Die Sensorik eines self balancing Robots

Alles wichtige über das MPU6050 & den HCSR-04

Verfasst von Tim Kreitmaier

Kurzzusammenfassung: In diesem Artikel werden Aufbau, Funktionsweise und richtige Initialisierung des MPU6050 Bewegungssensors und des HCSR-04 Ultraschallsensors erläutert.

Der Umsatz von Sensoren und Messtechnik ist 2020 im Vergleich zu 2005 um ungefähr 225% gestiegen. Dieser enorme Anstieg allein zeigt schon, dass Sensoren und Messtechnik immer stärker Einfluss auf unser Leben nehmen. Egal ob im Smartphone, das heutzutage fast jeder in der Hosentasche mit sich trägt, in der Industrie, im Auto, im Smarthome oder allgemein im Haushalt. Überall sind Sensoren wie Temperatursensoren, Bewegungssensoren, Beschleunigungssensoren und viele weitere zu finden. Auch im Messtechnik Praktikum im Sommer 2021 wurde von uns entschlossen einen self-balancing Robot durch den Einsatz verschiedener Sensoren zu realisieren. Als Ziel wurde gesetzt, dass der Roboter, der auf zwei Rädern steht, von sich aus balanciert, sich durch einen Handschuh steuern lässt und Hindernisse in seiner Umgebung meidet. Zur Umsetzung wurden zwei Sensoren verbaut, auf die im nachfolgenden Artikel genau eingegangen wird. Für das selbstständige Balancieren des Roboters wurde ein Sensor gebraucht, der Daten über die Ausrichtung und die Neigung gibt. Hierfür wurden MPU 6050 Module verwendet. Für das Erkennen von Hindernissen in der Umgebung wurden zwei HC-SR04 Ultraschallsensoren besorgt. Im folgenden Text wird auf den Aufbau, die Funktionsweise und die richtige Initialisierung, beziehungsweise auf die Ansteuerung dieser zwei Sensoren detailliert eingegangen. Das Ziel dieses Artikels ist, dass Interessierte und Hobby Arduino Bastler den Aufbau und die genaue Funktionsweise dieser beiden Sensoren verstehen und außerdem durch kurze Initialisierungscodebeispiele diese für eigene Projekte verwenden können.

Abbildung: MPU6050 (von oben)

Abbildung 1: HC-SR04 (von oben)

Im ersten Teil des Artikels wird auf den 3-Achsen Beschleunigungssensor und 3-Achsen Gyroskop, das MPU6050 Modul eingegangen. Dieses ist für ungefähr drei Euro im Internet erhältlich. Als erstes werden der Aufbau und die technischen Eigenschaften des MPUs erläutert. Anschließend werden die Grundlagen des Messprinzips und als letztes die richtige Initialisierung in Code erklärt.

Im MPU6050 befinden sich MEMS die bei Beschleunigung ihre Lage im Raum verändern. Die genaue Funktionsweise der MEMS wird jetzt erklärt.

Grundlagen des Messprinzips:

Üblicherweise arbeiten Micro-Electro-Mechanical Systems (MEMS)-Beschleunigungssensoren bzw. Gyratoren nach dem Feder-Masse- oder Stimmgabelprinzip. Bei ersterem erfassen eingebaute Signalverstärker die Verformung oder Bewegung einer Feder über Kapazitätsveränderungen oder den Piezoeffekt. Beim Stimmgabelprinzip schwingen magnetisch angeregte Kammstrukturen, deren Auslenkung von Sensoren erfasst werden.

Aufbau eines MEMS-Beschleunigungssensors:

Im Folgenden wird der Aufbau eines Beschleunigungssensors nach dem Feder-Masse-Prinzip erläutert. Dabei ist zu berücksichtigen, dass es Beschleunigungssensoren gibt, die auch nach anderen Prinzipien arbeiten. Das MPU6050 allerdings funktioniert nach genau diesem Prinzip.

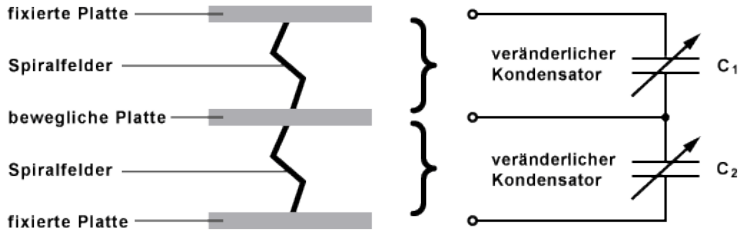

Abbildung 2: Aufbau eines MEMS-Beschleunigungssensors

Der Aufbau dieses MEMS-Beschleunigungssensors besteht aus drei gestapelten Platten, die über Spiralfedern miteinander verbunden sind. Die äußeren Platten sind fest und die mittlere Platte ist beweglich. Ihre Beweglichkeit wird durch die Spiralfedern begrenzt. Durch diesen Aufbau entsteht eine Reihenschaltung von zwei Kondensatoren mit veränderlicher Kapazität. Veränderlich deshalb, weil sich der Plattenabstand der beiden Kondensatoren durch die mittlere Platte verändert. Der Plattenabstand verändert die Kapazität eines Kondensators nach der Formel:

$$Q = \varepsilon_0 * \frac{A}{d} * U$$

mit ε_0= elektrische Feldkonstante, A=Plattenfläche, d=Abstand, U=Spannung.

Funktionsweise eines MEMS-Beschleunigungssensors:

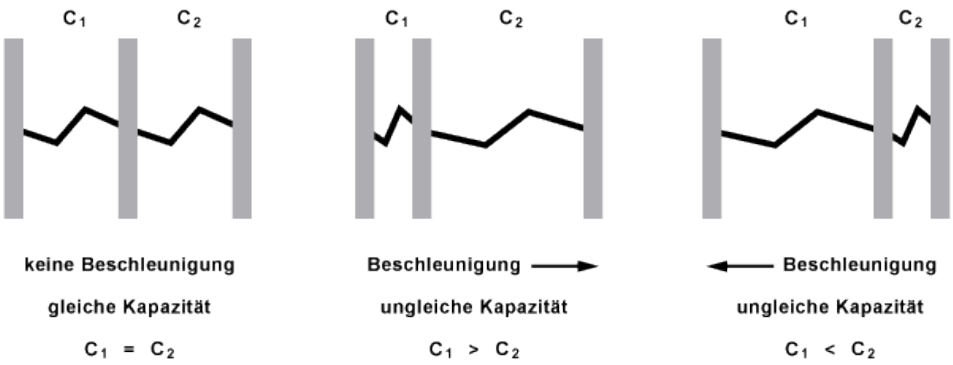

Abbildung 3: Funktionsweise eines MEMS-Beschleunigungssensors

Wenn keine Bewegung stattfindet oder die Beschleunigung gleich Null ist, dann befinden sich die Platten in gleichem Abstand zueinander. Findet nun eine Bewegung, also eine Beschleunigung, statt, dann wird die mittlere Kondensatorplatte zur Seite gedrückt. Dadurch entsteht eine zur Beschleunigung proportionale Kapazitätsänderung. Wenn eine konstante Geschwindigkeit eingehalten wird, dann tritt zwar immer noch eine Bewegung auf, aber keine Beschleunigung mehr. Die mittlere Kondensatorplatte geht in ihre Ursprungsposition zurück. Durch Integrieren der Beschleunigung kann die Geschwindigkeit berechnet werden und aus der Geschwindigkeit durch Integration die zurückgelegte Strecke.

Der Unterschied zwischen Beschleunigungssensor und Gyroskop ist lediglich, dass der Beschleunigungssensor die Beschleunigung in Richtung der x-, y- und z-Achse detektiert, wohingegen das Gyroskop die Bewegung um die Achsen ermittelt. Wenn das Modul in Ruhe ist, liefert das Gyroskop für x, y und z den Wert Null. Der Beschleunigungssensor allerdings detektiert die Erdbeschleunigung auch im Ruhezustand.

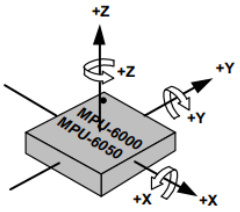

Abbildung 4: Messungen des Beschleunigungssensors und des Gyroskops

Aufbau und Pinout des MPU6050 Modul:

Aufbau:

7.5 Block Diagram

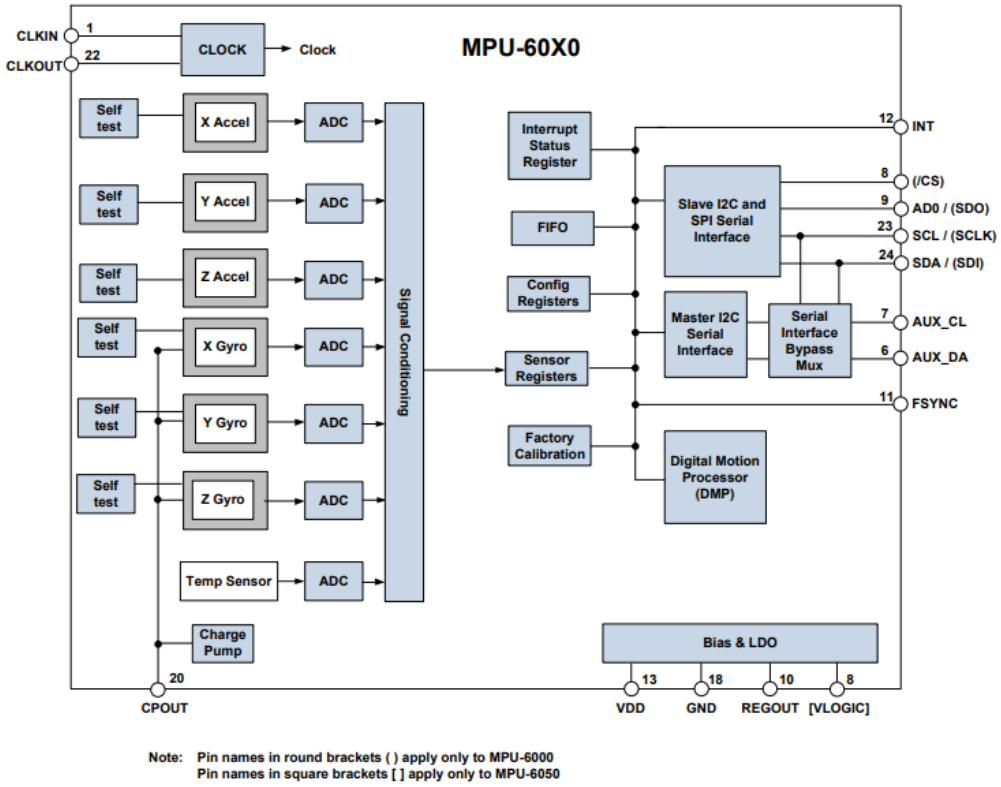

Abbildung 5: Blockschaltbild MPU6050

In diesem Diagramm sieht man das Blockschaltbild eines MPU6050 und alle Komponenten des Moduls. Es besteht aus X, Y und Z Beschleunigungssensoren und X, Y und Z Gyroskopen, einem Temperatursensor, den Clock Modulen, verschiedenen Registern, die für die Kommunikation und den Austausch der Daten über I2C und SPI zuständig sind und mehreren Analog-Digital-Convertern.

Pinout:

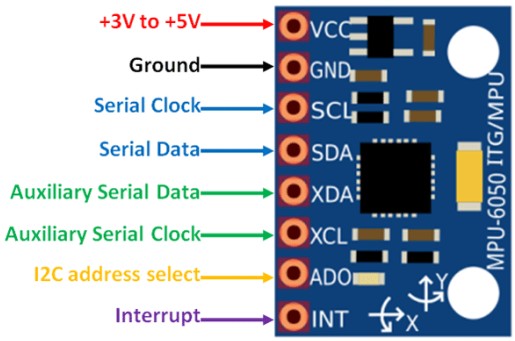

Abbildung 6: Pinout MPU 6050 Modul

Pin Name:	Beschreibung:
Vcc	Anschlusspin für 3-5V (normalerweise 5V)
Gnd (Ground)	Ground Anschluss des Sensors
SCL (Serial Clock)	Sendet Clock Impulse für eine I2C Kommunikation
SDA (Serial Data)	Zuständig für Datentransfer über I2C
XDA (Auxiliary Serial Data)	Optionaler Pin für andere Geräte mit I2C Modulen und Weiter MPU6050
XCL (Auxiliary Serial Clock)	Optionaler Pin für andere Geräte mit I2C Modulen und Weiter MPU6050
AD0	Kann die Adresse verändern, falls mehrere MPUs angeschlossen sind
INT (Interrupt)	Falls Daten vom MPU bereitstehen, zeigt das dieser Pin an

Diverse wichtige technische Eigenschaften:

- Spannungsversorgung (VDD/GND): das Modul besitzt einen LDO Spannungskonverter, kann also sowohl mit 3,3 V wie auch mit 5 V versorgt werden.
- Stromverbrauch (Modul): ca. 5,1 mA, im Sleep-Mode ca. 1,4 mA (eigene Messungen).
- Beschleunigungssensor: 16-Bit Auflösung, Messbereiche +/- 2, 4, 8 oder 16 g.
- Gyroskop: 16-Bit Auflösung, Messbereiche: +/- 250, 500, 1000 oder 2000°/s.
- Kommunikation (SDA/SCL): I2C, Adresse 0x68 oder 0x69.
- Interrupt Funktion: „Data Ready“, „Free Fall“, Beschleunigungslimit.

Initialisierung in Code:

Zur Benutzung des MPU6050 Moduls muss eine Steuereinheit, wie zum Beispiel ein Arduino Nano verwendet werden. Nach dem richtigen Verkabeln des MPUs flasht man den benötigten Code auf die Steuereinheit. In diesem Beispiel wird eine Bibliothek namens Arduino-MPU6050-master genutzt, um den unvermeidlichen Standardfehler eines jeden MPU6050 zu kompensieren.

Initialisierung der Gyroskope

Das erste Code Beispiel behandelt die Initialisierung der Gyroskope und das Auslesen der korrekten Daten:

```
1  #include <MPU6050.h>
2
3  //**********************mpu
4  MPU6050 mpu;                                          //initialise a MPU6050 element
5  //*********************/mpu
6
7  //_____________________________________________________________________________-setup
8  void setup()
9  {
10   while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))    //init MPU
11   {
12     delay(500);
13   }
14   mpu.calibrateGyro();                                 //calibrate to eliminate static error
15   mpu.setThreshold(3);                                 //set sensibility
16 }
17
18 //_____________________________________________________________-loop
19 void loop()
20 {
21   Vector normGyro = mpu.readNormalizeGyro();
22   Serial.print(" Xnorm = ");
23   Serial.print(normGyro.XAxis);
24   Serial.print(" Ynorm = ");
25   Serial.print(normGyro.YAxis);
26   Serial.print(" Znorm = ");
27   Serial.println(normGyro.ZAxis);
28
29   delay(10);
30 }
31
```

Zeile 1 inkludiert die Arduino-MPU6050-master Bibliothek, mit der der Standardfehler des Gyros behoben wird und die Daten ausgelesen werden.

Zeile 4 initialisiert ein Objekt der Arduino-MPU6050-master Bibliothek.

In der Setup-Funktion wird zuerst gewartet, bis das MPU Modul komplett eingerichtet ist und anschließend wird der oben genannte Fehler durch die eingebundene Bibliothek behoben.

In der Loop-Funktion werden alle korrigierten Werte der drei Gyros ausgegeben.

Initialisierung der Beschleunigungssensoren

Das zweite Code Beispiel zeigt die richtige Initialisierung der Beschleunigungssensoren, ebenfalls mit der Arduino-MPU6050-master Bibliothek:

```
1  #include <MPU6050.h>
2
3  //**********************mpu
4  MPU6050 mpu;                                          //initialise a MPU6050 element
5  //**********************/mpu
6
7  //________________________________________________________________________________-setup
8  void setup()
9  {
10   while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))    //init MPU
11   {
12     delay(500);
13   }
14 }
15
16 //_____________________________________________________________-loop
17 void loop()
18 {
19   Vector normAccel = mpu.readNormalizeAccel();
20
21   Serial.print(" Xnorm = ");
22   Serial.print(normAccel.XAxis);
23   Serial.print(" Ynorm = ");
24   Serial.print(normAccel.YAxis);
25   Serial.print(" Znorm = ");
26   Serial.println(normAccel.ZAxis);
27   delay(10);
28 }
```

Zeile 1 inkludiert die Arduino-MPU6050-master Bibliothek, mit der die Daten ausgelesen werden.

Zeile 4 initialisiert ein Objekt der Arduino-MPU6050-master Bibliothek.

In der Setup-Funktion wird zuerst gewartet, bis das MPU Modul komplett eingerichtet ist.

In der Loop-Funktion werden alle korrigierten Werte der drei Beschleunigungssensoren ausgegeben.

Im zweiten Teil des Artikels wird der HCSR-04 Ultraschallsensor genauer erklärt. Dieser ist für vier Euro im Internet bestellbar. Genau wie bei dem MPU Modul wird hier auf den Aufbau und die wichtigsten technischen Eigenschaften, auf die Funktionsweise und letztens auf die richtige Initialisierung in Code eingegangen.

Aufbau und Pinout des HCSR-04:

Pin Name:	Beschreibung:
Vcc	Anschlusspin für 5V
Trig (Trigger)	Triggert die Ultraschallimpulse
Echo	Gibt einen Impuls aus, wenn das reflektierte Signal empfangen wird.
GND (Ground)	Groundanschluss des Sensors

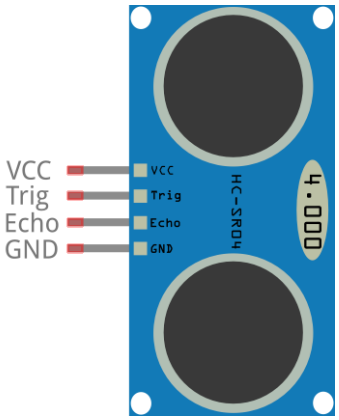

Abbildung 7: Pinout HCSR-04

Diverse wichtige technische Eigenschaften:

- Spannungsversorgung des HSR-04: Der Ultraschallsensor wird mit 5V(+/- 10%) Vcc betrieben
- Stromverbrauch: ca. 2mA pro Messung
- Signal Level: TTL-Pegel
- Min./max. Entfernung: ca. 2cm bis 3 Meter
- Maximale Messungen: 50 Messungen pro Sekunde
- Genauigkeit: Der Ultraschallsensor misst ungefähr auf 3mm genau

Funktionsweise:

Der Messvorgang wird durch eine fallende Flanke am Trigger-Eingang ausgelöst. Der Ultraschallsensor sendet daraufhin ein 40 kHz Burst-Signal für die Dauer von 200 µS zur eigentlichen Sensorkapsel (Transducer). Danach geht der Echopin sofort auf einen High-Pegel und der Ultraschallsensor wartet auf den Empfang des akustischen Echos. Sobald das Echo registriert wird, fällt der Ausgang auf einen Low-Pegel. Um die genaue Entfernung zu ermitteln, muss ein Mikrocontroller also lediglich für 10 us ein High-Signal an den Trigger-Eingang legen und danach messen, wie lange das High-Signal am Echopin anliegt. Wenn das High Signal länger als 200 ms angelegt war, dann wurde kein Hindernis vom Sensor erkannt oder es ist außerhalb der maximalen Reichweite.

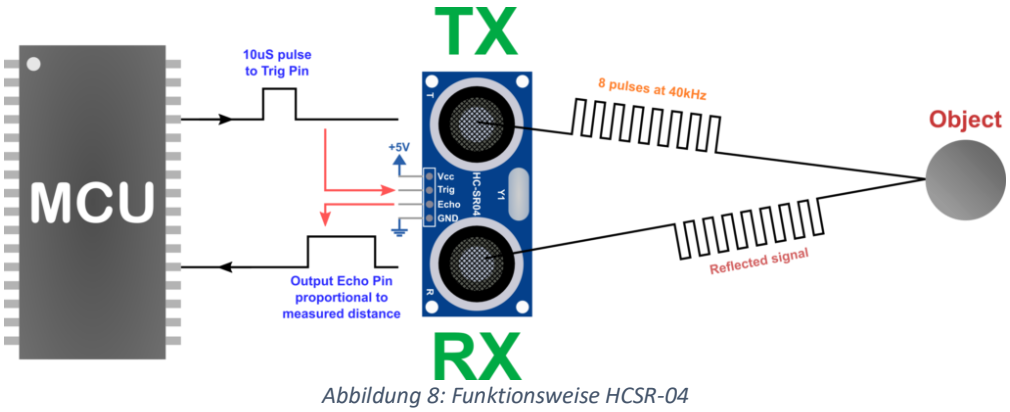

Abbildung 8: Funktionsweise HCSR-04

Zur Berechnung der Entfernung benötigt man nun noch die zugehörige Schallgeschwindigkeit. Die Schallgeschwindigkeit in Luft berechnet sich aus folgender Formel:

$$331{,}5 + (0{,}6 * Temperatur\ [°])$$

Geht man nun von einer Raumtemperatur von 20° aus, ergibt sich mit der Formel der Wert 343,5 m/s. Pro gemessener Mikrosekunde ist der Schall also 0,03434 cm unterwegs. Für die Berechnung der Entfernung teilen wir nun noch durch zwei, um nur die einmalige Wegstrecke zu bekommen und nicht den Hin- und Rückweg. Mit gerundet 0,017 nehmen wir also unsere gemessene Zeit mal und erhalten so die Entfernung in Zentimeter bzw. Meter, die der Schall zurückgelegt hat.

Initialisierung in Code:

```
1   #define tr1 3                       //defines for trigger pins
2   #define tr2 5
3   #define ech1 2                      //defines for echo pins
4   #define ech2 4
5
6
7   int duration;                       //helper variable for the pulse duration
8   int distance_v;                     //distance to front
9   int distance_r;                     //distance to back
10
11  void setup() {
12    pinMode(tr1,OUTPUT);              //pinmodes
13    pinMode(tr2,OUTPUT);
14    pinMode(ech1,INPUT);
15    pinMode(ech2,INPUT);
16  }
17
18  void loop()
19  {
20    digitalWrite(tr1,HIGH);           //distance measurement for side 1
21    delayMicroseconds(10);
22    digitalWrite(tr1,LOW);
23
24    duration=pulseIn(ech1,HIGH);
25    distance_v=0.017*duration;
26
27
28    digitalWrite(tr2,HIGH);           //distance measurement for side 2
29    delayMicroseconds(10);
30    digitalWrite(tr2,LOW);
31
32    duration=pulseIn(ech2,HIGH);
33    distance_r=0.017*duration;
34  }
35
```

In Zeile 1-4 werden die genutzten Pins mit Namen versehen.

In Zeile 7-9 werden für beide HC-SR04 Sensoren die notwendigen Variablen initialisiert.

In der Setup Funktion (Zeile 11-16) werden die Trigger Pins auf Output und die Echo Pins auf Input gesetzt.

Die Zeilen 18-34 sind für die Ultraschallmessungen und die Distanzberechnungen zuständig. In den beiden Distanz Variablen stehen nun die gemessenen Entfernungen.

Schlusswort:

Abschließend kann ich nur sagen, dass die Arbeit mit diesen beiden Sensoren sehr interessant und aufschlussreich war. Vor allem für Beginner und Einsteiger würde ich die HCSR-04 Ultraschallsensoren empfehlen. Diese sind, ohne viel Vorwissen oder Erfahrung, sehr leicht zu verwenden. Das MPU 6050 Modul hingegen benötigt wesentlich mehr Zeit, Wissen und Erfahrung, um es zu initialisieren und zu verwenden. Für jemanden mit Erfahrung in der Programmierung von Mikrocontrollern und Sensoren sollte das MPU allerdings auch keine allzu große Hürde sein. Anwendungsideen für das MPU wären zum Beispiel: Drohnen und Quadrocopter, Beschleunigungstachos, Messungen von Vibrationen oder auch kleine Alarmanlagen bei beweglichen Gütern. Die Ultraschallsensoren kann man in Anwendungen zu Füllstandsmessung, in jeglichen entfernungsmessenden Geräten und auch zur Erkennung von Bewegungen im Raum nutzen. Ich hoffe, dass der Artikel lehrreich war und eventuell dem Leser auch Lust gemacht hat, diese Sensoren selbst mal zu testen, zu programmieren und in eigenen Projekten zu verwenden. Falls der Artikel das Interesse für mehr Details zu unserem Projekt geweckt hat, ist hier noch ein Link zu allen Dateien des Roboters und drei weiteren Artikel. Einer befasst sich mit dem Aufbau und der kompletten Hardware des Roboters, ein weiterer mit unserem geschriebenen Code und der letzte mit der genauen Handschuhsteuerung über Bluetooth.

https://github.com/Stromi1011/SelfBalancingRobot_Woita

Quellen:

MPU6050 Datasheet:
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
Funktionsweise HCSR-04 Bild:
https://trionprojects.org/ultrasonic-sensor-hc-sr04-with-pic-microcontroller/
Bild MPU6050:
https://components101.com/sensors/mpu6050-module
Ultraschall Pinout:
https://ozeki.hu/attachments/1643/ultras-pinout.png
Bild MPU6050:
https://www.arduiner.com/wp-content/uploads/2013/03/377-GY-521-MPU-6050-Modulo-triaxial-Digitale-Giroscopio-6DOF-Modulo-code-schematic.jpg
Daten Ultraschallsensor:
https://www.mikrocontroller-elektronik.de/ultraschallsensor-hc-sr04/
Funktionsweise MPU6050:
https://wolles-elektronikkiste.de/mpu6050-beschleunigungssensor-und-gyroskop
Arduion-Master Bibliothek:
https://github.com/jarzebski/Arduino-MPU6050