
Aufbau und Regelung eines Ballbots

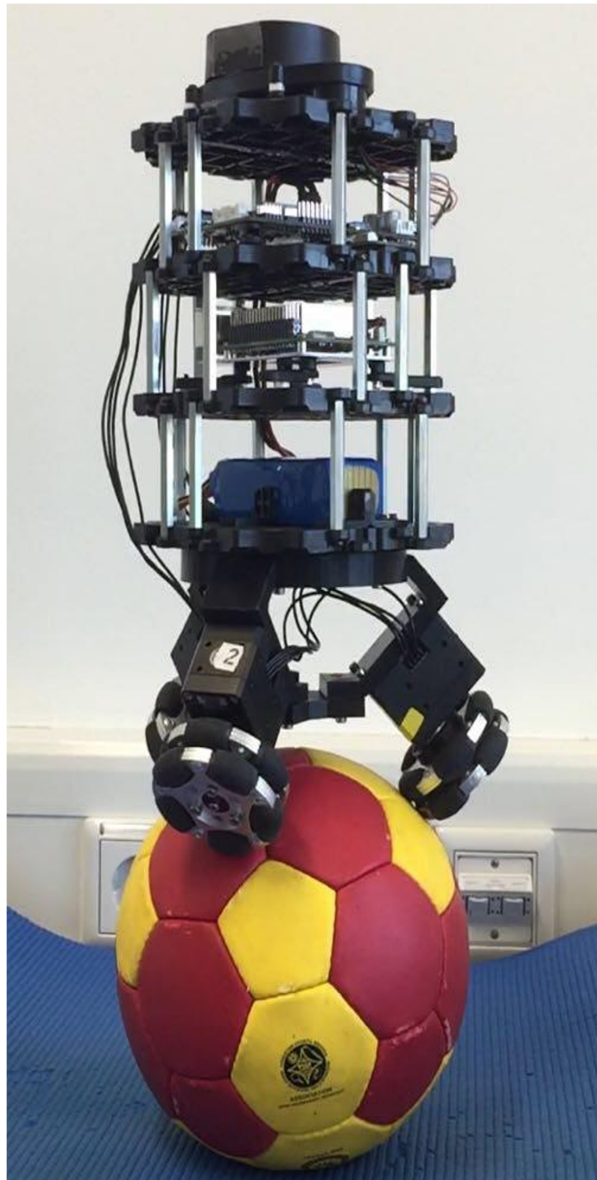
Florian Müller

Markus Lamprecht

Michael Suffel

Projektseminar – 16. Februar 2018

Betreuer: Dr.-Ing. Eric Lenz



TECHNISCHE
UNIVERSITÄT
DARMSTADT

REGELUNGSTECHNIK *rtm*
UND MECHATRONIK

Aufgabenstellung

Im Rahmen dieses Projektseminars soll basierend auf am Fachgebiet SIM (Fachbereich Informatik) vorhandenen Komponenten ein Ballbot-Roboter nach dem in [1] beschriebenen Beispiel aufgebaut und in Betrieb genommen werden. Im Einzelnen sind dabei die Arbeitspakete:

- Konstruktion der notwendigen mechanischen Baugruppen
- Aufbau des Roboters
- Inbetriebnahme von Aktorik und Sensorik über On-Board-Rechner
- Identifikation der Systemparameter sowie
- Modellierung und Regelung

zu bearbeiten.

Ziel der Arbeit ist es, einen funktionsfähigen Roboter zu haben, auf dem ein einfaches Balancieren auf der Stelle implementiert ist. Dabei kann die Regelung durchaus von bestehenden Arbeiten übernommen werden. Unabhängig von den letztendlich implementierten Regelkonzepten wird jedoch eine ausführliche Recherche zu den Regelkonzepten aus der Literatur gefordert.

Beginn: 16. Oktober 2017

Ende: 16. Februar 2018

Präsentation: 09. April 2018

Prof. Dr.-Ing. Ulrich Konigorski

Dr.-Ing. Eric Lenz

Technische Universität Darmstadt
Institut für Automatisierungstechnik und Mechatronik
Fachgebiet Regelungstechnik und Mechatronik
Prof. Dr.-Ing. Ulrich Konigorski

Landgraf-Georg-Straße 4
64283 Darmstadt
Telefon 06151/16-4167
www.rtm.tu-darmstadt.de





Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 16. Februar 2018

Florian Müller

Markus Lamprecht

Michael Suffel

Kurzfassung

Inhalt dieser Arbeit ist der mechanische Aufbau, die Modellierung, der Reglerentwurf und die Simulation eines Ballbots. Aufgebaut wurde der Ballbot mittels Komponenten eines Turtlebot3 Roboters, der um einen selbst entwickelten Unterbau ergänzt wurde. Anschließend erfolgte die mathematische Modellbildung und eine linear-quadratische Reglerauslegung (LQR) in MATLAB/Simulink. Bei der Modellbildung wurde das reale dreidimensionale System durch zwei unabhängige planare Ebenen (xz und yz) approximiert. Der entworfene Regler wurde schließlich mit dem 3D-Simulator Gazebo validiert.

Es konnte gezeigt werden, dass sich das System mittels der entworfenen Regelung in der xz - und yz -Ebene stabilisieren lässt. Essentiell für das Balancieren des Ballbots ist eine optimale Abstimmung des Roboters auf den Ball. Es sollte darauf geachtet werden dass:

- Der Reibkoeffizient im Kontaktpunkt zwischen Boden und Ball, sowie zwischen Ball und omnidirektionalem Rad muss möglichst groß sein.
- Die Motoren eine hohe maximale Drehzahl ($> 4.5 \text{ rad/s}$) aufweisen.
- Die Abtastrate des Systems möglichst hoch ($> 130 \text{ Hz}$) ist.

Weiterhin kann das System durch eine genauere Modellierung (3D-Modellierung) sowie der Berücksichtigung der Ball-Odometrie stabilisiert werden.

Schlüsselwörter: Ballbot, Omnidirektionale Räder, Gazebo, Matlab, Arduino

Abstract

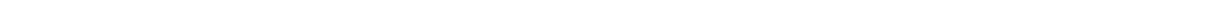
Goal of this research project was the construction, modelling, controller implementation and the simulation of a Ballbot. This Ballbot was built using components of a Turtlebot3 Robot. Additionally a specific designed structure was created to attach the omnidirectional wheels to the ballbot. After that MATLAB/Simulink was used to design a linear-quadratic regulator(LQR). The modelling was done by approximating the real three dimensional System with two planes (xz and yz). At the end the Ballbot was validated with the 3D-Simulator Gazebo.

This project revealed that the system could be stabilized around the xz- and the yz-plane. Essential for the balancing behaviour of the ballbot was an optimal adaption of the Robot to the ball. Thereby it should be considered:

- The friction coefficient between the ball and the surface and between the ball and the omnidirectional wheel should be as high as possible.
- The maximum revolution speed of the motors should be as high as possible ($> 4.5 \text{ rad/s}$).
- The sampling frequency of the system should be higher than 130 Hz.

In order to improve the stabilization of the system the 2D-modelling can be replaced by a more exact 3D-modelling. Additionally the Ball's odometry can be included in the design of the controller.

Keywords: Ballbot, Omnidirectional Wheels, Gazebo, Matlab, Arduino



Inhaltsverzeichnis

Symbole und Abkürzungen	x
1. Einleitung	1
2. Konstruktion	3
2.1. Konzeptionierung	3
2.2. Verwendete Hardware	5
2.2.1. OpenCR-Board	5
2.2.2. Sensorik	6
2.2.3. Motoren	7
2.3. Ballbot-Design	8
2.3.1. Oberbau	10
2.3.2. Unterbau	10
2.3.3. Wahl des Balls	12
2.4. Konstruktion mittels SolidEdge	12
2.4.1. SolidEdge - Eine kurze Einführung	12
2.4.2. SolidEdge - Umsetzung	13
2.5. Fertigung der Komponenten	15
3. Modellbildung und Regelung	17
3.1. Model	17
3.2. Minimalkoordinaten	18
3.3. Energien	18
3.3.1. Externe Kräfte und Drehmomente	19
3.4. Bewegungsgleichungen	20
3.5. Zustandsraumdarstellung	20
3.6. Reglerentwurf	22
3.7. Simulation des Modells	23
3.7.1. Implementierung ideales Modell	24
3.7.2. Reale Gegebenheiten	24
4. Simulation	27
4.1. 3D Simulatoren	27
4.2. Simulation von omnidirektionalen Rädern	27

4.3. Simulations Aufbau	30
4.3.1. Kommunikation zwischen ROS und Gazebo	30
4.3.2. Plugins der Simulation	31
4.3.3. Simulierte Dynamik-Eigenschaften	31
4.3.4. Visualisierungsprogramme	32
5. Implementierung	37
5.1. Entwicklungsumgebung	37
5.2. Hauptprogramm	38
5.2.1. Initialisierung Komponenten	38
5.2.2. Einlesen der Sensordaten	39
5.2.3. Verarbeitung Sensordaten	40
5.2.4. Ausgabe Drehmomente	41
5.3. Auswertung	41
6. Zusammenfassung	45
A. Parameterliste	47
B. Bestands-Liste	49
C. ROS-Graph Simulation	51
D. Simulink Simulationsaufbau	53
Literaturverzeichnis	55



Symbole und Abkürzungen

Lateinische Symbole und Formelzeichen

Symbol	Beschreibung	Einheit
A	Systemmatrix linearisierten Modell	
A*	Systemmatrix reduziertes Modell	
B	Eingangsmatrix linearisierten Modell	
B*	Eingangsmatrix reduziertes Modell	
C	Ausgangsmatrix linearisierten Modell	
C*	Ausgangsmatrix reduziertes Modell	
C_x	Corioliskraft-Matrix der <i>yz</i> -Ebene	
D	Durchgangsmatrix linearisierten Modell	
D*	Durchgangsmatrix reduziertes Modell	
f_{NP,yz1}	Kraftvektor des virtuellen Drehmomentes in <i>yz</i> -Ebene	Nm
f_{NP,yz2}	Kraftvektor des Gegendrehmoments in der <i>yz</i> -Ebene	Nm
f_{NP,yz}	Summe der einzelnen Kraftvektoren in der <i>yz</i> -Ebene	Nm
G_x	Gravitationskraft-Matrix der <i>yz</i> -Ebene	N
I_{B,yz}	Massenträgheitsmoment Körper in der <i>yz</i> -Ebene	kg · m ²
I_{B,xz}	Massenträgheitsmoment Körper in der <i>xz</i> -Ebene	kg · m ²
I_{B,xy}	Massenträgheitsmoment Körper in der <i>xy</i> -Ebene	kg · m ²
I_S	Massenträgheitsmoment Ball	kg · m ²
I_{W,yz}	Massenträgheitsmoment virtuelles Rad in der <i>yz</i> -Ebene	kg · m ²
I_{W,xz}	Massenträgheitsmoment virtuelles Rad in der <i>xz</i> -Ebene	kg · m ²
I_{W,xy}	Massenträgheitsmoment virtuelles Rad in der <i>xy</i> -Ebene	kg · m ²
J	Gütemaß für den linear-quadratischen Regelentwurf	
k_{motor}	Drehmomentkonstante	Nm/A
k_{unit}	Umrechnungskonstante Drehmoment-Unit	Nm/Unit
K	Matrix mit den Verstärkungsfaktoren des Regler	
l	Pendellänge	m
l_{pruef}	Länge Hebelarm des Prüfhammers	m
L	LANGRANGEsche Funktion	J
M_x	Massenträgheitsmatrix der <i>yz</i> -Ebene	
m_B	Masse Körper des Roboters	kg
m_S	Masse Ball	kg
m_W	Masse virtuelles Rad	kg
r_B	Radius Körper des Roboters	m
r_S	Radius Ball	m
r_W	Radius virtuelles Rad	m
R	Gewichtungsmatrix der Stellgrößen	

Symbol	Beschreibung	Einheit
$\mathbf{q}_{yz,xz,xy}$	minimaler Koordinatenvektor der jeweiligen Ebene	
\mathbf{Q}	Gewichtungsmatrix der Zustände	
$T_{1,2,3}$	reale Drehmomente der einzelnen Motoren	Nm
$T_{x,y,z}$	virtuelle Drehmomente	Nm
$T_{B,yz}$	kinetische Energie Körper des Roboters in der yz -Ebene	J
$T_{S,yz}$	kinetische Energie Ball in der yz -Ebene	J
$T_{W,yz}$	kinetische Energie virtuelles Rad in der yz -Ebene	J
\mathbf{u}	Stellgrößenvektor	
$V_{B,yz}$	potentielle Energie Körper des Roboters in der yz -Ebene	J
$V_{S,yz}$	potentielle Energie Ball in der yz -Ebene	J
$V_{W,yz}$	potentielle Energie virtuelles Rad in der yz -Ebene	J
\mathbf{x}	Zustandsvektor ursprüngliches System	
\mathbf{x}^*	Zustandsvektor reduziertes System	
$\dot{\mathbf{x}}$	zeitliche Ableitung des Zustandsvektors des linearisierten Systems	
$\dot{\mathbf{x}}_{nl}$	zeitliche Ableitung des Zustandsvektors des ursprünglichen nichtlinearen Systems	
$\dot{\mathbf{x}}^*$	zeitliche Ableitung des Zustandsvektors des reduzierten Systems	

Griechische Symbole und Formelzeichen

Symbol	Beschreibung	Einheit
$\vartheta_{x,y,z}$	Orientierung des Roboters	rad
$\varphi_{x,y,z}$	Orientierung des Balles	rad
$\psi_{x,y,z}$	Orientierung des virtuellen Rades	rad
λ	Eigenwerte ursprüngliches System	
λ^*	Eigenwerte reduziertes System	

Abkürzungen

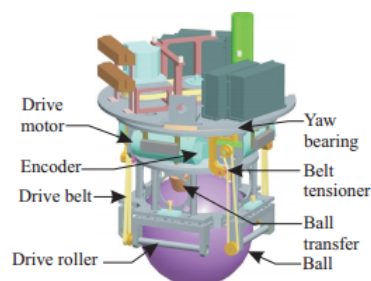
Kürzel vollständige Bezeichnung

CAD	Computer Aided Design
CMU	Carnegie Mellon University
EVA	Einlesen-Verarbeiten-Ausgeben
ETHZ	Eidgenössische Technische Hochschule Zürich
IMU	inertiale Messeinheit
LQR	linear-quadratische Regelung
ODE	Open Dynamics Engine
PLA	Polylacide
ROS	Robot Operating System

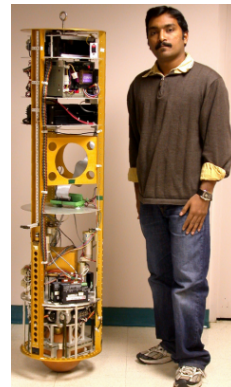
1 Einleitung

Seit dem Star Wars Film "Das Erwachen der Macht", gibt es neben r2d2 und c3po einen neuen Roboter, den BB8. Inspiriert ist dieser Roboter an dem sogenannten Ballbot. Ein Ballbot ist ein Roboter, der auf einem Ball balanciert. Wie im Film zu sehen, fasziniert der BB8 mit seinen einzigartigen Bewegungen, denn er kann sich nicht nur in jede Richtung fortbewegen, sondern auch um sich selber auf dem Ball drehen.

Der erste Ballbot wurde von dem in Computer-Mäusen angewendetem Maus-Roller Prinzip inspiriert. Er wurde 2006 an der Carnegie Mellon Universität (CMU) in den Vereinigten Staaten entwickelt. Abbildung 1.1 (a) zeigt, dass der Ballbot von vier Laufrollen mittels vier DC Motoren angetrieben wird. Seit 2006 wurde dieser Ballbot weiterentwickelt [2] und schließlich mit einem fünften DC Motor ausgestattet, sodass er sich auch um seine eigene Achse drehen kann.



(a) Skizze des ersten Ballbots mit vier Laufrollen.

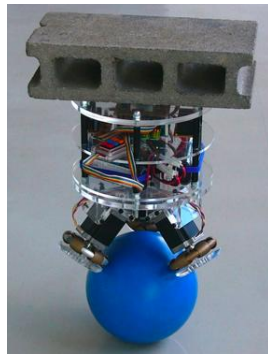


(b) Ballbot der CMU entwickelt 2006.

Abbildung 1.1.: Aufbau Skizze und menschengroßer erster Ballbot [2].

Dieser erste Ballbot benötigt mit vier bzw. fünf Motoren sehr viele Ressourcen. 2009 stellte M. Kumaga [3] einen Ballbot, siehe Abbildung 1.2 (a), der durch drei omnidirektionale Räder angetrieben wird, vor. Dieser Ballbot kann sich ebenfalls um seine eigene Achse drehen und sogar eine Last von mindestens 10 kg tragen.

2010 stellte die Eidgenössische Technische Hochschule Zürich (ETHZ) den Rezero vor [1], siehe Abbildung 1.2 (b). Dies ist ebenfalls ein Ballbot der mittels drei omnidirektionalen Rädern angetrieben wird. Er hat eine hohe dynamische Performanz und kann somit einen Neigungswinkel von bis zu 20° ausregeln, sowie eine maximale lineare Geschwindigkeit von 2 m/s erreichen.



(a) Ballbot der Tohoku Gakuin Universität (2009) [3].



(b) Ballbot Rezero der ETHZ (2010) [1].

Abbildung 1.2.: Omnidirektionale Ballbot's der TGU und der ETHZ.

Inspiziert durch diese erfolgreichen Beispiele balancierender Roboter, soll in diesem Projektseminar ein Ballbot aufgebaut werden. Primäres Ziel war das Balancieren des Roboters auf einem Ball zu erreichen.

Die Umsetzung umfasste dabei den mechanischen Aufbau (Kap. 2), die mathematische Modellierung (Kap. 3), die Simulation (Kap. 4) und die Implementierung (Kap. 5).

2 Konstruktion

2.1 Konzeptionierung

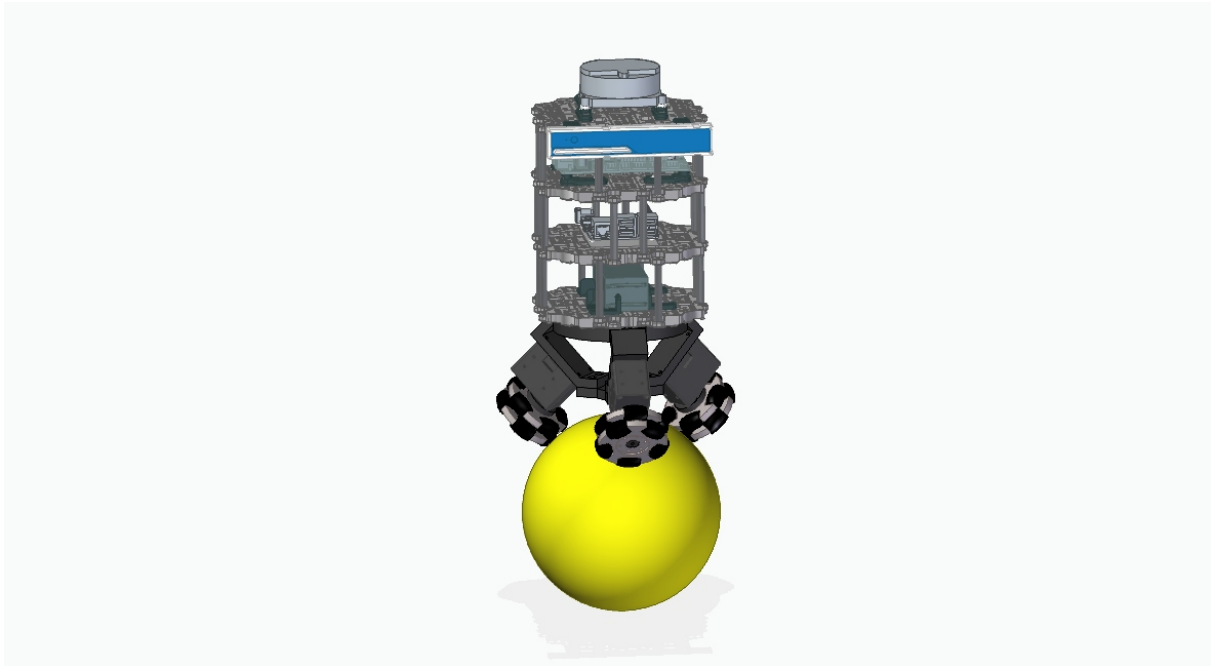


Abbildung 2.1.: Oberbau, bestehend aus vier Etagen

Für den Entwurf eines auf einem Ball balancierenden Roboters kann das System zur Vereinfachung als ein inverses Pendel betrachtet werden, dessen Drehachse auf einer beweglichen Plattform gelagert ist. Dies ist in Abbildung 2.2 beispielhaft dargestellt. Unter dieser Annahme können erste Abschätzungen über das dynamische Verhalten getroffen werden. Aus der Anschauung heraus kann direkt gefolgert werden, dass der Pendelarm mit der Länge l einen Einfluss auf die Pendeldynamik des Systems haben muss. Leitet man nun wie in [4], unter Verwendung der Momentenbilanz das nichtlineare Differenzialgleichungssystem zweiter Ordnung des Auslenkwinkels ϑ sowie der Schlittenstrecke x_w her, so erhält man die Gleichungen:

$$\ddot{\vartheta} = \frac{l \cdot m}{I} \cdot [-\cos \vartheta \cdot \ddot{x}_w + g \cdot \sin \vartheta], \quad (2.1)$$

$$\ddot{x}_w = \frac{F_{motor} - F_{Reib}}{M + m} + \frac{m \cdot l}{M + m} \dot{\vartheta}^2 \cdot \sin \vartheta - \ddot{\vartheta} \cdot \cos(\vartheta). \quad (2.2)$$

Setzt man nun das Massenträgheitsmoment des inversen Pendels

$$I = \frac{4}{3}m \cdot l^2, \quad (2.3)$$

in Gl. 2.1 ein, so ergibt sich:

$$\ddot{\vartheta} = \frac{3}{4 \cdot l} \cdot [-\cos \vartheta \cdot \ddot{x}_w + g \cdot \sin \vartheta]. \quad (2.4)$$

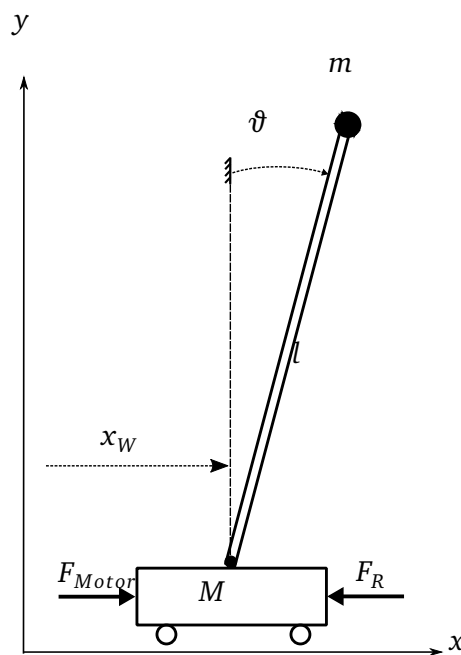


Abbildung 2.2.: Darstellung eines Ballbots als inverses Pendel

Anhand der Gleichung (2.4) lässt sich wie bereits vermutet eine Abhängigkeit der Winkelbeschleunigung $\ddot{\vartheta}$ von der Hebelarmlänge l erkennen. Da sich die Hebelarmlänge jedoch aus der Distanz zwischen Drehachse und Massenschwerpunkt zusammensetzt, bestimmt die Verteilung der Ballbotmasse die Hebelarmlänge des inversen Pendels. Durch das Vergrößern der Hebelarmlänge l kann so die Winkelbeschleunigung $\ddot{\vartheta}$ für kleine Winkelauslenkungen $\Delta\vartheta$ verringert werden. Dadurch ist das System leichter zu stabilisieren. Bei Betrachtung der Gleichung (2.2) lässt sich jedoch erkennen, dass mit einer Verlängerung des Hebelarms l eine größere Beschleunigung \ddot{x}_w des Schlittens einhergehen muss, die durch entsprechend starke Drehmomente realisiert werden kann.

Wendet man dieses Prinzip eines großen Hebelarms auf den Ballbot an, so stellt sich jedoch heraus, dass es nur bedingt anwendbar ist. Ursache hierfür ist die Schnittstelle zwischen omnidirektionalen Rädern und Ball, da hier aufgrund des Reibkoeffizienten nur ein begrenztes

Drehmoment übertragen werden kann. Der Reibkoeffizient begrenzt somit die gewünschte Sollbeschleunigung des Balles.

Da eine optimale Auslegung der Pendelarmlänge rechnerisch nur schwer zu bestimmen ist, wäre es wünschenswert, die Hebelarmlänge durch eine flexible Hardware verändern zu können. Durch das Hinzufügen bzw. Entfernen von weiteren Ebenen im Aufbau des Ballbots kann der Massenschwerpunkt und somit die Pendelarmlänge des Ballbots variabel eingestellt werden. So kann die optimale Konfiguration sehr einfach experimentell bestimmt werden.

Als Hardware für den Ballbot hat man sich daher für das TurtleBot3-Paket in der Bürger-Variante entschieden. Auf diese soll im Kapitel 2.2 tiefer eingegangen werden. Bei der Wahl des Balles sowie der Antriebsanordnung hat man sich die Konzepte bereits funktionierender Ballbots als Grundlage hergenommen. Die folgenden Kapitel gehen dabei näher auf die einzelnen Konstruktionsschritte ein.

2.2 Verwendete Hardware

Beim Entwurf des Ballbots wurde, wie bereits erwähnt, auf die leistungsfähige Hardware eines TurtleBot3 in der „Bürger“-Variante zurück gegriffen. Dieses Paket umfasst neben einem robusten mechanischen Aufbau auch alle grundlegenden elektronischen Komponenten wie Sensoren, Mikrocontrollern und Antriebseinheiten, die für die Entwicklung eines auf einem Ball balancierenden Roboters benötigt werden. Die wichtigsten Komponenten werden in den nachfolgenden Unterkapitel näher erläutert.

2.2.1 OpenCR-Board

Das im Projekt verwendete OpenCR-Board (in der Version 1.0), wie in Abbildung 2.3 dargestellt, ist ein Mikrocontroller, der für Roboterprojekte in der Größe des TurtleBot3-Pakets perfekt geeignet ist. Das gesamte Board, von der Hardware bis zur Software, ist durch Open-Source-Lizenzen frei verfügbar. Es bietet mit insgesamt 10 verschiedenen Kommunikationsstellen und einer integrierten inertialen Messeinheit (IMU)¹ bei der Entwicklung von Robotersystemen. Darüber hinaus ist das OpenCR-Board mit einem leistungsstarken ARM Cortex-M7 Prozessor ausgestattet.

¹ Die verwendete IMU ist die MPU-9250 siehe <https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/>.

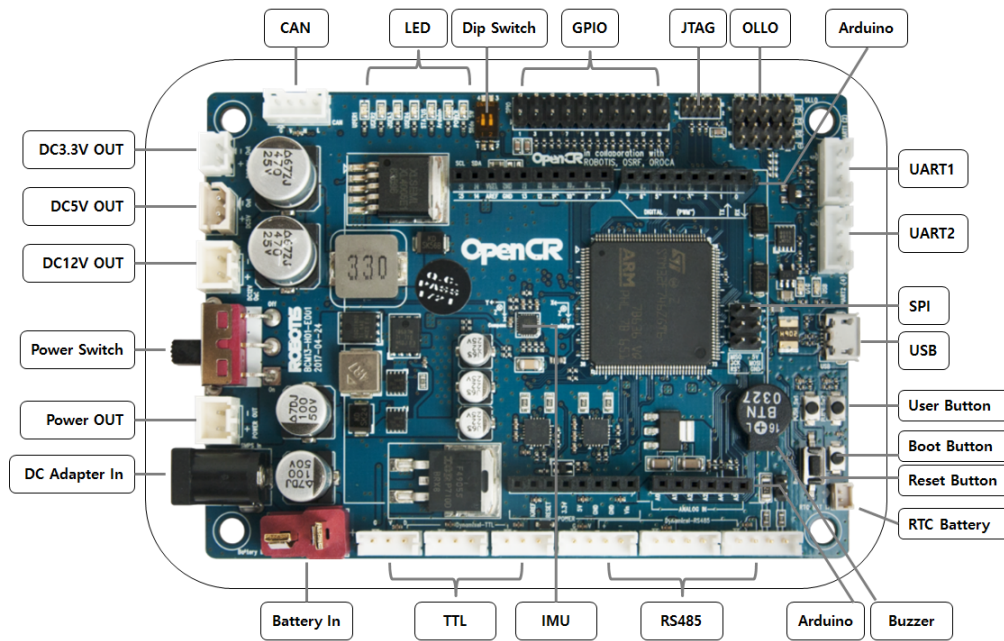


Abbildung 2.3.: Übersicht der Schnittstellen des OpenCR-Boards [5]

2.2.2 Sensorik

Am Ballbot wurden mehrere Sensoren verbaut, die für verschiedene Aufgabenbereiche benötigt werden. Tabelle 2.1 zeigt dabei eine Übersicht der Sensoren, die zur Stabilisierung bzw. zur Lokalisation verwendet werden.

Tabelle 2.1.: Sensoren, die im Ballbot verbaut wurden

Übersicht Sensoren			
Kategorie	Stabilisierung	Lokalisierung/Interaktion	
Bezeichnung	IMU MPU-9250	Kamera Intel Realsense R200	Laser Distance Sensor LDS-01
Taktfrequenz:	4Hz - 8kHz	2.5GHz	1,8 kHz
Features:	Gyroskop Accelerometer Magnetometer	IR Laser Projector System Full HD RGB Color Stream Onboard Imaging ASIC	DA: 120mm - 3500mm Scan Rate: 300±10 rpm Angular Resolution: 1°



Abbildung 2.4.: Motorenspezifikationen

Im TurtleBot3-Paket enthalten sind weiterhin auch die Motoren, welche für den Ballbotaufbau verwendet werden. Es handelt sich dabei um Dynamixel Servomotoren der Baureihe XM430-W350-T. Die Motoren haben sich als sehr robust erwiesen und können durch eine hohe Übersetzung von $i = 1 : 353.5$ ein variables ausgangsseitiges Drehmoment von bis zu 3.8 Nm bei 11.1 V [6] erzeugen. Dies ist für die gegebenen Systemkonfiguration mehr als ausreichend. Weiterhin werden von Dynamixel bereits verschiedene Betriebsarten bereitgestellt, in denen die Motoren betrieben werden können. So bieten die Motoren neben einer Positionsregelung, einer Geschwindigkeitsregelung auch eine Stromregelung. Durch die lineare Abhängigkeit zwischen Drehmoment und Strom kann somit das abtriebseitige Drehmoment geregelt werden.

Vom OpenCR-Board werden mittels dem RS485-Protokoll sogenannte Units übertragen. Die Units werden dann intern im Motor in einen realen Sollstrom umgewandelt. Der Strom kann über die Formel

$$I = k_{unit} \cdot Units, \quad \text{mit} \quad k_{unit} = 2,69 \cdot 10^{-3} \frac{A}{Unit} \quad (2.5)$$

berechnet werden. Anschließend kann mit Hilfe der Drehmoment(Nm)-Strom(A)-Kennlinie aus dem Datenblatt [6] die spezifische Motorkonstante zu $k_{motor} = 1,63 \text{ Nm/A}$ bestimmt werden. bestimmt. Dadurch kann das vom Motor erzeugte Drehmoment aus den Units mittels der Formel

$$M = k_{motor} \cdot I \quad (2.6)$$

$$= k_{motor} \cdot k_{unit} \cdot Units \quad (2.7)$$

berechnet werden. Insgesamt ergibt sich folgender Gesamtumrechnungsfaktor

$$k = k_{motor} \cdot k_{unit} = 228 \cdot \frac{Nm}{Unit} \quad (2.8)$$

In der technischen Umsetzung hat sich herausgestellt, dass die Motoren zueinander ein unterschiedliches Verhalten aufweisen und somit die Konstante k nicht für alle drei Motoren verwendet werden kann. Daher wurde die Konstante k experimentell bestimmt. Hierzu wird ein Prüfhebel der Länge l_{pruef} für die Motoren konstruiert, der während des Betriebs der Motoren auf eine Waage drückt. Die Motoren werden mit einer bestimmten Folge von dimensionslosen Einheiten angesteuert und jeweils das angezeigte Gewicht der Waage m notiert. Mit der Beziehung

$$F = m \cdot g$$

eingesetzt in

$$M = F \cdot l_{pruef} \quad (2.9)$$

kann das resultierende Drehmoment M berechnet werden und somit die Konstante k für jeden einzelnen Motor.

Bei den Messungen wurde festgestellt, dass das von den Motoren erzeugte Drehmoment einen Drift² aufweist. Dieser Drift konnte reduziert werden, in dem man statt einmaligen Drehmomentbefehlen, die Befehle in Form einer Pulsung wiederkehrend an den Motor übergeben hat. Die neuen k -Faktoren sind über die Ergebnisse der Messungen, dargestellt in den Abbildungen 2.5, 2.6 und 2.7, mittels der Methode der kleinsten Quadrate berechnet worden. Aus diesen Abbildungen ist zu erkennen, dass sich die Drehmoment-Unit-Konstanten für die einzelnen baugleichen Motoren unterscheiden. Dieses Verhalten wurde bei der Implementierung in Kapitel 5 berücksichtigt.

2.3 Ballbot-Design

Die mechanische Systemkonfiguration kann in drei Teile untergliedert werden. Dies ist zum einen der Oberbau (Kap.2.3.1), der Unterbau (Kap.2.3.2) sowie der Ball (Kap.2.3.3), auf dem das Gesamtsystem bestehend aus Ober- und Unterbau balancieren wird. Nachfolgende Kapitel dokumentieren die Entwicklungsprozesse der einzelnen Teile.

² Unter einem Drift ist hier gemeint, dass bei konstantem Anlegen von Units (Stromwert) das Drehmoment über die Zeit abnimmt.

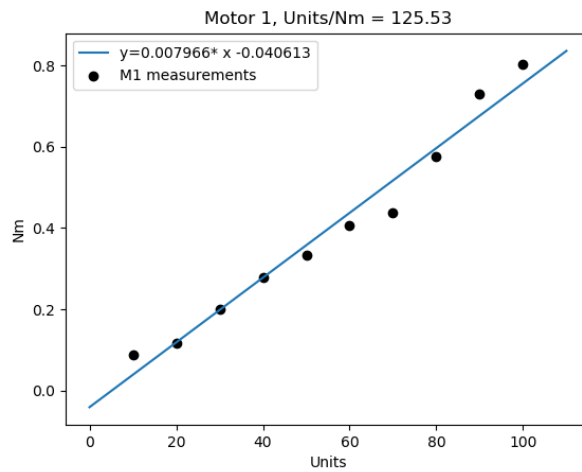


Abbildung 2.5.: Bestimmung der Drehmoment-Unit-Konstante für Motor 1

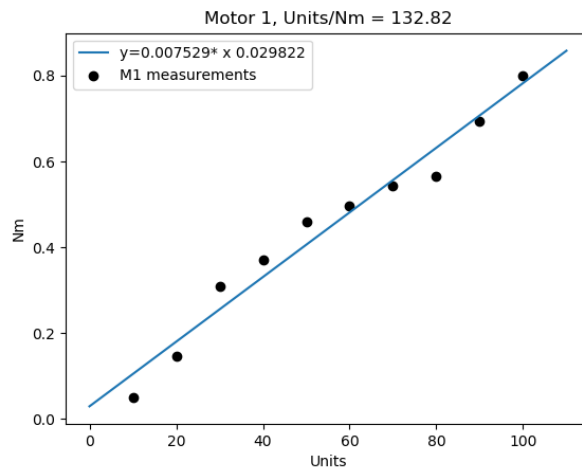


Abbildung 2.6.: Bestimmung der Drehmoment-Unit-Konstante für Motor 2

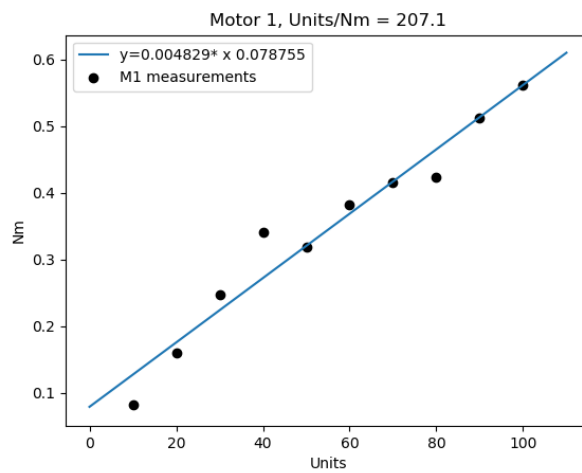


Abbildung 2.7.: Bestimmung der Drehmoment-Unit-Konstante für Motor 3

2.3.1 Oberbau



Abbildung 2.8.: Oberbau, bestehend aus vier Etagen

Die Konfiguration des Oberbaus hat sich experimentell unter Berücksichtigung der in Kapitel 3 hergeleiteten Beziehungen ergeben. In der untersten Ebene ist die Batterie platziert. Darüber folgt eine Ebene mit einem Up-Level Computer, der für rechenintensiver Aufgaben wie der Lokalisation benötigt wird. Dieser wird jedoch in dieser Arbeit nicht weiter berücksichtigt. Anschließend folgt das OpenCR-Board mit den darüber platzierten Sensoren zur Lokalisierung des Roboters im Raum, vgl. Abbildung 2.8.

2.3.2 Unterbau

Die Aufgabe des Unterbaus besteht in der Positionierung der Antriebe. Dabei sollte dieser möglichst leicht und zugleich stabil konstruiert werden.

Damit das Drehmoment der Motoren optimal auf den Ball übertragen werden kann, müssen die Räder gegenüber der z -Achse³ des Ballbots einen Winkel von $\alpha = 45^\circ$ einschließen. Weiterhin wurden die Räder um einen Winkel von $\beta = 120^\circ$ zueinander versetzt befestigt. Diese konstruktionsbedingten Winkel sind in Abbildung 2.10 dargestellt.

³ Die z -Achse ist die zum Ballbot vertikale Achse.

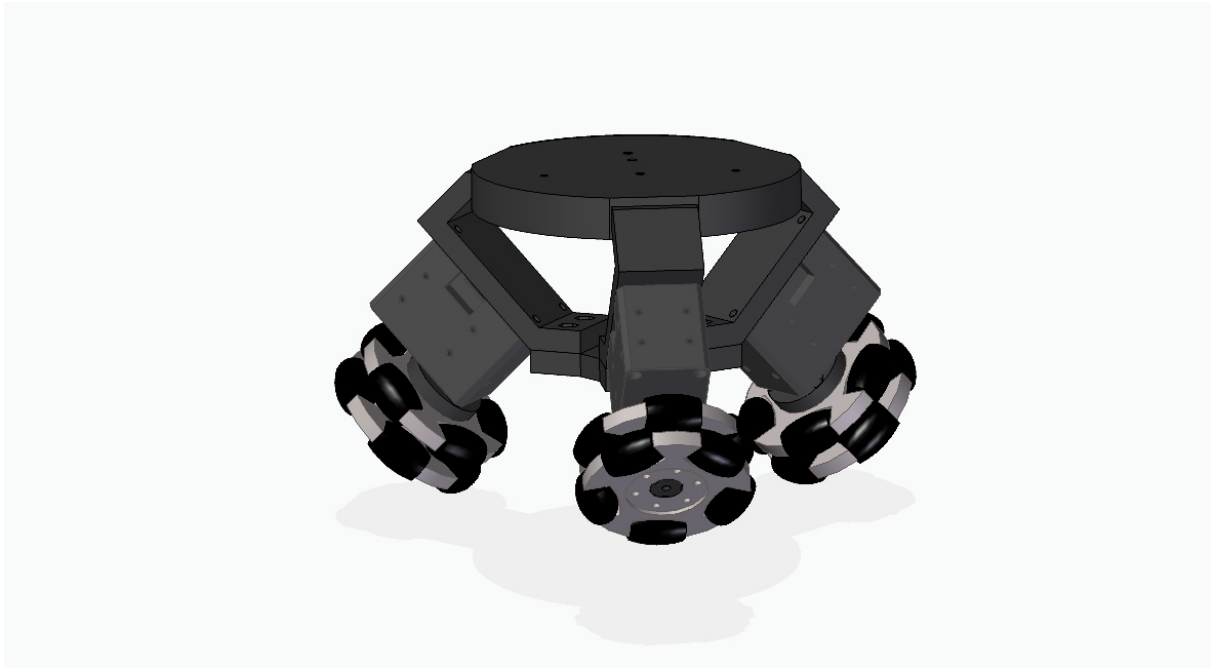


Abbildung 2.9.: Unterbau

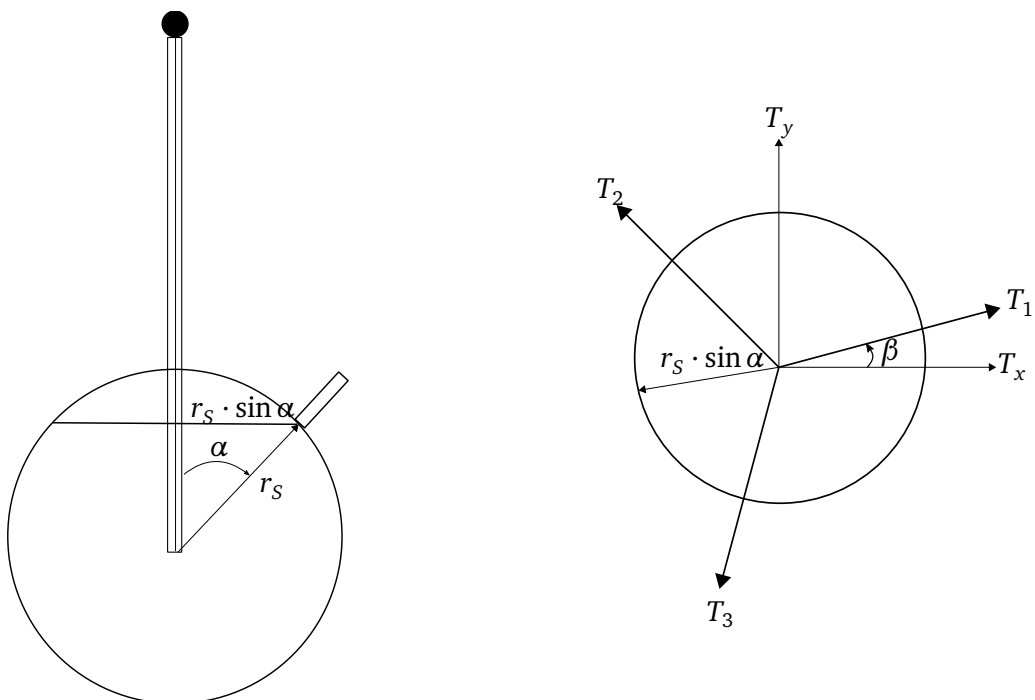


Abbildung 2.10.: Links: Seitenansicht der Ball-Rad- Konfiguration.

Rechts: Draufsicht der Ball-Rad -Konfiguration

Unter Berücksichtigung dieser Vorgaben ist der Unterbau entwickelt worden. Die dafür notwendigen Konstruktionen sind mit dem CAD-Tool SolidEdge durchgeführt und anschließend auf einen 3D-Drucker übertragen und gedruckt worden. Auf diesem Weg war es möglich schnell

und kostengünstig neue Ideen und Designs umzusetzen. Diese Schritte werden im Kapitel 2.4 erläutert. Zunächst soll jedoch noch auf die Wahl des Balls eingegangen werden.

2.3.3 Wahl des Balls

Der Ball stellt eine weitere Schlüsselkomponente bei der Entwicklung eines Ballbots dar. Im Verlauf der Balancierungstest sind einige verschiedene Bälle zum Einsatz gekommen. Die Suche nach einem optimalen Ball hat sich dabei als äußerst schwierig herausgestellt, da mehrere Parameter des Balls Einfluss auf das Balancierverhalten haben. Neben der Größe und Kompressibilität spielen besonders auch der Reibkoeffizient und das Trägheitsmoment eine große Rolle.

Die Größe des Balles muss auf die Auslegung des Unterbaus ausgerichtet werden, da sich sonst ein falscher Winkel α für das reale System ergibt. Weiterhin ist bei vielen Bällen immer wieder Schlupf zwischen Omniwheels und Ball aufgetreten. Sobald Schlupf auftrat, konnte der Ballbot durch die Regelung nicht mehr stabilisiert werden. Dies gilt es daher unbedingt zu vermeiden. Zudem dürfen keine Verformungen des Balles bei Belastungen durch den Ballbot auftreten, da sich der Ballbot sonst aufschaukelt und das Regelverhalten einer ungedämpften Schwingung gleicht. Am Ende der Experimente hat sich ein Handball der Größe II auf einer dünnen Schaumstoffmatte als bester Kompromiss herausgestellt. Die Schaumstoffmatte wurde dabei zur Dämpfung der Ballbewegung verwendet.

2.4 Konstruktion mittels SolidEdge

SolidEdge ist eine Computer gestütztes Design(CAD)-Software, die ein rechnergestütztes Konstruieren einzelner Bauteile sowie ganzer Baugruppen eines Produktes, einer Maschine, etc. ermöglicht. Weiterhin gibt es auch Tools zur Bauteil-Optimierung und Simulation von Strömungen. SolidEdge wird für Studenten kostenlos von der Siemens Industry Software GmbH zur Verfügung gestellt und kann unter Angabe der persönlichen Daten heruntergeladen werden⁴.

2.4.1 SolidEdge - Eine kurze Einführung

Bei der Konstruktion der einzelnen Unterbaukomponenten des Ballbots sind hauptsächlich zwei Funktionen von SolidEdge genutzt worden. Zum einen die Erzeugung einzelner Bauteile, worunter im Folgenden ein einzelner Körper⁵ verstanden wird, sowie die Erzeugung von Baugruppen⁶. Bei einer Baugruppe werden die Bauteile automatisch je nach Montagevorschrift zusammengefügt und über sogenannte Beziehungen miteinander verbunden. Wird eine Baugruppe in eine

⁴ https://www.plm.automation.siemens.com/plmapp/education/solid-edge/en_us/free-software/student (Stand: 07.02.2018)

⁵ Unter dem Begriff Körper soll in diesem Anwendungsfall ein einzelnes, nicht aus mehreren Teilkomponenten bestehendes, Werkstück verstanden werden.

⁶ Zusammensetzung/Montage einzelner Körper zu einer Gruppe.

neue Datei geladen, verhält sich die Baugruppe durch die definierten Beziehungen wie ein einzelner Körper. Auf diesem Weg können in eine Baugruppe auch andere Baugruppen geladen und miteinander in Beziehung gestellt werden. Das macht ein schrittweises und übersichtliches Zusammensetzen der Gesamtkonstruktion möglich.

2.4.2 SolidEdge - Umsetzung

Es soll nun ein Einblick in die Vorgehensweise bei der Konstruktion des Unterbaus mit SolidEdge gegeben werden. Abbildung 2.9 zeigt die Baugruppe „Unterbau“. Folgende Kriterien galt es dabei zu erfüllen: Die Baugruppe sollte aus mehreren Körpern bestehen, sodass ein einfaches Montieren der einzelnen Komponenten möglich ist. Weiterhin sollte eine flexible Konstruktion entwickelt werden, um den Ballbot auf Bällen mit verschiedenen Radien testen zu können. Dies wurde durch ein Teleskopsystem ermöglicht, sodass die Motoren inklusive Räder radial nach außen bzw. innen geschoben werden können. Dies ist in Abbildung 2.11 dargestellt.

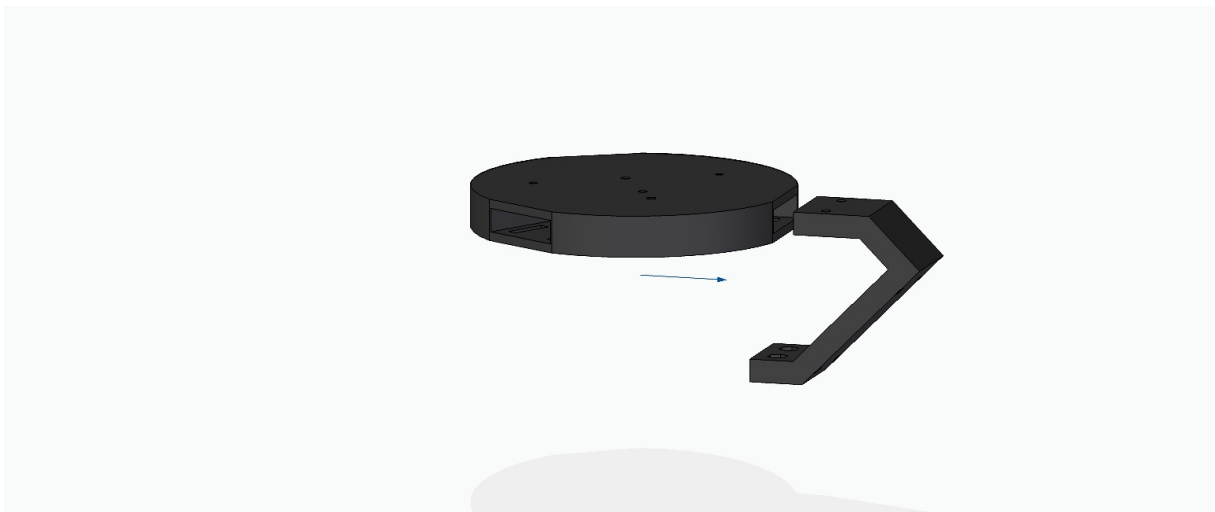


Abbildung 2.11.: Teleskopsystem bestehend aus Trägerplatte(links) und Motorenhalter(rechts)

Rücksicht wurde weiterhin auf eine stabile Integration der Motoren in die Gesamtkonstruktion genommen. So soll bei den wirkenden Kräften und Momenten, Bewegungen der Unterbaukomponenten relativ zu einander vermieden werden. Integriert wurden die Motoren daher über vier M3 Zylinderkopfschrauben an den sogenannten Motorenhaltern, wie in Abbildung 2.12 dargestellt. Die Führungsschiene des Motorenhalters wird in eine Führungsnut der Trägerplatte passgenau eingeführt und verschraubt.

Um die Räder, dargestellt in Abbildung 2.13 an die Motorwellen zu befestigen, sind spezielle Mitnehmer konstruiert worden, die in Abbildung 2.12 dargestellt sind.

Wichtige Kriterien, die es hierbei zu erfüllen gab, waren eine spielfrei Übertragung der Momente und Kräfte sowie eine möglichst kurze Distanz zwischen den Rädern und den Motoren.



Abbildung 2.12.: Antriebsvorrichtung bestehend aus Mitnehmer (unten), Motor (mitte) und Motorenhalter (oben).

Nach einigen Tests hat sich herausgestellt, dass die gewünschte Stabilität des Unterbaus nicht gegeben war. Die Konstruktion musste also noch verstärkt werden. Um die nötige Stabilität herzustellen, sind die Motorenhalter um eine Anflanschfläche erweitert worden. So können die Motorenhalter über ein Y-förmiges Verbindungsstück, dargestellt in Abbildung 2.14, miteinander gekoppelt werden. Hierbei wurde eine flexible Anpassung an unterschiedlichen Ballradien berücksichtigt.



Abbildung 2.13.: Modell eines omnidirektionalen Rades

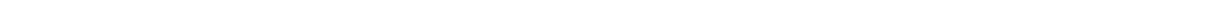


Abbildung 2.14.: Y-förmiges Verbindungsstück des Unterbaus.

2.5 Fertigung der Komponenten

Neben dem Zugang zu einem professionellen CAD-Tool bestand auch Zugang zu einem 3D-Drucker der Firma Oktoprint. Dies ermöglichte eine kostengünstige und schnelle Fertigung der konstruierten Bauteile. Verwendete wurde dabei das Filament Polylactide (PLA). PLA ist ein sehr verbreiteter biokompatibler Kunststoff, der eine hohe Oberflächenhärte, hohe Steifigkeit und eine hohe Zugfestigkeit bietet [7].

Der fertige Entwurf wurde schließlich als .stl exportiert und in das 3D-Druckprogramm CURA geladen. Mit CURA konnte anschließend die Fülldichte, die Qualität, Druckgeschwindigkeit und der Einsatz von Stützhilfen für den jeweiligen Druck eingestellt werden. Die gewünschten Genauigkeiten konnten mittels dieser Fertigungsweise eingehalten werden. So hat der Druck auch an kritischen Stellen wie der Einschubverbindung zwischen Trägerplatte und Motorenhalter überzeugt. Es war nach Einschub und Verschraubung der Motorenhalter mit der Trägerplatte kein Spiel vorhanden.



3 Modellbildung und Regelung

In diesem Kapitel wird anhand des Vorgehens in [1] die Modellbildung des Ballbots erläutert, die für die Auslegung eines geeigneten Regler benötigt wird. Im Anschluss kann mit Hilfe der Simulationsumgebung MATLAB/Simulink sowohl das Modell als auch der Regler verifiziert werden.

3.1 Model

Für die Modellbildung wird der dreidimensionale Ballbot in die yz -, xz - und xy -Ebenen aufgeteilt, das in Abbildung 3.1 dargestellt ist. In jeder Ebene wird das System vereinfacht als Zusammensetzung von drei starren Körpern, bestehend aus einer Kugel, einem virtuellem Rad und einem Körper betrachtet. Das Modell jeder Ebene besitzt somit zwei Freiheitsgrade, die sich in eine Translation bzw. Rotation des Balles und eine Rotation des Körpers aufteilen lassen [1].

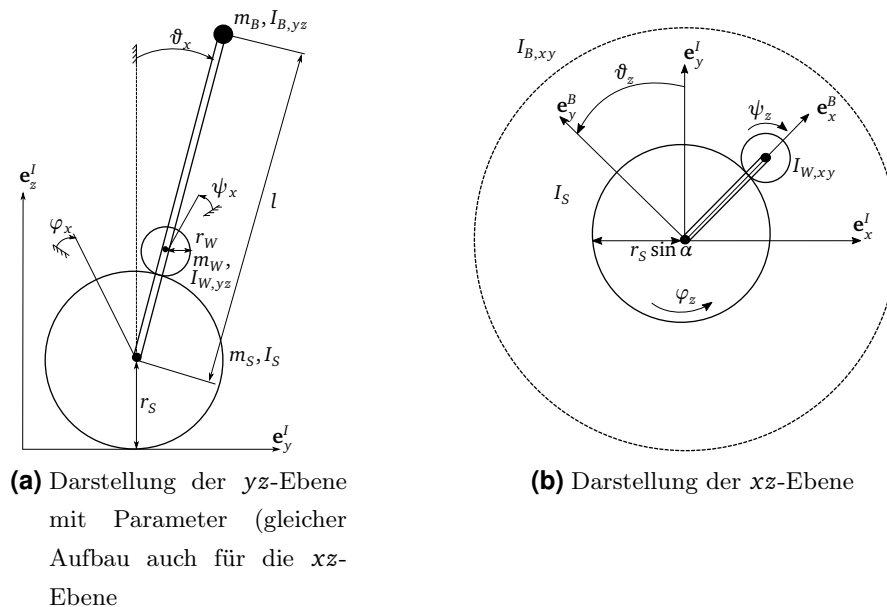


Abbildung 3.1.: Aufteilung des Ballbots in drei unabhängigen Ebenen [1]

Um möglichst vereinfachte Modelle der drei Ebenen zu erhalten, werden weitere Annahmen getroffen, die im Folgenden erläutert werden [1]:

- Kein Schlupf: Das System besitzt zwei Kontaktpunkte, in denen ein Schlupf auftreten kann. Hierzu zählt zum einen der Kontaktpunkt von Ball und Boden und zum anderen

der Kontaktpunkt zwischen Rädern und Ball. Damit kein Schlupf auftritt, müssen die angelegten Drehmomente begrenzt werden.

- Keine Reibung: Für die Modellierung wird ein reibungsfreies System angenommen. Lediglich bei der Rotation des Balles um die z-Achse wird die Reibung berücksichtigt.
- Keine Deformation: Der Ball wird als nicht deformierbar angenommen.
- Schnelle Motorendynamik: Für die Gleichgewichtsstabilisierung des Roboters ist es wichtig, dass die Motoren eine schnellere Dynamik als das Gesamtsystem aufweisen.
- Horizontale Bewegung: Die Ballbewegung wird für die horizontale Bewegung auf einer flachen Oberfläche ohne starken Neigungen eingeschränkt. Somit wird die vertikale Bewegung vernachlässigt.

Mit den getroffenen Annahmen ist es möglich, das Modell des Ballbots aufzustellen.

3.2 Minimalkoordinaten

Für die Beschreibung der einzelnen Modellebenen werden bestimmte Winkel verwendet, die in Abbildung 3.1 zu sehen sind. Dabei beschreiben die Winkel ϑ_{xyz} die Orientierung des Roboters, die Winkel φ_{xyz} die Orientierung des Balles und die Winkel ψ_{xyz} die Orientierung des virtuellen Rades in den jeweiligen planaren Ebenen. Unter Verwendung folgender minimalen Koordinaten

$$\mathbf{q}_{yz} = \begin{bmatrix} \varphi_x \\ \vartheta_x \end{bmatrix} \quad \mathbf{q}_{xz} = \begin{bmatrix} \varphi_y \\ \vartheta_y \end{bmatrix} \quad \mathbf{q}_{xy} = \begin{bmatrix} \varphi_z \\ \vartheta_z \end{bmatrix} \quad (3.1)$$

kann das komplette System mit Hilfe von Energien beschrieben werden [1].

3.3 Energien

Das Aufstellen der Bewegungsgleichungen für die einzelnen Ebenen wird anhand der LAGRANGEschen Gleichungen zweiter Art durchgeführt. Dazu müssen im Voraus die potentiellen und kinetischen Energien der einzelnen Körper aufgestellt werden. Dabei ist darauf zu achten, dass die Formeln der potentiellen und kinetischen Energien für die yz- und der xz-Ebene identisch sind und sich nur durch die entsprechenden Winkel der Ebenen unterscheiden. Damit der Ballbot zunächst seine Gleichgewichtslage halten kann, hat die Orientierung um die z-Achse eine vernachlässigbare Bedeutung. Aus diesem Grund wird die xy-Ebene für den Entwurf eines Reglers nicht berücksichtigt und somit auch für die Herleitung der Bewegungsgleichungen.

Die Herleitung der Bewegungsgleichungen wird nachfolgend für die yz-Ebene durchgeführt. Die kinetische und potentielle Energie des Balles entsprechen folgenden Formeln:

$$T_{S,yz} = \frac{1}{2} \cdot m_S \cdot (r_S \cdot \dot{\varphi}_x)^2 + \frac{1}{2} \cdot I_S \cdot \dot{\varphi}_x^2 \quad (3.2)$$

$$V_{S,yz} = 0 \quad (3.3)$$

Die potentielle Energie kann mit Null gleichgesetzt werden, da der Ursprung des Weltkoordinatensystems in den Mittelpunkt des Balles gelegt wird.

Die kinetische und potentielle Energie des virtuellen Rades in der yz -Ebene wird mit

$$T_{W,yz} = \frac{1}{2} \cdot m_W \cdot ((r_S \cdot \dot{\varphi}_x)^2 + 2 \cdot (r_S + r_W) \cdot \cos(\vartheta_x) \cdot \dot{\vartheta}_x \cdot (r_S \cdot \dot{\varphi}_x) + (r_S + r_W)^2 \cdot \dot{\vartheta}_x^2) + \frac{1}{2} \cdot I_W \cdot \left(\frac{r_S}{r_W} \cdot (\dot{\varphi}_x - \dot{\vartheta}_x) - \dot{\vartheta}_x\right)^2 \quad (3.4)$$

$$V_{W,yz} = m_W \cdot g \cdot (r_S + r_W) \cdot \cos(\vartheta_x) \quad (3.5)$$

berechnet. Dabei besteht die kinetische Energie ebenfalls aus einem translatorischem und rotatorischem Teil und einem zusätzlichem Kopplungsteil. Die potentielle Energie ist abhängig von dem Winkel ϑ_x .

Für den Körper bzw. den Roboter sind die Formeln für die kinetische und potentielle Energie ähnlich zu denen des virtuellen Rads und lauten:

$$T_{B,yz} = \frac{1}{2} \cdot m_A \cdot ((r_S \cdot \dot{\varphi}_x)^2 + 2 \cdot l \cdot \cos(\vartheta_x) \cdot \dot{\vartheta}_x \cdot (r_S \cdot \dot{\varphi}_x) + l^2 \cdot \dot{\vartheta}_x^2) + \frac{1}{2} \cdot I_A \cdot \dot{\vartheta}_x^2 \quad (3.6)$$

$$V_{B,yz} = m_A \cdot g \cdot l \cdot \cos(\vartheta_x) \quad (3.7)$$

Die Gesamtenergie setzt sich aus allen Teilenergien der einzelnen Komponenten des Ballbots zusammen [1].

3.3.1 Externe Kräfte und Drehmomente

Die Krafteinwirkung auf das Gesamtsystem erfolgt über das Drehmoment der Motoren. Für die einzelnen Ebenen der Modellbildung wirkt daher über das entsprechende virtuelle Rad ein virtuelles Drehmoment $T_{x,y,z}$. Daher kann die resultierende Krafteinwirkung in zwei Teile aufgeteilt werden. Ein Teil resultiert aus dem Effekt der Motordrehmomente und der andere Teil beschreibt die Krafteinwirkung durch das erzeugte Gegendrehmoment. Die gesamte Krafteinwirkung kann durch die Summe von

$$\mathbf{f}_{NB,yz1} = \begin{bmatrix} \frac{r_S}{r_W} \cdot T_x \\ -\left(1 + \frac{r_S}{r_W}\right) \cdot T_x \end{bmatrix} \quad \mathbf{f}_{NB,yz2} = \begin{bmatrix} 0 \\ T_x \end{bmatrix} \quad (3.8)$$

für die yz -Ebene angeben werden und für die Modellierung als Kraftanregung genutzt werden.

Ein weiterer, wichtiger Punkt in Bezug auf die spätere Simulation und Implementierung ist die Umrechnung von den virtuellen Drehmomenten $T_{x,y,z}$ auf die realen Drehmomente $T_{1,2,3}$ der Motoren. Die Umrechnung erfolgt mit

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \frac{1}{3} \cdot \begin{bmatrix} \frac{2 \cdot \cos \beta}{\cos \alpha} & \frac{-2 \cdot \sin \beta}{\cos \alpha} & 1 \\ \frac{-\sin \beta \cdot \sqrt{3} - \cos \beta}{\cos \alpha} & \frac{\sin \beta - \cos \beta \cdot \sqrt{3}}{\cos \alpha} & 1 \\ \frac{\sin \beta \cdot \sqrt{3} - \cos \beta}{\cos \alpha} & \frac{\sin \beta + \cos \beta \cdot \sqrt{3}}{\cos \alpha} & 1 \end{bmatrix} \cdot \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \quad (3.9)$$

Für die genaue Herleitung der in diesem Kapitel beschriebenen Gleichungen ist auf [1] verwiesen.

3.4 Bewegungsgleichungen

Mit den kinetischen und potentiellen Energien können nun die Bewegungsgleichungen des Ballbots für die jeweiligen Ebenen mit Hilfe der LAGRANGEschen Gleichungen zweiter Art hergeleitet werden. Als Beispiel wird auch hier die Herleitung für die yz -Ebene angegeben.

Zunächst wird aus den einzelnen Summen der kinetischen und potentiellen Energien die LAGRANGEsche Funktion gebildet [8].

$$L = T - V \quad (3.10)$$

Die Bewegungsgleichungen für den Ballbot resultieren aus dem Lösen der LAGRANGEschen Gleichungen zweiter Art.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\mathbf{q}}} \right)^T - \left(\frac{\partial L}{\partial \mathbf{q}} \right)^T = \mathbf{f}_{NP,yz} \quad (3.11)$$

Mit mathematische Umformungen können diese Bewegungsgleichungen in eine Matrixform überführt werden:

$$\mathbf{M}_x(\mathbf{q}, \dot{\mathbf{q}}) \cdot \ddot{\mathbf{q}} + \mathbf{C}_x(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}_x(\mathbf{q}) = \mathbf{f}_{NP,yz} \quad (3.12)$$

Hierbei ist:

$$\begin{aligned} \mathbf{M}_x &= \begin{bmatrix} m_{tot} \cdot r_S^2 + I_S + \left(\frac{r_S}{r_W}\right)^2 \cdot I_W & -\frac{r_S}{r_W} \cdot r_{tot} \cdot I_W + \gamma \cdot r_S \cdot \cos \vartheta_x m_{tot} \\ -\frac{r_S}{r_W} \cdot r_{tot} \cdot I_W + \gamma \cdot r_S \cdot \cos \vartheta_x m_{tot} & \frac{r_S^2}{r_W} \cdot I_W + I_B + m_b \cdot l^2 + m_W \cdot r_{tot}^2 \end{bmatrix}, \mathbf{M}_x \in \mathbb{R}^{2 \times 2} \\ \mathbf{C}_x &= \begin{bmatrix} -r_S \cdot \gamma \cdot \sin \vartheta_x \cdot \dot{\vartheta}_x^2 \\ 0 \end{bmatrix}, \mathbf{C}_x \in \mathbb{R}^{2 \times 1} \\ \mathbf{G}_x &= \begin{bmatrix} 0 \\ -g \cdot \sin \theta_x \cdot \gamma \end{bmatrix}, \mathbf{G}_x \in \mathbb{R}^{2 \times 1} \end{aligned} \quad (3.13)$$

mit den Substitutionen:

$$\begin{aligned} m_{tot} &= m_B + m_S + m_W \\ r_{tot} &= r_S + r_W \\ \gamma &= l \cdot m_B + (r_S + r_W) m_W \end{aligned} \quad (3.14)$$

3.5 Zustandsraumdarstellung

Bei den hergeleiteten Bewegungsgleichungen für die yz -Ebene handelt es sich um nichtlineare Funktionen. Um ein lineares Zustandsraummodell zu erhalten, müssen zunächst die Zustände

bzw. der Zustandsvektor festgelegt werden. Mit $\mathbf{x} = [\varphi_x \ \vartheta_x \ \dot{\varphi}_x \ \dot{\vartheta}_x]^T$ können die nichtlinearen Funktionen in folgende Form überführt werden.

$$\dot{\mathbf{x}}_{nl} = \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{M}_x^{-1} \cdot (\mathbf{f}_{Np,yz} - (\mathbf{C}_x + \mathbf{G}_x)) \end{bmatrix} \quad (3.15)$$

Das Ziel dieser Arbeit ist das Auslegen eines Reglers, damit der Roboter innerhalb eines kleinen Bereiches um den Arbeitspunkt $\mathbf{x}_0 = [0 \ 0 \ 0 \ 0]^T$ auf dem einen Ball balanciert. Hierzu muss zunächst das nichtlineare Modell um den Arbeitspunkt mit der Taylor-Reihenentwicklung linearisiert werden, um die lineare Zustandsraumdarstellung

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u} \\ \mathbf{y} &= \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{u} \end{aligned} \quad (3.16)$$

zu erhalten. Mit den zum Roboter dazugehörigen Parametern (siehe Anhang A) und dem eingesetzten Arbeitspunkt lauten die konkreten Matrizen für die lineare Zustandsraumdarstellung der yz -Ebene folgendermaßen:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.4886 & 0 & 0 \\ 0 & 17.1119 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 63.5516 \\ -10.6945 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Damit der Roboter auf einen Ball die eigene Gleichgewichtslage einhalten kann, ist es zunächst nebensächlich, welche Position und Geschwindigkeit der Ball dabei einnimmt. Vielmehr von Bedeutung ist die aufrechte Haltung des Ballbots. Dadurch sind für den späteren Entwurf einer Regelung nur die Zustände des Roboters von Bedeutung und die Zustände für den Ball müssen nicht für eine Regelung berücksichtigt werden. Mit dem neuem Zustandsvektor $\mathbf{x}^* = [\vartheta_x \ \dot{\vartheta}_x]^T$ wird die Ordnung des Modells 3.16 reduziert, das zu folgenden Zustandsraummatrizen führt:

$$\mathbf{A}^* = \begin{bmatrix} 0 & 1 \\ 17.1119 & 0 \end{bmatrix} \quad \mathbf{B}^* = \begin{bmatrix} 0 \\ -10.6945 \end{bmatrix} \quad \mathbf{C}^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{D}^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Die Prüfung auf Stabilität des Modells erfolgt anhand der Lage der Eigenwerte. Das ursprüngliche und das reduzierte Modell besitzt folgende Eigenwerte:

$$\lambda = \begin{pmatrix} 0 \\ 0 \\ 4.1367 \\ -4.1367 \end{pmatrix} \quad \lambda^* = \begin{pmatrix} 4.1367 \\ -4.1367 \end{pmatrix} \quad (3.17)$$

Das ursprünglich Modell besitzt zwei Eigenwerte $\lambda_{1,2} = 0$ auf der imaginären Achse und einen instabilen Eigenwert $\lambda_3 = 4.1367$ in der rechten Halbebene des Bildbereiches. Somit ist das Modell instabil. Bei dem reduzierten Modell handelt es sich ebenfalls um ein instabiles System, da es auch einen Eigenwert $\lambda_1^* = 4.1367$ besitzt.

Die Untersuchung der Steuer- und Beobachtbarkeit erfolgt in beiden Modelle mit den entsprechenden Kriterien von KALMANN. In beiden Fällen besitzen die Steuerbarkeits- und Beobachtbarkeitsmatrizen vollen Rang, wodurch beide Modelle vollständig steuer- und beobachtbar sind. Mit diesen Systemeigenschaften ist es mit Hilfe eines Reglers möglich, das entsprechende instabile System zu stabilisieren. Das Auslegen eines geeigneten Regler wird nachfolgend erläutert [1][9].

3.6 Reglerentwurf

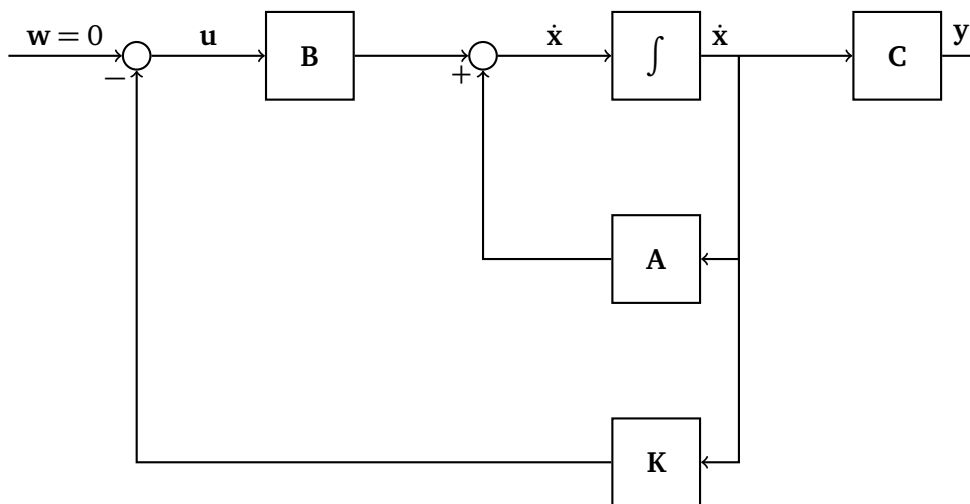


Abbildung 3.2.: Linearisierte Zustandsraumdarstellung mit rückgekoppelter Reglerverstärkung [1]

Der Entwurf eines geeigneten Reglers zur Stabilisierung der Gleichgewichtslage des Roboters wird im Folgenden auf das reduzierte Modell beschränkt. Für die beiden Modelle der yz - und xz -Ebene wird jeweils ein linear-quadratischer Zustandsregler (LQR) ausgelegt, dessen Funktionsweise in Abbildung 3.2 dargestellt ist und auf der Minimierung des Gütemaß

$$J = \int_0^{\infty} \mathbf{x}^T(t) \cdot \mathbf{Q} \cdot \mathbf{x}(t) + \mathbf{u}^T(t) \cdot \mathbf{R} \cdot \mathbf{u}(t) dt \quad (3.18)$$

durch einen vorgegebenen Stellgrößenverlauf basiert. In das Güteintegral geht quadratisch der Trajektorienverlauf $\mathbf{x}(t)$ und der Stellgrößenverlauf $\mathbf{u}(t)$ ein. Dabei handelt es sich bei der symmetrischen und positiv semidefiniten Matrix \mathbf{Q} um eine Gewichtungsmatrix für den Trajektorienverlauf. Für ein rasches Einschwingen eines bestimmten Zustandes ist dieser entsprechend mit einem größeren Gewicht in der Gewichtungsmatrix zu versehen. Die symmetrische und positiv definite Matrix \mathbf{R} gewichtet dagegen die Stellgröße. Müssen Stellgrößenbeschränkungen berücksichtigt werden, sind die entsprechenden Stellgrößen in dieser Matrix mit größeren

Gewichten zu versehen. Wird insgesamt die Gewichtungsmatrix \mathbf{Q} im Verhältnis zur der Gewichtungsmatrix \mathbf{R} vergrößert, führt das zu einem schnelleren Einschwingen der Zustände mit höheren Stellgrößen. Im umgekehrten Fall stabilisiert sich das System langsamer, da die Höhe der Stellgrößen abnimmt [10][11].

Das Regelgesetz, das aus der Minimierung des Gütemaß resultiert, ergibt sich zu

$$\mathbf{u}(t) = -\mathbf{K} \cdot \mathbf{x}(t) \quad \text{mit} \quad \mathbf{K} = \mathbf{R}^{-1} \cdot \mathbf{B}^T \cdot \mathbf{P}. \quad (3.19)$$

Die symmetrisch positiv definite Matrix \mathbf{P} errechnet sich aus dem algebraischen Teil der Riccati-Differentialgleichung

$$\mathbf{A}^T \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{A} - \mathbf{P} \cdot \mathbf{B} \cdot \mathbf{R}^{-1} \cdot \mathbf{B}^T \cdot \mathbf{P} = -\mathbf{Q}. \quad (3.20)$$

Das Lösen der Riccati-Differentialgleichung und somit die Berechnung der Reglermatrix \mathbf{K} kann mit Hilfe von MATLAB und dem dazugehörigen Befehl `lqr()` durchgeführt werden. Zu Beginn werden die einzelnen Gewichtungen der einzelnen Zustände und Stellgrößen zu

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 \\ 0 & 50 \end{bmatrix} \quad \mathbf{R} = 200$$

festgelegt. Mit dieser Konfiguration werden die einzelnen Verstärkungsfaktoren des Regler zu

$$\mathbf{K} = \begin{bmatrix} -3.3494 & -0.9362 \end{bmatrix} \quad (3.21)$$

berechnet.

Diese Verstärkungsfaktoren sind nicht ideal für das System. Durch manuelles Anpassen der Werte kann das Balancieren des Roboters um die Gleichgewichtslage schrittweise angepasst werden, das schließlich zu folgende Reglermatrix

$$\mathbf{K} = \begin{bmatrix} -8.6953 & -2.0932 \end{bmatrix} \quad (3.22)$$

führt.

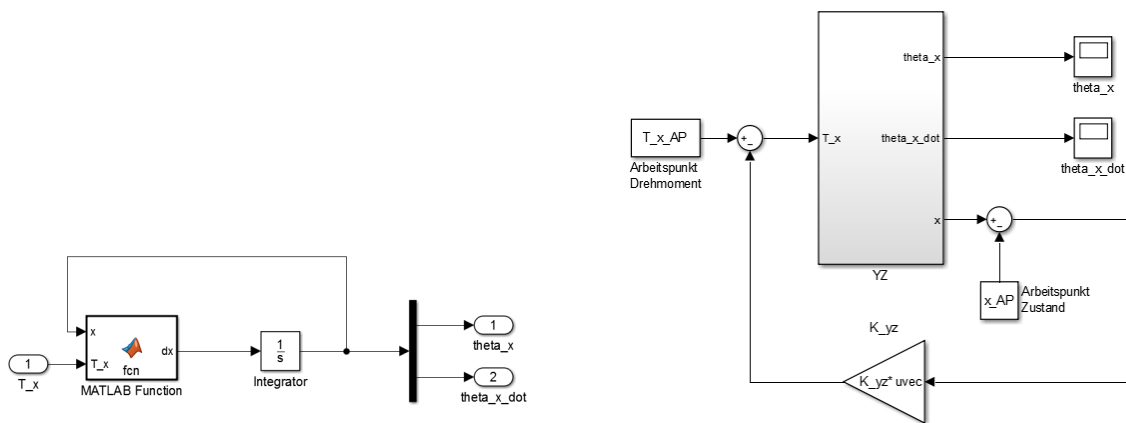
3.7 Simulation des Modells

Mit dem Aufbau und Durchführung einer Simulation mittels MATLAB/Simulink können sowohl die nichtlinearen Bewegungsgleichungen und somit das dynamische Systemverhalten des Ballbots als auch der entworfene Zustandsregler verifiziert werden. Weiterhin besteht die Möglichkeit, sowohl die Verstärkungsfaktoren des Regler zu optimieren, um die Dynamik des Systems zu erhöhen oder Stellgrößen zu beschränken, als auch Grenzsituationen zu testen. Dadurch entfällt das Testen am realen System, das zeitaufwendig ist und zu eventuellen Beschädigungen führen kann [12].

Im Folgenden wird zunächst die allgemeine Implementierung in MATLAB/Simulink erläutert, die dann schrittweise an die realen Gegebenheiten angepasst wird.

3.7.1 Implementierung ideales Modell

Ein wichtiger Bestandteil im Aufbau der Simulation besteht in dem Lösen der nichtlinearen Differentialgleichungen. Für diese Aufgabe wird in MATLAB/Simulink für jede Ebene ein MATLAB-Funktion-Block eingesetzt. Die Programmierung bzw. Parametrierung dieses Blockes erfolgt nicht grafisch, sondern mit Unterstützung eines Editors. Als Eingang wird dem Block in jedem Berechnungsschritt der aktuelle Zustandsvektor \mathbf{x} und das aktuelle, virtuelle Drehmoment T der entsprechenden Ebene übergeben. Mit den Werten wird dann die Bewegungsdifferentialgleichung für den aktuellen Schritt berechnet. Als Rückgabewert bzw. Ausgang wird die berechnete, zeitliche Ableitung des Zustandsvektors $\dot{\mathbf{x}}$ zurückgegeben. Die Abbildung 3.3 (a) zeigt die Funktionsweise des Lösens der nichtlinearen Bewegungsgleichungen für die yz-Ebene [8].



(a) Aufbau Matlab-Funktion-Block mit anschließenden Integrierer für yz-Ebene

(b) Aufbau Gesamtsystem inklusive Regler für yz-Ebene

Abbildung 3.3.: Aufbau Simulinkmodell für yz-Ebene bestehen aus Subsystem und Gesamtsystem mit Regler

Mit der Hinzunahme der Reglerverstärkungsfaktoren als Zustandsrückführung ergibt sich das Gesamtsystem, das in der Abbildung 3.3 (b) abgebildet ist. Hier ist zu beachten, dass der Zustandsregler für das linearisierte Modell ausgelegt wurde. Deshalb muss der Arbeitspunkt des Zustandes vor der Rückführung abgezogen werden. Nach dem Regler muss der Arbeitspunkt des Drehmomentes analog wieder auf das Δu addiert werden. Im Anhang D ist eine komplette Übersicht des Simulink-Simulationsaufbaus zu sehen.

3.7.2 Reale Gegebenheiten

Die inertielle Messeinheit liefert in der Realität verrauschte Signale, die für die Berechnung des Regelgesetzes benötigt werden. Die Stärke dieses Rauschen kann dem Datenblatt der IMU entnommen werden und in der Simulation durch ein weißes Rauschen integriert werden. Die

Abbildung 3.4 zeigt den Einfluss des Rauschen auf den Winkel ϑ_x und das reale Drehmoment T_1 in der Simulation.



Abbildung 3.4.: Rauscheinfluss auf den Winkel ϑ_x und auf das reale Drehmoment T_1 in der Simulation

Ein weiterer Effekt, der in der Realität auftritt, ist das Auslesen der Sensorsignale, die nur in bestimmten Zeitabständen aktualisiert ausgelesen werden können. Daraus ergibt sich eine Abtastfrequenz, die sich additiv aus der Aktualisierungsfrequenz der IMU, der benötigten Zeit für das Auslesen/ Verarbeiten der Sensordaten und dem Ausgeben der Drehmomente zusammensetzt. Mit der eingesetzten Hardware des Ballbots ergibt sich eine Abtastfrequenz (engl. Sampling Time) von $f = 200$ Hz.

In der Realität ergibt sich durch das Schreiben der Motorenmomente eine Totzeit von $t_{V,Motoren} = 3$ ms.

Diese Auswirkungen werden in der Simulation durch ein Abtast-Halteglied und ein Totzeitglied berücksichtigt. Sie sind anhand des Winkels ϑ_x und des Drehmoments T_1 in der Abbildung 3.5 dargestellt.

Durch die zusätzliche, integrierte Abtastung des Sensorsignals der IMU ist zu erkennen, dass der Verlauf des Drehmomentes T_1 und somit auch der anderen Drehmomente einen Treppenförmigen-Effekt hat. Durch das Lösen der nichtlinearen Bewegungsgleichungen mit Hilfe des Matlab-Funktion-Blocks stellt sich für den Ausgang des Systems, den Sensorsignalen, ein rampenförmiger Verlauf ein.

Der Verlauf des Winkels ϑ_x und des Drehmomentes T_1 in Abbildung 3.5 ist zur besseren Übersicht für den Zeitraum $\Delta t = [2 \ 3]$ dargestellt. Zudem wurde eine Anfangsauslenkung von 5° um die x -Achse vorgegeben, die mit Hilfe des Reglers ausgeregelt werden kann, damit das System stabilisiert werden kann.

Als Ergebnis der Simulation unter Einbinden möglichst aller realen Gegebenheiten führt zu einer Stabilisierung des System, d.h. der Roboter kann um seine Gleichgewichtslage balancieren ohne Stellgrößenüberschreitung. Eine Erklärung liegt hierbei in der Vernachlässigung der Reib- und Schlupfeffekte, die im Kapitel 3.1 eingeführt wurden.

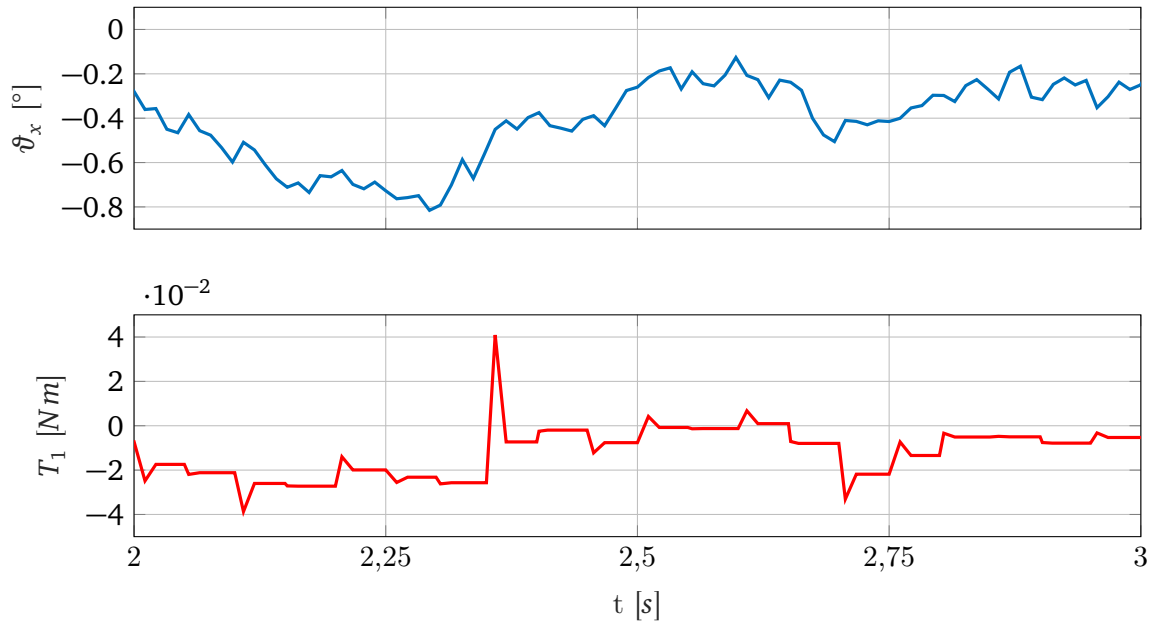


Abbildung 3.5.: Berücksichtigung von Rauschen, Abtastung und Zeitverzug im Bezug auf Winkel ϑ_x und reales Drehmoment T_1 in der Simulation

4 Simulation

Die Simulation des Ballbots wurde mittels Gazebo realisiert und kann mittels eines globalen ROS launch files gestartet werden. Die Simulation steht frei zur Verfügung und kann über [13] abgerufen werden. Zudem gibt es ein Youtube Video [14] das zeigt, wie man die Simulation auf einem Ubuntu-Betriebssystem mit ROS-Kinetic und Gazebo7 testen kann.

4.1 3D Simulatoren

Für eine 3D-Simulation des Ballbots bieten sich grundsätzlich zwei 3D Simulatoren an: Gazebo und V-Rep. Die Unterschiede dieser beiden Simulatoren sind in Tabelle 4.1 aufgeführt.

Auf dem Up-Board des Ballbots soll ROS¹ laufen. Aus diesem Grund wurde der Gazebo Simulator gewählt, denn dieser ist der Standard Simulator von ROS und daher besser integriert als V-REP[15].

4.2 Simulation von omnidirektionalen Rädern

Bei der Simulation des Ballbots in Gazebo stellt sich zunächst die Frage, wie man die omnidirektionalen Räder simulieren soll. Hierzu gibt es zwei Möglichkeiten:

Die erste Möglichkeit besteht darin, das omnidirektionale Rad ohne frei drehende kleine Räder zu simulieren. Um stattdessen das omnidirektionale Rad zu simulieren, gibt man dem Rad zwei verschiedene Reibungskoeffizienten für die unterschiedlichen Reibungsrichtungen vor. Die Reibungskoeffizienten werden in Gazebo μ_1 und μ_2 genannt. Der erste Reibungskoeffizient entspricht der Reibung zwischen Ball und Rad. Der zweite Reibungskoeffizient muss zu 0 gewählt werden, denn dieser simuliert die kleinen frei drehenden Räder des omnidirektionalen Rades. Zusätzlich muss man noch den $fdir_1$ Parameter von Gazebo setzen. Dieser gibt an in welche Richtung des aktuellen Gelenks(Joints) der μ_1 Parameter zeigen soll.

In Abbildung 4.1 (a) ist die einfachste Modellierungs-Möglichkeit eines Ballbots dargestellt. Abbildung 4.1 (b) zeigt, wie man die Reibungskoeffizienten μ_1 und μ_2 sowie den $fdir_1$ Parameter für dieses Rad einstellen müsste, um unendlich viele freie Räder zu simulieren. Leider

¹ Das Robot Operating System (ROS) ist eine Open Source Middleware. Diese ermöglicht es die verschiedenen Komponenten eines Roboters (Sensoren und Aktoren) geschickt miteinander zu verbinden. So ist es möglich einen Roboter zu simulieren und mittels eines UP-Level Computers zu steuern.

Tabelle 4.1.: Unterschiede zwischen den 3D Simulatoren V-REP und Gazebo.

Kriterium	Gazebo-Simulator	V-Rep-Simulator
Lizenz	Open Source Programm	Kommerzielle und kostenlose Version
ROS Integration	Gazebo ist der Standard Simulator von ROS. Gazebo wird als ein ROS Node behandelt und kann daher sehr gut in ROS integriert werden.	V-REP hat keine direkte Anbindung an ROS. Es läuft neben ROS in einem extra Terminal. Jedoch existiert ein Plugin mit dem auf ROS Topics und Services zugegriffen werden kann.
Plugins für Sensoren	Gazebo stellt bereits einige Plugins für Kameras, Laser Scanner etc. bereit. Diese können in einem xml file definiert werden.	V-REP stellt eine sehr benutzerfreundliche graphische Methode zur Verfügung um ein Modell mit Sensoren auszustatten.
CPU Auslastung	Gazebo lastet die Hardware sehr stark aus und ist bis zu 20% langsamer als V-REP.	V-REP hat im Gegensatz zu Gazebo eine konstante CPU Auslastung beim Zugriff auf ROS Nodes.
Community	Gazebo hat eine riesige Community die sehr viele Plugins für neue Sensoren selbst entwickelt und zur Verfügung stellt.	V-REP ist nicht ganz so bekannt und hat lediglich 2570(01.2018) Forenmitglieder. Gazebo hat dagegen 3200.

hat diese einfache Modellierungs-Möglichkeit beim Testen nicht funktioniert, da in Gazebo7 der `fdir1` Parameter nicht richtig funktioniert [16].²

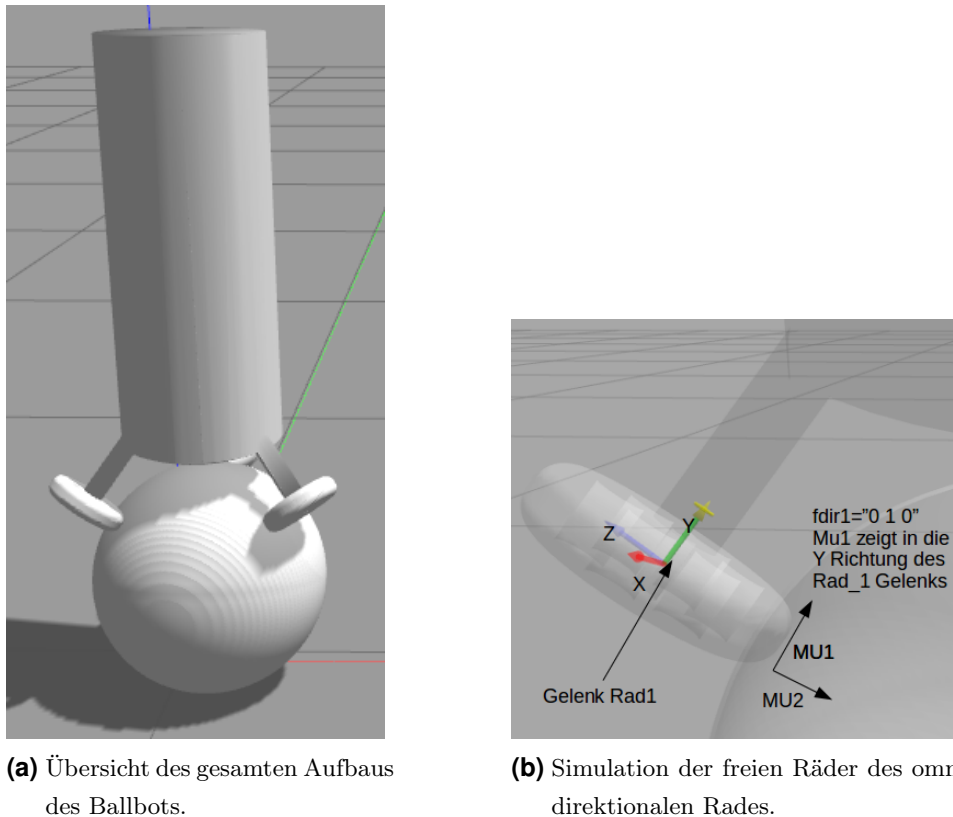


Abbildung 4.1.: Die einfachste Simulations Möglichkeit eines Ballbots in Gazebo7.

Die zweite Möglichkeit ist sehr viel aufwendiger, denn sie besteht darin, das echte omnidirektionale Rad mit allen kleinen Rädern zu simulieren. Hierfür muss zunächst das omnidirektionale Rad ohne die frei laufenden kleine Subräder in die Gazebo Simulation geladen werden. Anschließend müssen die Subräder mit richtiger Orientierung und Position ebenfalls in die Simulation geladen werden. Das omnidirektionale Rad sowie ein einzelnes Subrad wurde hierfür zunächst in SolidEdge konstruiert und anschließend als `.stl` exportiert. Diese `.stl` Dateien werden dann mittels einer `.xml` Datei in die Gazebo-Simulation geladen.

Für die finale Ballbot-Simulation wurde die zweite Möglichkeit benutzt, denn sie ist der Realität sehr viel näher als die Modellierung von unendlich vielen kleinen Subrädern. Nachteilig bei der Simulation aller 30 Subräder ist jedoch der deutlich größere Berechnungsaufwand, der die Simulation auf einen RealTime Faktor von 0.2 verlangsamt.

² Seit Gazebo8 sollte der `fdir1` Parameter wieder richtig funktionieren. Dies konnte jedoch in dieser Arbeit nicht weiter betrachtet werden.

4.3 Simulations Aufbau

Dieses Kapitel zeigt zunächst exemplarisch die Kommunikation zwischen ROS und Gazebo. Anschließend wird näher auf die verwendeten Plugins eingegangen. Zudem wird auf dynamische Eigenschaften der Simulation wie etwa die Steifigkeit einzelner Elemente eingegangen. Zum Schluss wird noch auf zwei Visualisierungsprogramme (RVIZ und PlotJuggler) eingegangen.

4.3.1 Kommunikation zwischen ROS und Gazebo

Die Simulation mittels Gazebo wurde komplett in ROS aufgebaut. Sie besteht aus mehreren Teilprogrammen(Nodes), die alle durch ein globales ROS launch file gestartet werden. Beim Ausführen der Simulation sind sehr viele Nodes aktiv, die untereinander über sogenannte Topics Nachrichten austauschen.

Anhang C zeigt den sogenannten ROS Graph der aktiven Nodes und deren Topics nach dem Starten des globalen ROS launch files. In der Mitte dieser Abbildung ist der Node `/ballbot/bb_control` zu erkennen. Dieser beinhaltet das Regelungsprogramm des simulierten Ballbots. Hierfür liest(subscribt) es Nachrichten der Topics `/ballbot/joints/joint_states`³ und `/ballbot/sensor/imu` ein, berechnet damit die entsprechenden Drehmomente für die einzelnen Räder und veröffentlicht(published) die Nachrichten mit den berechneten Drehmomenten auf den Topics `/ballbot/wheelx_effort_controller/command`. Der Node `/gazebo` wiederum liest diese Drehmomentbefehle der einzelnen omnidirektionalen Räder ein und dreht entsprechend in der sichtbaren Simulation die Räder.

Es sei noch darauf hingewiesen, dass es möglich ist, Gazebo mit ROS zu synchronisieren. Möchte man zum Beispiel einen Regler implementieren, der eine sehr hohe Updatefrequenz (bzw. eine geringe Abtastzeit) aufweist, die Berechnung der Verstärkungsfaktoren dieser Regelung jedoch länger dauert als die Abtastzeit, so muss die Regelung mit Gazebo synchronisiert werden. Hierfür gibt es die Möglichkeit, Gazebo pausiert zu starten und auch die ganze Zeit pausiert zu lassen. Ist nun die Berechnung der Regelung fertig, schickt man einen rosservice call an Gazebo um die Simulation einen Schritt weiterlaufen zu lassen. Anschließend wird der nächste Regelungswert berechnet. Bei dem simulierten Ballbot wurde eine Update Frequenz von 100 Hz benutzt. Die Verstärkungsfaktoren der 2D-Regelung wurden jedoch mit mindestens 1000 Hz berechnet. Daher musste Gazebo nicht mit dem entsprechendem ROS Node, der die Regelung darstellt `/ballbot_controll` synchronisiert werden.

³ Dieses Topic enthält eine Nachricht die die aktuellen Rad Drehmomente, Geschwindigkeiten [rad/sec] und deren absolute Positionen enthält. Für die Regelung der Odometrie werden jedoch nur die Rad Drehgeschwindigkeiten verwendet.

4.3.2 Plugins der Simulation

Die Sensoren des Ballbots können in Gazebo durch Plugins simuliert werden. Abbildung 4.2 zeigt die simulierten Sensoren und deren Plugin Bibliotheken. Hierbei wurde die IMU mit einer `update_rate` von 200Hz sowie mit einem Gaußschen Rauschen von $0.01^\circ/\sqrt{\text{Hz}}$ simuliert. Als Motoren-Interface wurde das Joint-Effort-Interface verwendet, welches über das `ros_control` Paket⁴ zur Verfügung steht. Die Motoren wurden dabei mit einem PID-Regler simuliert. Dabei wurden die Verstärkungsfaktoren standardmäßig zu $P=100$, $I=0.01$ und $D=10$ gewählt. Für weitere Details zur Implementierung der RealSense-Kamera sowie des LDS-Laser Scanners sei auf den Programmcode [13] verwiesen.

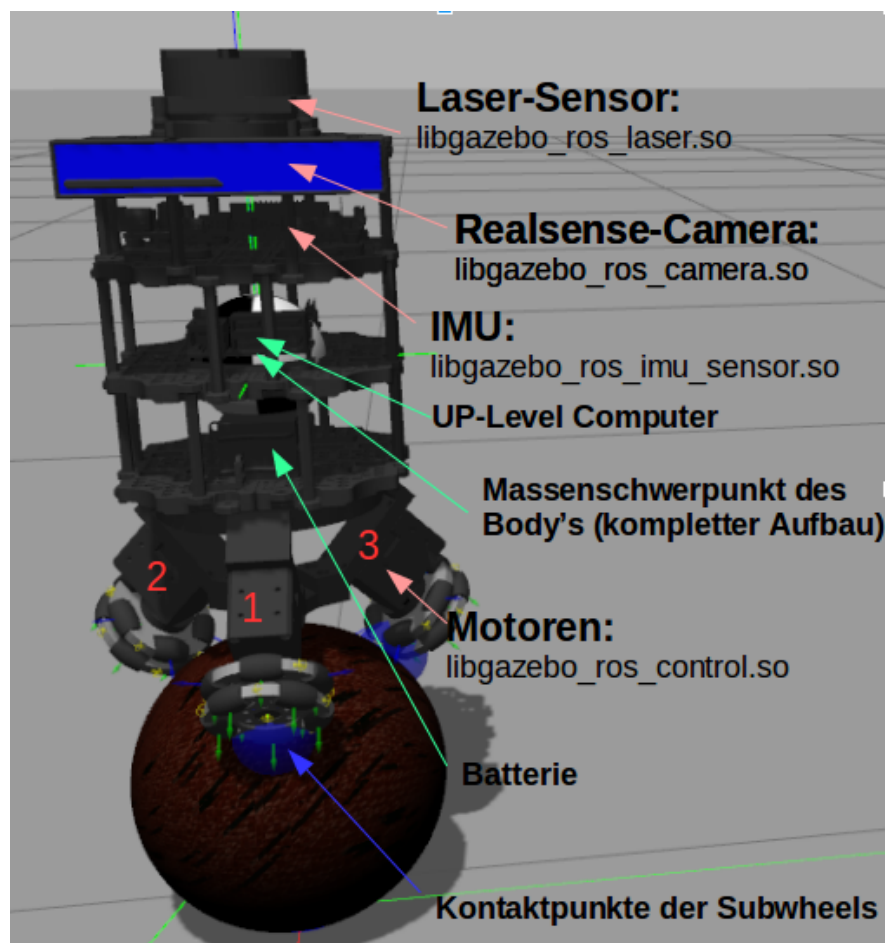


Abbildung 4.2.: Der simulierte Ballbot mit den simulierten Sensoren und deren Plugin-Bibliotheken.

4.3.3 Simulierte Dynamik-Eigenschaften

In Gazebo kann man für jedes Gelenk(Joint) ein maximales Drehmomentlimit und ein maximales Geschwindigkeitslimit setzen. Dieses maximale Drehmomentlimit kann für die verwendeten

⁴ Dieses Paket ist standardmäßig in der ROS-Kinetic Version enthalten. Weitere Informationen zu diesem Paket gibt es unter: http://wiki.ros.org/ros_control.

Tabelle 4.2.: Dynamische Parameter der Bindeglieder der Gazebo Simulation.

Parameter Name	Beschreibung	Beispiel
mu1	Der erste Coulombsche Reibungskoeffizient.	0.6 (Holz)
mu2	Der zweite Coulombsche Reibungskoeffizient.	0.7 (Gummi)
kp	Steifigkeit im Kontaktpunkt, festlegt gemäß Open Dynamics Engine (ODE) für Festkörper-Kontaktpunkte.	$1+e5\dots 1+e15$
kd	Dämpfungskonstante im Kontaktpunkt, festlegt gemäß ODE für Festkörper-Kontaktpunkte.	1...100

Dynamixel XM430-W350-R Motoren aus dem Datenblatt [6] entnommen werden. Betrachtet man zusätzlich, dass eine Batterie von 11.1 Volt verwendet wurde, so beträgt das maximale Drehmomentenlimit 3.8Nm und das maximale Geschwindigkeitslimit⁵ 4.5 rad/s. Diese beiden Begrenzungswerte wurden ebenfalls in der Simulation verwendet.

Für jedes Bindeglied(link) kann man in Gazebo dynamische Eigenschaften festlegen⁶. Die wichtigsten dieser dimensionsloser Größen sind in Tabelle 4.2 aufgeführt.

Diese dynamischen Größen müssen speziell für den Ball, die Subwheels (die kleinen frei laufenden Räder der omnidirektionalen Räder) sowie den Untergrund festgelegt werden. Das Einstellen dieser Größen gestaltet sich als sehr kompliziert, da Gazebo diese Parameter nicht ausführlich dokumentiert hat. In [17] heißt es lediglich, dass von zwei zusammenstoßenden Objekten immer der kleinste Coulombsche Reibungskoeffizient benutzt wird. Aus diesem Grund wurden die mu1 und mu2 Konstanten für die Subwheels, den Ball und den Untergrund alle auf den gleichen Wert von 0.7 gesetzt. Dieser Wert entspricht einem Reibkoeffizient von Asphalt oder Gummi.

In den Annahmen (siehe Kapitel 3) wurde erläutert, dass der Ball optimalerweise so steif wie möglich sein sollte. Aus diesem Grund wurde der *kp* Parameter des Balls auf $1 + e15$ und der *kd* Parameter zu 50 gewählt.

Die dynamischen Parameter *kp* und *kd* der Subwheels wurden basierend experimentell gewonnenen Erkenntnissen zu $kp_{subwheel} = 1 + e10$ und $kd_{subwheel} = 5$ gewählt. Dies entspricht einem relativ hartem und nur sehr gering dämpfendem Rad.

4.3.4 Visualisierungsprogramme

Zum testen der Simulation ist es wichtig bestimmte Werte (wie zum Beispiel die Winkel der IMU) live darstellen zu können. Hierfür stehen verschiedene Programme zur Verfügung:

⁵ Dieses Geschwindigkeitslimit konnte zusätzlich durch eine Messung bestätigt werden.

⁶ Gazebo stellt verschiedene Physics Engines zur Verfügung. Je nach Physics Engine haben die dynamischen Parameter unterschiedliche Namen. In diesem Projekt wurde die Standard Physics Engine der Open Dynamics Engine(ODE) verwendet.

1. Das Standardprogramm von ROS namens `rqt_plot` [18].
2. `Rqt_Multiplot` [19], ein bedienerfreundliches Paket der ETHZ zum Darstellen mehrerer Topics.
3. `PlotJuggler` [20], ein sehr einfach zu bedienendes Paket, mit dem man sehr schnell live Daten plotten kann.

In dieser Arbeit hat sich `PlotJuggler` als das beste dieser drei Programme herausgestellt, da es die Daten am schnellsten plotten konnte und dabei im Gegensatz zu `Rqt_Multiplot` nie abgestürzt ist. Abbildung 4.3 zeigt die Auswertung der gewünschten und tatsächlichen Drehmomente sowie der Winkel der IMU mittels `PlotJuggler`. In diesem Experiment wurde die maximale Geschwindigkeit der Motoren (vgl. Kapitel 4.3.3) auf 2 rad/s begrenzt. Setzt man die maximale Geschwindigkeit der Motoren auf 4.5 rad/s , wie in Abbildung 4.4 dargestellt, balanciert der Roboter. Anhand dieses Experiments konnte simulativ gezeigt werden, dass eine Begrenzung der maximalen Motordrehzahlen ein instabiles Verhalten des Ballbots zur Folge hat.

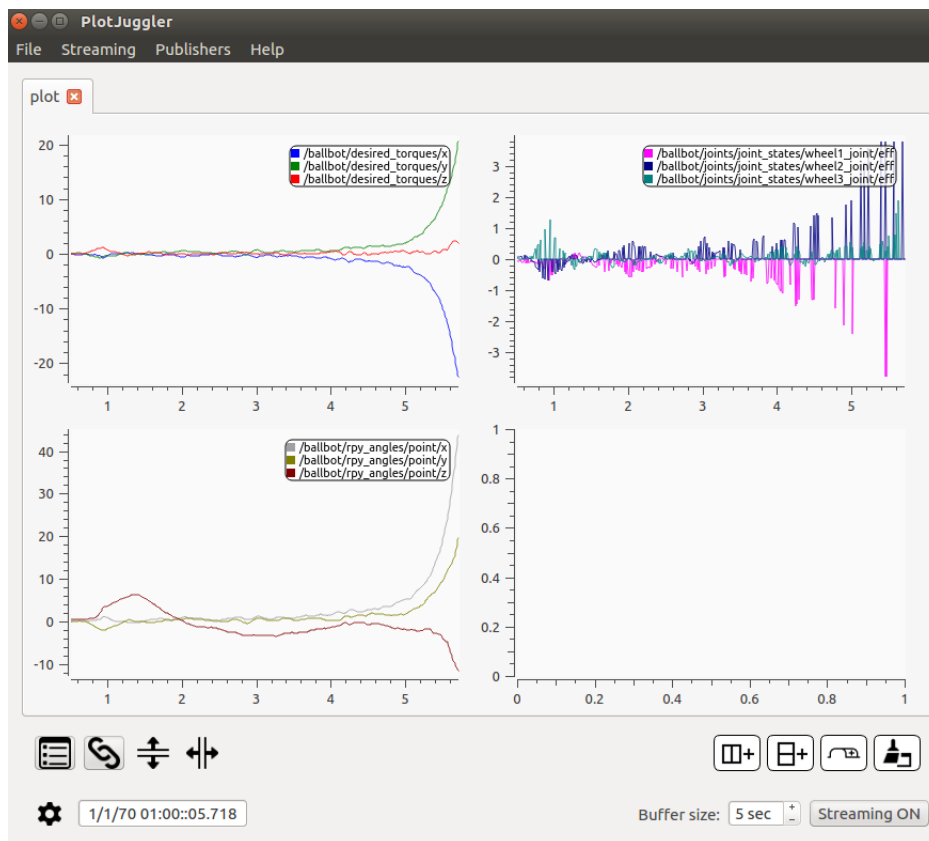


Abbildung 4.3.: Auswertung eines fallenden Ballbots mit `PlotJuggler`. Dargestellt sind: Oben links die gewünschten Drehmomente [Nm], oben rechts die tatsächlichen Drehmomente [Nm] und unten links die Winkel der IMU [°]. Für diesen Plot wurde das Geschwindigkeitslimit der Motoren auf 2.0 rad/s begrenzt.

Neben `Gazebo` gibt es ein weiteres nützliches 3D-Visualisierungs Programm namens `RVIZ`[21]. `RVIZ` bietet die Möglichkeit, Sensoren, wie zum Beispiel Kameras oder Laser-Sensoren, zu vi-

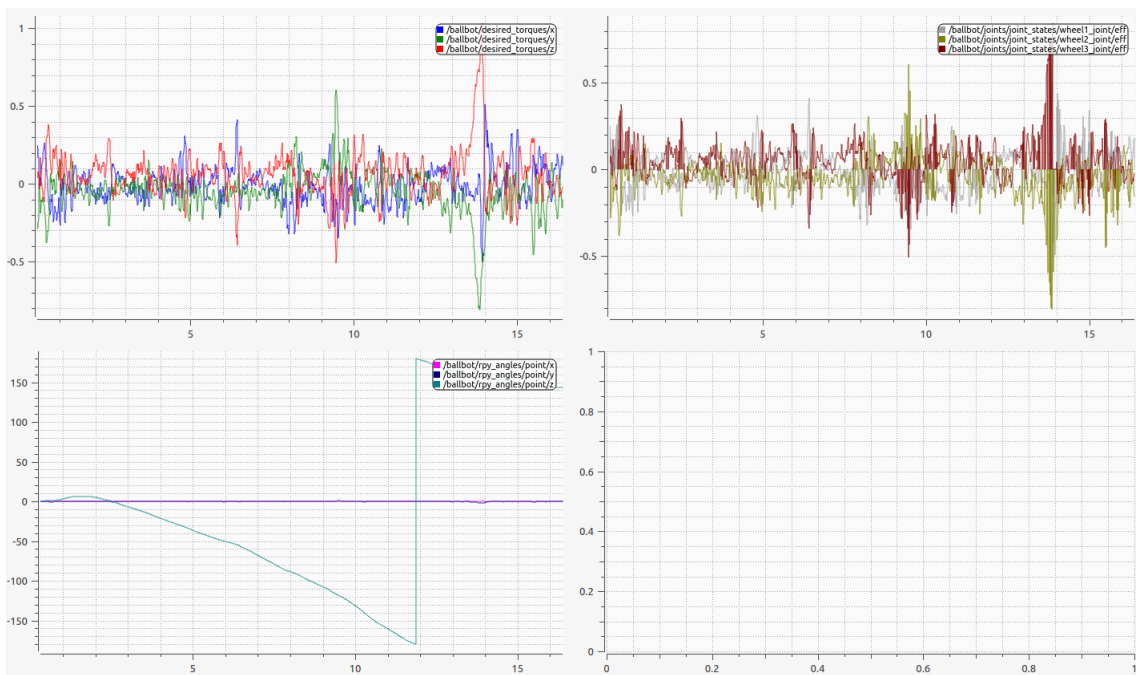


Abbildung 4.4.: Auswertung eines balancierenden Ballbots mit PlotJuggler. Dargestellt sind: Oben links die gewünschten Drehmomente [Nm], oben rechts die tatsächlichen Drehmomente [Nm] und unten links die Winkel der IMU [°]. Für diesen Plot wurde das Geschwindigkeitslimit der Motoren auf 4.5 rad/s begrenzt. Gut zu sehen ist in der Abbildung links unten, dass sich der Ball um die eigene Achse dreht, denn es wurde keine Regelung um die z-Achse implementiert.

sualisieren. Zudem lassen sich in RVIZ die Links mit Namen anzeigen und bietet so eine Möglichkeit, den Roboteraufbau näher zu betrachten. Abbildung 4.3 zeigt den Ballbot dargestellt mittels RVIZ. Zu beachten ist hierbei, dass RVIZ lediglich die Nachrichten von Gazebo bzw. ROS liest und anhand dieser Daten den Roboter mittels Gazebo simuliert. Um in RVIZ den Roboter und den Ball anzuzeigen, ist es nötig, zusätzlich eine Transformations-Nachricht (/tf) zu veröffentlichen. Diese stellt einen Bezug zwischen dem Fixed Frame world und dem Hauptlink des Balls bzw. des Ballbots dar⁷.

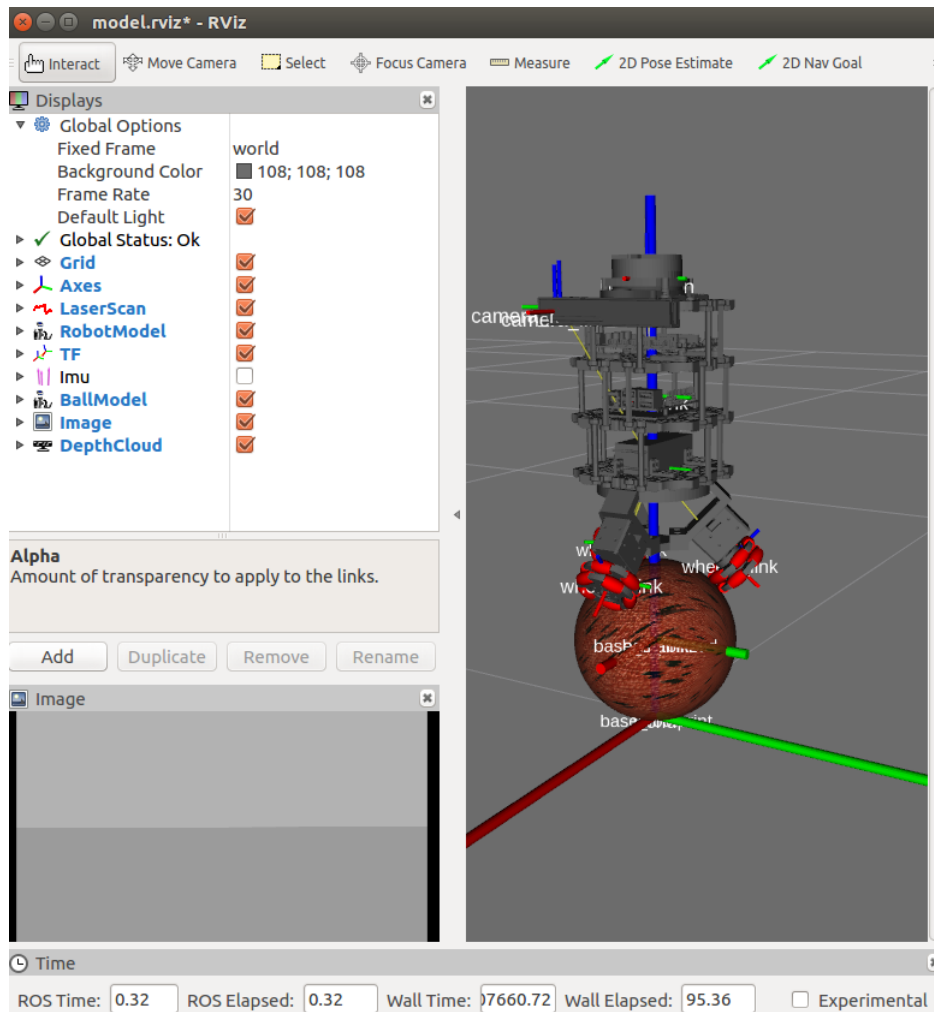
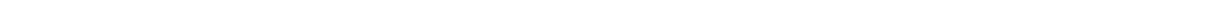


Abbildung 4.5.: Darstellung des Ballbots mittels RVIZ.

Des Weiteren gibt es für RVIZ nützliche Plugins wie zum Beispiel MoveIt [22]. Dieses Plugin bietet die Möglichkeit der Planung von Trajektorien, die ein Roboter abfahren soll. Im Rahmen dieser Arbeit konnte dieses Plugin noch nicht getestet werden, stellt aber für eine zukünftige Positionsplanung des Ballbots eine sehr gute Möglichkeit dar.

⁷ Das /tf Topic sowie die nodes /odom_to_tf_robot und /odom_to_tf_ball sind in Anhang C dargestellt.



5 Implementierung

In diesem Kapitel wird zunächst die eingesetzte Entwicklungsumgebung für die Programmierung des Regelgesetzes vorgestellt. Daraufhin wird die allgemeine Vorgehensweise der praktischen Umsetzung der Implementierung des Ballbot's erläutert.

5.1 Entwicklungsumgebung

Als Entwicklungsumgebung für den Zustandsregler des Ballbot's wird die ArduinoIDE ¹ eingesetzt, die in Abbildung 5.1 abgebildet ist. Die Hauptbestandteile dieser Entwicklungsumgebung umfassen einen Texteditor für das eigentliche Programm, eine Konsole für Fehler- und Kompilierungsmeldungen und einen seriellen Monitor für das Senden und Empfangen von Daten zwischen Computer und Board. Weiterhin eignet sich die Entwicklungsumgebung für den Einsatz, da ab der Version 1.6.4 das eingesetzte OpenCR-Board unterstützt wird. Allerdings muss die dafür nötige Bibliothek über den Boardverwalter nachinstalliert werden [5] [23].

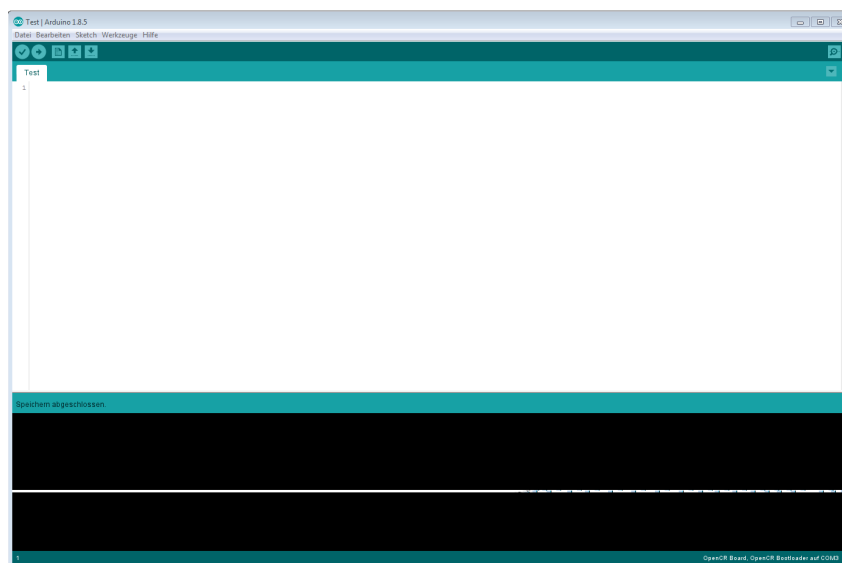


Abbildung 5.1.: Arduino Entwicklungsumgebung

Die Grundstruktur eines Programms, die im Listing 5.1 dargestellt ist, gliedert sich in zwei Bereiche. In der Setup-Funktionen, die nur einmal beim Start des Board ausgeführt wird, werden verschiedenen Initialisierungen für den späteren Programmablauf festgelegt. Bei der Loop-Funktion handelt es sich um das eigentliche Programm, die als endlose Schleife durchlaufen wird. Werden für das Programm noch weitere Funktionen bzw. Variablen benötigt, müssen diese am Anfang oder am Ende eines Programms definiert werden [24].

¹ <https://www.arduino.cc/en/Main/Software>

Listing 5.1: Grundstruktur eines Arduino-Programmes[24]

```
void setup()    {
    // Diese Funktion wird nur beim Starten
    // des entsprechen Boards einmal ausgeführt
}

void loop() {
    // Diese Funktion in einer endlosen Schleife
    // durchlaufen
}
```

Um die Übersichtlichkeit zu steigern, können Funktionen und Klassendefinitionen in separaten Dateien ausgelagert werden und im Hauptprogramm eingebunden werden.

5.2 Hauptprogramm

Die benötigten Daten für die Regelung werden nach dem EVA-Prinzip (Eingabe, Verarbeitung, Ausgabe) in einem Interrupt verarbeitet. Das bedeutet, dass das OpenCR-Board die benötigten Sensordaten des Gyroskops empfängt, entsprechend dem Regelgesetz aus Gl. 3.19 verarbeitet und die entsprechenden Stellgrößen an die einzelnen Motoren weiterleitet. Im Hauptprogramm sind folgende Bibliotheken aus der OpenCR-Bibliothek inkludiert worden:

- IMU.h : Schnittstelle zur IMU.
- DynamixelSDK.h : Schnittstelle zu den Motoren.

Nachfolgend wird das Hauptprogramm näher erläutert.

5.2.1 Initialisierung Komponenten

Für die Implementierung der Zustandsregelung müssen zunächst die inertielle Messeinheit, die Motoren und der Hardware-Interrupt initialisiert werden. Hierzu wird die Setup-Funktion benutzt.

Bei der Initialisierung der IMU bzw. des gesamten Systems muss darauf geachtet werden, die richtige Update-Frequenz festzulegen bzw. zu übergeben, damit das Gyroskop in möglichst kurzen Zeitabständen aktualisierte Daten zur Verfügung stellt und das System die aktuellen Stellgrößen berechnen kann.

Da der Ballbot seine Gleichgewichtslage durch das Aufbringen von entsprechenden Drehmomenten auf den Ball versucht zu halten, stellen diese gleichzeitig die Stellgröße des Systems dar. Daher müssen die drei Motoren zu Beginn auf die stromgeregelte Betriebsart konfiguriert und für den Einsatz freigegeben werden. Die entsprechenden Adressen und Werte sind dem Datenblatt [6] zu entnehmen.

Damit die Berechnung des Regelgesetzes nach Gl. 3.19 zu den Zeitpunkten der Aktualisierungsfrequenz durchgeführt werden kann, wird ein Hardware-Timer initialisiert, der als Interrupt arbeitet und in festgelegten Zeitabständen das Hauptprogramm des Mikrocontrollers unterbricht und einen Interrupt (Ausnahmeroutine) aufruft. Die auszuführenden Funktionen der Routine sind für das Einlesen der Daten und die Berechnung der Stellgröße zuständig. Die Abbildung 5.2 zeigt die Funktionsweise eines Interrupts.

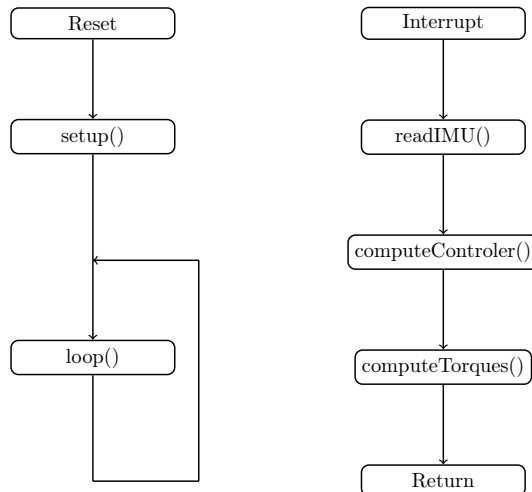


Abbildung 5.2.: Links: Hauptprogramm. Rechts: Aufrufen und Abarbeiten der Serviceroutine

5.2.2 Einlesen der Sensordaten

Der Zugriff auf die Winkel und Winkelgeschwindigkeiten wird durch das Einbinden der Bibliothek der inertialen Messeinheit hergestellt.

Für die weitere Datenverarbeitung müssen die Winkel und Winkelgeschwindigkeiten, die in der Einheit Grad und Umdrehungen pro Minute angegeben sind, mit einer entsprechenden Funktion in die Einheit rad und rad/s konvertiert werden. Die Abbildung 5.3 zeigt die Winkel ϑ_x und die entsprechenden Winkelgeschwindigkeiten $\dot{\vartheta}_x$. Aufgrund der höheren Übersichtlichkeit sind diese in Grad und Grad/s angegeben. Zudem ist zu erkennen, dass die Winkel mit einem Rauschen überlagert sind. Zu beachten ist, dass die eingebundene IMU Bibliothek (IMU.h) intern einen MadgwickAHRS-Filter² verwendet. Dieser wurde so übernommen. Abbildung 5.3 zeigt eine beispielhafte Ausgabe dieses Filters bei aufrechter Ruhelage des Roboters.

Um das Rauschen zu verringern, werden die letzten drei Sensorwerte der Winkel und Winkelgeschwindigkeiten gespeichert, aufsummiert und davon der Mittelwert gebildet. Das Ergebnis ist in Abbildung 5.4 zu sehen.

² Für weitere Informationen siehe: <https://github.com/arduino-libraries/MadgwickAHRS>.

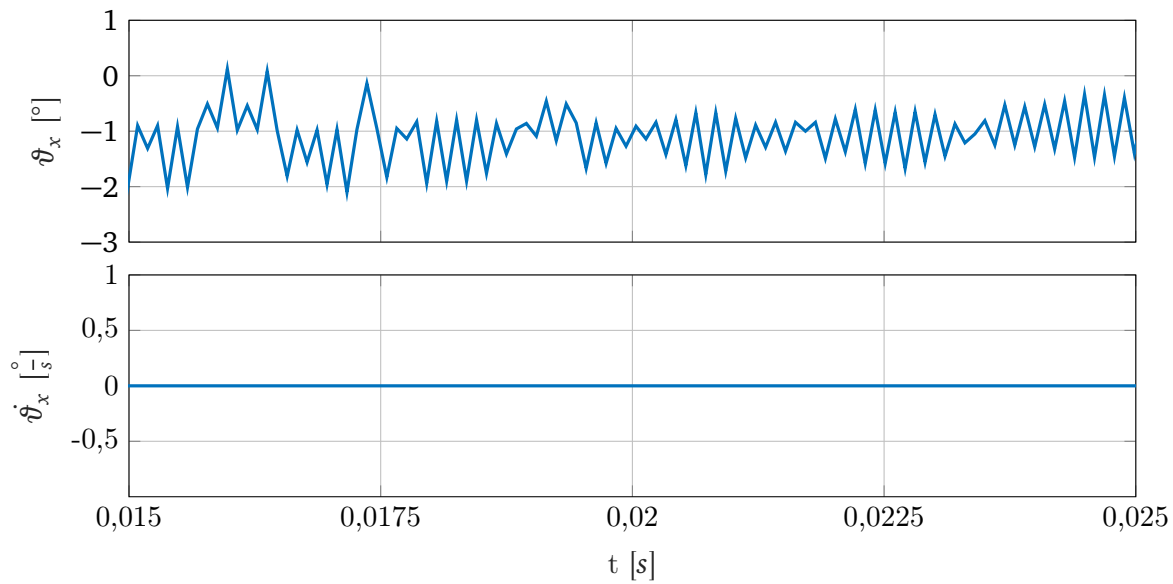


Abbildung 5.3.: Darstellung der Winkel ϑ_x und Winkelgeschwindigkeit $\dot{\vartheta}_x$

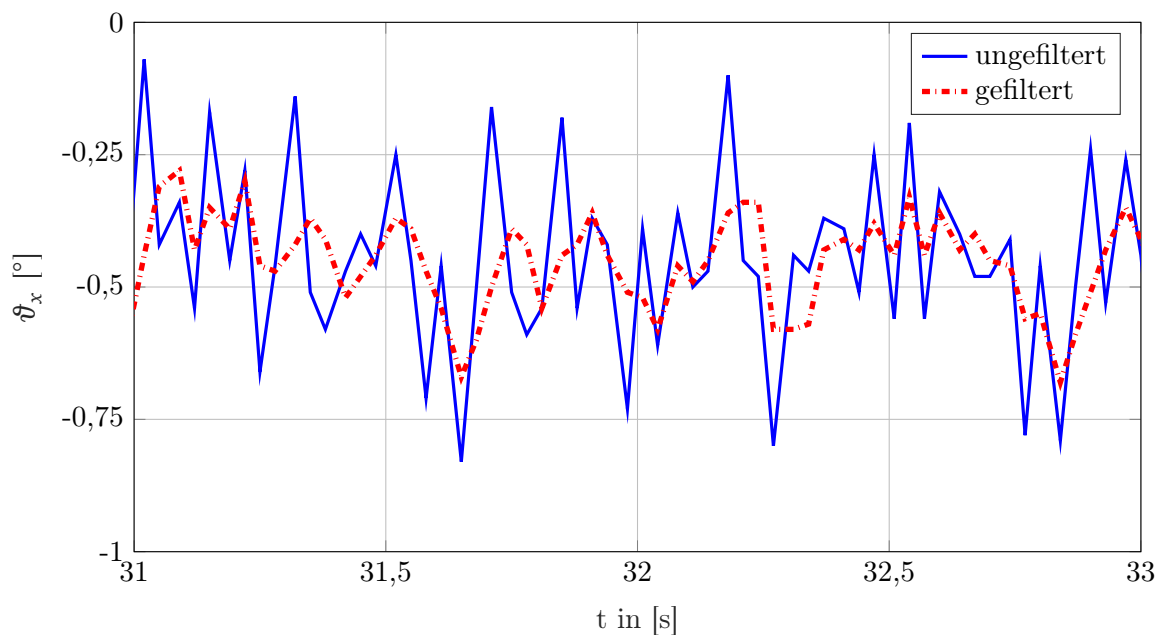


Abbildung 5.4.: Beispielhafter Vergleich zwischen ungefilterten(blau) und gefilterten Messwinkel

5.2.3 Verarbeitung Sensordaten

Mit den ausgelesenen Sensordaten können die einzelnen virtuellen Drehmomente für die entsprechenden Ebenen berechnet werden. Das virtuelle Drehmoment für die xy -Ebene wird auf Null gesetzt, da diese Orientierung um die z -Achse für die Regelung nicht berücksichtigt wird. Mit den berechneten, virtuellen Drehmomenten kann auf die realen Drehmomente für die einzelnen Motoren nach Gl. 3.9 umgerechnet werden. Damit der Zeitaufwand für Berechnung

möglichst klein ist, werden konstante, mathematische Operationen durch feste Zahlenwerte festgelegt. Das Listing 5.2 zeigt einen Ausschnitt dieser Definitionen.

Listing 5.2: Zuweisen fester Zahlenwerte für eine kürzerer Berechnungsdauer

```
#define ALPHA          PI/4
#define BETA           0
#define COS_ALPHA     0.70710678118
#define SIN_ALPHA     0.70710678118
#define SIN_BETA      0.0
#define COS_BETA      1.0
#define SQRT3         1.73205080757
```

5.2.4 Ausgabe Drehmomente

Das OpenCR-Board kommuniziert mit den Motoren über die RS485-Schnittstelle. Daher müssen für die Übertragung die berechneten Drehmomente in äquivalente, dimensionslose Einheiten (Unit) umgerechnet werden. Diese Umrechnungskonstanten sind experimentell im Kapitel 2.2.3 bestimmt worden.

Da in der Modellbildung die Reibung zwischen Motor und Ball vernachlässigt wurde, muss ein Offset der Drehmomente eingestellt werden, damit diese Reibung überwunden werden kann [1].

5.3 Auswertung

Nach der Implementierung des Mikrocontroller-Programms wurde die Regelung, wie in Kapitel 3 beschrieben, am realen Ballbot getestet. Abbildung 5.5 zeigt den Aufbau des Ballbots wie er auf dem Ball balanciert. Für dieses balancierende Experiment wurden zusätzlich die Winkel der IMU und die realen Drehmomente der Motoren aufgenommen. Abbildung 5.7 zeigt, dass die Motorendrehmomente nach Ausregelung einer kleinen Anfangsstörung³, zwischen -0.7 Nm und $+0.7\text{ Nm}$ pendeln. Anhand der Winkel ϑ_x, ϑ_y in Abbildung 5.6 sieht man, dass eine Anfangsauslenkung von 5° ausgeglichen werden konnte. Balanciert der Roboter so wurde ein maximaler Winkel von $|\vartheta|_{max} 2^\circ$ nicht überschritten. Die Standardabweichungen der Auslenkungswinkel des balancierenden Roboters lagen bei: $\sigma_{\vartheta_x} = 1.19^\circ$ und $\sigma_{\vartheta_y} = 1.11^\circ$. Die Standardabweichungen der Motoren lagen bei $\sigma_{T_1} = 0.24\text{ Nm}$ $\sigma_{T_2} = 0.23\text{ Nm}$ und $\sigma_{T_3} = 0.28\text{ Nm}$. Insgesamt hat sich ein stabiles Verhalten ergeben.

Das Regelverhalten weist jedoch Grenzen auf. Weitere Optimierungen für ein robusteres Regelverhalten sind:

- Ball: Es konnte zwar ein Ball gefunden werden, der einen guten Kompromiss(vgl. Kapitel 2.3.3) zwischen den gewünschten Eigenschaften bietet, allerdings könnte dieser Kompro-

³ Die kleine Anfangsstörung hat die Ursache, dass der Ballbot nicht perfekt vertikal auf dem Ball platziert wurde.



Abbildung 5.5.: Aufbau des realen Ballbots.

miss noch weiter verbessert werden. So kann zum Beispiel mit einer maßgenauen Aluminiumhohlkugel mit Gummibeschichtung die nötige notwendige Steifigkeit und den Reibwert bereitstellen.

- **Filterung des Messdaten:** Auch die Filterung hat einen großen Einfluss auf die Regelgüte. Im bestehenden Ballbot ist ein einfacher Mittelwertfilter verwendet worden. Dadurch konnte das Rauschen in einigen Fällen schon um den Faktor 2 reduziert werden, jedoch ist das Rauschen weiterhin im Regelverhalten zu spüren. Eine Verbesserung in diesem Verhalten könnte durch ein besseres Filter beispielsweise ein Kalmanfilter erzielt werden. Dies hätte zudem den Vorteil, dass kein Zeitverzug entstehen würde.
- **Modellierung:** Auch bei der Modellierung des Ballbots sind Vereinfachungen getroffen worden. Es hat sich gezeigt, dass die Vereinfachung zulässig sind und eine zweckmäßige Regelung implementiert werden kann. Allerdings werden damit Verkopplungen zwischen der xz - und yz -Ebene vernachlässigt. Diese sind zwar relativ klein, bei Berücksichtigung dieser Verkopplungen könnte jedoch sicherlich ein noch besseres Regelverhalten erzielt werden. Dies könnte durch eine 3D-Regelung erzielen werden.
- **Motoren:** Die im Projekt verwendeten Motoren weisen Drehzahlbegrenzungen auf. Sowohl Experimente als auch Simulation konnten zeigen, dass das System nicht wie vorgesehen stabilisiert werden kann. Durch die Verwendung anderer Motoren, die eine höhere Drehzahlbegrenzung aufweisen kann dieser Effekt umgangen werden.

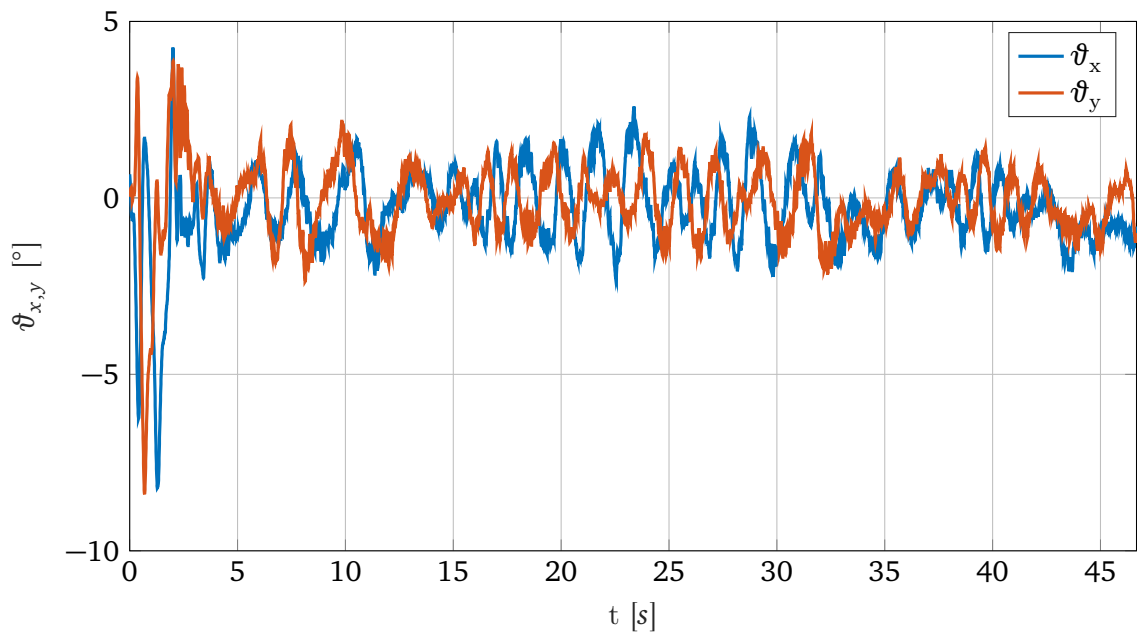


Abbildung 5.6.: Darstellung der Winkel ϑ_x und Winkelgeschwindigkeit $\dot{\vartheta}_x$ des balancierenden Ballbots.

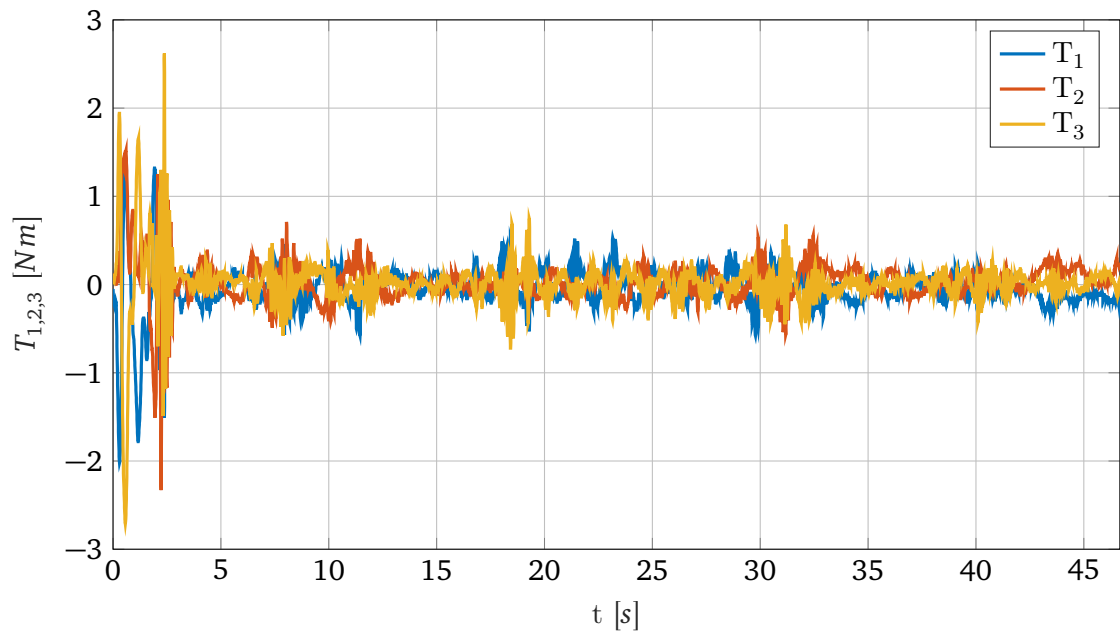
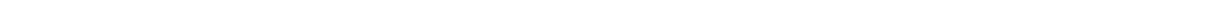


Abbildung 5.7.: Darstellung der Drehmomente des balancierenden Ballbots.



6 Zusammenfassung

Diese Arbeit behandelt die Implementierung eines balancierenden Ballbot basierend auf einer gegebenen Hardware. Im Verlauf der Arbeit ist dazu die gegebene Hardware durch eine selbst hergestellte Hardware ergänzt, mehrere Simulation mit Gazebo sowie MATLAB/Simulink zur Validierung der erzeugten Modelle durchgeführt, sowie eine Regelung für das reale System entworfen worden.

Der zeitliche Ablauf der Arbeit gliedert sich wie folgt. Zu Beginn wurde ein Konzept, basierend auf bereits bestehenden Ballbot-Projekten entworfen. Dieses hat sich dann als eine zylinderförmiger Körper ergeben, der über drei, um 120 Grad verdrehte, omnidirektionale Räder auf einem Ball balancieren soll. Im Anschluss wurde mit dem Entwurf des gesamten Ballbots in SolidEdge begonnen. Dabei wurde die Konstruktion zur Aufnahme der Antriebe entwickelt und mit der basierenden Hardware verknüpft. Neben einer genauen Entwicklung des Aufbaus zur Aufnahme der Antriebe, hatten die Arbeiten mit SolidEdge weiterhin zum Ziel ein detailgetreues Modell der Realität in der Simulationsumgebung Gazebo simulieren zu können. Mittels dieser Simulationsumgebung konnte so im Anschluss an die Konstruktionsarbeiten und des Reglerentwurfs, das Gesamtsystem unter labor- als auch realitätsnahen Bedingungen ausgiebig getestet werden. Dies hat sich für die späteren Test am realen Ballbot als sehr nützlich erwiesen, da man das Verhalten bereits untersuchen und kennenlernen konnte. So konnte gezeigt werden, dass die verwendeten Motoren für die Ballbot Anwendung nur bedingt geeignet sind, da sie nur ein sehr kleines Drehzahlband besitzen.

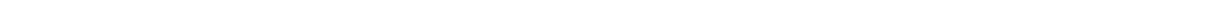
Zeitgleich zum Einlesen in die Simulationsumgebung ist der Regler mit MATLAB/Simulink entworfen worden. Für das Gesamtsystem wurde ein LQR-Zustandsregler, unter der Annahme das reale System durch drei unabhängige planare Ebenen zu approximieren, entworfen. Durch eine mathematische Abbildung des realen Systems, implementiert in Simulink, wurde der entworfene Regler in seiner Wirkungsweise überprüft. So konnte gezeigt werden, dass der Regler in der Lage ist den Ballbot um den betrachteten Arbeitspunkt zu stabilisieren. In Simulink berücksichtigt wurden dabei real auftretende Störungen wie Schlupf und Totzeiten.

Nach diesem Schritt wurde die Regelung auf dem realen System realisiert. Dazu wurde ein Programm auf einem Mikrocontroller erstellt, das die Sensorik ausliest, diese Werte durch das in Kapitel 3 gewonnene Regelgesetz weiterverarbeitet und die Stellgröße schließlich auf die Antriebe gibt. Beim Einlesen der Sensorwerte wurde weiterhin zur Rauschreduzierung ein Filter implementiert. Die Funktionsweise des so implementierten realen Gesamtsystems konnte im Anschluss auf einer dünnen Schaumstoffmatte bestätigt werden.



A Parameterliste

Parameter	Variable	Wert	Quelle
Masse Gesamtaufbau (alles)	---	1,731 kg	Gemessen
Masse Ball	m_S	0,3280 kg	Gemessen
Masse Motor	m_M	0,0820 kg	Datenblatt
Masse omnidirektionales Rad	m_{OW}	0.0520 kg	Gemessen
Masse virtuelles Rad	m_W	0,4020 kg	Gemessen
Masse Roboterkörper (mit Motoren/Räder)	m_B	1,603 kg	Gemessen
Masse Roboterkörper (ohne Motoren/Räder)	m_B	1,2010 kg	Gemessen
Radius Ball	r_S	0,0800 m	Datenblatt
Radius virtuelles Rad	r_W	0,0300 m	Datenblatt
Radius Körper	r_B	0,0703 m	Gemessen
Höhe Massenschwerpunkt	l	0.236 m	SolidEdge
Höhe Körper	h	0.366 m	SolidEdge
Trägheitsmoment Ball	I_S	0,0013 kgm^2	Berechnet
Trägheitsmoment Rotor	I_M	3.8e-8 kgm^2	Datenblatt
Trägheitsmoment omnidirektionales Rad	I_{OW}	2,34e-5 kgm^2	Berechnet
Trägheitsmoment der virtuellen Räder (yz- und xz-Ebene)	$I_{W,yz,xz}$	0.00357 kgm^2	Berechnet
Trägheitsmoment virtuelles Rad (xy-Ebene)	$I_{W,xy}$	0.0143 kgm^2	Berechnet
Trägheitsmoment Körper (yz-Ebene)	$I_{B,yz}$	0.0880 kgm^2	SolidEdge
Trägheitsmoment Körper (xz-Ebene)	$I_{B,xz}$	0.0880 kgm^2	SolidEdge
Trägheitsmoment Körper (xy-Ebene)	$I_{B,xy}$	0.0070 kgm^2	SolidEdge
Übersetzungsverhältnis	i	353,5	Datenblatt
Erdbeschleunigung	g	9,81 $\frac{m}{s^2}$	Datenblatt

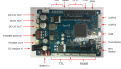




B Bestands-Liste

Tabelle B.1.: Screws:

Type	Size	Amount	Place
Cylinderhead screw	M3 x 11mm	8	Motor mounts
Cylinderhead screw	M2,5 x 22mm	16	Motor plate
Cylinderhead screw	M2 x 6 mm	18	Wheel shaft
Cylinderhead screw	M2,5 x 36 mm (38 mm)	5	Wheel shaft cover
Cylinderhead screw	M3 x 20 mm (21mm)	4	Layer mounting
Nut	M2	5	Layer mounting
Cylinderhead screw	M2,5 x 22mm (23mm)	4	Layer mounting

Tabelle B.2.: Übersicht über die verwendeten Bauteile

Item	#	W.[g]	Weblink	Bild
OpenCR Board (Controlling the motors, IMU)	1	60	github_wiki	
UpBoard (Main PC)	1	96	127€	
Intel RealSense R200	1	9.4	datasheet , 84.15€	

Die Gesamtkosten der verwendeten Bauteile belief sich auf ca. 1200€.

Tabelle B.3.: Übersicht über die verwendeten Bauteile

Item	#	W.[g]	Weblink	Bild
Laser Distance Sensor	1	124	specs, 100€	
Battery: LI-PO 11.1 1800mAh LB-12 19	1	132	44.90€	
Turtlebot3 Layers(125cmx125cm)	4			
XM430-W350-R Dynamixel (Motors)	3	82	robotis,250€	
Ball(alum., dia.: 140mm, material thickness 2.5mm)	1	400	ball-tech gmbh,40€.	
Omni wheels(dia: 60mm, thickness:25mm)	3	51.46	10.38€	
Kreisring (PLA, 3D printeted)	1	28		
Halterung (PLA, 3D printeted)	3	18		
Mitnehmer (PLA, 3D printeted)	3	8		
Plain washer (Beilagscheibe),(PLA, 3D printeted)	3	0.45		
Omni double wheels(dia: 56mm, thickness:25mm)	3	62	15€	
Mitnehmer double wheels	3	7		
Ball(alum., dia.: 140mm, material thickness 2.5mm)	1	400	ball-tech gmbh,40€.	
Ball(gummi, diameter: 150mm)	1	326	link and cost	

C ROS-Graph Simulation

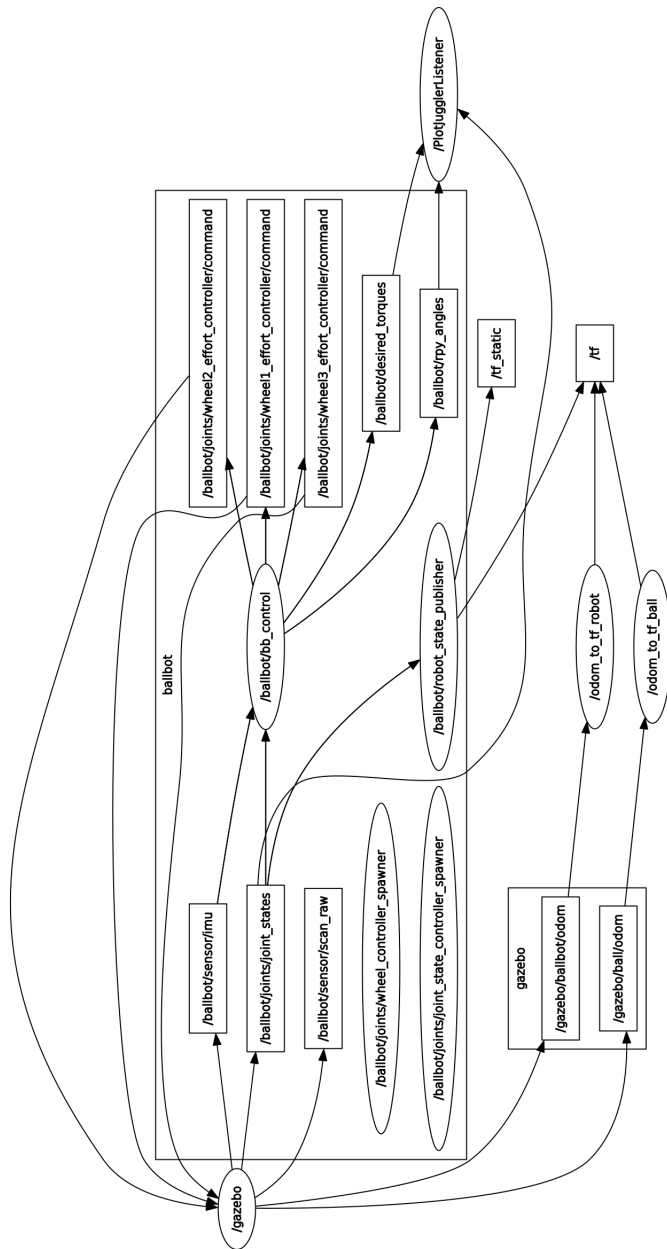
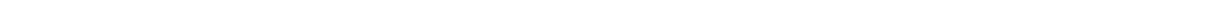


Abbildung C.1.: Übersicht aller Teilprogramme(Nodes) und deren Topics, die beim Starten der Simulation aktiv sind und Nachrichten austauschen. Hierbei sind die Nodes mit Ellipsen und die Topics mit Rechtecken gekennzeichnet. Das Bild wurde mit dem ROS Programm rqt_graph erstellt.



D Simulink Simulationsaufbau

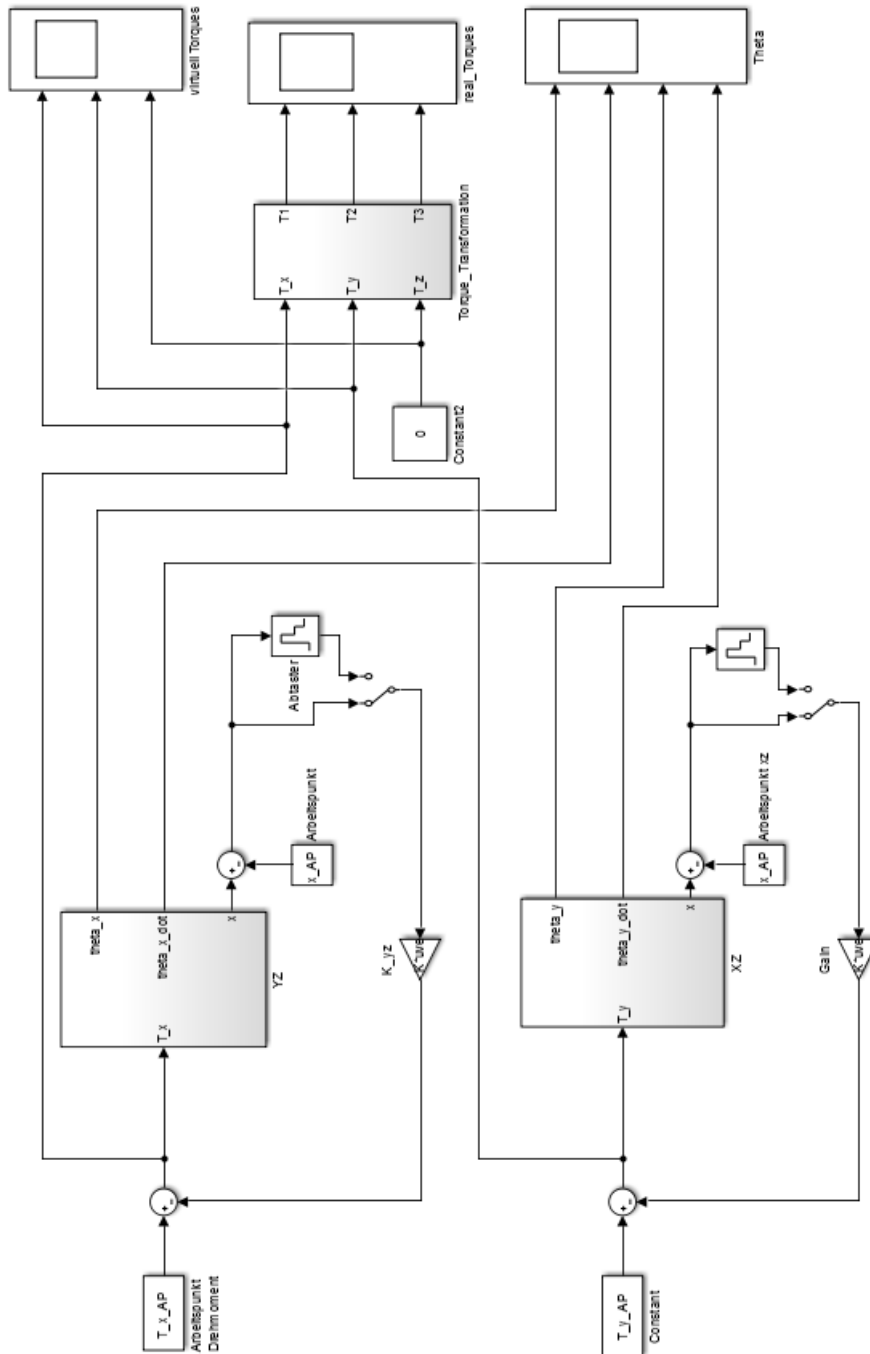


Abbildung D.1.: Übersicht des Simulink Simulationsaufbaus



Literaturverzeichnis

- [1] Fankhauser, Peter und Corsin Gwerder: Modeling and Control of a Ballbot. ETH Zürich, Bachelor Arbeit Auflage, 2010.
- [2] Umashankar Nagarajan, George Kantor, Ralph Hollis: The ballbot: An omnidirectional balancing mobile robot. The International Journal of Robotics Research, 2013.
- [3] Kumaga, M. und T. Ochiai: Development of a robot balanced on a ball: Application of passive motion to transport. In: IEEE International Conference on pages 4106-4111., 2009.
- [4] Buchholz, Dr.-Ing. Michael: Regelung eines inversen Pendels, Institut für Mess-, Regel- und Mikrotechnik, Universität Ulm. 29.10.2010.
- [5] OpenCR Board E-Manual.
<http://emanual.robotis.com/docs/en/parts/controller/opencr10/>. [Zugriff am 06.02.18].
- [6] Dynamixel Motor E-Manual.
http://support.robotis.com/en/product/actuator/dynamixel_x/xm_series/xm430-w350.htm. [Zugriff am 06.02.18].
- [7] Was ist PLA.
<https://www.filamentworld.de/3d-druck-wissen/was-ist-pla/>. [Zugriff am 06.02.18].
- [8] Konigorski, U.: Praktikum MATLAB/Simulink II. TU Darmstadt, 2016/2017.
- [9] Konigorski, U.: Mehrgrößenreglerentwurf im Zustandsraum. TU Darmstadt, 2016/2017.
- [10] Adamy, Jürgen: Systemdynamik und Regelungstechnik II. Shaker Verlag, 2016.
- [11] Ament, Christoph: Regelungstechnik 2. Universität Augsburg, 2017.
- [12] Konigorski, U.: Modellbildung und Simulation. TU Darmstadt, 2017.
- [13] Code des Ballbotprojekts.
<https://github.com/CesMak/bb>. [Zugriff am 06.02.18].
- [14] Installationsanleitung der Simulation des Ballbots.
<https://www.youtube.com/watch?v=a7RVCsyNebY>. [Zugriff am 06.02.18].
- [15] Serena Ivaldi, Vincent Padois, Francesco Nori: Tools for dynamics simulation of robots: a survey based on user feedback. Cornell University, 2014.
- [16] Code für die Ballbot Gazebo Simulation.
<https://bitbucket.org/osrf/gazebo/issues/463/ode-fdir1-parameter-broken>.

-
- [Zugriff am 06.02.18].
- [17] Festlegung der Reibungskoeffizienten in Gazebo.
<http://gazebosim.org/tutorials?tut=friction>. [Zugriff am 06.02.18].
- [18] RQT_Plot.
http://wiki.ros.org/rqt_plot. [Zugriff am 06.02.18].
- [19] RQT Multiplot.
https://github.com/ethz-asl/rqt_multiplot_plugin. [Zugriff am 06.02.18].
- [20] Plot Juggler Github.
<https://github.com/facontidavide/PlotJuggler>. [Zugriff am 06.02.18].
- [21] Rviz Website.
<https://github.com/ros-visualization/rviz>. [Zugriff am 06.02.18].
- [22] MoveIt Dokumentation.
http://docs.ros.org/hydro/api/moveit_ros_visualization/html/doc/tutorial.html. [Zugriff am 06.02.18].
- [23] Arduino Webpage.
<https://www.arduino.cc/en/Guide/Environment>. [Zugriff am 06.02.18].
- [24] Brühlmann, Thomas: Arduino Praxiseinstieg. Mitp, 2010.