist nun so auf eine Flache Oberfläche zu platzieren, dass die aufgedruckte Z-Achse nach oben zeigt. Auf der Oberfläche kann es nun um die Z-Achse gedreht werden, und es leuchtet je nach Drehrichtung bzw. Drehgeschwindigkeit des Sensorboards die grüne bzw. blaue LED in ihrer entsprechenden Helligkeit. Bei Drücken des blauen „USER“-Buttons wird nun der Winkel tariert und in der nähe des Tara-Werts blinkt je nach Abweichungsrichtung die grüne oder blaue LED in einer langsamen Frequenz von 2Hz.

5 Ausblick

Das System ist in der vorliegenden Art noch nicht vollständig ausgereift. Folgende Features könnten in weiterer Bearbeitung implementiert werden:

- Bessere Rauschreduktion durch einen Low-Pass-filter
- Sensorfusion aus Gyroskop und Magnetometer, um eine höhere Genauigkeit zu erreichen
- Manuelle Kalibrierung des Magnetometers durch Messung von Minimal- und Maximalwerten des Magnetfelds in jede Richtung
- Verwenden des Beschleunigungssensors zur Bestimmung, ob der Sensor in einer geeigneten Position (Ausrichtung in horizontaler Richtung) liegt

Verweise

[1] E. Bonet, "Converting three axis magnetometer to Degrees - arduino stack exchange," 27. 10. 2018. [Online]. Available: https://arduino.stackexchange.com/questions/18625/converting-three-axis-magnetometer-to-degrees.