Sensor-Fusion zur Lagebestimmung im R^3

Pablo Grimm
Archenhold-Gymnasium

Niklas Schmidt
Meteum

22. Februar 2016

Inhaltsverzeichnis

1. Messgrößen **4**
- 1.1. Raumachsen und -winkel . . . 4
- 1.2. Notwendige Messdaten . . . 5

2. Hardwarebeschreibung **6**
- 2.1. Beschleunigungssensor . . . 6
- 2.2. Gyrometer . . . 7

3. Aufbereitung der Messwerte **8**
- 3.1. Glättung . . . 8
  - 3.1.1. Lineare Glättung . . . 8
  - 3.1.2. Exponentielle Glättung . . . 8
  - 3.1.3. Eigenschaften . . . 9
- 3.2. Bestimmung der Neigungswinkel . . . 9
- 3.3. Filter . . . 11
  - 3.3.1. Komplementärfilter . . . 11

4. Softwarebeschreibung **12**

5. Bisherige Ergebnisse **13**
- 5.1. Bestimmung der Raumwinkel . . . 13
- 5.2. Drift des Gyrometers . . . 13
- 5.3. Ausblick . . . 13

A. Quellcode **16**
- A.1. Hauptprogramm . . . 16
- A.2. Angepasste libmaple-Bibliotheken . . . 20

Mitschüler der Robotik AG wiesen uns darauf hin, dass die Bestimmung der Neigung und Position eines Roboters relativ zu seinem Ausgangspunkt Schwierigkeiten bereitet. Sollen Sensoren zur Lagebestimmung im Raum, also zur Messung von Neigungswinkeln oder Position eines Objekts relativ zum Koordinatenursprung, eingesetzt werden, so kommt es zu Messfehlern.

Beschleunigungssensoren messen axial gerichtete Beschleunigungen und sind von Vibrationen und plötzlichen Bewegungen beeinflusst. Aufgrund dessen, dass dieser Sensor von äußeren Kräften beeinflusst wird, können Berechnungen der Neigung verfälscht werden, wenn der Sensor bewegt wird.

Gyrometer, die Winkelgeschwindigkeiten messen, haben ein Problem mit Drift. Andererseits arbeiten sie relativ genau und werden von äußeren Kräften nicht beeinflusst.

Wir wollen die Messwerte eines Beschleunigungssensors und eines Gyrometers miteinander verknüpfen.

Ziel des Projekts ist es, einen Überblick über verwendbare Filtermethoden zu gewähren und eine Implementierung in C++ aufzuzeigen. Als Hardwareplattform wird ein Evaluierungsboard „STM32F3 Discovery“ des Herstellers STMicroelectronics verwendet.

1. Messgrößen

Interessant für unsere Anwendung ist die Rotation um die z-Achse, die axiale Beschleunigung und die Neigung des Roboters.

1.1. Raumachsen und -winkel

Es wird das für Landfahrzeuge gedachte ENU-System (East-North-Up) verwendet. Dabei wird die Postion des Fahrzeugs im Raum in einem festen Referenzsystem (engl. world frame) angegeben. Die Neigung des Fahrzeugs wird mittels folgender drei Winkel bestimmt:

- Rollwinkel Φ (engl. roll): Drehung um x-Achse
- Nickwinkel Θ (engl. pitch): Drehung um y-Achse
- Gierwinkel Ψ (engl. yaw): Drehung um z-Achse

> „Im Einzelnen beschreiben die Winkel dabei drei aufeinander folgende Drehungen, die ein festes Referenzsystem (engl. world-frame) in ein objektbezogenes rechtshändiges Koordinatensystem (engl. body frame) überführen.“ [15a]

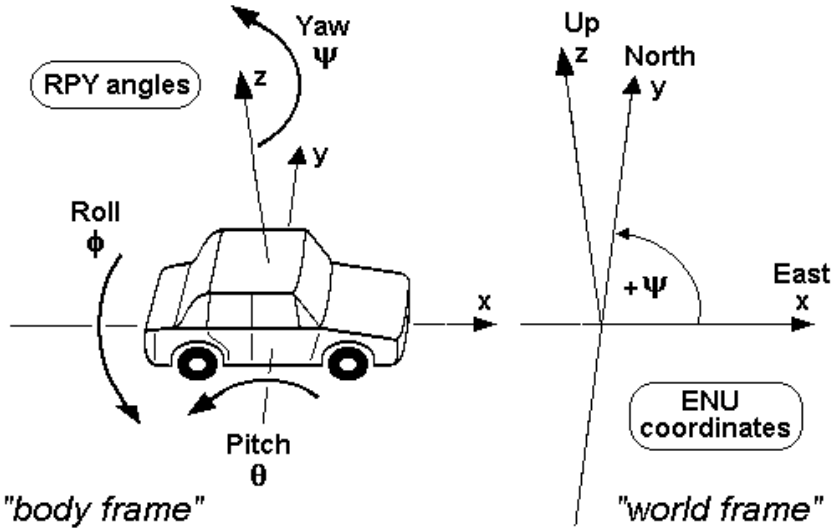

Abbildung 1.1.: Roll-, Nick- und Gierwinkel [Qni10]

1.2. Notwendige Messdaten

Die Rotation um die z-Achse (Gierwinkel) ist nötig, um die Richtung zu bestimmen, in die der Roboter fährt.

Axial gerichtete Beschleunigungen sind notwendig, um die Geschwindigkeit des Roboters bestimmen zu können. Aus ihnen lässt sich auch die zurückgelegte Stecke ableiten.

Die Neigung wird gebraucht, um festzustellen, ob eine Rampe befahren wird oder der Roboter droht umzukippen.

Wenn all diese Daten verwendet werden, kann die Position des Roboters im Raum bestimmt werden.

2. Hardwarebeschreibung

Das verwendete Evaluierungsboard „STM32F3 Discovery“ verfügt über ein dreiachsigen Beschleunigungssensor und ein dreiachs-Gyrometer.

Die Messdaten werden seriell per UART mit 3,3V TTL-Pegel übertragen. Der empfangende Rechner hat einen USB-UART.

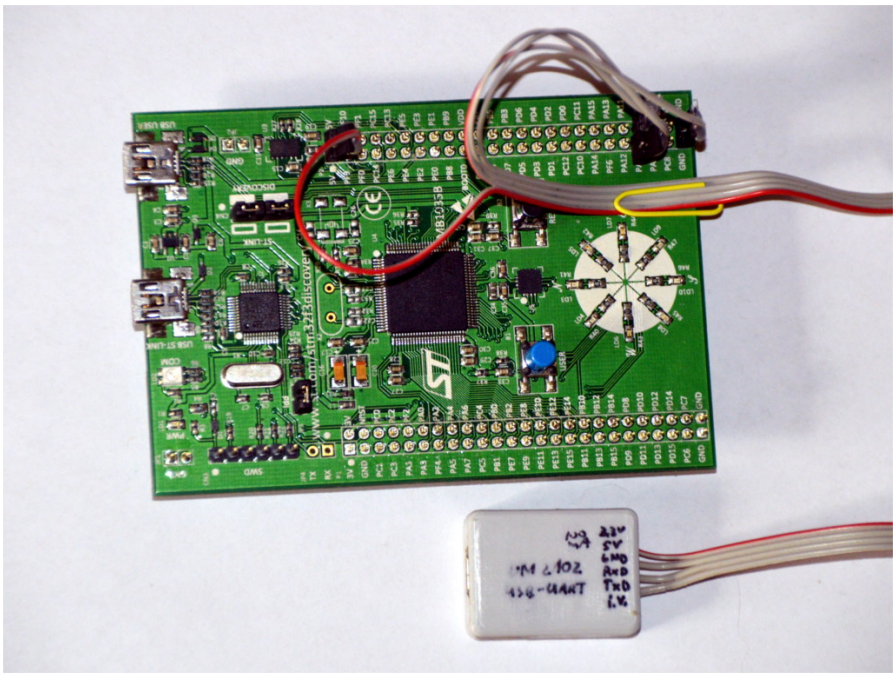

Abbildung 2.1.: STM32F3 Discovery mit USB-Seriell-Wandler

2.1. Beschleunigungssensor

Der verwendete Sensor LSM303 ist über den I^2C-Bus angebunden. Es handelt sich um ein Mikrosystem (MEMS, Mikroelektromechanisches System), was bedeutet, dass Sensormechanik und Auswerteelektronik auf einem Chip untergebracht sind.

Wird der Sensor beschleunigt, so wird eine federnd befestigte Probemasse aufgrund ihrer Massenträgheit ausgelenkt. An der Probemasse befestigte Kondensatorplatten verändern ihren Abstand zu am Sensorgehäuse befestigten Platten. Die so entstehende Kapazitätsänderung des Plattenkondensators kann elektronisch erfasst werden.

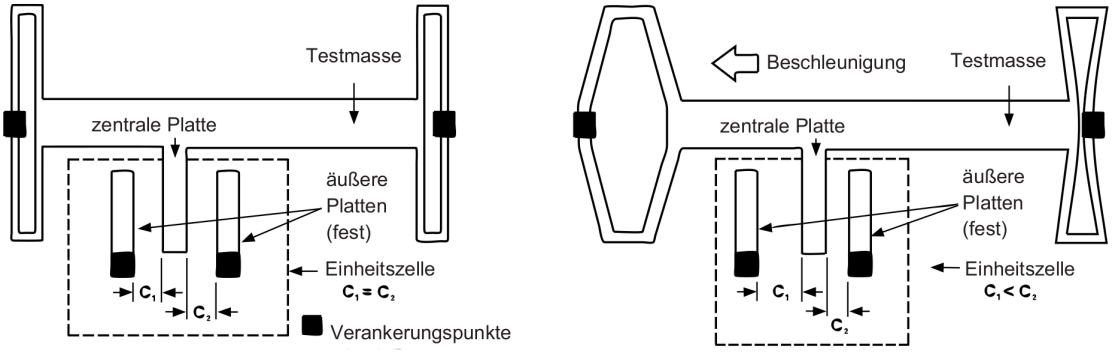

Abbildung 2.2.: MEMS Beschleunigungssensor [Kli06]

2.2. Gyrometer

Es wird ein stabförmiges Piezoelement in Schwingung versetzt. Wird es verdreht, so wird diese vertikale Oszillation mit einer horizontalen, durch die Corioliskraft hervorgerufenen, überlagert.

Durch den piezoelektrischen Effekt können vom Piezoelement zur Auslenkung der Schwingungen proportionale Spannungen abgenommen werden. Aus den horizontalen und vertikalen Komponenten der Schwingung lässt sich die Rotationsgeschwindigkeit ermitteln.

Diese Technologie wird *vibrating structure gyroscope* [Vgl. 15b] genannt. Auch hierbei handelt es sich um ein MEMS.

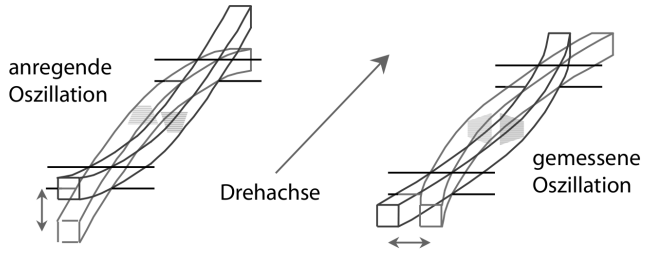

Abbildung 2.3.: MEMS Gyrometersensor [Kli06]

3. Aufbereitung der Messwerte

3.1. Glättung

Glättung dient dazu, Ausreißer[1] zu entfernen, damit die weitere Signalverarbeitung nicht gestört wird.

Um die qualitative Eignung einer Glättungsmethode zu beurteilen ist es sinnvoll, ihre Reaktion auf Sprünge des Eingangssignals zu betrachten, also auf plötzliche Anstiege oder Abfälle der Sensormesswerte.

3.1.1. Lineare Glättung

Die einfachste Art der Glättung ist die Bildung des arithmetischen Mittels $\bar{x}$ aus den letzten n Messwerten. Es gilt:

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

Dies wird als lineare Glättung, gleitender Mittelwert oder engl. linear smoothing bezeichnet.

Implementierung Die letzten n Messwerte werden in einem Array, welches als Ringpuffer arbeitet, vorgehalten. Zusätzlich zum Array wird eine Variable mitgeführt. Es wird pro neuem Messwert der älteste Wert von dieser Variablen subtrahiert und jener neue addiert. Die Variable enthält dann den n-fachen Mittelwert.

3.1.2. Exponentielle Glättung

Umso neuer der Messwert, desto größer die ihm zugewiesene Gewichtung. Das Filter hat eine unendliche Impulsantwort.

$$\begin{aligned} y_t^* &= a \cdot y_t + a \cdot (1-a) \cdot y_{t-1} + a \cdot (1-a)^2 \cdot y_{t-2} + ... \ + a \cdot (1-a)^{t-1} \cdot y_1 - (1-a)^t \cdot y_0^* \\ &= (1-a)^t \cdot y_0^* + \sum_{i=1}^{t} a \cdot (1-a)^i \end{aligned}$$

[1] sprunghafte Abweichung des Messsignals, etwa durch Rauschen oder Einstrahlung in analogen Schaltungsteilen

3.1.3. Eigenschaften

Lineare Glättung

Eine lineare Glättung kann jeweilig für ihren Bereich n, in dem die Glättung durchgeführt wird, nur eine allmähliche, lineare Annäherung an den neuen Wert erreichen. Sie nimmt dann aber, sofern dieser konstant bleibt, diesen auch vollständig an. Dies wird als endliche Impulsantwort bezeichnet.

Exponentielle Glättung

Eine exponentielle Glättung gleicht sich hingegen recht schnell an den neuen realen Wert an, wie der Name verrät geschieht dies exponentiell. Hier ist zu beachten, dass vorherige Werte mit der Zeit zwar an Gewichtung verlieren, aber nie ganz unbeachtet sein werden. Dies wird als unendliche Impulsantwort bezeichnet.

Ihre Sprungantwort entspricht zum Beispiel der Lade- bzw. Endladekurve eines über einen Ohmschen Widerstand geladenen Kondensators. Somit ist die exponentielle Glättung in ihrer Funktion mit einem RC-Tiefpass vergleichbar.

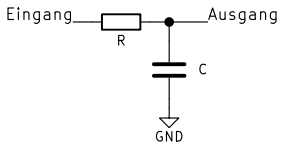

Abbildung 3.1.: RC-Tiefpassfilter

Die Sprungantworten beider Glättungsmethoden sind in Abbildung 3.2 dargestellt.

3.2. Bestimmung der Neigungswinkel

Befindet sich der Beschleunigungssensor in Ruhe oder der gleichförmig-geradlinigen Bewegung, so wirken – mit Ausnahme der senkrecht nach unten gerichteten Fallbeschleunigung $g = 9,81\frac{m}{s^2}$ – keine Beschleunigungen auf ihn ein.

Je nach Neigung des Sensors verteilt sich die Fallbeschleunigung auf alle drei Achsen, vergleichbar mit der in Abbildung 3.3 gezeigten Kräftezerlegung an einer schiefen Ebene.

Roll- und Nickwinkel lassen sich mithilfe der Seiten-Winkel-Beziehungen am rechtwinkligen Dreieck ermitteln.

$$\Phi = \arctan \frac{a_y}{\sqrt{a_x^2 + a_z^2}}$$
$$\Theta = \arctan \frac{a_x}{\sqrt{a_y^2 + a_z^2}}$$

Der Gierwinkel ϕ lässt sich nicht mit dem Beschleunigungssensor bestimmen, da es sich bei ϕ lediglich um eine Rotation um die z-Achse handelt und sich damit, im ruhenden

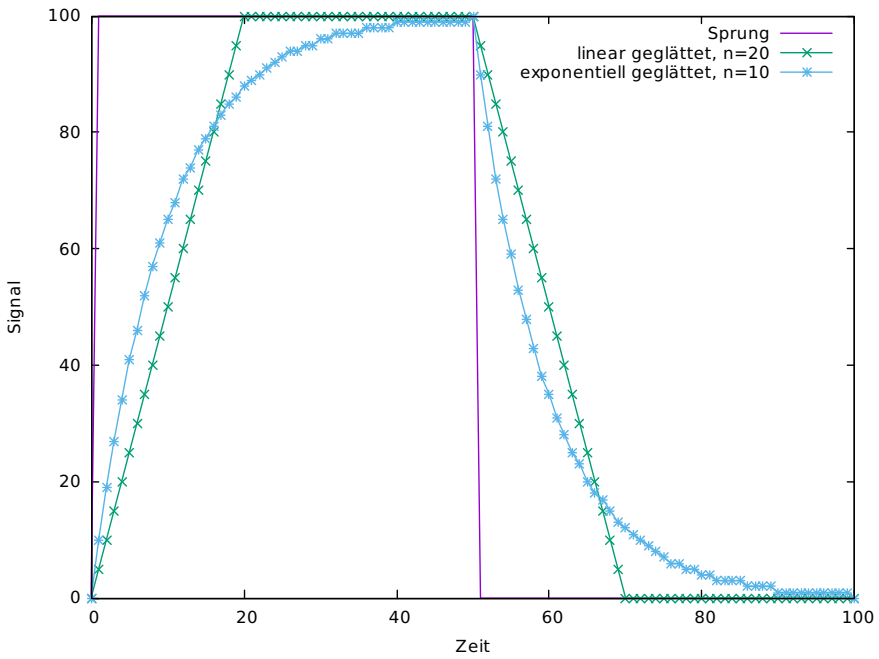

Abbildung 3.2.: Sprungantwort der linearen und exponentiellen Glättung

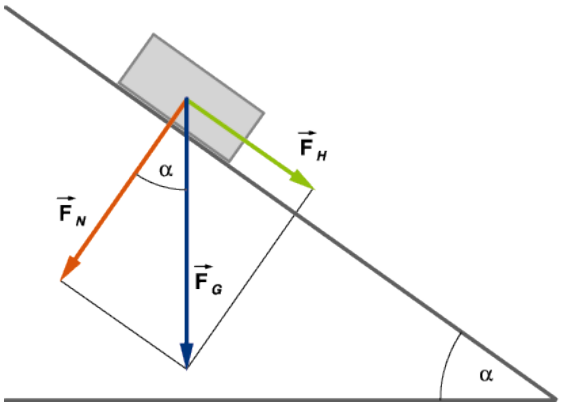

Abbildung 3.3.: Kräftezerlegung an der schiefen Ebene [11]

System, keine Beschleunigung auf andere Achsen verteilt. Hier können nur Drehraten des Gyrometers zurate gezogen werden.

3.3. Filter

3.3.1. Komplementärfilter

Die Messwerte zweier Sensoren werden dem Korrekturfaktor $0 \leq M \leq 1$ entsprechend gemischt. Der Filter ist somit zweidimensional.

Dabei wirkt der Komplementärfilter für den Gyrosensor als Hochpassfilter, dass die niederfrequente Drift herausfiltert. Die Messdaten des Beschleunigungssensors werden durch einen Tiefpass vom Rauschen befreit.

Hier ist die Gleichung für die Rotation um eine Achse gegeben, wobei ω die Winkelgeschwindigkeit vom Gyro und $\arctan \frac{a_{x|y}}{a_z}$ der Winkel aus der Kräftezerlegung ist.

$$\alpha' = (1 - M) \cdot (\alpha + \omega \Delta t) + M \cdot \arctan \frac{a_{x|y}}{a_z}$$

Dieser Filter ist mathematisch relativ einfach. Obwohl der Mahony-Madgwick-Filter oder der Kalmanfilter eine höhere theoretische Genauigkeit bieten, reicht die Präzision dieses Filters für viele Anwendungen aus. [Vgl. Mae13]

4. Softwarebeschreibung

Aufgrund der relativ hohen Komplexität des ARM Cortex M3 Prozessorkerns und seiner Peripherie ist es sinnvoll, auf diese Hardware zugeschnittene Softwarebibliotheken zu verwenden. Durch diese Abstraktion wird der eigentliche Programmcode übersichtlicher und plattformunabhängiger.

Da die von STMicroelectronics zum STM32F303 mitgelieferten Bibliotheken unter einer unfreien Lizenz stehen, was das Verändern und Weitergeben erschwerte, wurde den unter der MIT-Lizenz stehenden Bibliotheken des libmaple-Projekts [Lea] Vorzug gegeben.

Die Übertragung der Messwerte der Sensoren an den Mikrocontroller erfolgt in einem Timerinterrupt. Aus den gemessenen Beschleunigungen werden Roll- und Nickwinkel berechnet und zusammen mit den vom Gyrometer erfassten Winkeländerungen an den Komplementärfilter gegeben. Das Integrieren der Messwerte des Gyrometers erfolgt dabei bereits durch dessen internen FiFo-Buffer.

Aus den Bibliotheken zur Ansteuerung der Sensoren wurden alle Maßnahmen zur Mittelwertbildung entfernt. Es wurde eine Funktion zum Auslesen des internen Temperatursensors des Gyrometers hinzugefügt.

Ein einachsiger Komplementärfilter ist als Klasse angelegt. Es werden drei Objekte dieser Klasse erzeugt: die Filter für den Roll-, Nick- und Gierwinkel.

5. Bisherige Ergebnisse

5.1. Bestimmung der Raumwinkel

Bisher können wir Roll- und Nickwinkel mithilfe des Komplementärfilters zuverlässig bestimmen. Dazu genügt ein Korrekturfaktor von $M = 0,02$; der aus den Messwerten des Beschleunigungssensors berechnete Winkel geht also nur mit 2% in das Ergebnis der Winkelmessung ein.

5.2. Drift des Gyrometers

Die Bestimmung des Gierwinkels bereitet Probleme, da hier keine Messwerte des Beschleunigungssensors als feste Referenzwerte herangezogen werden können. Es gilt folglich, die Drift des Gyrometersensors so gut wie möglich zu kompensieren. Sie ist temperaturabhängig. Zu diesem Zweck wurde die gesamte Platine in einem Ofen erwärmt und die Drift kontinuierlich gemessen. Wie in Abbildung 5.1 zu sehen ist, verhält sich die Drift über einen weiten Temperaturbereich annähernd linear.

5.3. Ausblick

Im weiteren Verlauf des Projektes wollen wir die Gyrodrift in Abhängigkeit zur Temperatur kompensieren, damit einhergehend auch die Drift der Gyrometermesswerte so gut wie möglich ausgleichen, um so eine genaue Messung des Gierwinkels zu ermöglichen.

Auch wollen wir verschiedene Bewegungen aufnehmen und gezielt Neigungen und Drehungen erzeugen, um die Funktionalität unserer Anwendung zu testen. Hierzu werden wir Beschleunigungsmesswerte auswerten, um einen qualitativen Vergleich mit der aufgenommenen Bewegung zu ermöglichen, die sich unabhängig messen lässt (Geschwindigkeit, Weg).

Mit Versuchsserien zur Neigung werden wir versuchen, einen möglichst optimalen Gewichtungsfaktor für den Komplementärfilter zu finden.

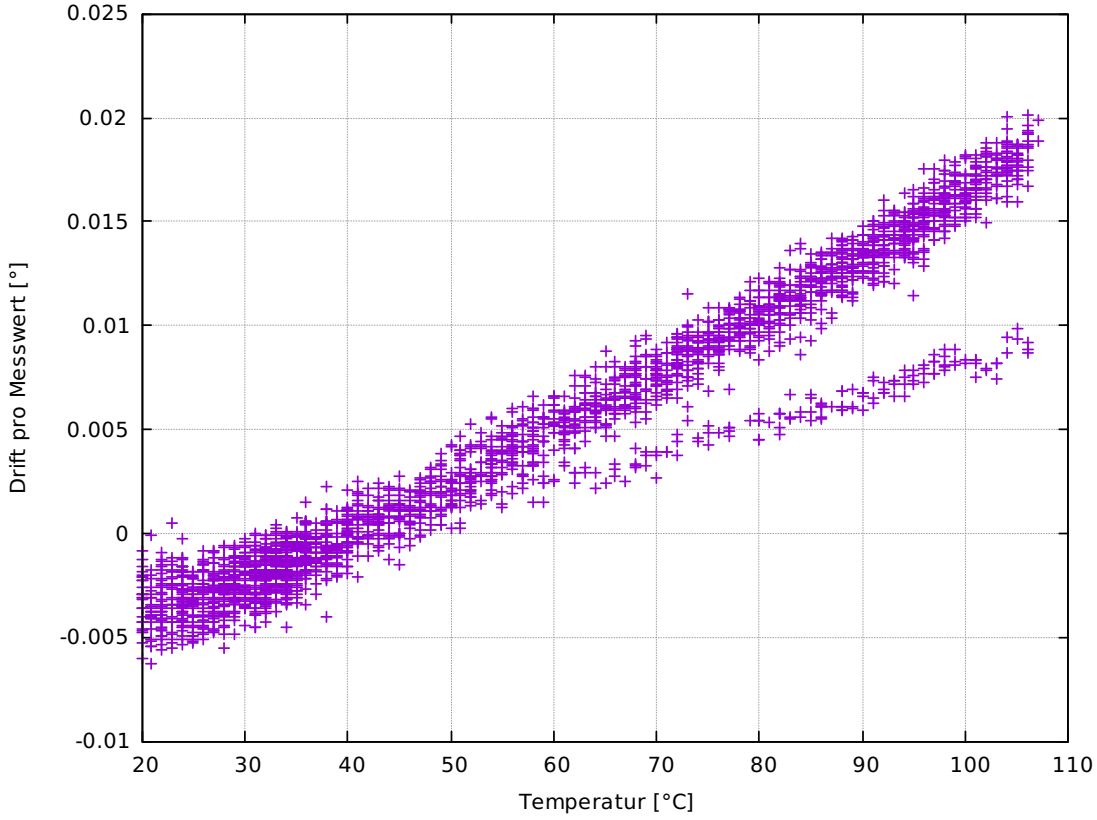

Abbildung 5.1.: Drift abhängig von der Chiptemperatur

Literatur

[11] *Nutzung der Kraftzerlegung, Anwendung auf die schiefe Ebene*. Universität Frankfurt. 9. Sep. 2011. URL: https://elearning.physik.uni-frankfurt.de/data/FB13-PhysikOnline/lm_data/lm_324/daten/bild_1/02_0070.gif (besucht am 13. 01. 2016).

[15a] *Roll-Nick-Gier-Winkel*. Wikipedia. 19. Dez. 2015. URL: http://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel.

[15b] *Vibrating structure gyroscope*. Wikipedia. 31. Dez. 2015. URL: https://en.wikipedia.org/wiki/Vibrating_structure_gyroscope.

[Kli06] Lasse Klingbeil. „Entwicklung eines modularen und skalierbaren Sensorsystems zur Erfassung von Position und Orientierung bewegter Objekte“. Dissertation. Universität Bonn, 2006. URL: http://nbn-resolving.de/urn:nbn:de:hbz:5N-07154 (besucht am 25. 01. 2016).

[Lea] LeafLabs. *libmaple*. URL: http://leaflabs.com/docs/libmaple.html (besucht am 31. 01. 2016).

[Mae13] Pieter-Jan van de Maele. *Reading IMU Whitout Kalman: The Complementary Filter*. 26. Apr. 2013. URL: www.pieter-jan.com/node/11 (besucht am 29. 01. 2016).

[Qni10] Qniemiec. *File:RPY angles of cars.png*. 15. Juli 2010. URL: https://commons.wikimedia.org/wiki/File:RPY_angles_of_cars.png.

A. Quellcode

A.1. Hauptprogramm

```
#include <wirish/wirish.h>   // IO-Zugriff
#include <libmaple/i2c.h>    // I2C-Bus
#include <math.h>
#include "lsm303.h"          // Beschleunigungssensor
#include "gyro.h"            // Gyro
#include "debug.h"           // serielles Debuggen

#define MS *1000
#define MESSRATE 10 MS       // in Mikrosekunden: 10ms

class ComplFilter {
    private:
    float angle;
    float m;                 // Korrekturfaktor

    public:
    ComplFilter( float correct ) {
        m = correct;
    }

    void update( float gyroAngleDiff, float accAngle ) {
        // Integration im Gyro-FiFo!
        angle = (1-m) * (angle + gyroAngleDiff) + m * accAngle;
    }
    float get( void ) {
        return angle;
    }
};

class Pos {
    private:
    float dt;
    float weg, geschw, beschleu;
```

```
    public:
    Pos( float timediff ) {
        weg = 0.0; geschw = 0.0;
        dt = timediff;
    }
    void update( float beschl ){
        beschleu = beschl;
        geschw += beschleu * dt;
        weg += geschw * dt;
    }
    float get() {
        return beschleu;
    }
};

float rad2deg( float rad ) {
    return rad*(180/PI);
}
float deg2rad( float deg) {
    return deg/(180/PI);
}

HardwareTimer timerimu(2);  // Timer für IMU-Update-Interrupt

ComplFilter roll(0.02), nick(0.02), gier(0.00);
                            // keine Korrektur für Gierwinkel möglich

Pos xpos(0.01), ypos(0.01), zpos(0.01);

bool printFlag = false;
int printCounter = 0;

void handler_imu(void);

int g_acc_norm;
// Ermitteln der Fallbeschleunigung
void acc_cal() {
    int i;
    long sum;
    sum =0;
    for(i=0;i<100;i++) {
        readAccValues();
        sum += sqrt(accx*accx + accy*accy + accz*accz);
```

```
    }
    g_acc_norm = sum/100;
}

void gyroLedInit() {
    /* Timer und Interupthandler */
    timerimu.pause();
    timerimu.setPeriod(MESSRATE);
    timerimu.setChannel1Mode(TIMER_OUTPUT_COMPARE);
    timerimu.setCompare(TIMER_CH1, 1);
    timerimu.attachCompare1Interrupt(handler_imu);
    timerimu.refresh();

    /* LED */
    pinMode(PE9, OUTPUT);

    /* Beschleunigungssensor initialisieren */
    i2c_master_enable(I2C1, 0);
    lsm303Init();

    /* Gyro initialisieren */
    gyroSpiInit();
    gyroInit();
    gyroStart();
    delay(100);
    digitalWrite(PE9, HIGH);// LED
    gyroCalibrate();
    delay(100);
    digitalWrite(PE9, LOW);
    gyroResetWinkel();
    delay(50);

    acc_cal();

    timerimu.resume();        // Timer starten

    Serial1.println("\n\rroll;\tnick;\tgier;\ttemp;\txweg;\tyweg");
}

void loop() {
    /* serielle Ausgabe der Messwerte */
    if( printFlag ) {

        Serial1.print( roll.get(), 0 );
```

```
        Serial1.print("; \t");
        Serial1.print( nick.get(), 0 );
        Serial1.print("; \t");
        Serial1.print( gier.get(), 0 );
        Serial1.print( "; \t" );
        Serial1.print( gyroGetTemp() );
        Serial1.print( "; \t" );
        Serial1.print( xpos.get(), 4 );
        Serial1.print( "; \t" );
        Serial1.print( ypos.get(), 4 );
        Serial1.print( "; \t" );
        Serial1.print( zpos.get(), 4 );

        Serial1.print("; \r\n");

        printFlag = false;
    }
}

void handler_imu(void) {
    digitalWrite(PE9, !digitalRead(PE9));    // LED togglen

    readAccValues();

    gyroUpdate();

    // winkel im bogenmaß
    float phi, theta, psi;
    phi = -atan2(accy, sqrtf(accz*accz + accx*accx) );
    theta = -atan2(accx, sqrtf(accz*accz + accy*accy) );

    /* Bogenmaß in Gradmaß umrechnen, Filter updaten */
    roll.update( winkely, phi * 180/PI );
    nick.update( winkelx, theta * 180/PI );
    gier.update( winkelz, 0 );

    float g_x, g_y, g_z;

    g_x = - sinf(theta);
    g_y = sinf(phi) * cosf(theta);

//  g_x = - sinf(deg2rad(nick.get()));
//  g_y = sinf(deg2rad(roll.get())) * cosf(deg2rad(nick.get()));
    g_z = cosf(deg2rad(roll.get())) * cosf(deg2rad(nick.get()));
```

```
    xpos.update(accx / float(g_acc_norm) );// - g_x );
    ypos.update(-accy / float(g_acc_norm) );//- g_y );
    zpos.update(accz / float(g_acc_norm) );
    /*
    printCounter ++;
    if( printCounter >= 10 ){    // Ausgabe alle 50 Messungen
        printCounter = 0;
        printFlag = true;
    }
    */
    printFlag = true;
}
```

A.2. Angepasste libmaple-Bibliotheken

Hierbei handelt es sich um die angepassten libmaple-Bibliotheken[Lea]. Sie wurden von Michael Krause auf das „STM32F3-Discovery“ portiert.

Im wesentlichen wurden Ringpuffer, die zur Glättung der Messwerte gedacht waren, entfernt. Diese Funktion übernehmen unsere Filteralgorithmen.

gyro.cpp

```
/*****************************************************************************
 * The MIT License
 *
 * Copyright (c) 2010 LeafLabs LLC.
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies
 * of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
```

```
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
 * BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 ****************************************************************************/

/**
 * @brief functions for accessing l3g20 gyro on stm32f3discovery board
 *
 * SPI1 is set up to be a master
 *      at 9 MHz (SPI_1_125MHZ),
 *      8 Bit
 *      MSB first in/out  (MSBFIRST)
 *      MODE 0 // following the datasheet it should be MODE 3 but 0 is
 *      working too
 *
 * Pin PE3 is used as slave select, as defined for SPI1 in board.h
 *
 */

#include <wirish/wirish.h>
#include "l3gd20.h"
#include "../RingBuffer/ringbuffer.h"
#include "gyro.h"


#define NSS PE3
#define GYRO_FIFO_WATERMARK 10

HardwareSPI spi1(1);
static uint8 gyroValuesSingle[6];

static int GYRO_BIAS_X =  -75;
static int GYRO_BIAS_Y =  94;
static int GYRO_BIAS_Z =  0;

float winkelx, winkely, winkelz;
int gyroxRaw, gyroyRaw, gyrozRaw;
            // bei jedem Update wird ein Datensatz der gelesenen
            // Werte in den Raw-Variablen gespeichert um den Bias
            // prüfen zu können
```

```
void readGyroValues()
{
    int j;
    digitalWrite(NSS, LOW);
    spi1.transfer(0xA8|0x40);

    for (j=0; j<6; j++)
        gyroValuesSingle[j] =  spi1.transfer(0);

    digitalWrite(NSS, HIGH);
}

uint8 gyroReadRegister(uint8 reg)
{
    digitalWrite(NSS, LOW);
    spi1.transfer(reg|0x80);

    uint8 val = spi1.transfer(0);

    digitalWrite(NSS, HIGH);
    return val;
}

void gyroWriteRegister(uint8 reg, uint8 val)
{
    digitalWrite(NSS, LOW);
    spi1.transfer(reg);
    spi1.transfer(val);
    digitalWrite(NSS, HIGH);
}

/**
 * Initialisiert das spi1 interface des stm32f3discovery
 */
void gyroSpiInit()
{
    spi1.begin(SPI_9MHZ, MSBFIRST, 0);
    /* NSS is usually active LOW, so initialize it HIGH */
    pinMode(NSS, OUTPUT);
    digitalWrite(NSS, HIGH);
}

#define GYRO_OP_PARAMS (GYRO_ODR_190|GYRO_LP_BW1|GYRO_ENABLE)
```

```
/**
 * konfiguriert den Gyro.
 * der Gyro ist nach dem Aufruf der Funktion aktiv. Wird er nicht mehr
 * benötigt, kann er mir gyroStop() wieder deaktiviert werden. Für eine
 * Reaktivierung kann die Funktion
 * gyroStart() verwendet werden.
 *
 * Im aktiven Zustand muss im Abstand von höchstens 120ms die Funktion
 * gyroUpdate() aufgerufen werden.
 *
 * Die globalen Variablen winkelx, winkely und winkelz enthalten den
 * jeweils aktuellen winkel. Die Funktion gyroResetWinkel setzt die
 * globalen Variablen wieder auf 0.
 */
void gyroInit()
{
    /* Initialize SPI */
    // gyroSpiInit(); --> called aus dem Anwendungsmodul,
    // da an SPI1 noch weitere Geräte betrieben werden

    // Initialize GyroChip
    // 190Hz data Rate, 50Hz Bandwidth
    gyroWriteRegister(GYRO_REG_CTRL_REG1, GYRO_OP_PARAMS);

    // CTRL_REG2 --> no High Pass --> nothing to change

    // enable Watermark interrupt on INT2 --> not working no int pulse
    // gyroWriteRegister(GYRO_REG_CTRL_REG3, GYRO_I2_EN|GYRO_I2_WTM);

    // set resolution to 500 dps
    gyroWriteRegister(GYRO_REG_CTRL_REG4, GYRO_FULL_SCALE_500);

    // enable FIFO, Output after second LP-Filter, no HP
    gyroWriteRegister(GYRO_REG_CTRL_REG5,
        GYRO_FIFO_ENABLE|GYRO_OUT_SELECT2);
    //gyroWriteRegister(GYRO_REG_CTRL_REG5, GYRO_OUT_SELECT2);

    //Fifomode and watermark
    gyroWriteRegister(GYRO_REG_FIFO_CTRL,
        GYRO_FIFO_MODE_STREAM|GYRO_FIFO_WATERMARK);
}

/**
```

```
 * Es werden 1000 Werte eingelesen (ca. 5,5s) und aus den Werten die
 * Drift-Kompensationswerte gemittelt. Der Gyro darf während der Zeit
 * nicht bewegt werden.
 */
void gyroCalibrate() {
    int count = 0;
    int sumx=0,sumy=0,sumz=0;

    #ifdef DEBUG_GYRO
        SerialUSB.print("Biaswerte ermitteln ");
    #endif
    while (count < 100) // war anfangs auf < 1000
    {
        delay(100);
        uint8 anz = gyroReadRegister(GYRO_REG_FIFO_SRC) & 0x1F;
        for (int i=0; i<anz; i++)
        {
            readGyroValues();
            int16 *p = (int16*)gyroValuesSingle;
            sumx += *p++;
            sumy += *p++;
            sumz += *p;
        }
        #ifdef DEBUG_GYRO
        SerialUSB.print("*");
        #endif
        count += anz;
    }
    GYRO_BIAS_X = sumx/count;
    GYRO_BIAS_Y = sumy/count;
    GYRO_BIAS_Z = sumz/count;

    #ifdef DEBUG_GYRO
    SerialUSB.println("");
    SerialUSB.print("BIAS ");
    SerialUSB.print(GYRO_BIAS_X); SerialUSB.print(" ");
    SerialUSB.print(GYRO_BIAS_Y); SerialUSB.print(" ");
    SerialUSB.println(GYRO_BIAS_Z);
    #endif
}

void gyroSetBiasX(int bias) { GYRO_BIAS_X = bias; }
void gyroSetBiasY(int bias) { GYRO_BIAS_Y = bias; }
void gyroSetBiasZ(int bias) { GYRO_BIAS_Z = bias; }
```

```
/**
 * Die Winkel-Variablen werden auf 0 zurückgesetzt
 */
void gyroResetWinkel()
{
    winkelx = 0;
    winkely = 0;
    winkelz = 0;
}

/**
 * Diese Funktion muss innehalb von 120ms mindestens einmal aufgerufen
 * werden. Es werden die vorhandenen Messwerte aus dem Fifo-Speicher des
 * Chips ausgelesen und die aktuellen Winkel berechnet
 */
void gyroUpdate()
{
    uint8 gyroFifoStatus = gyroReadRegister(GYRO_REG_FIFO_SRC);
    uint8 anz = gyroFifoStatus & 0x1F;

    #ifdef DEBUG_GYRO
    SerialUSB.print("fifo anz = ");
    SerialUSB.print(anz);
    SerialUSB.print(" ");
    #endif

    int16* p;
    int sumx=0,sumy=0,sumz=0;

    for (int i=0; i<anz; i++)
    {
        readGyroValues();
        p = (int16*)gyroValuesSingle;
        gyroxRaw = *p++ - GYRO_BIAS_X;
        gyroyRaw = *p++ - GYRO_BIAS_Y;
        gyrozRaw = *p++ - GYRO_BIAS_Z;
        sumx += gyroxRaw;
        sumy += gyroyRaw;
        sumz += gyrozRaw;

    }
/* Winkelgeschwindigkeiten!! in Grad/Messintervall */
    winkelx = (sumx*500.0F)/(160.0F*32767.0F);
```

```
    winkely = (sumy*500.0F)/(160.0F*32767.0F);
    winkelz = (sumz*500.0F)/(160.0F*32767.0F);

    #ifdef DEBUG_GYRO
    SerialUSB.print(waktX); SerialUSB.print("\t");
    SerialUSB.print(waktY); SerialUSB.print("\t");
    SerialUSB.print(waktZ); SerialUSB.print("\t");
    SerialUSB.println("");
    #endif
}

int gyroGetTemp( void ) {
    int temp;
    temp = (int) gyroReadRegister(0x26); // Register 26 Hex lesen
    return temp;
}

/**
 * Versetzt den GyroChip in den aktiven Zustand, leert den Fifo und
 * setzt die Winkel zurueck
 */
void gyroStart()
{
    gyroWriteRegister(GYRO_REG_CTRL_REG1, GYRO_OP_PARAMS);
    // Empty Fifo
    gyroUpdate();
    gyroResetWinkel();
}

/**
 * Versetzt den gyro in den Power Down mode
 */
void gyroStop()
{
    gyroWriteRegister(GYRO_REG_CTRL_REG1, 0);
}
```

lsm303.cpp

```
/*****************************************************************************
 * The MIT License
 *
 * Copyright (c) 2010 LeafLabs LLC.
```

```
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies
 * of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
 * BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 ******************************************************************************/

/**
 * @brief functions for accessing l3g20 gyro on stm32f3discovery board
 */

#include <wirish/wirish.h>
#include <libmaple/i2c.h>
#include "lsm303Reg.h"
#include "../RingBuffer/ringbuffer.h"
#include "lsm303.h"

#define ACC_FIFO_WATERMARK 10

static int ACC_BIAS_X;
static int ACC_BIAS_Y;
static int ACC_BIAS_Z;


int accx, accy, accz;

#define RB_SIZE 64
#define ACC_AVERAGE_X 60
#define ACC_AVERAGE_Y 60
```

```
#define ACC_AVERAGE_Z 60


static uint8 write_msg_data[3];
static i2c_msg write_msg;

static uint8 read_msg_data[7];
static i2c_msg read_msg;


void readAccValues()
{
    int16 *p;
    write_msg_data[0] = LSM303_REG_OUT_X_L_A|LSM303_READ_MULTI_BYTES;
                                // 0x80 -> AutoIncrement address on read
    write_msg.length = 1;
    i2c_master_xfer(I2C1, &write_msg, 1, 0);

    read_msg.length = 6;
    i2c_master_xfer(I2C1, &read_msg, 1, 2);

    p = (int16*)read_msg_data;
    accx = *p++ - ACC_BIAS_X;
    accy = *p++ - ACC_BIAS_Y;
    accz = *p++ - ACC_BIAS_Z;
}

uint8 accReadRegister(uint8 reg)
{
    write_msg_data[0] = reg;
    write_msg.length = 1;
    i2c_master_xfer(I2C1, &write_msg, 1, 0);

    read_msg.length = 1;
    i2c_master_xfer(I2C1, &read_msg, 1, 2);

    return read_msg_data[0];
}

void accWriteRegister(uint8 reg, uint8 val)
{
    write_msg_data[0] = reg;
    write_msg_data[1] = val;
    write_msg.length = 2;
```

```
    i2c_master_xfer(I2C1, &write_msg, 1, 0);
}


void lsm303I2CInit()
{
    //i2c_master_enable(I2C1, 0);
    // Aktivierung des I2C-Busses muss im Hauptprogramm erfolgen,
    // da mehrere Komponenten das I2C-Interface benutzen können
    // und eine mehrfache Initialisierung nicht funktioniert

    write_msg.addr = LSM303_ADDR_A;
    write_msg.flags = 0; // write, 7 bit address
    write_msg.length = sizeof(write_msg_data);
    write_msg.xferred = 0;
    write_msg.data = write_msg_data;

    read_msg.addr = LSM303_ADDR_A;
    read_msg.flags = I2C_MSG_READ;
    read_msg.length = sizeof(read_msg_data);
    read_msg.xferred = 0;
    read_msg.data = read_msg_data;

}

/**
 * Initialisiert und aktiviert den Beschleunigungssensor
 * Aktivierung des I2C-Busses muss im Hauptprogramm erfolgen,
 * da mehrere Komponenten das I2C-Interface benutzen können
 * und eine mehrfache Initialisierung nicht funktioniert
 */
void lsm303Init()
{
    /* Initialize I2C Transfer-Strukturen für den lsm303 */
    lsm303I2CInit();

    // Initialize ACC-Chip
    // 50Hz data Rate
    accWriteRegister(LSM303_REG_CTRL_REG1_A,
        LSM303_A_ODR_50HZ|LSM303_A_ALL_AXIS_EN);

    // CTRL_REG2 --> no High Pass --> nothing to change

    // enable Watermark interrupt on INT2 --> not working no int pulse
```

```
    // gyroWriteRegister(GYRO_REG_CTRL_REG3, GYRO_I2_EN|GYRO_I2_WTM);

    // sensitivity 2G is default

    // enable FIFO ?

    //Fifomode and watermark
}

void accCalibrate() {
    int count = 0;
    int sumx=0,sumy=0,sumz=0;

    #ifdef DEBUG_ACC
        SerialUSB.print("Biaswerte ermitteln ");
    #endif
    while (count < 100)
    {
        delay(1);
            readAccValues();
            sumx += accx;
        sumy += accy;
        sumz += accz;
            #ifdef DEBUG_ACC
        SerialUSB.print("*");
        #endif
        count += 1;
    }
    ACC_BIAS_X = sumx/count;
    ACC_BIAS_Y = sumy/count;
    ACC_BIAS_Z = sumz/count;

    #ifdef DEBUG_ACC
    SerialUSB.println("");
    SerialUSB.print("ACC-BIAS ");
    SerialUSB.print(ACC_BIAS_X); SerialUSB.print(" ");
    SerialUSB.print(ACC_BIAS_Y); SerialUSB.print(" ");
    SerialUSB.println(ACC_BIAS_Z);
    #endif
}

void accUpdate()
{
    readAccValues();
```

```
    #ifdef DEBUG_ACC
    SerialUSB.print(accx); SerialUSB.print("\t");
    SerialUSB.print(accy); SerialUSB.print("\t");
    SerialUSB.print(accz); SerialUSB.print("\t");
    SerialUSB.println("");
    #endif
}
```