Maturaarbeit 2014

Bau eines selbstbalancierenden Roboters

Lionel Peer Martin Ulmer

Kantonsschule Kreuzlingen

Betreuung: Daniel Zurmühle

Inhaltsverzeichnis

Abbildungsverzeichnis

Zusammenfassung

Im Rahmen der Maturaarbeit 2014 wurde an der Kantonsschule Kreuzlingen durch Lionel Peer und Martin Ulmer, unter der Leitung von Herrn D. Zurmühle, während einem Semester ein selbstbalancierender Roboter konstruiert und ein Bericht dazu verfasst. Während der Entwicklungsphase konnten über Experimente und den Bau von Prototypen wesentliche Erkenntnisse gewonnen werden.

Der Roboter kann über mehrere Minuten unabhängig balancieren. Dazu nutzt er zwei Räder, die sich auf der gleichen Achse befinden. Möglich ist auch das Balancieren auf Kies oder auf einem dünnen Kissen. Die Zeit des Balancierens ist nur durch die Kapazität der Batterien und durch zu grosse Hindernisse begrenzt. Die Regelung des Roboters übernimmt ein PID-Regler, der von einem Arduino Mikrocontroller ausgeführt wird und der seine Daten aus einer MEMS (Microelectromechanical systems) IMU bezieht.

Vorwort

Seit langem haben wir beide Interesse an Elektronik und Informatik. Bei Recherchen im Internet, bspw. in Internetforen und auf YouTube, sahen wir einige fertige Prototypen von Robotern zum Personentransport, die dem Segway sehr ähnlich sind und uns faszinierten. Aufgrund der Erfahrungen aus vorangegangenen Elektronikkursen und der Naturwissenschaftswoche (NWW) schien uns die Umsetzung eines ähnlichen Projektes möglich. Als aus der vagen Idee eine konkrete Vorstellung wurde, mussten wir feststellen, dass ein Gerät zum Personentransport schon aus finanziellen Gründen schwierig umsetzbar sein würde. Daher entschlossen wir uns, ein Modell zu konstruieren.

1 Einleitung

Ziel dieser Maturaarbeit ist die Entwicklung eines selbstbalancierenden Roboters.

> Ein selbstbalancierender Roboter wird im Rahmen dieser Arbeit als Gerät bezeichnet, welches zwei motorisierte Räder auf einer Achse besitzt und diese nutzt, um sich selbst im Gleichgewicht zu halten.

In dieser Arbeit wird versucht, die folgende Leitfrage möglichst vollständig zu beantworten: Ist es möglich mit geringem Vorwissen und im vorgegebenen Zeitrahmen einen funktionsfähigen selbstbalancierenden Roboter zu entwickeln?
Diese Fragestellung ist insofern interessant, als selbstbalancierende Roboter eine Verbindung zwischen physikalischen Vorgängen und deren Umsetzung in Elektronik, Informatik und Regeltechnik herstellen. Ausserdem könnten selbstbalancierende Roboter die heutigen Transportmöglichkeiten – insbesondere die des Fussgängers – revolutionieren.

Der Theorieteil beschäftigt sich einerseits mit den physikalischen Vorgängen des Balancierens und andererseits mit der technischen und elektronischen Umsetzung des Projektes. Die wichtigsten Komponenten werden vorgestellt und ihre Funktionsweise wird erläutert. Ausserdem wird auf die Geschichte des selbstbalancierenden Personentransportes sowie auf die federführenden Entwicklungs- und Produktionspioniere eingegangen.

Im praktischen Teil finden sich einleitende Beispielexperimente, um das Zusammenfügen von Programmierung und Hardware zu zeigen. Ausserdem wird schrittweise der Aufbau des Roboters beschrieben. Es wird versucht, möglichst genau darzustellen, was während des Zeitraumes der praktischen Arbeit erreicht wurde und wo Probleme auftraten. Der praktische Teil richtet sich hauptsächlich an Leser, die ein ähnliches Projekt ins Auge fassen möchten.

2 Theorieteil

In diesem Teil werden zwei verschiedene Firmen vorgestellt, die auf kommerzielle Weise selbstbalancierende Roboter für den Personentransport entwickeln und herstellen.
Zusätzlich wird auch auf die im praktischen Teil verbaute Hardware sowie ihre Funktionsweise eingegangen. Diese beinhaltet die verwendeten Sensoren, den verwendeten Mikrocontroller, den Antrieb und alle für deren Zusammenspiel notwendigen Bauteile.
In einem weiteren Teil werden die physikalischen Vorgänge behandelt, die notwendig sind, um einen Gegenstand zu balancieren. Ausserdem wird anhand von Berechnungen festgestellt, wie ein selbstbalancierender Roboter idealerweise auszusehen hätte.

2.1 Segway

Segway Inc. wurde im Jahre 1999 mit dem Ziel gegründet, ein Transportmittel für Personen zu entwickeln, das die Technologie der dynamischen Stabilisation nutzt. Das Fahrzeug sollte sehr kompakt und wendig sein und vor allem auf Fussgängerstreifen oder in Fussgängerzonen operieren können.

Die ersten Segway PTs (*Personal Transporter*) wurden im Jahre 2002 ausgeliefert, wobei es sich jedoch um Vorserienmodelle handelte. Ab dem Jahr 2003 konnten Segways erstmals von Privatpersonen erworben werden.
Segway Inc. produziert mittlerweile verschiedene Versionen des ehemaligen Segway PT, darunter auch Spezialversionen für Militär und für Blaulichtorganisationen. [19] Der Fahrer bewegt den Segway vorwärts oder rückwärts, indem er sich jeweils in die gewünschte Richtung lehnt und somit seinen Schwerpunkt verschiebt. Der verschobene Schwerpunkt würde den Segway aus dem Gleichgewicht bringen und umfallen lassen. Somit müssen sich die Räder ebenfalls in diese Richtung bewegen, um wieder senkrecht unter den Schwerpunkt des Systems zu kommen. Da der Segway dies nicht vollständig erreicht, entwickelt

[1] Quelle: GHW Bergisch-Land; *Segway Pt – Modelle*; URL: http://www.ghw-bergisch-land.de/segway-pt/modelle/index.php (besucht am 28.06.2014)

Abbildung 2.1: Segway PT i2[1]

sich eine Bewegung in die gewünschte Richtung. Um eine Kurve zu fahren, wird die Lenkstange nach rechts oder links gekippt, worauf der kurveninnere Motor langsamer und der äussere Motor schneller wird.
Der Segway PT i2, wie er in Abbildung 2.1 zu sehen ist, erreicht Geschwindigkeiten bis zu 20km/h und hat eine Reichweite von bis zu 40 km. Er kann eine einzelne Person transportieren und wird sehr häufig für Stadttouren vermietet. Der i2 ist in der Schweiz für die Strasse zugelassen, sofern er mit dem Swiss Road Kit ausgerüstet wurde. Der Preis eines Segways bewegt sich in der Grössenordnung von 5'000 Schweizer Franken, jedoch sind Unterhalts- und Betriebskosten mit knapp 10 Rappen für eine vollständige Akkuladung sehr tief. [25]

2.2 Ryno

Ryno Motors wurde von Chris Hoffmann in Portland, Oregon gegründet. Seine Idee entstand, als seine Tochter ihn fragte, ob es möglich sei ein futuristisches Motorrad mit nur einem Rad zu bauen, wie sie es in einem Videospiel gesehen hatte. Hoffmann brauchte insgesamt sechs Jahre, um drei Prototypen zu konstruieren. Die Realisierung wurde jedoch erst durch die Mitarbeit von Tony Ozrelic ermöglicht, der imstande war die Software für das Fahrzeug zu schreiben. Der Name des Ryno entstammt ebenfalls dem

Videogame, wurde jedoch zu einem späteren Zeitpunkt durch das Akronym *Ride Your New Opportunity (dt.: Fahre deine neue Möglichkeit)* ergänzt, welches den derzeitigen Werbespruch darstellt.
Das Serienmodell des Ryno wird ab Ende 2014 erhältlich sein und soll umgerechnet rund 4'800 Schweizer Franken kosten.

Abbildung 2.2: Das elektronisch betriebene Einrad Ryno[2]

Der Ryno, wie er in Abbildung 2.2 zu sehen ist, kann auf Fahrradstreifen und in Fussgängerzonen gefahren werden und erreicht durch seine Geschwindigkeit von bis zu 16 km/h und seine Reichweite von 16 Kilometern jeden Ort, der auch zu Fuss erreicht werden kann. Er wiegt ohne Batterien ungefähr 73 Kilogramm und kann bis zu 118 Kilogramm befördern. Der Ryno wird in der Standardversion durch schwere 12 V-Bleibatterien betrieben, ist mittlerweile aber auch – gegen einen Aufpreis – mit Lithium-Ionen-Akkus erhältlich. Die Ladezeit für die Bleibatterien beträgt dabei ungefähr sechs Stunden.

Sowohl Motor als auch Batterien sind im Innern des Rades verbaut. Zudem verfügt der Ryno über einen verstellbaren Sitz, einen 12 V-Zigarettenanzünder um ein Smartphone aufzuladen, eine Federung und vorne über eine Stossstange, die zugleich als Parkständer dient. [23]

[2]Quelle: Ryno Motors; *Ryno*; URL: http://rynomotors.com/wp-content/uploads/2013/04/RYNO12.jpg (besucht am 28.06.2014)

2.3 Verwendete Hardware

In diesem Kapitel wird die verwendete Hardware vorgestellt, die für den Bau des selbstbalancierenden Roboters benötigt wird. Es werden sowohl Funktionen als auch Funktionsweise der wichtigsten Bauteile erklärt und die Gründe für deren Integration in den Roboter erörtert.

2.3.1 Arduino Board

Der Arduino in der Ausführung Uno R3 (siehe Abbildung 2.3), wie wir ihn für unsere Arbeit gebrauchen, ist ein Mikrocontroller-Board, basierend auf dem Mikrocontroller ATmega328 von Atmel. Hergestellt wird er in Italien durch Smart Projects. Der Uno ist die einfachste Ausführung eines Arduino. [3, Arduino Uno]

> Ein Mikrocontroller besteht aus einem Mikroprozessor und Zusatzmodulen. Prozessor und Module befinden sich auf einem einzigen Halbleiterchip. Module können beispielsweise ein USB-Anschluss oder digitale bzw. analoge Ein- oder Ausgänge sowie weitere Controller sein. Mikrocontroller finden in unzähligen alltäglichen Dingen Anwendung: In Autos, Smartphones, Fernsehern, Uhren u.v.m. [28]

Allgemeine Daten

Mikrocontroller	ATmega328
Betriebsspannung	5 V
Eingangsspannung (empfohlen)	7-12 V
Eingangsspannung (min.-max.)	6-20 V
Digitale Pins (Eingang/Ausgang)	14
Analoge Input Pins	6
Strom (DC; max.) auf I/O-Pins	40 mA
Flash Speicher	32 KB (ATmega328)
Taktrate Mikrocontroller	16 MHz
Abmessungen	7 cm × 5 cm × 1 cm

Hardware

Das Arduino-Board verfügt neben dem wechselbaren Mikrocontroller über eine Vielzahl von Modulen:

USB-Anschluss Der Arduino Uno hat einen eingebauten USB-2.0-B-Anschluss. Dieser dient in erster Linie der Kommunikation mit dem Computer, z.B. um Maschinencode (kompilierte Sketches) auf den Arduino hochzuladen oder Werte auszulesen, also serielle Werte zu übertragen. Solange der Arduino an einen Computer angeschlossen ist, dient ihm dieser als Spannungsquelle. Der Arduino nutzt dabei die 5 V-Standardspannung des USBs und zieht maximal 500 mA Strom.

Stromanschluss Gleich neben dem USB-Anschluss, auf derselben Seite, befindet sich der Stromanschluss, um den Arduino unabhängig von USB und somit vom Computer zu betreiben. So kann das Board einerseits mit einem Gleichstromnetzteil, andererseits mit einer Batterie betrieben werden. Beim Betrieb durch den Stromanschluss sollte darauf geachtet werden, den Arduino mit einer Spannung von 7 bis 12 V zu speisen, obwohl nach Daten 6 bis 20 V möglich wären. Beim Betrieb mit 6 oder 13 bis 20 V, läuft man Gefahr, dass das Board beschädigt wird.

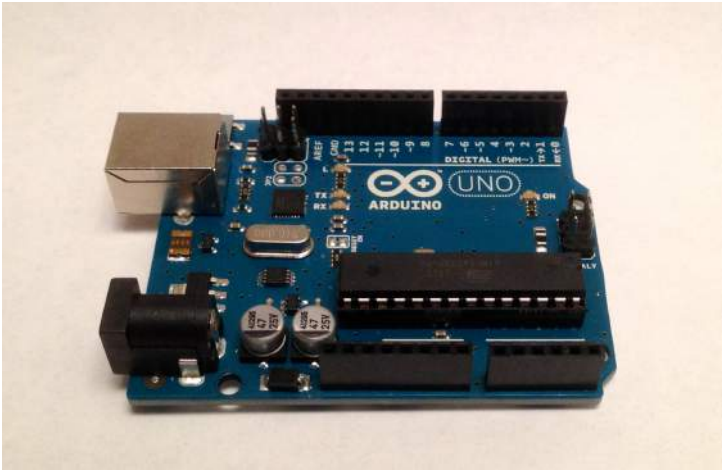

Abbildung 2.3: Arduino Board Uno R3

Reset-Knopf Direkt neben dem USB-Anschluss befindet sich der Reset-Knopf. Wird er gedrückt, startet der Arduino neu.

Digitale Pins Das Arduino-Board hat 14 digitale Pins, die von 0 bis 13 nummeriert sind. Es kann jeder Pin mit dem Befehl *pinMode()* entweder als Ein- oder als Ausgang definiert werden, die Argumente dafür sind entweder *INPUT* oder *OUTPUT*. Ist

ein Pin als Ausgang definiert worden, kann per Befehl *digitalWrite()* entweder eine Spannung von 5 (HIGH) oder 0 V (LOW) bei maximal 40 mA ausgegeben werden. Wurde ein Pin als Eingang definiert, kann mit dem Befehl *digitalRead* ausgelesen werden, ob den Pin entweder 5 oder 0 V erreichen.

Einige der digitalen Pins haben neben den grundlegenden Funktionen, über die jeder digitale Pin verfügt, noch weitere Spezialfunktionen. So können die Pins 0 und 1 serielle Daten empfangen oder senden und die Pins 2 und 3 bei leichten Spannungsanstieg oder -abfall sowie Spannungsänderungen eine Unterbrechung im laufenden Programm auslösen. Das ist dort wichtig, wo spezifische Werte von Bedeutung abgewartet werden müssen; die Pins 3, 5, 6, 9, 10, und 11 verfügen über Pulsweitenmodulation (PWM), welche in Abschnitt 2.3.4 genauer beschrieben wird. Diese Pins sind mit einer Tilde (∼) gekennzeichnet. Die Pins 10, 11, 12 und 13 werden benötigt, wenn mit anderen Geräten kommuniziert werden muss, und Pin 13 ist zusätzlich noch mit einer im Board integrierten LED verbunden, so dass man immer sieht, ob auf dem Pin gerade 5 oder 0 V ausgegeben werden.

Analoge Pins Zum Arduino gehören auch noch sechs analoge Pins, welche als A0 bis A5 bezeichnet werden. Die Standardaufgabe der analogen Pins ist es, bei einer eingehenden Spannung den Unterschied zu einer Referenzspannung zu messen. Da die Referenzspannung im Normalfall 0 V ist, messen die Pins die Spannung, die auf den Pin abgegeben wird.

Auch bei den analogen Pins gibt es welche mit Sonderfunktionen. *A4* und *A5*, auch *SDA*- und *SCL*-Pin genannt, werden für eine Kommunikation mit anderen Geräten benötigt. Beispielsweise wird auch der von uns benötigte Gyrosensor an diese beiden Pins angeschlossen, deren Funktion in Abschnitt 2.3.2 (Der Datenbus I^2C) genauer erörtert wird.

Weitere Pins Der Arduino verfügt noch über zwei weitere Pins: *AREF* und den Reset-Pin. An *AREF* liegt die Referenzspannung für die analogen Pins, und der Reset-Pin bewirkt, dass der Arduino neu gestartet wird, sobald den Pin 0 V erreichen. Der Reset-Pin wird vor allem benötigt, wenn der Reset-Knopf auf dem Board nicht erreichbar ist. [3, Arduino Uno]

2.3.2 Was ist eine IMU?

In diesem Kapitel wird erklärt, was eine Inertial Measurement Unit (IMU) ist, wie sie funktioniert und wo sie im Alltag eingesetzt wird. Zusätzlich wird die verwendete MPU-6050 vorgestellt, eine MEMS (Microelectromechanical systems) IMU, die von verschiedenen Produzenten auf dem Markt angeboten wird und in diesem Projekt Anwendung findet.

Inertialsensoren und die IMU

Inertialsensoren sind Sensoren zur Bestimmung von Drehraten und Beschleunigungen. Der Begriff entstammt der Funktionsweise der beiden Sensoren, die auf der Nutzung der Trägheit (lat. inertia) beruht. Eine IMU ist eine Kombination mehrerer Inertialsensoren, typischerweise drei orthogonal angeordneter Beschleunigungssensoren oder drei orthogonal angeordneter Gyrosensoren, welche die Winkelgeschwindigkeit messen. Orthogonal bedeutet, dass die Richtungsvektoren aller drei Sensoren jeweils das Skalarprodukt 0 ergeben, sie stehen somit alle senkrecht aufeinander. Eine beschreibende Skizze zeigt die Abbildung 2.4. Durch die doppelte Integration der Beschleunigung ergibt sich die Translationsbewegung und die durch die einmalige Integration der Drehgeschwindigkeit ergibt sich die Rotationsbewegung. Dieses Prinzip kann für die Trägheitsnavigation verwendet werden, wie sie im Abschnitt *Anwendungen von IMUs* beschrieben wird. [9]

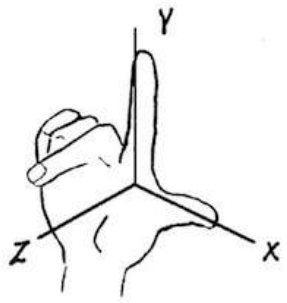

Abbildung 2.4: Räumliche orthogonale Anordnung von Fingern[3]

[3]Quelle: Improbable Research; *Orthogonal judged interesting*; URL: http://www.improbable.com/2010/06/21/orthogonal-judged-interesting/ (besucht am 08.08.2014)

Freiheitsgrade

Ein im Raum freier starrer Körper besitzt drei Freiheitsgrade der Translation (gleiche Verschiebung aller Massepunkte) und drei Freiheitsgrade der Rotation (Drehung entlang einer Achse). [7, Freiheitsgrad]
Der MPU-6050 kann mit dem Beschleunigungssensor die linearen Beschleunigungen messen und verfügt somit über Informationen zur Translation. Mit dem Gyro-Sensor ist es ihm zudem auch möglich die Drehgeschwindigkeit zu messen, er verfügt dadurch auch über Informationen zur Drehbewegung. Da beide Sensoren im dreidimensionalen Raum arbeiten (drei Achsen), spricht man beim MPU-6050 von einer 6 degrees of freedom (engl: Freiheitsgrade) IMU. Andere Sensoren verfügen zusätzlich beispielsweise über einen 3-Achsen Kompass oder über einen barometrischen Drucksensor (eine Achse für die Höhe) um die Navigation zu erleichtern und die Position genauer zu bestimmen. [20]

Beschleunigungssensor

Beschleunigungssensoren verwenden verschiedene Techniken um die Beschleunigung zu messen. In dieser Arbeit wird nicht auf alle Methoden eingegangen. Die verschiedenen Methoden beruhen alle auf der Trägheit der Masse, welche an deformierbaren oder beweglichen Materialen befestigt ist. Mikrosysteme, wie sie in der MPU-6050 verwendet werden, beruhen meist auf Änderungen der elektrischen Kapazität eines Bauteils, das starke Ähnlichkeit mit einem Plattenkondensator aufweist. Die Kapazität eines Plattenkondensators kann mit folgender Formel berechnet werden:

$$C = \varepsilon \frac{A}{d} \tag{2.1}$$

ε bezeichnet die Permittivität, welche die Durchlässigkeit des Materials für elektrische Felder beschreibt. Im Vakuum entspricht ε der elektrischen Feldkonstante ε_0. A ist der Flächeninhalt einer Platte des Kondensators und d beschreibt den Abstand der Platten.

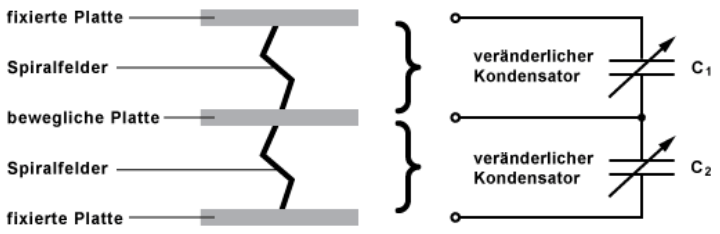

Abbildung 2.5: Plattenkondensator in einem Beschleunigungssensor[4]

Die Kapazität eines Plattenkondensators ist somit umgekehrt proportional zum Abstand der Platten. Eine Kondensatorplatte, die auf einer Art Feder gelagert ist, verschiebt sich anhand von Kräften, welche durch die Beschleunigungen hervorgerufen werden. Diese Beschleunigungen können als Änderung der Kapazität des Kondensators gemessen werden. In Abbildung 2.5 ist ein solcher Plattenkondensator zu sehen. Eine Besonderheit gegenüber einem herkömmlichen Kondensator stellt die Konstruktion mit drei Platten dar. Dadurch können Beschleunigungen in beide Richtungen gemessen werden. [8]

Gyrosensor

Das Prinzip des Gyrosensors ist dem Beschleunigungssensor sehr ähnlich. Im Gegensatz zum Beschleunigungssensor misst er nicht die Linearbeschleunigung sondern die durch die Zentripetalkraft verursachte Beschleunigung, welche bei einer Drehbewegung auftritt. Das Wort Gyro ist griechischen Ursprungs und bedeutet *Runde* oder *Drehung*. [21]

Anwendungen von IMUs

Inertial Measurement Units werden für den Privatgebrauch vor allem in der Robotik und in Smartphones und Digitalkameras eingesetzt, wovon sie bei letzteren hauptsächlich eine Rolle bei der Bildstabilisierung spielt. IMU's werden zusätzlich auch für die Navigation in der Luft- und Raumfahrt angewendet. Sie sind somit neben der zivilen Anwendung auch wichtige Bestandteile von militärischen Marschflugkörpern oder von Langstreckenraketen.

Durch ständige Messung der Beschleunigungen lässt sich die Position des Luftfahrzeuges gegenüber einer bekannten Position bestimmen. Dafür werden in einem Bordrechner die Beschleunigungen laufend integriert, so erhält man zuerst die Geschwindigkeit und dann die zurückgelegte Distanz. Aufgrund von Ungenauigkeiten der Messungen und den Integrationsfehlern ist die Trägheitsnavigation ungeeignet, falls sie nicht in Kombination mit anderen Systemen wie Funknavigation oder ähnlichem verwendet wird. [7, Trägheitsnavigation]
Das gleiche gilt für Navigationssysteme im Automobilbereich. Für Tunnelfahrten verfügen moderne Navigationssysteme ebenfalls über die Fähigkeit zur Trägheitsnavigation, weil dort kein Satellitensignal empfangen werden kann. [9]

[4]Quelle: Elektronik Kompendium; *Veränderlicher Kondensator – Beschleunigungssensor*; URL: http://www.elektronik-kompendium.de/sites/bau/bilder/15030411.gif (besucht am 2.08.2014)

Sparkfun MPU-6050

Die MPU-6050, wie sie in Abbildung 2.6 gezeigt wird, vereint einen 3-Achsen Beschleunigungssensor, einen 3-Achsen-Gyro-Sensor und ein Digital-Motion-Processor (DMP) auf einem Board. Es können damit sowohl Beschleunigungswerte als auch Rotationsgeschwindigkeit im Rohformat ausgegeben werden, oder der DMP kann sie in verschiedene Formate umrechnen und ausgeben. Ausserdem kann der DMP aus den beiden Sensoren Quaternionen berechnen, um Eulersche Winkel auszugeben, welche die Lage des Sensors im Raum zeigen. [13] Quaternionen sind ein Zahlenbereich, der die komplexen Zahlen erweitert. [1] In dieser Arbeit wird jedoch nicht auf deren genaue Funktionsweise eingegangen.

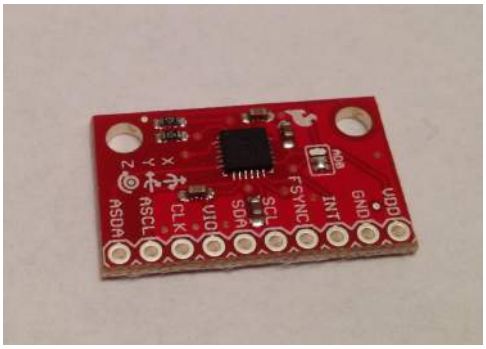

Abbildung 2.6: Die IMU MPU6050 von Sparkfun

Der hier verwendete Chip wurde durch Sparkfun verbaut, es gibt jedoch etliche andere Anbieter wie beispielsweise Invensense, die den MPU-6050 auf ihren Boards verbauen. Ebenfalls interessant ist der MPU-9150, der neben dem MPU-6050 auch noch einen 3-Achsen-Kompass enthält, mit dem auch eine Orientierung anhand der Himmelsrichtungen oder anhand von Magneten möglich ist. [20]

Eulersche Winkel Die Eulerschen Winkel wurden von Leonard Euler (1707–1783) eingeführt, um die Bewegung des Kreisels zu behandeln. Sie bestehen aus drei Winkeln, welche die gegenseitige Lage zweier Koordinatensysteme mit identischem Ursprung beschreiben. Eine beschreibende Grafik anhand eines Flugzeuges kann unter folgendem Link betrachet werden:

Wikipedia – Eulersche Winkel:
`http://de.wikipedia.org/wiki/Eulersche_Winkel#mediaviewer/Datei:Plane.svg`

Der Datenbus I²C Die Kommunikation mit dem MPU-6050 wird über den Zweidraht-Datenbus I²C sichergestellt, welcher aus der Taktleitung (Serial Clock Line – SCL) und der Datenleitung (Serial Data Line – SDA) besteht. I²C wurde in den frühen 1980er Jahren von Philips entwickelt. Die Vorteile einer seriellen Datenübertragung sind vor allem die wenigen Verbindungen, die im Vergleich zur parallelen Datenübertragung gemacht werden müssen, um verschiedene integrierte Schaltkreise (engl. integrated circuits – IC) zusammenzufügen. Dies führte nicht nur dazu, dass die Platinen kleiner wurden, sondern auch günstiger zu produzieren waren. [26]

I²C arbeitet nach dem Prinzip der *bidirektionalen Master/Slave-Architektur*. Alle *Slave*-Komponenten stellen auf Anforderung der *Master*-Komponente ihre Daten frei. Damit die Kommunikation funktioniert, müssen alle Komponenten über bestimmte Adressen verfügen, die von der *Master*-Komponente verliehen werden. [5, S. 489 – 491]

2.3.3 Logic Level Converter Bi-Directional

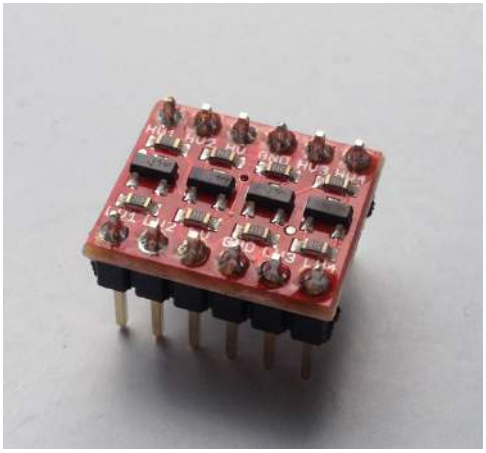

Abbildung 2.7: Logic Level Converter Bi-Directional von Sparkfun

Da der MPU-6050 mit einer Spannung von 3.3 V betrieben wird, das Arduino Board jedoch nicht zuverlässig alle 3.3 V-Signale als ein *HIGH*-Input erkennt, ist es zu empfehlen Zweifelsfälle zu vermeiden und einen Levelshifter zu benutzen. Die 3.3 V-Signale können beispielsweise mit einem Logic Level Converter auf 5 V angehoben werden. Der in Abbildung 2.7 gezeigte Logic Level Converter von Sparkfun ermöglicht es, zwei Systeme, die mit verschiedenen Spannungen betrieben werden, zu verbinden. Er funktioniert in beide Richtungen, also auch von 5 V auf 3.3 V. Deswegen wird er als *bidirektional* bezeichnet.

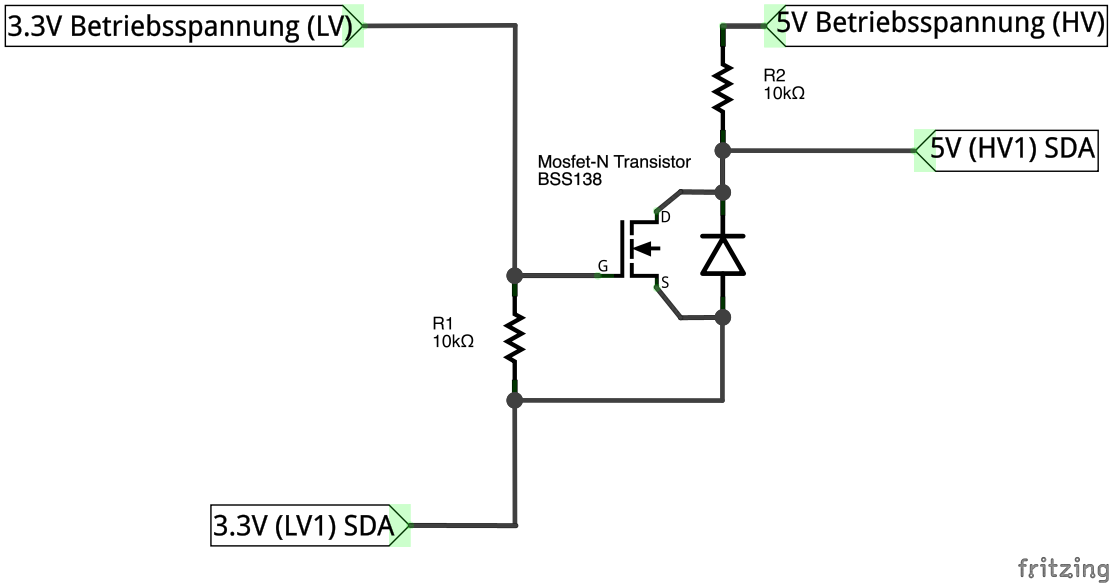

Abbildung 2.8: Schaltplan des Logic Level Converter Bi-Directional an Pin 1

Anwendung

Der Level Converter besitzt auf beiden Seiten jeweils sechs Pins, wobei die eine Seite unter der tieferen Spannung läuft und die andere unter der höheren. Es können bis zu 4 Datenleitungen tieferer Betriebsspannung mit 4 Datenleitungen höherer Betriebsspannung in beide Richtungen verbunden werden. Die *HV* und *LV* Pins müssen mit den jeweiligen Betriebsspannungen verbunden werden und die beiden *GND* Pins mit der Masse des verwendeten Mikrocontrollers. [12]

Funktionsweise

Ein Plan für die Schaltung einer einzigen Datenleitung kann in Abbildung 2.8 angesehen werden. In dieser Arbeit wird nicht auf die genaue Funktionsweise eingegangen. Bei besonderem Interesse wird an diesem Punkt auf die auf der Adafruit Website verfügbaren Datenblätter verwiesen. Das System wurde ehemals von Philips entwickelt: 4-channel I2C-safe Bi-directional Logic Level Converter: `http://www.adafruit.com/products/757`

2.3.4 Servo

Um den bei den ersten Prototypen dieser Arbeit verwendeten Antrieb zu verstehen, werden in diesem Kapitel der klassische Analogservo und der hier verwendete Rotationsservo erklärt. Der Fokus liegt dabei in der Ansteuerung und der Technik eines Servos.

Analogservo

Analoge Servos werden per Pulsweitenmodulation angesteuert. Die Periode liegt im Normalfall zwischen 18 und 25 ms und die Pulsweite zwischen 1 und 2 ms (Mittelstellung = 1.5 ms). Bei kürzeren Signalen bewegt er sich in die eine Richtung und bei längeren in die entgegengesetzte. Normalerweise hat ein Servo einen Umfang von 180°, das heisst jeweils 90° nach rechts und nach links.

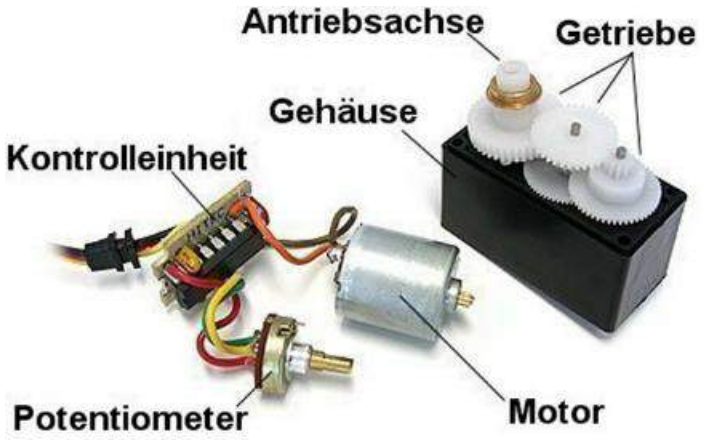

Abbildung 2.9: Bestandteile eines Servos[5]

Im Gehäuse eines Servos befindet sich entweder ein Wechselstrom- oder ein Gleichstrommotor, der dazu dient, den Servo in die Position zu drehen. Um die Stellungsposition zu bestimmen, befindet sich unter dem letzten Zahnrad, das direkt mit dem Hebel des Servos verbunden ist, ein Potentiometer (siehe Abbildung 2.9). Ein Potentiometer verstellt seinen inneren Widerstand abhängig von der Drehrichtung. Die Steuerelektronik des Servos kann somit den aktuellen Winkel des Hebels bestimmen. Ein Servo dreht sich dadurch so lange bis er den ihm übergebenen Winkel erreicht hat. Um ein Überdrehen zu verhindern befindet sich im Normalfall in beide Richtungen eine mechanische Sperre. Wird diese Sperre sowie das Potentiometer entfernt, entsteht ein Motor, der sehr einfach

[5]Quelle: Robotrack; *Das Servo und seine Technik*; URL: http://www.robotrack.org/ (besucht am 14.03.2014)

via Pulsweitenmodulation in beide Richtungen ansteuerbar ist. Ein Analogservo verfügt über drei Anschlüsse: Die Versorgungsspannung, die Masse und die Steuerleitung, über welche die Pulsweitenmodulation übertragen wird. [24]

Rotationsservo

Ein Rotationsservo (siehe Abbildung 2.10) lässt sich auf dieselbe Weise ansteuern wie ein normaler Analogservo. Die einzigen Unterschiede bestehen darin, dass er keine mechanischen Sperren enthält, die eine vollständige Drehung behindern und dass kein Potentiometer vorhanden ist. Anstatt des Potentiometers wird ein Rotationsencoder verwendet, wie er in 2.3.5 vorgestellt wird. Die Elektronik des Servos kann dadurch die Geschwindigkeit sehr exakt feststellen.

Abbildung 2.10: Fitec FS5106R 360° Robot Servo[6]

Da bei kleinen Winkeldifferenzen bei einem Analogservo die Drehgeschwindigkeit verringert ist, rotiert der Rotationsservo bei kleinen Winkelabweichung ebenfalls nur langsam. Jedoch spielt es ab einem gewissen Winkel keine Rolle mehr. Er dreht sich dann ohnehin mit der höchsten Geschwindigkeit. Beispielsweise gibt es keinen Geschwindigkeitsunterschied zwischen 60° und 70°, jedoch zwischen 10° und 20°. [10]

[6] Quelle: boxtec onlineshop; *Fitec FS5106R 360° Robot Servo (JR interface)*; URL: https://shop.boxtec.ch/fitec-fs5106r-360-robot-servo-interface-p-40730.html?language=de (besucht am 07.07.2014)

2.3.5 Bürstenloser DC-Motor

Bei den beiden letzten Prototypen wurden zwei Gleichstrommotoren mit Encodern verwendet. Die Funktionsweise der Motoren wird in diesem Kapitel erläutert.

Allgemeine Daten

Betriebsspannung	12 V
Eingangsspannung(min.-max.)	6-24 V
Strom(DC)	250 mA
Drehzahl	122 Rpm (ohne Last)
Drehmoment	0.85 Nm
Leistung	1,25 W
Besonderheit	Encoder [16]

Bürstenlose Motoren haben – im Gegensatz zu standardmässigen Gleichstrommotoren – die Wicklung der Kupferdrähte nicht in dem sich drehenden Teil des Motors, dem Rotor, sondern im Gehäuse, dem Stator. Sie sind ausserdem mehrphasig und haben mehrere Wicklungen. Bürstenlose Motoren brauchen auch zwingend eine Treiberschaltung, welche das Zusammenspiel der verschiedenen Wicklungen und den daraus entstehenden Magnetpolen koordiniert, also die Geschwindigkeit und Drehrichtung des Motors bestimmt. Normale Motoren hingegen funktionieren mit einem Plus- und einem Minuspol. Die Richtung wird dabei mit der Richtung des Stromflusses gegeben.

Eine Animation, die das Verhalten darstellt, kann unter folgendem Link abgerufen werden: `http://www.der-moba.de/images/9/99/Motor_bürstenlos.gif`

Ein bürstenloser Motor hat einige Vorteile gegenüber einem normalen Gleichstrommotor: Die bei einem Gleichstrommotor ständig auf dem Rotor "kratzenden" Bürsten, die einen Kontakt mit der Wicklung herstellen und ihn dadurch abnützen, fallen bei einem bürstenlosen Motor weg. Da sich die Wicklungen bei bürstenlosen Motoren im Gehäuse befinden, haben sie eine viel bessere Hitzeableitung und können dadurch ohne Kühlung länger auf höheren Geschwindigkeiten laufen als Gleichstrommotoren. Insgesamt ist ein bürstenloser Motor einem herkömmlichen Gleichstrommotor überlegen, jedoch kann auch sein Preis einiges höher liegen. [11]

Abbildung 2.11: Aslong JGA25 Gear Motor mit Encoder[7]

Der in dieser Arbeit verwendete bürstenlose Gleichstrommotor (siehe Abbildung 2.11) verfügt über eine integrierte Treiberschaltung. Dadurch kann er – genau wie ein normaler Gleichstrommotor – über zwei Kabel, den Plus- und den Minuspol angesteuert werden.

Encoder

Der bürstenlose Gleichstrommotor verfügt über einen Encoder (auf Deutsch auch *Kodierer*). Ein Encoder ist ein Messgerät, welches Umdrehungen, Drehzahl, Drehwinkel, Geschwindigkeit und Richtung eines Motors auslesen und kodieren kann. Encoder arbeiten entweder mechanisch, magnetisch oder optisch. [2]

Im Falle des JGA25 befindet sich direkt auf dem Rotor eine transparente Scheibe, auf der in regelmässigen Abständen eine Folie angebracht wurde, die ein Durchdringen des Lasers verhindert. Die Auflösung des Encoders konnte nicht in Erfahrung gebracht werden, jedoch wird sie durch das Getriebe noch einmal sehr stark erhöht, da pro Radumdrehung mehrere Rotorumdrehungen vollführt werden.

Verbindung zum Motor Controller Board und dem Arduino

Der Aslong JGA25 besitzt 6 Anschlüsse, die jeweils mit verschiedenen Kabelfarben gekennzeichnet sind: Das gelbe und das orange Kabel sind die Spannungseingänge und

[7]Quelle: Play-Zone GmbH; *Aslong JGA25 Gear Motor mit Encoder*; URL: http://www.play-zone.ch/de/ (besucht am 07.07.2014)

Ausgänge, je nachdem in welche Richtung der Motor drehen soll; das rote Kabel bezeichnet die Betriebsspannung für die Encoder und das grüne deren Masse; das schwarze und das weisse Kabel sind die Ausgangsleitungen der beiden Encoder.

2.3.6 Motor Controller Board

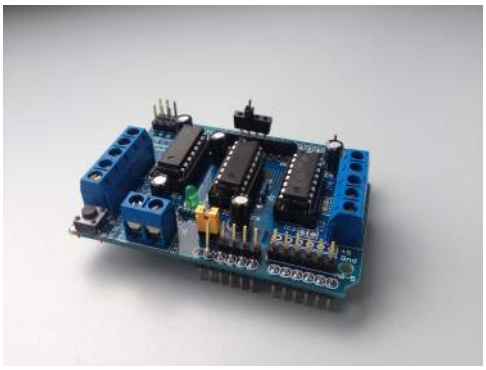

Abbildung 2.12: DK Motor/Stepper/Servo Shield für Arduino

Das DK Motor Shield eignet sich für das Ansteuern von bis zu vier DC-Motoren oder zwei Schrittmotoren. Es besitzt zwei L293D Motor-Controller, die jeweils zwei DC-Motoren in beide Richtungen, mit einer Auflösung von 8 bit mittels Pulsweitenmodulation, steuern können. Das DK Motor Shield ist ein Klon des Adafruit Motor Shield V.1, womit die selbe Library für die Ansteuerung verwendet werden kann, die Adafruit für ihr Shield bereitstellt. Der einzige Unterschied besteht darin, dass das DK Motor Shield nur Motoren bis 16V unterstützt, das originale Shield von Adafruit jedoch bis 25V. Für die in dieser Arbeit verwendeten Motoren, die mit einer Spannung von 12V betrieben werden, ist dies jedoch ausreichend.

Das DK Motor-Shield verfügt über Anschlüsse für bis zu zwei Servos und zwei Schrittmotoren oder bis zu vier DC-Motoren und zwei Servos. [17]

2.4 Theorie des Balancierens

In diesem Teil wird die Winkelgeschwindigkeit eines umfallenden Stabes in Abhängigkeit des Winkels berechnet, wenn dieser aus seiner stabilen Lage umfällt. Dies geschieht anhand der Energieerhaltung.

Berechnung der Winkelgeschwindigkeit

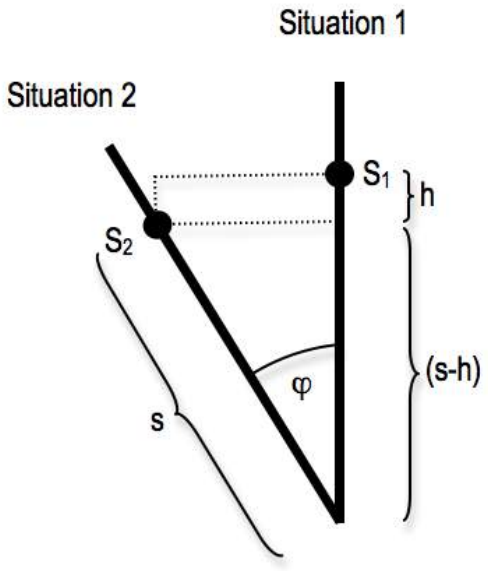

Abbildung 2.13: Skizze eines umfallenden Stabes mit Drehpunkt am Boden

In Abbildung 2.13 werden zwei Situationen eines umfallenden Stabes gezeigt. Es handelt sich bei dem Stab, genau wie bei einem balancierenden Roboter, um keinen Körper mit homogener Massenverteilung. Der Schwerpunkt S ist somit frei gewählt. Der Winkel φ bezeichnet die Winkeldifferenz zwischen Situation 1 und Situation 2. Die Strecke s ist der Abstand zwischen dem Drehpunkt und dem Schwerpunkt. Der Drehpunkt befindet sich auf der Oberfläche, auf welcher der Stab steht. Die Strecke h bezeichnet die Höhendifferenz des Schwerpunktes zwischen den beiden Situationen 1 und 2.

In Situation 1 – der Stab befindet sich im Gleichgewicht – besitzt der Schwerpunkt S_1 des Stabes potentielle Energie, jedoch keine Rotationsenergie.

$$E_{pot_1} = mgs \tag{2.2}$$

$$E_{rot_1} = 0 \tag{2.3}$$

Während des Fallens verliert der Stab an potentieller Energie und gewinnt dafür an Rotationsenergie. Aufgrund der Energieerhaltung müssen der Verlust an potentieller Energie und der Gewinn an Rotationsenergie identisch sein.

Die Gesamtenergie in Situation 2 stellt sich wie folgt dar, wobei Θ das Trägheitsmoment des Stabes darstellt und ω die Winkelgeschwindigkeit:

$$E_{pot_2} = mg(s-h) \tag{2.4}$$

$$E_{rot_2} = \frac{1}{2}\Theta\omega^2 \tag{2.5}$$

Der Winkel wird mithilfe von $(s-h)$ ausgedrückt:

$$\cos\varphi = \frac{(s-h)}{s} \tag{2.6}$$

$$s\cos\varphi = (s-h) \tag{2.7}$$

Nun wird in Situation 2 (2.4) eingesetzt:

$$E_{pot_2} = s\cos\varphi mg \tag{2.8}$$

$$E_{rot_2} = \frac{1}{2}\Theta\omega^2 \tag{2.9}$$

Aufgrund der bereits zuvor genannten Energieerhaltung müssen die Gesamtenergien der beiden Situationen gleich sein.

$$E_{pot_1} + E_{rot_1} = E_{pot_2} + E_{rot_2} \tag{2.10}$$

Es wird in 2.10 eingesetzt und da aus 2.3 klar ist, dass $E_{rot_1} = 0$, folgt:

$$mgs = mgs\cos\varphi + \frac{1}{2}\Theta\omega^2 \tag{2.11}$$

$$mgs - mgs\cos\varphi = \frac{1}{2}\Theta\omega^2 \tag{2.12}$$

$$mgs(1-\cos\varphi) = \frac{1}{2}\Theta\omega^2 \tag{2.13}$$

$$\omega^2 = \frac{2mgs(1-\cos\varphi)}{\Theta} \tag{2.14}$$

$$\omega = \sqrt{\frac{2mgs(1-\cos\varphi)}{\Theta}} \tag{2.15}$$

Der Term von E_{pot_2} wird auf die andere Seite gebracht, um den Term mgs ausklammern zu können. Nun wird nach ω aufgelöst, dazu wird mit zwei multipliziert und durch Θ dividiert, damit diese auf die andere Seite gelangen. Da in (2.14) ω nun noch quadratisch vorhanden ist, muss noch die Wurzel gezogen werden.

Aus (2.15) können nun alle Konstanten vor den Bruch genommen werden, damit der veränderliche Term gezeigt werden kann.

$$\omega = \sqrt{\frac{2mgs}{\Theta}}\sqrt{(1 - \cos\varphi)} \tag{2.16}$$

Als einziger veränderlicher Term bleibt folgender: $\sqrt{(1 - \cos\varphi)}$. Somit hängt die Funktion der Winkelgeschwindigkeit in Abhängigkeit des Winkels nur noch von diesem Term ab.

$$\omega(\varphi) = \sqrt{1 - \cos\varphi} \tag{2.17}$$

Interpretation des Ergebnisses

Aus diesem Ergebnis lässt sich sehr gut ableiten, wie der Roboter auf die beste Art gebaut werden sollte. Im konstanten Term $\sqrt{\frac{2mgs}{\Theta}}$ sind die gerätespezifischen Variablen der Masse und der Höhe des Schwerpunktes enthalten: Wenn diese Werte zunehmen, wird die Winkelgeschwindigkeit grösser. Ein idealer Roboter sollte somit möglichst leicht sein und über einen tiefen Schwerpunkt verfügen. Das Trägheitsmoment ist ebenfalls gerätespezifisch und steht unter dem Bruch. Somit sollte dieses idealerweise möglichst gross sein. Ein grosses Trägheitsmoment, trotz tiefem Schwerpunkt, ist nur möglich, wenn der Roboter möglichst breit konstruiert wird und die Masse gleichmässig so weit wie möglich nach aussen verlagert wird.

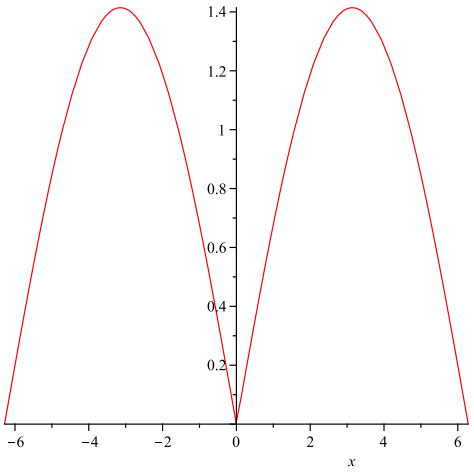

Abbildung 2.14: Winkelgeschwindigkeit eines Starrpendels von -2π bis 2π

In Abbildung 2.14 ist die Kurve der Winkelgeschwindigkeit zu sehen. Die Kurve zeigt die Entwicklung der Geschwindigkeit von -2π bis 2π, was heisst, dass jeweils in beide

Richtungen eine ganze Umdrehung um den Drehpunkt gezeigt wird und der Stab somit als Starrpendel angenommen wird. Für den selbstbalancierenden Roboter ist höchstens der Bereich von $-\frac{\pi}{2}$ bis $\frac{\pi}{2}$ entscheidend, weil der Roboter ansonsten bereits vollständig auf der Oberfläche liegt. Die Winkelgeschwindigkeit verändert sich in diesem Intervall in einem beinahe linearen Bereich. Noch kleiner sind die Änderungen der Winkelbeschleunigung in den Bereichen bis 20° (≈ 0.35 rad), wo die meisten Aktionen des Roboters stattfinden werden.

Der Seiltänzer mit der Balancierstange

Falls der Seiltänzer zur Balancierstange greift, versucht er sein Trägheitsmoment zu erhöhen, genau wie in Abschnitt 2.4 beschrieben. Selbstverständlich kann der Seiltänzer seine Masse nicht beeinflussen, jedoch hält er die Stange absichtlich so tief als möglich, um einen tiefen Schwerpunkt zu erreichen. [18]

Abbildung 2.15: Ein Seiltänzer balanciert mit Hilfe einer Balancierstange

2.5 Regelungstechnik

Der Roboter muss einen Regelalgorithmus ausführen, um einen Soll-Wert beizubehalten, der in diesem Falle ein bestimmter Winkel ist, der ihm als stabile Position übergeben wird. Einer der besten Regler zu diesem Zweck ist der PID-Regler, wie er in diesem Teil erklärt wird.

Die Differenz zwischen dem Ist-Wert I und dem Soll-Wert S wird als Regeldifferenz e bezeichnet. Im Falle des selbstbalancierenden Roboters wird diese durch eine Winkeldifferenz bestimmt, die sich aus dem aktuellen Winkel und dem stabilen Winkel ergibt.

$$I - S = e \tag{2.18}$$

2.5.1 P-Regler – Proportional-Glied

Der P-Regler wirkt proportional zur Regeldifferenz e.

$$P \propto e \tag{2.19}$$

Somit gilt für den Verstärkungsfaktor K_p:

$$P = e \cdot K_p \tag{2.20}$$

Der P-Regler multipliziert die Regeldifferenz, die er vom Sensor empfängt, mit seinem Verstärkungsfaktor und gibt den Output sofort an den Aktor weiter, der daraufhin eine bestimmte Funktion ausführt. Im Falle des Roboters werden die Sensordaten von der IMU, welche die Winkelabweichung zeigen, im Microcontroller mit dem Verstärkungsfaktor multipliziert und anschliessend an die Aktoren, die beiden DC-Motoren, weitergeleitet.

2.5.2 I-Regler – Integrator

Der I-Regler wirkt proportional zum Integral der Funktion der Regeldifferenzänderung.

$$I \propto \int_{t_0}^{t} e(t)dt \tag{2.21}$$

In Gleichung (2.21) bezeichnet t_0 einen Anfangszeitpunkt, t bezeichnet den jetzigen Zeitpunkt. Der I-Regler summiert bei jedem Durchlauf des Sketches die Regeldifferenz und wirkt proportional zu dieser Summe. Somit gilt für den Verstärkungsfaktor K_i:

$$I = \sum_{i=i}^{n} e_i \cdot K_i = (e_i + e_2 + e_3 + ... + e_n) \cdot K_i \tag{2.22}$$

In Gleichung (2.22) entspricht i dem Zähler der Durchläufe und n entspricht der Anzahl der Durchläufe, die während Δt, der Zeit zwischen t_0 und t, vergangen ist.

Im Falle des Roboters werden somit die Winkeldifferenzen bei jedem Durchlauf summiert. Dadurch agiert der I-Regler bei einer anhaltenden Regeldifferenz mit jedem Durchlauf des Sketches stärker und kann somit die Regeldifferenz weiter minimieren. Der Nachteil des I-Reglers ist seine Trägheit, deshalb wird er im Normalfall durch andere Regelglieder ergänzt.

2.5.3 D-Regelglied – Differential-Glied

Der D-Regler ist proportional zur Ableitung der Veränderung gegenüber des Soll-Wertes.

$$D \propto e'(t) \tag{2.23}$$

Das D-Regelungsglied multipliziert somit die Geschwindigkeit der Regeldifferenzänderung mit einem Verstärkungsfaktor. Unter Einbezug des Verstärkungsfaktors K_d gilt:

$$D = e'(t) \cdot K_d \tag{2.24}$$

Der D-Regler kann somit eine Art Prognose erstellen, was als nächstes passieren wird, da die Änderung der Regeldifferenz bei natürlichen Vorgängen immer dynamisch geschieht.

Im Falle des selbstbalancierenden Roboters ist die Ableitung des Winkels nach der Zeit genau die Winkelgeschwindigkeit $\dot{\varphi} = \omega$, die mit dem integrierten Gyrosensor gemessen werden kann. Aus diesem Grund sind dafür keine Berechnungen nötig und ω kann direkt mit dem Verstärkungsfaktor multipliziert werden.

$$D = \omega \cdot K_d \tag{2.25}$$

Der D-Regler kann nur zusammen mit anderen Regelungsgliedern eingesetzt werden, da er nur Zugriff auf die Ableitung der Regeldifferenzänderung hat, nicht aber auf die Regeldifferenz selbst. Er kennt somit weder den Soll– noch den Ist–Wert.

2.5.4 PID-Regler

Der PID-Regler vereint alle drei hier vorgestellten Regelglieder. Er ist sehr schnell und genau und gilt somit als einer der universellsten klassischen Regler. [27]

$$R = e \cdot K_p + \sum_{i=1}^{n} e_i \cdot K_i + e'(t) \cdot K_d \tag{2.26}$$

Da die Regeldifferenz einem Winkel entspricht, wird er nun als $\Delta\varphi$ bezeichnet. Die Ableitung des Winkels entspricht der Winkelgeschwindigkeit und wird mit ω bezeichnet.

$$R = \Delta\varphi \cdot K_p + \sum_{i=1}^{n} \Delta\varphi_i \cdot K_i + \omega \cdot K_d \tag{2.27}$$

3 Praktischer Teil

Im praktischen Teil dieser Arbeit wird auf das Kennenlernen und auf die spätere Evaluation der einzelnen Bauteile eingegangen. Er befasst sich ausserdem mit der Ansteuerung der Komponenten und dem Bau erster Prototypen. Im letzten Teil der Arbeit werden der fertige selbstbalancierende Roboter sowie der zusammengefügte Programmcode präsentiert.

3.1 Anwendung der einzelnen Bauteile

In diesem Teil werden Experimente durchgeführt, um die einzelnen Bauteile des Roboters zu verstehen, so dass sie schlussendlich zusammengefügt werden können. Der Fokus liegt auf der Programmierung sowie auf der Schaltung der Bauteile.

3.1.1 Ansteuerung zweier Rotations-Servos

In diesem Experiment wird die Ansteuerung zweier Servos anhand eines Schaltplans und eines Programmcodes erläutert. Die beiden Servos sollen in Gegenrichtung laufen und die Geschwindigkeit soll mit einem Potentiometer jederzeit anpassbar sein.

Schaltung

Die beiden verwendeten Rotationsservos werden mit 4.8 – 6 V betrieben, aber da sie mit grösster Wahrscheinlichkeit mehr Strom benötigen als der Arduino Uno liefern kann, ist es zu riskant, die Motoren über den Arduino zu speisen. Aus diesem Grund werden als Stromquelle vier in Serie geschaltete Mignon-Batterien (AA, 1.5 V) verwendet, wie sie in den Abbildungen 3.1 sowie 3.2 gezeigt werden.
Die Signalleitungen der beiden Servos werden an zwei digitale Pins angeschlossen, welche PWM unterstützen. Die beiden Versorgungsspannungen werden durch den Batterie-Block gespeist. Die Masse ist für die ganze Schaltung identisch. Somit spielt es keine

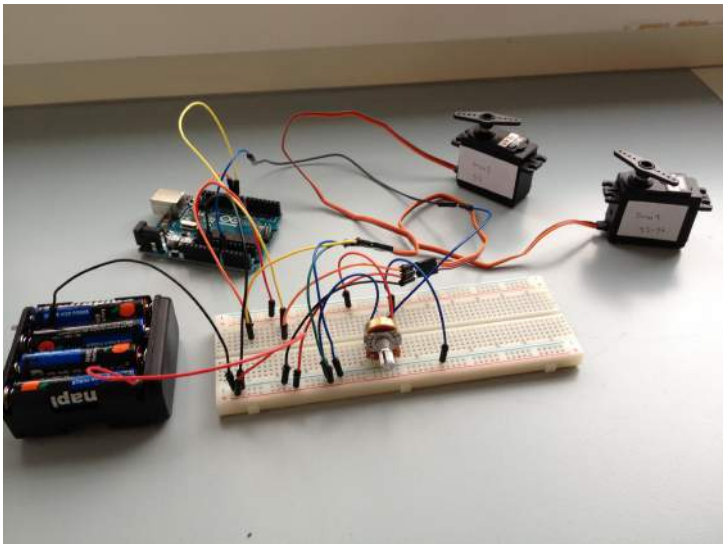

Abbildung 3.1: Ansteuerung zweier Rotationsservos

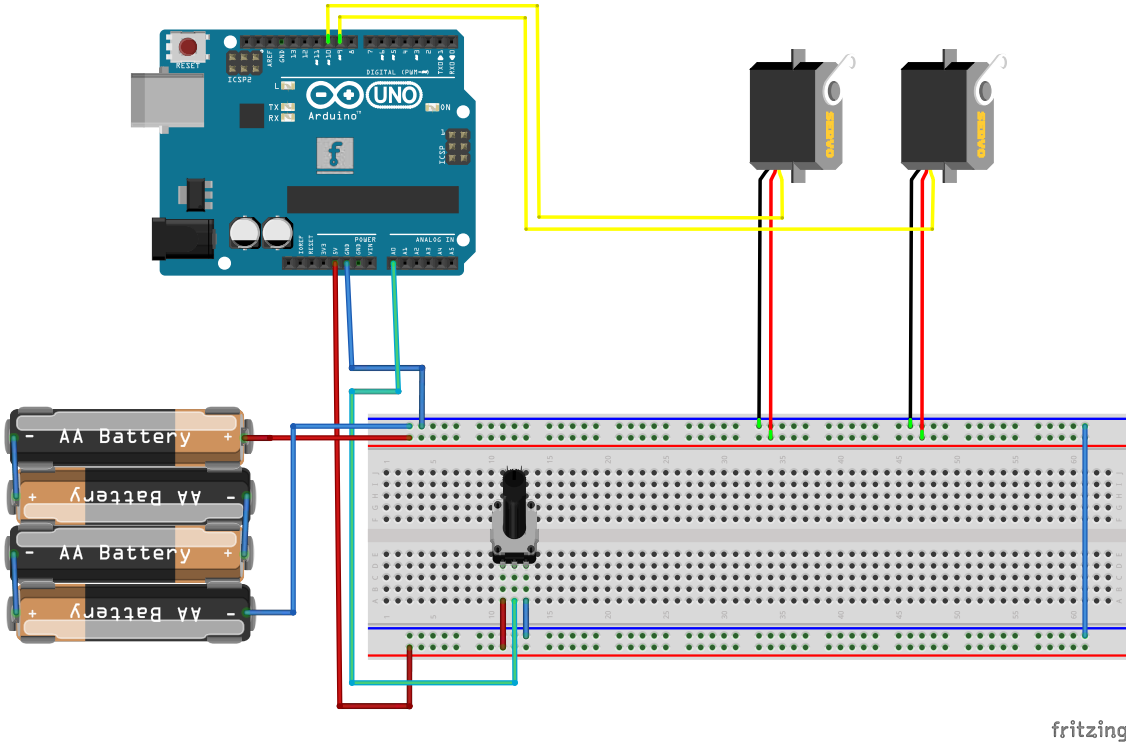

Abbildung 3.2: Ansteuerung zweier Rotationsservos – Fritzing

Rolle, wo die Servos an die Masse angeschlossen werden.
Das Potentiometer wird über die 5 V-Versorgung des Arduino gespeist, ist ebenfalls an die Masse angeschlossen und wird mit dem dritten, dem mittleren Anschluss mit einem analogen Pin des Arduino verbunden.

Programmierung

```
1  #include <Servo.h>
2
3    Servo Servo1; // erster Servo
4    Servo Servo2; // zweiter Servo
5    int potiPin = 0; // Variable fuer Nummer des analogen Pins
6    int potiWert; // Variable fuer Potentiometerwert
7
8  void setup(){
9
10   Servo1.attach(9); // Servo1 ist an PWM-Pin 9 angeschlossen
11   Servo2.attach(10); // Servo2 ist an PWM-Pin 10 angeschlossen
12
13   Serial.begin(9600); // Serielle Schnittstelle starten
14 }
15 void loop(){
16
17   potiWert = map(analogRead(potiPin), 0, 1023, 0, 179); // Potentiometerwert wird
18                                                         // ausgelesen und skaliert
19
20   Serial.println(potiWert); // skalierter Potentiometerwert wird auf
21   // Serial Monitor ausgegeben
22
23   Servo1.write(potiWert); // skalierter Wert wird an Servo1 weitergegeben
24   Servo2.write(179-potiWert); // skalierter Wert wird in entgegengesetzter
25   // Richtung an Servo2 weitergegeben
26
27   delay(20); // kurze Verzoegerung
28 }
```

In Zeile 1 wird die Servo-Library eingebunden. Diese ist bereits in der Arduino Entwicklungsumgebung vorinstalliert. In Zeilen 3 und 4 werden zwei Servos, mit Hilfe des Konstruktors der Servo-Library, erstellt. Die beiden Servos sind somit eingebunden und besitzen je eine eigene spezifische Bezeichnung. Die Variable für den analogen Pin, an welchen das Potentiometer angeschlossen wird, bekommt den Wert 0. Für den Potentiometerwert wird die Variable *potiWert* definiert.
In der Setup-Methode werden auf Zeile 10 und 11 die Ports definiert, an denen die Signalleitungen angeschlossen werden. In Zeile 13 wird die serielle Schnittstelle gestartet.
In der Loop-Methode wird jeweils als erstes, in der Zeile 17, der Potentiometerwert ausgelesen und gleich danach auf ein Intervall von 0 bis 179 (180 Werte) skaliert. Anschliessend wird auf Zeile 20 der Potentiometerwert auf dem Serial Monitor ausgegeben, so dass sich

das Verhalten der Motoren beobachten lässt. In Zeile 23 wird der skalierte Potentiometerwert an den ersten Servo gesendet. Die Library erlaubt das direkte Übergeben eines Gradwinkels. Beim Rotationsservo entspricht dies der Geschwindigkeit, wobei er sich bei 90°nicht bewegt. Um entgegengesetzte Drehrichtungen zu erreichen, muss der übergebene Wert beim zweiten Servo von 179 subtrahiert werden.

3.1.2 Ansteuerung zweier DC-Motoren

In diesem Experiment wird die Ansteuerung zweier DC-Motoren erläutert. Die DC-Motoren ersetzen die zu schwachen Rotationsservos und werden mit Hilfe des Adafruit Motor Shields angesteuert.

Schaltung

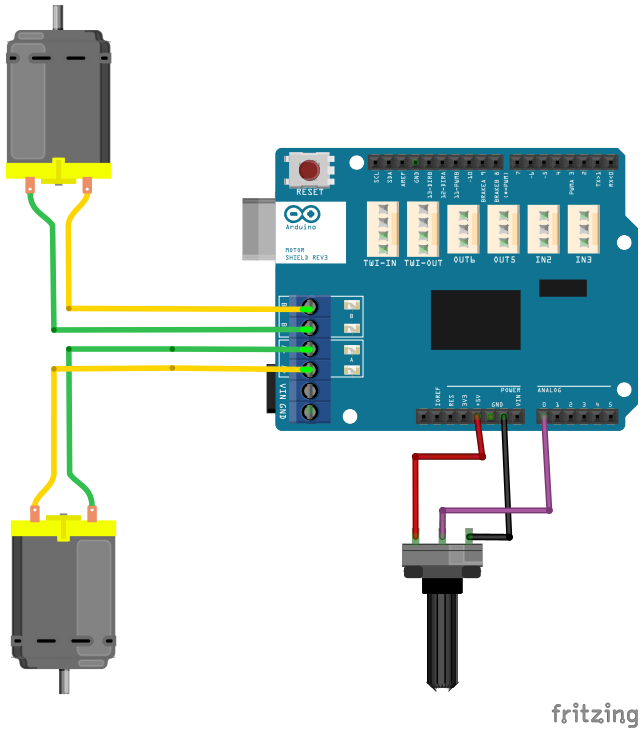

Abbildung 3.3: Ansteuerung zweier DC-Motoren

Das in Abbildung 3.3 gezeigte Motor-Shield ist nicht identisch mit dem in der Arbeit verwendeten Adafruit-Shield. Die Funktionsweise ist dieselbe, jedoch kann die Adafruit

Library nicht für das gezeigte Shield verwendet werden.

Die beiden DC-Motoren werden an die vorgesehenen beiden Ports angeschlossen. Zu beachten ist hierbei, dass sie im gegengleichen Sinn angeschlossen werden. Der Code wird damit vereinfacht, weil die Motoren ohnehin in entgegengesetzte Richtungen laufen sollen. Das Potentiometer wird an die 5 V-Versorgungsspannung, die Masse und an einen analogen Pin (in diesem Fall *A0*) angeschlossen.

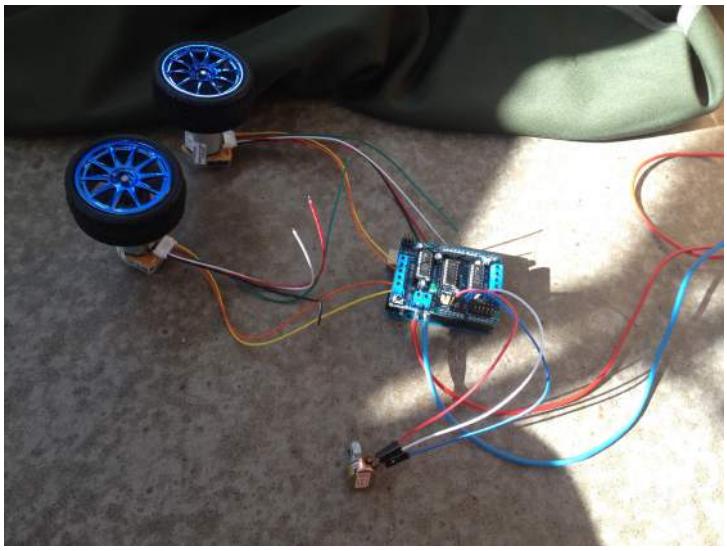

Abbildung 3.4: Ansteuerung zweier DC-Motoren mit dem Adafruit Motor Shield

Programmierung

Die Ansteuerung zweier DC-Motoren mithilfe des Adafruit Motor-Shield ist mit der dazu erhältlichen Library am einfachsten. Diese wird am Anfang des Sketches geladen und kann unter folgendem Link heruntergeladen werden:

Download Adafruit Motor Shield Library: `https://learn.adafruit.com/`

Diese Library wird in der ersten Zeile eingebunden. Es folgt der Konstruktor *AF_DCMotor*, der in Zeile 3 und 4 sowohl *motor1* sowie *motor2* erstellt. Diese beiden erhalten ein Argument, das den Port bezeichnet, an dem sie angeschlossen werden. Des Weiteren werden in den Zeilen 5 und 6, wie in Abschnitt 3.1.1, Variablen für den analogen Pin sowie den Potentiometerwert gesetzt. Danach wird noch eine Variable für die Geschwindigkeit definiert.

In der Setup-Methode werden die serielle Schnittstelle und die Motoren gestartet.

```
1  #include "AFMotor.h"
2
3  AF_DCMotor motor1(2); // Motor1 an Port 2 des Motor Shields
4  AF_DCMotor motor2(1); // Motor2 an Port 1 des Motor Shields
5  int analogPin = 0; // Analoger Pin fuer Potentiometerwert
6  int potiwert; // Variable fuer Aufnahme des Potentiometerwertes
7  int geschw;
8
9  void setup() {
10
11   Serial.begin(9600);          // Serielle Schnittstelle starten
12
13   motor1.setSpeed(200); // Motor1 einschalten
14   motor1.run(RELEASE);
15   motor2.setSpeed(200); // Motor2 einschalten
16   motor2.run(RELEASE);
17 }
18
19 void loop() {
20
21   potiwert = map(analogRead(analogPin), 0, 1023, -255, 255);
22   // Der Input des analogen Pins wird gelesen
23   // und von -255 bis 255 skaliert
24
25   geschw = map(abs(potiwert), 0, 255, 50, 255);
26   // Die Geschwindigkeit ist der absolute Betrag
27   // von potiwert und wird von 50 bis 255 skaliert
28
29   motor1.setSpeed(geschw);  // Geschwindigkeiten beider Motoren werden gesetzt
30   motor2.setSpeed(geschw);
31
32   if(potiwert < 0){// falls potiwert kleiner als Null ist fahren die Motoren vorwaerts
33     motor1.run(FORWARD);
34     motor2.run(FORWARD);
35   }
36   else{ // falls nicht, fahren sie rueckwaerts
37     motor1.run(BACKWARD);
38     motor2.run(BACKWARD);   }
39 }
```

In der Loop-Methode wird als erstes der Potentiometerwert skaliert. Dies ist nötig, weil die PWM des Arduino über eine Auflösung von 8-bit (256 Werte) verfügt. Somit sind auch die Motoren mit dieser Auflösung steuerbar. In Zeile 25 wird die Geschwindigkeit der Motoren definiert. Sie ist so gross wie der absolute Betrag des Potentiometerwerts, der auf ein Intervall von 50 bis 255 skaliert wurde. Dies ist nötig, weil die Motoren ihre eigene Trägheit – vor allem die des Getriebes – erst bei einem Wert von 50 überwinden und anfangen sich zu drehen. Der nächste Befehl *.setSpeed* gibt die Geschwindigkeit an die Motoren weiter. Nun ist die Geschwindigkeit zwar gesetzt, nur fehlt den Motoren noch eine Drehrichtung. Sie wird ihnen in Abhängigkeit des Vorzeichens des Potentiometerwertes in den Zeilen 32 bis 38 übertragen. Die Motoren erhalten jeweils die gleiche

Richtung, weil sie gegengleich an ihre Ports angeschlossen wurden. Somit drehen sie in entgegengesetzter Richtung.

3.1.3 Lesen der Sensordaten

Das korrekte Lesen der Sensorwerte ist für die Arbeit des Roboters entscheidend, weshalb diese durch den DMP verarbeitet werden müssen. Die Ansteuerung des DMP ist sehr anspruchsvoll, weswegen eine für den MPU-6050 geschriebene Library Anwendung findet. Der Code entstammt ebenfalls der Library, wurde jedoch geändert, um die für den PID-Regler wichtigen Daten lesen zu können.

Schaltung

Der MPU-6050 hat eine Betriebsspannung von 3.3 V, deshalb wird ein bidirektionaler Levelconverter benötigt, wie er in Abschnitt 2.3.3 beschrieben wurde. Die *Serial Clock Line (SCL)* wird über den Levelconverter mit dem analogen Pin *A5* verbunden und die *Serial Data Line (SDA)* mit dem analogen Pin *A4*. Dies ist nötig, weil die *Wire*-Library mit diesen zwei Pins arbeitet. [4] Der Interrupt-Pin *INT* wird ebenfalls über den Levelconverter mit dem digitalen Pin 2 des Arduino verbunden. Der Arduino Uno besitzt zwei externe Interrupts wovon der 0 Interrupt auf dem digitalen Pin 2 liegt. [3, attachInterrupt()] Die beiden *GND*-Pins am Levelconverter werden mit der Masse des Arduino verbunden, genau wie der des MPU-6050. VCC sowie der Low-Voltage-Pin *(LV)* am Levelconverter werden mit der 3.3 V-Spannung des Arduino verbunden und der High-Voltage-Pin *(HV)* mit der 5 V-Betriebsspannung.

Programmierung

Der in den folgenden Abschnitten beschriebene Quelltext kann bei Interesse in Abschnitt 5.1 eingesehen werden.

In den ersten drei Zeilen werden die drei Libraries eingebunden: *Wire.h* ist in der Arduino Entwicklungsumgebung bereits enthalten, *I2Cdev.h* und *MPU6050_6Axis_MotionApps20.h* werden von Jeff Rowberg zum Download angeboten:

I2C Device Library: `http://www.i2cdevlib.com/`

Sobald die Library installiert ist, können die zugehörigen Beispiele unter dem Reiter *Datei; Beispiele; MPU6050; Examples* gefunden werden. Um diese zu verändern, müssen

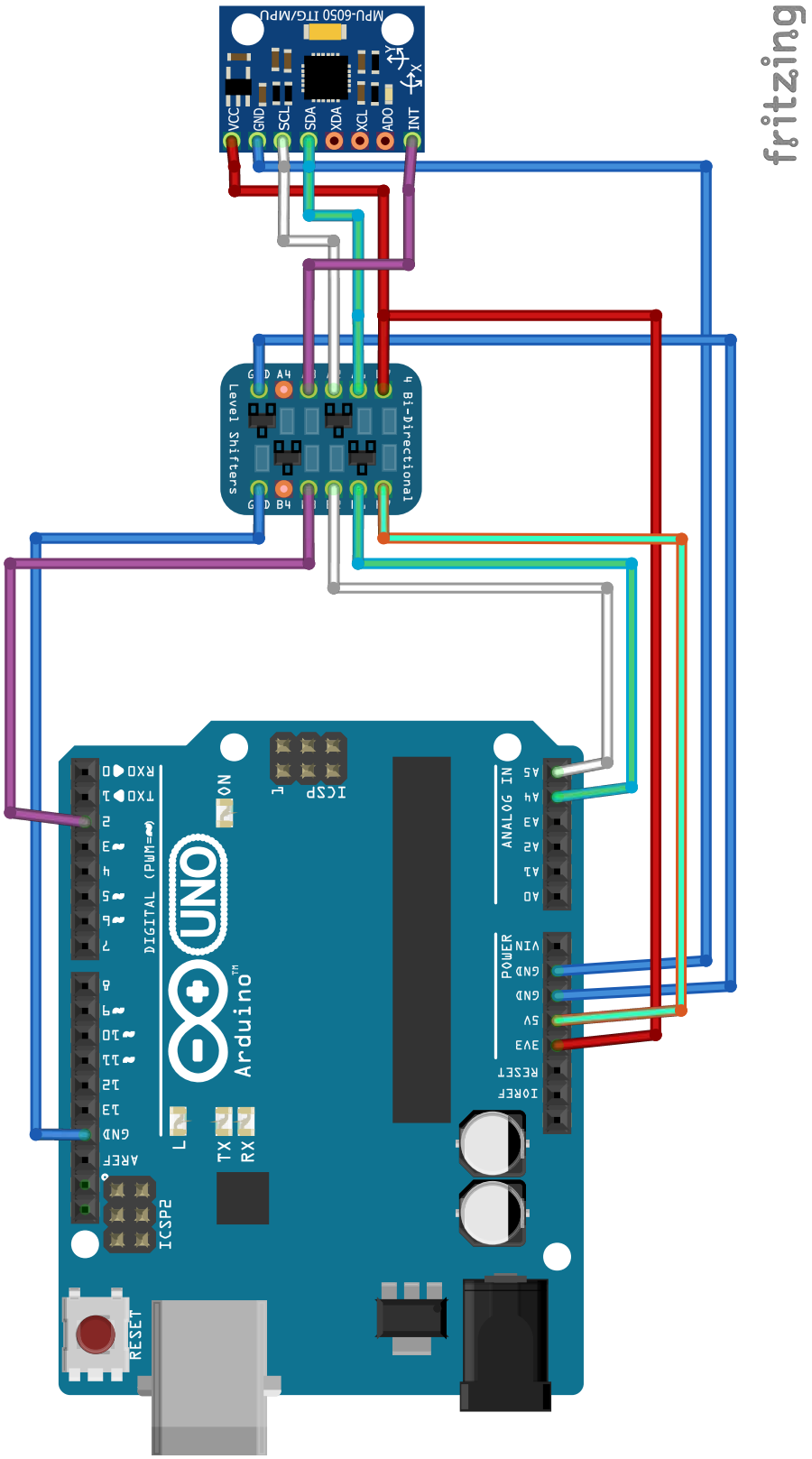

Abbildung 3.5: Schaltplan für das Lesen der Sensordaten aus der MPU-6050

sie in einem separaten Ordner neu abgespeichert werden. Das Listing, welches für unser Experiment verwendet wird, beeinhaltet nur einen bestimmten Teil des Beispiels *MPU-6050_DMP6*. Dabei wurde alles entfernt, was nicht für die Auslesung des Winkels in Gier-, Nick- und Rollachse (yaw, pitch & roll) benötigt wird.

Der PID-Regler benötigt zusätzlich zum Winkel auch die Winkelgeschwindigkeit. Aus diesem Grund wird in Zeile 23 ein Array des Typs *int16_t* (16 bit-Integer-Variable) mit dem Namen *gyros* erstellt. Um die Gyrowerte auch auszulesen wird in Zeile 99 folgender Befehl eingefügt: *mpu.dmpGetGyro(gyros, fifoBuffer);*. Jede Spalte des Arrays speichert Werte für eine ausgelesene Achse. In diesem Projekt wird nur die dritte – die Rollachse – verwendet. Diese steht an dritter und letzter Stelle im Array, und hat den Wert 2 (Arrays werden von 0 nummeriert).

3.2 Zusammenfügen der Bauteile

In diesem Teil wird schrittweise der Bau der verschiedenen Prototypen erklärt und wie für den fertigen Roboter der stabile Winkel sowie die PID-Werte bestimmt wurden, so dass dieser stabil balanciert.

3.2.1 Aufbau

Der Bau des Roboters wurde bei allen Prototypen und auch dem fertigen Roboter mit Fischertechnik realisiert. Alternativ hätte auch Lego verwendet werden können, oder es hätte eine Holzkonstruktion gebaut werden können. Die Holzkonstruktion bestand aus verschiedenen Plattformen für die Hardware, welche durch Gewindestangen und Muttern zu einem Gestell zusammengefügt waren. Die Vorteile einer Holzkonstruktion ist die Flexibilität bezüglich der Befestigung von Motoren und restlicher Elektronik, weil alles verschraubt werden kann. Die Nachteile bestehen darin, dass für die Rotationsservos keine genug grossen Räder verfügbar waren. Dadurch berührte schon bei sehr kleinen Winkeln die unterste Plattform des Untergrundes. Aus diesem Grund wurde die Idee verworfen.
In diesem Teil werden die verschiedenen Prototypen gezeigt. Ihre Funktionsweise wird erläutert, und ihre Schwächen werden diskutiert. Denn diese waren dafür verantwortlich, dass eine Weiterentwicklung notwendig wurde. Der Leser soll darüber informiert werden, was es bei einem eigenen Projekt zu beachten gilt.

Prototyp mit Servo

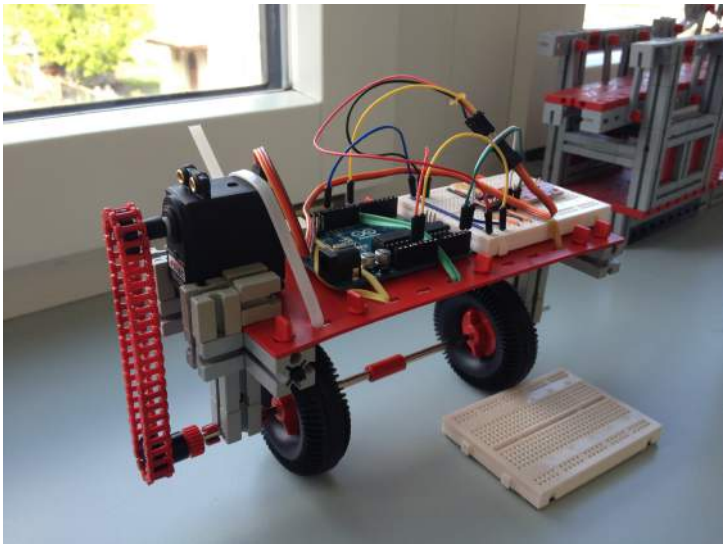

Abbildung 3.6: Erster Prototyp mit Rotationsservo

Der erste Prototyp, welcher in Abbildung 3.6 zu sehen ist, wurde durch einen einzigen Rotationsservo betrieben. Dieser befand sich wie der Rest der Hardware auf einer Plattform über der Achse, welche durch den Servo per Kette angetrieben wurde. Der Arduino wurde dabei durch eine 9 V-Blockbatterie gespiesen. Für den Servo befand sich eine 4,5 V-Flachbatterie unter der Plattform.

Zu Beginn wurde ausschliesslich der Beschleunigungssensor ausgelesen, weil der DMP sehr kompliziert anzusteuern ist. Die praktische Anwendung scheiterte daran, dass sich der Beschleunigungssensor, wie zuvor vermutet, von den Korrekturbewegungen zu stark beeinflussen liess und dadurch keine korrekten Werte lieferte. Daraufhin wurde versucht den Gyro-Sensor zu benutzen, um diesen mit dem Beschleunigungssensor zu kombinieren als eine Lösung ohne die Ansteuerung des DMPs. Die Experimente mit dem Gyro-Sensor waren vielversprechend. Doch weil dieser keinen stabilen Winkel kennt, war ein Balancieren nur während wenigen Sekunden möglich. Ausserdem war die Regelung mit dem Gyro mit sehr grossen Schwingungen des Roboters verbunden.

Aufgrund dieser schlechten Ergebnisse wurde der Programmcode zur Ansteuerung des DMP angepasst, womit der Sketch nun Zugriff auf genaue Winkeldaten hatte. Es wurde nur ein Proportional-Regler verwendet, weil angenommen wurde, dass dieser ausreichend wäre. Der Proportional-Regler funktionierte jedoch nur bei sehr kleinen Winkeln und war überfordert, sobald der Roboter leicht angestossen wurde. Zu diesem Zeitpunkt war noch

nicht klar, dass ein PID-Regler deutliche Vorteile bringen würde, deshalb wurde dieser erst bei einem späteren Prototypen verwendet.
Probleme verursachte auch, dass die Kette, welche vom Motor zur Achse verlief, und dieser ein wenig Freiraum liess. Dadurch wurden Ausgleichsmanöver jeweils verzögert an die Räder übertragen.

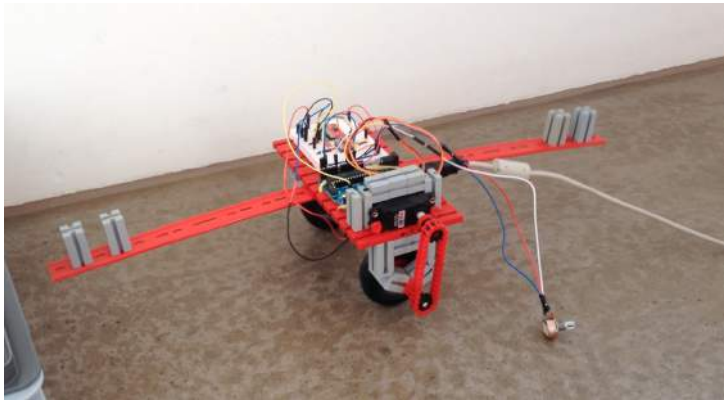

Abbildung 3.7: Prototyp mit aussen angebrachter zusätzlicher Masse

Der Rotationsservo stellte sich ebenfalls als zu schwach heraus, als Masse an äusseren Stationen angebracht wurde, um das Trägheitsmoment zu erhöhen. Dies geschah, nachdem die Resultate aus Abschnitt 2.4 ausgewertet waren. Abbildung 3.7 zeigt den Roboter mit vergrössertem Trägheitsmoment.
Dieser Prototyp nutzte nur einen Proportional-Regler und war dank des hohen Trägheitsmoments und eines genauen Verstärkungsfaktors der erste Prototyp, der über längere Zeit selbständig balancieren konnte. Sobald er jedoch in Bewegung versetzt wurde, war der Servo zu schwach, um ihn wieder aufzurichten. Aus diesem Grund wurde nach diesem Versuch entschieden, auf stärkere DC-Motoren umzusteigen, bei welchen ausserdem das Rad direkt auf die Nabe montiert werden kann. Dadurch wird verhindert, dass die Räder Freilauf haben.
Ein weiteres Problem stellten die Halterungen für die zusätzliche Masse dar, weil sie flexibel waren. Durch die ständige Bewegung begannen sie zu Schwingen und dies nicht im gleichen Rhythmus wie die Schwingungen des Roboters. Dadurch erzeugte Gegendrehmomente führten ebenfalls zu leichten Unsicherheiten beim Balancieren. Es ist also sehr wichtig, keine flexiblen Teile zu verwenden, die sich unabhängig vom Roboter in Bewegung setzen könnten.

Prototyp mit DC-Motoren

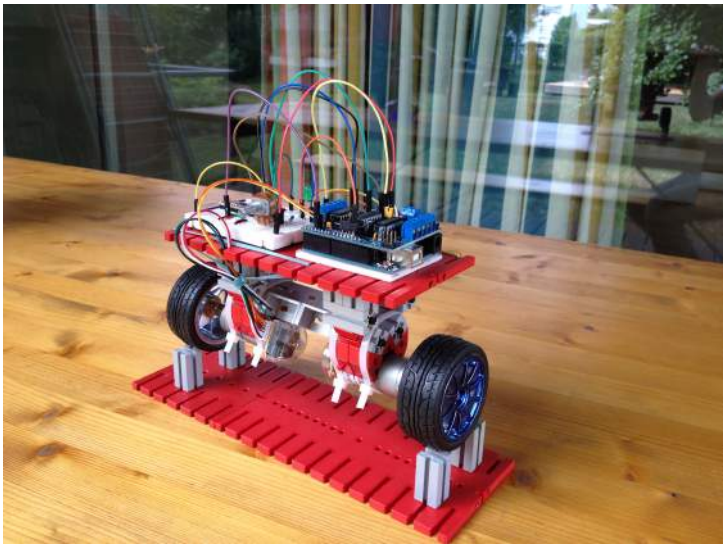

Abbildung 3.8: Erster Prototyp mit DC-Motoren

Da sich die Servos als ungeeignet herausstellten, wurden für die folgenden Prototypen bürstenlose DC-Motoren des Typs JGA25, wie sie in Abschnitt 2.3.5 beschrieben sind, verwendet (siehe Abbildung 3.8). Als Gerüst diente erneut Fischer-Technik. Der Prototyp besass eine Plattform, auf welchem der Arduino und ein Breadboard mit Levelconverter und MPU-6050 festgeklebt wurden. Auf den Arduino wurde das Adafruit Motor Shield gesteckt. An das Shield wurden die beiden Motoren sowie die Stromversorgung, die bei diesem Prototypen noch extern war, angeschlossen. Der MPU-6050 wurde wie in den vorherigen Prototypen an das Arduino-Board angeschlossen. Es gilt dabei zu beachten, dass einige Pins bereits durch das Shield reserviert sind. Auf der Adafruit-Website sind die Pinbelegungen beschrieben:
`https://learn.adafruit.com/adafruit-motor-shield/faq`

Dank eines Programmcodes eines ähnlichen Projekts wurde das Auslesen der Sensordaten sowie das Ansteuern der Motoren um einiges einfacher:

Self Balancing Robot:
`http://trandi.wordpress.com/2014/01/05/self-balancing-robot/`

Der Code machte es möglich, die Sensordaten auszulesen und zu verarbeiten, um die Motoren richtig anzusteuern. Der Sketch arbeitete mit einem PID-Regler, dank dem es schliesslich möglich wurde, den Prototypen zu balancieren. Es mussten nur noch die passenden PID-Werte und ein exakter stabiler Winkel gefunden werden.

Das Einzige, was diesem Prototypen noch fehlte, war eine selbständige Stromversorgung durch Batterien oder einen Akku. Durch den Umbau des Roboters würden jedoch neue PID-Werte gesucht werden müssen, weil sich sein Verhalten durch die zusätzliche Masse und neue Massenverteilung verändert.

3.2.2 Ermitteln des stabilen Winkels

Um einen dem stabilen Winkel ähnlichen Wert zu erhalten, wurde zuerst der Roboter in eine stabile Position gebracht. Die stabile Position ist daran zu erkennen, dass der Roboter nur sehr schwach auf den Gegenstand drückt, welcher ihn in dieser Lage hält. Aufgrund der Masseverteilung ist diese Position nicht zwingend, wenn der Roboter senkrecht steht. Der dabei ausgelesene Winkel wurde für die ersten Balancierversuche verwendet. Bei diesem Experiment gilt es zu beachten, dass der USB-Anschluss des Arduino in Richtung der Radachse zeigt. Es hat sich ergeben, dass das Gewicht des Kabels auf einer Seite des Roboters einen entscheidenden Einfluss auf das Gleichgewicht hat.

Da sich der Roboter ständig auf die linke Seite neigte und somit nur in eine Richtung fuhr, wurde der Wert solange angepasst, bis er sich nur noch an Ort und Stelle bewegte. Dies sollte, aufgrund der Erkenntnis mit dem USB-Kabel, mit Batterien als Stromversorgung durchgeführt werden.

3.2.3 Ermitteln der PID-Werte

These

Es wird angenommen, dass der P-Regler der entscheidendste aller drei Glieder sein wird. Aus diesem Grund werden die anderen Regler zu Beginn mit einem Nullfaktor ausgeschaltet. Sobald ein passender Wert gefunden worden ist, werden der I-Regler und der D-Regler hinzugefügt. Welchen Einfluss diese beiden Regler auf das Verhalten des Roboters haben werden, war zu diesem Zeitpunkt unbekannt. Diese Faktoren werden durch Probewerte so weit eingegrenzt, bis der ideale Wert gefunden ist.

Ausführung & Auswertung

Der P-Regler wurde soweit angepasst, dass der Roboter bereits balancierte. Wichtig ist es zu beachten, dass der Roboter bei einem zu grossen Wert zu schwingen beginnt. Der Wert sollte zu Beginn lieber zu klein gewählt sein, weil er später noch durch die

anderen Regler ergänzt wird. Beim Hinzufügen des I-Reglers wurde festgestellt, dass der Roboter deutlich besser balanciert, umso grösser der I-Regler gewählt wird. Dies liegt vermutlich daran, dass der I-Regler kleine Ungenauigkeiten des Sensors ausgleicht und damit Ruhe in das gesamte System bringt. Der D-Regler wurde vorerst ebenfalls klein gewählt. Auf das Balancieren hatte er kaum Einfluss. Wenn man den Wert vergrössert, ist zu beachten, dass dann der P-Reglerwert etwas kleiner gemacht werden sollte. Wird der D-Regler zu gross gewählt, fängt der Roboter an zu Zittern. Dieses Zittern kann in so starke Schwingungen ausschlagen, dass der Roboter umfällt. Der D-Regler sollte also nicht zu gross gewählt werden.

3.2.4 Encoder

Der Aslong JGA 25 verfügt über sehr hochauflösende optische Encoder. Auf jedem Motor befinden sich zwei Sensoren zur optischen Abtastung einer Gitterscheibe. Je nachdem wie die Signale gegenüber der beiden Sensoren verschoben sind, dreht der Motor in die eine oder in die andere Richtung. Um hochauflösende Encoder zu nutzen, ist es zu empfehlen, diese mit einem Interrupt zu versehen. Das Problem ist, dass der Arduino Uno nur über zwei dieser externen Interrupts verfügt. Einer davon wird bereits für den Sensor verwendet, somit ist nur einer übrig. Für einen Vergleich der Geschwindigkeiten der Motoren wären mindestens zwei weitere nötig. Um auch die Drehrichtung zu bestimmen bräuchte es weitere zwei Interrupt Pins. Aus diesem Grund konnten die Encoder keine Anwendung in diesem Projekt finden, es müsste ein besserer Mikrocontroller verwendet werden, wie bspw. der Arduino Due, welcher an jedem Pin eine Interrupt-Funktion ausführen kann und zusätzlich noch über eine höhere Rechenleistung verfügt.

3.3 Das Endprodukt: Selbstbalancierender Roboter

Der letzte Prototyp kann unabhängig von einer externen Stromversorgung betrieben werden und auf verschiedenen Oberflächen wie einem dünnen Kissen oder im Kies balancieren. Durch den Batterieblock wird die Masse des Roboters massiv erhöht, weswegen sich sein Verhalten bei Erschütterungen deutlich verschlechtert hat gegenüber dem ersten Prototypen mit DC-Motor. Die Batterien sind nicht optimal angebracht, besser wäre eine Positionierung unterhalb der Elektronik, um den Schwerpunkt möglichst tief zu halten. Aufgrund der Konstruktion des vorhergehenden Prototypen war dies jedoch schwierig umsetzbar.

3.3.1 Aufbau

Das Gerüst aus Fischertechnik besteht hauptsächlich aus zwei durch Stützen verbundene Ebenen und die darunter angebrachten Motoren. Auf der unteren Ebene befindet sich die gesamte Elektronik und auf der oberen Ebene ist der Akku befestigt. Die Motoren sind zwischen Winkelelementen eingebettet und mit Kabelbindern festgezurrt. Breadboard und Arduino (inklusive Motorshield) sind auf die Platten geklebt. Die Batteriehalterung ist mit den Akkus zu schwer, um aufgeklebt zu werden, deshalb wurde sie ebenfalls mittels Kabelbinder fixiert.

3.3.2 Besonderheiten zur Schaltung

Der Schaltplan in Abbildung 3.9 kombiniert die am Arduino angeschlossenen IMU und Levelconverter mit dem Motor Shield und den Motoren. Auf den Arduino wird das Motor Shield gesteckt, die beiden DC-Motoren werden an die vorgesehenen Anschlüsse des Shields angeschlossen.
Der Anschluss des Sensors erfolgt genau wie in Abschnitt 3.1.3 mithilfe des Levelconverters. Die Stromversorgung des gesamten Systems läuft über den *VIN*- und *GND*-Ausgang auf dem Motor Shield. Hier können eine externe Stromversorgung, Batterien oder ein Akku angeschlossen werden. Im finalen Prototypen des Roboters werden dafür zehn 1.2 V-Mignon-Akkus verwendet, um die optimale Spannung der Motoren zu erreichen. Zur Sicherheit wurde zwischen dem Motorshield und der Stromversorgung ein 220 μF Kondensator eingesetzt. Dieser soll Ungenauigkeiten in der Stromversorgung ausgleichen. Es konnte jedoch nicht festgestellt werden, ob sich das Verhalten des Roboters dadurch ändert. Der Arduino regelt die eingehenden 12 V automatisch auf die eigene Betriebsspannung von 5 V sowie auf 3.3 V herunter. Alle elektronischen Komponenten ausser dem *ON/OFF*-Schalter befinden sich auf einem Breadboard. Dazu gehört zusätzlich noch ein Potentiometer, um während dem Balancieren den stabilen Winkel zu verstellen und eine LED. Die LED zeigt den Fortschritt des Setups an und während der Ausführung der Loop-Methode gibt sie Aufschluss darüber, ob die Geschwindigkeit der Räder einen positiven oder einen negativen Wert hat.

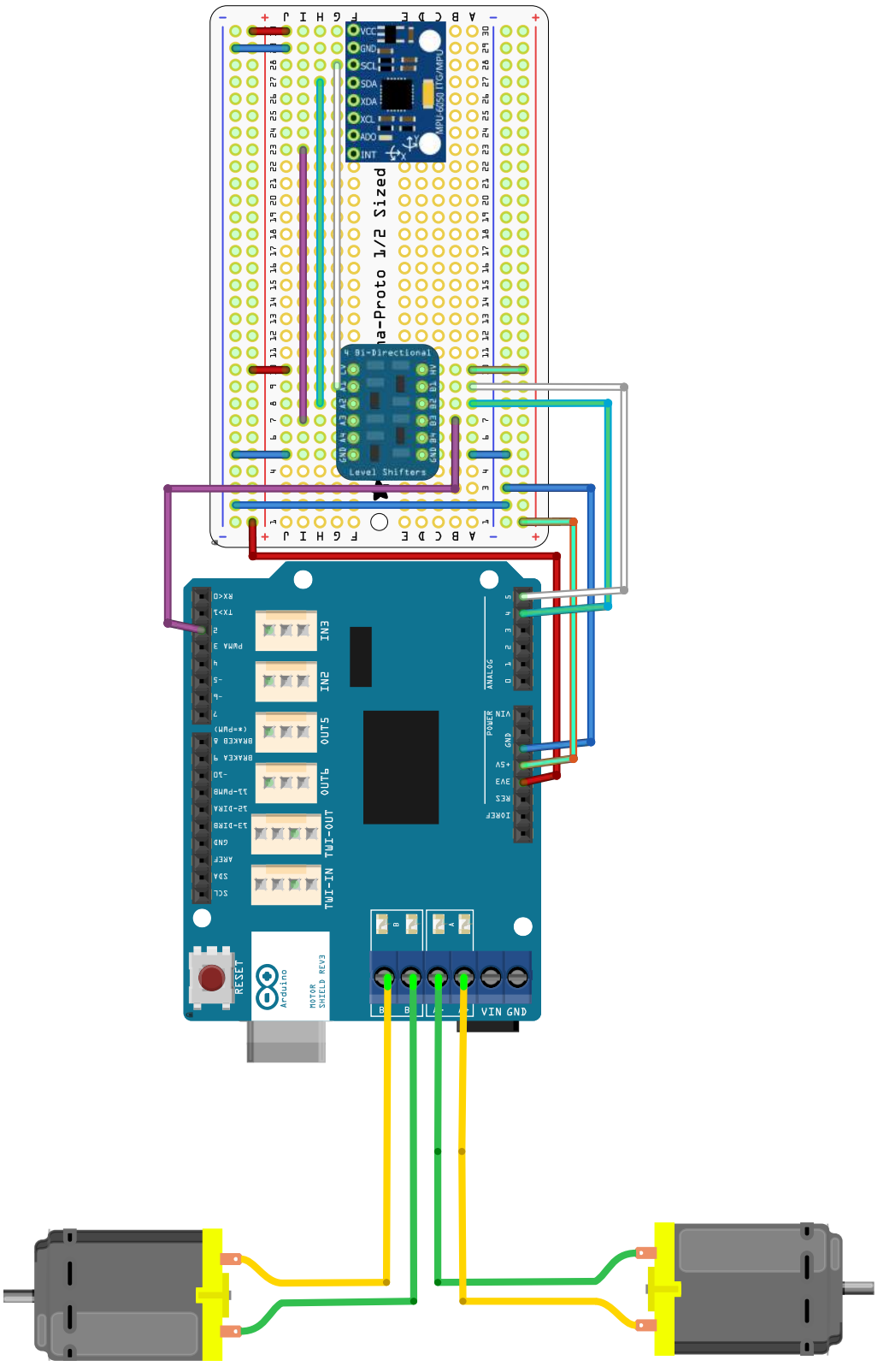

Abbildung 3.9: Schaltung des fertigen selbstbalancierenden Roboters

3.3.3 Programmierung

Der gesamte Quelltext, der für den Roboter verwendet wurde, kann in Abschnitt 5.2 eingesehen werden.
Der Quelltext besteht aus zwei *.ino*-Dateien und einer Headerdatei. In der Headerdatei wird einzig ein Struct erstellt, dass gleichzeitig die Winkeldaten und die Daten des Gyrosensors speichern kann. Die zweite Datei ist für die Ansteuerung des Sensors bestimmt und stammt wie die Headerdatei von der folgenden Website:

Self Balancing Robot:
`http://trandi.wordpress.com/2014/01/05/self-balancing-robot/`

In der Arduino Programmierung ist es üblich benutzerdefinierte Structs in einem separaten Header-File zu platzieren, weil Arduino nicht den vollen Umfang von C zur Verfügung stellt und es somit zu Problemen kommen kann.[6]

Hauptprogramm Der Hauptprogrammcode ist für die Verarbeitung der Sensordaten und Ansteuerung der Motoren verantwortlich. Die Header-Datei für das Struct *AngleGyroData* sowie die Library für das Motor-Shield werden in den ersten zwei Zeilen eingebunden. Danach werden die beiden Motoren definiert. Der Übergabewert markiert den Port, an dem sie angeschlossen sind. In Zeile 12 wird ein neues Struct des Typs *AngleGyroData* erstellt. In der Variable *anfangswinkel* wird der ermittelte stabile Winkel gespeichert. Wichtig ist zu beachten, dass die Winkelwerte in Radianten und die Gyrowerte in Radianten pro Sekunde aus dem Sensor ausgegeben werden. In der Setup-Methode werden die serielle Schnittstelle und die Motoren gestartet. In Zeile 25 wird die *mpu_setup()*-Methode ausgeführt. Diese Methode befindet sich in der zweiten *.ino*-Datei und wird zu einem späteren Zeitpunkt erklärt. Bei erfolgreichem Durchlaufen der Setup-Methode leuchtet die LED für drei Sekunden. In Zeile 35 werden zwei Variablen für die Aufnahme der Zeit erstellt, in Zeile 36 drei verschiedene Variablen für den PID-Regler. Die Variable *integrator* wird für die Aufnahme des Verstärkungsfaktors des I-Reglers verwendet. Die Variable *integratorMAX* kann dafür verwendet werden, um den Integrator zu begrenzen und *aktuellerwinkel* nimmt die Differenz zwischen dem ausgegeben Winkel des Sensors und dem stabilen Winkel auf.
In der Loop-Methode wird als erstes die bis dahin vergangene Zeit gespeichert. Diese Zeit wird in Zeile 76 an die Variable *vorher* übergeben. Der Sensor gibt die Daten mit einer Frequenz von 1 kHz aus. Ist bei einem erneuten Loop-Durchlauf weniger als eine Millisekunde vergangen, würden die vorherigen Daten verwendet werden. Aus diesem Grund wird die Datenaufnahme übersprungen, wenn dem so ist.

Der Methode *mpu_getData*, welche sich in der zweiten *.ino*-Datei befindet, wird das vordefinierte Struct übergeben. Falls der Rückgabewert der Methode Null ist, wurden die Daten erfolgreich im Struct gespeichert. Daraufhin wird der aktuelle Winkel berechnet. Ist *aktuellerwinkel* kleiner als Null, dann leuchtet die LED, ist der Wert grösser, bleibt sie dunkel.
In der für die Geschwindigkeit reservierten Variablen *speed* werden die Verstärkungsfaktoren mit den Regelgliedern multipliziert, dies ist der eigentliche PID-Regler.

Der PID-Regler wird auf die Geschwindigkeit der Motoren angewendet. Da die Geschwindigkeit nur 256 Werte kennt, muss diese auf das Intervall von -255 bis 255 begrenzt werden. Ansonsten könnte der Wert zu gross werden und einen Überlauf verursachen. In diesem Falle wäre der Roboter kaum mehr kontrollierbar und würde umfallen. Die Geschwindigkeiten der beiden Motoren werden in Zeile 62 und 63 übergeben. Dafür wird der Betrag der Geschwindigkeit von 0 bis 255 auf 50 bis 255 skaliert. Dadurch wird die Eigenschaft ausgeglichen, dass sich die Motoren erst bei einer bestimmten Geschwindigkeit zu drehen beginnen. Normalerweise ist diese Schwelle bei Getriebemotoren etwas höher als bei Motoren ohne Getriebe, weil ihre eigene Trägheit erhöht ist. Grundsätzlich sollte sie jedoch für jeden Motor einzeln ermittelt werden, nicht nur für den gleichen Typ, denn kaum ein Motor funktioniert genau gleich wie ein anderer. Eine gute Art diesen Wert zu bestimmen, findet sich in 3.1.3, wenn der Geschwindigkeitswert auf dem Serial-Monitor ausgegeben wird.

Der Integrator wird in Zeile 65 aufsummiert, dazu wird immer der alte Wert zum aktuellen Winkel addiert und neu gespeichert. Zusätzlich kann hier auch eine Begrenzung für den Integrator eingebaut werden, damit dieser nicht zu stark wird. Das Problem eines sehr starken Integrators ist, dass sich eine plötzliche Änderung der Regeldifferenz vom positiven in den negativen Bereich – oder andersherum – nicht mehr ausgleichen lässt, weil der Integrator noch einige Zeit sehr stark bleiben wird. Können nun P- und D-Regler den Integrator nicht ausgleichen, dann unterstützt der I-Regler sogar die Bewegung in die Fallrichtung des Roboters. Diese Situation taucht häufig auf, wenn der Roboter sich ständig in eine Richtung bewegt, weil der stabile Winkel nicht genau bestimmt wurde und er aus diesem Grund mit einem Hindernis kollidiert. Die Räder drehen sich in diesem Augenblick jedoch noch nicht in die Richtung des Ausgleichs sondern sie bringen den Roboter zu Fall.

Im letzten Teil wird anhand des Vorzeichens der Geschwindigkeit bestimmt, in welche Richtung sich die Räder der beiden Motoren drehen müssen. Die Motoren wurden hier in die gleiche Richtung angeschlossen, deshalb müssen im Programmcode auch die Richtungen unterschiedlich definiert werden.

Methoden der IMU Die zweite *.ino*-Datei enthält eine Methode für das Setup der IMU und eine für das Auslesen der Daten. Die Setup-Methode gibt im Falle eines erfolgreichen Setups ein *true* an den Hauptprogrammcode zurück, so dass dieser weiterfahren kann.

Die zweite Methode, die sich *mpu_getData* nennt, wird in Zeile 53 definiert. Sie hat als Übergabewert das AngleGyroData-Struct aus der Header-Datei, welches in diesem Falle *_winkeldaten* getauft wurde. Der Aufruf findet in Zeile 46 statt. Bei einem erfolgreichen Speichern der Daten hat die Methode den Rückgabewert Null. Wichtig zu wissen ist, wie auf die beiden Daten des Structs zurückgegriffen werden kann. Sie können mithilfe des Punktoperators aufgerufen werden. Somit wäre der Aufruf des Gyrowertes *_winkeldaten.gyro* und der Winkelwert *_winkeldaten.angle*. Die beiden werden in den Zeilen 50 und 57 angewendet.

4 Schluss

In der Einleitung wurde die Leitfrage gestellt, ob es möglich sei, mit geringem Vorwissen und im vorgegebenen Zeitrahmen, einen selbstbalancierenden Roboter zu konstruieren: Diese Frage kann positiv beantwortet werden. Der Roboter kann mehrere Minuten balancieren. Die Zeitdauer ist nur begrenzt durch die Batteriekapazität oder durch zu grosse Hindernisse und die Oberflächenbeschaffenheit der Fahrbahn.

Im Theorieteil konnte festgestellt werden, dass ein selbstbalancierender Roboter über einen tiefen Schwerpunkt, jedoch über ein grosses Trägheitsmoment verfügen sollte, damit er einfacher geregelt werden kann.

Während des Entwicklungsprozesses wurden vorerst Prototypen erstellt, die über Rotationsservos als Antrieb zum Ausgleich verfügten. Diese Servos und der programmierte Proportionalregler stellten sich als unzureichend heraus. Mithilfe von DC-Motoren und einem PID-Regler konnten diese Hürden überwunden werden.

Das Ermitteln der PID-Werte stellte eine weitere Schwierigkeit dar und sie konnten auch bis zum Ende dieser Arbeit nicht perfekt gewählt werden, was auch durch das Fehlen von Encodern negativ beeinflusst wurde. Die Encoder hätten die Geschwindigkeit der Räder kontrollieren können. Aus diesem Grund fährt der Roboter ständig leicht nach vorne und nach hinten und kann keine ruhige, aufrechte Position einnehmen.

Die Konstruktion war bereits so weit fortgeschritten, dass die Unterbringung der Batterien nur noch zuoberst möglich war. Die Batterien sind die schwerste Komponente am ganzen Roboter und durch ihre Position wurden die Erkenntnisse aus Abschnitt 2.4 nicht genutzt. Dadurch lässt sich auch das schlechte Verhalten bei Stössen gegen den Roboter erklären.

Gestaltung

Für die Erstellung dieser schriftlichen Arbeit wurden zwei verschiedene Softwarepakete verwendet um die schriftliche Arbeit zu gestalten und ein Web Hosting Service genutzt um einen einfachen und kostenlosen Internetauftritt zu erstellen. Bei LaTeX handelt es sich um ein Textsatzsystem und Fritzing ist ein Tool zur Erstellung von elektronischen Schaltplänen. Für die Erstellung der Website wurde Jimdo verwendet.

LaTeX

LaTeX ist ein Satz von Makros für das Textsatzsystem TeX. LaTeX enthält verschiedene vordefinierte Dokumentklassen, die Strukturierungsmöglichkeiten anbieten. Ausserdem sind verschiedene Werkzeuge zur Erstellung von Verzeichnissen verfügbar. LaTeX wird durch Pakete von Autoren ergänzt, die beinahe jede Anwendung abdecken. Ursprünglich wurde LaTeX von Leslie Lamport geschrieben, gepflegt wird es jedoch mittlerweile von einer kleinen Gruppe von Entwicklern.

Das Textsatzsystem TeX wurde in den Jahren 1977 bis 1986 durch Donald E. Knuth entwickelt, da seiner Meinung nach keine akzeptable Lösung für das Setzen von mathematischem Text am Computer existierte. Er entwickelte es deshalb für seine Buchserie *The Art of Computer Programming*. TeX ist sehr beliebt bei Autoren und Verlegen, die wissenschaftliche Texte bearbeiten, welche viele technische oder mathematische Formeln enthalten. [14]

Für das Erlernen von LaTeX wurde auf folgendes Buch zurückgegriffen:

Joachim Schlosser. *Wissenschaftliche Arbeiten schreiben mit LaTeX – Leitfaden für Einsteiger*. 5. überarbeitete Auflage. mitp, 2014.

Fritzing

Fritzing ist eine Open-Source Anwendung, die entwickelt wurde, um Entwicklern von Physical Computing Projekten eine Möglichkeit zu geben, ihre Projekte übersichtlich zu gestalten und zu präsentieren. Es kann zwischen drei Ansichten gewählt werden: Der Breadboard-Ansicht, der Schaltplanansicht und der Leiterplattenansicht. Fertig gezeichnete Projekte können auch als Leiterplatten bei Fritzing bestellt werden, beispielsweise ein selber erstelltes Arduino-Shield. Fritzing kommt mit einem Standardsatz an elektronischen Bauteilen, viele weitere Bauteile können zusätzlich im Internet bezogen werden. Einige Hersteller wie Sparkfun oder Adafruit bieten bereits ihre eigenen Bibliotheken zum Download an. Ursprünglich wurde Fritzing im Jahre 2007 an der Universität Potsdam gestartet, mittlerweile wird die Entwicklung von der Friends-of-Fritzing Foundation geführt. [15]

Fritzing kann unter folgendem Link bezogen werden:

Offizielle Fritzing Website: `http://fritzing.org/home/`

Jimdo

Jimdo wurde im Jahre 2004 in Hamburg gegründet, wo sie auch heute noch ihren Hauptsitz hat. Die Entwickler wollten das Internet jedem zur Verfügung stellen und jeden daran teilhaben lassen. Der einfache Baukasten mit dem auf Jimdo eine Website erstellt werden kann ist eines der erfolgreichsten Angebote weltweit, das diesen Service anbietet. [22]

Die im Rahmen dieser Arbeit erstellte Website und das zugehörige YouTube-Video können unter folgendem Link aufgerufen werden:

Self Balancing Robot – Jimdo `http://selfbalancingrobot.jimdo.com`

Der Blog auf der Website wird auch eine Plattform für Fragen an die Autoren und für regelmässige Updates bezüglich weiterer Projekte sein.

Nachwort

Die Arbeit war herausfordernd, teilweise überfordernd, aber interessant und sehr lehrreich. Wir mussten uns mit den Disziplinen Mechanik, Elektronik, Regeltechnik und Programmierung auseinandersetzen und dank der Einarbeitung in das für technisch-naturwissenschaftliche Publikationen geeignete Textsatzprogramm LaTeX und in das, für die Darstellung von Schaltplänen optimierte Programm *Fritzing*, gelang uns eine präzise Darstellung.

Wir sind sehr stolz auf das Endergebnis, weil es uns gelungen ist, einen selbstbalancierenden Roboter zu konstruieren, den Entwicklungsprozess schriftlich zu dokumentieren und seine Funktionstüchtigkeit in einem Internetauftritt zu präsentieren.

Im schriftlichen Teil dieser Arbeit hätten wir gerne ein Experiment zur experimentellen Ermittlung der Winkelgeschwindigkeit eines fallenden Stabes dokumentiert. Aufgrund der Ungenauigkeit des Sensors war dies jedoch nicht nach unseren Vorstellungen umsetzbar.

Weitere Schritte beim Bau des Roboters könnte das Hinzufügen der Encoder sein, beispielsweise mithilfe eines stärkeren Mikrocontrollers. Zusätzlich möchten wir den Roboter noch über eine Fernbedienung steuern können, so dass er sich fortbewegen und auch Kurven fahren kann. Möglich wäre dies mit einer RC-Fernbedienung, mit einem Playstation 3 Controller über Bluetooth oder mit einem Smartphone.

Danksagung

Wir möchten gerne der betreuenden Lehrperson, Herrn D. Zurmühle für die tatkräftige Unterstützung sowohl im Freifach "Physical Computing", als auch während unserer Freizeit, recht herzlich danken. Es war uns immer möglich Fragen zu stellen, welche in allen Fällen beantwortet wurden.
Ausserdem möchten wir uns bei Herrn M. Pickert für die Bereitstellung vieler von uns benötigten Materialien und für die Hilfestellungen im Labor bedanken.
Unseren Eltern möchten wir für die finanzielle Unterstützung und den Support generell danken.

Buchquellen

[5] Erik Bartmann. *Die elektronische Welt mit Arduino entdecken*. 1. Auflage. Köln: O'Reilly Verlag, 2011.

[7] *Brockhaus*. 21. völlig neu überarbeitete Auflage. Leipzig, 2006.

Onlinequellen

[1] Universität Braunschweig A. Formella D. Fellner. *Rotation mit Quaternionen*. URL: http://trevinca.ei.uvigo.es/~formella/doc/ig04/node97.html (besucht am 29. 06. 2014).

[2] SICK AG. *Encoder*. URL: http://www.sick.com/group/DE/home/products/product_portfolio/encoders/Seiten/positioning_encoders.aspx (besucht am 29. 07. 2014).

[3] Arduino. *Arduino*. URL: http://arduino.cc/ (besucht am 07. 07. 2014).

[4] Arduino. *Wire Library*. URL: http://arduino.cc/en/pmwiki.php?n=Reference/Wire (besucht am 04. 08. 2014).

[6] Alexander Brevig. *Resource for Arduino*. URL: http://playground.arduino.cc/Code/Struct (besucht am 15. 08. 2014).

[8] Steffen Buchner. *Beschleunigungssensoren*. URL: http://userpages.uni-koblenz.de/~physik/informatik/Sensoren/beschleunigung.pdf (besucht am 10. 08. 2014).

[9] Harvey Mudd College. *Inertial Measurement Units*. URL: http://www.eng.hmc.edu/NewE80/IMUDetails.htm (besucht am 08. 08. 2014).

[10] Pololu Corporation, Hrsg. *Continuous-rotation servos and multi-turn servos*. URL: http://www.pololu.com/blog/24/continuous-rotation-servos-and-multi-turn-servos (besucht am 07. 04. 2014).

[11] Johnson Electric. *EC Motoren (bürstenlos)*. URL: http://www.johnsonelectric.com / de / resources - for - engineers / automotive - applications / motion - technology/ec-motor-brushless.html (besucht am 29. 07. 2014).

[12] Sparkfun Electronics. *Logic Level Converter Bi-Directional*. URL: https://www.sparkfun.com/products/12009 (besucht am 08. 07. 2014).

[13] Sparkfun Electronics. *Triple Axis Accelerometer and Gyro Breakout - MPU-6050*. URL: https://www.sparkfun.com/products/11028 (besucht am 29. 06. 2014).

[14] Deutschsprachige Anwendervereinigung TeX e.V. *TEX& Co.* URL: http://www.dante.de/tex.html (besucht am 02. 08. 2014).

[15] Friends-of-Fritzing. *Fritzing.* URL: https://code.google.com/p/fritzing/.

[16] Play-Zone GmbH. *Aslong JGA25 Gear Motor mit Encoder.* URL: http://www.play-zone.ch/de/aslong-jga25-gear-motor-mit-encoder.html (besucht am 29. 07. 2014).

[17] Play-Zone GmbH. *DK Motor/Stepper/Servo Shield für Arduino.* URL: http://www.play-zone.ch/de/ (besucht am 09. 07. 2014).

[18] Universität Heidelberg. *Die Kunst des Balancierens.* URL: http://www.tphys.uni-heidelberg.de/~huefner/PhysikUeberall/V04S.pdf (besucht am 29. 07. 2014).

[19] Segway Inc., Hrsg. *Segway.* URL: http://www.segway.com/ (besucht am 28. 06. 2014).

[20] Inc InvenSense. *MPU-9150.* URL: http://www.invensense.com/mems/gyro/mpu9150.html (besucht am 09. 08. 2014).

[21] ITWissen. *Gyrosensor.* URL: http://www.itwissen.info/definition/lexikon/Gyrosensor-gyro-sensor.html (besucht am 08. 08. 2014).

[22] Jimdo. *Über Jimdo.* URL: http://de.jimdo.com/%C3%BCber-jimdo/ (besucht am 16. 08. 2014).

[23] Ryno Motors. *Ryno.* URL: rynomotors.com (besucht am 31. 07. 2014).

[24] RoboTrack, Hrsg. *Das Servo und seine Technik.* URL: http://www.robotrack.org/include.php?path=article&contentid=210 (besucht am 14. 03. 2014).

[25] Segway Schweiz. *Wie dynamische Stabilisierung funktioniert.* URL: http://www.segway.ch/de/infos/funktionsweise.php (besucht am 28. 06. 2014).

[26] Ralph Timmermann. *Elektronik – I2C Grundlagen.* URL: http://www.ralph.timmermann.org (besucht am 09. 07. 2014).

[27] Benutzer: Waste. *Regelungstechnik.* URL: http://rn-wissen.de/wiki/index.php/Regelungstechnik (besucht am 09. 07. 2014).

[28] Andreas Schwarz (Webmaster). *Mikrocontroller.* URL: http://www.mikrocontroller.net/articles/Mikrocontroller (besucht am 07. 07. 2014).

5 Anhang

5.1 Listing *Lesen der Sensordaten*

Das nachfolgende Listing entspricht der in Abschnitt 3.1.3 erklärten Programmierung um die Sensordaten lesen zu können.

```
#include "Wire.h" // Die Wire-Library wird eingebunden
#include "I2Cdev.h" // I2C-Device-Library wird eingebunden
#include "MPU6050_6Axis_MotionApps20.h" // Die Library fuer die MPU-6050 einbinden

MPU6050 mpu; // Konstruktor erstellt eine MPU-6050 mit der Variable mpu

bool dmpReady = false;  // wird auf true gesetzt, falls DMP bereit ist
uint8_t mpuIntStatus;   // beschreibt aktuellen INT-Status des MPU
uint8_t devStatus;      // Status der Kommunikation (0 = erfolgreich)
uint16_t packetSize;    // Variable fuer Paketgroesse (Standard = 42 Bytes)
uint16_t fifoCount;     // alle Werte in FIFO gezaehlt
uint8_t fifoBuffer[64]; // FIFO-Speicher

Quaternion q;           // [w, x, y, z]         Array fuer Quaternionen
VectorInt16 aa;         // [x, y, z]            Array fuer Beschleunigungskomponenten
VectorInt16 aaReal;     // [x, y, z]
// Array fuer Beschleunigung (gravitationsbereinigt)
VectorFloat gravity;    // [x, y, z]
// Array fuer Gravitationsvektorkomponenten
float euler[3];         // [psi, theta, phi]    Array fuer Eulersche Winkel
float ypr[3];           // [yaw, pitch, roll]
// Array fuer yaw (gieren)-, pitch (nicken)- & roll (rollen)
int16_t gyros[3];       // [x, y, z]            Array fuer Winkelgeschwindigkeit

volatile bool mpuInterrupt = false;     // hat Interrupt-Pin einen HIGH-Wert?
void dmpDataReady() {
    mpuInterrupt = true;
}

void setup() {

    Wire.begin(); // Wire-Library wird gestartet

    Serial.begin(115200); // Serielle Schnittstelle wird gestartet

    Serial.println(F("MPU-6050 initialisieren..."));
    // MPU-6050 initialisieren wird ausgegeben
```

```
38      mpu.initialize();                                   // mpu wird initialisiert
39
40      Serial.println(F("Geraeteverbindungen testen..."));
41      // Geraeteverbindung wird getestet wird ausgegeben
42      Serial.println(mpu.testConnection() ? F("verbunden") : F("Verbindungsfehler"));
43      // mpu-Verbindung wird getestet
44
45      Serial.println(F("DMP intialisieren..."));  // DMP wird initialisiert
46      devStatus = mpu.dmpInitialize();
47      // Geraete-Status wird geprueft --> funktioniert falls Rueckgabe = 0
48
49      if (devStatus == 0) {  // Falls Geraete-Status i.O.
50
51          Serial.println(F("DMP einschalten"));  // DMP einschalten ausgeben
52          mpu.setDMPEnabled(true);                // DMP einschalten
53
54          Serial.println(F("Interrupt-Erkennung einschalten..."));  // Textausgabe
55          attachInterrupt(0, dmpDataReady, RISING);
56          // Interrupt Pin verbunden mit Interrupt 0 (digitaler Pin 2);
57          // Interrupt aktivieren wenn von LOW auf HIGH
58          mpuIntStatus = mpu.getIntStatus();
59
60          Serial.println(F("DMP ist bereit. Warten auf den ersten Interrupt..."));
61          dmpReady = true; // dmpReady auf true --> loop() kann ausgefuehrt werden
62
63          packetSize = mpu.dmpGetFIFOPacketSize();
64
65      } else {              // Falls Geraete-Status nicht i.O.
66
67          Serial.println(F("DMP Initalisierungsfehler..."));  // Textausgabe
68      }
69
70  }
71
72  void loop() {
73
74      if (!dmpReady) return; // loop wird nur durchgefuehrt falls dmpReady = true
75
76      mpuInterrupt = false; // mpuInterrupt wird auf false gesetzt
77      mpuIntStatus = mpu.getIntStatus(); // Interrupt-Status wird ausgelesen
78
79      fifoCount = mpu.getFIFOCount(); // FIFOCount auslesen (zaehlt auszulesende Werte)
80
81      // Falls Programmcode zu ineffizient
82      if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
83          mpu.resetFIFO(); // Werte loeschen --> neu beginnen
84          Serial.println(F("FIFO overflow!"));  // Textausgabe
85
86      } else if (mpuIntStatus & 0x02) { // Falls Programmcode genug effizient
87
88          while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
89          // solange nicht 42 Bytes in fifoCount --> weitere Werte aufnehmen
90
91          mpu.getFIFOBytes(fifoBuffer, packetSize);
92
93          fifoCount -= packetSize;
```

```
94
95              mpu.dmpGetQuaternion(&q, fifoBuffer);         // Quaternionen auslesen
96              mpu.dmpGetQuaternion(&q, fifoBuffer);         // Quaternionen auslesen
97              mpu.dmpGetGravity(&gravity, &q);              // Gravitation auslesen
98              mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);    // ypr auslesen
99              mpu.dmpGetGyro(gyros, fifoBuffer);            // Gyrosensor auslesen
100             Serial.print(ypr[2]);                         // x-Komponente ypr ausgeben
101             Serial.print("\t");                           // neuer Tabulator
102             Serial.println(gyros[2]);                     // x-Komponente Gyro ausgeben
103     }
104 }
```

5.2 Listing *Endprodukt: Selbstbalancierender Roboter*

Das nachfolgende Listing entspricht der in Abschnitt 3.3.3 erklärten Programmierung des letzten Prototypen.

5.2.1 Definition des AngleGyroData-Structs

```
1  #ifndef BBUTIL_h              // falls BBUTIL_h noch nicht definiert;
2  #define BBUTIL_h              // dann definiere BBUTIL_h
3
4  #include <Arduino.h>          // bindet Standardfunktionen fuer Arduino ein
5
6  struct AngleGyroData {        // Erstellung eines structs fuer die gleichzeitige Aufnahme
7                                // von Winkelgeschw. und Winkel
8      float angle;
9      int16_t gyro;
10 };
11
12 #endif
```

5.2.2 Hauptprogrammcode

```
#include "BButil.h"      // Library fuer die Einbindug des struct AngleGyroData
#include "AFMotor.h"     // Library fuer das Motorcontroller-Board

AF_DCMotor motor1(3);   // motor1 an Port 3 des Motorcontroller-Boards
AF_DCMotor motor2(4);   // motor2 an Port 4 des Motorcontroller-Boards

int ledpin = 13;        // LED an Pin 13

// Der Winkel ist in Radianten, die Winkelgeschwindigkeit (Gyrosensor) in rad/s
// Winkel und Winkelgeschwindigkeit haben verschiedene Vorzeichen

AngleGyroData _winkeldaten = {0,0}; // Konstruktor des Typs AngleGyroData (aus BButil.h)
// erstellt das struct _winkelDaten und setzt beide Komponenten auf Null

float anfangswinkel = -0.024;           // Der Wert anfangswinkel wird gesetzt

void setup(){
  Serial.begin(115200);                 // Die Serielle Schnittstelle wird gestartet

  motor1.setSpeed(200);                 // Startvorgang der Motoren
  motor2.setSpeed(200);                 //
  motor1.run(RELEASE);                  //
  motor2.run(RELEASE);                  //

  mpu_setup();                          // Aufruf der setup-Methode fuer den MPU

  pinMode(ledpin, OUTPUT);              // Der ledpin wird als Ausgang definiert

  digitalWrite(ledpin, HIGH);           // LED auf High --> leuchtet
  delay(3000);                          // Sketch wird 3s gestoppt --> LED leuchtet 3s
  digitalWrite(ledpin, LOW);            // LED wird ausgeschaltet

}

long jetzt, vorher = 0; // Variablen des Typs long fuer die Zeitmessung werden definiert
float integrator = 0, integratorMAX = 200, aktuellerwinkel; // Variablen des Typs float
// fuer PID-Algorithmus werden definiert

void loop(){

  jetzt = millis();         // jetzt nimmt die Zeit auf, die bis dahin vergangen ist

  if(jetzt - vorher > 1){ // MPU gibt Daten mit 1 kHz aus; falls seit letztem Durchlauf
  // weniger als 1 ms vergangen --> nicht durchfuehren

    int8_t res = mpu_getData(&_winkeldaten); // res nimmt den Status des MPU auf

    if(res == 0){

      aktuellerwinkel = _winkeldaten.angle - anfangswinkel;
      if(aktuellerwinkel < 0){
            digitalWrite(ledpin, HIGH);
          }else{
            digitalWrite(ledpin, LOW);
```

```
55            }
56
57      int16_t speed = 2600 * aktuellerwinkel + 100 * integrator - 2.5 * _winkeldaten.gyro;
    // PID-Regler; alle 3 Komponenten werden mit Verstaerkungsfaktoren multipliziert
58  // Gyro-Wert wird subtrahiert, weil anderes Vorzeichen
59
60        speed = constrain(speed, -255, 255);
    // Die Geschwindigkeit wird auf ein Intervall von -255 bis 255 begrenzt
61
62        motor1.setSpeed(map(abs(speed), 0, 255, 50, 255));
    // Der Betrag der Geschwindigkeit wird auf einen Intervall von 50 bis 255 skaliert und
63        motor2.setSpeed(map(abs(speed), 0, 255, 50, 255));
    // an die beiden Motoren weitergegeben (Motoren beginnen sich bei 50 zu drehen)
64
65      integrator = constrain(integrator + aktuellerwinkel, -integratorMAX, integratorMAX);
    // Integratorwert des Algorithmus wird begrenzt auf -10 bis 10
66
67        if(speed > 0){
68         motor1.run(FORWARD);
    // Falls speed positiv; Motoren fahren vorwaerts
69         motor2.run(BACKWARD);
    //
70        }
71        else{
72         motor1.run(BACKWARD);
    // Falls speed negativ; Motoren fahren rueckwaerts
73         motor2.run(FORWARD);
    //
74        }
75
76        vorher = jetzt; // Die Variable "vorher" nimmt die Zeit bei Start der loop um sie
77                        // im naechsten Durchlauf mit der aktuellen Zeit zu vergleichen
78     }
79   }
80 }
```

5.2.3 Methoden für Setup und Ansteuerung der MPU-6050

```
#include "Wire.h"                          // Einbindung der Wire-Library
#include "I2Cdev.h"                        // Einbindung der I2C-Device-Library
#include "MPU6050_6Axis_MotionApps20.h"    // Einbindung der MPU-6050-Library

MPU6050 mpu;   // Definition von mpu des Typs MPU6050

uint16_t packetSize;
uint8_t fifoBuffer[64];

Quaternion q;            // Array fuer Quaternionen
VectorFloat gravity;     // Array fuer Gravitationsvektorkomponenten
float yawPitchRoll[3];
// Array fuer yaw (gieren)-, pitch (nicken)- & roll (rollen)-Neigungen
int16_t gyros[3];        // Array fuer Kompenenten der Winkelgeschwindigkeit

bool mpu_setup() {
    Wire.begin(); // Wire Library wird gestartet

    Serial.print(F("MPU initialisieren ")); // Textausgabe im serial Monitor
    mpu.initialize();

    Serial.print(F("Geraete-ID: ")); Serial.println(mpu.getDeviceID());
    // gibt die Geraete-ID aus

    Serial.println(F("Verbindungstest")); // Verbindungstest ausgeben
    Serial.println(mpu.testConnection() ? F("Verbunden") : F("Verbindungsfehler"));
    // Verbindungstest durchfuehren und Resultat ausgeben

    Serial.println(F("DMP initialisieren")); // DMP(Digital Motion Processor)
    uint8_t devStatus = mpu.dmpInitialize(); // initialisieren

    if (devStatus == 0) { // wenn devStatus = 0 --> Initialisierung erfolgreich
        Serial.println(F("Starte DMP"));
        mpu.setDMPEnabled(true); // DMP funktionstuechtig --> sendet ein true an den MPU

        packetSize = mpu.dmpGetFIFOPacketSize();

        Serial.println(F("Warte auf ersten Interrupt..."); // gibt Text aus

        return true; // gibt setup() ein "true" zurueck; teilt mit, dass der MPU
                     // startbereit ist und auf den 1st Interrupt wartet
        }

    else { // else tritt in Kraft, wenn devStatus != 0 ist. Die Schleife gibt einen Text
           // aus gibt dem setup() ein "false" zurueck

        Serial.print(F("DMP Fehler ")); Serial.println(devStatus);

        return false;
    }
}

int8_t mpu_getData(AngleGyroData* data){ // Erstellung einer Methode zur Auslesung des
                                         // Winkels und W-Geschw.; Datentyp int8_t mit
```

```
55                                                  // AngleGyroData struct als Uebergabewert
56      uint8_t mpuIntStatus = mpu.getIntStatus();
57
58      uint16_t fifoCount = mpu.getFIFOCount();
59
60      if ((mpuIntStatus & 0x10) || fifoCount >= 1024) {
61          mpu.resetFIFO();
62          return -1;
63
64      } else if (mpuIntStatus & 0x02) {
65          if (fifoCount >= packetSize) {
66
67            mpu.getFIFOBytes(fifoBuffer, packetSize);
68            mpu.resetFIFO();
69          }
70
71          mpu.dmpGetQuaternion(&q, fifoBuffer);
72          mpu.dmpGetQuaternion(&q, fifoBuffer);
73          mpu.dmpGetGravity(&gravity, &q);
74          mpu.dmpGetYawPitchRoll(yawPitchRoll, &q, &gravity);   // DMP berechnet ypr-Werte
75          mpu.dmpGetGyro(gyros, fifoBuffer);   // DMP berechnet Gyrosensor-Werte
76
77          data->angle = yawPitchRoll[2]; // falls data = angle --> Winkelwert
78          data->gyro =  gyros[2];   // falls data = gyro --> Gyrowert
79
80          return 0;
81      }else {
82        return -2;
83      }
84  }
```