OTH Regensburg Sommersemester 2021

Robotersteuerung mit einem Handschuh über Bluetooth

von Lukas Weber

Abbildung 1: Foto Handschuh

Grundlegende Idee

Auf einem Handschuh wird eine Schaltung befestigt mit der ein selbstbalancierender Roboter gesteuert werden soll. Neigt man die Hand nach vorne, hinten, links oder rechts werden über Bluetooth Steuersignale an den Roboter gesendet. Der Roboter soll die Steuersignale verarbeiten und dann entsprechend nach vorne und hinten fahren oder sich an der Stelle nach links oder rechts drehen. Währenddessen soll er sich ständig selbst ausbalancieren, was in den vorangehenden Artikeln meiner Kommilitonen behandelt wurde.

Die Hardware und ihre Verschaltung

Die Abbildung zeigt die Bauteile und ihre Verschaltung miteinander. Das Herzstück ist ein Arduino Nano, der auf dem Mikrocontroller ATmega328 basiert. Dieser verarbeitet die Daten, die er vom MPU-6050 Modul erhält. Das MPU-6050 oder auch GY-521 genannt beinhaltet ein 3-Achsen Gyroskop und einen 3-Achsen Beschleunigungssensor mit einem digitalen Bewegungsprozessor. Der Nano schickt Steuersignale an das HC-05 Bluetooth Modul, das diese an den selbstbalancierenden Roboter weiterleitet. Außerdem sind auf der Platine vier LEDs angebracht, die die Neigung des Handschuhs in die entsprechende Richtung anzeigen sollen. Abweichend vom Schaltbild haben die LEDs vier verschiedene Farben, was auf Abbildung 1 zu sehen ist. Die rote LED leuchtet bei einer Neigung nach vorne, die blaue bei einer nach hinten, die grüne bei einer nach links und die gelbe wiederum bei einer nach rechts. Die LEDs erleichtern auch das Finden von Fehlern,

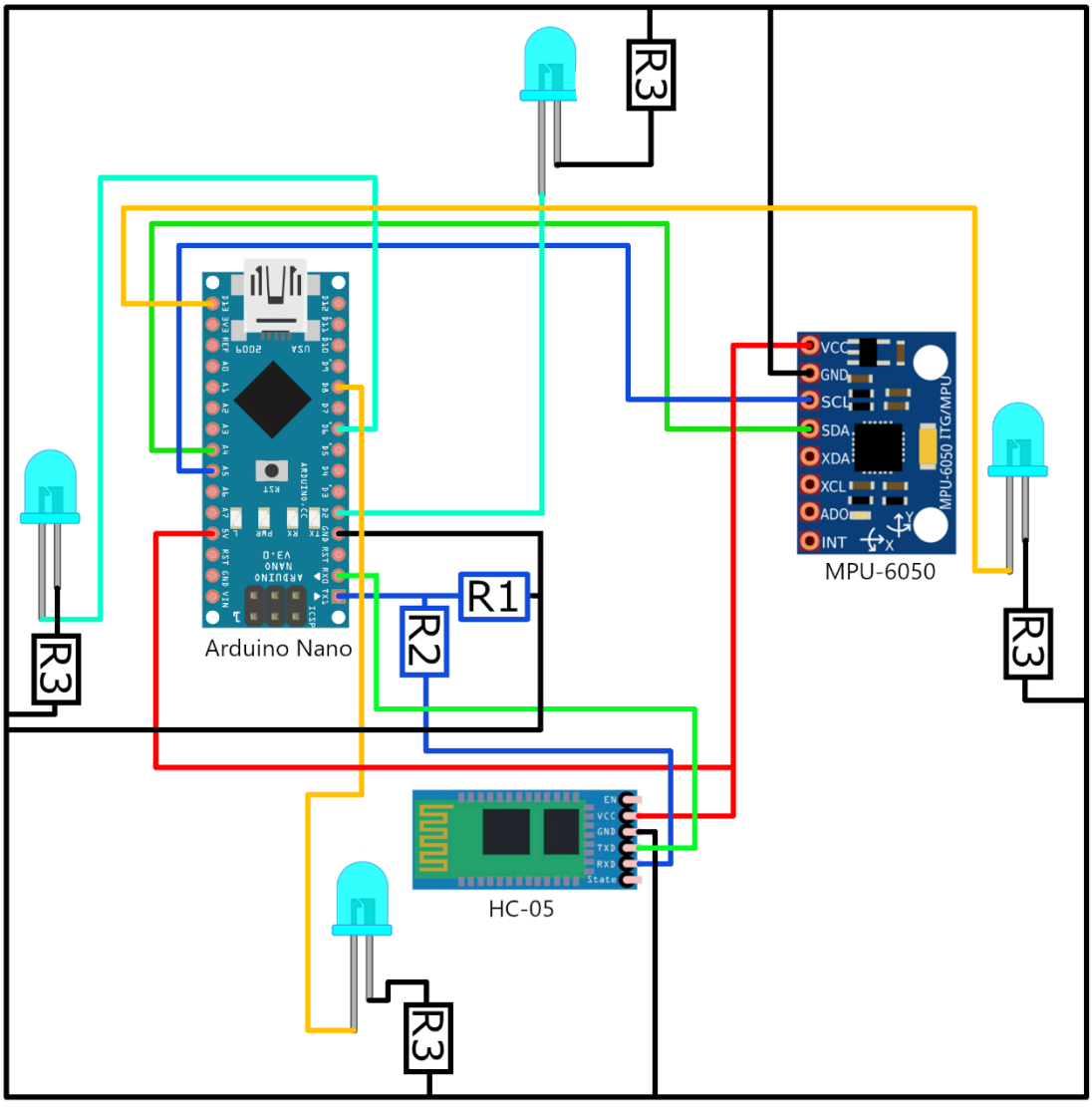

Abbildung 2: Schaltbild des Handschuhs

da in der benutzten Arduino IDE kein Debuggen möglich ist. Am Mini USB-Anschluss des Arduino Nanos ist ein 5V Akkupack als Energiequelle angeschlossen.

Der Arduino Nano gibt über seinen 5V-Pin jeweils die benötigte Spannung an die V_{CC} Pins des HC-05s und des MPU-6050. Für die Datenübertragung vom Tx-Pin des Nanos zum Rx-Pin des Bluetooth Moduls wird ein Spannungsteiler benötigt. Der Rx-Pin des Bluetooth Moduls braucht nach Datenblatt ein Spannungslevel von 3,3V. Hierzu wird für R_1 ein 2kΩ und für R_2 ein 1kΩ Widerstand verwendet. Für die Datenübertragung vom Tx-Pin des Bluetooth Moduls zum Rx-Pin des Arduino Nanos muss man keinen derartigen Spannungsteiler verwenden.

Das MPU-6050 kommuniziert über I^2C mit dem Arduino Nano. Hierfür wird nur der SCL-Pin ($I^2$2 serial clock) mit dem A4-Pin und der SDA-Pin (I^2C serial data) mit dem A5-Pin des Arduino Nano verbunden.

Die LEDs sind zur Strombegrenzung mit einem 100Ω Widerstand in Reihe geschaltet. Wird kein Widerstand verwendet, kann die Stromaufnahme einer LED durch ihre Halbleitereigenschaften sehr stark ansteigen. Gleichzeitig wird die LED wärmer, wodurch sie noch besser leitet. Dieser Kreislauf kann theoretisch bis zur Zerstörung der LED führen. Das ist hier zwar relativ unwahrscheinlich, weil die LEDs nicht im Dauerbetrieb benutzt werden, aber man sollte die Widerstände trotzdem einbauen. Im Schaltbild sind sie mit R_3 gekennzeichnet.

Konfiguration der Bluetooth Module

Das Bluetooth Modul HC-05 kann in zwei Modi benutzt werden. Im Daten-Modus findet die eigentliche Datenübertragung statt und im AT-Command-Modus können verschiedene Einstellungen vorgenommen werden. Dazu gehört zum Bespiel die Auswahl zwischen Master oder Slave. Bei diesem Projekt muss das Modul auf dem Handschuh als Master konfiguriert werden und das Modul auf dem Roboter wird als Slave konfiguriert. Zuerst wird das Empfänger-Modul auf dem Roboter eingestellt. Das Wechseln in den AT-Command-Modus ist ziemlich simpel. Grundsätzlich verwendet man dieselbe Schaltung wie die Teilschaltung zwischen HC-05 und Arduino Nano auf Abbildung 2. Zusätzlich muss nur der 3,3V-Pin des Nanos mit dem Key-Pin des Bluetooth Moduls verbunden werden. Der Key-Pin ist auf dem HC-05 nicht beschriftet. Es ist der erste Pin oben links, wenn man das Bluetooth Modul wie auf Abbildung 3 vor sich liegen hat. Zum Schluss wird mit dem Restart-Taster über dem EN-Pin der Wechsel in den AT-Command-Modus beendet.

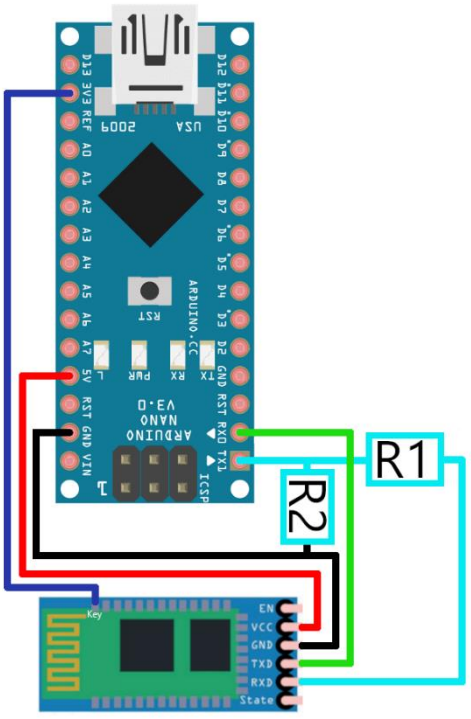

Abbildung 3: Schaltung für AT-Command-Modus

Um in der Arduino IDE einen leeren Sketch auf den Nano zu flashen, muss der Tx-Pin und der Rx-Pin des HC-05 zuerst vom Nano getrennt werden, da der Arduino über die gleiche serielle Schnittstelle programmiert wird, die das HC-05 benutzt. Jetzt kann man den seriellen Monitor öffnen, wo man „Sowohl NL als auch CR“ und eine Baudrate von 38400 auswählt. „AT+UART?“ liefert die Baudrate, die der HC-05 nutzt. Entspricht diese nicht 38400 Baud, muss sie mit „AT+UART=<Param1>,<Param2>, <Param3>“ noch eingestellt werden. Param1 ist dabei die Baudrate. Param2 das Stopbit und Param3 die Parität. Für das Stopbit und die Parität kann default 0 benutzt werden. Die 0 beim Stopbit steht dafür, dass ein Stopbit benutzt wird und durch die 0 bei der Parität gibt es kein Paritätsbit. Mit dem Befehl „AT+ADDR?“ gibt das Modul seine Bluetooth-Adresse zurück, welche man später noch braucht. Mit „AT+ROLE=0“ wird das Modul als Slave konfiguriert.

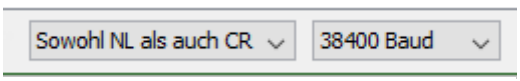

Abbildung 4: Enstellungen in der Arduino IDE

Als Nächstes wird auf dem Sender-Modul der Master-Modus eingestellt. Hier wird zunächst genauso wie oben in den AT-Command Modus gewechselt. Auch hier muss überprüft werden, ob die Baudrate 38400 ist, weil die HC-05s mit dieser Baudrate kommunizieren. Durch das Kommando „AT+ROLE=1“ wechselt das HC-05 in die Master Einstellung.

„AT+CMODE=0“ stellt den Verbindungsmodus „Feste Adresse“ ein. „AT+Bind=<Bluetooth-Adresse>“ legt die Adresse des Empfänger-Moduls fest. Die Bluetooth-Adresse wird hierbei mit der zuvor abgefragten Adresse ersetzt.

Nach diesem Schritt kann in den Daten-Modus gewechselt werden, indem man für beide Bluetooth-Module eine Schaltung wie die Teilschaltung in Abbildung 1 aufbaut. Wenn man alles richtig gemacht hat, werden sie sich von selbst koppeln.

Software

Die Software wird in der Arduino IDE programmiert. In ihr soll das MPU-6050 ausgelesen werden und den erhaltenen Werten entsprechend, die LEDs zum Leuchten gebracht werden. Ebenfalls soll ein Steuersignal über Bluetooth gesendet werden. Wie jedes Arduino-Programm besteht das Programm hauptsächlich aus einer setup()-Funktion-, die nur einmal am Anfang ausgeführt wird, und einer loop()-Funktion, die unendlich oft durchlaufen wird.

```
#include <MPU6050.h>

#define outputPin_f 2
#define outputPin_r 13
#define outputPin_l 6
#define outputPin_h 8
```

Als Erstes wird die MPU6050-Bibliothek von jarzebski inkludiert. Sie stellt verschiedene Funktionen für die einfachere Arbeit mit dem Sensor zur Verfügung. Es werden auch die Digital-Pins, an denen die einzelnen LEDs angeschlossen sind, als Konstanten definiert.

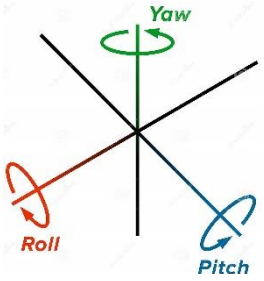

Abbildung 5: Pitch, Roll, Yaw

```
MPU6050 mpu;
const float timeStep = 0.01;
unsigned long timer = 0;
float pitch = 0;
float roll = 0;
```

Die inkludierte Bibliothek ist in C++ verfasst und besteht aus der MPU6050.h und der MPU6050.cpp. Die Bibliothek enthält eine Klasse namens „MPU6050“. Es wird ein Objekt dieser Klasse erstellt. Außerdem werden die Zeitvariablen „timeStep“ und „timer“ angelegt. Pitch beschreibt die Neigung des Handschuhs nach hinten und vorne und Roll die Neigung nach links und rechts. Abbildung 5 zeigt die zwei betroffenen Achsen. Yaw ist für diese Anwendung nicht relevant.

```
struct Vector
{
    float XAxis, YAxis, ZAxis;
};
```

Nun wird die Funktion „readMPU()“ definiert, die dem Auslesen der MPU-Daten dient. In ihr wird als Erstes ein Millisekunden-Timer mit der zuvor angelegten Variable timer gestartet. Dann wird ein Objekt des structs „Vector“ erstellt, das in der Bibliothek definiert ist. Auf der Abbildung ist zu sehen, wie das struct angelegt ist. Das struct wird durch readNormalizeGyro() mit Werten befüllt. readNormalizeGyro() ist eine Member-Funktion der Klasse MPU6050. Diese wiederum ruft als erstes die Methode readRawGyro() auf, die mit Hilfe der Arduino Standard-Bibliothek Wire.h die Daten vom MPU-6050 ausliest. In der readNormalizeGyro() werden diese Daten dann mit den zuvor bestimmten Kalibrierungswerten und dem gesetzten Threshold verrechnet und verbessert. Zurück in der readMPU() werden noch die neuen, tastsächlichen pitch und roll Werte durch die Addition der alten Werte mit den neuen Messwerten bestimmt. Am Ende wird noch ein Delay eingefügt, damit bis zu einer vollen timeStep Periode gewartet wird.

```
void readMPU()
{
  timer = millis();

  Vector norm = mpu.readNormalizeGyro();

  pitch = pitch + norm.YAxis * timeStep;
  roll = roll + norm.XAxis * timeStep;


  delay((timeStep*1000) - (millis() - timer));
}
```

```
void setup()
{
  Serial.begin(9600);

  while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
  {
    delay(500);
  }

  pinMode(outputPin_f, OUTPUT);
  pinMode(outputPin_h, OUTPUT);
  pinMode(outputPin_r, OUTPUT);
  pinMode(outputPin_l, OUTPUT);

  mpu.calibrateGyro();
  mpu.setThreshold(3);
  digitalWrite(outputPin_l, LOW);
}
```

In der setup() wird zunächst die serielle Datenübertragung mit einer Baudrate von 9600 begonnen. Dann wird das MPU mit Hilfe einer weiteren Bibliotheksfunktion initialisiert und gewartet, bis dieses bereit ist. Die vier Pins für die LEDs werden als OUTPUT konfiguriert. Als Nächstes wird das Gyroskop kalibriert, da dessen Ausgangslage in der Regel nicht genau bei 0° ist. Das Ergebnis wird wie oben erwähnt in readNormalizeGyro() verwendet. Außerdem wird ein Threshold von drei gesetzt, der ebenfalls in der readNormalizeGyro() einberechnet wird. Zum Schluss wird die grüne LED noch manuell ausgeschalten, da der D6-Pin anscheinend während des Flashens auf high gesetzt wird.

```
void loop()
{
  readMPU();
```

In der loop()-Funktion wird als erstes die readMPU() aufgerufen, die die Neigungsdaten aktualisiert. Dann wird in vier Blöcken pitch/roll mit selbst definierten Grenzbereichen verglichen. Diese Grenzbereiche können sehr großzügig dimensioniert werden. Allerdings sollten sie in einem Größenbereich sein, dass nicht bereits kleine, versehentliche Bewegungen Befehle senden oder dass die Hand zum Senden unangenehm stark gebeugt werden muss. In der Abbildung ist nur ein Block zu sehen, da alle vier Blöcke fast identisch aufgebaut sind. Falls sich pitch/roll in dem vorgegeben Bereich befindet, wird zuerst die passende LED ein geschaltet. Dann wird nur, wenn pitch sich in dem definierten Bereich befindet entweder ein ´r´ für rückwärts oder ein ´v´ für vorwärts über Serial an das Bluetooth-Modul gesendet. Das ist so, weil im Code des selbstbalancierenden Roboters aus Zeitgründen keine Drehbefehle implementiert sind. So wäre es überflüssig etwas zu senden. Dann wird nach einem kleinen Delay die LED wieder ausgeschaltet.

```
else if(pitch > 15)
{
  digitalWrite(outputPin_h, HIGH);
  Serial.write('r');
  delay(100);
  digitalWrite(outputPin_h, LOW);
}
```

Schlusswort

Falls Interesse an der Funktionsweise des selbstbalancierenden Roboters besteht, gibt es weitere Fachartikel. Darin werden die Verschaltung und Software des Roboters und die bei ihm verwendeten Sensoren MPU-6050 und Ultraschallsensor HC-SR04 genauer behandelt. Zusammen mit dem vollständigen Code des Projekts sind sie hier zu finden: https://github.com/Stromi1011/SelfBalancingRobot_Woita

Abbildungsverzeichnis und Quellen:

Abbildungsverzeichnis:

Quellen:

Inkludierte MPU6050 Datenbank von jarzebski:
https://github.com/jarzebski/Arduino-MPU6050
Informationen zu LEDs:
https://www.elektronik-kompendium.de/sites/bau/0201111.htm
Datenblatt HC-05 Bluetooth Modul:
https://components101.com/asset/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf
Datenblatt MPU-6050:
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
Informationen zum GY-521:
https://www.amazon.de/AZDelivery-MPU-6050-3-Achsen-Gyroskop-Beschleunigungssensor-Arduino/dp/B07N2ZL34Z/ref=sr_1_3?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=gy-521&qid=1624522480&sr=8-3
Konfiguration von Bluetooth Modulen1:
https://www.youtube.com/watch?v=hyME1osgr7s
Konfiguration von Bluetooth Modulen2:
https://www.teachmemicro.com/hc-05-bluetooth-command-list/
AT-Commands:
https://content.instructables.com/ORIG/FKY/Z0UT/HX7OYY7I/FKYZ0UTHX7OYY7I.pdf
Abbildung 5: Pitch, Roll, Yaw:
https://thumbs.dreamstime.com/z/roll-pitch-yaw-three-rotation-angles-corresponding-to-euler-angles-available-high-resolution-good-quality-to-fit-needs-200880861.jpg
Schaltsymbol MPU-6050:
https://components101.com/asset/sites/default/files/component_pin/MPU6050-Pinout.png

Schaltsymbol Arduino Nano:
https://892962.smushcdn.com/2087382/wp-content/uploads/2019/08/Arduino-Nano-Pinout-1.png?lossy=1&strip=1&webp=1
Schaltsymbol HC-05:
https://cdn.shopify.com/s/files/1/1509/1638/files/HC-05_Bluetooth_Modul_Pinout.pdf?v=1608537378
Schaltsymbol LED:
https://projects.raspberrypi.org/en/projects/getting-started-crumble/3