“Tutorial und Testapplikationen zum Bau eines mobilen Roboters auf Basis der RCUBE-Plattform mit dem AKSEN-Board“

D I P L O M A R B E I T
zur Erlangung des akademischen Grades
Diplom-Informatiker (FH)

vorgelegt von Steffen Lehnau
an der Fachhochschule Brandenburg
Fachbereich Informatik und Medien

1. Gutachter: Prof. Dr. H.Loose
2. Gutachter: Dipl.Inf. I.Boersch

Brandenburg, 18. Januar 2005

Aufgabenstellung

“Tutorial und Testapplikationen zum Bau eines mobilen Roboters auf Basis der RCUBE-Plattform mit dem AKSEN-Board“

Es ist zu untersuchen, welche Bestandteile für ein Autonomes Mobiles System notwendig sind und wie diese zusammenwirken. Auf Grund dieser Analysen ist ein autonomer, mobiler Roboter mit modularen, erweiterbaren und vielseitig verwendbaren Bestandteilen zu bauen. Auftretende Schwierigkeiten in der Entwicklung sind anzuführen und in allgemeingültigen Lösungsansätzen zu beschreiben.

Weiter ist zu untersuchen, welche Eigenschaften bzw. Skills ein AMS besitzen soll. Diese Skills sind zu implementieren und in die Bibliothek des AKSEN- Boards aufzunehmen.

Traditionelle Ansätze zur Roboterprogrammierung, wie Weltenmodellierung oder Subsumtionsarchitektur, sind zu beleuchten.

In einem Tutorium sind an Hand des zu entwickelnden universalen, flexiblen Vehikels die Bestandteile und die zu einem komplexen Verhalten zusammengefassten, vorimplementierten Skills zu erklären.

In umfangreichen Tests sollen Funktionsweise, Einfachheit und Robustheit aller Komponenten verifiziert werden.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die vorgelegte Diplomarbeit von mir selbst und nur unter Zuhilfenahme der im Literaturverzeichnis genannten Quellen verfasst wurde.

Brandenburg, 18. Januar 2005

Steffen Lehnau

Inhaltsverzeichnis

Aufgabenstellung II

Selbstständigkeitserklärung III

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Definition „autonom“ . . . 2
- 1.3 Definition „mobile“ . . . 2
- 1.4 Definition „Roboter“ . . . 3
- 1.5 Einsatzgebiete der Roboter . . . 4
  - 1.5.1 Forschung . . . 4
  - 1.5.2 Industrie . . . 4
  - 1.5.3 Medizin . . . 4
  - 1.5.4 Militär/Polizei . . . 4
  - 1.5.5 Unterhaltung . . . 5
  - 1.5.6 Spielzeug . . . 5
  - 1.5.7 Humanoide Roboter . . . 5
  - 1.5.8 Haushalt . . . 5
- 1.6 Zielsetzung . . . 6
- 1.7 Gliederung . . . 6

2 Grundbestandteile eines Robotes 7
- 2.1 Aktoren (Stellglieder) . . . 7
  - 2.1.1 Motoren . . . 8
  - 2.1.2 Getriebe . . . 8
  - 2.1.3 Antriebsarten . . . 9
  - 2.1.4 Sonstige . . . 11
- 2.2 Sensoren . . . 11
- 2.3 Steuerungseinheit . . . 13
  - 2.3.1 Aufbau (AKSEN-Board) . . . 14
  - 2.3.2 Schnittstellen (AKSEN-Board) . . . 14
  - 2.3.3 Das Programm . . . 15

2.4 Kinematik 15
2.5 Energieversorgung 16
2.5.1 Netz 16
2.5.2 Akku/Batterien 16
2.5.3 Solar 16
2.5.4 Brennstoffzelle 16

3 Roboterdesign **17**
3.1 Anordnung der Komponenten 17
3.2 Gehäuse 17
3.3 Kosten 18

4 Roboterprogrammierung **19**
4.1 Funktionen der AKSEN-Bibliothek 19
4.2 Erstes Programm 21
4.3 Weltmodell-Architektur 22
4.4 Subsumtionsmethode 23
4.5 Prozesse und Multitasking 25
4.6 Spezieller Roboterbefehlssatz 26
4.6.1 Bewegungsbefehle 26
4.6.2 Konfigurationsbefehle 27
4.6.3 Verhaltensbefehle 28
4.7 Regelkreise 28
4.7.1 P-Regler 29
4.7.2 PI-Regler 29
4.7.3 PID-Regler 29
4.8 Positionierung und Orientierung 30
4.8.1 Encoder 30
4.8.2 Kompass 31
4.8.3 GPS 31

5 Tutorial Teil 1: Bauanleitung eines AMS **32**
5.1 Konzeption 32
5.1.1 Form 32
5.1.2 Stabilität 33
5.1.3 Schwerpunkt 35
5.2 Stromversorgung 35
5.3 mechanischer Aufbau 36
5.3.1 Motorwahl 36
5.3.2 Getriebewahl 36
5.3.3 Radwahl 39
5.4 Wahl der Antriebsart 39

5.4.1 Nachlauf-/Stützrad 40
5.5 Sensorik und Informationsverarbeitung 41
5.5.1 AKSEN-Board 41
5.5.2 Konstruktion eines optischen Encoders 43
5.5.3 Entfernungsmessung mit IR 46
5.5.4 Sensorinterface zur Linienverfolgung 48
5.5.5 Sensorinterface zur Wandverfolgung 51
5.5.6 Sensorinterface zur Lichtverfolgung 53
5.5.7 Bumper 53

6 Tutorial Teil 2: Programmierung eines AMS **55**
6.1 AKSEN-Roboter konfigurieren 55
6.2 Die Geradeausfahrt 56
6.3 Das Drehen 57
6.4 Fahren im Quadrat 57
6.5 Abfahren einer Fläche 58
6.6 Linienverfolgung 60
6.7 Verhalten avoid_collision 60
6.8 Verhalten bumper 61

7 Zusammenfassung und Ausblick **62**
7.1 Zusammenfassung 62
7.2 Ergebnis 62
7.3 Ausblick 63

Literaturverzeichnis **64**

Abbildungsverzeichnis **67**

Tabellenverzeichnis **68**

A Fotos **69**
A.1 Roboterbeispiele 69
A.2 Weitere Fotos von „Eighteen“ 70
A.3 Auswahl von Segmentscheiben 73

B Messwert-Tabellen **74**
B.1 Encoder-Eingang AKSEN-Board 74
B.2 Sensorinterface Variante 1 75
B.3 Sensorinterface Variante 2 76
B.4 Sensorinterface Wandverfolgung 77

C Quellcode **78**
C.1 Testprogramm Sensorinterface-Linie 78
C.2 Testprogramm AKSEN-Encoder-Eingang 79
C.3 Header arbs.h 83

D CD-Inhalt **85**

1 Einleitung

1.1 Motivation

„Das Jahrhundert der Roboter“ - wie überall zu hören ist, ist nicht nur so dahin gesagt. Im 21. Jahrhundert wird der Roboter mit der gleichen Intelligenz ausgestattet werden wie der Mensch.

> *„In einem neuen Evolutionssprung werden Mensch und Maschine verschmelzen...“* [Ku01]

Im Prinzip wird ein Roboter alles das tun können, was ein Mensch auch kann und noch vieles mehr. Roboter werden in der Lage sein, sich selbst zu reproduzieren. Wird sich dann die Maschinenintelligenz der des Menschen unterordnen oder sogar zu einer Gefahr der Spezies Mensch werden? Dazu schlug schon 1940 Issac Asimaov [As04] die drei Gesetze der Robotik vor, die wie folgt lauten:

1. Ein Roboter darf keinen Menschen verletzen.
2. Ein Roboter muss den Anweisungen gehorchen, die ihm von Menschen gegeben werden, außer wenn diese dem ersten Gesetz widersprechen.
3. Ein Roboter muss seine eigene Existenz solange zu sichern versuchen, wie dies nicht dem ersten oder zweiten Gesetz widerspricht.

Jeder Roboterentwickler sollte sich auch mit gesellschaftlichen Aspekten (ethische Dimensionen) auseinandersetzen, um nicht noch einmal etwas derart Vernichtendes wie die Atombombe zu schaffen.

Die interdiziplinäre Wissenschaft der Roboter ist die **Robotik**. Sie setzt sich aus den Teilgebieten Informatik, Künstliche Intelligenz, Elektrotechnik, Maschinenbau, Mathematik, Kognitionswissenschaften, Psychologie und Biologie zusammen. [St04]

1.2 Definition „autonom“

> *“(griech.) selbstständig, unabhängig - In autonomen Systemen ergeben sich deren zukünftige Zustände und Wirkungen allein aus inneren Ursachen, d.h. sie werden nicht durch äußere Einwirkung beeinflusst.“* [Bh91]

Der Begriff „autonom“ in AMS bedeutet, dass das System sich ohne externe Unterstützung in einer veränderlichen Umgebung bewegen kann. Hier ist es je nach Einsatzumgebung notwendig, entsprechende Sensoren in ausreichendem Maße einzusetzen. Mit den Sensordaten und entsprechender Verarbeitung wird dann die an das System gestellte Aufgabe selbstständig nach eigener Planung ausgeführt. [Kn91]

Verschiedene Robotertypen können in verschiedene Grade ihrer Autonomie unterteilt werden.

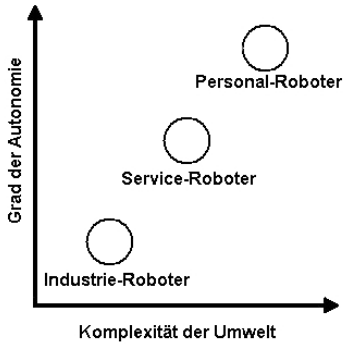

Abbildung 1.1: Autonomie-Grade

1.3 Definition „mobile“

„(lat.-franz.) beweglich, nicht an einen festen Standort gebunden ...“ [Bh91]

Als mobile Roboter werden Maschinen bezeichnet, die sich auf dem Land, im Wasser oder in der Luft frei bewegen. Für die Fortbewegung gibt es eine Vielzahl von Bewegungsarten, wie das Laufen, Springen, Kriechen, Fahren, Schwimmen, Hangeln, Rollen usw. Der einfachste Fall von AMS sind die radgetriebenen Roboter für den Einsatz auf ebenen Flächen, wie auf Straßen oder der Indoor-Umgebung. [Br03]

1.4 Definition „Roboter“

Die Bezeichnung „Roboter“ wurde erstmals 1921 in einem Theaterstück von Karel Capek [Ca11] vom tschechischen „robota“ für „Zwangsarbeit“ verwendet. In diesem Theaterstück „RUR“ („Rossum's Universal Robots“) beschreibt Capek mit „Roboter“ ein mechanisches Gerät, das wie ein Mensch aussieht und menschliche Bewegungen automatisch nachahmen kann. Im Laufe des Stücks entwickelt sich dieser „Roboter“ immer weiter, bis er schließlich seinen Erbauer tötet.

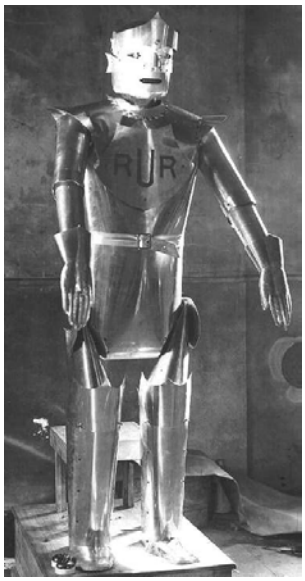

Abbildung 1.2: „RUR“[1]

Definition Industrieroboter nach ISO-Regelung TR 8373 von 1993: Ein Industrieroboter ist ein universelles Handhabungsgerät mit mindestens drei Achsen, dessen Bewegungsmuster frei programmierbar ohne mechanische Hilfsmittel entsteht. Es kann ein Endeffektor, wie z.B. Greifer oder Werkzeug, angebracht werden.

Im allgemeinen Sprachgebrauch wird als Roboter eine Maschine bezeichnet, die dem Aussehen von Lebewesen nachgebildet ist und zumindest teilweise deren Funktionen nachbilden kann.

1.5 Einsatzgebiete der Roboter

1.5.1 Forschung

Roboter werden zur Erforschung solcher Orte, wie Tiefsee und Weltall, eingesetzt, die für den Menschen unzugänglich oder gefährlich sind. Diese tragen Kameras und Instrumente, um Informationen zu sammeln. Beim Einsatz in der Tiefsee werden aus Kostengründen meist ferngesteuerte Systeme eingesetzt. Im Weltall dagegen werden durch die großen Entfernungen und die dadurch bedingte Zeitverzögerung autonome Systeme bevorzugt.

1.5.2 Industrie

In der Industrie ersetzen Industrieroboter den Menschen bei Arbeiten, die sich ständig wiederholen, sehr genau sein müssen oder gefährlich sind. Außerdem können Maschinen rund um die Uhr laufen, brauchen fast keine Pausen und vereinigen sich nicht in Gewerkschaften. Sie fördern die Effizienz zur Herstellung eines Produktes.

1.5.3 Medizin

Der Vorteil von Robotern in der Medizin ist die konstante Arbeitsleistung, unabhängig von mentalen Einflüssen. Die Ausführungen mit hoher räumlicher und zeitlicher Präzision sind reproduzierbar, protokollierbar und aus der Ferne bedienbar. Ein typischer Einsatz ist das Ausfräsen für Hüftprothesen, um einen besseren Sitz und dadurch eine längere Lebensdauer der Prothese zu erreichen. [VR02]
Neben der Präzision der Arbeiten wird auch die Operationszeit verkürzt und damit die Belastung des Patienten verringert.
Neuerdings werden auch prothetische (bionische) Glieder (Prothesen), mit Robotersystemen ausgestattet. Sie bilden natürliche Bewegungsabläufe nach.

1.5.4 Militär/Polizei

Beim Militär oder bei der Polizei werden Roboter eingesetzt, um beispielsweise Bomben zu finden und zu entschärfen. Auch um feindliches Gebiet auszuspionieren werden Roboter genutzt. Im Einsatz sind schon sogenannte „Dronen“ als unbemannte Flugobjekte, die selbständig in feindliches Gebiet eindringen und dort Aufgaben verrichten.

1.5.5 Unterhaltung

Viele kennen die weltberühmten Roboter „R2D2“, „C3PO“ oder „Nummer 5“ aus Filmen, wie „star wars“ oder „Nummer 5 lebt“. Die Filmindustrie kommt heutzutage ohne Spezialeffekte nicht mehr aus. Hier werden Roboter genutzt, um Illusionen bei dem Betrachter hervorzurufen.

1.5.6 Spielzeug

Zu Weihnachten 1998 kam ein Roboterspielzeug namens „Furby“ auf den Markt und wurde in kurzer Zeit sehr populär. Heute gibt es Roboterhunde wie den „Aibo“ (Abbildung A.2 mit erstaunlichen Fähigkeiten. Vor allem aus Japan kommt die Entwicklung des Haustierersatzes. Mit dem „Aibo“ wird auch schon eine Weltmeisterschaft im Fußball ausgespielt. Für kleine bis große Roboterentwickler gibt es den „LEGO Mindstorms“ Baukasten zum Bauen eigener Roboter.

1.5.7 Humanoide Roboter

Humanoide Roboter sind, wie der Name schon sagt, dem Menschen nachgebildet. Sie sind dazu gedacht sich in der Umgebung des Menschen zurechtzufinden. Humanoide Roboter sollen in allen für den Menschen geschaffenen Räumen (z.B. Büros, Krankenhäuser usw.) ohne spezielle Einrichtungen agieren können. Durch die natürlichen, menschlichen Funktionen wird die Akzeptanz und Kommunikation zwischen Mensch und Roboter verbessert. Der zur Zeit wohl beste Roboter mit menschlichen Bewegungsabläufen ist der in Abbildung A.1 gezeigte „ASIMO“.

1.5.8 Haushalt

Wer hätte nicht gern einen Roboter, der einem alle lästigen Arbeiten wie Staubsaugen, Rasenmähen, Fensterputzen usw. im Haushalt abnimmt. Hier liegt auch für die Zukunft ein riesiges Potential, wenn jedes Haus und jede Wohnung mit einem Roboter ausgestattet würde. Ein Beispiel für einen vollautomatischen Roboterstaubsauger ist der „ROBOTER STAUBSAUGER AEG Trilobite lectrolux Bodenstaubsauger“.

1.6 Zielsetzung

Ziel dieser Diplomarbeit ist die Entwicklung eines Autonomen Mobilen Systems (AMS) mit einer modularen, erweiterbaren und vielseitig verwendbaren Konstruktion. So sollen bestimmte Fragestellungen beim Bau eines Roboters beleuchtet und diskutiert werden. Ergebnisse hieraus sollen zu allgemein gültigen Lösungsansätzen zusammengefasst werden.

Die Konstruktion des AMS ist beschränkt auf rutschfeste, ebene Indoor-Umgebungen und auf eine Vortriebsart nach Wahl.

Zentraler Baustein für das AMS ist das AKSEN-Board, dessen Bibliotheksfunktionen erweitert werden sollen. Diese Erweiterungen beziehen sich auf neue Skills (Fähigkeiten) und Behavior (Verhalten). Dadurch entsteht eine höhere Ebene zur Roboter-Programmierung.

1.7 Gliederung

In den ersten drei Kapiteln werden Grundlagen theoretisch betrachtet, auf die im weiteren Verlauf Bezug genommen wird. Zu den Grundlagen gehören die Begriffserklärungen von „Roboter“, „autonom“ und „mobile“, sowie Bestandteile eines AMS.

Im Kapitel Roboterprogrammierung geht es um die unterschiedlichen Arten der Programmierung und um die Entwicklung neuer Funktionen, mit denen der Roboter einfacher programmiert werden kann.

Das Tutorium als Übungskurs unterteilt sich in den Bereich der Bauanleitung (Kapitel 5) und in den Bereich der Programmierung (Kapitel 6). Die Bauanleitung hilft dabei, ein AMS nachzubauen und auf bestimmte Fragestellungen einzugehen. Kapitel 6 beinhaltet die Anleitung zur Nachprogrammierung bestimmter Verhalten.
Das Tutorium richtet sich sowohl an Schüler und Studenten, als auch an erfahrene Roboterentwickler zur Lösung bestimmter Problemstellungen.

Eine Zusammenfassung mit Ergebissen aus den vorherigen Kapiteln beendet die Diplomarbeit. Außerdem wird ein Ausblick gegeben, wie bearbeitete Themen weitergeführt werden und eventuell auf andere Bereiche übertragen werden können.

2 Grundbestandteile eines Robotes

Ein Roboter besteht immer aus mehreren Teilsystemen. Die häufigsten sind die Aktoren, die Sensoren und die Steuerelektronik, ohne die ein Roboter nicht mobil und autonom sein kann. Mit den Sensoren nimmt das AMS seine Umwelt wahr. Mit der Steuerelektronik werden diese Informationen verarbeitet und wiederum daraus die Aktoren angesteuert.

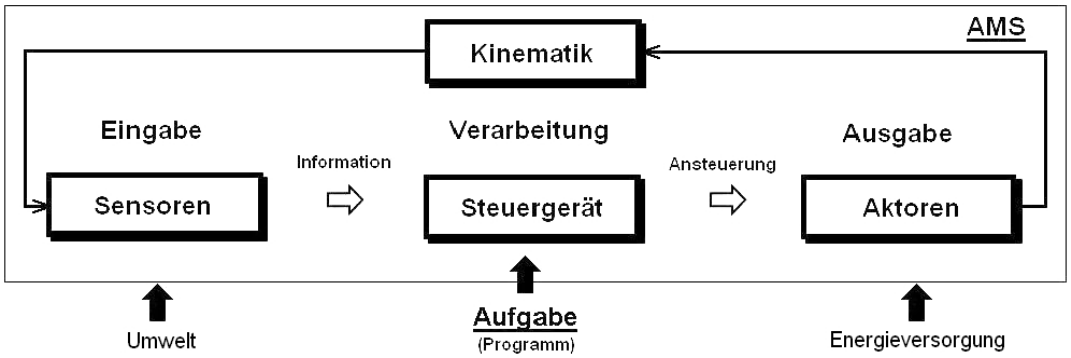

Abbildung 2.1: Ablaufplan

2.1 Aktoren (Stellglieder)

Als Aktoren werden Geräte bezeichnet, die Signale in meist mechanische Arbeit umsetzen. Vorrangig handelt es sich in mobilen Systemen um Antriebsaktoren, die für die Fortbewegung eines Roboters verantwortlich sind. Weiterhin versteht man unter Aktoren Manipulatoren, mit deren Hilfe der Roboter in die Lage versetzt wird, seine Umwelt zu verändern. Bestes Beispiel hierfür ist ein Greifer, der der menschlichen Hand nachempfunden ist.

Auch mit Licht (z.B. Laser) oder Schallwellen kann ein AMS seine Umwelt beeinflussen. Es gibt hauptsächlich drei Arten, elektrische Signale in mechanische Arbeit umzusetzen; das sind die Pneumatik, die Hydraulik und die Elektrik. Die am häufigsten eingesetzte Antriebselemente bei AMS sind die Elektromotoren. Diese sind in der Regel einfach anzusteuern und leicht in das System zu integrieren.

2.1.1 Motoren

Motoren sind Maschinen, die zum Bewegen und Antreiben von mechanischen Geräten verwendet werden. Motoren, die elektrische Energie in mechanische Energie umwandeln, sind einfache Elektromotore, Servomotore und Schrittmotore.

Ein **Elektromotor** ist ein elektrisches Gerät, das mit Hilfe von magnetischen Feldern hauptsächlich elektrische in mechanische Arbeit umwandelt. Elektromotoren zeichnen sich aus durch hohe Zuverlässigkeit, ein günstiges Verhältnis von Leistung bezogen auf das Gewicht und sehr hohe Wirkungsgrade über einen großen Leistungsbereich.

Ein **Servomotor** ist ein Elektromotor mit einer Einrichtung zur Bestimmung des zurückgelegten Drehwinkels. Beispielsweise handelt es sich hier um ein Potentiometer, welches die aktuelle Ankerstellung an ein elektronisches System liefert. Dieses System vergleicht den Ist- mit dem Sollwert und regelt gegebenenfalls bis zur Verringerung der Abweichung nach. Die Vorteile von Servomotoren bei Positionierungsaufgaben sind vor allem die hohe Dynamik und Präzision.

Ein **Schrittmotor** ist ein Elektromotor, der in Schritten arbeitet und sehr präzise eine Position auf dem Umfang seiner Kreisbewegung ansteuern kann. Eingesetzt wird er vor allem in Druckern und in Laufwerken. Der Vorteil eines Schrittmotors besteht darin, dass er exakt an einer bestimmten Position angehalten werden kann. Nachteile sind der hohe Preis und die aufwändige Ansteuerung.

2.1.2 Getriebe

Getriebe dienen zur Übertragung und Umformung von Bewegungen, Energie und Kräften. Es gibt bei den mechanischen Getrieben die formschlüssigen Getriebe, wozu Zahnradgetriebe, Schneckengetriebe, Kettengetriebe und Zahnriemengetriebe gehören und die kraftschlüssigen Getriebe, wie Reibradgetriebe, Riemengetriebe und Wälzkörpergetriebe. Beim Bau eines AMS werden Getriebe vorwiegend eingesetzt, um die Drehzahl und das Drehmoment an die Erfordernisse anzupassen. Die folgenden Formeln dienen zur Berechnung der Getriebe aus den Abbildungen 2.2 und 2.3.

$$i_{ges} = \frac{z_2 \cdot z_4}{z_1 \cdot z_3} = \frac{n_1}{n_4} \tag{2.1}$$

$$i_{ges} = i_1 \cdot i_2 \tag{2.2}$$

$$n_1 \cdot z_1 = n_2 \cdot z_2 \tag{2.3}$$

$$M_4 = M_1 \cdot i_{ges} \tag{2.4}$$

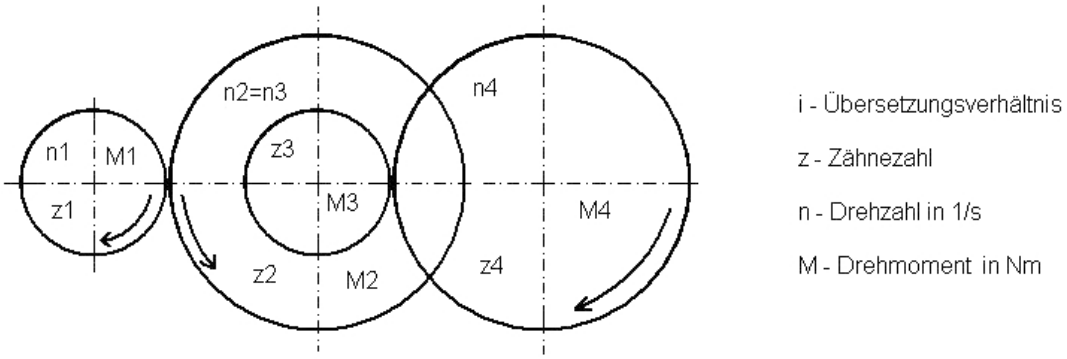

Abbildung 2.2: Getriebe

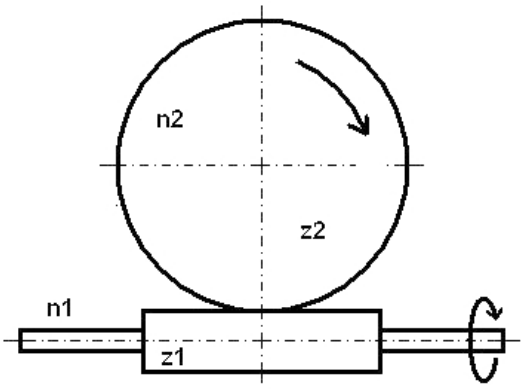

Abbildung 2.3: Schneckenrad

2.1.3 Antriebsarten

Je nach Einsatzgebiet muss für ein AMS eine spezielle Antriebsart gewählt werden. Abgesehen von Rädern sind diese meistes von Lebewesen abgeschaut. Nachgebildet wurden dann das Fliegen, Schwimmen, Laufen, Kriechen, Rollen usw. Für den Einsatz eines AMS in der Luft, im Wasser oder im Weltraum sind spezielle Einrichtungen (z.B. Flügel) notwendig. Der häufigste Einsatz ist jedoch auf dem Land zu finden. Hier sind wiederum unterschiedliche Oberflächen zu beachten. Ist die Umgebung sehr uneben und lose beschaffen, bieten sich Kettenantrieb, Schreitantrieb oder Kriechantrieb an. Beispiele für diese Antriebsarten sind in der Abbildung 2.4 zu sehen.

Abbildung 2.4: Spreng-[4],Schreit-[5],Kriech-Roboter[6]

Die am häufigsten entwickelten Systeme sind für den Indoor- Bereich gedacht. Hier bietet der ebene und rutschfeste Untergrund die optimalen Bedingungen für den Einsatz von Rädern. Diese Arbeit wird sich ausschließlich mit den Radantrieben befassen. Roboter mit Rädern haben eine einfache Mechanik und lassen sich leicht zusammenbauen. Die Anordnung und Funktion der einzelnen Räder ist verantwortlich für die verschiedenen Bewegungseigenschaften. So kann ein Rad die Funktion zum Lenken, Antreiben oder Stützen allein oder in Kombination übernehmen.

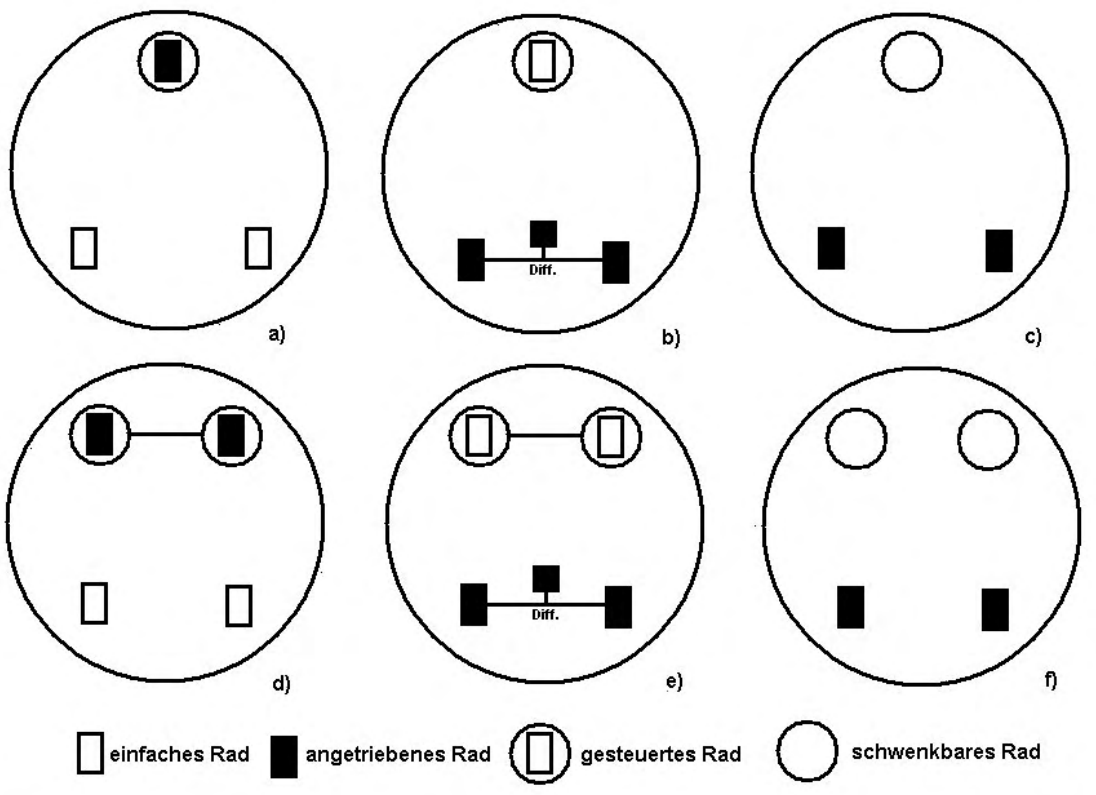

Abbildung 2.5: Einige Radanordnungen

In der Abbildung 2.5 sind verschiedene Lenk- und Steuergeometrien denkbarer Roboterkonstruktionen, aber längst nicht alle, dargestellt. In den ausgewählten Beispielen werden immer zwei Motoren eingesetzt. Entweder ein Motor für den Vortrieb und einer zur Lenkung (Abb.2.5 a,b,d,e) oder beide Motoren zum Vortrieb und zur Steuerung (Abb. 2.5 c,f). Es gibt noch eine große Menge von Radanordnungen mit verschiedenen Radtypen, die hier aus Gründen der Konzentration auf ein einfaches AMS nicht ausführlich erklärt werden.

2.1.4 Sonstige

Um die Roboter-Mensch-Schnittstelle zu verbessern, werden Roboter mit Systemen ausgestattet, die beim Menschen Emotionen hervorrufen. Dies soll es dem Menschen erleichtern, mit dem Roboter in Kontakt zu treten. Hier kommen zum Beispiel künstliche Muskeln zum Einsatz, die eine Gesichtsmimik imitieren.

2.2 Sensoren

> *" Ein Sensor (zu lateinisch sensus „Gefühl") oder (Mess-)Fühler ist in der Technik ein Bauteil, das neben bestimmten physikalischen oder chemischen Eigenschaften (z.B.: Wärmestrahlung, Temperatur, Feuchtigkeit, Druck, Helligkeit, Magnetismus, Beschleunigung, Kraft) auch die stoffliche Beschaffenheit seiner Umgebung qualitativ oder als Messgröße quantitativ erfassen kann."* [WI05]

Die bereitgestellten Sensordaten unterliegen im Allgemeinen einer Störung. Als Vorbild für die Enwicklung von Sensoren diente die Natur. So wurde die Kamera dem Auge nachempfunden, der Drucksensor dem Tastsinn unserer Haut und das Radar der Ultraschallortung von Fledermäusen.

Sensoren werden in zwei Sensorgruppen unterteilt:

- interne Sensoren zur Erfassung der internen Zustände (z.B. Geschwindigkeitssensoren oder Positionssensoren)
- externe Sensoren, die Informationen über den Zustand der Umwelt liefern (z.B. Abtastsensoren oder visuelle Sensoren)

Die Abbildung 2.6 gibt einen Überblick über die Klassifikation von Sensorsystemen in der Robotik.

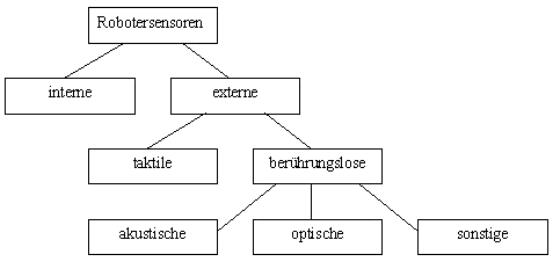

Abbildung 2.6: Klassifikationsschema für Sensorsysteme der Robotik

Messgröße	**Sensor**	**Variante**	**Preis**
Kollision	Schaltkontakte	MIKRO-SCHALTER 250VAC/3A	1,51€
	IR-Sensor	PHOTO-TRANS. SFH309FA-3/4/Q62702-P3590	0,32€
	kapazitiver Sensor	SENSOR-SCHALTER	40,39€
Kraft	Kraftsensor	DRUCKSENSOR MPXA4100A6U	25,54€
Entfernung	Sharp-Sensoren	DISTANZ-SENSOR GP2D120	25,54€
	Ultraschall-Sensoren	Sender - MA40S	4,58€
		Empfänger - MA 40 E7 R	17,87€
	Radar-Sensoren	SMX-1 (Siemens)	49,95€
Licht	Fotodiode	PHOTODIODE SFH213FA/Q62702-P1671	0,55€
	Fototransistor	PHOTO-TRANS. SFH309-5/6/Q62702-P3594	1,00€
Schall	Mikrofon	MINI-MIKROFON MIT CLIP	15,95€
Wärme	Widerstand	KTY84-130	1,51€
Orientierung/	Potentiometer	POTENTIOMETER PT 10 LV	0,38€
Positionierung	Rad-Encoder	LICHTSCHRANKE GP1A71R + TAKTSCHEIBE 120	19,40€
	Drehimpulsgeber	DREHIMPULSGEBER 427-011151AL021	6,95€
	Gyroskop	MINI GYRO	54,95€
Magnetfeld	Hall-Schalter	UNIPOLARER HALLSCHALTER	1,15€
	Reed-Schalter	REED-SCHALTER 6 VA	1,51€

Tabelle 2.1: Sensorenübersicht(Quelle:Conrad '05)

2.3 Steuerungseinheit

Die Steuerungseinheit ist das Herz jedes AMS. Hier werden Informationen der Sensoren gesammelt, verarbeitet und die Aktoren (an)gesteuert. Die Steuerungseinheit kann aus einer analogen elektronischen Schaltung bestehen. Dort sind Änderungen nur sehr aufwändig durch eine Manipulation der Schaltung möglich. Moderner und verbreiteter ist der Einsatz von Mikrocontrollern oder PC's. Durch die Software bzw. das Programm bekommt das System sein Verhalten. Änderungen können leicht durch die Manipulation des Programms durchgeführt werden. Bekannte Steuerungseinheiten sind das 6.270-Board, RCX von LEGO, C-Control usw.

Geeignet für die Steuer- und Regelungsaufgaben in einem AMS ist das AKSEN-Board mit dem Mikroprozessor SAB 80C515A im Kern. Das AKSEN-Board (**Ak**toren/**Sen**soren) ist ein Kernmodul aus der RCUBE-Plattform. Diese wurde von der Fachhochschule Brandenburg für autonome intelligente Systeme entwickelt. Weitere Kernmodule, wie VIO-Board (Video-Board) und CPU-Board lassen sich mit dem AKSEN-Board über einen Feldbus zu einem kostengünstigen, portablen und mobilen System verbinden.

Nähere Erläuterungen zum AKSEN-Boards finden Sie auf der beigefügten CD und unter [KI05].

Abbildung 2.7: AKSEN-Board [KI05]

2.3.1 Aufbau (AKSEN-Board)

Das AKSEN- Board besitzt folgende Merkmale [KI05]:

- 15 analoge Eingänge (Sensoren für Licht, Infrarot, Abstand, Linien, Akku usw.)
- 16 digitale Ein-/Ausgänge (frei konfigurierbar als Ein-/ oder Ausgang)
- 4 Motortreiber (in Drehzahl und Richtung variierbar)
- 4 schaltbare Leistungstreiber (z.B. Infrarotsender, Lämpchen, LED)
- 3 Servo-Ausgänge (durch Software erweiterbar)
- 1 Infrarotausgang mit Leistungstreiber
- 3 Encoder-Eingänge zum Erfassen von Drehzahlen (z.B. Odometrie)
- vierfach-DIP-Schalter
- V.24-Schnittstelle o zweizeiliges LCD
- 64 KB Flash, 8 KB Flash
- CAN-Interface 1 Mbit (optional)
- Bluetooth-Verbindung zum PC (optional)
- Mikroprozessor SAB 80C515A

2.3.2 Schnittstellen (AKSEN-Board)

Damit die Steuerungseinheit mit anderen Geräten kommunizieren kann, ist sie mit einigen Schnittstellen ausgerüstet. Einige sollen kurz beschrieben werden:

Die **Serielle Schnittstelle** bezeichnet den Ein-/Ausgang eines Computers oder eines Peripheriegerätes. Die Bits werden bei der seriellen Datenübertragung nacheinander übertragen.

I2C (Inter-Integrated Circuit) ist ein serieller Bus für Computersysteme. Er dient zum Anschluss von Geräten mit geringer Übertragungsgeschwindigkeit an ein Embedded System. Der Bus arbeitet im Standard-Modus mit 100 kbit/s über zwei bidirektionale Pins (Takt und Daten).

Das **CAN** (Controller Area Network) verbindet mehrere gleichberechtigte Komponenten über einen 2-Draht Bus miteinander. Das CAN-Protokoll wurde 1983 von Bosch für den Einsatz in Kraftfahrzeugen entwickelt. Das CAN Netzwerk kann prinzipiell Bitraten bis zu 1 Mbit/s übertragen.

2.3.3 Das Programm

Die Programmierung erfolgt üblicherweise an einem PC. Dort wird das Programm in einer mikrocontrollernahen Assembler-Sprache oder in einer Hochsprache wie C geschrieben. Dieses Programm wird dann in die eigentliche Maschinensprache kompiliert und in den Speicher des Mikrocontrollers geschrieben („flashen“). Hier kann das Programm zur Ausführung gebracht werden und bleibt auch ohne Stromversorgung resistent.

Weitere Informationen zum Schreiben, Kompilieren und Flashen mit dem AKSEN-Board finden Sie im Handbuch des Boardes.

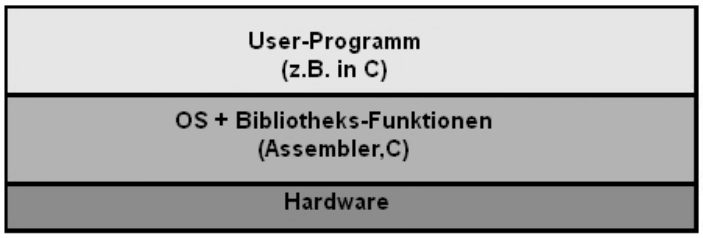

Abbildung 2.8: Programmstruktur

In der Abbildung 2.8 ist zu sehen, wie Software und Hardware auf verschiedenen Ebenen zusammenarbeiten. Von der höheren bis zur unteren Schicht wird die Formulierung eines Algorithmus bis zu einem für einen Computer ausführbaren Code angepasst.

Das für einen Roboter existentielle Programm kann mit seinen Eigenschaften auch als Komponente eines Roboters angesehen werden. Im Kapitel 4 werden die unterschiedlichen Roboterarchitekturen beschrieben.

2.4 Kinematik

Die Kinematik ist die Lehre der Bewegung. In diesem Fall bezieht sich die Kinematik auf die Bewegungsabläufe des Roboters.
Von Interesse ist es, wo sich der Roboter befindet und wie er an eine bestimmte Position kommt. Bei Robotern mit Rädern, die sich in der Ebene bewegen, sind drei Freiheitsgrade von Bedeutung. Das sind die X-Y-Koordinaten und der Winkel, d.h. die Richtung, in die der Roboter zeigt. [JF96]

2.5 Energieversorgung

Ein nicht zu unterschätzender Bestandteil des ASM ist die Stromversorgung. Was nützt die Unabhängigkeit, wenn kein Strom mehr da ist. Mögliche Versorgungseinheiten sollen hier beschrieben werden.

2.5.1 Netz

Ist das Einsatzgebiet klein und überschaubar, kann man sich auch für eine drahtgebundene Stromversorgung entscheiden. Hier sind die Kabel so anzuordnen, dass sie das AMS nicht behindern. Eine Variante ist, das Kabel zentral von der Decke hängen zu lassen, so dass der Roboter jeden Punkt im Einsatzgebiet erreichen kann und das Kabel nicht den Boden bzw. den Roboter berührt.

2.5.2 Akku/Batterien

Ein Akkumulator, bzw. eine Batterie, ist eine elektromechanische Einrichtung, in der elektrische Energie in Form von chemischer Energie gespeichert ist und nach Bedarf entnommen werden kann. Es gibt unterschiedliche Arten, wie Alkaline, Lead-Acid (Säure-Blei), Lithium, Mercury (Quecksilber), NiCd, NiMH, Silver, Zinc-Air und Carbon-Zinc.

2.5.3 Solar

Solarmodule wandeln Sonnenlicht in elektrische Energie um. In diesem Fall besteht eine gewisse Abhängigkeit vom Sonnenlicht und das AMS wäre in seiner Autonomie eingeschränkt. Die Größe der Module kann die Größe und Form des AMS beeinflussen.

2.5.4 Brennstoffzelle

Bei den Brennstoffzellen handelt es sich um eine Zukunftstechnologie, bei der die elektrische Energie aus Wasserstoff gewonnen wird. Bei diesem Prozess fallen so gut wie keine Schadstoffe an. Für ein autonomes, mobiles System wäre eine Kombination aus Solarmodul und Brennstoffzelle denkbar.

3 Roboterdesign

Im Prinzip sind der Phantasie jedes Roboterentwicklers keine Grenzen gesetzt, wenn es darum geht, dem Roboter seine Gestalt zu geben. Nur sind je nach Einsatzgebiet, Aufgabenstellung und Kosten, bestimmte Kriterien zu beachten. Diese sollen im folgenden Kapitel näher erläutert werden.

Hinzuweisen ist bei dem Design auch auf die Lego-Technik. Mit Hilfe der Lego- Bausteine können Prototypen leicht entwickelt werden. Änderungen sind einfach und bequem vorzunehmen.

3.1 Anordnung der Komponenten

Bei der Anordnung der Komponenten ist darauf zu achten, dass sie unkompliziert und wartungsfreundlich ist, d.h. alle Teile sollten leicht zugänglich sein. Außerdem ist auf die Gewichtsverteilung zu achten, um eine Schlagseite zu verhindern. Auf Grund der Gewichtsverteilung haben sich Gebilde entwickelt, bei denen die zum Antrieb benötigten Teile zentral angeordnet sind. Die Akku-Pakete sind ideal dafür das Fahrzeug auszubalancieren. Wegen des Zugangs zur Steuerungseinheit sind sie oft auf dem System platziert. Die Sensoren werden, je nach Funktion, im äußeren Bereich des Roboters angebracht.

3.2 Gehäuse

Das Gehäuse sollte das AMS gegen äußere Einwirkungen schützen und demzufolge robust sein. Außerdem sollte es leicht sein und dem Roboter ein ansprechendes Äußeres verleihen.

> *„Die Form eines Roboters kann einen großen Einfluss auf seine Leistungsfähigkeit haben. Ein nicht- zylindrischer Roboter läuft eher Gefahr als andere, bei einer ungünstigen Anordnung der Hindernisse in zu engen Passagen oder überfüllten Räumen steckenzubleiben.“* [JF96]

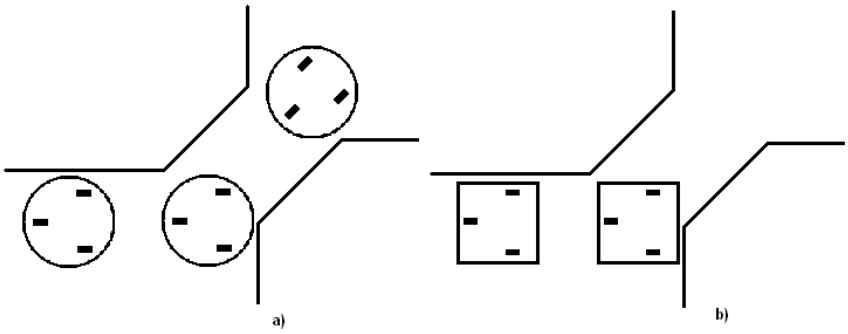

Abbildung 3.1: Roboterformen

In der Abbildung 3.1 ist zu sehen, dass der runde Roboter a) die Passage mit einer einfachen Drehbewegung leichter durchqueren kann als der quadratische Roboter b) bei gleicher Breite.

3.3 Kosten

Die Kosten sind vor allem abhängig von der Komplexität des AMS. Steigt zum Beispiel mit der Wahl der Sensoren die Flexibilität, dann steigen sehr wahrscheinlich auch die Kosten. Je nach Einsatzgebiet und Anforderungen sind spezielle Größen (Nano-Roboter) und verschiedene Materialien ein Maß für die Kosten. Wie fast immer, sind auch hier für die Kosten nach oben keine Grenzen gesetzt.

In den letzen Jahren ist es immer häufiger möglich geworden, kleine Bausätze zu erwerben. Zum Beispiel sind mit LEGO-MINDSTORMS ROBOTICS INVENTION schon beachtliche Ergebnisse erzielt worden. In der folgenden Tabelle werden mögliche Kosten aufgelistet, die notwedig sind, ein kleines, autonomes, mobiles System aufzubauen. Die Preise sind zum Teil aus dem AKSEN Shop unter [AS05].

Bestandteil	**Kosten in [€]**
AKSEN-Board	239,-
Akkupack mit Schnellladegerät	159,-
Lego-Bausatz	ca. 50,-
Sensorkit	199,-
Aktorenkit	189,-
Summe	836,-

Tabelle 3.1: Kostenbeispiel [AS05]

4 Roboterprogrammierung

Ein Roboter muss frei programmierbar sein. Grundsätzlich kann man die Programmierung in zwei Gruppen unterteilen. Das sind die on-line Methode, bei der die Programmierung direkt am Roboter stattfindet, und die off-line Methode, bei der die Programmierung ohne Roboter stattfindet und nur zum Testen notwendig ist. [St04]

4.1 Funktionen der AKSEN-Bibliothek

Das AKSEN-Board besitzt neben den C-Standardfunktionen auch eine Menge an Funktionen zum Nutzen der Eigenschaften des AKSEN-Boards. Diese werden in den drei folgenden Tabellen kurz beschrieben. Ausführliche Informationen befinden sich im AKSEN-Handbuch (beigefügte CD) auf den Seiten 43 bis 49.

Funktion	**Kurzbeschreibung**
Encoder, Servos und Motoren	
void motor_pwm(unsigned char motor,unsigned char geschw);	schaltet Motor-Port *motor* auf *geschw*
void motor_richtung(unsigned char motor,unsigned char richtung);	schaltet Drehrichtung des Motor-Port *motor*
void servo(unsigned char sv, unsigned int laufzeit);	stellt Pulsweite *laufzeit* am Servo-Port *sv* ein
void servo_arc(unsigned char sv, unsigned char winkel);	dreht aus Position *winkel* am Servo-Port *sv*
void servoD0_on(void);	Digital-Port 0 als Servo-Port
void servoD0_off(void);	Digital-Port 0 auf Normal-Betrieb
void servoD0(unsigned int laufzeit);	stellt Pulsweite laufzeit am Digital-Port 0
unsigned int encoder0(void);	Anzahl der neg.Flanken am Encoder-Port 0
unsigned int encoder1(void);	Anzahl der neg.Flanken am Encoder-Port 1
unsigned int encoder2(void);	Anzahl der neg.Flanken am Encoder-Port 3
Analoge und Digitale Anschlüsse	
unsigned char analog(unsigned char pin);	gibt an Spannung an Port *pin* zurück
void led(unsigned char laempchen, unsigned char wert);	schaltet LED-Port *laempchen* ein/aus
void digital_out(unsigned char ausgabepin, unsigned char wert);	schaltet Digital-Port *ausgabepin* ein/aus
unsigned char digital_in(unsigned char pin);	gibt Pegel an Port *pin* zurück

Tabelle 4.1: AKSEN-Funktionen Teil1(Version 0.956)

Funktion	**Kurzbeschreibung**
Multitasking	
unsigned char process_start(void(*funktionszeiger(),unsigned char zeit);	startet neuen Prozess
unsigned char process_kill(unsigned char pid);	beendet Prozess mit *pid*
void process_hog();	Tickzähler des akt.Prozesses auf 255
void process_defer();	veranlasst vorzeitigen Prozesswechsel
void process_set_ticks(unsigned char pid, unsigned char ticks);	Neuzuweisung einer Prozess-Zeit
unsigned char process_get_pid();	PID des akt.Prozesses ermitteln
LCD-Display	
void lcd_cls(void);	LCD löschen
void lcd_home(void);	Cursor auf 0,0
void lcd_setup(unsigned char DisplayOnOff, unsigned char CursorOnOff, unsigned char CursorBlink);	erlaubt grundlegende Einstellungen
void lcd_setxy(unsigned char zeile, unsigned char spalte);	Cursor an Position (*zeile,spalte*)
void lcd_putchar(char c);	gibt *c* aus
void lcd_puts(const char *string);	gibt *string* aus
void lcd_ubyte(unsigned char c);	gibt *c* aus
void lcd_byte(char wert);	gibt *wert* aus
void lcd_hbyte(unsigned char wert);	gibt *wert* aus
void lcd_uint(unsigned int i);	gibt *i* aus
void lcd_hint(unsigned int i);	gibt *i* aus
void lcd_ulong(unsigned long wert);	gibt *wert* aus
void lcd_long(long wert);	gibt *wert* aus
void lcd_hlong(unsigned long wert);	gibt *wert* aus
Erweiterungen	
void CanInit(void);	initialisiert CAN-Schnittstelle
int CanEmpfang(struct CAN_MSG xdata *CanBotschaft);	empfängt CanBotschaft von der CAN-Schnittstelle
int CanSenden(struct CAN_MSG xdata *CanBotschaft);	sendet CanBotschaft
void serielle_init(void);	initialisiert serielle Schnittstelle
void serielle_putchar (char c);	sendet Zeichen *c* über die serielle Schnittstelle
void serielle_puts (const char* string);	sendet null-terminierten *string*
char serielle_gets (char * string);	schreibt String von der seriellen Schnittstelle in *string*
Diverses	
unsigned char dip(void);	gibt die Stellungen der Dip-Schalter zurück
unsigned char dip_pin(unsigned char dip_schalter);	gibt die Stellungen des *dip_schalter* zurück
void version(char *versionstext);	gibt die aktuelle Version zurück

Tabelle 4.2: AKSEN-Funktionen Teil2(Version 0.956)

Funktion	Kurzbeschreibung
Zeitsteuerung	
void sleep(unsigned long wartezeit);	warte *wartezeit* in ms
unsigned long akt_time(void);	akt.Systemzeit in ms seit Systemstart
void clear_time(void);	setzt Zeitzähler auf 0
Infrarot	
#define mod_ir_an()	schaltet IP-Port ein
#define mod_ir_aus()	schaltet IP-Port aus
#define setze_ir_senden(wert)	setzt halbe Periodendauer auf *wert* in ms
#define mod_ir0_takt(takt)	setzt halbe Periodendauer
#define mod_ir1_takt(takt)	setzt halbe Periodendauer
#define mod_ir2_takt(takt)	setzt halbe Periodendauer
#define mod_ir3_takt(takt)	setzt halbe Periodendauer
#define mod_ir0_status()	Wert für Stabilität des IR-Signals
#define mod_ir1_status()	Wert für Stabilität des IR-Signals
#define mod_ir2_status()	Wert für Stabilität des IR-Signals
#define mod_ir3_status()	Wert für Stabilität des IR-Signals
#define mod_ir0_maxfehler(fehler)	setzt *fehler* für maximale Zeitdauer
#define mod_ir1_maxfehler(fehler)	setzt *fehler* für maximale Zeitdauer
#define mod_ir2_maxfehler(fehler)	setzt *fehler* für maximale Zeitdauer
#define mod_ir3_maxfehler(fehler)	setzt *fehler* für maximale Zeitdauer

Tabelle 4.3: AKSEN-Funktionen Teil3(Version 0.956)

4.2 Erstes Programm

Es wird beschrieben, welche Bestandteile zum Programm gehören. Das Programm kann grundsätzlich in zwei Abschnitte unterteilt werden. Das ist der erste Teil mit den Präprozessoranweisungen. Hier wird zum Beispiel mit der Anweisung „#include“ der Bibliotheksquellcode aus der dahinter aufgeführten Datei in das Programm eingefügt. Im Programmbeispiel Abbildung 4.1 werden in den Zeilen 1 und 2 die Standardbibliotheken und in Zeile 3 die Bibliothek mit den Funktionen aus den Tabellen 4.1, 4.2 und 4.3 eingefügt.
Im zweiten Teil des Programms stehen die Funktionen. Mit dem Schlüsselwort „AksenMain“ wird die Hauptfunktion, d.h. der Startpunkt, an dem die Programmausführung beginnen soll, beschrieben. Im Programmbeispiel Abbildung 4.1 geschieht das in der Zeile 4. Die Zeile 6 stellt die Funktion „printf“ zur Ausgabe einer Zeichenfolge dar und „while(1)“ in der Zeile 6 ist eine Endlosschleife.

```
1  #include <stdio.h>
2  #include <regc515c.h>
3  #include <stub.h>
4  void AksenMain(void)
5  {
6      lcd puts("Hallo Welt");
7      while(1);
8  }
```

Abbildung 4.1: Programmbeispiel

4.3 Weltmodell-Architektur

Bei der Weltmodell-Architektur handelt es sich um das traditionelle Modell in der Roboterprogrammierung. Dieses Modell schließt alle geometrischen Details, die Position und die Ausrichtung aller Objekte in die Umwelt des Roboters ein. [JF96]
Die Bewertung aller eingehenden Sensorsignale und der entsprechenden Reaktionen erfolgt durch ein Programm. Wegen der ständigen Bewertung und Anpassung widersprüchlicher Sensordaten (Sensordatenfusion) kommt es zu einem hohen Rechenaufwand. Die dadurch fehlende Echtzeitverarbeitung führt zu Schwächen beim Manövrieren in natürlicher oder wechselnder Umgebung. Es kommt zu Problemen, wenn die Zeit zwischen der Aufnahme der Sensordaten und der resultierenden Aktion zunimmt.
Die Weltmodellierung zeichnet sich durch ihre universellen Fähigkeiten aus. Konzepte aus der KI-Forschung werden zur Realisierung herangezogen. Die Programmierung der Künstlichen Intelligenz beschränkt sich wegen des hohen Aufwandes meist auf größere Systeme. [AA99]

In der folgenden Abbildung ist zu sehen, wie nach dem Paradigma der Weltmodellierung und der Aktionsplanung ein Roboterprogramm in eine geordnete Folge von Funktionseinheiten zerlegt wird. Die zu einem Zeitpunkt aufgenommenen Sensordaten werden zunächst gesammelt. Für ein stimmiges Weltmodell werden unwichtige und widersprüchliche Daten angepasst. Das entstandene Weltmodell führt, je nach Programm, zur Planung von Aktionen. Als Ergebnis werden entsprechende Befehle an die Motoren gesendet. [JF96]

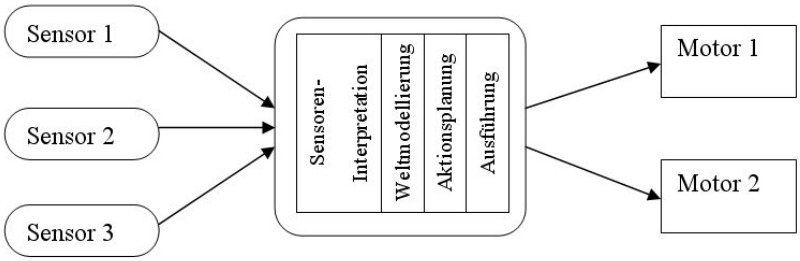

Abbildung 4.2: Weltmodell-Architektur

4.4 Subsumtionsmethode

Eine Alternative zur Weltmodell-Architektur ist die von Rodney Brooks und der Arbeitsgruppe für Mobile Roboter am MIT Artificial Intelligence Laboratory entwickelte Subsumtionsmethode. Anstatt alle Sensordaten auf einmal zu beurteilen, werden die einzelnen Sensorinformationen mit einem dazugehörigen Verhaltensmuster verknüpft. Diese Verhaltensmodule entsprechen parallel laufenden Prozessen in einer hierachischen Ordnung (siehe Abbildung 4.3).
Widersprüchliche Sensorinformationen führen zu widersprüchlichem Verhalten, das durch logische Verzweigungen gelöst wird. So unterbricht ein hohes, priorisiertes Verhalten das niedriger Priorisierte.
Für einfache, mobile Roboter ist das Modell gut geeignet. Eine steigende Komplexität führt zu einer großen Verzweigungsstruktur und damit auch zu höherer Rechenleistung, was den Einsatz wieder einschränken kann.

Weiterhin soll beschrieben werden, wie die Subsumtion zu programmieren ist (bezogen auf Abbildung 4.3). Dazu noch einmal der Hinweis, dass es sich bei den Verhaltensmodulen um Prozesse handelt. So ist es erforderlich, zum Programmstart alle Prozesse anzustoßen.

```
1 void main(){
2    process_start(motor_command,50);
3    process_start(verhalten3,50);
4    process_start(verhalten2,50);
5    process_start(verhalten1,50);
6    process_start(arbitrate,50);
7
8    process_kill(process_get_pid());
9 }
```

Die Funktionen gehören zur AKSEN-Bibliothek und können in den Tabellen 4.1, 4.2 und 4.3 nachgeschlagen werden. Prozesse müssen entweder in einer Endlosschleife laufen oder beendet werden. Da der Main-Prozess keine weiteren Aufgaben zu erledigen hat, wird er in der Zeile 8 beendet.
Der Prozess *motor_command* in Zeile 2 ist dazu da, die globale Variable *command* zu überwachen und die Motoren, je nach Inhalt der Variablen, anzusteuern.

```
1 int command;
2
3 void motor_command(void){
4    while(1){
5            switch(command){
6                   case Anweisung1:    motor_pwm(0,10);
7                                       motor_pwm(1,10);
8                                       break;
9                   case Anweisung2:    motor_pwm(0,10);
10                                      motor_pwm(1,0);
11                                      break;
12                  case Anweisung3:    motor_pwm(0,0);
13                                      motor_pwm(1,0);
14                                      break;
15              }
16     }
17 }
```

Der Prozess *arbitrate* in Zeile 6 der Main-Funktion dient, wie aus dem Namen hervorgeht, zum Schlichten. Das bedeutet, Meldungen der anderen Prozesse zu empfangen und je nach Priorität einer konkreten Anweisung zuzuordnen.

```
1 void arbitrate(void){
2     while(1){
3         if(verhalten1_flag == 1)command = command_Anweisung1;
4         if(verhalten2_flag == 1)command = command_Anweisung2;
5         if(verhalten3_flag == 1)command = command_Anweisung3;
6      }
7 }
```

Die Verhaltensstruktur ist mit einer Folge von If-Anweisungen dargestellt. Je tiefer sich die Anweisung befindet, desto höher ist die Priorität. Die globale Variable *command* wird entsprechend überschrieben.

```
1 int verhalten1_flag;
2 int command_Anweisung1;
3
4 void verhalten1(){
5     while(1){
6        if(sensor1 == Bedingung){
7                                verhalten1_flag = 1;
8                                command_Anweisung1 = Anweisung1;
9          }
10        else verhalten1_flag = 0;
11     }
12 }
```

Wie alle Prozesse muss auch *verhalten1* in einer Endlosschleife laufen. Je nach Bedingung soll das Verhalten aktiv werden und die Anweisung in der Variablen *command* zur Ausführung gelangen. Mit der Variablen *verhalten1_flag* wird das Verhalten mit 1 = aktiv und 0 = inaktiv gesetzt.
Die anderen Verhalten haben den gleichen Aufbau mit der gleichen Funktionalität.

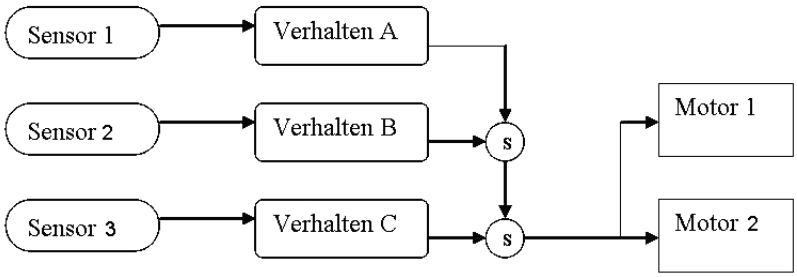

Abbildung 4.3: Subsumtions-Architektur

4.5 Prozesse und Multitasking

Bei einem Prozess, oder auch Task genannt, handelt es sich um ein in sich abgeschlossenes Programm. Werden den Prozessen in einer bestimmten Reihenfolge kurze Prozessorzeiten zugeteilt, so dass mehrere Prozesse quasi gleichzeitig ablaufen, spricht man von Multitasking. Die Prozesse laufen scheinbar parallel ab.

4.6 Spezieller Roboterbefehlssatz

Mit den vorhandenen AKSEN-Funktionen und den neuen Funktionen sollen dem AMS eine Menge an Skills (Fähigkeiten) bereitgestellt werden, d.h. konkrete Funktionen zur Programmierung eines Roboters auf einer höheren Ebene.
Diese Skills werden in drei Gruppen eingeteilt. Das sind die Bewegungsbefehle, Konfigurationsbefehle und Verhaltensbefehle.

4.6.1 Bewegungsbefehle

Bei den Bewegungsbefehlen handelt es sich um Funktionen, die abgearbeitet werden und zur aufrufenden Stelle zurückkehren (d.h. sie blockieren).

move

Fahre den Roboter einen angegebenen Weg (mm)! Ein positiver Parameter bedeutet vorwärts und ein negativen rückwärts.

Syntax: *void move(int mm);*

turn

Drehe den Roboter in einem angegebenen Winkel (deg)! Ein positiver Parameter bedeutet vorwärts und ein negativen rückwärts.

Syntax:*void turn(int deg);*

halt

Halte den Roboter an, d.h. alle Antriebsmotoren auf Null!

Syntax: *void halt(void);*

4.6.2 Konfigurationsbefehle

Mit den Konfigurationsbefehlen ist es möglich, den AKSEN-Roboters an die herrschenden Bedingungen anzupassen.

Befehl	Kurzbeschreibung	Default
void init_arbs(void);	initialisiert den Roboter Befehlssatz	
unsigned int get_raddurchmesser();	gibt den akt. Randdurchmesser in mm zurück	42
void set_raddurchmesser(unsigned int durch);	konfiguriert den Durchmesser	
unsigned char get_seganz();	gibt die akt.Segmentanzahl zurück	12
void set_seganz(unsigned int anz);	konfiguriert die Segmentanzahl	
unsigned int get_radabstand();	gibt den akt.Radabstand zurück	110
void set_radabstand(unsigned int abst);	konfiguriert den Radabstand	
unsigned char get_speed();	gibt die akt.Geschwindigkeit zurück	10
void set_speed(unsigned char sp);	konfiguriert die Geschwindigkeit 0-10	
unsigned char get_turnspeed();	gibt die akt.Drehgeschwindigkeit zurück	7
void set_turnspeed(unsigned char tusp);	konfiguriert die Drehgeschwindigkeit 0-10	
void set_encoder_links (unsigned char el);	konfiguriert den linken Encoder	E-Port0
void set_encoder_rechts (unsigned char er);	konfiguriert den rechten Encoder	E-Port1
void set_motor_links (unsigned char ml);	konfiguriert den linken Motor	M-Port0
void set_motor_rechts (unsigned char mr);	konfiguriert den rechten Motor	M-Port1
unsigned char get_motor_links ();	gibt den Port des linken Motor zurück	0
unsigned char get_motor_rechts ();	gibt den Port des rechten Motor zurück	1
unsigned char get_bumper1();	gibt den Port des ersten Bumper zurück	0
void set_bumper1(unsigned int bum1);	konfiguriert den Port des ersten Bumper	
unsigned char get_bumperN();	gibt den Anzahl der Bumper zurück	6
void set_bumperN(unsigned int bumN);	konfiguriert die Anzahl der Bumper	
unsigned char get_wand1();	gibt den Port des ersten Wand-IR-Sensors zurück	5
void set_wand1(unsigned int wan1);	konfiguriert den Port des ersten Wand-IR-Sensors	
unsigned char get_wandN();	gibt die Anzahl der Wand-IR-Sensoren zurück	5
void set_wandN(unsigned int wanN);	konfiguriert die Anzahl der Wand-IR-Sensoren	
unsigned char get_line1();	gibt den Port des ersten Linien-IR-Sensors zurück	5
void set_line1 (unsigned int lin1);	konfiguriert den Port des ersten Wand-IR-Sensoren	
unsigned char get_lineN();	gibt die Anzahl der Linien-IR-Sensors zurück	3
void set_lineN(unsigned int linN);	konfiguriert die Anzahl der Linien-IR-Sensors	

Tabelle 4.4: Konfigurationsbefehle

4.6.3 Verhaltensbefehle

Mit Verhaltensbefehlen wird das Kombinieren verschiedener Funktionen zu einem konkreten Verhalten(behavior) bezeichnet. Ein Verhaltensbefehl startet einen nebenläufigen Prozess (d.h. er ist nicht blockierend). Ein separater Prozess überwacht alle Verhaltensprozesse und leitet nach der Subsumtionsmethode die Verhalten zur Ausführung.

Befehl	Kurzbeschreibung	Quelle
void start_follow_line(void);	starte Verhalten zur Linienverfolgung	follow_line.c
void stop_follow_line(void);	beende Verhalten zur Linienverfolgung	follow_line.c
void start_follow_light(void);	starte Verhalten zur Lichtverfolgung	follow_light.c
void stop_follow_light(void);	beende Verhalten zur Lichtverfolgung	follow_light.c
void start_region(unsigned int breite, unsigned int xmax, unsigned int ymax);	starte Verhalten zum Abfahren einer Fläche	region.c
void stop_region(void);	beende Verhalten zum Abfahren einer Fläche	region.c
void start_bumper(void);	starte Verhalten zum Reagieren auf Bumper	bumper.c
void stop_bumper(void);	beende Verhalten zum Reagieren auf Bumper	bumper.c
void start_avoidcollision(void);	starte Verhalten zur Vermeidung von Kollisionen	avoid_collision.c
void stop_avoidcollision(void);	beende Verhalten zur Vermeidung von Kollisionen	avoid_collision.c
void start_go(void);	starte Verhalten zum Anschalten der Motoren	go.c
void stop_go(void);	beende Verhalten zum Anschalten der Motoren	go.c

Tabelle 4.5: Verhaltensbefehle

4.7 Regelkreise

In einem AMS ist es notwedig, das gewünschte Verhalten der Aktoren zu überwachen. D.h., über Sensoren wird gemessen, wie weit der Ist- vom Soll-Wert entfernt ist und entsprechend reagiert. Das wiederkehrende, zyklische Messen, Vergleichen und Stellen wird als Regelkreis bezeichnet. Zu den Bestandteilen des Regelkreises gehören Regler, Stelleinrichtung, Regelstrecke und Messeinrichtung. [Sc86]
Regler können nach dem Wertebereich ihres Ausgangssignals klassifiziert werden. Nimmt das Ausgangssignal nur bestimmte (abzählbar viele) Werte an, so spricht man von einem **unstetigen Regler**. Typische Anwendung ist die Zweipunktregelung („Fenster auf - Fenster zu“), bei der die Stellgröße den Wert 0% oder 100% annehmen kann.
Kann das Ausgangssignal einen beliebigen Wert zwischen 0% und 100% annehmen, handelt es sich um einen **stetigen Regler**.
Stetige Regler wiederum können nach ihrem dynamischen Verhalten wie folgt unterteilt werden :

- proportional wirkende Regler (P-Regler)
- integrierend wirkende Regler (I-Regler)
- differenzierend wirkende Regler (D-Regler)

Häufig sind die Eigenschalften miteinander verknüpft. [AB87]

4.7.1 P-Regler

Der P-Regler wird dort eingesetzt, wo keine hohen Anforderungen an die Regelgenauigkeit gestellt werden. Der Zusammenhang zwischen der Regelabweichung e und der Stellgröße y wird in der klassischen P-Regler-Gleichung beschrieben. (KR -Proportionalitätsfaktor, w-Sollwert)

$$e = w - x \tag{4.1}$$

$$y(t) = K_R \cdot e(t) \tag{4.2}$$

4.7.2 PI-Regler

Dem P-Regler wird ein intergral wirkender Teil hinzugefügt, um ein Schwingen des Reglers zu verhindern. Die nachfolgend dargestellte Reglergleichung stellt die beiden Anteile dar (KI -Integralfaktor).

$$y(t) = K_R \cdot e(t) + K_I \cdot \int_0^t e(\tau)d\tau \tag{4.3}$$

4.7.3 PID-Regler

Ein PID-Regler setzt sich aus einem P-Anteil, einem I-Anteil und einem D-Anteil zusammen. P steht für den proportional wirkenden, I für den integral wirkenden und D für differential wirkenden Teil der Gleichung. Der D-Anteil sorgt für ein schnelles Reagieren auf Änderungen der Regelabweichung. Die entsprechende Reglergleichungen ist im Folgendem aufgeführt (KD -Differentialfaktor).

$$y(t) = K_R \cdot e(t) + K_I \cdot \int_0^t e(\tau)d\tau + K_D \cdot \frac{de(t)}{d\tau} \tag{4.4}$$

Wie ein PID-Regler in C programmiert werden kann, ist als nächstes dargestellt. [Br03]

```
1 int y_old=0, e_old=0, e_old2=0;
2
3 int regler(int e,float kr,float ki,float kd){
4
5      int y;
6
7      y=((y_old+kr*(e-e_old)+ki*(e-e_old)/2+kd*(e-2*e_old+e_old2)));
8
9      y_old=y;
10     e_old2=e_old;
11     e_old=e;
12
13     return y;
14 }
```

4.8 Positionierung und Orientierung

In diesem Kapitel werden verschiedene Möglichkeiten betrachtet, wie sich ein AMS positionieren und orientieren kann. Das heißt, immer zu wissen, an welchem Punkt im Raum sich das System befindet und, je nach Aufgabenstellung, darauf zu reagieren.

4.8.1 Encoder

Encoder sind Sensoren, die sich mit dem Rad oder Motor auf einer Welle befinden. Sie haben die Aufgabe, die Drehbewegung für eine weitere Verarbeitung zu ermitteln. Dies kann auf unterschiedliche Art und Weise erfolgen. In der weiteren Arbeit soll der optische Rad-Encoder genauer beschrieben werden.
Es gibt unterschiedliche Arten optischer Rad-Encoder. Das sind absolute und inkrementelle Encoder. Je nach Ausrichtung der Welle wird die Position in Form eines Codes oder als Folge von Impulsen verarbeitet. Bei dem **absoluten Encoder** wird die genaue Stellung der Achse durch eine Positionscodierung bestimmt. Bei dem **inkrementellen Encoder** wird im Gegensatz dazu die Position durch das Zählen der Impulse bestimmt. [JF96]

4.8.2 Kompass

Mit einem elektronischen Kompass kann sich das AMS anhand eines absoluten Koordinatensystems orientieren. Im Outdoor-Bereich funktioniert der Betrieb sehr zuverlässing. Dagegen im Indoor-Bereich führen Kabeln, Baustahl und das magnetischen Feld vom Roboter selber zu Störungen. Ist ein Fehler bis zu ± 45 Grad tolerierbar, sind Kompasse auch im Indoor-Bereich einsetzbar. [JF96]

4.8.3 GPS

Das GPS (Global Positioning System) wurde von den USA für militärische Zwecke, wie Zielführung von Raketen, entwickelt. Daraus entstanden später auch Systeme für den zivilen Gebrauch. Moderne Navigationssysteme in Kraftfahrzeugen, Schiffen und mobilen Robotern arbeiten mit GPS. [Da03]
Im Aufbau befindet sich das Europäische Global Navigations-Satelliten-System GALILEO unter ziviler Leitung.

5 Tutorial Teil 1: Bauanleitung eines AMS

In diesem Kapitel soll dem Roboterentwickler ein Hilfsmittel bereit gestellt werden, mit dem er übliche wiederkehrende Probleme lösen kann. Teil 1 gibt Hilfestellung bei der Hardware-Entwicklung und Teil 2 bei der Software-Entwicklung.

Bekommt der Roboter einen Namen, erhöht dies die Motivation. Das hier im Weiteren beschriebene AMS soll den Namen „Eighteen“ erhalten. „Eight“ steht für die acht Ecken der äußeren Form und „Teen“ als Bezeichnung für Teenager mit der Bedeutung nicht mehr ganz Kind aber auch noch nicht Erwachsen. Das ist eine Bezeichnung für ein entwickeltes System, welches jedoch noch nicht ganz ausgereift ist .

5.1 Konzeption

Ziel der Entwicklung eines autonomen, mobilen Roboters ist es, ein System zu konstruieren, das einfach, modular, robust, langlebig und billig ist, und trotzdem eine vielseitige Funktionalität aufweist. Die Einfachheit hilft schon in der Planungsphase, sich auf das Notwendigste, ohne großen „Schnickschnack“, zu beschränken. In der Entwicklungsphase ist es von Vorteil, das System in einzelne funktionsbezogene Module zu zerlegen. So können diese Module unabhängig voneinader entwickelt und später auch gewartet werden. In der Modulbauweise lässt sich der Roboter auch leichter verändern oder erweitern. Bei jedem Roboter-Konstrukteur sollte die Robustheit seines Roboters an erster Stelle stehen. Robost in der Funktionalität und Haltbarkeit müssen einfach vorausgesetzt werden.

5.1.1 Form

Problem: Gibt es bestimmte Roboterformen, die Vorteile gegenüber anderen Formen bieten?

Diskussion: Wie im Kapitel 3.2 beschrieben ist, führt eine zylindrische Form zu einer höheren Bewegungsfreiheit. Nun sind zylindrische Formen nicht immer leicht nachzubauen. Kompromisse, zum Beispiel mit einer Achteckform, sind dennoch tragbar. Wie so eine Form mit LEGO-Bausteinen ausehen kann, sieht man in der folgenden Abbildung.

Vorschlag: Annäherung an eine zylindrische Form - z.B. ein Zylinder mit einer achteckigen Grundfläche.

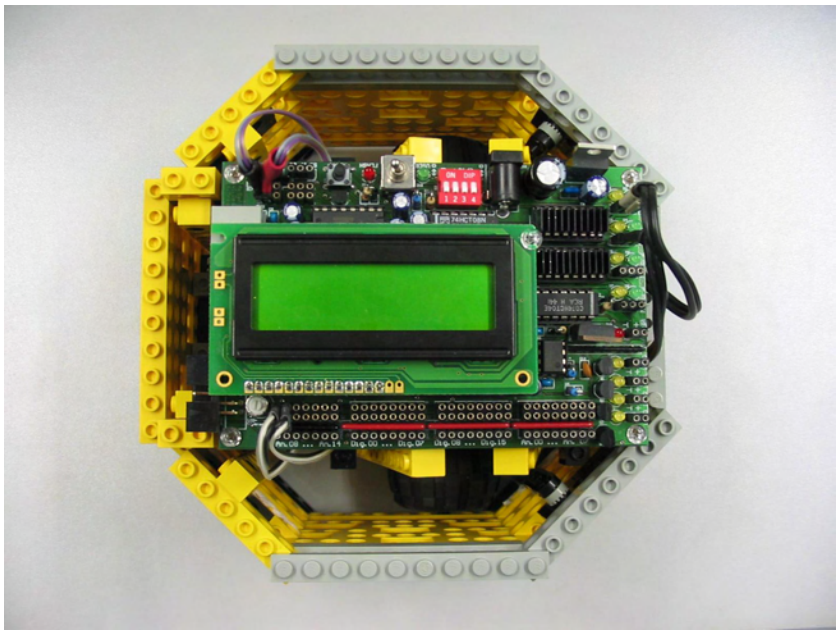

Abbildung 5.1: Draufsicht „Eighteen“

5.1.2 Stabilität

Problem: Wie erreicht man eine optimale Stabilität?

Diskussion: Für eine ausreichende Stabilität muss jeder einwirkenden Kraft eine entsprechende Kraft entgegen wirken, d.h. ein Körper behält seine Beständigkeit. Dementsprechend sind die Wahl des Materials und der Kostruktionsart von ausschlaggebender Bedeutung. Für das Material aus Lego-Bausteinen ist eine Größe vorgegeben. Also sind die Form und der Zusammenbau der Teile eine Maß für die Stabilität.

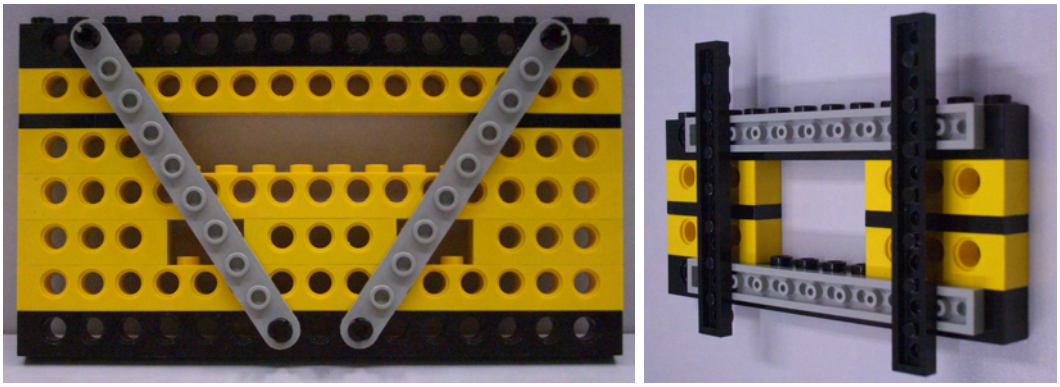

Abbildung 5.2: Vertikale Verbindungen

Vorschlag: Die horizontale Stabilität ist durch die Form mit den Noppen gegeben. Die vertikale Stabilität wird durch Konstruktionen wie in Abbildung 6.1 erreicht. Bei „Eighteen“ wird das Gehäuse in ähnlicher Konstruktion zusammengehalten, wie es in der Abbildung 5.3 zu sehen ist.
Weiterhin ist für eine ausreichende Stabilität die Verbindung der beiden Module Antriebsblock und Gehäuse nötig. An dem Antriebsblock sind vorn zwei und hinten ein Aufhängungspunkt angebracht. Damit wird das Antriebsmodul fest mit dem Gehäuse verbunden. Zu sehen sind die drei vom Antriebsblock vorstehenden Punkte in der Abbildung A.7 . Außerdem sind die Antriebswellen an den äußeren Punkten im Gehäuse stabilisiert.

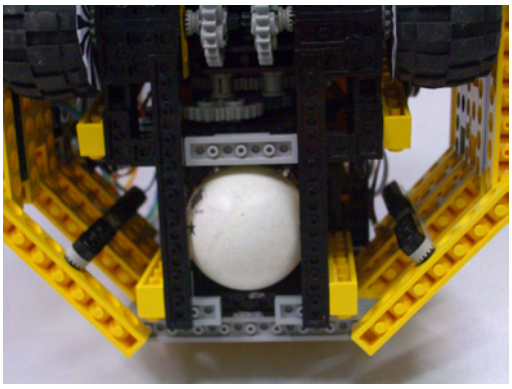

Abbildung 5.3: Verstrebungen am Gehäuse von „Eighteen“

5.1.3 Schwerpunkt

Problem: Wie erreicht man eine günstige Gewichtsverteilung?

Diskussion: Generell muss der Schwerpunkt eines Fahrzeuges so tief wie möglich liegen und alle Räder müssen gleichmäßig belastet sein. So erreicht man eine günstige Spurstabilität.

Der Schwerpunkt sollte gleichfalls auf einer Linie zwischen der Antriebswelle und dem Stützrad bzw. Ball liegen (siehe Abbildung). In Richtung der Position a) neigt der Roboter beim Abbremsen zum Kippen nach vorn. In der Position a) kippt er schon im Ruhezustand. Das Gleiche trifft auch für die Position c) zu. In Richtung Position c) kann es zum seitlichen Kippen kommen. Ideal ist eine mittlere Position wie zum Beispiel in der Position b).

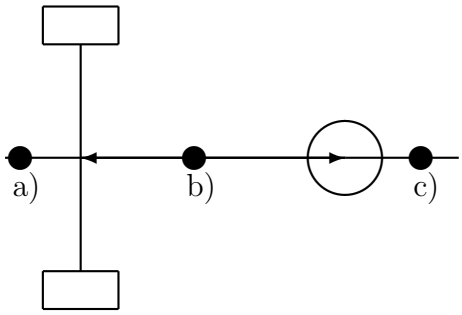

Abbildung 5.4: Position des Schwerpunktes

Vorschlag: Bei dreirädrigen Fahrzeugen sollte der Schwerpunkt auf der Achse zwischen der Antriebswelle und dem Stützrad bzw. Ball liegen.

5.2 Stromversorgung

Problem: Welche Stromversorgung bietet sich für ein AMS an?

Diskussion: Die Stromversorgung eines automobilen Roboters erfolgt üblicherweise mit Akkus. Dieses Thema ist aber sehr umfangreich und müßte deshalb gesondert behandelt werden.

Vorschlag: Für den Einsatz in mobilen Systemen bieten sich die Nickel-Metall-Hybrid-Akkus (NiMH) an. Sie haben eine bis zu 100% höhere Kapazität als NiCd, sie haben kei-

nen Memory-Effekt und sie haben eine geringere Umweltbelastung als die NiCd-Akkus. Außerdem ist die Lebensdauer, aufgrund des weniger häufigen Ladebedarfes, länger.

5.3 mechanischer Aufbau

5.3.1 Motorwahl

Problem: Was für ein Motor wird benötigt?

Diskussion: Im Allgemeinen werden Elektromotoren in Robotern eingesetzt. Für eine genaue Bewegung auf kurzen Strecken kommen die Schritt- bzw. Servomotoren zum Einsatz. Beim Antrieb sind, im Gegensatz dazu, leistungsstärkere Motoren notwendig.

Abbildung 5.5: Lego-Motor

Vorschlag: Für den Einsatz bei Verwendung von AKSEN-Board und Lego-Bausteinen bieten sich auch die Lego-Motoren an. Sie können fest und einfach in das System integriert werden und bieten auch noch alle notwendigen Leistungsmerkmale. Hat man diese Möglichkeit nicht, können auch einfache E-Motoren, wie in „Eighteen“, genutzt werden. Diese müssen dann stabil zwischen den Lego-Bausteinen eingesetzt werden.

5.3.2 Getriebewahl

Problem: Wann wird ein Getriebe notwendig und wie wird es konstruiert?

Diskussion: Ein Getriebe wird dann notwendig, wenn die Motorbewegung nicht direkt an die Räder weiter gegeben werden kann. Das ist der Fall bei hoher Drehzahl und niedrigem Drehmoment eines Elektromotors. LEGO bietet eine einfache Möglichkeit, Getriebe selbst nach seinen Bedürfnissen zu kreieren. Dazu hier ein paar Grundlagen [Wü04]:

- 1 LU(Lego Unit) = 8 mm horizontaler Noppenabstand
- 1,2 LU = vertikalen Noppenabstand bzw. Lochabstand
- 0,4 LU = flache Bausteine vertikal
- 8er Zahnradradius = 0,5 LU
- 24er Zahnradradius = 1,5 LU
- 40er Zahnradradius = 2,5 LU

D.h. der Achsabstand ist die Summe der Radien ineinander greifender Zahnräder. So sind horizontal alle Kombinationen ohne weiteres möglich. Bei vertikalen Kombinationen wird der Abstand entsprechend der LU mit den flachen Bausteinen angepasst. Zum Beispiel bei einem 8er und 24er Zahnrad mit 2 LU Achsabstand sind zu dem 1,2 LU vertikalen Rastermaß noch zwei flache Bausteine mit je 0,4 LU hinzuzufügen. Bei der vertikalen und horizontalen Verschiebung erfolgt die Berechnung über den Satz des Pythagoras. In der folgenden Abbildung sieht die Berechnung so aus:

0,5 + 2,5 Radius = 3,0 LU Hypotenuse

1 LU horizontal versetzt

vertikale Abstand = $\sqrt{3,0^2 - 1,0^2} = 2,8$

2,8 entspricht zwei großen Steinen mit je 1,2 LU und einem flachen Stein mit 0,4 LU.

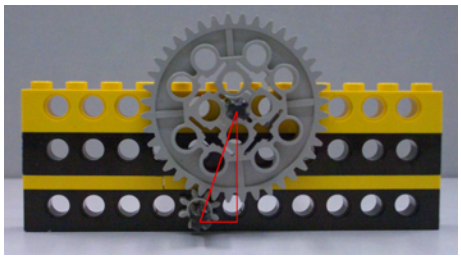

Abbildung 5.6: Beispiel Zahnradanordnung

Nach den Grundlagen eines Lego-Getriebes nun zu dem eigentlichen Getriebe: Um ein Getriebe konstruieren zu können, sind folgende Daten notwendig:

- Motordaten wie Drehmoment, Leistung oder Umdrehungszahl
- gewünschte Geschwindigkeit des mobilen Roboters
- Durchmesser der Antriebsräder

Aus den Daten wird das Übersetzungsverhältnis durch folgende Formeln ermittelt:

$$n_{Motor} = i_{ges} \cdot n_{Antriebsrad} \tag{5.1}$$

$$n_{Antriebsrad} = \frac{V}{\Pi \cdot d_{Antriebsrad}} \tag{5.2}$$

$$i_{ges} = n_{Motor} \cdot \frac{\Pi \cdot d_{Antriebsrad}}{V} \tag{5.3}$$

Vorschlag: Mit einem Motor, dessen Welle sich 26000 mal pro Minute dreht und einem Antriebsrad von 42 mm Durchmesser ergibt sich ein ungefähres Übersetzungsverhältnis von 1 zu 120 für eine Geschwindigkeit von $0{,}5\frac{m}{s}$. Mit der Zahnradanordnung 8 zu 40 und 1 zu 24 (Schneckenrad) wird nach der Formel 2.1 das Gesamtübersetzungsverhältnis 1 zu 120 erreicht. In der Abbildung 5.7 ist das Getriebe von „Eighteen“ mit noch einer zusätzlichen Übersetzung von 1 zu 1 aus Konstruktionsgründen zu sehen.

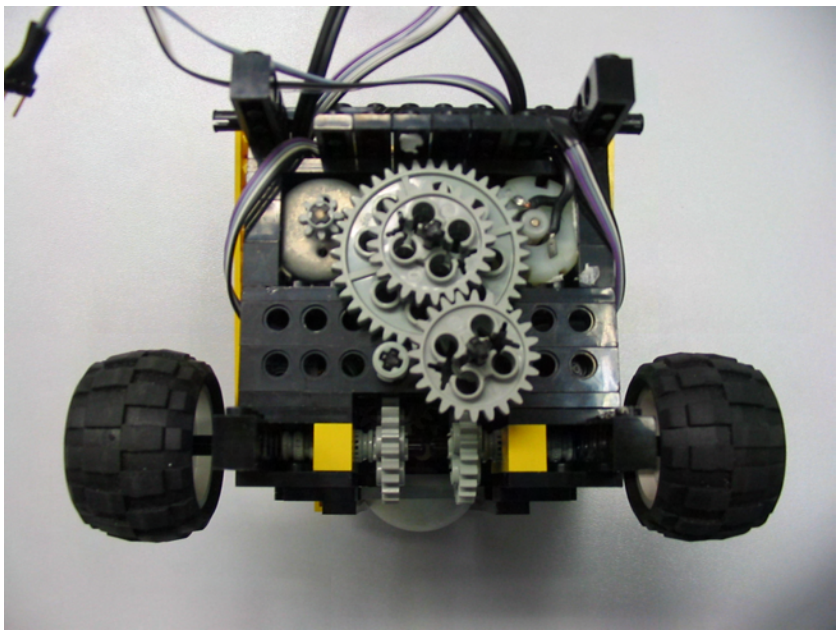

Abbildung 5.7: Getriebe „Eighteen“

5.3.3 Radwahl

Problem: Welche Räder sollen für ein AMS genutzt werden?

Diskussion: Gerade für den Indoor-Bereich sollten gummibereifte Räden bevorzugt werden. Sie erreichen den notwendigen Kontakt zum Boden ohne größeren Schlupf. Hier kann der Bodenkontakt noch durch eine höhere Reifenbreite verstärkt werden. Mit den Reifenbreiten und der damit höheren Reibung steigt auch der Energieverbrauch. So ist je nach Anforderung ein günstiger Kompromiss zu suchen.
Mit der Vergrößerung des Raddurchmessers kann ein mobiler Roboter auch kleinere Hindernisse (z.B. Kabel, kleine Kanten) überfahren. Größere Räder sind, wegen ihrer Ausmaße nicht so leicht in einen Roboter zu integrieren. Außerdem erhöht sich bei gleicher Kraft mit steigendem Raddurchmesser das Drehmoment. Mit gleichbleibender Motorleistung wird das höhere Drehmoment mit einem größeren Übersetzungsverhältnis ausgeglichen. Das bedeutet, das größere Rad deht sich langsamer.

Vorschlag: Je nach Aufgabe bietet LEGO eine Auswahl an gummibereiften Rädern, die leicht einzusetzen sind. Für kleinere AMS bieten sich, im Verhältnis zur Gesammtgröße, Räder mit gößerem Raddurchmesser an. Sie bleiben nicht so leicht hängen. Bei größeren Systen lassen sich große Räder nur schwer händeln, deshalb bieten sich kleinere eher an.

5.4 Wahl der Antriebsart

Problem: Mit welcher Antriebsart läßt sich ein AMS gut monövrieren?

Diskussion: Ein zentraler Punkt bei der Diskussion um die Monövrierfähigkeit ist der Wendekreis. Mit kleinem Wendekreis gelingt es am einfachsten sich an jede Position zu bewegen. Die Anordnung der Räder eines Dreiradantriebes bietet eine günstige Konstellation für einen kleinen Wendekreis. Sind die zwei Antriebsräder unabhängig voneinander in entgegengesetzter Richtung zu betreiben, liegt der Drehpunkt zwischen beiden Rädern. Der Roboter ist in der Lage, sich auf der Stelle zu drehen. Die folgende Abbildung soll dies verdeutlichen.

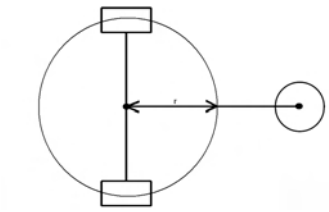

Abbildung 5.8: Wendekreis eines Dreiradantriebes

Vorschlag: Wenn es möglich ist den Schwerpunkt nach Abschnitt 5.4 so auszubalacieren, ist ein Dreirandantrieb immer eine günstige Wahl.

5.4.1 Nachlauf-/Stützrad

Problem: Wie wird das Umkippen bei einem AMS verhindert?

Diskussion: Mindestens drei Räder sind notwendig, um ein AMS ohne zusätzliche Stabilisierungseinrichtungen am Umkippen zu hindern.
Zwei Räder sind schon für den Vortrieb vorhanden, so fehlt nur noch eins als Stütze. Zwei Möglichkeiten werden hier vorgeschlagen:

- Stützrad (Abbildung: 5.9)
- Stützkugel (Abbildung: 5.10)

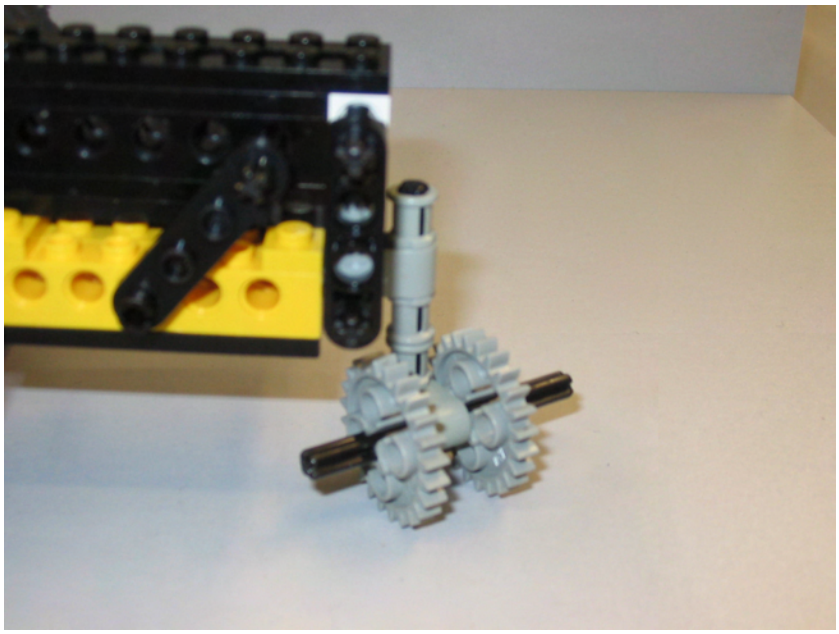

Abbildung 5.9: Nachlaufrad

Das Nachlaufrad hat den Nachteil der verzögerten Ausrichtung, d.h. bei einer größeren Richtungsänderung braucht das Rad seine Zeit, sich in die geeignete Position zu stellen. Ein weiterer Nachteil ist die mangelnde Stabilität. Durch das Verlagern der Drehachse ist es nicht so einfach, größere Kräfte zu bewältigen.

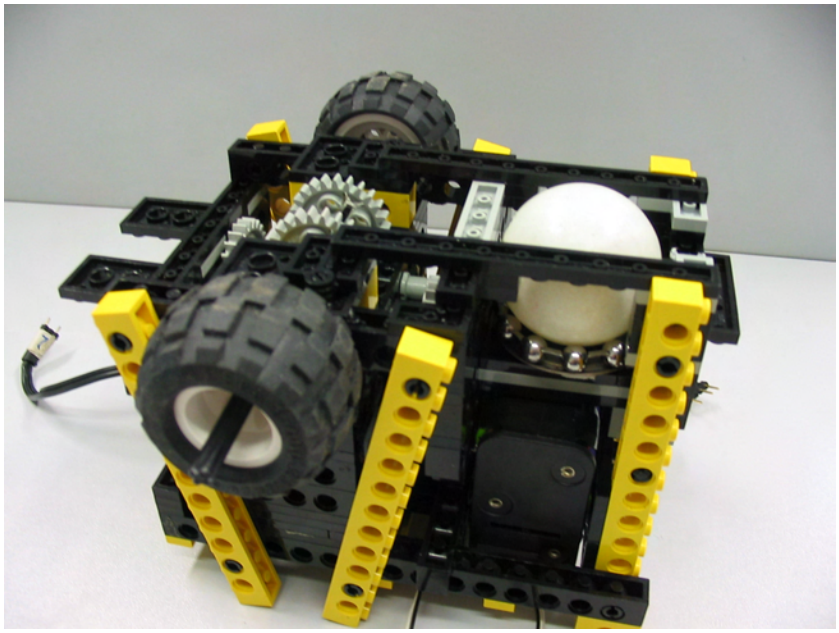

Abbildung 5.10: Stützkugel

Vorschlag: Mit der Kugel sind alle Nachteile eines Stützrades aufgehoben. Zu berücksichtigen ist hier, dass die Reibungskräfte verringert werden. Das geschieht zum Beispiel durch ein Kugellager (siehe Bild).

5.5 Sensorik und Informationsverarbeitung

5.5.1 AKSEN-Board

Problem: Wie können die Anschlüsse am AKSEN-Board belegt werden?

Diskussion: Anschlussbeschreibung (Parameter)

Motor-Port	:	1000mA pro Kanal
LED	:	Ptot=1W bei 25 °C (Ptot =VCC2*I)
IR	:	Ptot =65W bei 25 °C (Ptot =VCC2*I)
I/O-Pin	:	3mA(L), -10 bis -80μA(H)
VCC	:	5V-Zweig auf 1A begrenzt
VCC2	:	abhängig Stromversorgung 5 bis 8V (Servo,Motor,LED)

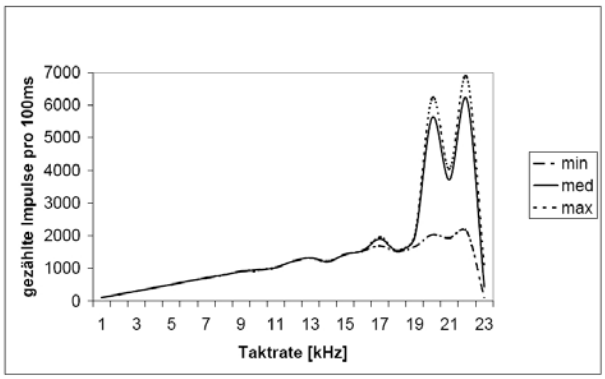

Abbildung 5.11: Diagramm Encoder-Eingang

In der Abbildung 5.11 ist zu sehen, dass über den Encoder-Port Taktraten bis zu 16 KHz verarbeitet werden können.

Aus technischen Gründen der Signalverarbeitung im AKSEN-Board werden nur die Flanken von high nach low gezählt. Bedingt wird dies durch die Interruptroutinen.
Das Diagramm zeigt den Verlauf einer Messreihe von 8 Werten pro Frequenz, das Minimum, den Durchschnitt und das Maximum (komplette Tabelle im Anhang B.4).
Gemessene Werte wurden über die CAN-Schnittstelle in einer CompactFlash gespeichert und zur Auswertung am PC wieder ausgelesen. Das Testprogramm zur Ermittlung der Daten befindet sich im Anhang (C.3).

Es kann vorkommen, dass mehr als vier schaltbare LED-Anschlüsse nötig sind. Zum Beispliel bei der der Wandverfolgung müssen bis zu fünf LED an- und ausgeschaltet werden. Hierfür ist der in der folgenden Abbildung zu sehende Verteiler geeignet. Er verteilt einen Port auf sieben weitere Anschlussmöglichkeiten.

In diesem Fall ist genau darauf zu achten, was angeschlossen wird. Da es sich um eine Parallelschaltung handelt, ist der Gesamtwiderstand kleiner als der kleinste Einzelwiderstand.

$$\frac{1}{R_{ges}} = \frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n} \tag{5.4}$$

Daraus ergibt sich eine zu geringe Leistungsaufnahme am LED-Port mit zum Beispiel fünf LED's je 80Ω. Als Alternative kommt der IR-Port in Frage. Er kann ebenfalls geschaltet werden und besitzt eine höhere Leistungsaufnahme.

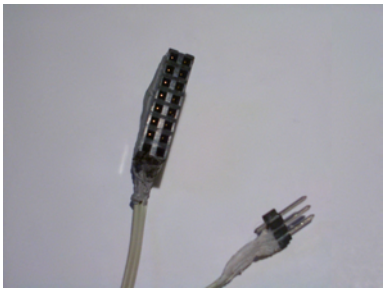

Abbildung 5.12: Verteiler

Vorschlag: Für jede Steuerungseinheit gibt es eigene Anschlussparameter. Für jedes zu installierende Bauteil müssen die Anschlüsse berechnet werden. Gegebenenfalls sind Vorwiderstände vorzusehen.

5.5.2 Konstruktion eines optischen Encoders

Problem: Welche Fragen sind bei der Konstruktion eines optischen Encoders zu klären?

Diskussion: Die wichtigsten Fragen bei der Konstruktion eines optischen Encoders sind die nach der Menge der Segmente und der Positionierung der Segmentscheibe.
Zur Positionierung einige Bemerkungen:
Die Segmentscheibe sollte so angebracht werden, dass sie alle Bewegungen des mobilen Roboters mitmacht. So bietet sich die Position direkt auf der Antriebswelle an. Der Nachteil hierbei ist der mögliche Schlupf der Räder. Dies führt zu fehlerhaften Daten. Das Problem kann behoben werden, in dem zusätzliche, freilaufende Räder nur für die Encoderdatenerfassung angebracht werden. Bei dem hier vorgestellten System wird aber von diesem zusätzlichen Aufwand abgesehen.
Da sich die Scheibe direkt auf der Antriebswelle befindet, kann höchstens der Durchmesser der Räder für die Segmentscheibe gewählt werden. Bevor jetzt die Menge der Segmente zum Theama wird, noch einige technische Bedingungen für einen optischen Encoder. Einerseits geht es darum, mit welcher Geschwindigkeit die anfallenden Signale von der Steuerungseinheit (hier AKSEN-Board) verarbeitet werden können. Das AKSEN-Board kann mit seinen Encodereingängen Signale bis zu 15 kHz verarbeiten (5.5.1). Hier noch einmal der Hinweis, dass nur die high-low-Flanken verarbeitet werden. D.h. nur die Übergänge von schwarz nach weiß werden gezählt. Diese Einschränkung führt somit auch zur Halbierung der möglichen Auflösung.
Andererseits stellt sich die Frage, welche Einschränkungen der optische Sensor, in diesem Fall der Optokoppler cny70, hat. Hier entstand bei der Untersuchung dieser Eigenschaften das Diagramm in Abbildung 5.13. Zu sehen ist der Signalverlauf schwarz-weiß-Wechsel in der Sekunde.

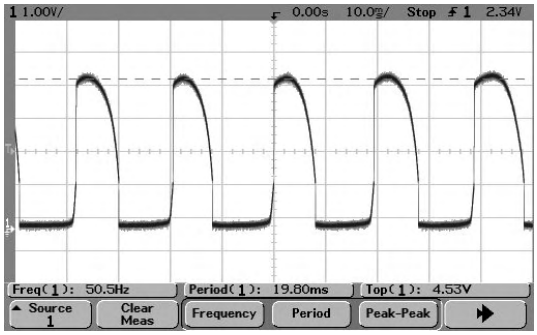

Abbildung 5.13: Signalverlauf des CNY70 bei 50Hz

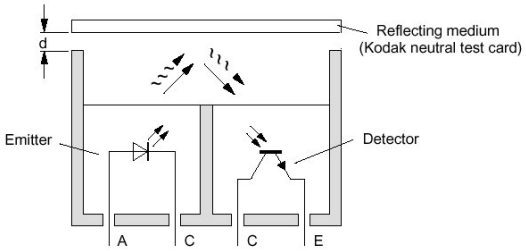

Abbildung 5.14: Funktionsabbildung eines Optokoppler [VT00-1]

Nach der Klärung einiger Bedingungen nun zu der eigentlichen Berechnung der maximalen Segmentanzahl. Dabei soll ein Ausschnitt einer Segmentscheibe zur Anschauung dienen.

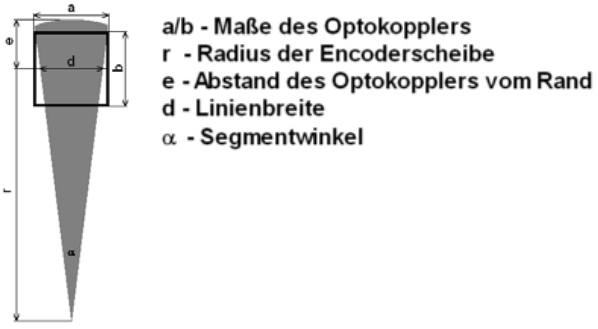

Abbildung 5.15: Skizze Anordnung

Die folgenden Formeln dienen zur Berechnung der maximalen Anzahl der Segmente.

$$\tan \alpha = \frac{d}{r - e} \tag{5.5}$$

$$e \geq \frac{b}{2} \tag{5.6}$$

$$max.Segmentanzahl = \frac{360^\circ}{2 \cdot \alpha} \tag{5.7}$$

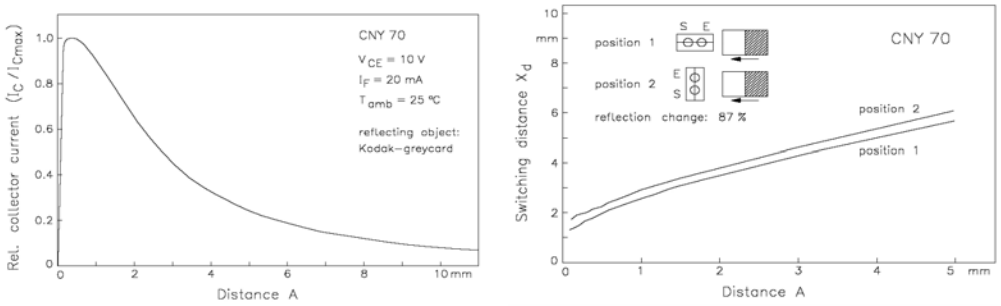

Abbildung 5.16: Techn. Daten von CNY70 [VT00-2]

Vorschlag: Mit den konkreten Werten aus den Datenblättern und der Positionierung in „Eighteen“ ergibt sich Folgendes:

- distance A von 1 mm
- switching distance von d = 3 mm
- Encoderscheibenradius von r = 21 mm
- Einbauabstand e = 10 mm
- berechnet $\alpha = 15^o$ nach Formel 5.4
- berechnete $max.Segmentanzahl = 12$ nach Formel 5.6

Für die gegebenen Vorraussetzungen ergibt sich eine Segmentscheibe mit 12 Segmenten. Als Segment wird nur der schwarze Anteil gesehen.

Zum Schluss noch eine Abbildung der 12er Segmentscheibe und eine Variante, wie diese auf der Antriebswelle befestigt werden kann. Um Stabilität herzustellen, ist eine Pappscheibe mit Silikon auf einer Lego-Riemenscheibe befestigt. Darauf ist die Segmentscheibe aufgeklebt.
Im Anhang gibt es eine kleine Auswahl anderer Segmentscheiben.

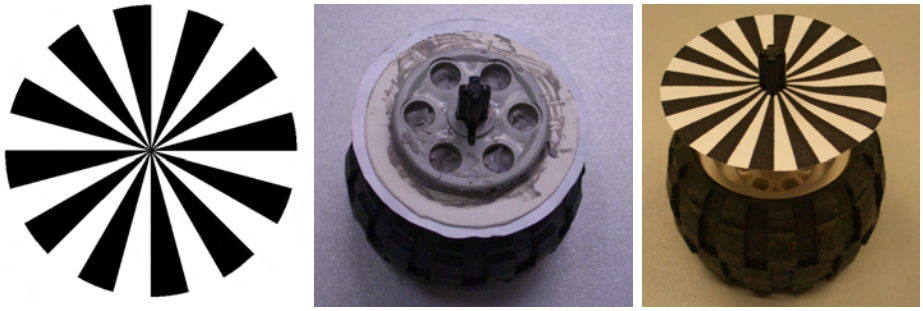

Abbildung 5.17: 12er Segmentscheibe und ihre Befestigung

5.5.3 Entfernungsmessung mit IR

Problem: Wie funktioniert die Entfernungsmessung mittels IR-Sensoren und wie werden diese in ein AMS integriert?

Diskussion Grundsätzlich gibt es zwei Arten, um eine Entfernungsmessung mit infrarotem Licht durchzuführen. Das sind die Weitenmessung mittels Lichteinfallwinkel und die der Lichtmenge. Typische Vertreter für die erste Art sind die SHARP-Sensoren. Wie in der folgenden Abbildung zu sehen ist, sendet der Sensor ein Licht aus, welches von dem Objekt reflektiert wird. Der Winkel, in dem das Licht wieder auf den Sensor auftritt, ist das Maß für die Entfernung des Objektes (siehe Formel und Funktionsprinzip).

$$d = \frac{2 \cdot \tan \alpha}{e} \tag{5.8}$$

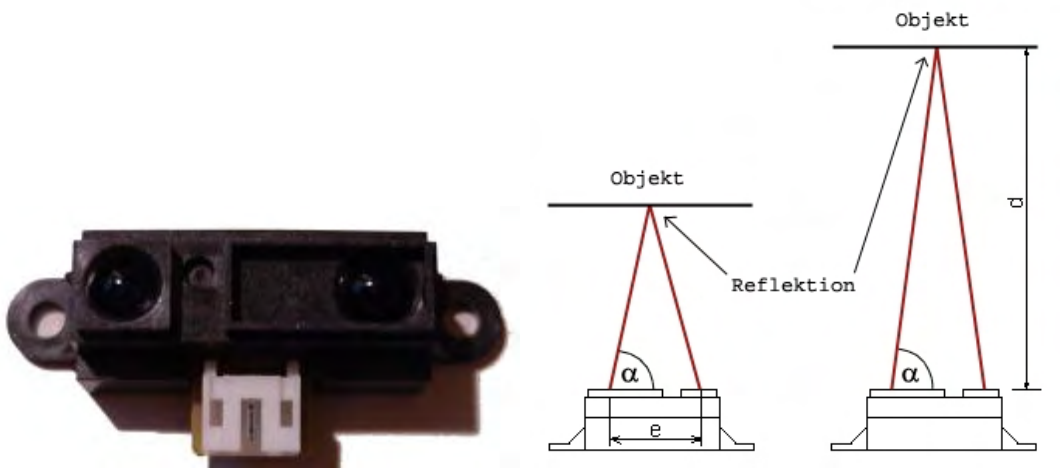

Abbildung 5.18: Foto und Funktionsprinzip der SHARP-Sensoren [Gr04]

Bei der zweiten Art der IR-Entfernungsmessung ist die reflektierte Lichtmenge das Maß der Entfernung. In der folgenden Abbildung sind der Lichtkegel der IR-Sendediode und der Kegel der Empfangsdiode sichtbar. Ein reflektierendes Objekt sorgt für eine Schnittmenge aus beiden Kegeln. Diese Menge kann ein Maß für die Entfernung des Objektes sein. Mehrere Faktoren sorgen für nicht unerhebliche Probleme. So ist es notwendig, Umgebungslicht weitestgehend auszublenden. Auch die Größe des Objektes sorgt für die unterschiedliche Interpretation der Messwerte. Ein größeres Objekt reflektiert mehr Licht als ein kleineres.

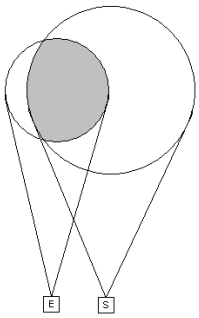

Abbildung 5.19: Funktionsprinzip lichtmengenabhängiger Entfernungsmessung

Aus diesem Funktionsprinzip entstand ein Versuch, bei dem der Abstand vom Sensorinterface zu einem Objekt (Pappe 18x13) untersucht wurde. Das Sensorsignal wurde von Umgebungslicht bereinigt, d.h. das tatsächlich reflektierten Licht wurde vom Umgebungslicht abgezogen.

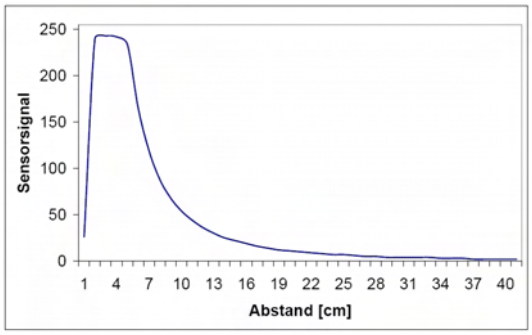

Abbildung 5.20: Signalverlauf lichtmengenabhängiger Entfernungsmessung

Vorschlag: Wenn gerade kein SHARP-Sensor zur Hand ist, kann mit einfachen IR-Sendern und IR-Empfängern ebenfalls eine, zwar nicht so exakte, aber doch wirksame IR-Entfernungsmessung aufgebaut werden.

5.5.4 Sensorinterface zur Linienverfolgung

Problem: Wie soll ein Sensorinterface zur Linienverfolgung aussehen, und welche Positionierung ist geeignet?

Was ist eine Linie?

Diskussion: Eine Linie ist eine sich vom Untergrund abhebende geometrische Fläche mit weitaus größerer Länge als Breite.
Mit drei Sensoren zur Linienverfolgung ergibt sich in erster Betrachtung eine Strategie, wie in Abbildung 5.21 zu sehen ist. Hier soll sich der mobile Roboter, je weiter er von der Linie abkommt, stärker zurück zur Linie drehen.

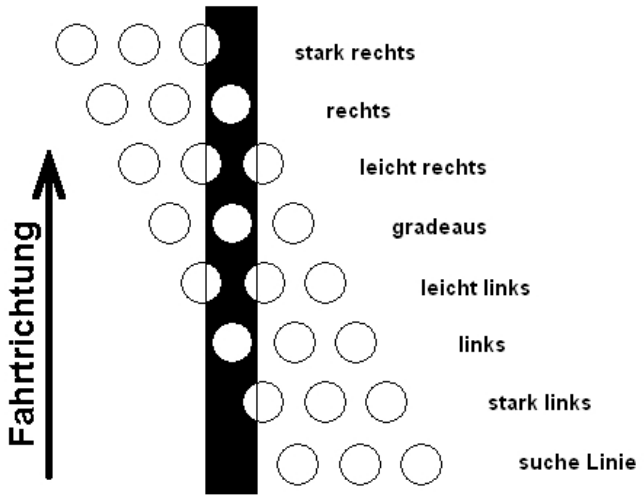

Abbildung 5.21: Strategie zur Linienverfolgung

In vielen Versuchen ergab sich ein Sensorinterface wie in Abbildung 5.22. Es befindet sich mittig an der vorderen Kante, 4 mm über dem Boden. Alle Sensoren sind knapp 4 mm auseinander.

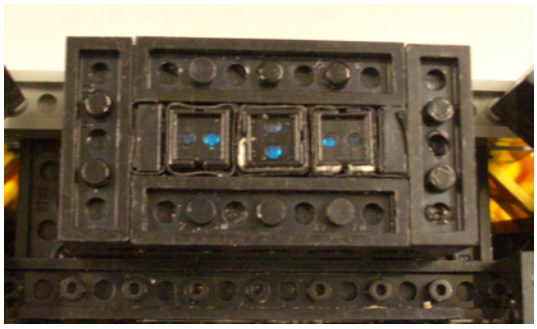

Abbildung 5.22: Sensorinterface zur Linienverfolgung, Variante 1

Mit einem Versuchsaufbau, bei dem ein Blatt Papier mit einer Linie(14 mm breit) um je 1 mm am Sensorinterface vorbei geführt wurde, entstand das Diagramm von Abbildung 5.24. Das entsprechende Testprogramm befindet sich im Anhang C.1. Bei jedem Millimeter Vorrücken werden die Sensordaten über die serielle Schnittstelle ausgegeben. Über das HyperTerminal von Microsoft können diese empfangen und an geeigneter Stelle weiter verarbeitet werden.

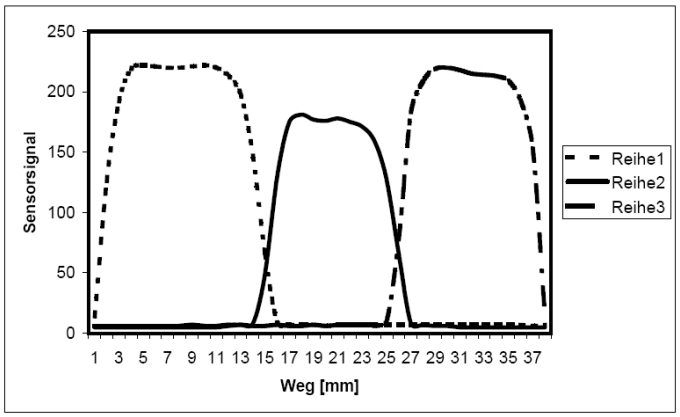

Abbildung 5.23: Signale eines Sensorinterface zur Linienverfolgung 1

Es ist zu erkennen, dass die Strategie aus der Abbildung 5.21 mit diesem Sensorinterface nicht zu realisieren ist. Grund ist die nicht eindeutige Interpretation der Signale für eine P-Regelung. D.h. bei fallender Kurve des mittleren Sensors ist nicht zu erkennen, ob sich das AMS zu weit links oder rechts befindet. Es können nur Aussagen getroffen werden, wenn die äußeren Sensoren eine Linie detektieren. Das bedeutet dann, das AMS ist rechts oder links. Daraus ergibt sich die neue Strategie: Wenn zu weit links, dann drehe solange nach rechts, bis der mittlere Sensor wieder die Linie detektiert. Das Gleiche trifft auch für die andere Richtung zu. Das Programm könnte folgendermaßen aussehen:

```
funktion follow_line1()
begin
      while true do
               if SensorMitte > SCHWELLE then
                     linkerMotor an
                     rechterMotor an

               else if SensorLinks > SCHWELLE then
                     linkerMotor aus
                     rechterMotor an

               else if SensorRechts > SCHWELLE then
                     linkerMotor an
                     rechterMotor aus

               //trifft nichts zu, bleibt alles beim alten
               end
       end
end
```

Eine Möglichkeit, vielleicht doch die erste Strategie verfolgen zu können ist die, mit nur zwei Optokopplern dicht nebeneinander zu arbeiten. Hierzu wieder das Diagramm zum Signalverlauf:

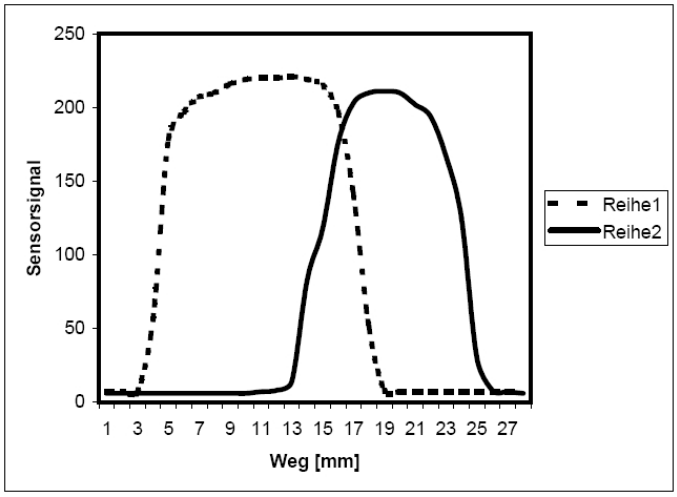

Abbildung 5.24: Signale eines Sensorinterface zur Linienverfolgung 2

Jetzt ist eine konkrete Interpretation der Sensordaten möglich. Jeweils ein Anstieg pro

Sensor bildet mit einem Anstieg des anderen eine Schnittmenge. Im Folgenden soll wieder ein Beispiel für die Umsetzung in C gegeben werden.

```
funktion follow_line2()
beginn
      while true do

            if sensorLinks or sensorRechts then

                  motorLinks = 10
                  motorRechts = 10

            else if sensorLinks > sesorRechts then

                  motorLinks = sensorLinks / 25
                  motorRechts = 10

            else if sensorRechts > sensorLinks then

                 motorLinks = 10
                 motorRechts = sensorRechts / 25
            //trifft nichts zu, bleibt letztes Kommando bestehen

            end
      end
end
```

Vorschlag: Sich für eine der beiden Varianten zu entscheiden ist stark von der Geschwindigkeit abhängig. So kann in der ersten Variante der mobile Roboter mit voller Geschwindigkeit fahren, dafür mitunter in starkem Zickzackkurs. Die zweite Variante funktioniert dagegen nur mit geringer Geschwindigkeit. Die Fahrt verläuft aber ruhiger an der Linie entlang.

5.5.5 Sensorinterface zur Wandverfolgung

Problem: Wie soll ein Sensorinterface zur Wandverfolgung aussehen und welche Positionierung ist geeignet?

Diskussion Für die Wandverfolgung sind die in dem Anschnitt 5.5.3 diskutierten IF-Sensoren notwendig. Diese Sensoren werden wie in der folgenden Abbildung am autonomen, mobilen Roboter angebracht.
Ähnlich wie bei dem Sensorinterface zur Linienverfolgung sind auch hier mindestens zwei

Stategien möglich. Bei der Ersten reagiert das AMS, wenn eine bestimmte Entfernung über- bzw. unterschritten ist. Bei Unterschreitung soll beispielsweise nach links und bei Überschreitung nach rechts gesteuert werden. Das Ziel ist es also, das AMS in einem bestimmten Korridor an der Wand entlang zu führen. Es handelt sich hier um die klassische Zweipunktregelung („Fenster auf- Fenster zu“).
Bei der zweiten Variante kommt wieder der PID-Regler zum Einsatz. D.h., ist der Roboter weit von der Wand entfernt, soll stark gegengeregelt werden und ist er dicht daran, weniger.
In der folgenden Abbildung ist die Anordnung der IR-Sensoren (Kapitel 5.5.3) zu sehen, die für die Wandverfolgung notwendig sind. Auf dem Foto ist die Sendediode links und die Empfangsdiode rechts zu sehen.

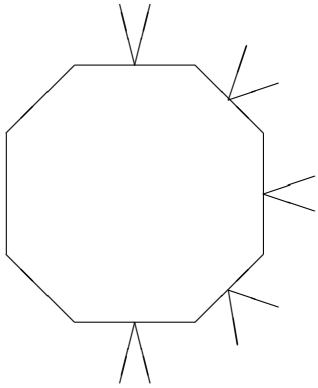

Abbildung 5.25: Sensoranordnung zur Wandverfolgung

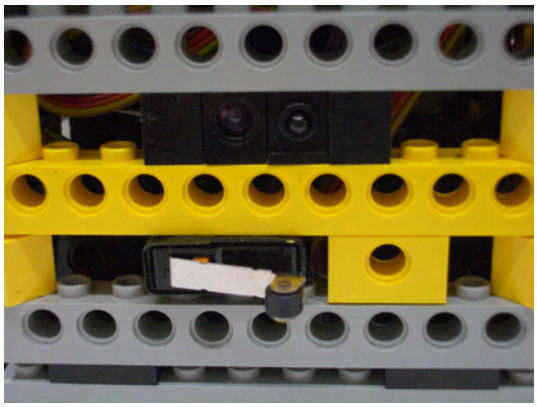

Abbildung 5.26: Sensorinterface zur Wandverfolgung

Vorschlag: Je nach Wahl, Menge und Anordnung der Entfernungssensoren hat die Wandverfolgung eine unterschiedliche Qualität. Der Vorschlag ist, lieber etwas mehr Geld für die Sensoren auszugeben, um eine ordentliche Funktionalität zu gewährleisten.

5.5.6 Sensorinterface zur Lichtverfolgung

Problem: Wie soll ein Sensorinterface zur Lichtverfolgung aussehen und welche Positionierung ist geeignet?

Diskussion Die Strategie ist es, den Roboter in die Richtung zu bewegen, aus der der Sensor die höchsten Werte liefert.
Begonnen wird mit einer 360 Grad Drehung, um das ganze Umfeld abzudecken. Danach kann davon ausgegangen werden, dass sich die Lichtquelle vor oder seitlich vom System befindet.

Vorschlag: Wird zur IR-Entfernungsmessung die Lichtmenge herangezogen, kann das selbe Interface wie bei der Wandverfolgung genutzt werden. Eine Abdeckung von 180 Grad sind für Lichtsuche und Lichtverfolgung völlig ausreichend.

5.5.7 Bumper

Problem: Wo und wofür sind Bumper bei einem autonomen, mobilen Roboter notwendig?

Diskussion Um ein autonomes, mobiles Fahrzeug gegen Zerstörung anderer oder sich selbst zu schützen, sind Mechanismen wie Bumper notwendig. Die Bumper führen nach mechanischem Kontakt zu einer vorher im Programm implementierten Reaktion.
Die Bumper dienen gewissermaßen als „Notaus“. Sie sind an den Seiten befestigt, in die sich der autonome, mobile Roboter bewegen kann.

```
funktion bumper()
var i: Integer
begin
   while true do

      for i=0 to i<AnzahlBumper do

         if bumperi = 0 then
                              motorLinks aus
                              motorRechts aus
         end
      end
   end
end
```

Vorschlag: In der Abbildung 5.26 ist neben dem Sensorinterface zur Wandverfolgung auch ein Taster zu sehen. Er ist mit Klebeband umwickelt und zwischen zwei Legosteinen eingeklemmt. Von den acht Kanten des „Eighteen“ sind die sechs Kanten mit einem Bumper versehen, in die sich das AMS bewegen kann. Das sind alle Seiten außer denen im rechten Winkel zur Fahrrichtung.

6 Tutorial Teil 2: Programmierung eines AMS

In diesem Kapitel wird beschrieben, wie ein bestimmtes Verhalten des Roboters durch die Software erreicht wird. Die Konfiguration der Anschlüsse und wie einige Funktionen arbeiten, wird ebenfalls beschrieben.

6.1 AKSEN-Roboter konfigurieren

In diesem Abschnitt soll beschrieben werden, wie ein AKSEN-Roboter konfiguriert wird. Bei der Menge der Anschlüsse muss dem System mitgeteilt werden, was wo angeschlossen ist. Dazu folgt ein Beispiel, wie die Konfigurationsbefehle aus Tabelle 4.5 anzuwenden sind.

```
1 #include <stdio.h>
2 #include <regc515c.h>
3 #include <stub.h>
4 #include "arbs.h"
5
6 void AksenMain(void){
7
8      init();
9
10     set_motor_links(0);     //linker Motor an Motor-Port 0
11     set_motor_rechts(1);     //rechter Motor an Motor-Port 1
12     set_encoder_links(0);    //linker Encoder an Encoder-Port 0
13     set_encoder_rechts(1);   //rechter Encoder an Encoder-Port 1
14     set_seganz(24);          //Encoderscheibe auf 24 Segmente gesetzt
15
16     set_linie1(0); //erste Liniensensor an Analog-Port 0
17     set_linieN(2); //Anzahl der Liniensensoren=2,Port 0-1
18
19     while(1);
20 }
```

Die Konfigurationen sind nur notwendig, wenn die Default-Werte nicht mit den aktuellen Bedingungen übereinstimmen. Als Default sind die Bedingungen von „Eighteen“ eingestellt. Die weiteren Set-Funktionen, wie für den Raddurchmesser oder Radabstand, sind in der Tabelle 4.5 nachzuschlagen.

Wie die einzelnen Verhalten in die Subsumtionsachitektur eingeordnet sind ist in der folgenden Abbildung zu sehen. Leider sind sie einzelnen Verhalten durch ihre Anordnung mit einer festen Priorität vorbestimmt. Der main-Prozess hat die höchte Priorität und kann somit alle anderen Prozesse überstimmen. Ansonsten ist in der folgenden Abbildung zu sehen, dass je höher der Prozess in der Struktur angeordnet ist, die Priorität wächst. In einer späteren Erweiterung könnte man sich mit dem Problem der festen Zuteilung befassen und eine flexiblere Prioritätenzuteilung entwickeln.

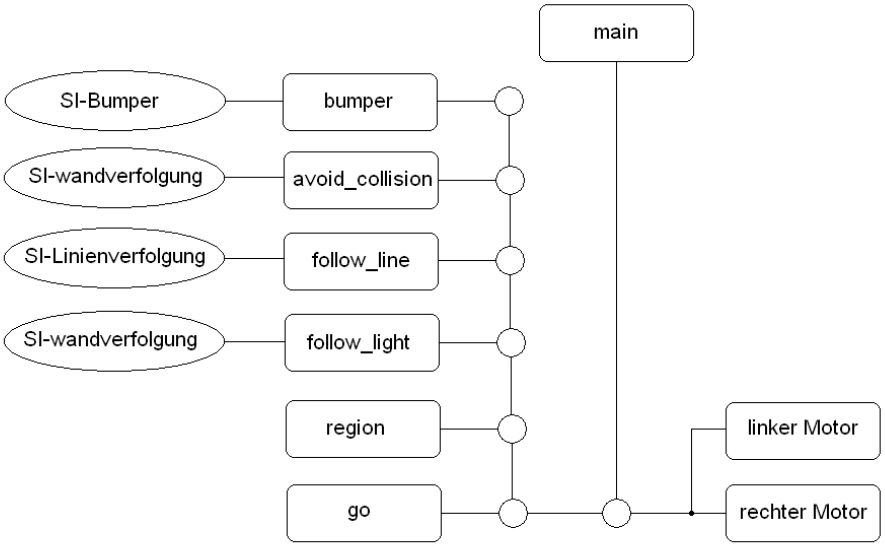

Abbildung 6.1: Subsumtion aller Verhalten

6.2 Die Geradeausfahrt

Mit **move(400)** soll der mobile Roboter einen Weg von 400 mm fahren. Für die Ermittlung der gefahrenen Wegstrecke dienen die Informationen des Encoders. Mit den gezählten Ticks lassen sich, je nach Raddurchmesser und Anzahl der Segmenten, Rückschlüsse auf den zurückgelegten Weg schließen. Die folgende Formel soll dies verdeutlichen.

$$Ticks = \frac{Weg_{soll} \cdot Segmentanzahl}{\Pi \cdot Raddurchmesser} \tag{6.1}$$

Außerdem soll der Roboter so weit wie möglich auf einer geraden Linie fahren. Dazu werden die Encoderdate über einen PID-Regler verarbeitet und die Motoren entsprechend angesteuert.

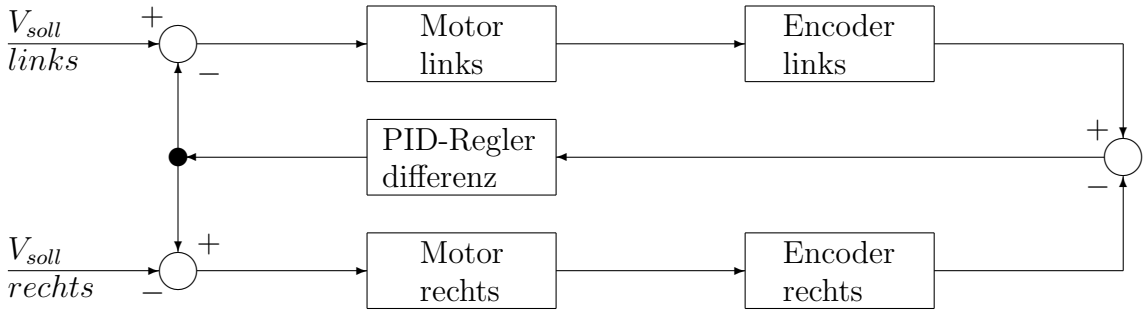

Abbildung 6.2: Synchronisieren der Räder für die Fahrt auf einer geraden Linie

6.3 Das Drehen

Mit **turn(90)** soll der mobile Roboter eine Drehung von 90 Grad durchführen. Ähnlich wie bei dem Move-Befehl sind auch hier die Encoderdaten notwendig. Mit der folgenden Formel werden die Ticks berechnet.

$$Ticks = Radabstand \cdot \frac{Grad_{soll}}{360} \cdot \frac{Segmentanzahl}{Raddurchmesser} \tag{6.2}$$

6.4 Fahren im Quadrat

Mit den neuen Skills soll ein Programm den Roboter dazu veranlassen in einem Quadrat zu fahren, d.h. viermal geradeaus fahren mit anschließender 90^o -Drehung.

```
1 #include <stdio.h>
2 #include <regc515c.h>
3 #include <stub.h>
4 #include "arbs.h"
5
6 void AksenMain(void){
7
8     int i;
9     init();
10
```

```
11    for(i=0;i<4;i++){
12
13              move(400); //fahre 400mm
14              sleep(200);
15              turn(90); //drehe 90 Grad
16              sleep(200);
17     }
18    while(1);
19 }
```

6.5 Abfahren einer Fläche

Im Gegensatz zu dem Fahren im Quadrat aus Abschitt 6.4 handelt es sich hier nicht um eine blockierende Funktion aus dem main-Prozess. Das Abfahren soll ein nicht blockierendes Verhalten darstellen.

Mit einer Strategie und internen Koordinaten soll der Roboter alle Punkte einer Fläche abfahren. Eine konkrete Anwendung könnte ein Rasenmäher oder ein Staubsauger sein.

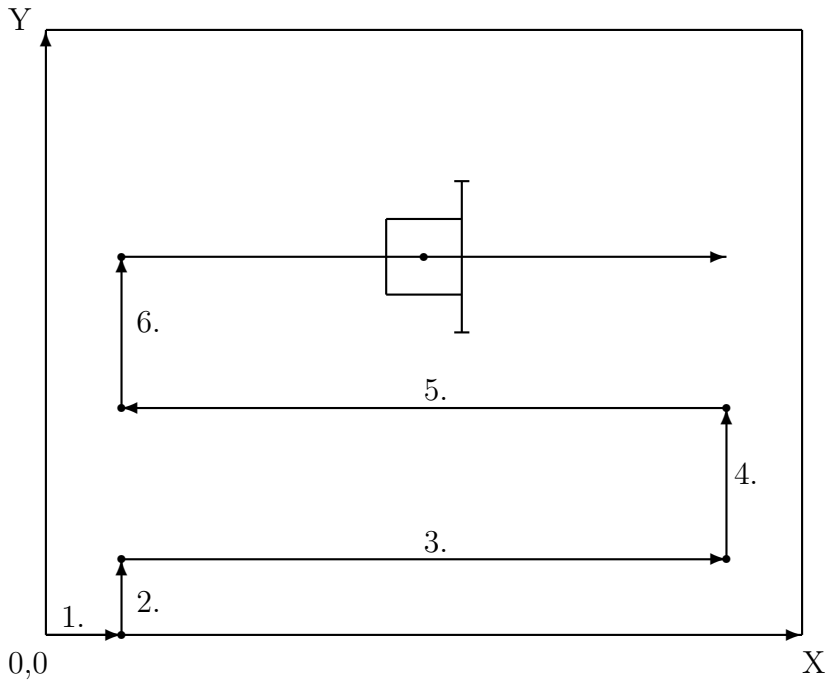

Abbildung 6.3: Stategie zur Abfahrt einer Fläche

Theoretische Betrachtung:
1. fahre bis x=0.5 * Breite(bzw. y=0.5 * Breite)
2. drehe auf 90 Grad (bzw. 0 Grad), fahre bis y=0.5 * Breite(bzw. x=0.5 * Breite)
3. drehe auf 0 Grad (bzw. 90 Grad), fahre bis x=xmax-(0.5 * Breite),y=konst
4. drehe auf 90 Grad , fahre bis y= yalt + Breite
5. drehe auf 0 Grad, fahre bis x = 0.5 * Breite
6. zu Punkt 2. bis y=ymax

Hierzu zur Veranschaulichung der Quellcode.

```
void region(void){

     unsigned int roboterbreite;
     unsigned int xachse;
     unsigned int yachse;
     unsigned int ycontrol;
     unsigned int ycheck;

     ycheck= yachse / roboterbreite;

     while(ycheck<ycontrol){

          move(roboterbreite * 0.5);
          turn(90);
          move(roboterbreite * 0.5);
          turn(-90);

          while(1){
                 move(xachse - roboterbreite);
                 turn(90);
                 move(roboterbreite);ycontrol++;
                 turn(90);
                 move(xachse - roboterbreite);
                 turn(-90);
                 move(roboterbreite);ycontrol++;
                 turn(-90);
          }
     }
}
```

6.6 Linienverfolgung

Besitzt der Roboter entsprechende Sensoren zum Erkennen einer Linie, soll das folgende Programm ihn dazu veranlassen, dieser zu folgen.

```
1 void follow_line()
2 {
3     unsigned int SCHWELLE = 180;
4
5     while(1)
6     {
7     unsigned int links  = analog(5);
8     unsigned int mitte  = analog(6);
9     unsigned int rechts = analog(7);
10
11    if(mitte > SCHWELLE)       command = MOVE;    //beide Motoren an
12    else if(links > SCHWELLE)  command = LINKS;   //nur Motor links an
13    else if(rechts > SCHWELLE) command = RECHTS;  //nur Motor rechts an
14    //trifft nichts zu, behalte letzte Kommando bei
15    }
16 }
```

6.7 Verhalten avoid_collision

In diesem Abschnitt soll der Quellcode vorgestellt werden der festlegt, wie das Verhalten *avoid_collision* implementiert ist. Wieder ist zu sehen, dass konkrete Konfigurationsbefehle herangezogen werden, hinter denen die entsprechende Belegung des AKSEN-Boards steht.

```
void avoid_collision(void){

unsigned char SCHWELLE = 80;
unsigned char motor_links = get_motor_links();
unsigned char motor_rechts = get_motor_rechts();
unsigned char wand1 = get_wand1(); //erster Wandsensor
unsigned char wandN = get_wandN();                //Anzahl der Wandsensoren
unsigned char i;

      while(1){
```

```
          for(i=wand1;i<wandN;i++){

                verglwert=analog(i);

                if(verglwert -(analog(i))>SCHWELLE) {

                      motor_pwm(motor_links,0);
                      motor_pwm(motor_rechts,0);
                  }
            }
        process_defer();
      }
}
```

6.8 Verhalten bumper

In diesem Abschnitt soll der Quellcode vorgestellt werden, wie die Verhalten *bumper* implementiert ist. Wieder ist zu sehen, dass konkrete Konfigurationsbefehle herangezogen werden, hinter denen die entsprechende Belegung des AKSEN-Boards steht.

```
void bumper(void){
unsigned char motor_links = get_motor_links();
unsigned char motor_rechts = get_motor_rechts();
unsigned char bumper1 = get_bumper1();
unsigned char bumperN = get_bumperN();
unsigned char i;

      while(1){

            for(i=bumper1;i<=bumperN;i++){

                  if(digital_in(i)==0){
                        motor_pwm(motor_links,0);
                        motor_pwm(motor_rechts,0);
                    }
             }
      }
}
```

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Zusammenfassend läßt sich sagen, dass sich diese Arbeit mit dem Bau eines autonomen, mobilen Systems befasst hat. Grundlage waren hierfür das AKSEN-Board als Steuerungseinheit, ein Lego-Bausatz, einfache Elektromotoren und diverse Sensoren. Das scheinbar einfache Zusammenbauen von Lego-Bausteinen erwies sich als nicht so günstig. Es fing an mit der dauernden Suche nach dem passenden Baustein und endete mit der Suche nach der richtigen Konstruktion. Etwas mehr Erfahrung mit den Lego-Bausätzen wäre hier sehr hilfreich gewesen.
Schwierig war es auch, aus dem riesigen Bereich von AMS eine Abgrenzung zu finden, und doch ein kompelettes System mit allen seinen Grundbestandteilen zu beschreiben. Die Kombination der Methoden aus den verschiedenen Fachdiziplinen, wie Elektronik, Maschinenbau und Informatik, stellte eine weitere Herausforderung dar. Trotz aller Schwierigkeiten ist eine robuste, funktionstüchtiger, autonomer, mobiler und formschöner Roboter entstanden.

7.2 Ergebnis

Dem Leser ist mit der vorliegenden Arbeit ein Hilfsmittel an die Hand gegeben, welches ihm erlaubt, ein AMS nachzubauen (Tutorial Teil1) und nachzuprogrammieren (Tutorial Teil2). Es liegt eine umfangreiche Bibliothek vor, in der die Bewegungsbefehle, Konfigurationsbefehle und Verhaltensbefehle enthalten sind.
Dennoch konnten nicht alle Probleme beim Bau eines autonomen, mobilen Roboters erfasst und Lösungsansätze entwickelt werden. Der Grund dafür ist in dem viel zu komplexen Gebiet der Roboterentwicklung zu sehen.

7.3 Ausblick

Der Ausblick auf Roboter mit eigenem Bewusstsein würde hier etwas zu weit gehen. Nur die Anregung zu einem „Roboterführerschein“ für alle Entwickler und Nutzer sei erlaubt. Wer weiß schon, was das Militär mit völlig autonomen Robotern erreichen will. Wünschenswerter ist da schon das spielerische Herangehen an Problemlösungen innerhalb von Roboterwettbewerben, z.B. dem Robocup oder der FIRST LEGO League.

Als zukünftige Erweiterung, bezogen auf das AKSEN-Board, könnte eine Simulationsumgebung auf dem PC sehr hilfreich sein. Eingangsparameter würden sich per Hand einstellen lassen und Ausgabedaten, je nach Progammabarbeitung, simuliert werden. Vielleicht bietet sich auch gleich eine Erweiterung zur Fernbedienung an. Über entprechende Schnittstellen könnten die Ausgangsports manipuliert werden. In der Zukunft wird das AKSEN-Board nach wie vor eine zentrale Rolle bei der Weiterentwicklung innerhalb der RCUBE-Plattform spielen.

Literaturverzeichnis

[AA99] Altenburg, J./Altenburg U. : *Mobile Roboter*, Carl Hanser Verlag München Wien, 1999

[AB87] Ausborn, W./Bätz, H./Heckert, J. : *Elektrische Bauelemente und Baugruppen der Automatisierungstechnik*, Verlag Technik, Berlin, 1987

[As04] Asimov, Issac : *http://www.asimovonline.com*, 2004

[AS05] AKSEN Shop: *http://www.aksen-roboter.de/stage/index.html*

[Bh91] Brockhaus : *Brockhaus-Enzyklopädie*, F.A. Brockhaus GmbH, Mannheim, 1991

[Br03] Bräunl, Thomas : *Emdebbed Robotics*, Springer-Verlag Berlin Heidelberg, 2003

[Ca11] Capek, Karel : *http://capek.misto.cz/english/index.html*, 2004

[Da03] Dahlmann, Oliver: *http://users.informatik.haw-hamburg.de/ kvl/ dahlmann/diplom.pdf*

[Gr04] Greif, Fabian : *http://www.kreatives-chaos.com/index.php*

[JF96] Jones, Joseph L. / Flynn, Anita. M. : *Mobile Roboter*, Addison-Wesley, Bonn, 1996

[KI05] Labor für Künstliche Intelligenz an der FHB : *http://ots.fh-brandenburg.de/index.php*

[Kn91] Knieriemen, Thomas : *Autonome Mobile Roboter*, BI-Wiss.-Verl., Mannheim, 1991

[Ku01] Kurzweil, Ray : *Homo Sapiens*, Econ-Taschenbuch-Verlag, München, 2001

[Ra97] von Randow, Gero : *Roboter - unsere nächsten Verwandten*, Rowohlt Verlag GmbH, Reinbek bei Hamburg, 1997

[Sc86] Scheffel, Detlev : *Automatische Steuerungen*, Verlag Technik, Berlin, 1986

[St04] Dr. Steinmüller, Johannes : *http://www-user.tu-chemnitz.de/ stj/lehre/ROBO.pdf*

[SW00] Schneider, U./Werner, D. : *Taschenbuch der Informatik*, Carl Hanser Verlag München Wien, 2000

[VR02] Vision Rundschau Nr.74/August2002 : *http://www.ams.or.at/wien/biz/vision/archiv/rund74.htm*

[Wi84] Wilke, Manfred : *Im Zeichen der Roboter*, VEB Deutscher Verlag der Wissenschaften, Berlin, 1984

[VT00-1] Vishay Telefunken : *http://frederic.ducrocq.free.fr/Station_meteo/ datasheet/CNY70.pdf*

[VT00-2] Vishay Telefunken : *http://www.build-a-bot.com/datasheets/ cny70_an.pdf*

[WI05] WIKIPEDIA : *http://de.wikipedia.org/wiki/Sensor*

[Wü04] Prof. Dr. Wülker, Michael : *http://mv-sirius.m.fh-offenburg.de/ robotik/SS04Material/Konstruktionen-2004-SS.pdf*

Bilder

[1] *http://www.delos.fantascienza.com/delos55/img/robot/cinema/rur.jpg*

[2] *http://www.honda-taiwan.com.tw/asimo/images/asimo.jpg*

[3] *http://pcweb.mycom.co.jp/special/2002/aibo/images/01.jpg*

[4] *http://www.br-online.de/kinder/fragen-verstehen/wissen/2003/00329/sprengstoff.jpg*

[5] *http://www.tappotec.com/Bilder/Arin/Gallery/NeuWalk1.jpg*

[6] *http://www.3sat.de/3sat.php?http://www.3sat.de/nano/news/26327/*

Abbildungsverzeichnis

1.1 Autonomie-Grade 2
1.2 „RUR“[1] 3

2.1 Ablaufplan 7
2.2 Getriebe 9
2.3 Schneckenrad 9
2.4 Spreng-[4],Schreit-[5],Kriech-Roboter[6] 10
2.5 Einige Radanordnungen 10
2.6 Klassifikationsschema für Sensorsysteme der Robotik 12
2.7 AKSEN-Board [KI05] 13
2.8 Programmstruktur 15

3.1 Roboterformen 18

4.1 Programmbeispiel 22
4.2 Weltmodell-Architektur 23
4.3 Subsumtions-Architektur 25

5.1 Draufsicht „Eighteen“ 33
5.2 Vertikale Verbindungen 34
5.3 Verstrebungen am Gehäuse von „Eighteen“ 34
5.4 Position des Schwerpunktes 35
5.5 Lego-Motor 36
5.6 Beispiel Zahnradanordnung 37
5.7 Getriebe „Eighteen“ 38
5.8 Wendekreis eines Dreiradantriebes 39
5.9 Nachlaufrad 40
5.10 Stützkugel 41
5.11 Diagramm Encoder-Eingang 42
5.12 Verteiler 43
5.13 Signalverlauf des CNY70 bei 50Hz 44
5.14 Funktionsabbildung eines Optokoppler [VT00-1] 44
5.15 Skizze Anordnung 44

5.16 Techn. Daten von CNY70 [VT00-2] 45
5.17 12er Segmentscheibe und ihre Befestigung 46
5.18 Foto und Funktionsprinzip der SHARP-Sensoren [Gr04] 46
5.19 Funktionsprinzip lichtmengenabhängiger Entfernungsmessung 47
5.20 Signalverlauf lichtmengenabhängiger Entfernungsmessung 47
5.21 Strategie zur Linienverfolgung 48
5.22 Sensorinterface zur Linienverfolgung, Variante 1 49
5.23 Signale eines Sensorinterface zur Linienverfolgung 1 49
5.24 Signale eines Sensorinterface zur Linienverfolgung 2 50
5.25 Sensoranordnung zur Wandverfolgung 52
5.26 Sensorinterface zur Wandverfolgung 52

6.1 Subsumtion aller Verhalten 56
6.2 Synchronisieren der Räder für die Fahrt auf einer geraden Linie 57
6.3 Stategie zur Abfahrt einer Fläche 58

A.1 „ASIMO“[2] 69
A.2 „Aibo“[3] 69
A.3 Draufsicht auf „Eighteen“ 70
A.4 Unterseite von „Eighteen“ 70
A.5 Unterseite von „Eighteen“ (seitlich) 71
A.6 Antriebsblock 71
A.7 Linke Seite Antriebsblock 72
A.8 Bodenfreiheit Antriebsblock 72
A.9 8er Segmentscheibe 73
A.10 12er Segmentscheibe 73
A.11 16er Segmentscheibe 73
A.12 32er Segmentscheibe 73

Tabellenverzeichnis

2.1 Sensorenübersicht(Quelle:Conrad '05) 12

3.1 Kostenbeispiel [AS05] 18

4.1 AKSEN-Funktionen Teil1(Version 0.956) 19
4.2 AKSEN-Funktionen Teil2(Version 0.956) 20
4.3 AKSEN-Funktionen Teil3(Version 0.956) 21
4.4 Konfigurationsbefehle 27
4.5 Verhaltensbefehle 28

B.1 Frequenz Encoder-Eingang 74
B.2 Sensordaten zur Linienverfolgung 1 75
B.3 Sensordaten zur Linienverfolgung 2 76
B.4 Sensordaten zur Wandverfolgung 77

A Fotos

A.1 Roboterbeispiele

Abbildung A.1: „ASIMO“[2]

Abbildung A.2: „Aibo“[3]

A.2 Weitere Fotos von „Eighteen“

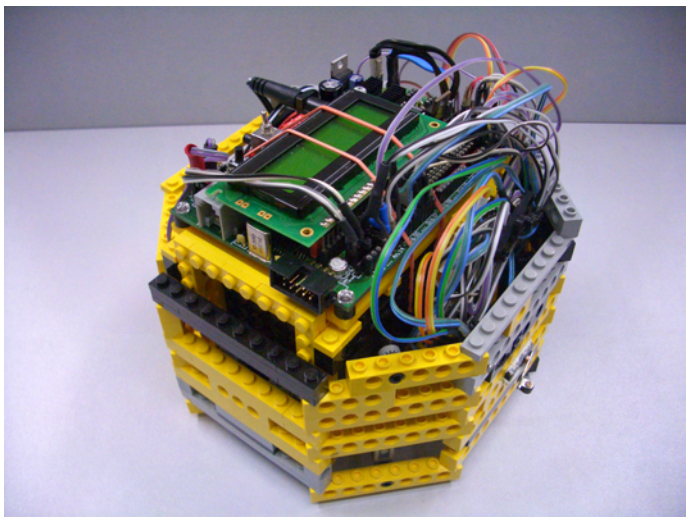

Abbildung A.3: Draufsicht auf „Eighteen“

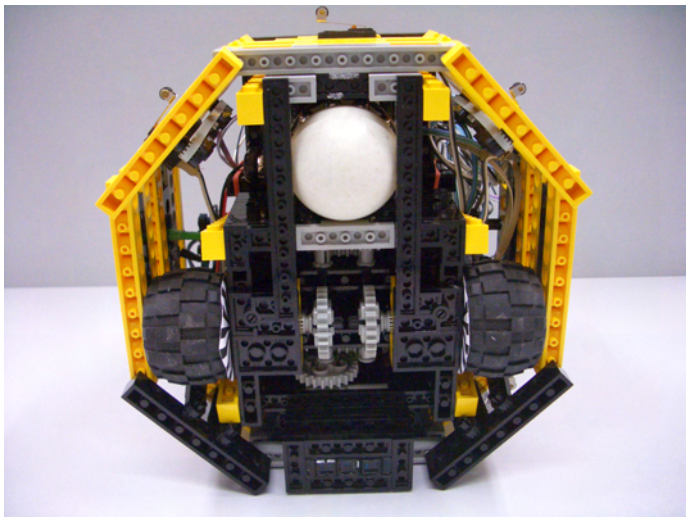

Abbildung A.4: Unterseite von „Eighteen“

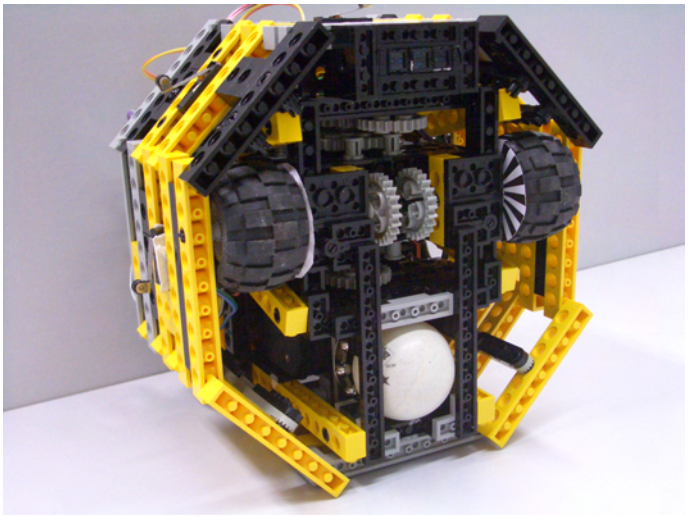

Abbildung A.5: Unterseite von „Eighteen“ (seitlich)

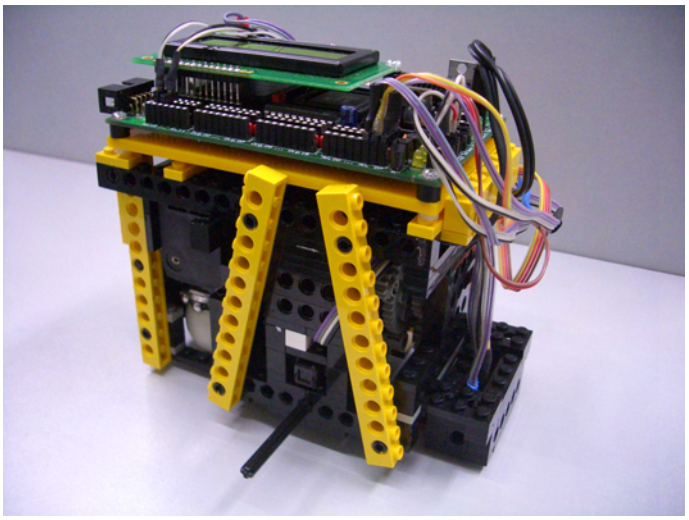

Abbildung A.6: Antriebsblock

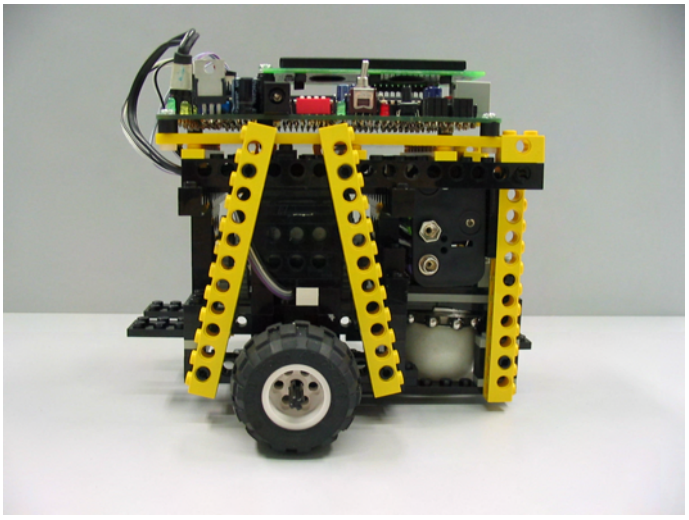

Abbildung A.7: Linke Seite Antriebsblock

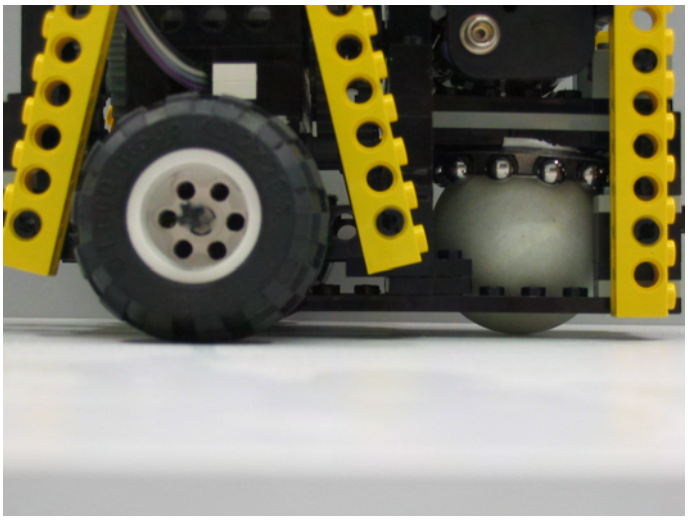

Abbildung A.8: Bodenfreiheit Antriebsblock

A.3 Auswahl von Segmentscheiben

Abbildung A.9: 8er Segmentscheibe

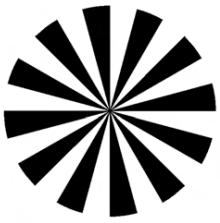

Abbildung A.10: 12er Segmentscheibe

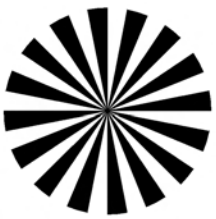

Abbildung A.11: 16er Segmentscheibe

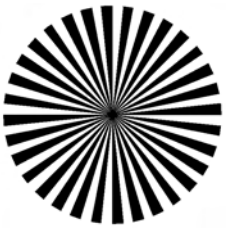

Abbildung A.12: 32er Segmentscheibe

B Messwert-Tabellen

B.1 Encoder-Eingang AKSEN-Board

Die Tabelle zeigt pro Frequenz eine Messreihe von acht Werten. Aus diesen Werten werden das Minimum, Maximum und das arithmetische Mittel ermittelt.

f [kHz]	1.	2.	3.	4.	5.	6.	7.	8.	min	med	max
1	101	100	100	101	100	101	101	100	100	100,5	101
2	202	201	202	201	202	201	202	201	201	201,5	202
3	300	303	302	302	302	302	303	302	300	302,0	303
4	404	404	404	404	404	404	404	403	403	403,9	404
5	502	503	503	503	504	504	503	504	502	503,3	504
6	609	608	607	607	607	607	607	607	607	607,4	609
7	703	706	706	705	706	707	706	705	703	705,5	707
8	787	802	803	790	798	795	801	801	787	797,1	803
9	907	908	909	908	909	909	908	910	907	908,5	910
10	938	952	953	956	945	954	951	960	938	951,1	960
11	1021	1025	1021	1025	1021	1021	1021	1023	1021	1022,3	1025
12	1218	1218	1217	1219	1219	1217	1219	1217	1217	1218,0	1219
13	1324	1320	1323	1321	1320	1321	1321	1321	1320	1321,4	1324
14	1208	1213	1211	1212	1214	1212	1214	1212	1208	1212,0	1214
15	1433	1430	1428	1428	1428	1427	1430	1428	1427	1429,0	1433
16	1540	1540	1540	1537	1538	1538	1541	1540	1537	1539,3	1541
17	1685	1925	1927	1941	1932	1916	1939	1943	1685	1901,0	1943
18	1539	1526	1528	1528	1527	1534	1533	1539	1526	1531,8	1539
19	1662	1989	1987	1991	2008	1950	1994	1991	1662	1946,5	2008
20	2027	5960	6186	6159	6220	6154	6145	6099	2027	5618,8	6220
21	1929	3836	3995	4044	3960	3994	4020	3940	1929	3714,8	4044
22	2123	6398	6775	6851	6762	6815	6795	6824	2123	6167,9	6851
23	1030	824	617	412	207	100	101	100	100	423,9	1030

Tabelle B.1: Frequenz Encoder-Eingang

B.2 Sensorinterface Variante 1

Weg [mm]	linker Sensor	mittlerer Sensor	rechter Sensor
1	14	5	6
2	126	5	6
3	192	5	6
4	219	5	6
5	222	5	6
6	221	5	6
7	220	5	6
8	220	5	6
9	221	5	7
10	222	5	6
11	220	5	6
12	213	6	7
13	198	7	7
14	147	8	6
15	66	49	6
16	9	130	7
17	7	175	6
18	7	181	6
19	7	177	7
20	6	176	6
21	7	178	7
22	7	175	7
23	7	171	7
24	6	159	7
25	7	127	10
26	7	65	79
27	7	8	183
28	7	7	209
29	7	6	219
30	7	6	220
31	7	5	218
32	7	5	215
33	7	5	214
34	7	5	213
35	7	5	209
36	7	5	193
37	6	5	153
38	7	5	15

Tabelle B.2: Sensordaten zur Linienverfolgung 1

B.3 Sensorinterface Variante 2

Weg [mm]	linker Sensor	rechter Sensor
1	7	6
2	7	6
3	7	6
4	56	6
5	179	6
6	197	6
7	207	6
8	210	6
9	216	6
10	219	6
11	220	7
12	220	8
13	221	14
14	219	83
15	215	118
16	197	176
17	141	203
18	51	210
19	8	211
20	7	210
21	7	202
22	7	194
23	7	167
24	7	125
25	7	30
26	7	8
27	7	7
28	7	6

Tabelle B.3: Sensordaten zur Linienverfolgung 2

Die Tabelle zeigt die Ausgabewerte von zwei Optokopplern für jeden Millimeter, einer Linie vorbei.

B.4 Sensorinterface Wandverfolgung

Weg [cm]	IR-Sensor
0	26
1	240
2	243
3	242
4	234
5	165
6	119
7	88
8	68
9	54
10	44
11	36
12	30
13	25
14	22
15	19
16	16
17	14
18	12
19	11
20	10
21	9
22	8
23	7
24	7
25	6
26	5
27	5
28	4
29	4
30	4
31	4
32	4
33	3
34	3
35	3
36	2
37	2

Tabelle B.4: Sensordaten zur Wandverfolgung

C Quellcode

C.1 Testprogramm Sensorinterface-Linie

```
#include <stdio.h>
#include <regc515c.h>
#include <stub.h>
#include "serielle.h"
static char line[30];
void AksenMain(void)
{
     unsigned long zeit;
     unsigned char in;
     unsigned int i=0,j=0;
     serielle_init();
     led(0,1);
     led(1,1);
     led(2,1);

     do {
        if(digital_in(7)==0){ //nach Betätigung des Tasters ->neuer Datensatz
        i++;
        for( j=0;j<=30;j++)line[j]=' '; //Datensätze durchnummerieren

        line[0]='\n';

        if(i<10){       line[5]= i+48;}
        else if(i<100){

                        line[4]=i/10+48;
                        line[5]=(i%10)+48;
                        }
        else{
                        line[3]=i/100+48;
                        line[4]=((i/10)%10)+48;
                        line[5]=(i%10)+48;
```

```
                    }
        in = analog(5);
        line[9] = in/100+48;
        line[10] = ((in/10)%10)+48;
        line[11] =  (in%10)+48;
        in = analog(6);
        line[13] = in/100+48;
        line[14] = ((in/10)%10)+48;
        line[15] =  (in%10)+48;
        in = analog(7);
        line[17] = in/100+48;
        line[18] = ((in/10)%10)+48;
        line[19] =  (in%10)+48;

        serielle_puts(line); //Ausgabe über serielle Schnittsttelle
        sleep(1000);
        }
      }
     while(1);
}
```

C.2 Testprogramm AKSEN-Encoder-Eingang

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <regc515c.h>
#include <stub.h>

#include "can.h"
#include "serielle.h"
#include "commands.h"
#include "can2CF.h"
#include "protos.h"
#include "mydefs.h"

#define LINELNG 80
static char line[LINELNG];

void AksenMain(void)
```

```
{

  int i,j;
  unsigned int length;
  unsigned long zeit;
  unsigned long deltat;
  unsigned long ges_deltat;
  for (i = 0; i < TEXTBUFLNG; i++) text[i] = 0;
  ts = text;
  te = text;
  ser_init();
  MyAdr =  readAddress();
  CanInit();
  sleep(50);

  lcd_puts("CAN-CompactFlash");
  lcd_setxy(1,0);
  lcd_puts("Encodertest");

  CFopen("protokoll.txt",F_WRITE);
  encoder1();
  deltat=akt_time();
  line[0]='\n';

  while(1){
           if ((akt_time()-deltat)>=100)break;
           }

           zeit=encoder1();
           deltat=akt_time();
           line[1]=zeit/10000 +48;
           line[2]= (zeit%10000)/1000+48;
           line[3]=(zeit%1000)/  100+48;
           line[4]=(zeit%100)/    10+48;
           line[5]=(zeit%10)        +48;

           while(1){if ((akt_time()-deltat)>=100)break;
           }
           zeit=encoder1();
           deltat=akt_time();
           line[7]=zeit/10000 +48;
           line[8]= (zeit%10000)/1000+48;
```

```
line[9]=(zeit%1000)/  100+48;
line[10]=(zeit%100)/    10+48;
line[11]=(zeit%10)        +48;

while(1){if ((akt_time()-deltat)>=100)break;
}
zeit=encoder1();
deltat=akt_time();
line[13]=zeit/10000 +48;
line[14]= (zeit%10000)/1000+48;
line[15]=(zeit%1000)/  100+48;
line[16]=(zeit%100)/    10+48;
line[17]=(zeit%10)        +48;

while(1){if ((akt_time()-deltat)>=100)break;
}
zeit=encoder1();
deltat=akt_time();
line[19]=zeit/10000 +48;
line[20]= (zeit%10000)/1000+48;
line[21]=(zeit%1000)/  100+48;
line[22]=(zeit%100)/    10+48;
line[23]=(zeit%10)        +48;

while(1){if ((akt_time()-deltat)>=100)break;
}
zeit=encoder1();
deltat=akt_time();
line[25]=zeit/10000 +48;
line[26]= (zeit%10000)/1000+48;
line[27]=(zeit%1000)/  100+48;
line[28]=(zeit%100)/    10+48;
line[29]=(zeit%10)        +48;

while(1){if ((akt_time()-deltat)>=100)break;
}
zeit=encoder1();
deltat=akt_time();
line[31]=zeit/10000 +48;
line[32]= (zeit%10000)/1000+48;
line[33]=(zeit%1000)/  100+48;
line[34]=(zeit%100)/    10+48;
```

```
        line[35]=(zeit%10)        +48;

        while(1){if ((akt_time()-deltat)>=100)break;
        }
        zeit=encoder1();
        deltat=akt_time();
        line[37]=zeit/10000 +48;
        line[38]= (zeit%10000)/1000+48;
        line[39]=(zeit%1000)/  100+48;
        line[40]=(zeit%100)/    10+48;
        line[41]=(zeit%10)        +48;

        while(1){if ((akt_time()-deltat)>=100)break;
        }
        zeit=encoder1();
        deltat=akt_time();
        line[43]=zeit/10000 +48;
        line[44]= (zeit%10000)/1000+48;
        line[45]=(zeit%1000)/  100+48;
        line[46]=(zeit%100)/    10+48;
        line[47]=(zeit%10)        +48;

        while(1){if ((akt_time()-deltat)>=100){
                        ges_deltat=akt_time()-deltat;
                        break;
                  }
        }

        CFwrite(line,50,&length);

        printStr(line);

        lcd_cls();
      lcd_puts("Protokoll  ");
        lcd_uint(zeit);
        lcd_setxy(1,0);
        lcd_ulong(ges_deltat);

        sleep(800);

        CFclose();
}
```

```
}
```

C.3 Header arbs.h

```
#include <stdio.h>
#include <regc515c.h>
#include <stub.h>

//arbs = Aksen Roboter Befehls Satz

void init_arbs(void);
void set_adresse(unsigned int * px,unsigned int * py);
void move(int weg);
void turn (int deg);
void halt();
void up_sem(void);
void down_sem(void);


unsigned int get_raddurchmesser();
void set_raddurchmesser(unsigned int durch);
unsigned char get_seganz();
void set_seganz(unsigned int anz);
unsigned int get_radabstand();
void set_radabstand(unsigned int abst);
unsigned char get_turnspeed();
void set_turnspeed(unsigned char tusp);
void set_encoder_links (unsigned char el);
void set_encoder_rechts (unsigned char er);
void set_motor_links (unsigned char ml);
void set_motor_rechts (unsigned char mr);
unsigned char get_bumper1();
void set_bumper1(unsigned int bum1);
unsigned char get_bumperN();
void set_bumperN(unsigned int bumN);
unsigned char get_wand1();
void set_wand1(unsigned int wan1);
unsigned char get_wandN();
void set_wandN(unsigned int wanN);
```

```
void start_follow_line(void);
void stop_follow_line(void);
void start_follow_light(void);
void stop_follow_light(void);
void start_region(unsigned int breite, unsigned int xmax, unsigned int ymax);
void stop_region();
void start_bumper();
void stop_bumper();
void start_avoidcollision();
void stop_avoidcollision();
```

D CD-Inhalt

- Diplomarbeit.pdf
- Poster.pdf
- Handbuch.pdf (AKSEN-Board)
- Verzeichnis „Quellcode“
  - arbs.h
  - main.c
  - init.c
  - move.c
  - follow_line.c
  - avoid_collision.c
  - bumper.c
  - region.c
- Verzeichnis „Quellen“
  - Datenblätter
  - Referenzen
- Verzeichnis „Fotos“