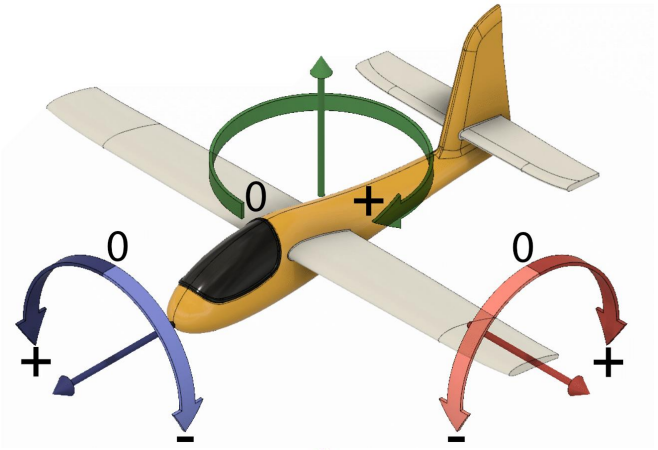
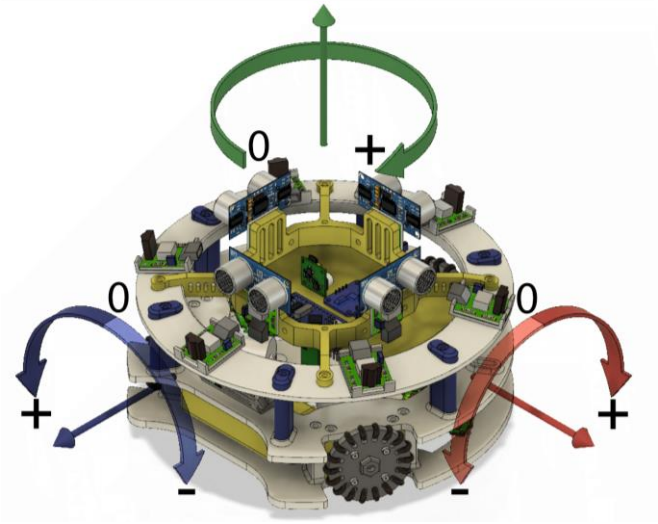
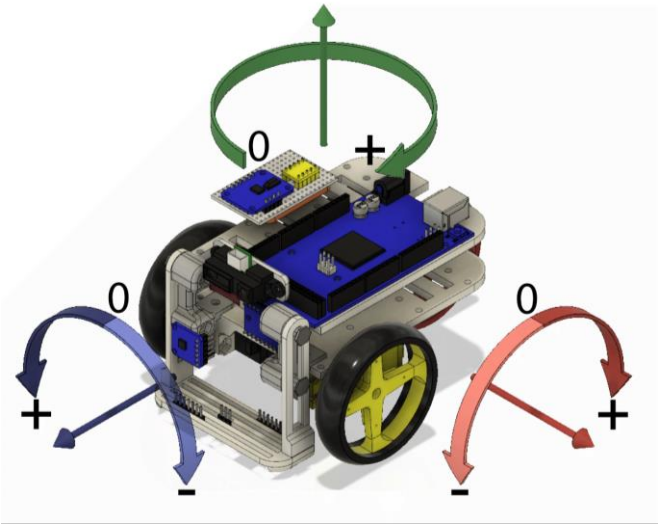
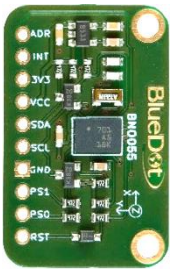
Robotik-AG-Tutorial zu

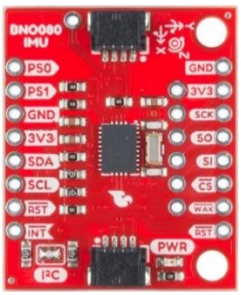
6- und 9-DOF IMU

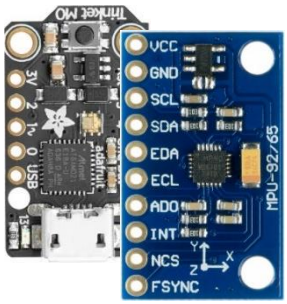


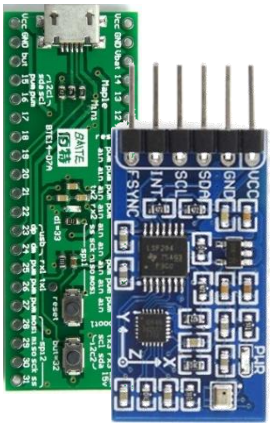












Version 0.7, Stand 22. Mai 2019

Autor: Thomas Jörg

Inhaltsverzeichnis

6- und 9-DOF IMU ... 1
Einleitung und Überblick ... 3
Anwendungsfälle ... 4
Soccer: ... 4
Rescue Line: ... 4
Herleitung Eulerwinkel ... 5
Gute Online-Simulationen dazu: ... 5
Verwendete & getestete IMU ... 5
Weitere Voraussetzungen ... 6
1. BNO055, Hersteller: Bosch Sensortec. ... 6
Bauformen der Breakouts: ... 6
Library: ... 6
Kalibrierungsprozedur: ... 6
Kalibrierungs- und Eulercode BNO055: ... 7
2. BNO080, Hersteller: Bosch Sensortec. ... 8
Dokumentation: ... 8
Bauformen der Breakouts: ... 8
Library: ... 8
Kalibrierungsprozedur: ... 8
Eulercode: ... 8
3. MPU9250 + M0 Cortex SAMD21 (Adafruit Trinket) ... 10
Library ... 10
Beschreibung der Library und Hardware ... 10
Hardware-Setup ... 11
Kalibrierungsprozedur: ... 11
Eulercode: ... 11
4. BMI088 + Maple Mini + SensorFusion-Bibliothek (Mahony-Algorithmus) ... 12
Verwendete Libraries ... 12
Dokumentation: ... 12
Kalibrierungs-Prozedur: ... 12
Eulercode: ... 13
5. ICM 20948 + AHRS SensorFusion ... 14
Verwendetes Board und weitere Bauformen: ... 14
Verwendete Libraries: ... 14
Kalibrierungsprozedur: ... 14
Eulercode mit Magnetisierungs-Kalbrierung: ... 16

Einleitung und Überblick

IMU ist die Abkürzung für „Inertial Measuring Unit“ und stellt eine Kombination mehrerer Inertialsensoren dar:

- **3-Achsen Beschleunigungs-Sensor (Accelerometer),**
  um Erd- und geradlinige Beschleunigung zu messen. Es werden drei unabhängige Werte in x, y, z gemessen
  Physikalische Größe: Beschleunigung in drei Raumrichtungen

- **3-Achsen Drehraten-Sensor (Gyroskop),**
  um die Drehbewegung um eine Achse zu messen. Es werden drei unabhängige Werte in x, y, z gemessen.
  Physikalische Größe: Winkelgeschwindigkeit in drei Raumrichtungen

- **3-Achsen Kompass-Sensor (Magnetometer),**
  um das Erdmagnetfeld zu messen. Es werden drei unabhängige Werte in x, y, z gemessen
  Physikalische Größe: Magnetische Flussdichten in drei Raumrichtungen.

Jeder dieser drei Sensoren liefert Messwerte in den orthogonalen Achsen x, y, z; in der Summe 9 Messwerte:

ax, ay, az, gx, gy, gz, mx, my, mz

‚**a**‘ *steht für ‚Beschleunigung,* ‚**g**‘ *für ‚gyro‘ – also ‚Drehgeschwindigkeit‘,* ‚**m**‘ *für ‚Magnetisierung‘.*

Jeder dieser Messwerte wird als sogenannter „Freiheitsgrad“ bezeichnet, bzw. als „**Degree of freedom**“, „DOF“. Daher spricht man üblicherweise von „**9DOF**“- oder „**6DOF**“- Sensoren.

Diese Messwerte werden meistens nicht isoliert genutzt, sondern werden miteinander verrechnet. Durch geeignete Berechnungen kann man unter anderem die ‚**Absolute Orientierung**‘ oder den aktuellen Ort des Sensors bestimmen. Da der Sensor an seinem Träger befestigt ist, bestimmt man damit also indirekt Ort und Ausrichtung des Trägers.

Den mathematischen Prozess dieser Verrechnung von vielen unterschiedlichen Sensorwerten zu neuen, nicht direkt gemessenen Werten bezeichnet man als sogenannte **Sensor-Fusion**. Dazu ist noch eine möglichst exakte Zeitmessung nötig, denn:

Um aus den Beschleunigungswerten des Accelerometers einen Ort zu berechnen, muss zweifach integriert werden:

- Die Integration der Beschleunigung nach der Zeit liefert die Momentangeschwindigkeit,
- Die Integration der Momentangeschwindigkeit nach der Zeit liefert den momentanen Ort.

Um aus den Winkelgeschwindigkeiten des Gyros den Drehwinkel zu bestimmen, muss einfach integriert werden :

- Die Integration der Winkelgeschwindigkeit nach der Zeit liefert den aktuellen Drehwinkel

Der Abgleich der so berechneten Werte mit dem gemessenen Erdmagnetfeld-Werten des Magnetometers verbessert die Genauigkeit, bzw. korrigiert die sogenannte **Drift**. Als **Drift** bezeichnet man die langsamen aber stetigen Wanderungen von Messwerten der Sensoren durch unterschiedlichen physikalische Einflüsse wie z.B. Temperaturrauschen, Fertigungs-Ungenauigkeiten, mangelnde Kalibrierung etc.

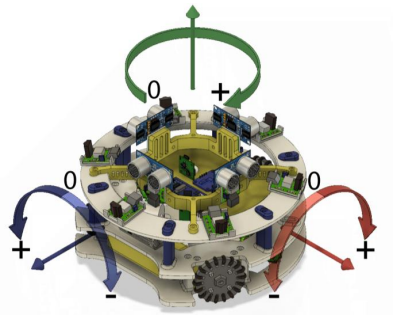
Bedeutung des Faktors ‚Zeit‘: Es wird nach der Zeit integriert; das bedeutet, dass die Genauigkeit der Sensorwerte empfindlich von der Genauigkeit und der Häufigkeit der gemessenen Zeitintervalle abhängt. Es gilt:
Je häufiger die Integration stattfindet und je genauer die Zeitmessungen sind, desto geringer der Messfehler.
Es ist daher nötig, für die Sensorfusion einen schnellen, präzisen Mikrocontroller zu verwenden.

In der Summe besteht eine voll ausgestattete IMU also aus **fünf voneinander unabhängigen Komponenten**:

Accelerometer, Gyroskop, Magnetometer, schneller Zeitmesser, Sensor-Fusion

Anwendungsfälle

Soccer:



Gierung/Yaw und initiales Heading:
Es wird eine möglichst genaue Ausrichtung des Roboters zum Tor hin angestrebt.

Diese Ausrichtung des Roboters auf dem Spielfeld wird als Englisch „**Yaw**“, bzw. deutsch „**Gierwinkel**“ bezeichnet. Der **optimale Gierwinkel** (Englisch „**Heading**“) wird zu Beginn der Halbzeit festgesetzt und bleibt während der Halbzeit erhalten. Dies muss im längsten Fall etwa 10-15 Minuten einer Halbzeit halten, bis neu kalibriert werden kann. **Kritisch ist daher die zeitliche Dauer des Headings!**

Die äußeren Dimensionen des Spielfelds liegen bei 243cm Länge gesamt zu 182cm Breite gesamt.

Die Breite des Tors liegt bei 60cm, die Länge der Stecke von Tor zu Tor bei 183cm. Der maximale Fehlerwinkel kann abgeschätzt werden zu:

$$\alpha_{max} = tan^{-1}\frac{30\ cm}{183\ cm} = 9{,}3°$$

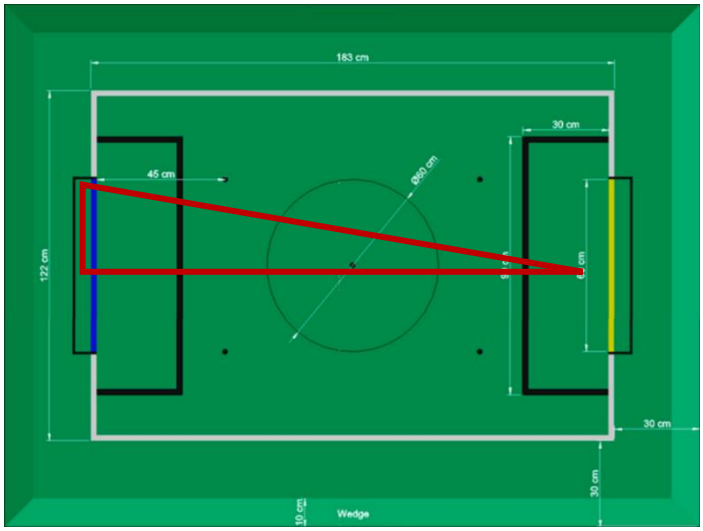
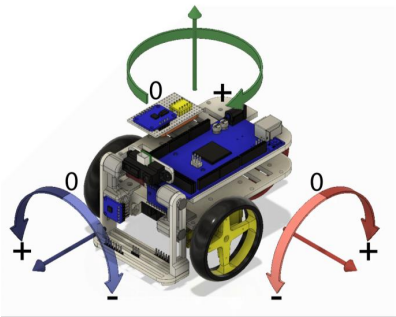


Abbildung 1, Abmessungen Soccer 2:2 Open und LWL

Rescue Line:



Gierung / Yaw:
an einer Schikane In der Disziplin ‚Rescue Line‘ muss der Roboter unter anderem um ein Hindernis fahren und danach exakt auf die Linie zurückfinden. Dazu muss der Roboter mehrere Male möglichst exakt jeweils um 90° gedreht werden.

An Kreuzungen muss sich der Roboter möglichst exakt um 90° oder 180° drehen.

Pitch / Roll:
Fährt der Roboter einen Abhang hinauf, so muss die Motorleistung für den Abhang angepasst werden. Ein positiver Pitch-Winkel zeigt die aktuelle Hangneigung an. Fährt er den Abhang hinab, so muss die Motorleistung ebenfalls angepasst werden; dies wird angezeigt durch einen negativen Pitch.

Bei Abhängen besteht die Gefahr, dass der Roboter rutscht und sich unter Umständen schiefstellt. Dies wird detektiert durch einen sich verändernden Roll, bzw. auch durch ein sich änderndes Yaw. Es sei dabei angenommen, dass Abhänge immer geradlinig verlaufen und keine Kurven enthalten.

Die Anforderung der zeitlichen Stabilität der absoluten Orientierung liegt bei weniger als einer Minute.

Abbildung 2, Schwierig zu fahren: Abhänge mit Bumpern und solche mit Linienunterbrechung

Herleitung Eulerwinkel

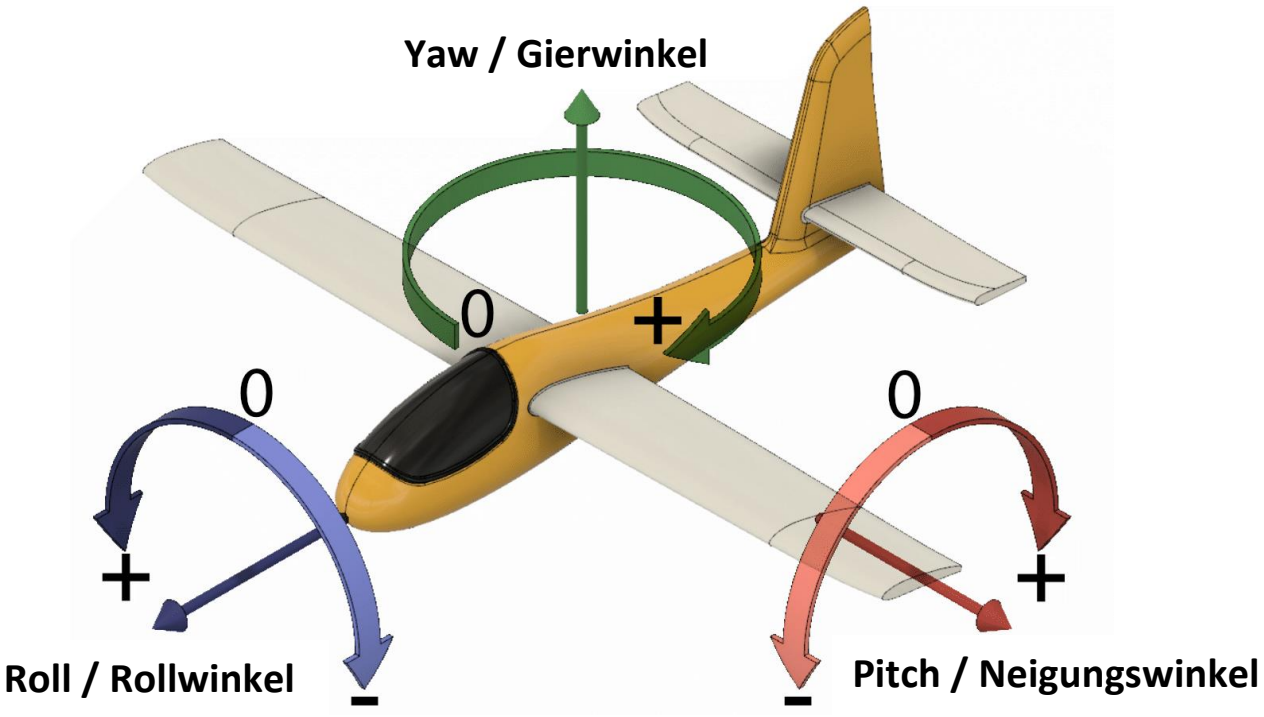


Abbildung 3: Darstellung der drei Eulerwinkel

Die mathematisch korrekte Definition sieht zwar etwas anders aus, allerdings wird oftmals so gerechnet:

- Der Gierwinkel / *eng. Yaw* startet bei 0° vorne und verläuft im Uhrzeigersinn bis 359°
- Der Rollwinkel / *eng. Roll* startet bei 0° senkrechter Ausrichtung und verläuft bis -179° gegen den Uhrzeigersinn, bzw. +179° mit dem Uhrzeigersinn
- Der Neigungswinkel / *eng. Pitch* startet bei 0° horizontaler Ausrichtung und verläuft bis +90 Grad bei Bergaufneigung und bis -90° bei Bergabneigung

Gute Online-Simulationen dazu:

http://www.ctralie.com/Teaching/COMPSCI290/Materials/EulerAnglesViz/

http://danceswithcode.net/engineeringnotes/rotations_in_3d/demo3D/rotations_in_3d_tool.html

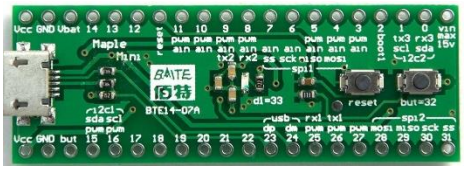
Verwendete & getestete IMU

Es wurden generell nur solche IMU verwendet, mit denen einen Sensorfusion durchgeführt werden kann. Dazu zählen Sensoren, die

- entweder einen **Koprozessor** für die Sensorfusion und Filterung bereits onboard besitzen
- oder solche, bei denen ein **externer Koprozessor mit spezifischer Firmware** dazugelinkt werden kann.

Weitere Voraussetzungen

- Leicht implementierbare Arduino-Bibliothek
- Ausgabe von Eulerwinkeln
- **Dokumentierte Kalibrierungsprozedur**
- Robustes Heading mit geringer Drift
- Gute Kommunikationsmöglichkeit mit dem Hauptprozessor



Alle Experimente wurden mit der Standard-Wire-Library der Arduino-Umgebung ausgeführt. Als Hauptprozessor wurde ein Maple Mini von Baite verwendet.

Abbildung 4: Maple Board von Baite

1. BNO055, Hersteller: Bosch Sensortec.

- Onboard Koprozessor: M0 Cortex.
- Kommunikation mit dem Hauptprozessor: **I2C**

Bauformen der Breakouts:

Abbildung 5: Bauformen BNO055

Library:

https://github.com/BoschSensortec/BNO055_driver

Kalibrierungsprozedur:

https://www.youtube.com/watch?v=Bw0WuAyGsnY

https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf
(Seite 47 von 105)

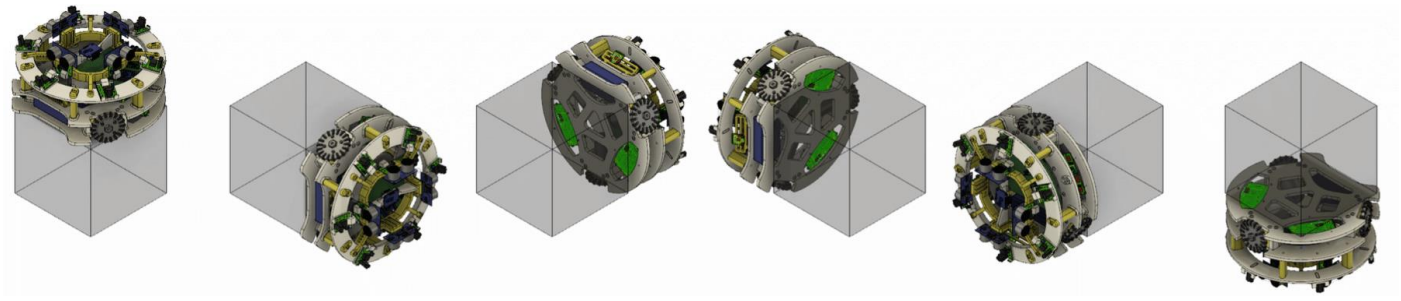


Abbildung 6: Kalibrierungsprozedur

Kalibrierungs- und Eulercode BNO055:

```
#include "BNO055_support.h"

struct bno055_t myBNO;
struct bno055_euler myEulerData;
unsigned char accelCalibStatus = 0;
unsigned char magCalibStatus = 0;
unsigned char gyroCalibStatus = 0;
unsigned char sysCalibStatus = 0;
unsigned long lastTime = 0;
unsigned char bnoState = 0; //0: unkalibriert, 1: vollständig kalibriert

void setup()
{
  Wire.begin();
  BNO_Init(&myBNO);
  bno055_set_operation_mode(OPERATION_MODE_NDOF);
  delay(1);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  delay(100);
  Serial.begin (115200);
}

void loop()
{
  if (bnoState == 0) {
    if ((millis() - lastTime) >= 200) //To read calibration status at 5Hz without using
additional timers
    {
      lastTime = millis();

      bno055_get_accelcalib_status(&accelCalibStatus);
      bno055_get_magcalib_status(&magCalibStatus);
      bno055_get_magcalib_status(&gyroCalibStatus);
      bno055_get_syscalib_status(&sysCalibStatus);
      Serial.println("Calibration:");
      Serial.println("Acc Mag Gyr Sys");
      Serial.print(accelCalibStatus); Serial.print(magCalibStatus);
      Serial.print(gyroCalibStatus); Serial.println(sysCalibStatus);

      if ((accelCalibStatus + magCalibStatus + gyroCalibStatus + sysCalibStatus) > 11){
        bnoState = 1;
      }
    }
  }
  if (bnoState == 1) {
    if ((millis() - lastTime) >= 100)
    {
      lastTime = millis();
      bno055_read_euler_hrp(&myEulerData);
      Serial.println("Yaw  Roll  Pitch:");
      Serial.print(int(myEulerData.h / 16.00)); Serial.print(" ");
      Serial.print(int(myEulerData.r / 16.00)); Serial.print(" ");
      Serial.println(int(myEulerData.p / 16.00)); Serial.print(" ");
    }
  }
}
```

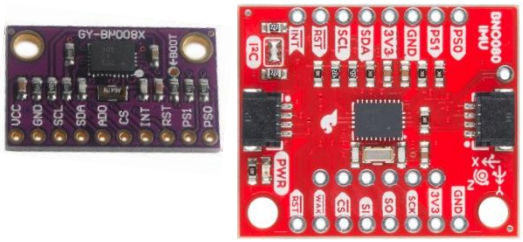
2. BNO080, Hersteller: Bosch Sensortec.

- Onboard Koprozessor: M0 Cortex.
- Kommunikation mit dem Hauptprozessor: **I2C**

Dokumentation:

https://learn.sparkfun.com/tutorials/qwiic-vr-imu-bno080-hookup-guide/all

Bauformen der Breakouts:



Library:

https://github.com/sparkfun/SparkFun_BNO080_Arduino_Library/archive/master.zip

Kalibrierungsprozedur:

https://www.hillcrestlabs.com/downloads/bno080-sensor-calibration-procedure#

https://cdn.sparkfun.com/assets/9/e/1/d/9/Sensor-Calibration-Procedure-v1.1.pdf

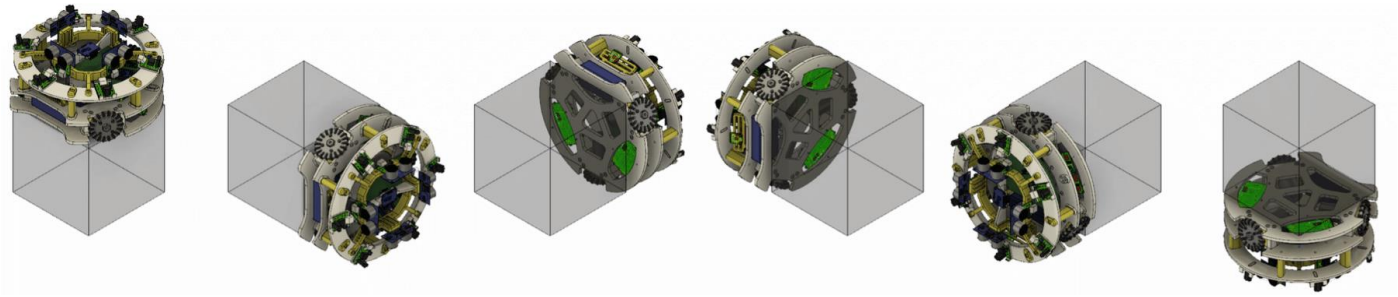


Abbildung 7: Kalibrierungsprozedur für den BNO080

Eulercode:

```
#include <math.h>
#include <Wire.h>
#include "SparkFun_BNO080_Arduino_Library.h"

BNO080 myIMU;

struct Euler {
  float yaw;
  float pitch;
  float roll;
};
struct Quat {
  float i;
  float j;
  float k;
  float real;
};
```

```
Euler getAngles(Quat q, bool degrees);
Quat myQuat;
Euler eul;

void setup()
{
  Serial.begin(115200);
  Wire.begin();
  Wire.setClock(400000); //Increase I2C data rate to 400kHz

  myIMU.begin();
  myIMU.enableRotationVector(10); //Send data update every 50ms

  Serial.println(F("Rotation vector enabled"));
  //Serial.println(F("Output in form i, j, k, real, accuracy"));
  Serial.println(F("Output in form time(ms), yaw, pitch, roll, accuracy"));
}

void loop()
{
  if (myIMU.dataAvailable() == true)
  {
    float quatI = myIMU.getQuatI();
    float quatJ = myIMU.getQuatJ();
    float quatK = myIMU.getQuatK();
    float quatReal = myIMU.getQuatReal();
    float quatRadianAccuracy = myIMU.getQuatRadianAccuracy();
    * /
    myQuat.i = quatI;
    myQuat.j = quatJ;
    myQuat.k = quatK;
    myQuat.real = quatReal;

    eul = getAngles(myQuat);
    Serial.print(eul.yaw, 2);
    Serial.print(eul.pitch, 2);
    Serial.print(eul.roll, 2);
    Serial.print(quatRadianAccuracy, 2);
    Serial.println();
  }
}
Euler getAngles(Quat q) {
  Euler ret_val;
  float x; float y;

  x = 2 * ((q.i * q.j) + (q.real * q.k));
  y = square(q.real) - square(q.k) - square(q.j) + square(q.i);
  ret_val.yaw = degrees(atan2(y, x));
  ret_val.pitch = degrees(asin(-2 * (q.i * q.k - q.j * q.real)));

  x = 2 * ((q.j * q.k) + (q.i * q.real));
  y = square(q.real) + square(q.k) - square(q.j) - square(q.i);
  ret_val.roll = degrees(atan2(y , x));

  return ret_val;
}

float square(float var) {
  return var * var;
}
```

3. MPU9250 + M0 Cortex SAMD21 (Adafruit Trinket)

- Externer Koprozessor: M0 Cortex SAMD21.
- Kommunikation mit dem Hauptprozessor: **Seriell**

Der 9DOF-Sensor MPU9250 liefert sehr präzise Daten, allerdings zunächst ohne Sensorfusion. Diese kann durch einen Board-eigenen Prozessor, den DMP, ausgeführt werden. Der MPU9250 ist eine Kombination aus dem Magnetometer AK8963 und dem 6DOF-Sensor MPU6500.

Library

Um die Sensorfusion mittels Koprozessor zu rechnen, existieren einige Hard- und Software-Varianten. Die Lösung von Sparkfun ist das sogenannte ‚Razor-Board', auf dem ein SAMD21-Prozessor mit verbaut ist. Die Firmware dieses Coprozessors ist Open Source und kann hier heruntergeladen werden:

https://github.com/sparkfun/SparkFun_MPU-9250-DMP_Arduino_Library

Beschreibung der Library und Hardware

Diese Firmware berechnet aus den MPU-eigenen DMP-Daten die gewünschten Eulerwinkel. Der DMP ist ein ‚Digital Motion Processor', der die Sensordaten aufbereitet, also die Sensorfusion durchführt. Allerdings ist der Zugriff auf den DMP aufwendig und rechenintensiv, weshalb für den DMP-Treiber ein schneller 32-Bit-Prozessor wie der SAMD21 (oder auch ein SAMD51) benötigt wird:

Der DMP-Prozessor des MPU9250 besitzt einen FIFO-Buffer für neue Daten. Das heißt: jedes Mal, wenn der Sensor einen neuen Datensatz generiert, wird er in einen Speicher geschrieben, der 512 Byte groß ist. Liest man Daten aus, so werden die gelesenen Daten gelöscht. Liest man zu langsam aus, so sind die Daten veraltet und neu gelesene Daten stimmen nicht mit dem aktuellen Orientierungs-Zustand des Sensors überein!

Es existieren zwei Methoden, diesem Problem zu begegnen:

a) Man nutzt einen speziellen Hardware-Interrupt des MPU9250-Sensors in Form eines Pins. Dieser Pin wird jedes Mal auf ‚High' geschaltet, wenn neue Daten vorhanden sind. Nachteil: Man benötigt ein Kabel mehr und muss am Koprozessor/Hauptprozessor einen Interrupt dafür bereitstellen.

b) Man pollt ‚oft genug', das heißt, man schaut häufig nach, ob frische Datensätze vorhanden sind. In der Sparkfun-Bibliothek ist der Befehl ‚imu.dmpUpdateFifo()' dafür vorgesehen:

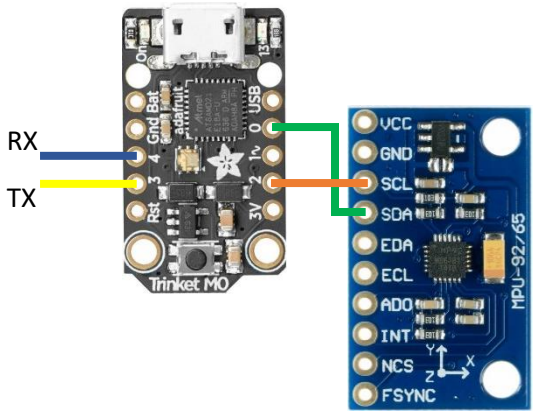
```
if ( imu.fifoAvailable() > 0 ) {
      if ( imu.dmpUpdateFifo() == INV_SUCCESS)    {
                  //lese die Daten aus
      }
}
```

Viele andere Bibliotheken wie z.B. die bekannten Varianten von Kris Winer verzichten daher auf den DMP-Zugriff. In der Konsequenz sind dann entweder die Werte recht unzuverlässig oder die Speicher- und Rechenintensität auf dem Hauptprozessor enorm.

Hardware-Setup

Getestet wurde die Kombination aus einem MPU9250 und einem Arduino Feather M0 / Arduino Trinket M0.

Wichtig: Auf anderen Prozessoren als den SAMD-Typen kompiliert die Sparkfun-Firmware nicht!



- Die Kommunikation zwischen MPU9250 und Koprozessor läuft über I2C. Die Kabelverbindung ist sehr kurz zu halten.
- Die Kommunikation zwischen Ko- und Hauptprozessor läuft über Serielle Schnittstelle (TX-RX).

Eine gemeinsame Nutzung eines I2C-Buses mit dem Hauptprozessor empfiehlt sich nicht, da der Koprozessor genauso wie der Hauptprozessor als I2C-Master agieren muss und auf dem Bus zwischen IMU und Koprozessor ein sehr intensiver Datenaustausch stattfindet.

Kalibrierungsprozedur:

Der Gyrosensor und das Magnetometer des MPU0250 müssen nach jedem Einschalten neu kalibriert werden.

https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide
(Stichwort: „Gyroscope calibration“: Das MPU-Board muss 8 Sekunden ruhig liegen)

https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration

Eulercode:

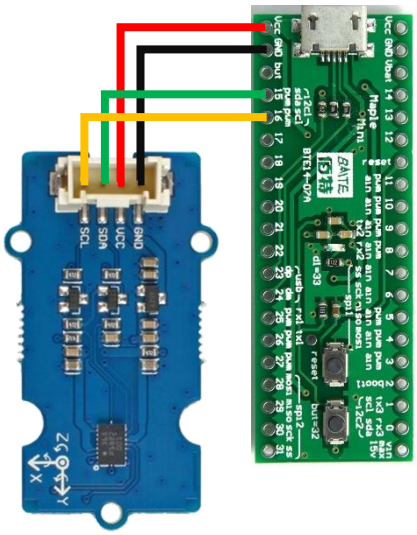
```
#include <SparkFunMPU9250-DMP.h>
MPU9250_DMP imu;

void setup()
{
  Serial.begin(115200);
  imu.begin();
  imu.setSensors(INV_XYZ_GYRO); // Enable gyroscope only
  imu.setGyroFSR(2000); // Set gyro to 2000 dps
  imu.dmpBegin(DMP_FEATURE_GYRO_CAL | DMP_FEATURE_SEND_CAL_GYRO, 10);
  Serial.println("Gyro ruhig stehen lassen und 8 Sekunden warten");
  delay(8000);

  imu.begin();
  imu.dmpBegin(DMP_FEATURE_6X_LP_QUAT | DMP_FEATURE_GYRO_CAL, 10);
}

void loop()
{
  if ( imu.fifoAvailable() )
  {
    if ( imu.dmpUpdateFifo() == INV_SUCCESS)
    {
      imu.computeEulerAngles();
      float q0 = imu.calcQuat(imu.qw);
      float q1 = imu.calcQuat(imu.qx);
      float q2 = imu.calcQuat(imu.qy);
      float q3 = imu.calcQuat(imu.qz);
      Serial.println(imu.yaw);
      Serial.println(imu.pitch);
      Serial.println(imu.roll);
    }
  }
}
```

4. BMI088 + Maple Mini + SensorFusion-Bibliothek (Mahony-Algorithmus)



- Externer Koprozessor: Maple Board (STM32)
- Kommunikation mit dem Hauptprozessor: **I2C**

Es handelt sich beim BMI088 um einen 6DOF-Sensor, auf dem ein Beschleunigungs-Sensor und ein Gyroskop eingebaut sind. Ein Magnetometer fehlt. Daher ist der Sensorfusions-Algorithmus nicht so aufwendig wie bei einem 9DOF und die Berechnungen können – eigentlich – auf den Hauptprozessor ausgelagert werden.

Allerdings ist hier die Häufigkeit der Berechnungen essentiell:
Je häufiger der „Mahony"-Sensorfusions-Algorithmus ausgeführt wird, desto genauer werden die berechneten Eulerwinkel.

Verwendete Libraries

Es werden zwei unterschiedliche Libraries verwendet, nämlich ein Hardware-Treiber, der vom Hersteller direkt angeboten wird UND eine OpenSource-Sensorfusions-Library, welche Fusion und Filterung übernimmt:

https://github.com/Seeed-Studio/Grove_6Axis_Accelerometer_And_Gyroscope_BMI088/archive/master.zip
(Library von SeeedStudio speziell für den BMI088)

https://github.com/aster94/SensorFusion
(Open Source Library für die Sensorfusion von 6DOF- und 9DOF-IMUs)

Dokumentation:

http://wiki.seeedstudio.com/Grove-6-Axis_Accelerometer&Gyroscope(BMI088)/#play-with-arduino

https://github.com/SeeedDocument/Grove-6-Axis_Accelerometer-Gyroscope-BMI088/raw/master/res/BMI088.pdf

Kalibrierungs-Prozedur:

Im Datenblatt „BMI088.pdf" findet sich auf Seite 2 die folgende Passage, **eine Kalibrierung ist nicht nötig**:

> An evaluation circuitry (ASIC) converts the output of the micro-electro-mechanical sensing structures (MEMS), which are developed, produced and tested in BOSCH facilities. The corresponding chip-sets are packed into one single LGA 3.0mm x 4.5mm x 0.95mm housing. For optimum system integration, BMI088 is fitted with digital interfaces (SPI or I2C), offering a wide VDDIO voltage range from 1.2V to 3.6V. To provide maximum performance and reliability, each device is tested and is ready-to-use calibrated.

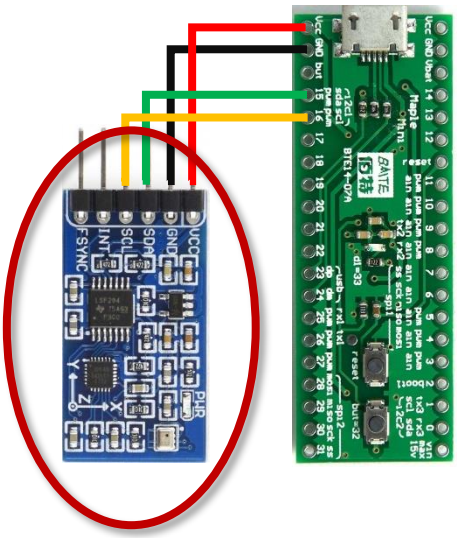
Anmerkungen:
Der BMI088 ist einfach und überraschend robust, die nicht notwendige Kalibrierung ist sehr komfortabel und reduziert falsche Bedienung. Die Werte zeigen ebenfalls eine geringe Drift, solange die Sensorfusion häufig genug ausgeführt werden. Optimales Berechnungsintervall: 1mal pro Millisekunde.

Eulercode:

```
void setup(void)
{
  Wire.begin();
  Serial.begin(115200);
  Serial.println("BMI088 Raw Data");
  while (1)
  {
    if (bmi088.isConnection()){
      bmi088.initialize();
      Serial.println("BMI088 is connected");
      break;
    }
    else Serial.println("BMI088 is not connected");
  }
}

void loop(void)
{
  bmi088.getAcceleration(&ax, &ay, &az);
  bmi088.getGyroscope(&gx, &gy, &gz);
  deltat = fusion.deltatUpdate();
  fusion.MahonyUpdate((3.1416/180)*gx,(3.1416/180)*gy,(3.1416/180)*gz,ax,ay,az,deltat);
  pitch = fusion.getPitch();
  roll = fusion.getRoll();
  yaw = fusion.getYaw();
  Serial.println(yaw);
  delay(1);
}
```

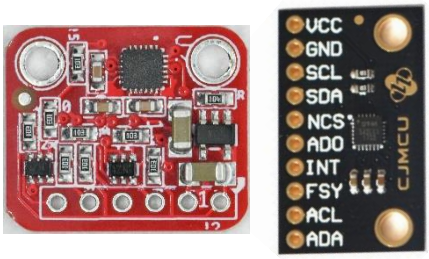
5. ICM 20948 + AHRS SensorFusion



Der ICM 20948 ist der Nachfolger des MPU9250. Zur Info: Die außergewöhnliche Namensbezeichnung rührt her vom Verkauf der Firma InvenSense an TDK. Der Sensor ist zum gegenwärtigen Zeitpunkt (22.Mai 2019) noch neu auf dem Markt, weshalb es zwar bereits einige Bibliotheken gibt, die aber nahezu allesamt noch sehr fehlerhaft sind. Insbesondere die fehlenden Kalibrierungsprozeduren bei allen bis auf der hier verwendeten Bibliotheken stellen eine starke Einschränkung dar.

Verwendetes Board und weitere Bauformen:

Waveshare 10DOF ICM 20948, **I2C-Adresse 0x68**



Verwendete Libraries:

https://github.com/drcpattison/DPEng_ICM20948_AK09916

https://github.com/adafruit/Adafruit_Sensor

Anmerkungen zur ICM20948-Bibliotkek:

Der üblicherweise verwendete I2C-Port des Sensors ist die 0x69. Um den Waveshare dennoch nutzen zu können, muss in der Datei „DPEng_ICM20948_AK09916.h“ in Zeile 39 der Eintrag abgeändert werden zu:

```
#define ICM20948_ACCELGYRO_ADDRESS (0x68)
```

Kalibrierungsprozedur:

In der Bibliothek von drcpattison finden sich zwei Dateien: **„ahrs_calibration_usb.ino**“ und „**ahrs_fusion.ino**“.

Mit der ersten Datei **„ahrs_calibration_usb.ino**“ bestimmt man die Kalibrierungswerte des Magnetometers. Die Kalibrierung wird ausgeführt mit der Software ‚MotionCal.exe“ die man hier herunterladen kann:

https://www.pjrc.com/store/prop_shield.html

Hat man den Sketch per Arduino-DIE auf den Mikrocontroller hochgeladen, öffnet man die App und wählt den Port aus, um die Magnetometer-Daten in Echtzeit zu visualisieren. Dabei sollte man so viele Punkte wie möglich erzeugen, die möglichst vollständig eine Kugeloberfläche bilden. Dies erreicht man, indem man den Sensor in möglichst alle Richtungen bewegt. Die App berechnet dazu drei unterschiedliche Magnetometer-Datensätze:

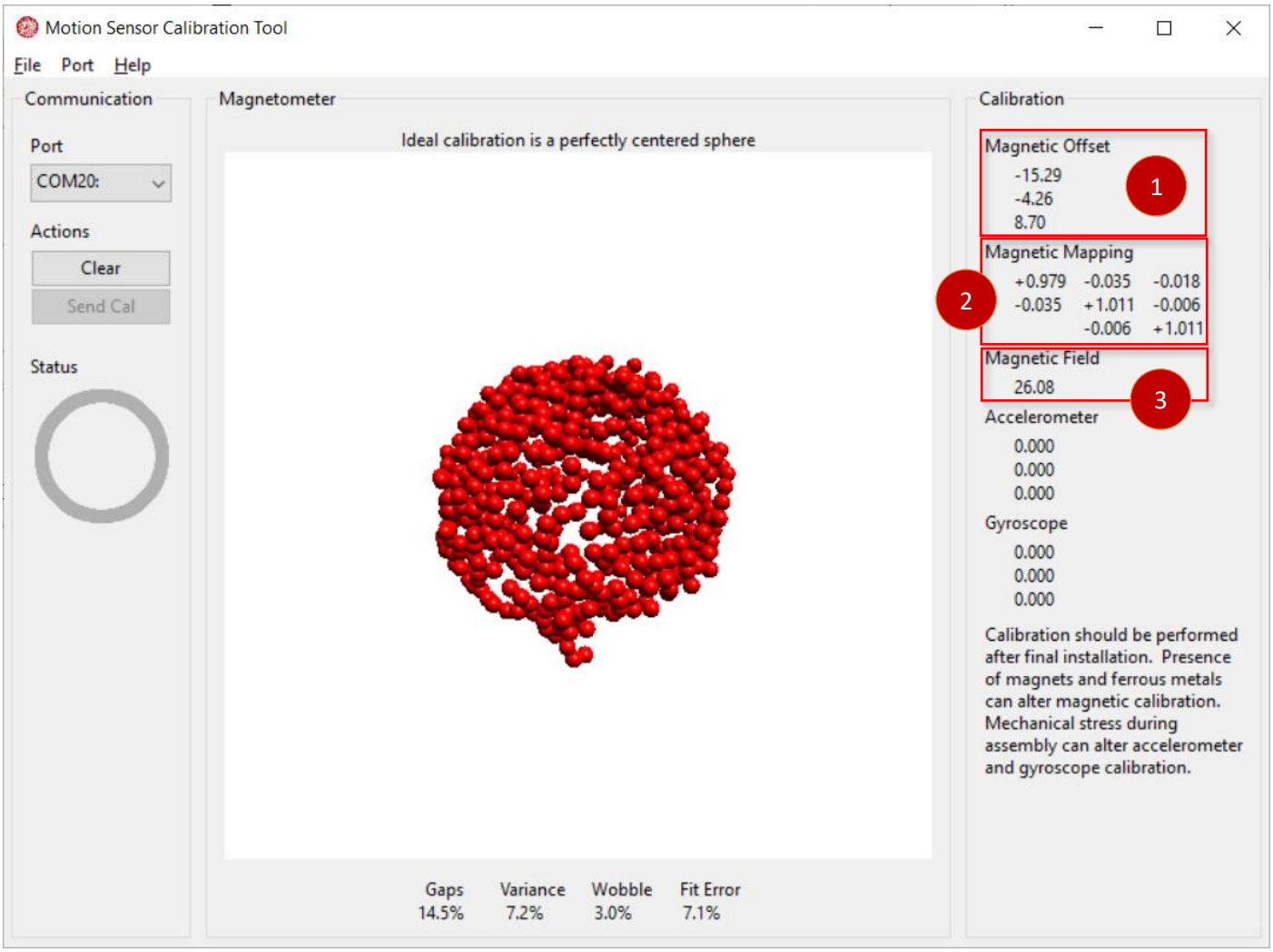


Abbildung 8: MotionCal in Serieller Verbindung mit dem ICM20948

Sind die Werte stabil und ändern sich nicht mehr, so sollte man sie direkt in die Arduino-Umgebung einpflegen und zwar in die zweite Datei „**ahrs_fusion.ino**“:

```
// Offsets applied to raw x/y/z mag values
float mag_offsets[3]            = { -15.29, -4.26, 8.70 };

// Soft iron error compensation matrix
float mag_softiron_matrix[3][3] = { {  0,979,  -0.0035,  -0.0018 },
                                    {  -0.035,  1.011, -0.006 },
                                    {  0.00,  -0.006,  +1.011 } };

float mag_field_strength        = 26.08;
```

Abbildung 9: Beispiel-Kalibrierung wie oben gemessen

Eulercode mit Magnetisierungs-Kalbrierung:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Madgwick_DPEng.h>
#include <DPEng_ICM20948_AK09916.h>

DPEng_ICM20948 dpEng = DPEng_ICM20948(0x948A, 0x948B, 0x948C);

float mag_offsets[3]            = { -14.0, -4.0, 8.0 };

float mag_softiron_matrix[3][3] = { {  1.0,  0.0,  0.0 },
                                    {  0.000,  1.02, 0.08 },
                                    {  -0.001,  +0.08,  0.99 } };
float mag_field_strength        = 28.0;
float gyro_zero_offsets[3]      = { 0.0F, 0.0F, 0.0F };

Madgwick_DPEng filter;

void setup()
{
  Serial.begin(115200);

  if(!dpEng.begin(ICM20948_ACCELRANGE_4G, GYRO_RANGE_250DPS, ICM20948_ACCELLOWPASS_50_4_HZ))
  {
    Serial.println("Ooops, no ICM20948/AK09916 detected ... Check your wiring!");
    while(1);
  }
  filter.begin();
}

void loop(void)
{
  sensors_event_t accel_event;
  sensors_event_t gyro_event;
  sensors_event_t mag_event;

  dpEng.getEvent(&accel_event, &gyro_event, &mag_event);

  float x = mag_event.magnetic.x - mag_offsets[0];
  float y = mag_event.magnetic.y - mag_offsets[1];
  float z = mag_event.magnetic.z - mag_offsets[2];

  float mx = x * mag_softiron_matrix[0][0] + y * mag_softiron_matrix[0][1] + z *
mag_softiron_matrix[0][2];
  float my = x * mag_softiron_matrix[1][0] + y * mag_softiron_matrix[1][1] + z *
mag_softiron_matrix[1][2];
  float mz = x * mag_softiron_matrix[2][0] + y * mag_softiron_matrix[2][1] + z *
mag_softiron_matrix[2][2];

  float gx = gyro_event.gyro.x + gyro_zero_offsets[0];
  float gy = gyro_event.gyro.y + gyro_zero_offsets[1];
  float gz = gyro_event.gyro.z + gyro_zero_offsets[2];

  filter.update(gx, gy, gz, accel_event.acceleration.x, accel_event.acceleration.y,
accel_event.acceleration.z, mx, my, mz);

  float roll = filter.getRoll();
  float pitch = filter.getPitch();
  float heading = filter.getYaw();
  Serial.println(heading);
  delay(10);
}
```