Reading MPU9250 sensors with Arduino

https://www.hackster.io/hibit/reading-mpu9250-sensors-with-arduino-a18f50

Hardware components

Arduino Nano (or another Arduino module)	× 1
MPU9250	× 1
Mini Breadboard	× 1
Dupont wires	× 1

Software apps and online services

Arduino IDE

Story

Hauptartikel: MPU9250 Sensoren mit Arduino lesen

MPU-9250 ist einer der fortschrittlichsten kombinierten Beschleunigungsmesser, Gyroskop und Kompass, die derzeit verfügbar sind. Es ersetzt die beliebte MPU-9150, die den Stromverbrauch senkt, was Gyro-Lärm und Kompass für die volle Reichweitenleistung verbessert. Es hat viele erweiterte Funktionen, darunter Low-Pass-Filterung, Bewegungserkennung und sogar einen programmierbaren spezialisierten Prozessor.

Intern umfasst es die MPU-6500, die ein 3-Achsen-Gyroskop sowie einen 3-Achs-Beschleunigungsmesser enthält, und den *AK8963*, den marktführenden 3-Achsen-Digitalkompass. Die MPU-9250 verwendet 16-Bit-Analog-Digital-Wandler (ADCs) zur Digitalisierung aller 9 Achsen.

Gyroskop

Ein Gyroskop ist ein Gerät zur Messung oder Aufrechterhaltung der Orientierung und Winkelgeschwindigkeit. Gemessen in Grad (oder Radian) pro Sekunde ist die Winkelgeschwindigkeit die Änderung des Drehwinkels des Objekts pro Zeiteinheit.

Je nach Richtung gibt es drei Arten von Winkelfrequenzmessungen:

- **Yaw** : die horizontale Drehung auf einer flachen Oberfläche, wenn man das Objekt von oben sieht.
- **Pitch** : vertikale Drehung, wie das Objekt von vorne gesehen.
- **Rolle** : horizontale Drehung, wenn man das Objekt von vorne sieht.

Fluguhr

Beschleunigungsmessersensoren sind integrierte Schaltungen (ICs), die die Beschleunigung messen, was die Geschwindigkeitsänderung pro Zeit der Einheit ist. Die Messung der Beschleunigung ermöglicht es, Informationen wie Objektneigung und Vibration zu erhalten.

Generell wird *g* als Einheit zur Beschleunigung verwendet, relativ zur Standard-Schwerkraft (1g x 9,80665 m/s2).

Magnetometer

Das Magnetometer liefert Informationen über das vom Gerätesensor erfasste Magnetfeld und kann theoretisch die Position eines Benutzers freilegen. Der Magnetometersensor misst das Magnetfeld für alle drei physikalischen Achsen (x, y, z) in T (micro Tesla).

Der Absolute Orientierungssensor ist einer der üblichen Anwendungsfälle eines Magnetometers und stellt eine stationäre Ausrichtung (fest zum Magnetfeldvektor und Schwerkraftvektor) zur Erdebene dar.

Verdrahtungsschema

Für Grundnutzung verbinden Sie die Netzteilstifte und *SDA*, *SCL*SCL-Pins nach dem unten gezeigten Verbindungsdiagramm:

Interpretation von Daten

Wir haben das offizielle MPU9250-Datenblatt und die Registerkarte in unserem offiziellen Repository. Es beschreibt die Funktion und den Inhalt jedes Registers innerhalb der MPU-9250, einschließlich des Magnetometers. Wir empfehlen Ihnen, Dezimal-, Binär- und Hex-Darstellungen zu lesen, um ein Register leicht zu identifizieren und die Bedeutung des darin enthaltenen Inhalts zu verstehen.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	SELF_TEST_X_GYRO	R/W	xg_st_data [7:0]							
01	1	SELF_TEST_Y_GYRO	R/W	yg_st_data [7:0]							
02	2	SELF_TEST_Z_GYRO	R/W	zg_st_data [7:0]							
0D	13	SELF_TEST_X_ACCEL	R/W	XA_ST_DATA [7:0]							
0E	14	SELF_TEST_Y_ACCEL	R/W	YA_ST_DATA [7:0]							
0F	15	SELF_TEST_Z_ACCEL	R/W	ZA_ST_DATA [7:0]							
13	19	XG_OFFSET_H	R/W	X_OFFS_USR [15:8]							
14	20	XG_OFFSET_L	R/W	X_OFFS_USR [7:0]							
15	21	YG_OFFSET_H	R/W	Y_OFFS_USR [15:8]							
16	22	YG_OFFSET_L	R/W	Y_OFFS_USR [7:0]							
17	23	ZG_OFFSET_H	R/W	Z_OFFS_USR [15:8]							
18	24	ZG_OFFSET_L	R/W	Z_OFFS_USR [7:0]							
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	FIFO_MODE	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	XGYRO_Cten	YGYRO_Cten	ZGYRO_Cten	GYRO_FS_SEL [1:0]		-	FCHOICE_B[1:0]	

.

Die ersten beiden Spalten der Registerkarte geben die Adresse in HEX und Dezimalformate mit anschließendem Namen an. Einige der Register sind nur gelesen (*mit R* gekennzeichnet) und andere erlauben uns zu schreiben (*mit R/W* gekennzeichnet). Der Inhalt des Registers wird mit 8 Bit dargestellt. In verschiedenen Fällen kann der Wert des Registers mehr als eine Sache darstellen, mit anderen Worten, jede Bit- oder Bitgruppe kann eine andere Bedeutung haben, z.B. *GYRO-CONFIG (Register 27*).

Die MPU-9250 hat 16-Bit-Präzision für jeden der Sensoren. Das bedeutet, dass zwei 8-Bit-Register verwendet werden, um die Ausgabe darzustellen. Wir lesen 8-Bit-Daten getrennt von jedem Register und verabreichen sie dann zu 16-Bit.

Arduino Bibliothek für I2C installieren

Wir verwenden das I2C-Protokoll, um mit Registern und Daten zu lesen/schreiben. Die I2C-Bibliothek bietet eine sehr einfache Schnittstelle zu diesem Zweck und kann in anderen Projekten, die das I2C-Protokoll verwenden, wiederverwendet werden. Es kann aus unserem offiziellen Repository heruntergeladen werden.

Um eine Bibliothek zu importieren, öffnen Sie die Arduino IDE, gehen Sie zu Sketch > Include Library > Add.ZIP Library und wählen Sie die Bibliotheksdatei aus unserem GitHub-Repository.

Dann können Sie einfach *eine* Erklärung verwenden:

```
#include "I2C.h"
```

Es wird die Bibliothek mit vordefinierten Funktionen enthalten, um mit Registern zu interagieren.

Arduino Code

Wir haben für jede Rohsensordaten einen separaten Konstrukt definiert: *Gyroskop-Rraw*, *Beschleunigungsmesser,* , *magnetometer-roh* und *temperatur-roh*. Jedes Rohwerk hat eine Datennormalisator-Funktion, die auf der aktuellen Konfiguration basiert. Menschliche lesbare normalisierte Werte werden in einer anderen als *normalisierten* Struktur gespeichert.

Achten Sie darauf, dem vorherigen Schritt zu folgen und die I2C-Bibliothek zu importieren, um sie mit der *include*Include-Anweisung zu verwenden:

```
#include "Wire.h"
#include "I2C.h"

#define MPU9250_IMU_ADDRESS 0x68
#define MPU9250_MAG_ADDRESS 0x0C

#define GYRO_FULL_SCALE_250_DPS  0x00
#define GYRO_FULL_SCALE_500_DPS  0x08
#define GYRO_FULL_SCALE_1000_DPS 0x10
#define GYRO_FULL_SCALE_2000_DPS 0x18

#define ACC_FULL_SCALE_2G  0x00
#define ACC_FULL_SCALE_4G  0x08
#define ACC_FULL_SCALE_8G  0x10
#define ACC_FULL_SCALE_16G 0x18

#define TEMPERATURE_OFFSET 21 // As defined in documentation

#define INTERVAL_MS_PRINT 1000

#define G 9.80665

struct gyroscope_raw {
  int16_t x, y, z;
} gyroscope;

struct accelerometer_raw {
  int16_t x, y, z;
```

```
} accelerometer;

struct magnetometer_raw {
  int16_t x, y, z;

  struct {
    int8_t x, y, z;
  } adjustment;
} magnetometer;

struct temperature_raw {
  int16_t value;
} temperature;

struct {
  struct {
    float x, y, z;
  } accelerometer, gyroscope, magnetometer;

  float temperature;
} normalized;

unsigned long lastPrintMillis = 0;

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

  I2CwriteByte(MPU9250_IMU_ADDRESS, 27, GYRO_FULL_SCALE_1000_DPS); // Configure
gyroscope range
  I2CwriteByte(MPU9250_IMU_ADDRESS, 28, ACC_FULL_SCALE_2G);        // Configure
accelerometer range

  I2CwriteByte(MPU9250_IMU_ADDRESS, 55, 0x02); // Set by pass mode for the
magnetometers
  I2CwriteByte(MPU9250_IMU_ADDRESS, 56, 0x01); // Enable interrupt pin for raw
data

  setMagnetometerAdjustmentValues();

  //Start magnetometer
  I2CwriteByte(MPU9250_MAG_ADDRESS, 0x0A, 0x12); // Request continuous
magnetometer measurements in 16 bits (mode 1)
}

void loop()
{
  unsigned long currentMillis = millis();

  if (isImuReady()) {
    readRawImu();

    normalize(gyroscope);
    normalize(accelerometer);
    normalize(temperature);
  }

  if (isMagnetometerReady()) {
    readRawMagnetometer();

    normalize(magnetometer);
  }
```

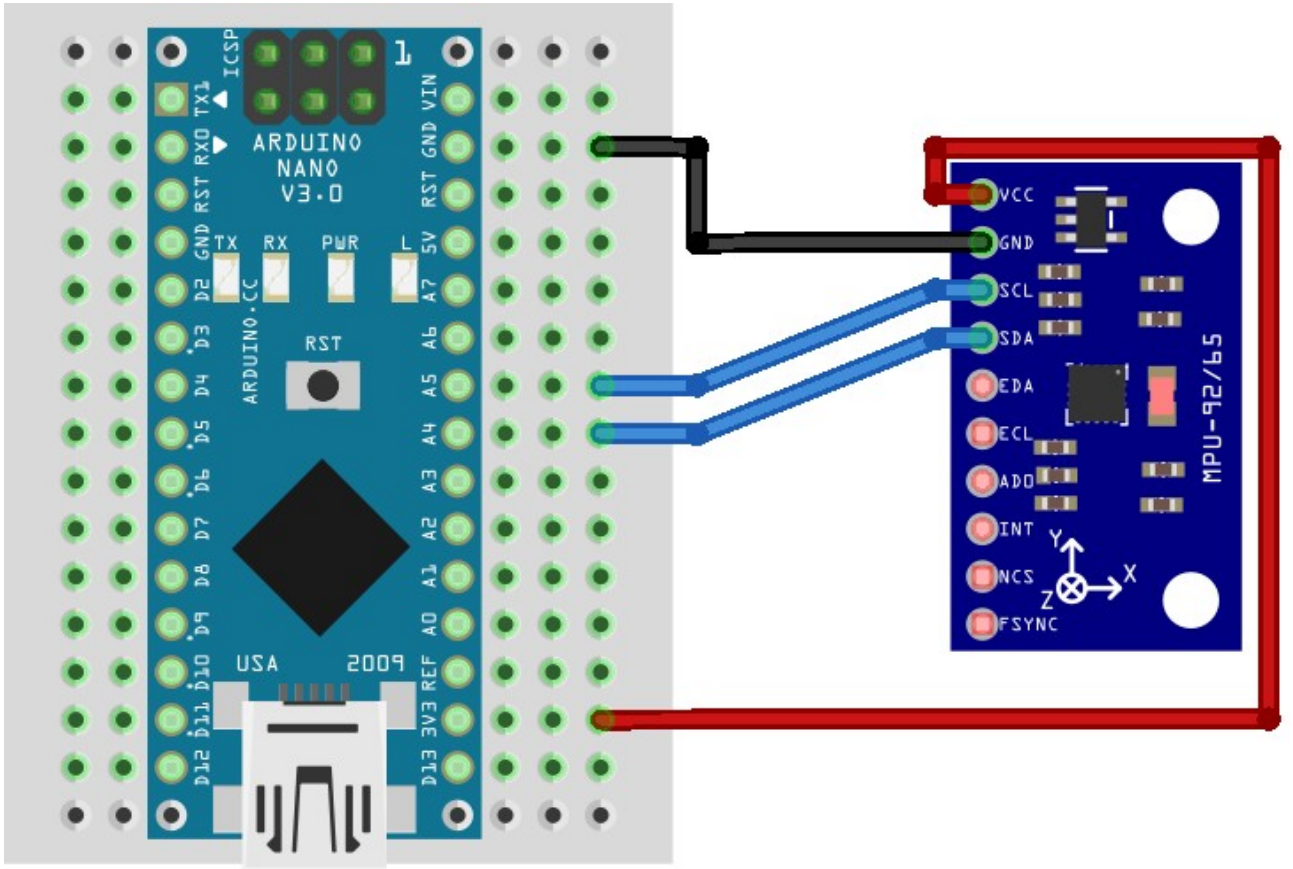
```
  if (currentMillis - lastPrintMillis > INTERVAL_MS_PRINT) {
    Serial.print("TEMP:\t");
    Serial.print(normalized.temperature, 2);
    Serial.print("\xC2\xB0"); //Print degree symbol
    Serial.print("C");
    Serial.println();

    Serial.print("GYR (");
    Serial.print("\xC2\xB0"); //Print degree symbol
    Serial.print("/s):\t");
    Serial.print(normalized.gyroscope.x, 3);
    Serial.print("\t\t");
    Serial.print(normalized.gyroscope.y, 3);
    Serial.print("\t\t");
    Serial.print(normalized.gyroscope.z, 3);
    Serial.println();

    Serial.print("ACC (m/s^2):\t");
    Serial.print(normalized.accelerometer.x, 3);
    Serial.print("\t\t");
    Serial.print(normalized.accelerometer.y, 3);
    Serial.print("\t\t");
    Serial.print(normalized.accelerometer.z, 3);
    Serial.println();

    Serial.print("MAG (");
    Serial.print("\xce\xbc"); //Print micro symbol
    Serial.print("T):\t");
    Serial.print(normalized.magnetometer.x, 3);
    Serial.print("\t\t");
    Serial.print(normalized.magnetometer.y, 3);
    Serial.print("\t\t");
    Serial.print(normalized.magnetometer.z, 3);
    Serial.println();

    Serial.println();

    lastPrintMillis = currentMillis;
  }
}
```

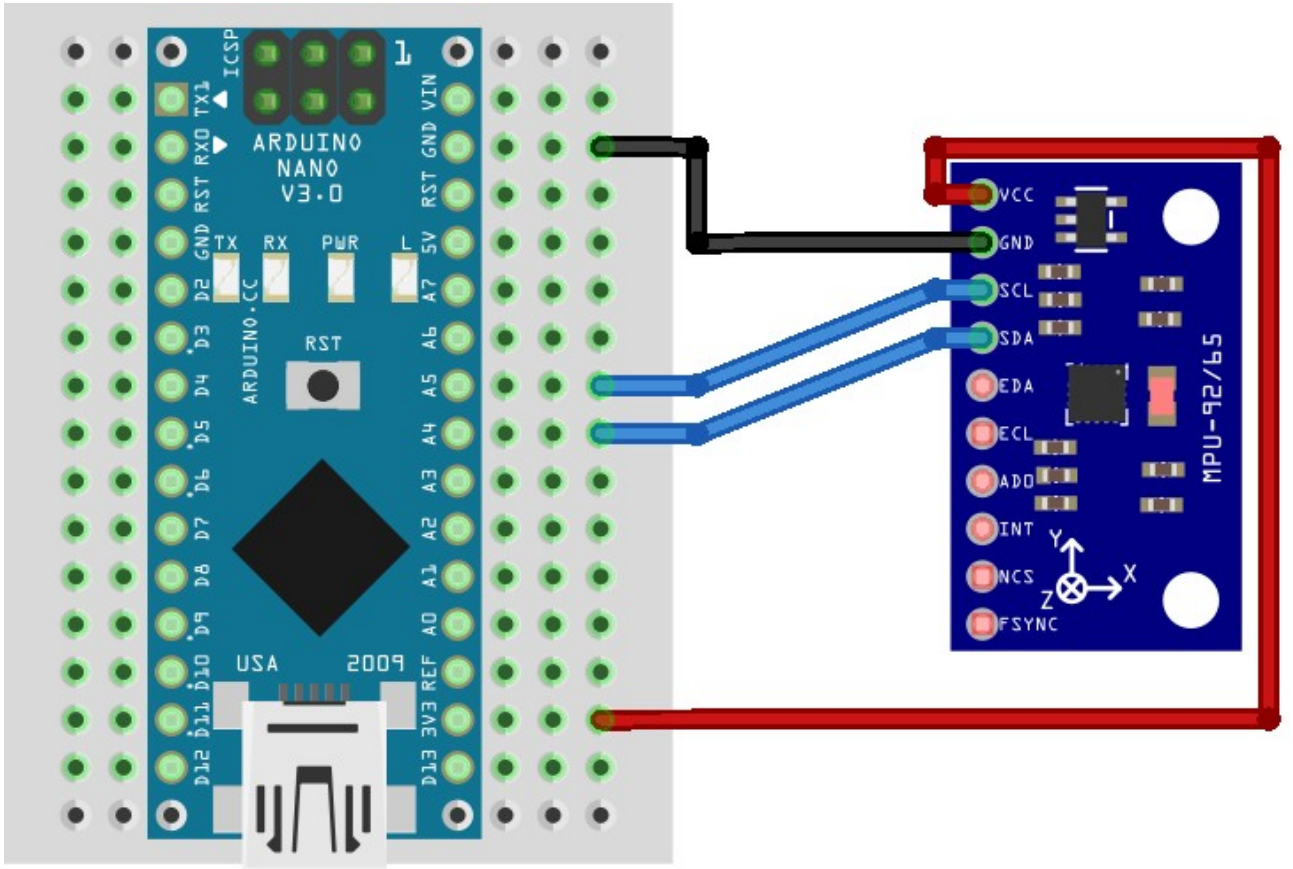
Hinweis: Der Snippet ist Teil des Arduino-Projekts in unserem GitHub-Repository mit dem in verschiedenen logischen Dateien getrennten Code.

Testen

Der serielle Monitor druckt die letzten verfügbaren Sensoren in jeder *INTERVAL_MS_PRINT*Millisekunden INTERVAL-MS-PRINT-MINT-Folgesse (im obigen Beispiel einmal pro Sekunde) und es sollte ähnlich sein wie:

Normalerweise wird die Orientierung auf dem physikalischen Modul gezeichnet, so dass Sie X-, Y- und Z-Achsen leicht erkennen können.


ARDUINO
NANO
V3.0
ARDUINO.CC
TX
RX
PWR
L
RST
USA
2009
VCC
GND
SCL
SDA
EDA
ECL
ADO
INT
NCS
FSYNC
MPU-92/65
Y
X
Z
fritzing


ARDUINO
NANO
V3.0
ARDUINO.CC
TX
RX
PWR
L
RST
USA
2009
VCC
GND
SCL
SDA
EDA
ECL
ADO
INT
NCS
FSYNC
MPU-92/65
Y
X
Z
fritzing