Aufbau und Regelung eines Ballbots

Masterarbeit

Technische Universität Graz
Institut für Regelungs- und Automatisierungstechnik

Ivan Sebastiani

Graz, Mai 2013

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..

(Unterschrift)

Kurzfassung

In der folgenden Masterarbeit geht es um die Realisierung eines Ballbots. Unter einem Ballbot kann man sich einen mobilen Roboter vorstellen, der auf einem Ball balanciert bzw. sich fortbewegt.
Ziel dieser Arbeit war es, für diesen Ballbot ein mathematisches Modell zu finden und einen geeigneten Regler zu entwerfen. In einem zweiten Schritt sollte dann der Ballbot mittels des LEGO Mindstorms-Baukastens aufgebaut werden und der zuvor entworfene Regler mittels der MATLAB/Simulink-Software realisiert werden. Dabei wurde für die Modellbildung auf den Lagrange-Formalismus zurückgegriffen und für den Regler wurde ein geeigneter Zustandsregler entworfen.
Abschließend wurde der Ballbot parametrisiert und getestet. Man unterscheidet dabei zwei verschiedene Betriebsarten, eine, in der der Ballbot im Stand balanciert und eine, in der sich der Ballbot auf einem vorgegebenen Weg fortbewegen kann. Durch Realisierung einer Applikation zur Datenauswertung war man dann auch in der Lage, die Funktionsweise des Ballbots am Rechner zu überprüfen.

Abstract

The following master thesis deals about the realization of a ballbot. A ballbot is a mobile robot, which is able to balance or to move on a ball.
First of all the aim of this work was it to find a mathematical model and to design a suitable controller for this ballbot. In a second step the ballbot was built up with the help of the LEGO Mindstorms construction kit but moreover the already well designed controller should be realized by the MATLAB/Simulink software. For modeling the Lagrange formalism was applied and a state controller was used for the controller.
Finally the ballbot was parameterized and tested. There are two different modes of operation to distinguish, the one in which the ballbot is balanced without moving and the other one in which the ballbot moves on a certain given path. Additionally by the realization of a software with a graphical interface for data evaluation, one can prove the functionality of the ballbot on a computer.

Nomenklatur

Formelzeichen für die Modellbildung

Formelzeichen	Bedeutung	Einheit
r_1, r_2	Ortsvektoren	-
l	Abstand Ball Mittelpunkt-Schwerpunkt	m
N	Anzahl der Massenpunkte	-
k	Anzahl der holonomen Zwangsbedingungen	-
n	Anzahl der Freiheitsgrade	-
L	Lagrange Funktion	-
T	Kinetische Energie	-
V	Potentielle Energie	-
q_1, q_2	Generalisierte Koordinaten	-
x_x	Position des Ballbots in x-Richtung	m
y_y	Position des Ballbots in y-Richtung	m
$\dot{x_x}$	Geschwindigkeit des Ballbots in x-Richtung	m s^{-1}
$\dot{y_y}$	Geschwindigkeit des Ballbots in y-Richtung	m s^{-1}
φ_x	Kippwinkel des Ballbots in x-Richtung	°
φ_y	Kippwinkel des Ballbots in y-Richtung	°
$\dot{\varphi_x}$	Kippwinkelgeschwindigkeit des Ballbots in x-Richtung	° s^{-1}
$\dot{\varphi_y}$	Kippwinkelgeschwindigkeit des Ballbots in y-Richtung	° s^{-1}
Q	Generalisierte Kraft	N
F_1, F_2	Äußere Kräfte	N
F_x	Antriebskraft in x-Richtung	N
F_y	Antriebskraft in y-Richtung	N
m_1	Masse Ball	kg
m_2	Masse Ballbot	kg
g	Erdbeschleunigung	m s^{-2}
J_0	Trägheitsmoment vom Antriebsrad	kg m^2
J_1	Trägheitsmoment vom Ball	kg m^2
ω	Kreisfrequenz vom Antriebsrad	s^{-1}
ω_a	Kreisfrequenz vom Ball	s^{-1}
R_0	Radius Antriebsrad	m
R_1	Radius Ball	m
i_a	Ankerstrom	A
u_a	Ankerspannung	V
R_a	Innenwiderstand Ankerkreis	Ω
L_a	Induktivität Ankerkreis	H

u_h	Induzierte Spannung im Ankerkreis	V
i_e	Erregerstrom	A
u_e	Erregerspannung	V
R_e	Innenwiderstand Erregerkreis	Ω
L_e	Induktivität Erregerkreis	H
k_m	Maschinenkonstante	-
Φ	Magnetischer Fluss	V s
M_a	Antriebsmoment	kg m²
M_l	Lastmoment	kg m²
M_b	Moment auf Ball	kg m²
k_r	Reibkonstante	-

Formelzeichen für die Regelung

A	Systemmatrix
b	Eingangsvektor
c	Ausgangsvektor
$\hat{A}$	Erweiterte Systemmatrix
$\hat{b}$	Erweiterter Eingangsvektor
$\hat{c}$	Erweiterter Ausgangsvektor
r	Führungsgröße
u	Stellgröße
u_x	Stellgröße für Regler in der ZX-Ebene
u_y	Stellgröße für Regler in der ZY-Ebene
y	Ausgangsgröße
x	Zustandsvektor
x_1	Zustandsvektor für Regler in der ZX-Ebene
x_2	Zustandsvektor für Regler in der ZY-Ebene
$\hat{x}$	Erweiterter Zustandsvektor
e	Regelfehler
S_u	Steuerbarkeitsmatrix
V	Vorfaktor
P	Strecke
h	Zustandsrückführung
h_I	Parameter für Integrierer
J	Gütekriterium
Q	Gewichtungsmatrix, um Zustandsgrößen zu gewichten
R	Skalarer Wert, um Stellgröße zu gewichten
T_s	Abtastzeit

Weitere Formelzeichen

U_{Stell}	Zugeführte Motorspannung	V
U_{Batt}	Versorgungsspannung des NXT-Bausteins	V
U_{Trans}	Spannungsabfall am Schalttransistor	V

Inhaltsverzeichnis

1 Einleitung

Thema dieser Masterarbeit ist der Aufbau und die Regelung eines Lego Ballbots mithilfe von MATLAB/Simulink.
Die erste Frage, die sich nun stellt, ist: Was ist eigentlich ein Ballbot? Unter einem Ballbot versteht man einen mobilen Roboter, der sich nicht wie gewöhnlich auf Rädern oder Ketten fortbewegt, sondern auf einem Ball balanciert. Durch die einzige Kontaktstelle, die der Roboter durch den Ball zum Boden hat, gibt es eine Reihe von Vorteilen, die ein Ballbot gegenüber anderen ähnlichen Robotern hat. Er ist omnidirektional, das heißt, er kann sich in alle Richtungen bewegen, ohne dass man einen Wenderadius benötigt und dadurch ist er außergewöhnlich agil und wendig. Ein weiterer Vorteil ist, dass er sich auf der eigenen Achse drehen kann.

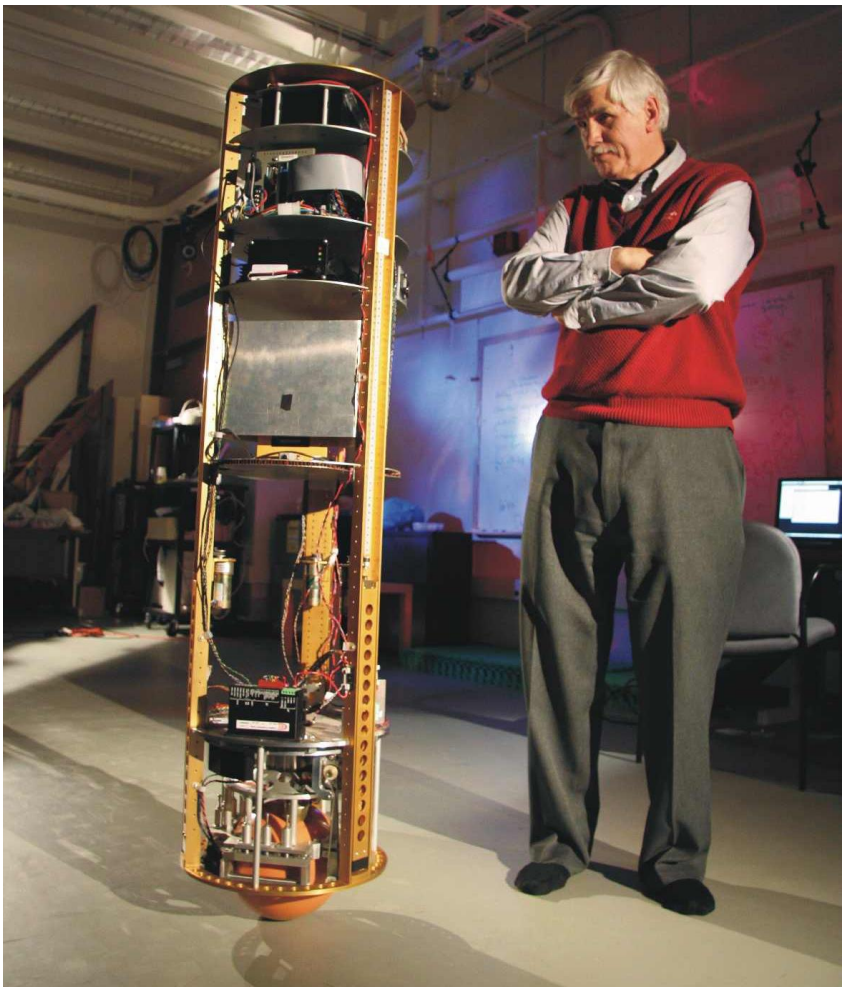


Abbildung 1.1: Prof. Ralph Hollis und sein Ballbot[1]

[1] Quelle: http://www.msl.ri.cmu.edu/projects/ballbot/

Der erste Ballbot wurde 2006 von Prof. Ralph Hollis an der Carnegie Mellon Universität in Pittsburgh entwickelt, siehe Abbildung 1.1. In den vergangenen Jahren wurde stetig an diesem Projekt weiter entwickelt, es wagten sich mehrere Universitäten an das Projekt Ballbot. Die Ballbots wurden stetig kleiner und präziser. 2010 entwickelte man an der ETH Zürich einen eigenen Ballbot mit dem Namen Rezero. Das Projekt war von einigem Erfolg gekrönt, sodass das Team 2012 ein Angebot von der Firma Disney bekam, solche Ballbots in den Vergnügungsparks der Firma einzusetzen.

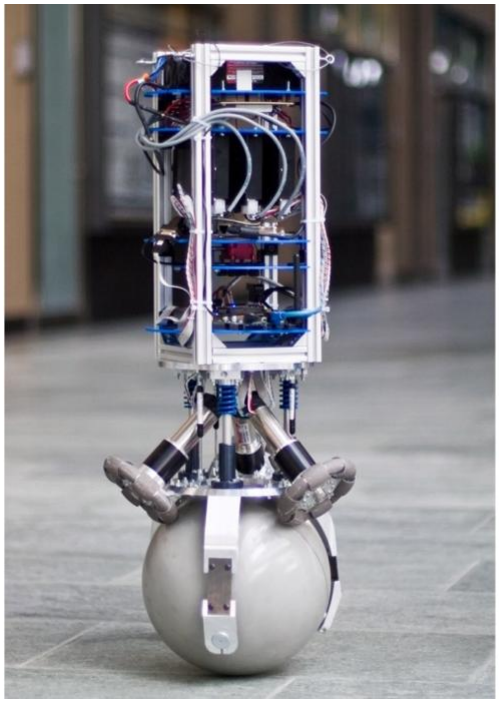


Abbildung 1.2: Links: Ballbot Rezero von der ETH Zürich[2]; Rechts: Rezero mit Infoscreen für die Firma Disney (Konzept)[3]

In der Zukunft kann man sich vorstellen, solche Ballbots in verschiedenen Bereichen einzusetzen. Man kann sie mit Infoscreens ausstatten oder mit anderen technischen Geräten wie z.B. Beamern oder Digitalkameras. Man würde sie auch gerne als Fortbewegungsmittel einsetzen.

[2] Quelle: http://en.wikipedia.org/wiki/Ballbot
[3] Quelle: http://rezero.ethz.ch/

2 Aufbau des Ballbots

LEGO Mindstorms ist eine Produktserie des dänischen Spielwarenherstellers LEGO. Sie wurde erstmals Anfang 2006 auf einer der größten Messen für Unterhaltungselektronik, der Consumer Electronics Show (CES), vorgestellt und sie ist seit Oktober 2006 im deutschsprachigen Raum erhältlich.
Kernstücke der Produktserie sind ein programmierbarer Legostein, sowie Elektromotoren, Sensoren und Lego-Technik-Teile, um Roboter oder andere autonome Systeme zu konstruieren und zu programmieren. Der LEGO Mindstorms-Baukasten ist nicht nur ein technisches Spielzeug, sondern wird auch als Lehrmittel in Hochschulen eingesetzt. Durch die Interaktion des Mikrocontrollers mit den Sensoren und Aktuatoren können verschiedene eingebettete Systeme realisiert werden.
Anfang Jänner 2013 wurde von LEGO auf der Consumer Electronics Show das Nachfolgesystem Mindstorms EV3 vorgestellt.

2.1 Hardware und Software

Hauptbestandteil des LEGO Mindstorms-Baukastens ist ein programmierbarer LEGO-Baustein, auch NXT genannt, welcher über verschiedene Programmiersprachen programmiert werden kann. Der NXT besitzt drei Motorausgänge und vier Sensoreingänge, die über einen I²C-Bus[4] kommunizieren. Die Kommunikation zwischen dem NXT und einem Rechner kann mittels einer Bluetooth-Schnittstelle oder einer USB 2.0-Schnittstelle geschehen. Folgende Aufzählung listet die wichtigsten Merkmale des programmierbaren NXT-Bausteins auf:

- CPU: 32 Bit Atmel ARM7 Mikroprozessor, 48 MHz, 256 kB Flash-Speicher, 64 kB RAM
- Koprozessor: 8 Bit Atmel ATmega48, 8 MHz, 4 kB Flash-Speicher, 512 Byte RAM
- USB 2.0-Anschluss
- Bluetooth-Schnittstelle
- 3 Motorausgänge
- 4 Sensoreingänge
- 100 x 64 Pixel LCD-Anzeige
- Open Source Firmware

[4] I^2C-Bus ist ein von Philips Semiconductors entwickelter serieller Datenbus, der hauptsächlich geräteintern für die Kommunikation zwischen verschiedenen Schaltungsteilen benutzt wird.

Ein großer Vorteil des LEGO Mindstorms-Baukastens ist, dass er mit verschiedenen Programmiersprachen betrieben werden kann. Die mitgelieferte Programmierumgebung ist eine grafische Programmiersprache (NXT-G), sie wurde in Zusammenarbeit mit National Instruments entwickelt und basiert auf LabVIEW. Neben der von LEGO bereitgestellten Programmierumgebung gibt es eine Vielzahl alternativer Programmiermöglichkeiten, wie z.B. NBC (Assembler ähnliche Syntax), NXC (C ähnliche Syntax), LabVIEW oder MATLAB/Simulink.

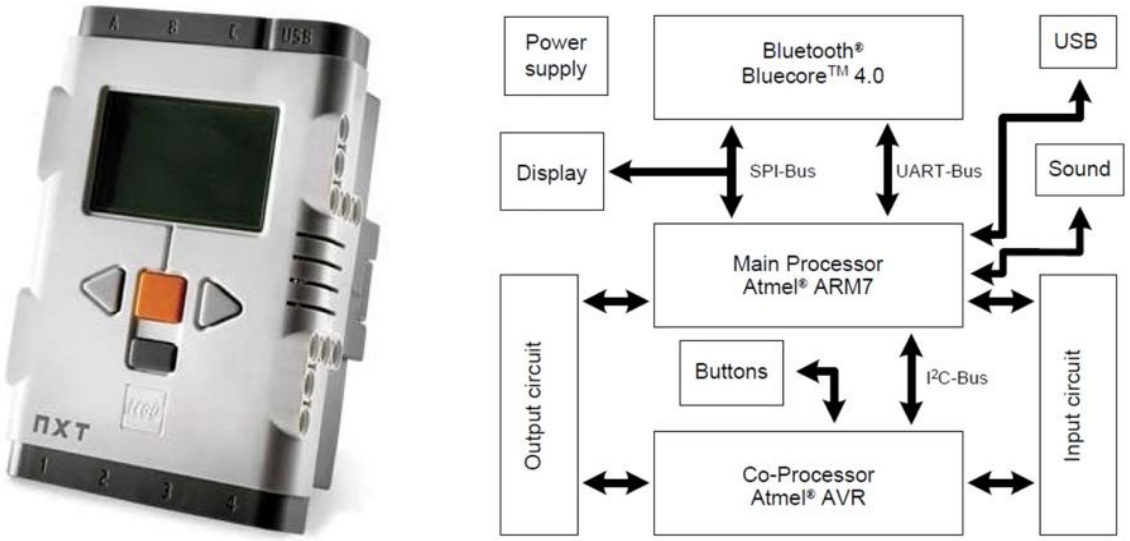


Abbildung 2.1: Links: Programmierbarer LEGO-Baustein NXT[5]; Rechts: Hardware-Architektur des NXT-Bausteins[6]

Damit man den NXT-Baustein mit anderen Programmiersprachen programmieren kann, ist es teils erforderlich den programmierbaren Legobaustein mit einer anderen Firmware zu bespielen, wie z.B. leJOS, ein Java-Betriebssystem oder nxtOSEK, ein Echtzeit-Betriebssystem, das als einziges die Ausführung von C- und C++-Code ermöglicht.

[5] Quelle: http://mindstorms.lego.com/en-us/default.aspx
[6] Quelle: http://legorobot.wiki-site.com/index.php/Lego_Mindstorm_Hardware

2.2 Sensoren und Aktuatoren

Der NXT-Baustein besitzt drei Ausgangsports für die Ansteuerung von Motoren und 4 Eingangsports für die Verbindung von Sensoren.

Abbildung 2.2: NXT-Baustein mit angeschlossenen Motoren und verschiedenen Sensoren[7]

Die Verbindung der Sensoren und Aktuatoren mit dem NXT geschieht über ein sechspoliges Kabel mit einer Art RJ12-Stecker. Abbildung 2.3 zeigt schematisch die Pinbelegung eines Ein- und Ausgangsports am NXT für die Motoren bzw. Sensoren.

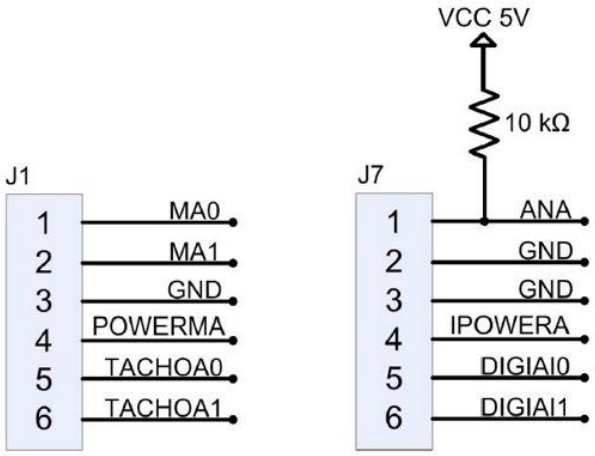


Abbildung 2.3: Schema eines Ein- (rechts) und Ausgangsports (links) am NXT-Baustein [2]

Die Signale MA0 und MA1 beim Ausgangsport werden für die Pulsweitenmodulation[8] benutzt, die man für die Ansteuerung der Motoren benötigt. Jeder Motor kann über diese

[7] Quelle: http://mindstorms.lego.com/en-us/default.aspx
[8] Pulsweitenmodulation (PWM) ist eine Modulationsart, bei der zwischen zwei Werten hin und her geschalten wird

Leitungen einen maximalen Strom von 700 mA beziehen. POWERMA ist für die Versorgungsspannung zuständig und die Signale TACHOA0 und TACHOA1 werden für die Auswertung der Anzahl der Rotationsimpulse und der Rotationsrichtung verwendet. Das Signal ANA beim Eingangsport ist ein analoger Eingang, IPOWERA ist wiederum für die Versorgungsspannung zuständig und die Signale DIGIAI0 und DIGIAI1 dienen der digitalen Kommunikation, welche über einen I^2C-Bus zustande kommt.

Die NXT-Motoren sind Gleichstrom-Servomotoren. Jeder Motor hat einen eingebauten Winkelsensor und wird mittels Pulsweitenmodulation angesteuert. Im Inneren des Motors befindet sich ein Getriebe (siehe Abbildung 2.4), durch das das Drehmoment vom Motor auf die Antriebswelle mit einem Übersetzungsverhältnis von 1:48 übertragen wird.

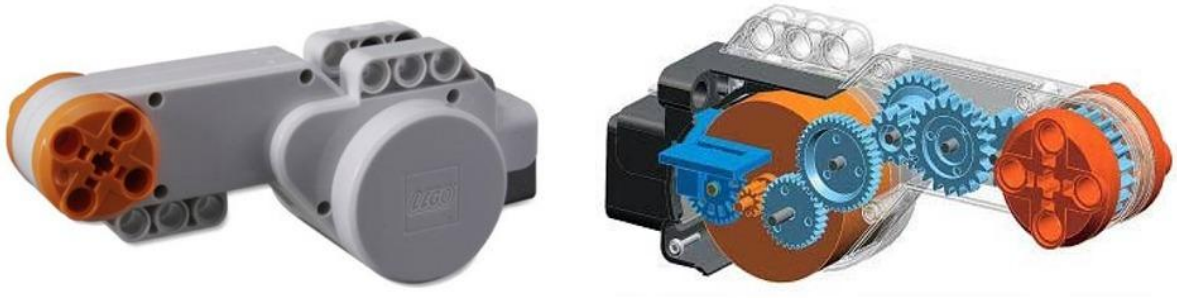


Abbildung 2.4: NXT Gleichstrom-Servomotor[9]

Die Spannungsversorgung der NXT-Motoren geschieht mittels des eingebauten Akkumulators am NXT-Baustein. Je nachdem, wie hoch die aktuelle Versorgungsspannung ist, ergeben sich verschiedene maximale Drehzahlen bzw. Drehmomente an der Motorwelle. In Abbildung 2.5 sieht man verschiedene Drehzahl-Kennlinien eines NXT-Servomotors bei unterschiedlichen Versorgungsspannungen und Lastmomenten. Auf der x-Achse ist jeweils die Motorleistung in Prozent angegeben. Wie man gut erkennen kann, ist die maximale Drehzahl bei voller Motorleistung abhängig von der Versorgungsspannung und vom Lastmoment. Je höher die Versorgungsspannung bzw. je kleiner das Lastmoment, desto höher ist die maximale Motordrehzahl, die erreicht werden kann bei voller Motorleistung.
Abbildung 2.6 zeigt die Drehzahl-Drehmoment-Kennlinie eines NXT-Motors bei unterschiedlicher Versorgungsspannung. Bei steigendem Drehmoment sinkt die maximale Drehzahl und bei sinkender Betriebsspannung verschiebt sich die gesamte Kennlinie nach unten.
In Abbildung 2.7 sieht man die Motorstrom-Drehmoment-Kennlinie eines NXT-Motors. Je nachdem, wie hoch die Versorgungsspannung ist, ändert sich das maximal zur Verfügung stehende Drehmoment des NXT-Servomotors. Um später den Ballbot balancieren zu können, muss ein gewisses Drehmoment zur Verfügung stehen. Ist die Versorgungsspannung zu gering, so wird man bei größeren Auslenkungen um die Ruhelage nicht mehr in der Lage sein, den Roboter in eine stabile Position zu regeln.

[9] Quelle: http://education.lego.com/de-de/

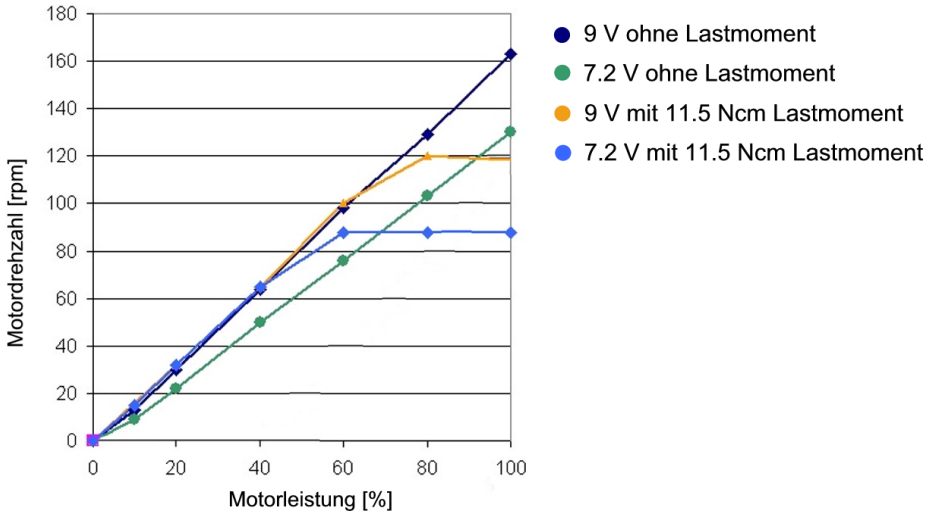


Abbildung 2.5: Drehzahl-Leistung-Motorkennlinien bei verschiedenen Betriebszuständen[10]

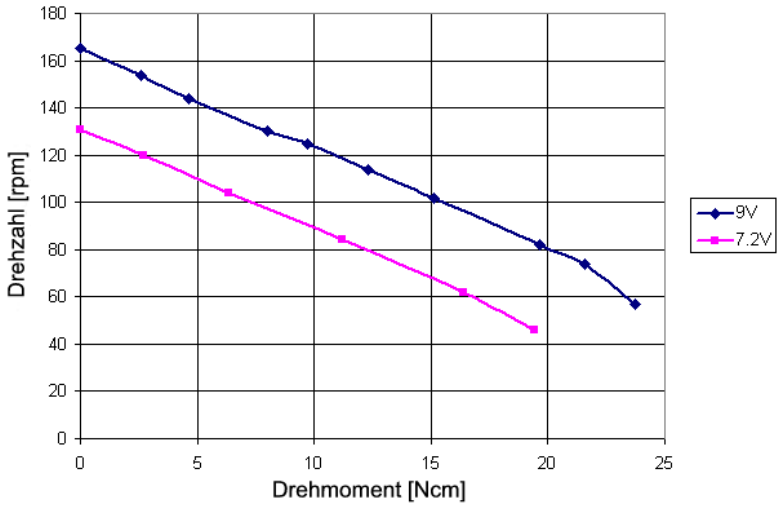


Abbildung 2.6: Drehzahl-Drehmoment Kennlinie eines NXT Gleichstrom-Servomotors[10]

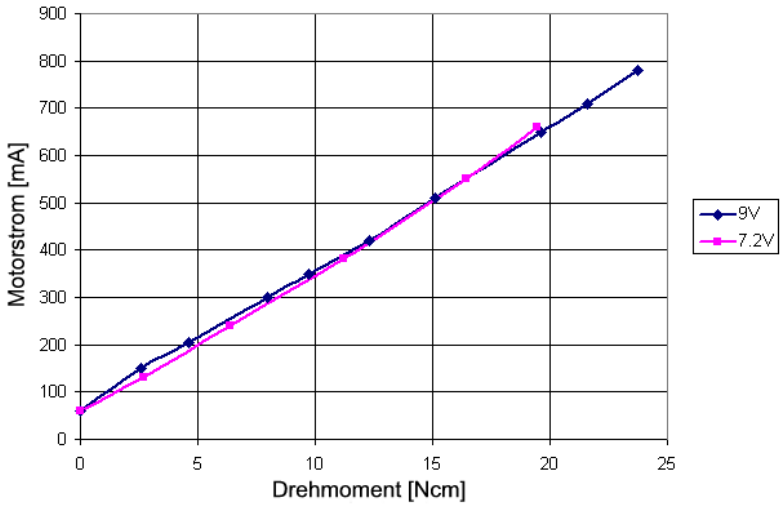


Abbildung 2.7: Motorstrom-Drehmoment Kennlinie eines NXT Gleichstrom-Servomotors[10]

[10] Quelle: http://www.philohome.com/nxtmotor/nxtmotor.htm

LEGO bietet eine Vielzahl von Sensoren an. Mittlerweile gibt es aber auch andere Firmen, die Sensoren für den NXT-Baustein herstellen. Eine Sonderstellung hat die Firma HiTechnic, die mit LEGO zusammenarbeitet und die Erlaubnis erhalten hat, das offizielle Sensorgehäuse von LEGO zu verwenden. Folgende Tabelle zeigt eine Auflistung der wichtigsten Sensoren für den NXT:

Sensorbezeichnung	Beschreibung	Hersteller
Tastsensor	Kann Berührungen feststellen	LEGO
Geräuschsensor	Misst Umgebungslautstärke	LEGO
Lichtsensor	Misst reflektiertes oder Umgebungslicht	LEGO
Ultraschallsensor	Misst Entfernung zu Gegenständen	LEGO
Temperatursensor	Misst Umgebungstemperatur	LEGO
Winkelsensor	Misst Motorumdrehung (im Motor integriert)	LEGO
Farbsensor	Kann Farben erkennen	LEGO/HiTechnic
Kompasssensor	Liefert Ausrichtung bezüglich des Erdmagnetfeldes	HiTechnic
Infrarotsensor	Detektiert Infrarotquellen und liefert deren Richtung	HiTechnic
Beschleunigungssensor	Misst Beschleunigungskräfte und Neigung des Sensors	HiTechnic
Gyroskopsensor	Misst Drehung/Neigung des Sensors	HiTechnic
Infrarotempfänger	Kann Infrarotsignale empfangen und dekodieren	HiTechnic
Stromsensor	Misst Stromstärke	Vernier
Spannungssensor	Misst Spannung	Vernier
Kraftsensor	Misst Kräfte	Vernier
Gasdrucksensor	Misst Druck in gasförmigen Medien	Vernier

Tabelle 2.1: Übersicht der wichtigsten Sensoren für den NXT-Baustein [3]

Für die Masterarbeit wurden nur der Winkelsensor von LEGO und der Gyroskopsensor und Kompasssensor von HiTechnic verwendet.

2.2.1 Winkelsensor

Jeder NXT-Motor besitzt einen eingebauten Winkelsensor, welcher die Umdrehungen in Winkelgrad mit einer Genauigkeit von 1° und einer maximalen Abtastrate von 1000 [1/Sek.] misst. Der Sensor besteht aus einem Encoder-Rad und basiert auf einer optischen Messung. Das Signal, welches der Sensor liefert, besteht einmal aus der Drehrichtung des Motors und einmal aus der Anzahl der Drehimpulse. Bei einer vollen Umdrehung sendet der Sensor 180 Impulse an den NXT, wobei sowohl bei der steigenden als auch bei der fallenden Flanke der NXT einen Drehimpuls zählt, und man erhält dadurch 360 Zählimpulse pro Umdrehung [3].

2.2.2 Kompasssensor

Der digitale Kompasssensor wird von der Firma HiTechnic für LEGO hergestellt und ist das elektronische Gegenstück zu einem handelsüblichen Kompass. Kompasse orientieren sich am Magnetfeld der Erde und richten ihre hochempfindliche Nadel immer Richtung Norden aus. Elektronische Kompasse bedienen sich des magnetoresistiven Effekts. Dieser Effekt nutzt aus, dass sich bei einigen Materialien, je nach Verlauf von magnetischen Feldern, der elektrische Widerstand verändert (siehe Abbildung 2.8). Diese Änderung drückt sich in Spannungsschwankungen aus, die gemessen werden können. In Abbildung 2.8 sieht man den dazugehörigen Messaufbau, der dabei eingesetzte magnetoresistive Widerstand Rφ ändert seinen Wert abhängig vom Winkel der Magnetfeldlinien und somit auch die gemessene Spannung. Je größer der Verdrehwinkel φ zum Bauteil ist, desto kleiner wird die gemessene Spannung U [6].

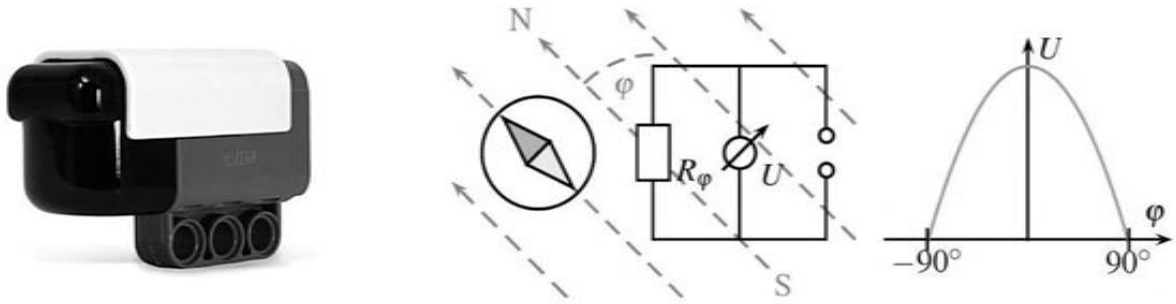


Abbildung 2.8: Links: Kompasssensor von der Firma HiTechnic[11]; Rechts: Messaufbau des Sensors und resultierender Spannungsverlauf[12]

Der Sensor besitzt einen Lese- und einen Kalibrier-Modus. Im Lese-Modus kann der Sensor die aktuelle Position des Roboters im Bezug zum Erdmagnetfeld ermitteln. Im Kalibrier-Modus kann der Sensor angepasst werden, um extern verursachte Schwankungen des Magnetfelds auszugleichen. Der Kompass funktioniert nur bei horizontaler Ausrichtung und sollte mindestens 15-20 cm vom NXT-Baustein und von den Motoren entfernt montiert werden, um den Einfluss von magnetischen Störfeldern zu vermeiden. Der Sensor hat eine Winkelauflösung von 1° und liefert Messwerte zwischen 0° und 359° mit einer maximalen Abtastrate von 100 [1/Sek.]. Der Wert 0 repräsentiert dabei den Norden, 90 Osten, 180 Süden und 270 Westen [3].

[11] Quelle: http://www.hitechnic.com/
[12] Quelle: [6]

2.2.3 Gyroskopsensor

Der Gyroskopsensor, oft auch Kreiselsensor genannt, ist ein rotierender Kreisel, der in einer kardanischen Aufhängung gelagert ist und aufgrund der Drehimpulserhaltung seine Orientierung im Raum beibehält. Wenn eine äußere Kraft die Drehachse des Kreisels zu drehen versucht, entsteht ein Drehmoment, welches über einen Encoder gemessen werden kann [6].

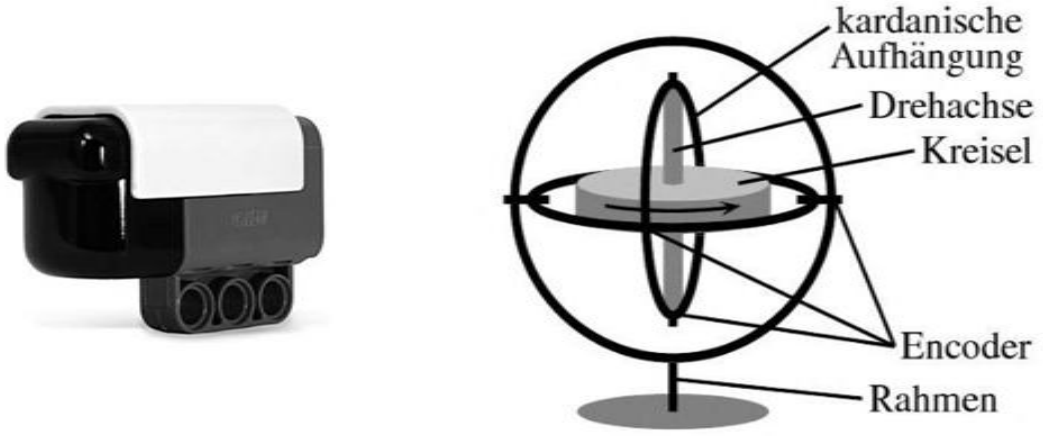


Abbildung 2.9: Links: Gyroskopsensor der Firma HiTechnic[13]; Rechts: Kreisel mit kardanischer Aufhängung[14]

Der Gyroskopsensor wird von der Firma HiTechnic hergestellt und misst die Winkelgeschwindigkeit auf einer Achse bis zu einer maximalen Winkelgeschwindigkeit von ±360° pro Sekunde mit einer maximalen Abtastrate von 300 [1/Sek.].

Das Sensorsignal des Gyroskopsensors ist mit einem Offset und einer Drift versehen. Bevor man dieses Signal verwenden kann, muss es dementsprechend gefiltert werden. In Kapitel 5 wird diese Problematik ausführlicher beschrieben.

[13] Quelle: http://www.hitechnic.com/
[14] Quelle: [6]

2.3 LEGO Ballbot

Der Aufbau des LEGO Ballbots mit dem LEGO Mindstorms-Baukasten basiert auf einer Bauanleitung von Yorihisha Yamamoto [4]. Abbildung 2.10 zeigt den fertig aufgebauten Roboter.

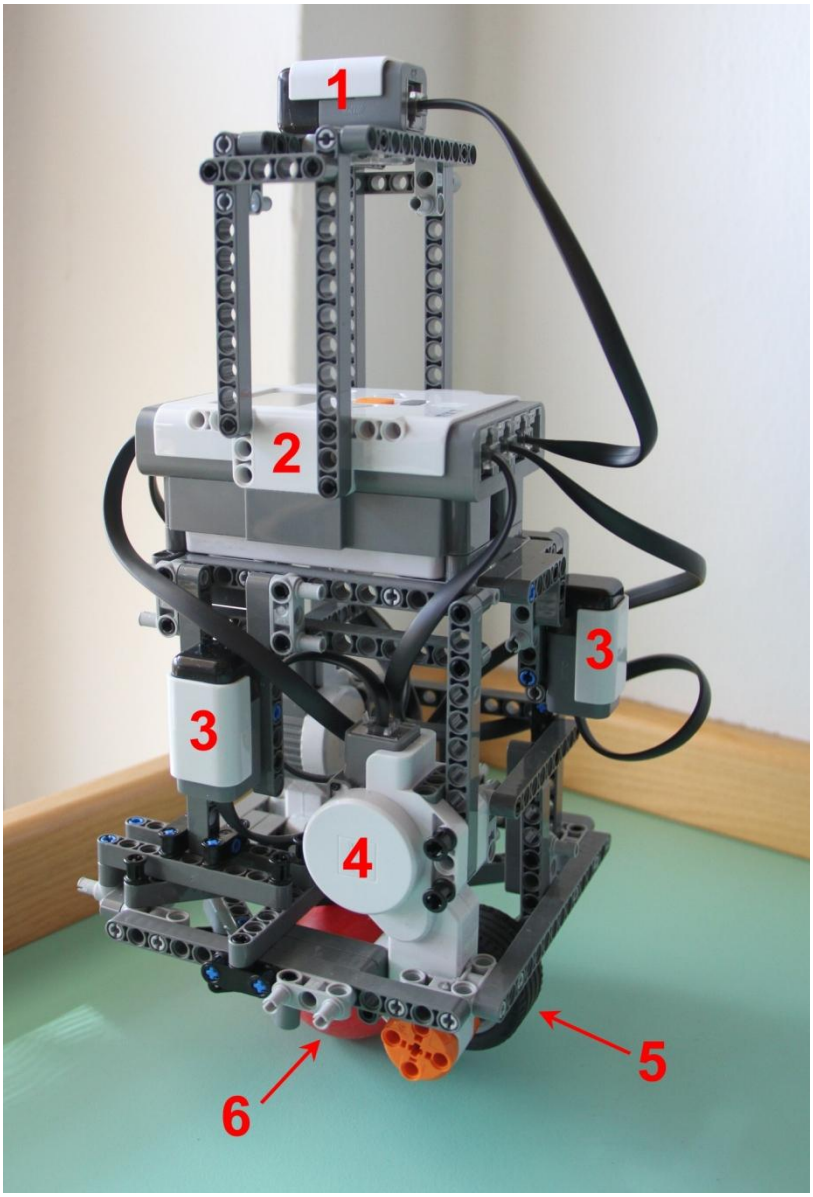


Abbildung 2.10: LEGO Ballbot; (1) – Kompasssensor, (2) – Programmierbarer NXT-Baustein, (3) – Gyroskopsensor, (4) – Gleichstrom-Servomotor, (5) – Gummirad, (6) – Plastikball

Wie man gut aus obiger Abbildung erkennen kann, besteht der LEGO Ballbot aus zwei Gleichstrom-Servomotoren, zwei Gyroskopsensoren, einem Kompasssensor und dem programmierbaren NXT-Baustein. Der gesamte Ballbot sitzt auf einem Plastikball und würde ohne Fremdeinwirkung umkippen. Der Ball wird von zwei Gummirädern, die über die Gleichstrom-Servomotoren angetrieben werden, in Bewegung gesetzt, siehe Abbildung 2.11. Ziel ist es jetzt, durch richtiges Bewegen des Plastikballes den Ballbot in Balance zu halten.

Die Messgrößen, die man dafür zur Verfügung hat, erhält man von den zwei Gyroskopsensoren und von den zwei Winkelsensoren, die in den Motoren eingebaut sind. Der NXT-Baustein liest diese Daten ein, verarbeitet diese und gibt dann eine entsprechende Stellgröße für die Servomotoren aus.

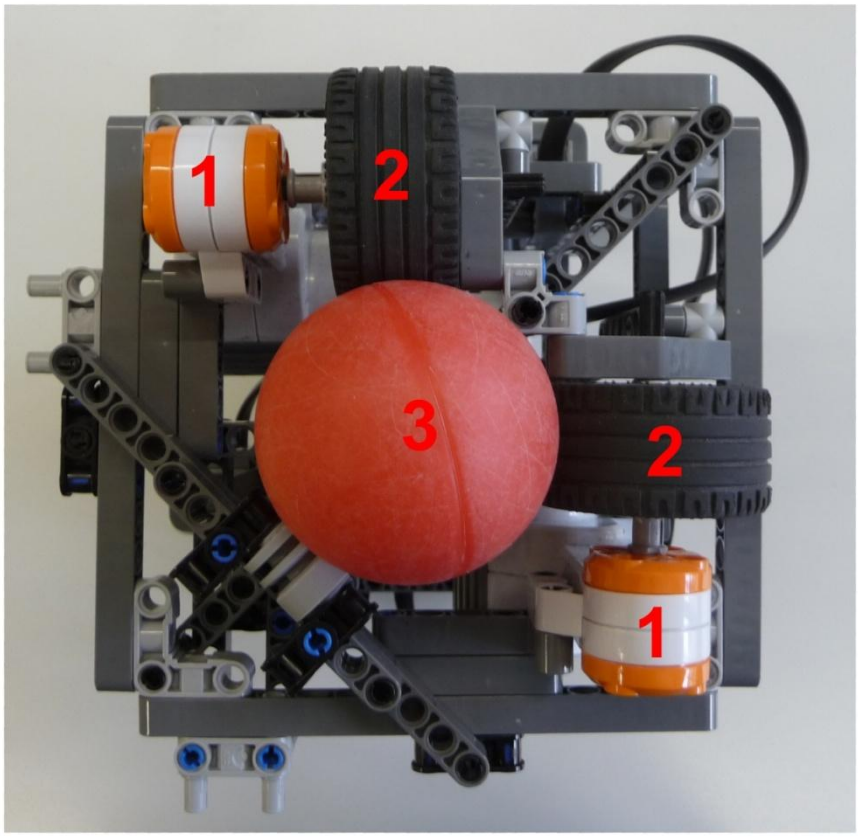


Abbildung 2.11: Unterseite des LEGO Ballbots; (1) – Gleichstrom-Servomotor, (2) – Gummirad, (3) Plastikball

Der digitale Kompass wird nicht für die Balancierung des Roboters benötigt, sondern für die Bestimmung der genauen Lage des Ballbots. Durch diese zusätzliche Information ist man in der Lage, den Ballbot in eine gewisse Position zu manövrieren.

3 Modellbildung

Wenn man sich den Ballbot näher anschaut, so stellt man fest, dass es sich um eine Art inverses Pendel handelt. Der Unterschied zu einem klassischen inversen Pendel ist, dass der Ballbot, bedingt durch seinen Aufbau, in alle Richtungen umkippen kann. Aufgrund der fixen Antriebskonfiguration des Ballbots ist eine vereinfachte Betrachtung möglich. Man kann das System in zwei idente Teilsysteme mit den Koordinatenachsen ZX und ZY aufteilen, wie in Abbildung 3.1 dargestellt.

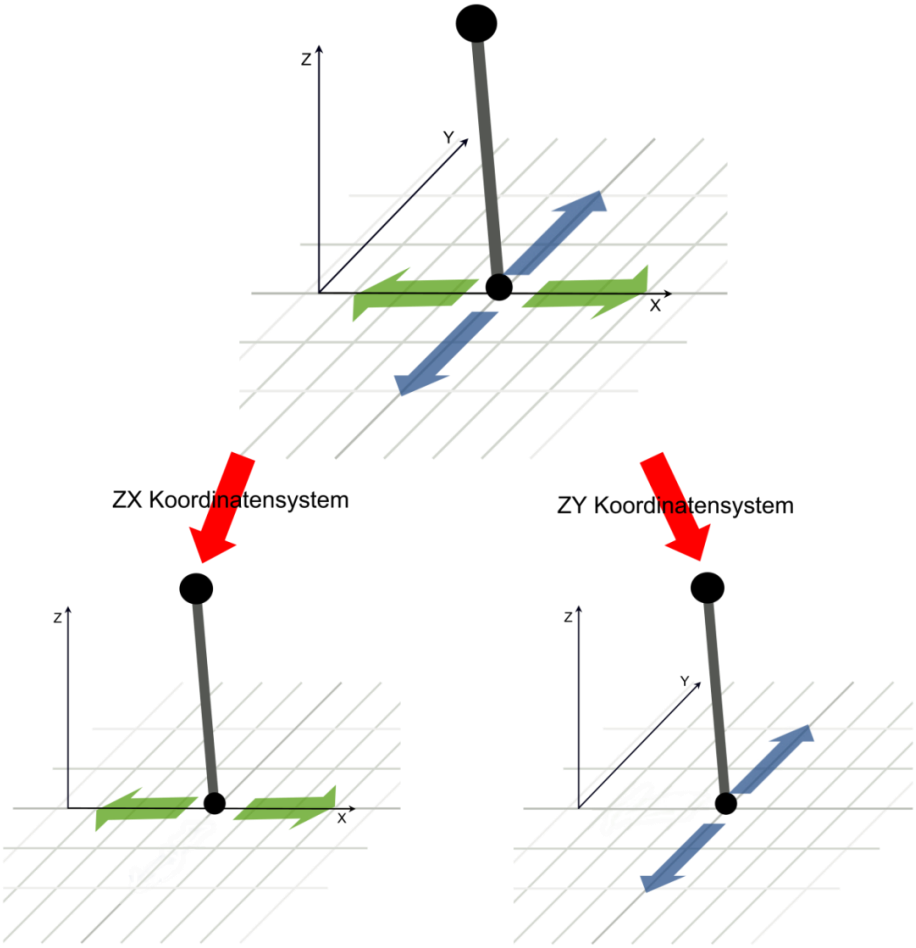


Abbildung 3.1: Aufteilung des Koordinatensystems in zwei zweidimensionale Koordinatensysteme ZX und ZY

Durch diese Aufteilung vereinfacht sich das mathematische Modell. Aus einem dreidimensionalen Koordinatensystem erhält man so zwei zweidimensionale Koordinatensysteme und man kann somit die Bewegungsgleichungen eines klassischen inversen Pendels verwenden.
Die Bewegungsgleichungen für das Modell werden mittels Lagrange-Formalismus berechnet.

3.1 Formalismus von Lagrange

Mit dem Lagrange-Formalismus ist es möglich, komplexe mechanische Systeme einfach zu beschreiben, indem man die kinetische und potentielle Energie eines Systems berücksichtigt.

Als Ausgangspunkt für die Berechnung der Bewegungsgleichungen dient das vereinfachte zweidimensionale mechanische Modell des Ballbots in Abbildung 3.2. Um die Bewegungsgleichungen möglichst einfach zu halten, werden folgende Annahmen getroffen:

1. Das Antriebsmoment liegt direkt am Ball
2. Zwischen Antriebsrad und Plastikball entsteht kein Schlupf
3. Es entstehen keine Verluste durch Reibung

Sozusagen wird das Antriebsrad ignoriert und man nimmt an, dass der Ballbot vom Plastikball direkt angetrieben wird. Diese Annahme kann man treffen, da das Trägheitsmoment des Antriebsrads gegenüber dem Trägheitsmoment des Plastikballs viel kleiner ist.

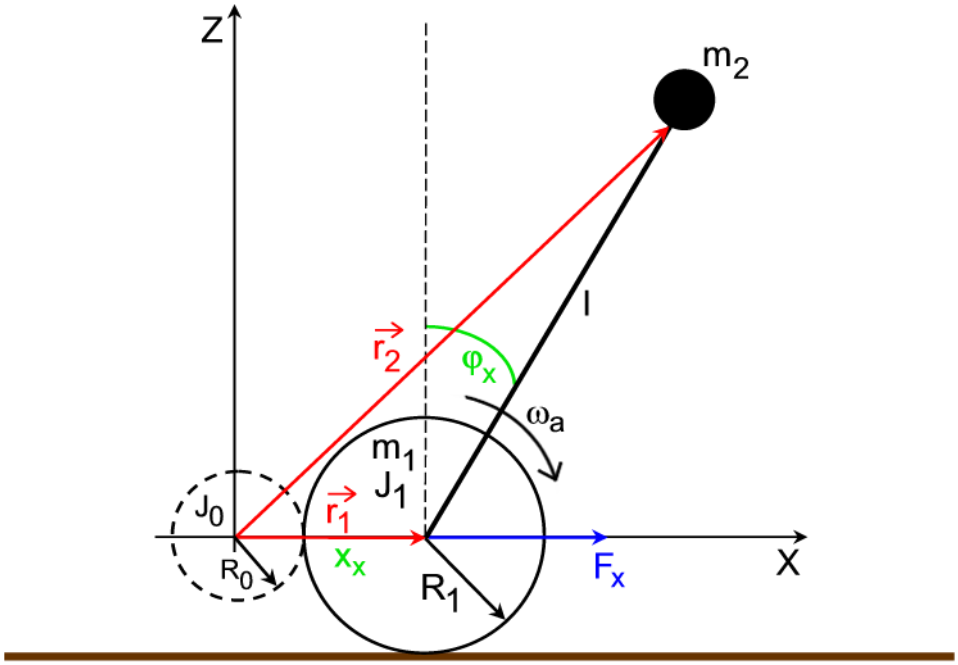


Abbildung 3.2: Vereinfachtes zweidimensionales mechanisches Modell des Ballbots

Aus Abbildung 3.2 ist ersichtlich, dass das Modell zwei Massenpunkte m_1 und m_2 besitzt. Durch die holonomen Zwangsbedingungen

$$z_1 = 0 \tag{3.1}$$

und

$$|\overrightarrow{r_2} - \overrightarrow{r_1}| - l = 0 \tag{3.2}$$

erhält man durch Einsetzen in folgender Gleichung

$$2N - k = n, \tag{3.3}$$

wobei k die Anzahl der holonomen Zwangsbedingungen und N die Anzahl der Massenpunkte ist, die Anzahl der Freiheitsgrade für das System. Durch Einsetzen von N und k in Gleichung 3.3 wird ersichtlich, dass das Modell zwei Freiheitsgrade besitzt und somit mit zwei Bewegungsgleichungen beschrieben werden kann.

Durch die Lagrange Funktion

$$L = T - V \tag{3.4}$$

kann man nun die Bewegungsgleichungen für das Modell berechnen, wobei T die potentielle Energie und V die kinetische Energie des Modells beschreibt.

Als erstes müssen die generalisierten Koordinaten bestimmt werden, wobei dafür die Position des Ballbots x_x und der Kippwinkel φ_x gewählt werden (siehe Abbildung 3.2):

$$q_1 = |\overrightarrow{r_1}| = x_x \tag{3.5}$$

$$q_2 = \varphi_x$$

Daraus können die Ortsvektoren r_1 und r_2 berechnet werden:

$$\overrightarrow{r_1} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_x \\ 0 \end{bmatrix} = \begin{bmatrix} q_1 \\ 0 \end{bmatrix} \tag{3.6}$$

$$\overrightarrow{r_2} = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_x + l * sin\varphi_x \\ l * cos\varphi_x \end{bmatrix} = \begin{bmatrix} q_1 + l * sinq_2 \\ l * cosq_2 \end{bmatrix}$$

Die generalisierten Kräfte berechnen sich wie folgt:

(3.7)

$$Q_j = \sum_{i=1}^{n} \overrightarrow{F_i^*} * \frac{\partial \vec{r}_i}{\partial q_j} \quad j = 1{,}2$$

wobei $\overrightarrow{F_1^*}$ und $\overrightarrow{F_2^*}$ die äußeren Kräfte sind

(3.8)

$$\overrightarrow{F_1^*} = \begin{bmatrix} F_x \\ 0 \end{bmatrix}$$

$$\overrightarrow{F_2^*} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Mit F_x wird die Antriebskraft auf die Masse m_1 bezeichnet, siehe Abbildung 3.2.

Die Gewichtskraft des Ballbots kann entweder über die äußeren Kräfte oder über die potentielle Energie berücksichtigt werden. In dieser Arbeit wurde die Gewichtskraft über die potentielle Energie berücksichtigt.

Aus Gleichung 3.7 folgt dann:

(3.9)

$$Q_1 = \overrightarrow{F_1^*} * \frac{\partial \overrightarrow{r_1}}{\partial q_1} + \overrightarrow{F_2^*} * \frac{\partial \overrightarrow{r_2}}{\partial q_1} = \begin{bmatrix} F_x \\ 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = F_x$$

$$Q_2 = \overrightarrow{F_1^*} * \frac{\partial \overrightarrow{r_1}}{\partial q_2} + \overrightarrow{F_2^*} * \frac{\partial \overrightarrow{r_2}}{\partial q_2} = \begin{bmatrix} F_x \\ 0 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} * \begin{bmatrix} l * cosq_2 \\ -l * sinq_2 \end{bmatrix} = 0$$

Um jetzt die Bewegungsgleichungen berechnen zu können, muss noch die potentielle Energie

(3.10)

$$V = m_2 * g * l * sin\varphi_x$$

und die kinetische Energie

(3.11)

$$T = \frac{1}{2} m_1 \dot{\overrightarrow{r_1}}^2 + \frac{1}{2} m_2 \dot{\overrightarrow{r_2}}^2 + \frac{1}{2} J_1 \dot{\omega_a}^2$$

bzw.

$$T = \frac{1}{2} m_1 \dot{\overrightarrow{r_1}}^2 + \frac{1}{2} m_2 \dot{\overrightarrow{r_2}}^2 + \frac{1}{2} J_1 \left(\frac{\dot{\overrightarrow{r_1}}}{R_1} \right)^2$$

berechnet werden, wobei die ersten zwei Terme die translatorische Bewegung und der letzte Term die rotatorische Bewegung beschreibt.

Mittels der Lagrange Funktion (siehe Gleichung 3.4) können nun die Bewegungsgleichungen wie folgt formuliert werden:

(3.12)

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q_1}}\right) - \frac{\partial L}{\partial q_1} = Q_1$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q_2}}\right) - \frac{\partial L}{\partial q_2} = Q_2$$

bzw.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x_x}}\right) - \frac{\partial L}{\partial x_x} = F_x$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi_x}}\right) - \frac{\partial L}{\partial \varphi_x} = 0$$

Man setzt nun für die Lagrange Funktion die Beziehungen 3.10 und 3.11 für die potentielle und kinetische Energie ein und stellt die Differentialgleichungen auf. Als Ergebnis erhält man folgende zwei Bewegungsgleichungen:

(3.13)

$$-\frac{1}{{R_1}^2}\left(-m_1 \ddot{q_1} {R_1}^2 - m_2 {R_1}^2 \ddot{q_1} + m_2 {R_1}^2 l \sin(q_2)\, \dot{q_2}^2 - m_2 {R_1}^2 l \cos(q_2)\, \ddot{q_2} - J_1 \ddot{q_1}\right) = F_x$$

$$m_2 l(\ddot{q_1}\cos(q_2) + l\ddot{q_2} - g\sin(q_2)) = 0$$

Um die Differentialgleichungen aufzustellen und das mathematische Modell zu berechnen, wurde die Mathematik-Software Maple verwendet. Im Anhang A sieht man den dazugehörigen Quellcode und die einzelnen Schritte und Ergebnisse im Detail.

3.2 Gleichstrommaschine

Aus Gleichung 3.9 erhalten wir für die generalisierte Kraft Q_1 die Kraft F_x. Die Kraft F_x ist die Antriebskraft des Ballbots. Aus Kapitel 2.2 weiß man, dass die Gleichstrom-Servomotoren mittels Pulsweitenmodulation angesteuert werden. Für die Modellbildung heißt das, dass unsere Eingangsgröße für den Regler ein Spannungssignal ist. Man benötigt also für die Kraft F_x eine Gleichung, in der die Eingangsspannung des Motors vorkommt. Um auf folgende Beziehung zu kommen, benötigt man das mathematische Modell der Gleichstrommaschine.

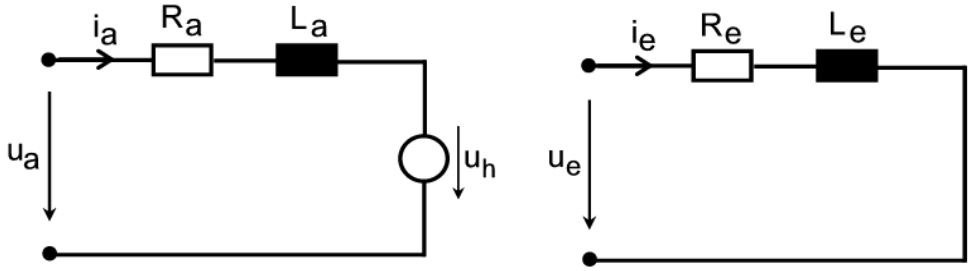


Abbildung 3.3: Ersatzschaltbild einer Gleichstrommaschine; Links: Ankerzweig; Rechts: Erregerzweig

Aus dem Ersatzschaltbild für die Gleichstrommaschine in Abbildung 3.3 kann man die elektrischen Gleichungen für die Gleichstrommaschine aufstellen:

(3.14)

$$L_e \frac{di_e}{dt} + R_e i_e = u_e$$

$$L_a \frac{di_a}{dt} + R_a i_a + k_m \Phi \omega = u_a$$

Wobei für den magnetischen Fluss Φ

$$\phi = f(i_e)$$

gilt.

Die Bewegungsgleichung für die Gleichstrommaschine sieht folgendermaßen aus

(3.15)

$$J \frac{d\omega}{dt} = M_a - k_r \omega - M_l$$

mit dem Trägheitsmoment des Rotors J und dem Antriebsmoment

(3.16)

$$M_a = k_m \phi i_a.$$

Der Term $k_r\omega$ in Gleichung 3.15 beschreibt näherungsweise die Reibung und M_l ist das Lastmoment.

Die vorher angeführten Gleichungen enthalten Nichtlinearitäten, da der magnetische Fluss vom Erregerstrom abhängt. Mit der Annahme, dass der verwendete Gleichstrommotor ein fremderregter Motor mit konstantem Erregerstrom I_e ist, kann man den magnetischen Fluss als konstant annehmen, d.h.

$$\Phi(\mathrm{t}) = \Phi_0$$

und die Nichtlinearitäten können ganz einfach beseitigt werden. Der Erregerkreis kann aus den Betrachtungen herausgenommen werden und damit vereinfachen sich obige Gleichungen zu:

(3.17)

$$L_a \frac{di_a}{dt} = -R_a i_a - k_m \phi_0 \omega + u_a$$

$$J \frac{d\omega}{dt} = k_m \phi_0 i_a - k_r \omega - M_l$$

Die Induktivität an der Ankerseite ist sehr klein, so dass man die linke Seite der Gleichung 3.17 auf Null setzen kann:

(3.18)

$$0 = -R_a i_a - k_m \phi_0 \omega + u_a$$

Löst man diese Gleichung nach i_a auf

(3.19)

$$i_a = \frac{1}{R_a}(u_a - k_m \phi_0 \omega)$$

und setzt sie in Gleichung 3.16 ein, so erhält man für das Antriebsmoment folgende Beziehung:

(3.20)

$$M_a = k_m \phi_0 \frac{1}{R_a}(u_a - k_m \phi_0 \omega),$$

wobei angenommen wurde, dass das Modell kein Lastmoment besitzt und dass es ohne Reibung behaftet ist.

3.3 Übersetzungsverhältnis

In Abschnitt 3.1 wurde angenommen, dass das Antriebsmoment direkt am Ball liegt und dass zwischen Antriebsrad und Plastikball kein Schlupf entsteht. Um diesen Gedanken korrekt weiter führen zu können, muss aber das Übersetzungsverhältnis zwischen Antriebsrad und Plastikball berücksichtigt werden.

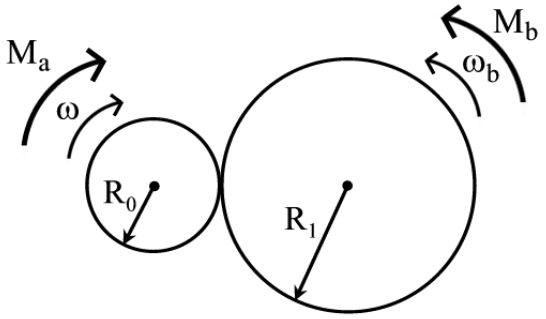


Abbildung 3.4: Übersetzung Antriebsrad-Plastikball

Wenn kein Schlupf zwischen Antriebsrad und Plastikball vorhanden ist, dann gilt folgende Beziehung zwischen dem Antriebsmoment und dem Moment am Plastikball:

(3.21)

$$M_a \omega = M_b \omega_b$$

bzw.

(3.22)

$$R_0 \omega = R_1 \omega_b$$

Löst man Gleichung 3.22 nach ω auf

(3.23)

$$\omega = \frac{R_1}{R_0} \omega_b$$

und setzt sie in Gleichung 3.21 ein

(3.24)

$$M_a \frac{R_1}{R_0} \omega_b = M_b \omega_b,$$

so erhält man für das Moment am Plastikball folgendes Ergebnis:

(3.25)

$$M_b = M_a \frac{R_1}{R_0}$$

3.4 Nichtlineares Modell

Da die rechte Seite der ersten Bewegungsgleichung eine Kraft benötigt und kein Moment (siehe Gleichung 3.13), kann Gleichung 3.25 durch die Beziehung:

(3.26)

$$M = F * r$$

in

(3.27)

$$F = \frac{M_b}{R_1} = \frac{M_a}{R_0}$$

umgeschrieben werden.

Setzt man nun in Gleichung 3.27 das Antriebsmoment (Gleichung 3.20) ein, so erhält man die benötigte Antriebskraft in Abhängigkeit der Motor-Eingangsspannung

(3.28)

$$F_x = \frac{1}{R_0} k_m \phi_0 \frac{1}{R_a} (u_a - k_m \phi_0 \omega),$$

wobei die Eingangsspannung des Motors zugleich die Stellgröße für den zukünftigen Regler ist.

Die Bewegungsgleichungen für den Ballbot lauten dann:

(3.29)

$$-\frac{1}{{R_1}^2}\left(-m_1 \ddot{q_1} {R_1}^2 - m_2 {R_1}^2 \ddot{q_1} + m_2 {R_1}^2 l \sin(q_2)\, \dot{q_2}^2 - m_2 {R_1}^2 l \cos(q_2)\, \ddot{q_2} - J_1 \ddot{q_1}\right) = \frac{k_m \Phi (u_a R_1 - k_m \Phi \dot{q_1})}{R_a R_0 R_1}$$

$$m_2 l(\ddot{q_1} \cos(q_2) + l\ddot{q_2} - g \sin(q_2)) = 0$$

Um nun von den Bewegungsgleichungen in eine Zustandsraumdarstellung zu kommen, führt man folgende Spaltenvektoren

(3.30)

$$q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}, \qquad \dot{q} = \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \end{bmatrix}, \qquad \ddot{q} = \begin{bmatrix} \ddot{q_1} \\ \ddot{q_2} \end{bmatrix}$$

ein und fasst die beiden Bewegungsgleichungen (3.29) zu einer vektorwertigen Beziehung der Form

(3.31)

$$l(q, \dot{q}, \ddot{q}) = r(q, \dot{q}, \dots)$$

zusammen.

Der linke Teil der Gleichung 3.31 kann wie folgt aufgeteilt werden:

$$M\ddot{q} + h(q, \dot{q}) = r(q, \dot{q}, \dots), \tag{3.32}$$

wobei mit

$$M = \frac{\partial l}{\partial \ddot{q}} \tag{3.33}$$

die Massenmatrix bezeichnet wird. Die Massenmatrix des vorliegenden Systems ist stets positiv definit.

Nach Umformen der Gleichung 3.32 in

$$\ddot{q} = -M^{-1}h(q, \dot{q}) + M^{-1}r(q, \dot{q}, \dots) \tag{3.34}$$

kann man das nichtlineare mathematische Modell des Ballbots wie folgt angeben:

$$\dot{x_1} = x_3 \tag{3.35}$$

$$\dot{x_2} = x_4$$

$$\dot{x_3} = \ddot{q_1} = \Big(R_1\big(k_m \Phi u_a R_1 - {k_m}^2 \Phi^2 \mathrm{x3} + \mathrm{m_2 l \sin(x_2)\, {x_4}^2 R_a R_0 R_1} - \cos(\mathrm{x_2})\, \mathrm{R_1 m_2 g \sin(x_2) R_a R_0}\big)\Big) / \Big(\big({m_1 {R_1}^2 + m_2 {R_1}^2 + J_1 - \mathrm{m_2 \cos(x_2)^2\, {R_1}^2}}\big)\mathrm{R_a R_0}\Big)$$

$$\dot{x_4} = \ddot{q_2} = -\big(\cos(\mathrm{x_2})\, {R_1}^2 k_m \Phi u_a - \cos(\mathrm{x_2}) R_1 {k_m}^2 \Phi^2 \mathrm{x_3} + \cos(\mathrm{x_2}) {R_1}^2 \mathrm{m_2 l \sin(x_2)\, {x_4}^2 R_a R_0} - \mathrm{g \sin(x_2) R_a R_0 m_1 {R_1}^2} - g \sin(\mathrm{x_2}) \mathrm{R_a R_0 m_2 {R_1}^2} - g \sin(\mathrm{x_2}) \mathrm{R_a R_0} J_1\big) / \big(l\big(\mathrm{m_1 {R_1}^2 + m_2 {R_1}^2 + J_1 - m_2 \cos(x_2)^2\, {R_1}^2}\big)\mathrm{R_a R_0}\big)$$

3.5 Linearisierung um die Ruhelage

Das mathematische Modell des Ballbots ist ein nichtlineares Modell. Um später einen Regler entwerfen zu können, muss es zuerst linearisiert werden. Die Methode, die dabei angewendet wird, ist die Linearisierung um die Ruhelage. Vorteil dieser Methode ist, dass man später die klassischen Reglerentwurfsmethoden verwenden kann. Nachteil ist aber, dass die Linearisierung nur für hinreichend kleine Auslenkungen um die Ruhelage mit dem nichtlinearen System übereinstimmt.

Das nichtlineare Regelungssystem

$$\frac{dx}{dt} = f(x,u) \qquad x \in \mathbb{R}^n, u \in \mathbb{R} \tag{3.36}$$

wird um die Ruhelage

$$\frac{dx_R}{dt} = 0 = f(x_{R,}u_R) \tag{3.37}$$

linearisiert, wobei die Ruhelagen des Ballbot Modells aus dem nichtlinearen mathematischen Modell aus 3.35 mit $u_a = 0$ wie folgt berechnet werden können:

$$0 = x_{3_R} \tag{3.38}$$

$$0 = x_{4_R}$$

$$0 = -R_1 \cos(\mathrm{x}_{2_\mathrm{R}})\, \mathrm{R}_1 \mathrm{m}_2 \mathrm{g} \sin(\mathrm{x}_{2_\mathrm{R}}) \mathrm{R}_\mathrm{a} \mathrm{R}_0$$

$$0 = \mathrm{g} \sin(\mathrm{x}_{2_\mathrm{R}}) \mathrm{R}_\mathrm{a} \mathrm{R}_0 \mathrm{m}_1 {R_1}^2 + g \sin(\mathrm{x}_{2_\mathrm{R}}) \mathrm{R}_\mathrm{a} \mathrm{R}_0 \mathrm{m}_2 {R_1}^2 + g \sin(\mathrm{x}_{2_\mathrm{R}}) \mathrm{R}_\mathrm{a} \mathrm{R}_0 J_1$$

Daraus ergeben sich folgende Ruhelagen:

$$x_{2_R} = 0, \pm\pi, \pm 2\pi, \pm 3\pi, \pm 4\pi, \ldots \tag{3.39}$$

$$x_{3_R} = 0$$

$$x_{4_R} = 0$$

Diese unendlich vielen Ruhelagen für x_{2_R} entsprechen den beiden möglichen Gleichgewichtszuständen eines realen Pendels in der hängenden bzw. in der aufrechten Position. Für das Ballbot Modell kommt nur die Ruhelage in der aufrechten Position in Frage, wobei zu beachten ist, dass es sich um eine instabile Ruhelage handelt. Die Komponente

x_{1_R} in der Ruhelage kann jeglichen Wert annehmen, d.h. der Ballbot kann auf jeder Position den aufrechten Gleichgewichtszustand erlangen.

Um ein lineares System zu erhalten, betrachtet man nur kleine Auslenkungen um die Ruhelage:

$$x \rightarrow x_R + \Delta x$$
$$u \rightarrow u_R + \Delta u$$

Setzt man diese Darstellung in der linken und rechten Seite der Gleichung 3.36 ein und entwickelt die rechte Seite in eine Taylorreihe um die Ruhelage

(3.40)

$$f(x_R + \Delta x, u_R + \Delta u) = f(x_{R,} u_R) + \left.\frac{\partial f}{\partial x}\right|_{x_R, u_R} * \Delta x + \left.\frac{\partial f}{\partial u}\right|_{x_R, u_R} * \Delta u + Terme\ höherer\ Ordnung,$$

wobei

$$f(x_{R,} u_R) = 0$$

ist, so erhält man bei Vernachlässigen der Terme höherer Ordnung der Taylorreihenentwicklung das linearisierte System

(3.41)

$$\frac{d\Delta x}{dt} = A\Delta x + b\Delta u.$$

Im linearisierten Modell gilt:

(3.42)

$$A = \left.\frac{\partial f}{\partial x}\right|_{x_R, u_R} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \\ \frac{\partial f_n}{\partial x_1} & & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{x_R, u_R}$$

$$b = \left.\frac{\partial f}{\partial u}\right|_{x_R, u_R} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} \\ \vdots \\ \frac{\partial f_n}{\partial u_1} \end{bmatrix}_{x_R, u_R}$$

3.6 Zustandsraummodell

Nachdem man die Linearisierung um die Ruhelage durchgeführt hat, erhält man für den Ballbot folgendes lineares Zustandsraummodell:

(3.43)

$$\frac{dx_1}{dt} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{g m_2 {R_1}^2}{m_1 {R_1}^2 + J_1} & -\frac{R_1 {k_m}^2 \Phi^2}{R_a R_0 (m_1 {R_1}^2 + J_1)} & 0 \\ 0 & \frac{g\left(m_1 {R_1}^2 + m_2 {R_1}^2 + J_1\right)}{l(m_1 {R_1}^2 + J_1)} & \frac{R_1 {k_m}^2 \Phi^2}{l R_a R_0 (m_1 {R_1}^2 + J_1)} & 0 \end{bmatrix} * \begin{bmatrix} x_x \\ \varphi_x \\ \dot{x}_x \\ \dot{\varphi}_x \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{{R_1}^2 k_m \Phi}{R_a R_0 (m_1 {R_1}^2 + J_1)} \\ -\frac{{R_1}^2 k_m \Phi}{l R_a R_0 (m_1 {R_1}^2 + J_1)} \end{bmatrix} u_x$$

mit den Zustandsgrößen ${x_1}^T = [x_x, \varphi_x, \dot{x}_x, \dot{\varphi}_x]$.

- x_x … Position des Ballbots in x-Richtung
- φ_x … Kippwinkel des Ballbots in x-Richtung
- $\dot{x_x}$ … Geschwindigkeit des Ballbots in x-Richtung
- $\dot{\varphi_x}$ … Kippwinkelgeschwindigkeit des Ballbots in x-Richtung

Obiges Zustandsraummodell gilt nur für den Regler in der ZX-Ebene. Um den Ballbot betreiben zu können, benötigt man einen zweiten identen Regler für die ZY-Ebene. Das dazugehörige Zustandsraummodell sieht wie folgt aus:

(3.44)

$$\frac{dx_2}{dt} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{g m_2 {R_1}^2}{m_1 {R_1}^2 + J_1} & -\frac{R_1 {k_m}^2 \Phi^2}{R_a R_0 (m_1 {R_1}^2 + J_1)} & 0 \\ 0 & \frac{g\left(m_1 {R_1}^2 + m_2 {R_1}^2 + J_1\right)}{l(m_1 {R_1}^2 + J_1)} & \frac{R_1 {k_m}^2 \Phi^2}{l R_a R_0 (m_1 {R_1}^2 + J_1)} & 0 \end{bmatrix} * \begin{bmatrix} y_y \\ \varphi_y \\ \dot{y}_y \\ \dot{\varphi}_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{{R_1}^2 k_m \Phi}{R_a R_0 (m_1 {R_1}^2 + J_1)} \\ -\frac{{R_1}^2 k_m \Phi}{l R_a R_0 (m_1 {R_1}^2 + J_1)} \end{bmatrix} u_y$$

mit den Zustandsgrößen ${x_2}^T = [y_y, \varphi_y, \dot{y}_y, \dot{\varphi}_y]$.

- y_y … Position des Ballbots in y-Richtung
- φ_y … Kippwinkel des Ballbots in y-Richtung
- $\dot{y_y}$ … Geschwindigkeit des Ballbots in y-Richtung
- $\dot{\varphi_y}$ … Kippwinkelgeschwindigkeit des Ballbots in y-Richtung

3.7 Modellparameter

Um einen passenden Regler für das Ballbot-Modell finden zu können, müssen zuerst die Parameter für das Modell bestimmt werden. Ein Großteil der Parameter kann gemessen werden, andere hingegen können nur geschätzt werden bzw. werden aus der Fachliteratur entnommen. In Tabelle 3.1 ist eine Auflistung aller verwendeten Parameter für das Ballbot-Modell zu sehen.

Parameter	Wert	Einheit	Beschreibung
m_1	0,012	kg	Masse des Plastik-Balls
m_2	0,734	kg	Masse des Ballbots
R_0	0,021	m	Radius Antriebsrad
R_1	0,02505	m	Radius Plastik-Ball
J_0	4,5e-8	kgm²	Trägheitsmoment Antriebsrad
J_1	5,02e-6	kgm²	Trägheitsmoment Ball
l	0,19	m	Abstand Ball Mitte - Schwerpunkt
R_a	4,6	Ω	Ankerwiderstand NXT-Motor
$k_m\Phi$	0,48	Vs/rad	Maschinenkonstante NXT-Motor

Tabelle 3.1: Parameter für das Ballbot Modell

Die Parameter für die Masse des Plastik-Balls und des Ballbots werden mittels einer Laborwaage ermittelt. Die Radien für das Antriebsrad und den Plastikball werden mittels einer Schieblehre bestimmt. Das Trägheitsmoment des Balls wird mittels folgender Formel berechnet:

(3.45)

$$J_0 = \frac{2}{3} * m_1 * {R_1}^2$$

Die Parameter für das Trägheitsmoment des Antriebsrads, für den Ankerwiderstand und für die Maschinenkonstante werden aus [1] entnommen, wobei bei der Maschinenkonstante das Übersetzungsverhältnis des Servomotors berücksichtig wird. Der Schwerpunkt des Ballbots kann nur abgeschätzt werden, so dass der Parameter für den Abstand vom Ball-Mittelpunkt bis zum Schwerpunkt nur geschätzt werden kann.

Verwendet man die Parameter aus Tabelle 3.1 für das lineare Zustandsraummodell aus 3.43 bzw. 3.44, so erhält man für die Systemmatrix bzw. für den Eingangsvektor folgende Werte:

(3.46)

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -360 & -4761 & 0 \\ 0 & 1947 & 25056 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ 0 \\ 248{,}4 \\ -1307{,}6 \end{bmatrix}$$

4 Reglerentwurf

Mit dem zuvor berechneten Zustandsraummodell ist man nun in der Lage einen geeigneten Regler für den Ballbot zu entwerfen. Aufgabe des Reglers ist es, den Ballbot in seiner Ruhelage auszubalancieren. Der Regler wird mittels der Software MATLAB/Simulink entworfen.

Um einen passenden Regler für das Ballbot-Modell finden zu können, wird zuerst die Systemmatrix aus 3.46 analysiert. Die Systemmatrix besitzt folgende Eigenwerte:

(4.01)

$$eig(A) = \begin{bmatrix} 0 \\ -4761{,}1 \\ 7{,}4 \\ -7 \end{bmatrix}$$

Wie man aus obigen Werten erkennen kann, hat die Systemmatrix einen Eigenwert mit positivem Vorzeichen. Das heißt, dass unsere Strecke instabil ist. Bei solchen Strecken, die Eigenwerte „rechts“ aufweisen, kann vorteilhaft ein Zustandsregler zur Stabilisierung eingesetzt werden.

Damit man überhaupt einen Zustandsregler verwenden kann, muss das System steuerbar sein. Durch die Steuerbarkeitsmatrix

(4.02)

$$S_u = (b\ Ab\ A^2b \dots A^{n-1}b)$$

kann dies überprüft werden. Damit das Modell steuerbar ist, muss folgende Beziehung gelten:

(4.03)

$$\det(S_u) \neq 0$$

Führt man diese Überprüfung mit den vorliegenden Werten aus 3.46 für die Systemmatrix A und den Eingangsvektor b in MATLAB durch, so erhält man folgendes Ergebnis:

(4.04)

$$\det(S_u) = -2{,}8316e + 14$$

Somit ist bewiesen, dass das System steuerbar ist und man einen Zustandsregler verwenden kann.

4.1 Zustandsregler

Es gibt mehrere Methoden, einen Zustandsregler zu entwerfen. Zwei klassische Ansätze sind der Zustandsregler durch Eigenwertvorgabe nach Ackermann oder der linear-quadratische Zustandsregler, auch LQR-Regler genannt.

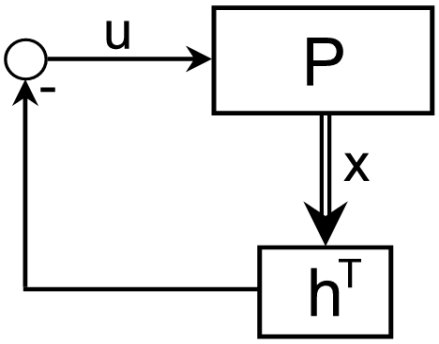


Abbildung 4.1: Struktur eines Zustandsreglers

In Abbildung 4.1 sieht man die Struktur eines Zustandsreglers. Wie man gut erkennen kann, werden die einzelnen Zustände zurückgeführt. Der Vorteil eines Zustandsreglers ist der, dass man mit relativ geringem Rechenaufwand den Regler auf einer Hardware implementieren kann. Man benötigt dazu nur einen Parameter, den Vektor h für die Zustandsrückführung. Mit der Zustandsrückführung kann man die Dynamik des Systems beeinflussen. Nachteil des Zustandsreglers ist aber, dass man ein mathematisches Modell der Strecke benötigt.

Beim Zustandsreglerentwurf durch Eigenwertvorgabe muss man die die Eigenwerte des geregelten Systems selber festlegen. Da es beim Ballbot-Modell nicht gleich ersichtlich ist, wo man die Eigenwerte platziert, hat es sich bewährt, einen linear-quadratischen Zustandsregler zu verwenden. Bei einem solchen Regler versucht man ein quadratisches Gütekriterium

(4.05)

$$J[u] = \int_0^\infty (x^T Q x + R u^2) dt$$

zu minimieren, deshalb auch der Name. Genauer gesagt, versucht man, dank einer Optimierung, den Rückführvektor h durch die Gewichtungsmatrix Q und den Skalar R so einzustellen, dass das quadratische Gütekriterium von Gleichung 4.05 minimal wird. Dieses integrale Gütekriterium bewertet sowohl den Verlauf der Zustandsgrößen als auch den der Stellgröße. Durch die Matrix Q lassen sich die einzelnen Zustandsgrößen gewichten, hingegen durch den Skalar R kann man die Stellgröße beeinflussen. Ergebnis ist ein Regelverhalten, das einen Kompromiss zwischen der Dynamik im Abbau von Zustandsfehlern und dem erforderlichen Stellaufwand darstellt.

Wie die Werte für Q und R gewählt werden, ist einem selbst überlassen. Es gibt keine genauen Regeln dafür, aber man muss auf folgende Dinge achten:

- Q muss symmetrisch, reell und positiv semidefinit sein

$$Q = Q^T \geq 0$$

- R muss symmetrisch, reell und positiv definit sein

$$R = R^T > 0$$

Dabei gilt die Voraussetzung, dass das Paar $(A, \bar{Q}')$ vollständig beobachtbar ist, wobei die Matrix $\bar{Q}'$ aus der Zerlegung

(4.06)

$$Q = \bar{Q}'\bar{Q}$$

der Gewichtungsmatrix Q hervorgeht. Diese Bedingung ist nicht erforderlich, wenn Q positiv definit gewählt wird [8].

In 4.07 sieht man ein Beispiel, wie eine solche Gewichtungsmatrix für das Ballbot-Modell z.B. aussehen könnte.

(4.07)

$$Q = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R = 0.0013$$

Die einzelnen Werte der Hauptdiagonalen der Gewichtungsmatrix sind für die Gewichtung der einzelnen Zustandsgrößen zuständig. Mit dem skalaren Wert R wird hingegen die Stellgröße beeinflusst. Je größer man R wählt, desto mehr wird die Stellgröße gewichtet und desto langsamer wird der Regler.

Führt man für obige Gewichtungsmatrix, die Zerlegung aus 4.06 durch, so erhält man für $\bar{Q}'$ folgende Matrix:

(4.08)

$$\bar{Q}' = \begin{bmatrix} \sqrt{3} & 0 & 0 & 0 \\ 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Überprüft man nun die vollständige Beobachtbarkeit des Paars $(A, \bar{Q}')$ durch Überprüfung des Ranges, so erhaltet man folgendes Ergebnis:

(4.09)

$$Rang\begin{bmatrix} \bar{Q}' \\ \bar{Q}'A \\ \bar{Q}'A^2 \\ \bar{Q}'A^3 \end{bmatrix} = 4$$

Wie man sieht ist der Rang gleich groß wie die Dimension des Zustandsraumes, demzufolge ist die vollständige Beobachtbarkeit des Paars $(A, \bar{Q}')$ bewiesen und man darf die Gewichtungsmatrix Q für weitere Berechnungen verwenden.

Um jetzt einen geeigneten Regler für das Ballbot-Modell finden zu können, müssen die Parameter aus 4.07 geändert werden und auf der Zielhardware ausgetestet werden. Je nachdem, wie stark gewisse Zustandsgrößen gewichtet werden bzw. der skalare Wert R gewählt wird, können dem Ballbot-Modell gewisse Eigenschaften wie Robustheit oder stationäre Genauigkeit aufgeprägt werden, wie man später in Kapitel 5 sehen wird.

Da der NXT-Baustein mit einer gewissen Abtastzeit arbeitet, ist es notwendig einen zeitdiskreten LQR-Zustandsregler zu entwerfen. Damit dies geschehen kann, müssen, wie in Abbildung 4.2 veranschaulicht, die Ein- bzw. Ausgangssignale der Strecke mit ein Halte- bzw. Abtastglied umgewandelt werden.

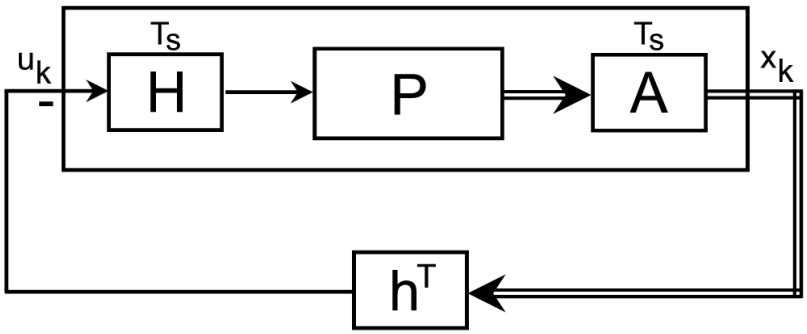


Abbildung 4.2: Abtastsystem bestehend aus einem Halteglied, einer zeitkontinuierlichen Strecke und einem Abtaster

Das zeitdiskrete Zustandsraummodell lässt sich wie folgt anschreiben:

(4.10)

$$\mathrm{x}_{k+1} = \Phi(T_s)x_k + h(T_s)u_k$$

$$y_k = c^T x_k$$

mit der Zustandsrückführung

(4.11)

$$u_k = -h^T x_k.$$

Das quadratische Gütekriterium sieht dann folgendermaßen aus:

(4.12)

$$J[u] = \sum_{k=0}^{\infty} {x_k}^T Q x_k + R {u_k}^2$$

In MATLAB/Simulink gibt es einen eigenen Befehl, welcher den Vektor h für die Zustandsrückführung berechnet, der wie folgt lautet:

$$lqrd(A, b, Q, R, Ts),$$

wobei A die Systemmatrix, b der Eingangsvektor, Q die Gewichtungsmatrix, R ein Skalar und Ts die gewünschte Abtastzeit ist. Führt man diesen Befehl in MATLAB mit den Werten aus 3.46 und 4.07 und einer Abtastzeit von 4 Millisekunden aus, so erhält man folgenden Vektor h für die Zustandsrückführung:

(4.13)

$$h^T = [-45{,}7306 \quad -95{,}9358 \quad -53{,}9445 \quad -10{,}2678]$$

4.2 Zustandsregler mit I-Anteil

Um die stationäre Genauigkeit des Systems zu verbessern, kann man einen Zustandsregler mit integrierendem Anteil verwenden, wie in Abbildung 4.3 dargestellt. Dabei ist die Position des Ballbots die Größe, die integriert wird.

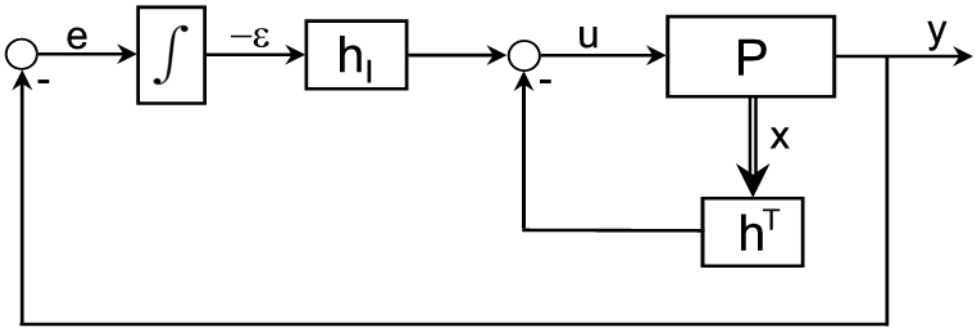


Abbildung 4.3: Struktur eines integrierenden Zustandsreglers

Um obige Struktur verwenden zu können, muss das klassische Zustandsraummodell

(4.14)

$$\frac{dx}{dt} = Ax + bu$$

$$y = c^T x$$

wie folgt angepasst werden, wobei die Ausgangsgröße des Reglers die Position des Ballbots ist.

Der neue Zustandsvektor lautet nun

(4.15)

$$\hat{x} = \begin{pmatrix} x \\ \varepsilon \end{pmatrix}$$

und der Eingang des Integrierers beträgt

(4.16)

$$\frac{d\varepsilon}{dt} = -e = -(-y) = y = c^T x.$$

Die neue Stellgröße lässt sich wie folgt beschreiben

(4.17)

$$u = -h^T x - h_I \varepsilon.$$

In Matrixschreibweise ergibt sich dann folgendes System:

(4.18)

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{d\varepsilon}{dt} \end{pmatrix} = \begin{pmatrix} A - bh^T & -bh_I \\ c^T & 0 \end{pmatrix} \begin{pmatrix} x \\ \varepsilon \end{pmatrix}$$

bzw.

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{d\varepsilon}{dt} \end{pmatrix} = \left(\begin{pmatrix} A & 0 \\ c^T & 0 \end{pmatrix} - \begin{pmatrix} b \\ 0 \end{pmatrix} \begin{pmatrix} h^T & h_I \end{pmatrix} \right) \begin{pmatrix} x \\ \varepsilon \end{pmatrix}$$

Um jetzt weiterhin den „lqrd“-Befehl in MATLAB verwenden zu können, müssen die Systemmatrix, der Eingangsvektor und der Ausgangsvektor folgendermaßen erweitert werden:

(4.19)

$$\hat{A} = \begin{pmatrix} A & 0 \\ c^T & 0 \end{pmatrix}$$

$$\hat{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

$$\hat{c}^T = \begin{pmatrix} c^T & 0 \end{pmatrix}$$

Berechnet man in MATLAB den Vektor h für die Zustandsrückführung mit der erweiterten Systemmatrix $\hat{A}$ und den erweiterten Eingangsvektor $\hat{b}$ aus 4.19 und verwendet für die Gewichtungsmatrix folgende Werte:

(4.20)

$$Q = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.00001 \end{bmatrix}$$

$$R = 0.0013$$

so erhält man folgendes Ergebnis:

(4.21)

$$h^T = [-45{,}7939 \quad -95{,}9542 \quad -53{,}9619 \quad -10{,}2711 \quad -0{,}0835]$$

Wie man gut aus dem Parametersatz 4.20 erkennen kann, wurde auch die Gewichtungsmatrix um ein Element erweitert. Durch dieses neue Element in der Matrix ist man in der Lage den Integrierer zu gewichten. Wie man sieht, wurde ein sehr kleiner Koeffizient gewählt, da der integrierende Zweig die Regelung nicht massiv beeinflussen soll, sondern lediglich die bleibende Regelabweichung beseitigen soll.

4.3 Simulation Balancierung

Mit dem in Abschnitt 3.7 ermittelten Parametern wird nun der integrierende Zustandsregler aus Kapitel 4.2 simuliert. In Abbildung 4.4 sieht man den Koppelplan für den Regler in Simulink, wobei die Strecke mittels einer MATLAB-Funktion implementiert ist und der Begrenzungsblock die Versorgungsspannung des NXT (±9V) berücksichtigt.

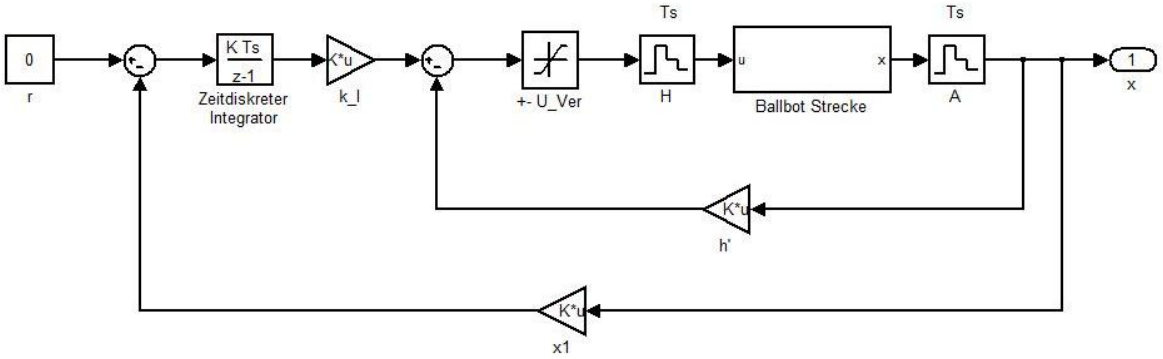


Abbildung 4.4: Simulink Koppelplan des integrierenden LQR-Zustandsreglers

Wenn man als Anfangsposition einen Kippwinkel von 1° wählt, so erhält man bei einer Abtastzeit von 4 Millisekunden folgende Simulationsergebnisse für die einzelnen Zustandsgrößen:

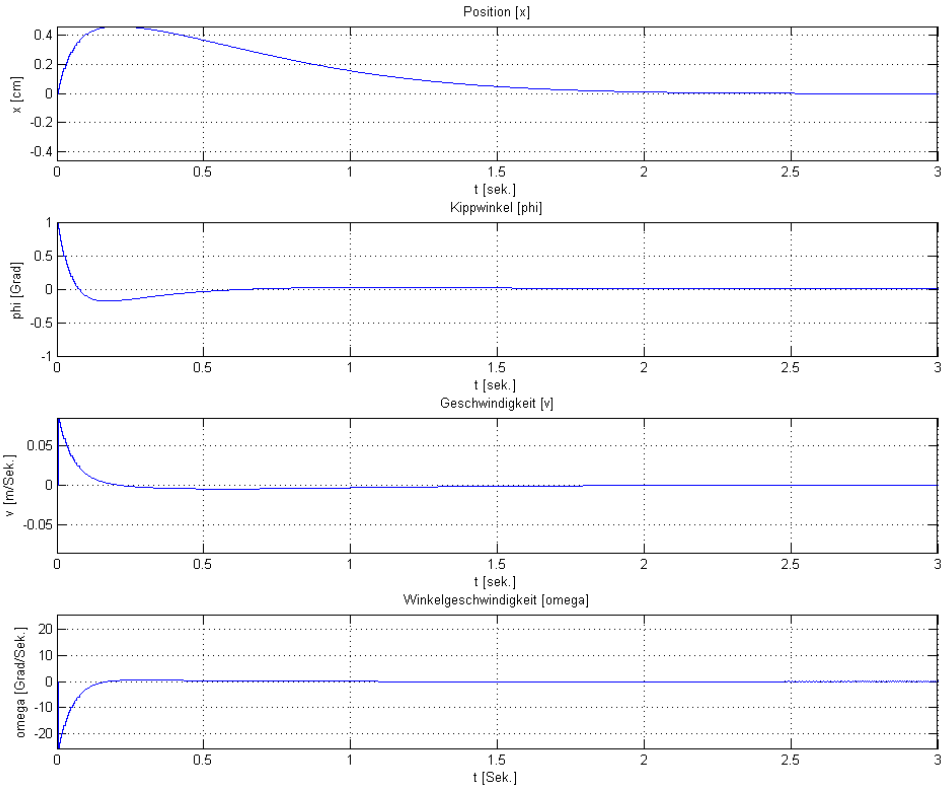


Abbildung 4.5: Grafische Darstellung der Zustandsgrößen bei einem Start-Kippwinkel von 1°

Wie man aus Abbildung 4.5 gut erkennen kann, schafft es der Regler ziemlich schnell, die Kippwinkel-Ausgangsposition auf Null zu bringen. Durch den integrierenden Anteil schafft man es, die bleibende Regelabweichung in der Position zu beseitigen. Später bei der Implementierung des Reglers auf der Zielhardware wird man sehen, dass dank des I-Anteils der Ballbot versucht, langsam in seine Ausgangsposition zurückzufahren. Auch die Stellgröße bleibt bei einer solchen Auslegung deutlich unter der Begrenzung von ±9V, siehe Abbildung 4.6.

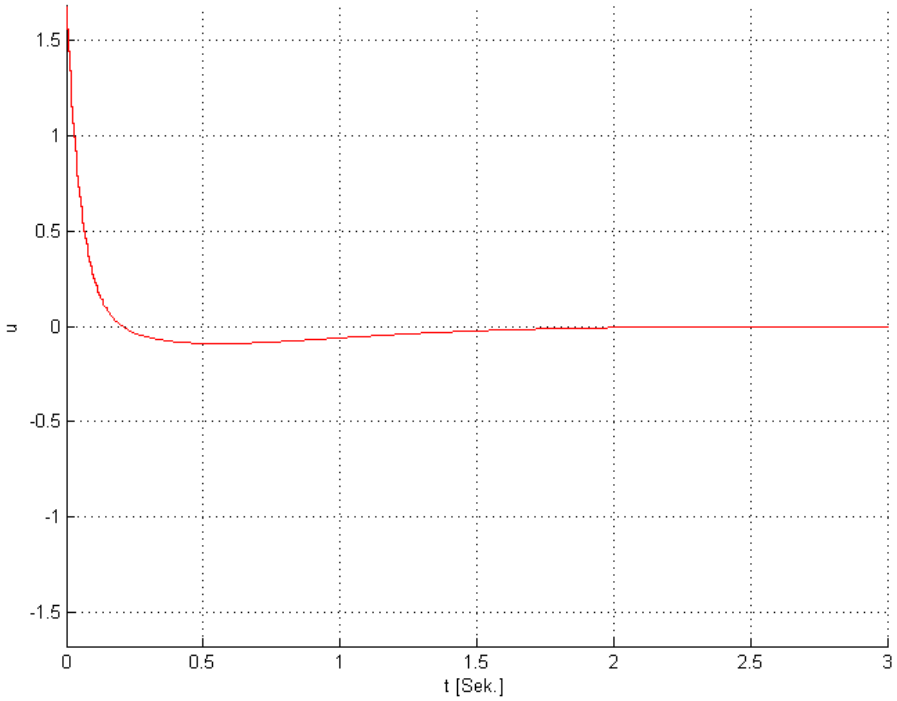


Abbildung 4.6: Grafische Darstellung der Stellgröße bei einem Start-Kippwinkel von 1°

4.4 Simulation Positionierung

Will man dem Ballbot eine Position in der XY-Ebene vorgeben, die dann angefahren werden soll, so liegt es nahe, eine Führungsregelung zu verwenden, da die Position des Ballbots eine Zustandsgröße des Systems ist. In Abbildung 4.7 ist die Struktur eines integrierenden Zustandsreglers mit Führungsregelung dargestellt.

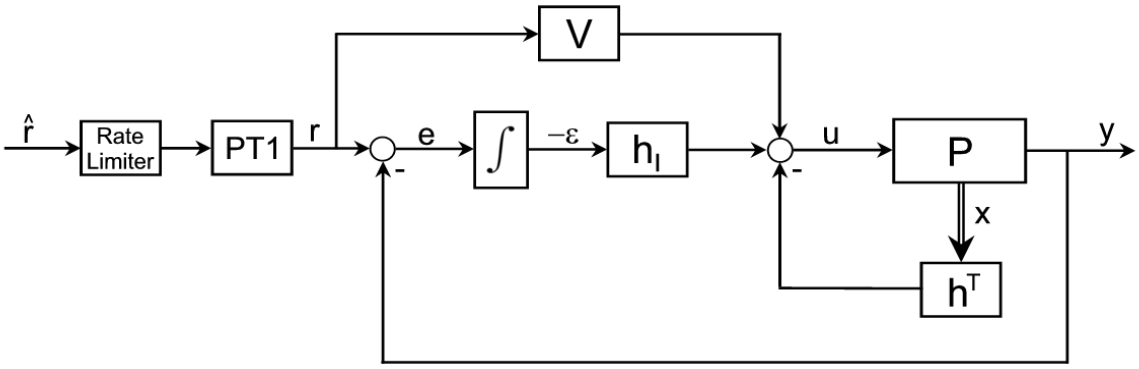


Abbildung 4.7: Struktur eines integrierenden Zustandsreglers mit Führungsregelung

Wie man aus obiger Abbildung gut erkennen kann, besteht die Regelkreisstruktur des integrierenden Zustandsreglers unter anderem aus einer Signalkonditionierung, bestehend aus „Rate Limiter“ und PT1-Glied, und einer Vorsteuerung V. Der Eingang des Integrierers lautet nun:

(4.22)

$$\frac{d\varepsilon}{dt} = -e = -(r - y) = y - r = c^T x - r$$

und die neue Stellgröße des Systems lässt sich wie folgt beschreiben

(4.23)

$$u = -h^T x - h_I \varepsilon + Vr.$$

Das dazugehörige Zustandsraummodell in Matrixschreibweise sieht dann wie folgt aus:

(4.24)

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{d\varepsilon}{dt} \end{pmatrix} = \begin{pmatrix} A - bh^T & -bh_I \\ c^T & 0 \end{pmatrix} \begin{pmatrix} x \\ \varepsilon \end{pmatrix} + \begin{pmatrix} bV \\ -1 \end{pmatrix} r$$

Schaltet man dem Regler eine sprungförmige Führungsgröße auf, ohne das Signal vorerst zu bearbeiten, so wird man gleich merken, dass der Regler nicht imstande ist, den Ballbot in eine stabile Position zu bringen. In Abbildung 4.8 ist dieses Verhalten deutlich zu erkennen. Grund dafür ist, dass der Ballbot einer so schnellen Änderung am Eingang nicht folgen kann. Um trotzdem eine Führungsgröße aufschalten zu können, muss das Signal zuerst gefiltert werden. Dazu bedient man sich eines sogenannten „Rate Limiters“ und eines PT1-Glieds. Dank dieser beiden Glieder wird ein „gemütlicher“ Führungsgrößenverlauf produziert. In Abbildung 4.9 sieht man die Verläufe der gefilterten und ungefilterten Signale. Wie man gut erkennen kann, wird aus dem Sollwertsprung dank des „Rate Limiters“ eine Rampe erzeugt. Durch das zusätzliche PT1-Glied werden noch die Kanten der Rampe geglättet. Durch diese Glättung will man einen sanften Start des Ballbots gewährleisten bzw. ein abruptes Anhalten vermeiden. Das verwendete PT1-Glied besitzt dabei folgende Übertragungsfunktion:

(4.25)

$$G(s) = \frac{1}{s+1}$$

Damit das Filter verwendet werden kann, muss es noch mit einer Abtastzeit von 4 Millisekunden zeitdiskretisiert werden.

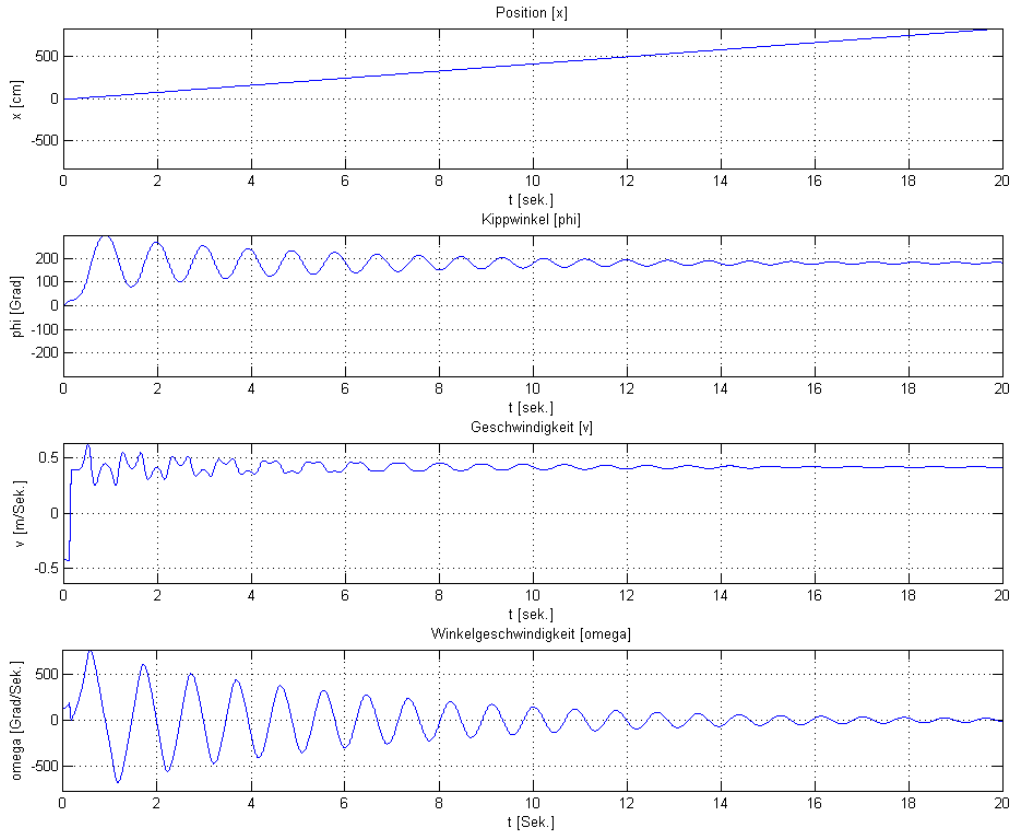


Abbildung 4.8: Grafische Darstellung der Zustandsgrößen bei Aufschalten einer ungefilterten Führungsgröße

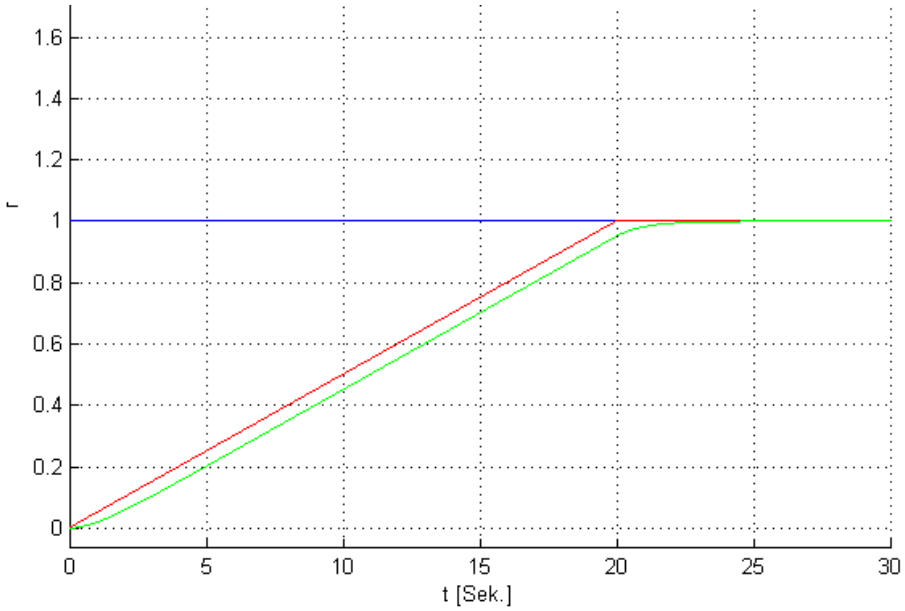


Abbildung 4.9: Grafische Darstellung der Führungsgröße; blau: ungefiltertes Signal, rot: Filterung mit „Rate Limiter“, grün: Filterung mit "Rate Limiter" und PT1-Glied

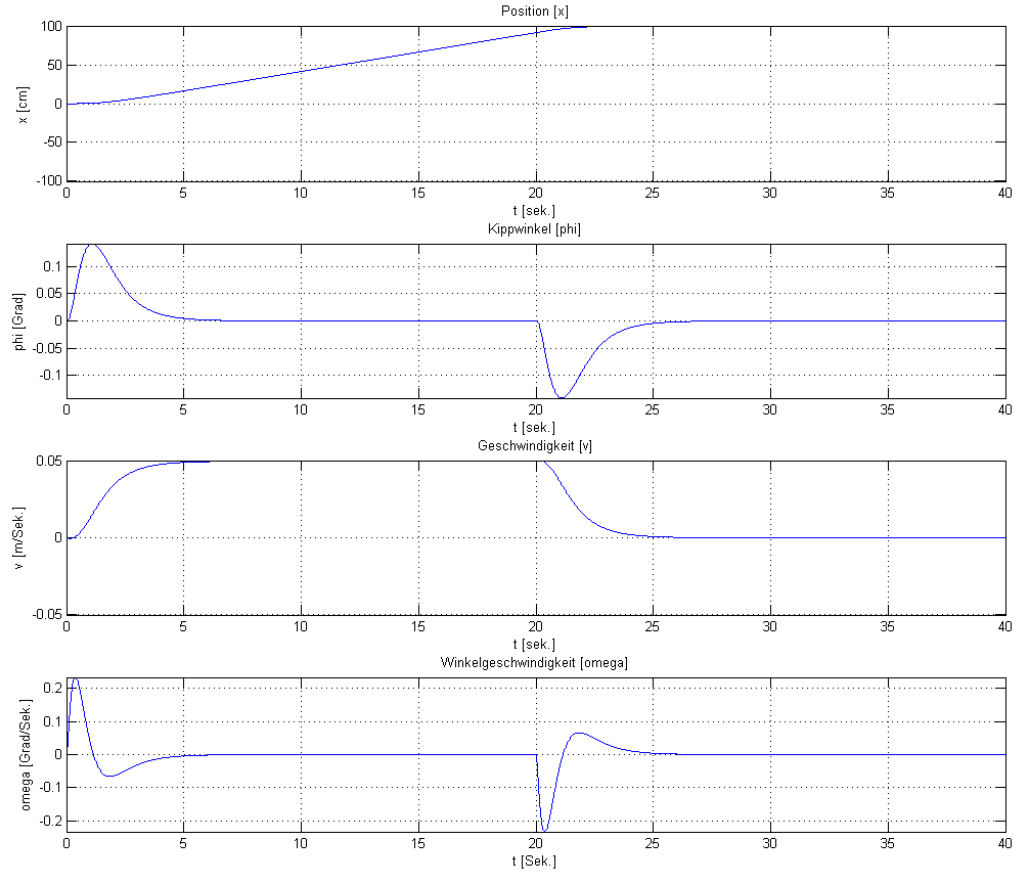


Abbildung 4.10: Grafische Darstellung der Zustandsgrößen bei Aufschalten einer gefilterten Führungsgröße

In Abbildung 4.10 ist der zeitliche Verlauf der Zustandsgrößen bei Verwendung des „Rate Limiters“ und des PT1-Gliedes als Filter für die Führungsgröße und bei Aufschalten einer konstanten Führungsgröße von einem Meter abgebildet. Man erkennt deutlich, dass dank der Filterung der Regler es nun schafft, den Ballbot zu stabilisieren und gleichzeitig die gewünschte Position zu erreichen.

In Abbildung 4.11 ist die dazugehörige Stellgröße ersichtlich. Auch hier wiederum ist deutlich zu erkennen, dass der Regler bei einer solchen Konfiguration nicht in die Begrenzung von ±9V kommt.

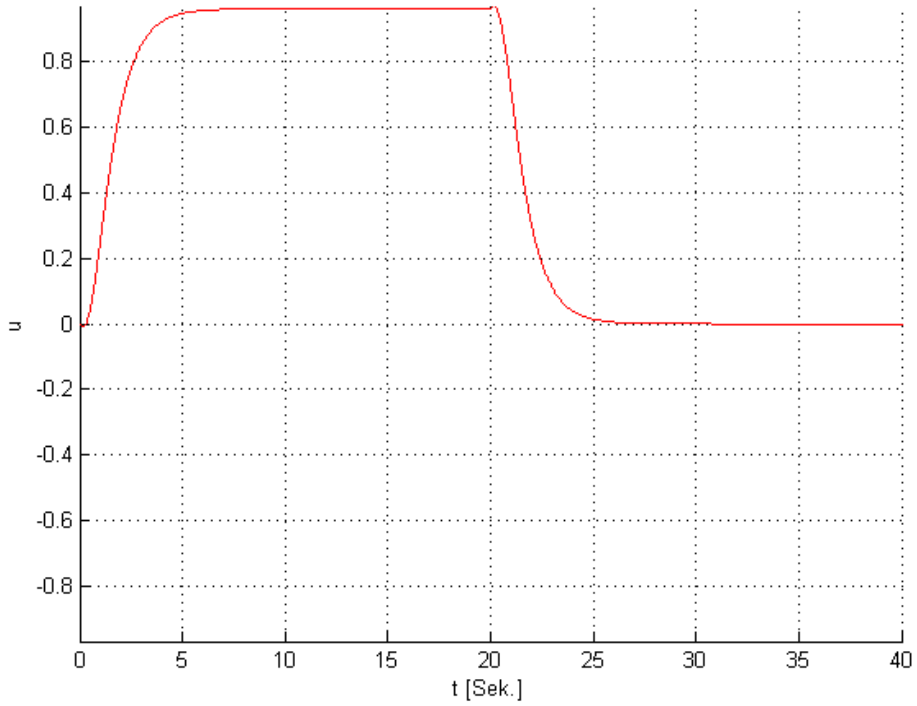


Abbildung 4.11: Grafische Darstellung der Stellgröße bei Aufschalten einer gefilterten Führungsgröße

Dank der Filterung der Führungsgröße wird man später in der Lage sein, dem Ballbot eine Position in der XY-Ebene vorzugeben, die dann angefahren wird. Man wird aber feststellen, dass gewisse Änderungen an den Parametern vorgenommen werden müssen, um diese Funktionalität des Ballbots zu gewährleisten.

5 Implementierung auf der Zielhardware

Der in Kapitel 4 beschriebene linear-quadratische Regler wird nun für die Zielhardware, den programmierbaren LEGO-Baustein NXT, angepasst und auf diese überspielt. Damit dies geschehen kann, muss zuerst für MATLAB/Simulink eine Toolbox installiert werden, welche die einzelnen Komponenten des LEGO Mindstorms-Baukastens unterstützt. Die Toolbox steht auf der Mathworks-Homepage zur Verfügung und nennt sich „Embedded Coder Robot for LEGO Mindstorms NXT“ bzw. „ECRobot“. Diese Toolbox ermöglicht das Erzeugen von Modellen in Simulink und das automatische Generieren von Maschinencode für den LEGO Mindstorms NXT-Baustein. Damit MATLAB/Simulink den Code generieren kann, benötigt man zusätzlich weitere Programme von Drittanbietern, die vorher installiert werden müssen. Die genaue Installationsanleitung der einzelnen Komponenten ist im Anhang B beschrieben. Voraussetzung für die Installation ist eine 7.2 MATLAB-Version (R2006a) mit dem Simulink Coder, Embedder Coder und Simulink. Ab der MATLAB-Version R2012a ist die LEGO Mindstorms NXT-Toolbox schon Bestandteil der MATLAB/Simulink-Toolboxen und es müssen keine weiteren Toolboxen mehr installiert werden. Nachfolgend eine kurze Auflistung der Programme, die installiert werden müssen:

1. *Cygwin*: Eine Unix ähnliche Umgebung auf Windows, erforderlich für GNU ARM Compiler
2. *GNU ARM Compiler*: Kompiliert C-Code für den ARM-Prozessor im LEGO NXT-Baustein
3. *LEGO Mindstorms NXT Treiber*: Treiber für die USB-Kommunikation mit dem NXT
4. *NeXTTool*: Kommandozeile, benötigt man für die Übertragung von Dateien vom PC auf den LEGO NXT
5. *NXT Enhanced standard firmware*: Ersatz Firmware für den LEGO NXT, die es erlaubt ARM-Programme zusätzlich zu den Standard NXT-Programmen auszuführen.
6. *nxtOSEK*: Ein Echtzeit-Betriebssystem für den LEGO Mindstorms NXT

Damit die kompilierten Simulink Programme auf den NXT-Baustein abgespielt werden können, muss auf dem NXT ein eigenes Betriebssystem installiert sein, das sich „nxtOSEK“ nennt. „nxtOSEK“ ist ein Echtzeitbetriebssystem für den LEGO Mindstorms NXT-Baustein und ermöglicht es, den Baustein in ANSI-C/C++ zu programmieren. Die Installation des Betriebssystems erfolgt automatisch, sobald man das kompilierte Programm auf den NXT-Baustein lädt.

5.1 „ECRobot“ Toolbox

Wurde die Toolbox korrekt installiert, wie in Anhang B beschrieben, so erscheint im Simulink Library-Browser ein neuer Unterpunkt mit dem Namen „ECRobot NXT Blockset“. In diesem Menüpunkt findet man nun alle wichtigen Blöcke, die man für die Programmierung des NXT-Bausteins benötigt. Die Sensoren und Aktuatoren besitzen immer einen „Interface“-Block und einen „Read“- oder „Write“-Block. Der „Interface“-Block ist eine Art Schnittstellenblock, den man benötigt, um die einzelnen Komponenten, die für das Modell verwendet werden, zu definieren. Unter anderem wird hier der NXT-Port angegeben, an den der Sensor bzw. Aktuator angeschlossen ist. Mit den „Read“-bzw. „Write“-Blöcken hat man dann Zugriff auf die einzelnen Sensoren bzw. Aktuatoren. In der Simulation sind diese Blöcke nur Platzhalter, aber sie werden verwendet, um eine entsprechende Programmierschnittstelle in den generierten Code zu implementieren.

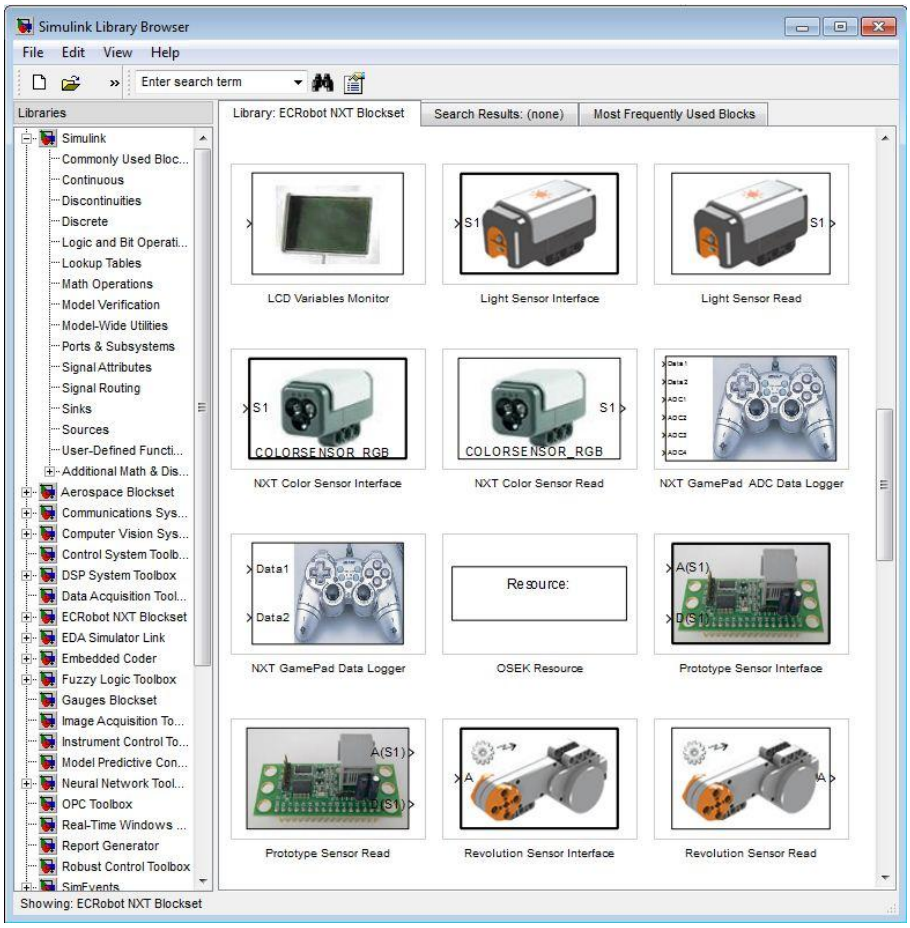


Abbildung 5.1: ECRobot Toolbox im Simulink Library-Browser

Nachfolgend sind die Blöcke beschrieben, die für den Ballbot benötigt werden. Für eine ausführlichere Beschreibung der Toolbox samt ihren Blöcken wird auf Quelle [5] verwiesen.

Gyro Sensor-Blöcke:

Abbildung 5.2: Gyro Sensor-Blöcke

Die Gyro Sensor-Blöcke werden verwendet, um die Winkelgeschwindigkeit zu messen. Mit dem „Read“-Block können die Gyro-Daten ausgelesen werden. Der Wert, der dabei ausgelesen wird, ist der Wert, den der Analog/Digital-Konverter des Sensors liefert. Wie man aus diesem Wert ein brauchbares Sensorsignal in Grad/Sekunde bekommt, wird in Abschnitt 5.2.1 beschrieben. Der verwendete Datentyp ist ein uint16.

Compass Sensor-Blöcke:

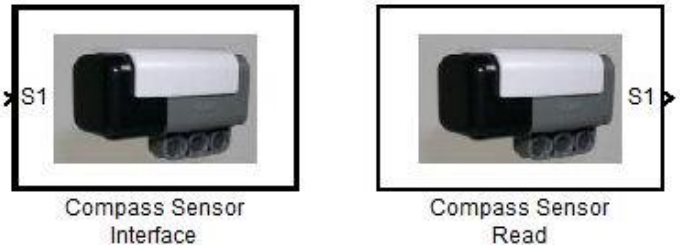


Abbildung 5.3: Compass Sensor-Blöcke

Die Compass Sensor-Blöcke werden verwendet, um die exakte Position des Ballbots zu bestimmen. Mit dem „Read“-Block kann die Position des Ballbots in Grad ausgelesen werden, wobei 0° Norden entspricht, 90° Osten, 180° Süden und 270° Westen.

Revolution Sensor-Blöcke:

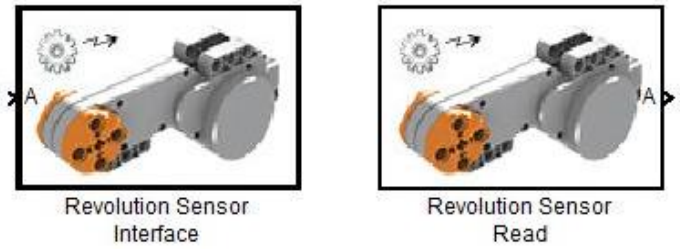


Abbildung 5.4: Revolution Sensor-Blöcke

Die Revolution Sensor-Blöcke werden verwendet, um die Umdrehung eines NXT-Servomotors zu messen. Mit dem „Read“-Block können die Daten des Motors ausgelesen werden. Der dabei verwendete Datentyp ist ein int32.

Servo Motor-Blöcke:

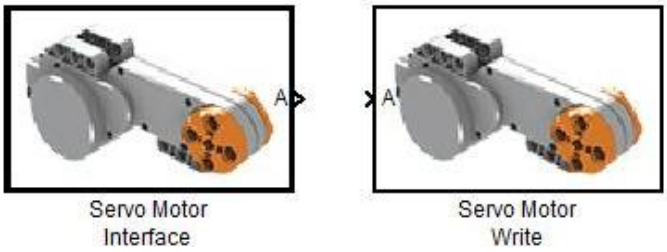


Abbildung 5.5: Servo Motor-Blöcke

Die Servo Motor-Blöcke werden verwendet, um einen Servomotor zu steuern. Mit dem „Write“-Block können die Servomotoren angesteuert werden, dabei kann ein Wert zwischen -100 und 100 angegeben werden. Der verwendete Datentyp ist ein int8.

Battery Voltage-Blöcke:

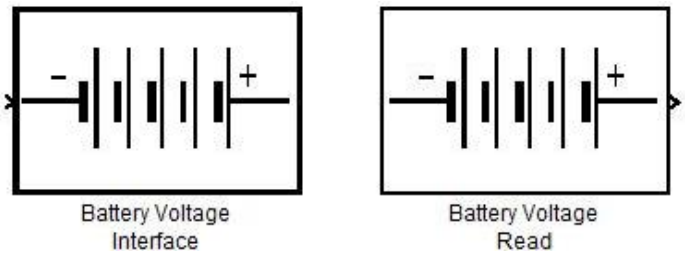


Abbildung 5.6: Battery Voltage-Blöcke

Die Battery Voltage-Blöcke werden verwendet, um die Batteriespannung des NXT zu überprüfen. Durch den „Read“-Block kann die Batteriespannung ausgelesen werden. Die Spannung wird dabei in mV angezeigt. Der verwendete Datentyp ist ein uint16.

System Clock-Blöcke:

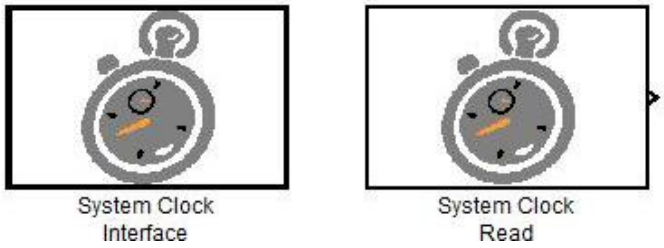


Abbildung 5.7: System Clock-Blöcke

Die Sysem Clock-Blöcke werden verwendet, um die System-Zeit auszulesen. Zu beachten ist, dass der Zähler anfängt zu zählen, sobald der NXT-Baustein eingeschaltet wird und nicht, sobald das „ECRobot“-Programm gestartet wird. Die Zeit wird in Millisekunden angezeigt und der verwendete Datentyp ist ein uint32.

LCD Variables Monitor-Block:

Abbildung 5.8: LCD Variables Monitor-Block

Durch diesen Block ist man in der Lage bis zu 16 Integer Werte am Display anzuzeigen. Der verwendete Datentyp ist ein int32.

Exported Function-Calls Scheduler-Block:

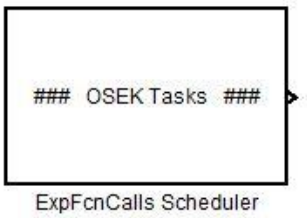


Abbildung 5.9: Exported Function-Calls Scheduler-Block

Durch diesen Block ist man in der Lage mehrere Funktionen gleichzeitig auszuführen und man kann jeweils die Laufzeiten der einzelnen Funktionen angeben. Zusätzlich kann man auch noch eine Initialisierungsfunktion definieren, die gleich am Anfang einmalig ausgeführt wird. Im Großen und Ganzen ist dieser Block nichts anderes als ein Scheduler.

NXT GamePad ADC Data Logger-Block:

Abbildung 5.10: NXT GamePad ADC Data Logger-Block

Durch diesen Block ist man in der Lage den NXT per Bluetooth zu steuern bzw. Daten abzuspeichern. Da der Ballbot sich autonom ausbalanciert bzw. bewegt, ist für diese Arbeit nur die Funktion für die Daten-Abspeicherung von Relevanz. Man kann bis zu sechs Signale abspeichern, wobei zwei Signale vom Typ int8 sind und die anderen vier vom Typ int16. Zu den sechs Signalen werden zudem noch die Zeit, die Batteriespannung, und die Umdrehung der drei Motoren mit abgespeichert. Die einzelnen Werte werden per Bluetooth an einen Rechner geschickt und in einer CSV-Datei abgespeichert. Ein Beispiel einer solchen CSV-Datei ist in Abbildung 5.11 zu sehen. Durch diese Funktion ist man dann imstande, das Verhalten verschiedener NXT-Anwendungen später am Rechner zu analysieren.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Time	Data1	Data2	Battery	Motor Rev A	Motor Rev B	Motor Rev C	ADC S1	ADC S2	ADC S3	ADC S4	I2C
2	0	-3	0	8142	0	0	0	0	-12	0	4	-1
3	4	-3	0	8142	0	0	0	-1	-12	0	4	-1
4	8	-3	2	8142	0	0	0	-1	-12	0	14	-1
5	12	-3	3	8142	0	0	0	-1	-12	0	14	-1
6	16	-3	5	8100	0	0	0	-1	-12	0	24	-1
7	20	-3	3	8100	0	0	0	-1	-12	0	14	-1
8	24	-3	5	8100	0	0	0	-1	-12	0	24	-1
9	28	-4	5	8142	0	0	0	-1	-12	0	24	-1
10	32	-4	7	8142	0	0	0	-1	-12	0	34	-1
11	36	-4	8	8156	0	0	0	-1	-12	0	34	-1
12	40	-6	8	8156	0	0	0	-1	-22	0	34	-1

Abbildung 5.11: Beispiel einer CSV-Datei für die Datensicherung

Um diesen Block verwenden zu können, muss eine aktive Bluetooth-Verbindung zwischen dem NXT-Baustein und einem Rechner bestehen. Eine detaillierte Beschreibung, wie man die Bluetooth-Verbindung einrichtet, befindet sich in Anhang C.

Die einzelnen Sensoren und Aktuatoren arbeiten mit unterschiedlichen Abtastraten. Damit der Regler korrekt arbeitet, wird die Abtastrate des Gyroskopsensors für alle Blöcke in Simulink verwendet. In unserem Fall wäre es eine Abtastrate von 300, d.h. alle 3,3 Millisekunden kann der Gyroskopsensor einen neuen Messwert liefern. Da der NXT in Millisekunden-Schritten arbeitet, wird für das gesamte Ballbot-System eine Abtastzeit von vier Millisekunden gewählt. In folgender Tabelle sind noch einmal die verwendeten Sensoren und Aktuatoren mit den einzelnen Abtastraten aufgelistet:

Sensor / Aktuator	Ausgang	Einheit	Max. Abtastrate [1/Sek]
Winkelsensor	Winkel	Grad	1000
Kompasssensor	Position	Grad	100
Gyroskopsensor	Winkelgeschwindigkeit	Grad/Sek.	300
DC Servomotor	PWM	%	500

Tabelle 5.1: Auflistung der verwendeten Sensoren und Aktuatoren mit den dazugehörigen Abtastraten

Wie man gerade gesehen hat, arbeiten alle NXT-Blöcke mit Integer-Werten. Um die Rechengenauigkeit zu erhöhen, wird intern mit Fließkommazahlen gerechnet, dazu müssen die Signale, die von den einzelnen Sensoren kommen bzw. den Aktuatoren zugeschalten werden, konvertiert werden. Dafür gibt es in Simulink einen eigenen Baustein, den „Data Type Conversion“-Block.

5.2 Simulink Koppelplan

Im folgenden Kapitel werden die einzelnen Ebenen des Simulink Koppelplans, die für den Ballbot notwendig sind, beschrieben. Ein paar Anregungen zum Aufbau und zur Struktur der einzelnen Koppelpläne wurden aus [7] entnommen.

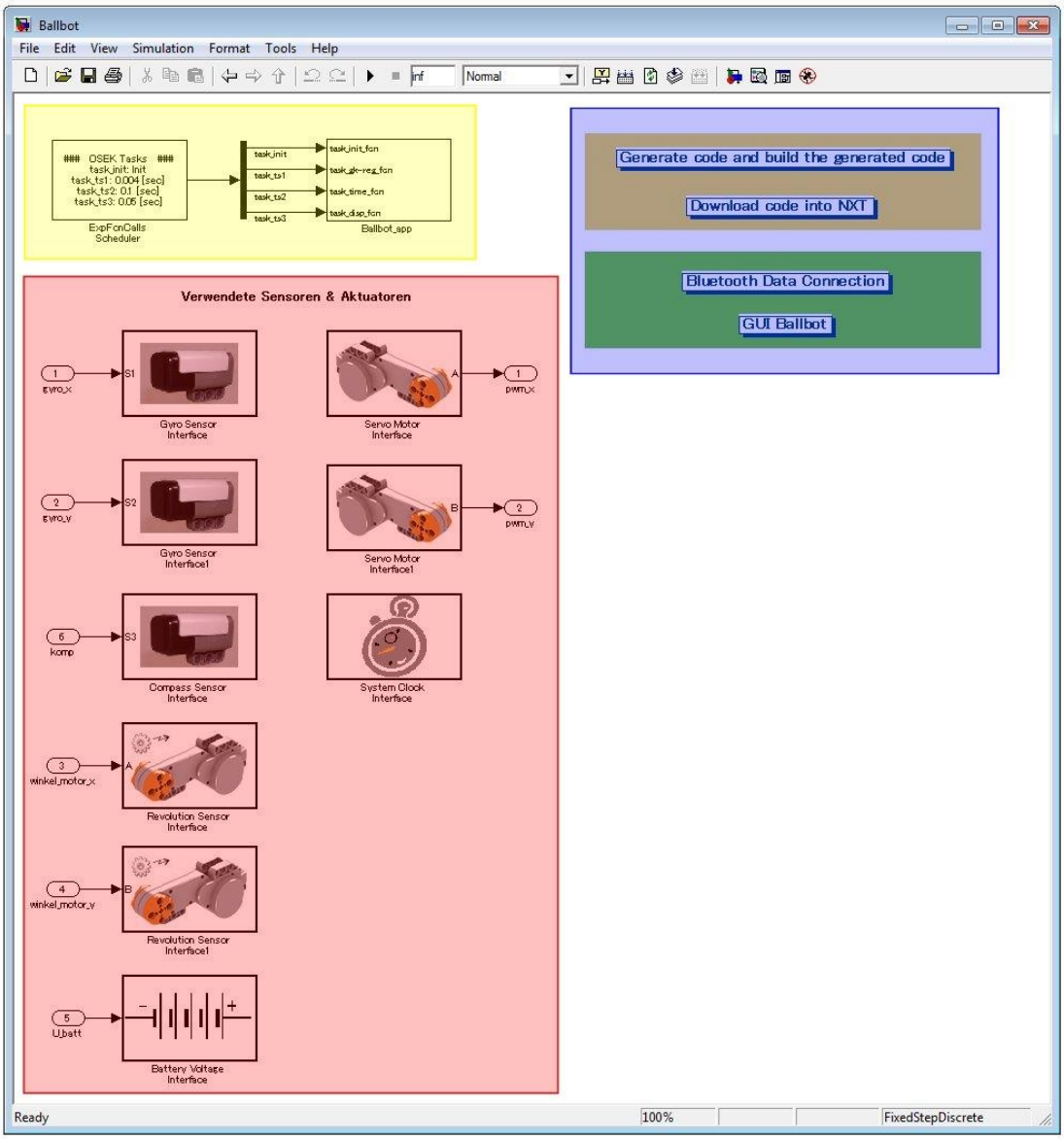


Abbildung 5.12: Simulink Koppelplan, oberste Ebene

Die erste Ebene des Koppelplans kann man in drei große Bereiche unterteilen, siehe Abbildung 5.12. Im roten Bereich sind alle Interface-Blöcke aufgelistet, welche später in den anderen Ebenen des Koppelplans benötigt werden. Im blauen Bereich kann man den Koppelplan kompilieren und per USB auf den NXT-Baustein laden. Des Weiteren kann man in diesem Bereich, per Bluetooth, Daten vom NXT an den Rechner senden und sie grafisch auswerten. Im gelben Bereich befindet sich das eigentliche Hauptprogramm. Im „ExpFcnCalls Scheduler“-Block sind die einzelnen Funktionen mit ihren Laufzeiten definiert. Die eigentlichen Funktionen befinden sich im Subsystem „Ballbot_app“.

Im Subsystem „Ballbot_app“ befinden sich 4 Funktionen: „task_init“, „task_ts1“, „task_ts2“ und „task_ts3“. In dieser zweiten Ebene werden auch alle globalen Variablen mit ihren Anfangswerten definiert, die für die Regelung des Ballbots benötigt werden.

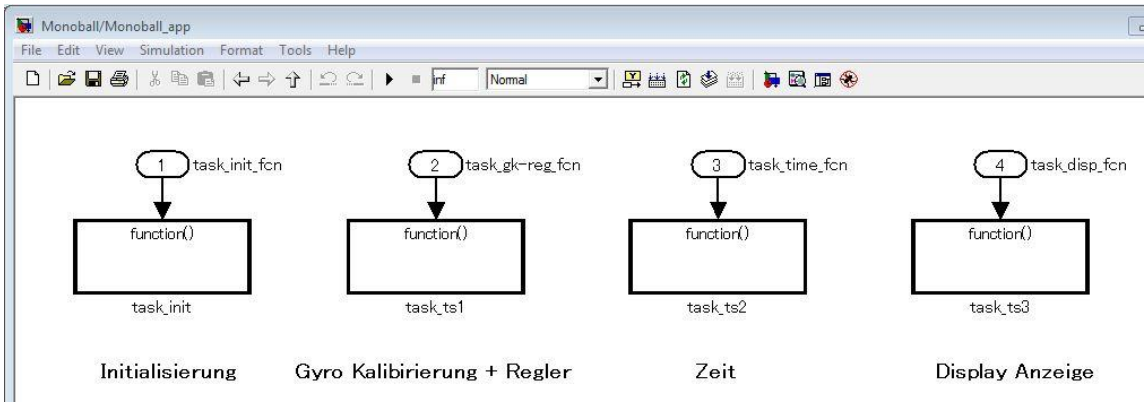


Abbildung 5.13: Zweite Ebene mit den dazugehörigen Funktionen für die Ballbot-Regelung

„task_init“:

Diese Funktion wird zum Initialisieren benötigt. Sie wird gleich bei Programm-Start ausgeführt. Es werden dabei die Anfangswerte der beiden Gyroskopsensoren und die System-Zeit in globalen Variablen abgespeichert. Es ist wichtig, die System-Zeit bei Programm-Start in eine globale Variable abzuspeichern, da der „System Clock“-Baustein anfängt zu zählen, sobald der NXT-Baustein eingeschalten wird.

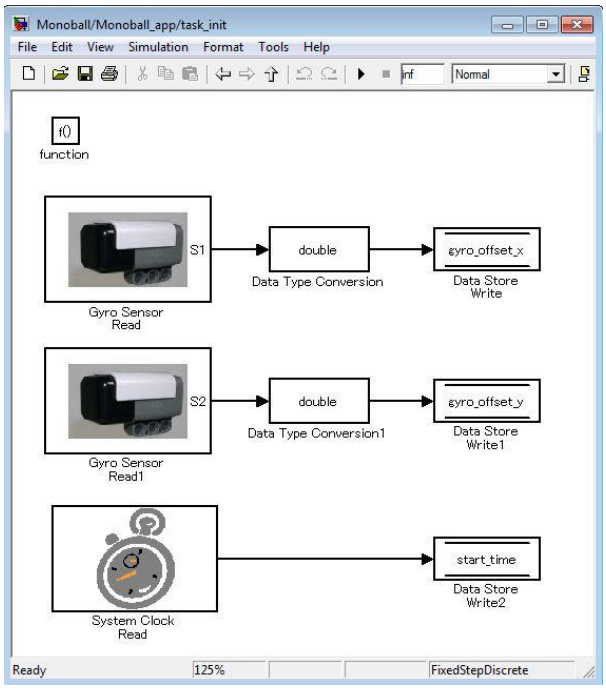


Abbildung 5.14: "task_init"-Funktion für die Initialisierung

„task_ts1“:

Das gesamte Ballbot Programm kann in vier Blöcke unterteilt werden: die Initialisierung die in der „task_init“-Funktion beschrieben ist, die Kalibrierung der Sensoren, den Regler und eine Funktion, die den Ballbot stoppt, falls er umkippt. Diese drei letzten Blöcke sind in der Funktion „ task_ts1“ implementiert. Durch einen „Switch Case“-Block kann jeweils auf einen dieser Blöcke zugegriffen werden. Diese Funktion wird alle 4 Millisekunden vom „ExpFcnCalls Scheduler“-Block in der obersten Ebene aufgerufen.

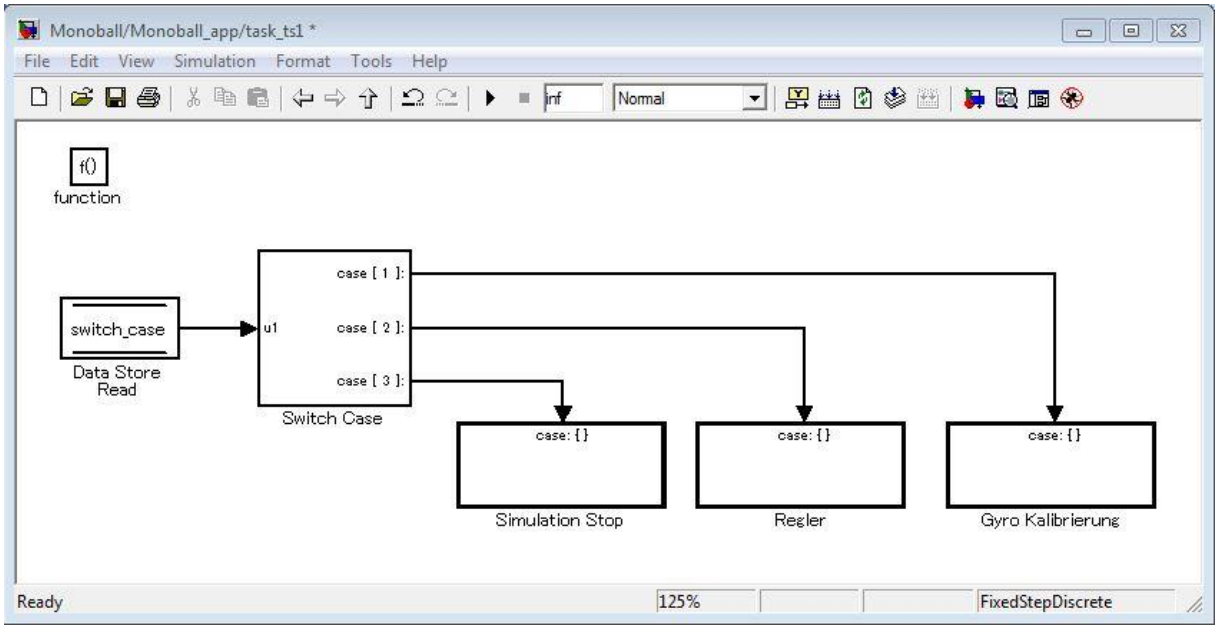


Abbildung 5.15: "task_ts1"-Funktion

„task_ts2“:

Diese Funktion ist für die korrekte zeitliche Ausführung der drei Blöcke, die in „task_ts1“ implementiert sind, verantwortlich und wird alle 100 Millisekunden aufgerufen. Sozusagen wird durch diese Funktion der „Switch Case“-Block in „task_ts1“ angesteuert. Sobald das Programm gestartet wird, werden eine Sekunde lang die Sensoren kalibriert. Anschließend fängt der Regler an zu arbeiten. Zugleich wird durch eine MATLAB-Funktion ständig nachgefragt, ob sich die Stellgröße für einen längeren Zeitraum in der Begrenzung befindet. Ist dies der Fall, so wird die Simulation gestoppt.

„task_ts3“:

Diese Funktion wird vom „ExpFcnCalls Scheduler“-Block alle 100 Millisekunden aufgerufen und ist für die Display-Anzeige zuständig. Zu beachten ist, dass die LCD-Anzeige nur int32-Werte ausgeben kann, deshalb müssen vorher alle Werte in diesem Datenformat konvertiert werden.

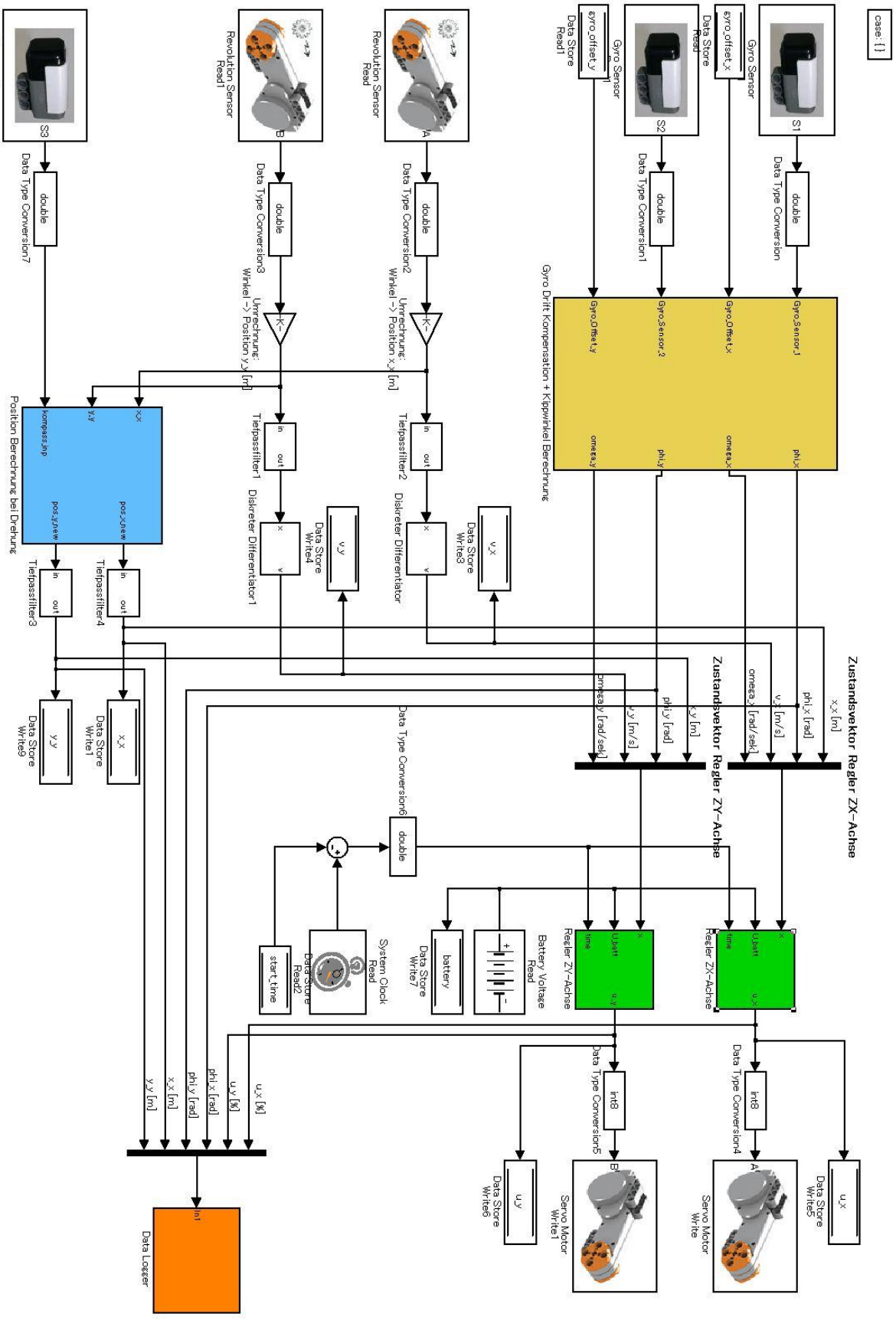


Abbildung 5.16: Simulink Koppelplan der Ballbot Regelung

Das eigentliche Hauptprogramm, welches für die Regelung des Ballbots zuständig ist, befindet sich in der Funktion „task_ts1“ im Block „Regler“. In Abbildung 5.16 sieht man den dazugehörigen Koppelplan.
Ein Teil des Koppelplans beschäftigt sich mit der Erzeugung des Zustandsvektors für den Regler. Aus den Sensordaten des Winkelsensors im DC-Servomotor wird die Position des Ballbots berechnet, welche zugleich die erste Zustandsgröße des Reglers ist. Differenziert man die Position, so erhält man die Geschwindigkeit des Ballbots bzw. die dritte Zustandsgröße. Das Differential wird mittels der Rückwärts-Differenzen-Methode ermittelt. Durch den Gyroskopsensor erhält man die vierte Zustandsgröße, die Kippwinkelgeschwindigkeit des Ballbots. Integriert man diese Größe mittels des Euler-vorwärts-Verfahrens, so erhält man den Kippwinkel bzw. die letzte notwendige Zustandsgröße für den Zustandsvektor. Diese vier Zustandsgrößen werden durch einen Multiplexer zusammengeführt und stehen dann dem Regler als Vektor zur Verfügung.
In Abbildung 5.16 sind auch deutlich die beiden Regler für die beiden Achsen zu erkennen. Die Sensordaten werden jeweils für die ZX- und ZY-Richtung eingelesen und in zwei unterschiedlichen Programmzweigen verarbeitet. Die daraus berechneten Stellgrößen werden dann an die dazugehörigen Motoren aufgeschaltet.

5.2.1 Filter

Leider ist der Gyroskopsensor mit einem Offset und einer Drift versehen. Bevor die Sensordaten verwendet werden können, muss zuerst der Offset und die Drift so gut wie möglich beseitigt werden. Der Gyroskopsensor liefert sogenannte „RAW“-Werte. Diese Werte sind diejenigen Werte, die am Ausgang des internen Analog/Digital-Konverters anliegen. Durch die Analog/Digital-Konverter Quantifizierung kommt es im stationären Zustand zu Schwankungen am Ausgang des Sensors. Meistens springt der Ausgang des Gyroskopsensors zwischen zwei Werten, siehe Abbildung 5.17.

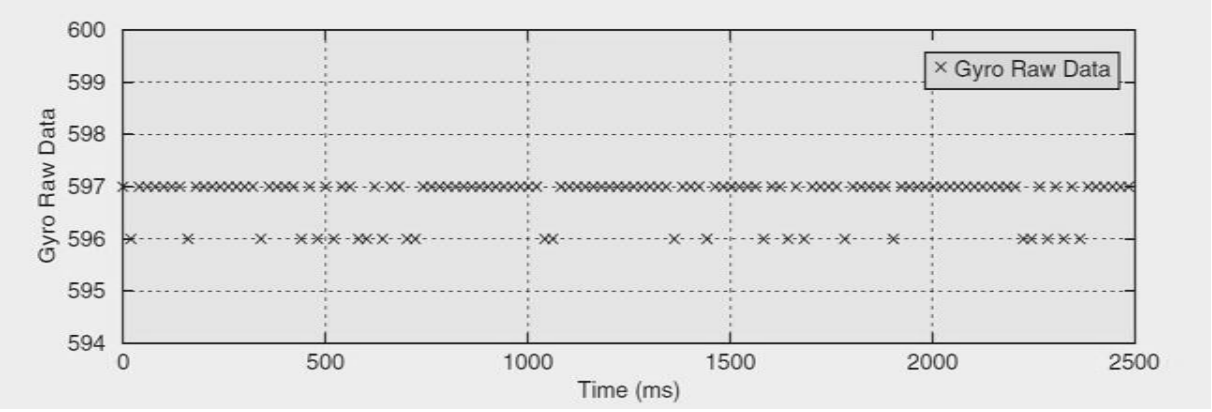


Abbildung 5.17: Ausgangswerte des Analog/Digital-Konverters am Gyroskopsensor bei Stillstand des Sensors[15]

[15] Quelle: [1]

Integriert man nun diese Werte auf, um den Kippwinkel zu erhalten, so wird der Winkel im stationären Zustand wegdriften. Man erhält also keinen brauchbaren Kippwinkel für den Regler. Nun gilt es dieses Driften zu beseitigen bzw. zu kompensieren. Bevor man aber das Drift-Problem löst, muss zuerst noch aus den „RAW"-Daten der zugehörige Wert in Grad berechnet werden. Wie man aus Abbildung 5.17 sieht, ist der Ausgangswert des Gyroskopsensors ungleich Null, er besitzt ein gewisses Offset. In der Funktion „task_ts1", siehe Abbildung 5.15, im Block „Gyro Kalibrierung", werden eine Sekunde lang die Werte des Gyroskopsensors gefiltert. Durch diese Filterung erhält man einen konstanten Wert, der dem tatsächlichen Sensorwert nahe kommt. Diese Filterung entspricht einer gleitenden Mittelwertbildung. Zieht man nun dem Ausgangssignal des Sensors dieses gefilterte Signal ab, so erhält man die Kippwinkelgeschwindigkeit in Grad pro Sekunde. Nun muss noch die Sensordrift kompensiert werden. Dafür gibt es in der Funktion „task_ts1" im Block „Regler" ein eigenes Subsystem („Gyro Drift Kompensation + Kippwinkel Berechnung"), siehe Abbildung 5.16, in dem ein zeitdiskreter Tiefpassfilter mit folgender Übertragungsfunktion implementiert ist:

$$Y(z) = H(z) * U(z) \tag{5.01}$$

$$H(z) = \frac{Y(z)}{U(z)} = \frac{1-a}{1-a*z^{-1}} \tag{5.02}$$

Die dazugehörige Differenzengleichung lautet:

$$y(n) = a * y(n-1) + (1-a) * u(n) \tag{5.03}$$

Durch diesen Filter und die richtige Wahl des Filterkoeffizienten a bekommt man die Drift einigermaßen in den Griff, so dass man den Sensorwert für die Regelung benutzen kann.

Ein weiteres Signal, das durch einen Tiefpass gefiltert werden muss, ist das Eingangssignal des Differentiators, für die Berechnung der Geschwindigkeit des Ballbots. Der Filter befindet sich in der Funktion „task_ts1" im Block „Regler" und besitzt folgende Übertragungsfunktion:

$$H(z) = \frac{0.1}{1-0.9*z^{-1}} \tag{5.04}$$

Dank dieser Filterung enthält das Positionssignal keine Sprünge mehr, siehe Abbildung 5.18. Führt man nun dieses gefilterte Signal dem Differentiator zu, so erhält man ein geglättetes Geschwindigkeitssignal, welches dann problemlos für die Regelung verwendet werden kann, siehe Abbildung 5.19.

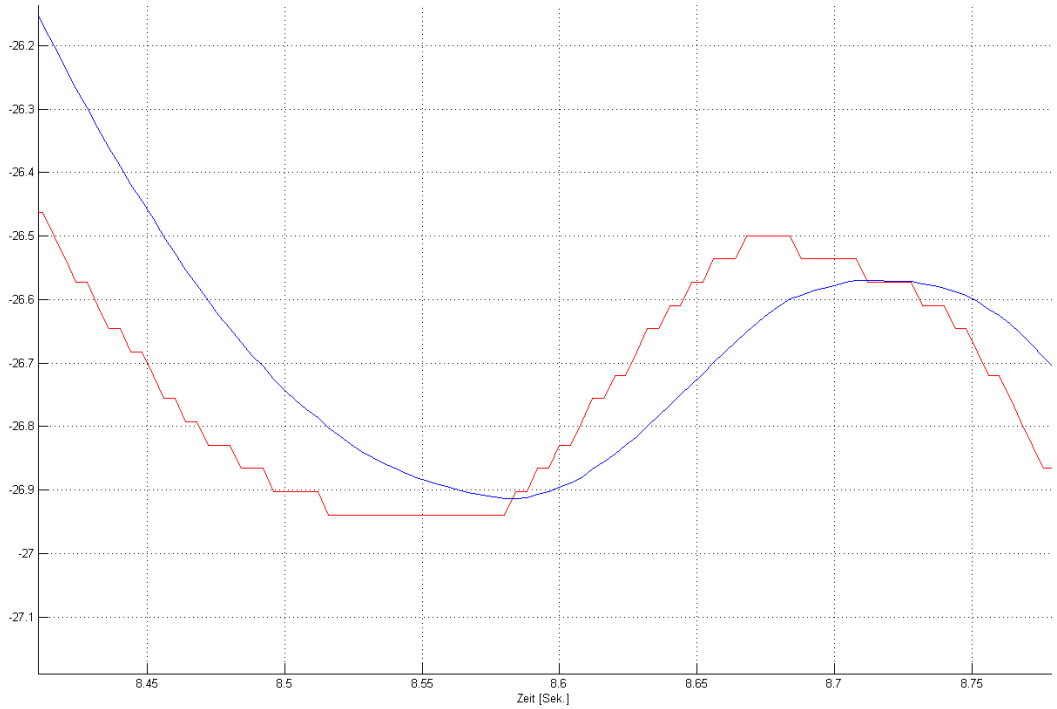


Abbildung 5.18: rot: ungefiltertes Positionssignal, blau: gefiltertes Positionssignal

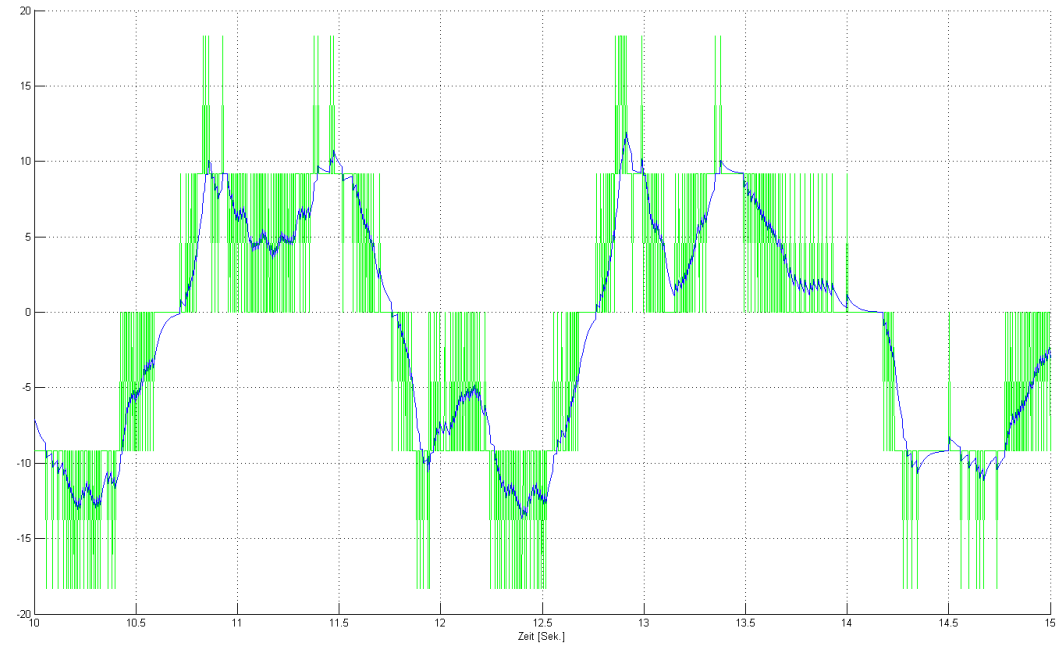


Abbildung 5.19: grün: ungefilterte Geschwindigkeit, blau: gefilterte Geschwindigkeit

Die verwendeten Koeffizienten der einzelnen Tiefpassfilter wurden durch mehrere Testversuche am Ballbot ermittelt.

5.2.2 Regler

Der eigentliche linear-quadratische Regler befindet sich im Subsytem „Regler ZX-Achse“ bzw. „Regler ZY-Achse“ in der Funktion „task_ts1“ im Block „Regler“. Die Eingangsgrößen dieses Subsystems sind der Zustandsvektor, die Versorgungsspannung und die Systemzeit. In Abbildung 5.20 ist der dazugehörige Simulink-Koppelplan zu sehen.

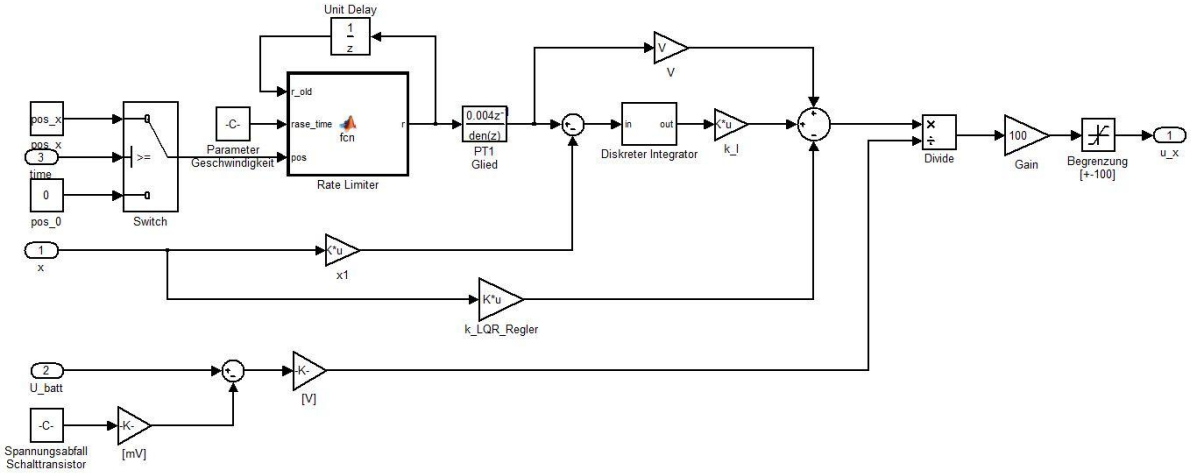


Abbildung 5.20: Simulink-Koppelplan des linear-quadratischen Reglers

Im Wesentlichen handelt es sich um jenen Regler, der im Abschnitt 4.2-4.4 entworfen und simuliert wurde, mit dem Unterschied, dass die Stellgröße in ein passendes PWM-Signal umgewandelt wird. Der Regler an sich wurde so modelliert, dass die Stellgröße die Eingangsspannung des Motors ist. Da aber die DC-Servomotoren als Eingangsgröße nur ein PWM-Signal in Prozent, das von -100 bis 100 geht, erwarten, muss die Stellgröße umgerechnet werden. Dazu verwendet man folgende Formel:

(5.05)

$$PWM_{\%} = \frac{U_{Stell}}{U_{Batt} - U_{Trans}} * 100,$$

wobei U_{Stell} die Stellgröße, die der Regler berechnet, U_{Batt} die aktuelle Versorgungsspannung des NXT-Bausteins und U_{Trans}, die Spannung die am internen Schalttransistor abfällt, ist. Wie hoch der Spannungsabfall am Transistor ist, kann nur geschätzt werden. Für das Ballbot-Modell wurde eine Spannung von 0,65V gewählt. Bevor das PWM-Signal dem „Servo Motor“-Block zugeführt wird, muss es noch auf 100% begrenzt werden.

Anzumerken ist, dass die Sensorsignale teilweise keine SI-Einheiten sind. Da der Regler aber in SI-Einheiten entworfen wurde, müssen für verschiedene Größen noch die dazugehörigen Umrechnungen durchgeführt werden.

5.2.3 Daten-Auswertung

Um die Arbeitsweise des Reglers überprüfen zu können, müssen verschiedene Daten analysiert werden. Der NXT-Baustein besitzt keinen internen Speicher und somit ist man nicht in der Lage Messdaten abzuspeichern. Durch den „NXT GamePad ADC Data Logger“-Block, welcher in Abschnitt 5.1 beschrieben wurde, ist man in der Lage per Bluetooth Daten vom NXT-Baustein zum Rechner zu senden und abzuspeichern. Damit dies geschehen kann, ist im Simulink-Koppelplan, welcher für die Regelung des Ballbots zuständig ist (siehe Abbildung 5.16), ein Subsystem namens „Data Logger“ enthalten, siehe Abbildung 5.21.

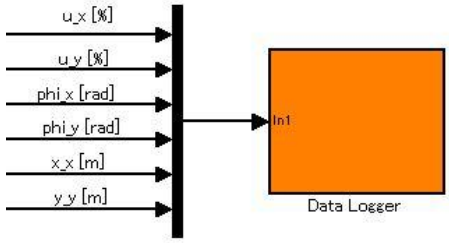


Abbildung 5.21: „Data Logger“-Subsystem für die Messdaten-Abspeicherung

Durch dieses Subsystem ist man in der Lage bis zu sechs Messdaten abzuspeichern. Da zusätzlich noch die Versorgungsspannung, die Zeit und die Ausgangswerte der einzelnen Servomotoren mit abgespeichert werden, ist es sinnvoll die Stellgrößen der beiden Regler, die Kippwinkel und die Position des Ballbots mit abzuspeichern. Durch diese Messdaten ist man dann in der Lage, das Ballbot-Modell am Rechner zu analysieren.
Die gesamte Messreihe wird in einer CSV-Datei abgespeichert. Damit die einzelnen Werte anschaulicher sind, wurde in MATLAB eine eigene Anwendung geschrieben, welche die gesamten Daten der CSV-Datei einliest und dann je nach Bedürfnis in einer Grafik anzeigt.

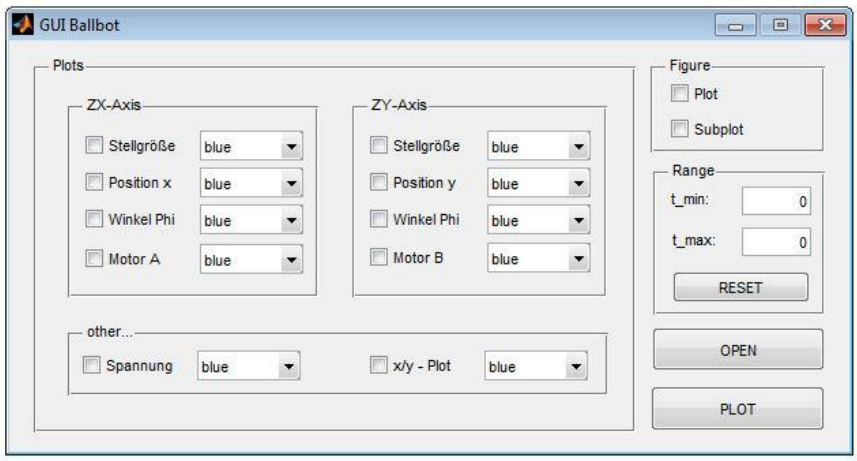


Abbildung 5.22: Bedienfeld der Auswertungssoftware

Die Bedienungsanleitung der Software sowie auch die Beschreibung für die Bluetooth Datenübertragung sind im Anhang C und D beschrieben.

5.3 Betriebsmodi

Der Ballbot kann in zwei Arten betrieben werden. Die erste Betriebsart ist die eigentliche Grundfunktion und zwar das Balancieren in der Ausgangsposition. In der zweiten Betriebsart ist man in der Lage, dem Ballbot eine gewisse Position in der Ebene vorzugeben und dieser bewegt sich dann autonom dorthin.
Die Parameter für die beiden Betriebsarten unterscheiden sich leicht. Bei der Balancierung wird vor allem auf die stationäre Genauigkeit und Robustheit geachtet. Bei der Fortbewegung hingegen ist vor allem das Erreichen der Wunschposition ohne umzukippen das primäre Ziel.

5.3.1 Balancierung

In diesem Betriebsmodus balanciert der Ballbot in seiner Ausgangsposition und versucht, so gut wie möglich, diese Position zu halten. Wichtige Kriterien sind dabei die Robustheit und die stationäre Genauigkeit. Um die stationäre Genauigkeit zu gewährleisten, genügt es, wenn man den Koeffizienten für den Integrierer ziemlich klein wählt. Will man den Regler robuster gegenüber Störungen machen, so muss man den skalaren Wert R kleiner wählen. Nach mehreren Testversuchen mit unterschiedlichen Parametern, hat sich ergeben, dass man mit folgendem Parametersatz den besten Kompromiss zwischen stationärer Genauigkeit, Robustheit und ruhiger Balancierung erzielt:

(5.06)

$$Q = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.00001 \end{bmatrix}$$

$$R = 0.0013$$

Wie man gut erkennen kann, ist der integrierende Anteil sehr klein. Trotzdem genügt er, um den Ballbot in seiner Ausgangsposition zu halten bzw. zu erreichen, dass er nach einiger Zeit in diese wieder zurückkehrt.
Würde man den Regler schneller machen, indem man den skalaren Wert R kleiner wählt, so würde der Ballbot zwar schneller auf Änderungen reagieren und somit auch robuster gegenüber gewisse Störungen sein, aber er würde auch insgesamt „unruhiger“ werden.

Führt man mit dem Parametersatz 5.06 einen Balancierversuch über einen längeren Zeitraum durch und speichert die Messdaten per Bluetooth ab, so erhält man für die Position des Ballbots folgendes Diagramm:

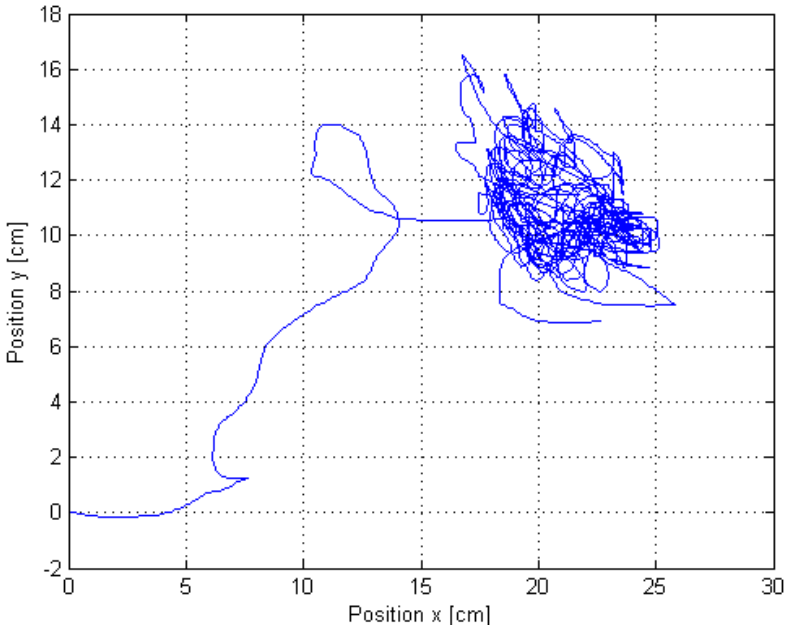


Abbildung 5.23: Position des Ballbots bei einem Balancier-Versuch

In Abbildung 5.23 ist gut zu erkennen, wie der Ballbot rund um einen Punkt balanciert. Die Abweichung zum Ursprung ergibt sich daraus, dass der Schwerpunkt des Ballbots, wie in der Simulation angenommen, wahrscheinlich nicht genau in der Mitte liegt und vor allem auch daraus, wie der Ballbot vor dem Versuchsstart gehalten wurde. Durch den integrierenden Anteil versucht jetzt der Ballbot langsam wieder in seine Ausgangsposition zurückzufahren. Da dieser Anteil ziemlich klein gewählt wurde, braucht der Ballbot eine gewisse Zeit, bis er zum Ursprung zurück findet. Durch Ungenauigkeiten bei der Parametrisierung des Ballbots und vor allem durch mögliches Auftreten eines Schlupfs zwischen dem Antriebsrad und dem Ball wird der Ballbot seine Ausgangsposition nicht exakt erreichen. Es wird immer eine gewisse Abweichung bleiben.

In Abbildung 5.24 sieht man für einen Balancier-Versuch die Winkelgeschwindigkeiten und Kippwinkel der beiden Kipprichtungen des Ballbots und die dazugehörigen Stellgrößen für die Antriebsmotoren. Wenn man die beiden Diagramme der Kippwinkel näher betrachtet, so sieht man, dass der Kippwinkel um die ZX-Achse um 0° schwankt, hingegen der Kippwinkel um die ZY-Achse um 2-3° hin und her pendelt. Dies ist wiederum ein Indiz, dass der Schwerpunkt des Ballbots wahrscheinlich nicht ganz in der Mitte liegt, wie es in den Berechnungen angenommen wurde. Es hängt aber auch davon ab, wie der Ballbot beim Start gehalten wird. Gut zu erkennen ist auch, dass die Stellgröße bei höheren Beträgen der Winkelgeschwindigkeit öfters kurzzeitig in die Beschränkung kommt.

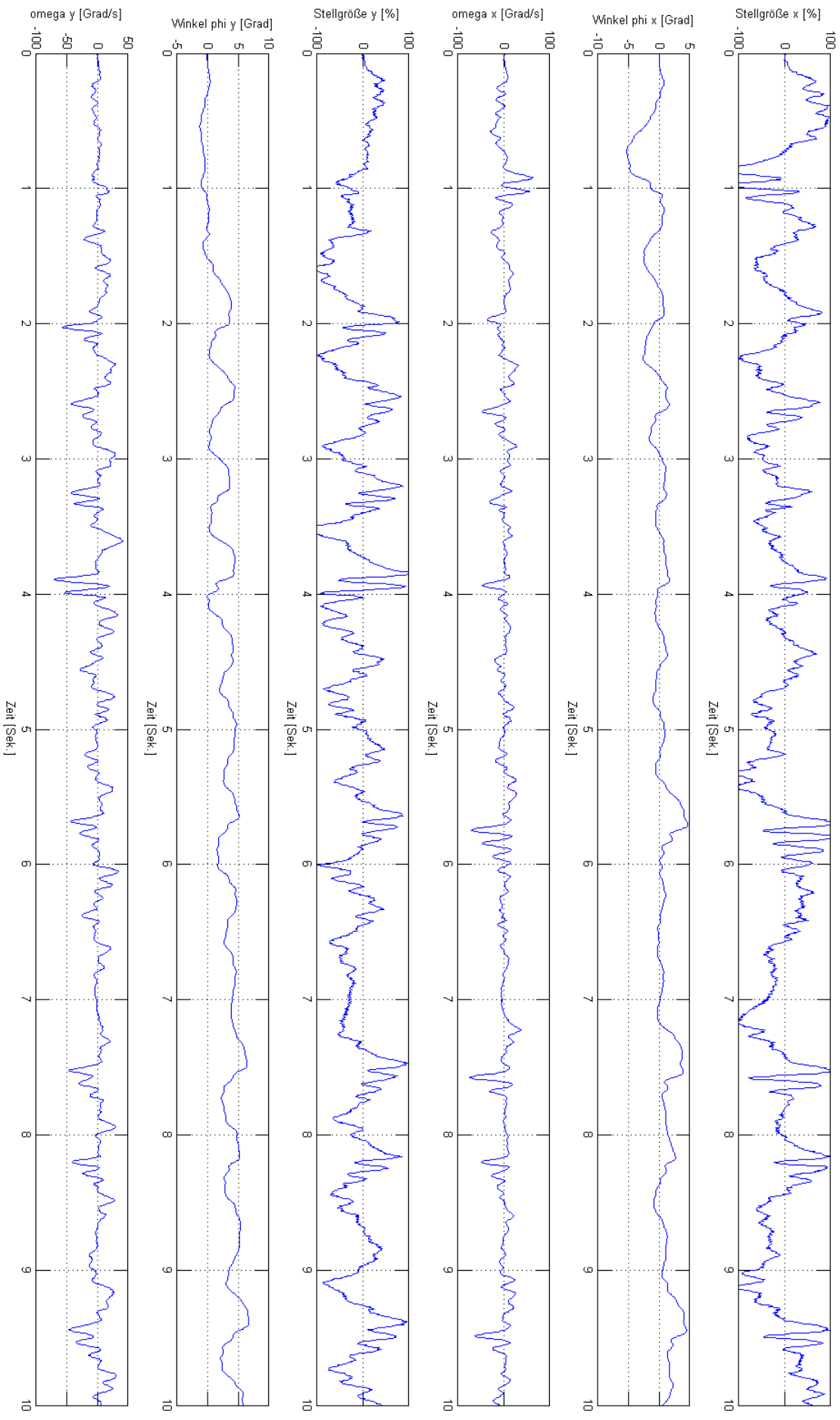


Abbildung 5.24: Stellgröße, Kippwinkel und Winkelgeschwindigkeit der beiden Regler bei Balancierung

Ein weiterer interessanter Versuch, ist die Balancierung des Ballbots mit äußeren Störeingriffen, siehe Abbildung 5.25. Bei 14,2 Sekunden wurde von außen eine Störung in Y-Richtung dem Ballbot zugeführt. Wie man gut aus der Abbildung erkennen kann, war die Störung so groß, dass der Kippwinkel 6° erreicht hat. Der Regler war längere Zeit in der Begrenzung, schafft es aber trotzdem, den Ballbot auszubalancieren. Zwischen dem Eintreten der Störung und dem Erreichen einer erneuten stabilen Position vergehen ca. 3 Sekunden. Um diese Zeit zu verkürzen, müsste man den Regler schneller machen. Dies hat aber den Nachteil, dass der Regler noch länger in der Begrenzung verharren würde. Durch einen schnelleren Regler wäre man in der Lage größere Störeingriffe zu kompensieren.

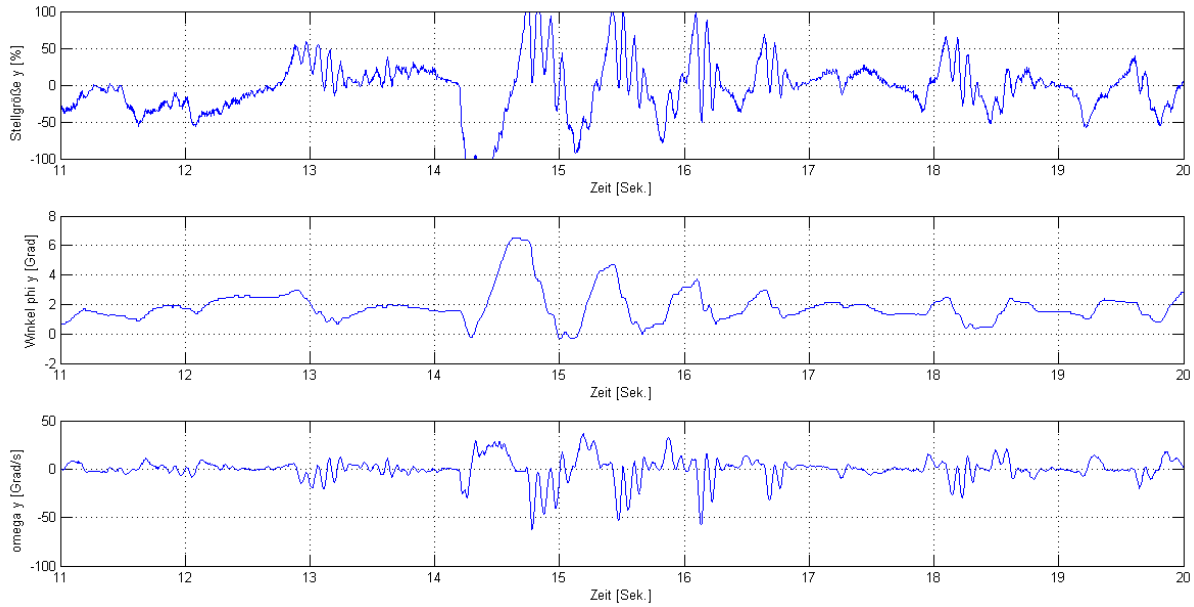


Abbildung 5.25: Stellgröße, Kippwinkel und Winkelgeschwindigkeit bei Störeingriff

5.3.2 Fortbewegung

In diesem Betriebsmodus fährt der Ballbot zu einer gewünschten Position in der XY-Ebene, die vorab mittels MATLAB übergeben wurde. Führt man mit obigen Parametern einen Testversuch durch, so wird man schnell feststellen, dass der Ballbot die gewünschte Position nicht erreicht. Grund dafür ist, dass der Roboter durch das Ausbalancieren und durch die Bewegung eine Eigenrotation aufweist. Durch diese Eigenrotation verliert der Ballbot seine Orientierung und der Roboter bewegt sich willkürlich im Raum. Um dieses Problem zu lösen, muss die genaue Lage des Ballbots bekannt sein. Da man mit den aktuellen Messdaten nicht in der Lage ist, die genaue Position des Ballbots zu bestimmen, wurde ein Kompasssensor eingebaut. Dank der Daten, die dieser Sensor liefert, kann man die tatsächliche Position berechnen und die aktuelle Position korrigieren.

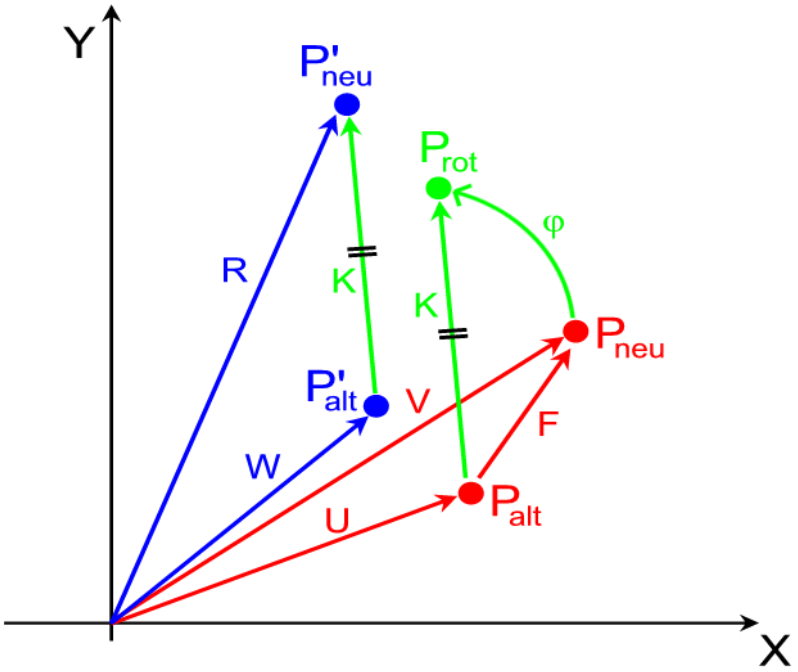


Abbildung 5.26: Grafische Darstellung eines Rechenschrittes zur Berechnung der tatsächlichen Position des Ballbots; rot: nicht korrigierte Positionen, blau: korrigierte Positionen, grün: Korrektur

Um die tatsächliche Position bestimmen zu können, muss ständig die aktuelle Position des Ballbots korrigiert werden. In Abbildung 5.26 ist die Berechnung für einen solchen Rechenschritt grafisch veranschaulicht. P_{alt} ist die aktuelle Position des Ballbots, die mittels der Winkelsensoren der NXT-Servomotoren bestimmt wurde. P'_{alt} ist die tatsächliche aktuelle Position des Ballbots. Kommt es bei der letzten Bewegung des Ballbots zu keiner Rotation, so entspricht die Position P_{alt} der Position P'_{alt}. Hat hingegen der Ballbot eine Rotation durchgeführt, so unterscheiden sich diese beiden Positionen. Bewegt sich nun der Ballbot von der aktuellen Position P_{alt} zur Position P_{neu} hin und findet während der Bewegung eine Rotation statt, so entspricht die neue Position P_{neu} nicht der tatsächlichen Position P'_{neu}. Nun gilt es, mit Hilfe der Daten, die der Kompasssensor liefert, die neue tatsächliche Position P'_{neu} zu berechnen.

Die aktuelle Position wird alle vier Millisekunden neu berechnet. Um diese Position bestimmen zu können, müssen mehrere Rechenschritte durchgeführt werden, die wie folgt lauten:

1. Berechnen der gefahrenen Strecke: Es wird die Differenz von der aktuellen Position zur letzten Position gebildet (siehe Skizze: $\vec{F} = \vec{V} - \vec{U}$)
2. Berechnen des Drehwinkels φ: Durch das Sensorsignal des Kompasssensors kann der Winkel, um den sich der Ballbot gedreht hat, bestimmt werden.
3. Rotation der aktuellen Lage um den Drehwinkel φ: Die aktuelle Position muss um den Drehwinkel mithilfe der Rotationsmatrix gedreht werden (siehe Skizze: $\vec{K} = R_M * \vec{F}$)
4. Berechnen der neuen tatsächlichen Position: Der in Punkt 3 berechnete Vektor $\vec{K}$ wird vektoriell zur alten tatsächlichen Position addiert. Das Ergebnis ist dann die neue tatsächliche Position (siehe Skizze: $\vec{R} = \vec{W} + \vec{K}$)

Die Rotation um den Drehwinkel wird mit Hilfe folgender Rotationsmatrix durchgeführt:

(5.07)

$$R_M = \begin{bmatrix} cos\varphi & -sin\varphi \\ sin\varphi & cos\varphi \end{bmatrix}$$

Für die oben beschriebenen Berechnungen wurde in Simulink ein eigenes Subsystem („Position Berechnung bei Drehung“) in der Funktion „task_ts1“ im Block „Regler“ realisiert, siehe Abbildung 5.16. In diesem Subsystem werden mittels einer MATLAB-Funktion alle vier Millisekunden die oben angeführten Berechnungen zur Bestimmung der tatsächlichen Position durchgeführt.

In diesem Betriebsmodus ist es vor allem wichtig, dass der Ballbot seine Wunschposition erreicht ohne umzukippen. Um dieser Anforderung gerecht zu werden, müssen die Parameter leicht geändert werden. Durch die Bewegung des Ballbots führt man dem Roboter sozusagen eine ständige Störung zu. Damit der Roboter trotzdem das Gleichgewicht nicht verliert, muss der Regler schneller reagieren. Es hat sich herausgestellt, dass man mit folgendem Parametersatz gute Ergebnisse erzielt und der Ballbot seine Wunschposition gut erreicht:

(5.08)

$$Q = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.005 \end{bmatrix}$$

$$R = 0.00095$$

Wenn man diesen Parametersatz mit jenen von 5.06 vergleicht, so sieht man, dass sich der integrierende Anteil und der skalare Wert R geändert haben. Der Regler wurde schneller gemacht und der integrierende Zweig des Reglers hat nun mehr Einfluss.

Durch die ständige Korrektur der Position und durch ungenaue Sensordaten des Kompasssensors erschwert sich die Fortbewegung des Ballbots. Abhilfe schafft ein Tiefpassfilter am Ausgang des Subsystems zur Positionsberechnung. Dank dieses Filters ist der Roboter nun in der Lage, seine Wunschposition ohne umzukippen zu erreichen.

Führt man mit dem Parametersatz 5.08 einen Versuch durch und gibt als Führungsgrößen für die Position x einen Meter und für die Position y einen halben Meter an und speichert die Messdaten per Bluetooth ab, so erhält man für die Position des Ballbots folgende Trajektorie:

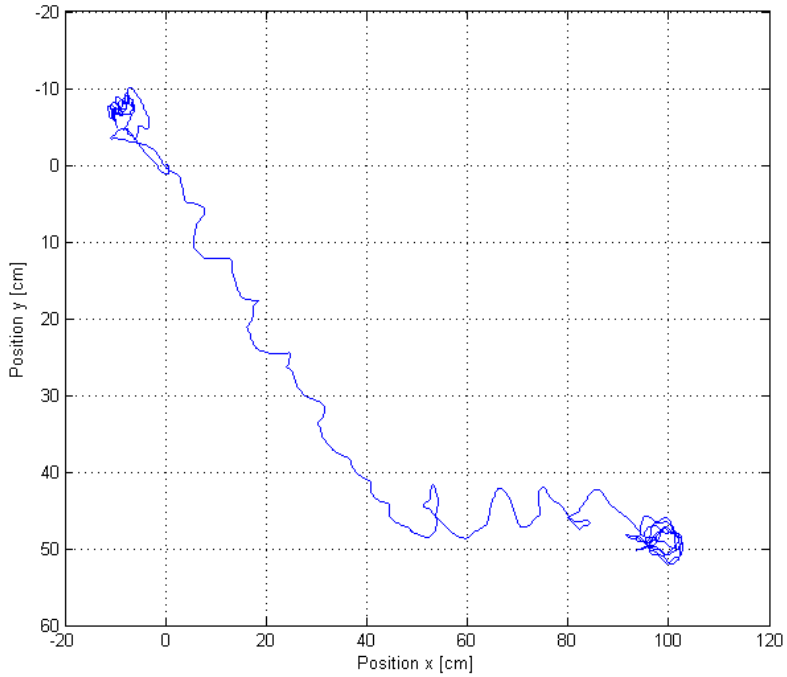


Abbildung 5.27: Trajektorie des Ballbots bei Fortbewegung

Wie man aus Abbildung 5.27 gut erkennen kann, balanciert der Ballbot einige Zeit in seiner Ausgangsposition, bevor er anfängt sich zu bewegen. Gut ersichtlich ist auch die Fortbewegung des Roboters in Richtung Sollposition und das Erreichen dieser Sollposition.

In Abbildung 5.28 sieht man die beiden Zustandsgrößen der beiden Regler für die Position des Ballbots. Sehr gut zu sehen ist, wie der Ballbot mittels einer Rampe sich an die vorgegebene Führungsgröße annähert. Die stationäre Genauigkeit ist dank des integrierenden Anteils sehr gut.

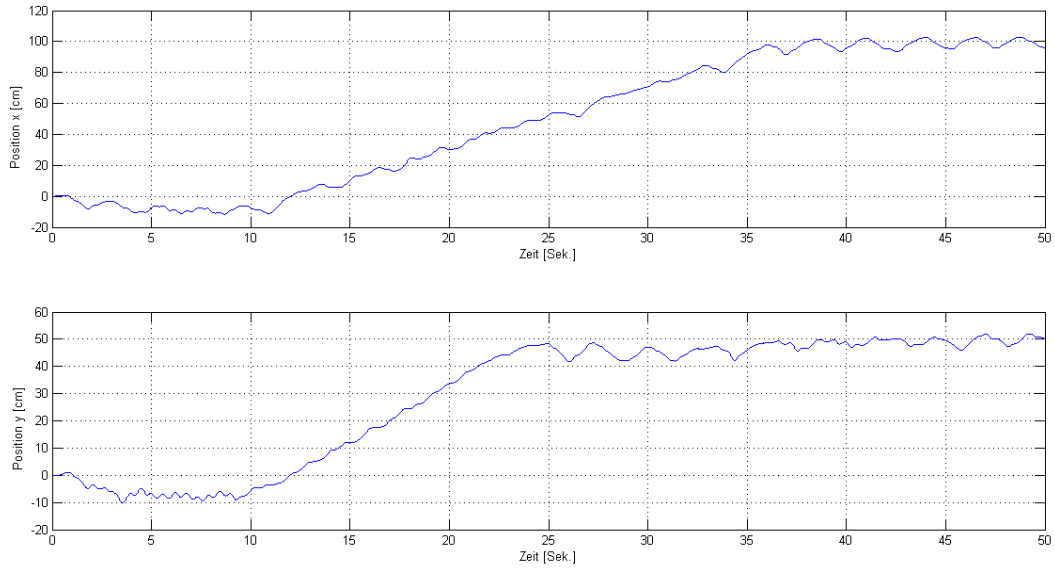


Abbildung 5.28: Position x und y bei Fortbewegung des Ballbots

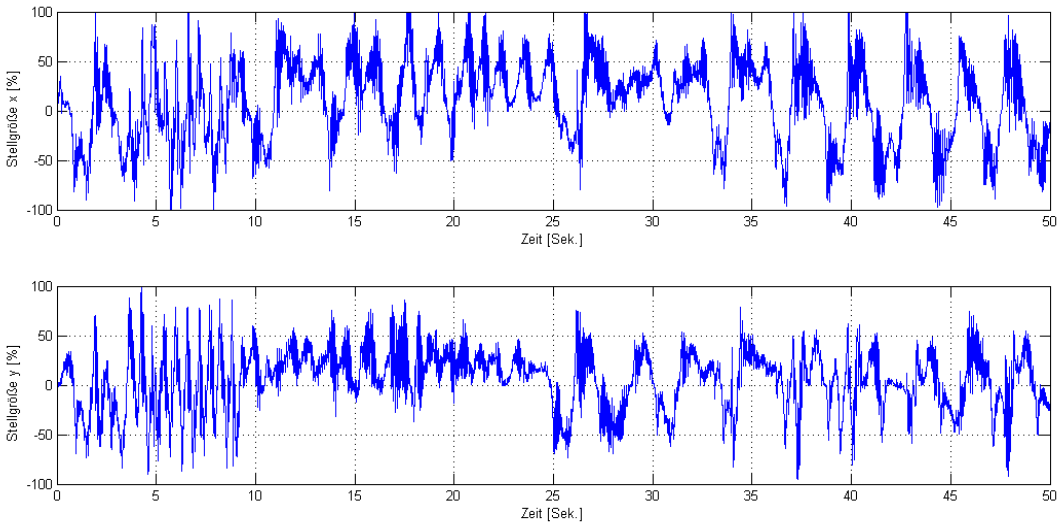


Abbildung 5.29: Stellgrößen der beiden Regler bei Fortbewegung des Ballbots

In Abbildung 5.29 sind für den Fortbewegungsversuch die Stellgrößen der beiden Regler grafisch dargestellt. Man erkennt, dass bei einer solchen Fortbewegung die Regler unterschiedlich beansprucht werden. Der Regler für die ZX-Achse gerät öfter in die Beschränkung als der Regler der ZY-Achse. Man sieht auch, dass die Regler in den ersten 10 Sekunden während der Balancierung seltener beansprucht werden als während der Fortbewegung.

Abschließend kann man sagen, dass die beiden Betriebsmodi, Balancierung und Fortbewegung, sich grundsätzlich von der Regelung nur geringfügig unterscheiden. Bei der Balancierung ist vor allem die stationäre Genauigkeit bzw. Robustheit von Relevanz. Bei der Fortbewegung ist hingegen eine korrekte Funktionsweise wichtig, d.h. der Ballbot soll ohne umzukippen die gewünschte Position erreichen. Um dies zu bewerkstelligen, mussten die Regelparameter leicht verändert werden und ein zusätzlicher Tiefpassfilter für die Position eingebaut werden. Zusätzlich stellte man fest, dass der Ballbot bei der Fortbewegung eine Eigenrotation aufwies und es dadurch unmöglich war, die Wunschposition präzise anzufahren. Durch Einbau eines digitalen Kompasssensors und der Implementierung einer Korrekturfunktion, welche ständig die aktuelle Position des Ballbots neu berechnet, war man dann in der Lage, den Ballbot korrekt zu manövrieren.
Es ist auch zu erwähnen, dass der Ballbot die Wunschposition zwar korrekt erreicht, aber diese in Wirklichkeit nicht ganz den gewünschten Längenmaßen entspricht. Diese Abweichung ist vor allem auf einen Schlupf zurückzuführen, der zwischen Antriebsrad und Plastikball auftritt. Auch die unterschiedliche Beschaffenheit des Untergrunds führt sicherlich zu Abweichungen. Zusätzlich ist zu erwähnen, dass die meisten Modell-Parameter zwar eigenständig bestimmt worden sind, aber einige nur geschätzt werden konnten bzw. aus der Literatur entnommen worden sind, wie z.B. der Schwerpunkt des Ballbots oder die Maschinenkonstante der Gleichstrom-Servomotoren. Durch all diese Aspekte können sich dann eben Abweichungen bei der angefahrenen Wunschposition ergeben. Auch durch ungenaue Sensordaten des Kompasssensors wird die Regelung sicherlich nicht erleichtert. In all diesen Bereichen sind für einen besseren Betrieb des Ballbots sicherlich noch Verbesserungen möglich.

6 Zusammenfassung und Ausblick

Am Institut für Regelungs- und Automatisierungstechnik an der Technischen Universität Graz entschied man sich im Sommer 2012, im Rahmen einer Masterarbeit einen Ballbot aus LEGO zu realisieren.
Der erste Teil der Arbeit bestand darin, mittels des Lagrange-Formalismus ein mathematisches Modell für den Ballbot zu finden. Dabei musste man auf die richtige Wahl der Stellgröße achten, da die LEGO-Motoren als Eingangsgröße ein PWM-Signal benötigen. Anschließend wurde dann ein passender Regler entworfen und das Modell in MATLAB/Simulink simuliert. Die Wahl fiel dabei auf einen linear-quadratischen Zustandsregler.
Der nächste Schritt war die Installation und Inbetriebnahme der MATLAB/Simulink-Toolbox für den LEGO Mindstorms-Baukasten. Große Schwierigkeiten hatte man mit der Genauigkeit der Gyroskopsensoren, da sie mit einem Offset und einer Drift versehen sind. Mittels eines Tiefpassfilters konnte das Problem dann zufriedenstellend gelöst werden. Anschließend wurde der zuvor entworfene und simulierte Regler mit der installierten Toolbox in Simulink implementiert und getestet. Nach den ersten erfolgreichen Balancier-Versuchen stellte man fest, dass mit dem aktuellen Regler die stationäre Genauigkeit nicht zufriedenstellend war. Der Regler wurde dann mit einem Integrierer erweitert.
Im zweiten Teil der Arbeit ging es darum, dem Ballbot eine gewisse Position in der Ebene vorzugeben und dieser sollte sich dann autonom dorthin bewegen. Da man die Position des Ballbots als Zustandsgröße hatte, lag es auf der Hand, die Wunschposition als Führungsgröße aufzuschalten. Bei den ersten Tests stellte man gleich fest, dass der Ballbot eine willkürliche Eigenrotation besitzt. Durch diese Rotation war man dann nicht mehr in der Lage die gewünschte Position anzufahren, da die genaue Lage des Ballbots unbekannt war. Man entschied sich dann, einen Kompasssensor einzubauen, um die genaue Position des Ballbots bestimmen zu können. Dank des Sensors war man dann in der Lage, gewisse Positionen in der Ebene problemlos anzufahren.
Ein weiterer Bestandteil der Masterarbeit war die Realisierung einer grafischen Auswertung. Die gemessenen Daten werden dabei mittels einer Bluetooth-Verbindung während der Inbetriebnahme des Ballbots an einen Rechner übermittelt. Dank einer MATLAB-Applikation können dann diese Messdaten grafisch ausgewertet werden.
Mit der erstellten Lösung ist man nun in der Lage, die Eigenschaften eines integrierenden linear-quadratischen Zustandsreglers zu demonstrieren und zu veranschaulichen. In weiteren Arbeiten könnte man die Regelung optimieren, vor allem im Bereich der Parameter Identifikation gibt es noch Verbesserungspotential. Des Weiteren könnte man den Schlupf berücksichtigen, der zwischen dem Antriebsrad und dem Ball auftritt. Auch im Bereich der Positionsbestimmung gibt es noch Verbesserungspotential. Anstelle des digitalen Kompasssensors könnte man die Position des Ballbots durch eine Kamera bestimmen.

Dadurch umgeht man die Störanfälligkeit und Ungenauigkeit des Kompasssensor-Signals und man könnte die Position des Ballbots exakt bestimmen.
Um eine ruhigere und exaktere Balancierung und Fortbewegung zu gewährleisten, könnte man auch den Aufbau des Ballbots ändern. Durch die Anordnung der beiden Motoren ist nur eine gewisse Bewegung möglich. Würde man die Motoren anders anordnen bzw. einen zusätzlichen Motor einbauen, sodass eine Rotation des Ballbots im Stand möglich wäre, so würde man deutliche Verbesserungen in der Regelung erzielen können.
In dieser Masterarbeit wurde ausschließlich mit dem linear-quadratischen Zustandsregler gearbeitet. Es wäre durchaus denkbar andere Regler zu benutzen, wie z.B. einen nichtlinearen Regler.
Zum Schluss ist noch zu erwähnen, dass es auch die Möglichkeit gibt, den Ballbot mittels Bluetooth-Verbindung per Rechner in Echtzeit anzusteuern. In einer weiterführenden Arbeit könnte man diese Zusatz-Funktion einbauen.

Anhang A: Maple Code

```
> restart;
with(LinearAlgebra):with(linalg):

> alias(q1=q1[t],q1p=q1p[t],q2=q2[t],q2p=q2p[t]);
```

$$q1, q1p, q2, q2p$$

```
> r1:=Vector([q1,0]);
r2:=Vector([q1+l*sin(q2),l*cos(q2)]);
r1p:=map(diff,r1,t);
r2p:=map(diff,r2,t);
T:=simplify(m[1]/2*(Transpose(r1p).r1p)+m[2]/2*(Transpose(r2p).r2p)+J[1]/(2*R[1]^2)*(Transpose(r
1p).r1p));
T:=subs(diff(q1,t)=q1p,diff(q2,t)=q2p,T);
f1:=diff(T,q1p);
f2:=diff(T,q2p);
```

$$r1 := \begin{bmatrix} q1 \\ 0 \end{bmatrix}$$

$$r2 := \begin{bmatrix} q1 + l \sin(q2) \\ l \cos(q2) \end{bmatrix}$$

$$T := \frac{1}{2} \frac{1}{R_1^2} \left(m_1 \, q1p^2 \, R_1^2 + m_2 \, R_1^2 \, q1p^2 + 2 \, m_2 \, R_1^2 \, q1p \, l \cos(q2) \, q2p + m_2 \, R_1^2 \, l^2 \, q2p^2 + J_1 \, q1p^2 \right)$$

```
> Q1:=((k[m]*phi)/R[a])/R[0]*U[a]-((k[m]*phi)^2/R[a])/R[0]*diff(q1,t)/R[1];
Q2:=0;
```

$$Q1 := \frac{k_m \, \phi \, U_a}{R_a \, R_0} - \frac{k_m^2 \, \phi^2 \left(\frac{\partial}{\partial t} \, q1 \right)}{R_a \, R_0 \, R_1}$$

$$Q2 := 0$$

```
> V:=simplify(m[2]*g*l*cos(q2));
L:=T-V;
ls1:=simplify(diff(diff(L,q1p),t)-diff(L,q1)); gl1:=simplify(ls1=Q1);
ls2:=simplify(diff(diff(L,q2p),t)-diff(L,q2)); gl2:=simplify(ls2=Q2);
```

$$V := m_2 \, g \, l \cos(q2)$$

$$L := \frac{1}{2}\,\frac{1}{R_1^2}\left(m_1\,q1p^2\,R_1^2 + m_2\,R_1^2\,q1p^2 + 2\,m_2\,R_1^2\,q1p\,l\cos(q2)\,q2p + m_2\,R_1^2\,l^2\,q2p^2 + J_1\,q1p^2\right) - m_2\,g\,l\cos(q2)$$

```
> subs({diff(q1p,t)=`q1''`,diff(q2p,t)=`q2''`,diff(q1,t)=`q1'`,diff(q2,t)=`q2'`,q2p=`q2'`},gl1);
subs({diff(q1p,t)=`q1''`,diff(q2p,t)=`q2''`,diff(q1,t)=`q1'`,diff(q2,t)=`q2'`,q1p=`q1'`,q2p=`q2'`},gl2);
```

$$-\frac{1}{R_1^2}\left(-m_1\,q1''\,R_1^2 - m_2\,R_1^2\,q1'' + m_2\,R_1^2\,l\sin(q2)\,q2'^2 - m_2\,R_1^2\,l\cos(q2)\,q2'' - J_1\,q1''\right) = \frac{k_m\,\phi\left(U_a\,R_1 - k_m\,\phi\,q1'\right)}{R_a\,R_0\,R_1}$$

$$m_2\,l\,(q1''\cos(q2) + l\,q2'' - g\sin(q2)) = 0$$

```
> L1:=subs({diff(q1p,t)=q1pp,diff(q2p,t)=q2pp,diff(q1,t)=q1p,diff(q2,t)=q2p},ls1);
L2:=subs({diff(q1p,t)=q1pp,diff(q2p,t)=q2pp,diff(q1,t)=q1p,diff(q2,t)=q2p},ls2);

> M:=jacobian([L1, L2],[q1pp, q2pp]);

> h:=Vector(simplify(matadd(Vector([L1, L2]),-multiply(M,Vector([q1pp, q2pp])))));
rs:=simplify(multiply(inverse(M),matadd(Vector([Q1, Q2]),-h)));

> f1:=x3;
f2:=x4;
f3:=subs({diff(q1,t)=x3,diff(q2,t)=x4,q1=x1,q2=x2,q1p=x3,q2p=x4},rs[1]);
f4:=subs({diff(q1,t)=x3,diff(q2,t)=x4,q1=x1,q2=x2,q1p=x3,q2p=x4},rs[2]);
```

$$f1 := x3$$

$$f2 := x4$$

$$f3 := \left(R_1\left(k_m\,\phi\,U_a\,R_1 - k_m^2\,\phi^2\,x3 + m_2\,l\sin(x2)\,x4^2\,R_a\,R_0\,R_1 - \cos(x2)\,R_1\,m_2\,g\sin(x2)\,R_a\,R_0\right)\right) \Big/ \left(\left(m_1\,R_1^2 + m_2\,R_1^2 + J_1 - m_2\cos(x2)^2\,R_1^2\right)R_a\,R_0\right)$$

$$f4 := -\left(\cos(x2)\,R_1^2\,k_m\,\phi\,U_a - \cos(x2)\,R_1\,k_m^2\,\phi^2\,x3 + \cos(x2)\,R_1^2\,m_2\,l\sin(x2)\,x4^2\,R_a\,R_0 - g\sin(x2)\,R_a\,R_0\,m_1\,R_1^2 - g\sin(x2)\,R_a\,R_0\,m_2\,R_1^2 - g\sin(x2)\,R_a\,R_0\,J_1\right) \Big/ \left(l\left(m_1\,R_1^2 + m_2\,R_1^2 + J_1 - m_2\cos(x2)^2\,R_1^2\right)R_a\,R_0\right)$$

```
> JA:=simplify(subs(x1=0,x2=0,x3=0,x4=0,U[a]=0,jacobian([f1, f2, f3, f4],[x1, x2, x3, x4])));
```

$$JA := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\dfrac{g\, m_2\, R_1^2}{m_1\, R_1^2 + J_1} & -\dfrac{R_1\, k_m^2\, \phi^2}{R_a\, R_0 \left(m_1\, R_1^2 + J_1\right)} & 0 \\ 0 & \dfrac{g \left(m_1\, R_1^2 + m_2\, R_1^2 + J_1\right)}{\left(m_1\, R_1^2 + J_1\right) l} & \dfrac{R_1\, k_m^2\, \phi^2 \cos(0)}{l\, R_a\, R_0 \left(m_1\, R_1^2 + J_1\right)} & 0 \end{bmatrix}$$

```
> JB:=simplify(subs(x1=0,x2=0,x3=0,x4=0,U[a]=0,jacobian([f1, f2, f3, f4],[U[a]])));
```

$$JB := \begin{bmatrix} 0 \\ 0 \\ \dfrac{R_1^2\, k_m\, \phi}{R_a\, R_0 \left(m_1\, R_1^2 + J_1\right)} \\ -\dfrac{R_1^2\, k_m\, \phi \cos(0)}{l\, R_a\, R_0 \left(m_1\, R_1^2 + J_1\right)} \end{bmatrix}$$

Anhang B: „ECRobot“ Toolbox - Installationsanleitung

Die Folgende Anleitung beschreibt die einzelnen Schritte, die für die Installation der Mindstorms Toolbox „ECRobot“ für MATLAB/Simulink notwendig sind.

1. Auf der Mathworks Seite unter folgendem Link: http://www.mathworks.com/matlabcentral/fileexchange/25207 den „ECRobotInstaller“ herunterladen und entpacken.

2. MATLAB öffnen und zum gerade heruntergeladenen „ECRobotInstaller“ Ordner wechseln.

3. Im MATLAB Command Window folgenden Befehl eintippen: download_ecrobot_tools

   Durch diesen Befehl werden alle benötigten Programme von Drittanbieter heruntergeladen. Bei der Ausführung von Cygwin erscheinen mehrere Dialogfenster mit verschiedenen Konfigurationsmöglichkeiten. Alle Optionen sind bereits vorkonfiguriert, man muss nur mit „Weiter“ die einzelnen Dialogfenster bestätigen.

4. Sobald alle Programme heruntergeladen sind, im MATLAB Command Window folgenden Befehl eintippen: install_ecrobot_tools

   Durch diesen Befehl werden die zuvor heruntergeladenen Programme installiert. Bei der Installation der Programme, kann es vorkommen, dass Dialogboxen mit Konfigurationsmöglichkeiten erscheinen. Alle Optionen sind bereits vorkonfiguriert, man muss nur mit „Weiter“ die einzelnen Dialogfenster bestätigen.

   Kommt es bei der Installation zu Problemen, so muss man Punkt 4 nochmal wiederholen.

5. Rechner neu starten

Die „ECRobot“ Toolbox ist nun installiert und kann in Simulink verwendet werden.

Falls auf dem NXT-Baustein noch eine alte Firmware vorhanden ist, ist es notwendig ein Update durzuführen. Dazu schließt man den NXT-Baustein per USB am Rechner an und öffnet MATLAB und wechselt wiederum zum „ECRobotInstaller“-Ordner und gibt im Command Window folgenden Befehl ein: update_nxt_firmware.

Falls die Toolbox nicht installiert wurde, gibt es im „ECRobotInstaller“-Ordner eine pdf-Datei (README.pdf) mit einer Installationsbeschreibung auf Englisch, wo abschließend verschiedene Fehlerursachen mit den dazugehörigen Lösungen aufgelistet sind.

Anhang C: Bluetooth-Verbindung einrichten

Damit der LEGO Mindstorms NXT-Baustein per Rechner gesteuert werden kann bzw. Daten an einen Rechner übermittelt werden können, muss eine aktive Bluetooth-Verbindung zwischen einem Rechner und den NXT-Baustein bestehen. Voraussetzung dafür ist ein Rechner mit Bluetooth Interface oder ein Bluetooth Adapter. Die folgende Anleitung beschreibt die einzelnen Schritte, wie man eine solche Bluetooth-Verbindung aufbaut.

1. Den NXT-Baustein einschalten und ein Programm ausführen, wo eine Bluetooth-Verbindung benötigt wird. Auf dem NXT-Display erscheinen folgende Zeilen:

   Solange diese Anzeige am Display zu sehen ist, wartet der NXT-Baustein auf eine Bluetooth-Verbindung.

2. Nach dem Anbringen bzw. Aktivieren des Bluetooth-Adapters am Rechner die Bluetooth-Konfiguration öffnen. Normalerweise gelangt man in das Konfigurationsmenü durch Doppelklick auf das Bluetooth-Symbol in der Taskleiste.

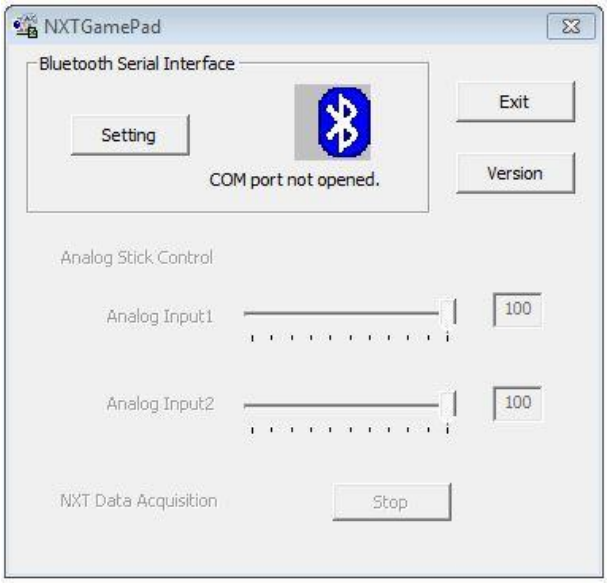
3. Eine neue Bluetooth-Verbindung erstellen oder Bluetooth-Geräte, die in Reichweite sind, suchen.

4. Nachdem der NXT-Baustein gefunden wurde, folgenden Bluetooth pass key eingeben: MATLAB

5. Sobald eine Verbindung steht, unter den Bluetooth-Einstellungen den dazugehörigen COM Port notieren.

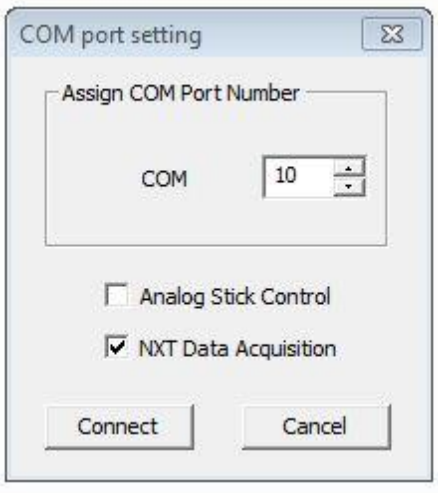
6. Den NXT-Baustein ausschalten.

7. Auf folgender Seite: http://lejos-osek.sourceforge.net/download.htm das Programm „NXTGamePad“ downloaden und ausführen. Das Programm kann auch über das MATLAB/Simulink Modell „Ballbot.mdl“ (siehe Anhang D) durch drücken auf die Schaltfläche „Bluetooth Data Connection“ gestartet werden. Folgendes Dialogfenster erscheint:

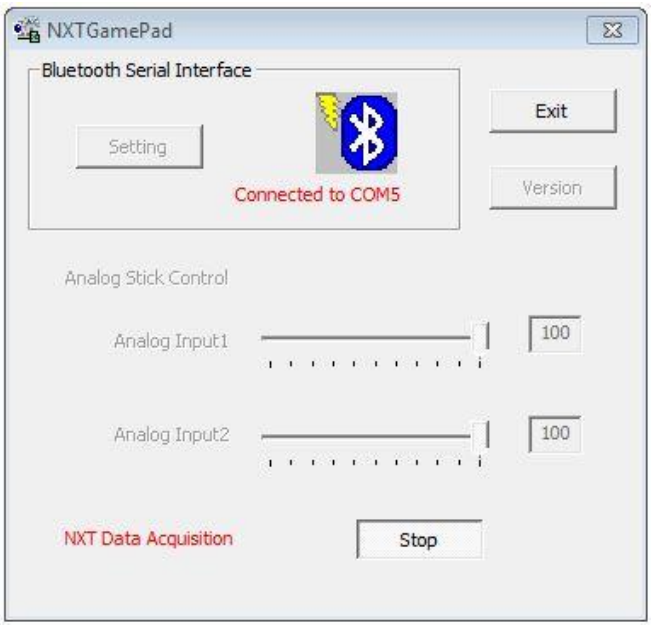


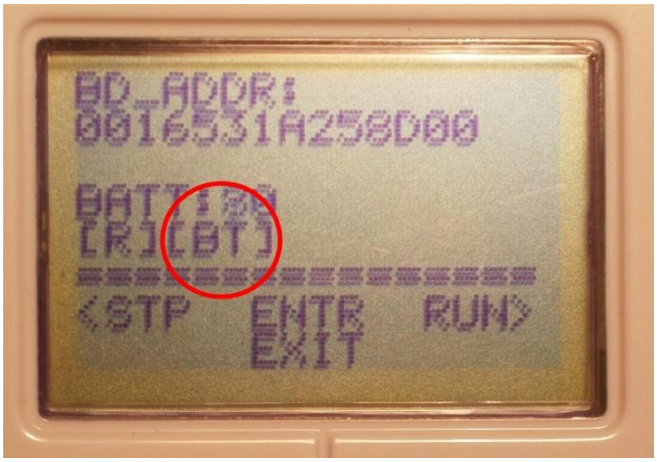
8. Den NXT-Baustein einschalten bzw. das gewünschte Programm starten, wo die Bluetooth-Verbindung benötigt wird.

9. Auf den Button „Setting“ klicken und den COM Port auswählen, der gerade für die Bluetooth-Verbindung benutzt wird. Die gewünschte Funktion „Analog Stick Control“ für die Steuerung des NXT oder „NXT Data Acquisition“ für die Daten-Übermittlung auswählen und dann auf „Connect“ klicken.



Sobald die Bluetooth-Verbindung aktiv ist, erscheint folgendes Dialogfeld und am NXT LCD Bildschirm erscheint ein [BT] für Bluetooth.





10. Auf den Button „Start“ klicken, um die Daten-Aufnahme zu starten. Sobald auf „Stop“ geklickt wird, wird die Daten-Aufnahme unterbrochen und man kann die bisherigen aufgenommenen Daten in einer CSV-Datei abspeichern.

Sobald die Bluetooth-Verbindung einmal eingerichtet ist, müssen, wenn man nochmals das „NXTGamePad“-Programm benutzen will, nur noch die Punkte 8 bis 10 durchgeführt werden.

Damit keine Verbindungsprobleme entstehen, sollte man zuerst das „NXTGamePad“-Programm schließen und dann erst das Programm am NXT-Baustein beenden.

Eine Installationsanleitung auf Englisch ist auch unter folgendem Link, auf der Herstellerseite der Bluetooth-Software, zu finden: http://lejos-osek.sourceforge.net/nxtgamepad.htm.

Anhang D: Bedienungsanleitung

Folgende Bedienungsanleitung erklärt schrittweise, wie man die MATLAB/Simulink Ballbot-Software bedient und wie man sie auf den NXT-Baustein lädt.

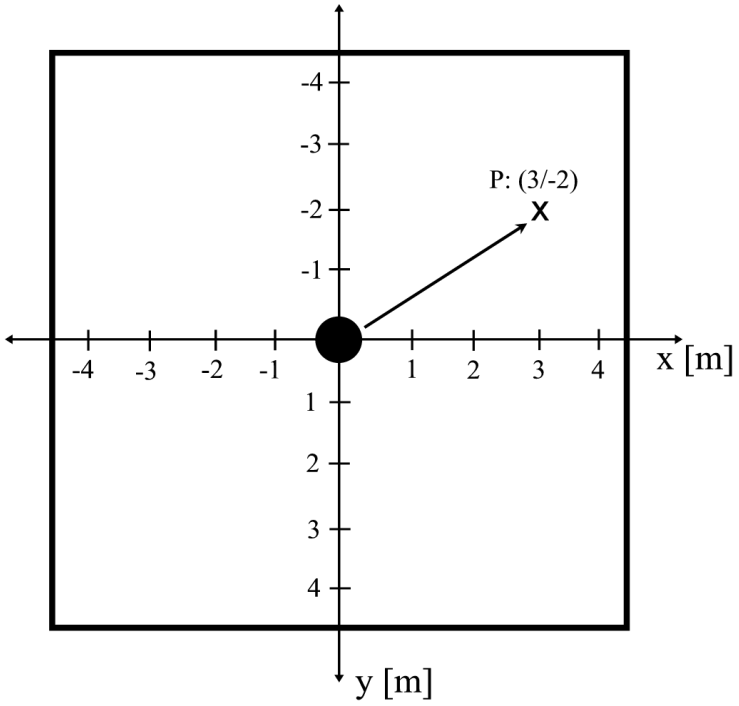
Die gesamte Software befindet sich auf der beigelegten CD im Ordner „Ballbot Programm“. Dieser Ordner muss auf den lokalen Rechner kopiert werden. Anschließend öffnet man in MATLAB das m-File „init_Ballbot.m“, welches sich in diesem Ordner befindet. Dieses m-File dient der Initialisierung des Ballbots und muss als erstes ausgeführt werden. Durch dieses m-File werden alle Parameter initialisiert, die für das Regeln des Ballbots nötig sind. Falls man die Parameter des linear-quadratischen Reglers ändern will, so muss man die Werte für k_LQR und k_I in den dazugehörigen m-Files ändern.
Den Ballbot kann man in zwei Betriebsmodi bedienen. Einmal, in dem er in seiner Ausgangsposition balanciert und einmal, in dem er bestimmte Koordinaten in der Ebene anfährt. Damit der Ballbot balanciert, muss in der „init_Ballbot.m“-Datei die Variable „flag_modus“ auf eins gesetzt werden. Will man hingegen den zweiten Betriebsmodus aktivieren, so muss die Variable auf zwei gesetzt werden. Wird dieser Betriebsmodus aktiviert, so müssen noch folgende Variablen bestimmt werden:

```
% pos_x = ???;
% pos_y = ???;
% start_time = ???;
```

Diese Variablen haben folgende Bedeutung:

pos_x: x-Koordinate der anzufahrenden Position in Meter. Ein positiver Wert bedeutet eine Bewegung nach rechts, ein negativer Wert eine Bewegung nach links.
pos_y: y-Koordinate der anzufahrenden Position in Meter. Ein positiver Wert bedeutet eine Bewegung nach vor, ein negativer Wert eine Bewegung zurück.
start_time: Zeit, ab wann die Fortbewegung beginnen soll. Bis dorthin balanciert der Ballbot in seiner Ausgangsposition.

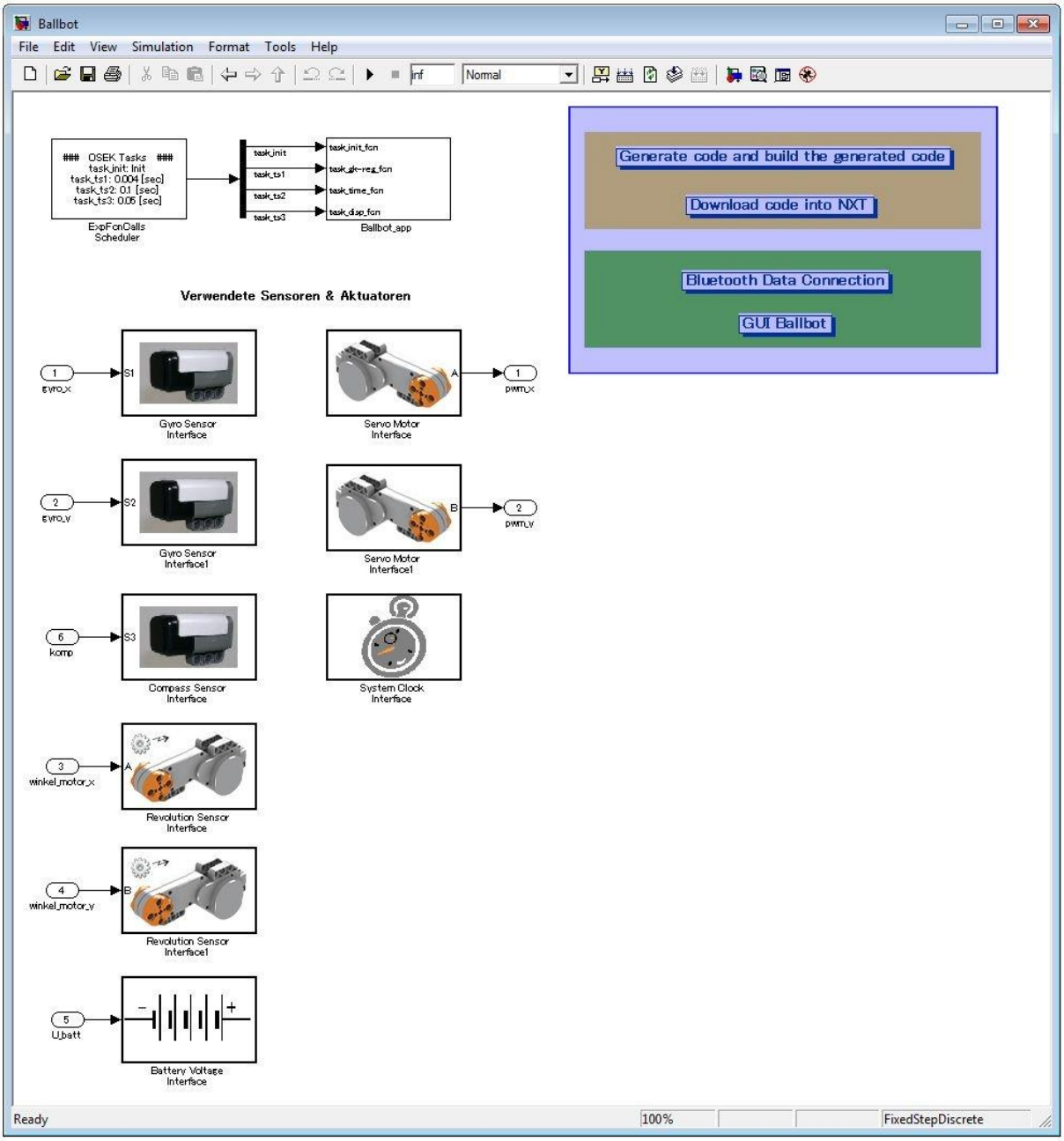
Will man z.B. den Ballbot, wie in folgender Abbildung grafisch dargestellt, auf eine bestimmte Position in der Ebene fortbewegen, so müssen die zuvor beschriebenen Variablen in der „init_Ballbot.m“-Datei folgendermaßen geändert werden.




```
flag_modus = 2;
pos_x = 3;
pos_y = -2;
start_time = 10000;
```

Ändert man die Codezeilen dementsprechend, so balanciert der Ballbot erst 10 Sekunden in seiner Ausgansposition und bewegt sich dann zur gewünschten Position hin.

Nachdem man alle Einstellungen in der MATLAB „init_Ballbot.m“-Datei vorgenommen hat und die Datei ausgeführt hat, öffnet man im selben Ordner das Simulink Modell „Ballbot.mdl“. Folgender Koppelplan ist dann am Bildschirm ersichtlich:



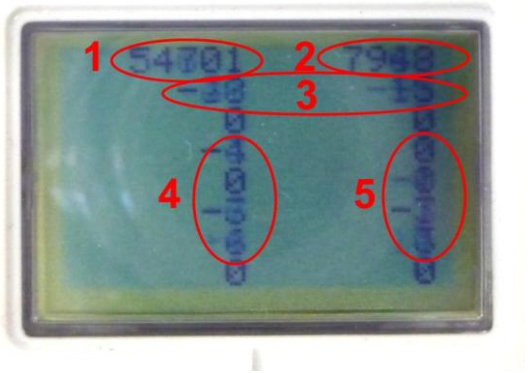
Für die Bedienung des Programms ist nur der blau markierte Bereich von Relevanz.

Will man nun das Programm mit dem zuvor eingegebenen Parameter auf den Ballbot laden, so muss man folgendermaßen vorgehen:

1. NXT-Baustein per USB mit dem Rechner verbinden
2. Auf die Schaltfläche „Generate code and build the generated code“ klicken
3. NXT-Baustein einschalten
4. Auf die Schaltfläche „Download code into NXT“ klicken

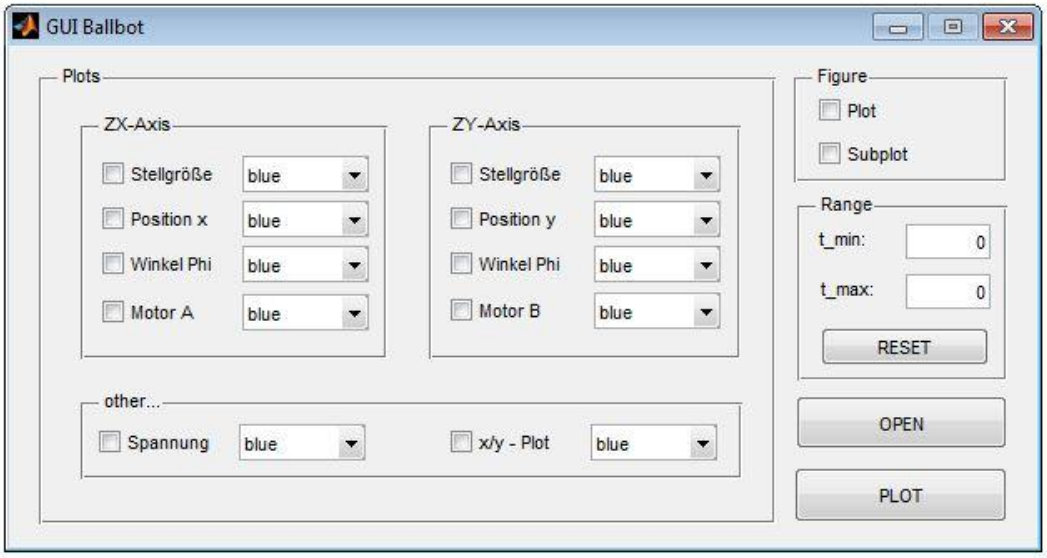
Nachdem man diese vier Punkte durchgeführt hat, befindet sich das Programm auf dem LEGO NXT-Baustein. Um das Programm starten zu können, muss noch durch die Betätigung der orangen Taste am NXT-Baustein in den Menüpunkt „My Files“ gewechselt werden, um dort ins Untermenü „Software files“ durch nochmaliges Betätigen der orangen Taste wechseln zu können. In diesem Untermenü befinden sich alle Programme, die auf den NXT Baustein geladen wurden. Hat man mit den Pfeiltasten am NXT das gewünschte Programm ausgewählt, so kann man es durch erneutes Betätigen der orangen Taste starten. Folgende Zeilen erscheinen dann am LCD-Bildschirm des LEGO-Bausteins:

Durch Betätigung der rechten Pfeiltaste am NXT-Baustein kann nun das Programm gestartet werden. Um das Programm zu stoppen, muss wieder die mittlere orange Taste gedrückt werden. Betätigt man die linke Pfeiltaste, so wird das Programm neu gestartet.
Sobald das Programm gestartet wurde, werden am LCD-Bildschirm des NXT folgende Werte angezeigt:

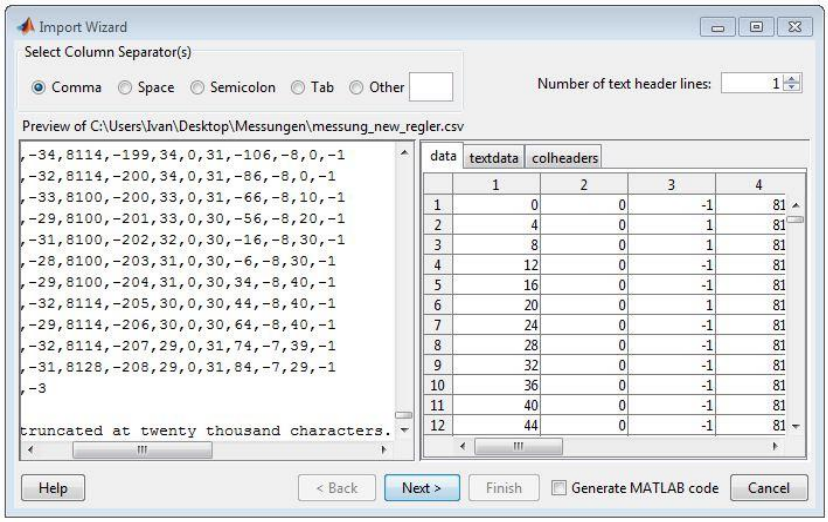


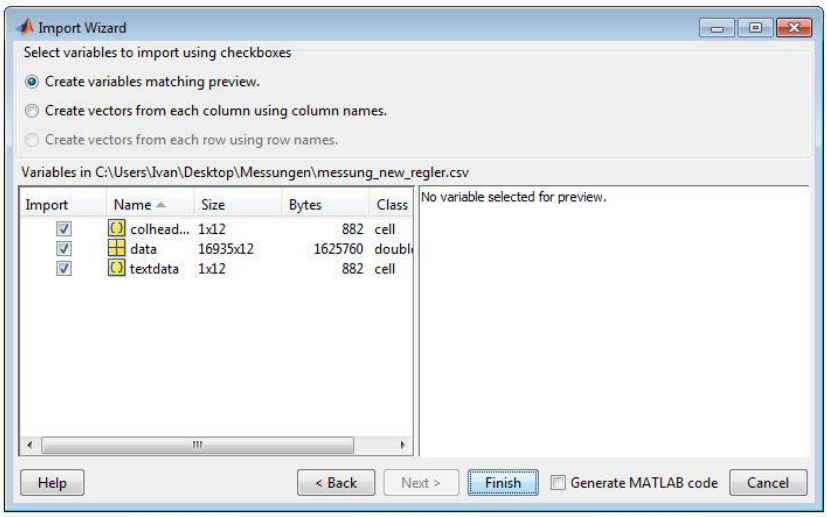
1: Systemzeit [Millisekunden]
2: Versorgungsspannung [mV]
3: Stellgrößen für die beiden Servomotoren [%]
4: Zustandsgrößen für die ZX-Achse (Position des Ballbots [cm], Kippwinkel des Ballbots [Grad], Geschwindigkeit des Ballbots [cm/Sek.], Kippwinkelgeschwindigkeit des Ballbots [Grad/Sek.])
5: Zustandsgrößen für die ZY-Achse (Position des Ballbots [cm], Kippwinkel des Ballbots [Grad], Geschwindigkeit des Ballbots [cm/Sek.], Kippwinkelgeschwindigkeit des Ballbots [Grad/Sek.])

Will man eine Messreihe aufnehmen, so muss zuerst eine Bluetooth-Verbindung zwischen dem NXT-Baustein und dem Rechner hergestellt werden. Damit dies geschehen kann, müssen zuerst, wie im Anhang C beschrieben, der Bluetooth-Adapter und die dazugehörige Software am Rechner installiert bzw. konfiguriert werden.
Durch Drücken auf die Schaltfläche „Bluetooth Data Connection“ im Simulink Koppelplan „Ballbot.mdl“ wird die Bluetooth-Software gestartet. Die Bedienung der Software ist im Anhang C beschrieben.
Hat man eine Messreihe aufgenommen, so sind die gesamten Daten in einer CSV-Datei hinterlegt. Um die Daten auszuwerten, drückt man auf die Schaltfläche „GUI Ballbot“ und es erscheint folgende grafische Oberfläche:



Durch dieses Programm ist man in der Lage, die Daten der CSV-Datei einzulesen und verschiedene Kennlinien darzustellen. Durch Klicken auf die Schaltfläche „OPEN“ kann eine CSV-Datei eingelesen werden. Zum Importieren der Daten muss dann anschließend, im sich öffnenden Fenster, auf „Next“ und dann auf „Finish“ gedrückt werden, ohne die Voreinstellungen zu ändern (siehe folgende Bilder):





Anschließend kann man im Abschnitt „Figure“ auswählen, ob man ein einzelnes Diagramm (Plot) oder ob man mehrere Diagramme in einem Fenster (sogenannte Subplots) darstellen lassen will. Unter dem Abschnitt „Plots“ kann man dann die Messreihen auswählen, die in dem Diagramm angezeigt werden sollen, dabei stehen folgende Messreihen zur Auswahl:

- Stellgröße der beiden Regler
- Position x und Position y des Ballbots
- Kippwinkel des Ballbots in beide Richtungen
- Motorumdrehung der beiden Motoren
- Betriebsspannung
- Die abgefahrene Strecke des Ballbots → x/y - Plot

Durch Klicken auf der Schaltfläche „PLOT“ werden dann die ausgewählten Diagramme gezeichnet.

Im Abschnitt „Range“ kann man durch Angabe von „t_min“ und „t_max“ den Bereich angeben, der im Diagramm geplottet werden soll. Durch Klicken auf den Button „RESET“ wird wieder der gesamte Messbereich geplottet.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] SEBASTIÁN SÁNCHEZ PRIETO, TOMÁS ARRIBAS NAVARRO, MARIANO GÓMEZ PLAZA, ÓSCAR RODRÍGUEZ POLO: A Monoball Robot Based on LEGO Mindstorms, IEEE Control Systems Magazine, April 2012, Seite 71 - 83

[2] CLAUDIA FRISCHKNECHT, THOMAS OTHER: LEGO Mindstorms NXT – Next Generation -, Semesterarbeit, ETH Zürich, Juli 2006

[3] THORSTEN LEIMBACH, SEBASTIAN TRELLA: Roberta® – Lernen mit Robotern, Spezifikation und Evaluation des LEGO Mindstorms NXT Systems, Fraunhofer-Institut IAIS, April 2011

[4] YORIHISA YAMAMOTO: NXT Ballbot Building Instructions, 2009, http://www.mathworks.com/matlabcentral/fileexchange/23931-nxt-ballbot-self-balancing-robot-on-a-ball-controller-design

[5] TAKASHI CHIKAMASA: Embedded Coder Robot NXT Modeling Tips, 2010, http://www.mathworks.com/matlabcentral/fileexchange/13399

[6] KARSTEN BERNS, DANIEL SCHMIDT: Programmierung mit LEGO MINDSTORMS NXT, Springer, 2010

[7] YORIHISA YAMAMOTO: NXT Ballbot Model-Based Design – Control of self-balancing robot on a ball, built with LEGO Mindstorms NXT, 2009, http://www.mathworks.com/matlabcentral/fileexchange/23931-nxt-ballbot-self-balancing-robot-on-a-ball-controller-design

[8] JAN LUNZE: Regelungstechnik 2 - Mehrgrößensysteme Digitale Regelung, 3., neu bearbeitete Auflage, Springer Verlag, Bochum 2004