Abschluss-Ausarbeitung
für das Modul **Elektronik für Lehrämtler** zu dem Projekt:

ARoPra

Arduino **Ro**boter mit **Pr**üfungs**A**ngst

Ein selbstbalancierender
Zweibein-Roboter basierend auf
der Arduino-Platform

vorgelegt von
Cornelius Brütt
MatNr: 1161780
Mail: stu231310@mail.uni-kiel.de

Seminarleitung: Prof. Dr. Dietmar Block

Kiel, am 05. Oktober 2023

Inhaltsverzeichnis

1 Kurzvorstellung

ARoPa ist ein Balancier-Robotikprojekt mit zwei Rädern basierend auf der Arduino-Plattform. Der Roboter ermittelt die momentane Orientierung im Raum mithilfe eines Beschleunigungssensors und errechnet über einen Algorithmus Steuerungsbefehle für zwei Schrittmotoren, die wiederum die Räder antreiben. Hierüber wird versucht, die Orientierung im Raum beizubehalten - Der Roboter versucht sich also aufrecht zu balancieren.

Die Entscheidung, dieses Projekt als Gegenstand der Abschlussprüfung auszuwählen, motivierte sich aufgrund der Komplexität: Die Software sollte anspruchsvoll ausfallen, da ich durch mein zweites Studienfach Informatik bereits viel Vorerfahrung mitbringe. Weiterhin reizte mich es, eine komplexere Schaltung zusammenzusetzen, die neben diversen Komponenten ebenfalls den Umgang mit dem Lötkolben erfordert. Ich wollte dieses Projekt zum Anlass nehmen, meine Fähigkeiten in der hardwarenahen Programmierung auszubauen und neue Fähigkeiten wie das Löten von Schaltkreisen zu erlernen.

2 Entwicklung

Das Projekt ist im Rahmen der Abschlusspräsentation des Moduls Ëlektronik für Lehrämtlerän der CAU Kiel entstanden. Die Projektplanung startete mit der Recherche im Oktober 2022, wo unter Anderem die Inspiration für das äußerliche Erscheinungsbild im Internet gefunden wurde[1]. In der Winterpause 2022/2023 startete der Bau des Grundgerüsts sowie die Bestellung der genutzten Komponenten. Die Softwareentwicklung startete mit dem Breadboarding Ende Juli und wurde im August 2023 über das Versionsmanagmentsystem git dokumentiert[2].

3 Details

3.1 Aufbau Schaltung

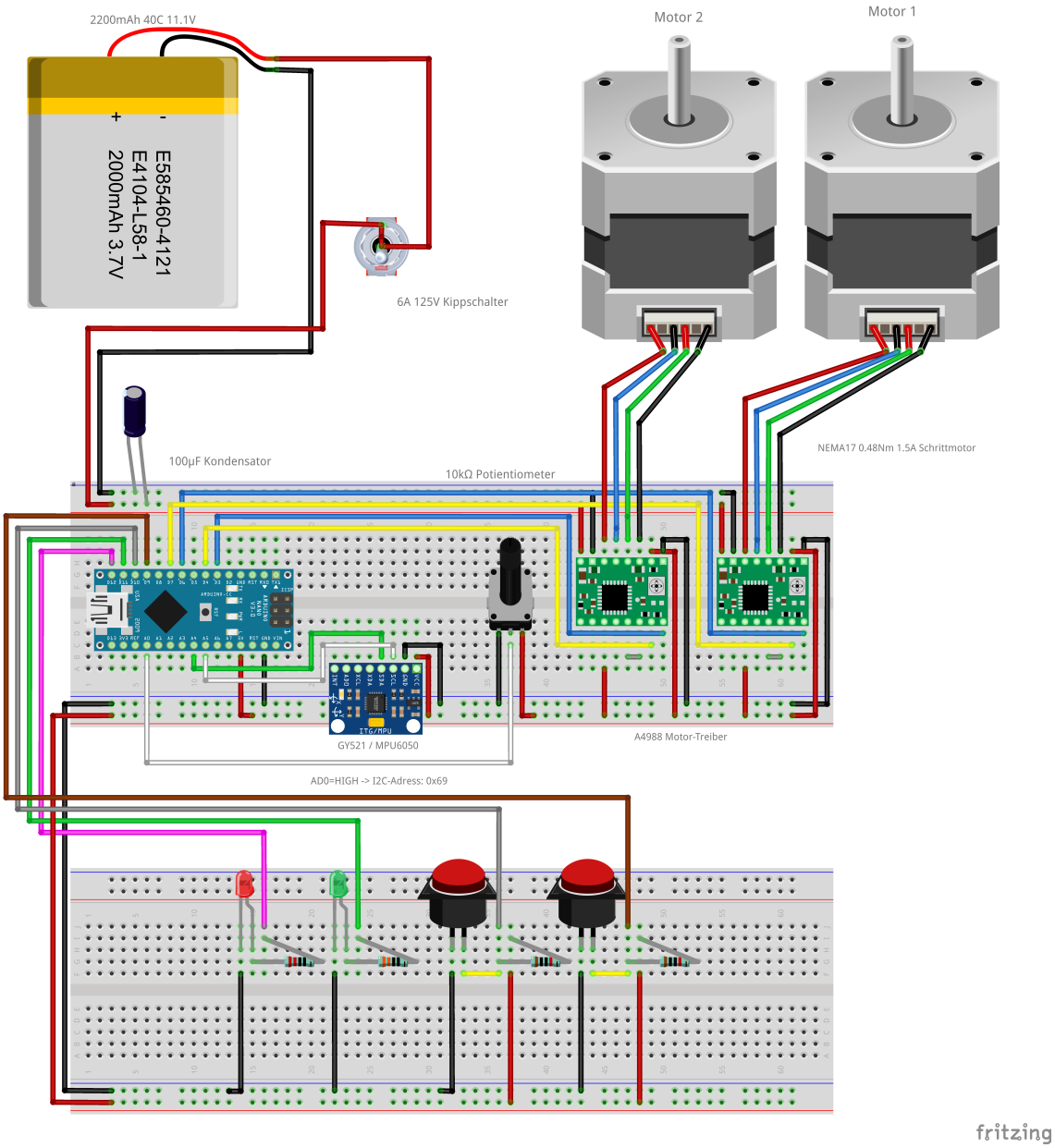

Abbildung 1: Die finale Schaltung im Breadboard-Aufbau. Eine Schaltskizze befindet sich im Anhang

Folgende Komponenten wurden verbaut:

- Arduino Nano Mikrocontroller
- MPU6050 Beschleunigungssensor und Gyroskop auf dem GY521 Breakout-Board
- 2x A4988 Motortreiber
- 2x NEMA17 0.48Nm 1.5A 4-Phasen Schrittmotoren
- 2200mAh 11.1V 40C LiPo-Batterie
- 100 µF Elektrolyt-Kondensator
- 10kΩPotentiometer
- 125V 6A Kippschalter
- diverse Widerstände, Knöpfe, Kabel

Die Schrittmotor-Treiber sind über die eingebauten Trimm-Potentiometer auf einen Betriebsstrom von ca 1.1A eingestellt worden. In der momentanen Beschaltung der MS1-3-Pins laufen die Schrittmotoren im sogenannten 1/4-Schritt-Modus - Statt 200 Schritte pro Umdrehung steuert der Treiber den Motor mit 800 Schritten pro Umdrehung an. Zusätzlich wurden die Treiber-Controller mit einem 6mm Aluminium Passivkühler versehen, um auch im längeren Betrieb eine einigermaßen geeignete Betriebstemperatur zu gewährleisten.
Alle Komponenten des oberen Breadboards befinden sich auf einer 70x60mm Rasterplatine. Dabei wurden für die größeren Komponenten Pin-Aufnahmeheader auf die Platine festgelötet sodass jene Bauteile einfach draufgesteckt und bei Bedarf wieder gelöst werden können.Die Knöpfe und LED des unteren Breadboards wurden zusammen mit dem Kippschalter des Batterie-Stromkreises auf einem Streifen Pappelholz als "Bedienoberfläche"geklebt.

Die Beine sowie das Skelett des Roboters bestehen aus 12mm starkem Spanholz, welches mit M5x65mm Gewindeschrauben an den Gelenken zusammengehalten werden. Das Gehäuse besteht aus Zuschnitten von 4mm Pappel-Bastelholzplatten und wurde größtenteils verleimt- Mithilfe von zwei Stoßdämpfern aus dem Modellbau wurde eine einfache Federlagerung realisiert. Die 70mm Aluminium-Räder stammen als einziges Bauteil aus dem Robotokbedarf.

3.2 Aufbau Software

Die Software, die den Roboter betreibt, lässt sich grob in drei wichtige Bestandteile trennen:

Messung der Orientierung

Um den Winkel θ des Roboters zum Boden (Gemessen von der Lochplatine zum Lot) zu ermitteln, werden Beschleunigungsdatena_y und a_z sowie Winkelgeschwindigkeit ω_x ermittelt. Der Winkel ergibt sich als

$$\theta_{acc} = arctan(a_y/a_z)$$

bzw.

$$\theta_{gyro} = \omega_x \cdot t_{sample} + \omega_0$$

wobei t_{sample} das vorab definierte Zeitintervall darstellt, indem gemessen wird und ω_0 der Winkel *vor* diesem Zeitintervall definiert.
Beide Winkel werden nun zusammengerechnet. Da die Werte des Beschleunigungssensors von hoher Messstreuung geprägt sind und die Werte des Gyroskop mit der Zeit von dem Nullpunkt abdriften, stehen jetzt mehrere Möglichkeiten zur Fusionierung offen [3]:

1. das einfache Mitteln beider Messergebnisse ohne Berücksichtigen der Fehlerquellen (Vogel-Strauß-Methode nach Tanenbaum)[4], [5].

2. ein **Komplementärfilter** agierend als Hochpass für den Beschleunigungssensor und als Tiefpass des Gyroskops, um die Hauptfehlerquellen jeder Messung zu minimieren.

3. ein **Kalman-Filter** als theoretisch optimales System, welches sich jedoch mathematisch als sehr komplex darstellt [3], [6].

Alle Optionen wurden im Verlaufe der Entwicklung implementiert und getestet, wobei sich der Kalman-Filter als superior herausstellte. Im Folgenden wurde daher eine MPU6050-Bibliothek mit integriertem Kalman-Filter zur Messwerterhebung genutzt [7].

Berechnen eines Steuerbefehls

Aus dem gemesssenen Winkel soll nun ein Befehl generiert werden, der als Resultat diesen Winkel möglichst nahe am gewünschten Wert (zum Beispiel 0°) hält. Hierfür

bietet sich die Implementation eines *Proportional-Integral-Derivative Controller*, eines sogenannten PID-Reglers an [8], [9].
Der Algorithmus nimmt einen festen, erwünschten Wert (*Set Point*, unsere 0°) und einen regulierenden Parameter (*Input*, unseren gemessenen Winkel θ) und generiert daraus einen *Output*, der einen regelbaren Parameter beinflussen soll (*Process Value*, unsere Motorgeschwindigkeit). Der Output wird dabei aus einem proportionalen, einen integrierenden und einen differentiellen Teil ermittelt, die jeweils unterschiedliche Charakteristika für das Ausgangssignal bedeuten. Zum Einstellen der Gewichtung dieser Teile werden drei Parameter *KP, KI* und *KD* genutzt, dessen numerische Werte für jeden Aufbau experimentell ermittelt werden müssen.

Betrieb der Motoren

Aus dem *Output* des PID-Algorithmus wird eine Motorengeschwindigkeit errechnet. Dafür wird der Output auf ein Intervall von $[-360, 360]$ abgebildet, welches jeweils einer vollständigen Rotation vor-/rückwärts entspricht. Dieser Wert wird als absolute Position interpretiert, auf die der Schrittmotor gedreht werden soll.
Die Motorsteuerung erfolgt hierbei über die weit verbreitete *AccelStepper*-Library, da diese das Beschleunigen und Abbremsen der Motoren automatisiert [10]. Anstelle der ständigen Aktualisierung der Geschwindigkeit beschleunigt diese Bibliothek die Motoren eigenständig auf eine eingestellte Maximalgeschwindigkeit und bremst entsprechend rechtzeitig wieder ab. Damit werden sonst übliche Probleme wie das ungewollte Überspringen von Schritten oder das „Überschießen“ der gewünschten Position vermindert.

An dieser Stelle soll eine Problematik der genutzten Bibliothek kurz erwähnt werden: Die Bibliothek erfordert ein möglichst häufiges Aufrufen der *run()* oder *runSpeed()*Funktion, welches üblicherweise in der loop()-Routine des Hauptprogrammes stattfindet. Da die Berechnung der PID-Werte diese Routine jedoch merkbar verlangsamt, resultiert diese "traditionelleÜmsetzung in deutlich langsameren Motoren-Geschwindigkeiten. Als Lösung habe ich mich sogenannten *Interrupt-Service-Routines* (kurz **ISR**) bedient, die in einem festgelegten Zeitintervall die Programmausführung unterbrechen und eben die gewünschte *run()*- Funktion aufrufen können [11].

4 Ausblick

Zu dem Stand dieser Ausarbeitung befindet sich die erste Version von*ARoPa* kurz vor der Fertigstellung. Kurzfristiges Ziel wird es also selbstverständlich sein, die Parameter des PID-Algorithmus sowie der Motorensteuerung weiter einzustellen und

zu verfeinern, um eine bessere Balance und Reaktion auf Umwelteinflüsse wie z.B. Stöße zu erzielen. Desweiteren sind diverse zusätzliche Fähigkeiten wie eine Fernsteuerung oder das Einbinden eines Abstandhalters über einen Ultraschall-Distanzmesser möglich, um den Algorithmus unter weiteren Herausforderungen zu verbessern. Auch das eigenständige Umprogrammieren der genutzten Bibliotheken zur besseren Abstimmung mit diesem Aufbau wäre möglich
Nach diesen kurzfristigen Schritten wäre es denkbar, dieses Projekt in mehrere Richtungen zu steuern. So ist eine zweite Version *ARoPA MK II* denkbar, die von den gewonnenen Erfahrungen und Problemfeldern der ersten Version profitiert und auf diesen Erkenntnissen aufbauen kann. Auch ein Hoch- oder Herunterskalieren des Aufbaus zum Entwickeln eines *ARoPA Mini* oder *Maxi* würde eine neue geeignete Herausforderung stellen. Selbstverständlich lassen sich die Erkenntnisse über Datenfilter, Steueralgorithmen und Schrittmotoren in einer Bandbreite an anderen Robotikprojekten

Literatur

[1] User "Pro Know". “Diy Arduino Based Self Balancing Robot | PROKNOW”. (21/01/2020), Adresse: `https://www.youtube.com/watch?v=DlsZUFGkmaQ&t=288s` (besucht am 7. Okt. 2023).

[2] Cornelius Brütt. “Dachgruber/self-balancing-robot: Arduino code needed for the self-balancing-robot project for ElektronikFuerLA SoSe23 at CAU Kiel”. (7/10/2023), Adresse: `https://github.com/Dachgruber/self-balancing-robot` (besucht am 7. Okt. 2023).

[3] Shane Colton. “The Balance Filter, A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform”. (7/10/2023), Adresse: `http://www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/complementary%20filter.pdf` (besucht am 7. Okt. 2023).

[4] Andrew S. Tanenbaum, *Moderne Betriebssysteme*, 3. Aufl. München: Person Studium, 2009. (besucht am 10. Juli 2023).

[5] Wikipedia, Hrsg. “Vogel-Strauß-Algorithmus”. de. Creative Commons Attribution-ShareAlike License Page Version ID: 202270320. (2020), Adresse: `https://de.wikipedia.org/w/index.php?title=Vogel-Strau%C3%9F-Algorithmus&oldid=202270320` (besucht am 10. Juli 2023).

[6] Wikipedia, Hrsg. “Kalman-Filter”. de. Creative Commons Attribution-ShareAlike License Page Version ID: 237041786. (2023), Adresse: `https://de.wikipedia.org/w/index.php?title=Kalman-Filter&oldid=237041786` (besucht am 10. Juli 2023).

[7] User ProFL. “ProFL/KalmanMPU6050: A Kalman filter library for usage with Arduino and MPU6050. Based on https://github.com/TKJElectronics/KalmanFilter and https://github.com/TKJElectronics/Example-Sketch-for-IMU-including-Kalman-filter”. (3/07/2018), Adresse: https://github.com/ProFL/KalmanMPU6050 (besucht am 8. Okt. 2023).

[8] Wikipedia, Hrsg. “Proportional–integral–derivative controller”. en. (2023), Adresse: https://en.wikipedia.org/w/index.php?title=Proportional%E2%80%93integral%E2%80%93derivative_controller&oldid=1178778094 (besucht am 10. Juli 2023).

[9] DC - PID Explained, “PID Controller Explained”, *PID Explained*, 24. Nov. 2018. Adresse: https://pidexplained.com/pid-controller-explained/ (besucht am 7. Okt. 2023).

[10] Mike McCauley. “AccelStepper - Arduino Reference”. (24/11/2022), Adresse: https://www.arduino.cc/reference/en/libraries/accelstepper/ (besucht am 7. Okt. 2023).

[11] “Arduino timer interruptions ISR Tutorial”. (28/02/2021), Adresse: https://electronoobs.com/eng_arduino_tut140.php (besucht am 7. Okt. 2023).