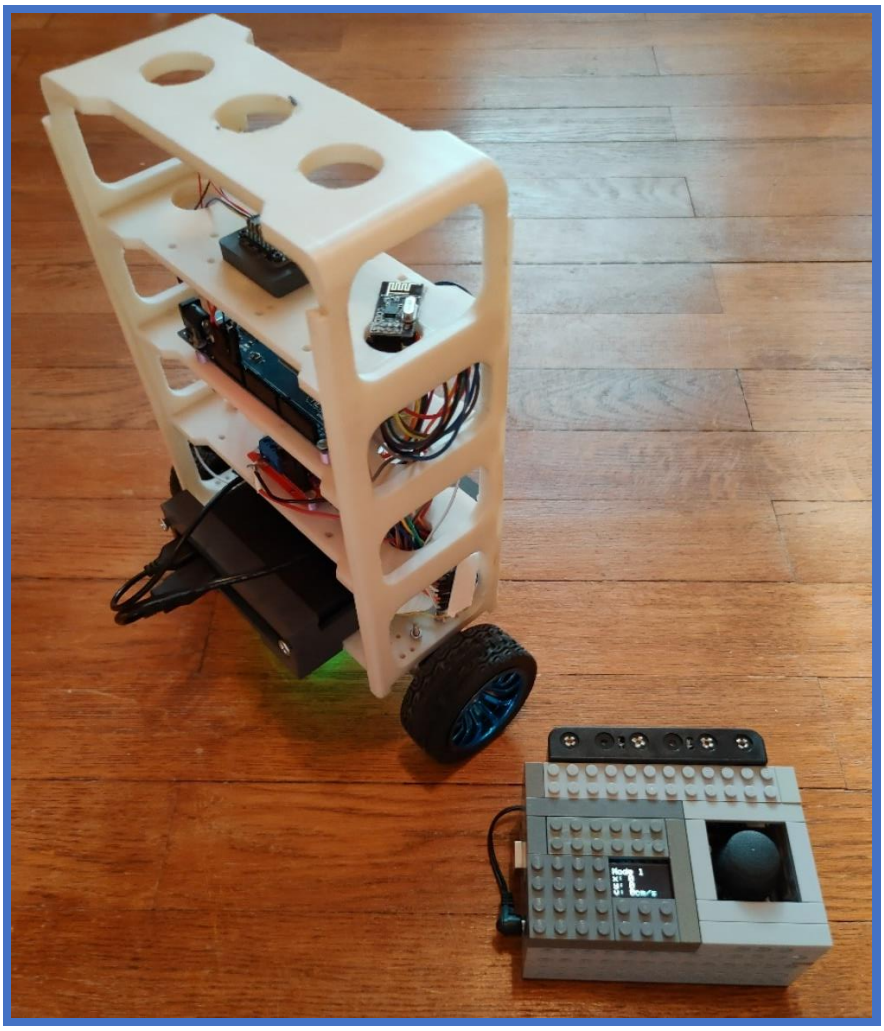

Selbstausgleichender 2-rädriger Roboter mit Fernsteuerung

ABSCHLUSSARBEIT 2022

MATAIGNE Philippe 0191336114
UNIVERSITÄT LUXEMBURG | ELEKTROTECHNIK SEMESTER 6 SS 2022
BETREUER: PROF. DR. ING. JEAN-RÉGIS HADJI-MINAGLOU

UNIVERSITÉ DU LUXEMBOURG

Zusammenfassung

Das Projekt baut auf einem zweirädrigen Roboter auf, der ursprünglich nur im Gleichgewicht stehen konnte. In dieser Arbeit wird der Roboter so erweitert, dass er von einer Fernbedienung gesteuert nach vorne und hinten fahren und wenden und dies ebenfalls aufgrund eines vorprogrammierten Pfades tun kann. Dies wird hauptsächlich durch den Zusatz eines Drehgebers ermöglicht. Dabei wird auf eine Regler Struktur aus kaskadierenden PID-Reglern gesetzt. Im Endeffekt kann der Roboter allerdings nur langsam fahren, da die Powerbank des Robotes viel wiegt und genug Spielraum gelassen werden muss damit er nicht kippt, weil die Böden oft nicht sehr uniform sind. Auch in der Fernbedienung vorprogrammierte Pfade kann der Roboter schlussendlich abfahren, aber die Positionserfassung ist, während dem Wenden nicht immer ganz zuverlässig und könnte mit zusätzlichen Sensoren verbessert werden.

Vorwort

Ich möchte mich bei den folgenden Personen bedanken.

Prof. Dr. Ing. Jean-Régis Hadji-Minaglou für die Betreuung der Arbeit.

Nishith Kapoor, Berg Robert und Abdoellah Adam für die geleistete Vorarbeit.

Daniel Warnimont für seine Hilfe im Labor.

Abbildungsverzeichnis

Abbildung 1-1: Foto eines Segway© i2 SE von Segway.com (1) 1
Abbildung 2-1: Fotos der ursprünglichen Teile: Motor (links), Arduino Uno (Mitte) und Motortreiber L298N mit Boost Converter XL6009 (rechts) 3
Abbildung 2-2: Arduino Mega 2560 Rev3 im Arduino Shop (5) 4
Abbildung 2-3: Pinbelegung Arduino Mega 2560 Rev3 (5) 5
Abbildung 2-4: Foto von BNO055 im Adafruit Shop (7) 6
Abbildung 2-5: Fotos von LN298N Motortreiber auf Amazon.de (9) 7
Abbildung 2-6: Foto von DC Boost Converter XL6009 auf Amazon.de (11) 7
Abbildung 2-7: Foto von Litionite Falcon Mini auf Amazon.de (12) 8
Abbildung 2-8: Foto der 3 Varianten des nRF24L01+ von HowToMechatronics.com (13) 8
Abbildung 2-9: Schema der Kanäle des nRF24L01+ von HowToMechatronics.com (13) 9
Abbildung 2-10: Schema eines Pakets des nRF24L01+ von LastMinuteEngineers.com (14) ... 10
Abbildung 2-11: Pinbelegung des nRF24L01+ von HowToMechatronics.com (13) 10
Abbildung 2-12: Foto von JGA25-371 mit Zubehör auf Amazon.de (15) 11
Abbildung 2-13: Foto von Größenvergleich zwischen JGA25-370 (oben) und JGA25-371 (unten) 11
Abbildung 2-14: Foto der Encoderseite des JGA25-371 (15) 12
Abbildung 2-15: Foto vom Motor JGA25-371 mit beigelieferten Kabel und Pinbelegung (15) 13
Abbildung 2-16: Foto vom fertig verkabelten Roboter 14
Abbildung 2-17: Schematische Darstellung der Verkabelung des Roboters 15
Abbildung 2-18: Foto von Arduino Uno Rev3 im Arduino Shop (16) 17
Abbildung 2-19: Schema der Pinbelegung des Arduino Uno Rev3 (16) 18
Abbildung 2-20: Foto des Joysticks auf Amazon.de (18) 19
Abbildung 2-21: Foto mit Potentiometer des Joysticks (links) und Schema des Koordinatensystems von LastMintuteEngineers.com (rechts) (19) 19
Abbildung 2-22: Foto des SSD1306 OLED im Adafruit Shop (20) 20
Abbildung 2-23: Foto eines 6xAA Batterienhalters auf Amazon.de (22) 21
Abbildung 2-24: Foto der Fernbedienung von vorne (rechts) und hinten (links) 22
Abbildung 2-25: Foto der Anschlüsse der Fernbedienung (links) und der Unterseite (rechts) 22
Abbildung 2-26: Foto der fertigen Verkabelung im Inneren der Fernbedienung 23
Abbildung 2-27: Schematische Darstellung der Verkabelung der Fernbedienung 24
Abbildung 2-28: Foto des fertigen Roboters und Fernbedienung im Betrieb 26
Abbildung 3-1: Mechanische Darstellung des Systems 27
Abbildung 3-2: Wurzelortskurve und Sprungantwort des Gleichgewichts ohne Regler mit Matlab gezeichnet 30
Abbildung 3-3: Wurzelortskurve und Sprungantwort des Gleichgewichts mit PI-Regler mit Matlab gezeichnet 32
Abbildung 3-4: Wurzelortskurve und Sprungantwort des Gleichgewichts mit PID-Regler mit Matlab gezeichnet Matlab-Befehle: 33

Abbildung 3-5: Wurzelortskurve und Sprungantwort der Fortbewegungsgeschwindigkeit ohne Regler mit Matlab gezeichnet 34
Abbildung 3-6: Wurzelortskurve und Sprungantwort der Fortbewegungsgeschwindigkeit mit I-Regler mit Matlab gezeichnet 35
Abbildung 3-7: Darstellung der Rotation des Systems 36
Abbildung 3-8: Wurzelortskurve und Sprungantwort der Wendegeschwindigkeit mit I-Regler mit Matlab gezeichnet 38
Abbildung 3-9: Wurzelortskurve und Sprungantwort der Wendegeschwindigkeit mit PI-Regler mit Matlab gezeichnet 39
Abbildung 3-10: Wurzelortskurve und Sprungantwort der abgefahrenen Distanz ohne Regler mit Matlab gezeichnet 40
Abbildung 4-1: Schema einer I2C Verbindung (26) 42
Abbildung 4-2: Schema einer SPI Verbindung (26) 43
Abbildung 4-3: Schematische Darstellung der Regler Struktur des Roboters 46
Abbildung 4-4: Verlauf der Encodersignale bei positiver (oben) und negativer (unten) Drehrichtung 48
Abbildung 4-5: Vereinfachtes Flussdiagramm von Robo.ino 52
Abbildung 4-6: Vereinfachtes Flussdiagramm von Controller.ino 62
Abbildung 5-1: Anschließen der Stromversorgung des Roboters (links) und des Kontrollers (rechts) 70
Abbildung 5-2: Foto der Fernbedienungsanzeige, wenn nicht verbunden (links) und verbunden (rechts) 70
Abbildung 5-3: Foto der Fernbedienungsanzeige im ersten Modus mit Joystick im Mittelpunkt (links) und Joystick unten rechts (rechts) 70
Abbildung 5-4: Foto der Fernbedienungsanzeige im zweiten Modus, wenn kein Pfad ausgewählt ist (links) und bei ausgewähltem Pfad (rechts) 71

Tabellenverzeichnis

Tabelle 2-1: Auflistung der Kabelverbindungen des Roboters 16
Tabelle 2-2: Auflistung der Kabelverbindungen der Fernbedienung 25
Tabelle 2-3: Auflistung der Bauteile des Projektes mit Logikspannung und Preis 25
Tabelle 4-1: Auflistung der Regler mit ihren Parametern 51

Inhaltsverzeichnis

1 Einleitung 1
1.1 Hintergrund 1
1.2 Ziele und Anforderungen 1
2 Hardware 3
2.1 Ursprüngliche Hardware 3
2.2 Anforderungen an die Hardware 3
2.3 Roboter Hardware 4
2.3.1 Arduino Mega 2560 Rev3 4
2.3.2 Accelerometer-Gyroskop-Magnetometer BNO055 6
2.3.3 Motortreiber L298N 6
2.3.4 DC Boost Converter XL6009 7
2.3.5 Litionite Falcon Mini Powerbank 8
2.3.6 nRF24L01+ 8
2.3.7 JGA25-371 Motoren 10
2.3.8 Verkabelung des Roboters 14
2.4 Fernbedienung Hardware 17
2.4.1 Arduino Uno Rev3 17
2.4.2 nRF24L01+ 18
2.4.3 Joystick 19
2.4.4 SSD1306 0,96“128x64 OLED Anzeige 20
2.4.5 6xAA Batterienhalter 21
2.4.6 Klemmbaustein Hülle 21
2.4.7 Verkabelung der Fernbedienung 23
2.5 Gesamtübersicht 25
3 Theorie 27
3.1 Regelung des Gleichgewichts 27
3.2 Regelung der Fortbewegungsgeschwindigkeit 33
3.3 Regelung der Wendegeschwindigkeit 36
3.4 Regelung der abgefahrenen Distanz 39
3.5 Regelung der Richtung 40
4 Software 41
4.1 Allgemeines 41

4.1.1 Arduino IDE 41
4.1.2 I2C 41
4.1.3 SPI 42
4.2 Datenübertragung zwischen Roboter und Kontroller 43
4.3 Roboter Software 44
4.3.1 Ursprünglicher Programmcode 44
4.3.2 Regler Struktur 44
4.3.3 Timing 47
4.3.4 Berechnung der Position und Drehgeschwindigkeit der Motoren 47
4.3.5 Implementierung der Regler Struktur 49
4.3.6 Regler Werte 50
4.3.7 Speicherung eines Pfadpunktes 51
4.3.8 Robo.ino 52
4.4 Fernbedienung Software 60
4.4.1 Joystick 60
4.4.2 Bildschirmanzeige 61
4.4.3 Pfadprogrammierung 61
4.4.4 Funktion zur einfachen Programmierung der Regler des Roboters 62
4.4.5 Controller.ino 62
5 Gebrauchsanleitung des fertigen Roboters 70
6 Schlussfolgerung 72
7 Literaturverzeichnis 74
8 Anhang 77
8.1 SelfRobo.h 77
8.2 SelfRobo.cpp 78

1 Einleitung

Es wird nun zuerst der Hintergrund von zweirädrigen Robotern erläutert und dann die Anforderungen und Ziele des Projektes aufgelistet.

1.1 Hintergrund

Mit steigendem Fokus auf elektronische Geräte, die sich selbst fortbewegen können und Aufgaben ausführen, sind verschiedene Konzepte aufgetreten. Eins davon, ist der Roboter auf 2 Rädern, denn dies ermöglicht eine beträchtliche Platzeinsparung in Vergleich zu 4 Rädern und gleichzeitig eine hohe Beweglichkeit. Allerdings ist die Umsetzung wesentlich komplexer als noch bei einem 3- oder 4-rädigen Roboter, der einen festen Stand hat und man sich nur um die Regelung der Fortbewegung an sich kümmern muss. Mit nur 2 Rädern und dem Schwerpunkt des Geräts über diesen, muss gleichzeitig auch das Gleichgewicht geregelt werden, denn der Roboter kann nun umkippen und auch die Kipplage hat einen Einfluss auf die Fortbewegung des Roboters. Ein Beispiel einer reellen Anwendung heutzutage, sind verschiedene Geräte von Segway zur persönlichen Fortbewegung, wie der Segway© i2 SE in Abbildung 1-1, ebenfalls auf 2 Rädern, den man in die gewünschte Fahrrichtung neigen muss, wenn man ihn steuert. Der Roboter in diesem Projekt ist vergleichbar mit einer Miniaturvariante davon, nur dass der Roboter sich auf Befehl einer Fernbedienung zur Fortbewegung selbst in die richtige Richtung neigt.

Abbildung 1-1: Foto eines Segway© i2 SE von Segway.com (1)

1.2 Ziele und Anforderungen

Das Projekt basiert auf einer bereits vorhandenen Basis von vorherigen Abschlussarbeiten von Nishith Kapoor (2), Berg Robert (3) und Abdoellah Adam (4). Diese Basis besteht aus einem Roboter mit 2 Rädern, der sich selbst balancieren kann, so dass er stehen bleibt und nicht

kippt. Dies ist allerdings die einzige Funktion, die der Roboter ursprünglich besitzt. Das Ziel dieser Arbeit besteht nun darin, Funktionen zu diesem Roboter hinzuzufügen. Diese Erweiterungen bestehen erstens aus einer kabellosen Fernbedienung mit Anzeige und Joystick, um dem Roboter Befehle vermitteln zu können als auch die Fähigkeit anschließend diese Befehle ausführen zu können. Diese Befehle bedeuten, dass der Roboter in beide Richtungen fahren als auch sich drehen kann. Eine weitere erforderte Funktion ist, dass der Roboter in der Fernbedienung vorprogrammierte Pfade abfahren können soll ohne weitere Nutzereingaben, nachdem der Pfad ausgewählt wurde.

2 Hardware

Im Folgenden wird, die benutze Hardware näher betrachtet. Zuerst wird die Hardware, die bereits am Anfang des Projektes vorhanden war, aufgezählt. Danach werden nach den Anforderungen des Roboters, die finale Hardware im Roboter und Kontroller erläutert und schlussendlich die Pinbelegung aufgelistet.

2.1 Ursprüngliche Hardware

Die am Anfang bereits vorhandene Hardware bestand aus zwei 1360rpm DC 12V Motoren ohne Drehgeber mit der Bezeichnung JGA25-370. Zum Ermitteln der Positionsdaten wurde der BNO055 Sensor, ein Accelerometer-Gyroskop-Magnetometer, benutzt. Zum Ausführen der Datenerfassung und Kontrollalgorithmen wurde ein Arduino Uno Rev3 eingesetzt. Schließlich, als Driver für die Motoren der L298N mit einem DC Boost Converter XL6009, der die 5V Spannung von der Litionite Falcon Mini Powerbank auf 12V konvertiert. Vom ursprünglichen Roboter wurden der BNO055, der L298N, der Boost Converter, die Powerbank und das 3D-gedruckte Gerüst schlussendlich beibehalten. Der Arduino Uno wurde allerdings in der Fernbedienung wiederverwendet. Die Abbildung 2-1 zeigt die ursprünglichen Hardwareteile.

Abbildung 2-1: Fotos der ursprünglichen Teile: Motor (links), Arduino Uno (Mitte) und Motortreiber L298N mit Boost Converter XL6009 (rechts)

2.2 Anforderungen an die Hardware

Der Roboter soll in beide Richtungen fahren und Pfade abfahren können, deswegen ist es nötig die Geschwindigkeit und Position des Roboters zu ermitteln. Diese lassen sich mithilfe von Drehgebern an den Motoren ermitteln. Das gleiche gilt für die Drehrichtung und Drehposition, diese Informationen sind jedoch schon mit dem bereits vorhandenen BNO055 ermittelbar. Des Weiteren soll der Roboter ferngesteuert werden können, dafür werden kabellose Sender und Empfänger am Roboter und der Fernbedienung benötigt. Für diese Ergänzungen am

Roboter reicht der vorherige Arduino Uno auch nicht mehr aus, da die Anschlussmöglichkeiten zu limitiert sind. An der Fernbedienung wird dann auch noch ein Mikroprozessor für die Befehle Übergabe benötigt. In diesem Fall reicht der vorherige Arduino Uno aus. Des Weiteren braucht die Fernbedienung eine Spannungsversorgung wie einen Akku oder Batterien, ein Eingabegerät wie ein Joystick und ein Ausgabegerät, also ein Bildschirm.

2.3 Roboter Hardware

Zuerst wird sich nun mit der am Roboter verbauten Hardware befasst.

2.3.1 Arduino Mega 2560 Rev3

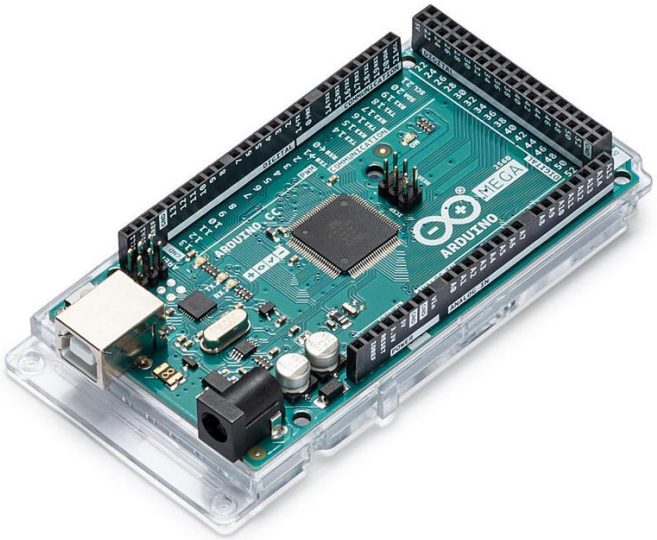

Abbildung 2-2: Arduino Mega 2560 Rev3 im Arduino Shop (5)

Als Mikrokontroller des Roboters wird der Arduino Mega 2560 Rev3 eingesetzt. Der Vorteil von Arduino ist die hohe Benutzerfreundlichkeit der Arduino Software IDE, die es ermöglicht den Mikroprozessor ohne Probleme zu programmieren. Auch die Kompatibilität mit den anschließbaren Bauteilen ist recht hoch, denn Arduino ist neben Raspberry Pi wohl einer der weit verbreitesten Mikroprozessor-Hersteller im „do-it-yourself“(DIY)-Bereich. Des Weiteren basierte der ursprüngliche Roboter auf einem Arduino Uno Rev3, so dass der Mikroprozessor ohne weitere Modifikationen am Projekt eingesetzt werden kann. Der Hauptgrund für den Wechsel vom Arduino Uno auf den Arduino Mega sind die verbesserten Anschlussmöglichkeiten, eine höhere Rechenleistung, sowie eine höhere Speicherkapazität. Die hohe Rechenleistung und der geringe Stromverbrauch sowie kompakter Formfaktor, wenngleich grösser als der Arduino Uno, machen diesen Mikrokontroller zur perfekten Auswahl für unsere Zwecke.

Der Mikroprozessor des Megas ist der ATmega2560. Dieser Chip ist ein Atmel AVR 8 Bit Chip mit einer RISCV Architektur und einem 16MHz Kristall Oszillator. Die Flashspeichergrösse beträgt 256KB mit 8KB SRAM und 4KB EEPROM. Der Mega besitzt auch einen USB-Port, der

eine einfache Verbindung zu einem PC ermöglicht, einen DC Stromanschluss, einen ICSP Header und einen Reset Schalter. Wie in Abbildung 2-3 gezeigt, besitzt der Arduino 54 Digitale Input/Output Pins, wobei 15 als PWM Output benutzt werden können, und 16 5V Analog Inputs mit 10 Bit Auflösung, also mit 1024 verschiedenen Werten. Zusätzlich gibt es 3 5V Pins, einen 3,3V Pin und 5 Erdungspins.

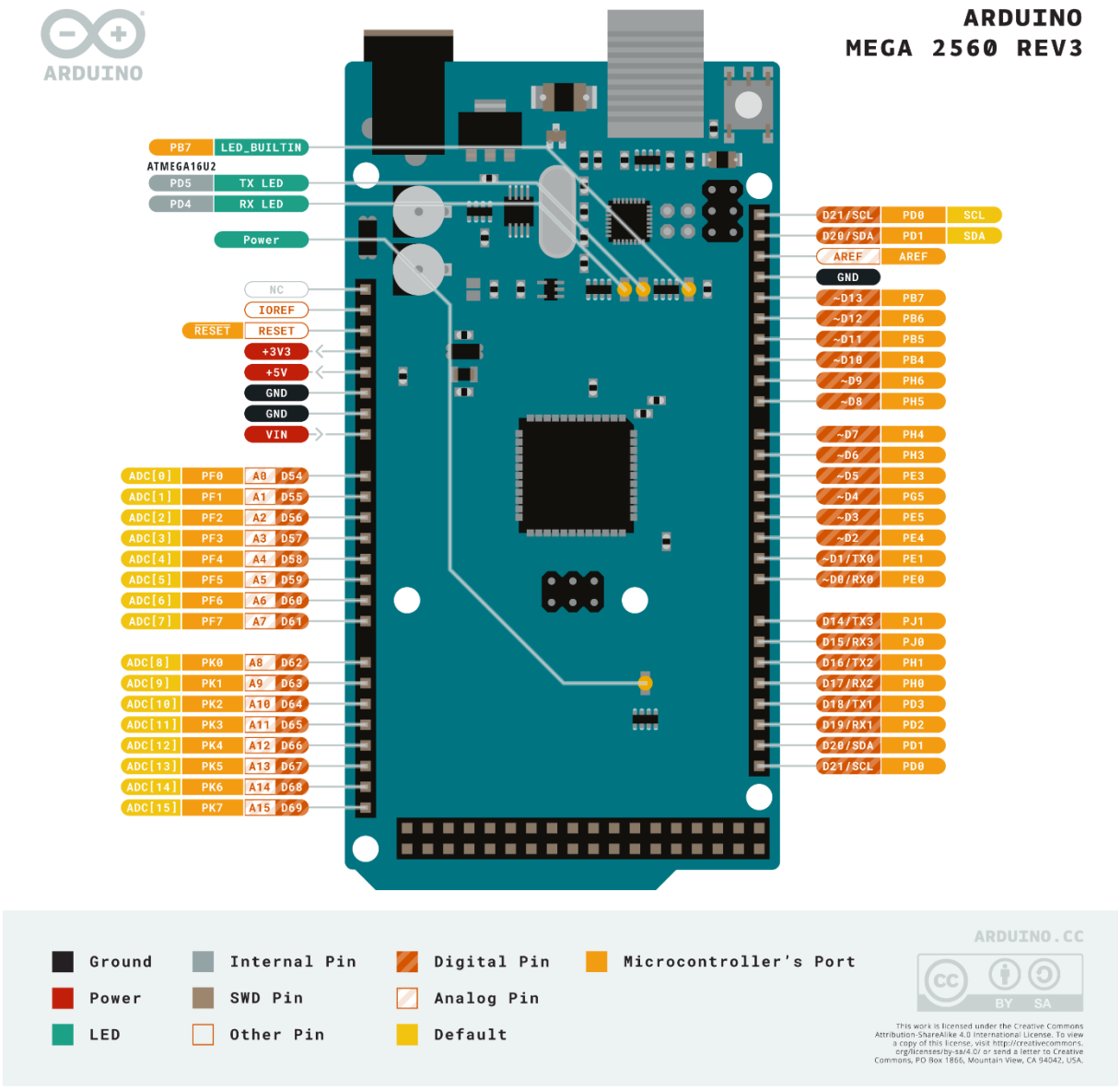

Abbildung 2-3: Pinbelegung Arduino Mega 2560 Rev3 (5)

Es werden offiziell für die Eingangsspannung 7-12 Volt empfohlen. Die Logikspannung des Arduinos beträgt 5 Volt, so dass man beachten muss, dass die zusätzlich angeschlossenen Geräte auch mit 5 Volt Logik funktionieren. Ansonsten muss man einen Spannungsteiler nutzen. Der maximale Strom für I/O-Pins sind 20mA und bei 3,3V sind 50mA.

Am Zeitpunkt Juli 2022 ist ein Arduino Mega Rev3 für 42,00€ erhältlich im offiziellen Arduino Shop. (5), (6)

2.3.2 Accelerometer-Gyroskop-Magnetometer BNO055

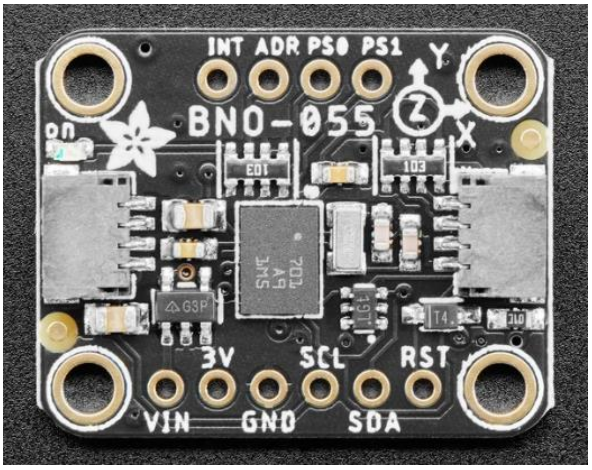

Abbildung 2-4: Foto von BNO055 im Adafruit Shop (7)

Als Sensor für die Ermittlung der absoluten Orientierung wird der BNO055 benutzt, der im ursprünglichen Roboter verbaut war. Der BNO055 von Bosch ist ein 9-Achsen absoluter Orientierungssensor und besteht aus einem ARM Cortex-M0, MEMS-Akzelerator, Magnetometer und Gyroskop. In unserem Fall wird eine Interface-Platine von Adafruit benutzt mit einem STEMNA QT 4-Pin Anschluss von Adafruit, der eine Plug-n-Play Verbindung ermöglicht mit I2C Unterstützung zwischen dem Sensor und dem Arduino Mega. Diese Verbindung ist auch kompatibel mit dem Qwiic Anschluss vom Hersteller Sparkfun.

Der Sensor kann folgendes mit einer Frequenz von 100Hz in 3 Achsen ausgeben: Die absolute Orientierung in der Quaternion oder in Euler Winkeln, die angulare Geschwindigkeit in rad/s, die Beschleunigung bestehend aus linearer Bewegung und Schwerkraft, die lineare Beschleunigung und die Beschleunigung der Schwerkraft jeweils in m/s^2. Mit 20Hz kann in 3 Achsen die magnetische Feldstärke in µT gemessen werden. Schlussendlich ist es auch noch möglich mit 1Hz die Umgebungstemperatur in °C auszulesen. Diese Platine von Adafruit erlaubt eine Spannungsversorgung mit 3,3 Volt und 5 Volt und auch die Logik verträgt beide Spannungen, so dass man den Sensor ohne Bedenken an den Arduino Mega anschließen kann.

Beim STEMNA QT Anschluss steht das rote Kabel für VCC, rot für GND, blau für SDA und gelb für SCL.

Am Zeitpunkt Juli 2022 ist ein Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055 - STEMMA QT / Qwiic für 29,95€ erhältlich im offiziellen Adafruit Shop. (7), (8)

2.3.3 Motortreiber L298N

Hier handelt es sich um den ursprünglichen Motortreiber mit doppelter H-Brücke, der Analogeingänge in PWM regulierte Ausgänge für die Motoren umwandelt. Der Treiber besitzt 2 PWM unabhängige Ausgänge, um Motoren anzusteuern. Die Eingangsspannung kann 3,2V

bis 40V Gleichspannung betragen und der Ausgang kann 5V bis 35V betragen. Zusätzlich gibt es auch noch einen 5V Ausgang mit dem man zum Beispiel einen Arduino versorgen könnte. Pro Kanal können 2A kontinuierlich ausgegeben werden und 3A Spitzenstrom.

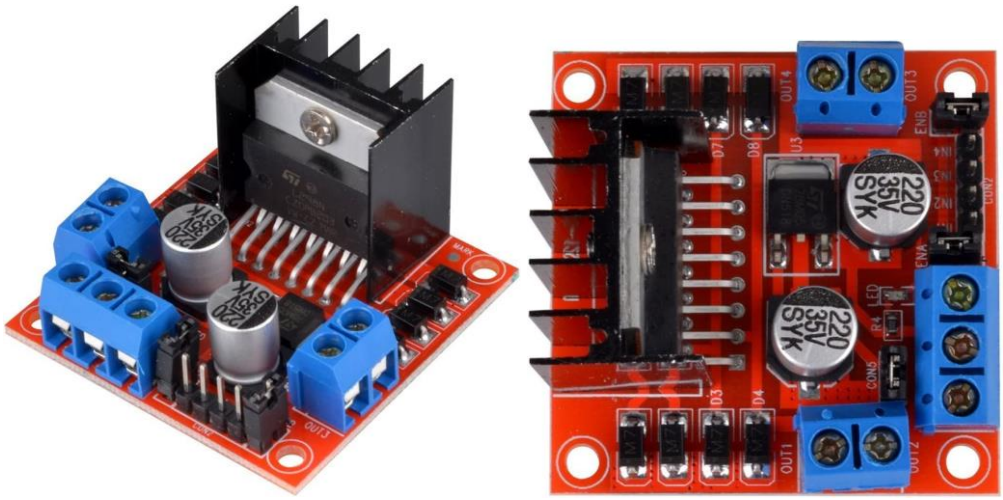

Abbildung 2-5: Fotos von LN298N Motortreiber auf Amazon.de (9)

Am Zeitpunkt Juli 2022 ist ein L298N für 6,99€ erhältlich auf Amazon.de. (9), (10)

2.3.4 DC Boost Converter XL6009

Hierbei handelt es sich um den ursprünglichen DC/DC Boost Konverter, um die 5V Versorgungsspannung des Akkus in 12V Ausgangspannung für den Motortreiber zu verwandeln. Der XL6009 ist vonnöten, denn der Akku kann nur 5V erzeugen. Dieses Modul ermöglicht es, eine schwankende Eingangsspannung zwischen 5V und 9V in gleichmäßige 12V DC zu konvertieren.

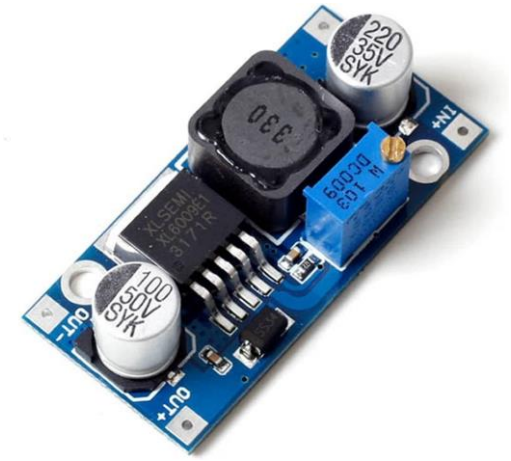

Abbildung 2-6: Foto von DC Boost Converter XL6009 auf Amazon.de (11)

Am Zeitpunkt Juli 2022 ist ein XL6009 für 3,99€ erhältlich auf Amazon.de. (11)

2.3.5 Litionite Falcon Mini Powerbank

Als Stromversorgung für den Roboter wird die ursprüngliche Lösung beibehalten. Es handelt sich dabei um die Litionite Falcon Mini, eine Lithium-Polymer Powerbank mit einer Kapazität von 12000mAh. Die Batterie lässt sich über Mikro-USB wieder aufladen und besitzt als Ausgang 2 USB-A Anschlüsse mit 5V und 2,4A und ein USB-A mit alternativ 5V 3A, 9V 2A und 12V 1,5A. Das Gewicht liegt bei 320g. Allerdings lassen sich in unserem Fall nicht mehr als 5V nutzen, da für eine höhere Spannung eine serielle Kommunikation über USB erforderlich ist und hier nur ein Kabel mit positivem und negativem Pol anschlossen wird.

Abbildung 2-7: Foto von Litionite Falcon Mini auf Amazon.de (12)

Am Zeitpunkt Juli 2022 wird dieses Modell nicht mehr angeboten, aber vergleichbare Alternativen sind für ungefähr 40€ erhältlich. (12)

2.3.6 nRF24L01+

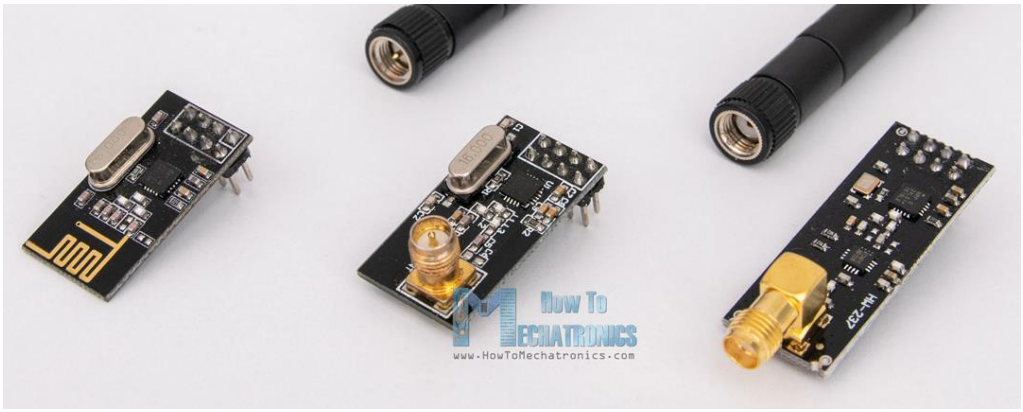

Abbildung 2-8: Foto der 3 Varianten des nRF24L01+ von HowToMechatronics.com (13)

Für die kabellose Kommunikation zwischen dem Roboter und der Fernbedienung, kommt das nRF24L01+ Modul zum Einsatz. Das Modul kann sowohl senden als auch empfangen und arbeitet auf dem 2,4GHz Band mit einer Baudrate wählbar zwischen 250kbps, 1Mbps und

2Mbps. Die Sendestärke kann 0, -6, -12 oder -18dBm sein. In freier Umgebung kann das Modul mit der niedrigsten Baudrate eine Reichweite von bis zu 100 Metern erreichen. Es stehen 125 mögliche Kanäle zur Auswahl, die von 2,401 bis 2,525GHz reichen und es kann mit 6 anderen Modulen gleichzeitig kommuniziert werden, wie in Abbildung 2-9 dargestellt

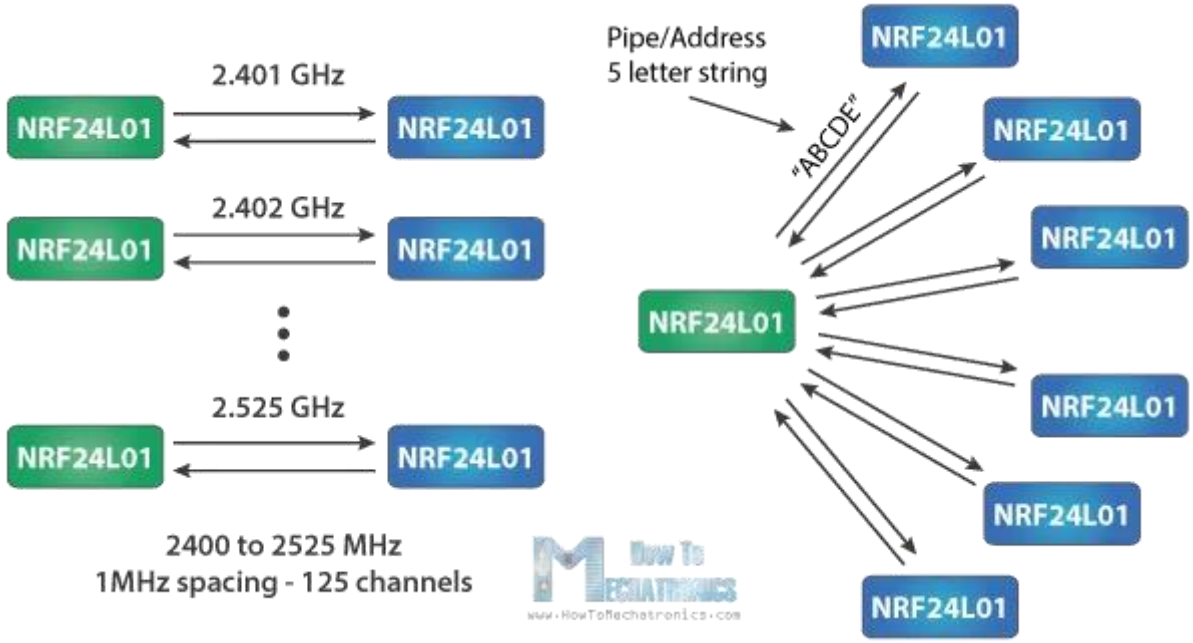

Abbildung 2-9: Schema der Kanäle des nRF24L01+ von HowToMechatronics.com (13)

Der Stromverbrauch des Moduls beträgt nur 12mA während der Übertragung, 26 µA im Standby und 900nA im Powerdown Modus. Durch den niedrigen Stromverbrauch, kombiniert mit dem kleinen Formfaktor und guter Latenz im Vergleich zu den meisten Bluetooth und Wifi Modulen, ist der nRF24L01+ ideal für unsere Zwecke geeignet. Die Eingangsspannung soll 3,3 Volt betragen, aber die Logik des Moduls ist mit den 5 Volt des Arduinos kompatibel. Es gibt noch zwei weitere gängige Varianten dieses Moduls, eine mit einem SMA Anschluss für eine Antenne für eine erhöhte Reichweite und eine andere mit einer Antenne und einem RFX2401C Chip mit einem Leistungsverstärker und einem Rauscharmen Verstärker für eine Reichweite bis zu einem Kilometer. Für unsere Zwecke sollte die Basisvariante aber genügen.

Das Modul kommuniziert mit dem Arduino über das SPI Protokoll und für die Kommunikation zwischen zwei Modulen wird als Datalink Layer das Enhanced Shockburst Protokoll eingesetzt. Dies ermöglicht eine variable Sendegröße von 1 bis 32 Bytes, gibt jedem Paket eine Nummer, um neue Pakete zu erkennen und erlaubt es dem Empfänger eine Bestätigung des Empfangs anzufordern. In Abbildung 2-10 wird das Paket dieses Protokolls dargestellt.

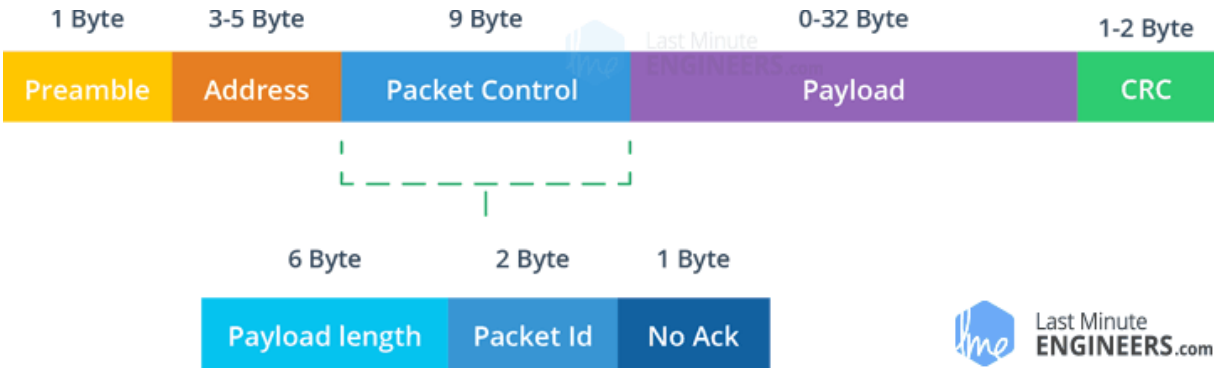

Abbildung 2-10: Schema eines Pakets des nRF24L01+ von LastMinuteEngineers.com (14)

In Abbildung 2-11 wird die Pinbelegung des nRF24L01+ dargestellt, wobei der IRQ Pin bei unserem Projekt ungenutzt bleibt.

Abbildung 2-11: Pinbelegung des nRF24L01+ von HowToMechatronics.com (13)

Am Zeitpunkt Juli 2022 ist ein nRF24L01+ für ungefähr 2,00€ pro Stück erhältlich auf Amazon.de. (13), (14)

2.3.7 JGA25-371 Motoren

Ursprünglich war der Motor JGA25-370 verbaut im Roboter. Da dieser aber keinen Encoder besitzt und so eine Geschwindigkeits- oder Positionsbestimmung nicht möglich ist, muss ein Motor her, der mit der gleichen Spannung von 12V funktioniert und den gleichen Formfaktor besitzt für eine möglichst einfache Montage wie in Abbildung 2-13 erkennbar ist. Der JGA25-371 erfüllt diese Anforderungen, weil es sich um praktisch den gleichen Typ Motor wie vorher handelt, jedoch mit dem Zusatz eines inkrementalen Drehgebers. Genau wie beim vorherigen Motor sind auch bei diesem die gleiche Motorhalterung, Räder und Schrauben enthalten wie in Abbildung 2-12 dargestellt.

Abbildung 2-12: Foto von JGA25-371 mit Zubehör auf Amazon.de (15)

Abbildung 2-13: Foto von Größenvergleich zwischen JGA25-370 (oben) und JGA25-371 (unten)

Die beiliegenden Räder haben einen Durchmesser von 6,5cm, der Radumfang beträgt also $\pi \cdot 6{,}5cm = 20{,}42cm$

Allerdings wurde sich dieses Mal für die 200rpm Variante anstatt wie vorher verbaute 1360rpm, denn mit 200 Umdrehungen pro Minute könnten theoretisch $\frac{200 \cdot 20{,}42}{60} = 68$cm pro Sekunde abgefahren werden, was mehr als ausreichend sein sollte. Durch die niedrigere maximale Drehzahl sollte auch ein höheres Drehmoment erreicht werden. Es wird allerdings angemerkt, dass in der Präsentation des vorherigen Roboters von einem 245rpm Motor die Rede war, aber ein 1360rpm Motor verbaut vorgefunden wurde.

Die 200rpm Variante des nun verbauten Motors hat laut Spezifikation ein Getriebeverhältnis von 21,3. Er ist ausgelegt für einen Strom von 0,16A und ein Drehmoment von 0,4kg·cm. Bei blockiertem Rad können allerdings 0,6A fließen und ein Drehmoment von 2kgcm auftreten. Ohne Last kann die Drehzahl 280rpm erreichen und es soll ein Strom von 0,04A fließen.

Der Encoder ist wie bereits erwähnt ein inkremental Drehgeber mit 2 Kanälen, so dass eine Bestimmung der Drehrichtung durch die Reihenfolge der Signale beider Kanäle erfolgen kann. Jeder Kanal gibt 11 Signale pro Motorumdrehung aus. In der folgenden Abbildung 2-14 ist der Drehgeber des Motors gut ersichtlich durch die Magnetscheibe, die am Ende der Motorwindung befestigt ist und durch einen Sensor pro Phase gemessen wird.

Abbildung 2-14: Foto der Encoderseite des JGA25-371 (15)

Der Motor wird mit 12 Volt Gleichspannung betrieben. Ein Verringern dieser Spannung senkt die Drehzahl. Mit negativer Spannung kehrt sich auch die Drehzahl um. Der Encoder benötigt eine separate Spannungsversorgung, die 3,3 bis 5 Volt betragen muss, also problemlos vom Arduino gespeist werden kann. Allerdings dürfen hier nicht die Pole vertauscht werden wie bei der Motorspannung. Anschließend gibt es für beide Kanäle jeweils einen Ausgang mit einer Logikspannung, die der Spannungsversorgung des Encoders entspricht. In der Abbildung 2-15 sieht man die Pinbelegung noch einmal erläutert.

Abbildung 2-15: Foto vom Motor JGA25-371 mit beigelieferten Kabel und Pinbelegung (15)

Am Zeitpunkt Juli 2022 ist ein Motor JGA25-371 mit 200rpm für ungefähr 28,00€ pro Stück erhältlich auf Amazon.de. Es ist allerdings empfehlenswert 3 Stück zu beschaffen für den Fall eines technischen Defektes. So war zum Beispiel einer der alten Motoren defekt und der exakt gleiche Motor nicht mehr verfügbar, wenn man nun den ursprünglichen Roboter wieder rekonstruieren hätte wollen. (15)

2.3.8 Verkabelung des Roboters

Nachdem nun die einzelnen Teile herausgesucht worden sind, müssen die Teile miteinander verbunden und am Roboter befestigt werden. Das resultiert dann in der Abbildung 2-16 im fertigen Roboter mit dem ursprünglichen 3D-gedruckten Gestell.

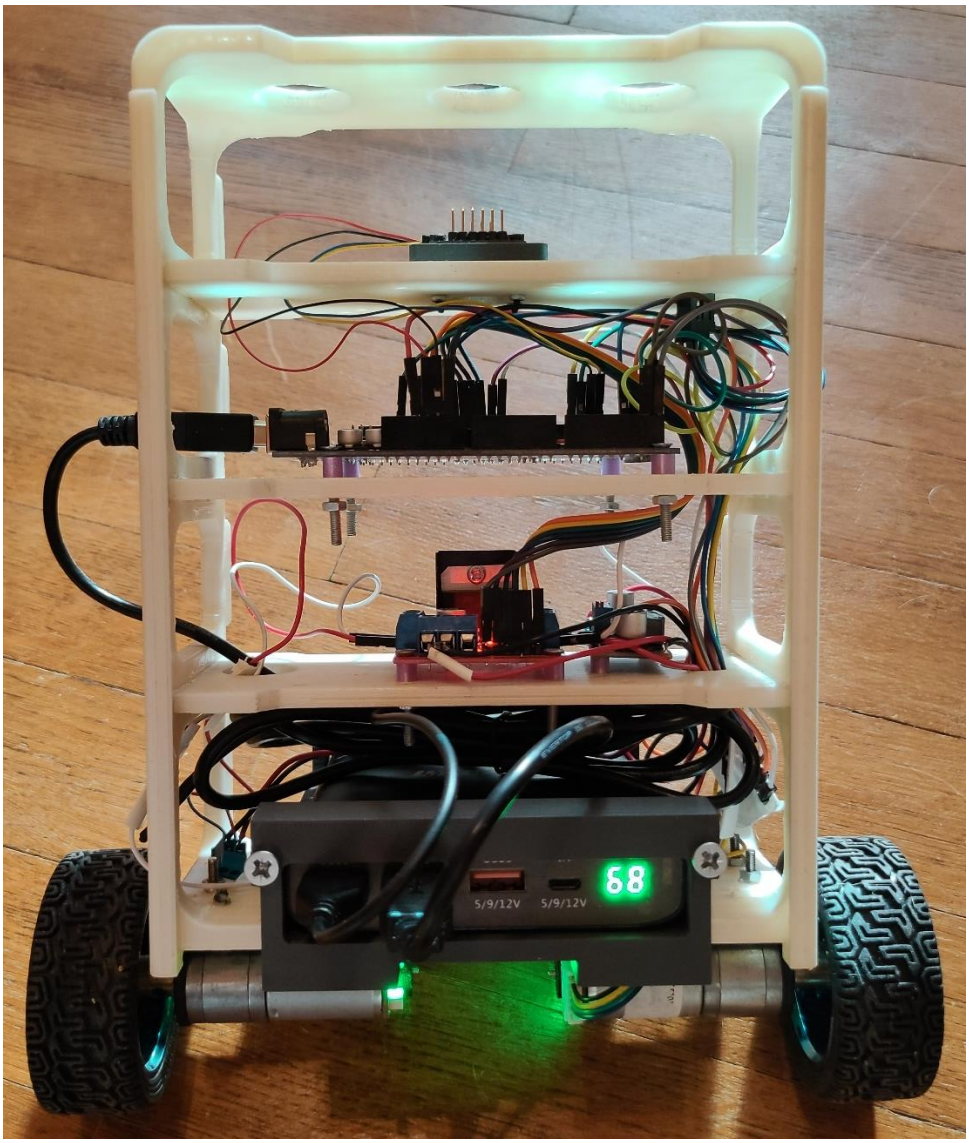

Abbildung 2-16: Foto vom fertig verkabelten Roboter

Bei der folgenden Abbildung 2-17 wird die Verkabelung des Roboters schematisch dargestellt.

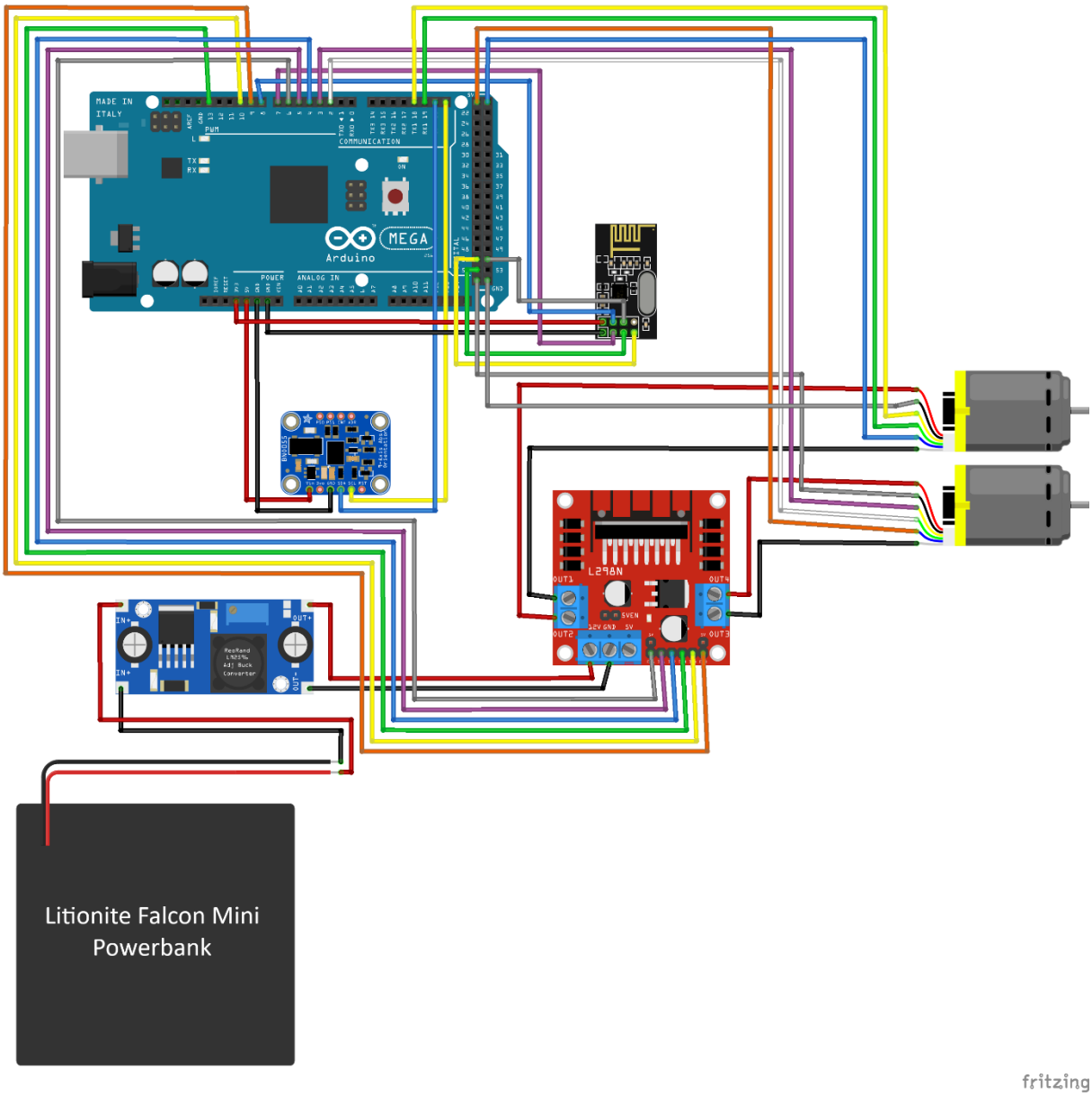

Abbildung 2-17: Schematische Darstellung der Verkabelung des Roboters

Da alle an den Arduino Mega angeschlossene Geräte 5V Logikspannung vertragen, ist kein Spannungsteiler nötig.

Für die I2C Kommunikation zwischen dem BNO055 und dem Arduino Mega müssen SDA und SCL an die digitalen Pins 20 und 21 angeschlossen werden. Bei beiden Encodern muss jeweils ein Kanal ebenfalls an Interrupt Pins verbunden werden. Die Interrupt Pins des Arduino Megas sind die digitalen Pins 2, 3, 18, 19, 20 und 21, wobei 20 und 21 schon für I2C reserviert sind.

In der Tabelle 2-1 sind alle Kabelverbindungen des Roboters noch einmal genau aufgelistet.

Tabelle 2-1: Auflistung der Kabelverbindungen des Roboters

Anfang des Kabels	**Ende des Kabels**
BNO055 VIN	Arduino Mega 5V Ausgang
BNO055 GND	Arduino Mega GND
BNO055 SDA (I2C)	Arduino Mega digital Pin 20
BNO055 SCL (I2C)	Arduino Mega digital Pin 21
nRF24L01+ VCC	Arduino Mega 3,3V Ausgang
nRF24L01+ GND	Arduino Mega GND
nRF24L01+ CSN (SPI)	Arduino Mega digital Pin 8
nRF24L01+ CE (SPI)	Arduino Mega digital Pin 7
nRF24L01+ SCK (SPI)	Arduino Mega digital Pin 52
nRF24L01+ MOSI (SPI)	Arduino Mega digital Pin 51
nRF24L01+ MISO (SPI)	Arduino Mega digital Pin 50
L298N 12V Eingang	XL6009 OUT+
L298N GND	XL6009 OUT-
L298N ENA	Arduino Mega digital Pin 6
L298N ENB	Arduino Mega digital Pin 9
L298N IN1	Arduino Mega digital Pin 5
L298N IN2	Arduino Mega digital Pin 4
L298N IN3	Arduino Mega digital Pin 13
L298N IN4	Arduino Mega digital Pin 10
L298N OUT1	1. JGA25-371 Motor negativer Pol
L298N OUT2	1. JGA25-371 Motor positiver Pol
L298N OUT3	2. JGA25-371 Motor negativer Pol
L298N OUT4	2. JGA25-371 Motor positiver Pol
Powerbank USB-A B 5V+	XL6009 OUT+
Powerbank USB-A 5V-	XL6009 OUT-
Powerbank USB-A 5V	Arduino Mega USB-B
1. JGA25-371 Encoder positiver Pol	Arduino Mega 5V Ausgang
1. JGA25-371 Encoder negativer Pol	Arduino Mega GND
1. JGA25-371 Encoder Kanal A	Arduino Mega digital Pin 18
1. JGA25-371 Encoder Kanal B	Arduino Mega digital Pin 19
2. JGA25-371 Encoder positiver Pol	Arduino Mega 5V Ausgang
2. JGA25-371 Encoder negativer Pol	Arduino Mega GND
2. JGA25-371 Encoder Kanal A	Arduino Mega digital Pin 3
2. JGA25-371 Encoder Kanal B	Arduino Mega digital Pin 2

2.4 Fernbedienung Hardware

Nachdem wir uns mit der Hardware des Roboters befasst haben, wenden wir uns nun dem Kontroller zu.

2.4.1 Arduino Uno Rev3

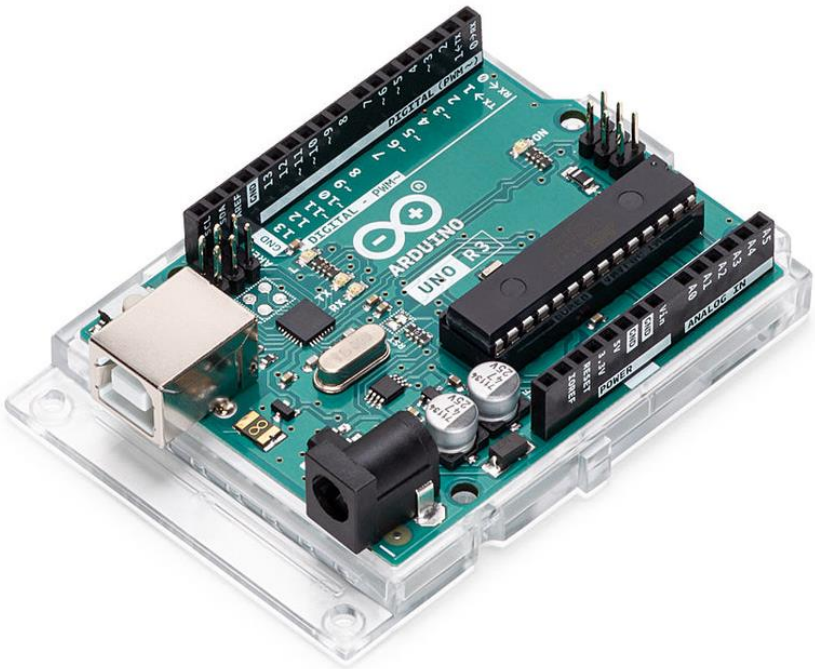

Abbildung 2-18: Foto von Arduino Uno Rev3 im Arduino Shop (16)

Auch in der Fernbedienung wird ein Mikrokontroller von Arduino verbaut. In diesem Fall der Arduino Uno Rev3 der ursprünglich im Roboter verbaut war. Da die Hardwareanforderungen an die Fernbedienung weit geringer sind als an den Roboter, reicht der Uno mit seinen wenigeren Anschlüssen und geringerer Speicherkapazität gegenüber dem Arduino Mega aus. Mit seinem dadurch geringeren Formfaktor ist der Arduino Uno ideal für die Fernbedienung geeignet. Die gleichen Arduino typischen Vorteile wie beim Mega treffen auch hier zu.

Der Mikroprozessor des Unos ist der ATmega328P. Dieser Chip ist ein AVR 8 Bit Chip mit einer RISCV Architektur und einem 16MHz Keramik Resonator. Die Flashspeichergröße beträgt 32 KB mit 2KB SRAM und 1KB EEPROM. Der Uno besitzt auch einen USB-Port, der eine einfache Verbindung zu einem PC ermöglicht, einen DC Stromanschluss, einen ICSP Header und einen Reset Schalter. Wie in Abbildung 2-19 gezeigt, besitzt der Arduino 14 Digitale Input/Output Pins, wobei 6 als PWM Output benutzt werden können, und 6 5V Analog Inputs mit 10 Bit Auflösung, also mit 1024 verschiedenen Werten. Zusätzlich gibt es einen 5V Pin, einen 3,3V Pin und 3 Erdungspins.

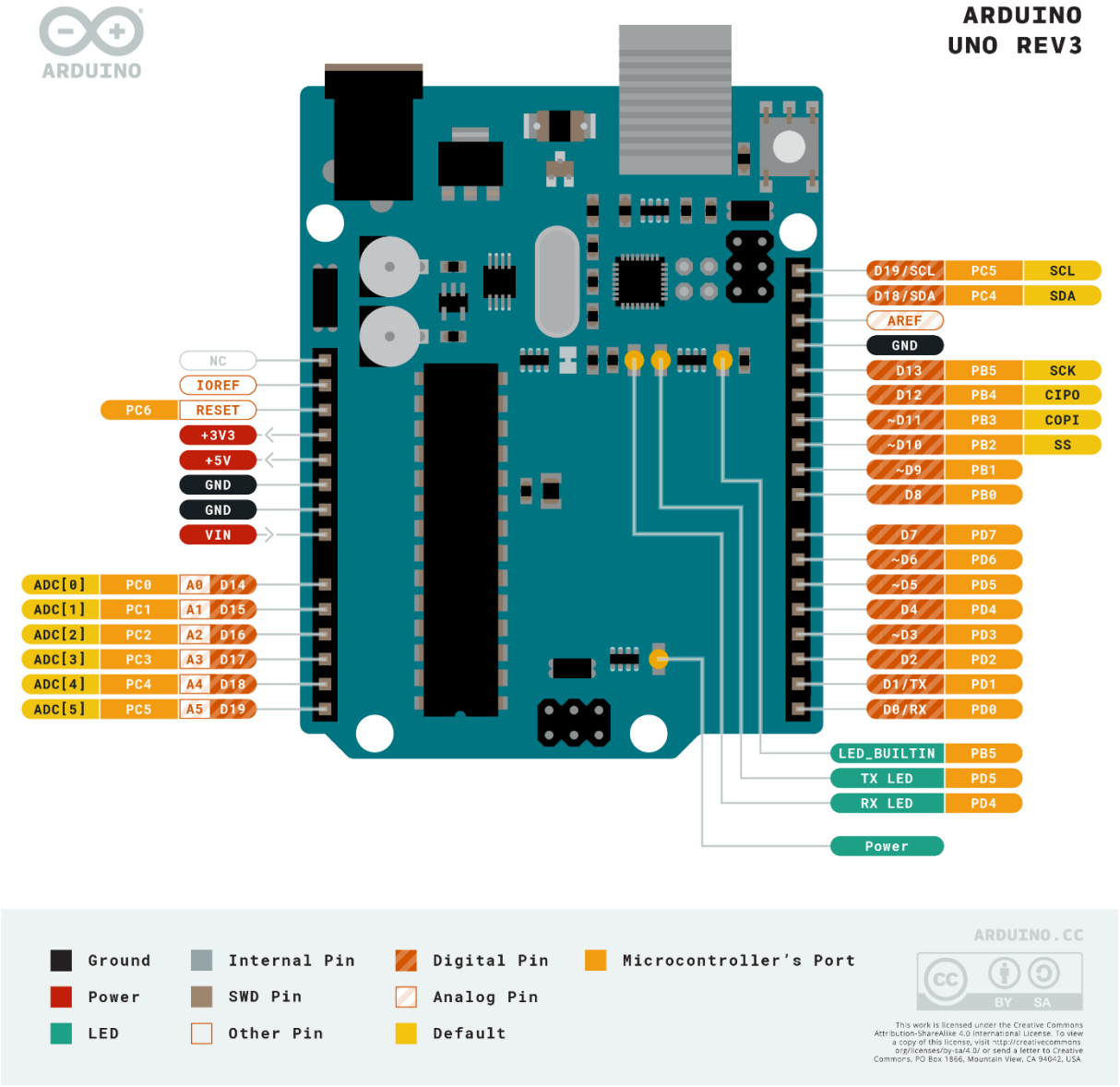

Abbildung 2-19: Schema der Pinbelegung des Arduino Uno Rev3 (16)

Es werden offiziell für die Eingangsspannung 7-12 Volt empfohlen. Die Logikspannung des Arduinos beträgt 5 Volt, so dass man beachten muss, dass die zusätzlich angeschlossenen Geräte auch mit 5 Volt Logik funktionieren. Ansonsten muss man einen Spannungsteiler nutzen. Der maximale Strom für I/O-Pins ist 20mA und bei 3,3V 50mA.

Am Zeitpunkt Juli 2022 ist ein Arduino Mega Rev3 für 24,00€ erhältlich im offiziellen Arduino Shop. (16), (17)

2.4.2 nRF24L01+

Wie beim Roboter, wird auch beim Kontroller logischerweise ein nRF24L01+ Modul zum Empfangen und Senden von Daten mit dem Roboter benutzt. (Siehe 2.3.6 nRF24L01+)

2.4.3 Joystick

Abbildung 2-20: Foto des Joysticks auf Amazon.de (18)

Als Eingabe für den Kontroller wird ein generischer analoger Joystick mit 5V Eingangsspannung genutzt. Der Joystick besitzt eine X- und Y-Achse dessen Ausgabe zwischen 0V bis 5V liegt. Dabei ist die Ausgabe beider Achsen 2,5V im Ruhezustand, wenn der Joystick sich in der Mitte befindet.

Dieser Joystick funktioniert ähnlich wie bei einer PS2 (Playstation 2) Fernbedienung. Eine Feder sorgt dafür, dass der Steuerknüppel sich automatisch wieder in der Mitte zentriert, wenn man ihn loslässt. Jede Achse besitzt einen Potentiometer, der den Spannungswert der jeweiligen Achse mit der Neigung des Joysticks auf der Achse variiert. Wird der Joystick diagonal bewegt, werden auf beiden Achsen eine Abweichung von der Mitte registriert. Wenn man den Joystick eindrückt, wird ein Hebel auf einen Knopf gepresst, der den Signalpegel auf dem anhängenden Pin ändert. In Abbildung 2-21 sind die vorher beschriebenen Potentiometer und das Koordinatensystem des Joysticks dargestellt.

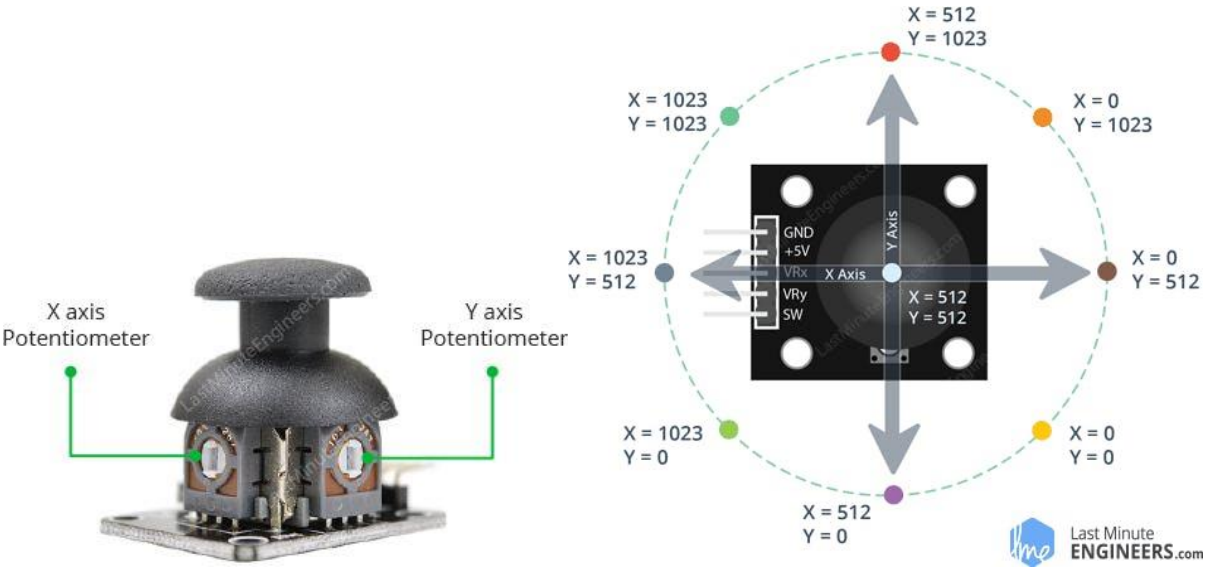

Abbildung 2-21: Foto mit Potentiometer des Joysticks (links) und Schema des Koordinatensystems von LastMintuteEngineers.com (rechts) (19)

Am Zeitpunkt Juli 2022 ist so ein Joystick für 3,69€ erhältlich auf Amazon.de. (18) (19)

2.4.4 SSD1306 0,96“128x64 OLED Anzeige

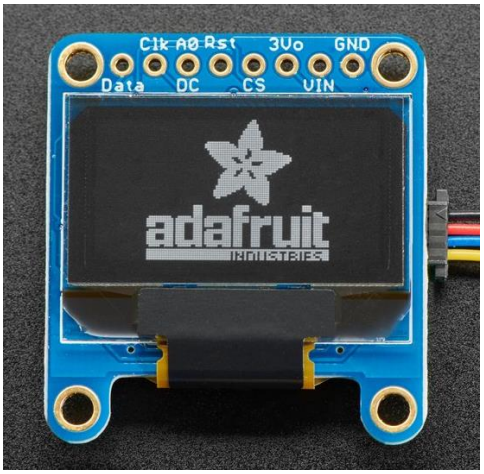

Abbildung 2-22: Foto des SSD1306 OLED im Adafruit Shop (20)

Als Anzeigeelement kommt der SSD1306 von Adafruit zum Einsatz. Es handelt sich dabei um einen monochromatischen OLED Bildschirm mit einer Auflösung von 128 auf 64 Pixeln und 0,96 Zoll Diagonale. Die Anzeige besteht aus weißen Pixeln, die individuell angesteuert werden und ohne Hintergrundbeleuchtung ihr eigenes Licht ausgeben, was zu einem besonders hohen Kontrast und damit guter Leserlichkeit führt. Die Abwesenheit einer Hintergrundbeleuchtung führt auch zu einem besonders niedrigen Stromverbrauch. Dieser Verbrauch hängt dadurch von der Anzahl der leuchtenden Pixel ab, aber in der Praxis sollen ungefähr 20mA erwartet werden. Die Anzeige hat eine Aktualisierungsrate von 60Hz. Die Kommunikation mit dem Arduino kann sowohl über SPI als auch I2C erfolgen. Zusätzlich verträgt die Logik sowohl 3,3V als auch 5V. Die Anzeige benötigt zwar eine 5V Spannung, ein interner Boost Konverter ermöglicht jedoch auch eine 3,3V Spannungsversorgung. Die OLED Anzeigeelemente können jedoch auch separat durch einen Pin mit 7 bis 9 Volt versorgt werden.

Am Zeitpunkt Juli 2022 ist eine neuere Version des SSD1306 für 17,50€ erhältlich im offiziellen Adafruit Shop. (20), (21)

2.4.5 6xAA Batterienhalter

Abbildung 2-23: Foto eines 6xAA Batterienhalters auf Amazon.de (22)

Als Stromversorgung für die Fernbedienung wird ein generischer Halter für 6 1,5V Batterien benutzt für einen maximalen Ausgang von 9V, um den Arduino zu versorgen. Da der Energieverbrauch der Fernbedienung viel niedriger ist als der Roboter mit seinen zwei Motoren, sollten die Batterien sehr lange halten, bevor sie nicht mehr die nötige Spannung liefern. Der Verbau eines aufladbaren Akkus ist also in diesem Fall unnötig.

Am Zeitpunkt Juli 2022 ist so ein ähnlicher Batteriehalter für 8,47€ im Doppelpack erhältlich auf Amazon.de. (22)

2.4.6 Klemmbaustein Hülle

Als Hülle oder Halterung zur Befestigung wurde sich für eine Konstruktion aus Klemmbausteinen entschieden, denn die Umsetzung ist sehr einfach und Klemmbausteine haben generell eine recht hohe Qualität. Außerdem ermöglicht die modulare Bauweise nachher eine einfache Anpassung, um Änderungen oder Erweiterungen vorzunehmen. Zusätzlich ist das Gewicht auch gering, da Klemmbausteine aus Plastik bestehen und innen hohl sind. Im Vergleich mit einer 3D gedruckten Version ist diese Lösung minimal weniger stabil und ein speziell entworfenes Modell ermöglicht vielleicht noch geringfügige Platzeinsparungen. Die Bausteine erlauben nur eine Anpassung der Dimensionen in vordefinierten Schritten und es ist daher nicht möglich, die Module nur mit reinen Klemmsteinen ohne Spielraum zu verankern. Da Schrauben, ohne die Steine zu beschädigen nicht passen, wurde auf doppelseitigen Klebestreifen zurückgegriffen, um die Module zu fixieren.

Bei dem nRF24L01+ muss allerdings die Platzierung bedacht werden, denn je nach Position in dem Kontroller wurde die Verbindungsqualität stark beeinträchtigt. In den Abbildungen 2-24 und 2-25 ist die Fernbedienung mit fertiger Hülle aus Klemmbausteinen in unterschiedlichen Ansichten gezeigt.

Abbildung 2-24: Foto der Fernbedienung von vorne (rechts) und hinten (links)

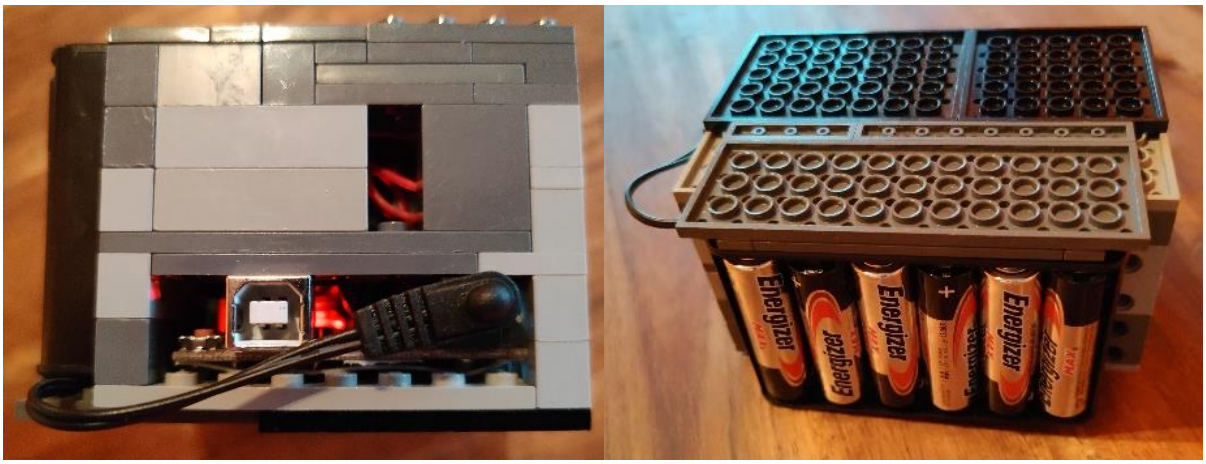

Abbildung 2-25: Foto der Anschlüsse der Fernbedienung (links) und der Unterseite (rechts)

2.4.7 Verkabelung der Fernbedienung

Zuerst werden die vorher aufgezählten Module miteinander verbunden. Anschließend wird das Ganze im Inneren der Hülle befestigt. In der Abbildung 2-26 sieht man im Inneren der geöffneten Fernbedienung dessen Verkabelung.

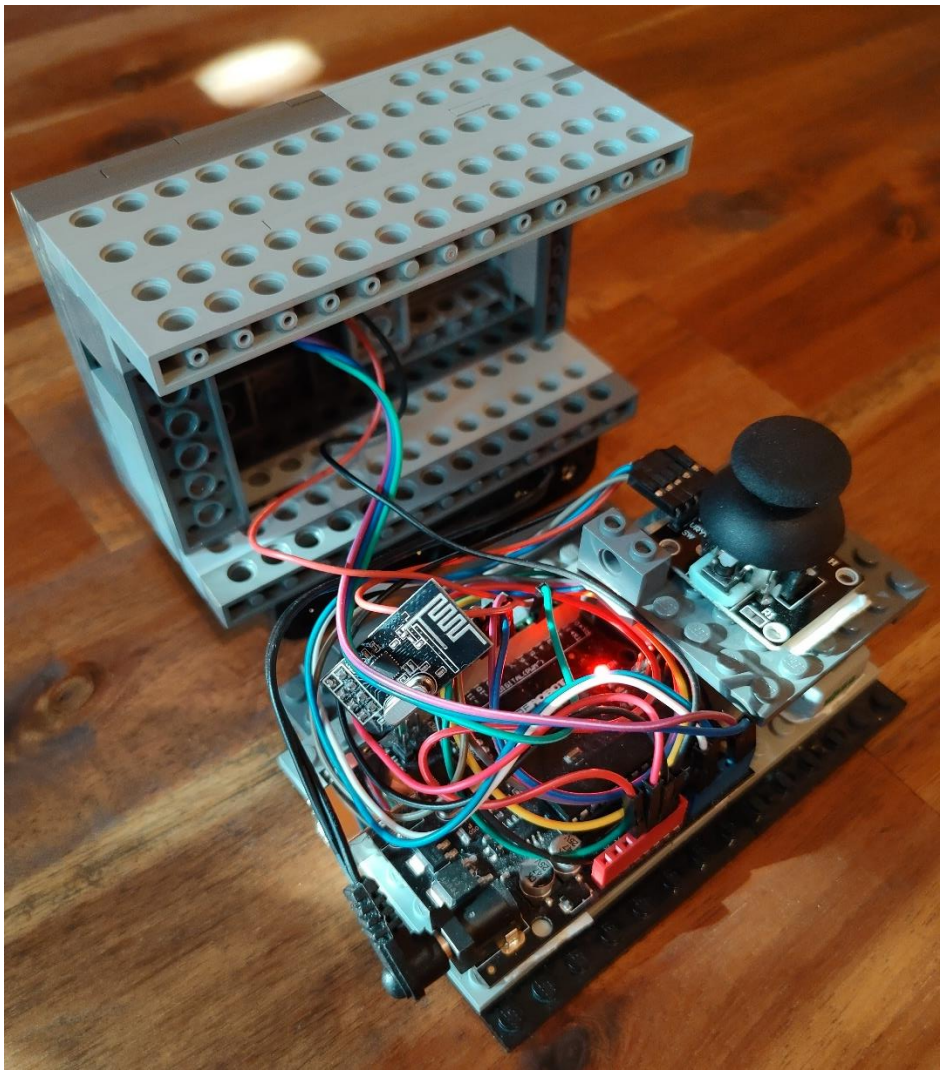

Abbildung 2-26: Foto der fertigen Verkabelung im Inneren der Fernbedienung

Bei der folgenden Abbildung 2-27 wird die Verkabelung der Fernbedienung schematisch dargestellt.

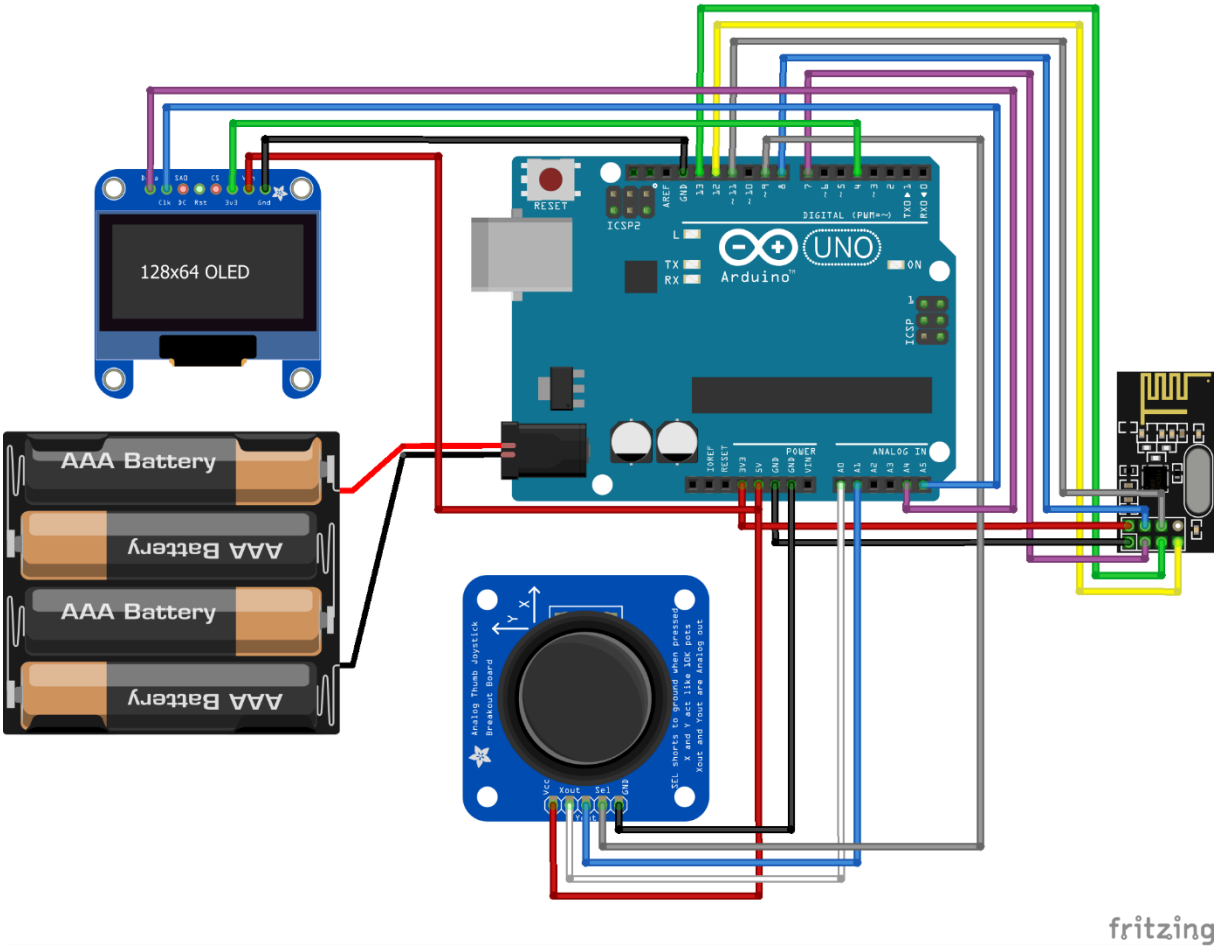

Abbildung 2-27: Schematische Darstellung der Verkabelung der Fernbedienung

Da alle an den Arduino Uno angeschlossene Geräte 5V Logikspannung vertragen, ist kein Spannungsteiler nötig.

Für die I2C Kommunikation zwischen der SSD1306 Anzeige und dem Arduino Uno müssen die analogen Pins A4 und A5 benutzt werden.

In der Tabelle 2-2 sind alle Kabelverbindungen der Fernbedienung noch einmal genau aufgelistet.

Tabelle 2-2: Auflistung der Kabelverbindungen der Fernbedienung

Anfang des Kabels	**Ende des Kabels**
SSD1306 VIN	Arduino Uno 5V Ausgang
SSD1306 GND	Arduino Uno GND
SSD1306 Rst	Arduino Uno digital Pin 4
SSD1306 SDA/Data (I2C)	Arduino Uno analog Pin A4
SSD1306 SCL/Clk (I2C)	Arduino Uno analog Pin A5
nRF24L01+ VCC	Arduino Uno 3,3V Ausgang
nRF24L01+ GND	Arduino Uno GND
nRF24L01+ CSN (SPI)	Arduino Uno digital Pin 8
nRF24L01+ CE (SPI)	Arduino Uno digital Pin 7
nRF24L01+ SCK (SPI)	Arduino Uno digital Pin 13
nRF24L01+ MOSI (SPI)	Arduino Uno digital Pin 11
nRF24L01+ MISO (SPI)	Arduino Uno digital Pin 12
Joystick 12V Eingang	Arduino Uno 5V Ausgang
Joystick GND	Arduino Uno GND
Joystick VRX	Arduino Uno analog Pin A0
Joystick VRY	Arduino Uno analog Pin A1
Joystick SW	Arduino Uno digital Pin 9
6xAA Batterienhalter 9V	Arduin Uno Gleichstrombuchse

2.5 Gesamtübersicht

In der folgenden Tabelle 2-3 werden die einzelnen Teile noch einmal aufgelistet mit ihrer Logikspannung und Preis.

Tabelle 2-3: Auflistung der Bauteile des Projektes mit Logikspannung und Preis

Bauteil	**Logikspannung**	**Preis**
Arduino Mega 2560 Rev3	5V	42,00€
Adafruit BNO055	3-5V	29,95€
L298N Motortreiber	5V	6,99€
DC Boost Converter XL6009	/	3,99€
Akku mit 12000mhA	/	40,00€
nRF24L01+	5V	2x2,00€
JGA25-371 Motor	3,3-5V	3x28,00€
Arduino Uno Rev3	5V	24,00€
Joystick	5V	3,69€
SSD1306 OLED	3,3-5V	7,50€
6xAA Batterienhalter	/	8,47€
Gesamt	/	254,59€

Der Gesamtpreis für Roboter und Fernbedienung ohne Halterung beträgt 254,59€.

Das Gesamtgewicht des Roboters beträgt ungefähr 1,1kg.

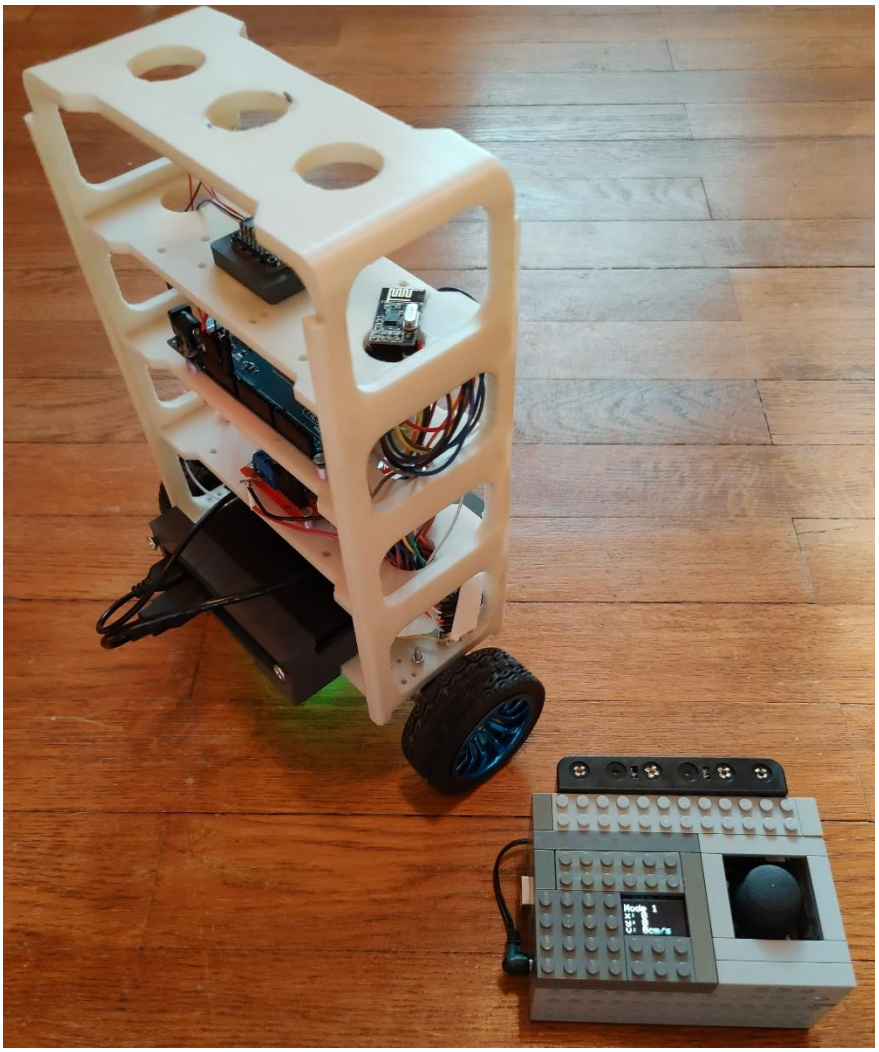

Abbildung 2-28: Foto des fertigen Roboters und Fernbedienung im Betrieb

3 Theorie

In diesem Teil werden zuerst die Regelung des Gleichgewichts und dann der Fortbewegungs- und Wendegeschwindigkeit theoretisch untersucht und mögliche Regler vorgeschlagen. Danach wird die Regelung der abgefahrenen Distanz und der Richtung des Roboters analysiert. Dabei handelt es sich um eine qualitative Analyse um einen geeigneten Regler Typ zu bestimmen. Zu beachten ist außerdem, dass bei der Regelung des Gleichgewichts die Fortbewegungsgeschwindigkeit und bei der Wendegeschwindigkeit dessen Differenz als Eingang der Regelstrecke angenommen werden, obwohl es in beiden Fällen in der Realität ein Spannungswert ist, der auf die Motoren gegeben wird. Da diese Spannung in einem annähernd linearen Verhältnis mit der Drehgeschwindigkeit und damit Fortbewegungsgeschwindigkeit steht, sollte dies auf die qualitative Untersuchung wenig Einfluss haben.

3.1 Regelung des Gleichgewichts

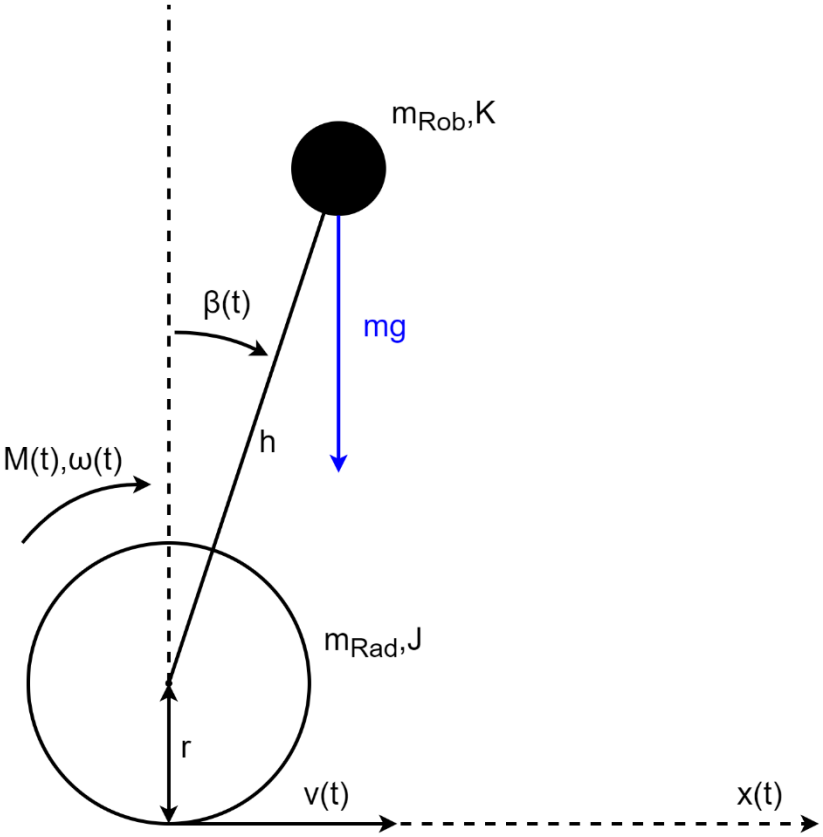

Abbildung 3-1: Mechanische Darstellung des Systems

$\beta(t)$: Pitch Winkel, Winkel des Roboters zur Vertikale

$\varepsilon(t)$: Radwinkel, Winkel des Rades zur Vertikale

$\omega(t)$: Drehgeschwindigkeit des Rades

$x(t)$: Zurückgelegte Distanz des Roboters

$v(t)$: Geschwindigkeit des Roboters auf der x-Achse

$M(t)$: Vom Motor produziertes Drehmoment

g: Erdanziehungskraft der Erde

r: Radius des Rades

h: Höhe des Schwerpunkts des Roboters vom Mittelpunkt der Räder

m_{Rob}: Masse des Roboters

m_{Rad}: Masse der Räder

K: Trägheitsmoment des Roboterkörpers

J: Trägheitsmoment der Räder

$$\dot{\varepsilon}(t) = \omega(t) \quad \text{Gl. 1}$$

$$\dot{x}(t) = v(t) = -r \cdot \dot{\varepsilon}(t) = -r \cdot \omega(t) \quad \text{Gl. 2}$$

Dabei wird davon ausgegangen, dass die Räder nicht rutschen und die Reibung zwischen den Rädern und dem Körper und der Luftwiderstand werden vernachlässigt.

<u>Energie des Systems:</u>

Kinetische Energie des Rades:

$$E_{kinRad} = \frac{1}{2} m_{Rad} v^2 + \frac{1}{2} J \omega^2 \quad \text{Gl. 3}$$

$$\Rightarrow E_{kinRad} = \frac{1}{2} m_{Rad} r^2 \dot{\varepsilon}^2 + \frac{1}{2} J \dot{\varepsilon}^2 \quad \text{Gl. 4}$$

Kinetische Energie des Zentrums der Masse:

$$E_{kinRob} = \frac{1}{2} m_{Rob} (v - h\dot{\beta} cos\beta)^2 + \frac{1}{2} m_{Rob} (L\dot{\beta} sin\beta)^2 + \frac{1}{2} K \dot{\beta}^2 \quad \text{Gl. 5}$$

$$\Rightarrow E_{kinRob} = \frac{1}{2} m_{Rob} (-r\dot{\varepsilon} - h\dot{\beta} cos\beta)^2 + \frac{1}{2} m_{Rob} (h\dot{\beta} sin\beta)^2 + \frac{1}{2} K \dot{\beta}^2 \quad \text{Gl. 6}$$

Potenzielle Energie des Zentrums der Masse:

$$E_{potRob} = m_{Rob} g h cos\beta \quad \text{Gl. 7}$$

Lagrange-Funktion:

$$\mathcal{L} = E_{kin} - E_{pot} = E_{kinRad} + E_{kinRob} - E_{potRob} \quad \text{Gl. 8}$$

Aus Gl. 4, Gl. 6 und Gl. 7:

$$\Rightarrow \mathcal{L} = \frac{1}{2}m_{Rad}r^2\dot{\varepsilon}^2 + \frac{1}{2}J\dot{\varepsilon}^2 + \frac{1}{2}m_{Rob}(-r\dot{\varepsilon} - h\dot{\beta}cos\beta)^2 + \frac{1}{2}m_{Rob}(h\dot{\beta}sin\beta)^2 + \frac{1}{2}K\dot{\beta}^2 - m_{Rob}ghcos\beta \quad \text{Gl. 9}$$

$$\Rightarrow \mathcal{L} = \frac{1}{2}(J + (m_{Rad} + m_{Rob})r^2)\dot{\varepsilon}^2 + \frac{1}{2}(K + m_{Rob}h^2)\dot{\beta}^2 + m_{Rob}rh(cos\beta)\dot{\beta}\dot{\varepsilon} - m_{Rob}ghcos\beta \quad \text{Gl. 10}$$

Euler-Lagrange Gleichung:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = Q_i \quad \text{Gl. 11}$$

mit generalisiertem Koordinatensystem $q = \begin{pmatrix}\beta \\ \varepsilon\end{pmatrix}$ und Q_i der generalisierten Kraft auf das System.

Lösung für jeden q_i und linearisiert um den Punkt $\beta = 0$, bzw. vertikale Position, also $cos\beta \approx 1$ angenommen:

$$(J + (m_{Rad} + m_{Rob})r^2)\ddot{\varepsilon} + (m_{Rob}rh)\ddot{\beta} = M \quad \text{Gl. 12}$$

$$(m_{Rob}rh)\ddot{\varepsilon} + (K + m_{Rob}h^2)\ddot{\beta} - (m_{Rob}gh)\beta = -M \quad \text{Gl. 13}$$

Aus Gl. 12 und Gl. 13:

$$(J + (m_{Rad} + m_{Rob})r^2)\ddot{\varepsilon} + (m_{Rob}rh)\ddot{\beta} = -(m_{Rob}rh)\ddot{\varepsilon} - (K + m_{Rob}h^2)\ddot{\beta} + (m_{Rob}gh)\beta \quad \text{Gl. 14}$$

$$\Rightarrow (J + (m_{Rad} + m_{Rob})r^2)\ddot{\varepsilon} + (m_{Rob}rh)\ddot{\beta} + (m_{Rob}rh)\ddot{\varepsilon} + (K + m_{Rob}h^2)\ddot{\beta} - (m_{Rob}gh)\beta = 0 \quad \text{Gl. 15}$$

$$\Rightarrow (J + m_{Rob}rh + (m_{Rad} + m_{Rob})r^2)\ddot{\varepsilon} + (K + m_{Rob}h^2 + m_{Rob}rh)\ddot{\beta} - (m_{Rob}gh)\beta = 0 \quad \text{Gl. 16}$$

Aus Gl. 1:

$$\Rightarrow (J + m_{Rob}rh + (m_{Rad} + m_{Rob})r^2)\dot{\omega} + (K + m_{Rob}h^2 + m_{Rob}rh)\ddot{\beta} - (m_{Rob}gh)\beta = 0 \quad \text{Gl. 17}$$

Aus Gl. 2:

$$\Rightarrow -(J/r + m_{Rob}h + (m_{Rad} + m_{Rob})r)\dot{v} + (K + m_{Rob}h^2 + m_{Rob}rh)\ddot{\beta} - (m_{Rob}gh)\beta = 0 \quad \text{Gl. 18}$$

$$\Rightarrow -K_1\dot{v} + K_2\ddot{\beta} - K_3\beta = 0 \quad \text{Gl. 19}$$

Wobei:

$K_1 = J/r + m_{Rob}h + (m_{Rad} + m_{Rob})r$

$$K_2 = K + m_{Rob}h^2 + m_{Rob}rh$$

$$K_3 = m_{Rob}gh$$

Laplace Transformation: (mit Anfangswerten = 0 angenommen)

$$-K_1pV(p) + K_2p^2B(p) - K_3B(p) = 0 \qquad \text{Gl. 20}$$

$$\Rightarrow -K_1pV(p) + (K_2p^2 - K_3)B(p) = 0 \qquad \text{Gl. 21}$$

$$\Rightarrow K_1pV(p) = (K_2p^2 - K_3)B(p) \qquad \text{Gl. 22}$$

$$\Rightarrow \frac{B(p)}{V(p)} = \frac{K_1p}{K_2p^2 - K_3} = \frac{X_a}{X_e} = F_s(p) \qquad \text{Gl. 23}$$

$$\Rightarrow F_s(p) = \frac{K_1p}{K_2p^2 - K_3} \qquad \text{Gl. 24}$$

In Abbildung 3-2 wird eine Wurzelortskurve und die Sprungantwort dieser Regelstrecke ohne Regler dargestellt. Dabei wurden für diese und die folgenden Matlab-Berechnungen die Konstanten als 1 angenommen um ein generelles Verhalten der Regelstrecke abzuleiten. Bei der Sprungantwort soll der Wert von 0 auf 1 geregelt werden.

Aus der Wurzelortskurve und der instabilen Sprungantwort wird erkennbar, dass dieses System nicht stabil ist und ein Regler benötigt wird. Dabei muss allerdings beachtet werden, dass ein positiver Pol nicht mit einem Regler kompensiert werden darf.

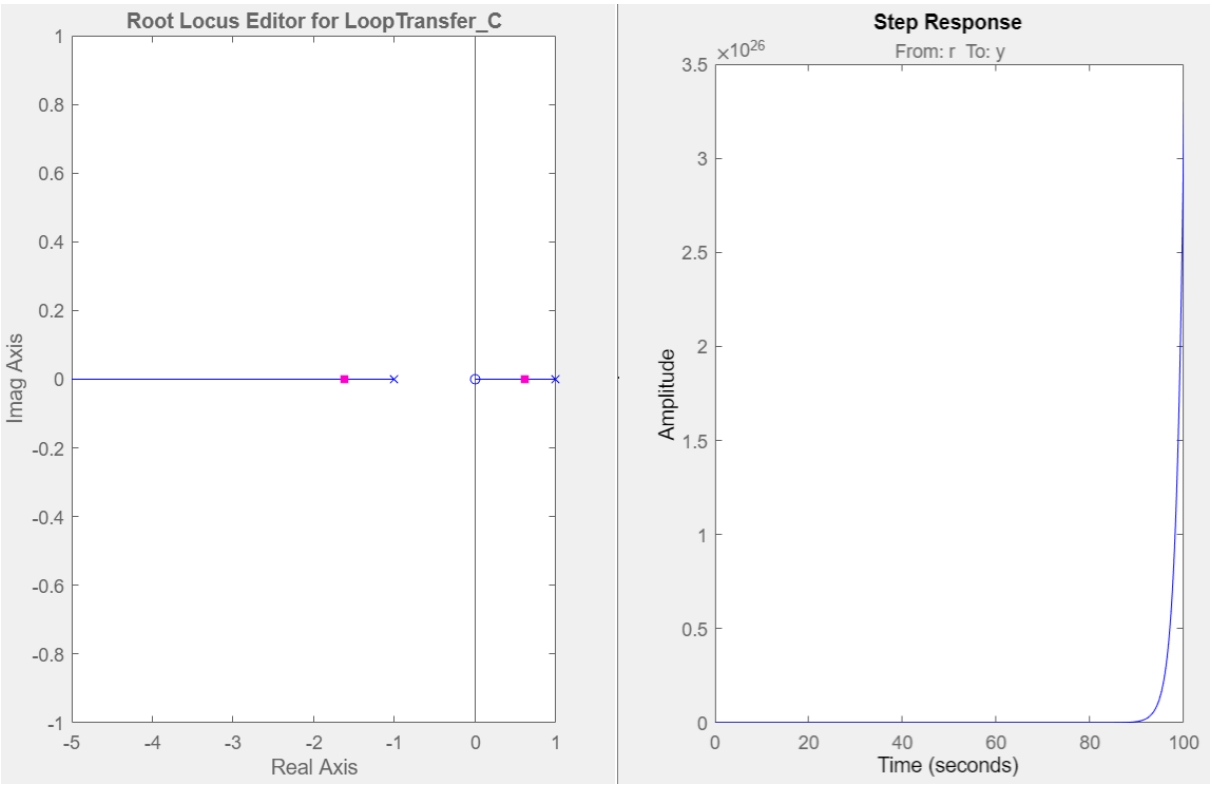

Abbildung 3-2: Wurzelortskurve und Sprungantwort des Gleichgewichts ohne Regler mit Matlab gezeichnet

Matlab-Befehle:

```
>> sys = tf([0 1 0],[1 0 -1]);
>> rltool(sys);
```

Es kann ein PI- oder PID-Regler genutzt werden.

Regelstrecke eines PID Regler (Parallelschaltung P-, I- und D-Regler):

$$F_R(p) = K_P + K_D \cdot p + \frac{K_I}{p} = K_P + \frac{p}{T_D} + \frac{1}{T_I p} \text{ mit } K_P, K_D, K_I \geq 0 \text{ und } T_I, T_D > 0 \qquad \text{Gl. 25}$$

PI-Regler:

Aus Gl. 24:

$$F_S(p) = \frac{K_1 p}{K_2 p^2 - K_3} = \frac{K_1 p}{(\sqrt{K_2} p + \sqrt{K_3})(\sqrt{K_2} p - \sqrt{K_3})} = \frac{\frac{K_1}{K_2} p}{(p + \sqrt{\frac{K_3}{K_2}})(p - \sqrt{\frac{K_3}{K_2}})} \qquad \text{Gl. 26}$$

Nullstellen: 1 → $p_z = 0$

Pole: 2 → $p_{k1} = +\sqrt{\frac{K_3}{K_2}}, p_{k2} = -\sqrt{\frac{K_3}{K_2}}$

$$F_R(p) = K_R \cdot \frac{(1 + pT_I)}{pT_I} = K \cdot \frac{(p + \sqrt{\frac{K_3}{K_2}})}{p} \qquad \text{Gl. 27}$$

$$F_0(p) = F_R(p) \cdot F_S(p) = K \cdot \frac{(p + \sqrt{\frac{K_3}{K_2}})}{p} \cdot \frac{\frac{K_1}{K_2} p}{(p + \sqrt{\frac{K_3}{K_2}})(p - \sqrt{\frac{K_3}{K_2}})} = K \cdot \frac{\frac{K_1}{K_2}}{(p - \sqrt{\frac{K_3}{K_2}})} \qquad \text{Gl. 28}$$

In Abbildung 3-3 erkennt man, dass das System nun stabil ist, allerdings muss die Verstärkung möglichst hoch gewählt werden, um die Regelabweichung zu minimieren.

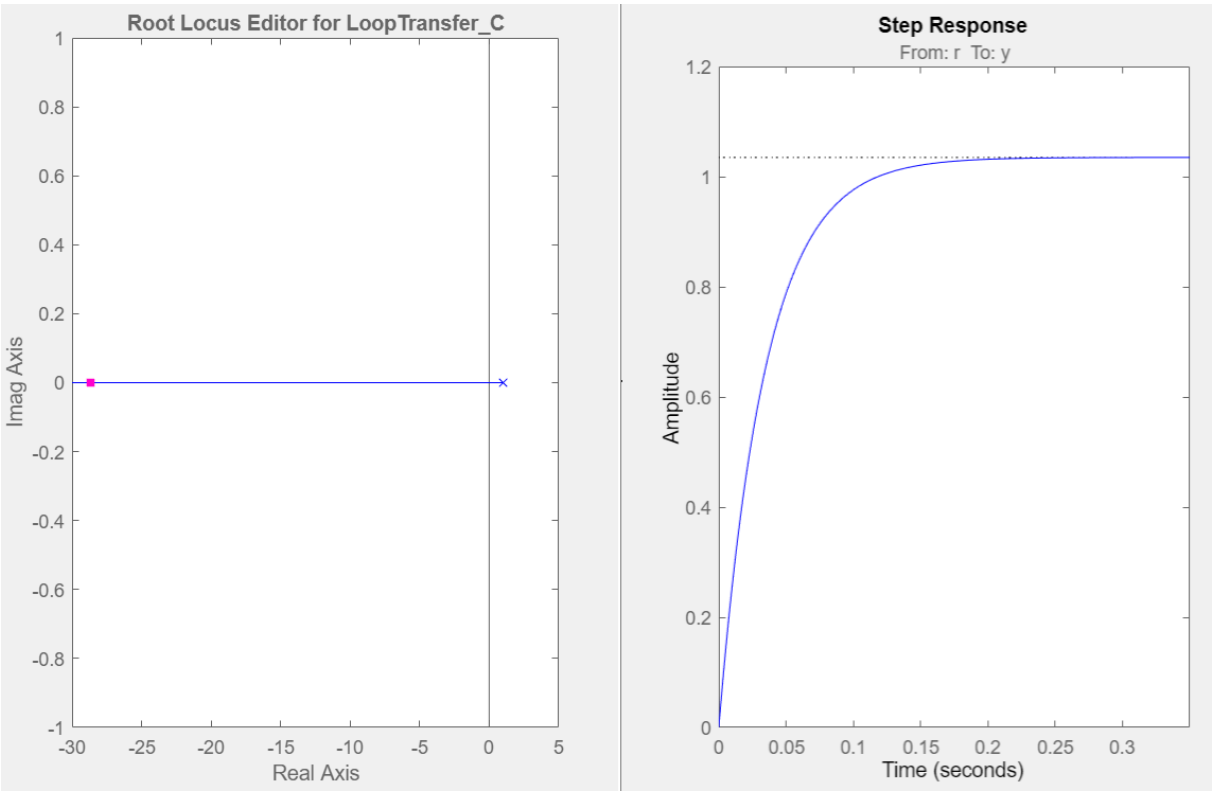

Abbildung 3-3: Wurzelortskurve und Sprungantwort des Gleichgewichts mit PI-Regler mit Matlab gezeichnet

Matlab-Befehle:

```
>> sys = tf([0 0 1],[0 1 -1]);
>> rltool(sys);
```

PID-Regler:

$$F_R(p) = K_R \cdot \frac{1+pT_I+p^2T_IT_D}{pT_I} = K \cdot \frac{(p+\sqrt{\frac{K_3}{K_2}})(p+T)}{p} \text{ mit T > 0} \qquad \text{Gl. 29}$$

$$F_0(p) = F_R(p) \cdot F_S(p) = K \cdot \frac{(p+\sqrt{\frac{K_3}{K_2}})(p+T)}{p} \cdot \frac{\frac{K_1}{K_2}p}{(p+\sqrt{\frac{K_3}{K_2}})(p-\sqrt{\frac{K_3}{K_2}})} = K \cdot \frac{(p+T)\frac{K_1}{K_2}}{(p-\sqrt{\frac{K_3}{K_2}})} \qquad \text{Gl. 30}$$

In Abbildung 3-4 ist erkennbar, dass die Systemantwort mit PID-Regler ähnlich wie mit PI-Regler ausfällt.

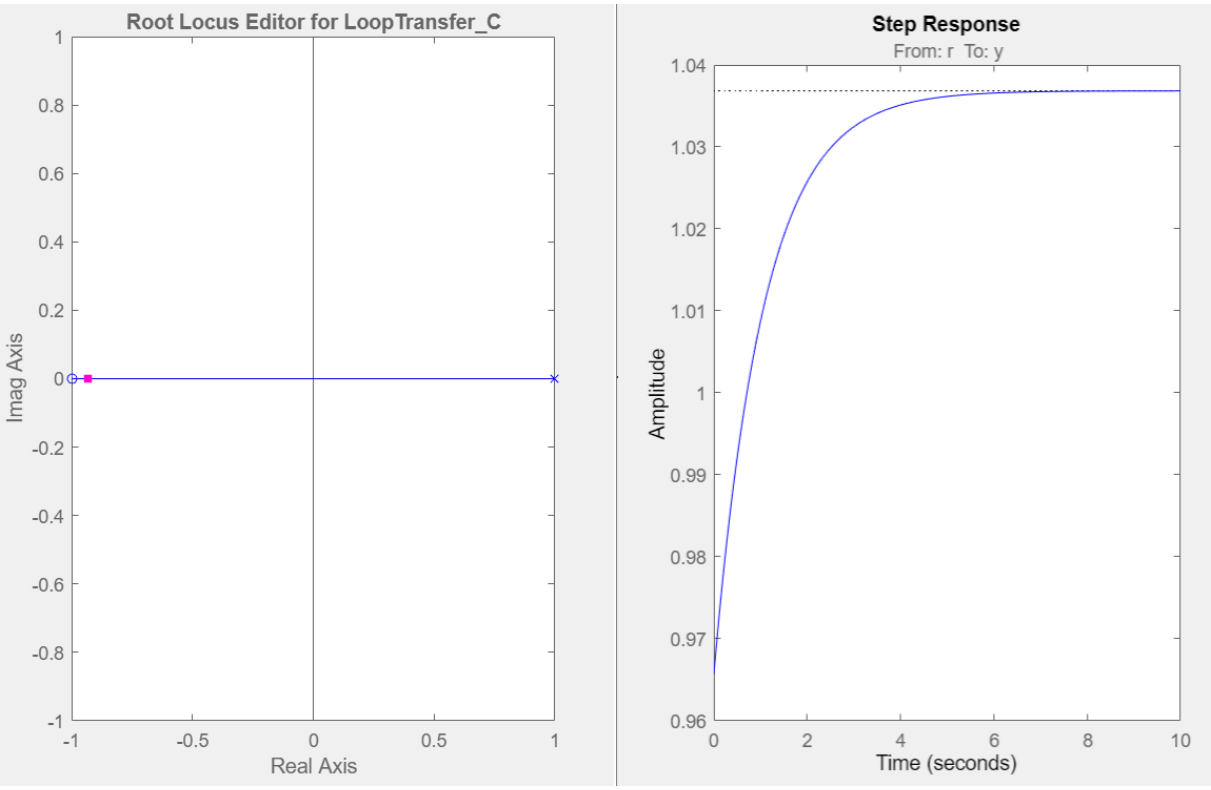

Abbildung 3-4: Wurzelortskurve und Sprungantwort des Gleichgewichts mit PID-Regler mit Matlab gezeichnet ***Matlab-Befehle:***

```
>> sys = tf([0 1 1],[0 1 -1]);
>> rltool(sys);
```

3.2 Regelung der Fortbewegungsgeschwindigkeit

Wenn man von einer vorherigen PI-Regelung für das Gleichgewicht ausgeht und die Rückkopplung berücksichtigt:

$$F_S(p) = \frac{V(p)}{B(p)} = \frac{F_{02}(p)}{1+F_{02}(p)} \qquad \text{Gl. 31}$$

Aus Gleichung Gl. 28:

$$F_S(p) = \frac{K \cdot \frac{\frac{K_1}{K_2}}{\left(p-\sqrt{\frac{K_3}{K_2}}\right)}}{1+K \cdot \frac{\frac{K_1}{K_2}}{\left(p-\sqrt{\frac{K_3}{K_2}}\right)}} = K \cdot \frac{\frac{K_1}{K_2}}{\left(p-\sqrt{\frac{K_3}{K_2}}\right)} \cdot \frac{\left(p-\sqrt{\frac{K_3}{K_2}}\right)}{K \cdot \frac{K_1}{K_2}+\left(p-\sqrt{\frac{K_3}{K_2}}\right)} = \frac{K \cdot \frac{K_1}{K_2}}{K \cdot \frac{K_1}{K_2}+\left(p-\sqrt{\frac{K_3}{K_2}}\right)} \qquad \text{Gl. 32}$$

$$\Rightarrow F_S(p) = \frac{K \cdot \frac{K_1}{K_2}}{p+K \cdot \frac{K_1}{K_2}-\sqrt{\frac{K_3}{K_2}}} \qquad \text{Gl. 33}$$

Angenommen $T = K \cdot \frac{K_1}{K_2} - \sqrt{\frac{K_3}{K_2}} > 0$:

$$\Rightarrow F_S(p) = K_{ges} \cdot \frac{1}{p+T}$$ Gl. 34

In Abbildung 3-5 wird eine Wurzelortskurve und die Sprungantwort dieser Regelstrecke ohne Regler dargestellt. Dieses System ist stabil, allerdings ist eine recht große Verstärkung vonnöten, um die Regelabweichung zu minimieren.

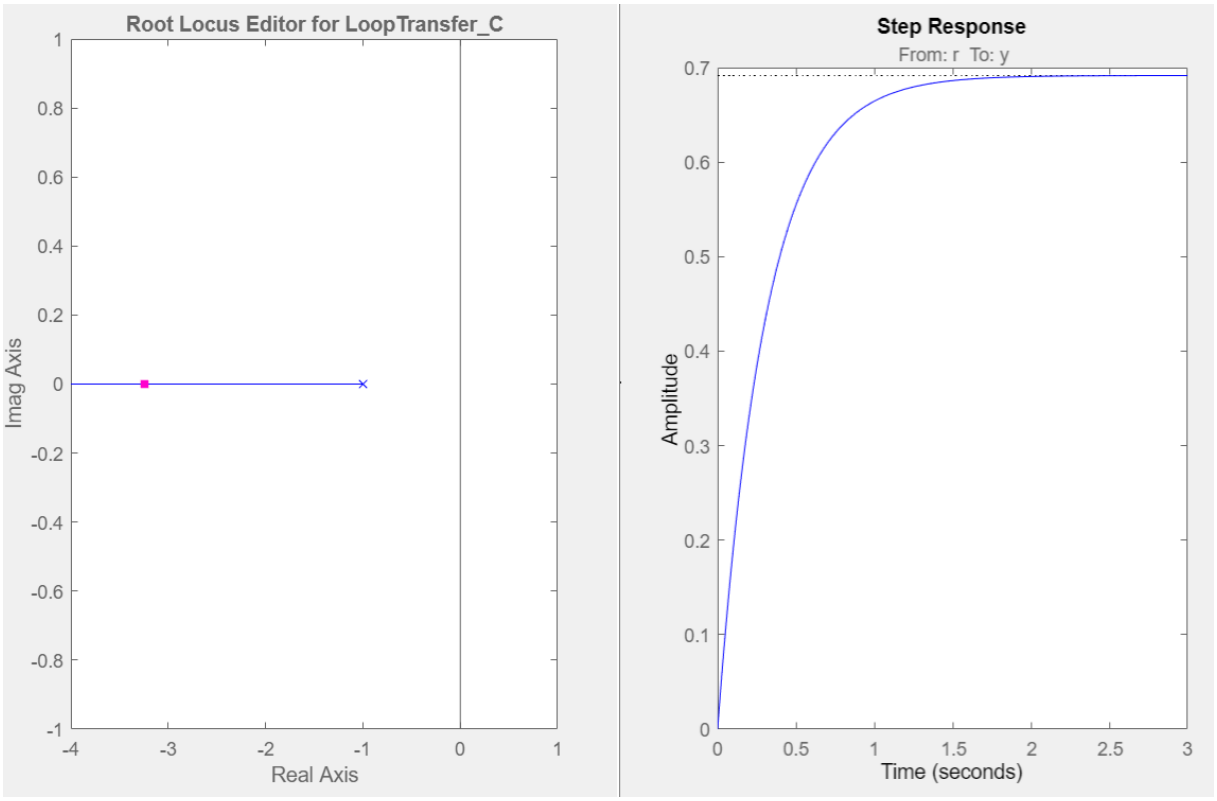

Abbildung 3-5: Wurzelortskurve und Sprungantwort der Fortbewegungsgeschwindigkeit ohne Regler mit Matlab gezeichnet

Matlab-Befehle:

```
>> sys = tf([0 0 1],[0 1 1]);
>> rltool(sys);
```

Es kann ein I-Regler eingesetzt werden, um die Regelabweichung zu eliminieren mithilfe eines Pols im Ursprung.

$$F_R(p) = \frac{1}{pT_I} = K_I \cdot \frac{1}{p}$$ Gl. 35

$$F_0(p) = F_R(p) \cdot F_S(p) = K_I \cdot \frac{1}{p} \cdot K_{ges} \cdot \frac{1}{p+T} = K_{ges2} \cdot \frac{1}{p(p+T)}$$ Gl. 36

In Abbildung 3-6 ist ersichtlich, dass die Regelabweichung erfolgreich eliminiert wurde.

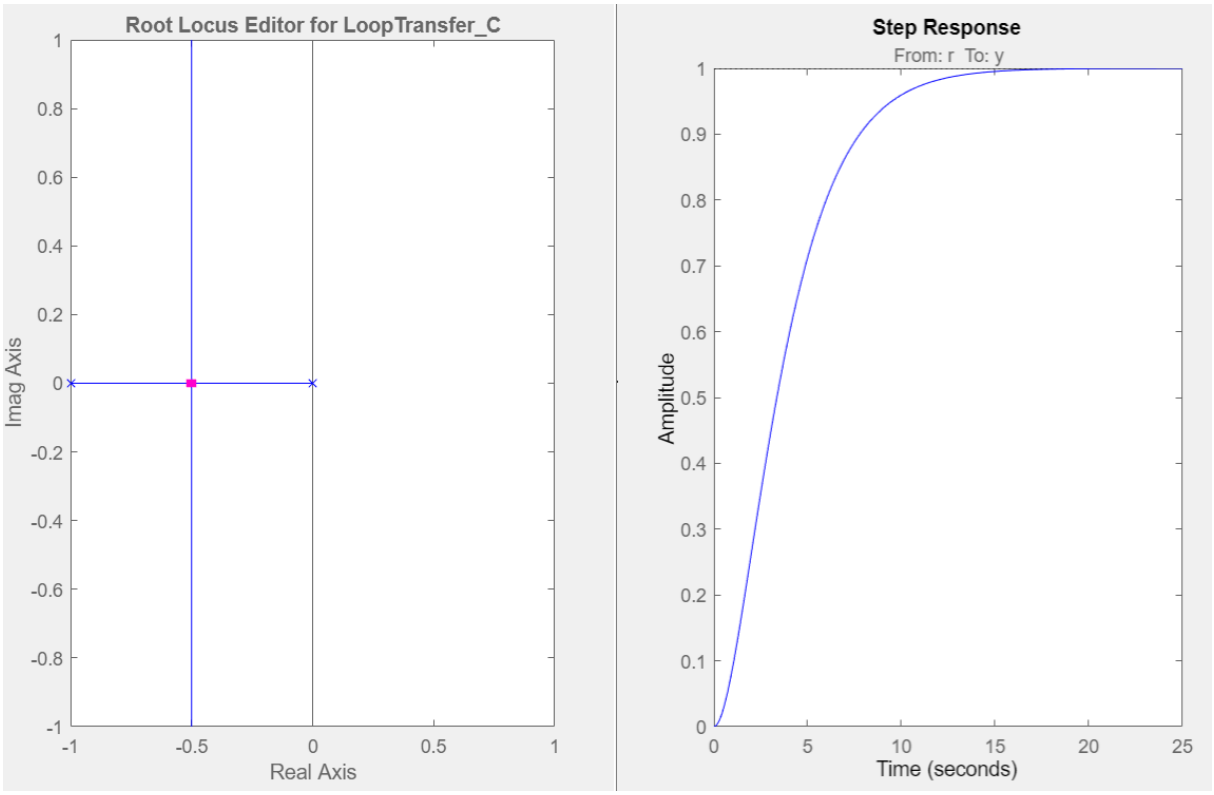

Abbildung 3-6: Wurzelortskurve und Sprungantwort der Fortbewegungsgeschwindigkeit mit I-Regler mit Matlab gezeichnet

Matlab-Befehle:

```
>> sys = tf([0 0 1],[1 1 0]);

>> rltool(sys);
```

3.3 Regelung der Wendegeschwindigkeit

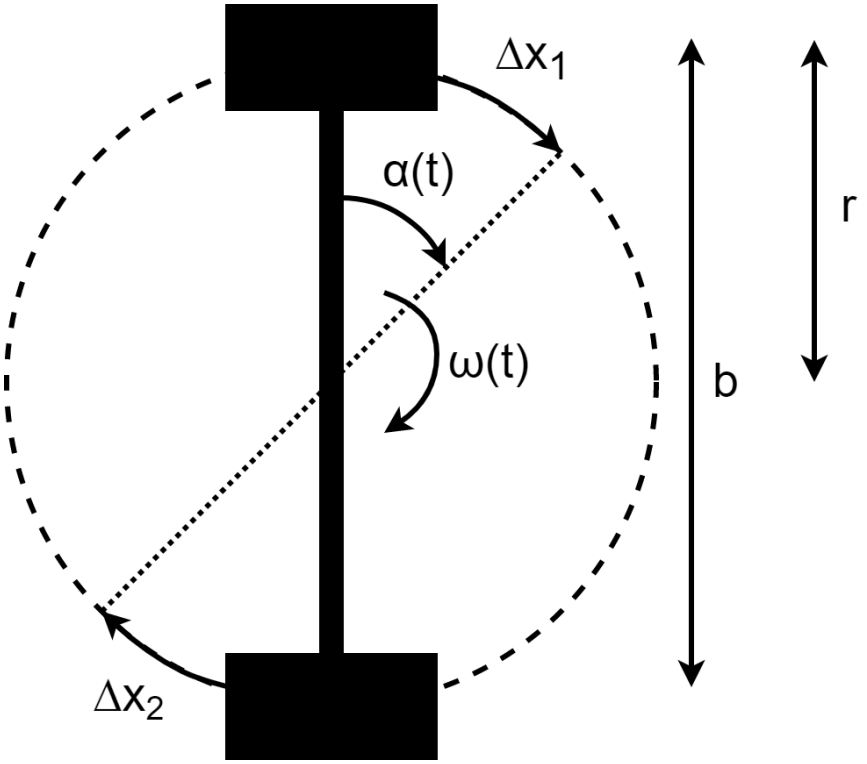

Abbildung 3-7: Darstellung der Rotation des Systems

$\alpha(t)$: Yaw Winkel

$\omega(t)$: Wendegeschwindigkeit

$x(t)$: Zurückgelegte Distanz des Roboters aus dem Durchschnitt beider Räder

$\Delta x_1(t), \Delta x_2(t)$: Abweichung der Distanz von Rad 1 und 2 in positive Wenderichtung zu $x(t)$

$\Delta x(t)$: Gesamtabweichung der Position

$v(t)$: Geschwindigkeit des Roboters aus dem Durchschnitt beider Räder

$\Delta v_1(t), \Delta v_2(t)$: Abweichung der Geschwindigkeit von Rad 1 und 2 zu $v(t)$

$\Delta v(t)$: Gesamtabweichung der Geschwindigkeit

b: Breite des Roboters

r: Radius des Wendekreises

Es wird davon ausgegangen, dass die Räder nicht rutschen.

$$\dot{\alpha}(t) = \omega(t) \quad \text{Gl. 37}$$

$$\dot{x}(t) = v(t) \quad \text{Gl. 38}$$

$$\Rightarrow \Delta\dot{x}(t) = \Delta v(t) \quad \text{Gl. 39}$$

$$\Delta x(t) = \frac{1}{2}(\Delta x_1(t) + \Delta x_2(t)) \quad \text{Gl. 40}$$

$$\Rightarrow \Delta \dot{x}(t) = \frac{1}{2}\left(\Delta \dot{x}_1(t) + \Delta \dot{x}_2(t)\right) \quad \text{Gl. 41}$$

$$\Rightarrow \Delta v(t) = \frac{1}{2}(\Delta v_1(t) + \Delta v_2(t)) \quad \text{Gl. 42}$$

Umfang des Kreises, wenn der Roboter auf der Stelle dreht:

$$l = 2\pi r = \pi b = 2\pi \cdot \frac{b}{2} \quad \text{Gl. 43}$$

$$\Delta x(t) = \alpha(t) \cdot \frac{b}{2} \quad \text{Gl. 44}$$

Ableitung:

$$\Delta \dot{x}(t) = \frac{b}{2} \cdot \dot{\alpha}(t) \quad \text{Gl. 45}$$

$$\Rightarrow \Delta v(t) = \frac{b}{2} \cdot \omega(t) \quad \text{Gl. 46}$$

Laplace Transformation: (mit Anfangswerten = 0 angenommen)

$$\Delta V(p) = \frac{b}{2} \cdot W(p) \quad \text{Gl. 47}$$

$$\Rightarrow \Delta V(p) = \frac{b}{2} \cdot W(p) \quad \text{Gl. 48}$$

$$\Rightarrow \frac{W(p)}{\Delta V(p)} = \frac{2}{b} = \frac{X_a}{X_e} = F_s(p) \quad \text{Gl. 49}$$

$$\Rightarrow F_s(p) = \frac{2}{b} = Kte \quad \text{Gl. 50}$$

Ohne Regler ist das System statisch und der Ausgangswert bleibt konstant.

Es bietet sich zum Regeln dieses Regelkreises ein I- oder PI-Regler an.

<u>I-Regler:</u>

$$F_R(p) = \frac{1}{pT_I} = K_I \cdot \frac{1}{p} \quad \text{Gl. 51}$$

$$F_0(p) = F_R(p) \cdot F_S(p) = K_I \cdot \frac{1}{p} \cdot \frac{2}{b} = K \cdot \frac{1}{p} \quad \text{Gl. 52}$$

In Abbildung 3-8 werden die Wurzelortskurve und die Sprungantwort des Systems mit I-Regler dargestellt. Mit einem I-Regler befinden sich alle Punkte der Wurzelortskurve im real negativen Teil und es bleibt keine permanente Regelabweichung.

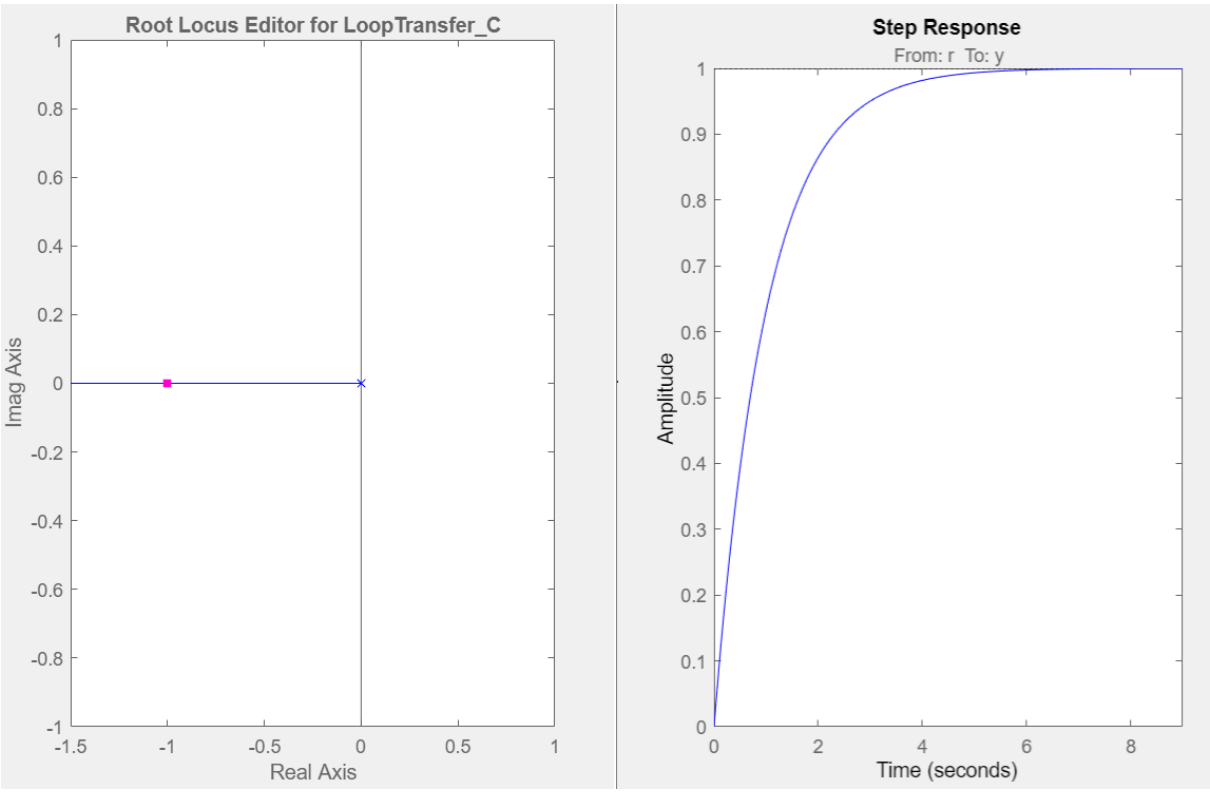

Abbildung 3-8: Wurzelortskurve und Sprungantwort der Wendegeschwindigkeit mit I-Regler mit Matlab gezeichnet

Matlab-Befehle:

```
>> sys = tf([0 0 1],[0 1 0]);
>> rltool(sys);
```

PI-Regler:

$$F_R(p) = K_R \cdot \frac{1+pT_I}{pT_I} \qquad \text{Gl. 53}$$

$$F_0(p) = F_R(p) \cdot F_S(p) = K_R \cdot \frac{1+pT_I}{pT_I} \cdot \frac{2}{b} = K \cdot \frac{\frac{1}{T_I}+p}{p} \qquad \text{Gl. 54}$$

In Abbildung 3-9 werden die Wurzelortskurve und Sprungantwort im Falle einer PI-Regelung dargestellt. Auch der PI-Regler erfüllt alle Voraussetzungen für eine erfolgreiche Regelung.

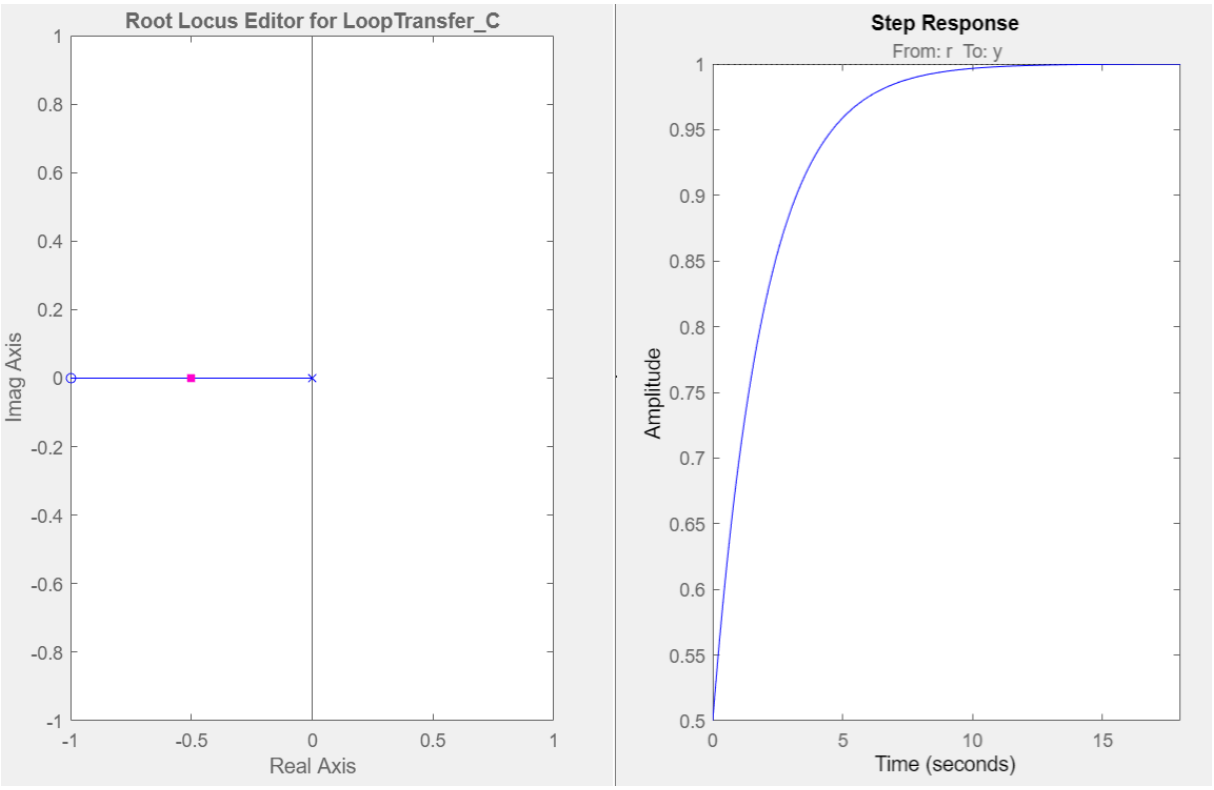

Abbildung 3-9: Wurzelortskurve und Sprungantwort der Wendegeschwindigkeit mit PI-Regler mit Matlab gezeichnet

Matlab-Befehle:

```
>> sys = tf([0 1 1],[0 1 0] );
>> rltool(sys);
```

3.4 Regelung der abgefahrenen Distanz

Aus Gl. 2: $\dot{x}(t) = v(t)$

Laplace Transformation: (mit Anfangswerten = 0 angenommen)

$$pX(t) = V(t) \qquad \text{Gl. 55}$$

$$\Rightarrow \frac{X(p)}{V(p)} = \frac{1}{p} = \frac{X_a}{X_e} = F_s(p) \qquad \text{Gl. 56}$$

$$\Rightarrow F_s(p) = \frac{1}{p} \qquad \text{Gl. 57}$$

In Abbildung 3-10 ist erkennbar, dass diese Regelstrecke stabil ist. Sie kann aber durch den Zusatz eines P-, PI- oder PD-Regler optimiert werden.

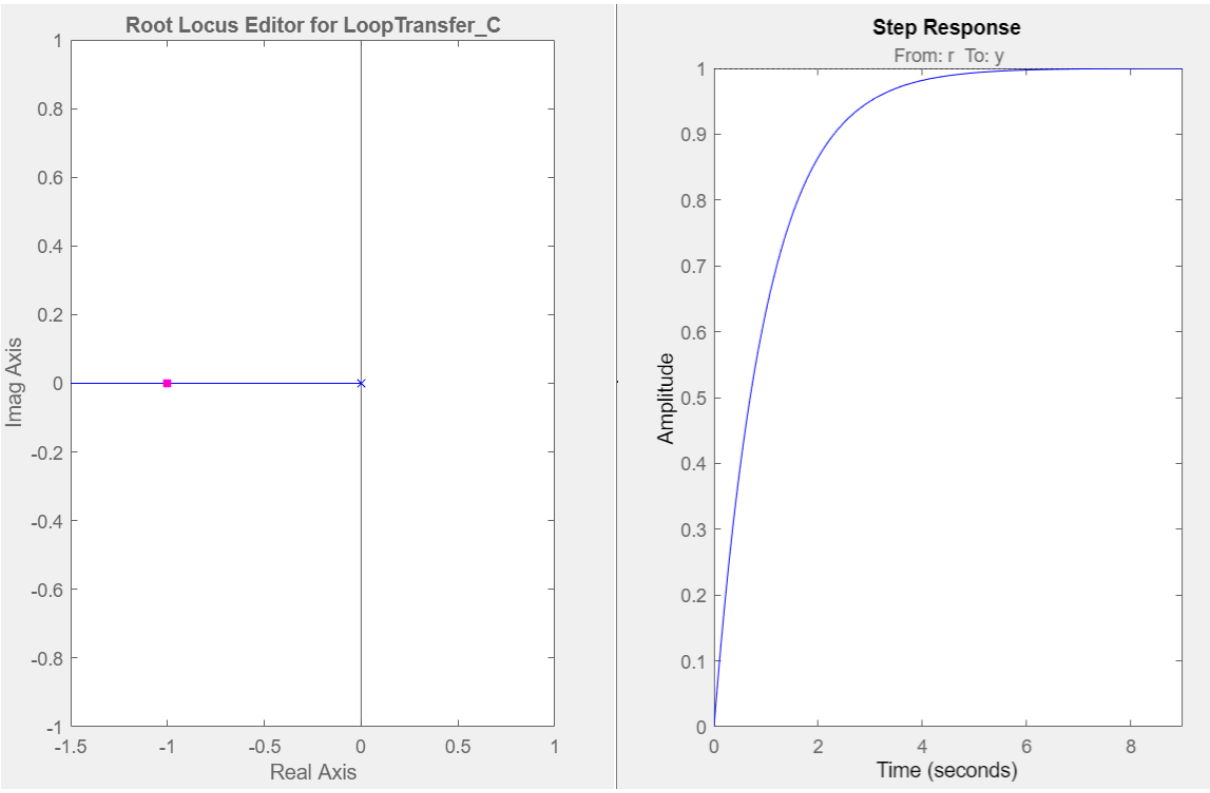

Abbildung 3-10: Wurzelortskurve und Sprungantwort der abgefahrenen Distanz ohne Regler mit Matlab gezeichnet

3.5 Regelung der Richtung

Aus Gl. 37: $\dot{\alpha}(t) = \omega(t)$

Laplace Transformation: (mit Anfangswerten = 0 angenommen)

$$pA(p) = W(p) \qquad \text{Gl. 58}$$

$$\Rightarrow \frac{A(p)}{W(p)} = \frac{1}{p} = \frac{X_a}{X_e} = F_s(p) \qquad \text{Gl. 59}$$

$$\Rightarrow F_s(p) = \frac{1}{p} \qquad \text{Gl. 60}$$

Diese Regelstrecke weist das Gleiche Verhalten wie die Regelung der abgefahrenen Distanz auf (Siehe Abbildung 3-10). Das bedeutet, dass diese Regelstrecke ebenfalls stabil ist und durch P-, PI- oder PD-Regler optimiert werden kann.

4 Software

Nachdem wir uns mit der Hardware befasst haben, wenden wir uns nun der Software zu. Zuerst werden allgemeinere Konzepte erläutert, dann die Kommunikation zwischen der Fernbedienung und dem Roboter und anschließend die Programmierung des Roboters und der Fernbedienung jeweils näher analysiert.

4.1 Allgemeines

In diesem Abschnitt werden die Arduino IDE, die zum Programmieren der benutzten Mikrokontroller eingesetzt wird, und die Protokolle, die zur Kommunikation zwischen den Arduinos und verschiedenen Modulen zum Einsatz kommen, erklärt.

4.1.1 Arduino IDE

Als Software zum Programmieren der Arduinos kommt die Open Source Arduino IDE von Arduino selbst zum Einsatz. Diese Software ist mit jedem Arduino Bord kompatibel und der Source Code der Anwendung ist auf GitHub verfügbar. Die Textdateien, die man auf den Arduino hochlädt, haben die Dateiendung „.ino“ und werden in einer C und C++ ähnlichen Programmiersprache geschrieben. Der Arduino wird über USB an den PC mit der Arduino IDE verbunden. Dies ermöglicht das Hochladen des Codes auf den Arduino und die Kommunikation zwischen beiden Geräten. Die Funktion Serial Monitor erlaubt es Kommunikationen vom Arduino zu empfangen und Eingaben an den Arduino zu senden. Mit Serial Plotter, kann man den Zustandsverlauf einer Variabel im laufenden Programmcode auf dem Arduino in einem Diagramm in der IDE anzeigen lassen. Es besteht auch die Möglichkeit Bibliotheken mit vorgefertigten Funktionen einzubinden, allerdings werden alle eingebundenen Bibliotheken in ihrer Gesamtheit auf den Arduino kopiert.

Stand Juli 2022 steht die Arduino IDE Version 2 mit neuerem Interface und neuen Funktionalitäten experimentell zur Verfügung. Für dieses Projekt wurde aber noch die ältere Version 1 benutzt. (23), (24)

4.1.2 I2C

Für die Kommunikation des Arduino Megas mit dem BNO055 Sensor und des Arduino Unos mit der SSD1305 OLED Anzeige, kommt das I2C Protokoll zum Einsatz. Der I2C Bus, auch „Inter-Integrated Circuit“, besitzt eine Master-Slave Struktur und verfügt über eine geringe Reichweite und Bandbreite. Dafür werden aber nur 2 Leitungen benötigt, um den Master mit den Slaves zu verbinden. Dies sind die serielle Datenleitung SDA und serielle Taktleitung SCL. Der Master kommuniziert immer nur mit einem Gerät gleichzeitig und jedes gesendete Datenbyte wird vom Empfänger mit ACK oder NAK beantwortet, um zu kommunizieren, ob die Information angekommen ist. Jeder Slave besitzt eine Adresse, die im Hexadezimalen

System ausgedrückt wird, wie zum Beispiel 0x3D für die SSD1306 Anzeige. Die Kommunikation läuft im Halbduplex Modus ab, es kann also in beide Richtungen gesendet werden, aber nicht in beide Richtungen gleichzeitig. Das Protokoll erlaubt auch die Nutzung von mehr als einem Master. Die Arduinos besitzen jeweils vordefinierte Pins für den I2C Anschluss. In Abbildung 4-1 wird eine I2C Verbindung zwischen 2 Mastern und 2 Slaves dargestellt, wobei Master durch „Controller“ und Slave durch „Peripheral“ ersetzt wurde. (25), (26)

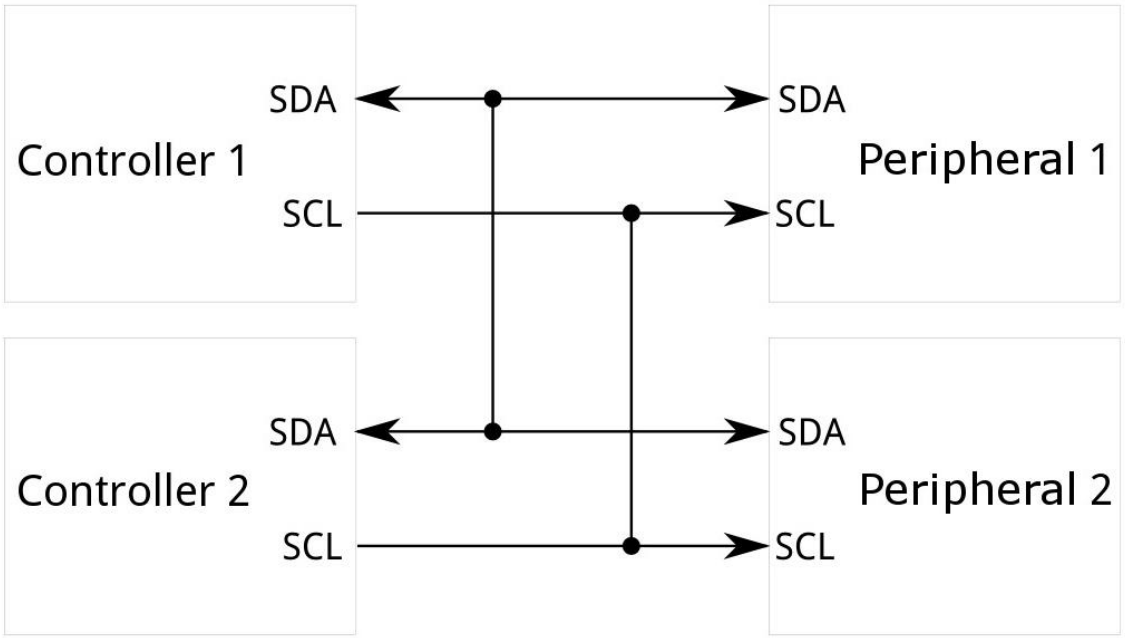

Abbildung 4-1: Schema einer I2C Verbindung (26)

4.1.3 SPI

Die Arduinos kommunizieren mit ihrem jeweiligen nRF24L01+ Modul über das SPI Protokoll, auch „Serial Peripheral Interface“. Der SPI Bus ist ebenfalls ein Master-Slave Protokoll, hat aber eine höhere Datenübertragungsrate als I2C. Dafür sind jedoch mindestens 4 Leitungen für die Verbindung nötig. Als erstes die Taktleitung SCK, dann die Datenleitungen MOSI („Master-Out-Slave-In“) und MISO („Master-In-Slave-Out“) für beide Richtungen oder COPI und CIPO, wenn Master als „Controller“ und Slave als „Peripheral“ bezeichnet wird, und dann die Slave Auswahl Leitung SS („Slave-Select“). Letztere wird separat für jeden angeschlossenen Slave benötigt, wenn jedes Gerät individuell angesteuert werden soll. Bei diesem Protokoll gibt es keinen Standard zum Überprüfen des Empfangs der Daten. Hierbei handelt es sich um eine Vollduplex Verbindung, weil die Daten in beide Richtungen gleichzeitig gesendet werden können. Im Gegensatz zu I2C ist hier nur ein Master erlaubt. Auch hier haben die Arduinos jeweils vordefinierte Pins für die SPI Kommunikation. In Abbildung 4-2 wird eine SPI Verbindung zwischen einem Master mit 2 Slaves dargestellt. (25), (26)

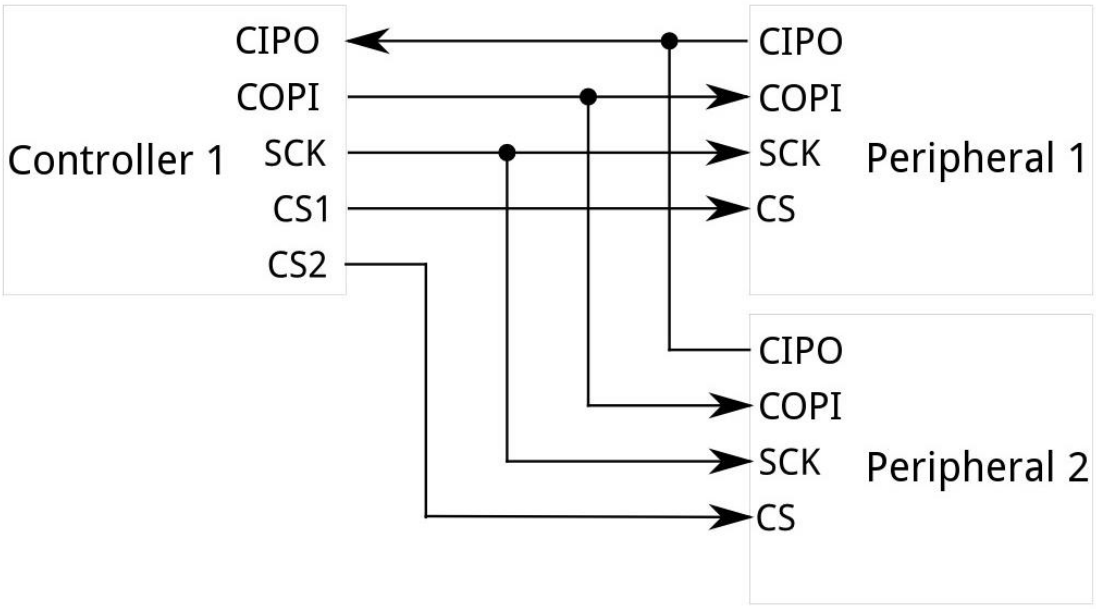

Abbildung 4-2: Schema einer SPI Verbindung (26)

4.2 Datenübertragung zwischen Roboter und Kontroller

Die Kommunikation zwischen der Fernbedienung und dem Roboter findet mithilfe der kabellosen Verbindungsmodule nRF24L01+ an beiden Geräten statt. Dafür werden zuerst 3 verschiedene Bibliotheken genutzt. Die erste „RF24.h“ muss zuerst heruntergeladen werden. Die beiden anderen sind „nRF24L01.h“ und „SPI.h.“ Dabei ist die Letztere nötig für die Kommunikation über das SPI Protokoll zwischen den Modulen und Arduinos. Anschließend werden zuerst CSN und CE Pins festgelegt. Dann werden die Kanäle für beide Senderichtungen festgelegt als Kanäle 100 und 101, also 2,5GHz und 2,501GHz. Zusätzlich werden auch noch einige Einstellungen vorgenommen: Die Datenrate wird vom Standartwert 1 Mbps auf den niedrigsten Wert von 250kbps gesetzt, denn es besteht keinen Bedarf an hoher Bandbreite und die Reichweite ist so verbessert. Außerdem wird die Sendestärke auf Maximum gesetzt, um die Reichweite zu maximieren. Als Datenpaket werden zwei Datenstrukturen definiert: „data1“ wird von der Fernbedienung gesendet und „data2“ vom Roboter. Dabei muss beachtet werden, dass jedes Paket nicht mehr als 32 Bytes beinhaltet, denn dies ist das größte Paket, das vom Modul gesendet werden kann. Ein Float, Integer oder String besteht zum Beispiel aus 4 Bytes, so dass nur 8 dieser Variablen in einem Paket übertragen werden können.

Beim Starten der Fernbedienung wird das Modul zuerst so eingestellt, dass es nur nach Signalen des Roboters sucht. Erst wenn die Fernbedienung das erste Mal Daten vom Roboter empfangen hat, wechselt der Roboter zwischen Senden und Empfangen von Daten. Wenn das Modul keine Daten empfängt, wird dieser Zeitpunkt vermerkt und wenn ab dann mehr als 2 Sekunden vergehen, ohne dass ein Paket erfolgreich empfangen wird, dann wird der Status auf den Wartezustand „Not connected“ geändert.

4.3 Roboter Software

Nun wenden wir uns der Roboter Programmierung zu. Zuerst wird der ursprüngliche Code kurzfassend erklärt, anschließend die Überlegungen hinter dem neuen Code und dann der Programmcode selbst mit Erläuterungen gezeigt.

4.3.1 Ursprünglicher Programmcode

Im ursprünglichen Programm war der Roboter so programmiert, dass er nur balancieren konnte, ohne sich fortzubewegen. Dafür wurde eine C++ Bibliothek namens „SelfRobo.h“ erstellt, auf die das Hauptprogramm zurückgreift. In dieser Bibliothek wurde eine Klasse „SelfRobo“ definiert, wo man die Werte für den Regler zum Ausgleichen und die Pins für die Ansteuerung des Motortreibers angeben kann. In der Klasse ist dann der PID-Regler zum Ausgleichen definiert, das Auslesen des Positionssensors BNO055 und die Ausgabe des Regler Ausgangs auf den Motortreiber, um die Motoren anzusteuern. Die Funktionen der Klasse wurden dann einfach in der Hauptschleife ausgeführt.

Von dieser Klasse werden nur noch die Funktionen zur Ansteuerung der Motoren und Auslesen des Positionssensors BNO055 genutzt. Die neue Regler Struktur befindet sich nun im Hauptprogramm.

Eine leicht modifizierte Version der „SelfRobo.h“ Bibliothek im Vergleich zur Ursprünglichen kann im Anhang gefunden werden.

4.3.2 Regler Struktur

Zur Regelung kommen PID-Regler zum Einsatz. Dabei handelt es sich um eine Kombination aus P-, I- und D-Regler. Dabei berücksichtigt der P-Anteil die Abweichung des Istwerts vom Sollwert, der I-Anteil das Integral dieser Abweichung und der D-Anteil die Ableitung der Abweichung.

Der ursprüngliche Roboter basierte auf nur einem einzigen PID, um den Roboter im Gleichgewicht zu halten. Dieser Regler wurde auch noch in dieser Version im Prinzip beibehalten. Allerdings wurden für die zusätzlichen Funktionen des Roboters noch 4 weitere PID Regler hinzugefügt.

<u>balancePID:</u>

Der Regler balancePID für das Gleichgewicht nimmt den Pitch Winkel in Grad, der mithilfe des Positionssensors BNO055 bestimmt wird als Istwert und als Sollwert den Ziel Pitch Winkel in Grad, auf den der Roboter stabilisiert werden soll. Der Ausgangswert ist ein 8 Bits Integer zwischen 0 und 255 und umgedrehte Richtung bei negativem Wert, die anschließend über den L298N Motortreiber in ein PWM Signal zwischen 0 und 12V für die Motoren übersetzt wird. Der Zielwinkel hat -1° als Anfangswert, weil dort der Roboter bei nicht geneigtem Roboter ungefähr im Gleichgewicht ist.

drivePID:

Anschließend wird dieser Wert von einem weiteren Regler drivePID vorgegeben, der die Fortbewegungsgeschwindigkeit des Roboters regelt, denn wenn der Zielwinkel nicht im Gleichgewichtspunkt des Robotes ist, balanciert dieser sich nicht mehr auf der gleichen Stelle, sondern beginnt sich in die geneigte Richtung fortzubewegen, da die Räder den Roboter nachschieben müssen, um der Beschleunigung durch die Schwerkraft entgegenzuwirken. Der Istwert ist die Drehgeschwindigkeit der Motoren aus der Ableitung der Positionssignale in Encodersignalen pro Millisekunde, auch ENC/ms. Der Sollwert ist dann die gewünschte Geschwindigkeit und das Ausgangssignal der Ziel Pitch Winkel oder auch der Sollwert des vorherigen balancePIDs. Auch wenn der Roboter sich nicht fortbewegen soll, ist dieser Regler nötig, denn der Gleichgewichtspunkt des Roboters kann sich verschieben, wenn die Bodenfläche unter den Rädern leicht geneigt ist, was in der Praxis generell der Fall ist.

Es wurde später festgestellt, dass ein abrupter hin und her Wechsel der Bewegungsrichtung des Roboters, zum Beispiel indem man den Joystick im Modus 1 abwechselnd nach oben und unten bewegt, destabilisieren kann. Deswegen wurde auf den Eingang dieses Reglers eine Dämpfung bzw. einen numerischen Filter implementiert. Dies resultiert darin, dass die dem Regler zugeführten Werte zu 99% aus dem vorherigen Wert und 1% dem neuen Wert bestehen. Dies erleichtert zusätzlich die Reglung der Geschwindigkeit im Allgemeinen, da der Regler dadurch eingestellt werden kann, ohne den Roboter bei der Beschleunigung aus dem Gleichgewicht zu bringen, weil der Roboter recht langsam reagiert.

turnPID:

Allerdings muss auch die Wendegeschwindigkeit über den Regler turnPID eingestellt werden. Dieser nimmt als Istwert die Wendegeschwindigkeit in Grad pro Sekunde, die durch die Ableitung des Yaw Winkels aus dem Positionssensor BNO055 ermittelt wird. Der Sollwert ist dann die Zielwendegeschwindigkeit und der Ausgangswert ist ein Wert, der vom Spannungswert eines Motors abgezogen und zum anderen addiert wird.

Wenn der Roboter im Modus 1 mit dem Joystick gesteuert wird, dann gibt dieser den Sollwert für die Fahr- und Wendegeschwindigkeit vor, wenn allerdings ein vorprogrammierter Pfad im Modus 2 vorgegeben wird, dann müssen zwei weitere Regler her, um diese beiden Geschwindigkeiten so einzustellen, dass die Zielposition erreicht wird.

distancePID:

Der erste dieser beiden Regler ist der distancePID, der die abgefahrene Distanz regelt. Dieser bekommt als Istwert die Summe der gezählten Encodersignale beider Motoren durch 2 geteilt. Wobei eine Drehrichtung die Zähler erhöht und die andere sie verringert. Der Sollwert ist dann die gewünschte Distanz in cm, die der Roboter abfahren soll. Das Ausganssignal ist dann eine gewünschte Geschwindigkeit in cm/s. Diese Geschwindigkeit wird dann in ENC/ms umgewandelt, indem mit 988 (Anzahl Encodersignale pro Umdrehung) multipliziert wird und anschließend durch 1000 (Umwandlung Sekunde in Millisekunde) und 20,42 (Radumfang in

cm) geteilt wird, also: $v\left[\frac{ENC}{ms}\right] = \frac{988}{1000 \cdot 20{,}42} \cdot v\left[\frac{cm}{s}\right]$. Dieser Wert wird dann als Sollwert für den drivePID vorgegeben.

directionPID:

Schlussendlich ist der Regler directionPID zuständig dafür, dass der Roboter in eine bestimmte Richtung zeigt, also ein bestimmter Yaw Winkel eingestellt wird. Als Istwert wird der Yaw Winkel in Grad aus dem BNO055 entnommen. Der Stellwert ist der Ziel Yaw Winkel in Grad und der Ausgang ist Wendegeschwindigkeit in Grad pro Sekunde, also der Sollwert vom turnPID.

Zusätzlich befindet sich am Ausgang am jedem der 5 PIDs eine Sättigung, die den Ausgangswert im positiven und negativen Bereich begrenzt. Es handelt sich also um eine Regler Struktur bestehend aus kaskadierenden PID-Reglern. Erstens sind die 3 Regler balancePID, drivePID und distancePID und dann separat die 2 Regler turnPID und directionPID in Reihe geschaltet.

In Abbildung 4-3 ist diese Regel Struktur schematisch dargestellt.

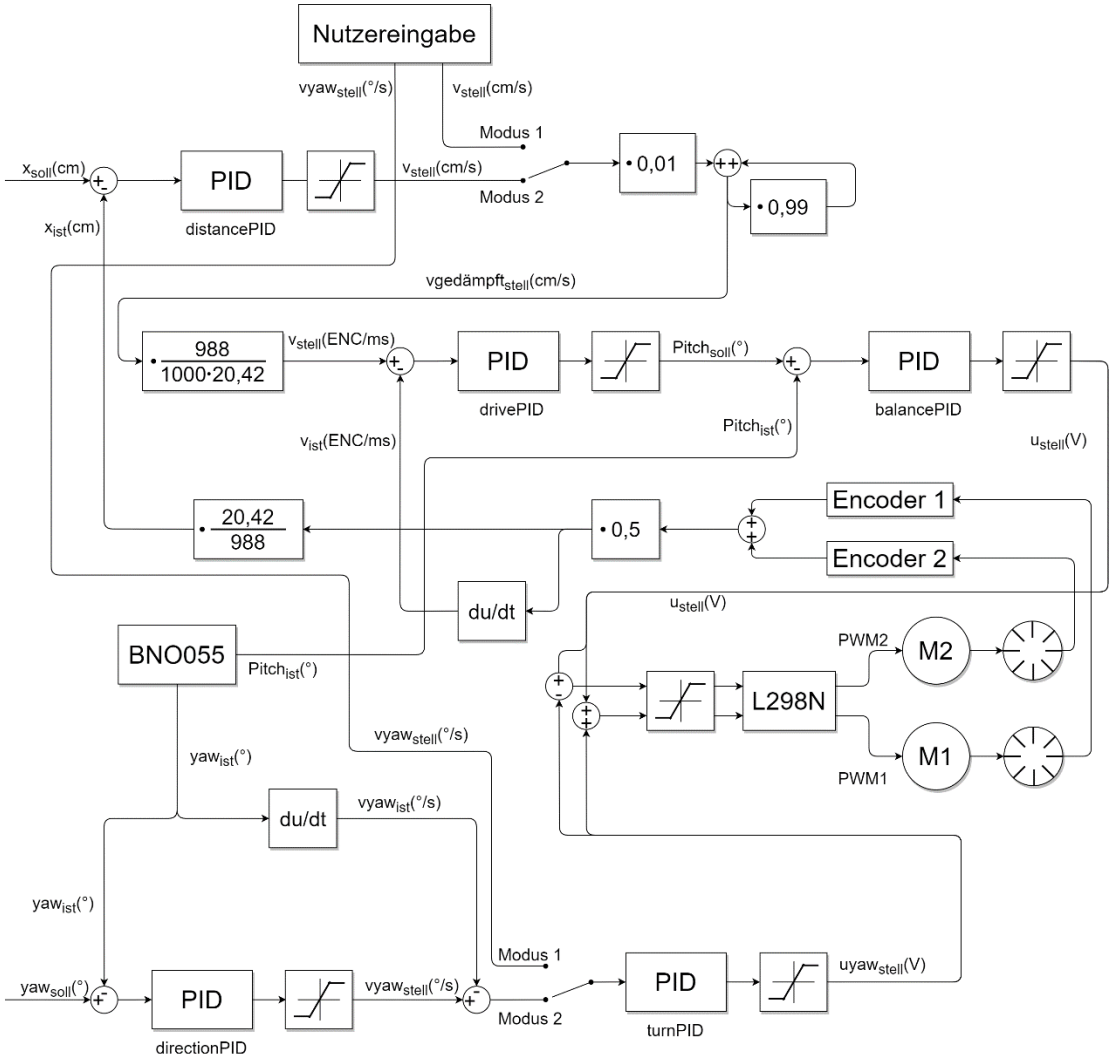

Abbildung 4-3: Schematische Darstellung der Regler Struktur des Roboters

4.3.3 Timing

Im ursprünglichen Programm des Roboters, wurde der einzige vorhandene Regler in der Hauptschleife des Programms ausgeführt. Dies führte dazu, dass der Regler nicht immer in den geplanten 5ms ausgeführt wurde, weil ein Durchlauf der Schleife 2 bis 4ms beansprucht und wegen anderer Codes in der Schleife in Dauer variiert. Bis auf die leichte Unregelmäßigkeit der Ausführung des Reglers, spielte dies vorher keine Rolle, weil die vom Positionssensor BNO055 benutzten Werte sowieso nur alle 10ms aktualisieren. Mit dem Hinzufügen zusätzlicher Funktionen in der Hauptschleife, hauptsächlich für die kabellose Kommunikation mit dem Kontroller, ist die Durchlaufzeit der Schleife auf ungefähr 70ms angestiegen, so dass die im vorherigen Teil beschriebene Regler Struktur nicht oft genug ausgeführt werden kann. Daher wird der Regelvorgang nun durch einen Timer Interrupt ausgelöst. Dies passiert alle 10ms, denn dies entspricht der Aktualisierungsrate des Positionssensors BNO055. Nichtsdestotrotz nimmt der gesamte Regelvorgang ungefähr 7ms in Anspruch, so dass der Regelvorgang nicht mehr alle 5ms ausgeführt werden kann. Des Weiteren wurde festgestellt, dass das Auslesen des Positionssensors BNO055 am Anfang des Regelvorgangs 1 bis 2,3ms in Anspruch nimmt, so dass ab Beginn des Auslesens des Sensors ein Timer beginnt, der erst nachdem 2,3ms vergangen sind, den nächsten Befehl erlaubt, damit die anschließenden Regler immer bei fast genau 10ms Abstand ausgeführt werden. Das Auslesen der Encoder Signale erfolgt mithilfe von Interrupts, so dass diese immer dann gezählt werden, wenn der Arduino eine steigende oder fallende Flanke auf einem Kanal der Encoder von beiden Motoren registriert. Die Hauptschleife wird also nur dann ausgeführt, wenn gerade kein Encoder Signal registriert wird und nicht die Regler Struktur des Roboters ausgeführt wird. Die zurückgelegte Distanz und Drehgeschwindigkeit werden aber aus den gezählten Encoder Signalen im gleichen Timer Interrupt wie die Regler berechnet, also ebenfalls mit einer Abtastrate von 10ms.

4.3.4 Berechnung der Position und Drehgeschwindigkeit der Motoren

Die Bestimmung der Distanz, die der Roboter zurückgelegt hat und der Drehgeschwindigkeit der Räder erfolgt mithilfe der Encoder an den Motoren. Zuerst wird das Signal, das vom Encoder auf die 2 Kanalausgänge gegeben wird, analysiert. Mithilfe des Serial Plotters der Arduino IDE, konnten folgende Abbildungen erstellt werden, wenn man das Rad mit der Hand in eine der beiden Richtungen dreht. In Abbildung 4-4 sieht man die Ausgänge beider Kanäle bei positiver und negativer Drehrichtung.

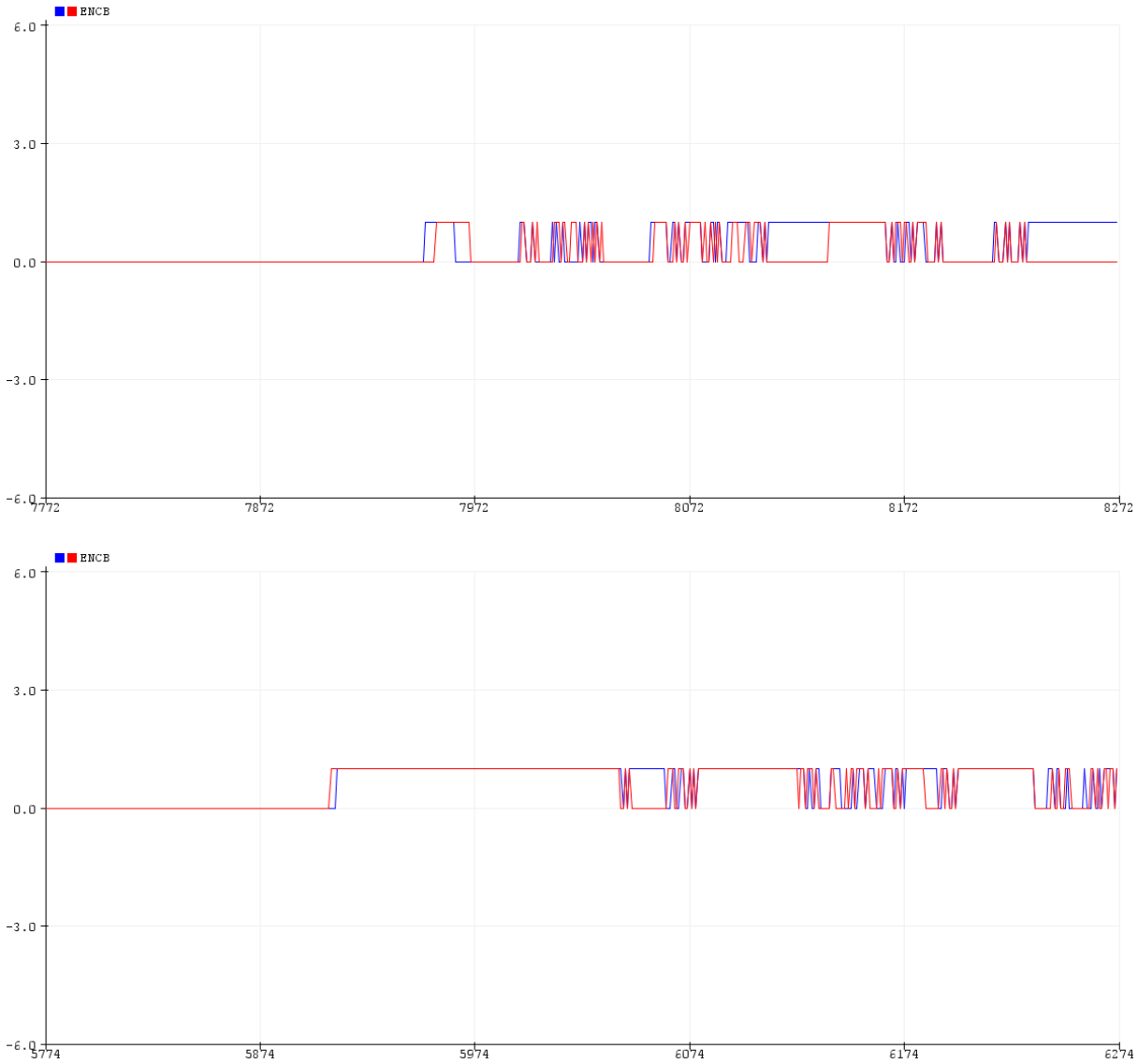

Abbildung 4-4: Verlauf der Encodersignale bei positiver (oben) und negativer (unten) Drehrichtung

Es wird schnell ersichtlich, dass bei positiver Drehrichtung das Signal von Kanal A zuerst von 0 auf 1 wechselt und dann, während dieser Kanal noch auf 1 ist, Kanal B nachzieht. Anschließend sinkt Kanal A wieder auf 0, während Kanal B auf 1 bleibt und danach erst mit Verzögerung sinkt auch Kanal B wieder auf 0. Kanal B ist also nacheilend zu Kanal A. Bei umgekehrter Drehrichtung kehrt sich diese Aussage um und Kanal A eilt Kanal B nach. Daraus lässt sich nun eine Methode erstellen, die Encoder Signale zu einem Zähler hinzuzählt oder abzählt je nach Drehrichtung. Beim zweiten Motor müssen Kanal A und B vertauscht werden, da der Motor um 180° gedreht ist und daher vom Motor aus gesehen in die umgekehrte Richtung zum anderen Motor drehen muss, damit beide Motoren vom Roboter aus gesehen in die gleiche Richtung drehen.

Zuerst werden an den beiden Pins, wo ein Encoder Kanal A angeschlossen ist, ein Interrupt angehängt, der ausgelöst wird, wenn Kanal A den Zustand wechselt, also auf dessen Pin eine wechselnde Flanke erkannt wird. Im Interrupt wird anschließend der Zustand von Kanal B überprüft. Ist dieser gleich wie Kanal A, so ist dieser voreilend und es wird +1 zu einem Zähler

addiert. Ist der Kanal B jedoch verschieden von Kanal A, so ist dieser nacheilend und es wird -1 vom Zähler subtrahiert. Dies führt dazu, dass bei positiver Drehrichtung der Zähler hochzählt und bei negativer runterzählt. Anschließend werden die Werte von beiden Zählern summiert und durch 2 geteilt, um die Fortbewegungsdistanz in gezählten Signalen auszudrücken.

Zusätzlich wird die Bibliothek „atomic/util.h“ importiert. Dies ermöglicht das Auslesen des Encoder Zählers ohne Lesefehler, wenn gleichzeitig das Interrupt ausgelöst und auf den Zählerwert zugegriffen wird.

Dies ist aber für den normalen Gebrauch ziemlich nichts sagend. Um dies in cm umzurechnen, muss daher ermittelt werden, wie viele Encoder Signale pro Umdrehung des Rades gezählt werden. Dafür wurde ein Rad an einer Stelle mit Klebestreifen markiert und anschließend das Rad so gedreht, dass diese Markierung 10-mal an der gleichen Position vorbeikommt. Danach wurden die gezählten Encoder Signale durch 10 geteilt, um eine relativ präzise Anzahl an gezählten Signalen pro Umdrehung des Rades zu ermitteln. Das Resultat ist 988 Signale pro Umdrehung. Nun kann man mithilfe dieser Zahl und des Radumfangs, der im ersten Teil dieser Arbeit auf 20,42cm bestimmt wurde, die zurückgelegte Distanz in cm umrechnen: $x[cm] = \frac{20{,}42cm}{988ENC} \cdot x[ENC]$ (ENC ist hier eine Abkürzung für gezählte Encoder Signale.)

Die Drehgeschwindigkeit lässt sich aus der Ableitung der zurückgelegten Distanz ermitteln, also auch $v = \dot{x} = \frac{dx}{dt} = \frac{\Delta x}{\Delta t}$. Es wird die Differenz der gezählten Encodersignale durch den Zeitunterschied geteilt:

$$v\left[\frac{ENC}{ms}\right] = \frac{(position - previousPostion)}{sampletime}$$

Auch dies kann man dann umrechnen in praktikablere Einheiten:

$$v[\frac{cm}{s}] = \frac{(position - previousPostion) \cdot 1000 \cdot 20{,}42}{988 \cdot sampletime}$$

$$v[RPM] = \frac{(position - previousPostion) \cdot 1000 \cdot 60}{988 \cdot sampletime}$$

(„position“ bezeichnet hier die aktuell gezählten Signale, „previousposition“ die bei der letzten Berechnung gezählten Signale und „sampletime“ steht für die Abtastrate.)

4.3.5 Implementierung der Regler Struktur

Die 5 PID Regler werden mithilfe der gleichen Bibliothek wie der alte Regler in der „SelfRobo“ Bibliothek erstellt. Daher wird zuerst die PID_v1 Bibliothek, die man auf der Arduino Website finden kann, importiert. Diese Bibliothek wurde auch für die ursprüngliche Implementierung des Roboters genutzt. Allerdings wurde eine kleine Änderung vorgenommen an der Bibliothek und eine Zeile ausgeklammert, so dass der PID auch ausgeführt wird, wenn weniger Zeit als die Abtastzeit vergangen ist. Dies resultiert darin, dass wenn einige Mikrosekunden zu wenig vergangen sind, nicht noch einmal die ganze Abtastzeit abgewartet werden muss. Anschließend wurden die 5 PID Regler balancePID, drivePID, distancePID, turnPID und directionPID erstellt. Dann werden für jeden Regler die Grenzwerte, Abtastzeit und K_P, K_I und

K_D Werte eingestellt. Für die Implementierung des Timers wurde dann die TimerOne Bibliothek importiert. Dann wurde der Timer1 mit 10ms Wartezeit eingestellt, der die Funktion „balanceRobot“ aufruft. In dieser Funktion werden dann alle nötigen Funktionen zum Regeln des Roboters ausgeführt, wie das Auslesen des Yaw und Pitch Winkels aus dem BNO055 mit Berechnung der Wendegeschwindigkeit, Zählen der Encoder Signale mit Berechnung der Distanz und Geschwindigkeit und anschließende Berechnung der Regler mit Ausgabe auf den Motortreiber zum Ansteuern der Motoren. Im Timer läuft also die vorher beschriebene Regler Struktur ab. Bei der Implementierung des Timers muss allerdings beachtet werden, dass die Pins 11 und 12 ihre PWM Funktion verlieren, wenn der Timer1 genutzt wird.

4.3.6 Regler Werte

Die Regler Werte müssen natürlich für jeden PID individuell eingestellt werden. Die Abtastrate wird logischerweise für jeden PID auf 10ms gesetzt, weil sie alle 10ms vom Timer ausgelöst werden. Die Grenzwerte für die Ansteuerung der Motoren des balancePID betragen -255 und 255, weil der Motortreiber maximal 255 als Wert zulässt. Die Grenzwerte der restlichen 4 PIDs und die K_P, K_I und K_D Werte der 5 Regler wurden anschließend experimentell ermittelt. Dafür wurde teilweise für eine genauere Bestimmung der Werte die Methode der Dichotomie genutzt. Dabei wird zunächst ein Wert ausprobiert und die Reaktion des Reglers beobachtet. Wenn der Wert zu hoch ist, wird ein niedriger genommen und wenn zu niedrig ein höherer Wert. Ist der Wert nun anstatt zu hoch zu niedrig oder umgekehrt, dann wird der Mittelwert vom nahestehenden hohen und niedrigen Wert genommen. Danach wird bis zum Erreichen einer zufriedenstellenden Regelung der Mittelwert aus dem vorherigen berechneten Mittelwert und den näheren der beiden vorherigen Werte genommen.

Beim balancePID musste mit dem Wechsel von den alten auf die neuen Motoren, die Regler Werte angepasst werden. Hauptsächlich der I-Anteil musste beträchtlich erhöht werden. Der drivePID war zuerst nur schwer zufriedenstellend einzustellen. Das Problem lag darin, dass der vorherige balancePID zu aggressiv eingestellt war und sofort auf die kleinste Abweichung reagierte. Dies ist im Stillstand unproblematisch, führt während dem Fahren jedoch zu einer sehr ungleichmäßigen Fahrt. Ein Senken der 3 Werte des balancePID führte dazu, dass der Roboter das Gleichgewicht mit weniger und schwächeren Impulsen regelt und damit zu einer viel ruhigeren und gleichmäßigeren Fahrt führt. Bei DrivePID erwies sich nur ein reiner I-Anteil als ideal. Ein kleiner zusätzlicher P-Anteil macht die Fahrt minimal weniger verzögert, führt insgesamt aber zu einem schlechteren Gleichgewicht des Roboters. Beim turnPID erwies sich nur ein PI-Regler als zufriedenstellend. Schlussendlich wurden für den distancePID eine PD-Regelung und für den directionPID eine PI-Regelung als optimal befunden.

Für den Grenzwert des drivePID wurde -6 und +6 Grad festgelegt, da mehr nicht stabil ist und im Falle, wo mehr bei einer Steigung benötigt wird, der Motor nicht die nötige Leistung besitzt, um das Gewicht des Roboters fortzubewegen in die gewollte Richtung. Bei turnPID erwies sich -30 und +30 als Grenzwert, wo der Roboter schnell drehen kann ohne das Gleichgewicht zu gefährden. Anschließend wurde bei directionPID eine Wendegeschwindigkeit von maximal

20°/s festgelegt. Beim distancePID wurde die maximale Fahrgeschwindigkeit auf 12cm/s begrenzt. Bei beiden letzteren Reglern führen höhere Geschwindigkeiten zu einer schnelleren Fortbewegung auf Kosten der Stabilität des Gleichgewichts. Insgesamt handelt es sich daher bei den Regler Werten um einen Kompromiss zwischen Stabilität des Gleichgewichts und Geschwindigkeit des Systems. Der Roboter wurde auf unterschiedlichen Unterlagen getestet bis er sie abfahren konnte, ohne aus dem Gleichgewicht zu kommen. Dabei handelte es sich hauptsächlich um recht welligen alten Laminatboden, Kachelboden und Tische. Trotzdem kann der Roboter bei plötzlichen abrupten Neigungen kippen und es wird generell empfohlen sichtbare Neigungswechsel nicht mit Höchstgeschwindigkeit zu nehmen.

In Tabelle 4-1 werden die endgültigen eingestellten Werte aufgelistet.

Tabelle 4-1: Auflistung der Regler mit ihren Parametern

Regler	K_P	K_I	K_D	Grenzwert 1	Grenzwert 2	Regler Typ
balancePID	30	275	3.5	-255	+255	PID
drivePID	0	6	0	-6	+6	I
turnPID	0,5	0,5	0	-30	+30	PI
distancePID	0,5	0	0,05	-12	+12	PD
directionPID	5	1	0	-20	+20	PI

4.3.7 Speicherung eines Pfadpunktes

Im zweiten Modus empfängt der Roboter Pfadpunkte von der Fernbedienung. Ein Pfadpunkt besteht aus einer Distanz, die der Roboter abfahren soll und einem Yaw Winkel, um den der Roboter seine Richtung anpassen soll. Dafür gibt es sowohl im Paket, das der Roboter sendet als auch dem Paket, das er empfängt, eine Variable „update“. Wenn der Roboter seinen Punkt erreicht hat, dann sendet der Roboter update=1. Am Anfang ist dies der Fall, da dann die Distanz und der Yaw Winkel 0 sind. Das heißt, dass der Roboter hochzählen soll beim Zähler, der den Pfadpunkt des Pfades bezeichnet, also den nächsten Punkt schicken soll. Wenn der Kontroller dann den nächsten Punkt sendet, sendet er ebenfalls update=1 damit der Roboter weiß, dass er den neuen Punkt erhalten hat. Dann sendet der Roboter wieder update=0, da er nun den neuen Pfadabschnitt abfährt und der Kontroller sendet auch wieder update=0, da er nun den aktuellen Punkt sendet. Jedes Mal, wenn der Roboter seinen Pfadabschnitt beendet, wiederholt sich dieser ganze Vorgang.

Die Distanz und der Yaw Winkel werden unter den gleichen Variablen x und y gesendet, wie im ersten Modus die Joystick Eingaben. Der Roboter hat einen Pfadpunkt dann erreicht, wenn die Abweichung der Ist- und Sollposition weniger als 1cm und 2° beträgt. Die Werte des folgenden Punkts werden immer zu dem vorherigen Punkt addiert und nicht zu der aktuellen Position des Roboters.

4.3.8 Robo.ino

Nachdem nun die Überlegungen hinter den einzelnen Aspekten des Codes erklärt wurden, folgt nun ein vereinfachtes Flussdiagramm in Abbildung 4-5 und anschließend der gesamte Programmcode des Roboters mit Kommentaren zur besseren Verständlichkeit. Die „SelfRobo.h“ Bibliothek kann im Anhang aufgesucht werden.

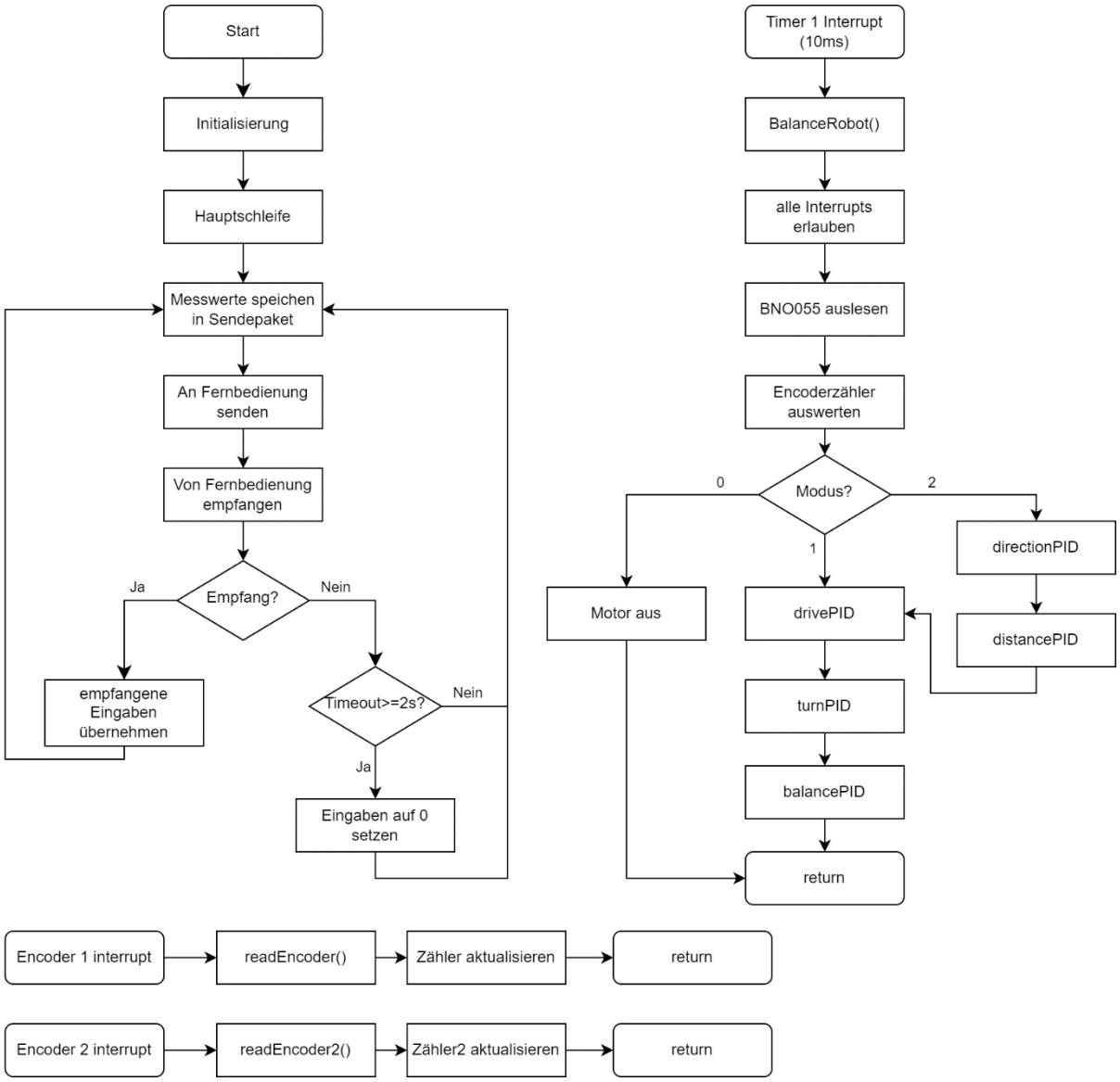

Abbildung 4-5: Vereinfachtes Flussdiagramm von Robo.ino

```
#include "SelfRobo.h"
#include <TimerOne.h>
#include "Wire.h"
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <util/atomic.h>
#include <PID_v1.h>
//Importieren der Bibliotheken

int CEPin = 7; //nrf24L01 CE Pin
```

```
int CSNPin = 8; //nrf24L01 CSN Pin

//Pins für Motortreiber
int leftForward = 13;
int leftBackward = 10;
int leftSpeedPin = 9;
int rightForward = 4;
int rightBackward = 5;
int rightSpeedPin = 6;

//Encoder Interrupt Pins
#define ENCA 18
#define ENCB 19
#define ENCA2 2
#define ENCB2 3

int sampletime = 10; //Abtastzeit in ms

volatile long posi = 0; //Zähler von Encoder1
volatile long posi2 = 0; //Zähler von Encoder2
double rpmAverage = 0; //Durchschnitt RPM der beiden Motoren
double motorPower1 = 0; //Motorwert1 für Motortreiber
double motorPower2 = 0; //Motorwert2 für Motortreiber
double turnSpeed = 0; //Wendegeschwindigkeit
double previousYawAngle = 0; //vorheriger Yawwinkel
double startYawAngle = 0; //Anfangsyawwinkel
long distance = 0; //abgefahrene Distanz von Encoder1
long previousDistance = 0; //vorherige abgefahrene Distanz von Encoder1
long distance2 = 0; //abgefahrene Distanz von Encoder2
long previousDistance2 = 0; //vorherige abgefahrene Distanz von Encoder2
double distanceAverage = 0; //Durchschnitt der abgefahrenen Distanz beider Motoren
double distanceCM = 0; //Abgefahrene Distanz in cm
double previousDistanceCM = 0; //Vorherige abgefahrene Distanz in cm
double startDistance = 0; //Anfangsdistanz
double speed = 0; //Geschwindigkeit Encoder1
double speed2 = 0; //Geschwindigkeit Encoder2
double speedAverage = 0; //Durchschnitt der Geschwindigkeit beider Motoren
double speedENCs = 0; //Geschwindigkeit in ENC/s
double speedENCms = 0; //Geschwindigkit in cm/s
double speedCMs = 0; //Geschwindigkeit in cm/s
double driveSetpointCMs = 0; //Zielgeschwindigkeit

//Istwerte für Regler
double balanceInput = 0;
double driveInput = 0;
double turnInput = 0;
double distanceInput = 0;
double directionInput = 0;

//Ausgangsvariablen für Regler
double balanceOutput = 0;
double driveOutput = 0;
double turnOutput = 0;
double distanceOutput = 0;
double directionOutput = 0;

//Sollwerte für Regler
double balanceSetpoint = -1;
double driveSetpoint = 0;
```

```
double turnSetpoint = 0;
double distanceSetpoint = 0;
double directionSetpoint = 0;

//Hilfsvariablen zum Einstellen der Regler
double Kp = 0.5;
double Ki = 0;
double Kd = 0.05;
float newSetpoint = -1;

//Deklarationen der PID Regler mit Zeiger auf Eingang-, Ausgang- und Stellwert und
Festlegung der Reglerwerte
PID balancePID(&balanceInput, &balanceOutput, &balanceSetpoint, 30, 275, 3.5,
DIRECT);
PID drivePID(&driveInput, &driveOutput, &driveSetpoint, 0, 6, 0, DIRECT);
PID turnPID(&turnInput, &turnOutput, &turnSetpoint, 0.5, 0.5, 0, DIRECT);
PID distancePID(&distanceInput, &distanceOutput, &distanceSetpoint, 0.5, 0, 0.05,
DIRECT);
PID directionPID(&directionInput, &directionOutput, &directionSetpoint, 5, 1, 0,
DIRECT);

//Deklaration aus SelfRobo Bibliothek für ursprüngliche Programmation des
Motortreibers und BNO055
SelfRobo mySelfRobo(leftForward, leftBackward, leftSpeedPin, rightForward,
rightBackward, rightSpeedPin, 170, 1.5, 8);
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28); //BNO055 mit I2C Adresse 0x28

RF24 radio(CEPin, CSNPin); //RF24 Objekt mit CE und CSN erstellen
const byte addresses[][6] = {"00100", "00101"}; //Ausgewählte Kanäle zur
Kommunikation

// Paket das vom Kontroller gesendet wird mit maximal 32 Bytes pro Paket
struct DataPackage1 {
   float x = 0;
   float y = 0;
   int power = 0;
   float Kp;
   float Ki;
   float Kd;
   float newSetpoint;
   int update = 0; //1 = Kontroller sendet nächster Punkt
};
//Paket das vom Roboter gesendet wird mit maximal 32 Bytes pro Paket
struct DataPackage2 {
   float angle = 0;
   float speed = 0;
   float position = 0;
   float Kp;
   float Ki;
   float Kd;
   float newSetpoint;
   int update = 0; //1 = Robo fordert nächster Pfadpunkt von Kontroller an
};

DataPackage1 data1; //Empfangspaket erstellen
DataPackage2 data2; //Sendepaket erstellen

String connection = "Not Connected"; //Verbindungszustand
int timeout = 0; //Verbindungsverlust Timer in ms
```

```
unsigned long firstDisconnect = 0; //Zeitpunkt erstes Verbindungsverlusts
unsigned long currentDisconnect = 0; //Zeitpunkt aktuellsten Verbindungsverlusts

unsigned long firstMillis = 0; //Zählervariablen zur Verzögerung des BNO055
unsigned long nextMillis = 0;

void setup()
{
  Serial.begin(9600); // Baud COM
  bno.begin(); //BNO055 starten
  Wire.begin(); //I2C Kommunikation starten
  Wire.setClock(400000); //I2C Kommunikationsfrequenz
  radio.begin(); //nRF24L01 Kommunikation starten
  radio.openWritingPipe(addresses[0]); // 00100 Sendeadresse setzen
  radio.openReadingPipe(1, addresses[1]); // 00101 Empfangsadresse setzen
  radio.setPALevel(RF24_PA_HIGH); //Sendestärke
  radio.setDataRate(RF24_250KBPS); //Senderate

  //Anfangsreglerwerte in Sendepaket speichern
  data2.Kp = Kp;
  data2.Ki = Ki;
  data2.Kd = Kd;
  data2.newSetpoint = newSetpoint;
  data1.Kp = Kp;
  data1.Ki = Ki;
  data1.Kd = Kd;
  data1.newSetpoint = newSetpoint;

  // Timer Interrupt
  Timer1.initialize(sampletime*1000); //Timer1 in Mikrosekunden (Timer1)
  Timer1.attachInterrupt(BalanceRobot); //BalanceRobot durch Timer1 auslösen

  //Pins für Encoder als Input
  pinMode(ENCA,INPUT);
  pinMode(ENCB,INPUT);
  pinMode(ENCA2,INPUT);
  pinMode(ENCB2,INPUT);
  attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,CHANGE); //readEncoder
bei wechselnder Flanke auf ENCA
  attachInterrupt(digitalPinToInterrupt(ENCA2),readEncoder2,CHANGE);
//readEncoder2 bei wechselnder Flanke auf ENCA2

  //Regler Grenzwerte, Abtastrate und Modus festlegen
  balancePID.SetOutputLimits(-255,255);
  balancePID.SetSampleTime(sampletime);
  balancePID.SetMode(AUTOMATIC);
  drivePID.SetOutputLimits(-6,6);
  drivePID.SetSampleTime(sampletime);
  drivePID.SetMode(AUTOMATIC);
  turnPID.SetOutputLimits(-30,30);
  turnPID.SetSampleTime(sampletime);
  turnPID.SetMode(AUTOMATIC);
  distancePID.SetOutputLimits(-12,12);
  distancePID.SetSampleTime(sampletime);
  distancePID.SetMode(AUTOMATIC);
  directionPID.SetOutputLimits(-20,20);
  directionPID.SetSampleTime(sampletime);
  directionPID.SetMode(AUTOMATIC);
}
```

```
void loop(){ //Hauptschleife
  data2.angle = directionInput; //Eingang in directionPID in Sendepaket
  data2.speed = speedCMs; //Geschwindigkeit in cm/s in Sendepaket
  data2.position = distanceInput; //Eingang in distancePID in Sendepaket

  //Geänderter Reglerwert auf nicht ausgeklammerten PID anwenden
  if(data1.Kp != Kp || data1.Ki != Ki || data1.Kd != Kd){
    Kp = data1.Kp;
    data2.Kp = Kp;
    Ki = data1.Ki;
    data2.Ki = Ki;
    Kd = data1.Kd;
    data2.Kd = Kd;
    //balancePID.SetTunings(Kp,Ki,Kd);
    //drivePID.SetTunings(Kp,Ki,Kd);
    //turnPID.SetTunings(Kp,Ki,Kd);
    //distancePID.SetTunings(Kp,Ki,Kd);
    //directionPID.SetTunings(Kp,Ki,Kd);
  }
  if(data1.newSetpoint != newSetpoint){
    newSetpoint = data1.newSetpoint;
    data2.newSetpoint = newSetpoint;
    //balanceSetpoint = newSetpoint;
    //driveSetpointCMs = newSetpoint;
    //turnSetpoint = newSetpoint;
    //distanceSetpoint = newSetpoint;
    //directionSetpoint = newSetpoint;
  }

  delay(5); //5ms warten
  radio.stopListening(); //Nicht mehr empfangen
  radio.write(&data2, sizeof(data2)); //Paket senden an Roboter
  delay(5); //5ms warten
  radio.startListening(); //Empfangsmodus
  if (radio.available()) { //Wenn erreichbar
    while (radio.available()) { //Während erreichbar
      radio.read(&data1, sizeof(data1)); //Paket von Roboter auslesen
      timeout = 0;
      firstDisconnect = 0;
      connection = "Connected";
      if(data1.power == 1){ //Wenn in Modus1
        driveSetpointCMs = 1.2*data1.x; //Fahrgeschwindigkeit übernehmen
        turnSetpoint = data1.y*2; //Wendegeschwindigkeit übernehmen
      }
      if(data1.power == 2){ //Wenn in Modus2
        distanceSetpoint = data1.x; //Pfadpunktdistanz übernehmen
        directionSetpoint = data1.y; //Pfadpunktrichtung übernehmen
      }

    }
  }
  else if(!radio.available()){ //Wenn nicht erreichbar
    currentDisconnect = millis(); //Aktueller Zeitpunkt in ms speichern
    if(firstDisconnect == 0){
      firstDisconnect = currentDisconnect;
    }
    timeout = currentDisconnect - firstDisconnect;
```

```
    if (timeout >= 2000) { //Wenn Timeout länger als 2 Sekunden dann Verbindung
verloren
      data1.x = 0;
      data1.y = 0;
      connection = "Not connected";
    }
  }
}

void BalanceRobot() { //Funktion die Reglerstruktur beinhaltet
  sei(); //Interrupts für IC2 ermöglichen innerhalb von Interrupt um BNO055
auslesen zu können
  firstMillis = micros(); //aktueller Zeitpinkt speichern
  mySelfRobo.Update(); //Yawwinkel und Pitchwinkel von BNO055 erhalten
  nextMillis = micros(); //aktueller Zeitpinkt speichern
  //Warten bis 2,3ms vergangen seit Anfang Auslesen BNO055, weil Sensor auslesen
1ms bis 2,3ms dauert damit PID gleichmässig rechnet
  while((nextMillis - firstMillis)<2300){
    nextMillis = micros();
  }

  if(data1.power >= 1){ //Wenn in Modus 1 oder 2
    long pos = 0; //Position lesen in atomic block um Fehler zu vermeiden, wenn
Interrupt gleichzeitig mit diesem Code ausgeführt
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
      pos = posi;
    }

    distance = pos; //Zählerwert speichern

    long pos2 = 0; //Position lesen in atomic block um Fehler zu vermeiden, wenn
Interrupt gleichzeitig mit diesem Code ausgeführt
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
      pos2 = posi2;
    }
    distance2 = pos2; //Zählerwert speichern

    distanceAverage = ((distance + distance2)/2) - startDistance;
//Durschnittdistanz rechnen
    distanceCM = distanceAverage*20.42/988; //in cm umrechnen

    speed = distance - previousDistance; //Geschwindigkeit 1 rechnen
    speed2 = distance2 - previousDistance2; //Geschwindigkeit 2 rechnen
    previousDistance = distance; //aktueller Distanz1 speichern
    previousDistance2 = distance2; //aktueller Distanz2 speichern
    speedAverage = (speed + speed2)/2; //Durchschnittgeschwindigkeit rechnen
    speedENCs = speedAverage*(1000/float(sampletime)); //Umrechnen in ENC/s
    speedENCms = speedENCs/1000; //Umrechnen in ENC/ms
    speedCMs = speedENCs*(20.42/988); //Umrechnen in cm/s

    if(data1.power == 2){ //Wenn in Modus 2
      //Wenn Pfadpunkt erreicht
      if(data2.update == 0 && abs(directionSetpoint - directionInput) < 2 &&
abs(distanceSetpoint - distanceInput) < 1){
        startDistance = startDistance + data1.x;
        startYawAngle = startYawAngle + data1.y;
        data1.x = 0;
        data1.y = 0;
        data2.update = 1;
```

```
      }

      //Wenn nächster Pfadpunkt erreicht
      if(data1.update == 1){
        data2.update = 0;
      }

      //Berechnung damit Startyawwinkel zwischen -360 und 0 liegt
      if(startYawAngle > 0){
        startYawAngle = startYawAngle - 360;
      }
      else if(directionInput < -360){
        startYawAngle = startYawAngle + 360;
      }

      directionInput = mySelfRobo.yawAngle - startYawAngle; //Richtung
zurücksetzen
      //Berechnung damit Istwinkel nicht mehr als 180° in eine Richtung beträgt
      if(directionInput > 180){
        directionInput = directionInput - 360;
      }
      else if(directionInput < -180){
        directionInput = directionInput + 360;
      }

      //Berechnung damit Yawwinkeldifferenz nicht mehr als 180° in eine Richtung
beträgt
      if(directionInput - directionSetpoint > 180){
        directionSetpoint = directionSetpoint + 360;
      }
      else if(directionInput - directionSetpoint < -180){
        directionSetpoint = directionSetpoint - 360;
      }

      directionPID.Compute(); //PID Berechnung
      turnSetpoint = directionOutput; //Ausgang als Eingang nächsten PIDs

      distanceInput = distanceCM - startDistance; //Distanz zurücksetzen
      distancePID.Compute(); //PID Berechnung
      driveSetpointCMs = distanceOutput; //Ausgang als Eingang nächsten PIDs
    }

    driveSetpoint = 0.99*driveSetpoint + 0.01*driveSetpointCMs*(988/(1000*20.42));
//Umwandlung von cm/s in ENC/ms und Dämpfung
    driveInput = speedENCms; //Istwert in PID
    drivePID.Compute(); //PID Berechnung

    //Berechnung damit Yawwinkeldifferenz nicht mehr als 180° in eine Richtung
beträgt und Berechnung der Wendegeschwindigkeit
    if(abs(mySelfRobo.yawAngle - previousYawAngle) <= 180){
      turnSpeed = (mySelfRobo.yawAngle -
previousYawAngle)*(1000/float(sampletime));
    }
    else if(mySelfRobo.yawAngle - previousYawAngle > 180){
      turnSpeed = (mySelfRobo.yawAngle - previousYawAngle -
360)*(1000/float(sampletime));
    }
    else if(mySelfRobo.yawAngle - previousYawAngle < -180){
```

```
      turnSpeed = (mySelfRobo.yawAngle - previousYawAngle +
360)*(1000/float(sampletime));
    }
    turnInput = turnSpeed; //Geschwindigkeit
    turnPID.Compute(); //PID Berechnung
    previousYawAngle = mySelfRobo.yawAngle; //aktueller Yawwinkel speichern

    balanceSetpoint = -1 + driveOutput; //Stellwert des PID
    balanceInput = mySelfRobo.pitchAngle; //Istwert des Pitchwinkels als Eingang
des PID
    balancePID.Compute(); //PID Berechnung

    //Reduzierung und Erhöhung der Geschwindigkeit des korrekten Motors um in die
richtige Richtung zu drehen
    motorPower2 = balanceOutput-turnOutput;
    motorPower1 = balanceOutput+turnOutput;

    //Sicherstellen, dass Motorwerte nicht ausserhalb des zugelassenen Bereichs
liegen
    if(motorPower1 > 255){
      motorPower1 = 255;
    }
    if(motorPower2 > 255){
      motorPower2 = 255;
    }
    if(motorPower1 < -255){
      motorPower1 = -255;
    }
    if(motorPower2 < -255){
      motorPower2 = -255;
    }

    mySelfRobo.SetDrive(motorPower2, motorPower1);//Motortreiber erhält
Motoransteuerungswerte
  }

  if(data1.power == 1){ //Wenn in Modus 1
    startYawAngle = mySelfRobo.yawAngle;
    startDistance = distanceCM;
  }

  if(data1.power == 0){ //Wenn Power aus alles zurücksetzen
    startDistance = distanceCM; //Startdistanz aktualisieren
    distanceSetpoint = 0;
    distanceInput = 0;
    distancePID.Compute();
    distanceOutput = 0;

    startYawAngle = mySelfRobo.yawAngle; //Startyawwinkel aktualisieren
    directionSetpoint = 0;
    directionInput = 0;
    directionPID.Compute();
    directionOutput = 0;

    driveSetpoint = 0;
    driveInput = 0;
    drivePID.Compute();
    driveOutput = 0;
```

```
    balanceSetpoint = -1;
    balanceInput = -1;
    balancePID.Compute();
    balanceOutput = 0;

    previousYawAngle = mySelfRobo.yawAngle; //vorherigen Yawwinkel aktualisieren
    turnSetpoint = 0;
    turnInput = 0;
    turnPID.Compute();
    turnOutput = 0;

    mySelfRobo.SetDrive(0, 0); //Motor aus
  }
}

void readEncoder(){ //Funktion um Encoder1 Interrupts zu zählen
  int a = digitalRead(ENCA); //Kanal A
  int b = digitalRead(ENCB); //Kanal B
  if(a == b){ //Wenn positive Drehrichtung hochzählen
    posi++;
  }
  else{ //Wenn negative Drehrichtung runterzählen
    posi--;
  }
}

void readEncoder2(){ //Funktion um Encoder2 Interrupts zu zählen
  int a2 = digitalRead(ENCA2); //Kanal A
  int b2 = digitalRead(ENCB2); //Kanal B
  if(a2 == b2){ //Wenn positive Drehrichtung hochzählen
    posi2++;
  }
  else{ //Wenn negative Drehrichtung runterzählen
    posi2--;
  }
}
```

4.4 Fernbedienung Software

Die Programmierung der Fernbedienung wird nun untersucht. Im Gegensatz zum Roboter gibt es hier keinen Code, auf den man aufbauen konnte. Zuerst wird nun die Umsetzung der einzelnen Modulfunktionen erklärt und schlussendlich der gesamte Code der Fernbedienung mit Kommentaren angezeigt.

4.4.1 Joystick

Zuerst werden die X und Y-Achse des Joysticks als die Pins A0 und A1 definiert. Diese Pins werden dann vom Arduino mit „analogRead“ in 10 Bit Auflösung ausgelesen. Das bedeutet, dass der ausgelesene Wert 0 bis 1023 betragen kann. Dieser Wert wird anschließend mit „map“ auf 0 bis 20 aufgeteilt, da dies leichter zu verwerten ist. Zusätzlich wird die Mitte des Joysticks als 10, also die Hälfte des Spektrums festgelegt. Die Distanz des Joysticks von der Mitte wird dann durch das Subtrahieren von 10, also der Mitte, vom ausgelesenen Wert von

0 bis 20 erhalten. Da der Joystick durch leichte Streuung nicht unbedingt genau die Hälfte des Spektrums in der Mitte ausgibt, muss die Distanz zur Mitte mehr als 2 betragen. Leider wird der maximale Wert beim Joystick recht schnell erreicht, so dass man für die Ausgabe einer kleinen Abweichung von der Mitte des Joysticks oder Geschwindigkeit für den Roboter, den Steuerknüppel nur wenig von der Mitte fortbewegen darf.

Beim Drücken des Joystick Knopfs wird digital 0 auf dem Pin 9 ausgelesen. Ist der Knopf nicht gedrückt, wird 1 ausgelesen. Wenn erkannt wird, dass der Knopfzustand nicht mit dem vorherigen Zustand übereinstimmt und er 1 beträgt, dann wird registriert, dass der Knopf gedrückt wurde und die damit Verbundene Aktion ausgeführt. Um ein Gedrückt Halten des Codes zu registrieren, muss der Zustand 0 betragen und dann dieser Zustand 2 Sekunden lang beibehalten werden, damit die damit verbundene Aktion ausgelöst wird.

4.4.2 Bildschirmanzeige

Für die SSD1396 Anzeige werden die Bibliothek „Adafruit_GFX.h“ und „Adafruit_SSD1306.h“ für die Anzeige selbst benötigt, „Wire.h“ für die I2C Kommunikation zwischen Anzeige und Arduino Uno und „TimerOne.h“ für den Timer1. Dann wird das Objekt „display“ der Anzeige im Code erstellt mit der Auflösung, die hier 128 auf 64 Pixel beträgt und den zugewiesenen Pins. Dann wird der Anzeige die I2C Adresse 0x3D zugewiesen. Als erstes werden alle Pixel des Bildschirms ausgeschaltet mit „display.clearDisplay()“ und anschließend die Anzeige mit „display.display()“ aktualisiert, um diese Änderung anzuzeigen. Der Timer1 wird dann so gesetzt, dass er alle 100ms die Funktion „RefreshDisplay“ ausgelöst wird, also mit 10Hz. Die Anzeige kann aber auch mit 60Hz aktualisieren. 10Hz ist aber ausreichend für unsere Zwecke. In dieser Funktion werden dann wieder zuerst alle Pixel zurückgesetzt, danach erst die Befehle zum Anzeigen des Tests und der Variablen ausgeführt und zum Abschluss die Anzeige aktualisiert.

Beim Anzeigen von Variablen Werten muss allerdings beachtet werden, dass diese zuerst in „char“ umgewandelt werden mithilfe des Befehls „itoa“. Wird dies nicht getan, so kann es zu visuellen Fehlern auf der Anzeige kommen, da es dann zu Überschneidungen in der Belegung des dynamischen Speichers des Arduinos kommen kann.

4.4.3 Pfadprogrammierung

Für die Pfadprogrammierung muss in der Kondition, wenn der gewünschte Pfad ausgewählt ist, die gewünschte Distanz und Richtung für ein Punkt mit einer Zahl des Zählers i assoziiert werden. Dabei ist i=0 der erste Punkt. Wenn der Pfad sich danach wiederholen soll, muss beim Zählerwert nach dem letzten Punkt, der Zähler i wieder auf 0 gesetzt werden, um den ersten Punkt wieder abzurufen. Obwohl man eine Distanz und eine Richtung einem Pfadpunkt zuweisen kann, ist es nicht empfohlen gleichzeitig beide Werte auf einen Wert ungleich 0 zu setzen, da der Roboter bei gleichzeitiger Richtungs- und Distanzanpassung nie am gleichen Punkt rauskommen wird. Dies ist der Fall deswegen, weil der Roboter nicht mit perfekt

gleichbleibender Geschwindigkeit fährt und wendet und daher, wenn der Roboter am Anfang langsamer fährt, aber schon dreht, weiter seitlich ankommt als dies anders der Fall sein könnte. Es wird also empfohlen abwechselnd mit den einzelnen Pfadpunkten die Distanz und Richtung anzupassen, damit der Pfad vorhersehbar bleibt.

4.4.4 Funktion zur einfachen Programmierung der Regler des Roboters

Für eine einfachere Einstellung der Regler, während der Roboter herumfährt, wurde die Fernbedienung an den PC angeschlossen. Dann wurde ein Code des ursprünglichen Roboters so angepasst, dass er es erlaubt, über die serielle Schnittstelle mit einer Baudrate von 9600 die Werte einzustellen, indem man den Namen der Variabel mit anschließendem gewünschtem Wert schreibt, wie zum Beispiel „Kp10“. Im Serial Monitor der Arduino IDE werden dann die gesetzten Werte angezeigt und die Werte kabellos zum Roboter geschickt. Diese Werte wurden allerdings immer nur von einem der 5 Regler übernommen, der vorher im Code des Roboters festgelegt wurde.

4.4.5 Controller.ino

Im Folgenden wird zuerst in Abbildung 4-6 ein vereinfachtes Flussdiagramm und dann der gesamte Programmcode der Fernbedienung mit Kommentaren zur Erklärung angezeigt.

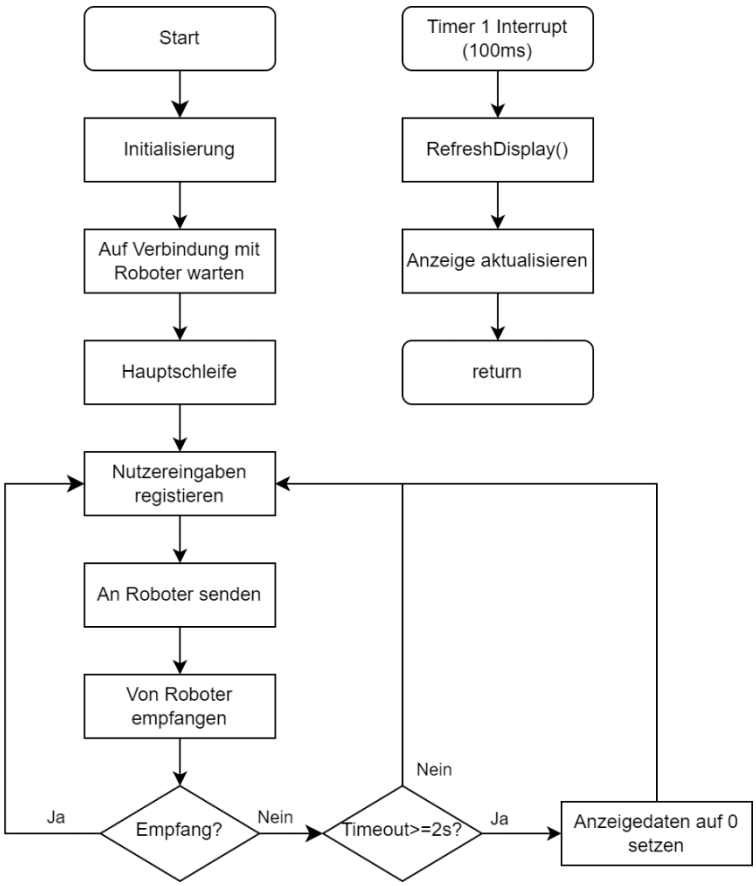

Abbildung 4-6: Vereinfachtes Flussdiagramm von Controller.ino

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <TimerOne.h>
//Importieren der Bibliotheken

#define SCREEN_WIDTH 128 // OLED Anzeige Pixelhöhe
#define SCREEN_HEIGHT 64 // OLED Anzeige Pixelbreite

//Deklaration für SSD1306 Anzeige verbunden mit I2C an SDA und SCL Pins
#define OLED_RESET     4 // Reset pin für Anzeige
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET); //Objekt
für Anzeige erstellen

int xAxis = A0; //Joystick X Achse
int yAxis = A1; //Joystick y Achse
int buttonPin = 9; //Joystick Drücken pin
int CEPin = 7; //nrf24L01 CE Pin
int CSNPin = 8; //nrf24L01 CSN Pin
int vrx; //Zum Speichern der X Position
int vry; //Zum Speichern der Y Position
int button = 1; //Zum Speichern des Drückenzustands
int lastButton = 1; //Zum Speichern des vorherigen Drückenzustands
int holdButton = 0; //Zeit in ms Joystick gedrückt
int power = 0; //Powerstatus
int path = 0; //Ausgewählter Pfad
int i = 0; //Zähler des aktuellen Pfadabschnitts
int distance = 0; //Distanz die abgefahren werden soll
int direction = 0; //gewünschte Yawwinkelkorrektur

//Parameter zum Auslesen des Joysticks
int range = 20; //Ausgabebereich für X und Y-Achse
int threshold = range / 10; //Toleranzschwelle
int center = range / 2; //Wert des Mittelpunkts

RF24 radio(CEPin, CSNPin); //RF24 Objekt mit CE und CSN erstellen

const byte addresses[][6] = {"00100", "00101"}; //Ausgewählte Kanäle zur
Kommunikation

// Paket das vom Kontroller gesendet wird mit maximal 32 Bytes pro Paket
struct DataPackage1 {
   float x = 0;
   float y = 0;
   int power = 0;
   float Kp;
   float Ki;
   float Kd;
   float newSetpoint;
   int update = 0; //1 = Kontroller sendet nächster Punkt
};
//Paket das vom Roboter gesendet wird mit maximal 32 Bytes pro Paket
struct DataPackage2 {
   float angle = 0;
   float speed = 0;
   float position = 0;
```

```
    float Kp;
    float Ki;
    float Kd;
    float newSetpoint;
    int update = 0; //1 = Roboter fordert nächster Pfadpunkt von Kontroller an
};
DataPackage1 data1; //Sendepaket erstellen
DataPackage2 data2; //Empfangspaket erstellen

char c; //Zum Speichern der seriellen Eingabe bei Verbindung mit PC
char string[20]; //Zum Umwandeln von Variablen in String für die Anzeige
String connection = "Not connected"; //Verbindungszustand
int timeout = 0; //Verbindungsverlust Timer in ms
unsigned long firstDisconnect = 0; //Zeitpunkt erstes Verbindungsverlusts
unsigned long currentDisconnect = 0; //Zeitpunkt aktuellsten Verbindungsverlusts
int hold = 0; //Joystick Gedrückt Timer in ms
unsigned long firstHold = 0; //Zeitpunkt Joystick angefangen zu drücken
unsigned long currentHold = 0; //Zeitpunkt aktuell gedrückt

void setup() {
  Serial.begin(9600); // Baud COM
  pinMode(buttonPin,INPUT); //Pin für Joystick Drücken als Input
  digitalWrite(buttonPin,HIGH); //interner Pullup-Resistor ermöglichen
  radio.begin(); //nRF24L01 Kommunikation starten
  radio.openWritingPipe(addresses[1]); // 00101 Sendeadresse setzen
  radio.openReadingPipe(1, addresses[0]); // 00100 Empfangsadresse setzen
  radio.setPALevel(RF24_PA_HIGH); //Sendestärke
  radio.setDataRate(RF24_250KBPS); //Senderate

  // SSD1306_SWITCHCAPVCC = Anzeigespannung von 3.3V intern
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3D)) { // Adresse I2C 0x3D für 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Nicht fortfahren, Schleife wiederholt sich für immer
  }
  display.clearDisplay(); //Pixel zurücksetzen
  display.display(); //Anzeige aktualisieren

  // Timer Interrupt
  Timer1.initialize(100000); //Anzeigeaktualisierungsrate in Mikrosekunden
(Timer1)
  Timer1.attachInterrupt(RefreshDisplay); //RefreshDisplay durch Timer1 auslösen

  radio.startListening(); //nRF24L01 im Empfangsmodus
  while(connection == "Not connected"){
    if(radio.available()){ //Wenn erreichbar
      while (radio.available()) { //Während erreichbar
        radio.read(&data2, sizeof(data2)); //Datenpaket von Roboter lesen
        timeout = 0;
        firstDisconnect = 0;
        connection = "Connected";
        data1.Kp = data2.Kp;
        data1.Ki = data2.Ki;
        data1.Kd = data2.Kd;
        data1.newSetpoint = data2.newSetpoint;
      }
    }
  }
}
```

```
void loop() { //Hauptschleife
  delay(5); //5ms warten
  radio.stopListening(); //Nicht mehr empfangen
  vrx = -readAxis(xAxis); //Funktion zur Joystickauswertung aufrufen
  vry = readAxis(yAxis);
  if(power == 1){ //Wenn in Modus 1
    data1.x = vrx;
    data1.y = vry;
  }
  if(power == 2){ //Wenn in Modus 2
    if(vry >= 5){ //Joystick nach links, dann Pfad 1
      path = 1;
      i = 0;
      data1.update = 1;
    }
    else if(vrx >= 5){ //Joystick nach oben, dann Pfad 2
      path = 2;
      i = 0;
      data1.update = 1;
    }
    else if(vry <= -5){ //Joystick nach rechts, dann Pfad 3
      path = 3;
      i = 0;
      data1.update = 1;
    }
    else if(vrx <= -5){ //Joystick nach unten, dann Pfad 4
      path = 4;
      i = 0;
      data1.update = 1;
    }
    if(path == 0){ //kein Pfad ausgewählt
      distance = 0;
      direction = 0;
    }
    if(path == 1){ //Pfad 1: Dreieck mit Seitenlänge 50cm
      if(i == 4){ //Pfad hat Ende erreicht, also zurücksetzen
        i = 0;
      }
      if(i == 0){
        distance = 50; //Distanz in cm zu Punkt 1
        direction = 0; //Direktion zu Punkt 1
      }
      if(i == 1){
        distance = 0;
        direction = 60;
      }
      if(i == 2){
        distance = -50;
        direction = 0;
      }
      if(i == 3){
        distance = 0;
        direction = 60;
      }
    }
    if(path == 2){ //Pfad 2: Quadrat mit 1m Seitenlänge
      if(i == 2){ //Pfad hat Ende erreicht, also zurücksetzen
        i = 0;
      }
```

```
      if(i == 0){
        distance = 100; //Distanz in cm zu Punkt 1
        direction = 0; //Direktion zu Punkt 1
      }
      if(i == 1){
        distance = 0;
        direction = 90;
      }
    }
    if(path == 3){ //Pfad 3: Kreis
       if(i == 2){ //Pfad hat Ende erreicht, also zurücksetzen
        i = 0;
      }
      if(i == 0){
        distance = 20; //Distanz in cm zu Punkt 1
        direction = 0; //Direktion zu Punkt 1
      }
      if(i == 1){
        distance = 0;
        direction = 45;
      }
    }
    if(path == 4){ //Pfad 4: Acht
       if(i == 32){ //Pfad hat Ende erreicht, also zurücksetzen
        i = 0;
      }
      if(i<16){
        if(i%2 == 0){
          distance = 20; //Distanz in cm zu Punkt 1
          direction = 0; //Direktion zu Punkt 1
        }
        if(i%2 == 1){
          distance = 0;
          direction = 45;
        }
      }
      if(i>15 && i<32){
        if(i%2 == 0){
          distance = 20; //Distanz in cm zu Punkt 1
          direction = 0; //Direktion zu Punkt 1
        }
        if(i%2 == 1){
          distance = 0;
          direction = -45;
        }
      }
    }
    if(path != 0){ //Wenn irgendein Pfad ausgewählt
      if(data2.update == 1 && data1.update == 0){ //Nächste Punkt auswählen, wenn
Roboter anfordert
        i = i + 1;
        data1.update = 1;
      }
      if(data2.update == 0 && data1.update == 1){ //Roboter hat nächster Punkt
erhalten
        data1.update = 0;
      }
    }
    data1.x = distance; //Punktdistanz in Sendepaket
```

```
    data1.y = direction; //Punktyawwinkeldifferenz in Sendepaket
  }

  button = digitalRead(buttonPin); //Joystickknopf auslesen
  if(button != lastButton){ //Wenn Knopfzustand verändert und
    lastButton = button; //Knopfzustand speichern
    if(hold < 2000){ //Wenn nicht gedrückt gehalten
      if(button == 1){ //Wenn nicht mehr gerdückt
        if(power == 0){ //Wenn Power aus
          power = 1;
          path = 0;
        }
        else if(power == 1){ //Wenn in Modus 1
          power = 2;
          path = 0;
          i = 0;
          data1.x = 0;
          data1.y = 0;
        }
        else if(power == 2){ //Wenn in Modus 2
          power = 1;
          path = 0;
          data1.x = 0;
          data1.y = 0;
          i = 0;
        }
      }
      if(button == 0){ //Wenn Joystick gedrückt
        currentHold = millis(); //aktueller Zeitpunkt in ms speichern
        firstHold = currentHold;
      }
    }
    if(hold >= 2000){ //Knopf nicht mehr gedrückt nach 2 Sekunden gedrückt halten,
also Timer zurücksetzen
      lastButton = button;
      hold = 0;
    }
  }
  if(button == 0 && lastButton == 0){ //Knopf gedrückt gehalten
    currentHold = millis();
    hold = currentHold - firstHold;
    if(hold >= 2000) { //Länger als 2 Sekunden gedrückt dann Power aus
      power = 0;
    }
  }

  data1.power = power; //Powerzustand in Sendepaket
  radio.write(&data1, sizeof(data1)); //Paket senden an Roboter
  delay(5); //5ms warten

  radio.startListening(); //Empfangsmodus
  if(radio.available()){ //Wenn erreichbar
    while (radio.available()) { //Während erreichbar
      radio.read(&data2, sizeof(data2)); //Paket von Roboter auslesen
      timeout = 0;
      firstDisconnect = 0;
      connection = "Connected";
    }
  }
```

```
  else if(!radio.available()){ //Wenn nicht erreichbar
    currentDisconnect = millis(); //Aktueller Zeitpunkt in ms speichern
    if(firstDisconnect == 0){
      firstDisconnect = currentDisconnect;
      }
    timeout = currentDisconnect - firstDisconnect;
    if(timeout >= 2000) { //Wenn Timeout länger als 2 Sekunden dann Verbindung
verloren
      data2.angle = 0;
      data2.speed = 0;
      connection = "Not connected";
    }
  }

  if(Serial.available()) { //Wenn serielle Kommunikation mit PC
    c = Serial.read();
  }
  switch(c){ //gewünschter Reglerparameter in Sendepaket speichern und in Serial
Monitor ausgeben
    case 's':
      Serial.print("Entered Set point Value : ");
      data1.newSetpoint = Serial.parseFloat();
      data2.newSetpoint = data1.newSetpoint;
      Serial.println(data1.newSetpoint);
      break;

    case 'p':
      Serial.print("Entered Kp Value : ");
      data1.Kp = Serial.parseFloat();
      data2.Kp = data1.Kp;
      Serial.println(data1.Kp);
      break;

    case 'i':
      Serial.print("Entered Ki Value : ");
      data1.Ki = Serial.parseFloat();
      data2.Ki = data1.Ki;
      Serial.println(data1.Ki);
      break;

    case 'd':
      Serial.print("Entered Kd Value : ");
      data1.Kd = Serial.parseFloat();
      data2.Kd = data1.Kd;
      Serial.println(data1.Kd);
      break;

    default :
      break;
  }
}

void RefreshDisplay() { //Funktion zur Anzeigenaktualisierung
  sei();  //Interrupts für IC2 ermöglichen innerhalb von Interrupt
  display.clearDisplay(); //Pixel zurücksetzen
  display.setTextSize(2); //Zeichne 2X-sekalierter Text
  display.setTextColor(SSD1306_WHITE); // Zeichne weisser Text
  display.setCursor(0,0); //Oben links beginnen
  if(connection == "Connected"){  //Wenn Verbunden
```

```
    if(power == 0){
      display.println(F("Power off")); //Text zeichnen
      display.println(F("Connected"));
    }
    if(power == 1){ //Wenn in Modus 1
      display.println(F("Mode 1"));
      display.print(F("x: "));
      itoa(data1.x,string,10); //Umwandeln von float in String mit Basis 10
      display.println(string);
      display.print(F("y: "));
      itoa(data1.y,string,10); //Umwandeln von float in String mit Basis 10
      display.println(string);
      display.print(F("v: "));
      itoa(data2.speed,string,10); //Umwandeln von float in String mit Basis 10
      display.print(string);
      display.println(F("cm/s"));
    }
    if(power == 2){ //Wenn in Modus 2
      display.println(F("Mode 2"));
      if(path == 0){ //Wenn kein Pfad ausgewählt
        display.println(F("No path"));

      }
      else{
        display.print(F("Path "));
        itoa(path,string,10); //Umwandeln von float in String mit Basis 10
        display.println(string);
      }
      display.print(F("Point "));
      itoa(i,string,10); //Umwandeln von int in String mit Basis 10
      display.println(string);
      display.print(F("v: "));
      itoa(data2.speed,string,10); //Umwandeln von float in String mit Basis 10
      display.print(string);
      display.println(F("cm/s"));
    }
  }
  else if(connection == "Not connected"){ //Nicht Verbunden
    display.print(F("Searching SelfRobo"));
  }
  display.display(); //Anzeige aktualisieren
}

int readAxis(int thisAxis) {
  int reading = analogRead(thisAxis); //Analoginput auslesen
  reading = map(reading, 0, 1023, 0, range); //Analoginput auf Ausgabebereich
umwandeln

  int distance = reading - center; //Ausgabedistanz zur Mitte berechnen
  if (abs(distance) < threshold) { //Wenn Ausgabedistanz ausserhalb
Toleranzbereich
    distance = 0;
  }
  return distance; //Distanz zuückgeben
}
```

5 Gebrauchsanleitung des fertigen Roboters

Im Folgenden handelt es sich um eine Anleitung zur Benutzung des Roboters.

Um den Roboter zu benutzen, ist es zuerst nötig den Roboter und die Fernbedienung einzuschalten. Der Roboter wird eingeschaltet, indem man die USB-Kabel vom Boost Converter und dem Arduino Mega an die Powerbank anschließt, damit diese mit Strom versorgt sind. Zum Aufladen des Roboters muss ein Micro-USB Kabel mit Stromquelle an die Powerbank angeschlossen werden. Die Fernbedienung wird durch das Verbinden der Batterien mit dem Arduino Uno eingeschaltet.

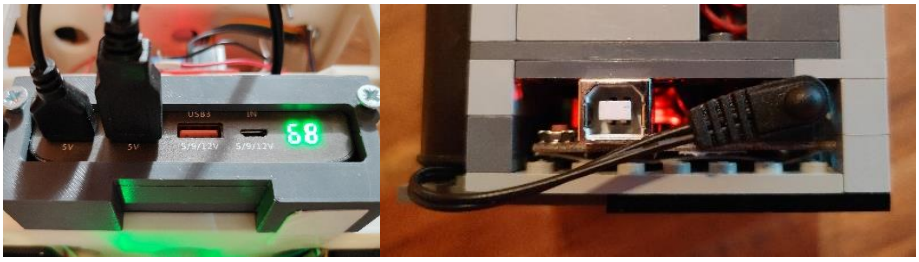

Abbildung 5-1: Anschließen der Stromversorgung des Roboters (links) und des Kontrollers (rechts)

Sobald die Fernbedienung eingeschaltet ist, wird diese nach dem Roboter suchen und „Searching Selfrobo“ anzeigen, bis er mit dem Roboter verbunden ist. Die Motoren des Roboters bleiben ausgeschaltet.

Abbildung 5-2: Foto der Fernbedienungsanzeige, wenn nicht verbunden (links) und verbunden (rechts)

Um dies zu ändern, muss der Joystick gedrückt werden. Nun befindet der Roboter sich im ersten Modus, wo man diesen mit dem Joystick steuern kann. Dabei ist zu beachten, dass der Joystick nach links den Roboter in die positive Drehrichtung wendet. Es werden zusätzlich die Ausgabewerte des Joysticks x und y und die Geschwindigkeit v des Roboters in cm/s angezeigt.

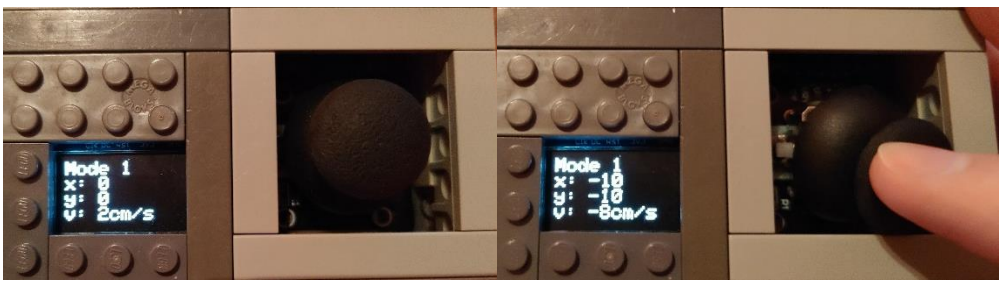

Abbildung 5-3: Foto der Fernbedienungsanzeige im ersten Modus mit Joystick im Mittelpunkt (links) und Joystick unten rechts (rechts)

Wenn man den Joystick nun nochmal drückt, wird der zweite Modus aktiviert und der Roboter kann vorprogrammierten Pfaden folgen. Am Anfang ist keiner ausgewählt, aber mit dem Joystick nach oben oder unter oder rechts oder links kann einer von vier Pfaden ausgewählt werden:

- Pfad 1: Dreieck mit 50cm Seitenlänge
- Pfad 2: Quadrat mit 1m Seitenlänge
- Pfad 3: Kreis als Hexagon mit 20cm Seitenlänge
- Pfad 4: Acht bestehend aus 2-mal dem vorherigen Kreis

Neben der Geschwindigkeit v in cm/s, werden nun auch der ausgewählte Pfad unter „Path“ und aktuelle Pfadabschnitt unter „Point“ angezeigt.

Abbildung 5-4: Foto der Fernbedienungsanzeige im zweiten Modus, wenn kein Pfad ausgewählt ist (links) und bei ausgewähltem Pfad (rechts)

Schlussendlich können die Motoren des Roboters jederzeit abgeschaltet werden, indem man den Joystick Knopf 2 Sekunden gedrückt hält. Bei Verlieren der Verbindung zwischen Fernbedienung und Roboter, bleiben die Motoren eingeschaltet, aber der Roboter balanciert, ohne sich fortzubewegen, damit dieser nicht umfällt oder wegrollt.

Es wird empfohlen bei sichtbaren Veränderungen der Bodenneigung die Geschwindigkeit des Roboters zu reduzieren, wenn man diesen Untergrund noch nicht getestet hat. Die Batterielaufzeit wird bei kontinuierlichen Pfadabfahren und am Anfang voll aufgeladenem Akku auf ungefähr 8 Stunden geschätzt.

6 Schlussfolgerung

Schlussendlich werden die Ziele dieses Projektes noch einmal wiederholt und dann ermittelt, ob diese erreicht wurden und was noch verbessert oder hinzugefügt werden könnte. Am Ende wird dann noch die Theorie mit der Praxis verglichen.

Das Ziel dieses Projektes war es den Roboter so zu erweitern, dass er sich nicht nur balancieren kann, sondern auch wenden und nach vorne fahren kann. Des Weiteren sollte der Roboter Anweisungen kabellos von einer Fernbedienung erhalten können. Schlussendlich sollte der Roboter auch vorprogrammierte Pfade abfahren können.

Diese Ziele wurden weitestgehend erfüllt. Durch das Hinzufügen von Drehgebern und mehr Rechenleistung mit dem Arduino Mega konnte das erste Ziel mithilfe von kaskadierenden PID-Reglern umgesetzt werden. Eine kabellose Verbindung mit hoher Reichweite und niedriger Latenz konnte durch zwei nRF24L01+ umgesetzt werden. Die Fernbedienung selbst wurde innerhalb einer Klemmbaustein Hülle mit einem Bildschirm und Joystick realisiert für eine leichte Kontrolle des Roboters. Vorprogrammierte Pfade sind auf dem Kontroller gespeichert und diese können an den Roboter zum Abfahren übermittelt werden. Der Roboter besitzt dann die nötigen Regler, um diese Pfade abfahren zu können.

Allerdings kann der Roboter nicht sehr schnell fahren (12 cm/s), denn die meisten Böden sind nicht sehr uniform und führen durch abrupte Neigungen schnell dazu, dass der Roboter bei höheren Geschwindigkeiten kippen würde. Deswegen muss viel Spielraum gelassen werden, damit der Roboter sich wieder ausgleichen kann bei Neigungen. Auch kann der Roboter nur geringe Steigungen fahren. Beides ist hauptsächlich auf die Powerbank zurückzuführen, die neben den Motoren den größten Teil der 1,1kg des Gewichts des Roboters ausmacht. Die Motoren sind trotz des deswegen hoch ausgewählten Drehmomentes nicht stark genug, um das Gewicht des Roboters aufwärtszubewegen. Für eine bessere Fahrleistung könnte also auf eine kleinere leichtere Batterie gesetzt werden, um das Gewicht zu verringern. Dies würde es auch ermöglichen einen Motor mit höherer Drehzahl und niedrigerem Drehmoment zu benutzen. Natürlich hat die große Powerbank auch seine Vorteile, denn sie ermöglicht bei anfangs maximaler Kapazität ein Fahren für geschätzte 8 Stunden. Auch könnte eine Umsetzung mit einer LQR Reglung eventuell bessere Resultate erzielen. Zusätzlich könnte ein Ultraschallsensor zum Scannen des Bodenprofils es ermöglichen, dass der Roboter langsamer fährt, wenn eine Bodenwelle kommt. Außerdem könnte man auch andere Räder mit einem glatteren Profil oder größerem Radumfang nutzen, um die Kontaktfläche mit dem Boden zu vergrößern. Zuletzt könnte der Einsatz von Motoren mit einem präziseren Getriebe die Gleichgewichtsregelung verbessern.

Beim Abfahren eines Pfades fällt auch auf, dass die Positionsmessung, wenn der Roboter dreht, nicht immer komplett zuverlässig ist, da sie nur über die Drehgeber ermittelt wird, so dass es nicht möglich ist, zum Beispiel die seitliche Abweichung vom Pfad genau zu ermitteln. Dies könnte mit einer Kamera oder einem Infrarotsensor an der Unterseite des Roboters eventuell verbessert werden. Dies würde auch ermöglichen, dass der Roboter bei Pfaden

gleichzeitig nach vorn fahren und drehen könnte, ohne eine starke Abweichung vom Pfad zu riskieren.

Der Kontroller besitzt eine Klemmbaustein Hülle. Dies macht das Umbauen der Fernbedienung sehr leicht, ist aber nicht die ergonomischste Lösung, denn die Noppen der Bausteine sind auf Dauer nicht die bequemste Oberfläche zum Anfassen. Auch die blockhafte Form lässt sich noch verbessern. Durch die Verwendung eines Arduino Nano könnte man den Formfaktor des Kontrollers noch verkleinern und in Kombination mit einer ergonomischeren 3D-gedruckten Hülle insgesamt die Haptik erhöhen.

Auch die Konnektivität kann noch erweitert werden, so könnte mit dem Hinzufügen von einem Bluetooth oder Wlan Modul eine Smartphone App oder eine PC-Applikation mit dem Roboter verbunden werden, um diesen mit mehr Funktionen steuern zu können. Allerdings haben viele solcher Module schlechtere Latenzen als das benutzte nRF24L01+ Modul, das leider ein weniger kompatibleres Protokoll benutzt. Alternativ könnte man auch der Kontrollapplikation einen Code hinzufügen, welcher das Umwandeln des Protokolls des nRF24L01+ ermöglicht.

Auch könnte das Hinzufügen eines Ultraschallsensor, der nach vorne und hinten zeigt am Roboter eine Hinderniserkennung ermöglichen und damit auch Umgehung. Dies ist aber nicht unbedingt nötig, denn der Roboter kippt meistens nicht, wenn er in eine Wand fährt, sondern stützt sich an dieser ab, bis der Roboter die Fahrrichtung wechseln soll.

Wenn man die Praxis mit der Theorie vergleicht, fällt auf, dass die schlussendlich genutzten Regler Typen auch in der Theorie als mögliche Optionen bestimmt wurden. Allerdings ist der PI-Regler für das Gleichgewicht in der Realität nicht anwendbar, was wahrscheinlich darauf zurückzuführen ist, dass dort als Eingang die Drehgeschwindigkeit anstatt der Motorspannung berücksichtigt wurde.

7 Literaturverzeichnis

1. **SEGWAY INC.** Segway i2 SE. [Online] Juli 2022. https://eu-en.segway.com/products/segway-i2-se.

2. **Kapoor, Nishith.** *Two Wheeled Self Balancing (The Doodle Robot).* 2022.

3. **Robert, Berg.** *Positionskontrolle eines aufstehenden zwei-rad Roboters.* 2019.

4. **Adam, Abdoellah.** *Two-Wheeled Self-Balancing Robot Design and control based on the concept of an inverted pendulum.* 2019.

5. **Arduino S.r.l. 2021.** Arduino Store Arduino Mega 2560 Rev3. [Online] Juli 2022. https://store.arduino.cc/products/arduino-mega-2560-rev3.

6. **Atmel Corporation 2014.** Datenblatt Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V. [Online] Juli 2022. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.

7. **Adafruit Industries, LLC.** Adafruit Shop Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055 - STEMMA QT / Qwiic. [Online] Juli 2022. https://www.adafruit.com/product/4646.

8. **Bosch Sensortec GmbH.** Datenblatt BNO055. [Online] Juli 2022. https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf.

9. **Amazon.com Inc. 1996-2022.** Amazon.de DollaTek Mini Stepper Motor Drive Controller Board-Modul L298N verdoppeln H-Brücke DC-Stepper für Arduino Smart-Auto-Roboter. [Online] Juli 2022. https://www.amazon.de/DollaTek-Controller-Board-Modul-verdoppeln-Smart-Auto-Roboter/dp/B07DK6Q8F9/ref=pd_lpo_1?pd_rd_i=B07DK6Q8F9&psc=1.

10. **Handson Technology 2008-2021.** Handson Technology User Guide L298N Dual H-Bridge Motor Driver. [Online] http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf.

11. **Amazon.com Inc. 1996-2022.** Amazon.de DollaTek XL6009 DC-DC-Step-Up. [Online] Juli 2022. https://www.amazon.de/DollaTek-XL6009-DC-DC-Step-Up-Stromnehmen-3V-32V/dp/B07DJ4F1BN/ref=sr_1_1_sspa?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=18H568HT65ANF&keywords=XL6009&qid=1659812765&s=ce-de&sprefix=xl6009%2Celectronics%2C51&sr=1-1-spons&psc=1&.

12. —. Amazon.de Litionite Falcon Mini 12000mAh Power Bank. [Online] Juli 2022. https://www.amazon.de/Falcon-Mini-Litionite/dp/B07H98DCNT?ref_=ast_sto_dp.

13. **HowToMechatronics.com 2022.** HowToMechatronics nRF24L01 – How It Works, Arduino Interface, Circuits, Codes. [Online] Juli 2022. https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/.

14. **LastMinuteEngineers.com 2022.** LastMinuteEngineers How nRF24L01+ Wireless Module Works & Interface with Arduino. [Online] Juli 2022. https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/.

15. **Amazon.com Inc. 1996-2022.** Amazon.de DC 12 V DIY Encoder Getriebemotor, Micro Drehzahlreduzierung Motor Vollmetallgetriebe, mit Halterung 65mm Magnetgetriebe Motor Rad Kit, für Smart Auto Roboter Modell DIY(200RPM). [Online] Juli 2022. https://www.amazon.de/Getriebemotor-Halterung-Magnetgetriebe-Drehzahlreduzierung-Vollmetallgetriebe/dp/B07WRYQZNR/ref=sr_1_12?adgrpid=114802779577&gclid=Cj0KCQjw_4-SBhCgARIsAAlegrUUENGq79zsDnr8NJdldHrrMpMQpCHClQ8HSZLdjefluITEgHRyCqEaAlQxEALw_wcB&hvadid=49.

16. **Arduino S.r.l 2021.** Arduino Store Arduino Uno Rev3. [Online] Juli 2022. https://store.arduino.cc/products/arduino-uno-rev3.

17. **Atmel Corporation 2015.** Datenblatt ATmega328P. [Online] Juli 2022. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

18. **Amazon.com Inc. 1996-2022.** Amazon.de AZDelivery Joystick Modul KY-023 kompatibel mit Arduino und Raspberry Pi inklusive E-Book! [Online] Juli 2022. https://www.amazon.de/AZDelivery-Joystick-KY-023-Parent/dp/B07V3HQSVY?th=1.

19. **LastMinuteEngineers.com 2022.** LastMinuteEngineers How 2-Axis Joystick Works & Interface with Arduino + Processing. [Online] Juli 2022. https://lastminuteengineers.com/joystick-interfacing-arduino-processing/.

20. **Adafruit Industries, LLC.** Adafruit Shop Monochrome 0.96" 128x64 OLED Graphic Display - STEMMA QT. [Online] Juli 2022. https://www.adafruit.com/product/326.

21. —. Adafruit Learn Monochrome OLED Breakouts Overview. [Online] Juli 2022. https://learn.adafruit.com/monochrome-oled-breakouts.

22. **Amazon.com Inc. 1996-2022.** Amazon.de Yantan 2X Batteriehalter Batteriefach Fuer 6 Stueck AA Batterien mit Anschlussdraht. [Online] Juli 2022. https://www.amazon.de/Yantan-Batteriehalter-Batteriefach-Batterien-Anschlussdraht/dp/B094HZ8QLR/ref=d_pd_sbs_sccl_2_25/257-3507962-9113411?pd_rd_w=JIxBz&content-id=amzn1.sym.dd12e595-59ba-462a-a82b-6daceb70766d&pf_rd_p=dd12e595-59ba-462a-a82b-6daceb70766d.

23. **Arduino S.r.l 2021.** Arduino Overview of the Arduino IDE 1. [Online] Juli 2022. https://docs.arduino.cc/software/ide-v1/tutorials/Environment#file.

24. —. Arduino Downloads. [Online] Juli 2022. https://www.arduino.cc/en/software.

25. **eVision Systems GmbH 2022.** eVision Systems I2C vs SPI Protokoll: Unterschiede und Ähnlichkeiten. [Online] Juli 2022. https://evision-webshop.de/Total-Phase-Knowledge-Base/Promira/I2C-vs-SPI-Protokoll-Analysatoren-Unterschiede-und-

Aehnlichkeiten#:~:text=I2C%20(Inter%2DIntegrated%20Circuit),die%20serielle%20Taktleitung%20(SCL)..

26. **SparkFun Electronics.** Sparkfun I2C. [Online] Juli 2022. https://learn.sparkfun.com/tutorials/i2c/all.

27. **Unbehauen, Heinz.** *Regelungstechnik I 15. Auflage.* s.l. : Vieweg+Teubner Verlag, 2008.

8 Anhang

Im Anhang befinden sich beide Dateien der „SelfRobo“ Bibliothek

8.1 SelfRobo.h

```
#ifndef SelfRobo_h
#define SelfRobo_h

#include "Arduino.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <PID_v1.h>



class SelfRobo {

  public:
    SelfRobo(int leftForward, int leftBackward, int leftSpeedPin,
        int rightForward, int rightBackward, int rightSpeedPin, double Kp, double
Ki, double kd);

    // Behavior
    bool Update();
    void Initialize(int sampleTime, int outputLimit1, int outputLimit2);
    void ComputeBalance();
    void ComputeSpeed(float rpm);
        bool Drive();
        bool SetDrive(int leftMotorSpeed, int rightMotorSpeed);

    // PID
    void SetTunings(double Kp, double Ki, double Kd);
        void UpdateSetpoint(double newSetpoint);

    // IMU
    volatile double rollAngle;
    volatile double yawAngle;
    volatile double pitchAngle;
    volatile double pitchSpeed;
    volatile double acc;

    // Get values printed
    float GetOutput();
    float GetInput();
    float GetSetpoint();



  private:

    // IMU
    Adafruit_BNO055 _bno;
    void initializeIMU();
    void readIMU();
```

```
    // Motors
    int _leftForward;
    int _leftBackward;
    int _leftSpeedPin;
    volatile int _leftSpeed = 0;
    int _rightForward;
    int _rightBackward;
    int _rightSpeedPin ;
    volatile int _rightSpeed = 0;
    void initializeMotors();
    PID selfRoboController;
    void setMotors(int leftMotorSpeed, int rightMotorSpeed);



    // PID

    volatile int motorPower;

    double _Kp, _Ki, _Kd;
    volatile double Setpoint = 0, Input, Output;
    void initializePID(int sampleTime, int outputLimit1, int outputLimit2);



};
#endif
```

8.2 SelfRobo.cpp

```
#include "Arduino.h"
#include "SelfRobo.h"
#include <PID_v1.h>


// constructor is made of self robo.
SelfRobo::SelfRobo(int leftForward, int leftBackward, int leftSpeedPin,
         int rightForward, int rightBackward, int rightSpeedPin, double Kp, double
Ki, double Kd):selfRoboController(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT)
{
    _leftForward   = leftForward;
    _leftBackward  = leftBackward;
    _leftSpeedPin  = leftSpeedPin;
    _rightForward  = rightForward;
    _rightBackward = rightBackward;
    _rightSpeedPin = rightSpeedPin;
       _Kp = Kp;
       _Ki = Ki;
       _Kd = Kd;
}


//Initialize() function is used to make sure that the robot sensors and motors are
ready for action.
void SelfRobo::Initialize(int sampleTime, int outputLimit1, int outputLimit2) {
  initializeIMU();
```

```
  initializeMotors();
  initializePID(sampleTime, outputLimit1, outputLimit2);
}

// Update() function is used to call certain behaviors during the main loop i.e.
read the IMU and Set motor power.
bool SelfRobo::Update() {
      readIMU();
}

// Set motor power to values from PID
bool SelfRobo::Drive() {
      setMotors(motorPower, motorPower);
}

// Set motor power to input values
bool SelfRobo::SetDrive(int leftMotorSpeed, int rightMotorSpeed) {
      setMotors(leftMotorSpeed, rightMotorSpeed);
}

// IMU BNO055 detection
void SelfRobo::initializeIMU() {
  _bno = Adafruit_BNO055(55);
  if (!_bno.begin())
  {
    Serial.print("No BNO055 detected");
    while (1);
  }
  delay(1000);
  _bno.setExtCrystalUse(true);
}
void SelfRobo::readIMU() {
  sensors_event_t event;
  _bno.getEvent(&event, Adafruit_BNO055::VECTOR_EULER);
  // euler angles are used in the program as they are stable and filter already
  //in BNO055 as compared to MPU6050 which has all the raw data.
  yawAngle = -event.orientation.x;
  //rollAngle = event.orientation.y;
  pitchAngle = event.orientation.z;

  //_bno.getEvent(&event, Adafruit_BNO055::VECTOR_GYROSCOPE);
  //pitchSpeed = event.gyro.x;

  //_bno.getEvent(&event, Adafruit_BNO055::VECTOR_LINEARACCEL);
  //acc = event.acceleration.y;
}


// Motors Pin numbers
void SelfRobo::initializeMotors() {
  pinMode(_leftForward  , OUTPUT);
  pinMode(_leftBackward , OUTPUT);
  pinMode(_leftSpeedPin , OUTPUT);
  pinMode(_rightForward , OUTPUT);
  pinMode(_rightBackward, OUTPUT);
  pinMode(_rightSpeedPin, OUTPUT);
}
void SelfRobo::setMotors(int leftMotorSpeed, int rightMotorSpeed) {
      if (rightMotorSpeed < 0 && leftMotorSpeed < 0) {
```

```
    digitalWrite(_rightBackward, LOW);
    digitalWrite(_rightForward, HIGH);
    analogWrite(_rightSpeedPin, abs(rightMotorSpeed));

    digitalWrite(_leftBackward, LOW);
    digitalWrite(_leftForward, HIGH);
    analogWrite(_leftSpeedPin, abs(leftMotorSpeed));
  }
  else if (rightMotorSpeed >= 0 && leftMotorSpeed >= 0) {

    digitalWrite(_rightBackward, HIGH);
    digitalWrite(_rightForward, LOW);
    analogWrite(_rightSpeedPin, rightMotorSpeed);

    digitalWrite(_leftBackward, HIGH);
    digitalWrite(_leftForward, LOW);
    analogWrite(_leftSpeedPin, leftMotorSpeed);
  }

}



// PID
void SelfRobo::initializePID(int sampleTime, int outputLimit1, int outputLimit2) {
    selfRoboController.SetMode(AUTOMATIC);
    selfRoboController.SetSampleTime(sampleTime);
    selfRoboController.SetOutputLimits(outputLimit1, outputLimit2);
}


void SelfRobo::ComputeBalance() {

  Input = pitchAngle;
  //Input = map(zTilt, -90.0, 90.0, -90.0, 90.0);

  selfRoboController.Compute();

  // Motor control
  motorPower = Output;

  // give up if there is no chance of success
  if (Input < -40 || Input > 40) motorPower = 0;
}

void SelfRobo::ComputeSpeed(float rpm) {
  Input = rpm;
  selfRoboController.Compute();
  motorPower = Output;
}

void SelfRobo::SetTunings(double Kp, double Ki, double Kd) {

  selfRoboController.SetTunings(Kp, Ki, Kd);
}

void SelfRobo::UpdateSetpoint(double newSetpoint) {
   Setpoint = newSetpoint;
```

```
}


float SelfRobo::GetOutput() {
  return Output;
}

float SelfRobo::GetInput() {
  return Input;
}

float SelfRobo::GetSetpoint() {
  return Setpoint;
}
```