



# Dynamisches Verhalten eines omnidirektionalen mobilen Roboters mit aktiven Schwenkrädern

## Masterthesis MSE

Institut für Entwicklung Mechatronischer Systeme EMS

Verfasser: Stefan Landis

Matrikel-Nr.: 09-250-747

Fachbereich: Automation und Robotik

Betreuer: Prof. Einar Nielsen

Prof. Dr. Urs Graf

Experte: PhD Sjur J. Vestli

Abgabedatum: Mai 2015

## Abstract

<b>Initial Status</b>	<p>An omnidirectional mobile robot is characterized by its capability to move freely in the plane and therefore has three degrees of freedom. This capability is achieved by using certain types of wheels. One of these wheel type, the caster wheel, has an eccentricity of the steering axis to the rotation axis of the wheel. The advantage of omnidirectional mobile robots with active caster wheels over other omnidirectional drive systems is that standard wheels can be used. So the most suitable wheel can be selected for each surface depending on the flatness, degree of soiling etc.</p> <p>Two previous master theses have focused on developing a highly dynamic omnidirectional mobile robot with active caster wheels (OmniMoBot) so far. However the work on OmniMoBot has not yet been completed.</p>
<b>Tasks</b>	<p>In this master thesis the OmniMoBot will be revised and put into operation. For commissioning the OmniMoBot the fitting on the Hall-Sensorprints, an additional safety circuit and the cabling is required. In addition the software needs to be redeveloped.</p> <p>To demonstrate the capabilities of mobile robots with active caster wheels, a pointed aluminum rod is balanced by the OmniMoBot during rest and during driving. In addition to balancing the OmniMoBot should be able to move with the help of a laser scanner without causing a collision.</p>
<b>Procedure</b>	<p>The kinematics of the OmniMoBots was derived. Afterwards various drive control concepts for the OmniMoBot were presented. The behavior of the balancing robot could be analyzed by a simulation with Matlab / Simulink. The setup of the simulated control structure was realized in C ++ by using the framework EEROS.</p>
<b>Key Results</b>	<p>A well comprehensible, simple and effective control structure for the OmniMoBot resulted. By the implemented control structure the OmniMoBot is able to balance the aluminum rod well.</p> <p>Using the data of laser scanner, a collision avoidance has been realized, which converts the operator's instructions or the instructions given by the path planner as closely as possible, while maintaining the speed as high as possible. The collision avoidance not only prevents colliding with an obstacle but it also actively supports the path planner or the operator by driving around the obstacle.</p>
<b>Recommendations</b>	<p>The speed of OmniMoBots is currently limited to 1 m/s. If the speed is to be increased, it is recommended to involve the dynamics because at higher speeds the influence of the dynamics increases.</p> <p>When the whirls change sides ("shopping-cart effect") very high accelerations can occur. The maximum speed, at which this changing sides process can be achieved by the motors, is 1.96 m/s.</p> <p>The OmniMoBot is suitable for further work in the field of path planning.</p>



OmniMoBot

## Kurzfassung

<b>Ausgangs-Situation</b>	<p>Ein omnidirektionaler mobiler Roboter zeichnet sich dadurch aus, dass er sich in der Ebene frei bewegen kann und somit drei Freiheitsgrade besitzt. Diese Fähigkeit wird mittels bestimmter Radtypen erreicht. Einer dieser Radtypen ist das Schwenkrad, welches eine Exzentrizität der Lenkachse zu der Drehachse des Rades aufweist. Der Vorteil von omnidirektionalen mobilen Robotern mit aktiven Schwenkrä dern gegenüber anderen omnidirektionalen Antriebssystemen ist, dass Standardräder verwendet werden können. Damit kann für jeden Untergrund je nach Ebenheit, Verschmutzungsgrad etc. das dafür geeignete Rad gewählt werden.</p> <p>Zwei vorangegangene Masterthesen haben sich bereits mit der Entwicklung eines hoch-dynamischen omnidirektionalen mobilen Roboters mit aktiven Schwenkrä dern (Omni-MoBot) befasst. Die Arbeiten am OmniMoBot konnten jedoch nicht abgeschlossen werden.</p>
<b>Aufgabenstellung</b>	<p>In dieser Masterthesis wird der OmniMoBot überarbeitet und in Betrieb genommen. Für die Inbetriebnahme sind das Einpassen des Hall-Sensorprints, eine zusätzliche Sicherheitsschaltung und die Verkabelung erforderlich. Zudem muss die Software neu entwickelt werden.</p> <p>Um die Fähigkeiten von mobilen Robotern mit aktiven Schwenkrä dern zu demonstrieren, wird mit dem OmniMoBot ein zugespitzter Aluminiumstab ausbalanciert, sowohl im Stillstand als auch während dem Fahren. Zusätzlich zum Balancieren soll sich der Roboter mit Hilfe eines Laserscanners kollisionsfrei bewegen.</p>
<b>Vorgehen</b>	<p>Die Kinematik des OmniMoBots wurde hergeleitet. Anschliessend wurden verschiedene Antriebs-Regelungskonzepte für den OmniMoBot vorgestellt. Mittels einer Simulation mit Matlab/Simulink konnte das Verhalten des balancierenden Roboters analysiert werden. Der Aufbau der simulierten Regelungsstruktur wurde mittels des Frameworks EEROS in C++ realisiert.</p>
<b>Wesentliche Ergebnisse</b>	<p>Für den OmniMoBot entstand eine gut nachvollziehbare, einfache und effektive Regelungsstruktur. Mittels der implementierten Regelungsstruktur ist der OmniMoBot in der Lage, den Aluminiumstab gut auszubalancieren.</p> <p>Mit den Daten eines Laserscanners wurde eine Kollisionsvermeidung realisiert, welche die Befehle des Bedieners oder die Anweisungen des Bahnplaners möglichst genau umsetzt und dabei die Geschwindigkeit so hoch wie möglich beibehält. Die Kollisionsvermeidung verhindert nicht nur die Kollision mit dem Hindernis, sondern unterstützt den Bahnplaner oder den Bediener aktiv beim Umfahren eines Hindernisses.</p>  <p style="text-align: right;">OmniMoBot</p>
<b>Empfehlungen</b>	<p>Die Geschwindigkeit des OmniMoBots ist momentan auf 1 m/s beschränkt. Falls die Geschwindigkeit erhöht werden soll, wird empfohlen, die Dynamik miteinzubeziehen, weil bei höheren Geschwindigkeiten der Einfluss der Dynamik steigt.</p> <p>Beim Umschwenken der Räder („shopping-cart effect“) können sehr hohe Beschleunigungen auftreten. Die maximale Geschwindigkeit, bei welcher dieses Umschwenken durch die Motoren noch erfüllt werden kann, liegt bei 1.96 m/s.</p> <p>Der OmniMoBot eignet sich für weitere Arbeiten im Bereich Bahnplanung.</p>

## Inhalt

<b>1 Einleitung</b>	<b>7</b>
1.1 Aufgabenstellung	8
1.2 Kapitelübersicht	9
<b>2 Theoretischer Hintergrund</b>	<b>10</b>
2.1 Mobile Roboter	10
2.2 Kinematik des OmniMoBots	11
2.2.1 Berechnung der direkten Jacobimatrix	14
2.2.2 Berechnung der inversen Jacobimatrix	20
2.3 Fahrmöglichkeiten	22
2.4 Einfluss der Exzentrizität des Schwenkrades	23
2.5 Regelungskonzepte der Antriebe	25
2.5.1 Allgemeine Regelungskonzepte in der Robotik für die Motorachsen	25
2.5.2 Antriebs-Regelungskonzept für den OmniMoBot	29
2.6 Die wirkenden Massenträgheitsmomente des OmniMobot	31
2.6.1 Antrieb	31
2.6.2 Lenkung	34
2.6.3 Vereinfachte Massenmatrix für den OmniMoBot	35
2.7 Inverses Pendel	35
2.7.1 Inverses Pendel: Theorie	36
2.7.2 Inverses Pendel: OmniMoBot	37
2.7.3 Inverses Pendel: Regelungskonzept	38
2.8 EEROS	39
2.9 fLINK	42
<b>3 Simulation mit Simulink</b>	<b>45</b>
3.1 Aufbau des Simulationsmodells	45
3.1.1 Subsystem „Trajectory generator“	46
3.1.2 Subsystem „Pendulum control“	47
3.1.3 Subsystem „OmniMoBot“	47
3.2 Simulationsergebnisse	49
3.2.1 Ergebnis: Spitzenregelungsfrequenz 0.159 Hz, Winkelregelungsfrequenz 1.59 Hz	51
3.2.2 Ergebnis: Spitzenregelungsfrequenz 0.915 Hz, Winkelregelungsfrequenz 3.183 Hz	51
<b>4 Praktische Umsetzung</b>	<b>53</b>

4.1	OmniMoBot .....	53
4.2	Hardware-Probleme und Entscheidungen .....	54
4.2.1	Drive-Board-Entscheidung.....	54
4.2.2	Mikrocontroller.....	56
4.2.3	Hall-Sensorprint.....	56
4.2.4	Sicherheitsschaltung .....	59
4.3	Software .....	60
4.3.1	Control System.....	60
4.3.2	Safety System .....	66
4.3.3	Zu eruierende Fehler in EEROS und fLink.....	69
4.4	Kollisionsvermeidung durch Laserscanner „URG-04LX“ .....	69
4.4.1	Vektorielle Funktionsweise der Kollisionsvermeidung.....	69
4.4.2	Softwarestruktur der Kollisionsvermeidung .....	76
4.5	Ergebnisse .....	80
4.5.1	Kalibrierung des Stabwinkels.....	81
4.5.2	Ausbalancieren des Stabes .....	82
4.5.3	Fazit der Fähigkeiten des OmniMoBots .....	87
<b>5</b>	<b>Schlussfolgerung und Ausblick.....</b>	<b>88</b>
<b>6</b>	<b>Verzeichnisse .....</b>	<b>89</b>
6.1	Formelzeichen .....	89
6.2	Abkürzungsverzeichnis.....	91
6.3	Abbildungen .....	92
6.4	Tabellen .....	94
6.5	Literatur- und Quellenverzeichnis .....	95
<b>7</b>	<b>Erklärung zur Urheberschaft .....</b>	<b>97</b>
<b>Anhang</b>	<b>.....</b>	<b>98</b>
A.	Simulink Simulation .....	98
I.	Spannungsvorgabe („Duty Cycle“ des PWMs).....	98
II.	M-File für Simulation (Abbildung 35, S. 46) .....	100
III.	S-Funktion Jacobimatrix .....	102
IV.	S-Funktion inverse Jacobi .....	103
B.	Test ESCON Module 50/5 .....	104
C.	Nullposition der Homingsensoren bestimmen.....	105
D.	Elektronische Bauteile .....	105
I.	Mainboard .....	106
II.	Drive-Board des OmniMoBots („NTB 6xDC“).....	107
III.	HMI .....	107

---

IV.	Hall-Sensorprint.....	110
V.	Verwendete Motoren .....	111
VI.	Laserscanner.....	112
VII.	Batterie.....	114
E.	Schaltplan OmniMoBot.....	114
F.	CD.....	119

## 1 Einleitung

### Motivation

Der Anwendungsbereich von mobilen Robotern kennt keine Grenzen. Anwendungen, in denen sich mobile Roboter bereits etabliert haben, sind zum Beispiel der Transport von Waren in der Logistikbranche, gefährliche Aufgaben wie das Entschärfen von Sprengsätzen oder das Erforschen von fremden Planeten im Rahmen von Raumfahrtprogrammen. Im Moment sind in der Industrie häufiger stationäre als mobile Roboter im Einsatz.

Die Fähigkeit, omnidirektional zu fahren, ist für jeden mobilen Roboter vorteilhaft. Omnidirektionales Fahren bedeutet, dass in alle drei Freiheitsgrade eines mobilen Roboters, gleichzeitig und unabhängig voneinander, eine Geschwindigkeit vorgegeben werden kann. Das selbständige omnidirektionale Fahren kann durch aktiv angetriebene Spezial- oder Schwenkräder erreicht werden. Beim Schwenkradantriebssystem muss die Aufhängung des Rades gegenüber der Lenkachse eine Exzentrizität aufweisen.

Der grösste Vorteil des Schwenkradantriebssystems gegenüber anderen omnidirektionalen Antriebssystemen besteht in der Verwendbarkeit von Standardrädern. Solche können für jeden Untergrund - je nach Ebenheit, Verschmutzungsgrad etc. - gezielt ausgewählt werden. Deswegen lohnt es sich, die Möglichkeiten und Fähigkeiten dieses Antriebssystems genauer zu erforschen.

Zukünftig kann ein solches Antriebssystem zum Beispiel in Fortbewegungsmitteln für gehbehinderte Menschen im Alltag eingesetzt werden. Ein Fortbewegungsmittel mit diesem Antriebssystem benötigt weniger Platz als ein Rollstuhl und kann aus dem Stand jede beliebige Position anfahren. Dies würde gehbehinderten Menschen die Arbeit in vielen Bereichen ermöglichen oder erleichtern, etwa beim Verkauf an einer Theke, Arbeiten im Labor oder in einer Arztpraxis.

### Stand der Technik

In der Publikation [3] wird die Balance eines Menschen mittels einer Plattform mit aktiven Schwenkrä dern analysiert, was einem Fortbewegungsmittel mit diesem Antriebskonzept nahekommt.

Die Kinematik von omnidirektionalen mobilen Robotern mit Rädern ist bereits in früheren Publikationen untersucht worden, zum Beispiel in der Publikation [12]. Einen umfassenden und aktuellen Überblick zu Regelung, Kinematik und Dynamik von mobilen Robotern mit aktiven Schwenkrä dern bietet die Doktorarbeit von El-Shemawy [4]. Diese Doktorarbeit wird mit der vorliegenden Masterthesis um eine einfache und effektive Regelungsstruktur ergänzt. Ferner werden mit dem neu konstruierten, omnidirektionalen mobilen Roboter (OmniMoBot) die Fähigkeiten dieses Antriebssystems verdeutlicht.

### Ziel der Arbeit

Das Ziel dieser Masterthesis ist die Realisierung eines hochdynamischen mobilen Roboters. Der omnidirektionale mobile Roboter OmniMoBot [6] soll überarbeitet und in Betrieb genommen werden. Mit diesem sollen die Fähigkeiten des Schwenkradantriebs aufgezeigt werden, indem er einen zugespitzten Aluminiumstab ausbalanciert, sowohl im Stillstand als auch während dem Fahren. Zusätzlich zum Balancieren soll er sich mit Hilfe eines Laserscanners kollisionsfrei bewegen.

Dabei müssen folgende Punkte (*Handbook of Robotics* [2], S. 407) berücksichtigt werden, die die Schwierigkeiten beschreiben, welche bei der Umsetzung von mobilen Robotern mit aktiven Schwenkrä dern auftreten:

- Der Schwerpunkt eines Fahrzeugs mit aktiven Schwenkrä dern kann sich gegenüber den Bodenkontaktpunkten der Räder verschieben. Daraus resultiert eine variable Kippstabilität, welche auf den Extremfall ausgelegt werden muss.
- Durch die Exzentrizität der Räder kann ein Umschwenken der Räder stattfinden, der „shopping-cart effect“, woraus hohe Beschleunigungen in der Lenkung resultieren.
- Für das Antreiben des Schwenkrades muss eine Übersetzung verwendet werden, um zu verhindern, dass bei einer Lenkbewegung das Kabel des Antriebsmotors aufgewickelt wird.
- Ein Fahrzeug mit aktiven Schwenkrä dern braucht für das omnidirektionale Fahren mindestens vier Aktoren, woraus ein überbestimmtes System entsteht. Beim überbestimmten System steigen die Ansprüche an die Regelung der Aktoren.

<b>Anwendungsziel des OmniMoBots</b>	Das Anwendungsziel ist, dass der OmniMoBot zu Demonstrationszwecken eingesetzt werden kann. Er soll durch das Ausbalancieren eines Stabes die Fähigkeiten des Schwenkradantriebs aufzeigen und sich zukünftig mit zusätzlichen Aufbauten autonom und kollisionsfrei durch ein Gebäude bewegen. Dabei soll er unter anderem auch einen Aufzug benutzen können.
--------------------------------------	---

## 1.1 Aufgabenstellung

<b>Ausgangslage</b>	Die Arbeiten an der Mechanik des auf aktiven Schwenkrä dern basierenden OmniMoBots sind grösstenteils abgeschlossen [6]. Die Elektronik [5] muss überarbeitet werden, was jedoch nicht Teil der vorliegenden Arbeit ist. Für die Inbetriebnahme des OmniMoBot ist weiter die Einpassung des Hall-Sensorprints, eine zusätzliche Sicherheitsschaltung und die Verkabelung erforderlich. Zudem muss die Software neu entwickelt werden.
---------------------	---

<b>Aufgabenbereich</b>	<p>Der Aufgabenbereich umfasst folgende Punkte:</p> <ul style="list-style-type: none"> <li>• Die Kinematik der Plattform untersuchen, indem der Roboter soweit ausprogrammiert wird, dass mit einem Joystick ein gesteuertes Fahren möglich ist.</li> <li>• Die Regelung der Antriebe optimal auslegen und mögliche Regelungskonzepte aufzeigen.</li> <li>• Eine Regelung für ein inverses Pendel (balancieren eines Stabs) mit Hilfe von Simulationen untersuchen, um sie anschliessend in die Plattform zu implementieren.</li> <li>• Erlangen einer Positionsschätzung aufgrund von Laserscannerdaten.</li> <li>• Entwicklung einer Strategie, mit welcher basierend auf Laserscannerdaten Kollisionen vermieden werden können.</li> <li>• Evtl. weitere Demonstrationsmöglichkeiten aufzeigen.</li> </ul>
------------------------	---

**Rahmen-  
bedingungen**

Es gelten folgende Rahmenbedingungen:

- Der OmniMoBot soll vollständig mit dem Framework „EEROS“ programmiert werden.
- Zukünftige Ergänzungen wie z.B. ein Roboterarm sollen bei der Überarbeitung der Elektronik berücksichtigt werden.
- Es soll untersucht werden, ob der Einsatz von ROS-Komponenten sinnvoll ist.

## 1.2 Kapitelübersicht

---

**Kapitel 2**

Die verschiedenen Antriebssysteme von omnidirektionalen mobilen Robotern werden erläutert. Anschliessend wird die Kinematik des OmniMoBots hergeleitet. Zudem werden verschiedene Fahrmöglichkeiten aufgeführt, Regelungskonzepte für den OmniMoBot vorgestellt, die wirkenden Massenträgheitsmomente berechnet, die Theorie zum Inversen Pendel dargelegt und fLink sowie das Software Framework EEROS vorgestellt.

**Kapitel 3**

Die Kinematik, das Regelungskonzept, die wirkenden Trägheitsmomente sowie das inverse Pendel in Simulink werden zusammengefügt und getestet.

**Kapitel 4**

Der OmniMoBot, die Hardware, die aufgetretenen Probleme, die Software des OmniMoBots sowie die Ergebnisse werden vorgestellt.

**Kapitel 5**

Das Ergebnis dieser Masterthesis wird zusammenfassend beschrieben. Zusätzlich werden offene Punkte behandelt und die Arbeiten, welche zur Erfüllung des Anwendungszwecks des OmniMoBots noch notwendig sind, werden aufgeführt.

## 2 Theoretischer Hintergrund

### 2.1 Mobile Roboter

**Allgemein [13]** Die Mobilität der Roboter kann mittels verschiedener Fortbewegungsarten erreicht werden. Es gibt Roboter mit Rädern und solche mit Beinen sowie kriechende, fliegende, schwimmende etc. In dieser Arbeit liegt das Augenmerk auf omnidirektionalen, mobilen Robotern mit Rädern.

Die Kriterien für die Definition eines Fahrwerks sind die Anzahl der Räder, der verwendete Radtyp und die geometrische Anordnung der Räder zueinander. Relevant für die Mobilität des Roboters ist die Anzahl und Position der Räder. Ein Roboter kann mit drei Rädern statisch stabil sein, vorausgesetzt der Schwerpunkt liegt innerhalb des Dreiecks, welches die drei Räder aufspannen. Ein mobiler Roboter mit nur zwei Rädern ist stabil, wenn sich der Schwerpunkt unterhalb der gemeinsamen Radachse befindet. Hat ein mobiler Roboter mehr als drei Räder, müssen diese gefedert sein, damit der Bodenkontakt von allen Rädern auch bei leichter Unebenheit des Untergrunds garantiert ist.

#### Omni-direktionale Radtypen

Mobile Roboter mit omnidirektionalen Rädern können nach vier Radtypen unterschieden werden [1]:

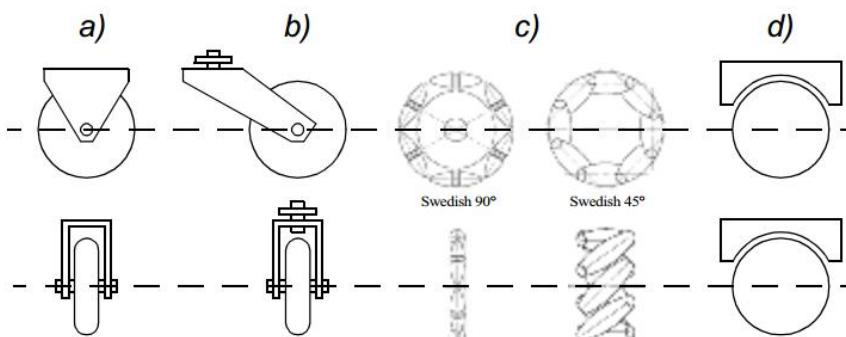


Abbildung 1: Radtypen [1]

- a) **Standardrad**

Das Standardrad ist am häufigsten anzutreffen, zum Beispiel bei einem Fahrrad mit gelenkter und fester Variante. Bei diesem Radtyp ist die Lenkachse über dem Kontaktpunkt von Rad und Ebene. Dieser Radtyp ist im Stillstand nicht-holonom, das bedeutet, dass von den drei Freiheitsgraden nur in zwei Freiheitsgraden eine Bewegung vorgegeben werden kann.

- b) **Schwenkrad (Caster Wheels)**

Das Schwenkrad ist unter anderem bei Bürostühlen und Einkaufswagen anzutreffen. Es erlaubt omnidirektionales Fahren auch aus dem Stillstand, was mit dem Versatz der Lenkachse zum Kontaktpunkt ermöglicht wird [2].

- c) **Mecanum-Rad (Swedish Wheels)**

Das Mecanum-Rad gestattet durch mehrere drehbar gelagerte Rollen, die auf dem Umfang des Rades angebracht sind, ebenfalls eine omnidirektionale Bewegung. Für diesen Radtyp ist ein ebener Boden wichtig, weil gegenüber Schwenkräden der Bodenkontakt schlechter ist [2].

Der grösste Nachteil dieses Radtyps ist, dass bereits im Verhältnis zu den

Rädern kleine Schwellen ein grosses Hindernis für die Rollen mit dem geringen Durchmesser darstellen können.

- d) **Kugel-Rad**

Ein Kugel-Rad muss mittels weiterer Rollen oder Räder angetrieben werden. Das hat zur Folge, dass diese Art von Antrieb sehr schmutzempfindlich ist. Schmutz auf der Oberfläche der Kugel verändert ihren Reibungskoeffizienten, dies hat einen direkten Einfluss auf die Kraft, die von den antreibenden Rollen übertragen werden kann.

Die Nachteile des Mecanum-Rades und des Kugel-Rades sowie das Fehlen des dritten Freiheitsgrads im Stillstand des Standardrads, sind die Gründe, weshalb das Antriebskonzept des „OmniMoBots“ mit aktiven Schwenkräder umgesetzt wurde.

**Annahmen** Für die weiteren Berechnungen werden für mobile Roboter mit Rädern folgende Vereinfachungen gemacht [14]:

- Sie bewegen sich ausschliesslich auf ebenem, horizontalem Untergrund.
- Jedes Rad hat nur einen Kontaktpunkt mit der Ebene.
- Es tritt kein Schlupf in den Kontaktpunkten auf, weder in Richtung der Rollbewegung des Rads noch quer dazu.

Durch diese Vereinfachungen führt das Rad eine reine Rollbewegung aus. Gleichzeitig ist eine Rotation in der Lenkachse möglich. Damit ist der Roboter nur in der Lage, Bewegungen in der Ebene durchzuführen.

**Roll- und Gleitbedingung** Durch die gemachten Annahmen ist es möglich, für jeden Radtyp zwei Bedingungen anzugeben [2]:

- **Rollbedingung:** Berücksichtigt das feste Verhältnis von der Rotation um die Radachse zu der zurückgelegten Strecke.
- **Gleitbedingung:** Bei den Radtypen a und b aus Abbildung 1 lässt die Gleitbedingung im Kontaktpunkt keine Bewegung quer zur Rollbewegung des Rades (orthogonal zur Fahrtrichtung des Rades) zu.

Aus diesen beiden Bedingungen kann direkt durch die Betrachtung der auftretenden Geschwindigkeitsvektoren die inverse Jacobimatrix für einen mobilen Roboter hergeleitet werden [1].

## 2.2 Kinematik des OmniMoBots

**Allgemein** Wie in der Abbildung 2 ersichtlich ist, hat der OmniMoBot drei Schwenkräder. Dabei besitzt er pro Schwenkrad zwei Motoren, einen für die Lenkachse und einen für die Radachse. Somit sind alle vorhandenen Gelenke der Plattform durch Motoren angetrieben.

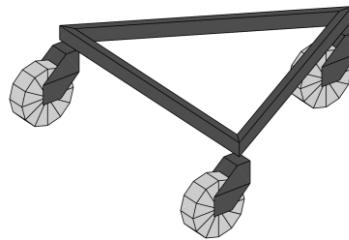


Abbildung 2: Plattform des OmniMbots

**Koordinaten-  
systeme**

Damit die Motoren wie gewünscht betrieben werden können, werden folgende Parameter und Koordinatensysteme (KS) eingeführt:

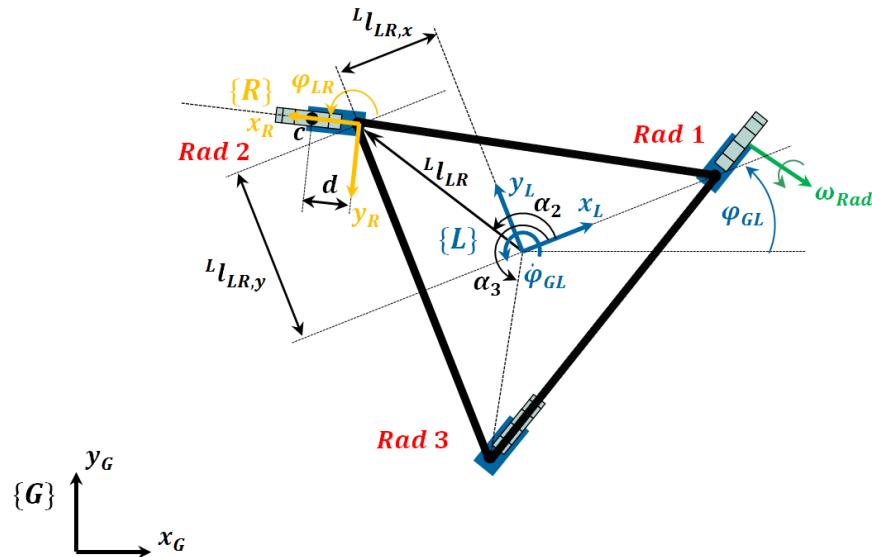


Abbildung 3: Parameter und Koordinatensysteme des OmniMoBot

Es werden drei verschiedene Koordinatensysteme eingeführt:  $\{G\}$  für das globale KS,  $\{L\}$  für das lokale und  $\{R\}$  für das KS des Rades. Der Kontaktpunkt  $c$  beschreibt die Position, bei welcher das Rad den Boden berührt. Mit einem Geschwindigkeitsvektor, welcher im Radkoordinatensystem  $\{R\}$  beschrieben wird, kann die Drehgeschwindigkeit des Rades und die Drehgeschwindigkeit der Lenkachse berechnet werden.

**Berechnung  
von  $\omega_{Rad}$  und  
 $\omega_{Lenk}$** 

Die Drehgeschwindigkeit des Rades kann für jedes Schwenkrad wie folgt berechnet werden:

$$\omega_{Rad} = {}^R v_{GR,x} \cdot \frac{1}{R} \quad (2-1)$$

- ${}^R v_{GR,x}$ : Ist die Geschwindigkeit, mit der sich das Radkoordinatensystem  $\{R\}$  bezogen auf das globale KS  $\{G\}$  bewegt. Diese Geschwindigkeit wird im Radkoordinatensystem  $\{R\}$  beschrieben. Das  $x$  steht für die  $x$ -Komponente des Radkoordinatensystems  $\{R\}$ .
- $R$ : Ist der Radius des Rades.

Die Drehgeschwindigkeit der Lenkachse wird folgendermassen berechnet:

$$\dot{\varphi}_{LR} = \omega_{Lenk} = -{}^R v_{GR,y} \cdot \frac{1}{d} \quad (2-2)$$

Aus der Abbildung 3 geht hervor, dass die Gleichung ( 2-2 ) negativ sein muss, weil die wirkende Geschwindigkeitskomponente  ${}^R v_{GR}$ , wird vom Kontaktpunkt  $c$  aus betrachtet, denn es wird nicht um die Lenkachse gedreht sondern um den  $c$ -Punkt.

**Position des Roboters [14]**

Das Fahrwerk (und damit der Roboter) wird als starrer Körper in einer waagrechten Ebene angesehen. In dieser Ebene hat das Fahrwerk drei Freiheitsgrade, zwei durch Translation und eine durch die Rotation. Das Fahrwerk selbst hat noch weitere innere Freiheitsgrade, welche durch die Rad- und Lenkachsen definiert sind. Diese sind jedoch für die Position des Roboters im Raum nicht von Belang.

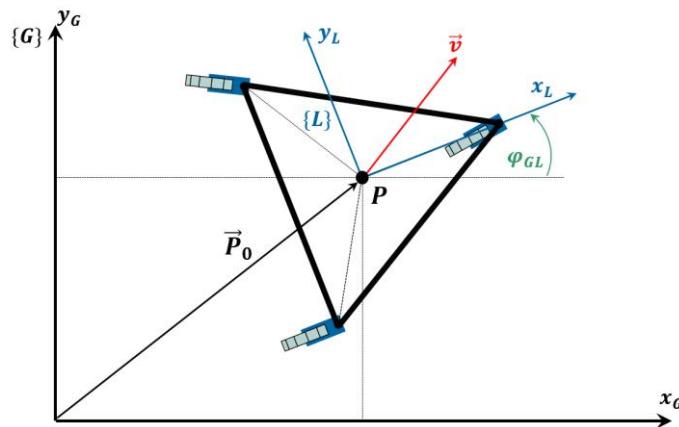


Abbildung 4: Globales und lokales Koordinatensystem

Durch die Transformation vom lokalen KS  $\{L\}$  in das globale KS  $\{G\}$  werden die drei Freiheitsgrade des Roboters im globalen KS  $\{G\}$  beschrieben.

Das globale KS  $\{G\}$  beschreibt die Position des Roboters  $\vec{P}_0$  aus der Sicht eines globalen Nullpunkts. Das lokale KS  $\{L\}$  hat den Referenzpunkt  $P$  als Ursprung und ist fix mit dem Roboter verbunden.

**Bewegung des Roboters [14]**

Die Bewegung (zeitliche Ableitung der Position und Orientierung) des Roboters kann im globalen KS  $\{G\}$  mit  ${}^G v_{GL,x}$ ,  ${}^G v_{GL,y}$  und  ${}^G \dot{\varphi}_{GL}$  eindeutig beschrieben werden. Um die Bewegung im lokalen KS  $\{L\}$  zu beschreiben, wird nur eine orthogonale Drehmatrix  ${}^L R_G$  benötigt.

$${}^L \vec{v}_{GL} = {}^L R_G \cdot {}^G \vec{v}_{GL} \quad (2-3)$$

In Komponentenschreibweise und mit dem dritten Freiheitsgrad wird die Rechnung wie folgt geschrieben:

$$\begin{bmatrix} {}^L v_{GL,x} \\ {}^L v_{GL,y} \\ {}^L \dot{\varphi}_{GL} \end{bmatrix} = \begin{bmatrix} \cos(\varphi_{GL}) & \sin(\varphi_{GL}) & 0 \\ -\sin(\varphi_{GL}) & \cos(\varphi_{GL}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^G v_{GL,x} \\ {}^G v_{GL,y} \\ {}^G \dot{\varphi}_{GL} \end{bmatrix} \quad (2-4)$$

**Direkte Kinematik**

Allgemein beschreibt in der Robotik die direkte Kinematik die Position und Orientierung des Endeffektors, kurz TCP (Vektor  $\vec{x}$  in Abbildung 5), als eine Funktion des Gelenkvektors  $\vec{q}$  (Winkel der Gelenke oder der Motoren). Aus den Positionen der Motoren kann also die Position des TCPs im kartesischen KS berechnet werden.

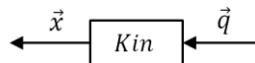


Abbildung 5: Direkte Kinematik

Bei einem mobilen Roboter existiert jedoch kein Zusammenhang zwischen der Position des Roboters und der Position der Motoren. Für die Regelung eines mobilen Roboters ist der dynamische Zusammenhang von Roboter und Motoren relevant. Wenn die Geschwindigkeiten der Räder bekannt sind, kann durch die Jacobimatrix der Geschwindigkeitsvektor  ${}^L\vec{v}_{GL}$  des mobilen Roboters berechnet werden. Im vorliegenden Fall hat jedes Schwenkrad zwei Freiheitsgrade. Somit haben die drei Schwenkräder zusammen sechs Freiheitsgrade.

Nun werden diese sechs Freiheitsgrade mittels der Jacobimatrix in die drei Freiheitsgrade des Roboters umgerechnet. Aus diesem Zusammenhang ist ersichtlich, dass das System überbestimmt ist.

**Inverser Kinematik**

Die inverse Kinematik berechnet aus gegebenen, kartesischen Koordinaten (Vektor  $\vec{x}$  in Abbildung 6) die zugehörigen Motorenstellungen  $\vec{q}$ . Bei einer gegeben Geschwindigkeit des Roboters können mit der inversen Jacobimatrix die Geschwindigkeiten der Räder definiert werden.

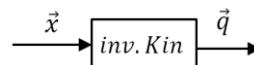


Abbildung 6: Inverse Kinematik

### 2.2.1 Berechnung der direkten Jacobimatrix

**Allgemein**

Bei der Herleitung der direkten Jacobimatrix werden die in Abbildung 3 definierten Parameter und Koordinatensysteme verwendet.

Die Ausgaben der direkten Jacobimatrix müssen als eine Schätzung angesehen werden. Die Berechnung der Geschwindigkeit des Roboters aus den Encoder-Daten ist niemals perfekt, denn es besteht immer die Möglichkeit, dass das hinterlegte Modell des Roboters, welches von der Jacobimatrix für die Berechnung verwendet wird, nicht exakt der Realität entspricht. Bei diversen Bereichen können kleine Abweichungen zur Realität auftreten, zum Beispiel bei der Grösse der Räder, der Grösse des Roboters, Abweichungen zwischen vorgegebener und richtiger Rotation, Rutschen der Räder, etc. Diese Fehler können alle einen Einfluss auf die Geschwindigkeitsberechnung haben (siehe [2], S. 477).

Die direkte Jacobimatrix des OmniMoBots lässt sich nicht als eine Matrix darstellen. Sie wird jedoch trotzdem als Matrix bezeichnet und beschreibt den vektoriellen Zusammenhang zwischen den Geschwindigkeiten in den drei Radkoordinatensystemen  $\{R\}_i$  und dem lokalen KS  $\{L\}$ .

**Odometrie [2]** Die Odometrie ist eine Methode zur Bestimmung von Position und Orientierung eines mobilen Roboters im globalen KS  $\{G\}$  mittels Integration der berechneten Geschwindigkeit. Wie zuvor erwähnt, ist die Geschwindigkeitsberechnung nur eine Schätzung, deswegen ist die daraus erhaltene Position ebenfalls eine Schätzung. Für die Überprüfung der Odometriedaten des Roboters sind zusätzliche Positionsschätzungen notwendig, welche mit verschiedenen Sensoren, zum Beispiel mit einem Laserscanner und/oder Beschleunigungssensor, durchgeführt werden können.

**Positions- und Geschwindigkeitsschätzung** Wenn eine Positionsschätzung mittels Beschleunigungssensoren erfolgt, besteht durch das Rauschen der Signale und durch dessen Integration die Gefahr, dass die geschätzte Position wegdriftet. Dieses Thema wird ausführlich in der Publikation [15] behandelt. Ein robuster Kalman-Filter, welcher mittels Beschleunigungssensordaten eine gute Geschwindigkeitsschätzung umsetzt, wird in der Publikation [16] vorgestellt.

**Durchdrehen eines Rades** Es besteht die Möglichkeit, dass ein Rad keinen Bodenkontakt mehr hat und frei durchdreht. Da die Geschwindigkeit der Räder geregelt wird, kann mit der direkten Jacobimatrix das Durchdrehen des Rades nicht ermittelt werden. Denn für die Berechnung der Geschwindigkeiten im kartesischen System verwendet die direkte Jacobimatrix nur die Encoder-Werte der Motoren. Trotzdem kann das Durchdrehen auf eine einfache Weise erkannt werden, indem der Stromverbrauch für ein frei durchdrehendes Rad gemessen wird. Dieser Wert kann dann als Vergleichswert verwendet werden. Wenn dieser Wert über eine bestimmte Dauer gemessen wird, ist anzunehmen, dass das Rad frei durchdreht. Gleichzeitig ist es sinnvoll den Stromverbrauch eines Motors bei konstanter Fahrt zu messen, damit ersichtlich wird, wieviel Strom die Reibung verbraucht. Dadurch wäre es wahrscheinlich möglich, das Verfahren etwas zu verfeinern.

**Rutschen eines Rades** Das Rutschen eines Rades kann, im Gegensatz zum Durchdrehen, mit der direkten Jacobimatrix erfasst werden. Da das System überbestimmt ist, kann ein rutschendes Rad, welches die Geschwindigkeitsschätzung der direkten Jacobimatrix beeinflusst, erkannt und ignoriert werden. Mit den zwei verbleibenden, nicht rutschenden Rädern kann trotzdem eine gute Geschwindigkeitsschätzung abgegeben werden.

- Ablauf des direkten Jacobi-Algorithmus**
1. Drehgeschwindigkeiten  $\omega_{Rad}$  und  $\omega_{Lenk}$  in die Geschwindigkeit  ${}^R\vec{v}_{GR}$  umrechnen (siehe Gleichung (2-1) und (2-2)). Daraus geht hervor, dass die Geschwindigkeit  ${}^R\vec{v}_{GR,y}$  negativ sein muss.
  2. Die Geschwindigkeit  ${}^R\vec{v}_{GR}$  mit einer Drehmatrix im lokalen KS  $\{L\}$  ausdrücken:

$$\begin{bmatrix} {}^L v_{GR,x} \\ {}^L v_{GR,y} \end{bmatrix}_i = \begin{bmatrix} \cos(\varphi_{LRi}) & -\sin(\varphi_{LRi}) \\ \sin(\varphi_{LRi}) & \cos(\varphi_{LRi}) \end{bmatrix} \cdot \begin{bmatrix} {}^R v_{GR,x} \\ {}^R v_{GR,y} \end{bmatrix}_i \quad (2-5)$$

$i$ : Ist ein Index, der für 1, 2 oder 3 stehen kann.

Wenn zu  $\varphi_{LRi}$  noch  $\varphi_{GL}$  addiert wird, ist die Geschwindigkeit nicht im lokalen KS  $\{L\}$  sondern im globalen KS  $\{G\}$  ausgedrückt.

3. Die durch die Encoder erhaltenen Geschwindigkeiten werden kontrolliert. Damit

lässt sich ermitteln, ob ein Rad gerutscht ist. Dies ist möglich, da die Räder mechanisch miteinander gekoppelt sind. Weiter wird durch Mitteln aus zwei kontrollierten Geschwindigkeiten eine mathematisch korrekte Geschwindigkeit bestimmt. Wie das gemacht wird, ist anschliessenden Erläuterungen zu entnehmen.

4. Die Geschwindigkeit  ${}^L\vec{v}_{GL}$  und die Rotationgeschwindigkeit  $\dot{\phi}_{GL}$  des Roboters können in acht Bewegungszuständen beschrieben werden. Die acht Zustände sind nachfolgend ab S. 18 genauer beschrieben.

#### Mechanische Kopplung

Die Geschwindigkeiten  ${}^L\vec{v}_{GR_i}$ , welche an den Rädern durch die Encoder-Daten berechnet werden, sind mechanisch miteinander gekoppelt. Dadurch kann mit zwei Radgeschwindigkeiten  ${}^L\vec{v}_{GR_i}$  die dritte berechnet werden. Aufgrund dieser Tatsache lässt sich ein Rutschen von einem Rad erkennen. Die Abbildung 7 zeigt die mechanische Kopplung der wirkenden Radgeschwindigkeiten. Es ist zu erkennen, dass in der Verbindung von zwei Rädern der gleiche Geschwindigkeitsvektor wirkt.

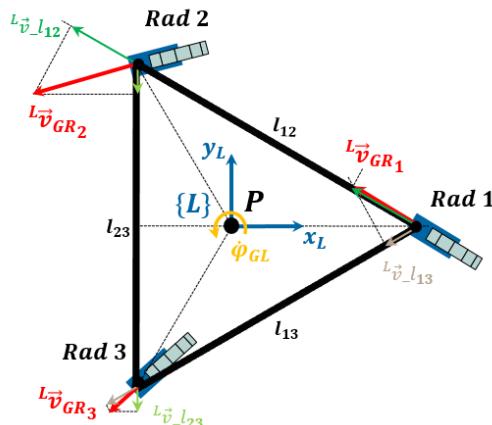


Abbildung 7: Mechanische Kopplung

- ${}^L\vec{v}_l_{AB}$ : Ist die auf die mechanische Verbindung  $l_{AB}$  projizierte Radgeschwindigkeit  ${}^L\vec{v}_{GR_i}$ . Dabei ist  $AB$  ein Index der für 12, 13 oder 23 stehen kann.
- $l_{AB}$ : Ist die mechanische Verbindung von zwei Rädern. Dadurch ist auch die mechanische Kopplung der Räder gegeben. Denn eine auf diese Verbindung projizierte Geschwindigkeit  ${}^L\vec{v}_l_{AB}$ , ist in der ganzen Verbindung  $l_{AB}$  gleich gross.

Die projizierte Geschwindigkeit  ${}^L\vec{v}_l_{AB}$  berechnet sich wie folgt:

$${}^L\vec{v}_l_{AB} = \frac{\vec{l}_{AB} \cdot {}^L\vec{v}_{GR_i}}{l_{AB}^2} \cdot \vec{l}_{AB} \quad (2-6)$$

#### Mathematisch korrekte, projizierte Geschwindigkeiten ${}^L\vec{v}_l_{AB}$

Da in einer mechanischen Verbindung  $l_{AB}$  die jeweils darauf projizierten Geschwindigkeiten  ${}^L\vec{v}_l_{AB}$  gleich gross sein müssen, können sie somit überprüft werden. Wenn die projizierten Geschwindigkeiten gleich oder annähernd gleich sind, werden sie gemittelt. Dadurch entsteht aus zwei Geschwindigkeiten eine mathematisch korrekte

Geschwindigkeit, mit welcher weiter gerechnet werden kann.

Wenn die beiden Geschwindigkeiten nicht annähernd gleich sind, kann davon ausgegangen werden, dass mindestens ein Rad gerutscht ist.

**Mathematisch korrekte Radgeschwindigkeit  ${}^L\vec{v}_{GR_i}$**

Mit den mathematisch korrekten, projizierten Geschwindigkeiten  ${}^L\vec{v}_l_{AB}$  werden die mathematisch korrekten Radgeschwindigkeiten  ${}^L\vec{v}_{GR_i}$  ermittelt.

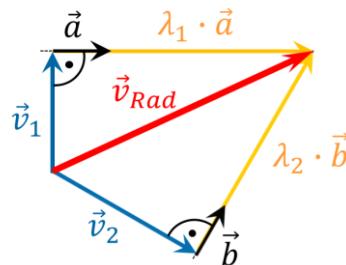


Abbildung 8: Vektorieller Zusammenhang

- $\vec{v}_i$ : Sind die mathematisch korrekten, projizierten Geschwindigkeiten  ${}^L\vec{v}_l_{AB}$ , die an einem Schwenkrad wirken.
- $\vec{v}_{Rad}$ : Ist die zu ermittelnde mathematisch korrekte Radgeschwindigkeit  ${}^L\vec{v}_{GR_i}$ .
- $\vec{a}, \vec{b}$ : Sind die Richtungsvektoren, die senkrecht zu den projizierten Geschwindigkeiten  $\vec{v}_i$  stehen.
- $\lambda_i$ : Sind die Faktoren für die Richtungsvektoren.

Mathematischer Zusammenhang:

$$\vec{v}_{Rad} = \vec{v}_1 + \lambda_1 \cdot \vec{a} = \vec{v}_2 + \lambda_2 \cdot \vec{b} \quad (2-7)$$

Die Richtungsvektoren  $\vec{a}$  und  $\vec{b}$  sind wie folgt definiert:

$$\vec{a} = \begin{bmatrix} v_{11} \\ v_{12} \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{und} \quad \vec{b} = -1 \cdot \begin{bmatrix} v_{21} \\ v_{22} \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (2-8)$$

Durch das Zusammenführen der Gleichungen ( 2-7 ) und ( 2-8 ) können die Faktoren für die Richtungsvektoren  $\lambda_i$  bestimmt werden:

$$\lambda_1 = \frac{b_1 \cdot (v_{12} - v_{22}) - b_2 \cdot (v_{11} - v_{21})}{a_1 \cdot b_2 - a_2 \cdot b_1} \quad (2-9)$$

$$\lambda_2 = \frac{a_1 \cdot (v_{12} - v_{22}) - a_2 \cdot (v_{11} - v_{21})}{a_1 \cdot b_2 - a_2 \cdot b_1} \quad (2-10)$$

Nun kann mittels einer der beiden Gleichungen von ( 2-7 ) die mathematisch korrekte Radgeschwindigkeit  $\vec{v}_{Rad}$  berechnet werden.

**Rotations-  
geschwindig-  
keit des Robo-  
ters**

Die Rotationsgeschwindigkeit  $\dot{\phi}_{GL}$  des Roboters kann durch einen Momentanpol bestimmt werden. Dazu wird ein Momentanpol in eine Lenkachse eines geeigneten Schwenkrades gesetzt.

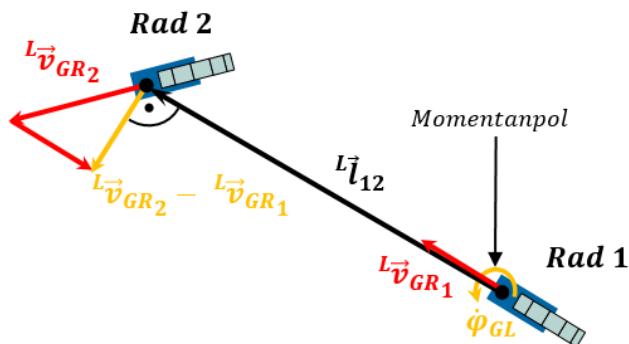


Abbildung 9: Momentanpolbestimmung des Roboters

Der Momentanpol ist, wie in Abbildung 9 ersichtlich, in die Lenkachse des Schwenkrades 1 gesetzt. Durch das Abziehen der Radgeschwindigkeit  $L\vec{v}_{GR_1}$  ist die Lenkachse des Schwenkrads 1 nicht mehr in Bewegung und kann so als Momentanpol genutzt werden.

Die Rotationsgeschwindigkeit  ${}^G\vec{\omega}_{GL} = \begin{bmatrix} 0 \\ 0 \\ \dot{\phi}_{GL} \end{bmatrix}$  kann mit dieser Annahme wie folgt berechnet werden:

$${}^G\vec{\omega}_{GL} = \frac{L\vec{l}_{12} \times (L\vec{v}_{GR_2} - L\vec{v}_{GR_1})}{|L\vec{l}_{12}|^2} \quad (2-11)$$

**Bewegungs-  
zustände**

Im Folgenden werden, wie auf S. 16 bei Punkt 4 erwähnt, die acht Bewegungszustände erläutert.

**Bewegungs-  
zustand 1**

Dieser Bewegungszustand beschreibt den Stillstand.

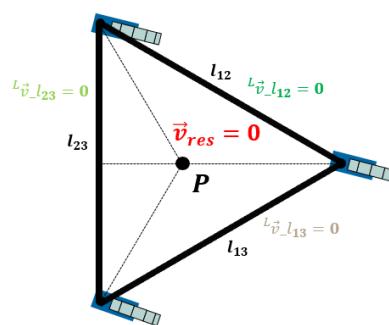


Abbildung 10: Stillstand

- $\vec{v}_{res}$ : Ist die resultierende Geschwindigkeit des Roboters im Punkt P. Es gilt folgender Zusammenhang:  $\vec{v}_{res} = {}^L\vec{v}_{GL}$ .

**Bewegungs-  
zustände  
2, 3 und 4**

Diese Bewegungszustände beschreiben die Bewegung orthogonal zu einer mechanischen Verbindung  $l_{AB}$ . Dadurch hat immer eine projizierte Geschwindigkeit den Wert null. Sobald dies der Fall ist, dürfen die Gleichungen ( 2-9 ) und ( 2-10 ) nicht benutzt werden, da sonst durch null dividiert wird. Die Abbildung 11 zeigt einen von drei solchen möglichen Fällen.

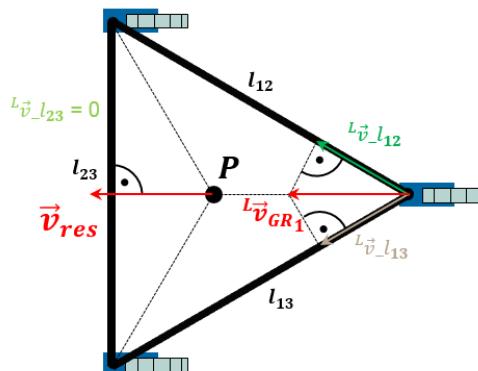


Abbildung 11: Orthogonale Bewegung zur mechanischen Verbindung  $l_{23}$ ,

In diesem Fall tritt keine Rotation des Roboters auf ( $\dot{\phi}_{GL} = 0$ ).

Die resultierende Geschwindigkeit  $\vec{v}_{res}$ , wie sie in der Abbildung 11 dargestellt wird, kann durch die projizierten Geschwindigkeiten  ${}^L\vec{v}_{l12}$  und  ${}^L\vec{v}_{l13}$  mit der Gleichung ( 2-7 ) berechnet werden ( $\vec{v}_{res} = {}^L\vec{v}_{GR_1}$ ).

**Bewegungs-  
zustände  
5, 6 und 7**

Diese Bewegungszustände beschreiben eine Drehung um die Lenkachse eines Schwenkrades. Die Abbildung 12 zeigt einen der drei möglichen Zustände.

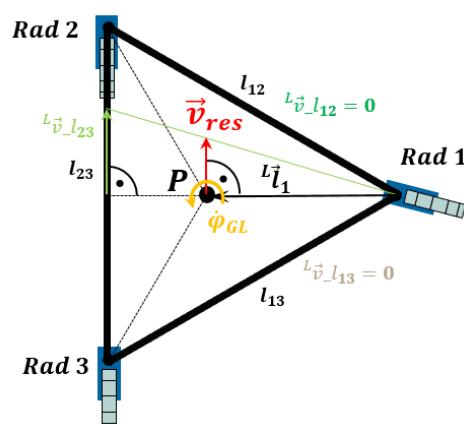


Abbildung 12: Drehung um die Lenkachse

Der in Abbildung 12 gezeigte Bewegungszustand beschreibt eine Drehung um die Lenkachse des ersten Schwenkrades. Somit haben zwei projizierte Geschwindigkeiten den Wert 0.

Die resultierende Geschwindigkeit  $\vec{v}_{res}$  im Punkt P wird wie folgt berechnet:

$$\vec{v}_{res} = \frac{2}{3} \cdot {}^L\vec{v}_{l23} \quad ( 2-12 )$$

Die Rotationsgeschwindigkeit  ${}^G\vec{\omega}_{GL} = \begin{bmatrix} 0 \\ 0 \\ \dot{\varphi}_{GL} \end{bmatrix}$  kann mit der Annahme, dass der Momentanpol in der Lenkachse des Schwenkrades 1 liegt, wie folgt berechnet werden:

$${}^G\vec{\omega}_{GL} = \frac{{}^L\vec{l}_1 \times \vec{v}_{res}}{|{}^L\vec{l}_1|^2} \quad (2-13)$$

**Bewegungs-  
zustand 8** Dieser ist der allgemeine Bewegungszustand, dargestellt beispielsweise in Abbildung 7, S. 16. Die resultierende Geschwindigkeit  $\vec{v}_{res}$  wird in diesem Fall durch das lineare Mittel der drei mathematisch korrekten Radgeschwindigkeiten  ${}^L\vec{v}_{GR_i}$  berechnet:

$${}^L\vec{v}_{GL} = \vec{v}_{res} = \frac{1}{3} \cdot \sum_{i=1}^3 {}^L\vec{v}_{GR_i} \quad (2-14)$$

Die Rotationsgeschwindigkeit  ${}^G\vec{\omega}_{GL}$  wird mit einem geeigneten Momentanpol berechnet (siehe Gleichung (2-11), S. 18). Ein geeigneter Momentanpol ist einer, der nicht wegen einem rutschenden Rad ausgeschieden ist.

**Transformation  
in das globale  
KS {G}** Um einen Istwert der Odometriedaten zu erhalten muss noch eine Transformation vom lokalen KS  $\{L\}$  in das globale KS  $\{G\}$  durchgeführt werden. Dazu ist wieder nur eine Rotationsmatrix  ${}^G R_L$  notwendig:

$$\begin{bmatrix} {}^G v_{GL,x} \\ {}^G v_{GL,y} \\ {}^G \dot{\varphi}_{GL} \end{bmatrix} = \begin{bmatrix} \cos(\varphi_{GL}) & -\sin(\varphi_{GL}) & 0 \\ \sin(\varphi_{GL}) & \cos(\varphi_{GL}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^L v_{GL,x} \\ {}^L v_{GL,y} \\ {}^L \dot{\varphi}_{GL} \end{bmatrix} \quad (2-15)$$

## 2.2.2 Berechnung der inversen Jacobimatrix

**Allgemein** Bei der Herleitung der inversen Jacobimatrix werden die in Abbildung 3, S. 12 eingeführten Parameter und Koordinatensysteme verwendet.

Die Herleitung der inversen Jacobimatrix drückt allgemein die Drehgeschwindigkeiten  $\omega_{Rad}$  und  $\omega_{Lenk}$  eines Schwenkrades mit der global auftretende Geschwindigkeit  ${}^G\vec{v}_{GL}$  und der Rotationsgeschwindigkeit  ${}^G\vec{\omega}_{GL}$  aus.

**Berechnung** Der auftretende Geschwindigkeitsvektor  ${}^G\vec{v}_{GR}$  des Roboters beschreibt die Geschwindigkeit vom Radkoordinatensystem  $\{R\}$  bezogen auf das globale KS  $\{G\}$ . Er kann im globalen KS  $\{G\}$  wie folgt ausgedrückt werden:

$${}^G\vec{v}_{GR} = {}^G\vec{v}_{GL} + {}^G\vec{\omega}_{GL} \times {}^G\vec{l}_{LR} \quad (2-16)$$

- ${}^G\vec{v}_{GL}$ : Ist der Geschwindigkeitsvektor, wie sich das lokale KS  $\{L\}$  bezogen auf

das globale KS  $\{G\}$  bewegt.

- ${}^G\vec{\omega}_{GL}$ : Diese Komponente beschreibt die Drehgeschwindigkeit vom lokalen KS  $\{L\}$  bezogen auf das globale KS  $\{G\}$ . Damit wird der Einfluss des dritten Freiheitsgrads des Roboters berücksichtigt. Die  $z$ -Komponente des Vektors  ${}^G\vec{\omega}_{GL}$  beschreibt die Rotationsgeschwindigkeit. Die  $x$  – und  $y$ - Komponenten haben immer den Wert null.

Der Geschwindigkeitsvektor  ${}^G\vec{v}_{GL}$  aus der Gleichung ( 2-16 ) kann mit der Drehmatrix von ( 2-4 ) im lokalen KS  $\{L\}$  ausgedrückt werden. Damit wird der Geschwindigkeitsvektor  ${}^G\vec{v}_{GR}$  ebenfalls im lokalen KS  $\{L\}$  beschrieben:

$${}^L\vec{v}_{GR} = {}^L R_G \cdot {}^G\vec{v}_{GL} + {}^L\vec{\omega}_{GL} \times {}^L\vec{l}_{LR} \quad (2-17)$$

- ${}^L R_G$ : Ist die Rotationsmatrix von der Gleichung ( 2-4 )
- ${}^L\vec{\omega}_{GL} = {}^G\vec{\omega}_{GL}$
- ${}^L\vec{l}_{LR} = {}^G\vec{l}_{LR}$

Wenn die Gleichung ( 2-17 ) im Radkoordinatensystem  $\{R\}$  ausgedrückt werden soll, muss der Geschwindigkeitsvektor  ${}^L\vec{v}_{GR}$  mit der Drehmatrix  ${}^R R_L$  multipliziert werden. Zusammen mit dem ausgerechneten Kreuzprodukt entsteht folgende Gleichung:

$${}^R\vec{v}_{GR} = {}^R R_L \cdot {}^L\vec{v}_{GR} = {}^R R_L \cdot \left( {}^L R_G \cdot {}^G\vec{v}_{GL} + \begin{bmatrix} -{}^L\omega_{GL,z} \cdot {}^Ll_{LR,y} \\ {}^L\omega_{GL,z} \cdot {}^Ll_{LR,x} \end{bmatrix} \right) \quad (2-18)$$

- ${}^R R_L$ : Diese Rotationsmatrix hat den selben Aufbau wie die Rotationsmatrix  ${}^L R_G$ , ausser dass um den Winkel  $\varphi_{LR}$  gedreht wird statt um den Winkel  $\varphi_{GL}$ .

Die Drehgeschwindigkeit  ${}^L\omega_{GL,z}$  kann ausgeklammert und die Rotationsmatrizen zusammengefasst werden:

$$\begin{bmatrix} {}^R v_{GR,x} \\ {}^R v_{GR,y} \end{bmatrix} = {}^R R_G \cdot \begin{bmatrix} {}^G v_{GL,x} \\ {}^G v_{GL,y} \end{bmatrix} + {}^L\omega_{GL,z} \cdot {}^R R_L \cdot \begin{bmatrix} -{}^L l_{LR,y} \\ {}^L l_{LR,x} \end{bmatrix} \quad (2-19)$$

Aus den Gleichungen ( 2-1 ) und ( 2-2 ) wird die inverse Jacobimatrix hergeleitet:

$$\begin{bmatrix} \omega_{Rad} \\ \omega_{Lenk} \\ {}^L\omega_{GL,z} \end{bmatrix} = J^{-1} \cdot \begin{bmatrix} {}^G v_{GL,x} \\ {}^G v_{GL,y} \\ {}^G\omega_{GL,z} \end{bmatrix} \quad (2-20)$$

$$J^{-1} = \begin{bmatrix} \frac{c(\varphi_{LR} + \varphi_{GL})}{R} & \frac{s(\varphi_{LR} + \varphi_{GL})}{R} & \frac{-{}^L l_{LR,y} \cdot c(\varphi_{LR}) + {}^L l_{LR,x} \cdot s(\varphi_{LR})}{R} \\ \frac{s(\varphi_{LR} + \varphi_{GL})}{d} & \frac{-c(\varphi_{LR} + \varphi_{GL})}{d} & \frac{-{}^L l_{LR,y} \cdot s(\varphi_{LR}) - {}^L l_{LR,x} \cdot c(\varphi_{LR})}{d} \\ 0 & 0 & 1 \end{bmatrix} \quad (2-21)$$

## 2.3 Fahrmöglichkeiten

Allgemein	<p>Das omnidirektionale Fahren ermöglicht eine neue Betrachtung der Bewegungsvorgaben, denn die drei Freiheitsgrade, welche die Bewegung eines mobilen Gefährts definieren, können unabhängig voneinander vorgegeben werden.</p> <p>Es ist möglich, die Bewegungsvorgabe für einen mobilen Roboter direkt im lokalen KS <math>\{L\}</math> zu beschreiben. Dabei ist die Bewegungsvorgabe trotzdem relativ zum globalen KS <math>\{G\}</math>. Die direkte Beschreibung im lokalen KS <math>\{L\}</math> bedeutet jedoch, dass keine Transformation vom globalen KS <math>\{G\}</math> in das lokale KS <math>\{L\}</math> stattgefunden hat (<i>siehe Gleichung (2-4), S. 13</i>). Dies hat zur Folge, dass in der Bewegung des Roboters ein Vorne und ein Hinten definiert ist.</p>
Im lokalen KS $\{L\}$ beschriebene Bewegung	<p>Wenn eine Geschwindigkeitsvorgabe im lokalen KS <math>\{L\}</math> beschrieben wird, bewegt sich der Roboter so in der Ebene, wie es durch die Vorgabe bestimmt wurde. Dabei hat er jedoch zu keinem Zeitpunkt einen Momentanpol. Wenn nun eine zusätzliche Drehung zu der Geschwindigkeitsvorgabe hinzu kommt, wird der vorgegebene Geschwindigkeitsvektor mit der Drehung überlagert. Dies hat zur Folge, dass nicht mehr genau nachvollzogen werden kann, welche Bewegungsvorgaben dem Roboter gemacht wurden.</p>
Fahren wie bei einem Auto	<p>Durch eine Bewegungsvorgabe im lokalen KS <math>\{L\}</math> kann sich ein Fahrverhalten wie bei einem Auto einstellen. Dazu ist es jedoch notwendig, dass die Geschwindigkeitsvorgabe auf eine Achse begrenzt wird, sodass entweder nur eine Vorgabe in x-Richtung oder eine in y-Richtung möglich ist.</p>
Omni- direktionales Fahren	<p>Beim omnidirektionalen Fahren (Bewegungsvorgabe im globalen KS <math>\{G\}</math>) gibt es weder Vorne noch Hinten. Der Roboter kann in der vorgegebenen Geschwindigkeit in der Ebene fahren und sich dabei gleichzeitig um seine eigene Achse drehen. Dadurch ist immer ersichtlich, welche Bewegung dem Roboter vorgegeben wurde.</p>
Drehung um eine Lenkachse	<p>Es ist möglich, eine Drehung um eine Lenkachse vorzugeben (<i>siehe Abbildung 12, S. 19</i>).</p> <p>In diesem Fall ist ebenfalls keine Transformation vom globalen KS <math>\{G\}</math> in das lokale KS <math>\{L\}</math> notwendig.</p> <p>Um die Drehung um eine Lenkachse zu erreichen, muss im lokalen KS <math>\{L\}</math> die resultierende Geschwindigkeit <math>{}^L\vec{v}_{res}</math> berechnet werden. Dies wird mittels der gewünschten Rotationsgeschwindigkeit <math>{}^L\vec{\omega}_{GL}</math> und dem jeweiligen Distanzvektor <math>{}^L\vec{l}_i</math> erreicht. Der Distanzvektor <math>{}^L\vec{l}_i</math> zeigt vom Radkoordinatensystems <math>\{R\}_i</math> zum Nullpunkt des lokalen KS <math>\{L\}</math>.</p>

$${}^L\vec{v}_{res} = {}^L\vec{\omega}_{GL} \times {}^L\vec{l}_i \quad (2-22)$$

Wenn nun die berechnete resultierende Geschwindigkeit  ${}^L\vec{v}_{res}$  mit der dazugehörigen Rotationsgeschwindigkeit  ${}^L\vec{\omega}_{GL}$  im lokalen KS  $\{L\}$  vorgegeben wird, erfolgt eine Drehung um die Lenkachse  $i$ .

## 2.4 Einfluss der Exzentrizität des Schwenkrades

**Allgemein** Die Besonderheit des OmniMoBots ist auf die Schwenkräfer zurückzuführen, welche ihm die Möglichkeit geben, aus dem Stillstand senkrecht zur letzten Fahrtrichtung zu beschleunigen. Diese Exzentrizität bringt jedoch auch neue, negative Faktoren, die berücksichtigt werden müssen. Dazu gehören starke Beschleunigungen beim Umschwenken der Schwenkräfer sowie eine verschlechterte Kippstabilität, je nach Position der Schwenkräfer.

**Geschwindigkeitsbegrenzung** In erster Linie geht es nicht darum, die Geschwindigkeit zu begrenzen, sondern um die Begrenzung der maximal auftretenden Beschleunigungen. Diese treten beim Umschwenken der Räder auf, wie es in Abbildung 13 dargestellt ist.

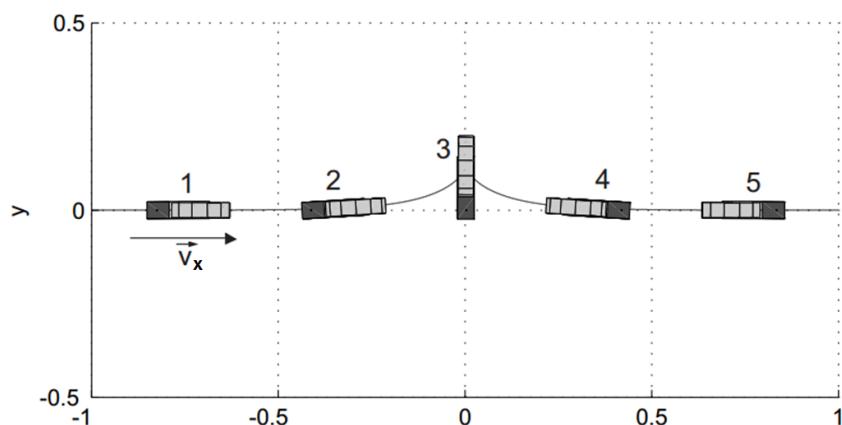


Abbildung 13: Umschwenken des Rades [5]

Die Abbildung 13 zeigt den Extremfall. Annahme: Der Roboter fährt mit seiner maximalen Geschwindigkeit in x-Richtung, das Rad schwenkt von seinem unnatürlichen Zustand zur Fahrtrichtung (Punkt 1) in seinen normalen Zustand in Fahrtrichtung (Punkt 5). Dabei hat das Rad im Punkt 1 eine negative Geschwindigkeit und im Punkt 5 eine positive. Das bedeutet, das Rad muss innerhalb des Zeitraumes des Umschwenkens von der maximalen negativen Geschwindigkeit in die maximale positive Geschwindigkeit beschleunigen. Dabei entstehen starke Beschleunigungen sowohl in der Lenkung wie auch im Antrieb.

**Max. Beschleunigung der Lenkung durch das Umschwenken**

Bei gleichbleibender Geschwindigkeit in x-Richtung  $\vec{v}_x$  (siehe Abbildung 13) treten die maximalen Beschleunigungen der Lenkung auf, wenn das Rad mit der Bewegungsrichtung einen Winkel von  $45^\circ$  einschließt. Daraus ergibt sich folgender Zusammenhang (siehe Herleitung [5], S. 18):

$$\dot{\omega}_{Lenk} = \frac{v_x^2}{2d^2} \quad (2-23)$$

Der Faktor  $d$  ist die Exzentrizität des Rades, also die Distanz von der Drehachse der Lenkung bis zum Auflagepunkt des Rades.

**Max. Beschleunigung des Antriebs durch das Umschwenken**

In dem vorliegenden Fall  $R < 2 \cdot d$  ist nicht wie bei [2] S. 407 erwähnt, die max. auftretende Beschleunigung der Lenkung die kritische Beschleunigung, sondern die auftretende max. Beschleunigung des Antriebs.  
(siehe Herleitung [5], S. 19):

$$\dot{\omega}_{Rad} = \frac{v_x^2}{d \cdot R} \quad (2-24)$$

**Bemerkung**

Im Fall  $R < 2 \cdot d$  ist die kritische Beschleunigung im Antrieb. Wenn  $R > 2 \cdot d$  ist die kritische Beschleunigung in der Lenkung, wie in [2] S. 407 erwähnt.

Da die Geschwindigkeit  $v_x$  bei beiden Gleichungen im Zähler im Quadrat auftaucht, ist eine Begrenzung der Geschwindigkeit ein vernünftiger Ansatz, um die auftretenden Beschleunigungen in den Griff zu bekommen. Auch eine Vergrösserung der Exzentrizität oder des Rades hätte einen positiven Einfluss auf die auftretenden Beschleunigungen. Diese sind jedoch bereits mit den folgenden Werten definiert:

$$R = 0.05 \text{ [m]} \quad \text{und} \quad d = 0.096505 \text{ [m]} \quad (2-25)$$

**Kippstabilität**

Die Kippstabilität des OmniMoBots ist nicht optimal, da die Schwenkräder eine Exzentrizität aufweisen. Wenn jedoch die Schwenkräder gar keine Exzentrizität aufweisen würden, wäre die Bewegung aus dem Stand in eine beliebige Richtung nicht mehr möglich. Deshalb ist in einer vorhergehenden Masterthesis [6] ein Optimum für die Exzentrizität bestimmt worden.

Die Exzentrizität der Schwenkräder  $d$  wie auch die Beschleunigung  $a$ , hat einen direkten Einfluss auf die Kippstabilität des Roboters.

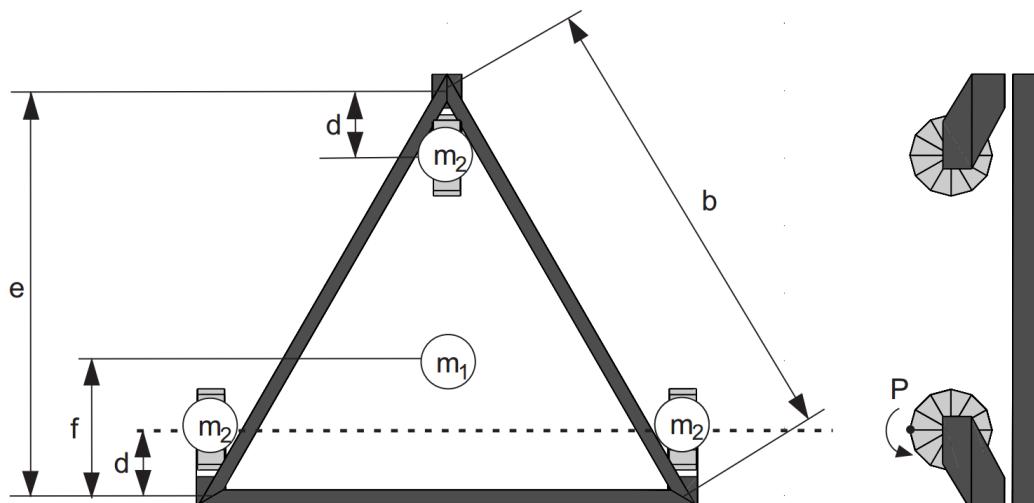


Abbildung 14: Berechnungsmodell Kippstabilität [5]

Die Abbildung 14 zeigt den schlimmsten Fall der Instabilität, bei der sich das vordere Rad kurz vor dem Abheben befindet. Dabei beschleunigt der Roboter nach oben. Durch eine Gleichgewichtsbetrachtung um den Punkt  $P$ , kann eine Gleichung hergeleitet werden, mit welcher die Einflüsse der Exzentrizität und anderer Faktoren ersichtlich werden (siehe Herleitung in [5], S. 16):

$$d = \frac{\sqrt{\frac{1}{12} \cdot b \cdot g + \frac{3}{4} \cdot b \cdot k \cdot g - (1 + 3 \cdot k) \cdot h_{sp} \cdot a}}{g + 2 \cdot k \cdot g} \quad (2-26)$$

- $k$ : Ist das Verhältnis der Masse der Radaufhängung  $m_2$  zu der Masse der Plattform  $m_1$ .
- $h_{sp}$ : Ist die zusammengefasste Höhe der Massenpunkte.

**Bemerkung** Die Gleichung ( 2-26 ) zeigt, dass bei grösseren Beschleunigungen  $a$  eine kleinere Exzentrizität  $d$  erwünscht ist.  
Die Exzentrizität wurde verwendet, um den Einfluss des Massenverhältnisses  $k$  zu bestimmen. Dabei wurde Folgendes festgestellt: Wenn das Verhältnis  $k$  von  $m_2$  zu  $m_1$  den Wert 10 erreicht, darf die Exzentrizität  $d$  einen Wert von ca. 0.15 Meter haben. Bei einem Verhältnis  $k$  von 0.1 darf die Exzentrizität  $d$  noch einen Wert von ca. 0.12 Meter haben. Folglich ist der Roboter kippstabiler, je grösser die Masse der Radaufhängung  $m_2$  im Verhältnis zur Masse der Plattform  $m_1$  ist. Weiter ist ersichtlich, dass ein grösserer Roboter auch eine grössere Exzentrizität haben darf.

## 2.5 Regelungskonzepte der Antriebe

**Allgemein** Mit der hergeleiteten Kinematik des OmniMoBots können nun Regelungskonzepte aufgeführt werden. Als erstes werden die allgemeinen Regelungskonzepte vorgestellt, anschliessend die spezifischen Regelungskonzepte für den OmniMoBot.

### 2.5.1 Allgemeine Regelungskonzepte in der Robotik für die Motorachsen

**Allgemein** In der Robotik wird oft ein Regelungsprinzip mit einer PD-Regelung verwendet. Die PD-Regelung liegt einem mechanischen System zugrunde. Das mechanische System besteht aus einem parallelen Feder-Dämpfer-Element und einer Masse in Serie (siehe [9]).

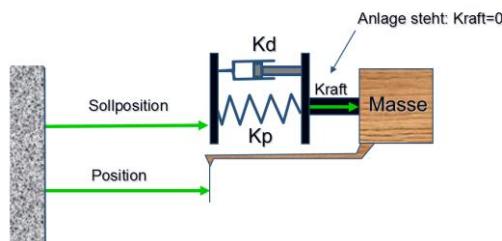


Abbildung 15: Mechanisches System [9]

**Herleitung der PD-Regelung [9]** Die Kraft, welche aufgewendet werden muss, um die Masse zu beschleunigen, ist wie folgt definiert:

$$F = \ddot{x} \cdot m \quad (2-27)$$

Die aus einer Sollpositionsänderung entstehende Kraft wird durch den dämpfenden und den federnden Teil beschrieben. Die Beschleunigung der Masse wirkt dabei der

Sollpositionsänderungs-Kraft  $F$  entgegen:

$$F = \ddot{x} \cdot m = -K_d \cdot \dot{x} - K_p \cdot x \quad (2-28)$$

Dies kann in die folgende Gleichung umgewandelt werden:

$$\ddot{x} + \frac{K_d \cdot \dot{x}}{m} + \frac{K_p \cdot x}{m} = 0 \quad (2-29)$$

Die ungedämpfte Eigenkreisfrequenz  $\omega_0$ , die Ferderkonstante  $c$ , die Masse  $m$ , die Dämpfungskonstante  $d$  und der Dämpfungsgrad  $D$  haben folgenden Zusammenhang:

$$\omega_0 = \sqrt{\frac{c}{m}} \quad \text{und} \quad D = \frac{d}{2m\omega_0} \quad (2-30)$$

Wenn  $K_d = d$  und  $K_p = c$  ist, kann die Gleichung ( 2-29 ) auch als Bewegungsgleichung für einen freien, gedämpften Schwinger verwendet werden [17]:

$$\ddot{x} + 2D\omega_0\dot{x} + \omega_0^2x = 0 \quad (2-31)$$

Daraus ergibt sich, dass  $K_p = m\omega_0^2$  und  $K_d = m2D\omega_0$  ist.

In der Physik existieren nur die Istwerte, der Sollwert ist stets gleich null. Daher kann die Gleichung ( 2-31 ) folgendermassen umgeformt werden:

$$\ddot{x} - 2D\omega_0(0 - \dot{x}_{ist}) - \omega_0^2(0 - x_{ist}) = 0 \quad (2-32)$$

Diese Betrachtungsweise hilft bei der Umsetzung in die Regelungstechnik. In dieser wird die Gleichung ( 2-32 ) wie folgt beschrieben:

$$\ddot{x} - 2D\omega_0(\dot{x}_{soll} - \dot{x}_{ist}) - \omega_0^2(x_{soll} - x_{ist}) = 0 \quad (2-33)$$

Die Differenz der  $x_{soll}$ - und  $x_{ist}$ -Werte wird als Error  $e$  bezeichnet. Aus dem Zusammenhang der Gleichungen ( 2-28 ) und ( 2-33 ) ergibt sich Folgendes:

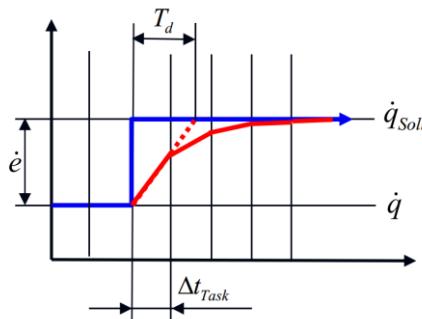
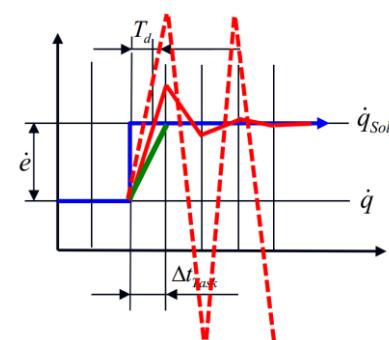
$$\ddot{e} = 2D\omega_0\dot{e} + \omega_0^2e = \frac{F}{m} \quad (2-34)$$

Aus der Gleichung ( 2-34 ) können die folgenden Regelungsparameter der PD-Regelung abgeleitet werden:

$$k_p = \omega_0^2 \quad \text{und} \quad k_d = 2D\omega_0 \quad (2-35)$$

Bei dieser Art der Regelung wird davon ausgegangen, dass der Geschwindigkeitsfehler der begrenzende Faktor für das Regelungssystem ist. Dabei wird die Steifigkeitsgrenze der Geschwindigkeitsrückführung erreicht, wenn der Geschwindigkeitsfehler in einem einzigen Regelungstakt  $\Delta t_{task}$  ausgeglichen wird (siehe Abbildung 17 grüne Linie). Damit die Regelung nicht instabil wird, muss die folgende Voraussetzung erfüllt sein (siehe [9]):

$$k_d = \frac{1}{T_d} < \frac{1}{\Delta t_{Task}} = f_{Task} \quad (2-36)$$

Abbildung 16: Stabile Regelung:  $T_d > \Delta t_{Task}$  [9]Abbildung 17: Instabile Regelung:  $T_d < \Delta t_{Task}$  [9]

Die ungedämpfte Eigenkreisfrequenz des geschlossenen Regelkreises  $\omega_0$  kann wie folgt definiert werden, unter der Annahme, dass die Regelungstakt-Rate  $f_{Task} = 1 \text{ kHz}$  entspricht:

$$\omega_0 = 2 \cdot \pi \cdot f_0 \approx 357 \left[ \frac{\text{rad}}{\text{s}} \right] \quad \text{und} \quad f_0 = \frac{1}{2 \cdot D \cdot T \cdot s \cdot (2 \cdot \pi)} \approx 56.8 \text{ [Hz]} \quad (2-37)$$

$2 = \text{Nyquist - Faktor}$

$D = \text{optimaler Dämpfungswert} = 0.7$

$T = \Delta t_{Task} = \text{Abtastzeit der Regelung} = 0.001 \text{ s}$

$s = \text{Sicherheitsfaktor} = 2$

$f_0 = \text{Eigenfrequenz in Hz}$

### 2.5.1.1 Positionsregelung im Gelenkkoordinatensystem

#### Blockbild

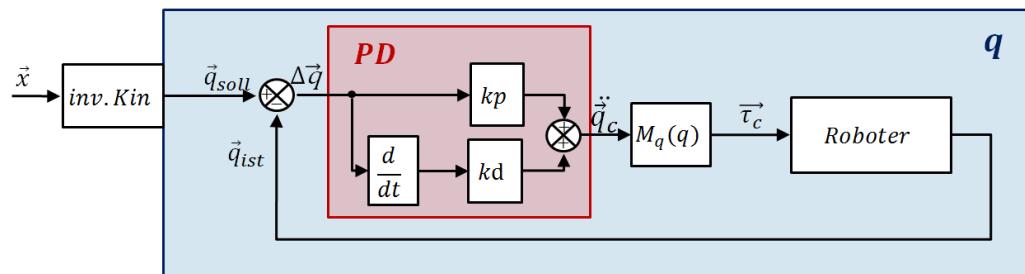


Abbildung 18: Positionsregelung

**Beschreibung** Die Position wird im kartesischen KS vorgegeben. Durch die inverse Kinematik findet eine Umwandlung in die Gelenkstellung  $q$  statt. Die Ist-Gelenkstellung kommt direkt aus den Encoder-Daten. Die Beschleunigung des Fehlers  $\ddot{q}_c$  wird anschliessend mit der Massenmatrix  $M_q$  multipliziert, daraus entsteht dann das geregelte Antriebsmoment  $\vec{\tau}_c$ . Die Verstärkungen  $kp$  und  $kd$  der PD-Regelung haben die zuvor bestimmten Werte. Dies ist das einfachste Regelungskonzept und hat den Vorteil, dass auftretende Störungen direkt im Gelenkkoordinatensystem ausgeregelt werden.

**Massenmatrix** Die Motoren werden auf ein Moment geregelt, aus dem Moment wird der Stromanteil ermittelt und mit diesem kann die benötigte Spannung („Duty Cycle“ des PWMs) für die Motoren berechnet werden.

Die Massenmatrix ist für die Berechnung der Momente notwendig, dabei werden alle vorhandenen Massen in dem zu regelnden System berücksichtigt. Konkret sind es nicht direkt die Massen, sondern die Massenträgheitsmomente, reduziert auf den Getriebeausgang der Motoren. Dabei enthält die Diagonale der Massenmatrix die höchsten Werte, wenn von Getriebemotoren mit einer hohen Übersetzung ausgegangen wird. Das Trägheitsmoment des Motors ist dabei in der Regel dominant, weil das Rotorträgheitsmoment des Motors mit dem Quadrat der Getriebeübersetzung multipliziert wird. Als Richtwert für eine noch akzeptable Regelung kann ein maximal um den Faktor vier höheres Lastträgheitsmoment im Vergleich zum Motorenträgheitsmoment angenommen werden. Mehr dazu ist in Kapitel 2.6 zu finden.

### 2.5.1.2 Kaskadierte Positions- und Geschwindigkeitsregelung

#### Blockbild

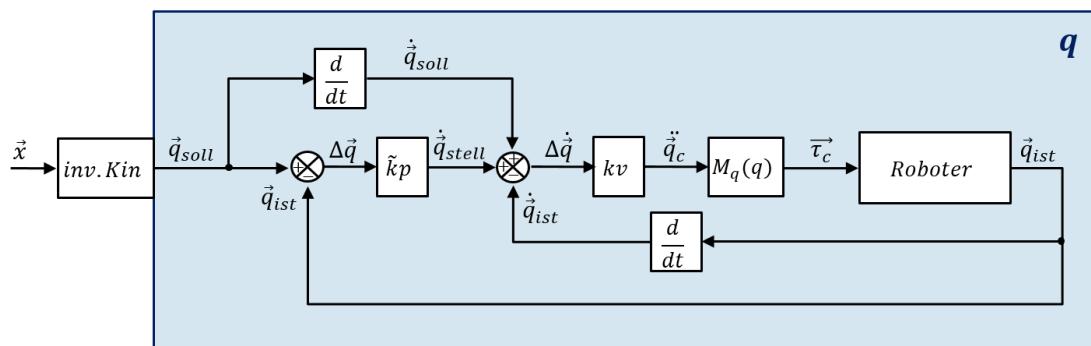


Abbildung 19: Kaskadierte Regelung

**Beschreibung** Dieses Regelungskonzept ist die kaskadierte Form der Positionsregelung. Dabei kann ebenfalls die Position vorgegeben werden. Zusätzlich wird jedoch die Stellgrösse mit dem diffenzierten Anteil der Position beaufschlagt. Dieser zusätzliche Anteil wird auch Vorsteuerung genannt.

Die Vorsteuerung hat den Vorteil, dass die Stellgrösse mit einem Wert beaufschlagt wird, der unabhängig von den Zuständen der Regelungsstrecke und den daraus resultierenden Messungen ist. Die Vorsteuerung verbessert das Führungsverhalten, indem der aufgrund des Sollwertverlaufs erwartete Stellgrößen-Bedarf berücksichtigt wird.

Mit diesem Regelungskonzept ist es somit auch möglich, statt des Positionsverlaufs  $\vec{q}_{soll}$  den Geschwindigkeitswert  $\dot{\vec{q}}_{soll}$  vorzugeben.

Im Fall eines mobilen Roboters kann die Position durch die Integration der Geschwindigkeitsvorgabe ermittelt werden (siehe Abbildung 22).

#### Regelungsparameter

Aus der Abbildung 19 kann folgende Gleichung hergeleitet werden:

$$\ddot{e} = kv \cdot \dot{e} + kv \cdot \tilde{kp} \cdot e \quad (2-38)$$

Dabei entspricht der Regelungsfehler  $\ddot{e}$  der geregelten Beschleunigung der Gelenkkachsen  $\ddot{q}_c$ . Der Vergleich mit der Gleichung (2-34) ergibt die neuen Regelungsparameter:

$$\tilde{k}p = \frac{\omega_0}{2D} \quad \text{und} \quad kv = 2D\omega_0 \quad (2-39)$$

## 2.5.2 Antriebs-Regelungskonzept für den OmniMoBot

**Allgemein** Da bei mobilen Robotern eine Geschwindigkeit und nicht eine Position vorgegeben wird, kann der in Kapitel 2.5.1.1 erwähnte Positionsregler nicht mit den hergeleiteten Regelungsparametern verwendet werden. Hingegen können aus der kaskadierten Regelung aus Kapitel 2.5.1.2 gleich drei verschiedene Regelungskonzepte mittels der hergeleiteten Regelungsparameter erstellt werden.

### 2.5.2.1 Regelung im kartesischen Koordinatensystem

**Allgemein** Bei dem folgenden Regelungskonzept wird berücksichtigt, dass ein mobiler Roboter geregelt werden soll. Aus diesem Grund wird eine Geschwindigkeitsvorgabe als Sollwert vorgegeben statt ein Positionsverlauf.

#### Blockbild

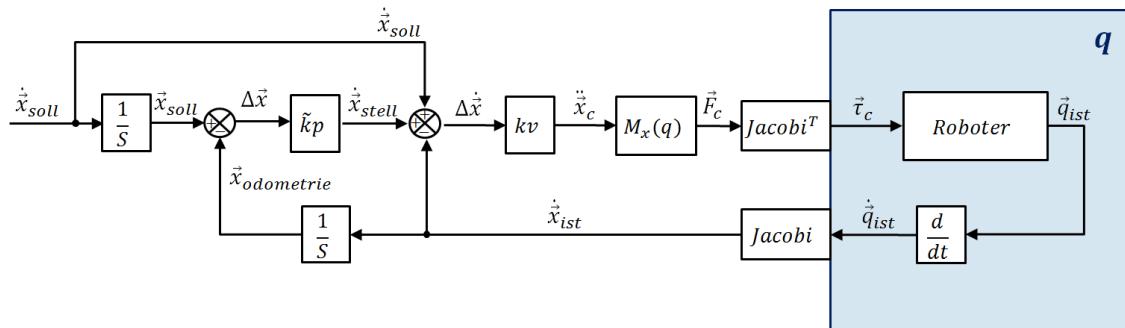


Abbildung 20: Regelungskonzept für mobile Roboter im kartesischen KS

**Beschreibung** Die Geschwindigkeit wird im kartesischen KS vorgegeben. Durch die Rückführung über die Jacobimatrix kann sie kaskadiert geregelt werden.

Die daraus erhaltene Beschleunigung des Fehlers  $\ddot{x}_c$  wird mit der Massenmatrix, welche im kartesischen KS definiert ist, verrechnet. Dabei entstehen die wirkenden Kräfte  $\vec{F}_c$  im kartesischen KS. Dies hat grosse Vorteile, weil auf diese Weise direkt die wirkenden Kräfte beeinflusst werden können. Durch das anschliessende Verrechnen der Kräfte  $\vec{F}_c$  mit der transponierten Jacobimatrix ergibt sich das geregelte Antriebsmoment  $\vec{\tau}_c$ .

Leider ist für dieses Regelungskonzept eine Jacobimatrix erforderlich und nicht, wie in Kapitel 2.2.1, S. 14 hergeleitet, ein Jacobi-Algorithmus.

**Kartesische Massenmatrix** Für die Berechnung der kartesischen Massenmatrix  $M_x$  wird ebenfalls eine Jacobimatrix  $J$  benötigt. Zusammen mit der Massenmatrix  $M_q$  wird sie wie folgt berechnet (siehe Beschreibung [55], Robot Control Part 6):

$$M_x = (J \cdot M_q^{-1} \cdot J^T)^{-1} \quad (2-40)$$

## 2.5.2.2 Kaskadierte Regelung im kartesischen und im Gelenkkoordinatensystem

### Allgemein

Die Verwendung der direkten Jacobimatrix ermöglicht eine Regelung im kartesischen Raum mit der Position aus der Odometrie sowie gleichzeitig eine Regelung im Gelenkkoordinatensystem mit den Motorgeschwindigkeiten.

### Blockbild

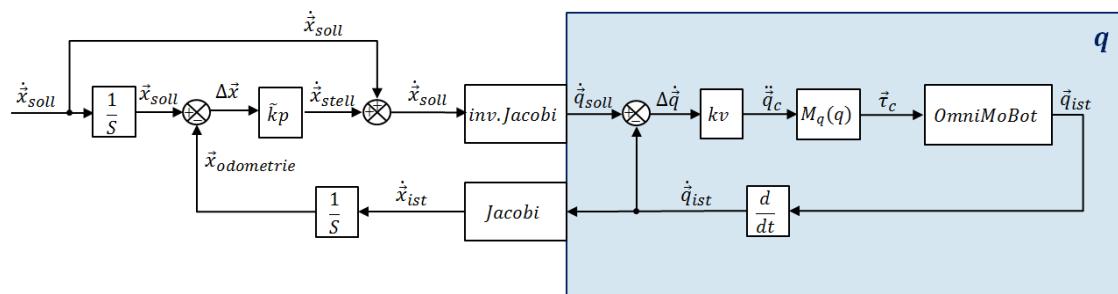


Abbildung 21: Regelkonzept für OmniMoBot im kartesischen KS und im Gelenkkoordinatensystem

### Erläuterung

Vorgegeben wird die Geschwindigkeit im kartesischen KS des Roboters. Durch die Verwendung der direkten Jacobimatrix kann die Position im kartesischen Raum geregelt werden. Die Sollposition  $\vec{x}_{soll}$  wird dabei mit der integrierten, geschätzten Geschwindigkeit  $\dot{\vec{x}}_{odometrie}$  aus der Jacobimatrix verglichen. Die Soll-Geschwindigkeit  $\dot{\vec{x}}_{soll}$ , welche aus der Summe von Stell und Soll entsteht, wird mit der inversen Jacobimatrix in die Motorstellungsgeschwindigkeiten umgerechnet. Das ermöglicht die kaskadierte Regelung der Geschwindigkeit im Gelenkkoordinatensystem  $q$ .

### Bemerkung

Ein Vorteil der Regelung im kartesischen Raum ist, dass mögliche Regelungsabweichungen der Motorachsen kompensiert werden können. Diese werden zwar im Gelenkkoordinatensystem  $q$  ausgeregelt, können sich aber nichtlinear auf die kartesische Position auswirken, was zu Positionsfehlern führt.

## 2.5.2.3 Kaskadierte Regelung im Gelenkkoordinatensystem

### Allgemein

Dieses Regelungskonzept ist einfach gehalten, da keine Jacobimatrix verwendet wird. Zudem regelt es Störungen und mögliche Fehler wegen Ungenauigkeiten direkt im Gelenkkoordinatensystem aus. Daher wird für die Inbetriebnahme des OmniMoBots dieses Regelungskonzept umgesetzt.

### Blockbild

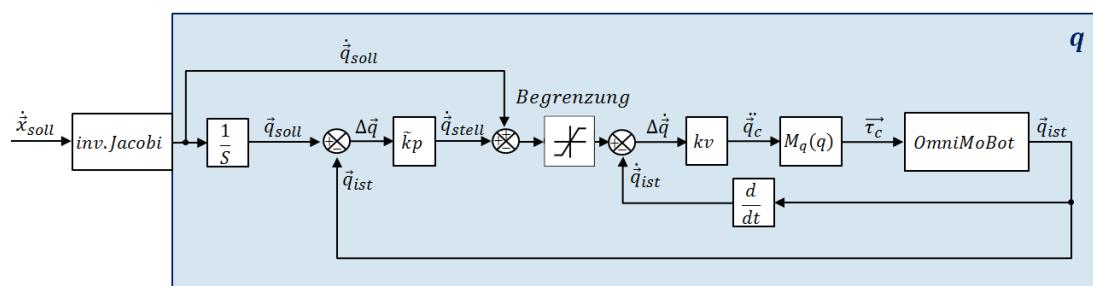


Abbildung 22: Regelungskonzept für OmniMoBot im Gelenkkoordinatensystem

**Erläuterung** Wieder wird die Geschwindigkeit im kartesischen Raum vorgegeben. Diese kann anschliessend durch die inverse Jacobimatrix direkt in die Motorstellungsgeschwindigkeit umgewandelt  $\dot{q}_{soll}$  werden. Danach wird sie integriert und kaskadiert geregelt.

**Bemerkung** Auch bei diesem Regelungskonzept ist es möglich, die Positionsregelung im kartesischen Raum mittels eines Beschleunigungs-Sensors und/oder eines Laserscanners zu verbessern. Für die Inbetriebnahme werden jedoch noch keine zusätzlichen Sensoren verwendet.

Weiter soll, wie in Kapitel 2.3, S. 22 erwähnt, die Geschwindigkeit begrenzt werden. Eine dynamische Begrenzung wäre optimal (siehe [5], S. 25). Für die Inbetriebnahme wird jedoch eine konstante Geschwindigkeitsbegrenzung von 1 m/sec im globalen KS {G} verwendet.

Für die Begrenzung aus Abbildung 22 gilt die Leerlaufdrehzahl der Motoren ( $7500 \text{ min}^{-1}$ ). Durch die jeweiligen Getriebeübersetzungen beträgt die Begrenzung der Antriebe ca.  $41.8 \text{ rad/sec}$  und die Begrenzung der Lenkungen beträgt ca.  $20.5 \text{ rad/sec}$ .

## 2.6 Die wirkenden Massenträgheitsmomente des OmniMobots

**Allgemein** In diesem Kapitel wird eine Handrechnung gemacht, um die dynamische Regelbarkeit zu prüfen. Dabei wird eine Vereinfachung des Roboterträgheitsmoments vorgenommen und mit dem Motorträgheitsmoment verglichen. Das Motorträgheitsmoment muss im Vergleich zum Roboterträgheitsmoment möglichst gross sein, denn in diesem Fall kann mit einer vereinfachten Massenmatrix gerechnet werden. Wird die Geschwindigkeit des OmniMoBots auf 1 m/s beschränkt, ist keine genauere Berechnung der Dynamik notwendig. Bei höheren Geschwindigkeiten steigt der Einfluss der Dynamik, deswegen ist es empfehlenswert, bei höheren Geschwindigkeiten den Einfluss der Dynamik miteinzubeziehen. Dazu kann die Publikation [7] beigezogen werden. Sie beschreibt die Berechnung der Dynamik eines mobilen Roboters, der ebenfalls von drei Schwenkrä dern angetrieben wird. Zusätzlich werden der Einfluss der Räder und mögliche Singularitäten berücksichtigt.

Die Trägheitsmomente werden jeweils auf ein Schwenkrad bezogen, welches aus einem antreibenden und einem lenkenden Teil besteht. Es wird davon ausgegangen, dass alle drei Schwenkrä der identisch sind. Das Trägheitsmoment der einzelnen Komponenten wird mittels CAD-Daten ermittelt.

### 2.6.1 Antrieb

**Allgemein** Der Antrieb hat drei Getriebe-Stufen. Dabei ist  $i_1 = 2$ ,  $i_2 = 4$  und  $i_3 = 2.25$ . Somit entsteht eine totale Untersetzung  $i_{tot}$  von 18.

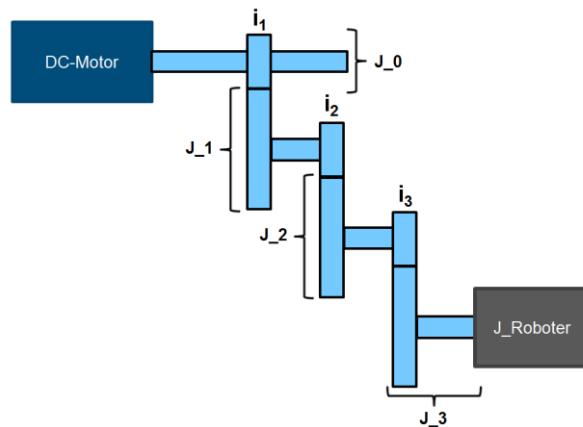
**Vereinfachte  
Darstellung**

Abbildung 23: Vereinfachte Darstellung des Antriebs

**Werte Tabelle**

Trägheitsmoment	Kürzel	Wert [kg*m²]
Ritzel mit Welle	$J_0$	$4.2675 \cdot 10^{-6}$
Übergang erster zur zweiter Stufe	$J_{-1}$	$30.643 \cdot 10^{-6}$
Übergang zweiter zur dritter Stufe	$J_{-2}$	$48.1519 \cdot 10^{-6}$
Abtrieb der dritten Stufe	$J_{-3}$	$3.247195 \cdot 10^{-3}$
Roboter	$J_{\text{Robot}}$	$2.225 \cdot 10^{-2}$

Tabelle 1: Parameterwerte des Antriebs

- $J_0$  bis  $J_3$ : Die Werte sind mit dem CAD berechnet.
- $J_{\text{Robot}}$ : Der Wert berechnet sich aus der Masse des Roboters und dem Radradius, wobei ein Drittel der Masse des Roboters 8.9 Kilogramm entspricht [6]:

$$J_{\text{Robot}} = \frac{1}{3} m_{\text{Robot}} \cdot R_{\text{Rad}}^2 = 8.9 \text{ [kg]} \cdot 0.05^2 \text{ [m}^2\text{]} \quad (2-41)$$

**Berechnung** Das Trägheitsmoment der Getriebe  $J_{\text{getr\_mot}}$  auf die Motorwelle reduziert:

$$J_{\text{getr\_mot}} = \frac{J_{-3}}{i_{\text{tot}}^2} + \frac{J_{-2}}{(i_1 \cdot i_2)^2} + \frac{J_{-1}}{i_1^2} = 1.8435 \cdot 10^{-5} \text{ [kg} \cdot \text{m}^2\text{]} \quad (2-42)$$

Das Trägheitsmoment des Roboters  $J_{\text{Robot\_mot}}$  auf die Motorwelle reduziert:

$$J_{\text{Robot\_mot}} = \frac{J_{\text{Robot}}}{i_{\text{tot}}^2} = 6.8673 \cdot 10^{-5} \text{ [kg} \cdot \text{m}^2\text{]} \quad (2-43)$$

Das gesamte Trägheitsmoment  $J_{\text{tot}}$  auf die Motorwelle reduziert:

$$J_{\text{tot}} = J_{\text{enc}} + J_{\text{mot}} + J_0 + J_{\text{getr\_mot}} + J_{\text{Robot\_mot}} = 10.58 \cdot 10^{-5} \text{ [kg} \cdot \text{m}^2\text{]} \quad (2-44)$$

Das gesamte Trägheitsmoment  $J_{\text{tot\_getr}}$  auf den Getriebeausgang reduziert:

$$J_{\text{tot\_getr}} = J_{\text{tot}} \cdot i_{\text{tot}}^2 = 34.279 \cdot 10^{-3} \text{ [kg} \cdot \text{m}^2\text{]} \quad (2-45)$$

**Überblick**

Trägheitsmoment	Kürzel	Auf Getriebeausgang reduziert [ $\text{kg}^*\text{m}^2$ ]	Auf Motorenausgang reduziert [ $\text{kg}^*\text{m}^2$ ]
Encoder	$J_{enc}$	$0.05508 \cdot 10^{-3}$	$0.017 \cdot 10^{-5}$
Motorwelle	$J_{mot}$	$4.6008 \cdot 10^{-3}$	$1.42 \cdot 10^{-5}$
Ritzel mit Welle	$J_0$	$1.3823 \cdot 10^{-3}$	$0.42675 \cdot 10^{-5}$
Übergang erste zu zweiter Stufe	$J_1$	$2.482 \cdot 10^{-3}$	$0.7661 \cdot 10^{-5}$
Übergang zweite zu dritter Stufe	$J_2$	$0.244 \cdot 10^{-3}$	$0.075237 \cdot 10^{-5}$
Abtrieb der dritten Stufe	$J_3$	$3.2472 \cdot 10^{-3}$	$1.0022 \cdot 10^{-5}$
Summe der konstanten Trägheitsmomente ( $J_{enc}$ bis $J_3$ )	$J_{konst,antr}$	$11.962 \cdot 10^{-3}$	$3.692 \cdot 10^{-5}$
Summe des dynamischen Trägheitsmoments	$J_{Roboter}$	$22.25 \cdot 10^{-3}$	$6.8673 \cdot 10^{-5}$
Gesamtes Trägheitsmoment	$J_{gesamt,antr}$	$34.279 \cdot 10^{-3}$	$10.58 \cdot 10^{-5}$

Tabelle 2: Trägheitsmomente des Antriebs im Überblick

**Bemerkung**

Da die totale Untersetzung  $i_{tot}$  relativ klein ist, hat das Trägheitsmoment des Roboters  $J_{Roboter\_mot}$  den grössten Einfluss auf das System. Dies hat zur Folge, dass bei höheren Geschwindigkeiten die Regelbarkeit mittels der vereinfachtem Massenmatrix abnimmt. Wie sich in Kapitel 4.1, S. 53 zeigt, reicht die vereinfachte Massenmatrix für das Balancieren eines Stabes bei einer max. Geschwindigkeit von 1 m/s aus.

Eine Beschleunigung des Roboters von  $a = 5 \text{ m/s}^2$  hat die folgende Winkelbeschleunigung  $\ddot{\varphi}_{getr}$ :

$$\ddot{\varphi}_{getr} = \frac{a}{R_{Rad}} = 100 [\text{rad/s}^2] \quad (2-46)$$

Die Winkelbeschleunigung  $\ddot{\varphi}_{getr}$  wird auf die Motorwelle reduziert:

$$\ddot{\varphi}_{mot} = \ddot{\varphi}_{getr} \cdot i_{tot} = 1800 [\text{rad/s}^2] \quad (2-47)$$

Das Motorenmoment  $M_{mot}$  wird wie folgt berechnet:

$$M_{mot} = \ddot{\varphi}_{mot} \cdot J_{tot} = 0.1904 [\text{Nm}] \quad (2-48)$$

Der Motor hat ein Nennmoment  $M_n$  von  $0.177 \text{ Nm}$ . Er ist demnach dieser Aufgabe knapp gewachsen.

**Max. Geschwindigkeit für das Umschwenken**

Mit der Gleichung ( 2-24 ), S. 24 kann die maximal zulässige Geschwindigkeit für das Umschwenken ermittelt werden:

$$\sqrt{\frac{3 \cdot M_n}{J_{konst,antr} \cdot i_{tot}}} \cdot d \cdot R = v_{max,Antrieb} = 1.96 [\text{m/s}] \quad (2-49)$$

Das Umschwenken muss somit erfolgen, bevor eine Geschwindigkeit von 1.96 m/s erreicht wird.

## 2.6.2 Lenkung

### Allgemein

Die Lenkung besitzt zwei Getriebe, ein Planetengetriebe mit 2 Stufen und ein Ritzel mit Tellerrad. Das Planetengetriebe hat eine Untersetzung von  $i_1 = 12.25$  und das Kegelrad eine von  $i_2 = 3$ . Somit entsteht eine totale Untersetzung  $i_{tot}$  von 36.75.

### Vereinfachte Darstellung

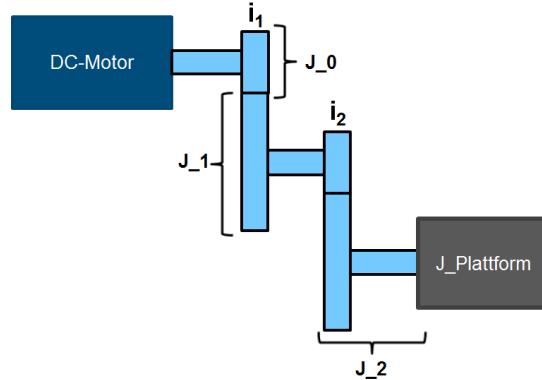


Abbildung 24: Vereinfachte Darstellung der Lenkung

### Werte Tabelle

Trägheitsmoment	Kürzel	Wert [ $\text{kg} \cdot \text{m}^2$ ]
Planetengetriebe auf Eingang	$J_0$	$2.4 \cdot 10^{-6}$
Übergang Planetengetriebe zum Ritzel	$J_1$	$5 \cdot 10^{-6}$
Schwenkrad	$J_2$	$6.863 \cdot 10^{-3}$
Plattform ohne Schwenkräder	$J_{Plattform}$	$37.249 \cdot 10^{-3}$

Tabelle 3: Parameterwerte der Lenkung

- $J_0$  bis  $J_2$ : Die Werte sind mit dem CAD berechnet.
- $J_{Plattform}$ : Der Wert berechnet sich aus der Masse der Plattform und der Exzentrizität  $d$  des Schwenkrades, wobei ein Drittel der Masse der Plattform 4 Kilogramm entspricht. [6]:

$$J_{Plattform} = \frac{1}{3} m_{Plattform} \cdot d^2 = 4 [\text{kg}] \cdot 0.0965^2 [\text{m}^2] \quad (2-50)$$

**Berechnung** Das Trägheitsmoment der Getriebe und des Schwenkrades auf die Motorwelle reduziert:

$$J_{getr\_schwenk\_mot} = J_0 + \frac{J_1}{i_1^2} + \frac{J_2}{i_{tot}^2} = 7.515 \cdot 10^{-6} [\text{kg} \cdot \text{m}^2] \quad (2-51)$$

Das Trägheitsmoment des Roboters  $J_{Plattform\_mot}$  auf die Motorwelle reduziert:

$$J_{Plattform\_mot} = \frac{J_{Plattform}}{i_{tot}^2} = 2.758 \cdot 10^{-5} [\text{kg} \cdot \text{m}^2] \quad (2-52)$$

Das gesamte Trägheitsmoment  $J_{tot}$  auf die Motorwelle reduziert:

$$J_{tot} = J_{enc} + J_{mot} + J_{getr\_schwenk\_mot} + J_{Plattform\_mot} = 4.95 \cdot 10^{-5} [\text{kg} \cdot \text{m}^2] \quad (2-53)$$

Das gesamte Trägheitsmoment  $J_{tot\_getr}$  auf den Getriebeausgang reduziert:

$$J_{tot\_getr} = J_{tot} \cdot i_{tot}^2 = 66.806 \cdot 10^{-3} [kg \cdot m^2] \quad (2-54)$$

Überblick	Trägheitsmoment	Kürzel	Auf Getriebeausgang reduziert [ $kg \cdot m^2$ ]	Auf Motorenausgang reduziert [ $kg \cdot m^2$ ]
Encoder	$J_{enc}$		$0.23 \cdot 10^{-3}$	$0.017 \cdot 10^{-5}$
Motorwelle	$J_{mot}$		$19.178 \cdot 10^{-3}$	$1.42 \cdot 10^{-5}$
Planetengetriebe auf Eingang	$J_0$		$3.241 \cdot 10^{-3}$	$0.24 \cdot 10^{-5}$
Übergang Planetengetriebe zum Ritzel	$J_1$		$0.045 \cdot 10^{-3}$	$0.003332 \cdot 10^{-5}$
Schwenkrad	$J_2$		$6.863 \cdot 10^{-3}$	$0.50816 \cdot 10^{-5}$
Summe der konstanten Trägheitsmomente ( $J_{enc}$ bis $J_2$ )	$J_{konst,lenk}$		$29.557 \cdot 10^{-3}$	$2.1885 \cdot 10^{-5}$
Summe des dynamischen Trägheitsmoments	$J_{Plattform}$		$37.249 \cdot 10^{-3}$	$2.758 \cdot 10^{-5}$
Gesamtes Trägheitsmoment	$J_{gesamt,lenk}$		$66.8063 \cdot 10^{-3}$	$4.947 \cdot 10^{-5}$

Tabelle 4: Trägheitsmomente der Lenkung im Überblick

**Bemerkung** Bei der Lenkung ist das dominanteste Trägheitsmoment dasjenige der Plattform. Da der selbe Motor auch beim Antrieb verwendet wird und das gesamte Massenträgheitsmoment der Lenkung dank des Getriebes kleiner ist als jenes des Antriebs, muss für die Lenkung das auftretende Motorenmoment nicht überprüft werden. Auch beim Umschwenken der Räder ist die Winkelbeschleunigung im Antrieb und nicht die der Lenkung kritisch.

### 2.6.3 Vereinfachte Massenmatrix für den OmniMoBot

**Allgemein** Aus den Kapiteln 2.6.1 und 2.6.2 kann nun die vereinfachte Massenmatrix hergeleitet werden:

$$\begin{bmatrix} 0.0343 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0343 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0343 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0668 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0668 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0668 \end{bmatrix} \quad (2-55)$$

Es ergibt sich eine 6x6-diagonale Matrix. Die Reihenfolge ist durch das Regelungskonzept gegeben. Zusätzlich wird davon ausgegangen, dass sich jeweils die drei Antriebe und die drei Lenkungen gleich verhalten.

## 2.7 Inverses Pendel

**Allgemein** Mit dem OmniMoBot soll einen Stab balanciert werden. Dieser verhält sich gleich wie ein inverses Pendel. In diesem Kapitel wird die Theorie dazu behandelt (siehe auch [8]).

## 2.7.1 Inverses Pendel: Theorie

**Definition** Das inverse Pendel hat den Schwerpunkt oberhalb der Drehachse, welche sich in der „ybase“ befindet (siehe Abbildung 25). Wenn der Schwerpunkt direkt über der Drehachse liegt, befindet sich das Pendel in einer instabilen Ruhelage.

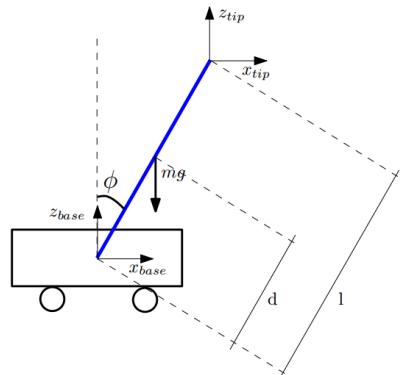


Abbildung 25: Inverses Pendel [8]

Die Pendel-Bewegung des Stabes kann durch horizontale Bewegungen des Wagens beeinflusst werden.

**Mathematische Definition** Das dynamische Verhalten des inversen Pendels kann durch das wirkende Momentengleichgewicht in der Drehachse hergeleitet werden:

$$\ddot{\phi} \cdot J_d = d \cdot m_{stab} \cdot (-\ddot{x}_{base} \cdot \cos(\phi) + g \cdot \sin(\phi)) - f_b \cdot \dot{\phi} \quad (2-56)$$

Mit dem Reibungsfaktor  $f_b$  können Reibungseinflüsse in der Drehachse berücksichtigt werden. Das in der Mitte des Stabes wirkende Trägheitsmoment  $J_d$  ist wie folgt definiert:

$$J_d = J_{stab} + d^2 \cdot m_{stab} \quad (2-57)$$

Das Trägheitsmoment des Stabes  $J_{stab}$  wird mittels dem „Satz von Steiner“ auf die Drehachse reduziert. Das wirkende Trägheitsmoment  $J_d$  kann dementsprechend wie folgt beschrieben werden:

$$J_d = r^2 \cdot m_{stab} + d^2 \cdot m_{stab} \quad (2-58)$$

Der Massen-Radius  $r$  steht dabei für  $\sqrt{\frac{J_{stab}}{m_{stab}}}$  (siehe [8]). Damit kann die Masse  $m_{stab}$  aus der Gleichung (2-56) gekürzt werden:

$$\ddot{\phi} = \frac{d \cdot (-\ddot{x}_{base} \cdot \cos(\phi) + g \cdot \sin(\phi))}{r^2 + d^2} - f_b \cdot \dot{\phi} \quad (2-59)$$

Weiter kann die Ist-Position der Stabspitze  $x_{tip}$  wie folgt definiert werden:

$$x_{tip} = x_{base} + l \cdot \sin(\phi) \quad (2-60)$$

Eine Beziehung zwischen der Beschleunigung in der Spitze des Stabes  $\ddot{x}_{tip}$  und dem Winkel  $\phi$  kann durch das Einbeziehen der Erdbeschleunigung erreicht werden:

$$\phi = \arctan\left(\frac{\ddot{x}_{tip}}{g}\right) \quad (2-61)$$

## 2.7.2 Inverses Pendel: OmniMoBot

Allgemein	Bei dem OmniMoBot wird das inverse Pendel in der Ebene ausbalanciert. Dafür muss der Winkel $\phi$ in der x-Achse und auch derjenige in der y-Achse bekannt sein. Diese Winkel werden mittels vier Hall-Sensoren ermittelt (siehe Anhang D IV, S. 110).
Inverses Pendel im 3D-Raum	In der vorhergehenden Theorie war nur die Rede von horizontaler Bewegung in x-Richtung, dasselbe gilt jedoch auch für die y-Richtung.

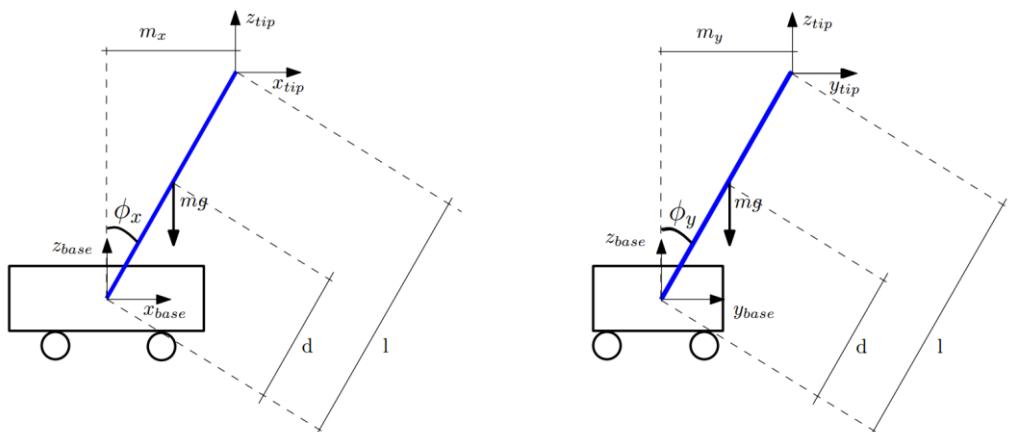


Abbildung 26: Inverses Pendel 3D [8]

Der Stab ist mit einem Magneten ausgestattet. Dieser befindet sich an der „x-base“ bzw. an der „y-base“. Der Magnet hält nicht den Stab an Ort und Stelle, sondern er wird benötigt für die Beschreibung der Winkel  $\phi_x$  und  $\phi_y$ . Dies wird erreicht, indem seine magnetische Flussdichte mittels der vier Hall-Sensoren in eine elektrische Spannung umgeformt wird, welche anschliessend in die Winkel  $\phi_x$  und  $\phi_y$  umgerechnet wird. Aus den Winkeln  $\phi_x$  und  $\phi_y$  kann die Position der Spitze des Stabes in x- und in y-Richtung bestimmt werden:

$$\begin{bmatrix} x_{tip} \\ y_{tip} \end{bmatrix} = \begin{bmatrix} x_{base} \\ y_{base} \end{bmatrix} + \begin{bmatrix} l \cdot \sin(\phi_x) \\ l \cdot \sin(\phi_y) \end{bmatrix} \quad (2-62)$$

Damit der OmniMoBot gleichzeitig fahren und den Stab balancieren kann, wird die Spitze des Stabs auf die Position geregelt.

## 2.7.3 Inverses Pendel: Regelungskonzept

**Allgemein** Damit die Position der Stabspitze geregelt werden kann, wird der Roboterregelung (siehe Abbildung 22, S. 30) eine Stabspitzenregelung vorgeschaltet.

**Konzept** Das Konzept, um die Position der Stabspitze  $x_{tip}$  und die Winkel  $\phi_x$  und  $\phi_y$  zu regeln, kann auf dieselbe Weise wie in Kapitel 2.5.1, S. 25 beschrieben umgesetzt werden. Dabei werden die Stabspitzenregelung und die Winkelregelung jeweils als ein mechanisches System betrachtet.

**Winkel-  
regelung**

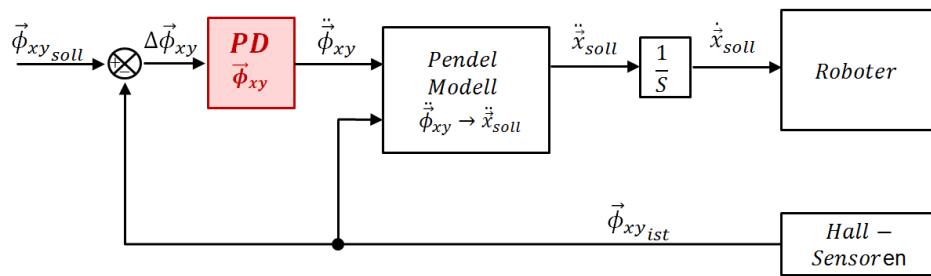


Abbildung 27: PD-Regelung des Stabwinkels

Der vorgegebene Winkel  $\vec{\phi}_{xy,soll}$  wird mit dem Winkel, welcher mittels der Hall-Sensoren ermittelt wurde, verglichen. Anschliessend wird er durch einer PD-Regelung ausgeregelt. Die daraus entstehende Winkelbeschleunigung  $\ddot{\phi}_{xy}$ , wird anhand der Gleichung ( 2-59 ) in eine Beschleunigung in x- und in y-Richtung umgeformt ( $\ddot{x}_{soll}$ ). In Kapitel 2.5.1, S. 25 wird erwähnt, dass von einer Eigenfrequenz in Hz  $f_0 = 56.8$  Hz für die Roboter Regelung ausgegangen wird. Bei einer kaskadierten Anordnung von Regelungen muss die innere Regelung mindestens um den Faktor 10 schneller sein als die äussere. Wenn eine Eigenfrequenz in Hz  $f_0$  von ca. 1.59 Hz mit einem Dämpfungsgrad  $D$  von 0.7 für die PD-Regelung des Stabwinkels  $\vec{\phi}_{xy}$  verwendet wird, ergibt sich mit der Gleichung ( 2-37 ) eine ungedämpfte Eigenkreisfrequenz des geschlossenen Regelungskreis  $\omega_0$  von  $10 \frac{rad}{sek}$ . Aus den Gleichungen ( 2-35 ) ergeben sich somit folgende Reglungsparameter:

$$k_p = 100 \quad \text{und} \quad k_d = 14 \quad (2-63)$$

**Stabspitzen-  
regelung**

Die Stabspitzenregelung wird vor die Stabwinkelregelung geschaltet. Dies hat zur Folge, dass die PD-Regelung für die Stabspitze mindestens den Faktor 10 langsamer sein muss als die Stabwinkelregelung.

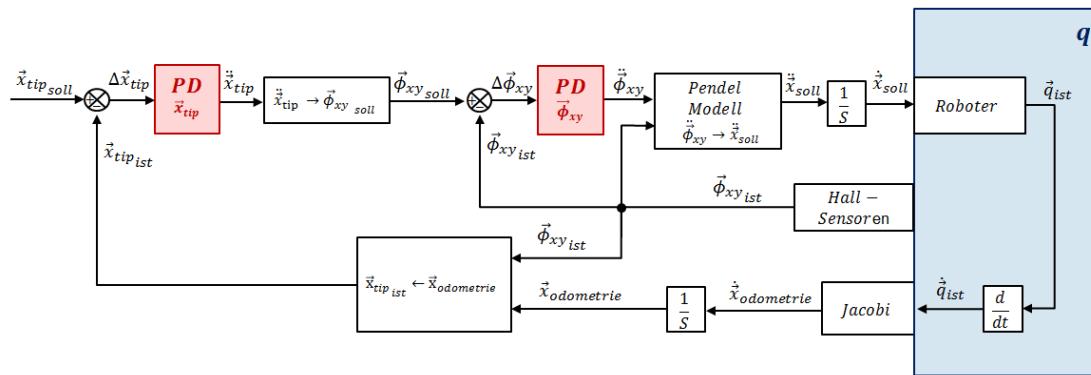


Abbildung 28: PD-Regelung der Stabspitze

Die Stabspitzenposition  $\vec{x}_{tip,soll}$  kann mittels Joystick oder Bahnplaner vorgegeben werden. Geregelt wird die Position auf den Wert, welcher durch die Gleichung ( 2-60 ) aus der Odometrie und den Winkeln  $\vec{\phi}_{xy,ist}$ , berechnet wurde. Nach der PD-Regelung kann mittels der Gleichung ( 2-61 ) auf den notwendigen Stabwinkel  $\vec{\phi}_{xy,soll}$  geschlossen werden.

Die Eigenfrequenz in Hz  $f_0$  für die Stabspitzenregelung wird auf  $0.159 \text{ Hz}$  gesetzt und ist somit um den Faktor 10 langsamer als die Winkelregelung. Daraus ergeben sich bei einem Dämpfungsgrad  $D$  von 0.7 folgende Werte:

$$k_p = 1 \quad \text{und} \quad k_d = 1.4 \quad (2-64)$$

Die Regelungsparameter ( 2-63 ) und ( 2-64 ) sind eine erste Annahme, sie werden durch eine Simulation noch genauer bestimmt.

## 2.8 EEROS

- Allgemein [51]** EEROS steht für **E**asy, **E**legant, **R**eliable, **O**pen and **S**afe Real-Time Robotics Software Framework. In diesem Framework können Roboter nach einer bestimmten Architektur programmiert werden. Das Framework erlaubt die Implementierung von hardwarenahen Echtzeit-Regelungen bis hin zu high-level Steuerungsapplikationen. EEROS verwirklicht eine industriefähige Open Source Robotersoftware, welche sowohl sicher als auch einfach zu warten ist.
- Aktueller Stand** EEROS befindet sich zurzeit immer noch in Entwicklung, wurde jedoch bereits bei verschiedenen Robotertypen angewandt, z. B. bei einem Delta-, einem Knickarm- und einem Scara-Roboter [52, 53].

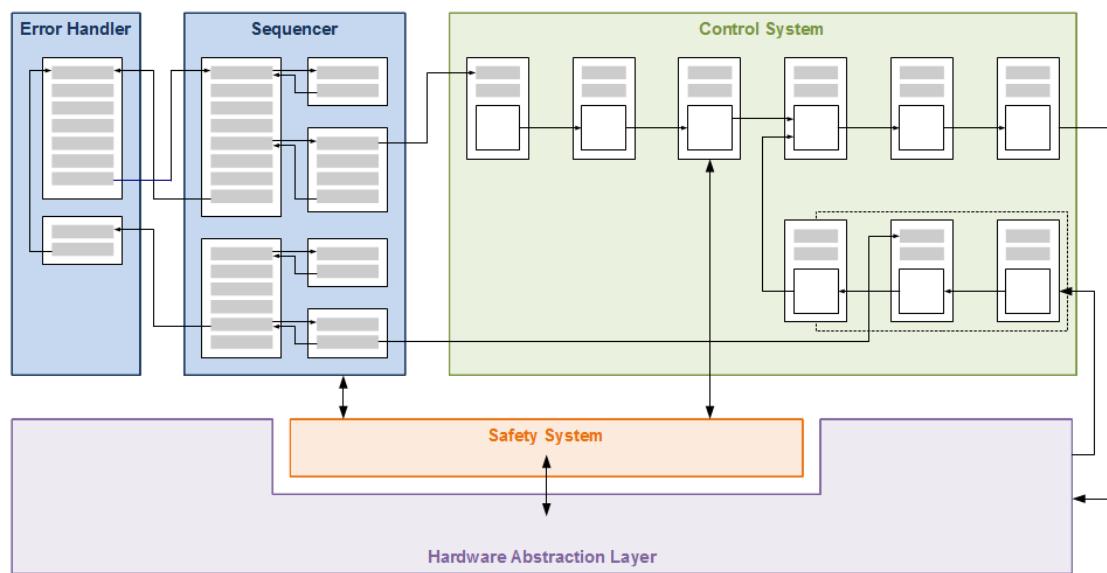
**Architektur**

Abbildung 29: EEROS Architektur [51]

EEROS besteht im Wesentlichen aus drei eng verknüpften Subsystemen:

- Bewegungssteuerung, Echtzeitverhalten (Control System)
- Sicherheitssystem (Safety System)
- Ablaufsoftware (Sequencer)

**Control  
System  
[51]**

Das Control System erlaubt das Erstellen von Regelungsstrukturen, ähnlich wie in Simulink. Um solche Strukturen zu erstellen, stehen dem User drei Komponenten zur Verfügung: Signale, Blöcke und Zeitdomänen.

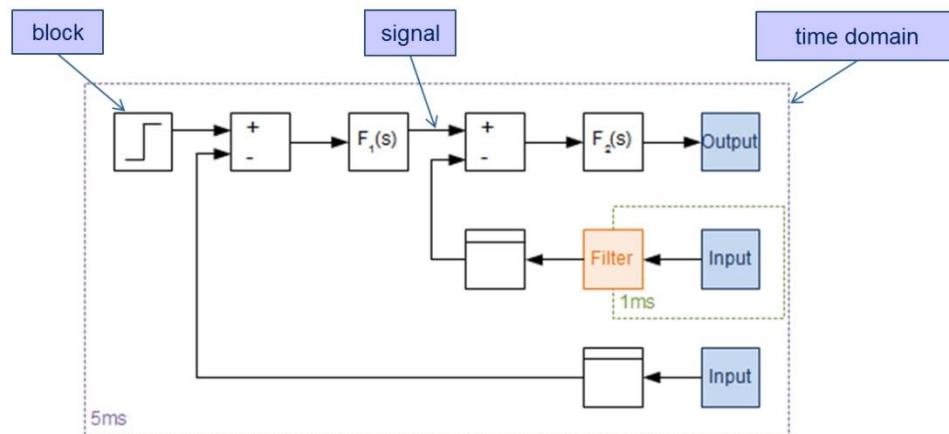


Abbildung 30: Beispiel einer Regelungsstruktur [51]

**Signale** repräsentieren ändernde Werte zu einem bestimmten Zeitpunkt. Jedes Signal enthält deshalb einen Zeitstempel, welcher den Zeitpunkt der Erfassung des Signals darstellt. Somit können Operationen, wie zum Beispiel das Differenzieren, angewendet werden ohne massgeblich von Effekten wie Jitter oder Totzeiten beeinflusst zu werden. Signale können Skalare, Vektoren oder sogar Matrizen sein und enthalten immer einen Namen und den erwähnten Zeitstempel.

In den **Blöcken** werden die Operationen realisiert, welche die Signale verändern.

Die einzelnen Blöcke werden konfiguriert und zusammengesetzt, jedoch ohne grafische Unterstützung. Die Verbindungen werden erzeugt, indem die „connect“-Methode der Eingänge aufgerufen wird: `f1block.in.connect(sumBlock.out)`

Jeder Block gehört zu einer **Zeitdomäne**, welche die Taktfrequenz der Ausführung steuert.

<b>Safety System</b>	<p>Das Safety System überwacht das Gesamtsystem und jede Teilkomponente. Dabei wird nicht nur der Endbenutzer und der Roboter, sondern bereits der Entwickler vor den Folgen allfälliger Programmierfehler geschützt. Das wird erreicht, indem verschiedene Sicherheitslevels (safety levels) eingeführt werden. Dabei hat jedes Level einen eindeutigen Namen und eine eindeutige Nummer. Je höher das Level, desto höher ist die Gefahr in einem Fehlerfall.</p> <p>Für jedes Level werden sicherheitsrelevante Ausgänge (z.B. Watchdog) sowie sicherheitsrelevante Eingänge (z.B. Bestätigungstaster) definiert. Zusätzlich wird festgelegt, wie die sicherheitsrelevanten Ausgänge gesetzt werden sollen, wenn die sicherheitsrelevanten Eingänge von den Sollwerten abweichen. Die sicherheitsrelevanten Ausgänge können vom Control System oder von den Sequenzen nur mit dem Einverständnis des Safety Systems angesprochen werden.</p> <p>Durch das Auslösen von Events kann zwischen den Sicherheitslevels gewechselt werden. Ein solcher Event kann von einem Sequencer, dem Control System, von sicherheitsrelevanten Eingängen oder vom Safety System selbst ausgelöst werden.</p>
----------------------	--

<b>Sequencer</b> [51]	<p>Allein mit dem Control System wird es schwierig, ein Roboterprogramm zu schreiben, welches eine Aufgabe ausführt, deren Level höher ist als nur einer bestimmten Bahn zu folgen. Es wird daher etwas benötigt, um solche Programme zu erstellen. In EROS wird dies mit dem Sequencer gelöst.</p> <p>Eine Sequenz ist eine lineare Liste mit Schritten, welche gewisse Aktionen auf dem Roboter ausführen. Ein Schritt kann Untersequenzen haben, welche blockierend aufgerufen werden oder es können Sequenzen definiert werden, welche parallel ablaufen. Solche Sequenzen können miteinander synchronisiert werden, indem auf das Ende eines bestimmten Schrittes gewartet wird.</p>
--------------------------	---

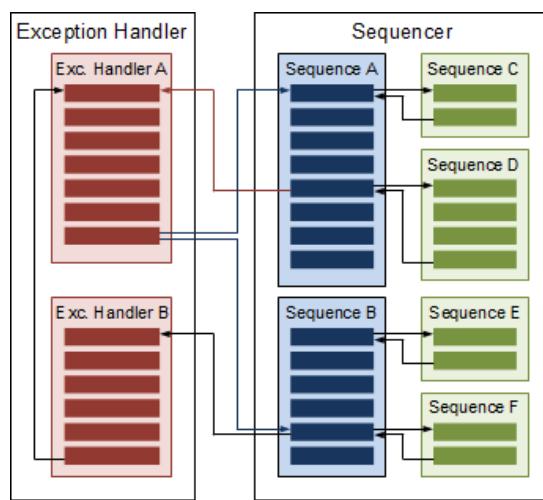


Abbildung 31: Beispiel Sequencer [51]

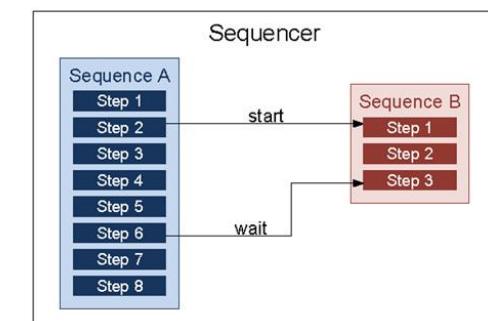


Abbildung 32: Parallel laufende Sequenzen [51]

Für jede Sequenz können spezielle Sequenzen definiert werden, die wiederum bestimmte Fehlerfälle behandeln (z.B. *Timeout*). Tritt nun solch ein Fehlerfall ein, dann wird die entsprechende Exception Handler Sequenz ausgeführt. Diese Sequenz

bestimmt dann, ob der fehlgeschlagene Vorgang wiederholt oder rückgängig gemacht wird oder ob ein anderer Prozess ausgeführt werden soll.

## 2.9 fLINK

<b>Allgemein</b>	<p>In diesem Kapitel wird kurz ein Software Layer mit dem Namen „fLink“ vorgestellt. Er wurde im Rahmen einer Masterthesis entwickelt [10]. Dabei wird ein ähnliches Ziel wie bei dem Software Framework EEROS verfolgt. Software soll nicht auf spezifische Probleme zugeschnitten werden, sondern sie soll so universell und einfach wie möglich eingesetzt werden können.</p> <p>In dieser Masterthesis wurde der Software Layer „fLink“ für die Kommunikation zwischen Prozessor und FPGA verwendet. „fLink“ steht einerseits für das englische <b>FPGA-Link</b> und andererseits für das deutsche flink im Sinne von schnell, zügig.</p> <p>Die Software „fLink“ ermöglicht eine universelle, flexible und hardware-unabhängige Schnittstelle für die Anbindung eines FPGAs an ein Echtzeitbetriebssystem.</p>
<b>Schnittstellen Abstraktion</b> [10]	<p>Eingänge und Ausgänge werden nach ihrer Funktion zusammengefasst und zu einer gemeinsamen, logischen Schnittstelle zusammengefasst. Um das zu erreichen, werden die spezifischen Eigenschaften der verwendeten Hardware abstrahiert. Auf diese Weise kann zum Beispiel eine „<i>Analog Input</i>“-Schnittstelle, welche einen n-kanaligen ADC repräsentiert, erstellt werden. Ob diese mit dem FPGA über SPI oder über eine parallele Schnittstelle verbunden ist, spielt durch das Abstrahieren der Schnittstelle für die Anwendung keine Rolle mehr. Für die abstrahierte Schnittstelle sind neben den allgemeinen Eigenschaften wie Auflösung, Anzahl Kanäle, etc. nur der eingelesene Wert und allenfalls noch Informationen zum Aufnahmezeitpunkt relevant.</p>
	<p>Für diese Abstraktion wird ein logischer Gerätebaum verwendet, der drei Ebenen kennt:</p> <ul style="list-style-type: none"> <li>• <b>Device:</b> Gruppe von Subdevices, üblicherweise entspricht ein Device einem FPGA.</li> <li>• <b>Subdevice:</b> Funktionale Einheit von einem Typ, zum Beispiel „<i>Analog Input</i>“ mit 8 Kanälen oder „<i>Digital I/O</i>“ mit 32 Kanälen.</li> <li>• <b>Channel:</b> Einzelner Kanal eines Subdevices, entspricht zum Beispiel bei einem digitalen I/O einem einzelnen Pin.</li> </ul>
<b>Beispiel eines Systemauf- baus</b> [10]	<p>Der Systemaufbau für eine fLink-Anwendung könnte aus folgenden Teilkomponenten bestehen:</p>

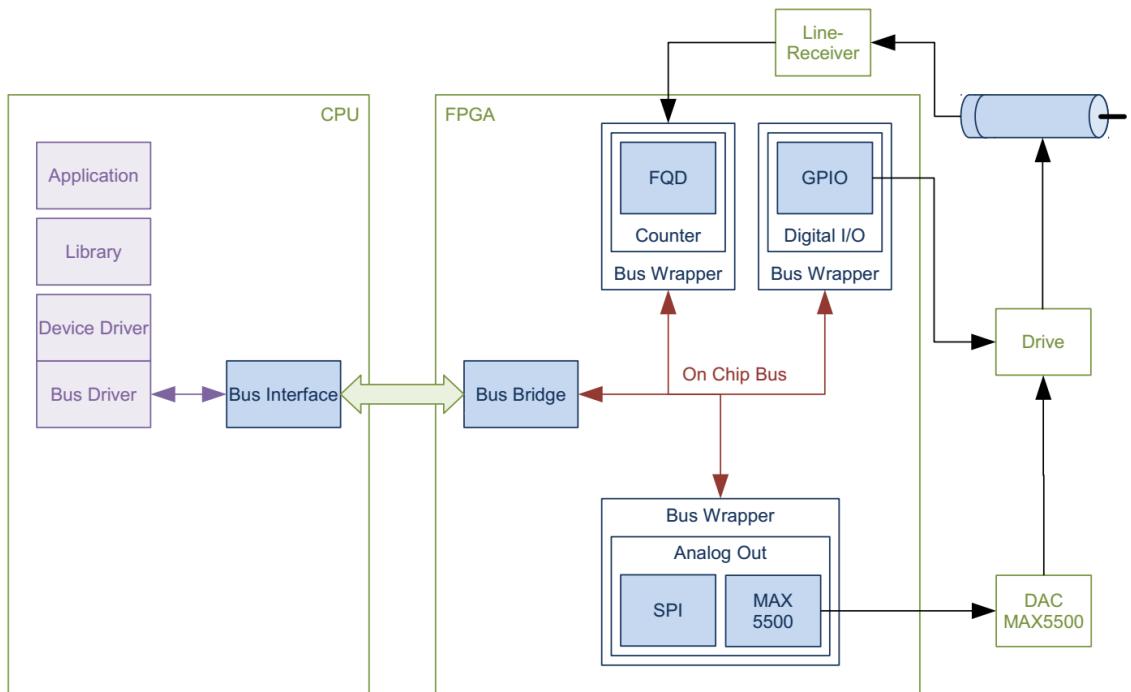


Abbildung 33: Ein Systemaufbau für eine fLink-Beispielanwendung [10]

Die Anwendung greift mit einer Anwendungsbibliothek auf den Gerätetreiber zu, welcher je nach verwendetem Kommunikationsbus über den entsprechenden Bus-Treiber mit dem FPGA kommuniziert. Letzterer enthält die nach Funktion getrennten Subdevices, welche über einen ON-Chip-Bus und die „Bus Bridge“ mit dem Kommunikationsbus zum Prozessor verbunden sind.

Für eine detailliertere Erklärung wird auf die Literatur [10] verwiesen.

#### Programmbibliothek und Anwendung

Wenn nun von einer einfachen Motorregelung als Anwendung ausgegangen wird, welche nicht direkt im FPGA sondern als Echtzeit-Anwendung unter Linux, z.B. Preempt\_RT, realisiert werden soll, müssen dem Entwickler für seine Regelung folgende Hardware-Zugriffe möglich sein:

- Auslesen des Inkremental-Encoders als Istwert.
- Ausgeben einer analogen Spannung über den DAC als Stellwert.
- Freigabe für den Motor-Drive über einen digitalen Ausgang.

Die Programmbibliothek stellt Methoden bereit, welche einen einfachen Zugriff auf diese Hardware-Ressourcen ermöglichen. Als Pseudo-Code könnte das Auslesen des Encoders etwa wie folgt aussehen:

```
encPos = readCounter(device, subdevice, channel);
```

#### fLink-Programmbibliothek [10]

Die fLink-Programmbibliothek „*flinklib*“ bietet dem Entwickler einen einfachen Zugriff auf ein fLink-Device. Dazu werden dem Programmierer zwei Anwendungsprogrammierschnittstellen (APIs) angeboten (siehe Abbildung 34): eine High-Level API für einen komfortablen Zugriff auf bekannte Subdevice-Typen und eine Low-Level API für den Zugriff auf benutzerdefinierte Subdevice-Typen.

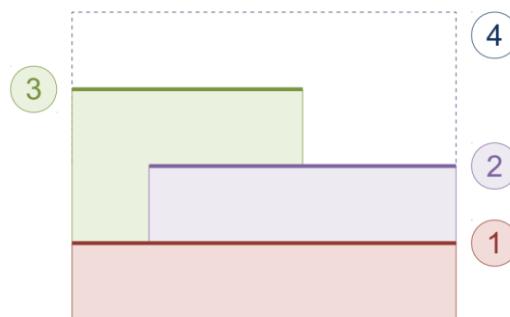


Abbildung 34: API Levels der fLink Library [10]

1. **Internal API:** Für Anwender nicht verwendbar.
2. **Low-Level API:** Für Anwender nutzbar, kann zum Beispiel für die Handhabung von benutzerdefinierten Modul-Schnittstellen verwendet werden. Sie bietet direkten Zugriff auf den Speicherbereich des ausgewählten Subdevices. Dazu stehen Methoden für das Lesen und Schreiben sowie für I/O-Control bereit.
3. **High-Level API:** Hauptschnittstelle, bietet komfortablen Zugriff auf bekannte Modul-Schnittstellen. Zusätzlich werden einige allgemeine Funktionen vorgegeben:
  - Device öffnen / schliessen
  - Anzahl Subdevices auslesen
  - Subdevice auswählen

Beispiele für bekannte Modul-Schnittstellen (Subdevices), welche mit fLink ange-  
sprochen werden können:

- **Digital I/O:** I/O Richtung festlegen, Ausgang setzen, Eingang lesen
- **Counter:** Countermode setzen, Counterwert lesen
- **PWM:** PWM Frequenz setzen, Hightime setzen

4. Anwendung oder externe Bibliothek, welche die fLink-Library verwendet.

Für detailliertere Erläuterungen wird wiederum auf die Literatur [10] verwiesen.

### 3 Simulation mit Simulink

<b>Allgemein</b>	In diesem Kapitel werden die Kinematik, das Regelungskonzept, die wirkenden Trägheitsmomente sowie das inverse Pendel in Simulink zusammengefügt und getestet.
<b>Sample time</b>	Die Simulation rechnet mit einem fixen Step von einer Millisekunde. Diese „Sample time“ entspricht der späteren Regelungstakt-Rate $f_{Task}$ des realen OmniMoBots.
<b>Parameter / Vereinfachungen</b>	<p>Die Simulationsparameter können dem m-File<sup>1</sup> oder dem Anhang A (S. 100) entnommen werden. Die Angaben entsprechen dem realen System, welches in Kapitel 4.1, S. 53 vorgestellt wird.</p> <p>Für die Simulation gelten folgende Vereinfachungen</p> <ul style="list-style-type: none"> <li>• In der gesamten Simulation werden die Reibung und der Luftwiderstand vollständig vernachlässigt.</li> <li>• Der zu balancierende Stab befindet sich konzentrisch zu der Roboterachse.</li> <li>• Die mechanische Verkopplung, welche in Kapitel 2.2.1, S. 16 behandelt wurde, wird in dieser Simulation nicht berücksichtigt.</li> </ul>
<b>Besonderes</b>	<ul style="list-style-type: none"> <li>• Es gilt zu beachten, dass in dieser Simulation gleich alle sechs Motoren auf einmal angesteuert werden. Dafür muss z.B. die in Kapitel 2.6.3, S. 35 erstellte Massenmatrix als Vektor in den „Jredgetr“-Block (siehe Abbildung 40, S. 49) implementiert werden.</li> <li>• Dank EEROS ist es möglich, die Form der Simulink-Implementation eins-zu-eins für C++ zu übernehmen. Aus diesem Grund wurde der Simulationsaufbau umfassender als notwendig modelliert.</li> </ul>

#### 3.1 Aufbau des Simulationsmodells

<b>Allgemein</b>	Dieses Simulink-Modell repäsentiert den OmniMoBot, welcher durch die Vorgabe der Stabspitzen-Position geregelt werden kann. Gleichzeitig ist er in der Lage, den Stab auszubalancieren. Mit diesem Modell können die Regelungsparameter für das inverse Pendel überprüft und optimiert werden.
------------------	--

<sup>1</sup> Anhang F CD in Ordner: 5\_Matlab\0\_Balance\_Simulation\0\_omnidirektionales\_Fahren, m-File: „mainStabbalance“

## Simulink- Modell

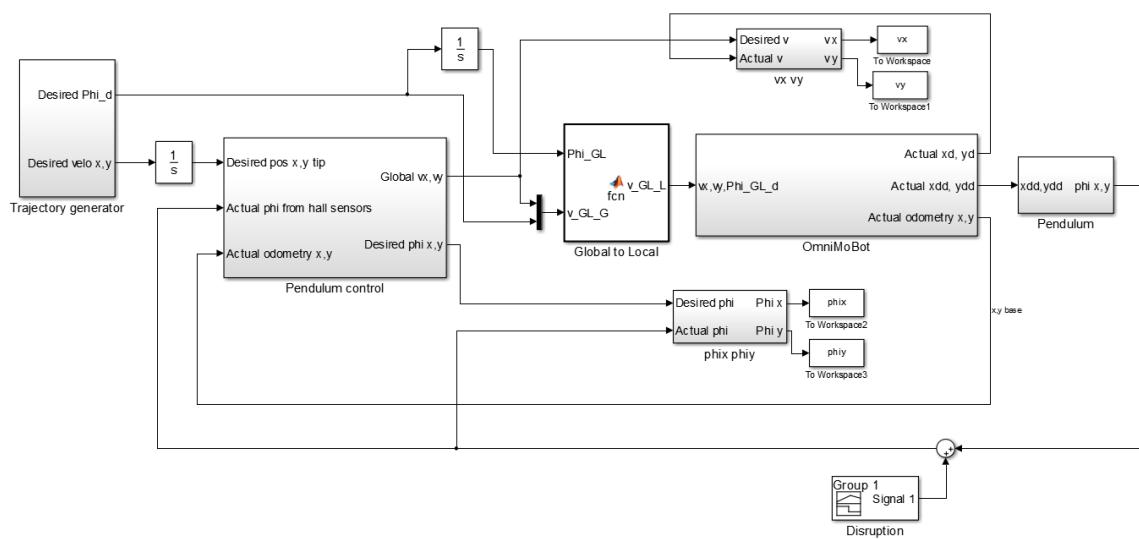


Abbildung 35: Simulink Modell stabbalancierender OmniMoBot<sup>2</sup>

### Erläuterung

Das Simulink-Modell (Abbildung 35) ist aus mehreren Subsystemen zusammengestellt. Diese wurden in den vorangegangenen Kapiteln behandelt.  
Mit einem Trajektorien-Generator wird in dieser Simulation eine Geschwindigkeit im globalen KS  $\{G\}$  vorgegeben. Diese wird integriert und als Soll-Vorgabe für die Stabspitzen-Position verwendet.  
Das Submodell „Pendulum control“ regelt die Stabspitzenposition sowie dessen Winkel Phi aus den Daten der Hall-Sensoren und aus den Odometriedaten des OmniMoBots. Das Subsystem gibt hieraus die Sollgeschwindigkeit in x- und y-Richtung vor. Dem OmniMoBot kann wegen seiner omnidirektionalen Fähigkeit eine zusätzliche Drehung vorgegeben werden. Da die Geschwindigkeit im globalen KS  $\{G\}$  vorgegeben wird, ist für die Transformation in das lokale KS  $\{L\}$  die Integration der Rotationsgeschwindigkeit notwendig.

Um die Daten der Hall-Sensoren zu simulieren, wurde der Subsystemblock „Pendulum“ generiert. Dieser beinhaltet die Gleichung (2-59) von S. 36. Mit dem Block „Disruption“ wird eine Störung von aussen simuliert, z.B. das Antippen des Stabes.

### 3.1.1 Subsystem „Trajectory generator“

#### Beschreibung

Der Trajektorien-Generator-Block besteht aus zwei „Signal Builder“-Blöcken, mit welchen ein Geschwindigkeitsverlauf in x- und y-Richtung vorgegeben wird. Der dritte „Signal Builder“-Block gibt einen Verlauf für die Rotationsgeschwindigkeit vor.

Eine Geschwindigkeitsvorgabe mit nachträglicher Integration hat den Vorteil, dass die vorgegebene Position keinen sprunghaften sondern einen rampenförmigen Positionsverlauf aufweist. Bei sehr steifen Regelungen dürfen keine zu grossen Sprünge als Sollwert vorgegeben werden, weil dies zu starken Überschwingungen führen kann. Solange nicht ein Bahnplaner eine Positions vorgabe für den mobilen Roboter errechnet, kann die Geschwindigkeit alternativ mittels eines Joysticks vorgegeben werden.

<sup>2</sup> Anhang F CD in Ordner: 5\_Matlab\0\_Balance\_Simulation\0\_omnidirektionales\_Fahren, slx-File: „Stabbalance“

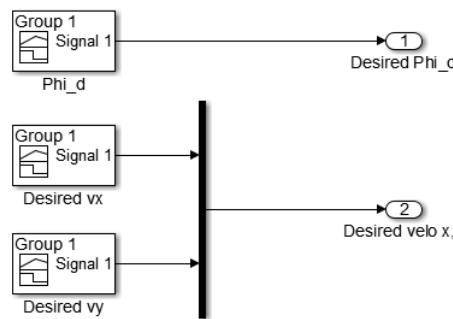
**Aufbau**

Abbildung 36: Subsystem „Trajectory generator“

**3.1.2 Subsystem „Pendulum control“****Allgemein**

Das Subsystem „Pendulum control“ entspricht der PD-Regelung der Stabspitze (siehe Abbildung 28, S. 39).

Der Zweck dieser Simulation ist, passende Regelungsparameter zu finden für die in Abbildung 37 rot markierten PD-Regelungen.

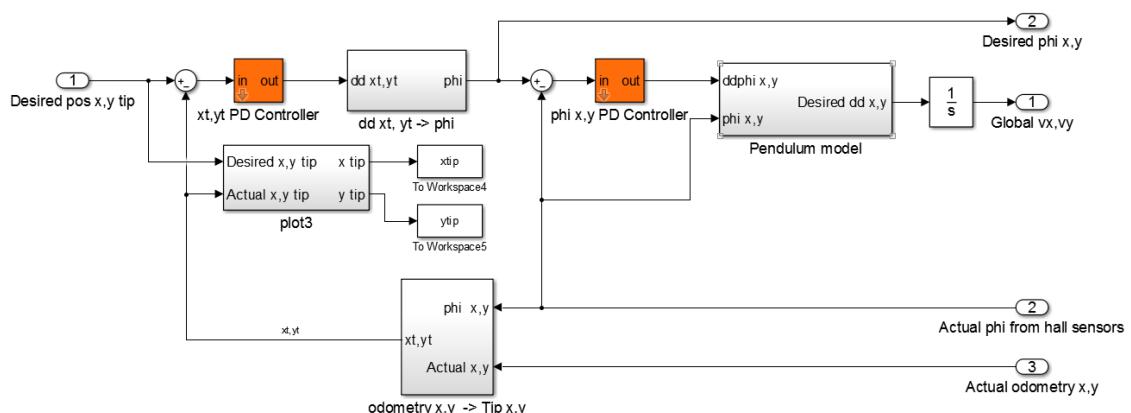
**Aufbau**

Abbildung 37: Subsystem „Pendulum control“

**3.1.3 Subsystem „OmniMoBot“****Allgemein**

Das Subsystem „OmniMoBot“ beinhaltet alle Komponenten, die sich auf den OmniMoBot beziehen. Sie werden in den Unterkapiteln detaillierter behandelt.

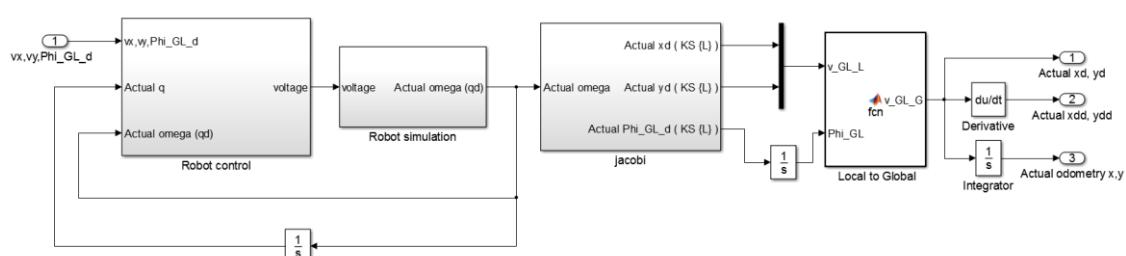
**Aufbau**

Abbildung 38: Subsystem „OmniMoBot“

**Beschreibung** Dieses Subsystem simuliert den OmniMoBot. Dazu ist im Subsystem „Robot control“ das Regelungskonzept „Kaskadierte Regelung im Gelenkkoordinatensystem“ (siehe S. 30) hinterlegt. Mittels des Subsystems „Robot simulation“ werden die „realen“ Bedingungen simuliert (siehe Abbildung 39). Für die Regelung des inversen Pendels sind die Daten der Odometrie des OmniMoBots erforderlich. Die Odometriedaten des OmniMoBots werden ausgegeben, indem die Geschwindigkeiten der Motoren des OmniMoBots in der Jacobimatrix umgerechnet und durch den Block „Local to Global“ in das globale KS  $\{G\}$  transformiert und anschliessend integriert werden. Im Block „Local to Global“ wurde die Gleichung (2-15) implementiert.

Die Jacobimatrix ist mittels einer S-Funktion umgesetzt. Der Code ist im Anhang A (S. 102) zu finden. Da bei dieser Simulation die mechanische Verkopplung nicht berücksichtigt wird, wurde mit einem vereinfachten Jacobi-Algorithmus simuliert. Dieser macht keine Fallunterscheidung, er verwendet ein lineares Mittel der Radgeschwindigkeit  ${}^L\vec{v}_{GR_i}$  zur Bestimmung der resultierenden Geschwindigkeit des Roboters  $\vec{v}_{res}$ .

**Subsystem  
„Robot  
simulation“**

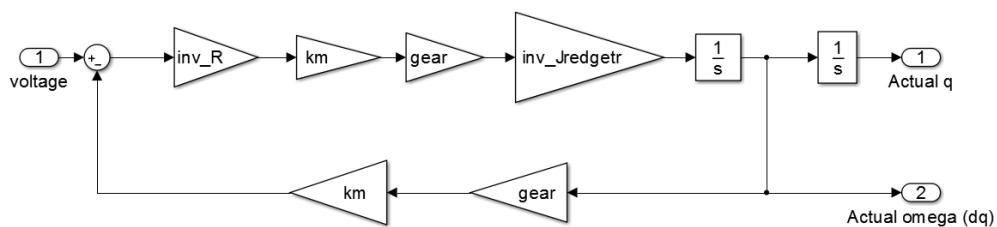


Abbildung 39: Subsystem „Robot simulation“

Diese Simulation könnte vereinfacht dargestellt werden, indem die Blöcke „Jredgetr“, das Subsystem „Motor model“ und das Subsystem „Robot simulation“ (Abbildung 40) durch zwei in eine Reihe gestellte Integratoren ersetzt werden.

### 3.1.3.1 Subsystem „Robot control“

**Allgemein** In diesem Subsystem ist neben dem zuvor beschriebenen Regelungskonzept (siehe S. 30) zusätzlich ein „Homingblock“ und ein „Motor model“-Block enthalten. Der „Homingblock“ ist für diese Simulation nicht relevant. Er wird erst später verwendet für die Implementierung des Control Systems mittels EEROS (siehe Kapitel 4.3.1.2, S. 64). Dasselbe gilt für den „Switch“-Block. Die inverse Jacobi ist, wie die direkte Jacobimatrix, mittels einer S-Funktion umgesetzt. Der Code dazu ist im Anhang A (S. 103) zu finden.

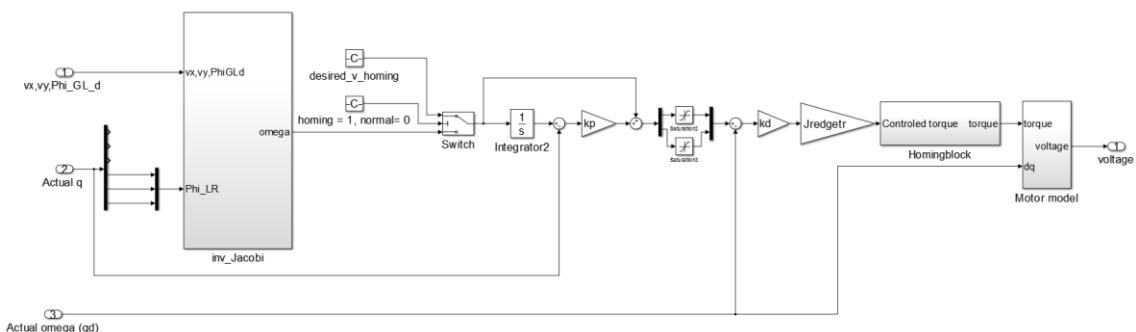
**Aufbau**

Abbildung 40: Subsystem „Robot control“

**Erläuterung**

Mit den Geschwindigkeitsvorgaben im lokalen KS  $\{L\}$  und den aktuellen Positionen der Lenkungsmotoren können mittels der inversen Jacobimatrix die Geschwindigkeiten  $\omega$  der Gelenkstellungen  $q$  berechnet werden. In der Gleichung (2-21) der inversen Jacobimatrix auf S. 21 ist der Winkel  $\varphi_{GL}$  enthalten. Auf diesen Winkel  $\varphi_{GL}$  kann hier verzichtet werden, weil sich die inverse Jacobimatrix aus der Abbildung 40 bereits im lokalen KS  $\{L\}$  befindet.

Für die Simulation wird beim „Switch“-Block der Wert 0 vorgegeben, auch der „Homing-block“ wird deaktiviert.

Mit dem „Motor model“-Block wird das benötigte Antriebsmoment  $\tau$  in eine Spannungsvorgabe für die Motoren umgewandelt. Die Begrenzungen sind auf die Lehrlaufdrehzahlen der Motoren eingestellt (siehe Kapitel 2.5.2.3, S. 30).

**Subsystem „Motor Model“**

Die Herleitung für die Umrechnung des Antriebsmoments in die Spannungsvorgabe kann im Anhang A (S. 98) nachvollzogen werden.

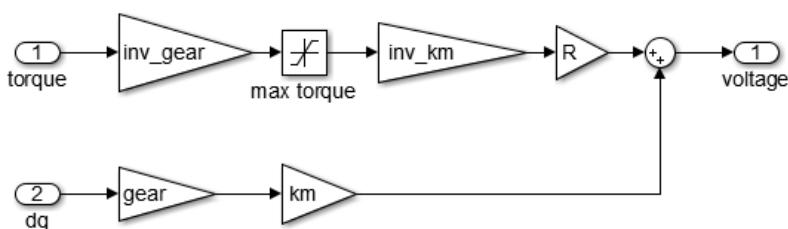


Abbildung 41: Subsystem „Motor model“

Die Begrenzung ist auf das dreifache des Nennmoments der Motoren festgelegt (0.531 Nm).

### 3.2 Simulationsergebnisse

**Allgemein**

Es werden verschiedene Regelungsparameter untersucht, als erstes die in Kapitel 2.7.3, S. 38 erwähnten. Anschliessend wird nach geeigneteren Parametern gesucht. Die nachfolgend aufgeführten Vorgaben bleiben für alle Simulationen dieselben.

**Dämpfungsgrad  $D$** 

Der Dämpfungsgrad  $D$  ist sowohl bei der Spitzen- als auch bei der Winkelregelung auf den Wert 1 gesetzt.

**Trajektorien-  
Generator  
Vorgabe**

Dem Trajektorien-Generator werden folgende Geschwindigkeitsprofile für die x- und y-Richtung vorgegeben:

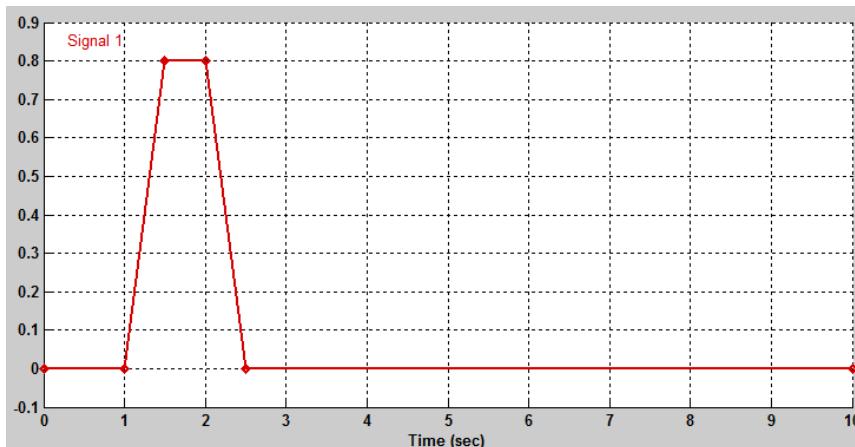


Abbildung 42: Sollvorgabe für den Trajektorien-Generator für die Geschwindigkeit in x- und y-Richtung, Einheit y-Achse [m/s]

Durch die Vorgabe einer Bahn für die Stabspitze kann die Ausregelung der Stabspitze bei einer zusätzlichen Bewegung des Roboters simuliert werden.

**Vorgabe der  
Rotations-  
geschwindig-  
keit**

Weiter wirkt folgende Sollwertvorgabe für die Rotationsgeschwindigkeit des Omni-MoBots:

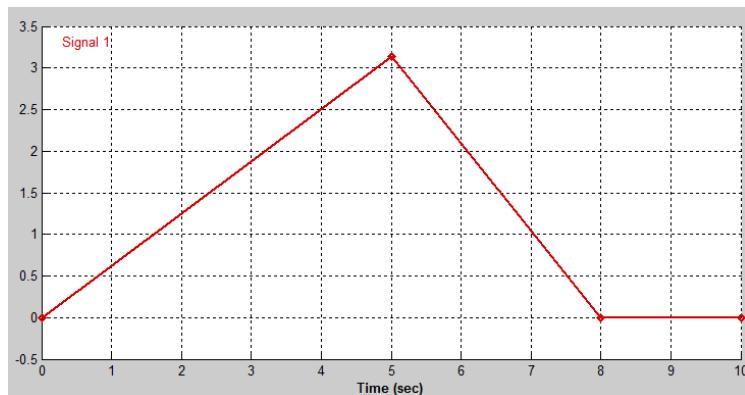


Abbildung 43: Sollwertvorgabe für die Drehgeschwindigkeit des OmniMoBots, Einheit y-Achse [rad]

Mit dieser Vorgabe kann der Einfluss einer Drehung des OmniMoBots simuliert werden. Die Zentrifugalkraft wird in dieser Simulation nicht berücksichtigt. Trotzdem hat eine zusätzliche Drehung des Roboters um seine eigene Achse einen starken Einfluss auf die Leistung der Antriebe. Wenn die Antriebe durch die Drehung in Bewegung sind und zur selben Zeit ein Stoß wirkt, treten hohe Beschleunigungen auf. Deshalb ist es möglich, dass durch eine zusätzliche Drehung um die eigene Achse die Begrenzung schneller erreicht wird als ohne eine solche Drehung. Die Drehung hat jedoch keinen Einfluss auf den Geschwindigkeitsverlauf des OmniMoBots (siehe Abbildung 46). Auch auf die Ausregelung der Stabspitze (siehe Abbildung 45) hat die Drehung keinen Einfluss.

**Störungs-  
Vorgabe**

Die Störung, welche ein Antippen des Stabes simulieren soll, tritt bei dem Zeitpunkt  $t = 5\text{s}$ , für eine Zeitspanne von 0.24 Sekunden, mit einem Sprung von 0.1 rad auf.

### 3.2.1 Ergebnis: Spitzenregelungsfrequenz 0.159 Hz, Winkelregelungsfrequenz 1.59 Hz

**Allgemein** Mit den Parametern aus dem Kapitel 2.7.3, S. 38 ist die Regelung zu träge. Dies wird veranschaulicht mit dem Soll- und Istwert der Stabspitzenposition in x-Richtung.

#### Ergebnis

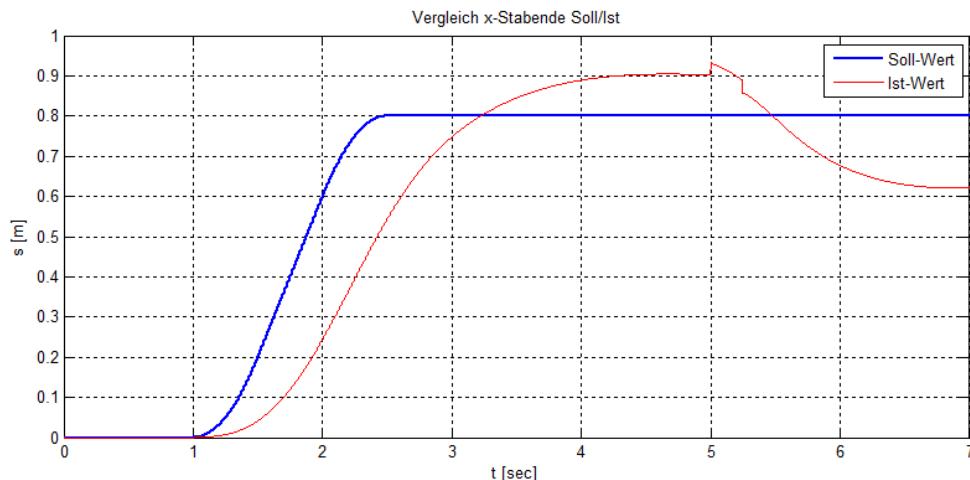


Abbildung 44: Träger Stabspitzenpositionsverlauf in x-Richtung

**Erkenntnis** In der Abbildung 44 stellt der Sollwert die Stellgrösse des Trajektorien-Generators dar. Es ist zu erkennen, dass der Istwert, welcher durch die Odometriedaten definiert ist, eine deutliche Abweichung vom Sollwert des Trajektorien-Generator hat. Aus diesem Grund werden die Eigenfrequenzen in Hz  $f_0$  der Spitzenregelung und der Winkelregelung erhöht.

### 3.2.2 Ergebnis: Spitzenregelungsfrequenz 0.915 Hz, Winkelregelungsfrequenz 3.183 Hz

**Allgemein** Die besten Ergebnisse werden mit den Eigenfrequenzen in Hz  $f_0$  0.915 Hz (Spitzenregelung) und 3.183 Hz (Winkelregelung) bei einem Dämpfungsgrad  $D$  von 1 erreicht. Dabei muss jedoch der D-Anteil der Winkelregelung um den Faktor 20 verkleinert werden.

Als erstes wird die Stabspitzenposition und anschliessend der Verlauf des Geschwindigkeitsvergleichs des Soll- und Istwerts des OmniMoBots dargestellt. Es ist bei beiden Abbildungen die Bewegung in x-Richtung dargestellt.

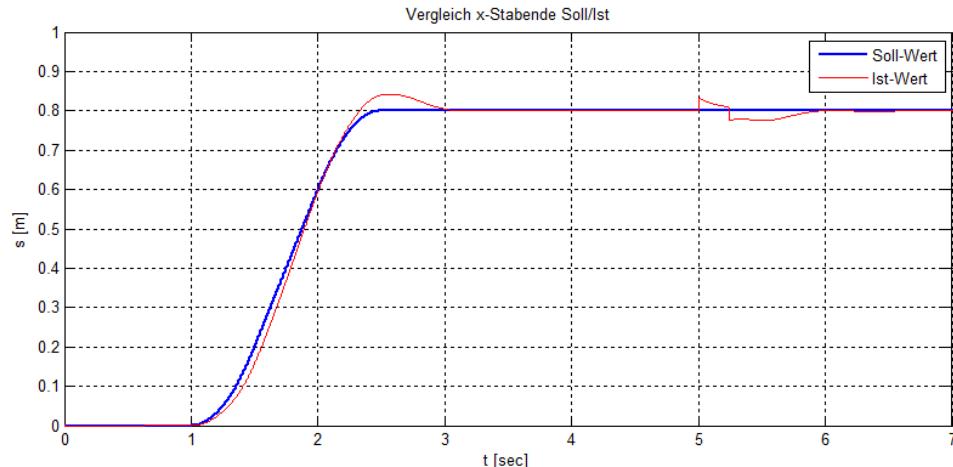
**Ergebnisse**

Abbildung 45: Verhalten der Stabspitze in x-Richtung

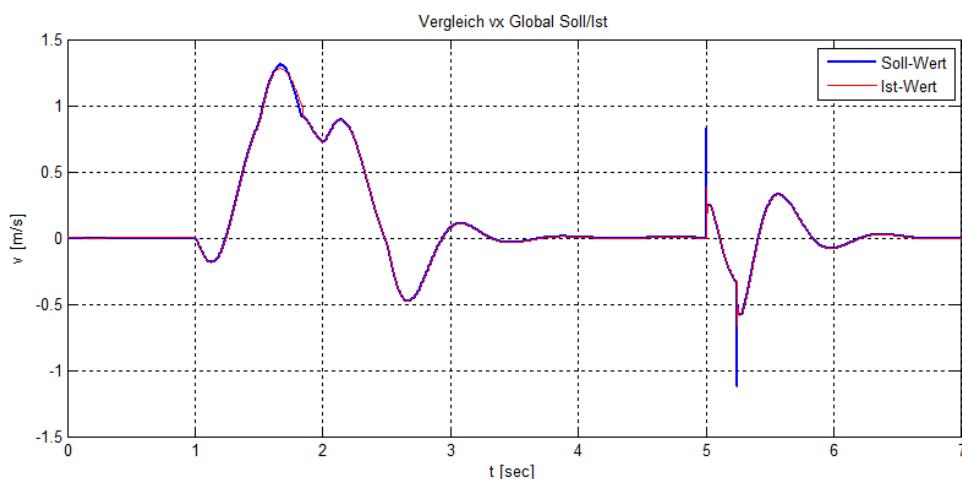


Abbildung 46: Geschwindigkeitsverlauf des OmniMoBots in x-Richtung

**Erkenntnis**

Die Abbildung 45 zeigt ein straffes Regelungsverhalten der Stabspitzenposition mit einer leichten Überschwingung des Istwerts. Auch die Störgrösse wird schnell ausgeregelt.

In der Abbildung 46 ist gut ersichtlich, wie der Roboter durch die Stabspitzenvorgabe seine Bahn fährt. Als erstes bewegt sich der Roboter in die entgegengesetzte Richtung der Stabspitzenvorgabe. Dies hat den Effekt, dass sich die Stabspitze in die gewünschte Richtung bewegt.

In beiden Abbildungen ist ersichtlich, wie gut eine Störung, welche bei  $t = 5\text{s}$  auftritt, ausgeregelt werden kann.

Durch das Simulieren wird klar, dass der OmniMoBot den Stab höchstwahrscheinlich nicht viel schneller als mit  $1\text{ m/s}$  balancieren kann. Denn um die Bewegungsenergie des Stabes beim Anhalten auszuregeln, sind hohe Beschleunigungen des OmniMoBots notwendig. Durch diese Beschleunigungen kommen die Motoren schnell an ihre Leistungsgrenzen.

Im Allgemeinen hat die Simulation gezeigt, dass ein Balancieren des Stabes möglich ist. Aufgrund der Vereinfachungen und Annahmen, welche für die Simulationen des Systems getroffen wurden, müssen die Regelungsparameter hinsichtlich des echten OmniMoBots nochmals justiert werden.

## 4 Praktische Umsetzung

**Allgemein** In diesem Kapitel werden der OmniMoBot, die Hardware, die aufgetretenen Probleme, die Software des OmniMoBots sowie die Ergebnisse vorgestellt.

### 4.1 OmniMoBot

**Hardware Übersicht** Der komplette Schaltplan des OmniMoBots ist im Anhang E ab S. 114 zu finden.

**Bild**

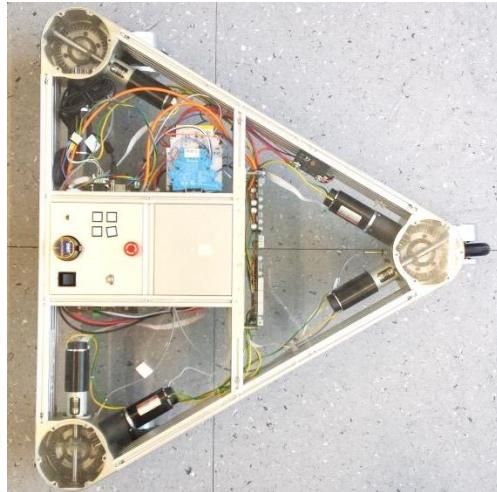


Abbildung 47: OmniMoBot

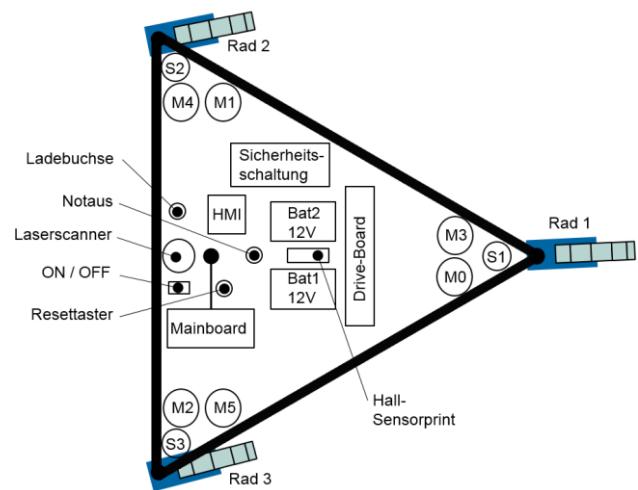


Abbildung 48: Komponenten des OmniMoBots

**Ziel des OmniMoBots** Mit dem OmniMoBot können die Fähigkeiten eines mobilen Roboters mit Schwenkrad-Antrieb aufgezeigt werden. Um die Fähigkeiten zu demonstrieren, wird vom OmniMoBot ein Stab ausbalanciert.



Abbildung 49: Balancierender OmniMoBot

**Beschreibung** Als Übergang vom Stab zur Plattform dient eine 60°-Spitze, welche auf ein kleines Loch in der Plattform platziert wird. Somit besteht keine feste mechanische Verbindung zwischen Stab und OmniMoBot. Der Stab ist aus einem Aluminiumrohr gefertigt.

Die Positionsvorgabe für den Stab bezieht sich auf das obere Ende des Stabes. Der Winkel des Stabes wird mittels dem Hall-Sensorprint, welcher bereits in Kapitel 2.7, S. 35 erwähnt wurde, gemessen. Damit ist der Roboter in der Lage, den Stab auszubalancieren. Zusätzlich zum Ausbalancieren kann sich der Roboter gleichzeitig um die eigene Achse drehen. Dies ist durch die Regelung im globalen KS  $\{G\}$  möglich, weil eine zusätzliche Drehung um die eigene Achse keinen Einfluss hat auf die Positionsbestimmung des oberen Endes des Stabes im globalen KS  $\{G\}$ .

## 4.2 Hardware-Probleme und Entscheidungen

**Allgemein** Die Mechanik [6] und die Elektronik [5] wurden in zwei vorangegangenen Masterthesen ausgelegt.

Sie wurden jedoch noch nie getestet, was natürlich Probleme mit sich bringen kann.

In diesem Kapitel wird beschrieben, welche Probleme mit der verwendeten Hardware auftraten und welche hardwarebezogenen Entscheidungen getroffen wurden.

**Verwendete Hardware** Die detailliertere Beschreibung der Hardware ist im Anhang D ab S. 105 zu finden. Folgende Hardware wurde verwendet:

- Mainboard („NTB C32 Carrier Board“<sup>3</sup>)
- Drive-Board („NTB 6xDC“<sup>4</sup>)
- Hall-Sensorprint
- HMI
- 6 DC-Motoren (RE40 148867)
- Laserscanner URG-04LX
- Batterie

### 4.2.1 Drive-Board-Entscheidung

**Allgemein** Es stehen zwei Drive-Boards zur Verfügung. Das erste ist von der Firma MAXON (Typ „ESCON Module 50/5“), das zweite wurde in einer vorangegangenen Masterthesis eigens für den OmniMoBot entwickelt („NTB 6xDC“). Da das „NTB 6xDC“ Drive-Board jedoch nicht wie geplant funktioniert, muss abgewogen werden, ob dieses Drive-Board trotz der auftretenden Fehler verwendet wird oder ob die Alternative, das „ESCON“-Board, vielversprechender ist.

**„NTB 6xDC“:  
Aufgetretene Fehler** Die Kommunikation mit dem Mainboard sollte bei diesem Drive-Board über SPI erfolgen. Bei der Inbetriebnahme hat sich jedoch gezeigt, dass die SPI-Kommunikation mit dem Mainboard viel zu langsam ist. Der Mikrocontroller ist zu langsam, was eine sehr hohe CPU-Auslastung zur Folge hat. Der Mikrocontroller kann mit dem Wegfallen der SPI-Kommunikation etwas entlastet werden.

<sup>3</sup> Anhang F CD in Ordner: 4\_Elektronik\0\_Prints\Mainboard

<sup>4</sup> Anhang F CD in Ordner: 4\_Elektronik\0\_Prints\Drive-Board(NTB6xDC)

Aus diesem Grund werden, bei einer Verwendung dieses Drive-Boards, die PWM-Outputs des Mainboards direkt mit den Mosfets des Drive-Boards verbunden. Diese Lösung kann jedoch zu EMV-Problemen führen und ist auch aus montagetechnischen Gründen nicht anzustreben.

#### 4.2.1.1 ESCON Module 50/5

**Allgemein** Ein „ESCON“-Board ist sehr klein und kann sowohl für DC- als auch für EC-Motoren verwendet werden. Für den OmniMoBot sind jedoch sechs dieser Boards notwendig.



Abbildung 50: ESCON Module 50/5

Die Drive-Boards müssen als erstes getestet werden, damit entschieden werden kann, welches verwendet werden soll. Die Einzelheiten zum Test sind im Anhang B (S. 104) aufgeführt.

**Erkenntnisse**

- Die **Hauptspannungsversorgung**, welche das Drive-Board versorgt, soll angelegt werden, **bevor die Spannung 5 VDC mit der Stromquelle verbunden wird**. Andernfalls muss im Programm ESCON Studio die Fehlermeldung gelöscht werden, damit das Drive-Board wieder richtig angesteuert werden kann.
- Das Mainboard hat die Möglichkeit, 8 PWM-Signale auszugeben. Wenn jedoch nur ein PWM-Signal vorgegeben wird und alle **PWM-Signale mit einem Flachbandkabel** weggeführt werden, entsteht ein **starkes Rauschen**. Der Grund dafür könnten die restlichen, nicht definierten PWM-Signale sein. Wenn nicht alle PWM-Signale verwendet werden, muss den nicht verwendeten Signalen ein Wert vorgegeben werden, damit diese nicht floaten. Dadurch wird das Rauschen minimiert.
- Wenn eine Fehlermeldung erscheint, muss diese mit einem Rechtsklick im Programm „ESCON Studio“ gelöscht werden. Ansonsten reagiert das Drive-Board auf keine Signale.
- Da der Duty Cycle im „ESCON Studio“ auf 10%, beziehungsweise 90%, begrenzt ist, werden im Testprogramm alle Werte, die kleiner als 10% Duty Cycle sind, gleich 10% gesetzt. Dementsprechend werden auch alle Werte, die höher als 90% Duty Cycle sind, gleich 90% gesetzt. Die Auflösung des berechneten Duty Cycle wird dadurch nicht verschlechtert. Eine Verschlechterung der Auflösung würde zu unkontrollierbaren Bewegungen des Rotors führen.
- Die Frequenz des PWM-Signals muss mindestens doppelt so hoch sein wie die Regelungstakt-Rate  $f_{Task}$ .

**Fazit**

Der Test hat gezeigt, dass mit dem „ESCON“-Board eine sehr straffe und robuste Regelung umgesetzt werden kann. Damit sind die „ESCON“-Boards eine mögliche Alternative zum fehlerbehafteten „NTB 6xDC“ Drive-Board.

**4.2.1.2 Entscheidung****Fazit**

Nach ausgiebigen Tests und Versuchen wurde entschieden, dass das „NTB 6xDC“ Drive-Board verwendet wird. Die ausschlaggebenden Gründe sind:

- Durch die direkte Verbindung der PWM-Outputs mit den Mosfets treten keine EMV-Probleme auf.
- Bei Verwendung der ESCON-Module müsste eine zusätzliche Platine gelayoutet werden.
- Die Leistung der ESCON-Module ist etwas zu gering.
- Geld- und Zeitmangel für die Fertigung einer alternativen Platine für die ESCON-Module.

**4.2.2 Mikrocontroller****Allgemein**

Der verwendete Mikrocontroller MPC5200 (*siehe Anhang D I, S. 106*) läuft nah an seiner Leistungsgrenze. Dies hat zur Folge, dass die Anforderungen, welche durch die Echtzeitprogrammierung gegeben sind, nicht in jedem Fall erfüllt werden.

**Echtzeit-  
anforderung**

Die Echtzeitanforderung ist erfüllt, wenn das Control System (*siehe Kapitel 4.3.1, S. 60*) mit all seinen Blöcken einmal komplett durchläuft und dabei die Periode, welche durch die Regelungstakt-Rate  $f_{Task}$  definiert ist, zu keinem Zeitpunkt überschritten wird.

**Lösungen**

1. Verwendung eines schnelleren Prozessors. Dies hat jedoch zur Folge, dass ein neues Mainboard entwickelt werden muss.
2. Anpassung der Regelungstakt-Rate  $f_{Task}$ , soweit bis eine Echtzeit Regelung möglich ist. Dies ist jedoch nur begrenzt empfehlenswert, denn durch die Anpassung der Regelungstakt-Rate  $f_{Task}$  werden auch die Regelungsparameter (*siehe Kapitel 2.5.1, S. 25*) angepasst. Die Anpassung der Regelungsparameter verhindert, dass die Regelung instabil wird. Jedoch werden die Regelungsparameter durch diese Anpassung etwas weicher, wodurch die straffe Regelung des Omni-MoBots verlorengeht.

**Regelungstakt-  
Rate 400 Hz**

Die Regelungstakt-Rate  $f_{Task}$  wird auf 400 Hz gesetzt. Mit dieser Takt-Rate ist die Regelung des Roboters noch genügend straff, um den Stab auszubalancieren. Bei einer höheren Takt-Rate haben Messungen gezeigt, dass die „run“-Methode des Control Systems (*siehe Kapitel 4.3.1*) nicht in jedem Fall innerhalb der vorgegebenen Periode ( $1/f_{Task}$ ) beendet wurde. Wenn die Periode nicht eingehalten wird, kann die Regelung instabil werden.

**4.2.3 Hall-Sensorprint****Allgemein**

Tests mit dem Hall-Sensorprint haben gezeigt, dass die Ausgaben des Prints willkürliche Ausschläge aufweisen. Diese konnten grösstenteils ausgefiltert werden, was jedoch nur

eine Übergangslösung ist. Der Print ist über SPI mit dem Mainboard verbunden.

### SPI-Signal

Das SPI wird über den folgenden ADC erzeugt:

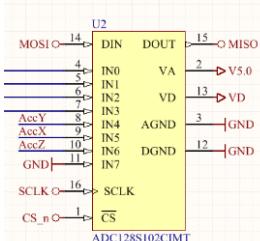


Abbildung 51: ADC128S102CIMT

Der ADC erfasst acht Signale. Bei IN0 bis IN3 kommen die Signale der Hall-Sensoren an, bei IN4 bis IN6 die Werte des Beschleunigungssensors und bei IN7 wird der GND des SPI angeschlossen.

### Testbedingungen

Bei den folgenden Tests werden alle Daten von IN0 bis IN7 mit Hilfe von fLink erfasst und mittels Matlab dargestellt.

Die Messung erfolgt über eine Dauer von 50 Sekunden.

Tests zufolge spielt es keine Rolle, ob der Hall-Sensorprint eingebaut oder frei hängend ist.

Für die folgenden Tests werden alle anderen Komponenten des OmniMoBots von der Stromversorgung getrennt. Damit können EMV-Einflüsse ausgeschlossen werden.

Alle acht erfassten Signale verhalten sich gleich.

### Messergebnis mit Störung

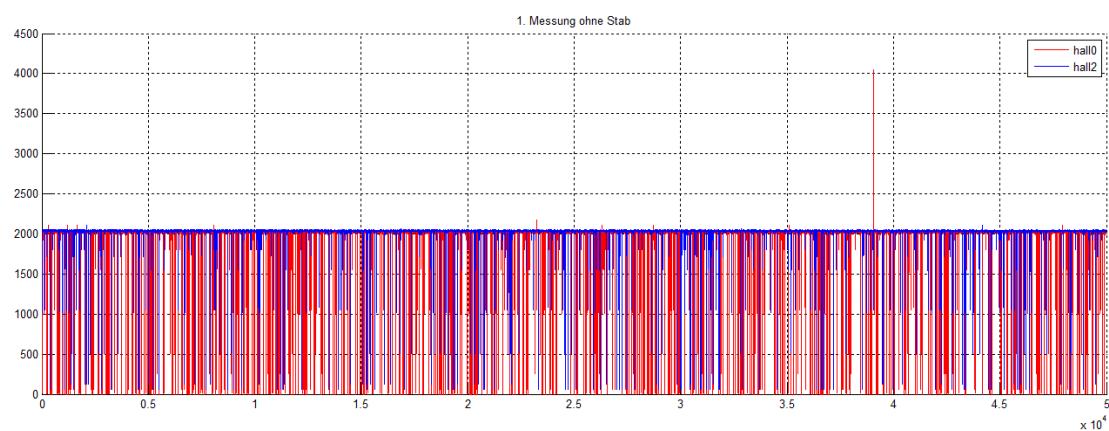


Abbildung 52: Hall-Sensorprint mit Störungen über 50 Sekunden

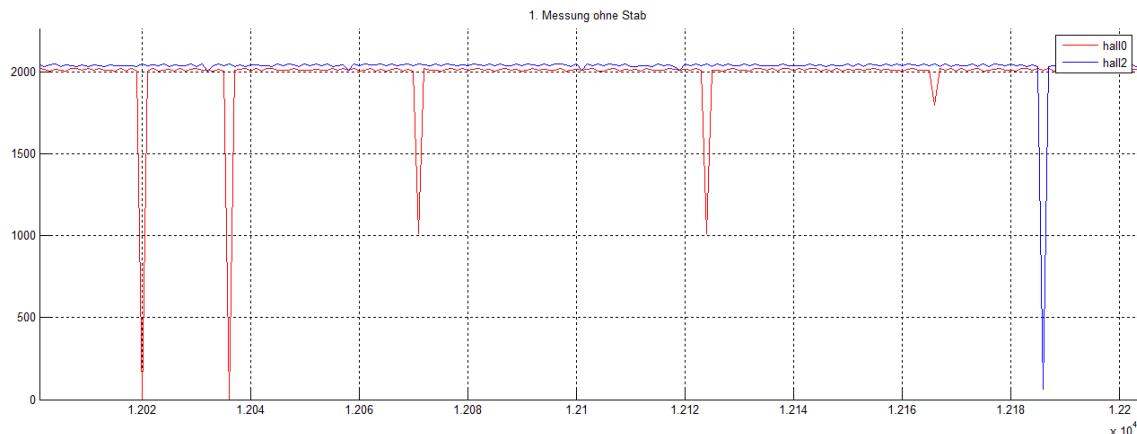


Abbildung 53: Hall-Sensorprint mit Störungen über 0.22 Sekunden

<b>Untersuchungen</b>	Aus den in Abbildung 52 und Abbildung 53 dargestellten Messungen können keine abschliessenden Erkenntnisse gewonnen werden. Auch das genauere Untersuchen des SPI-Signals sowie Messungen direkt auf dem Hall-Sensorprint führen zu keinen schlüssigen Ergebnissen.
<b>Anhaltspunkt</b>	<p>Den ersten Anhaltspunkt lieferte eine Messung bei verringter Stromversorgung, als der Akku fast leer war.</p> <p>Diese Messung zeigt keine einzige Störung. Somit wird deutlich, dass es sich bei dem aufgetretenen Fehler um keinen Softwarefehler handeln kann.</p> <p>Zusätzlich kann aufgrund dieser Messung der Fokus auf die Stromversorgung gelegt werden.</p>
<b>Fehlerursache</b>	Der Print wird mit 12 Volt vom Mainboard versorgt. Das SPI-Signal wird mit dem FPGA erzeugt. Durch die zusätzliche 12 Volt Spannungsversorgung wird eine Schlaufe mit der Masse (GND) erzeugt. Das bedeutet, dass die 12 Volt Spannungsversorgung nicht das selbe Massepotenzial hat wie das SPI-Signal vom FPGA.
<b>Messergebnis ohne Störung</b>	Nach der Behebung dieser Schlaufe werden die folgenden Signale gliefert:

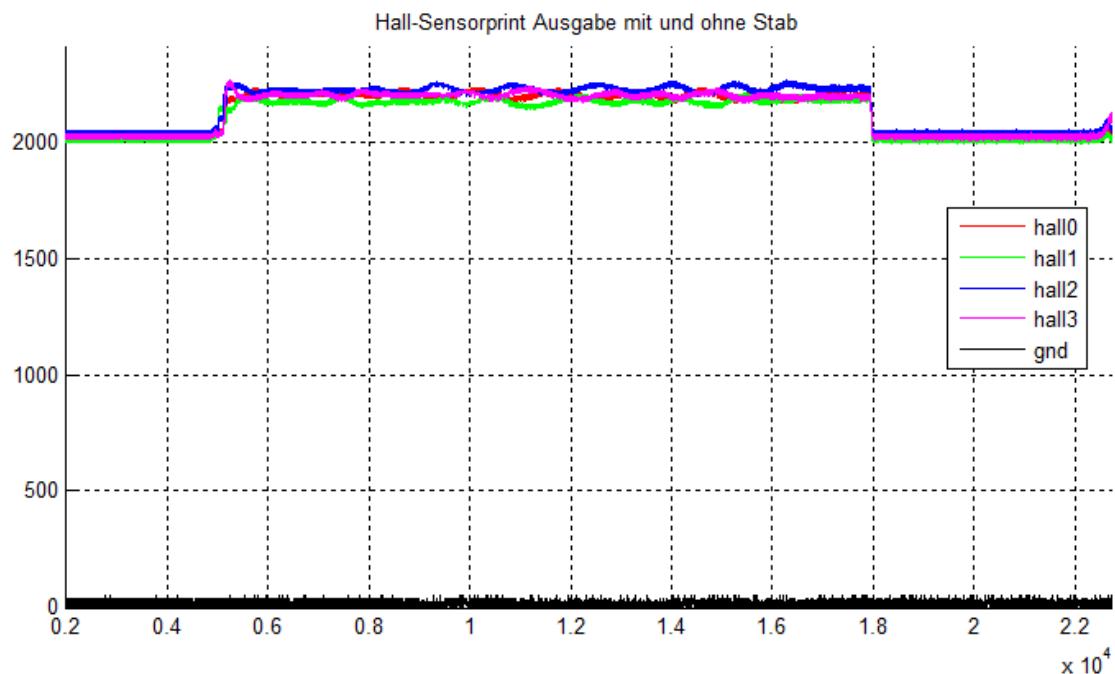


Abbildung 54: Hall-Sensorprint voll funktionsfähig

Die Abbildung 54 zeigt die neuen Ausgaben des Hall-Sensorprints. Der Hall-Sensorprint ist dabei in den OmniMoBot eingebaut und alle anderen Komponenten sind ebenfalls angeschlossen. Der zu detektierende Stab wird nach ca. fünf Sekunden hinzugefügt, etwas hin und her bewegt und nach ca. 18 Sekunden wieder entfernt. Mit diesen Ausgaben ist es nun möglich, den Stab sauber auszubalancieren.

#### 4.2.4 Sicherheitsschaltung

##### Allgemein

Der Schaltplan zu der Sicherheitsschaltung ist im Anhang E ab S. 114 zu finden.

Die Sicherheitsschaltung stellt sicher, dass die Motoren nur mit Strom versorgt werden, solange der Mikrocontroller noch das Watchdog-Signal sendet und der Not-Aus-Taster nicht betätigt ist.

##### Funktion

Das Kernstück der Sicherheitsschaltung ist das Sicherheits-Schaltgerät „SNO 4062KM“, welches jedoch nur einen Eingang überwacht. Um nun das Watchdog-Signal zusätzlich zum Not-Aus-Taster auszuwerten, muss die Auswertung des Watchdogs über drei weitere Relais gehen. Das erste Relais befindet sich auf dem Mainboard, es ist die eigentliche Watchdog-Auswertung. Dieses Relais schaltet zwei Koppelrelais. Es sind zwei notwendig, weil das Sicherheits-Schaltgerät nur mit einem Reset-Taster wieder freigeschaltet werden kann. Damit der Reset-Taster aber freigeschaltet wird, muss auch der Pfad des Reset-Tasters unterbrochen werden. Die beiden Koppelrelais werden mit dem Not-Aus-Taster in Serie geschaltet.

Mit dem Sicherheits-Schaltgerät kann nicht direkt die Spannungsversorgung der Motoren unterbrochen werden, weil nur ein maximaler Dauerstrom von sechs Ampere über die Schaltung fliessen darf. Dies ist eindeutig zu wenig, denn ein Motor hat bereits sechs Ampere als Nennstrom. Aus diesem Grund wird mit dem Sicherheits-Schaltgerät ein zusätzliches Halbleiterrelais angesprochen. Mit diesem können bis zu 60 Ampere geschaltet werden.

<b>Mögliche Probleme</b>	Wenn die Sicherheitsschaltung die Spannungsversorgung aus scheinbar unerklärlichen Gründen unterbricht, muss Folgendes unternommen werden: <ul style="list-style-type: none"> <li>• Batteriestand prüfen</li> <li>• Watchdog-Zeit kontrollieren (in welcher Zeit der Watchdog wieder gesetzt werden muss)</li> <li>• Die Regelungstakt-Rate <math>f_{Task}</math> überprüfen. Bei zu hoher Takt-Rate ist die CPU zu langsam, wodurch der Watchdog nicht rechtzeitig gesetzt wird.</li> </ul>
--------------------------	--

## 4.3 Software

**Allgemein** In diesem Kapitel wird das implementierte Control- und Safety-System vorgestellt.

Auf einen Sequencer kann verzichtet werden, weil der OmniMoBot keine Aufgabe erfüllen muss, die mehr erfordert, als einer Bahn zu folgen.

Nach der Vorstellung des Safety Systems werden die Probleme, die mit fLink und EROS aufgetreten sind, behandelt.

Die Kernelkonfiguration für das c32 Board, das CMake und der Bootcode sind im NTB-Student's Wiki [56] und auf der CD im Anhang in Ordner 6\_Software zu finden.

**Software Stack** Der Software Stack des OmniMoBots ist wie folgt definiert:

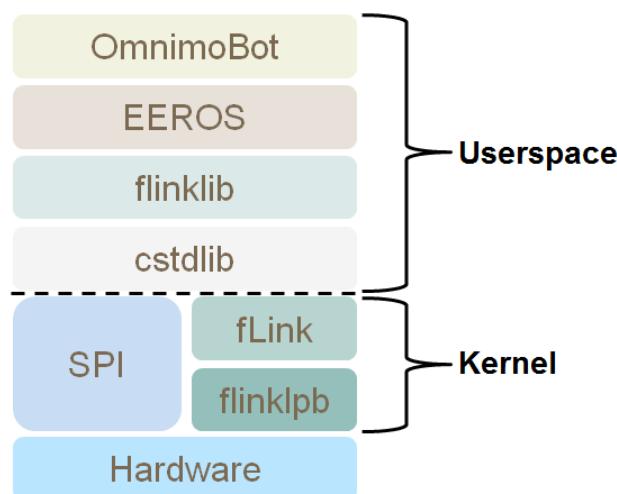


Abbildung 55: Software stack

Der Treiber von fLink ist der flinklpb (flink- locale plus bus).

### 4.3.1 Control System

**Allgemein** In diesem Kapitel wird die Regelungsstruktur, wie sie mit C++ umgesetzt wurde, vorgestellt. Diese ähnelt sehr stark dem bereits vorgestellten Simulink-Modell.

Mit EROS (siehe Kapitel 2.8, S.39) ist es möglich, eine Regelungsstruktur, welche

graphisch dargestellt ist, mit einfachen Befehlen in C++ in Textform wiederzugeben. Dieses Kapitel ist zur Ergänzung zu den C++-Codes<sup>5</sup> gedacht, damit diese beim Lesen leichter zu verstehen sind.

Die Regelungstakt-Rate  $f_{Task}$  liegt bei 400 Hz, weil der Mikrocontroller für 1 kHz zu langsam ist.

**Blockfarben** In den Kapiteln 4.3.1 bis 4.3.1.2 werden Blöcke in verschiedenen Farben dargestellt, dabei gilt Folgendes:

- **Grün:** Diese Blöcke werden von EEROS zur Verfügung gestellt. Es sind Blöcke, welche Input-Schnittstellen beschreiben. Dabei kann es sich um „Peripheralinput“-Blöcke oder um Subsystem-Input-Blöcke handeln. Mit einem Subsystem ist ein Block gemeint, welcher weitere Blöcke enthält. Die Subsystem-Input-Blöcke existieren nicht als solche im EEROS. Wenn ein Subsystem erstellt werden soll, können Methoden, die einen Input übergeben, definiert werden. So wird zum Beispiel für das Subsystem „Robot control“ folgende Methode erstellt:

```
eeros::control::Input<eeros::math::Vector<3>>& getInVelo()
{
    return invjacobi.getInVglobal();
}
```

Mit dieser Methode wird der Input des „invjacobi“-Blocks mit einem Vektor übergeben, welcher drei Komponenten enthält. Dadurch kann nun im Control System (siehe Abbildung 57) der „Robot control“-Block mit dem „Switch“-Block durch „connect“ verbunden werden.

Wenn ein Input bei mehreren Blöcken verwendet werden soll (siehe „qd actual“-Inputblock, Abbildung 59, S. 64), kann als Übergang ein normaler „gain“-Block von EEROS mit der Verstärkung eins definiert werden.

- **Grau, Mux und Demux:** Diese Blöcke werden ebenfalls von EEROS zur Verfügung gestellt. Es handelt sich dabei um vordefinierte Blöcke, wie sie beispielsweise auch bei Simulink vorkommen. Der Integrator-Block stellt eine Ausnahme dar, denn er wurde noch nicht von EEROS zur Verfügung gestellt.
- **Blau:** Diese Blöcke wurden neu implementiert. Bei den blauen Blöcken kann es sich um einen Subsystem-Block oder um einen Block mit einem eigenen Algorithmus handeln.
- **Rot:** Für diese Blöcke gilt dasselbe wie für die grünen Blöcke, nur dass es sich dabei um Outputs statt um Inputs handelt.

**Software-  
struktur eines  
blauen Blocks**

Die zuvor erwähnten blauen Blöcke können gemäss folgender Struktur mit EEROS implementiert werden.

<sup>5</sup> Anhang F CD in Ordner: 6\_Software\1\_OmniMoBot\src\omnimobot\control

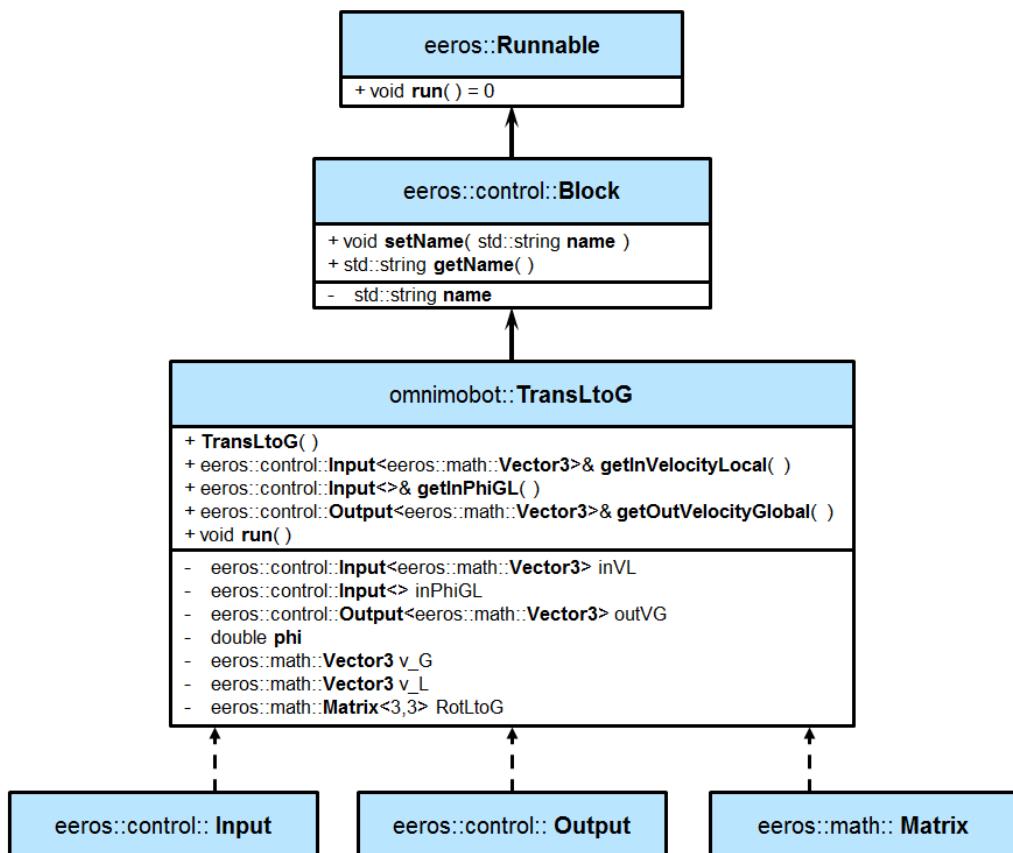


Abbildung 56: Aufbau eines Blockes mit EEROS

**Beschreibung** Die Abbildung 56 zeigt die Implementierung des Blockes „TransLtoG“. Er transformiert eine Geschwindigkeit mit einer Rotationsmatrix vom globalen KS {G} ins lokale KS {L} und entspricht dem Block „{L} to {G}“ in der Abbildung 57.

Ein auf diese Weise implementierter Block erbt üblicherweise von der Klasse „Block“ und diese wiederum erbt von der Klasse „Runnable“. Dadurch besitzt jeder Block die Methode „run“. Diese Methode wird im Control System in einer bestimmten Reihenfolge der Zeitdomäne übergeben. Die Reihenfolge ist wichtig, denn durch das Einhalten der Reihenfolge wird sichergestellt, dass die Inputs eines Blockes bereits berechnet wurden. Konkret bedeutet dies: Die „run“-Methode des Blocks 1, dessen Output der Input des Blocks 2 ist, muss vor der „run“-Methode des Blocks 2 ausgeführt werden.

Mittels der Methode „connect“ wird immer ein Input mit einem Output verbunden. Sie wird mit der Klasse „Input“ zur Verfügung gestellt. Die Klasse „Matrix“ ermöglicht ein Rechnen mit Matrizen und Vektoren.

### Implementierte Regelungs- struktur

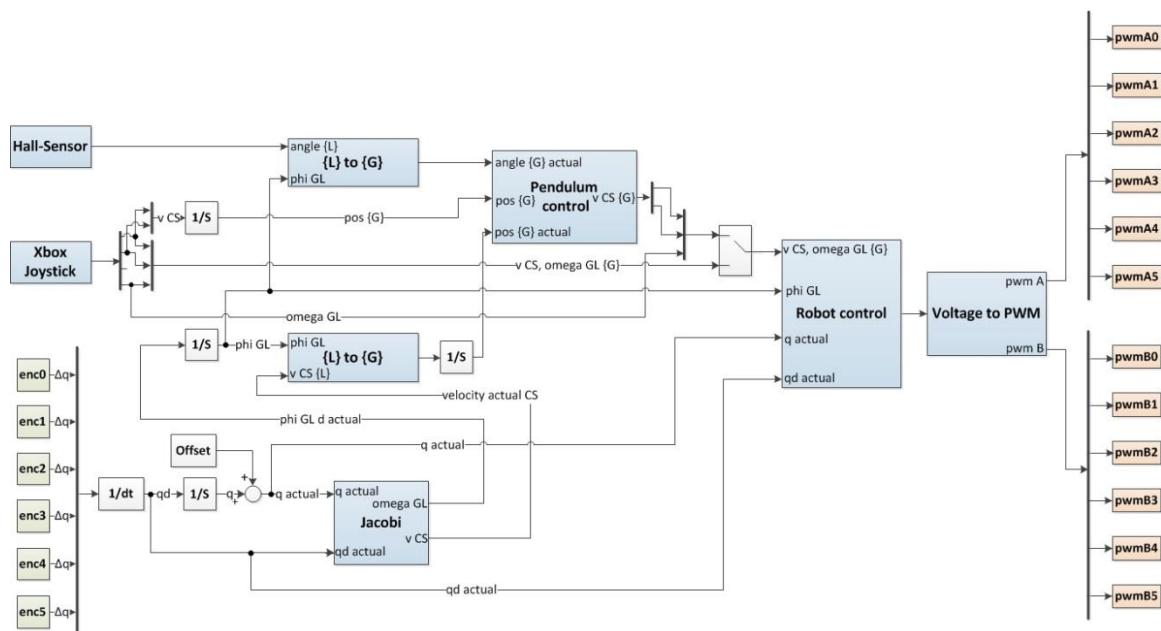


Abbildung 57: Control System OmniMoBot in C++

- Beschreibung** Die „enc“-Blöcke geben die delta-Werte der Encoder aus. Damit definieren sie zusammen mit dem „Hall-Sensor“-Block den Ist-Zustand des Systems. In den delta-Encoderwerten sind die Getriebeübersetzungen bereits berücksichtigt, somit handelt es sich dabei um die Delta-Encoderwerte am Getriebeausgang. Wenn diese Werte durch die Periodenzeit des Control Systems geteilt werden, ergibt sich daraus die Geschwindigkeit. Da mit EEROS eine Echtzeitregelung implementiert wird, ist die Periodendauer genau bekannt.  
Mit dem Homing werden die richtigen Werte für den Offset-Block bestimmt.  
Die Geschwindigkeit  $qd$  wird mit dem Jacobi-Algorithmus (siehe Kapitel 2.2.1, S. 14) in die lokale kartesische Geschwindigkeit  $v CS$  umgerechnet.  
Die Regelung des inversen Pendels erfolgt im globalen KS  $\{G\}$ . Aus diesem Grund werden alle Sensordaten in das globale KS  $\{G\}$  umgerechnet, gleich wie beim Simulink-Modell (siehe Kapitel 3.1.3, S.47).  
Anders als im Simulink-Modell werden hier für alle Transformationen von lokal  $\{L\}$  zu global  $\{G\}$  oder umgekehrt mit demselben  $\varphi_{GL}$  gerechnet.  
Der „Xbox Joystick“-Block gibt die Geschwindigkeiten in x-, y- und z-Richtung vor, der vierte Wert bestimmt die Rotationsgeschwindigkeit. Die z-Richtung wird noch nicht benötigt.  
Das Subsystem „Pendulum control“ wurde auf dieselbe Weise, wie in Kapitel 3.1.2, S. 47 beschrieben, implementiert.  
Mit dem Switchblock kann zwischen normalem, omnidirektionalem Fahren und Balancieren gewechselt werden.

- PWM-Blöcke** Die „pwmA“-Blöcke beschreiben für jeden Motor die Drehung in positiver Richtung, die „pwmB“-Blöcke sind für die negative Drehrichtung der Motoren zuständig. Somit muss darauf geachtet werden, dass pro Motor jeweils nur ein PWM aktiv und somit das andere auf null gesetzt ist. Dies wird vom „Voltage to PWM“-Block bewältigt.

#### 4.3.1.1 „Hall-Sensor“-Block

**Allgemein** Die Funktionsweise und die Kalibrierung des Hall-Sensorprints ist im Anhang D (IV, S. 110) zu finden. Die Kalibrierung des Stabwinkels  $\vec{\phi}_{xy}$  wird in Kapitel 4.5.1, S. 81 beschrieben.

##### Aufbau

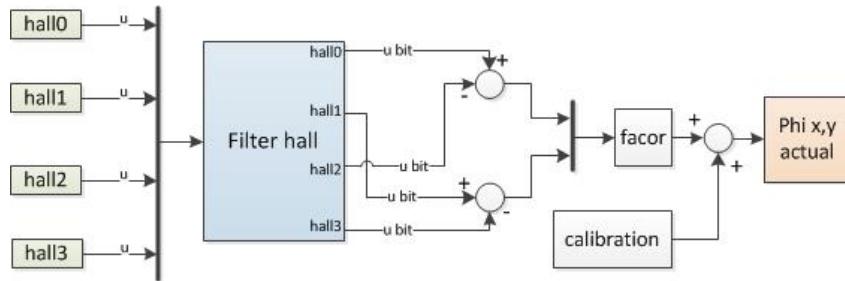


Abbildung 58: Datenverarbeitung des Hall-Sensorprints in C++

**Beschreibung** Im Block „Filter hall“ wird nicht mehr das Signal gefiltert, sondern er stellt eine Methode zur Verfügung, mit welcher abgefragt werden kann, ob der Stab vorhanden ist. Zudem werden die Kalibrierungswerte von den Hall-Sensordaten abgezogen. Der „calibration“-Block ist für die Kalibrierung des Stabwinkels  $\vec{\phi}_{xy}$  zuständig.

**Stab vorhanden?** Wenn alle Sensordaten zusammengezählt werden und der Wert 8400 [V in  $2^{12}$  Bit] niemals erreicht wird, ist kein Stab vorhanden. Zudem wurde für jeden der vier Hall-Sensoren der grösste mögliche Winkel ermittelt. Wenn einer von diesen Werten unterschritten wird, ist folglich der Winkel zu gross, um den Stab noch auszubalancieren.

#### 4.3.1.2 „Robot control“-Block und Homing der Lenkachsen

**Allgemein** Der „Robot control“-Block unterscheidet sich nur leicht vom Simulink-Modell (siehe Kapitel 3.1.3.1, S. 48). Die Transformation vom globalen KS  $\{G\}$  in das lokale KS  $\{L\}$  wird direkt in der inversen Jacobimatrix gemacht, wie es in Kapitel 2.2.2 ab S. 20 vorgestellt wird. Der „Motor model“-Block entspricht dem bereits vorgestellten Subsystem (siehe Abbildung 41, S. 49).

##### Aufbau

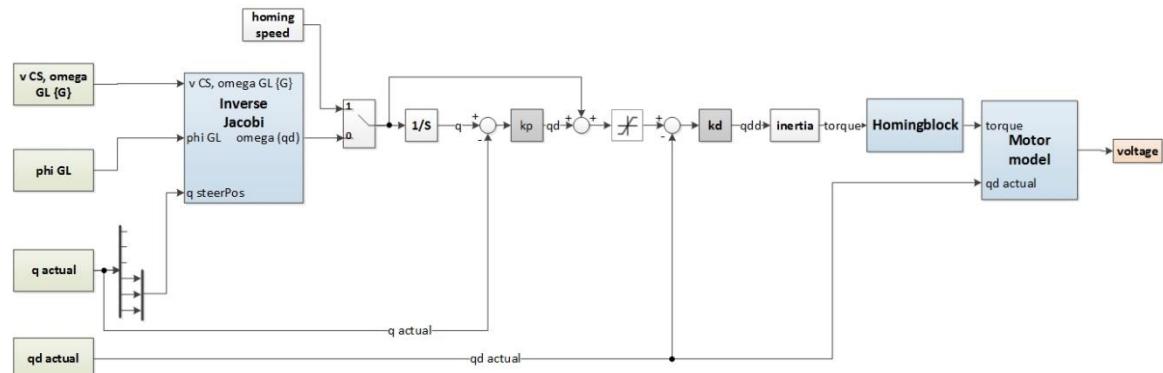


Abbildung 59: Robot control in C++

**Homing der Lenkachsen**

Die Homing Kalibrierung ist im Anhang C (S. 105) zu finden.

Das Homing erfolgt ausschliesslich im „Robot control“-Block.

Um die Position einer Lenkachse zu bestimmen, dreht sich das Schwenkrad einzeln um seine eigene Achse, bis der Homingsensor anschlägt. Die Lenkachse ist dabei positions- und geschwindigkeitsgeregt. Damit die anderen Achsen das sich drehende Schwenkrad möglichst nicht behindern, wird diesen im „Homingblock“ der Wert null vorgegeben. Der „Homingblock“ gibt das berechnete benötigte Moment weiter, bei einer Nullvorgabe werden somit die Achsen aktiv auf ein Null-Moment geregelt.

Das bedeutet, dass die durch die Drehung induzierte Spannung aktiv durch den „Motor model“-Block ausgeregelt wird. Damit ist der Widerstand, welchen die nicht aktiven Achsen dem drehenden Schwenkrad entgegenbringen, auf die Reibung reduziert.

Wenn jeweils von einem Schwenkrad, welches initialisiert wurde, auf das nächste zu initialisierende Schwenkrad gewechselt wird, müssen folgende Punkte ausgeführt werden:

- Im „Homingblock“ den Ausgang der initialisierten Achse auf null zurücksetzen.
- Dem „Offset“-Block (siehe Abbildung 57) die neuen Werte übergeben. Dabei entspricht der Wert der initialisierten Achse dem negativen Ausgang des Encoder-Integrators (siehe Abbildung 57) plus dem in Anhang C (S. 105) beschriebenen Offset. Der Offset-Wert der neu zu initialisierenden Achse wird ebenfalls auf den negativen Wert des Encoder-Integrators gesetzt.
- Der Integrator aus der Abbildung 59 muss auf null zurückgesetzt werden.
- Im „Homingblock“ den Ausgang für die neu zu initialisierende Lenkachse freischalten.

Dieser Ablauf wird im Safetysystem bei einem Levelwechsel (siehe Kapitel 4.3.2, S. 66) ausgeführt.

Nach dem Homing wird im „Switch“-Block (siehe Abbildung 59) auf den Eingang 0 gewechselt.

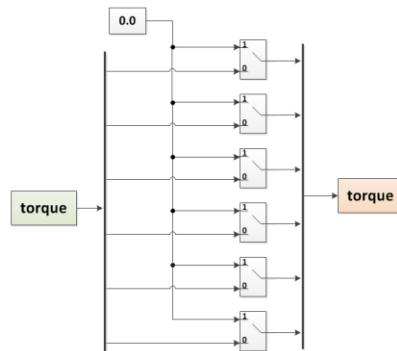
**Aufbau Homingblock**

Abbildung 60: Homingblock in C++

Dieser Block ermöglicht aus dem zuvor erwähnten Grund, dass sich der Roboter bei Bedarf mit geringem Widerstand umpositionieren lässt, wenn allen Achsen der Wert null vorgegeben wird.

#### 4.3.1.3 „Voltage to PWM“-Block

**Allgemein** Wie im Anhang A (I, S. 98) beschrieben, wird die benötigte Spannungsvorgabe für die Motoren berechnet. Für die Ansteuerung der Motoren ist ein PWM-Signal notwendig. Dieses wird über fLink generiert.

Aus der Spannungsvorgabe  $u_{in}$  und der Brückenspannung  $U_{Bridge}$  wird nun der Duty Cycle  $D$  für das PWM bestimmt:

$$D = \frac{u_{in}}{U_{Bridge}} \quad \text{mit} \quad U_{Bridge} = 24 [V] \quad (4-1)$$

Der Duty Cycle  $D$  darf nicht höher als 95% sein, da dies zu Störungen auf dem Drive Board („NTB 6xDC“) führt. Bei einem höheren Duty Cycle funktionieren die Ladungspumpen, welche in den Mosfet-Treiber-ICs eingebaut sind, nicht mehr. Die Mosfets schalten in solch einem Fall nicht mehr korrekt.

#### 4.3.2 Safety System

**Allgemein** Das Safety System läuft wie das Control System mit einer Regelungstakt-Rate  $f_{Task}$  von 400 Hz.

**Sicherheits-relevante Eingänge** Die sicherheitsrelevanten Eingänge bestehen aus zwei Tastern und einem Boolean, welcher den Status der Sicherheitsschaltung wiedergibt.

- **Approval:** Ist ein Taster, welcher den nötigen Event liefert, um vom Level „Ready“ in das Level „Drive\_Joystick“ zu wechseln.
- **Emergency Reset:** Ist der Taster, mit welchem aus dem „Emergency“-Level zum Level „Ready“ gewechselt werden kann.
- **Not-Aus:** Ist der Boolean, welcher von der Sicherheitsschaltung ausgegeben wird. Damit lässt sich überprüfen, ob die Sicherheitsschaltung freigegeben ist oder nicht. Mit diesem Signal ist gleichzeitig der Reset-Taster der Sicherheitsschaltung gekoppelt, weil der Not-Aus-Boolean erst „false“ ist, nachdem die Sicherheitsschaltung mit dem Reset-Taster freigeschaltet wurde. Dieser Zusammenhang wird als Event für das Starten des „Homing\_Rad1“-Levels verwendet.

Der Laserscanner kann ebenfalls als sicherheitsrelevanter Eingang verwendet werden. In diesem Fall werden jedoch die Daten des Laserscanners im Control System für die Kollisionsvermeidung verwendet (siehe Abbildung 69, S. 78). Wenn ein vom Laserscanner erfasstes Hindernis zu nahe ist, muss nicht notwendigerweise gleich das Level gewechselt werden. Besser ist es in diesem Fall, die Geschwindigkeitsvorgabe auf null zu setzen, denn eventuell verschwindet das Hindernis wieder.

**Sicherheits-  
relevante  
Ausgänge**

Die sicherheitsrelevanten Ausgänge sind der Watchdog, die Duty Cycle der PWMs und zwei LEDs.

- **Watchdog:** Ist für die Überwachung des Prozessors zuständig.
- **Duty Cycle:** Sobald das Safety System den Wert null ausgibt, ist es für das Control System nicht möglich, einen Duty Cycle auszugeben.
- **Emergency LED:** Das Leuchten dieser LED zeigt an, dass eine Störung vorliegt.
- **Power on LED:** Das Leuchten dieser LED zeigt an, dass die Motoren mit Strom versorgt werden.

**Safety System-  
Levels und  
Events**

Auf der folgenden Seite wird der Zusammenhang von Levels und Events dargestellt:

Aktuelles Level	Bemerkung	Sicherheitsrelevante Steuerungs-Ausgänge										Mögliche Events	Nächstes Level
		Sicherheitsrelevante Eingänge											
Nr.	Name												
100 Off	Ausgeschalteter Zustand	↑	F	F	F	F	0	0	0	0	0	0	100 Initializing
110 Initializing	Aufstart der Software	↓	T	F	F	F	0	0	0	0	0	0	200 Initialized
200 Initialized	System bereit, nicht initialisierte Position, ohne Leistung, warten auf bestätigung	↑	T	F	F	F	0	0	0	0	0	0	210 Power_on
500 ControlStopping	Controlsystem wird gestoppt	→	F	F	F	F	0	0	0	0	0	0	Do_Off
210 LowEmergency	Fehler vor dem Homing	→	F	T	F	F	0	0	0	0	0	0	Control_Stopping_done
220 Power_on	Leistung wird angelegt und Regelerstellungs-task / Homing wird gestartet	→	T	F	T	L	L	L	L	C	I	I	Do_Control_stop
300 Homing_Rad1	Zu der Lenkung_1 wird ein Offset bestimmt	→	T	F	T	L	L	L	L	C	I	I	Do_LowEmergency
310 Homing_Rad2	Zu der Lenkung_2 wird ein Offset bestimmt	→	T	F	T	L	L	L	L	C	I	I	Homing_Rad1_done
320 Homing_Rad3	Zu der Lenkung_3 wird ein Offset bestimmt	→	T	F	T	L	L	L	L	C	I	I	Do_LowEmergency
400 Emergency	Fehler nach dem Homing	→	F	T	F	0	0	0	0	I	C	I	Homing_Rad2_done
410 Ready	Roboter ist Bereit	→	T	F	T	L	L	L	L	C	I	C	420 Drive_joystick
420 Drive_joystick	Mit joystick fahren	→	T	F	T	L	L	L	L	C	I	I	440 Drive_joystick_Pendulum
440 Drive_joystick_Pendulum	Mit joystick fahren und balancieren	→	T	F	T	L	L	L	L	C	I	I	440 Emergency
													410 Ready

i = ignoriert, T = true, F = false, C = check bool, L = leave

Die Achsen 3,4,5 sind die Motoren der Lenkung (brauchen ein Homing)  
Die Achsen 0,1,2 sind die Motoren der Antriebe

Events:

- zur Reduktion des Levels ↑
- zum Erhöhen des Levels ↓

Levels:

- Hauptlevel
- Transientes Level zum hinunterfahren ↑
- Transientes Level zum hochfahren ↓

Abbildung 61: Levels und Events vom Safety System

### 4.3.3 Zu eruierende Fehler in EEROS und fLink

Allgemein	Bei Tests aufgetretene Fehler lagen zum Teil an Fehlern in der Programmierung von fLink und EEROS, was darauf zurückzuführen ist, dass diese noch in der Entwicklungsphase befinden. In solchen Fällen mussten die Fehler in fLink und EEROS eruiert werden. In diesem Kapitel werden die aufgetretenen Probleme kurz beschrieben.
fLink	Die Kommunikation zwischen fLink und dem Kernel war noch nicht blockierend implementiert. Da zwei unterschiedliche Threads gleichzeitig über fLink mit dem Kernel kommunizierten, kam es zu nicht nachvollziehbaren Fehlern. Zum Beispiel erhielten digitale Eingänge ein „true“, welches elektronisch jedoch nicht existierte.
EEROS	Die Blöcke „mux“ und „demux“, welche von EEROS für den Aufbau eines Control Systems zur Verfügung gestellt werden, gaben den Zeitstempel nicht weiter. Dies führte bei einer nachträglichen Integration, welche mit diesen Zeitstempeln arbeitet, zu Fehlern.
	Mit EEROS ist es möglich, verschiedene Zeitdomänen zu definieren. Jedoch ist es noch nicht möglich, den Zeitdomänen unterschiedliche Prioritäten zuzuweisen. Momentan besitzen alle Zeitdomänen, welche sich im Modus Echtzeit befinden, dieselbe Priorität. Dies kann zu sehr starken Jittern führen.

### 4.4 Kollisionsvermeidung durch Laserscanner „URG-04LX“

Definition	Die Kollisionsvermeidung ist unabhängig von der Bahnplanung. Mit der Kollisionsvermeidung wird überprüft, ob die vorgegebene Geschwindigkeit und Richtung des Bahnplaners oder des Joysticks zu einer Kollision mit der Umgebung führt.
Allgemein	Der OmniMoBot soll einen Stab ausbalancieren können und sich dabei zusätzlich sicher und kollisionsfrei bewegen. Um das zu erreichen, wurde eine Kollisionsvermeidung mittels Laserscanner implementiert.
	Die Kollisionsvermeidung funktioniert nach dem Prinzip, dass der OmniMoBot stehen bleibt und sich die Hindernisse um ihn bewegen. Somit wird die Bewegung der Hindernisse relativ zum OmniMoBot berechnet.
	Im Anhang D (S. 112) sind die technischen Daten des Laserscanners zu finden.

Als erstes werden im folgenden die mathematischen Zusammenhänge der Kollisionsvermeidung erläutert, anschliessend die Softwarestruktur der Kollisionsvermeidung sowie die Implementierung mit EEROS vorgestellt.

#### 4.4.1 Vektorielle Funktionsweise der Kollisionsvermeidung

Allgemein	In diesem Kapitel wird genauer auf die vektoriellen Zusammenhänge der Kollisionsvermeidung eingegangen. Die dazugehörige Matlab-Simulation ist auf der CD <sup>6</sup> zu finden. Die Kollisionsvermeidung besteht aus zwei Teilen. Der erste Teil (CollisionDetection)
-----------	---

<sup>6</sup> Anhang F CD in Ordner: 5\_Matlab\2\_Kollisionsschutz\Variante\_2\_implementierte\move, m-File: „main“

sucht aus allen Distanzen, die vom Laserscanner erfasst werden, die wichtigsten Distanzen heraus. Der zweite Teil (SafetyVelocityDesired) definiert aufgrund der wichtigsten Distanzen und der Geschwindigkeitsvorgabe, welche vom Bahnplaner ausgegeben wird, die neue Geschwindigkeitsvorgabe.

Es wird auf die mathematischen Eigenschaften der beiden Klassen eingegangen. Auch werden die nötigen Transformationen in die verschiedenen Koordinatensysteme vorgestellt. Weiter wird in diesem Kapitel die Berechnung der Hindernisse aufgezeigt.

#### 4.4.1.1 Die Klasse „CollisionDetection“

##### Allgemein

Mit der „CollisionDetection“-Klasse werden die wichtigsten Distanzen aus den Laserscannerdaten ermittelt. Die Definition der „wichtigsten“ Distanzen folgt in diesem Kapitel. Die Aufgabe dieser Klasse ist es, die Datenmenge etwas zu reduzieren, damit bei den darauf folgenden Klassen weniger Rechenleistung notwendig ist.

##### Funktionsprinzip

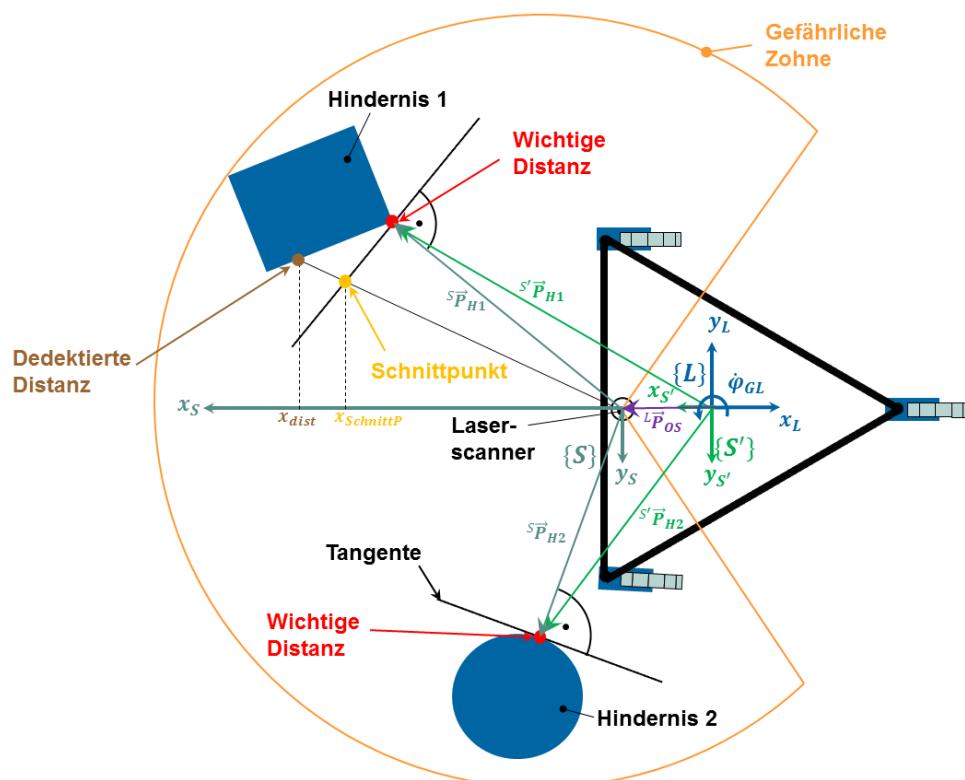


Abbildung 62: Funktionsprinzip der Klasse „CollisionDetection“

##### Bemerkung

Dieser Algorithmus wirkt im Scannerkoordinatensystem  $\{S\}$ .

Das Funktionsprinzip dieser Klasse ist sehr einfach. Es wird ein Bereich (gefährliche Zone) vorgegeben, innerhalb dessen alle Distanzen als Distanzpunkte gespeichert werden. Von diesen gespeicherten Distanzpunkten werden jedoch nur die wichtigsten an den nachfolgenden Algoritmus weiter gegeben. Um herauszufinden, welche dieser Distanzen als wichtig gelten, wird mit der geringsten Distanz zum Hindernis begonnen. Von dieser wird die Tangente berechnet. Anschliessend wird von jeder gespeicherten Distanz der Schnittpunkt mit der berechneten Tangente ermittelt. Nun kann der x-Wert des gespeicherten Distanzpunktes mit dem x-Wert des Schnittpunktes verglichen werden. Wenn der x-Wert des Schnittpunkts grösser ist als der x-Wert des

gespeicherten Distanzpunktes, wird dieser Distanzpunkt ebenfalls zu einer wichtigen Distanz. Dasselbe Prinzip funktioniert auch mit dem y- statt dem x-Wert.

**Mathematisch** Eine Distanz kann mittels einer Geradengleichung ausgedrückt werden:

$$y_d = m_d \cdot x_d + q_d \quad (4-2)$$

Die Steigung  $m_d$  ist der Tangens des Winkels  $\alpha_d$  der jeweiligen Distanz. Da die Gerade durch den Ursprung geht, ist  $q_d = 0$ . Die Steigung der Tangente  $m_t$  ist der negative reziproke Wert der Steigung  $m_d$ :

$$m_t = -\frac{1}{\tan(\alpha_d)} \quad (4-3)$$

Durch das Einsetzen der Werte  $y_d$  und  $x_d$  des Distanzpunktes, von welchem die Tangente berechnet wird, kann der noch fehlende Faktor  $q_t$  der Geradengleichung der Tangente ermittelt werden:

$$q_t = -m_t \cdot x_d + y_d \quad (4-4)$$

Der x-Wert des Schnittpunktes  $x_{SchnittP}$  der Tangente und einer zu untersuchenden Distanz wird durch Gleichsetzen der beiden Geradengleichungen ermittelt:

$$x_{SchnittP} = \frac{q_t}{m_d - m_t} \quad (4-5)$$

### Ergebnis

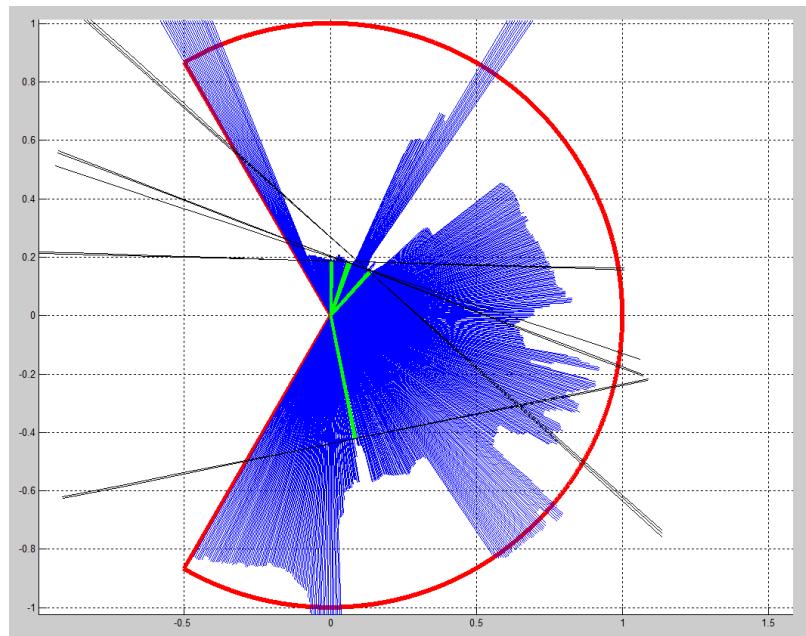


Abbildung 63: Ergebnis der Klasse „CollisionDetection“

Die Abbildung 63 zeigt in Blau alle Laserscannerdaten und in Grün die durch die Klasse CollisionDetection ermittelten „wichtigen“ Distanzen.

- Transformation** Die ermittelten Positionsdaten müssen nun transformiert werden, damit sie in der Klasse „SafetyVelocityDesired“ richtig verwendet werden. Die Klasse „SafetyVelocityDesired“ wirkt im Scannerkoordinatensystem  $\{S'\}$  statt im Scannerkoordinatensystem  $\{S\}$ , was einer Translation um den Vektor  ${}^{S'}\vec{P}_{OS}$  gleichkommt. (siehe Abbildung 62):

$${}^{S'}\vec{P}_H = {}^S\vec{P}_H + {}^{S'}\vec{P}_{OS} \quad (4-6)$$

- ${}^{S'}\vec{P}_{OS}$ : Der Betrag dieses Vektors ist 0.2075 Meter.

#### 4.4.1.2 Bewegung der Hindernisse

- Allgemein** Dem Laserscanner ist es nicht möglich, bei einer Takt-Rate  $f_{Task}$  von 400 Hz neue Scandaten zu senden. Somit muss während der Zeit, in welcher der Laserscanner keine neuen Positionsdaten der Hindernisse sendet, die Bewegung der Hindernisse berechnet werden. Da die Übertragungsdauer des Laserscanners ca. 150 ms beträgt (siehe Kapitel 4.4.2.1, ab S. 76) sind die „neuen“ Positionsdaten bei der Ankunft im Prozessor bereits wieder veraltet. Aus diesem Grund müssen die „neuen“ Positionsdaten mit der währenddessen erfolgten Positionsverschiebung des Roboters ergänzt werden.
- Mathematisch** Die neue Position  ${}^{S'}\vec{P}_{H\_neu}$  des Hindernisses wird folgendermassen definiert: Letzte Position  ${}^{S'}\vec{P}_{H\_alt}$  abzüglich der gefahrenen Geschwindigkeit  ${}^{S'}\vec{v}_{ist}$  multipliziert mit der Zeitstempeldifferenz  $dt$ .

$${}^{S'}\vec{P}_{H\_neu} = {}^{S'}\vec{P}_{H\_alt} - {}^{S'}\vec{v}_{ist} \cdot dt \quad (4-7)$$

Mit einer zusätzliche Rotation  $\phi_{GL}$  des OmniMoBots um die eigene Achse ist die neue Position  ${}^{S'}\vec{P}_{H\_neu\_phi}$  des Hindernisses wie folgt definiert:

$$\begin{bmatrix} {}^{S'}P_{H\_neu\_phi,x} \\ {}^{S'}P_{H\_neu\_phi,y} \end{bmatrix} = \begin{bmatrix} \cos(\phi_{GL} \cdot dt) & \sin(\phi_{GL} \cdot dt) \\ -\sin(\phi_{GL} \cdot dt) & \cos(\phi_{GL} \cdot dt) \end{bmatrix} \cdot \begin{bmatrix} {}^{S'}P_{H\_neu,x} \\ {}^{S'}P_{H\_neu,y} \end{bmatrix} \quad (4-8)$$

Aus der neuen Position  ${}^{S'}\vec{P}_{H\_neu\_phi}$  können nun wiederum die Distanz sowie der Winkel des Hindernisses neu berechnet werden.

#### 4.4.1.3 Die Klasse „SafetyVelocityDesired“

- Anforderung** Mit der Klasse „SafetyVelocityDesired“ wird aus der Geschwindigkeitsvorgabe des Bahnplaners und den wichtigen Distanzdaten eine neue, sichere Geschwindigkeitsvorgabe erstellt, mit welcher keine Kollisionen auftreten. Die Anforderung an diese Klasse ist, die Befehle des Bedieners oder die Anweisungen des Bahnplaners möglichst genau umzusetzen und dabei die Geschwindigkeit so hoch wie möglich zu halten. Zusätzlich darf der Roboter mit keinem Hindernis kollidieren.

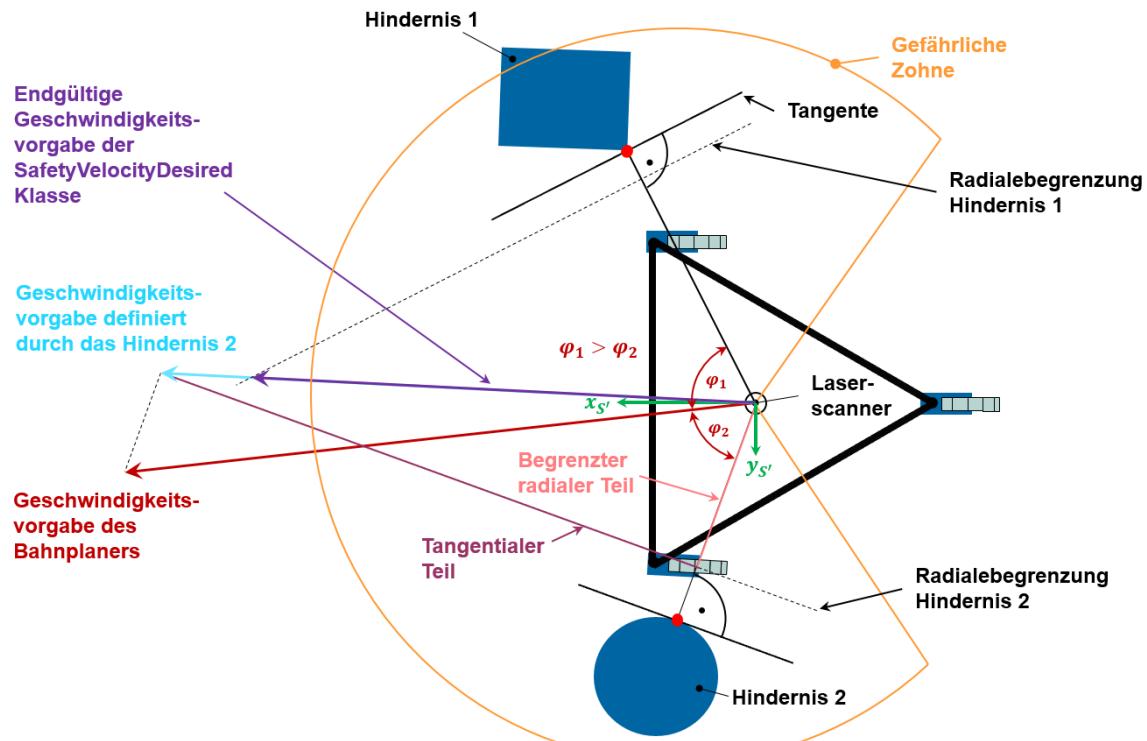
**Funktions-prinzip**

Abbildung 64: Funktionsprinzip der „SafetyVelocityDesired“ Klasse

**Bemerkung**

Der Algorithmus ist im Scannerkoordinatensystem  $\{S'\}$  definiert.

Der durch den Bahnplaner vorgegebene Geschwindigkeitsvektor wird an jede detektierte Distanz projiziert. Dabei entsteht ein tangentialer- und ein radialer Anteil des Geschwindigkeitsvektors für jedes Hindernis. Der radialem Anteil des Geschwindigkeitsvektors wird nun begrenzt. Diese Begrenzung wird aufgrund der aktuellen Distanz zum Hindernis ermittelt. Damit die vorgegebene Geschwindigkeit des Bahnplaners möglichst nicht durch mehrere Hindernisse verfälscht wird, wird immer als erstes mit dem Hindernis begonnen, welches den kleinsten Winkel zum Vektor der Geschwindigkeitsvorgabe einnimmt (siehe Hindernis 2 in Abbildung 64). Wenn dann ein zweites Hindernis (Hindernis 2) die bereits reduzierte Geschwindigkeitsvorgabe beeinflusst, wird dieses Hindernis ebenfalls in die Berechnung der sicheren Geschwindigkeitsvorgabe mit einbezogen.

**Radiale Begrenzung**

Es werden zwei Möglichkeiten zur Begrenzung des radialen Teils des Geschwindigkeitsvorgabe-Vektors vorgestellt. Die eine ist eine proportionale Begrenzung, die andere ist eine Begrenzung, welche durch die maximale negative Beschleunigung definiert ist.

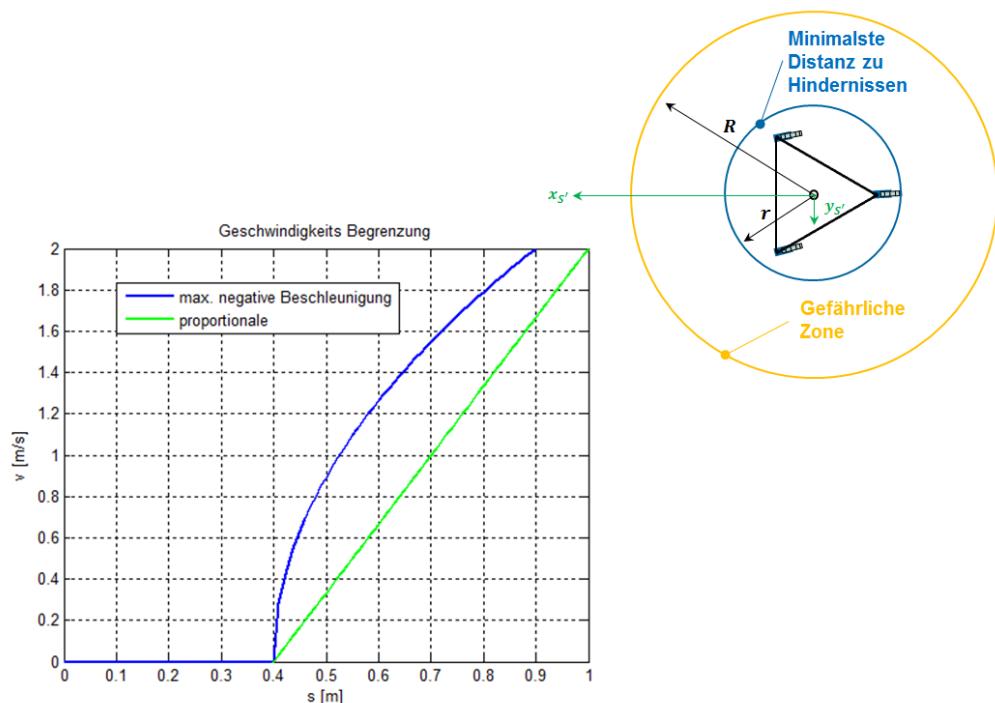


Abbildung 65: Radiale Begrenzung der Geschwindigkeit

- $r$ : Ist die minimalste Distanz zu den Hindernissen ( $r = 0.4[m]$ ).
- $R$ : Ist der Radius der gefährlichen Zone ( $R = 1[m]$ ).
- $a_{max}$ : Ist die maximale negative Beschleunigung ( $a_{max} = 4 \frac{m}{s^2}$ ).
- $V_{max}$ : Ist die maximale Geschwindigkeit des OmniMoBots ( $V_{max} = 2 \frac{m}{s}$ ).

Die Funktion der proportionalen Begrenzung wird durch die maximale Geschwindigkeit wie folgt definiert:

$$v = \frac{V_{max} \cdot s}{R - r} - \frac{V_{max} \cdot r}{R - r} \quad (4-9)$$

Um den Einfluss von Hindernissen auf den Geschwindigkeitsverlauf zu reduzieren, ist die Verwendung einer Wurzelfunktion sinnvoll. Eine solche kann mittels der maximalen negativen Beschleunigung erreicht werden:

$$v = \sqrt{2 \cdot a_{max} \cdot (s - r)} \quad (4-10)$$

#### Simulations- ergebnis Beispiel 1

Dieses Beispiel zeigt, wie der Roboter durch die Kollisionsvermeidung selbständig um ein Hindernis fährt, obwohl sich die Gewindigkeitsvorgabe des Bahnplaners nicht verändert.

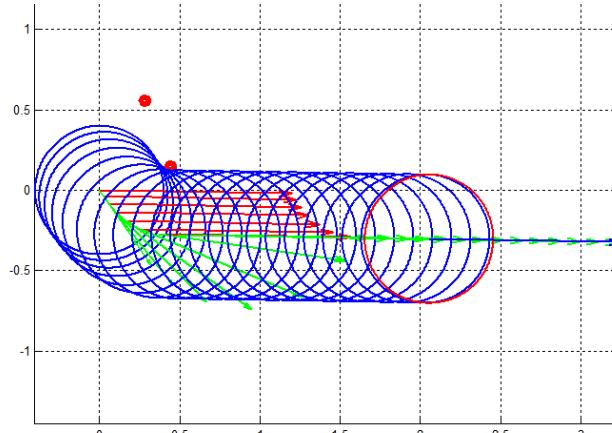


Abbildung 66: Beispiel 1 der „SafetyVelocityDesired“ Klasse

Der blaue Kreis stellt den Roboter dar. Die roten Punkte symbolisieren die detektierten Hindernisse. Der rote Vektor steht für die Geschwindigkeitsvorgabe durch den Bahnplaner, der Grüne für die sichere Geschwindigkeitsvorgabe (mit Ausnahme der letzten Vorgabe, welche blau ist).

#### Simulations- ergebnis Beispiel 2

Dieses Beispiel zeigt, dass der Roboter nicht weiter fährt, wenn weitere Hindernisse im Weg stehen.

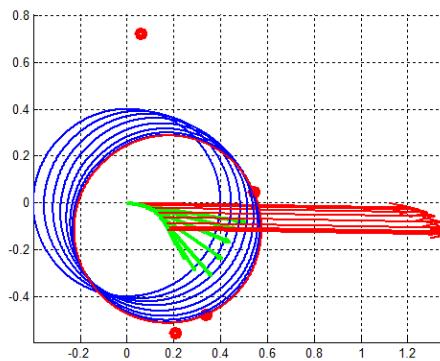


Abbildung 67: Beispiel 2 der „SafetyVelocityDesired“ Klasse

#### 4.4.1.4 Transformationen

<b>Transformation KS <math>\{L\}</math> zu KS <math>\{S'\}</math> (omni- direktionales Fahren)</b>	Beim omnidirektionalen Fahrverhalten muss lediglich die Geschwindigkeitsvorgabe, welche im lokalen KS $\{L\}$ vorgegeben ist, in das Scannerkoordinatensystem $\{S'\}$ transformiert werden. Dadurch ist es der „SafetyVelocityDesired“ Klasse möglich, mit dieser Vorgabe zu rechnen. Da eine Geschwindigkeit um einen fixen Winkel $\varphi_{LS}$ transformiert wird, ist eine orthogonale Drehmatrix ${}^{S'}R_L$ mit dem Winkel $\varphi_{LS}$ notwendig. ( $\varphi_{LS} = \pi [rad]$ )
--	---

$$\begin{bmatrix} {}^{S'}v_{GL,x} \\ {}^{S'}v_{GL,y} \end{bmatrix} = \begin{bmatrix} \cos(\varphi_{LS}) & \sin(\varphi_{LS}) \\ -\sin(\varphi_{LS}) & \cos(\varphi_{LS}) \end{bmatrix} \cdot \begin{bmatrix} {}^L v_{GL,x} \\ {}^L v_{GL,y} \end{bmatrix} \quad (4-11)$$

Da der Winkel  $\varphi_{LS} = \pi$  ist, kann  ${}^{S'}R_L$  auch als Rücktransformation verwendet werden.

- Transformation KS  $\{L\}$  zu KS  $\{S'\}$  (Bewegung in KS  $\{L\}$  beschrieben)** Wenn eine Bewegung im lokalen KS  $\{L\}$  beschrieben ist (siehe Kapitel 2.3, S. 22), muss die Drehung um den Momentanpol ebenfalls in die Transformationsmatrix  ${}^{S'}R_L$  mit einbezogen werden. Der Grund dafür ist die Überlagerung der Drehung mit der Geschwindigkeitsvorgabe.

$$\begin{bmatrix} {}^{S'}v_{GL,x} \\ {}^{S'}v_{GL,y} \end{bmatrix} = \begin{bmatrix} \cos(\varphi_{LS} + \varphi_{GL}) & \sin(\varphi_{LS} + \varphi_{GL}) \\ -\sin(\varphi_{LS} + \varphi_{GL}) & \cos(\varphi_{LS} + \varphi_{GL}) \end{bmatrix} \cdot \begin{bmatrix} {}^L v_{GL,x} \\ {}^L v_{GL,y} \end{bmatrix} \quad (4-12)$$

Für die Rücktransformation kann  ${}^{S'}R_L^T$  verwendet werden.

- Transformation KS  $\{G\}$  zu KS  $\{S'\}$**  Diese Transformation entspricht der Gleichung (4-12), nur dass statt der Geschwindigkeit  ${}^L\vec{v}_{GL}$  die im globalen KS  $\{G\}$  ausgedrückte Geschwindigkeit  ${}^G\vec{v}_{GL}$  verwendet wird.

## 4.4.2 Softwarestruktur der Kollisionsvermeidung

- Allgemein** In diesem Kapitel werden die Softwarestruktur der „LaserScanner“-Klasse sowie die Softwarestruktur der Kollisionsvermeidung vorgestellt.

### 4.4.2.1 Die Klasse „LaserScanner“

- Vorgaben** Der Klasse „LaserScanner“ kann ein Bereich in Steps vorgegeben werden. Mit dieser Step-Vorgabe wird definiert, welcher Abschnitt des 240°-Bereichs gemessen werden soll. Im Rahmen dieser Arbeit wird immer der ganze Bereich abgefragt, das heisst der Bereich vom Startstep 44 bis zum Endstep 725.

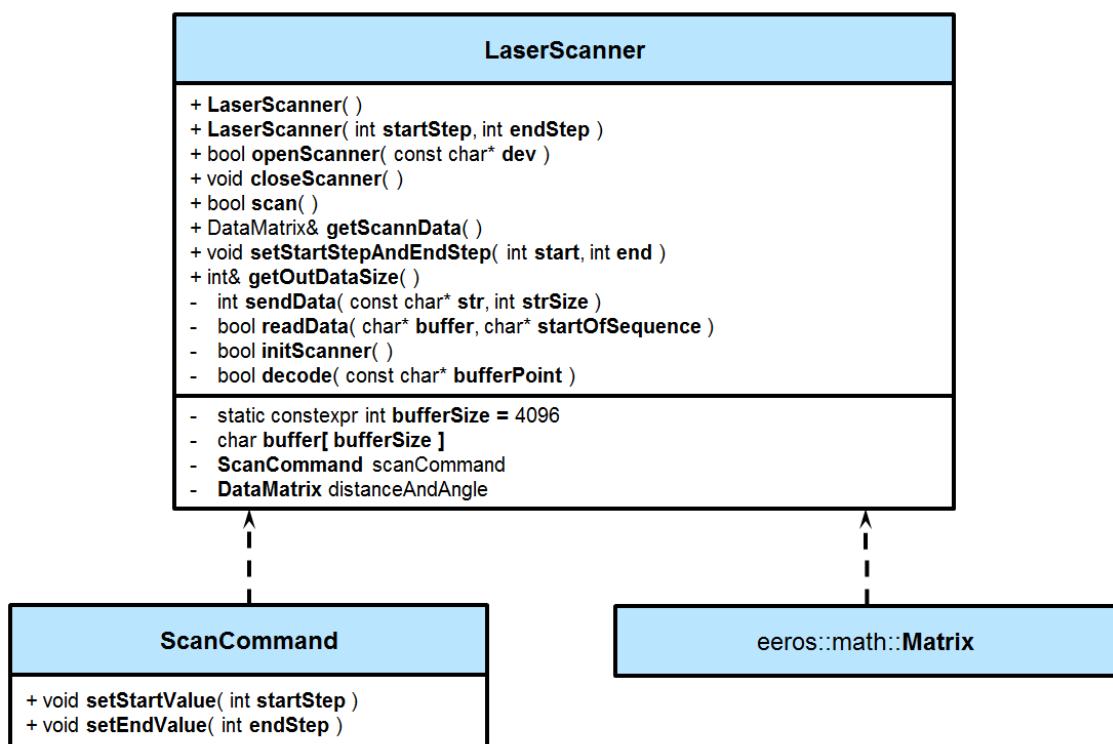
**Struktur**

Abbildung 68: Softwarestruktur der Laserscanner-Datengewinnung

**Bemerkung**

Ein Objekt der Klasse „LaserScanner“ erzeugt je eine Instanz der Klassen „ScanCommand“ und „Matrix“. Die Klasse Matrix wird von EROS zur Verfügung gestellt. Die „DataMatrix“-Definition entstand durch „typedef“, auf diese Weise wird der Instanz „Matrix“ gleich eine Grösse zugewiesen.

Mit dem Objekt der „LaserScanner“-Klasse kann die Methode „scan“ aufgerufen werden. Diese Methode managed den ganzen Laserscannerdaten-Gewinnungsablauf. Dabei wird als erstes die „sendData“-Methode aufgerufen, welcher zuvor ein Pointer auf den String übergeben wird, welcher vom Objekt „scanCommand“ vorbereitet wurde. Anschliessend wird die „readData“-Methode von der „scan“-Methode aufgerufen, diese füllt den Buffer mit den empfangenen Rohdaten des Laserscanners. Zuletzt ruft die Methode „scan“ die Methode „decode“ auf. Diese decodiert die Rohdaten des Laserscanners und speichert die Resultate in die „distanceAndAngle“-Matrix. Der Methode „decode“ muss dafür ein Pointer auf den Buffer der Rohdaten übergeben werden.

**Übertragungszeit**

Die Übertragungszeit ist die Zeit, welche benötigt wird, um die Daten des Laserscanners über die RS232-Schnittstelle auf das Mainboard zu übertragen. Die Übertragungszeit der Laserscannerdaten wird durch die Zeichencodierung und durch den abgefragten Scanbereich definiert. Eine gemessene Distanz wird mittels Zeichencodierung durch **drei „char“ Einheiten** ausgedrückt. Da eine „char“ Einheit nur 1 Byte und nicht wie beispielsweise ein „double“ 8 Bytes benötigt, ist es möglich, die Grösse des Datenpakets möglichst klein zu halten. Wie das Decodieren funktioniert, kann dem Datenblatt<sup>7</sup> entnommen werden.

Da jeweils immer der ganze Bereich abgefragt wird, lässt sich die Grösse eines Datenpakets mittels des Datenblatts genau definieren. Die Distanzen werden vom

<sup>7</sup> Anhang F CD in Ordner: 4\_Elektronik\2\_Datenblätter\Laserscanner, PDF-File: „URG SCIP20“

Laserscanner in Blöcken ausgegeben:

- Anzahl abgefragte Distanzen in Byte:  $n_D = (725 - 44 + 1) \cdot 3$  [Byte]
- Anzahl Bytes pro Block:  $n_{D/Block} = 64$  [Byte]
- Anzahl Datenblöcke:  $n_B = \frac{n_D}{n_{D/Block}}$  [-]
- Anzahl Bytes bevor die Blöcke ausgegeben werden:  $n_V = 47$  [Byte]

Nach jedem Block werden zwei Bytes gesendet. Die Gesamtgrösse  $n_{tot}$  ist somit wie folgt definiert:

$$n_{tot} = n_B \cdot 2 \text{ [Byte]} + n_D + n_V = 2157 \text{ [Byte]} \quad (4-13)$$

Die Baudrate wird auf 115200 Bit pro Sekunde gesetzt. Dadurch ergibt sich eine Übertragungszeit von ca. 150 Millisekunden.

#### 4.4.2.2 Control System mit Kollisionsvermeidung

##### Allgemein

Im Kapitel 4.3.1 auf S. 60 wird das Control System des OmniMoBots vorgestellt. Dieses Control System wird in diesem Kapitel noch um ein paar Blöcke ergänzt. Weiter wird ein zusätzliches Control System vorgestellt. Es wird „LaserScannerContol“ genannt. Es besitzt wie das in Kapitel 4.3.1 vorgestellte Control System eine eigene Zeitdomäne. Jede Zeidomäne entspricht einem eigenen Thread.

##### Implementierte Struktur

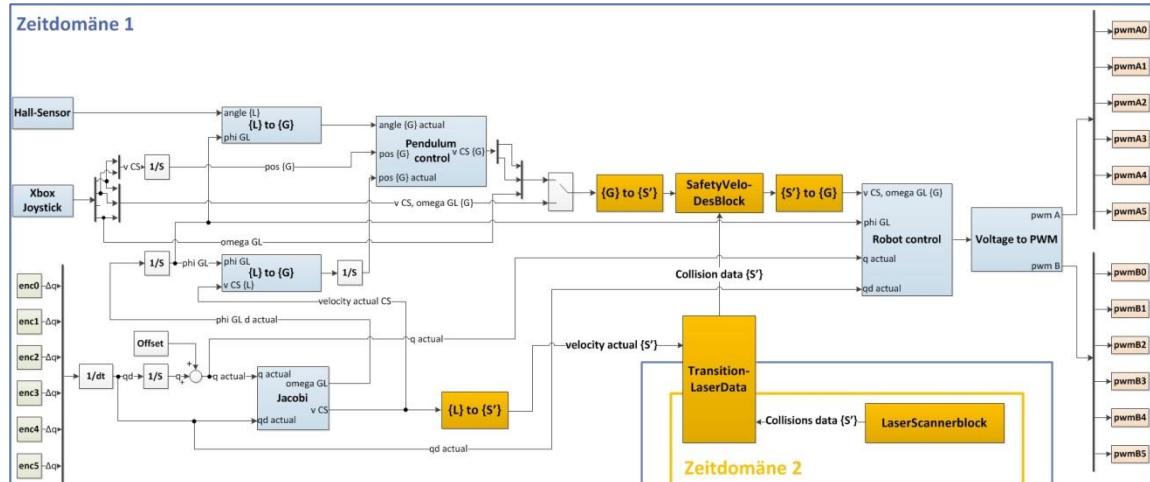


Abbildung 69: Control System mit Kollisionsvermeidung

##### Beschreibung

Neu gegenüber dem Control System von Kapitel 4.3.1 sind die in Abbildung 69 orange gefärbten Blöcke. Es sind zwei Zeitdomänen zu sehen, welche mit einem Transitionsblock miteinander verbunden sind. Der Output des „LaserScannerblocks“ ist eine Matrix, welche alle wichtigen Kollisionsdaten enthält. In dem Transitionsblock werden mittels der aktuellen Geschwindigkeit die vom „LaserScannerblock“ erhaltenen Kollisionsdaten verarbeitet. Die Kollisionsdaten sind immer im Scannerkoordinatensystem  $\{S'\}$  (siehe Abbildung 62, S. 70) beschrieben, deshalb sind die zusätzlichen Transformationsblöcke notwendig. Der „SafetyVeloDesBlock“ gibt einen Geschwindigkeitsvektor vor, mit welchem eine kollisionsfreie Bewegung gewährleistet ist.

**LaserScanner-block** Der „LaserScannerblock“ aus Abbildung 69 hat folgende Softwarestruktur:

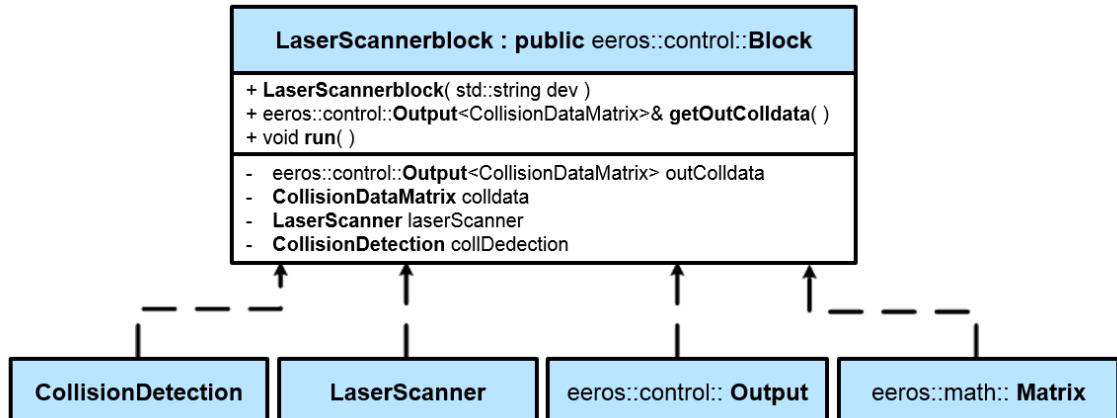


Abbildung 70: Struktur des „LaserScannerblocks“

**Beschreibung** Der „LaserScannerblock“ inkludiert die zwei Klassen „Output“ und „Matrix“ von EROS, sowie die „LaserScanner“- und die „CollisionsDetection“-Klasse. Damit lassen sich mit diesem Block die „wichtigsten“ Distanzen in Form einer „CollisionsDataMatrix“ an den „TransitionLaserData“-Block übergeben.

**Transition-LaserData** Der „TransitionLaserData“-Block aus Abbildung 69 ist wie folgt implementiert:

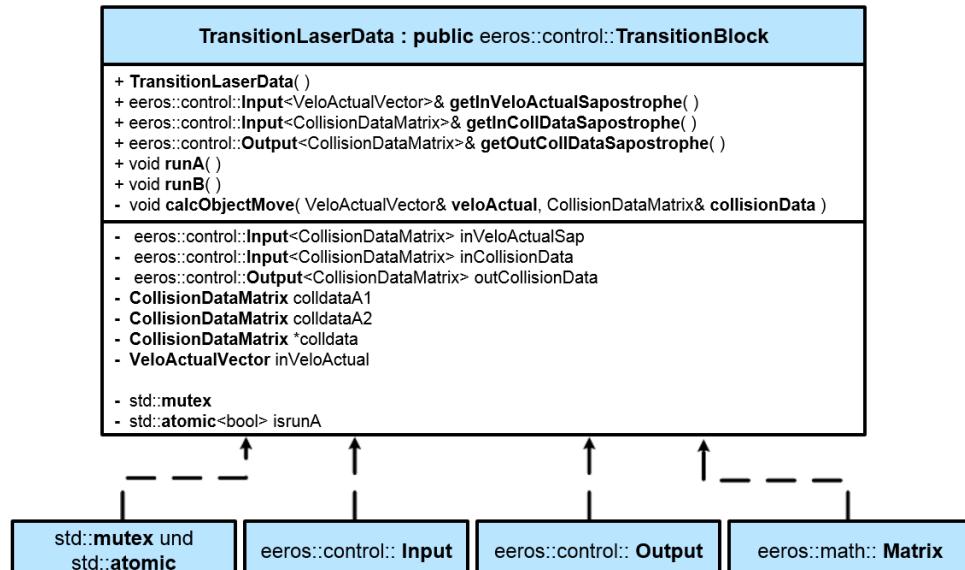


Abbildung 71: Struktur des „TransitionLaserData“-Blocks

**Beschreibung** Die Aufgabe des „TransitionLaserData“-Blocks ist es, die Kollisionsdaten von möglichen Hindernissen über zwei Zeitdomänen zu handhaben. Die Zeitdomäne 1 hat eine Takt-Rate  $f_{Task}$  von 400 Hz, was einer Periode von 2.5 ms entspricht. Die Periode der Zeitdomäne 2 beträgt 170 ms. Die Klasse „TransitionBlock“, welche von EROS zur Verfügung gestellt wird, gibt zwei „run“-Methoden vor. Die „runA“-Methode wird in der Zeitdomäne 2 ausgeführt, die „runB“-Methode in der Zeitdomäne 1.

Der „TransitionLaserData“-Block muss gewährleisten, dass die Kollisionsdaten nicht gleichzeitig ausgelesen und beschrieben werden. Dies wird mit „mutex“, mit der „atomic“-Variable „isrunA“ und dem „double buffering“-Prinzip erreicht.

Sobald die „runA“-Methode ausgeführt wird, wird der „mutex“ gelockt. Während dieser Zeit ist es dem zweiten Thread (Zeitdomäne 1) nicht möglich, ebenfalls in den gelockten Bereich zu gelangen. Dieser wartet währenddessen vor seinen „lock“-Bereich, bis die „runA“-Methode den „mutex“ wieder freigibt. Im gelockten Bereich in der „runA“-Methode wird dem Pointer „colldata“ die aktuelle Adresse der Kollisionsmatrix übergeben und die „atomic“-Variable „isrunA“ wird auf „true“ gesetzt. Im gelockten Bereich der „runB“-Methode wird die „isrunA“-Variable wieder auf „false“ gesetzt. Die „atomic“ Variable „isrunA“ kann nicht gleichzeitig gelesen und beschrieben werden. Durch den „mutex“ und die Eigenschaft von „atomic“ ist sichergestellt, dass neue Kollisionsdaten vorhanden sind, wenn die „isrunA“-Variable auf „true“ gesetzt wird. Somit ist mit dieser Variable bestimmt, ob die „runB“-Methode die Position der Hindernisse aus den alten Kollisionsdaten berechnen muss oder ob sie die neuen Kollisionsdaten verwenden kann.

Die Kollisionsdatenmatrix wird mittels „double buffering“ beschrieben. Das bedeutet, es sind zwei Buffer „colldataA1“ und „colldataA2“ vorhanden. In der „runA“-Methode wird dem „colldata“-Pointer die Adresse jenes Buffers übergeben, welche nicht bereits im vorangegangenen Durchlauf übergeben wurde. Dadurch ist sichergestellt, dass der Thread 2 immer denjenigen Buffer neu beschreibt, welcher nicht gerade für die Berechnung der Hindernisse im Thread 1 verwendet wird.

## 4.5 Ergebnisse

**Allgemein** Die Notwendigkeit einer Kalibrierung des Stabwinkels  $\phi$  (siehe *Stabwinkel  $\phi$  Abbildung 25, S. 36*) wird aufgezeigt. Weiter wird das Ausbalancieren des Stabes untersucht und ein Fazit über die Fähigkeiten des OmniMoBot gezogen.

Wie bereits im Kapitel Simulation (*Kapitel 3, S. 45*) werden in diesem Kapitel zur Veranschaulichung die Position der Stabspitze und der Verlauf der Geschwindigkeit des OmniMoBots in x-Richtung dargestellt. Die dazu notwendigen Daten wurden während dem Betrieb mit einem „MeasuringDataBlock“ aufgezeichnet und am Ende des „main“-Programms in ein File geschrieben. Die Plots wurden mit Matlab erstellt.

Der Stab lässt sich mit dem OmniMoBot unterschiedlich gut ausbalancieren, auch wenn keine Änderungen an der Hardware oder Software durchgeführt wurden. Die möglichen Gründe werden in diesem Kapitel behandelt.

**Regelungsparameter** Die Regelungsparameter sind wie folgt definiert:

- **Spitzenregelung**

Eigenfrequenz in Hz  $f_0$ : 0.196 Hz, Dämpfungsgrad  $D$ : 0.7

Aus den Gleichungen ( 2-35 ) und ( 2-37 ) folgt:

→  $k_p = 1.5125$ ,  $k_d = 0.77$  ( $k_d$  wurde zusätzlich durch den Faktor  $\sqrt{5}$  dividiert)

- **Winkelregelung**

Eigenfrequenz in Hz  $f_0$ : 1.194 Hz, Dämpfungsgrad  $D$ : 0.7

Aus den Gleichungen ( 2-35 ) und ( 2-37 ) folgt:

→  $k_p = 56.25$ ,  $k_d = 5.25$  ( $k_d$  wurde zusätzlich durch den Faktor 2 dividiert)

- „Robot control“

Eigenfrequenz in Hz  $f_0$ : 11.368 Hz, Dämpfungsgrad  $D$ : 0.7

Aus den Gleichungen ( 2-37 ) und ( 2-39 ) folgt:

➔  $\tilde{k}_p = 51.02$ ,  $k_v = 100$  ( $k_v$  entspricht in Abbildung 59,  $k_d$ )

Die Regelungsparameter bleiben bei den nachfolgenden Abbildungen stets dieselben.

#### 4.5.1 Kalibrierung des Stabwinkels

<b>Problem- beschreibung</b>	Der Roboter fährt, sobald er zu balancieren beginnt, eine gewisse Distanz in eine Richtung und balanciert auf der neuen Position den Stab weiter aus.
----------------------------------	---

**Offset beim  
Start des  
Balancier-  
modus**

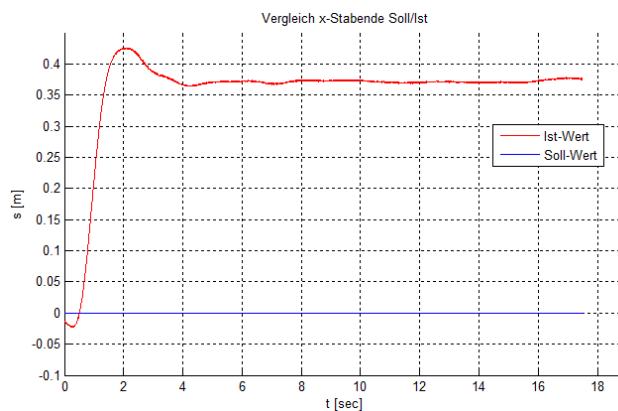


Abbildung 72: Offset beim Start des Balanciermodus

**Beschreibung** Abbildung 72 stellt die gemessene Anfangsbewegung anhand der Stabspitzenposition in x-Richtung dar. Die Anfangsbewegung in y-Richtung war sogar doppelt so gross. Es ist zu erkennen, dass der Roboter gleich nach dem Start auf die Position 0.357 m zufährt, obwohl die Position null vorgegeben wurde. Diese konstante Regelungsabweichung hält der Roboter auch bei nachfolgenden Positionsänderungen aufrecht. Des weiteren ist aus dieser Abbildung ersichtlich, wie gut der OmniMoBots den Stab an Ort und Stelle ausbalanciert. Bei der Position 0.357 m ist die Bewegung des Stabes noch minimal.

**Offset in  
Winkel-  
messung**

Die Abbildung 73 stellt eine Messung des Stabwinkels  $\phi_x$  dar.

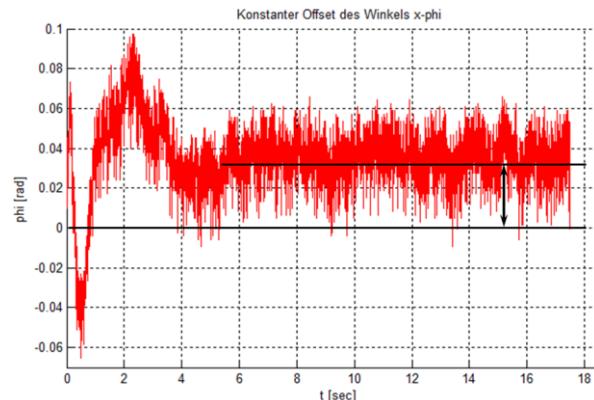


Abbildung 73: Offset des eingependelten Stabwinkels  $\phi_x$

**Beschreibung** Während der in Abbildung 73 dargestellten Messung verhielt sich der OmniMoBot unruhig, was dem Stabwinkelverlauf anzusehen ist (*mehr dazu folgt in Kapitel 4.5.2.3*). Weiter veranschaulicht die Abbildung 73, dass der Stabwinkel  $\phi_x$  auch bei eingependeltem Zustand eine konstante Abweichung von ca. 0.03 rad aufweist. Beim ausgependelten Zustand müsste sich der Stabwinkel  $\phi_x$  um null rad herum bewegen. Aufgrund dieser Abweichung geht die Regelung des Stabes fälschlicherweise davon aus, dass der Stab schief ist, obwohl er in Wirklichkeit gerade steht. Dadurch wird die Geschwindigkeitsvorgabe, welche aus der Integration der Beschleunigung  $\ddot{x}_{soll}$  resultiert (siehe Abbildung 27, S. 38), nicht korrekt berechnet. Dies führt zu einer bleibenden Regelungsabweichung.

**Kalibrierung des Stabwinkels** Die Kalibrierung des Stabwinkels  $\vec{\phi}_{xy}$  wird im „Hall-Sensor“-Block umgesetzt (siehe Kapitel 4.3.1.1, S. 64).

Das beste Resultat konnte mit einem Offset von 0.024 rad für  $\phi_y$  und keinem Offset für  $\phi_x$  erreicht werden.

Das Problem kann nicht vollumfänglich mit der Kalibrierung des Stabwinkels  $\vec{\phi}_{xy}$  gelöst werden. Der Offset, welcher beim Start des Balanciermodus auftritt, konnte mit Hilfe der Kalibrierung von ca. 0.8 m auf 0.15 m reduziert werden.

## 4.5.2 Ausbalancieren des Stabes

**Allgemein** Wenn der OmniMoBot in einer unruhigen Phase den Stab balanciert, wird der Stab dennoch zuverlässig und ausdauernd balanciert, jedoch etwas weniger straff im Vergleich zu einer ruhigen Phase. Während dem Balancieren wechselt er niemals von einer ruhigen in eine unruhige Phase oder umgekehrt. Die Phase bestimmt sich zu Beginn des Balanciermodus.

### 4.5.2.1 Ausbalancieren einer Störung

**Ergebnis Stabspitzenposition**

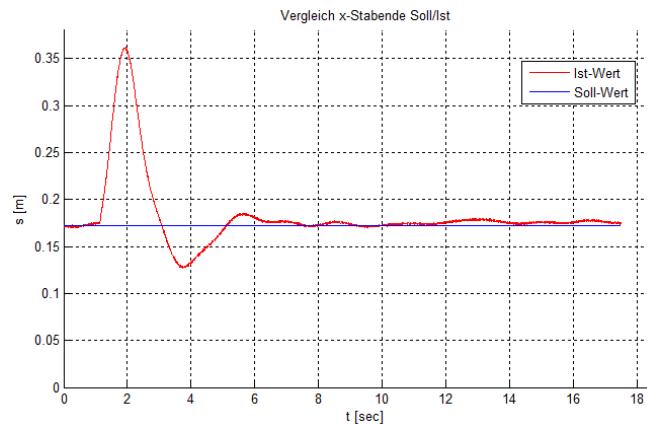
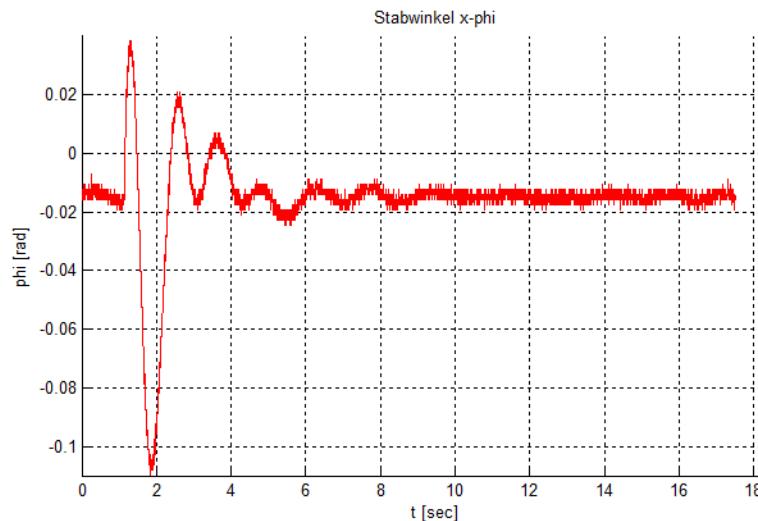
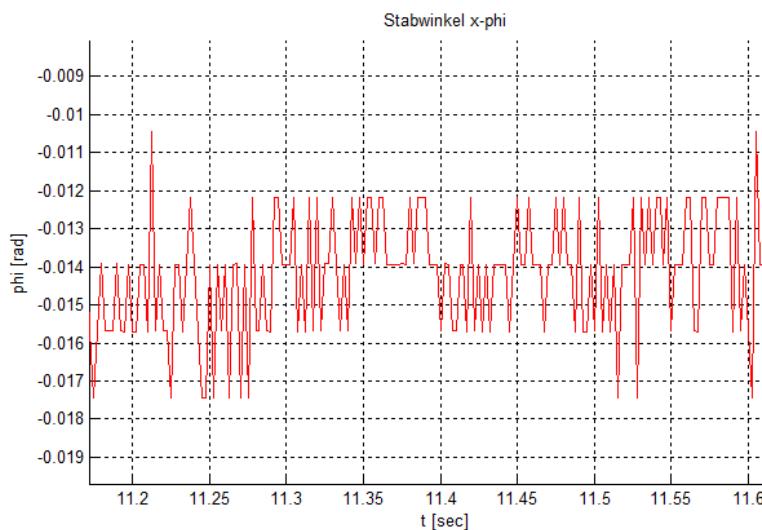


Abbildung 74: Stabspitzenposition (Störung des Stabes)

**Beschreibung** Die Abbildung 74 stellt das Ausbalancieren einer Störung anhand der Stabspitzenposition dar. Diese Messung wurde in einer ruhigen Phase des OmniMoBots aufgezeichnet. Die Störung wird schnell und straff ausbalanciert.

**Ergebnis****Stabwinkel  $\phi_x$** Abbildung 75: Stabwinkel  $\phi_x$  (Störung des Stabes)Abbildung 76: Zoom von „Stabwinkel  $\phi_x$  (Störung des Stabes)“**Beschreibung**

In den Abbildung 75 und Abbildung 76 ist der Stabwinkel  $\phi_x$  dargestellt, welcher während dem Ausbalancieren der Störung des Stabes gemessen wurde. Die Daten entsprechen dem Input „angle {G} actual“ des „Pendulum Control“-Blocks (siehe Abbildung 69, S. 78). In Abbildung 76 ist die ursprüngliche Quantisierung der Hall-Sensorsignale durch den 12-bit ADC, welcher sich auf dem Hall-Sensorprint befindet, gut zu erkennen.

**Ergebnis**  
**Geschwindig-  
keitsverlauf**

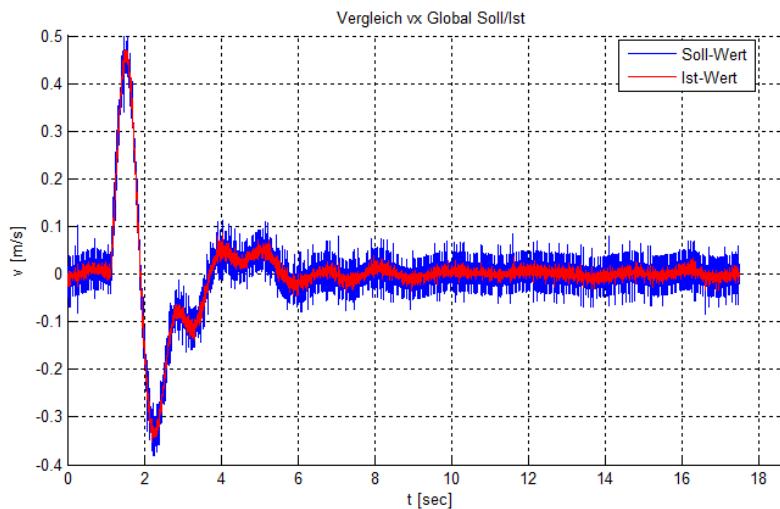


Abbildung 77: Geschwindigkeitsverlauf des OmniMoBots in x-Richtung (Störung des Stabes)

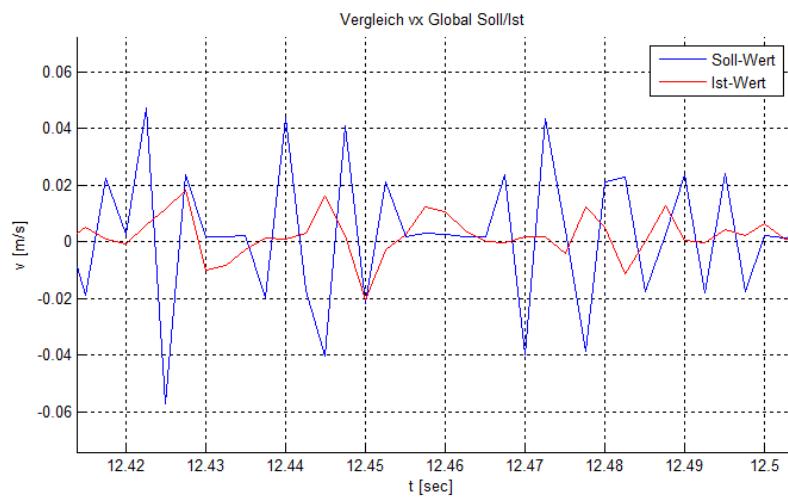


Abbildung 78: Zoom von „Geschwindigkeitsverlauf des OmniMoBotss in x-Richtung (Störung des Stabes)“

**Beschreibung**

Die Abbildung 77 zeigt den Geschwindigkeitsverlauf des OmniMoBots in x-Richtung während dem in Abbildung 74 dargestellten Stabspitzenpositionsverlauf. Die Ausschläge des Sollwerts der Gschwindigkeitsvorgabe entstehen durch das Differenzieren des quantisierten Stabwinkels  $\phi_x$  (siehe Abbildung 76) (siehe dazu auch Abbildung 96, S. 112).

#### 4.5.2.2 Ausbalancieren einer Stabspitzenpositionsvorgabe

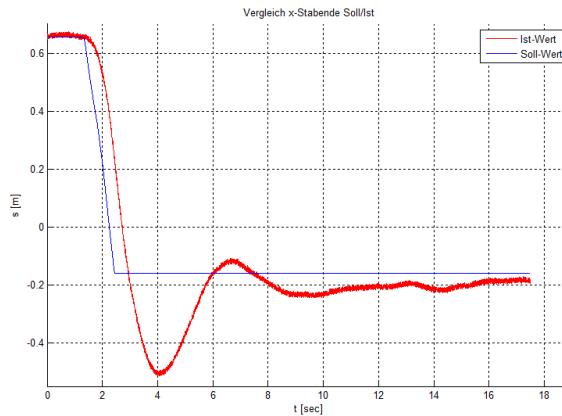
**Ergebnis Stab-  
spitzenposition**


Abbildung 79: Stabspitzenposition (Positionsvorgabe)

**Beschreibung** In Abbildung 79 ist die Reaktion des OmniMoBots auf eine Stabspitzenpositionsvorgabe zu sehen. Diese Messung wurde während einer unruhigen Phase des OmniMoBots durchgeführt. Es ist deutlich zu erkennen, dass der OmniMoBot den Stab weniger straff ausregelt, als bei der zuvor vorgestellten Messung. Dennoch reicht die Balancierfähigkeit für das Steuern des OmniMoBots mittels einer Stabspitzenpositionsvorgabe aus.

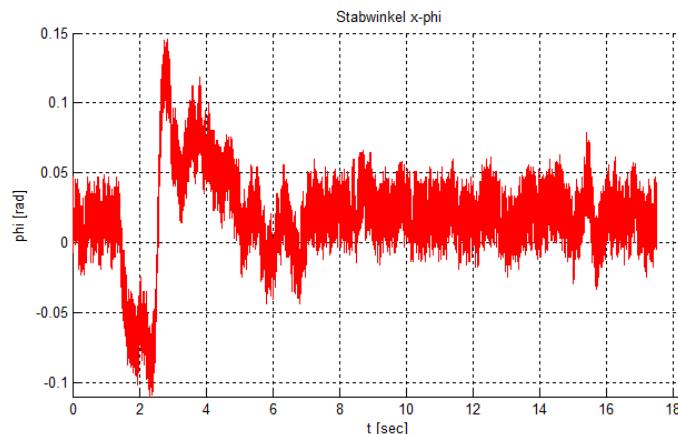
**Ergebnis  
Stabwinkel  $\phi_x$** 


Abbildung 80: Stabwinkel  $\phi_x$  (Positionsvorgabe)

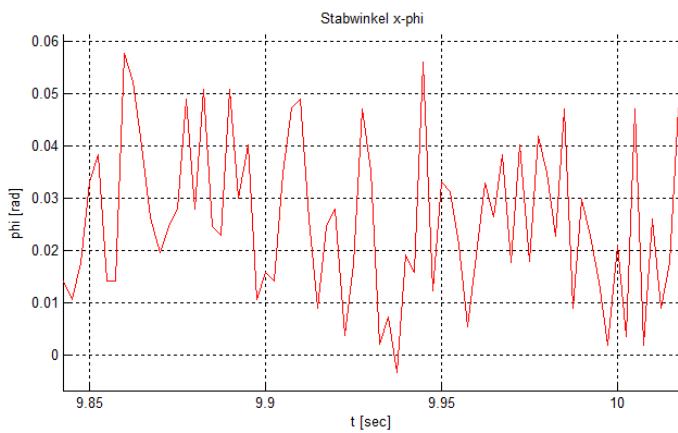


Abbildung 81: Zoom von „Stabwinkel  $\phi_x$  (Positionsvorgabe)“

- Beschreibung** Wenn sich der OmniMoBot in einer unruhigen Phase befindet, hat dies auch einen Einfluss auf den aktuellen Stabwinkel  $\phi_x$ . In Abbildung 81 ist das ursprüngliche quantisierte Signal der Hall-Sensoren nicht zu erkennen, zudem ist es viel stärker verrauscht. Warum das Signal stärker verrauscht ist, muss noch weiter untersucht werden. Die Messdaten stammen vom „{L} to {G}“-Block (siehe Abbildung 69, S. 78), welcher den Stabwinkel  $\vec{\phi}_{xy}$  in das globale KS {G} transformiert.

#### 4.5.2.3 Untersuchung des unruhigen Verhaltens

- Allgemein** Das unruhige Verhalten des Roboters bzw. des Stabwinkels  $\phi_x$  wird untersucht. Zu diesem Zweck werden Messergebnisse des Stabwinkels  $\phi_x$  dargestellt, welche direkt vom „Hall-Sensor“-Block (siehe Abbildung 58, S. 64) ausgegeben werden.

**Ergebnis  
unruhiges  
Verhalten**

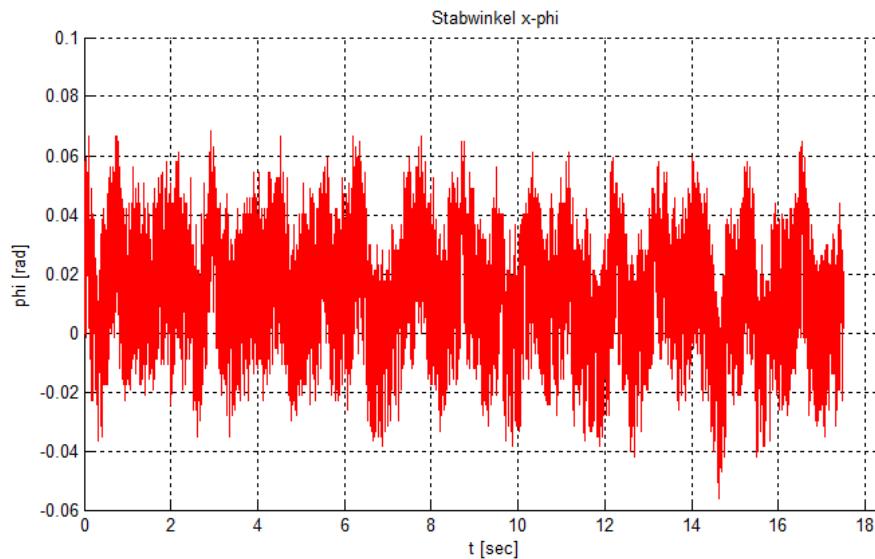
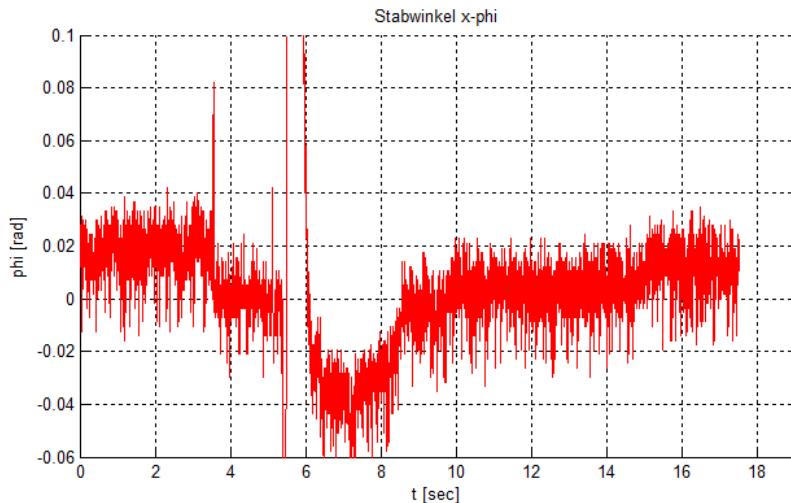


Abbildung 82: Stabwinkel  $\phi_x$  bei unruhigem Verhalten

- Bemerkung** In Abbildung 82 ist der Stabwinkel  $\phi_x$  bei balancierendem, unruhigem Roboter abgebildet. Nun wird untersucht, ob die unruhige Bewegung der Plattform einen Einfluss auf den Stabwinkel  $\phi_x$  hat oder ob der Stabwinkel  $\phi_x$  das unruhige Verhalten der Plattform verursacht.

**Testergebnis**Abbildung 83: Testergebnis Stabwinkel  $\phi_x$  bei etwas ruhigerem Verhalten

**Beschreibung** Bei diesem Test wird wieder der Stabwinkel  $\phi_x$  ausgegeben, jedoch bei etwas ruhigerem Verhalten als bei Abbildung 82. Der Stab wird mit dem OmniMoBot während der Zeit  $t = 0\text{s}$  bis  $t = \text{ca.} 4.5\text{s}$  balanciert, dann wird der Stab entfernt, worauf der OmniMoBot die Motoren abstellt und sich nicht mehr bewegt. Anschliessend wird der Stab von Hand bei ca.  $t = 6\text{s}$  wieder auf den OmniMoBot gesetzt. In der Zeit  $t = 6\text{s}$  bis  $t = 17\text{s}$  wird der Stab von Hand balanciert bei stillstehenden Motoren.

**Erkenntnis** Mit dem Test (siehe Abbildung 83) wird gezeigt, dass die Regelung und die Motoren keinen Einfluss auf den Stabwinkel  $\phi_x$  haben, weil das Rauschen vor und nach dem Abstellen und Regeln der Motoren gleich stark ist. Somit ist die Ursache für das unruhige Verhalten nicht durch die Regelungsstruktur bedingt. Die Ursache für das Rauschen könnte in der Hardware des Mainboards oder in der Hardware des Hall-Sensorprints zu finden sein. Es kann nicht ausgeschlossen werden, dass die Hardware auf leicht unterschiedliche Stromversorgung empfindlich reagiert, was bei einer Stromversorgung mittels Bleiakku durchaus der Fall ist.

#### 4.5.3 Fazit der Fähigkeiten des OmniMoBots

**Balancieren** Der OmniMoBot ist in der Lage, den Stab im Stillstand gut auszubalancieren. Dadurch veranschaulicht er die Dynamik und die drei Freiheitsgrade im Stillstand des Schwenkradantriebssystems. Des weiteren ist er in der Lage, den Stab auszubalancieren während er durch die Stabspitzenpositions vorgabe zusätzlich gesteuert wird.

**Kollisions-vermeidung** Die Kollisionsvermeidung für den OmniMoBot funktioniert sehr gut. Das gesamte Potential der implementierten Kollisionsvermeidung kann jedoch erst vollständig zutage treten, wenn eine zusätzliche Positionsschätzung des OmniMoBots implementiert wird, denn die Positions differenz, welche während der Übertragungszeit der Laserscannerdaten entsteht, lässt sich mittels einer zusätzlichen Positionsschätzung genauer bestimmen. Im Moment wird diese Positions differenz alleine durch die Positionsschätzung, welche durch die Odometrie bestimmt wird, berechnet. Das hat zu Folge, dass beim Rutschen oder hohen Geschwindigkeiten die Position der Hindernisse nicht exakt geschätzt wird. Aus diesem Grund wird bei detektierten Hindernissen die Geschwindigkeit stärker gedrosselt, als es notwendig wäre. Dadurch ist jedoch gewährleistet, dass der OmniMoBot kollisionsfrei bleibt.

## 5 Schlussfolgerung und Ausblick

<b>Ergebnisse</b>	<p>In der vorliegenden Arbeit wurden die Fähigkeiten eines omnidirektionalen mobilen Roboters mit aktiven Schwenkrä dern durch das Ausbalancieren eines zugespitzten Aluminiumstabs aufgezeigt. Zu diesem Zweck wurden die Kinematik des Roboters hergeleitet und verschiedene Regelungskonzepte vorgestellt. Mittels einer Simulation mit Matlab/Simulink konnte das Verhalten des balancierenden Roboters analysiert werden. Gleichzeitig konnte auf diese Weise die Realisierbarkeit dieses Projekts überprüft werden. Der Aufbau der simulierten Regelungsstruktur wurde mittels des Frameworks EEROS in C++ realisiert. Durch diese Vorgehensweise entstand eine gut nachvollziehbare, einfache und effektive Regelungsstruktur für einen omnidirektionalen, auf aktiven Schwenkrä dern basierenden, mobilen Roboter.</p> <p>Mit den Daten eines Laserscanners wurde eine Kollisionsvermeidung realisiert, welche die Befehle des Bedieners oder die Anweisungen des Bahnplaners möglichst genau umsetzt und dabei die Geschwindigkeit so hoch wie möglich beibehält. Dies wird erreicht, indem der Vektor der Geschwindigkeitsvorgabe unterteilt wird in eine radiale und in eine tangentiale Komponente, bezogen auf die Position des Hindernisses. Anschliessend wird der radiale Anteil der Geschwindigkeitsvorgabe, je nach Distanz zu dem Hindernis, begrenzt. Der tangentiale Anteil wird nicht begrenzt. Auf diese Weise verhindert die Kollisionsvermeidung nicht nur die Kollision mit dem Hindernis, sondern unterstützt den Bahnplaner oder den Bediener aktiv beim Hindernis-Ausweichen.</p>
<b>Probleme / offene Punkte</b>	<p>Mit der aktuellen Elektronik ist es nicht möglich, zusätzliche Aufbauten direkt anzusteuern. Das Ansteuern von zusätzlichen Aufbauten, wie z.B. einem Roboterarm, konnte beim Überarbeiteten der Elektronik nicht berücksichtigt werden, da die Elektronik nicht nur im OmniMoBot verwendet wird.</p> <p>EEROS soll in naher Zukunft auch mit ROS-Komponenten kompatibel werden. Dazu müssen die für EEROS nützlichen ROS-Komponenten noch evaluiert werden.</p> <p>Es wurde keine Positionsschätzung mit Hilfe von Laserscanner oder Beschleunigungssensoren implementiert. Für eine zukünftige Bahnplanung und um das volle Potential der implementierten Kollisionsvermeidung nutzen zu können, ist eine zusätzliche Positionsschätzung notwendig. Die Publikationen [15] und [16] könnten dafür hilfreich sein (siehe auch Kapitel 2.2.1, ab S. 14).</p>
<b>Ausblick</b>	<p>Die Geschwindigkeit des OmniMoBots ist momentan auf 1 m/s beschränkt. Falls die Geschwindigkeit erhöht werden soll, wird empfohlen, die Dynamik miteinzubeziehen weil bei höheren Geschwindigkeiten der Einfluss der Dynamik steigt. Dazu kann die Publikation [7] beigezogen werden (siehe auch Kapitel 2.6, S. 31).</p> <p>Beim Umschwenken der Räder („shopping-cart effect“) können sehr hohe Beschleunigungen auftreten. Die maximale Geschwindigkeit, bei welcher dieses Umschwenken durch die Motoren noch geleistet werden kann, liegt bei 1.96 m/s (siehe Gl. (2-49), S. 33).</p> <p>Das Signal des Hall-Sensorprints muss noch weiter untersucht werden, da es zu einem unruhigen Verhalten des OmniMoBots beim Balancieren führen kann (siehe Kapitel 4.5.2.3, S. 86).</p> <p>Um mit dem OmniMoBot bei hoher Geschwindigkeit autonom durch ein Gebäude zu fahren, sind zusätzlich zu den bereits erwähnten Arbeiten noch folgende notwendig:</p> <ul style="list-style-type: none"> <li>• Erstellen eines Bahnplaners</li> <li>• Positionsschätzung mittels Beschleunigungssensoren oder Laserscanner</li> <li>• Evtl. Implementierung einer dynamischen Geschwindigkeitsbegrenzung (siehe dazu [5])</li> </ul>

## 6 Verzeichnisse

### 6.1 Formelzeichen

Formelzeichen	Bezeichnung	Si-Einheit
$\omega$	Winkelgeschwindigkeit	rad/s
$v$	Geschwindigkeit	m/s
$R_{Rad}, R$	Radradius oder Radius der gefährlichen Zohne	m oder m
$d$	Exzentrizität oder Dämpfungskonstante	m oder Nsm/rad
$\varphi$	Winkel zwisch zwei KS	rad
${}^i R_j$	Rotationsmatrix { j } zu { i }	-
$q$	Gelenk- / Motorstellung	rad
$x$	Position im kartesischen KS	m
$l_{AB}$	Mechanische Verbindung von zwei Rädern	m
$v_{-l_{AB}}$	Auf mechanische Verbindung projizierte Geschwindigkeit	m/s
$\vec{a}, \vec{b}$	Richtungsvektoren	m/s
$\lambda_i$	Faktor für Richtungsvektor	-
$c$	Kontaktpunkt c oder Federkonstante	- oder N/m
{ i }	Koordinatensystem i	-
$l_{LR}$	Distanz von { L } zu { R }	m
$l_i$	Distanz von { R } i zu { L }	m
$a$	Beschleunigung	m/s <sup>2</sup>
$g$	Erdbeschleunigung = 9.81	m/s <sup>2</sup>
$e$	Error (differenz von Soll und Ist)	m
$m$	Masse	kg
$k$	Verhältnis m2 zu m1	-
$h_{sp}$	Zusammengefasste Höhe der Massenpunkte	m
$F$	Kraft	N
$\omega_0$	Ungedämpfte Eigenkreisfrequenz des geschlossenen Regelkreises	rad/s, 1/s
$f_0$	Ungedämpfte Eigenfrequenz des geschlossenen Regelkreises	Hz
$K_d$	Dämpfungskonstante	Nsm/rad
$K_p$	Federkonstante	N/m
$D$	Dämpfungsgrad oder Duty Cycle	- oder -
$k_d, k_v$	Diffrenzierender Anteil	1/s
$k_p$	Proportionalen Anteil	1/s <sup>2</sup>
$\tilde{k}p$	Ermöglicht in kaskadierter Regelungsstruktur proportionales Übertragungsverhalten	1/s
$T_d$	Periode des diffrenzierenden Anteils	s

$f_{Task}$	Regelungstakt-Rate	Hz
$T, \Delta t_{Task}$	Regelungstakt	s
$s$	Sicherheitsfaktor oder Distanz: Hindernis zu {s'}	- oder m
$\tau_c$	Geregeltes Antriebsmoment	Nm
$M_q$	Massenmatrix im { q }	$\text{kgm}^2$
$M_x$	Massenmatrix im kartesischen KS	kg
$J$	Jacobimatrix oder Massenträgheitsmoment	- oder $\text{kgm}^2$
$i$	Getriebeuntersetzung oder Index für 1,2,3	- oder -
$AB$	Index für 12, 13 oder 23	-
$M_{mot}$	Motormoment	Nm
$\phi$	Stabwinkel	rad
$f_b$	Reibungseinflüsse	$\text{Nsm/rad}$
$r$	Radius des Massenpunktes oder Radius der Minimalstendistanz	m oder m
$x_{tip}$	Position der Stabspitze (obere Ende des Stabes)	m
$x_{odometrie}$	Die geschätzte Position aus der Odometrie	m
$u, U$	Spannung	V
$P_H$	Position eines Hindernisses	m
$m_d$	Steigung der Distanz des Hindernisses (Geradengleichung)	-
$y_d$	y-Wert: Distanz des Hindernisses	m
$x_d$	x-Wert: Distanz des Hindernisses	m
$m_t$	Steigung der Tangente eines Hindernisses (Geradengleichung)	-
$q_t$	y-Achsenabschnitt der Tangente eines Hindernisses	m
$x_{Schnittp}$	x-Wert des Schnittpunktes: Tangente eines Hindernisses mit neuer Distanz	m
$s' \vec{P}_{os}$	Ortsvektor des { S } im { S' } beschrieben	m
$dt$	Zeitstempeldifferenz	s

## 6.2 Abkürzungsverzeichnis

<b>ADC</b>	Analog-to-Digital-Converter (Analog-Digital-Wandler)
<b>API</b>	Application Programming Interface (Anwendungsprogrammierschnittstelle)
<b>CPU</b>	Central Processing Unit (Hauptprozessor)
<b>DAC</b>	Digital-to-Analog-Converter (Digital-Analog-Wandler)
<b>EEROS</b>	Easy, Elegant, Reliable, Open and Safe (Open Source Robotik Software Framework)
<b>FPGA</b>	Field Programmable Gate Array, ein integrierter Schaltkreis, in welchem eine logische Schaltung programmiert werden kann.
<b>GPIO</b>	General Purpose Input/Output (konfigurierbarer, digitaler Ein- und Ausgang)
<b>HMI</b>	Human-Machine Interface (Mensch-Maschinen-Schnittstelle)
<b>KS</b>	Koordinatensystem
<b>LED</b>	Light-emitting diode (Leuchtdiode)
<b>PD</b>	Übertragungsglied der Regelungstechnik, Regler mit proportionalem und differentiellem Übertragungsverhalten
<b>PWM</b>	Pulse-Width Modulation (Pulsbreitenmodulation)
<b>SPI</b>	Serial Peripheral Interface (serieller Datenbus)
<b>TCP</b>	Tool center point, (Endeffektor)
<b>UML</b>	Unified Modeling Language (grafische Modellierungssprache)
<b>VHDL</b>	Very High Speed Integrated Circuit Hardware Description Language (Hardwarebeschreibungssprache)

## 6.3 Abbildungen

Abbildung 1: Radtypen [1].....	10
Abbildung 2: Plattform des OmniMobs .....	12
Abbildung 3: Parameter und Koordinatensysteme des OmniMoBot .....	12
Abbildung 4: Globales und lokales Koordinatensystem.....	13
Abbildung 5: Direkte Kinematik .....	14
Abbildung 6: Inverse Kinematik.....	14
Abbildung 7: Mechanische Kopplung .....	16
Abbildung 8: Vektorieller Zusammenhang.....	17
Abbildung 9: Momentanpolbestimmung des Roboters .....	18
Abbildung 10: Stillstand .....	18
Abbildung 11: Orthogonale Bewegung zur mechanischen Verbindung l23,.....	19
Abbildung 12: Drehung um die Lenkachse.....	19
Abbildung 13: Umschwenken des Rades [5].....	23
Abbildung 14: Berechnungsmodell Kippstabilität [5].....	24
Abbildung 15: Mechanisches System [9] .....	25
Abbildung 16: Stabile Regelung: $T_d > \Delta t_{Task}$ [9].....	27
Abbildung 17: Instabile Regelung: $T_d < \Delta t_{Task}$ [9].....	27
Abbildung 18: Positionsregelung.....	27
Abbildung 19: Kaskadierte Regelung .....	28
Abbildung 20: Regelungskonzept für mobile Roboter im kartesischen KS .....	29
Abbildung 21: Regelkonzept für OmniMoBot im kartesischen KS und im Gelenkkoordinatensystem .....	30
Abbildung 22: Regelungskonzept für OmniMoBot im Gelenkkoordinatensystem .....	30
Abbildung 23: Vereinfachte Darstellung des Antriebs .....	32
Abbildung 24: Vereinfachte Darstellung der Lenkung .....	34
Abbildung 25: Inverses Pendel [8] .....	36
Abbildung 26: Inverses Pendel 3D [8] .....	37
Abbildung 27: PD-Regelung des Stabwinkels .....	38
Abbildung 28: PD-Regelung der Stabspitze .....	39
Abbildung 29: EEROS Architektur [51].....	40
Abbildung 30: Beispiel einer Regelungsstruktur [51] .....	40
Abbildung 31: Beispiel Sequencer [51] .....	41
Abbildung 32: Parallel laufende Sequenzen [51].....	41
Abbildung 33: Ein Systemaufbau für eine fLink-Beispielanwendung [10] .....	43
Abbildung 34: API Levels der fLink Libarary [10].....	44
Abbildung 35: Simulink Modell stabbalancierender OmniMoBot .....	46
Abbildung 36: Subsystem „Trajectory generator“ .....	47
Abbildung 37: Subsystem „Pendulum control“ .....	47
Abbildung 38: Subsystem „OmniMoBot“ .....	47
Abbildung 39: Subsystem „Robot simulation“.....	48
Abbildung 40: Subsystem „Robot control“ .....	49
Abbildung 41: Subsystem „Motor model“ .....	49
Abbildung 42: Sollvorgabe für den Trajektorien-Generator für die Geschwindigkeit in x- und y-Richtung, Einheit y-Achse [m/s] .....	50
Abbildung 43: Sollwertvorgabe für die Drehgeschwindigkeit des OmniMoBots, Einheit y-Achse [rad]....	50
Abbildung 44: Träger Stabspitzenpositionsverlauf in x-Richtung .....	51

---

Abbildung 45: Verhalten der Stabspitze in x-Richtung .....	52
Abbildung 46: Geschwindigkeitsverlauf des OmniMoBots in x-Richtung .....	52
Abbildung 47: OmniMoBot .....	53
Abbildung 48: Komponenten des OmniMoBots .....	53
Abbildung 49: Balancierender OmniMoBot .....	53
Abbildung 50: ESCON Module 50/5 .....	55
Abbildung 51: ADC128S102CIMT .....	57
Abbildung 52: Hall-Sensorprint mit Störungen über 50 Sekunden.....	57
Abbildung 53: Hall-Sensorprint mit Störungen über 0.22 Sekunden.....	58
Abbildung 54: Hall-Sensorprint voll funktionsfähig .....	59
Abbildung 55: Software stack.....	60
Abbildung 56: Aufbau eines Blockes mit EEROS.....	62
Abbildung 57: Control System OmniMoBot in C++.....	63
Abbildung 58: Datenverarbeitung des Hall-Sensorprints in C++.....	64
Abbildung 59: Robot control in C++ .....	64
Abbildung 60: Homingblock in C++ .....	65
Abbildung 61: Levels und Events vom Safety System .....	68
Abbildung 62: Funktionsprinzip der Klasse „CollisionDetection“.....	70
Abbildung 63: Ergebnis der Klasse „CollisionDetection“.....	71
Abbildung 64: Funktionsprinzip der „SafetyVelocityDesired“ Klasse .....	73
Abbildung 65: Radiale Begrenzung der Geschwindigkeit .....	74
Abbildung 66: Beispiel 1 der „SafetyVelocityDesired“ Klasse .....	75
Abbildung 67: Beispiel 2 der „SafetyVelocityDesired“ Klasse .....	75
Abbildung 68: Softwarestruktur der Laserscanner-Datengewinnung .....	77
Abbildung 69: Control System mit Kollisionsvermeidung.....	78
Abbildung 70: Struktur des „LaserScannerblocks“ .....	79
Abbildung 71: Struktur des „TransitionLaserData"-Blocks .....	79
Abbildung 72: Offset beim Start des Balanciermodus .....	81
Abbildung 73: Offset des eingependelten Stabwinkels $\phi_x$ .....	81
Abbildung 74: Stabspitzenposition (Störung des Stabes).....	82
Abbildung 75: Stabwinkel $\phi_x$ (Störung des Stabes) .....	83
Abbildung 76: Zoom von „Stabwinkel $\phi_x$ (Störung des Stabes)“ .....	83
Abbildung 77: Geschwindigkeitsverlauf des OmniMoBots in x-Richtung (Störung des Stabes).....	84
Abbildung 78: Zoom von „Geschwindigkeitsverlauf des OmniMoBotss in x-Richtung (Störung des Stabes)“ .....	84
Abbildung 79: Stabspitzenposition (Positionsvorgabe) .....	85
Abbildung 80: Stabwinkel $\phi_x$ (Positionsvorgabe) .....	85
Abbildung 81: Zoom von „Stabwinkel $\phi_x$ (Positionsvorgabe)“ .....	85
Abbildung 82: Stabwinkel $\phi_x$ bei unruhigem Verhalten .....	86
Abbildung 83: Testergebnis Stabwinkel $\phi_x$ bei etwas ruhigerem Verhalten .....	87
Abbildung 84: Elektromechanisches Modell eines DC Motors [11].....	98
Abbildung 85: Gekoppelte Differentialgleichungen des elektromechanischen Modells [11].....	98
Abbildung 86: Schema des ESCON Testaufbaus .....	104
Abbildung 87: Regelungsstruktur, welche im Testprogramm implementiert wurde .....	105
Abbildung 88: Nullposition des OmniMoBots .....	105
Abbildung 89: Headerboard phyCORE-MPC5200B-I/O .....	106

---

---

Abbildung 90: Mainboard „NTB C32 Carrier Board“ mit Headerboard.....	106
Abbildung 91: Drive-Board „NTB 6xDC“ .....	107
Abbildung 92: HMI-Print [5].....	108
Abbildung 93: Schaltplan HMI-Print .....	109
Abbildung 94: Hall-Sensorprint .....	110
Abbildung 95: Anordnung der Hall-Sensoren [8] .....	110
Abbildung 96: Problem von differenzierten Encoder-Werten (Symbolbild) [5] .....	112
Abbildung 97: Laserscanner "URG-04LX".....	112
Abbildung 98: Messrichtung und Datenpunkte des Laserscanners .....	113
Abbildung 99: LC-R127R2PG1 .....	114

## 6.4 Tabellen

---

Tabelle 1: Parameterwerte des Antriebs .....	32
Tabelle 2: Trägheitsmomente des Antriebs im Überblick .....	33
Tabelle 3: Parameterwerte der Lenkung .....	34
Tabelle 4: Trägheitsmomente der Lenkung im Überblick .....	35
Tabelle 5: Maxon DC motor RE 40 148867.....	111
Tabelle 6: Technische Daten von Laserscanner .....	113

## 6.5 Literatur- und Quellenverzeichnis

Auf die nachstehenden Werke wird in der Arbeit mit der in der eckigen Klammer angegebenen Nummer verwiesen.

- Literatur
- [1] **Roland Siegwart, Illah R. Nourbakhsh, Davide Scaramuzza** „Introduction to Autonomous Mobile Robots“ second edition, MIT Press, ISBN: 9780262015356, 2004
  - [2] **Bruno Siciliano, Oussama Khatib** „Handbook of Robotics“, Springer Verlag, ISBN: 9783540239574, 2008
  - [3] **H. Hharboutly, A. Benali, F. B. Amar, M. Bouzit, J. Ma** „Design of Omnidirectional Mobile Platform for Balance Analysis“, IEEE/ASME Trans. Mechatronics, vol. 19, no. 6, pp. 1872-1881, Dez. 2014
  - [4] **Ahmed Khamies El-Shenawy** „Motion control of Holonomic wheeled mobile robot with modular actuation“, Universität Mannheim, 2010
  - [5] **David Frommelt** Masterthesis „Hochdynamische Antriebseinheit für omnidirektionale mobile Roboter“, NTB Interstaatliche Hochschule für Technik Buchs, Feb. 2013
  - [6] **Claudio Roffler** Masterthesis „Hochdynamische Antriebseinheit für omnidirektionale mobile Roboter“, NTB Interstaatliche Hochschule für Technik Buchs, Feb. 2013
  - [7] **J. H. Chung, Byung-Ju Yi, W. K. Kim, Seog-Young Han** „Singularity-Free Dynamic Modeling Including Wheel Dynamics for an Omni-Directional Mobile Robot with Three Caster Wheels“, IJCAS, vol. 6, no. 1, pp.86-100, Feb. 2008
  - [8] **Claudia Visentin** „Stabilisierung eines Pendels auf einem Scara Roboter“, NTB Interstaatliche Hochschule für Technik Buchs, Nov. 2012
  - [9] **Einar Nielsen** Skript „Roboterkinematik-4“, NTB Interstaatliche Hochschule für Technik Buchs, 2014
  - [10] **Martin Züger** Masterthesis „fLink An universal FPGA device interface for realtime Linux“, NTB Interstaatliche Hochschule für Technik Buchs, April. 2014
  - [11] **Josef Goette** Skript MSE „Control of DC Motors“, 2013
  - [12] **Patrick F. Muri, Charles P. Neuman** „Kinematic Modeling of Wheeled Mobile Robots“, The Robotics Institute Carnegie-Melon University Pittsburgh, Pennsylvania 15213, Juni 1986
  - [13] **Simon Jantscher** Diplomarbeit „Modulares Rekonfigurierbares Roboterfahrwerk“, Technische Universität Graz, Dez. 2008
  - [14] **Johannes Köb** Diplomarbeit „Modellbasierte Regelung eines Roboterfahrwerks“, Technische Universität Graz, Okt. 2005

[15] **Y. K. Thong, M. S. Woolfson, J. A. Crowe, B. R. Hayes-Gill, D. A. Jones** „Numerical double integration of acceleration measurements in noise“, School of Electrical and Electronic Engineering, University of Nottingham, Nottingham NG7 2RD, United Kingdom, April 2004

[16] **Soo Jeon, Masayoshi Tomizuka** „Benefits of acceleration measurement in velocity estimation“, Department of Mechanical Engineering, University of California at Berkeley, Berkeley, CA 94720, USA, Dez. 2005

[17] **Ernst-Günter Paland** „Technisches Taschenbuch“ INA, Schaeffler KG, Mai 2002

**Links**

[50] <http://studywolf.wordpress.com/category/robotics/operational-space-control/>,

aufgerufen am 19.11.2013

[51] <http://www.eeros.org>

aufgerufen am 29.12.2014

[52] [https://www.youtube.com/watch?v=7fTq\\_YM1Ha8&feature=youtu.be](https://www.youtube.com/watch?v=7fTq_YM1Ha8&feature=youtu.be),

aufgerufen am 29.12.2014

[53] <https://www.youtube.com/watch?v=rG3bmQ7xzZc&feature=youtu.be>,

aufgerufen am 29.12.2014

[54] [http://wiki.ntb.ch/infoportal/embedded\\_systems/mpc5200/phycore-mpc5200b-io/c32-carrierboard/start](http://wiki.ntb.ch/infoportal/embedded_systems/mpc5200/phycore-mpc5200b-io/c32-carrierboard/start), aufgerufen am 05.04.2015

[55] <https://studywolf.wordpress.com/category/robotics/operational-space-control/>,

aufgerufen am 06.04.2015

[56] <http://wiki.ntb.ch/stud/omnimobot/start>,

aufgerufen am 07.05.2015

## 7 Erklärung zur Urheberschaft

**Erklärung** Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

**Ort/Datum** Buchs SG, 22. Mai 2015

**Unterschrift**

Stefan Landis

# Anhang

---

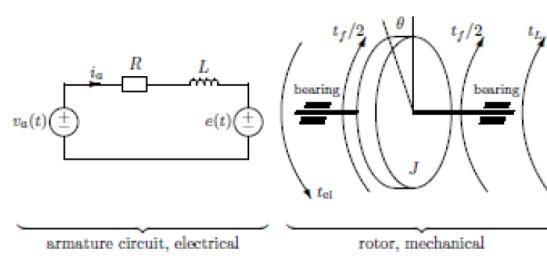
## A. Simulink Simulation

---

### I. Spannungsvorgabe („Duty Cycle“ des PWMs)

**Elektro-  
mechanisches  
Modell**

Das elektromechanische Modell eines DC Motors kann wie folgt beschrieben werden:



$v_a$ : armature voltage	$t_{el}$ : electrical torque
$i_a$ : armature current	$t_L$ : load torque
$R$ : terminal resistance	$t_f$ : friction torque
$L$ : terminal inductance	$J$ : rotor inertia
$e$ : back emf voltage	$\theta$ : angular position

Abbildung 84: Elektromechanisches Modell eines DC Motors [11]

Daraus lassen sich die gekoppelten Differentialgleichungen herleiten:

$$\begin{aligned} \text{electrical: } & \frac{d}{dt} i_a + \frac{R}{L} i_a + \frac{K_g}{L} \dot{\theta} = \frac{1}{L} v_a \\ \text{mechanical: } & \ddot{\theta} + \frac{b_f}{J} \dot{\theta} - \frac{K_m}{J} i_a = -\frac{1}{J} t_L \end{aligned}$$

with the newly introduced symbols meaning

$K_g$  : generator constant

$K_m$  : motor constant or torque constant

$b_f$  : viscous friction constant

Abbildung 85: Gekoppelte Differentialgleichungen des elektromechanischen Modells [11]

Mit diesen zwei gekoppelten Differentialgleichungen kann die benötigte Spannung des Systems bestimmt werden. In diesem Fall wird angenommen, dass die Generatorkonstante  $K_g$  gleich der Motorkonstante  $K_m$  ist.

**Mechanischer  
Teil**

Die mechanische Gleichung sieht nach der Vernachlässigung der Reibung wie folgt aus:

$$\tau_{Motor} = i_a \cdot K_m = J \cdot \dot{\omega} + t_L \quad (0-1)$$

Die mechanische Zeitkonstante eines DC-Motors ist wie folgt definiert:

$$t_{mech} = \frac{R \cdot J}{K_m^2} \quad (0-2)$$

**Elektrischer Teil** Die Definition der elektrischen Zeitkonstante ist:

$$t_{el} = \frac{L}{R} \quad (0-3)$$

Die Veränderung des Stroms wird durch eine e-Funktion beschrieben:

$$i(t) = \frac{U}{R} \cdot \left(1 - e^{-\frac{t}{t_{el}}}\right) \quad (0-4)$$

Da die elektrische Zeitkonstante  $t_{el}$  des Motors sehr viel kleiner ist als die mechanische  $t_{mech}$ , kann die Vereinfachung  $L \cdot \frac{di_a}{dt} \approx 0$  gemacht werden. Dadurch wird der Einfluss der Stromänderung, welche über die Induktivität  $L$  beeinflusst wird, vernachlässigt. Somit wird der elektrische Teil durch die folgende Gleichung bestimmt:

$$v_a = R \cdot i_a + K_m \cdot \omega = R \cdot \frac{\tau_{Motor}}{K_m} + K_m \cdot \omega \quad (0-5)$$

## II. M-File für Simulation (Abbildung 35, S. 46)

```

clear all
close all
clc

%% Parameter

% Regelungsparameter OmniMoBot
fcontroller = 1000;
D = 0.7; % Dämpfungskonst
S = 2; % Sicherheitsfaktor
f0 = fcontroller/(4*pi*D*S);
omega0 = 2*pi*f0;
kp = omega0/(2*D);
kd = 2*D*omega0;
dt = 0.001;

% Regelungsparameter Stabregelung
DTip = 1;
fTip = 23;

fTip0 = fTip/(4*pi*DTip*S); % Frequenz der Spitzen-Regelung
omegaTip = 2*pi*fTip0;
kpTip = omegaTip*omegaTip;
kdTip = 2*omegaTip*DTip;

DAngle = 1;
fAngle = 80;

fAngle0 = fAngle/(4*pi*DAngle*S); % Frequenz der Winkel-Regelung
omegaAngle = 2*pi*fAngle0;
kpPhi = omegaAngle*omegaAngle;
kdPhi = 2*omegaAngle*DAngle / 20;

% Mechanik
Jredgetr =[0.03426;0.03426;0.03426;0.06672;0.06672;0.06672]; % 1-3 = 0.03426: Trägheitsmoment bei Ausgang Getriebe (Antrieb)
% 3-6 = 0.06672: Trägheitsmoment bei Ausgang Getriebe (Lenkung)
inv_Jredgetr =[1/0.03426;1/0.03426;1/0.03426;1/0.06672;1/0.06672;1/0.06672];

gear = [18;18;18;36.75;36.75;36.75]; % 1-3 = 18: Getriebeübersetzung (Antrieb),
% 3-6 = 36.75: Getriebeübersetzung (Lenkung)

inv_gear = [1/18;1/18;1/18;1/36.75;1/36.75;1/36.75];
km = 0.0302; % Drehmomentkonstante
inv_km = 1/km;
R = 0.299; % el.Widerstand des Motors [ohm]
inv_R = 1/R;
L = 0.0823*10^-3; % Induktivität des Motors
omegaMax_lenk = 20.5; % Leerlaufdrehzahl Getriebeausgang in rad/s für Motor 7200 min^-1
omegaMax_an = 41.8; % Leerlaufdrehzahl Getriebeausgang in rad/s für Motor 7200 min^-1
Mmotmax = 0.531; % 3x Dauerdrehmoment [Nm]

% Stab
d = 0.08; % Abstand Spitze - Schwerpunkt
Jstab = 0.00038; % Massenträgheit [kg*m^2]
mstab = 0.037; % Masse von Stab [Kg]
r = sqrt(Jstab/mstab); % Länge des Stabes [m]
lstab = 0.324; % Länge des Stabes [m]
g = 9.81; % Erdbeschleunigung

```

```

%% RoboterDefinition

N = 3;                                % Anzahl Räder (gleichverteilt)
offset = 0.0965049;                      % Abstand Radachse zu drehachse in doku Exzentrizität d [m]
l = 0.392598183;                        % Grösse Roboter (Mittelachse zu Drehachse der Räder)
rRad=0.05;                               % Rad radius [m]
soll_geschw_homing = [0.2;0.2;0.2;0.2;0.2;0.2;]; % Geschw. von lenkung beim homing
state = 4;
homingSwitchOn = 0;                      % 1 = homing, 0 = joystick

offsetEnc = [0;0;0;0;0;0];                % real: [0;0;0;1.776;2.833;5.6997] gemessen in rad (wenn Motor in positiver
                                         % Richtung dreht (gegenurzeigersinn von oben)

dx_v = [1;-1*0.5;-1*0.5];               % grössenvorgabe des OmniMoBots wird für die Jacobi benutzt
dy_v = [0;1*0.866;1*-0.866];

sim('Stabbalance');

%% Plots
h1 = figure;
subplot(211)
plot(vx.time, vx.signals.values(:,1),'b','LineWidth',2)
grid on
set(gca,'FontSize',10)
title('Vergleich vx Global Soll/Ist')
xlabel('t [sec]')
ylabel('v [m/s]')
hold on
plot(vx.time, vx.signals.values(:,2),'r','LineWidth',1)
axis([0 7 -1.5 1.5])
legend('Soll-Wert','Ist-Wert')
set(h1,'color',[1 1 1])

subplot(212)
plot(vy.time, vy.signals.values(:,1),'b','LineWidth',2)
grid on
title('Vergleich vy Global Soll/Ist')
xlabel('t [sec]')
ylabel('v [m/s]')
hold on
plot(vy.time, vy.signals.values(:,2),'r','LineWidth',1)
axis([0 7 -1.5 1.5])
legend('Soll-Wert','Ist-Wert')

h2 = figure;
subplot(211)
plot(xtip.time, xtip.signals.values(:,1),'b','LineWidth',2)
grid on
set(gca,'FontSize',10)
title('Vergleich x-Stabende Soll/Ist')
xlabel('t [sec]')
ylabel('s [m]')
hold on
plot(xtip.time, xtip.signals.values(:,2),'r','LineWidth',1)
axis([0 7 0 1])
legend('Soll-Wert','Ist-Wert')
set(h2,'color',[1 1 1])

```

```

subplot(212)
plot(ytip.time, ytip.signals.values(:,1), 'b','LineWidth',2)
grid on
title('Vergleich y-Stabende Soll/Ist')
xlabel('t [sec]')
ylabel('s [m]')
hold on
plot(ytip.time, ytip.signals.values(:,2), 'r','LineWidth',1)
axis([0 7 0 1])
legend('Soll-Wert','Ist-Wert')

h3 = figure;
subplot(211)
plot(phix.time, phix.signals.values(:,1), 'b','LineWidth',2)
grid on
set(gca,'FontSize',10)
title('Vergleich phiX-Stab Soll/Ist')
xlabel('t [sec]')
ylabel('phiX [rad]')
hold on
plot(phix.time, phix.signals.values(:,2), 'r','LineWidth',1)
axis([0 7 -1.8 1.8])
legend('Soll-Wert','Ist-Wert')
set(h3,'color',[1 1 1])

subplot(212)
plot(phiy.time, phiy.signals.values(:,1), 'b','LineWidth',2)
grid on
title('Vergleich phiY-Stab Soll/Ist')
xlabel('t [sec]')
ylabel('phiY [rad]')
hold on
plot(phiy.time, phiy.signals.values(:,2), 'r','LineWidth',1)
axis([0 7 -1.8 1.8])
legend('Soll-Wert','Ist-Wert')

```

### III. S-Funktion Jacobimatrix

```

function [x_d,y_d,Phi_GL_d] = fcn( omega_phi,omega_lenk,rRad,offset,dx,dy,Phi_LR)
v_LR_L_i = zeros(3,3);
l_i = zeros(3,3);

v_LRx_R = omega_phi*rRad; % v_GRx ist ein Vektor mit allen vx komponenten (Räder)

v_LRY_R = -omega_lenk*offset;

for i=1:3
    v_LR_R = [v_LRx_R(i);v_LRY_R(i);0];

    Rot_rad2lokal = [ cos(Phi_LR(i)) -sin(Phi_LR(i)) 0
                      sin(Phi_LR(i))  cos(Phi_LR(i)) 0
                      0                 0                 1];
    v_LR_L_i(:,i) = Rot_rad2lokal*v_LR_R;

    l_i(:,i) = [dx(i);dy(i);0];
end

% lineares Mittel
v_LR_L = sum(v_LR_L_i,2)/3;

x_d = v_LR_L(1);
y_d = v_LR_L(2);

Phi_GL_d = sum(cross(l_i,v_LR_L_i)./(ones(3,1)*sum(l_i.^2,1)),2)/3;
end

```

**IV. S-Funktion inverse Jacobi**

```

function [Koordi_R1, Koordi_R2, Koordi_R3, Phi_LR_d] = fcn( x_d, y_d, Phi_GL_d, rRad, offset, dx1, dy1, dx2, dy2, dx3, dy3, Phi_LR)
Phi_LR_d = zeros(3,1);

Koordi_G = [x_d; y_d; Phi_GL_d]; % _d bedeutet abgeleitet, Koordi_G bedeutet Korrdinatensystem global

inv_jacobi1 = [ cos(Phi_LR(1))/rRad      sin(Phi_LR(1))/rRad      (-dy1*cos(Phi_LR(1))+sin(Phi_LR(1))*dx1)/rRad;
                -sin(Phi_LR(1))/offset   cos(Phi_LR(1))/offset   (-dy1*-sin(Phi_LR(1))+dx1*cos(Phi_LR(1)))/offset;
                                0                      0                      1];
];

Koordi_R1 = inv_jacobi1 * Koordi_G; % Koord_R bedeutet Korrdinatensystem beim Rad = [phi_d; Phi_LR_d; Phi_GL_d]

Phi_LR_d(1) = -Koordi_R1(2);

%%

inv_jacobi2 = [ cos(Phi_LR(2))/rRad      sin(Phi_LR(2))/rRad      (-dy2*cos(Phi_LR(2))+sin(Phi_LR(2))*dx2)/rRad;
                -sin(Phi_LR(2))/offset   cos(Phi_LR(2))/offset   (-dy2*-sin(Phi_LR(2))+dx2*cos(Phi_LR(2)))/offset;
                                0                      0                      1];
];

Koordi_R2 = inv_jacobi2 * Koordi_G; % Koord_R bedeutet Korrdinatensystem beim Rad = [phi_d; Phi_LR_d; Phi_GL_d]

Phi_LR_d(2) = -Koordi_R2(2);

%%

inv_jacobi3 = [ cos(Phi_LR(3))/rRad      sin(Phi_LR(3))/rRad      (-dy3*cos(Phi_LR(3))+sin(Phi_LR(3))*dx3)/rRad;
                -sin(Phi_LR(3))/offset   cos(Phi_LR(3))/offset   (-dy3*-sin(Phi_LR(3))+dx3*cos(Phi_LR(3)))/offset;
                                0                      0                      1];
];

Koordi_R3 = inv_jacobi3 * Koordi_G; % Koord_R bedeutet Korrdinatensystem beim Rad = [phi_d; Phi_LR_d; Phi_GL_d]

Phi_LR_d(3) = -Koordi_R3(2);

end

```

## B. Test ESCON Module 50/5

**Hardware des Testaufbaus**

- **Netzteil:** 24V (25 A möglich)
- **CPU-Board:** Phytec phyCORE-MPC5200B-I/O
- **Drive-Board:** ESCON Module 50/5
- **DC-Motor:** RE 40 148867
- **Getriebe:** 36.75 (Lenkung)
- Trägheitsmoment:  $0.029557 \text{ [kgm}^2\text{]}$  Trägheitsmoment des statischen Systems (siehe Tabelle 4, S. 35). Der OmniMoBot wurde dazu auf den Kopf gedreht.

## **Schema des Testaufbaus**

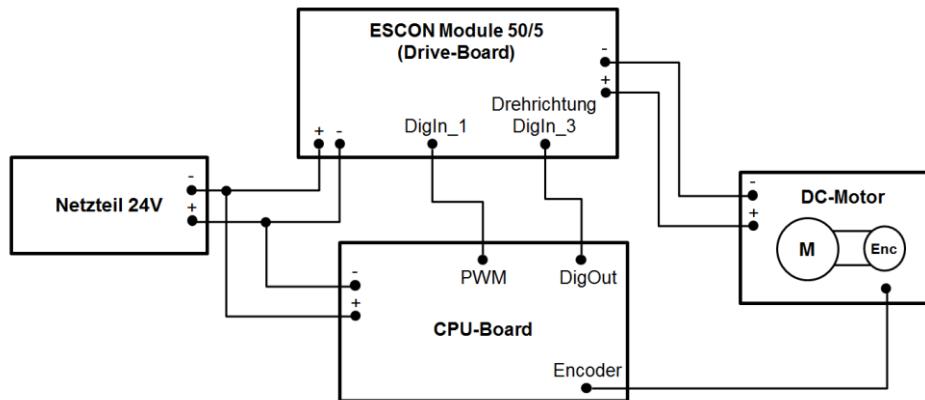


Abbildung 86: Schema des ESCON Testaufbaus

**ESCON Studio** Dem „ESCON“-Board müssen als erstes mit einem Programm (ESCON Studio) die Parameter des anzutreibenden Motors angegeben werden. Folgende Schritte wurden durchgeführt:

1. **Drehzahlkonstante:**  $317.0 \text{ [min}^{-1}\text{V}^{-1}\text{]}$
2. **Thermische Zeitkonstante der Wicklung:**  $41.5 \text{ sec}$
3. **Grenzdrehzahl:**  $12000.0 \text{ [min}^{-1}\text{]}$
4. **Nennstrom:** 5 A
5. **Max. Ausgangstrom:** 15 A
6. Kein Sensor vorhanden
7. Stromregler
8. **Freigabe und Drehrichtung:** DigIn 2 High aktiv und DigIn 3 High aktiv
9. **PWM-Sollwert:** DigIn 1, 10% = -0.2 A, 90% = 15 A (**DigIn 1 muss frei sein, sonst kann im Register der PWM-Sollwert nicht ausgewählt werden**)
10. **Fixer Offset:** = 0

**Testprogramm** Das Testprogramm ist auf der CD im Anhang<sup>8</sup> zu finden. Der Motor der Lenkung wurde mit folgender Regelungsstruktur betrieben:

<sup>8</sup> Anhang F CD in Ordner: 6 Software\1 OmniMoBot\test, „test-Maxon“

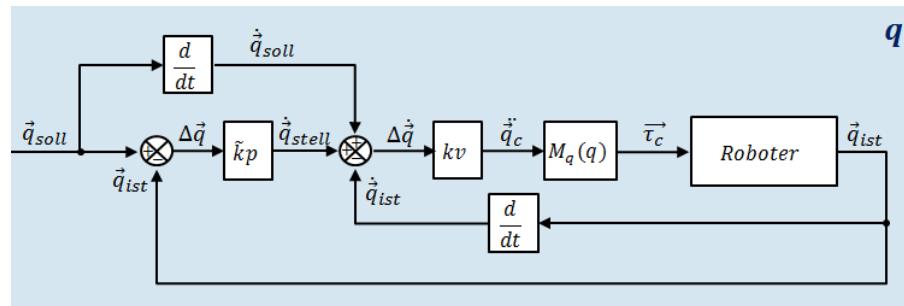


Abbildung 87: Regelungsstruktur, welche im Testprogramm implementiert wurde

## C. Nullposition der Homingsensoren bestimmen

- 1. Schritt** Jedes Schwenkrad wird auf die Nullposition ausgerichtet:

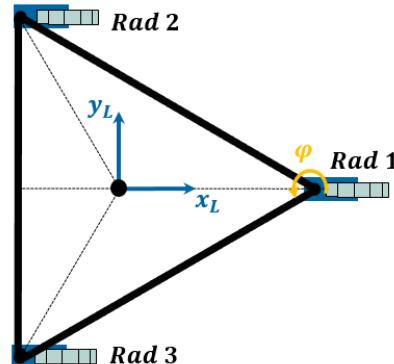


Abbildung 88: Nullposition des OmniMoBots

- 2. Schritt** Jedes Schwenkrad wird in positiver Richtung gedreht, bis der jeweilige Homingsensor anschlägt. Die Homingsensoren befinden sich an den folgenden Positionen:

**Schwenkrad 1 = 1.776 [rad]**

**Schwenkrad 2 = 2.833 [rad]**

**Schwenkrad 3 = 5.6997 [rad]**

## D. Elektronische Bauteile

- Allgemein** Die bisherige Elektronik des OmniMoBots wurde nicht vollständig übernommen. Das Mainboard wurde neu entwickelt und das Drive-Board wurde verändert. Zusätzlich wurde der OmniMoBot mit einer Sicherheitsschaltung ergänzt.

## I. Mainboard

Allgemein	Als Mainboard wird das „NTB C32 Carrier Board“ <sup>9</sup> [54] verwendet. Es ist Träger eines Headerboards der Firma Phytec (phyCORE-MPC5200B-I/O). Dieses Board enthält einen MPC5200 Mikrocontroller aus der Familie der PowerPC-Prozessoren des Herstellers Freescale. Zusätzlich besitzt das Headerboard einen FPGA.
MPC5200 Headerboard	<p>Der MPC5200 ist ein 32-Bit-RISC- Mikrocontroller. Der Kern basiert vorwiegend auf dem PowerPC 603e (G2_LE-Kern), welcher eine grosse Ähnlichkeit mit dem e300-Kern aufweist. Er enthält eine Double Precision FPU, eine MMU für Daten und Instruktionen, 16 KiB Daten- und Instruktionsscache, einen komplexen DMA-Controller (BestComm) für I/O-Operationen, einen SDR/DDR-RAM Controller, einen Ethernet MAC (100 MBps), ein USB 1.1 Interface, sechs programmierbare serielle Controller und zwei I<sup>2</sup>C Controller.</p> <p>Der MPC5200B ist eine Die-Shrink-Version mit einem etwas effizienteren DMA-Controller und einem geringeren Stromverbrauch. Beide Varianten sind mit bis zu 400 MHz erhältlich.</p>

Prints



Abbildung 89: Headerboard phyCORE-MPC5200B-I/O

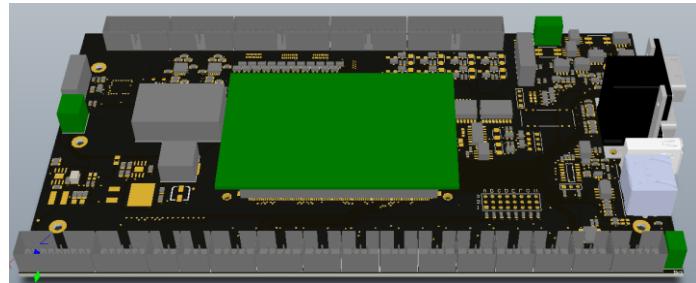


Abbildung 90: Mainboard „NTB C32 Carrier Board“ mit Headerboard

Schnittstellen

- **Servoantrieb:** Der FPGA des Headerboards generiert über fLink (siehe Kapitel 2.9, S. 42) ein PWM. Beim Mainboard sind acht auf diese Weise erzeugte PWMs herausgeführt. Diese acht PWMs werden über einen Pegelwandler von 3.3 auf 5 Volt erhöht. Pro Achse werden zwei PWMs benötigt, im Ganzen somit 12 PWMs. Die restlichen vier PWMs (Achse 4 und 5) werden ohne Pegelwandler direkt vom FPGA ausgelesen. Alle PWMs werden direkt mit den Mosfets des Drive-Boards verbunden.
- **Sensorprint:** Der FPGA des Headerboards erstellt ein SPI-Signal. Dieses wird über einen Adapterprint, welcher einen Line Receiver enthält (siehe Adapterprint im Schaltplan), geschickt. An den Adapterprint wird der Sensorprint angeschlossen.
- **Ethernet:** Die Ethernet-Ansteuerung wird ebenfalls vom Headerboard zur Verfügung gestellt. Das Mainboard ist mit einer 10/100 Mbit Ethernet-Buchse bestückt.
- **W-LAN:** Funktioniert noch nicht.

<sup>9</sup> Anhang F CD in Ordner: 4\_Elektronik\0\_Prints\Mainboard

- **2x RS232 auf 9pol D-Sub:** Das Headerboard stellt zwei RS232 Schnittstellen zur Verfügung. Diese wurden durch das Mainboard hinausgeführt. Die Schnittstelle „ttyPSC1“ funktioniert erst, wenn der FPGA-loader im Bootcode entfernt wird, weil dieser dieselben Pins des FPGAs anspricht.
- **Laserscanner Hokuyo URG-04LX:** Dieser wird mit der RS232-Schnittstelle „ttyPSC1“ verbunden. Zusätzlich wird er über das Mainboard mit fünf Volt versorgt.
- **Digitale Outputs:** Der FPGA des Headerboards generiert mit fLink ein digitales Output Signal von 0 bis 3.3 Volt. Mit einer zusätzlichen Sink/Source-Schaltung auf dem Mainboard wird der Signalbereich vergrössert auf 0 bis 24 Volt.
- **Digitale Inputs:** Die digitalen Inputs sind durch eine Transistorschaltung in der Lage, Spannungen von bis zu 30 Volt auszuhalten.
- **Joystick:** Es steht ein XBOX-Kontroller zu Verfügung, welcher über einen USB-Transceiver mit dem Mainboard verbunden ist.

**Digital I/O  
Adapterprint**

Die digitalen Out- und Inputs des Mainboards wurden durch einen Adapterprint nach aussen geführt. Durch diesen Print können alle digitalen Anschlüsse angehängt werden (es existiert kein Schaltplan von diesem Print).

**II. Drive-Board des OmniMoBots („NTB 6xDC“)**

Allgemein	Die Daten zum „NTB 6xDC“ Drive-Board, welches für den OmniMoBot entwickelt wurde, sind auf der CD zu finden <sup>10</sup> . Mehr über den Print ist in der Dokumentation [5] zu lesen.
-----------	--

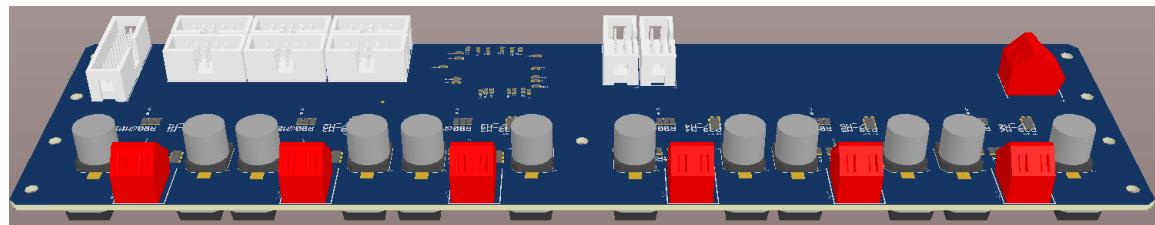
**Print**

Abbildung 91: Drive-Board „NTB 6xDC“

**III. HMI**

Allgemein	Im Rahmen einer Masterthesis [5] wurde ein Benutzerinterface aus vier Tastern und LEDs im Design des NTB-Logos aufgebaut.
-----------	---

<sup>10</sup> Anhang F CD in Ordner: 4\_Elektronik\0\_Prints\Drive-Board(NTB6xDC)

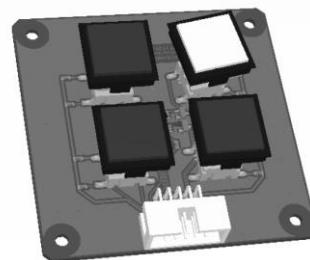
**Print**

Abbildung 92: HMI-Print [5]

- Änderungen** Die Widerstände wurden so verändert, dass die LEDs mit einer Logikspannung von 5 Volt angesprochen werden.

## Schaltplan

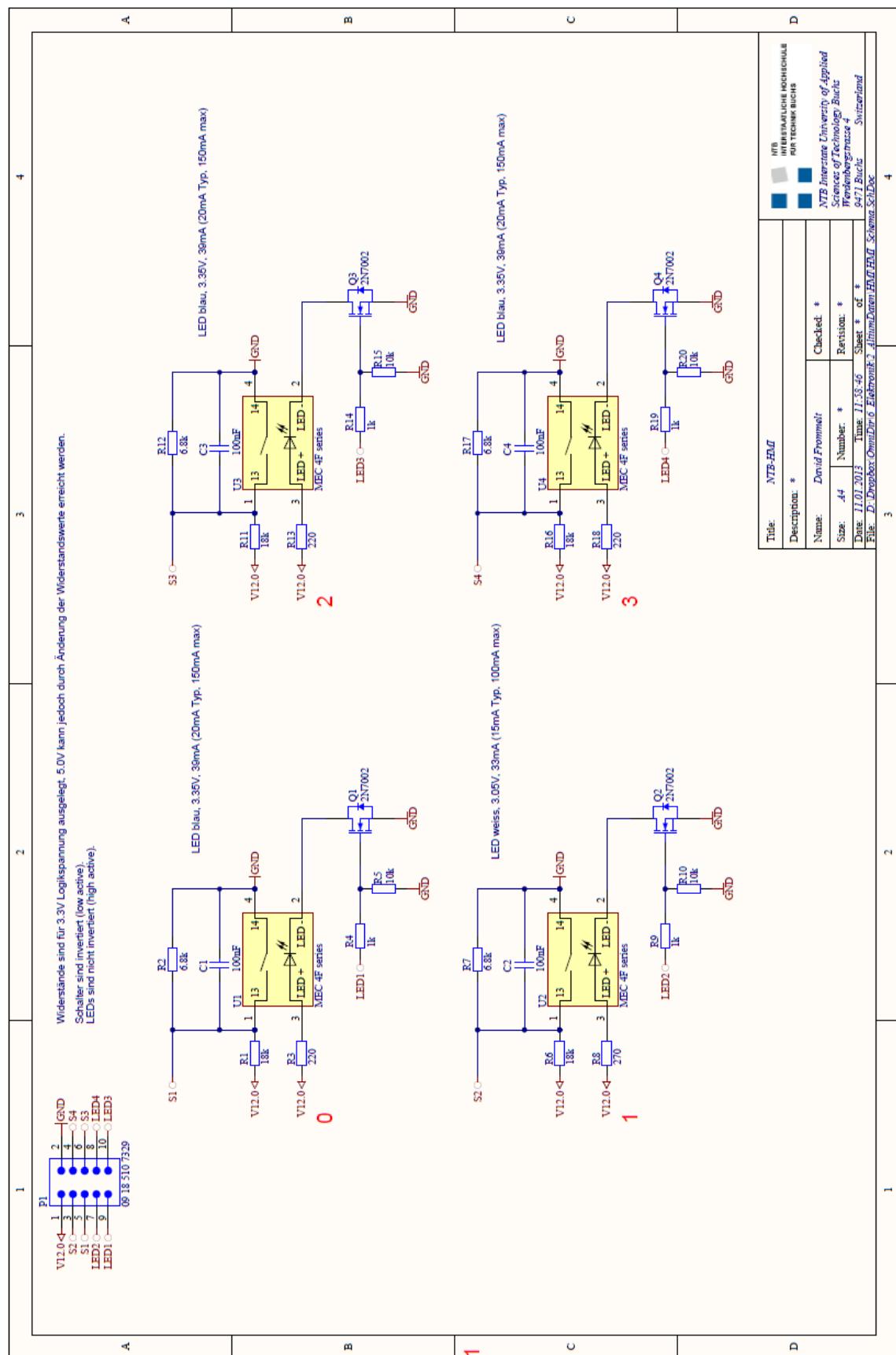


Abbildung 93: Schaltplan HMI-Print

## IV. Hall-Sensorprint

### Allgemein

Der Hall-Sensorprint wurde im Rahmen einer Masterthesis [5] entwickelt und anschliessend in der NTB weiter optimiert. Die Daten zum Hall-Sensorprint sind auf der CD zu finden<sup>11</sup>. Es werden die folgenden Fragen beantwortet:

- Welchen Wert gibt der Sensorprint aus?
- Wie müssen die Ausgaben umgerechnet werden?

### Print

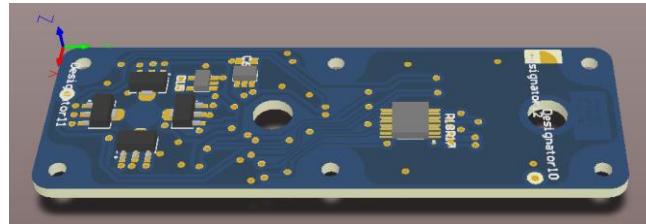


Abbildung 94: Hall-Sensorprint

### Funktions- prinzip

Der Sensorprint besitzt vier Hallsensoren, die folgendermassen angeordnet sind:

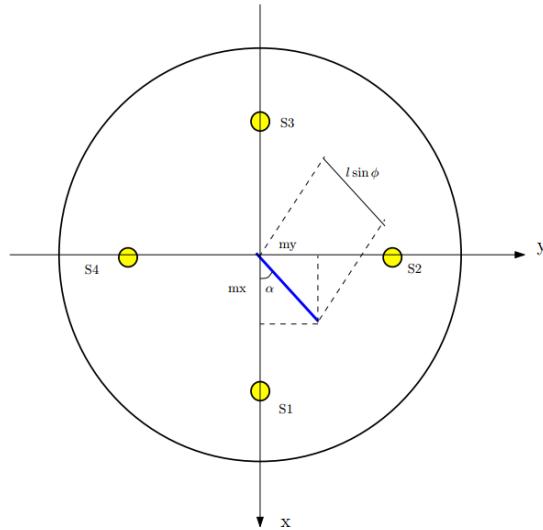


Abbildung 95: Anordnung der Hall-Sensoren [8]

Da nur zwei Freiheitsgrade gemessen werden, müssten nur zwei Sensoren verwendet werden. Damit die Messung jedoch weniger störanfällig ist, werden die Winkel differential gemessen. Die Sensoren (S1, S2, S3, S4) wandeln die einwirkende magnetische Flussdichte fast proportional in eine elektrische Spannung um. Diese Spannung wird anschliessend mit einem ADC digitalisiert. Die Sensoren können positive von negativen magnetischen Flussdichten unterscheiden. Damit ist es möglich, zwei verschiedene Stäbe zu erkennen.

Der Hallsensor-Print muss nur im Arbeitspunkt linear sein, daher reicht eine lineare Abbildung der Differenz der Hall-Sensoren. Somit werden die Winkel  $\phi_x$  und  $\phi_y$  mit folgender Gleichung bestimmt:

$$\begin{bmatrix} \phi_x \\ \phi_y \end{bmatrix} = 0.00175 \cdot \begin{bmatrix} n_{S1} - n_{S3} \\ n_{S2} - n_{S4} \end{bmatrix} \quad (0-6)$$

<sup>11</sup> Anhang F CD in Ordner: 4\_Elektronik\0\_Prints\Sensorprint

Der Faktor 0.00175 wurde im Rahmen einer Masterthesis [5] experimentell ermittelt.

Der Wert  $n_{Si}$  ist die digitalisierte, elektrische Spannung, welche vom Sensor  $i$  ausgegeben wird.

- Kalibrieren** Wie in Abbildung 54, S. 59 zu sehen ist, sind die Messdaten der Hallsensoren zueinander etwas verschoben. Um dies auszugleichen, wird der Mittelwert von jedem Hallsensor, welcher über 50 Sekunden und ohne Einwirkung des Stabes gemessen wird, dem jeweiligen Hallsensorwert abgezogen.  
Folgende Kalibrierungswerte wurden ermittelt:  
**Hall0** = 2020 [V in  $2^{12}$  Bit]  
**Hall1** = 2013 [V in  $2^{12}$  Bit]  
**Hall2** = 2044 [V in  $2^{12}$  Bit]  
**Hall3** = 2028 [V in  $2^{12}$  Bit]

- Hall-Sensor Adapterprint** Die SPI Kommunikation mit dem Hall-Sensorprint erfolgt differentiell. Dadurch ist ein zusätzlicher line driver-receiver notwendig. (siehe Schaltplan „Hall-Sensor Adapterprint“).

## V. Verwendete Motoren

- Technische Daten** Der OmniMoBot verfügt über sechs gleiche Maxon DC-Motoren mit den folgenden technischen Daten:

Motordaten	Kürzel	Wert	Einheit
Nennspannung	$U_n$	24	V
Nennstrom	$i_n$	6	A
Anschlussinduktivität	$L$	$0.0823 \cdot 10^{-3}$	H
Anschlusswiderstand	$R$	0.299	$\Omega$
Leerlaufdrehzahl	$n_{leer}$	7580	$\text{min}^{-1}$
Drehmomentkonstante	$K_m$	0.0302	Nm/A
Rotorträigesmomentsmoment	$J_m$	$1.42 \cdot 10^{-5}$	Kgm <sup>2</sup>
Nennmoment (Dauerdrehmoment)	$M_n$	0.177	Nm
Ticks pro Umdrehung	-	1024	-

Tabelle 5: Maxon DC motor RE 40 148867

- Problem von differenzierten Encoder-Werten** Die Encoder geben die Position des Rotors als Impulse heraus. Dies und die Abtastzeit führen zu einer Quantisierung des Signals. Die Ableitung eines quantisierten Signals führt folglich zu hohen Ausschlägen (siehe Abbildung 96). Diese Ausschläge können bei einer langsamen Bewegung ein Problem werden, zum Beispiel wenn ein Stab auf eine Position geregelt wird.

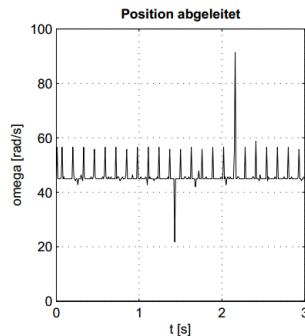


Abbildung 96: Problem von differenzierten Encoder-Werten (Symbolbild) [5]

Eine einfache Lösung ist, das Signal der differenzierten Encoder-Werte mit einem einfachen Tiefpassfilter zu filtern. Dies ist mit einer Gewichtung der neuen und alten Signale möglich:

$$1. \omega_{filtered} = \frac{99 \cdot \omega_{old} + \omega_{new}}{100} \quad 2. \omega_{old} = \omega_{filtered} \quad (0-7)$$

Durch die stärkere Gewichtung des alten Wertes wird das Signal geglättet. Dabei wird jedoch auch die Zeitkonstante des Signals vergrössert.

## VI. Laserscanner

**Beschreibung des Laserscanners** Der Laserscanner „URG-04LX“ ist ein Distanzsensor, es wird ein einzelner Laserstrahl um eine Achse rotiert. Damit werden die Distanzen in einem Winkel von 240° in einer Ebene erfasst. Die Auflösung des Laserscanners bestimmt, wie viele Distanzen damit erfasst werden. Da der Laserscanner immer auf die gleiche Weise (von rechts nach links) die Daten ausgibt, kann mittels des Zeitpunktes der erfassten Distanz und der Auflösung des Laserscanners der jeweilige Winkel der Distanz berechnet werden.

Der Laserscanner wird von der Firma Hokuyo angeboten.



Abbildung 97: Laserscanner "URG-04LX"

Das Datenblatt<sup>12</sup> des Laserscanners sowie die Software<sup>13</sup> sind auf der CD zu finden.

<sup>12</sup> Anhang F CD in Ordner: 4\_Elektronik\2\_Datenblätter\Laserscanner

<sup>13</sup> Anhang F CD in Ordner: 6\_Software\2\_Laserscanner

**Update** Der verwendete Laserscanner wurde auf die Version **SCIP2.0** Standard upgedatet. Damit sind mehr Befehle verfügbar.

**Technische Daten** Der Messbereich wird in Steps vorgegeben:

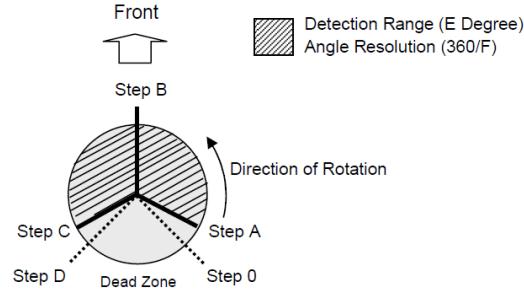


Abbildung 98: Messrichtung und Datenpunkte des Laserscanners

Der Step A ist der erste mit dem eine Messung durchgeführt werden kann, er hat den Wert **44**. Der letzte messbare Step C hat den Wert **725**. Weitere technische Daten:

- **Nennspannung:** 5 VDC
- **Nennstrom:** 500mA, Startstrom bis 800mA
- **Messbereich:** 20mm ~ 5600mm
- **Auflösung:**  $\pm 1\%$  des Messwerts
- **Scanwinkel:** 240°
- **Winkelauflösung:**  $\frac{2\pi}{1024}$

#### Werte Tabelle

URG-04LX	
Lichtquelle	Laser
Abmessungen (BxTxH)	50 x 50 x 70 mm
Gewicht	160 g
Versorgungsspannung	5V DC
Leistungsaufnahme	2,5 W
Reichweite	4 m
Öffnungswinkel	240°
Genauigkeit	$\pm 10\text{mm}$
Auflösung	360°/1024
Scanfrequenz	10 Hz
Schnittstellen	USB 2.0, RS232
Schutzart	Optik: IP64 Gehäuse: IP40

Tabelle 6: Technische Daten von Laserscanner

## VII. Batterie

### Allgemein

Der OmniMoBot wird mit zwei Bleibatterien von Panasonic betrieben.

### Technische Daten



Abbildung 99: LC-R127R2PG1

- Hersteller: Panasonic
- Typ: LC-R127R2PG1
- Spannung: 12 Volt
- Kapazität: 7.2 Ah
- Gewicht: 2470g
- Grösse (LxBxH): 151x 65x 94mm
- Anschluss: Faston 6.3 mm

### Aufladen

Aufladen mit ALCS 2-24 A. Wenn die Batterien vollständig leer sind, muss mit einer Laufzeit von ca. 28 Stunden gerechnet werden.

## E. Schaltplan OmniMoBot<sup>14</sup>

1	2	3	4	5	6	7	8	9	10	11
Schaltungsunterlagen / electrical circuit diagrams										
Projekt: OmniMoBot Version: V1.0.0.PT Autor: S.Landis / NTB										
Änderungen	Datum	Name		Datum	Name	 Titelblatt				
		gez.:	10.04.2015	SLS		 Projekt Bez.: OmniMoBot Projekt Nr.: -				
		gepr.:				Date: Schaltungsunterlagen1.spf7 Version: V1.0.0.PT				
										Seite 1 von 8

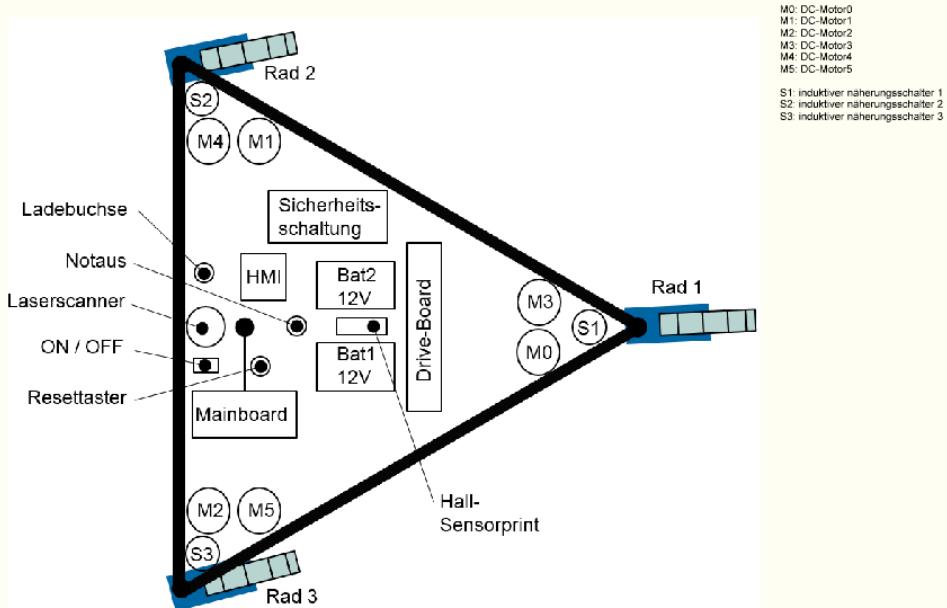
<sup>14</sup> Anhang F CD in Ordner: 4\_Elektronik\1\_Schaltplan

1	2	3	4	5	6	7	8	9	10	11

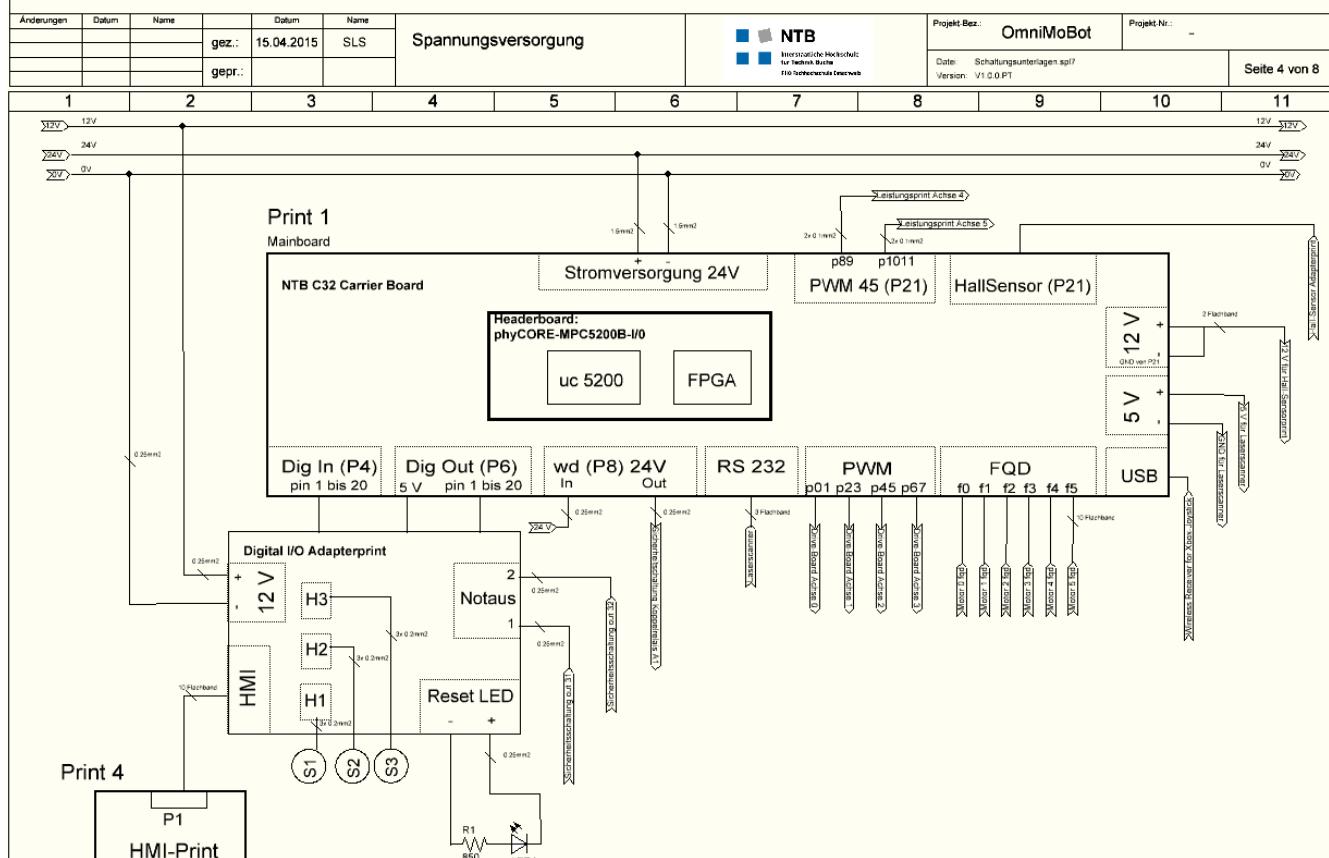
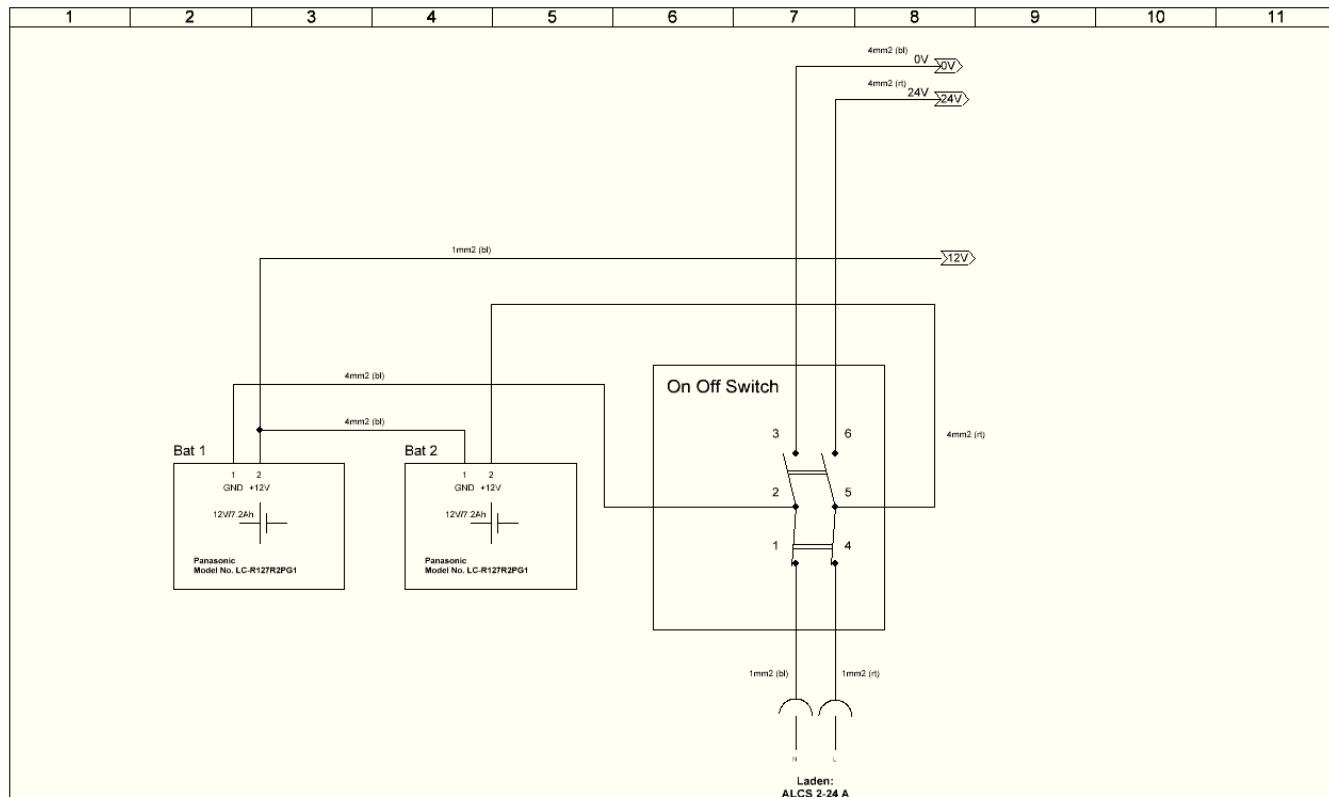
Seite:	Beschreibung:	Datum:	Ersteller:
1	Titelseite	10.04.2015	SLS
2	Inhaltsverzeichnis	10.04.2015	SLS
3	Schaltungsaufbau	10.04.2015	SLS
4	Spannungsversorgung	10.04.2015	SLS
5	Mainboard	03.04.2014	SLS
6	Sicherheitsschaltung	03.04.2014	SLS
7	Drive-Board	03.04.2014	SLS
8	Hall-Sensor Adapterprint	03.04.2014	SLS

Aenderungen	Datum	Name	Datum	Name	Inhaltsverzeichnis	 NTB Interstaatliche Hochschule für Technik Rapperswil Fachhochschule Ostschweiz	Projekt Bez.: OmniMoBot	Projekt Nr.: -
		gez.:	10.04.2015	SLS				
		gepr.:						
					Date: Schaltungsauflagen1.sp7 Version: V1.0.0.PT			Seite 2 von 8

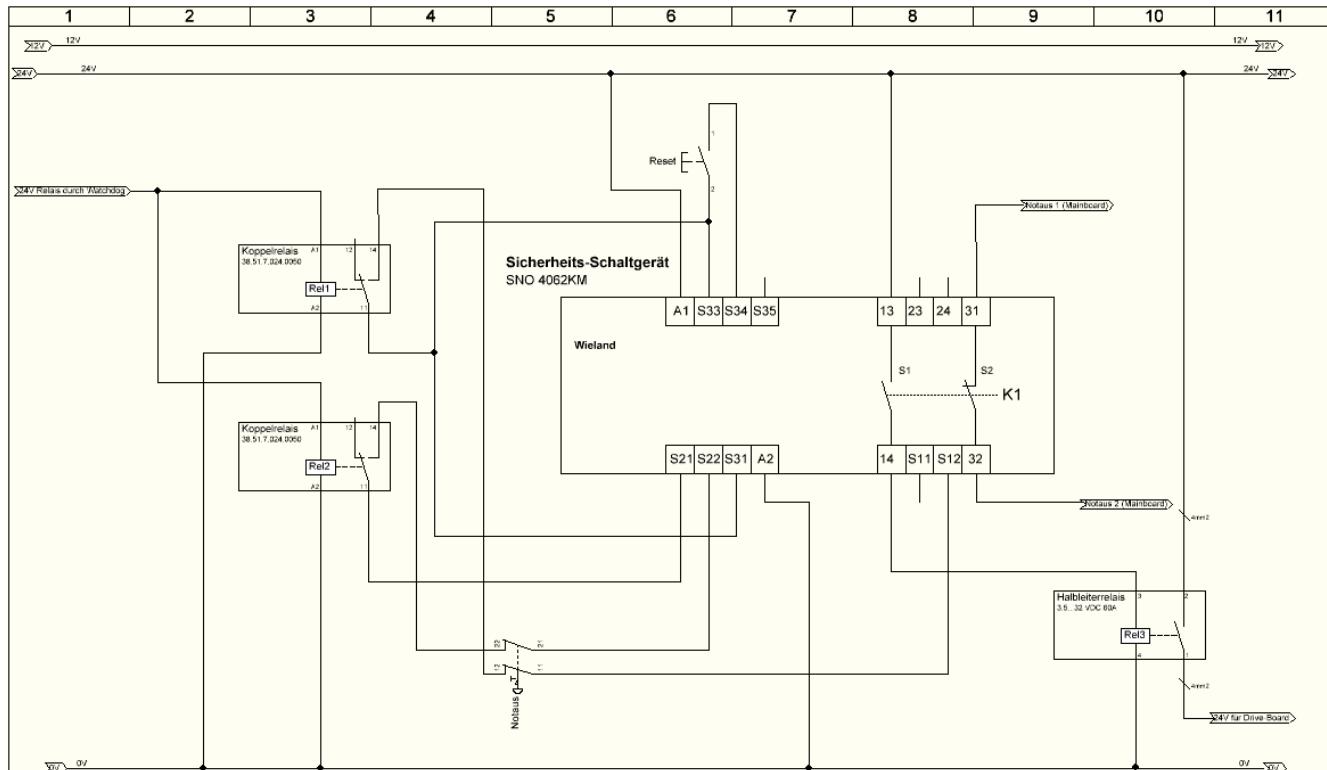
1	2	3	4	5	6	7	8	9	10	11



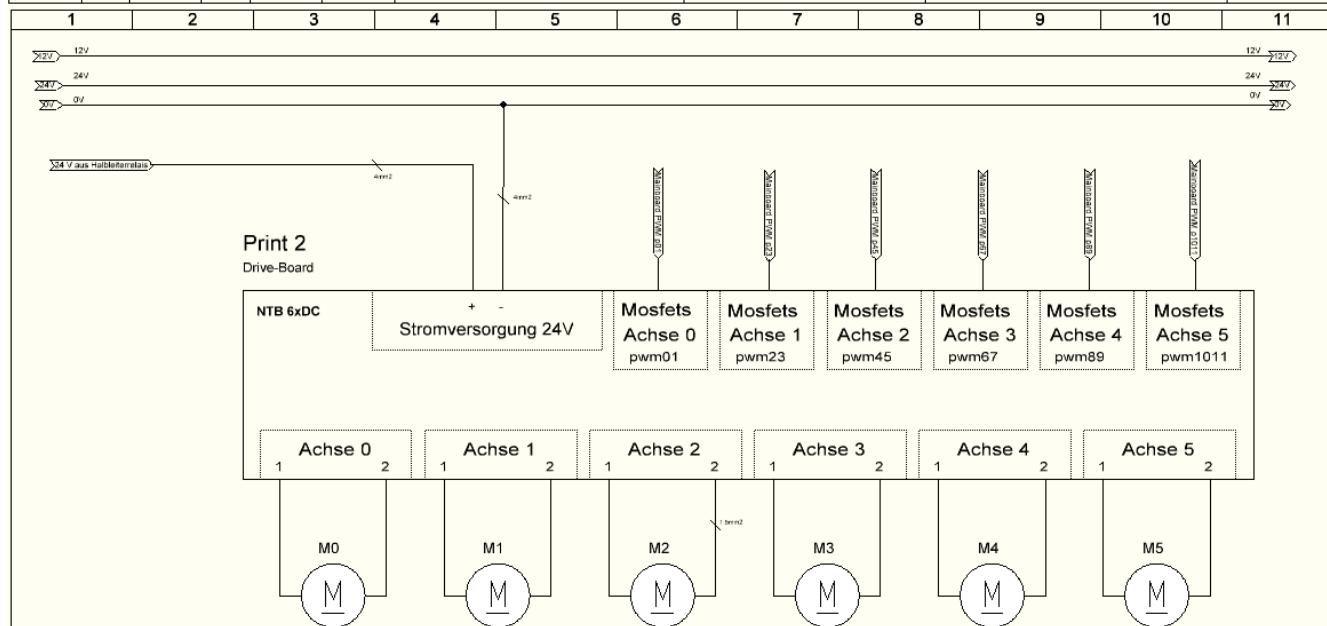
Aenderungen	Datum	Name	Datum	Name	Schaltungsaufbau	 NTB Interstaatliche Hochschule für Technik Rapperswil Fachhochschule Ostschweiz	Projekt Bez.: OmniMoBot	Projekt Nr.: -
		gez.:	10.04.2015	SLS				
		gepr.:						
					Date: Schaltungsauflagen1.sp7 Version: V1.0.0.PT			Seite 3 von 8



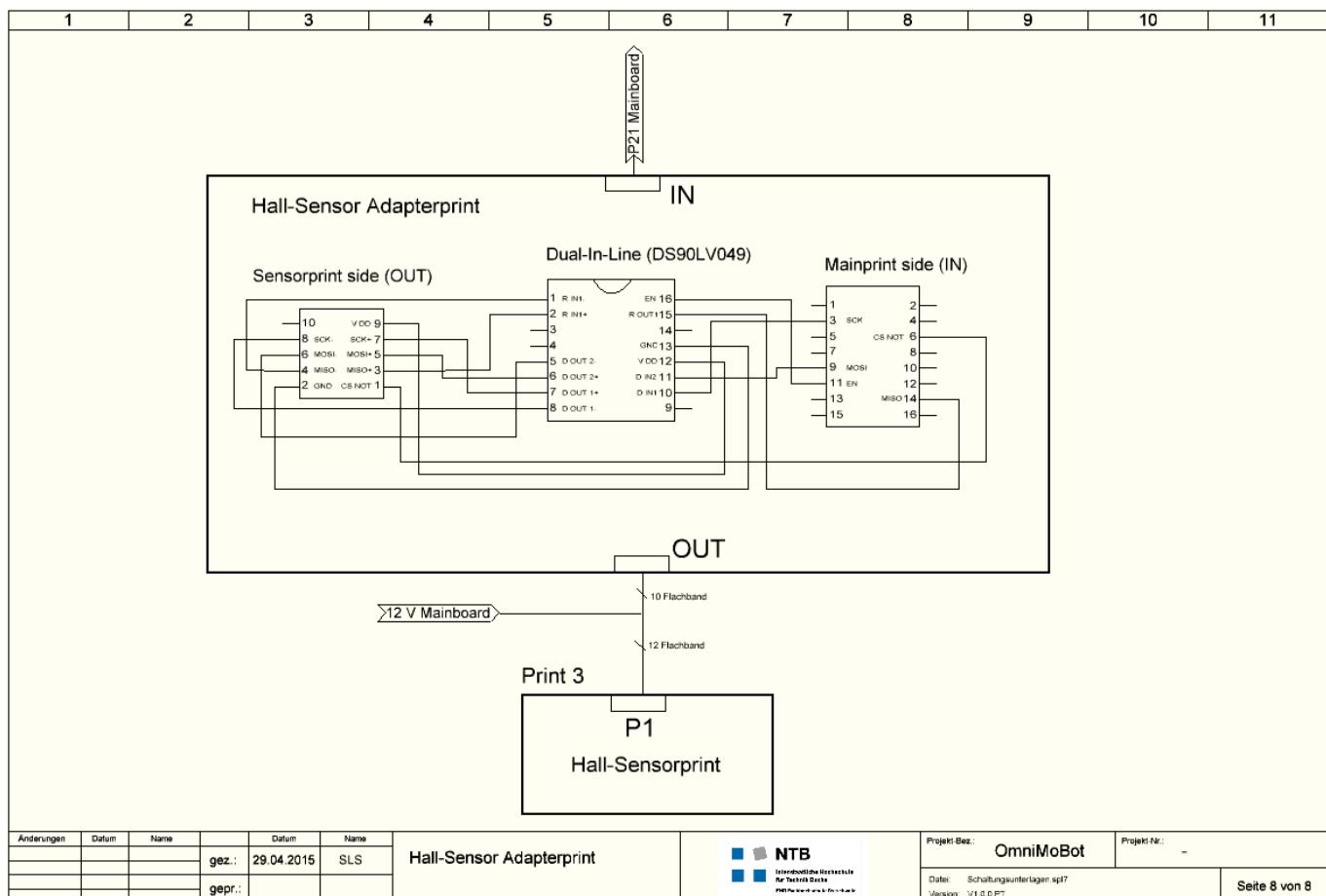
Änderungen	Datum	Name	Datum	Name	Mainboard		Projekt-Bez.	OmniMoBot	Projekt-Nr.:
	gez.:	15.04.2015	SLS				Date:	Schaltungsunterlagen sp1/ Version: V1.0.0 PT	
	gepr.:								Seite 4 von 8



Aenderungen	Datum	Name	Datum	Name	Sicherheitsschaltung		NTB	Projekt Bez.	Projekt Nr.
		gez.:	29.04.2015	SLS				OmniMoBot	-
		gepr.:					Datei: Schaltungsunterlagen spt7 Version: V1.0.D.PT		Seite 6 von 8



Aenderungen	Datum	Name	Datum	Name	Drive-Board		NTB	Projekt Bez.	Projekt Nr.
		gez.:	10.04.2015	SLS				OmniMoBot	-
		gepr.:					Datei: Schaltungsunterlagen 1.spt7 Version: V1.0.C.PT		Seite 7 von 8



Änderungen	Datum	Name	Datum	Name	Hall-Sensor Adapterprint	NTB Interstaatliche Hochschule für Technik Buchs <a href="http://www.ntb.ch">www.ntb.ch</a>	Projekt Bez.:	Projekt Nr.:
		gez.:	29.04.2015	SLS			Datei: Schaltungsunterlagen.spl7	-
		gepr.:					Version: V1.0.PT	Seite 8 von 8

## F. CD

---

**Allgemein**

Die folgenden Ordner sind auf der CD zu finden:

- 0\_Anforderung\_Aufgabenstellung
- 1\_Doku
- 2\_Literatur
- 3\_Mechanik
- 4\_Elektronik
- 5\_Matlab
- 6\_Software
- 7\_Bilder\_Videos