

Dokumentation MarsRiders

Eurobot 2008



Fabian Bürli
Florian Neuhaus
Moritz Kobel
Philipp Haslebacher
Samuel Hubacher
Simon Eggimann
Simon Kunz

Beginn der Arbeit: 20.September 2007
Abgabe der Arbeit: 1. Oktober 2012

angefertigt an der

Berner Fachhochschule
für Technik und Informatik

Abstract

Management Summary

Das Projekt Eurobot 2008 befasst sich an der Berner Fachhochschule mit dem Thema Robotik. Wir als Team von sieben Studenten haben in 8 Monaten einen fahrbaren Roboter entwickelt. Dieser soll die Aufnahmebedingungen des international ausgeschriebenen Eurobot-Wettbewerbs erfüllen. Bei der Entwicklung kann auf Erfahrungen von drei vor-gängigen Teams zurückgegriffen werden. An Budget werden 4000 CHF von der BFH zur Verfügung gestellt. Zudem sind Sponsorenbeiträge von über 2000 CHF vorhanden.

Erstmals setzt unser Team namens **MarsRiders** innovative Technologien, wie eine Java-basierte Hauptsteuerung ein. Das verteilte Rechnen auf mehreren Knoten ermöglicht uns nicht nur eine gezielte Arbeitsteilung, sondern auch ein ausgeklügeltes Sicherheitskonzept.

The project Eurobot 2008 from the Berne University of Applied Sciences (BFH) concerns the issue robotics. Our Team of seven students developed during 8 months a driving roboter. It should conform the terms of admission of the international Eurobot Challenge. We could refer to the experience of three former teams. We could use a budget of 4000 CHF from the BFH. We have also investors who gave us over 2000 CHF.

For the first time, our team called **MarsRiders** used innovative technologies, like a java-based main control. Distributed calculation on multiple nodes allowed us to distribute the work and to elaborate the security system.

Zuerst ignorieren sie dich, dann lachen sie über dich, dann bekämpfen sie dich und dann gewinnst du.

Mahatma Gandhi

Inhaltsverzeichnis

1 Einführung	1
1.1 Einleitung	1
1.2 Überblick	1
1.3 Eurobot 2008	1
1.4 Inhalte	3
1.4.1 Zusammenfassung	3
1.4.2 Änderungen und zusätzliche Kapitel aus der Projektarbeit 2	4
2 Mechanik	5
2.1 Ideenfindung	5
2.1.1 Ideensuche	5
2.2 Konzeptphase	6
2.2.1 Variantenbildung	6
2.2.2 Variantenbewertung	9
2.2.3 Variantenentscheid	11
2.2.4 Aufbau des Roboters	11
2.3 Technische Umsetzung	12
2.3.1 Roboter Kurzbeschreibung	12
2.3.2 Ballaufnahme	14
2.3.3 Speicherung	15
2.3.4 Ballausgabe	15
2.3.5 Antriebskonzept	16
2.3.6 Motorenauslegung	16
2.4 Neuerungen	19
2.4.1 Neue Antriebsauslegung	19
2.4.2 Konstruktion neuer Antriebsräder	23
2.4.3 Startvorrichtung	24
2.4.4 Verschalung	24
2.4.5 Schiebe-Arm	25
3 Sensoren und Aktoren	27
3.1 Sensoren	27
3.1.1 CAN Knoten	27
3.1.2 Erweiterung CAN Knoten	28

3.1.3 Sensorboard	29
3.1.4 Distanzsensor	31
3.1.5 Drehgeber Trommel	33
3.1.6 Endschalter	34
3.1.7 Kompass	34
3.1.8 Gyroskop	35
3.1.9 Farbsensor	36
3.1.10 Software Sensorknoten	38
3.2 Aktoren	46
3.2.1 Hardware	46
3.2.2 Software	51
3.3 Speisungsanschlüsse	62
3.4 Navigationssystem	63
3.4.1 Kommunikation über RS232	63
4 Industrie PC	65
4.1 Hardware	65
4.1.1 Navigation	65
4.2 Betriebssystem & Tools	65
4.2.1 Installation Betriebssystem	65
4.2.2 Anwendungsumgebung Roboterseitig	70
4.2.3 Skripte auf dem Roboter	72
4.2.4 Bedienungsanleitung für den Roboter	74
4.2.5 Konfiguration Entwicklungsumgebung	75
4.2.6 IDE	75
4.3 Java-Software	76
4.3.1 Konzept	76
5 UML Dokumentation	77
5.1 Übersicht / Komponenten	78
5.2 Use-Case	81
5.2.1 Wettbewerb gewinnen	81
5.2.2 Position anfahren	83
5.2.3 Bälle einladen	84
5.2.4 Gegner ausweichen	85
5.2.5 Roboter kontrollieren	86
5.3 Sequenzdiagramme und Beschreibungen	88
5.3.1 CAN-Bus	88
5.3.2 Routing	89
5.3.3 BatchStrategy	92
5.3.4 BatchStrategy.dispenserAction()	93

5.4	Klassendiagramme	96
5.4.1	eurobot	96
5.4.2	eurobot.analyse	97
5.4.3	eurobot.can	98
5.4.4	eurobot.field	101
5.4.5	eurobot.hw	101
5.4.6	eurobot.hw.sensor	102
5.4.7	eurobot.imp	104
5.4.8	eurobot.move	106
5.4.9	eurobot.move.exception	106
5.4.10	eurobot.move.odometrie	107
5.4.11	eurobot.move.place	108
5.4.12	eurobot.play	108
5.4.13	eurobot.remote	109
5.4.14	eurobot.remote.gui	109
5.4.15	eurobot.rs232	112
5.4.16	Hardwareabstraktion	112
5.4.17	Threads	112
5.4.18	Routing	114
6	Abschluss der Arbeit	117
6.1	Schweizermeisterschaft	117
6.2	Weltmeisterschaft	119
6.3	Fazit Projektarbeit 1	121
6.3.1	Arbeitsvorgehen	121
6.3.2	Soll / Ist	122
6.4	Ausblick auf Projektarbeit 2	122
6.4.1	Fertigstellung angefangener Teilprojekte	122
6.4.2	Optimierung	122
6.4.3	Erweiterung	122
6.4.4	Mechanik	123
6.5	Was wir dem nächsten Team weitergeben möchten	123
6.5.1	Hardware	124
6.5.2	Software	125
6.5.3	Wettkampfvorbereitung	125
6.5.4	Wettkampf	126
6.5.5	Sponsoren	126
6.5.6	Medien	126
6.5.7	Diverses	126
6.6	Danksagung	127
6.7	Schlusswort	128

6.8 Selbststndigkeitserklrung	128
A Pflichtenheft	129
A.1 Analyse des technischen Prozesses: Ist-Aufnahme	129
A.1.1 Bestandsaufnahme	129
A.1.2 Aufgabenstellung	130
A.1.3 Anforderungsliste	130
A.2 Ressourcen und Infrastruktur.....	133
A.2.1 Rume, Dozenten und Personal.....	133
A.2.2 Dokumentation und Datenaustausch.....	133
A.2.3 Mechanik	134
A.2.4 Programmierumgebung Atmel	134
A.2.5 Programmierumgebung Java mit Eclipse	134
A.3 Definition der Aufgaben	135
A.3.1 Mechanik	135
A.3.2 Hardware.....	139
A.3.3 Menschliche Eingriffe	142
A.4 Umgebungsbedingungen: Betriebsbedingungen	142
A.4.1 Angaben zum Projektablauf	142
A.4.2 Zeitplan	142
B Anleitungen	145
B.1 AT90CAN128 Debugging on Vista	145
B.2 Bedienungsanleitung Roboter	150
B.3 Anschlussbelegung Servo-Board	153
C Resultate Swisseurobot	155
D Newsletter Team MarsRiders	175
E Software	183
E.1 Dokumentation, Prsentationen und Plakate	183
E.2 Korrespondenz	183
E.3 mechanische Zeichnungen	183
E.4 Schemas.....	183
E.5 Datenbltter	183
E.6 Latex Sourcecode und sonstige Sourcecodes	183
E.6.1 Dokumentation und Sourcecode der Projektarbeit 1	183
E.6.2 Sourcecode Antriebsknoten RTOS	183
E.6.3 Sourcecode Sensorikknoten RTOS	184
E.6.4 Sourcecode Rampenberechnung in Java	184
E.6.5 Sourcecode Java Eurobot	184

E.6.6 Javadoc vom Sourcode Java.....	184
E.6.7 Sourcecode diese Dokumentation	184
E.7 Doxygen Antriebssystem	184

Abbildungsverzeichnis

1.1	Spielfeld	2
2.1	Morphologischer-Kasten	5
2.2	Bewertung	10
2.3	Stärkediagramm	11
2.4	Modelle	11
2.5	Aufbau Roboter	12
2.6	Roboter	13
2.7	Ballaufnahme	14
2.8	Trommelspeicher	15
2.9	Ballausgabe	16
2.10	Kennlinie	18
2.11	Antriebseinbau	19
2.12	ST4118L1804	20
2.13	ST6018M2008	21
2.14	Antrieb	22
2.15	Räder	23
2.16	Entgültige Lösung	23
2.17	Startvorrichtung	24
2.18	Verschalung	24
2.19	fig:Schiebe-Arm unten und oben	25
3.1	Hardwareaufbau	28
3.2	Schema Atmel Erweiterung	29
3.3	Schema Sensorplatine	30
3.4	Sensorplatine	31
3.5	IR-Distanzsensor	32
3.6	GP2D12	33
3.7	Contelec Wal305	33
3.8	Kompassmodul cmp03	34
3.9	Sensor ADXRS300	35
3.10	Prototyp Gyroskop	36
3.11	Schema Farbsensorplatine	37
3.12	grobe Softwarestruktur Sensorik	38
3.13	Softwarestruktur Sensorik	39
3.14	Detail CAN	41
3.15	CAN Nachrichten Sensorik	42
3.16	Detail I2C	42

3.17	Detail IO	43
3.18	Bänder Motor	47
3.19	Trommel Motor	48
3.20	Hitec Servo HS-645MG	48
3.21	Graupner Micro Servo C271	49
3.22	Schema Atmel AddOn-Board	51
3.23	CAN-ID's für Aktorknoten	52
3.24	Struktogramm TaskSpeedControl	56
3.25	Struktogramm motorControl	56
3.26	Beschleunigungsrampe	57
3.27	Datenflussdiagramm	59
3.28	Datenflussdiagramm	60
3.29	Speisungssprint	62
5.1	Komponentendiagramm	80
5.2	Anwendungsfall 1: Wettbewerb gewinnen	82
5.3	Anwendungsfall 2: Position anfahren	84
5.4	Anwendungsfall 3: Ball aufnehmen	85
5.5	Anwendungsfall 4: Gegner ausweichen	86
5.6	Anwendungsfall 5: Roboter kontrollieren	87
5.7	Sequenzdiagramm CAN-Bus	88
5.8	Sequenzdiagramm Routing	91
5.9	Sequenzdiagramm BatchStrategy	93
5.10	Sequenzdiagramm BatchStrategy.dispenserAction()	95
5.11	Analyse GUI gegnerischer Pfad	97
5.12	Klassendiagramm eurobot.can	100
5.13	Klassendiagramm eurobot.hw	102
5.14	Klassendiagramm eurobot.hw.sensor	104
5.15	Klassendiagramm eurobot.remote	111
6.1	Siegerfoto	118
6.2	Grillhilfe	121
A.1	CAD-Zeichnung Roboter	135
A.2	CAD-Zeichnung Ballaufnahme	136
A.3	CAD-Zeichnung Trommelförderer	137
A.4	CAD-Zeichnung Auswurfmechanik	138
A.5	schematischer Aufbau der Hardware	139
A.6	Speisungskonzept	141

Tabellenverzeichnis

1.1	Version Pflichtenheft	3
1.2	Ergänzungen Projektarbeit 2	4
2.1	Bewertung Ballspeicherung	6
2.2	Bewertung Ballaufnahme	7
2.3	Bewertung Ballablage	8
2.4	Varianten	9
3.1	Taskliste	39
3.2	Flagliste	40
3.3	Message Queues	40
3.4	Semaphoren	40
3.5	Kommunikation über RS232	63
3.6	Legende Abkürzungen	63
4.1	MySQL-Datenbank Tabelle LogTable	71
5.1	Übersicht der Komponenten	79
A.1	Ressorts	129
A.2	Anforderungsliste	131

1. Einführung

1.1. Einleitung

Dieses Dokument beschreibt unsere Projektarbeit 1 vom 5. Semester und unsere Projektarbeit 2 vom 6. Semester. Unsere Aufgabe bestand darin einen Roboter zu bauen, mit welchem wir an einem internationalen Wettkampf teilnehmen können. Nach einer Einführung in das Thema welche unter anderem unser Pflichtenheft enthält wird der Entwicklungsweg der mechanischen Komponenten erklärt. Anschliessend werden in einem Kapitel die Entwicklung der Sensoren und Aktoren erläutert. Darauf folgt eine Beschreibung des Industrie-PC's mit dessen Software. Abgeschlossen wird diese Arbeit mit einem rückblickenden Fazit und einem Ausblick auf die weiteren Arbeiten welche uns im nächsten Semester erwarten.

1.2. Überblick

Eurobot ist ein internationaler Wettbewerb für Roboterbegeisterte. Zugelassen sind Gruppen aus Jugendlichen, Studenten oder unabhängige Klubs. Der Eurobot Wettkampf findet stets in Europa statt, ist aber für alle Länder aller Kontinente zugänglich. Diejenigen Länder, welche mehr als drei Teams stellen können, müssen in einer nationalen Ausscheidung drei Teams für die internationalen Wettkämpfe bestimmen. Die Aufgabe, welche den Teams gestellt wird, ist jedes Jahr eine andere. Auf diese Art werden die neuen Teams nicht benachteiligt.

1.3. Eurobot 2008

Dieses Jahr besteht die Aufgabe daraus, nach Bausteine des Lebens auf dem Planeten Mars zu suchen. Die blauen und roten Unihockeybälle stellen Bodenproben dar, welche eingesammelt werden müssen. Für den Rücktransport stehen zwei Systeme zur Verfügung: Das eine besteht aus einem gekühlten Behälter, in den die Bodenproben ohne weiteres geworfen werden können, das andere besteht aus einem gewöhnlichen Behälter, in dem die Bodenproben gemischt mit Eis (durch weisse Bälle dargestellt) gelagert werden müssen.

Jedes Team darf nur einen Roboter einsetzen. An jedem Match, welcher 90 Sekunden dauert, nehmen jeweils zwei Teams teil. Die Bohrkerne aus denen die Bodenproben zu entnehmen sind, werden durch stehende und liegende Ballspender dargestellt. Jeder Roboter hat Bodenproben zu sammeln und diese entweder in den gekühlten Behälter zu werfen oder in den gewöhnlichen Behälter gemischt mit Eis zu legen.

Der gewöhnliche Behälter befindet sich entlang der längeren Tischseite gegenüber dem Startfeld. Jedem Roboter ist eine Hälfte des Behälters zugewiesen. Für jeden Ball der eigenen

1. Einführung

Farbe zwischen zwei weissen Bällen bekommt der Roboter Zusatzpunkte. Jeder Roboter darf die Bälle im gewöhnlichen Behälter bewegen, welche entnehmen oder hinzufügen, um sein Ergebnis zu beeinflussen.

Informationen zu den Reglementen und den einzelnen Wettkämpfen sind unter folgenden Adressen verfügbar:

- Schweizer Ausscheidung: [swisseurobot](http://www.swisseurobot.ch)¹
- Internationale Ausscheidung: [eurobot](http://www.eurobot.org/de/)²

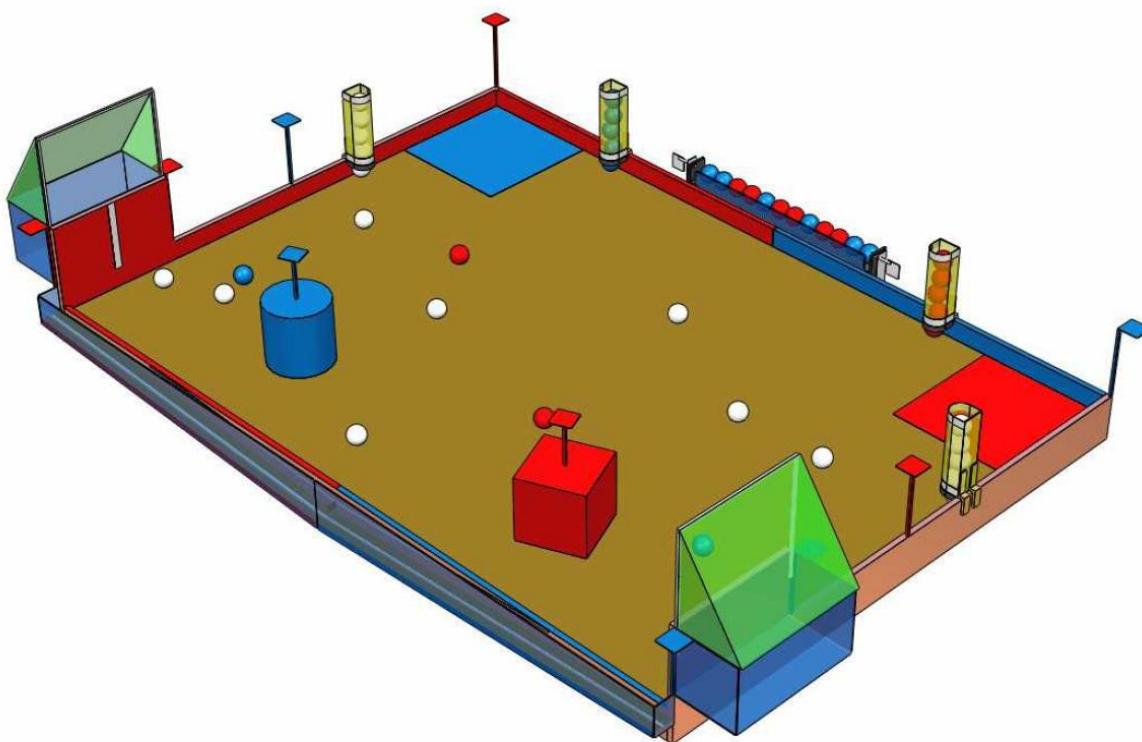


Abbildung 1.1.: Spielfeld

¹<http://www.swisseurobot.ch>

²<http://www.eurobot.org/de/>

1.4. Inhalte

1.4.1. Zusammenfassung

In diesem Dokument sind die Fakten der Konzeptpräsentation aufgeführt. Das Dokument wurde nach Vorlage von Herrn Felsner geordnet und ist mit den nötigen Kapiteln ergänzt worden. Das Pflichtenheft gibt einen Überblick über das Vorgehen unserer Projektarbeit. Da dieses Dokument nicht abschliessend ist, benötigt es eine Angabe der Version.

1.4.1.1. Version

Version	Datum	Ergänzungen
V001	24.Oktober 2007	Prototyp in Word von Simon Kunz
V002	November 2007	Portierung in Latex von Simon Kunz
V003	Dezember 2007	Ergänzungen Projektmanagment
V004	Januar 2008	Änderungen bei Personal, Speisungskonzept und schematischer Aufbau
V005	Mai 2008	Ergänzung Projektarbeit 2

Tabelle 1.1.: Version Pflichtenheft

1.4.2. Änderungen und zusätzliche Kapitel aus der Projektarbeit 2

Kapitel	Beschreibung	Wer?
3.4	neues Kapitel	Kunz
3.2.2	RTOS-Dokumentation	Neuhaus, Kobel und Kunz
E.7	Dxygen RTOS	Neuhaus, Kobel und Kunz
3.2.1	aktualisiert	Eggimann
6.1	neu erstellt	Eggimann
6.2	neu erstellt	Eggimann
5	Projektarbeit OOP	Kobel, Neuhaus
5.4.2	neu erstellt	Kunz
5.4.7	Projektarbeit Bildverarbeitung	Eggimann, Kunz
3.1.3	neu	Haslebacher
3.1.6	neu	Haslebacher
3.1.8	neu	Haslebacher
3.1.9	neu	Haslebacher
3.1.10	aktualisiert	Haslebacher
3.1.2	neu	Haslebacher

Tabelle 1.2.: Ergänzungen Projektarbeit 2

2. Mechanik

2.1. Ideenfindung

Am Anfang der Ideenfindung haben wir entschieden, das Antriebskonzept, die Navigation und die Gegnererkennung des letztjährigen Teams [kju:] zu übernehmen. So konnten wir sehr viel Zeit sparen. Da die Navigation mit integrierter Gegnererkennung letztes Jahr schon sehr zuverlässig funktioniert hat, konnten wir sie als Modul in unseren Roboter einbauen.

2.1.1. Ideensuche

Um geeignete Ideen zu finden hat unsere Gruppe zuerst einen Morphologischen Kasten gemacht, in dem wir in den Sparten Ballaufnahme, Ballspeicherung und Ballabgabe verschiedene Ideen sammelten.

Ballaufnahme: Der Roboter muss Unihockeybälle von den vertikalen Spendern oder auf dem Feld aufnehmen können.

Speicherung: Maximal fünf Bälle dürfen vom Roboter aufgenommen und gespeichert werden. Die Farbe der Bälle muss mittels Sensor erkannt werden.

Ausgabe: Die Bälle müssen bei einem definierten Punkt möglichst schnell und nach Farbe sortiert ausgeladen werden. Ein Ablegen in den gekühlten Behälter (werfen der Bälle) ist in unserer Strategie nicht vorgesehen. Daher wird keine Konstruktion zum Werfen der Bälle benötigt. Unihockeybälle müssen nur im Standard-Behälter abgelegt werden.



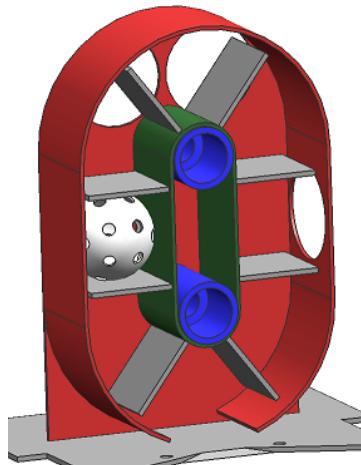
Abbildung 2.1.: Morphologischer-Kasten

2.2. Konzeptphase

2.2.1. Variantenbildung

Wir haben aus dem morphologischen Kasten die für uns in Frage kommenden Aufnahme-, Speicher- und Ablagesysteme ausgewählt. Anschliessend haben wir zu jedem System die Vor- und Nachteile aufgelistet. Wir haben nicht alle Systeme bewertet, da einige nicht realisierbar oder für diese Anwendungen nicht geeignet waren.

Ballspeicherung

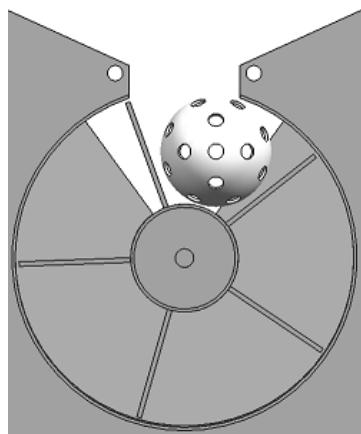


Vorteile:

- + Mehr Platz für Antrieb
- + Kleinerer Umfang des Roboters

Nachteile:

- Weniger Platz für Hardware
- Beförderung der Bälle nach oben bringt kein Vorteil zum Abladen



Vorteile:

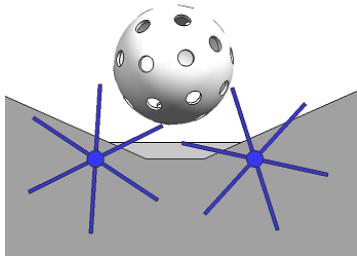
- + Kleiner Platzbedarf
- + Einfache Mechanik
- + Aufwurf einfach realisierbar

Nachteile:

- Benötigt viel Platz im Umfang (Roboterumfang)

Tabelle 2.1.: Bewertung Ballspeicherung

Ballaufnahme

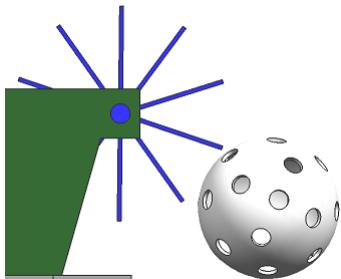


Vorteile:

- + Einfach realisierbar
- + Kleiner Platzbedarf

Nachteile:

- Kleiner Aufnahmebereich (nur Ballbreite)
 - Verklemmen mehrerer Bälle möglich, ohne Fahrbe-
wegung (bei Spender)
-

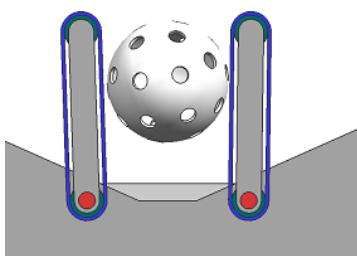


Vorteile:

- + Optimiert für Ballaufnahme auf dem Spielfeld
- + Einfach realisierbar (nur ein Antriebsmotor)

Nachteile:

- Verklemmen mehrerer Bälle möglich (Trichter)
 - Aufnahme von Bällen aus dem Spender nicht mög-
lich
-



Vorteile:

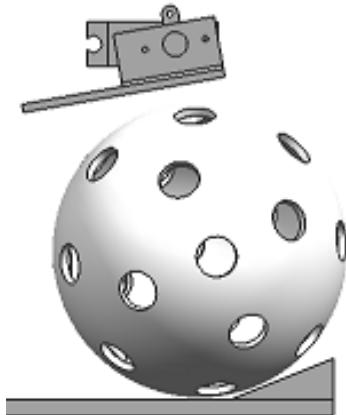
- + Einsammeln von Bällen auf dem Feld und von den
Sendern möglich
- + Aufnahme mehrerer Bälle ohne Fahrbewegung
(Spender)
- + Breiter Aufnahmebereich durch Schwenkarme

Nachteile:

- Mechanisch schwierige Lösung
 - Bei Crash kann viel kaputt gehen
-

Tabelle 2.2.: Bewertung Ballaufnahme

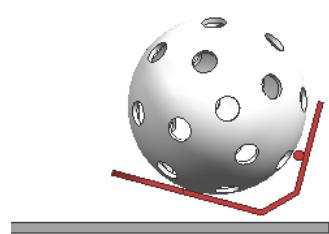
Ballablage

**Vorteile:**

- + Einfache Realisierung mittels Servo und Auswerfer

Nachteile:

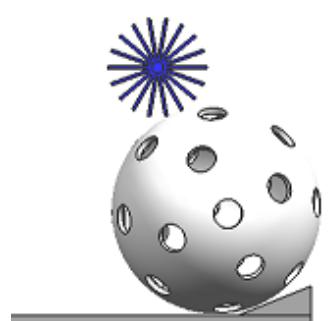
- Gefahr von Verklemmen möglich
- Bälle könnten während der Fahrt herausfallen

**Vorteile:**

- + Einfache Realisierung mittels Servo und Auswerfer
- + Bälle können während der Fahrt nicht herausfallen
- + Bälle liegen immer auf einer Ebene und werden nur beim Ausladen angehoben

Nachteile:

- Die Schwenkbewegung braucht viel Platz

**Vorteile:**

- + Bürste zum Auswerfen könnte kontinuierlich laufen
(Bälle werden schneller ausgeworfen)

Nachteile:

- Das Ablegen der Bälle erfolgt relativ unkontrolliert
(rotierende Bürste)
- Bälle könnten während der Fahrt herausfallen

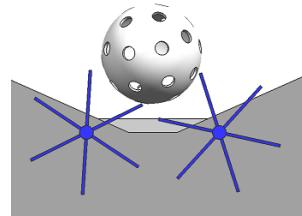
Tabelle 2.3.: Bewertung Ballablage

2.2.2. Variantenbewertung

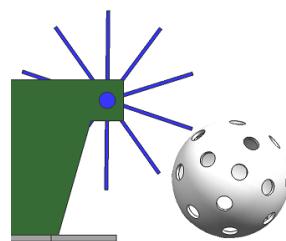
Wir haben anschliessend uns entschieden nur die Ballaufnahme zu bewerten, da für uns bei der Speicherung nur eine Speichertrömmel und bei der Ballablage nur der Schwenkmechanismus in Frage kam. Die anderen Systeme hatten leider zu viele Nachteile gegenüber den von uns gewählten Systemen.

Unsere Varianten:

Variante 1



Variante 2



Variante 3

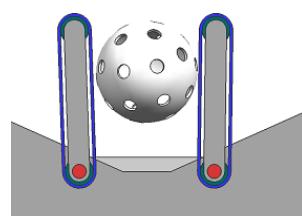


Tabelle 2.4.: Varianten

Diese Varianten haben wir technisch und wirtschaftlich bewertet (siehe Abbildung 2.2 und 2.3).

2. Mechanik

Technische Bewertung					Wertskala nach Richtlinie VDI 2225	
Variante Kriterien	Gewichtung	1	2	3	Pkt.	Bedeutung
Gesammtumfang	5	4	3	3	0	unbefriedigend
Komplexität	3	3	4	3	1	gerade noch tragbar
Spielstrategie	4	2	1	4	2	ausreichend
Funktionssicher	5	2	2	4	3	gut
Flexibilität	2	1	2	2	4	sehr gut (ideal)
Montageaufwand	1	3	3	3		
Herstellbarkeit	3	3	3	2		
Total Punkte	92.00	61	57	73		
$W_t = \text{Tot. P} / 92$		0.66	0.62	0.79		

Wirtschaftliche Bewertung				
Variante Kriterien	Gewichtung	1	2	3
Materialkosten	1	3	3	2
Herstellungskosten / Fertigung	1	2	2	3
Entwicklungs-kosten	2	4	3	2
letztjährige Teile verwendbar	5	3	1	4
Eigenherstellung möglich	4	4	4	4
Total Punkte	52.00	44	32	45
$W_w = \text{Tot. P} / 52$		0.85	0.62	0.87

Abbildung 2.2.: Bewertung

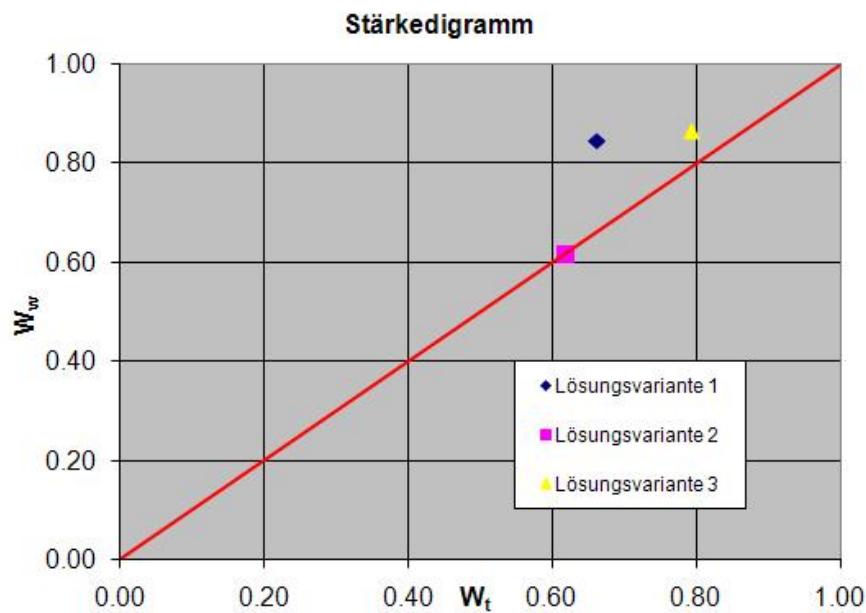


Abbildung 2.3.: Stärkediagramm

2.2.3. Variantenentscheid

Wir haben uns für die Variante 3 entschieden. Danach haben wir einen kleinen Prototypen aus LEGO gebaut, um die Zuverlässigkeit und Funktionssicherheit unserer Lösung herauszufinden. Bei der Aufnahme der Bälle hat sich gezeigt, dass die Variante mit den Förderbändern sehr zuverlässig funktioniert.

Auch für die Speichertrommel und die Ausgabeklappe wurde ein Modell aus Karton erstellt. Daran konnte die Funktionssicherheit der Speichertrommel überprüft werden.

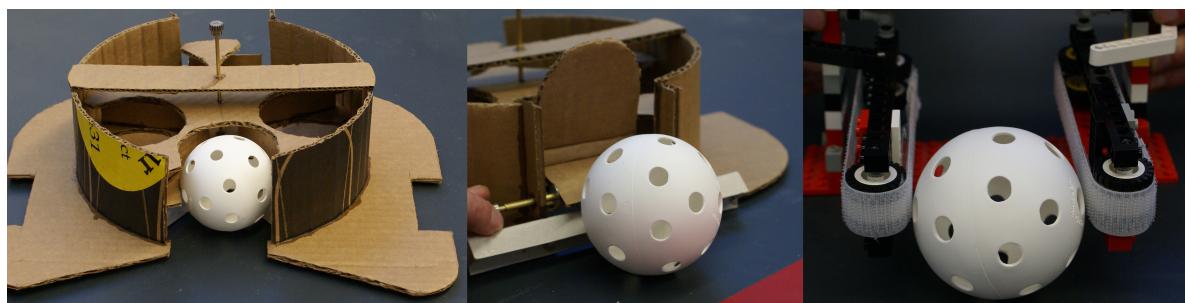


Abbildung 2.4.: Modelle

2.2.4. Aufbau des Roboters

Wir haben uns entschlossen, den ganzen Roboter in einzelne Stockwerke einzuteilen. Das unterste Stockwerk beinhaltet die gesamte Mechanik zur Aufnahme, Speicherung und Ausgabe der Bälle.

Oberhalb dieses Stockwerks werden die Prints mit den Motorentreibern untergebracht. Ebenfalls finden hier auch diverse Sensoren zur Ballerkennung und der Farbsensor ihren Platz. Auf dem nächsten Stock befindet sich die Steuerung unseres Roboters, sie besteht aus dem Industrie-PC, mehreren Atmel-Boards und den Akkus. Um eine optimale Gewichtsverteilung zu erhalten, werden die beiden Bosch-Akkus möglichst im hinteren Teil des Roboters montiert.

Auf der obersten Etage bleibt dann noch Platz für diverse Schalter, Stecker, Notaus und die Startvorrichtung. Der Kompass zur Lagebestimmung unseres Roboters wird ebenfalls auf dieser Ebene eingebaut, somit kann die Störung von den Elektromotoren auf ein Minimum beschränkt werden.

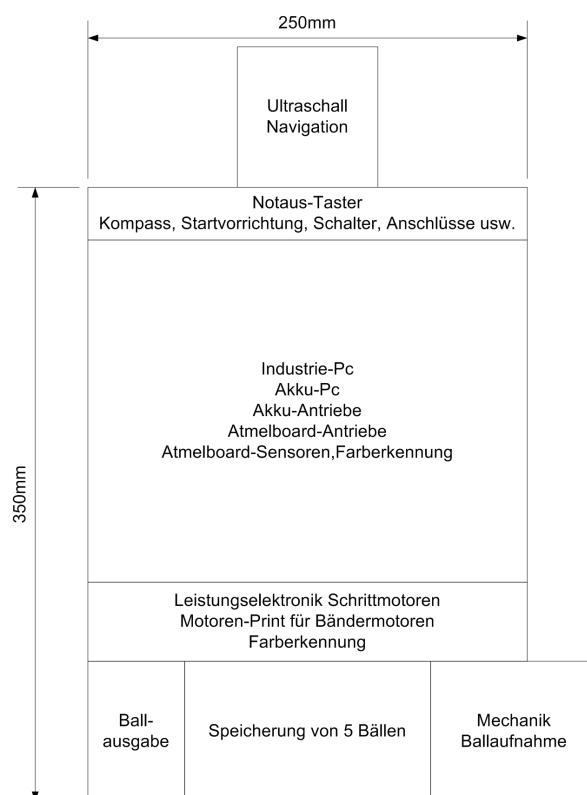


Abbildung 2.5.: Aufbau Roboter

2.3. Technische Umsetzung

2.3.1. Roboter Kurzbeschreibung

2.3.1.1. Mechanischer Aufbau

Die Grundplatte und alle Zwischenplatten sind aus 3 mm dickem Aluminium gelasert worden. Als Abstandshalter werden M5 Abstandsbolzen aus Kunststoff verwendet. Alle Leistungsprints sind mit Abstandsbolzen auf dem jeweiligen Stockwerk befestigt. Die beiden Atmel-Boards sind auf Laborplatten verbaut; so können sie einfach ausgetauscht oder neu

programmiert werden. Für die Akkus sind die beiden Halterungen vom letzten Team verwendet worden, da wir für unsere verbauten Komponenten ebenfalls die Spannungen 24 V und 36 V benötigen. Das Navigationssystem wurde vom letztjährigen Team übernommen.

2.3.1.2. Sensoren/Aktoren

Alle Sensoren des Roboters wurden auf ein Atmel Board geführt. Um die analogen Signale des Potentiometers und der Distanzsensoren auszuwerten, ist auf der Laborplatte ein A/D-Wandler untergebracht. Die Ansteuerung aller Antriebe und der Servos erfolgt über das zweite Atmel Board. Beide Boards werden dann via CAN-Bus auf den Industrie-PC geführt.

2.3.1.3. Software

Für die Hauptsteuerung steht ein Industrie-PC der Firma Digitallogic (Modell MSM855) zur Verfügung. Seine grosszügige Hardwareausstattung und Leistung (Siehe 65) erlaubt uns die Steuerung in Java zu implementieren. Dabei wird die komplette Hardware abstrahiert. Alle notwendigen Daten der Sensoren und Aktoren werden auf den PC per CAN-Bus übertragen. Hier findet auch das Routing und die Spielstrategie statt. Die Steuersoftware kommuniziert zusätzlich über RS232 (Navigation). Die Software bestimmt die auszuführenden Aktionen anhand der aktuellen Sensorwerte und dem Zustand der Strategiefunktionen.

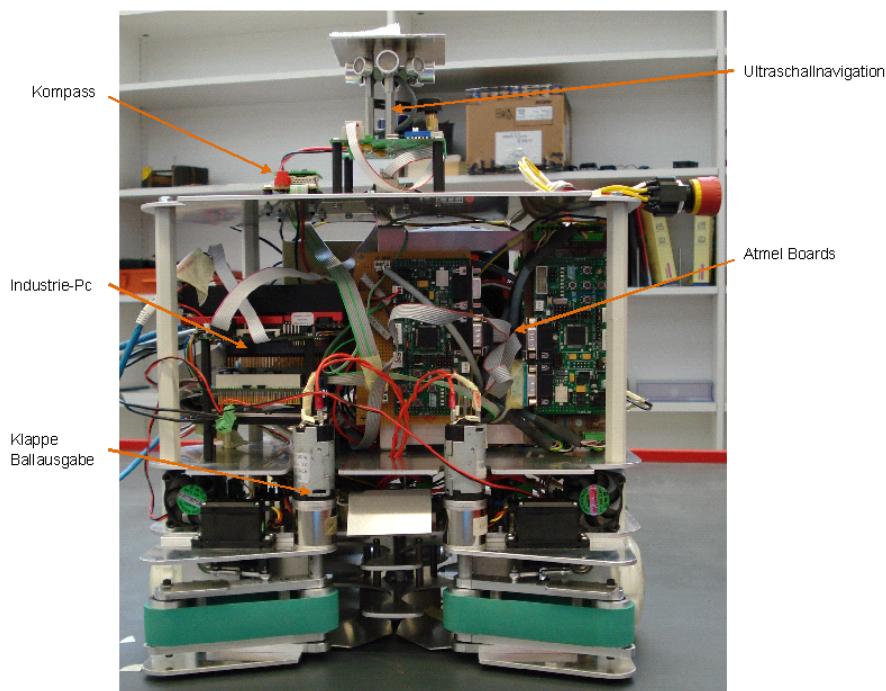


Abbildung 2.6.: Roboter

2.3.2. Ballaufnahme

Die Ballaufnahme ist mittels zwei beweglichen Armen realisiert. Die vordere Lagerung des Armes ist als angestellte Lagerung mittels zwei Rillenkugellagern (Typ 626-2ZR) ausgeführt. Die Lagerung der Antriebswelle erfolgt mit zwei Kugellagern des Typs 624-2ZR. Bei dieser Lagerung wurde das obere Lager fest auf die Antriebswelle gepresst. Nachträglich wurde das untere Lager fest mit dem Arm verklebt, damit sich die Arme axial auf der Antriebswelle nicht mehr verschieben können.

Die Arme werden mit zwei Servos bewegt, hierbei wird die Drehbewegung des Servohorns mit einem Gestänge aus zwei Gelenkkopflagern auf den Arm übertragen. Die Servos besitzen ein Drehmoment von 80 Ncm. Damit kann der Arm maximal einer seitlichen Kraft von 13 N entgegenwirken. Erste Versuche zeigten, dass die Haltekraft nicht sehr gross sein muss um den Unihockeyball in den Roboter zu befördern.

Als Riemen wurden endlos vulkanisierte Vollgummiriemen der Firma Graf endless belts eingesetzt. Die Riemen wurden uns von der Firma gesponsert, zusätzlich haben wir noch zwei Reserveriemen erhalten. Um zu verhindern dass die Riemen von den Scheiben abrutschen können, wurden die Scheiben mit einer Wölbung versehen.

Die Riemen werden mit zwei DC-Motoren betrieben, welche wir vom letztjährigen Team übernommen haben. An den Motoren ist ein Planetengetriebe angeflanscht, um ein höheres Drehmoment zu erhalten.

Damit während dem Fahren keine Bälle aus dem Roboter rollen können, wurde eine Klappe gebaut, welche mit einem Microservo bewegt wird. Die Klappe funktioniert eigentlich sehr gut, ist aber noch nicht sehr stabil gebaut. Das Drehmoment des Microservos ist eher zu schwach. Daher muss diese Konstruktion eventuell im weiteren Verlauf des Projekts noch verbessert werden.

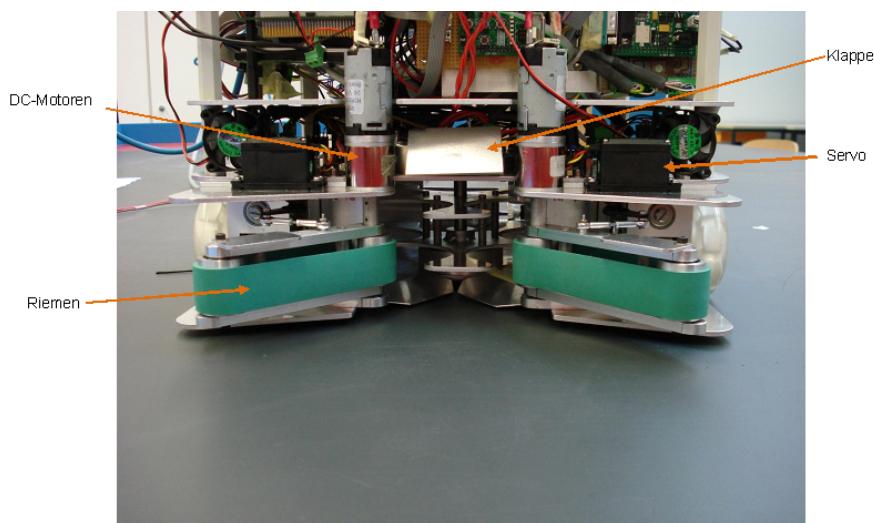


Abbildung 2.7.: Ballaufnahme

2.3.3. Speicherung

Die Speicherung erfolgt mit einem Trommelspeicher, welcher fünf Unihockeybälle aufnehmen kann. Der Speicher wird mit einem Schrittmotor der Firma McLennan angetrieben. Die Trommel ist mit einem Rillenkugellager Typ 608-2ZR gelagert. Unterhalb dieser Lagerung ist ein Potentiometer der Firma Contelec eingebaut, mit welchem die genaue Winkellage der Trommel bestimmt werden kann. Ein Deckel, welcher an der Unterseite festgeschraubt wird, schützt das Potentiometer vor Beschädigung und Verschmutzung.

Um sicherzustellen, dass die Trommel in der richtigen Position steht wenn ein Ball aus dem Roboter befördert werden soll, wird mittels eines Induktiven Sensors immer die Ausgabeposition der Trommel überprüft. Diese Sicherheit haben wir vorgesehen, damit die Ausgabeklappe nicht mit der Trommel verklemmen kann.

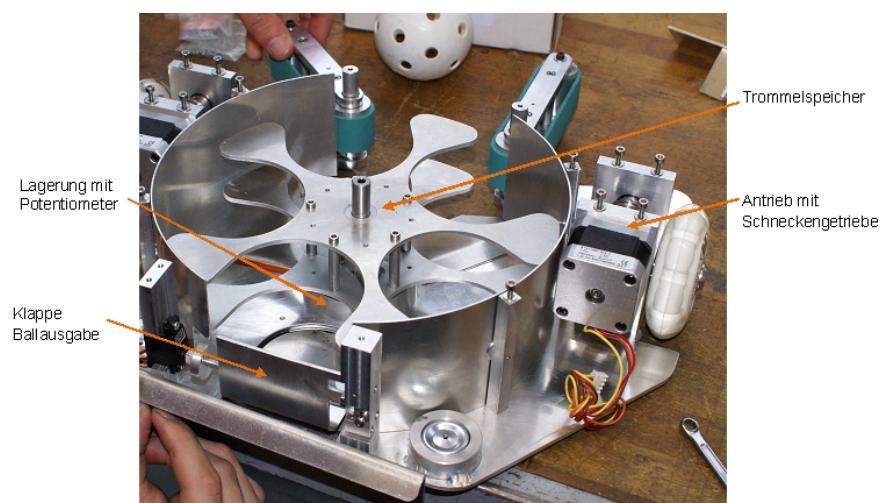


Abbildung 2.8.: Trommelspeicher

2.3.4. Ballausgabe

Die Ballausgabe ist mittels einer Klappe realisiert, welche den Ball nicht nur auswirft, sondern auch gleich auf die richtige Höhe befördert. Die Klappe ist an eine Achse geschweisst, welche auf einer Seite mit einem Lager abgestützt wird. Gegenüber der Lagerung ist ein Microservo montiert, mit welchem die Klappe bewegt wird. Beide Endpositionen der Klappe werden ebenfalls mit induktiven Näherungsschaltern detektiert.

Um den Roboter für die Ballausgabe genau an der Bande zu positionieren, werden Distanzsensoren eingesetzt. Diese sind bereits in der Steuerung integriert, aber müssen noch mit einer Halterung am Roboter befestigt werden.

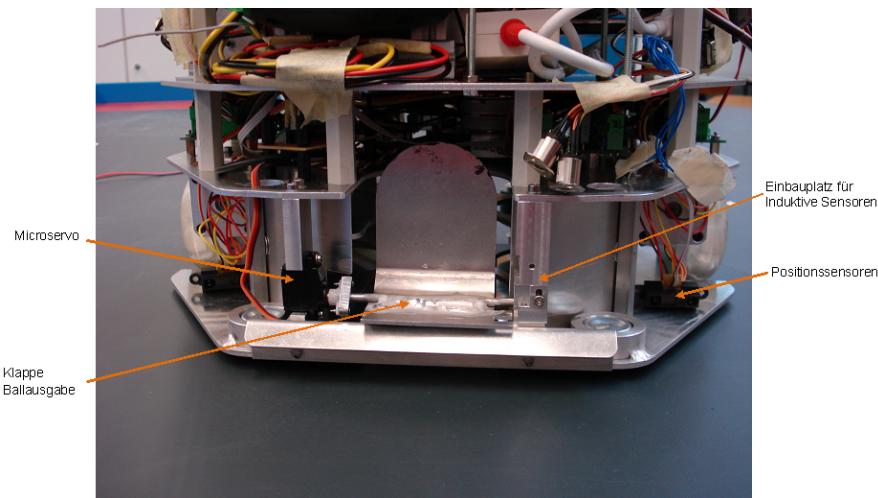


Abbildung 2.9.: Ballausgabe

2.3.5. Antriebskonzept

Wir haben uns entschieden, das Antriebskonzept des letztjährigen Teams [kju:] für unseren Roboter zu übernehmen. Es war uns so möglich, erheblich Zeit zu sparen, da kein neues Antriebskonzept entwickelt werden musste.

Aus den Erfahrungen des Team [kju:] haben wir allerdings gemerkt, dass ihr Antrieb zu schwach ausgelegt war und dadurch bei einem Crash immer wieder aus dem Schritt fiel und der Roboter einfach auf dem Spielfeld stehen blieb. Daher haben wir - nach einigen Abklärungen und Rücksprache mit Herrn Bircher - entschieden, die Motoren neu auszulegen und das Antriebskonzept an diese Motoren anzupassen. Wir sind allerdings noch nicht ganz sicher, ob der Antrieb so funktioniert. Deshalb könnte es noch Änderungen am Antriebskonzept geben.

2.3.6. Motorenauslegung

2.3.6.1. getroffene Annahmen

Für die Auslegung der Motoren haben wir nachfolgende Annahmen über unseren Roboter getroffen:

Robotergeschwindigkeit:

$$v_{Roboter} = 0.5 \text{ m/s}$$

Beschleunigung:

$$a_{Roboter} = 0.5 \text{ m/s}^2$$

Gewicht Roboter:

$$m_{Roboter} = 18 \text{ kg}$$

Reibwert:

$$\mu = 0.1$$

Übersetzungsverhältnis :

$$i_{Getriebe} = 21$$

Getriebewirkungsgrad:

$$\eta_{Getriebe} = 0.55$$

Das Team hat letztes Jahr sehr grosse Räder für ihren Roboter verwendet, um das nötige Drehmoment zu verringern haben wir kleinere Räder gewählt.

Raddurchmesser:

$$d_{Rad} = 0.076m$$

Radumfang:

$$U_{Rad} = d_{Rad} \times \pi = 0.239m$$

2.3.6.2. Berechnung

Aus den getroffenen Annahmen können nachfolgende Kräfte ermittelt werden:
Gewichtskraft Roboter:

$$F_G = m_{Roboter} \times g = 18kg \times 9.81m/s^2 = 176.58N$$

Reibkraft pro Rad:

$$F_R = \mu \times \frac{F_G}{2} = 0,1 \times \frac{176.58N}{2} = 8.83N$$

Beschleunigungskraft:

$$F_B = a_{Roboter} \times \frac{m_{Roboter}}{2} = 0.5m/s^2 \times \frac{18kg}{2} = 4.5N$$

daraus berechnet man die Momente des Rades

Drehmoment Rad:

$$M_{Rad} = (F_R + F_B) \times \frac{d_{Rad}}{2} = (8.83N + 4.5N) \times \frac{0.076}{2} = 0.51Nm$$

Drehzahl Radachse:

$$n_{Radachse} = v_{Roboter} \times U_{Rad} = 0.5m/s \times 0.239m = 2.09s^{-1}$$

so kommt man auf das notwendige Nennmoment des Motors

Nennmoment Motor:

$$M_{Motor} = \frac{M_{Rad}}{i_{Getriebe} \times \eta_{Getriebe}} = \frac{0.51Nm}{21 \times 0.55} = 0.043Nm$$

Drehzahl Motor in s^{-1} :

$$n_{Motor} = n_{Radachse} \times i_{Getriebe} = 2.09s^{-1} \times 21 = 43.98s^{-1}$$

Drehzahl Motor in min^{-1} :

$$n_{Motor} = n_{Motor} \times \frac{60s}{min} = 43.98s^{-1} \times \frac{60s}{min} = 2639min^{-1}$$

2.3.6.3. Motorauswahl

Auf Grund der Berechnungsresultate brauchen wir einen Motor mit einem Nennmoment von 0.043Nm . Da das Team ihre Motoren letztes Jahr bei der Firma Nanotec bezogen hat, entschieden wir uns für den gleichen Lieferanten, damit wir die alte Motorensteuerung weiterhin verwenden können und so keine neuen Steuerungen entwickeln müssen.

Mit dem Evaluationstool auf der Homepage der Firma Nanotec konnten wir einige für uns geeignete Motoren auflisten und haben uns schliesslich für das Modell ST4118M1404 entschieden.

Dieser Motor hat ein Drehmoment von 0.043Nm bei einer Drehzahl von 3000min^{-1} . Das Nennmoment ist damit zwar exakt so viel wie wir gerechnet haben, doch da wir mit einer niedrigeren Drehzahl fahren, wird das Drehmoment höher, wie man in Abbildung 2.10 sieht.



Abbildung 2.10.: Kennlinie

2.3.6.4. Antriebseinbau

Die Motoren mit dem Getriebe sind leicht vor der Mittelachse des Roboters angeordnet, somit mussten nur zwei Kugelrollen im hinteren Bereich angebracht werden. Der Roboter liegt somit auf vier Punkten auf, dies verhindert eine Nickbewegung welche beim Beschleunigen und Bremsen sonst vorkommen kann. Als Abstützung verwenden wir Kugelrollen der Firma FTA mit einer Nylonkugel (siehe 2.11).

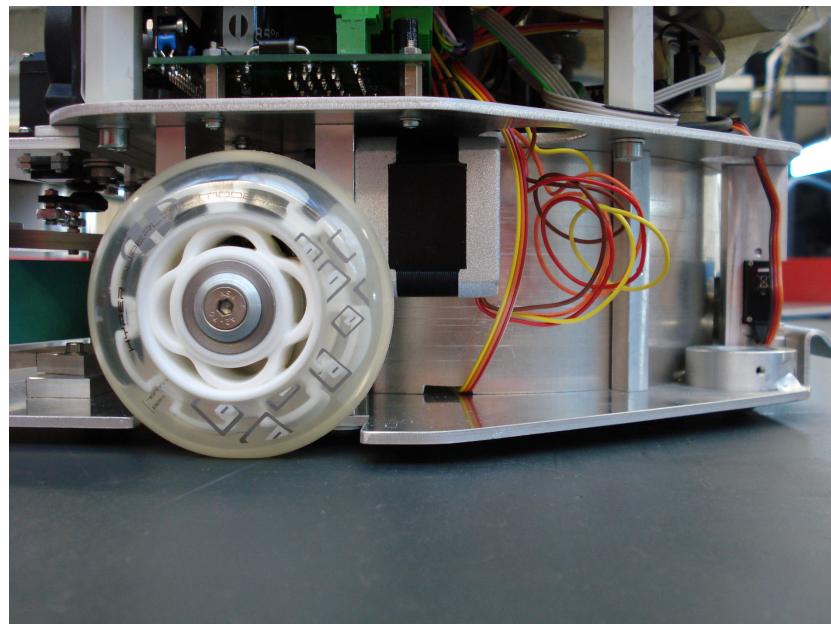


Abbildung 2.11.: Antriebseinbau

2.4. Neuerungen

2.4.1. Neue Antriebsauslegung

Mit den bestehenden Motoren und Schneckengetriebe konnten wir bis jetzt maximal mit 0.35 m/s fahren. Mit dieser Geschwindigkeit ist es uns nur möglich einmal die vertikalen Dispenser anzufahren und die aufgenommenen Bälle abzuladen. Unser Ziel ist es aber die Spender zweimal anzufahren, damit wir insgesamt 10 Bälle im Standard-Behälter ablegen können.

Bei der Auswahl eines neuen Antriebs galten folgende Vorgaben:

- Ansteuerelektronik muss weiter verwendet werden können
- Antriebsachse muss an gleicher Position bleiben wegen Drehmittelpunkt des Roboters

Um einen schnelleren Antrieb zu realisieren wurden zwei mögliche Varianten untersucht:

2.4.1.1. Variante 1

Verwendung eines stärkeren Schrittmotors der gleichen Baureihe und einer tieferen Übersetzung des Schneckengetriebes.

Abbildung 2.12 zeigt die Kennlinie des stärksten Schrittmotors der 42-er Baureihe von nanotec. Aus dem Diagramm ist ersichtlich dass bei höherer Drehzahl das Moment stark abnimmt. Wenn wir nun die bestehende Getriebeübersetzung verändern, sinkt zwar die Drehzahl, aber das benötigte Moment steigt an. Nach einer ersten Abschätzung zeigte sich dass wir mit dieser Variante keine befriedigende Lösung realisieren können.

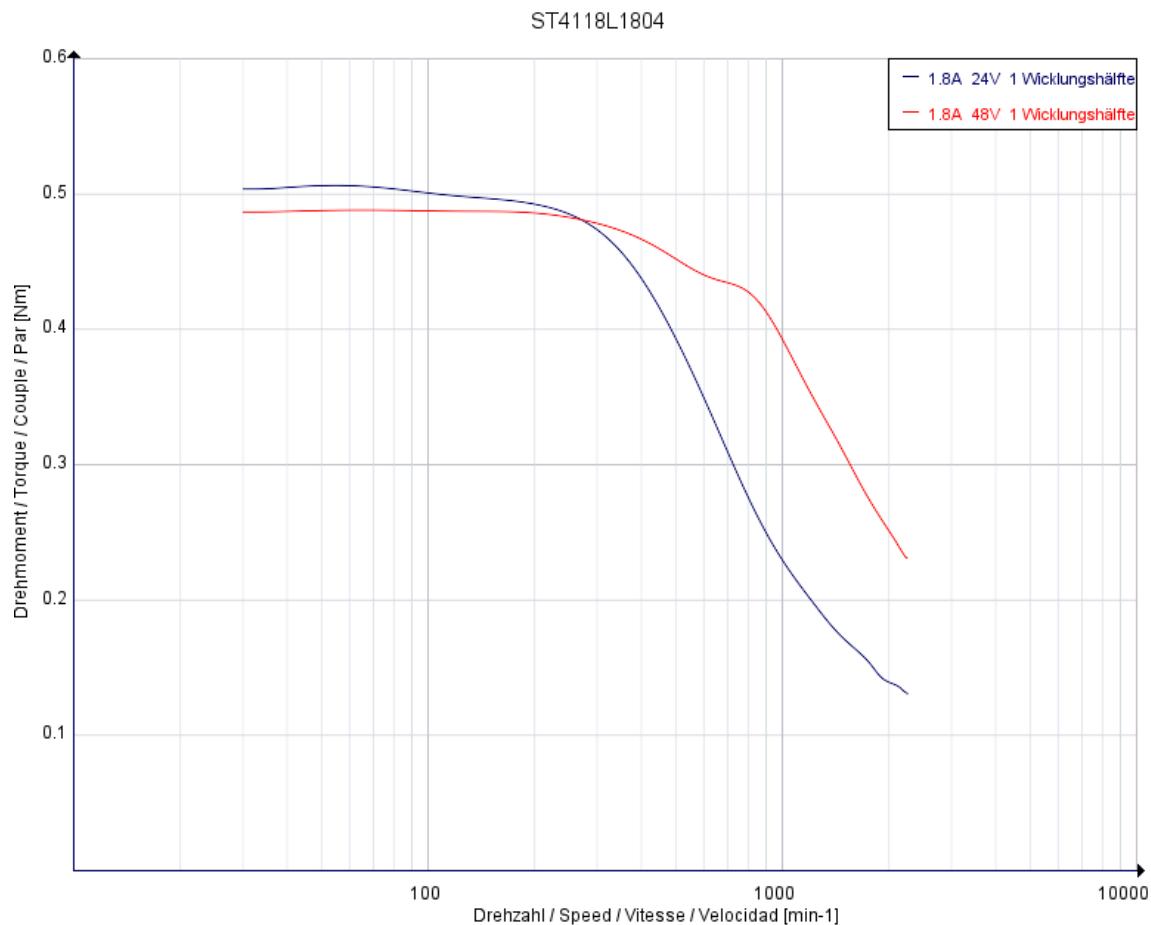


Abbildung 2.12.: ST4118L1804

2.4.1.2. Variante 2

Direktantrieb mit grösseren Schrittmotoren.

Bei dieser Variante wird der Roboter direkt mit zwei grossen Schrittmotoren angetrieben. Durch diese Variante entfällt der schlechte Wirkungsgrad des Schneckengetriebes, zudem können wir den Motor im unteren Drehzahlbereich betrieben wo er noch ein hohes Drehmoment besitzt.

Abbildung 2.13 zeigt die Kennlinie des gewählten Schrittmotors aus der 60-er Baureihe von nanotec. Wir werden den Motor Seriell betreiben und mit einer Spannung von 36 Volt versorgen. Auf diese Weise zieht ein Motor nur 1.4 A Strom, was kein Problem für den Motorentreiber darstellt.

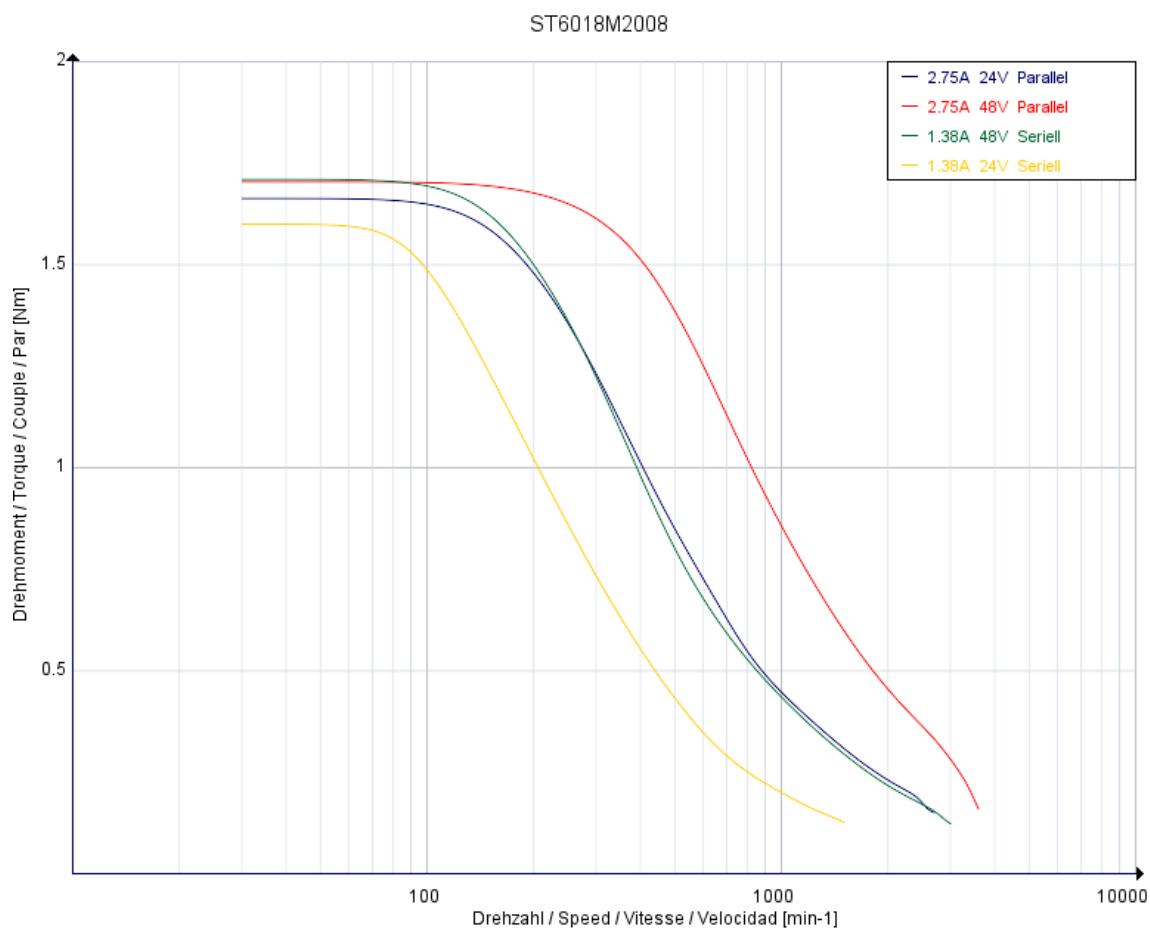


Abbildung 2.13.: ST6018M2008

2.4.1.3. Rechnerische Auslegung des neuen Antriebs

Robotergeschwindigkeit:

$$v_{Roboter} = 1 \frac{m}{s}$$

Beschleunigung:

$$a_{Roboter} = 2 \frac{m}{s^2}$$

Gewicht Roboter:

$$m_{Roboter} = 13 kg$$

Reibwert:

$$\mu = 0.1$$

Raddurchmesser:

$$d_{Rad} = 0.090 m$$

Motorenträgheitsmoment:

$$J_{Motor} = 0.000045 kg * m^2$$

Gewichtskraft eines Antriebsrades:

$$F_G = \frac{m_{Roboter} \times g}{2} = \frac{13kg \times 9.81 \frac{m}{s^2}}{2} = 63.765N$$

Reibkraft eines Antriebsrades:

$$F_R = F_G \times \mu = 63.743N \times 0.1 = 6.377N$$

Beschleunigungskraft:

$$F_B = a_{Roboter} \times \frac{m_{Roboter}}{2} = 2 \frac{m}{s^2} \times \frac{13kg}{2} = 13N$$

Trägheitsmoment:

$$M_T = J_M \times \frac{a_{Roboter}}{\frac{d_{Rad}}{2}} = 0.000045kg * cm^2 \times \frac{2 \frac{m}{s^2}}{\frac{0.090m}{2}} = 0.002Nm$$

Drehmoment Rad:

$$M_{Rad} = M_T + (F_R + F_B) \times \frac{d_{Rad}}{2} = 0.002Nm + (6.377N + 13N) \times \frac{0.090m}{2} = 0.874Nm$$

Drehzahl Rad:

$$n_{Rad} = \frac{v_{Roboter}}{\pi \times d_{Rad}} = \frac{1 \frac{m}{s}}{\pi \times 0.090m} = 5.537 \frac{1}{s} = 212.207 \frac{1}{min}$$

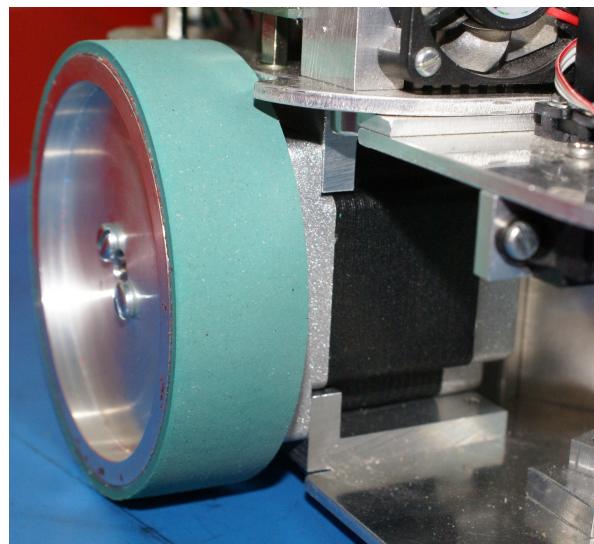


Abbildung 2.14.: Antrieb

Bei ersten Tests mit den Motoren stellten Wir fest dass im unteren Drehzahlbereich (ca. 100 Hz) grosse Resonanzen auftreten. Um diese Erscheinungen zu beseitigen betreiben wir die Motoren nun im Halbschritt (Auflösung pro Umdrehung 400 Schritte).

2.4.2. Konstruktion neuer Antriebsräder

Mit dem neuen Antrieb benötigten wir jetzt auch grössere Antriebsräder. Zuerst verwendeten wir die Inline-Rollen des Teams [Kju] von letztem Jahr. Bei Testfahrten fiel uns aber auf dass die Räder beim Anfahren oder Abbremsen immer wieder durchdrehten. Als Verbesserungsmassnahme wurden grössere Inline-Rollen auf einen Durchmesser von 90mm abgedreht. Dadurch hatten die Räder eine grössere Auflagefläche und sollten dadurch mehr Haftung entwickeln.



Abbildung 2.15.: Räder

Der Erfolg war aber nicht sehr befriedigend, also musste nach einer neuen Lösung gesucht werden.

Da wir für unsere Ballaufnahme elastische Gummiringe verwenden und diese eine sehr gute Haftung besitzen, kam uns die Idee Zwei Räder zu fertigen auf welche wir diese aufziehen können. Erste Tests zeigten dass wir die Riemen zusätzlich festkleben müssen weil die seitlichen Querkräfte bei den Drehbewegungen die Riemen sonst von den Rädern schieben.

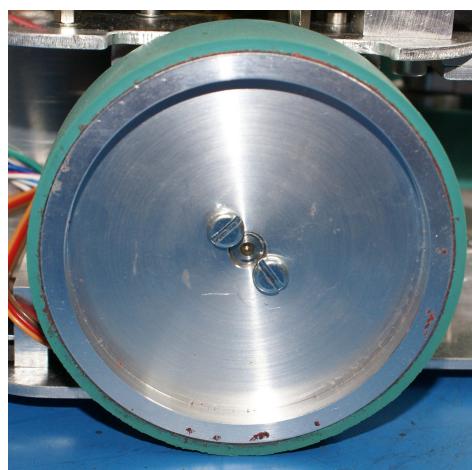


Abbildung 2.16.: Entgültige Lösung

2.4.3. Startvorrichtung

Von der Eurobot Vereinigung ist vorgeschrieben dass der Roboter mittels einer Startschnur gestartet werden muss. Da das System des Teams [kju] ziemlich unzuverlässig aussah, entschlossen wir uns eine bessere Version zu bauen. Die neue Startvorrichtung ist mittels eines induktiven Sensors aufgebaut. Dieser Sensor erkennt die Anwesenheit eines Metallstiftes aus Stahl, welcher beim Start aus der Vorrichtung gezogen wird.

Ein Nachteil bietet das System. Es ist eine Versorgungsspannung des Sensors nötig, welche bei einem einfachen Schalter entfällt.

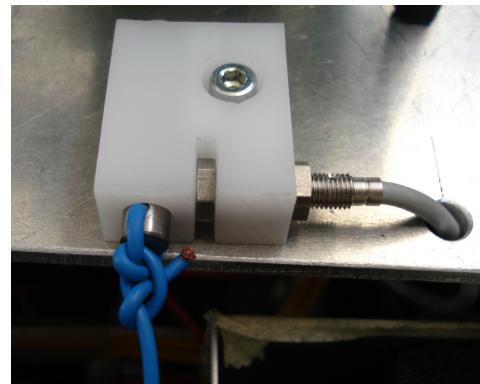


Abbildung 2.17.: Startvorrichtung

2.4.4. Verschalung

Unser Roboter hatte noch keine Verschalung und da unsere Elektronik zum Teil sehr weit aussen angebracht ist, war es uns zu gefährlich, das Turnier ohne eine Verschalung zu bestreiten.

Unsere erste Idee war ein Gehäuse aus durchsichtigem Plexiglas. Unser Roboter hat jedoch eine sehr komplizierte Aussenform und so wäre die Herstellung sehr anspruchsvoll und zeitaufwendig gewesen. Daher entschlossen wir uns das Gehäuse aus Aluminium zu fertigen und es anschliessend schwarz zu lackieren.



Abbildung 2.18.: Verschalung

2.4.5. Schiebe-Arm

Kurz vor dem Wettkampf in Rapperswil hatten wir noch die Idee einen beweglichen Arm an unseren Roboter zu bauen, um damit Bälle im Standartbehälter verschieben zu können. Mit dieser Taktik können wir dem Gegener Punkte klauen, indem wir seine Bälle auf unsere Seite des Standartbehälters schieben.

Der Arm besteht aus 4mm Plexiglas und ist direkt mit einem Servo verbunden. Bei ersten Test ging das Getriebe des Servos regelmässig kaputt, da es zu grosse Momente aufnehmen musste. Um das Servo bei ausgeklapptem Arm zu entlasten, wurde eine zusätzliche Stütze eingebaut, welche einrastet, wenn der Arm seine Endposition erreicht hat. Diese Stütze nimmt die anfallenden Kräfte beim Verschieben der Bälle auf, damit keine Gegenmomente mehr auf das Servo wirken.

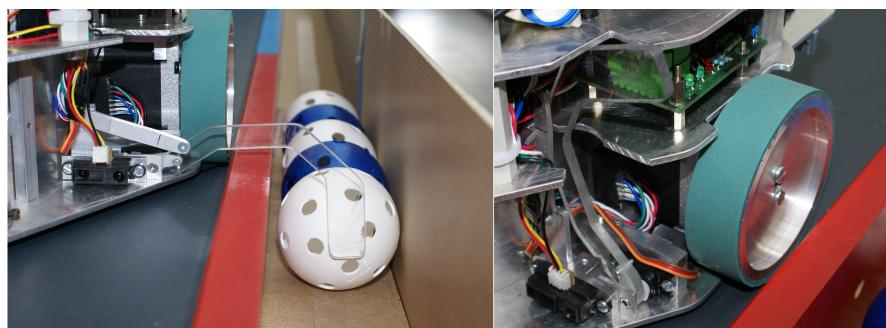


Abbildung 2.19.: fig:Schiebe-Arm unten und oben

3. Sensoren und Aktoren

3.1. Sensoren

Der Sensorik wurden folgende Aufgaben zugewiesen:

- Starten mittels Startschnur
- Fahrtrichtungsbestimmung des Roboters
- Drehposition der Speichertrommel ermitteln
- Farberkennung des aufgenommenen Balls
- Ballerkennung beim Einziehen
- Endlagen der Auswurfklappe erkennen
- Distanzmessung zur Bande beim Ausladen

Während der Projektarbeit 1 wurde die Richtungsbestimmung, die Startschnur, die Drehposition und die Distanzmessung implementiert. Die Farberkennung ist für die momentane Spielstrategie nicht erforderlich und die Endschalter sind rein zur Sicherheit der Servos. Für die ersten Tests sind diese nicht erforderlich und werden aus Zeitgründen erst in der Projektarbeit 2 realisiert.

3.1.1. CAN Knoten

Als Schnittstelle zwischen dem Industrie-PC und den Sensoren kommt ein Entwicklungsboard DVK90CAN1 der Herstellers Atmel zum Einsatz.

Wir entschieden uns dafür aus folgenden Gründen:

- kompakte Abmessungen (5cm x 12cm)
- umfangreiche Peripherie vorhanden (Tasten, LED, Treiberbausteine, ..)
- viele nützliche Schnittstellen (I2C, RS232, CAN, SPI, JTAG)
- vorhandenes, bereits portiertes Echtzeitbetriebssystem uC-OS II
- Debugging mit JTAG ICE mk2 möglich

3. Sensoren und Aktoren

Da die meisten Ports des Controllers auf Buchsenleisten geführt sind, ist es zudem einfach mit einer Erweiterungsplatine das Board für unsere Anwendung anzupassen.

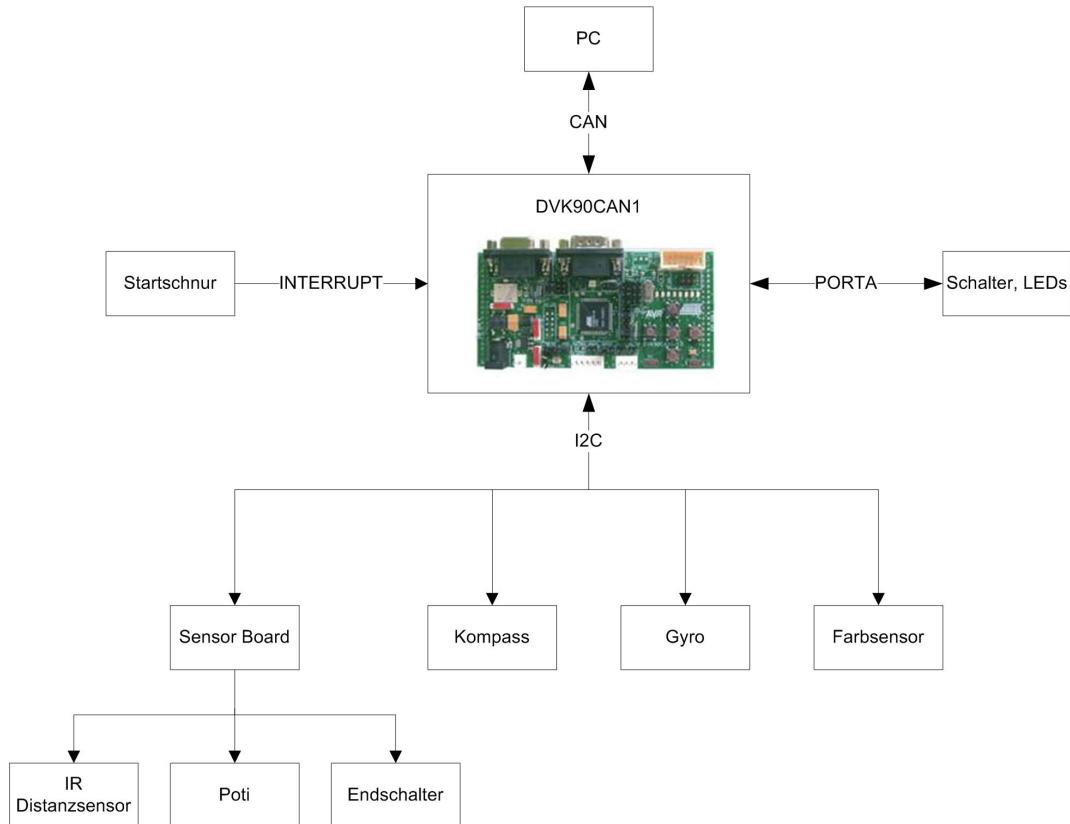


Abbildung 3.1.: Hardwareaufbau

3.1.2. Erweiterung CAN Knoten

Um den Startvorgang komfortabler zu gestalten wurden gegen Ende der Arbeit noch verschiedene Schalter und Taster eingebaut.

- Taster Reset, setzt beide CAN Knoten zurück
- Taster Init, CAN-Nachricht wird an PC gesendet, Programmneustart
- Taster Eject, CAN-Nachricht wird an PC gesendet, Roboter wirft Bälle aus
- Farbwahlschalter, Status kann vom PC abgefragt werden
- neue Startschnur, vom einfachen Jumper zum edlen Stahlbolzen mit induktivem Sensor

Die Startschnur ist auf einen Interrupteingang (INT2) geführt. Leider sind die anderen externen Interrupts des Controllers nicht auf die Kontaktleisten geführt. Deshalb sind die

weniger wichtigen Tasten auf einen normalen IO-Port geführt und werden in einem Task mit niedriger Priorität periodisch abgefragt.

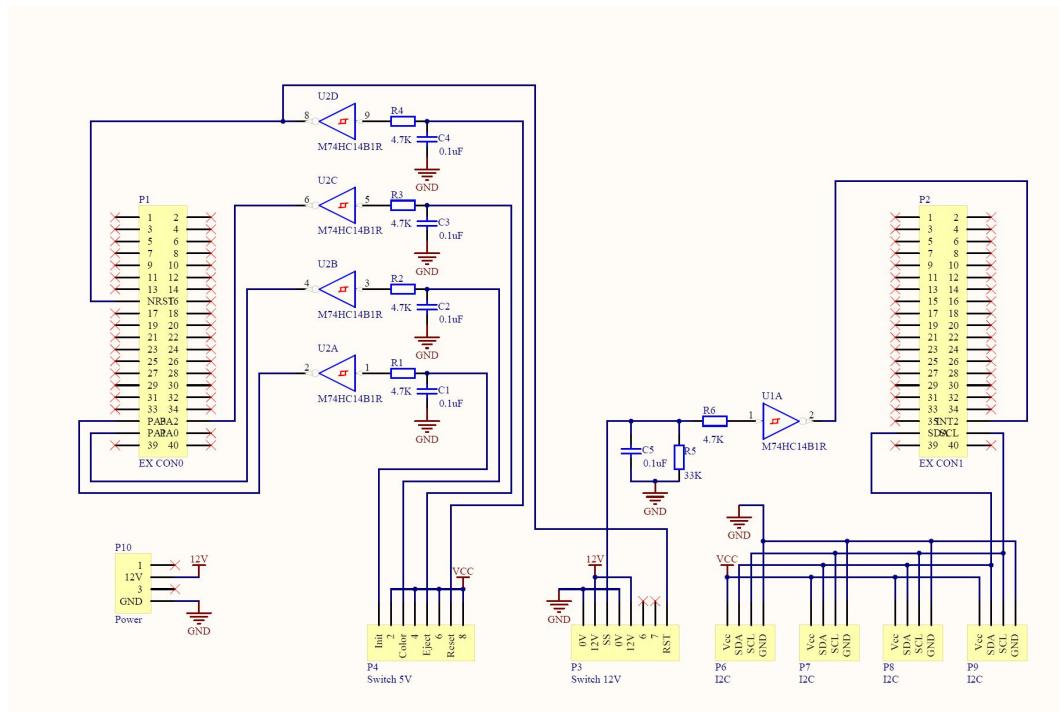


Abbildung 3.2.: Schema Atmel Erweiterung

3.1.3. Sensorboard

Mit dieser Erweiterungsplatine wurden die Sensoren der unteren Ebene des Roboters für eine einfachere Verdrahtung zusammengefasst.

Sensoren:

- 1 x Induktiv für Auswurfklappe
- 1 x Induktiv für Trommelposition
- 1 x Lichtreflex für Ballaufnahme
- 4 x IR Distanzsensor für Bandenabstand
- 1 x Endlospotentiometer für Trommelposition

Sämtliche Sensorwerte können über den I2C-Bus abgefragt werden.

Die drei Endschalter werden mit einem Spannungsteiler auf 5V angepasst, mit einem einfachen RC-Glied und mit einem Schmitt-Trigger (74HC14) entprellt und auf einen I2C IO-Expander (PCF8574) geführt.

Alle IR-Distanzsensoren (GP2D12 / GP2D120) werden mit 5V gespiesen und liefern eine Analoge Spannung entsprechend der gemessenen Distanz. Diese Spannung wird direkt auf einen I2C AD-Wandler (PCF8591) geführt. Die Referenzspannung des Wandlers liegt auf

3. Sensoren und Aktoren

5V.

Das Potentiometer wird mit 12V und einem Vorwiderstand betrieben. Dieses Signal ist ebenfalls auf einen I2C AD-Wandler geführt. Die höhere Spannung wurde gewählt, weil für die Montage unter der Trommel kein Abgeschirmtes Kabel verlegt werden konnte und die Kabelführung direkt neben dem Motor durchläuft. Damit sollten Störungen verringert werden.

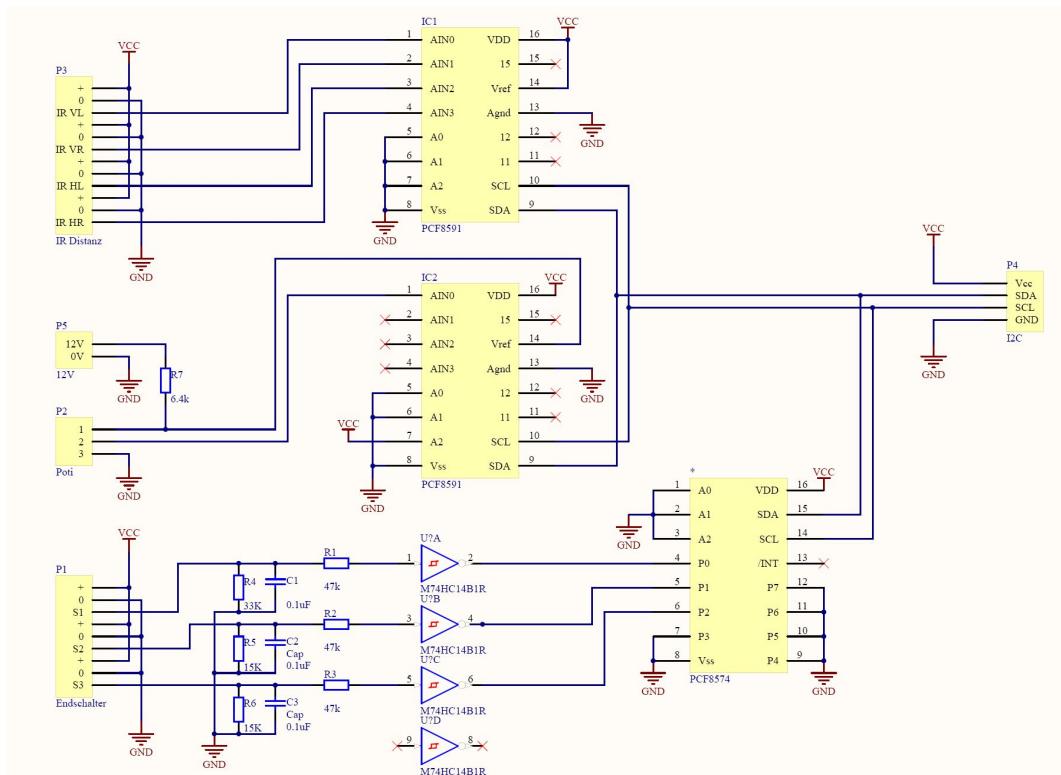


Abbildung 3.3.: Schema Sensorplatine

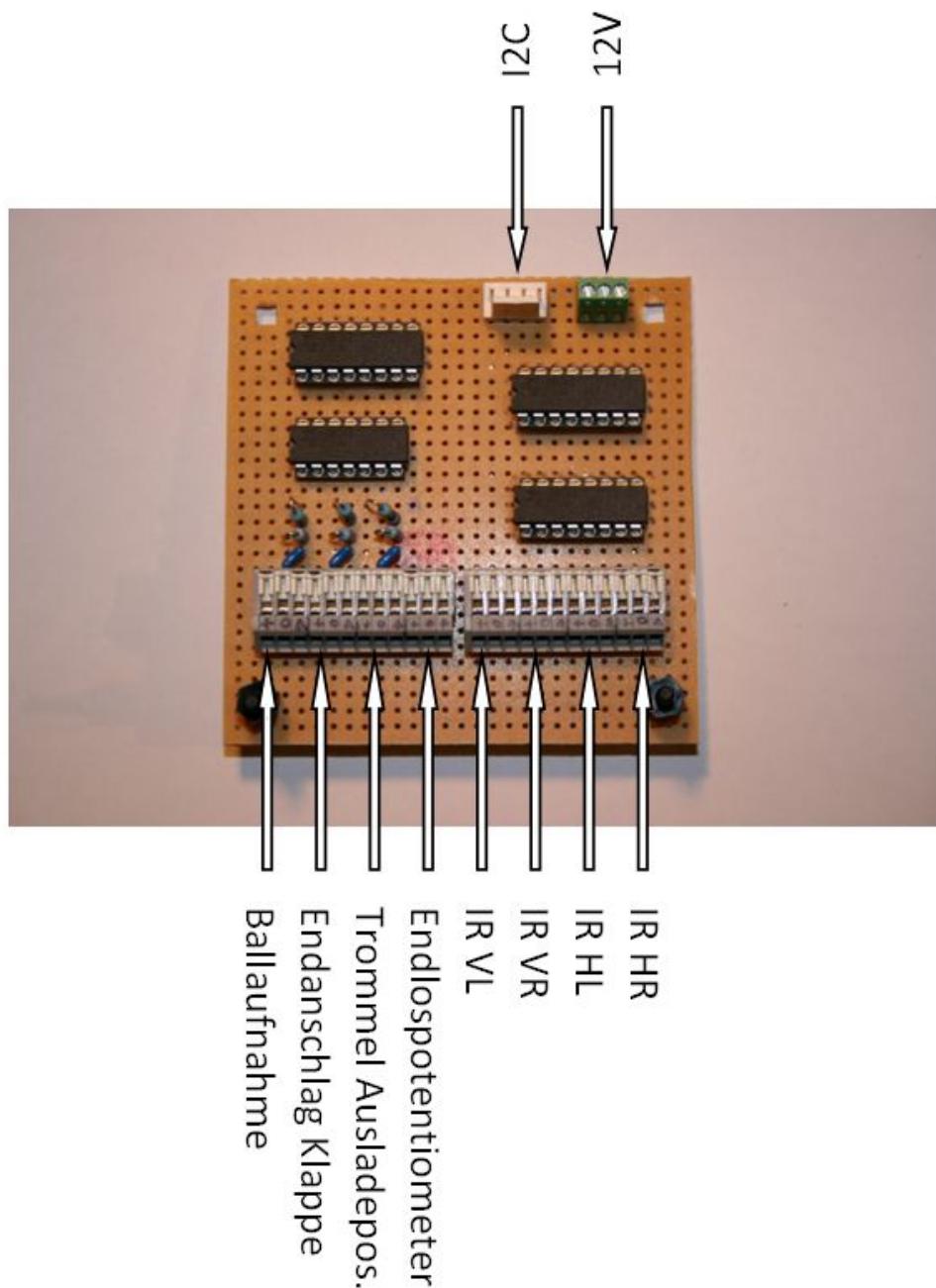


Abbildung 3.4.: Sensorplatine

3.1.4. Distanzsensor

Die Distanzmessung vom Roboter zur Bande benötigen wir als Kollisionsschutz beim Anfahren der Ausladeposition. Die Distanz wird nur gemessen, wenn der Roboter kurz vor der Bande ist. Damit entfallen Störeinflüsse vom gegnerischen Roboter oder Bälle, die im Weg stehen.

Mögliche Distanzmessverfahren wären Laser, IR oder Ultraschall.

3. Sensoren und Aktoren

Lasermessung ist in Anbetracht des begrenzten Raum- und Finanzbudgets ungeeignet.

Ein Ultraschallmodul könnte im Zusammenspiel mit unserem Navigationssystem zu Problemen führen und wurde deshalb nicht weiter evaluiert.

IR- Distanzsensoren sind günstig in der Anschaffung, klein in der Bauform und einfach in der Ansteuerung. Einzig die Präzision ist schlechter als Ultraschall und Laser, reicht für unsere Aufgabe aber voll aus.



Abbildung 3.5.: IR-Distanzsensor

Der gewählte Sensor Sharp GP2D12 wird mit 5V gespiesen und liefert eine Analogspannung gemäss der Kennlinie in Abbildung 3.6. Sie ist nicht ausgesprochen linear und hat zudem bei 8cm einen Knick. Weil wir die Sensoren aber 8cm hinter der Auswurftasse montiert haben und das Signal nur in der Endanfahrt auf die Bande auswerten, benötigen wir nur den Ausschnitt von 8 bis 20 cm.

Da wir links und rechts einen Sensor montiert haben, können wir zudem ermitteln, ob der Roboter parallel zur Bande steht und so einen fehlerhaften Auswurf verhindern.

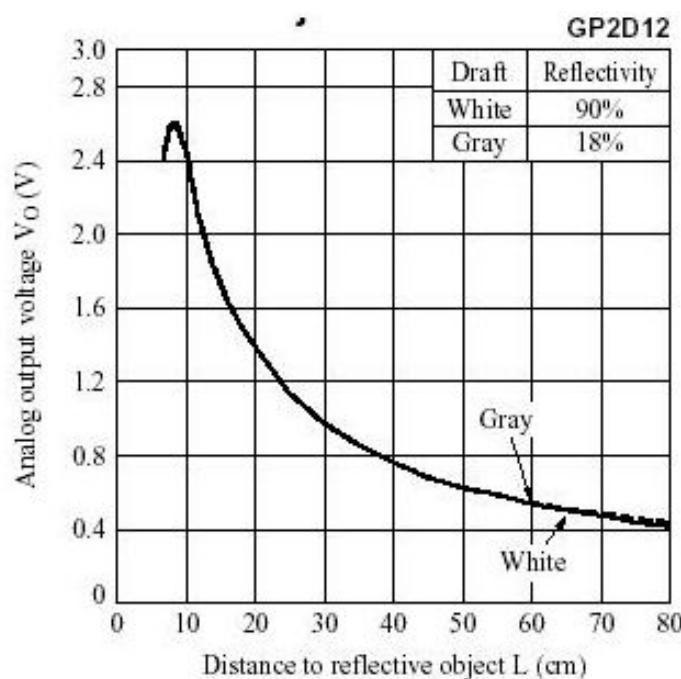


Abbildung 3.6.: GP2D12

3.1.5. Drehgeber Trommel

Die Ballspeichertrommel wird mit einem Schrittmotor angetrieben. Damit kann um jeden beliebigen Winkel gedreht werden. Wenn jedoch ein Ball kurz eingeklemmt wird, kann der Motor aus dem Schritt fallen. Dies hätte eine falsche Drehposition zur Folge, die mögliche Beschädigungen der Mechanik nach sich ziehen kann.



Abbildung 3.7.: Contelec Wal305

Aus diesem Grund wird die absolute Position der Trommel mit einem Endlospotentiometer bestimmt. Dieses konnte dank der flachen Bauweise unter der Trommel montiert werden. Die Auswertung der Analogspannung geschieht mit dem Wandler, der auch die Distanzwerte aufnimmt.

3.1.6. Endschalter

Wir setzen Endschalter ein zur

- Detektion von Bällen im Einzug
- Kontrolle der Auswurfklappe
- Überprüfung der Trommel auf Ausladeposition
- Startschnur

Bei der Aufnahme setzen wir für eine berührungslose Erkennung der Bälle einen Lichtreflektaster Baumer FHDK-10 ein. Die Schaltdistanz kann an der Stellschraube am Gehäuse in einem Bereich von 20 bis 120 mm eingestellt werden.

Die Trommelposition, die Klappenendlage und das Startsignal werden jeweils mit einem Induktiven Näherungsschalter erfasst. Die schlanken M4 Sensoren von Contrinex konnten auch an schlecht zugänglichen Orten montiert werden. Mit einem Schaltabstand von nur einem Millimeter stellen sie dafür höhere Präzisionsanforderungen an die Mechanik.

Sämtliche Endschalter werden mit 12V betrieben und werden auf der Sensorplatine ausgewertet.

3.1.7. Kompass

Die Drehrichtung des Roboters ist lebensnotwendig für unsere Navigation. Diese messen wir mit dem digitalen Kompassmodul cmp03.



Abbildung 3.8.: Kompassmodul cmp03

Der Messwert wird auf Grund der horizontalen Komponente des vorhandenen Erdmagnetfeldes mit zwei Magnetfeldsensoren KMZ51 bestimmt. Die Messwerte werden mit einem PIC-Mikrocontroller in einen Winkelwert umgewandelt. Es stehen zwei Genauigkeitsstufen zur Verfügung: 256 oder 3600 Werte pro 360°. Für Manöver innerhalb der Tischdimensionen reicht uns der Bereich von 0 bis 255. Durch eine komfortable Kalibrierungsfunktion besteht zudem die Möglichkeit das Modul an räumliche Besonderheiten und vorhandene statische Felder anzupassen. Zum Schutz vor den Magnetfeldern der Antriebsmotoren wurde der Kompass auf dem obersten Stockwerk des Roboters montiert. Sehr wichtig für hohe Messgenauigkeit ist die genaue horizontale Ausrichtung des Kompassmoduls.

Die Werte können als PWM-Signal oder über I2C ausgelesen werden. Wir verwenden dafür den I2C-Bus mit einer Taktfrequenz von von 100kHz. Das Übertragungsprotokoll ist ähnlich dem eines I2C-Eproms und kann dem Datenblatt im Anhang entnommen werden.

Da während des Wettbewerbes unvorhersehbare Magnetfelder durch die gegnerischen Roboter entstehen können, sollte im 2. Teil der Arbeit noch ein zweites System ausprobiert werden.

3.1.8. Gyroskop

In der ersten Projektarbeit wurde für die Richtungsbestimmung ein elektronischer Kompass (cmp-03) eingesetzt. Dieser zeigte aber langsame Reaktionszeiten bei schnellen Drehungen und starke Abweichungen bei dynamischen Magnetfeldern (z.B. gegnerischer Roboter). In einer Projektarbeit im Wahlmodul Sensorik wurde eine Schaltung zur Auswertung eines elektronischen Gyroskops (Drehratensensor) vom Typ ADXRS300 erstellt. Dieser Sensor hat einen Signalausgang, welcher im Ruhezustand auf 2.5V liegt. Wird der Sensor gedreht, verändert sich das Ruhesignal mit einer Empfindlichkeit von 5mV/ $^{\circ}$ /s. dreht sich der Sensor z.B. mit 100 $^{\circ}$ /s liegt am Ausgang eine Spannung von 2V resp. 3V, je nach Drehrichtung.

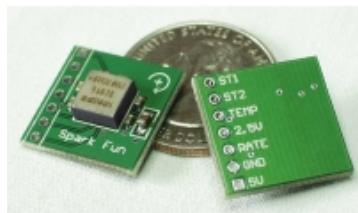


Abbildung 3.9.: Sensor ADXRS300

Mit einem Mikrokontroller wird das Signal eingelesen und integriert. Über eine I2C Schnittstelle kann die aktuelle Drehrichtung ausgelesen werden. Die Geräteadresse ist 0x50. Als Antwort auf eine Read-Anfrage erhält man 2 Bytes, die zusammen eine 16 Bit Variable Bilden. Der Wert dieser Variable bewegt sich zwischen 0 und 36000, also Grad mit Hundertstelauflösung.

Die Schwierigkeit des Gyroskopes liegt bei kleinen Integrationsfehlern in der Auswertung, die zu einem konstanten Drift führen. Dieser ist zudem noch temperaturabhängig. Da das Gyroskop erst gegen Ende der Arbeit aufgebaut wurde, konnte keine saubere Driftkompen-sation erreicht werden. Korrigiert wird nur statisch! Für die Spieldauer von 90 Sekunden reicht es aber aus.

Weitere Details können der Gyroskop-Dokumentation entnommen werden.

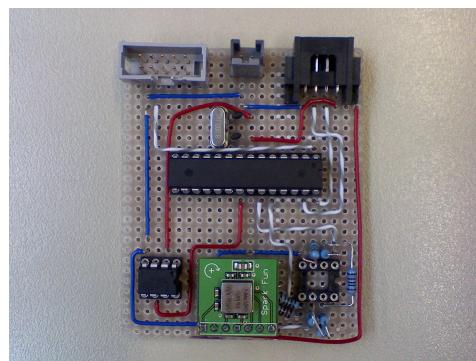


Abbildung 3.10.: Prototyp Gyroskop

3.1.9. Farbsensor

Das Eurobot Reglement schreiben vor, dass Bälle der gegnerischen Farbe nicht im Roboter gespeichert werden dürfen. Unsere Strategie sollte so ein Fall eigentlich ausschliessen. Tests haben aber gezeigt, dass im ungünstigsten Fall ein fremder Ball direkt vor den Dispenser rollen kann und von unserem Roboter aufgenommen wird.

Das Eurobot Team Zamboni hat für ihre Aufgabe gute Erfahrungen mit dem Sensor MCS3BT von Matzet gemacht. Vom Prinzip her handelt es sich um einen RGB Sensor mit drei Fotodioden mit entsprechenden Farbfiltern. Der Sensor wird mit einer Referenzspannung gespiesen und liefert für jeden Farbkanal ein Stromsignal. Für die Auswertung mit einem AD-Wandler muss das Signal mit einer geeigneten Verstärkerschaltung angepasst werden.

Die Beleuchtung spielt bei der Farberkennung eine wesentliche Rolle. Deshalb ist es wichtig, den Sensor an einem vor Fremdlicht geschützten Ort mit einer definierten, konstanten Beleuchtung zu platzieren.

Wir entschieden uns der Einfachheit halber ein eigenständiges Farbmodul auf einer kleinen Platine aufzubauen. Darauf befinden sich der Sensor, die weissen LEDs für die Beleuchtung, die Verstärkerschaltung und ein 8-Bit AD-Wandler mit einem I2C Bus Interface. Somit beschränkt sich der Verdrahtungsaufwand auf ein I2C-Kabel.

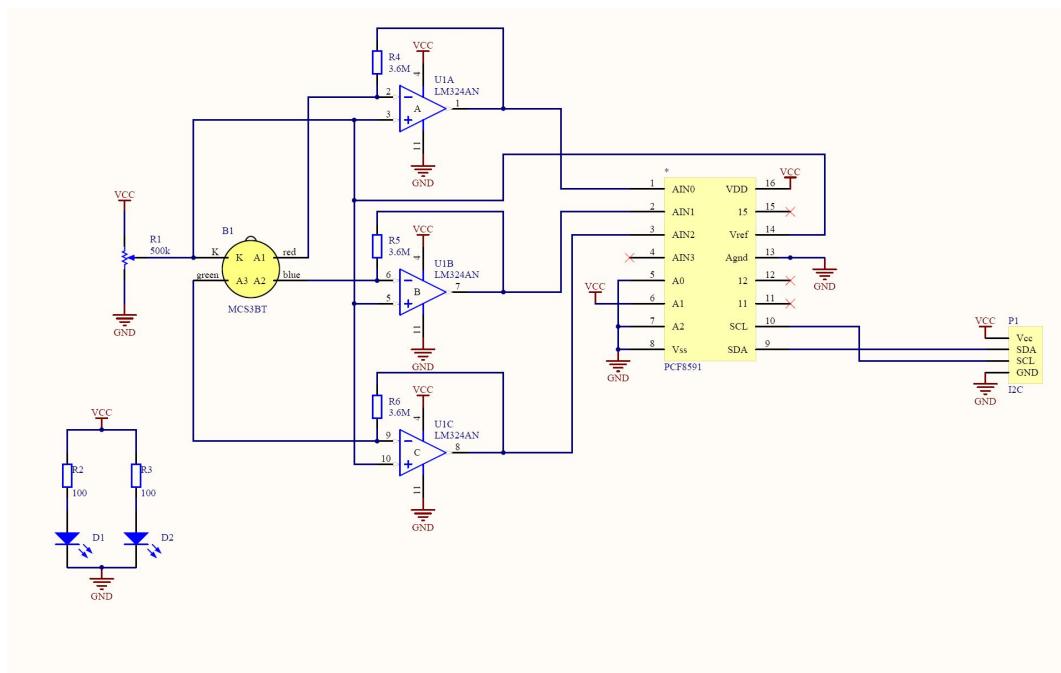


Abbildung 3.11.: Schema Farbsensorplatine

3.1.10. Software Sensorknoten

Programmiert wird in C mit der Entwicklungsumgebung AVR Studio (Version 4.13, Build 528). Als Compiler verwenden wir WinAVR. Zum Debuggen steht ein Atmel Programmiergerät JTAG ICE MK2 zur Verfügung.

Der PC soll sämtliche Sensorwerte abfragen können. Bei den Schaltern soll eine Zustandsänderung einen Interrupt auslösen, der die entsprechende CAN-Message absetzt. Leider sind bei dem Entwicklungsboard DVK90CAN1 nur zwei externe Interrupts auf die Stifitleisten geführt. Deshalb hat nur die Startschnur einen eigenen Interrupt. Die anderen Tastenzustände werden in einem periodischen Task ausgelesen. Zustandsänderungen werden über CAN gemeldet.

Die Aufgabe wurde in 3 Teile gegliedert:

- CAN Kommunikation
- I2C Kommunikation
- Bord IO

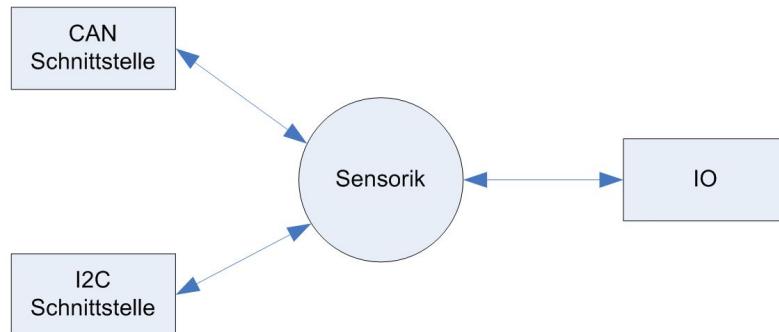


Abbildung 3.12.: grobe Softwarestruktur Sensorik

Da wir mit dem Echtzeitbetriebssystem μ C-OS arbeiten, ist es uns möglich eine Struktur mit Tasks, Message Queues und Flags aufzubauen.

Softwaredesign mit μ C-OS

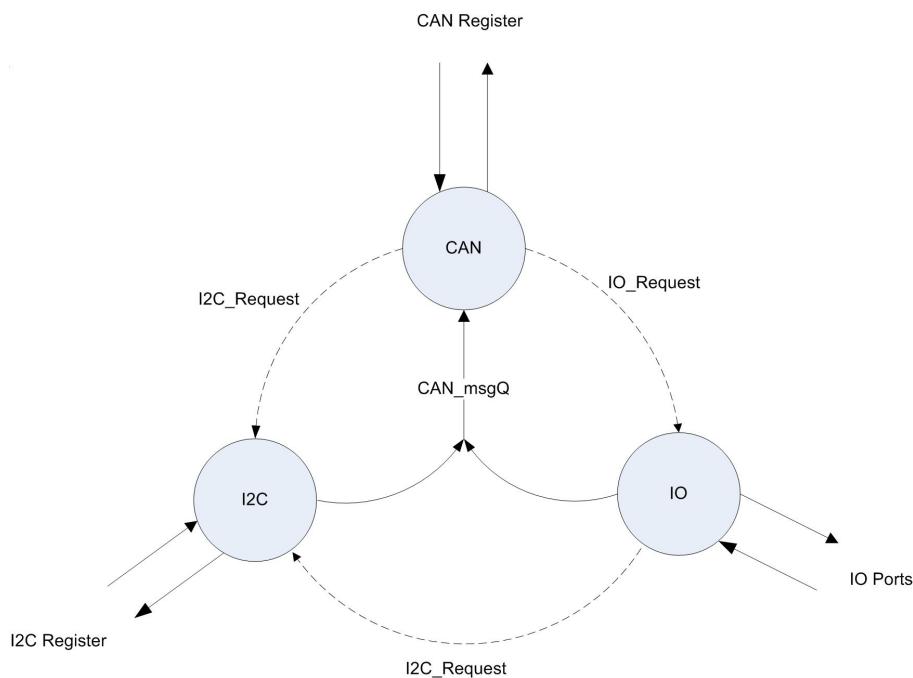


Abbildung 3.13.: Softwarestruktur Sensorik

Das CAN Modul ist verantwortlich für die Kommunikation mit dem Industrie-PC. Alle eingehende Anfragen werden hier ausgewertet, worauf die gewünschten Sensorwerte bei den jeweiligen Modulen mit Flags angefordert werden. Diese stellen eine neue CAN Nachricht mit den Messwerten zusammen und legen sie in die Message Queue des CAN Moduls.

μ C-OS-Objekte:

Task	Priorität	Aufgabe
Task_Start	0 / 7	System starten / periodische Aufgaben bearbeiten
Task_Can_Reception	1	Empfang der CAN Nachrichten
Task_Can_Transmission	2	Senden der CAN Nachrichten
Task_I2C	3	Senden und Empfangen über I2C
Task_Switch	3	IO Handling

Tabelle 3.1.: Taskliste

3. Sensoren und Aktoren

Flaggruppen	Flags
can_mob_status_tx	FLAG_MOB_STATUS_TX FLAG_MOB_SWITCH_TX FLAG_MOB_CMP_TX FLAG_MOB_DIST_TX FLAG_MOB_DRUM_TX
I2C_request	FLAG_I2C_CMP FLAG_I2C_DIST FLAG_I2C_DRUM FLAG_I2C_SWITCH FLAG_I2C_COLOR
I2C_IR	FLAG_I2C_IR
switch_status	FLAG_START FLAG_STOP FLAG_COLOR

Tabelle 3.2.: Flagliste

Message Queues	Task
can_transmission_msgq	Task_Can_Transmission
can_reception_msgq	Task_Can_Reception

Tabelle 3.3.: Message Queues

Semaphore	Task
I2C_Semaphore	Task_I2C

Tabelle 3.4.: Semaphoren

CAN Kommunikation

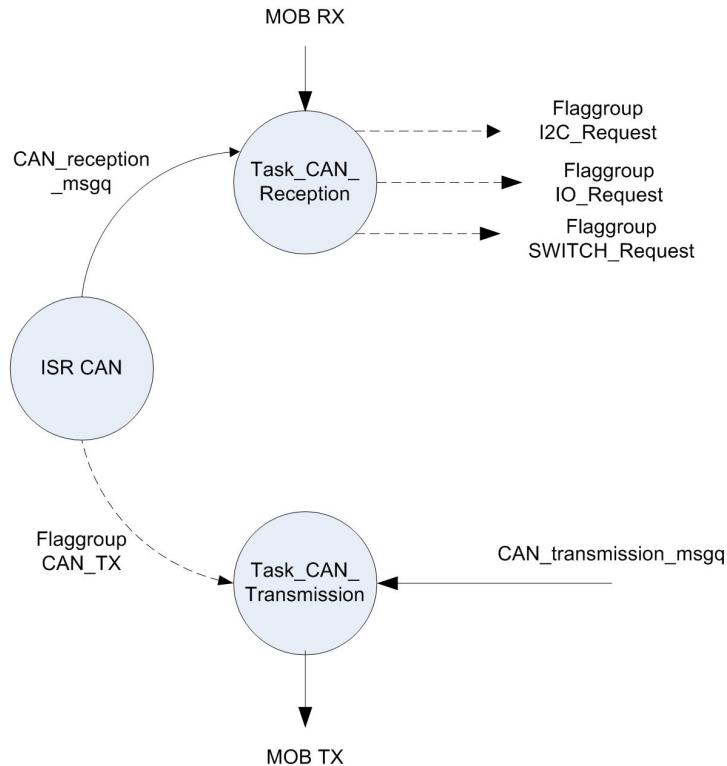


Abbildung 3.14.: Detail CAN

Der Controller AT90CAN128 besitzt eine integrierte CAN Schnittstelle. Atmel stellt für die Benutzung der Schnittstelle eine Reihe von praktischen Registern zur Verfügung, welche das Kommunikationshandling stark erleichtern. Es können sogenannte MOBs (Message Object) definiert werden. Jeder dieser MOBs hat einen eigenen Nachrichtenfilter. Ein MOB ist wie ein Briefkasten für eine CAN-Nachricht mit einer bestimmten Message-ID. Mit MOBs können Nachrichten gesendet und empfangen werden.

Hat einer der MOBs eine Nachricht erhalten oder erfolgreich gesendet, wird der CAN Interrupt ausgelöst. Eingehende Nachrichten werden in der ISR sofort aus dem MOB ausgelesen und an den Empfangstask weitergeleitet, damit bei einer hohen Abfragerate keine Anfragen verloren gehen (beim bearbeiten muss der MOB deaktiviert werden).

Der Sendetask CAN_Transmission erhält die zu sendenden Nachrichten in der Message Queue. Die Nachricht wird in den entsprechende MOB kopiert und zum Senden freigegeben. Ist die Nachricht erfolgreich abgesetzt worden, wird in der Interruptroutine ein Flag gesetzt, welches im Transmission Task den MOB wieder freigibt.

3. Sensoren und Aktoren

Aktion:	CAN-ID:	Inhalt:	von Atmel	von PC
STATUS_INIT	0x100	-	X	
STATUS_GAME_START	0x102	-	X	
STATUS_GAME_STOP	0x103	>> nicht verwendet		
COLOR_GET	0x104	-		X
COLOR_SEND	0x105	data[0] = Status Farbschalter, data[1]..[3] RGB	X	
RGB_GET	0x106	-		X
RGB_SEND	0x107	data[0] = Farbe: (0 = nichts, 1 = Rot, 2 = Blau, 3 = Weiss)	X	
CMP_VALUE_GET	0x110	-		X
CMP_VALUE_SEND	0x111	data[0] ..[1] = Gyro Wert 16Bit	X	
CMP_VALUE_INIT	0x112	nicht verwendet		X
DIST_VALUE_GET	0x120	-		X
DIST_VALUE_SEND	0x121	data[0] = D1, data[1] = D2, data[2] = D3, data[3] = D4;	X	
DRUM_VALUE_GET	0x130	-		X
DRUM_VALUE_SEND	0x131	data[0] = Trommelposition 8-Bit	X	
SWITCH_WALUE_GET	0x140	-		X
SWITCH_VALUE_SEND	0x141	data[0] = Schalterzustand -> [-, out, in pick]]	X	
SWITCH_EJECT_OUT	0x142	nicht verwendet	X	
SWITCH_EJECT_IN	0x143	nicht verwendet	X	
SWITCH_BALL_PICK	0x144	nicht verwendet	X	
SWITCH_INIT	0x145	-	X	
SWITCH_EJECT	0x146	-	X	

Abbildung 3.15.: CAN Nachrichten Sensorik

I2C

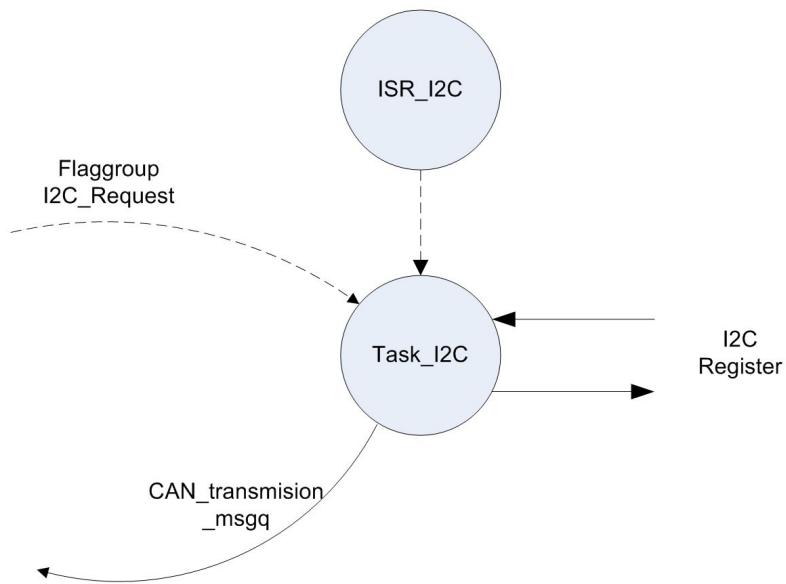


Abbildung 3.16.: Detail I2C

Sämtliche Sensoren die am I2C-Bus angeschlossen sind, werden über diesen Task ausgelesen. Dafür stehen Sende- und Empfangsfunktionen zur Verfügung.

IO

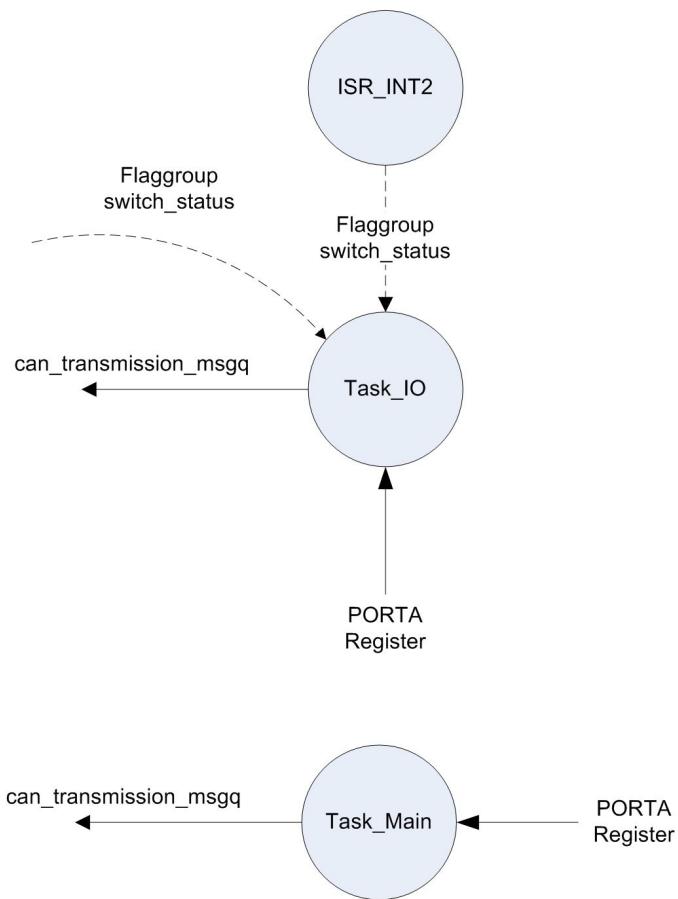


Abbildung 3.17.: Detail IO

Dieses Modul bearbeitet die Abfrage der Schalterzustände und die externen Interrupts, z.B. die Startschnur.

Für die beiden Taster 'Init' und 'Eject' reichten die externen Initerrupts nicht aus. Deshalb werden sie im Task_Main periodisch abgefragt. Positive Flanken werden dem PC über CAN gemeldet.

Umsetzung

1:1 verwendet wurden die Dateien des Betriebssystems und die des CAN-Treibers von T.Reber aus dem letzten Semester.

Neue Files und ihre Funktionen:

can.c - Funktionen die die Kommunikation über CAN ermöglichen

Task_Can_Reception	Wartet auf Flags des CAN Interrupts und holt die entsprechende Message aus dem Mob ¹ und leitet sie weiter
Task_Can_Transmission	Wartet auf Messages in der MSQ ² , legt diese im gewünschten Mob ab und setzt die Sendefreigabe
can_get_message	liest eine Nachricht aus dem Mob
can_put_message	legt Message in Mob ab
can_init	initialisiert die CAN Schnittstelle
ISR_CAN_TRANSFER_COMPLETE	Interrupthandling CAN

I2C.c - Funktionen für den I2C Bus

I2C_init	initialisiert die I2C Register
Task_I2C	Wartet auf Anfragen, holt die Sensorwerte und stellt eine neue CAN Nachricht zusammen
I2C_get	liest Daten von einem Slave
I2C_send	sendet Daten an einen Slave
I2C_interrupt	wird ausgelöst wenn I2C sendet oder empfängt

Kompass.c - Funktionen für den Kompass

getcompass	spezielle I2C Funktion, die auf das Protokoll des Kompass abgestimmt ist
------------	--

RTOS.c - Initialisierung und Start des Betriebssystems

main	
Task_Main	
init	Task für periodische Aufgaben alle Initialisierungen werden hier zusammengefasst ausgeführt
AppCreateTasks	
OSTickInit	

switch.c - Endschalter

interrupt_init	Initialisierung der Interruptregister
Task_Switch	Task für Endschalter
Start_Interrupt	Interrupt der Startschnur
AppCreateTasks	

interrupt.s - Interruptregister Eintrag

¹Mob = Message Object

²MSQ = Message Queue

3.2. Aktoren

3.2.1. Hardware

Es sind folgende Aktoren vorhanden:

- 2 Schrittmotoren der Firma Nanotec für den Antrieb
- 2 DC-Motoren der Firma VDO Antriebstechnik für den Einzugsbänderantrieb
- 1 Schrittmotor der Firma McLennan für die Speichertrommel
- 2 Servos der Firma HiTec für die Einzugsbänder
- 1 Servo der Firma HiTec für die Einzugsklappe
- 1 Servo der Firma Graupner für die Auswurfklappe
- 1 Servo der Firma Graupner für die Ballschiebevorrichtung

3.2.1.1. Schrittmotor für den Antrieb

Es wird der bestehende Antrieb des letztjährigen Teams [kju:] verwendet, allerdings werden stärkere Motoren eingesetzt. Die neuen Motoren sind wiederum Schrittmotoren der Firma Nanotec. Die Leistungselektronik sollte dabei noch verwendet werden können.

Die neuen Motoren sind vom Typ ST6018M2008 und werden als Direktantrieb (ohne Getriebe) verwendet.

Sie haben folgende Leistungsdaten:

Spannung: 36 VDC
Strom pro Phase: 1.4 A
Drehmoment Mot: 1.6 Nm

Die Logikprinte für die Ansteuerung der Motoren wurde vom Team [kju:] übernommen. Da wir die Motoren mit etwas Überlast betreiben soll ein Strom von 1.7 A fliessen. $R_S = 0.8V/1.7A = 0.47\Omega$ gemäss Datenblatt des Treibers IMT901.

3.2.1.2. Motor für die Einzugsbänder

Als Motoren für die Einzugsbänder werden ebenfalls bestehende Motoren des Teams [kju:] verwendet. Es handelt sich um einen Gleichstrommotor der Firma VDO Antriebstechnik des Typs M28x20/S.

Angaben zum Motor:

Spannung: 24V
Stromaufnahme: 120 mA
Drehzahl: 2800 rpm
Drehmoment Mot: 1.6 Ncm
Getriebe: 20:1



Abbildung 3.18.: Bänder Motor

Die Logikprinte für die Ansteuerung der DC-Motoren wurde ebenfalls übernommen. Da allerdings auf dieser Printplatte viel gelötet wurde und diese deshalb in sehr schlechtem Zustand war, wurde eine neue Printplatte angefertigt. Sie wurde ebenfalls mit neuen Bau- teilen bestückt. Die Einstellungen jedoch haben wir aus der Dokumentation des Teams [kju:] übernommen.

3.2.1.3. Schrittmotor der Speichertrommel

Auch dieser Motor ist bereits vom Team [kju:] her vorhanden. Es handelt sich um einen Schrittmotor der Firma McLennan vom Typ M82201-P2SB.

Angaben zum Motor:

Spannung: 36V
Strom pro Phase: 180 mA
Drehmoment Mot: 2.75 Ncm
Getriebe: 250:3
Drehmoment Getr: 80 Ncm



Abbildung 3.19.: Trommel Motor

Die Logikprinte für die Ansteuerung des Trommelmotors wurden übernommen. Auch diese Printplatte befand sich in sehr schlechtem Zustand. Deshalb wurde auch hier eine neue Printplatte angefertigt. Sie wurde ebenfalls mit neuen Bauteilen bestückt. Auch hier wurden die Einstellungen aus der Dokumentation des Teams [kju:] übernommen.

3.2.1.4. Servo für Einzugbänder

Als Servos für die Einzugbänder werden Servos vom Typ Hitec HS-645MG mit Metallgetriebe verwendet. Sie bewegen die Einzugbänder von der Grundstellung in die Arbeitsstellung.

Sie haben folgende Leistungsdaten:

Betriebsspannung:	4.8-6VDC
Leerlaufstrom:	9 mA
Betriebsstrom:	max. 400 mA
Stellmoment:	ca. 80 Ncm
Stellzeit:	0.24 s/60°
Gewicht:	55.2g

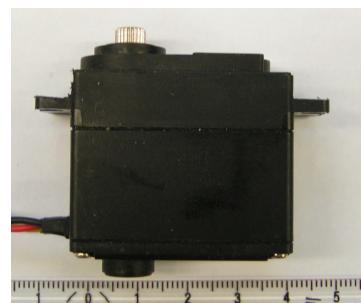


Abbildung 3.20.: Hitec Servo HS-645MG

3.2.1.5. Servo für Einzugklappe

Als Servo für die Einzugklappe wird ebenfalls ein HiTec HS-645MG Servo mit Metallgetriebe verwendet.

Sie haben folgende Leistungsdaten:

Betriebsspannung:	4.8-6VDC
Leerlaufstrom:	9 mA
Betriebsstrom:	max. 400 mA
Stellmoment:	ca. 80 Ncm
Stellzeit:	0.24 s/60°
Gewicht:	55.2g

3.2.1.6. Servo für Auswurfklappe

Als Servo für die Auswurfklappe wird ein Graupner C271 Micro Servo mit Metallgetriebe verwendet.

Sie haben folgende Leistungsdaten:

Betriebsspannung:	4.8-6VDC
Leerlaufstrom:	ca. 8mA
Betriebsstrom:	max. 560mA
Stellmoment:	ca. 14 Ncm
Stellzeit:	0.14 s/40°
Gewicht:	8g

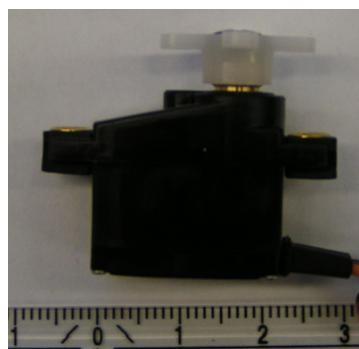


Abbildung 3.21.: Graupner Micro Servo C271

3.2.1.7. Servo für Ballschiebevorrichtung

Als Servo wird ein Graupner C261 Micro Servo mit Kunststoffgetriebe verwendet. Die Vorrichtung wird mechanisch stark belastet und die Getriebe werden dabei schnell zerstört. Bei diesem Servo kann jedoch das Getriebe schnell gewechselt werden, was auch der Grund zur Verwendung ist. Dieses Servo ist baugleich wie das Graupner Servo C271.

Das Servo hat folgende Leistungsdaten:

Betriebsspannung:	4.8-6VDC
Leerlaufstrom:	ca. 5mA
Betriebsstrom:	max. 450mA
Stellmoment:	ca. 14 Ncm
Stellzeit:	014 s/40°
Gewicht:	8g

3.2.2. Software

3.2.2.1. Controller

Die Software für den Aktorknoten läuft auf einem Atmel DVK90CAN1 Entwicklungsboard. Dieses Board enthält einen Controller des Typs AT90CAN128.

Auf dem Controller läuft das Echtzeitbetriebssystem μ C-OS/II. Dieses System wurde bereits im letzten Jahr vom Team [kju:] für diesen Controller portiert und konnte mit Anpassungen in gewissen Funktionen wie z.B. der CAN-Schnittstelle übernommen werden.

Die Programmiersprache für diesen Controller ist C.

Für die Schnittstelle zu den Logikprintplatten ist ein AddOn-Board angefertigt worden. Es ist nach folgendem Schema angeordnet:

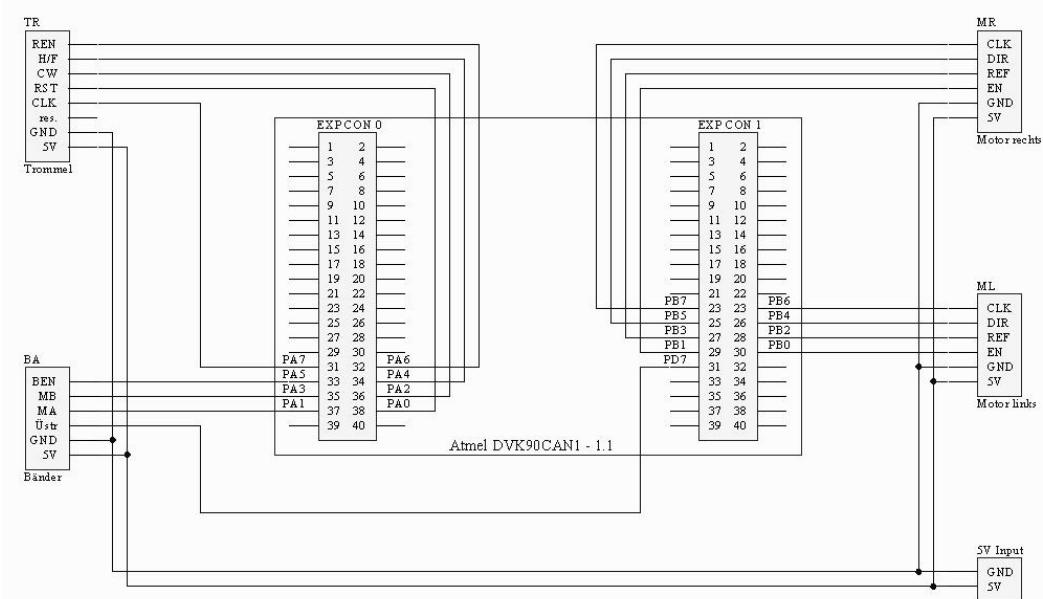


Abbildung 3.22.: Schema Atmel AddOn-Board

3. Sensoren und Aktoren

3.2.2.2. Definition der CAN-Messages

Um korrekt und koordiniert mit dem Aktorknoten kommunizieren zu können, müssen die CAN Nachrichten und die CAN-ID's festgelegt werden.

Dies geschah nach folgender Tabelle:

CAN Kommunikation Aktoren

Aktion	CAN-ID	Inhalt	data[0]	data[1]	data[2]	data[3]	data[4]	data[5]	data[6]	data[7]
STATUS_INIT_GET	0x200		---	---	---	---	---	---	---	---
STATUS_INIT_SEND	0x201		---	---	---	---	---	---	---	---
STATUS_GAME_START	0x202		---	---	---	---	---	---	---	---
STATUS_GAME_STOP	0x203		---	---	---	---	---	---	---	---
DRIVE_MOD	0x210	MOD	---	---	---	---	---	---	---	---
DRIVE_DRIVE	0x211	DIR_MR	EN_MR	SPEED_MR	---	DIR_ML	EN_ML	SPEED_ML	---	---
DRIVE_STEP	0x212	DIR_MR	ASR_100	ASR_10	ASR_1	DIR_ML	ASL_100	ASL_10	ASL_1	---
DRIVE_STEP_END	0x213	AN_FINISHED	---	---	---	---	---	---	---	---
SERVO_COMMAND	0x220	SERVO_COM	---	---	---	---	---	---	---	---
DRUM_COMMAND	0x230	DIR_TR	EN_TR	ANZ_POS_TR	AS_1000	AS_100	AS_10	AS_1	---	---
DRUM_STEP_END	0x231	TR_FINISHED	---	---	---	---	---	---	---	---
BAND_COMMAND	0x240	DIR_BA	EN_BA	---	---	---	---	---	---	---

SERVO_COMMAND:	data[0]
SERVO_BAND_AUF	0x01
SERVO_BAND_ZU	0x02
SERVO_KLAPPE_AUF	0x04
SERVO_KLAPPE_ZU	0x08
SERVO_AUSWURF_EIN	0x10
SERVO_AUSWURF_AUS	0x20
SERVO_ARM_EIN	0x40
SERVO_ARM_AUS	0x80

DRIVE_MOD:	data[0]
DRIVE	0x00
STEP	0x01

Abbildung 3.23.: CAN-ID's für Aktorknoten

3.2.2.3. Servoansteuerung

Im RS232-Task werden die Servobefehle an einen Micro Serial Servo Controller der Firma Pololu gesendet. Das Servoboard ist bei www.robotikhardware.de³ erhältlich.

Die Ansteuerung dieses Boards kann über mehrere Modes erfolgen. Wir benutzen den Pololu-Mode. Die genaue Beschreibung kann im Benutzerhandbuch nachgelesen werden.

Wie sieht das Protokoll aus:

start byte = 0x80	device ID = 0x01	command	servo num	data 1	data 2
-------------------	------------------	---------	-----------	--------	--------

Es stehen folgende Commands zur Verfügung:

- Command 0: Set Parameters
- Command 1: Set Speed
- Command 2: Set Position, 7-bit
- Command 3: Set Position, 8-bit

³<http://www.robotikhardware.de>

- Command 4: Set Position, absolute
- Command 5: Set Neutral

Wir haben das Command 3 verwendet. Damit können wir die Servoposition, aufgeteilt in data 1 und data 2, eingeben und senden. Die Werte für die Positionen wurden mittels Tests ermittelt.

Für die Anschlussbelegung auf dem Servo-Board siehe auch den Anleitung im Anhang auf Seite B.3.

3.2.2.4. Redesign Antriebssystem

3.2.2.4.1. Ausgangslage

Im Rahmen des Vertiefungsmoduls Echtzeitbetriebssysteme versuchten wir das bestehende Antriebssystem für den Eurobot zu Überarbeiten und echtzeitfähig zu machen. Folgende Komponenten waren bereits vorhanden:

- Atmel AT90CAN128 Mikrokontroller
- 2 Schrittmotoren ST6018M2008 von Nanotec für die Antriebe
- 1 Schrittmotor für die Trommel
- 2 DC-Motoren für die Bänder
- portierte Version von μ COS für den Atmel Kontroller
- zu diversen Aktoren existierten bereits Codefragmente
- Leistungstreiber, Ansteuerung per Rechtecksignal

Das Ziel war ein System zu entwickeln, dass für beliebige Schrittmotoren mit beliebigen Kenndaten funktioniert. Zudem sollten alle Aktoren weiterhin auf demselben System laufen.

3.2.2.4.2. Entwicklungsumgebung

Als Entwicklungsumgebungen evaluierten wir folgende IDEs:

Eclipse mit AVR-Plugin

Um mit **verschiedenen Betriebssystemen** kompatibel zu sein, versuchten wir unseren Atmel Knoten in Eclipse zum Laufen zu bringen. Dazu war folgendes notwendig:

- Die CDT-Version⁴ von Eclipse
- AVR Plugin für Eclipse⁵B.1
- Anleitung auf tinkerlog⁶

⁴<http://www.eclipse.org/cdt/>

⁵http://www.mikrocontroller.net/articles/AVR_Eclipse

⁶<http://tinkerlog.com/2007/06/03/programming-avr-with-eclipse-and-winavr/>

- für Windows den **libusb-Treiber**⁷, um das JTAG-Device in Eclipse ansprechen zu können
- die Tools aus der Sammlung **WinAVR**⁸

Eine Anleitung für Windows XP/Vista findet sich im AnhangB.1. Dazu ist folgendes anzumerken: Auf den Illustrationen den Atmel-Prozessor durch den richtigen ersetzen (hier at90can128). Zudem ist das Wort Debug einmal gross und einmal klein geschrieben - dies sollte (vor allem auf Linux-Systemen) einheitlich sein.

Das Debugging funktioniert in 3 Schritten:

1. Mikrokontroller flashen
2. Debug-Server starten
3. Debug-Client starten

Das Debugging ist leider auf Grund des GDB-Servers recht träge. Zur Entwicklung eignet sich Eclipse aber hervorragend.

AVR-Studio

Das AVR-Studio läuft bisher ausschliesslich unter Windows. Die neuste Version kann kostenlos von der **Atmel-Homepage**⁹ bezogen werden. Wenn ein neues Projekt erstellt wird, ist darauf zu achten, dass auch die μ COS spezifischen Files in die Projektmappe eingefügt werden - ansonsten werden sie anscheinend vom Compiler nicht gefunden. Das AVR-Studio unterstützt zudem **kein** „code completion“. Vor allem bei μ COS spezifischen Funktionen ist diese Funktion sehr nützlich. Das Flashen geht bei dieser Umgebung sehr flott.

⁷<http://libusb.wiki.sourceforge.net>

⁸http://sourceforge.net/project/showfiles.php?group_id=68108

⁹http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725

3.2.2.4.3. Realisation

Der C-Code des Projekts ist kommentiert. Eine entsprechende Doxygen-Dokumentation findet sich im Anhang unter E.7. Die Motoren werden mit Hilfe von C-Structs abstrahiert. Diese Strukturen bilden den Kern der Anwendung und werden in allen Programmteilen verwendet.

Um nun eine Regelung zu realisieren, wurde für jeden Motor zwei Strukturen angelegt: Eine

```

1 typedef struct MOTOR_PROPERTIES{
2     INT8U running;          // läuft der Motor?
3     INT8U direction;       // Drehrichtung
4     INT16U frequency;      // aktuelle Frequenz
5     INT32U steps;          // Schritte
6     INT8U stepMode;        // Schritt-Modus?
7 }MOTOR_PROPERTIES;
8
9 typedef struct MOTOR_STATUS{
10    MOTOR_PROPERTIES actual; // aktuelle Werte
11    MOTOR_PROPERTIES nominal; // gewünschte Werte
12    INT8U flagNumber;        // Motorennummer
13    INT16U freqIndex;        // aktueller Rampenindex
14    INT16U startFreq;        // Startfrequenz der Rampe
15    INT16U maxFreq;          // maximal zulässige Frequenz
16    INT16U minFreq;          // minimale Frequenz des Motors
17    INT8U changeingDirection; // wird momentan die Richtung geändert?
18 }MOTOR_STATUS;
```

enthält die aktuellen Daten und eine die gewünschten. Die Bändermotoren unterscheiden sich vom oben erwähnten Prinzip nur dadurch, dass für sie keine Regelung existiert - also auch nur eine Struktur pro Motor existiert. Zudem kennen die DC-Motoren keine Schrittmodus usw., es wurden also andere Strukturen verwendet.

Die Regelung erfolgt vom TaskSpeedControl aus (siehe 3.2.2.4.5). Um die Taktung der Steuerung kümmert sich Timer 2. Dieser setzt jede Millisekunde ein Flag für jeden Motor, welches in der Funktion motorSettings konsumiert wird. So kann die Zeit für die Anfahrt resp. Bremsrampe beeinflusst werden.

Je nachdem wie gross der Unterschied zwischen Soll und Ist-Frequenz ist, wird die Rampe langsamer oder schneller abgefahren, resp. Punkte in der Rampe übersprungen. Die Art (Funktion) der Rampe wird im Array freqRamp als Wertetabelle abgelegt. Momentan wird eine \sin^2 Funktion verwendet.

Die Motorenansteuerung kann im Wesentlichen durch folgende Struktogramme abstrahiert werden:

Der Task TaskSpeedControl ruft seriell und periodisch die beiden Funktionen beltControl und motorControl auf:

3. Sensoren und Aktoren



Abbildung 3.24.: Struktogramm TaskSpeedControl

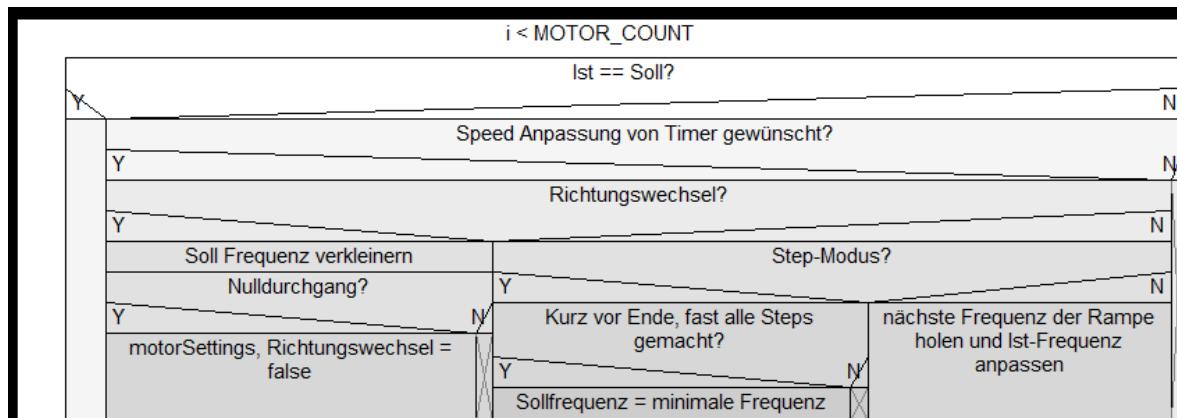


Abbildung 3.25.: Struktogramm motorControl

3.2.2.4.3.1. Rampe Zum Anfahren und Bremsen wird jeweils eine \sin^2 Rampe gefahren (Plot der verwendeten Wertetabelle siehe 3.26). Die Rampe ist als Wertetabelle im Programmspeicher implementiert. Bei kleinen Frequenzunterschieden wird auch die Rampe nicht mehr vollständig abgefahrene. Der Bremszeitpunkt im Schrittmodus¹⁰ wird ebenfalls mit Hilfe einer Wertetabelle ermittelt. Diese Wertetabelle wurde mit Hilfe eines selbstgeschriebenen Java-Tools RampCalcToolJ erstellt¹¹. Allerdings ist zu sagen, dass der Bremsvorgang auch experimentell angepasst werden muss, da die berechneten Werte nicht unbedingt mit der Realität korrelieren.

¹⁰Fahre Anzahl Schritte

¹¹Der Code dazu ist im Subversions-Backup enthalten

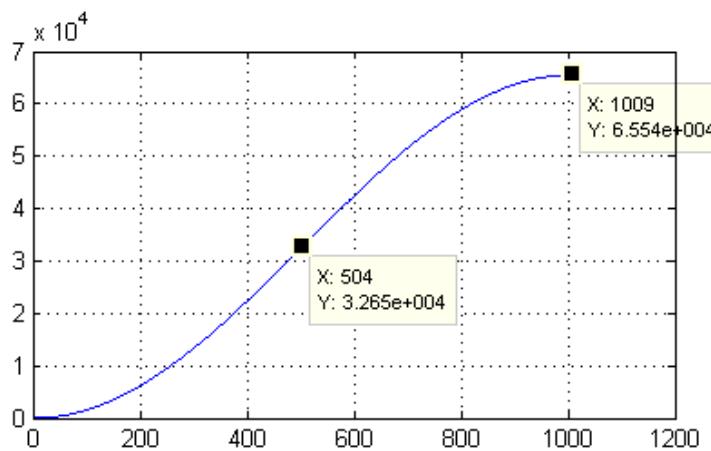


Abbildung 3.26.: Beschleunigungsrampe

3.2.2.4.3.2. Probleme/Tipps Probleme gab es vor allem im Zusammenhang mit Timern und der CAN-Ansteuerung. Sobald die Timer-ISR zu viele Instruktionen enthält oder der Timer zu schnell läuft, kann das zum Systemabsturz führen. Der Prozessor blockiert während einer ISR anscheinend nicht den aktuellen Interrupt, sodass dieser auch während der ISR verschachtelt auftreten kann und so zum Stack-Overflow führen kann. Ein Hinweis darauf liefert der Stack-Pointer SP oder auch die μ COS interne Variable OSIntNesting, die angibt wie viele verschachtelte Interrupts zurzeit vorhanden sind.

Eine weitere Fehlerquelle kann eine fehlende Interrupt-Routine darstellen. Wenn zum Beispiel für einen Timer mittels Register der Overflow-Interrupt aktiviert wurde, dann gibt es bei jedem Überlauf des Timers einen Interrupt. Wenn nun keine entsprechende ISR vorhanden ist, dann ruft der Prozessor die Standard-ISR auf, welche den Prozessor neu startet. Um das zu verhindern oder dem Problem auf die Schliche zu kommen, kann folgender Code verwendet werden:

```

1 SIGNAL (__vector_default){
2     int testVariable = 1; // irgendeine Variable anlegen, um einen Breakpoint
                           machen zu können
3 }
```

Nun wird standardmäßig die eigene ISR aufgerufen. Hier sollte man einen Breakpoint setzen. Natürlich sollte die eigene ISR nur zu Debugging-Zwecken verwendet werden.

Die CAN-Implementierung wurde vom Team kju realisiert und enthält gewisse Unschönheiten. So wird bei jeder empfangenen CAN-Nachricht ein Task angeworfen, nur um die Nachricht dann in einen Message-Queue zu speichern. Bei dieser Bearbeitung können allerdings weitere CAN-Interrupts auftreten und die im CAN-Register liegenden Daten können überschrieben werden. Zudem kann es vorkommen, dass ein MOB noch nicht wieder freigegeben wurde, bevor der nächste Interrupt auftritt. Dies hat zur Folge, dass gewisse CAN-IDs plötzlich „tot“ sind. Diese Probleme treten meist erst bei schnellen Poll-Zyklen im Bereich von unter 50ms auf. **Das Problem konnte im Rahmen dieser Arbeit noch nicht be-**

hoben werden! Philipp Haslebacher hat aber auf seinem Sensorknoten eine entsprechende Anpassung vorgenommen. Für das Antriebssystem, welches im Rahmen der Bachelor-Arbeit entwickelt wird, ist ebenfalls eine Anpassung geplant.

Zu Gunsten der Performance wurden einige Semaphoren wieder entfernt. Dies meist im Zusammenhang mit den Motoren-Strukturen. Auf diese wird meistens nur lesend zugegriffen, sodass eine Korruption der Daten weitgehend ausgeschlossen werden kann.

Der RAM-Speicher ist mit 4K für grosse Anwendungen recht knapp bemessen. Jeder zusätzliche Tasks braucht seinen eigenen Stack, der meistens ca. 300 Byte gross sein sollte. Daher ist es sinnvoll, grosse Lookup-Tabellen in den Programmspeicher zu verlegen. Auf diesen Speicher kann dann logischerweise nunmehr lesend zugegriffen werden. Folgende Funktionen werden von Atmel zur Verfügung gestellt:

```
1 #include <avr/io.h>
2 #include <avr/pgmspace.h>
3
4 const INT16U freqRamp[] PROGMEM =
5 {0,
6 1,
7 2};
8
9 INT16U getRampValue(INT16U index)
10 {
11     INT16U value = pgm_read_word(&freqRamp[index]);
12     return value;
13 }
```

Weitere Tipps finden sich auf [mikrokontroller.net](http://www.mikrokontroller.net)¹².

¹²<http://www.mikrocontroller.net/articles/AVR-GCC-Codeoptimierung>

3.2.2.4.4. Datenflussdiagramm

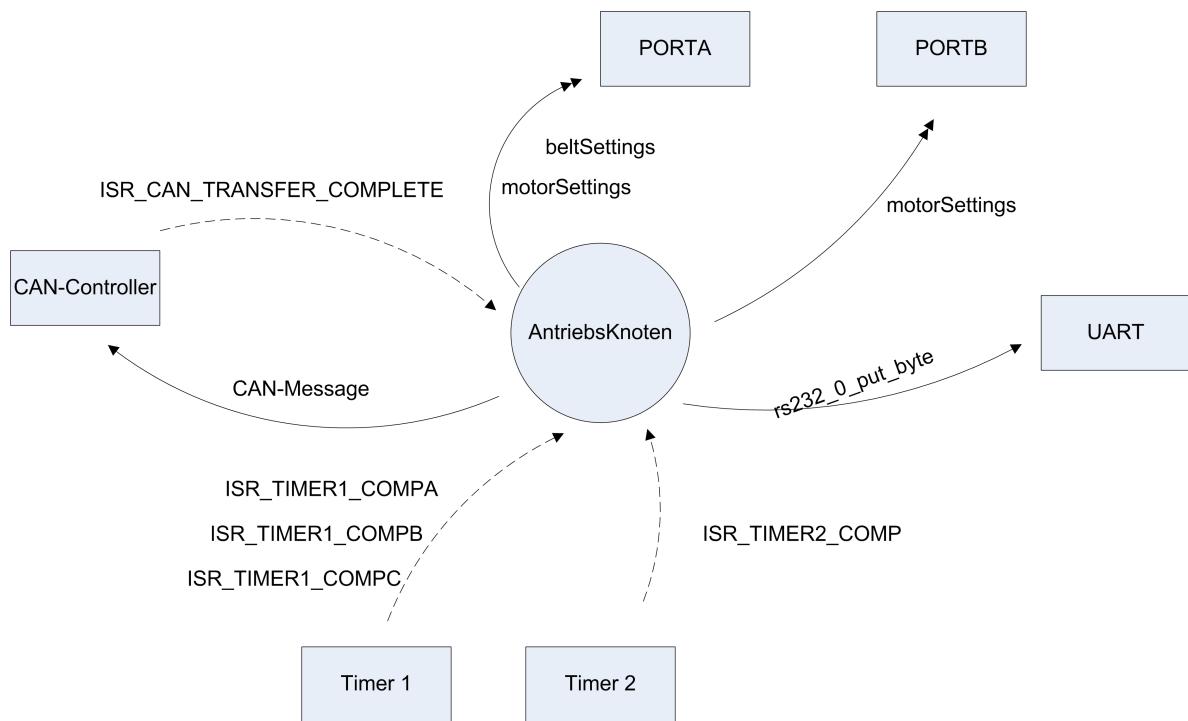


Abbildung 3.27.: Datenflussdiagramm

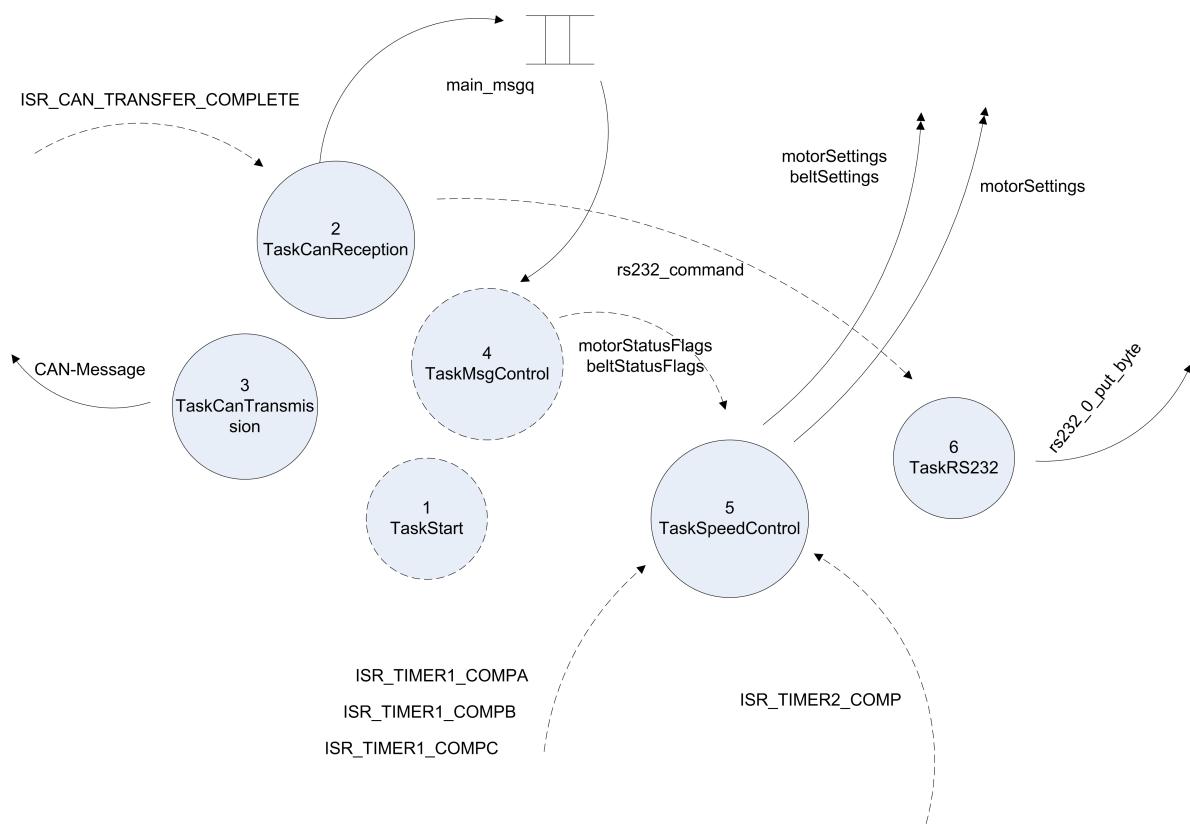


Abbildung 3.28.: Datenflussdiagramm

3.2.2.4.5. Mini-Specs

Task 1: TaskStart

Dieser Task dient allein dazu, die anderen Tasks zu initialisieren. Insofern hat er eine reine Kontrollfunktion. Er geht nach Beendigung der Aufgabe in den Suspend-Zustand über.

Task 2: TaskCanReception

Nachdem eine CAN-Nachricht empfangen wurde, teilt dies der CAN-Controller dem Betriebssystem per Interrupt (ISR_CAN_TRANSFER_COMPLETE) mit. In der entsprechenden Interrupt-Routine wird ein Flag für Reception-Task gesetzt. Dieser verarbeitet die erhaltenen Daten und füllt sie in die Message-Queue `main_msq` ab.

Task 3: TaskCanTransmission

Zur Übermittlung einer CAN-Nachricht werden Daten in eine Message-Queue `can_transmission_msq` abgefüllt. Sobald eine Nachricht vorhanden ist, wird diese an den Controller weitergeleitet. Eine Interrupt-Routine wird aufgerufen, wenn der Transfer abgeschlossen ist. In dieser Routine wird ein entsprechendes Flag gesetzt, dass der Controller erneut sendebereit sei.

Task 4: TaskMsgControl

Dieser Task wartet auf eine Nachricht vom Task TaskCanReception. Sobald sich eine solche in der Message-Queue befindet, wird mit einer weiteren Verarbeitung der Nachricht

begonnen und entsprechende Kontrollstrukturen aktiviert. Beispielsweise wird die Motoren-Struktur mit Daten (wie Geschwindigkeit und Richtung) aus der CAN-Nachricht versorgt.

Task 5: TaskSpeedControl

Hier werden die Geschwindigkeiten der verschiedenen Motoren geregelt. 5 Motoren sind vorhanden:

- 2 Antriebsmotoren (Schrittmotoren)
- 2 Bändermotoren (DC-Motoren)
- 1 Trommelmotor (Schrittmotoren)

Als Zeitbasis für die Regelung dient der Timer 2, der periodisch ein Flag setzt und so eine Anfahr- oder Bremsrampe ermöglicht. Die Bändermotoren werden nicht geregelt - für sie gibt es nur die Zustände ein oder aus. Das Taktsignal für die Schrittmotoren wird jeweils im Timer 1 generiert. Es ist auch dieser Timer, der das Rechtecksignal auf PORTA resp. PORTB gibt.

Task 6: TaskRS232

Dieser Task wird direkt per Flag aus dem Task TaskCanReception aktiviert. Sobald eine CAN-Nachricht mit einem Servo-Befehl ankommt, wird ein Flag gesetzt, dass die gewünschte Aktion dem RS232 mitteilt. Der RS232-Task generiert Befehle, die über die serielle Schnittstelle an das Servo-Board geschickt werden.

3.3. Speisungsanschlüsse

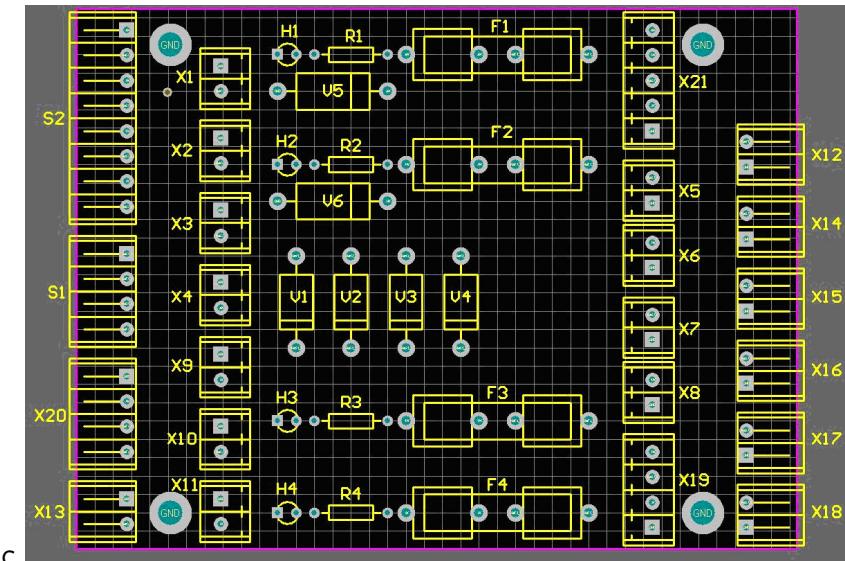


Abbildung 3.29.: Speisungsprint

Anschlüsse:

S1	Ein/Aus Schalter
S2	NOT-AUS
X1	36V Netzgeräteinspeisung
X2	24V Netzgeräteinspeisung
X3	36V Reserve
X4	36V Reserve
X5	24V Trommelmotorspeisung
X6	24V Reserve
X7	12V Reserve
X8	12V Aktoren-Atmel- und Servoboardspeisung
X9	5V Navigationssystemspeisung
X10	5V Aktoren-Atmel- und Servospeisung
X11	5V Reserve
X12	36V Akkueinspeisung
X13	24V Akkueinspeisung
X14	36V Antriebmotorspeisung links
X15	16V Antriebmotorspeisung rechts
X16	24V Bänderantriebsspeisung
X17	12V Reserve
X18	12V Reserve
X19	12V und 5V Sensoren-Atmelspeisung
X20	Industrie PC
X21	NC

3.4. Navigationssystem

3.4.1. Kommunikation über RS232

Einstellungen:

- 57600 Baud
- 8 Datenbits
- 1 Stopbit
- keine Parität
- keine Flusskontrolle

Aktion	Anfrage	Antwort
Position eigener Roboter bestimmen	\$PO<CR>	\$PO,e,xxxx,yyyy<CR>
Position gegnerischer Roboter bestimmen	\$PE<CR>	\$PE,e,xxxx,yyyy<CR>
Ping an Komponenten des Ortungssystems	\$PG<CR>	\$PG,a,b,c,d<CR>
Schallgeschwindigkeit abfragen	\$GC<CR>	\$GC,cccc<CR>
Schallgeschwindigkeit setzen	\$SC,cccc<CR>	\$SC,cccc<CR>

Tabelle 3.5.: Kommunikation über RS232

Abkürzung	Beschreibung
e:	Error-Code
xxxx:	x-Position in mm
yyyy:	y-Position in mm
a:	Status (1/0) Sender gegnerischer Roboter
b:	Status (1/0) Empfänger 1
c:	Status (1/0) Empfänger 2
d:	Status (1/0) Empfänger 3
cccc:	neue oder aktuelle Schallgeschwindigkeit in m/4s

Tabelle 3.6.: Legende Abkürzungen

Error-Code:

0: Ok

1: ungültige oder keine Werte von Baken

2: Position nicht auf dem Tisch nach erster Iterationsphase

3: Position nicht auf dem Tisch nach zweiter Iterationsphase

4: Position nicht gültig (Vergleich mit drittem Baken)

4. Industrie PC

4.1. Hardware

Den Embedded PC haben wir von unseren Vorgängern übernommen, die ihn bereits mit Erfolg eingesetzt haben. Dabei handelt es sich um das Model MSM855 der Firma Digitallogic. Verbaut ist ein Intel Celeron M Prozessor mit 1 GHz, 512MB RAM und eine 32GB Flash-Festplatte als Datenspeicher. Zu Beginn verwendeten wir eine herkömmliche Festplatte. Es zeigte sich aber, dass diese auf Dauer den Beschleunigungsbelastungen nicht gewachsen ist, so dass wir während der Testfahrten teilweise nicht mehr auf die Harddisk zugreifen konnten.

4.1.1. Navigation

Zur Navigation setzen wir das Ultraschall-Navigationssystem unserer Vorgänger ein. Diese haben dieses soweit fertig gestellt, dass wir es nur noch an der seriellen Schnittstelle des Embedded PC anschliessen und die entsprechende Java-Klassen schreiben (5.4.15) mussten.

4.2. Betriebssystem & Tools

Um eine hohe Stabilität und Funktionalität zu erhalten, haben wir uns entschieden, ein Debian GNU/Linux als Betriebssystem zu verwenden.

4.2.1. Installation Betriebssystem

Da wir ursprünglich die CAN-Bus Karte des PCs verwenden wollten, welche über einen Treiber verfügt, der primär für den 2.4er Kernel geschrieben ist, installierten wir zu Beginn die Debian Version 3.1 sarge.

Bei der Evaluation des Treibers mussten wir feststellen, dass dieser zu wenig gut arbeitet. Daher haben wir uns für ein USB-CAN-Dongle entschieden, wessen Treiber am Besten mit dem 2.6er Kernel arbeitet. Daher haben wir das System nachträglich auf die Version 4.0, Codename etch, aktualisiert.

4.2.1.1. Installationsvorbereitung

(basiert auf: <http://www.de.debian.org/releases/sarge/i386/apcs04.html.de>)

- Windowspartition verkleinern
- PC mit Rettungssystem (RIP Linux) starten

- 2 Partitionen anlegen: /dev/hda2 für swap und /dev/hda3 als ext3
- Swap einbinden: mkswap /dev/hda2 ; sync ; swapon /dev/hda2
- Zielpartition mounten:

```
mkdir /mnt/debinst ; mount /dev/hda3 /mnt/debinst
```

- debootstrap holen:

```
wget http://ftp.debian.org/debian/pool/main/d/debootstrap/  
debootstrap_1.0.3_all.deb
```

- debootstrap entpacken: ar - x debootstrap_1.0.3_all.deb
- debootstrap einrichten: cd / ; zcat /root/work.tar.gz | tar xv
- debootstrap ausführen:

```
/usr/sbin/debootstrap --arch i386 sarge /mnt/debinst http://ftp  
.ch.debian.org/debian
```

- ins System wechseln:

```
LANG=C chroot /mnt/debinst /bin/bash  
export TERM=xterm-color
```

4.2.1.2. Installation

- fstab editieren: vi /etc/fstab

```
/dev/hda3      /      ext3      defaults      0 1  
/dev/hda2      none    swap      sw          0 0  
proc          /proc   proc      defaults      0 0
```

- Devices mit mount -a einbinden
- Zeitzone einrichten

```
vi /etc/default/rcS  
UTC=no  
tzconfig      -> Europe -> Zurich
```

- Netzwerk konfigurieren: vi /etc/network/interfaces

```
auto lo  
iface lo inet loopback  
auto eth0  
iface eth0 dhcp
```

- Hostname setzen: echo eurobot > /etc/hostname

- /etc/hosts konfigurieren:

```
127.0.0.1 localhost eurobot
```

- /etc/apt/sources.list konfigurieren

```
deb http://ftp.ch.debian.org/debian sarge main contrib non-free
deb-src http://ftp.ch.debian.org/debian sarge main contrib non-free
deb http://security.debian.org/sarge/updates main contrib non-free
deb-src http://security.debian.org/sarge/updates main contrib non-free
```

- Paketcache aktualisieren: apt-get update

- Locales installieren: apt-get install locales : de_CH.UTF-8 auswählen

- Tastatur einrichten: dpkg-reconfigure console-data : quertz -> Swiss -> German -> latin1 wählen

- apt-get install vim (mit vim arbeitet es sich leichter als mit vi)

- /etc/kernel-img.conf anpassen:

```
do_symlinks = yes
relative_links = yes
do_bootloader = yes
do_bootfloppy = no
do_initrd = yes
link_in_boot = no
```

- Festplatte verfügbar machen: mknod /dev/hda3 b 3 3 (ansonsten macht mkinird beim Installieren des Kernels einen Fehler)

- Kernel-Image installieren: aptitude install kernel-image-2.4-686

- Bootmanager lilo installieren: apt-get install lilo

- lilo konfigurieren: vim /etc/lilo.conf

- allgemeine Devices erstellen: cd /dev ; ./MAKEDEV generic

- lilo installieren

```
mknod /dev/hda b 3 0
lilo

reboot
```

4.2.1.3. System einrichten

Da beim ersten Start Modulfehler auftreten (via 2.6er ein 2.4er Kernel installiert), muss die Modulkonfiguration angepasst werden:

```
depmod -a  
update-modules  
echo e100 >> /etc/modules  
reboot
```

Nach dem erneuten Reboot kann nun mit der wirklichen Einrichtung begonnen werden. Zuerst braucht es noch etwas Software:

```
aptitude install less make gcc kernel-source-2.4 ssh unzip texinfo  
tex
```

Der Kernel Quellcode muss entpackt und vorbereitet (für OCAN) werden:

```
cd /usr/src  
tar -xjf kernel-source-2.4.27.tar.bz2  
ln -s kernel-source-2.4.27 linux  
cd linux  
cp /boot/config/2.4.27-3-686 .config  
vim Makefile ( EXTRAVERSION = -3-686 )  
make oldconfig  
make dep  
OCAN herunterladen:  
cd  
mkdir ocan  
cd ocan  
mkdir dl-drv ( für DigitalLogic Driver )  
cd dl-drv  
LINUX.zip von Digital-Logic hier hin kopieren  
unzip LINUX.zip  
cd LINUX  
make  
chmod +x ocan_load  
./ocan_load type=4 base=0x340 irq=9
```

Für die Module: aptitude install module-assistant

Für java: aptitude install java-package

Nun stellte sich heraus, dass OCAN keine Lösung ist und wir nun mit dem CAN-USB-Dongle arbeiten.

4.2.1.4. System Upgrade auf Debian Etch mit 2.6er Kernel

- in /etc/apt/sources.list sarge durch etch ersetzen
- aptitude update && aptitude dist-upgrade

- aptitude install linux-image-2.6-686
- aptitude install linux-headers-2.6-686
- aptitude remove lilo
- aptitude install grub
- /boot/grub/device.map editieren: (hd0) /dev/hda hinzufügen
- /boot/grub/menu.lst erstellen:

```
default 0
timeout 4
title Debian
root (hd0,2)
kernel /vmlinuz root=/dev/hda3 ro
initrd /initrd.img

title WinXP
root (hd0,0)
makeactive
chainloader +1
```

- aptitude remove java-package
- aptitude install sun-java5-jre

4.2.1.5. PCAN-Treiber installieren

Der Treiber kann von <http://peak-system.com/linux/> heruntergeladen werden.

- peak-linux-driver-6.4.tar.gz nach /root/pcan kopieren
- tar -xzf peak-linux-driver-6.4.tar.gz
- mv peak-linux-driver-6.4 /usr/src
- cd /usr/src
- tar -xjf linux-headers-2.6.18-5-686.tar.bz2
- ln -s linux-headers-2.6.18-5-686 linux
- cd peak-linux-driver-6.4
- make
- make install

4.2.1.6. RS232 Installation

Achtung: /dev/ttyS0 scheint defekt zu sein, daher verwenden wir /dev/ttyS1

Zur Kommunikation verwenden wir die Java Communication API für Linux. Wichtig bei deren Installation ist, dass libLinuxSerialParallel.so nach /usr/lib und javax.comm.properties nach JVM/jre/lib kopiert wird.

4.2.2. Anwendungsumgebung Roboterseitig

Aus verschiedenen Gründen haben wir uns entschieden, Java als Programmiersprache für die Programmierung des Hauptrechners einzusetzen. Nicht zuletzt haben dabei Komfort, Flexibilität und der hohe Abstraktionsgrad eine Rolle gespielt. Hauptnachteil ist der Performance-Verlust auf Grund des Interpreters. Dies sollte aber dank des performanten Rechners und den bescheidenen Anforderungen keine Probleme darstellen. Nachfolgend gehen wir auf die Konfiguration des Rechners ein, die uns folgende Dinge ermöglicht:

- Update des Source-Codes per SVN
- Einfaches Kompilieren mittels shell-Skripten
- Hohe Flexibilität beim Debugging dank MySQL-Datenbank und Remote-Zugriff

4.2.2.1. Kompilation

Das Kompilieren wird in mehreren Stufen automatisiert. Zum Einen setzen wir das Build-Tool ANT ein, welches mittels einer entsprechenden Konfigurationsdatei¹ automatisch die Klassen des Projektes erstellt und auch ein JAR-File zur einfachen Ausführung und Distribution generiert. Der Build-Befehl wird wiederum in shell-Skripten² gekapselt, welche auch aktuelle Sourcen abholen und die Main-Klasse festlegen können.

Da wir einige Libraries verwenden, wird der eigentliche Build-Befehl durch ANT stark vereinfacht:

```
ant -f build-jar.xml
```

Im Gegensatz zu:

```
javac -encoding utf-8 -cp src:lib:lib/log4j.jar:lib/mysql-
      connector-java-5.1.5-bin.jar src/eurobot/*.java
javac -encoding utf-8 -cp src:lib:lib/log4j.jar:lib/mysql-
      connector-java-5.1.5-bin.jar src/eurobot/test/*.java
```

4.2.2.2. Debugging

Von Anfang an vorgesehen war eine komfortable Debugging-Möglichkeit um die Fehlersuche zu erleichtern. Diese Lösung ist in Form des Java-Paketes eurobot.log implementiert. Es ermöglicht die Verwendung mehrerer Logger, welche unter anderem in Log-Files, auf

¹build-jar.xml

²getAndBuild.sh, run.sh, runJar.sh

die Konsole und/oder in eine Datenbank schreiben können. Als Datenbank verwenden wir MySQL:

4.2.2.3. Datenbank

Die verwendete MySQL-Datenbank erlaubt auch Zugriffe aus einem Netzwerk. So können wir mit geeigneter Filterung einfach Fehler oder sonstige Informationen auf dem Roboter analysieren. Mehrere Spalten enthalten wichtige Informationen. Hier das Grundgerüst mit einem Test-Eintrag:

Date	Logger	Priority	ID	...
2007-10-29 16:26:01	MultiLogger	ERROR	E000	...

...	Thread	Class	Message
...	main	eurobot.test.TestLogging	Testnachricht!

Tabelle 4.1.: MySQL-Datenbank Tabelle LogTable

Der Datenbank-Zugriff ist sowohl lokal als auch aus einem Netzwerk per MySQL-Konsole möglich:

```
mysql -uroot -peurobot -heurobot.kicks-ass.org eurobotlog
```

Die URL eurobot.kicks-ass.org zeigt übrigens per ddclient immer auf die aktuelle IP, sofern diese dynamisch bezogen ist und ddclient aktiviert ist. Siehe unten.

4.2.2.4. Netzwerkkonfiguration

Über automatisch ausgeführte Skripte kann sich das System im Netzwerk selber konfigurieren. Für die Wettkämpfe ist dies aber nicht gewünscht, so wird dort eine statische Netzwerkkonfiguration verwendet.

4.2.2.4.1. dynamisch

Bei der dynamischen Netzwerkkonfiguration sind in der *crontab* (bearbeiten mit *crontab -e*) die Befehle *reconfigureNetwork.sh* und *update_etc_hosts.sh* aktiviert.

Das Skript *reconfigureNetwork.sh* stellt fest welche Devices aktiv sind und verändert die Netzwerkkonfiguration dementsprechend. *update_etc_hosts.sh* verändert den Eintrag in */etc/hosts* entsprechend. Dies ist für die RMI-Verbindung nötig.

Die Datei */etc/network/interfaces* ist wie folgt anzupassen:

```
auto eth0
iface eth0 inet dhcp
```

Die weiteren Netzwerkdevices sind entsprechend zu konfigurieren.

4.2.2.4.2. statisch

Bei der statischen Konfiguration sind die oben erwähnten Skripte in der *crontab* deaktiviert. Weiter ist dann in */etc/network/interfaces* eine statische IP-Adresse zu konfigurieren:

```
auto eth0
iface eth0 inet static
    address 192.168.3.2
    netmask 255.255.255.0
```

Andere Netzwerkdevices sind zu deaktivieren um allfällige Probleme zu vermeiden.

4.2.2.5. commandPrompt.sh

Das Shell-Skript `commandPrompt.sh` (in `/root`) bietet die Möglichkeit einige Aktionen einfach auszuführen:

```
BITT:~# ./commandPrompt.sh

Eurobot commandPrompt.sh

_____
1) getAndBuild.sh      (svn export) ITDS
2) updateAndBuild.sh   (svn update) ITDS
3) updateAndBuildVaio.sh (svn update) VAO
4) combat mode (NONE)  1.+2. Mal normal ausladen
5) combat mode A       1. Mal bei Gegner ausl., 2. Mal normal
6) combat mode B       2. Mal bei Gegner ausl., 1. Mal normal
7) combat mode A+B     1.+2. Mal zum Gegner (ausladen)
8) reset battery timer
9) run AkkuSparen.sh
0) quit

enter a digit
```

Durch das Drücken der entsprechenden Ziffer wird der entsprechende Befehl ausgeführt. Die Befehle 4-7 werden in einer Endlosschleife ausgeführt welche erst unterbrochen wird, wenn der Rückgabewert der Java-Software, und somit auch von `run.sh`, welches den Rückgabewert weiterreicht, nicht gleich 1 ist. Dadurch kann sich die Java-Software mit dem Rückgabewert 1 selbst neu starten.

4.2.3. Skripte auf dem Roboter

4.2.3.1. /root/scripts/battMonitoringCheck.sh

Schau ob der Batterietimer abgelaufen ist. Wenn ja, wird ein Javaprogramm ausgeführt, welches sagt, dass man die Batterie wechseln solle. Dieses Skript kann per crontab aufgerufen werden.

4.2.3.2. /root/scripts/battMonitoringReset.sh

Setzt den Batterietimer zurück

4.2.3.3. /root/scripts/checkNetDevices.sh

Schau welche Netzwerkdevices vorhanden sind

4.2.3.4. /root/scripts/eth0_up.sh

Ethernet einschalten (alte version)

4.2.3.5. /root/scripts/reconfigureETHERNET.sh

Ethernet einschalten/Wlan deaktivieren

4.2.3.6. /root/scripts/reconfigureNetwork.sh

Schaut welche Netzwerkdevices aktiv sind und konfiguriert entsprechend eines. Kabel hat vor Drahtlos Priorität.

4.2.3.7. /root/scripts/reconfigureWLAN.sh

WLAN einschalten/Ethernet deaktivieren

4.2.3.8. /root/scripts/update_etc_hosts.sh

Die eigene IP-Adresse am richtigen Ort ins /etc/hosts schreiben. Das ist für die Java-RMI-Verbindung notwendig.

4.2.3.9. /root/scripts/wlan0_up.sh

WLAN einschalten (alte version)

4.2.3.10. /root/eurobot/build.sh

Kompliert den Code in /root/eurobot/Eurobot.

4.2.3.11. /root/eurobot/getAndBuild.sh

Führt getBuild.sh und build.sh aus.

4.2.3.12. /root/eurobot/getBuild.sh

Löscht /root/eurobot/Eurobot und lädt das Verzeichnis neu aus der Versionsverwaltung auf <https://svn.itds.ch/> herunter.

4.2.3.13. /root/eurobot/run.sh

Führ die als erstes Argument angegebene Java-Klasse aus. Als zweites Argument kann optional der Devicenamen des CAN-Bus angegeben werden. Der Rückgabewert vom der ausgeführten Java-Klasse wird vom Skript als Rückgabewert weitergegeben.

4.2.3.14. /root/eurobot/sayChangeBattery.sh

Führt die Java-Klasse SayChangeBattery im eurobot.test-Paket aus, welche per Text-To-Speech sagt, dass die Batterie/der Akku gewechselt werden solle.

4.2.3.15. /root/eurobot/sayNetworkConfigChanged.sh

Äquivalent zum Skript oben lässt dieses Skript den Roboter sagen, dass die Netzwerkkonfiguration geändert worden ist.

4.2.3.16. /root/eurobot/updateAndBuild.sh

Aktualisiert den Sourcecode in Eurobot2 mit dem Repository von <https://svn.itds.ch> und kompiliert den Code. Anschliessend werden die Bytecodedateien nach Eurobot kopiert.

4.2.3.17. /root/eurobot/updateAndBuildVaio.sh

Arbeitet wie das oben genannte Skript, verwendet aber das Repository von <http://svn.bittnet>. Dieses Repository wurde während der Wettkämpfe auf dem Sony Vaio Notebook inkl. DHCP/DNS-Server installiert.

4.2.3.18. /root/eurobot/scripts/

Hier befinden sich diverse Skripte welche mit dem Can-Bus von der Konsole aus kommunizieren.

4.2.4. Bedienungsanleitung für den Roboter

4.2.4.1. System starten

- 36V Speisung anschliessen (Akku)
- Navigationssystem einschalten (Baken, 9V Speisung)
- 24V Speisung anschliessen (Akku/Netzgerät)
- Kontrolle ob Lüfter arbeiten (CPU,Motorentreiber)
- Servoboard über Kippschalter aus- und wieder einschalten
- Beide Atmel-Boards mittels roter Reset-taste neu starten
- Sound + Lauflicht über Kippschalter aktivieren :-)

4.2.4.2. Einloggen

Per ssh (unter Windows mit putty.exe) mit dem Roboter verbinden. Der Servername ist eurobot.kicks-ass.org (dynamisch) oder 192.168.3.2 (statisch). Der Benutzername lautet root und das Passwort ist eurobot.

4.2.4.3. Anwendung starten

- Trommel auf Einladeposition bringen
- Notstop kontrollieren
- in der Shell ./commandPrompt.sh starten
- Nötigenfalls mit der Taste 2 oder 3 ein Softwareupdate einspielen
- Programm mit Taste 4-7 starten

4.2.4.4. Fehlerquellen

Navigationsfehler

Position wird nicht erkannt, Roboter dreht im Kreis, Gegner wird nicht erkannt
 → Navibaken richtig aufgestellt?
 → Kanal/Frequenzeinstellungen prüfen
 → Speisung prüfen

CAN-Bus Error

Java startet dauernd neu, Roboter bleibt stehen, Befehle werden nicht ausgeführt
 → Notaus kontrollieren
 → Kontrollieren ob Sensorboard blinkt, wenn nicht: reset.

4.2.5. Konfiguration Entwicklungsumgebung

4.2.5.1. Quellcodeverwaltung

Meist arbeiten mehrere Team-Mitglieder am selben Source-Code. Deshalb ist die Verwendung eines Quellcodeverwaltungssystems wie SVN unerlässlich. In der IDE Eclipse können wir mittels **Subclipse**³ den Quellcode einfach aktuell halten und Konflikte verhindern.
 Auf dem Industrie-Rechner automatisiert ein shell-Skript das Aktualisieren und Kompilieren des Quellcodes. Im Wesentlichen ist dabei folgender Befehl von Bedeutung:

```
svn export https://svn.itds.ch/svn/eurobot/Eurobot Eurobot
```

4.2.6. IDE

Als Integrierte Entwicklungsumgebung (IDE) setzen wir **Eclipse Europa**⁴ ein. Diese Umgebung eignet sich besonders gut für Java-Quelltext und ist sowohl auf Windows als auch unter Linux lauffähig.

³<http://subclipse.tigris.org>

⁴<http://www.eclipse.org>

4.3. Java-Software

4.3.1. Konzept

Unser Konzept versucht die Funktionalität des Roboters in verschiedene Java-Pakete auszulagern. Dies ermöglicht den höchsten Grad an Abstrahierung und lässt uns somit die Vorteile der objektorientierten Programmierung (Kapselung, Polymorphie, Vererbung) voll ausnutzen.

Die Software ist im Kapitel UML Dokumentation (5) im Detail beschrieben.

5. UML Dokumentation

5.1. Übersicht / Komponenten

Die einzelnen Komponenten des Roboters B.I.T.T. des Teams MarsRiders wurde in Java als Pakete realisiert. Beispielsweise existiert für die gesamte Abstraktion der Hardware ein eigenes Paket namens `eurobot.hw` (5.4.5). Im Folgenden soll kurz erläutert werden, was die Aufgabe der einzelnen Komponenten (beziehungsweise Pakete) ist. Eine detailliertere Beschreibung der Funktionalität kann unter 5.4 gefunden werden. Eine Übersicht der Komponenten bietet das Diagramm 5.1.

Komponente	Schnittstelle	Beschreibung
<code>eurobot</code>	<code>Eurobot/Run</code>	Stellt den Roboter dar. Die einzelnen Komponenten werden hier vereint. Von hier aus kann auf die gesamte Funktionalität des Roboters zugegriffen werden. Enthält zusätzlich Klassen zum Starten des Roboters und zur Aktualisierung der Umgebungsbedingungen.
<code>eurobot.analyse</code>	keine	Ermöglicht die nachträgliche Analyse eines Spiels. Die Fahrt kann auf dem virtuellen Spielfeld nachverfolgt werden.
<code>eurobot.can</code>	<code>PCANSupport</code>	Ermöglicht die Kommunikation mit der CAN-Schnittstelle.
<code>eurobot.field</code>	diverse	Modelliert das Spielfeld und seine Objekte wie Dispenser und Gegner.
<code>eurobot.hw</code>	<code>Device</code>	Jeder Aktor wird in diesem Paket abstrahiert.
<code>eurobot.hw.sensor</code>	<code>Sensor</code>	Jeder Sensor ist ein Gerät und wird demzufolge der <code>eurobot.hw</code> (5.4.5) Komponente untergeordnet.
<code>eurobot.imp</code>	<code>BallDetection</code>	Diese Komponente beschäftigt sich mit der Bildverarbeitung und stellt diverse Tools zur Verfügung. Die konkrete Anwendung findet sich in der Schnittstellenklasse <code>eurobot.imp.BallDetection</code> (5.4.7.3).
<code>eurobot.log</code>	<code>Log</code>	Alle wichtigen und unwichtigen Informationen werden mit Hilfe dieses Pakets geloggt. Dabei stehen verschiedene Senken zur Verfügung (Datenbank, File-System, Konsole, ...).
<code>eurobot.move</code>	<code>Drive</code>	Übernimmt die Ansteuerung der Motoren und übersetzt abstrakte Befehle (drehen um Winkel, ...) in konkrete Fahrbefehle, die der <code>eurobot.hw</code> (5.4.5) Komponente weitergegeben werden.
<code>eurobot.move.exception</code>	diverse	Stellt Fehlermeldungen für Bewegungen zur Verfügung.
<code>eurobot.move.odometrie</code>	<code>OdometrieDrive</code>	Ermöglicht die odometrische Navigation. Arbeitet eng mit <code>eurobot.move</code> (5.4.8) zusammen.
<code>eurobot.move.place</code>	<code>Place</code>	Abstrahiert die Plätze auf dem Spieltisch und stellt verschiedene Informationen über sie zur Verfügung.

eurobot.play	Strategy	Als Kern der Anwendung wird hier der Ablauf des Spielverlaufs beeinflusst.
eurobot.remote	Server/Client	Ermöglicht die Kommunikation mit der Aussenwelt über ein Netzwerk. Kann dazu verwendet werden, um Statusinformationen oder Steuerinformationen zu übertragen.
eurobot.remote.gui	keine	Stellt den aktuellen Status des Roboters dar. Kann einige Befehle auf dem Roboter ausführen.
eurobot.rs232	RS232Support	Ermöglicht die Kommunikation mit der RS232 Schnittstelle. Wird zudem zur Kommunikation mit dem Ultraschall Navigationsgerät verwendet.
eurobot.test	keine	Dient zum Ausprobieren der einzelnen Komponenten.

Tabelle 5.1.: Übersicht der Komponenten

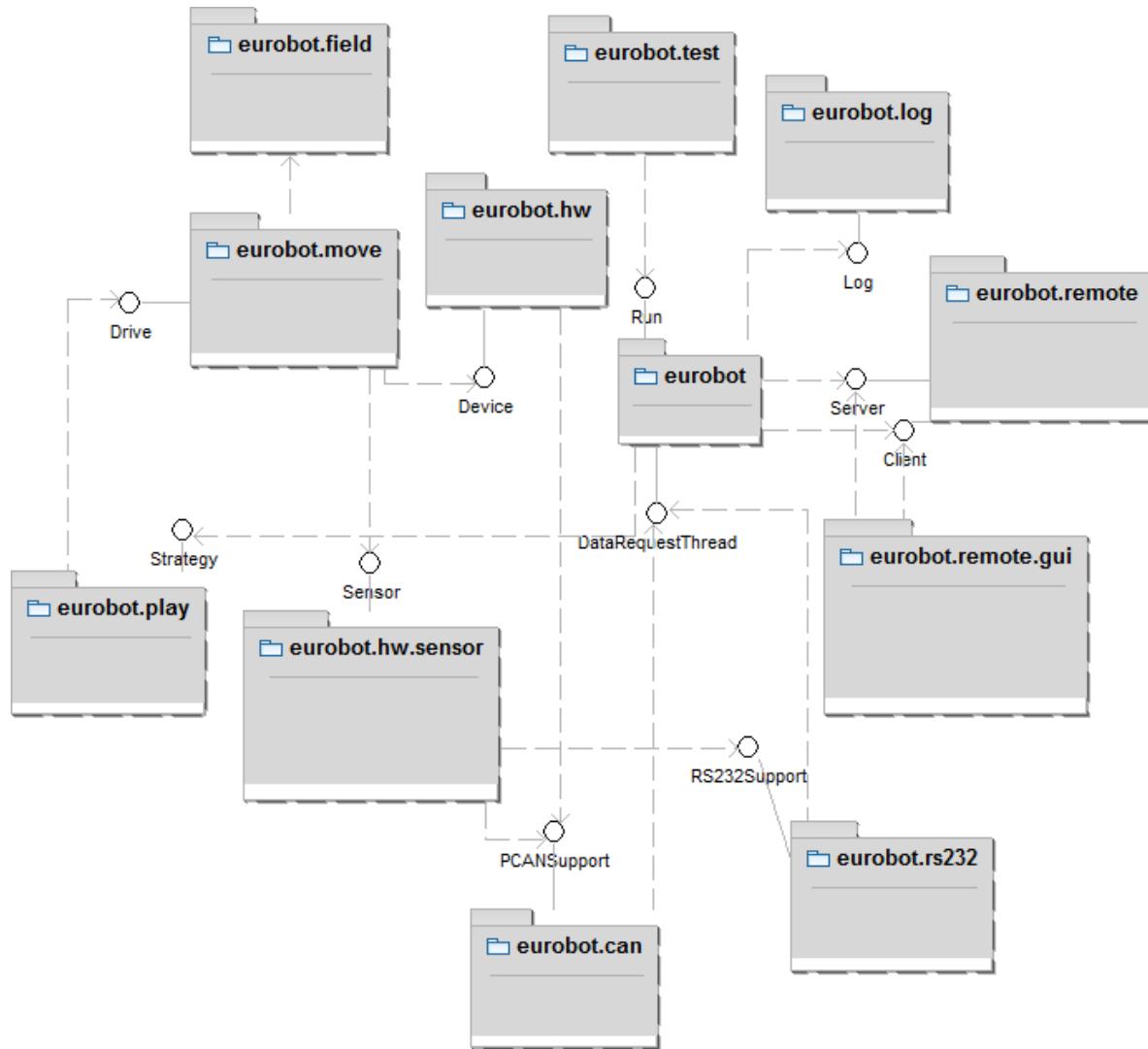


Abbildung 5.1.: Komponentendiagramm

5.2. Use-Case

Hier folgen die Use-Case Diagramme mit Beschreibung.

5.2.1. Wettbewerb gewinnen

Fall Nr. 1	„Wettbewerb gewinnen“
Beschreibung:	Der Roboter startet das Spiel, erfasst laufend die aktuelle Situation und versucht dabei Dispenser anzufahren und Bälle daraus zu entnehmen. Danach wird zur Ausladeposition gefahren und die Bälle ausgeworfen. Nach Ende der Spielzeit stoppt der Roboter.
Vorbedingungen:	Alle Systeme sind eingeschaltet und funktionieren einwandfrei. NotAus ist nicht gedrückt. Navigationsbaken sind auf dem korrekten Kanal.
Beschreibung des Ablaufs	
E1)	Der Anwender startet das Programm
A1)	Der Roboter überprüft die aktuelle Situation (gewählte Farbe) und initialisiert seine Systeme
E2)	Der Anwender gibt das Startsignal
A2)	Der Roboter erfasst laufend seine aktuelle Position und fährt je nach gewählter eine Position an. Zu Fall 2 wechseln
E3)	Motoren drehen
A3)	Position wird aktualisiert. Roboter erhält eine Nachricht, wenn die gewünschte Anzahl Schritte gefahren wurde
E4)	Kollision
A4)	Ausweichen
A4A1)	Warten bis Objekt wegfährt
E5)	Position erreicht
A5)	Wenn wir vor einem Dispenser stehen, dann Bälle einladen. Zu Fall 3 wechseln
A5A1)	Wenn wir an der Ausladeposition stehen, dann Bälle ausladen
E6)	Timer ist abgelaufen
A6)	Alle Aktoren werden gestoppt
E6A1)	Notstop wird gedrückt
A6A1)	Speisungen unterbrechen. Zurück zu A4 wechseln

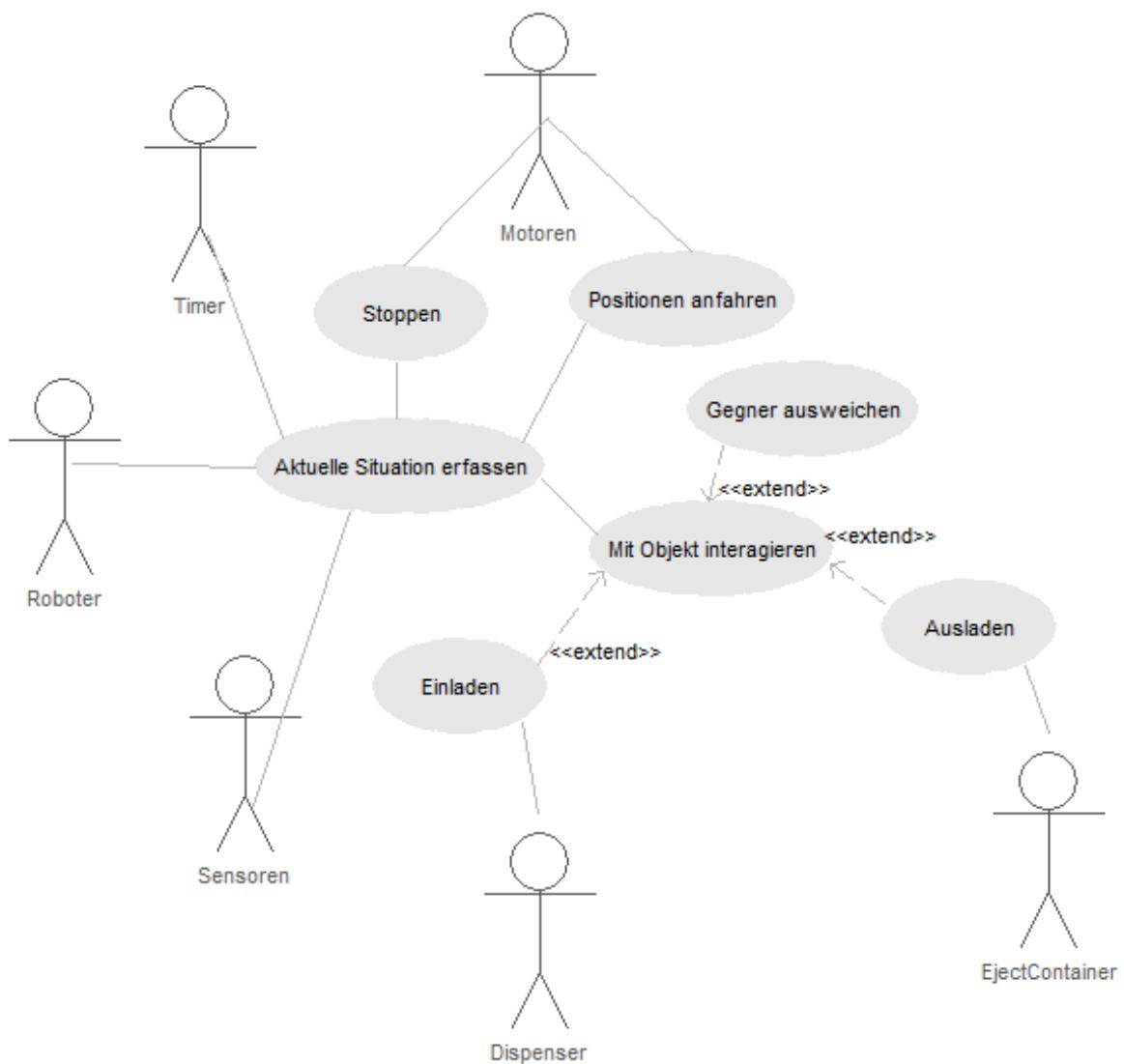


Abbildung 5.2.: Anwendungsfall 1: Wettbewerb gewinnen

5.2.2. Position anfahren

Fall Nr. 2	„Position anfahren“
Beschreibung:	Der Roboter erhält von der Strategie den Befehl, ein Objekt anzufahren. Es werden Fahrbefehle an die Motoren geschickt.
Vorbedingungen:	Speisung vorhanden. Kein Objekt im Weg.
Beschreibung des Ablaufs	
E1)	Die Strategie möchte einen Dispenser anfahren
A1)	Die Fahrbefehle werden vorbereitet
E1A1)	Der Roboter möchte den Eject-Container anfahren
A1A1)	Zu A1
E2)	Der Anwender gibt das Startsignal
A2)	Der Roboter gibt den ersten Fahrbefehl an die Motoren
E3)	Motoren drehen
A3)	Position wird aktualisiert. Roboter erhält eine Nachricht, wenn die gewünschte Anzahl Schritte gefahren wurde
E4)	Kollision
A4)	Ausweichen
A4A1)	Warten bis Objekt wegfährt
E5)	Position erreicht
A5)	Wenn wir vor einem Dispenser stehen, dann Bälle einladen. Zu Fall 3 wechseln
A5A1)	Wenn wir an der Ausladeposition stehen, dann Bälle ausladen
E6)	Timer ist abgelaufen
A6)	Alle Aktoren werden gestoppt
E6A1)	Notstopp wird gedrückt
A6A1)	Speisungen unterbrechen. Zurück zu A6 wechseln

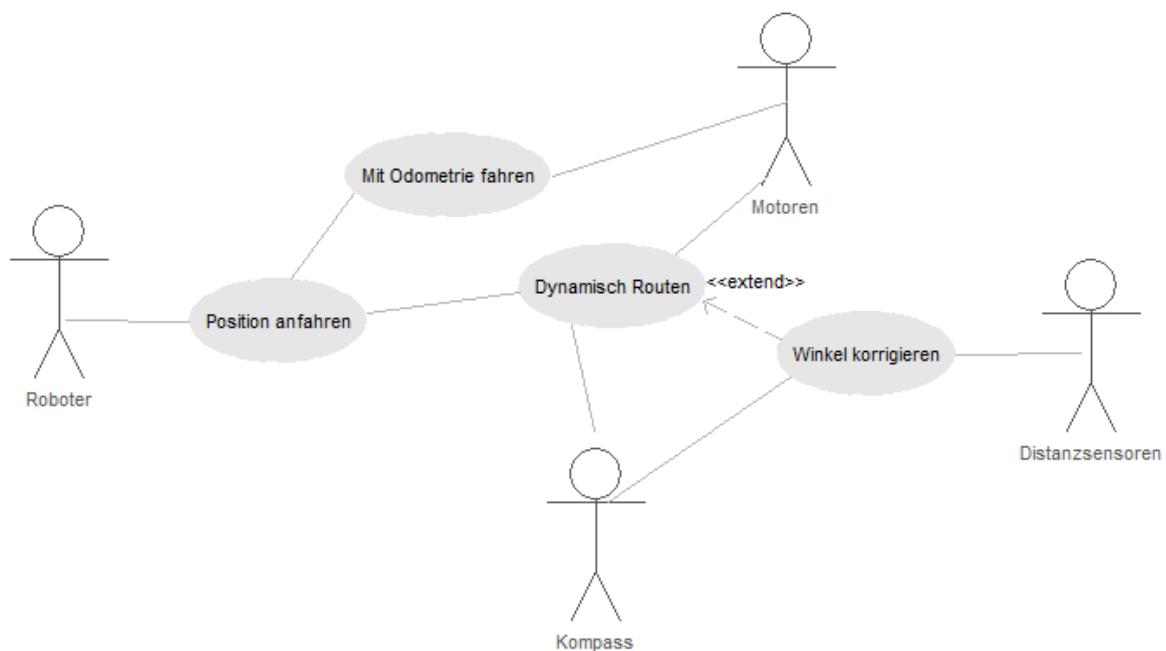


Abbildung 5.3.: Anwendungsfall 2: Position anfahren

5.2.3. Bälle einladen

Fall Nr. 3 „Bälle einladen“

Beschreibung:	Der Roboter erhält den Befehl einen Ball einzuziehen. Dabei schliesst er die Bänder, dreht die Trommel, öffnet die Klappe und schaltet die Motoren ein.
Vorbedingungen:	24V Speisung vorhanden.
Beschreibung des Ablaufs	
E1)	Die Hauptsteuerung fordert eine Ballaufnahme
A1)	Die Trommelsteuerung überprüft den Status der Trommel und deren Position und liefert diese Informationen zurück
E2)	Die Hauptsteuerung will auch unter Einbezug der neuen Informationen einen Ball aufnehmen
A2)	Die Aktoren werden per CAN initialisiert. Trommel drehen. Bänder öffnen. Klappe öffnen.
A2A1)	Die Aktoren sind bereits initialisiert. Zu E3
E3)	Alle Systeme sind initialisiert. Der Speicherplatz für den Ball ist nicht belegt
A3)	Bänder einschalten. Bänder schliessen. Der Ball wird eingezogen
E4)	Die Hauptsteuerung wartet auf das Ende des Vorgangs.
A4)	Per CAN erhalten wir eine Status-Nachricht, wenn die Trommel gedreht hat. Nun zu A1 wechseln

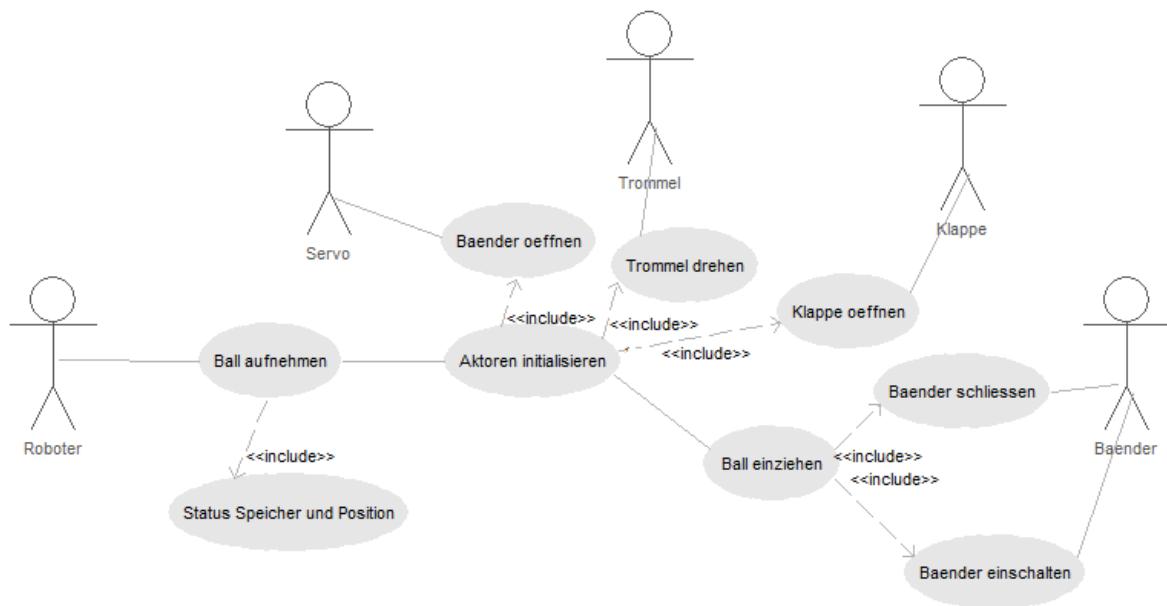


Abbildung 5.4.: Anwendungsfall 3: Ball aufnehmen

5.2.4. Gegner ausweichen

Fall Nr. 4

„Gegner ausweichen“

Beschreibung:	Der Roboter entdeckt über eines der Navigationssysteme ein Hindernis. Das Hindernis soll umfahren werden. Bei einem Zusammenstoss hält der Roboter an und weicht aus.
Vorbedingungen:	Navigationssysteme bereit. Fahrtgeschwindigkeit tief.
Beschreibung des Ablaufs	
E1)	Gegner steht mitten in der Fahrtroute des Roboters
A1)	Unsere und des Gegners Position wird vom Ultraschall-Navigationsgerät erfasst und verglichen. Die Fahrtroute wird neu berechnet, so dass der Gegner umfahren werden sollte
E2)	Der Roboter kollidiert mit dem Gegner
A2)	Die Motoren werden gestoppt. Über das Gyroskop wird die Winkelveränderung festgestellt und beide Positionen (gegnerische und die des Roboters) abgeglichen.
E3)	Der Gegner bewegt sich nach der Kollision von uns weg
A3)	Zu A1)
E3A1)	Der Gegner bleibt stehen oder kollidiert erneut
A3A1)	Günstigstes Ausweichmanöver fahren. Vorwärts oder rückwärts flüchten. Unter Umständen zuerst drehen.
E4)	Das Ziel wurde erreicht
A4)	Zu Fall 3 wechseln oder „Bälle ausladen“

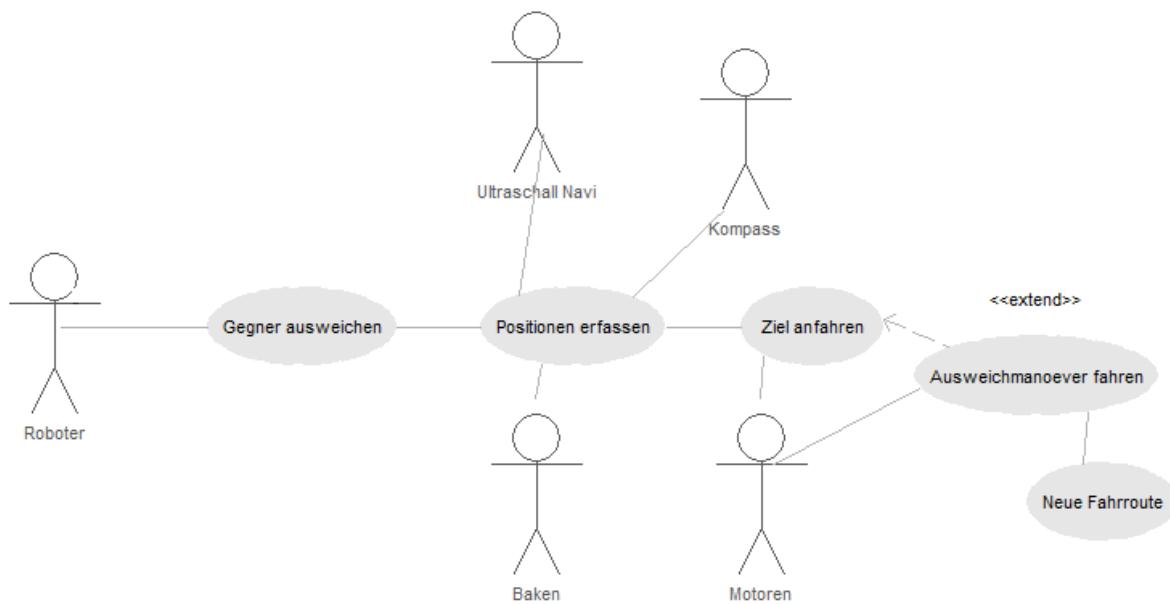


Abbildung 5.5.: Anwendungsfall 4: Gegner ausweichen

5.2.5. Roboter kontrollieren

Fall Nr. 5 „Roboter kontrollieren“

Beschreibung:	Der Roboter soll drahtlos per graphischer Oberfläche kontrollierbar sein. Dabei kann er gestartet, gestoppt und bewegt werden. Statusinformationen müssen angezeigt werden.
Vorbedingungen:	Netzwerkverbindung vorhanden.
Beschreibung des Ablaufs	
E1)	Die Java Software wird im autonomen Modus gestartet
E1A1)	Die Java Software wird im manuellen Modus gestartet
A1)	Alle nötigen Netzwerkverbindungen werden vorbereitet. Der Server wird initialisiert. Eine akustische Rückmeldung meldet den Erfolg
A1A1)	Wie A1)
E2)	Der Techniker verbindet sich per Netzwerk mit dem Roboter
A2)	Aktuelle Informationen und das Spielfeld werden auf dem GUI angezeigt
E3)	Der Roboter wird per Startschur gestartet
E3A1)	Der Roboter wird per Startknopf im GUI gestartet
A3)	Autonomer Modus: Fahrbefehle werden vorbereitet. Siehe Fall 1
A3A1)	Manueller Modus: Nichts geschieht
E4)	Ein Zielpunkt wird per GUI an den Roboter geschickt
A4)	Autonomer Modus: Verhalten undefiniert, da evtl. Fahrbefehle ausgeführt werden. In einem weiteren Schritt sollte die Anfrage ignoriert werden
A4A1)	Manueller Modus: Zu Fall 2

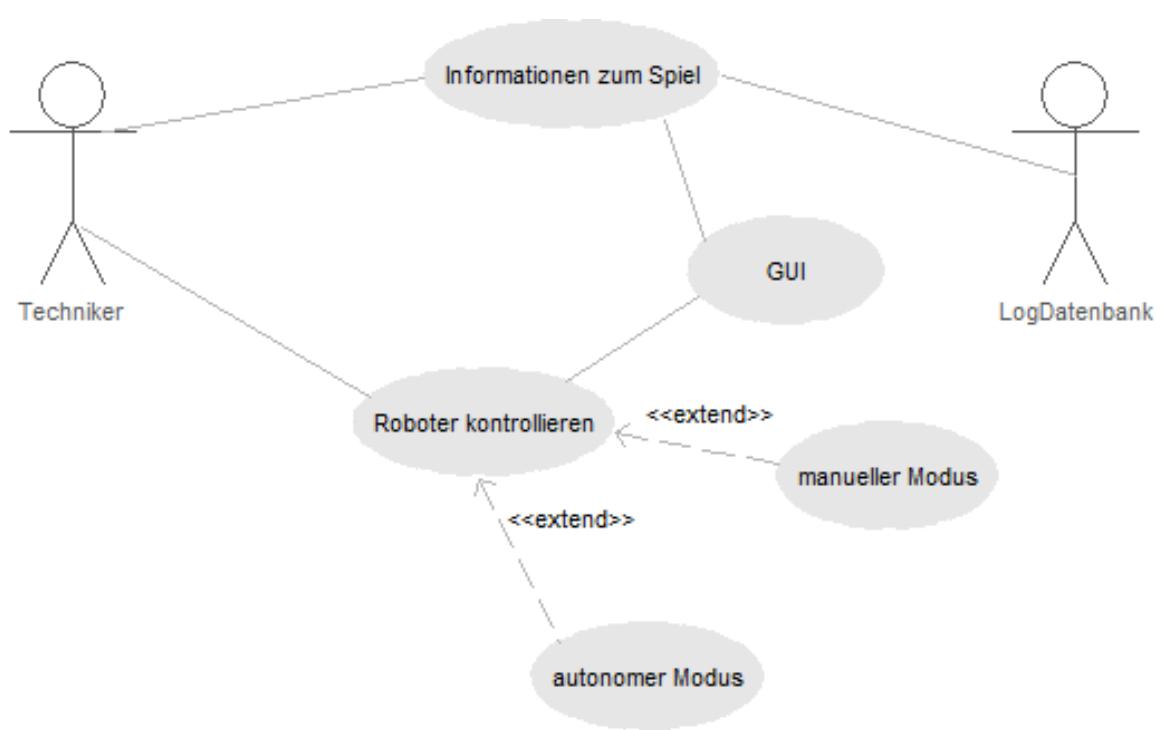


Abbildung 5.6.: Anwendungsfall 5: Roboter kontrollieren

5.3. Sequenzdiagramme und Beschreibungen

5.3.1. CAN-Bus

Im Sequenzdiagramm zum CAN-Bus (Bild 5.7) wird neben dem Initialisieren des PCANSupport auch das Senden und Empfangen von Nachrichten gezeigt.

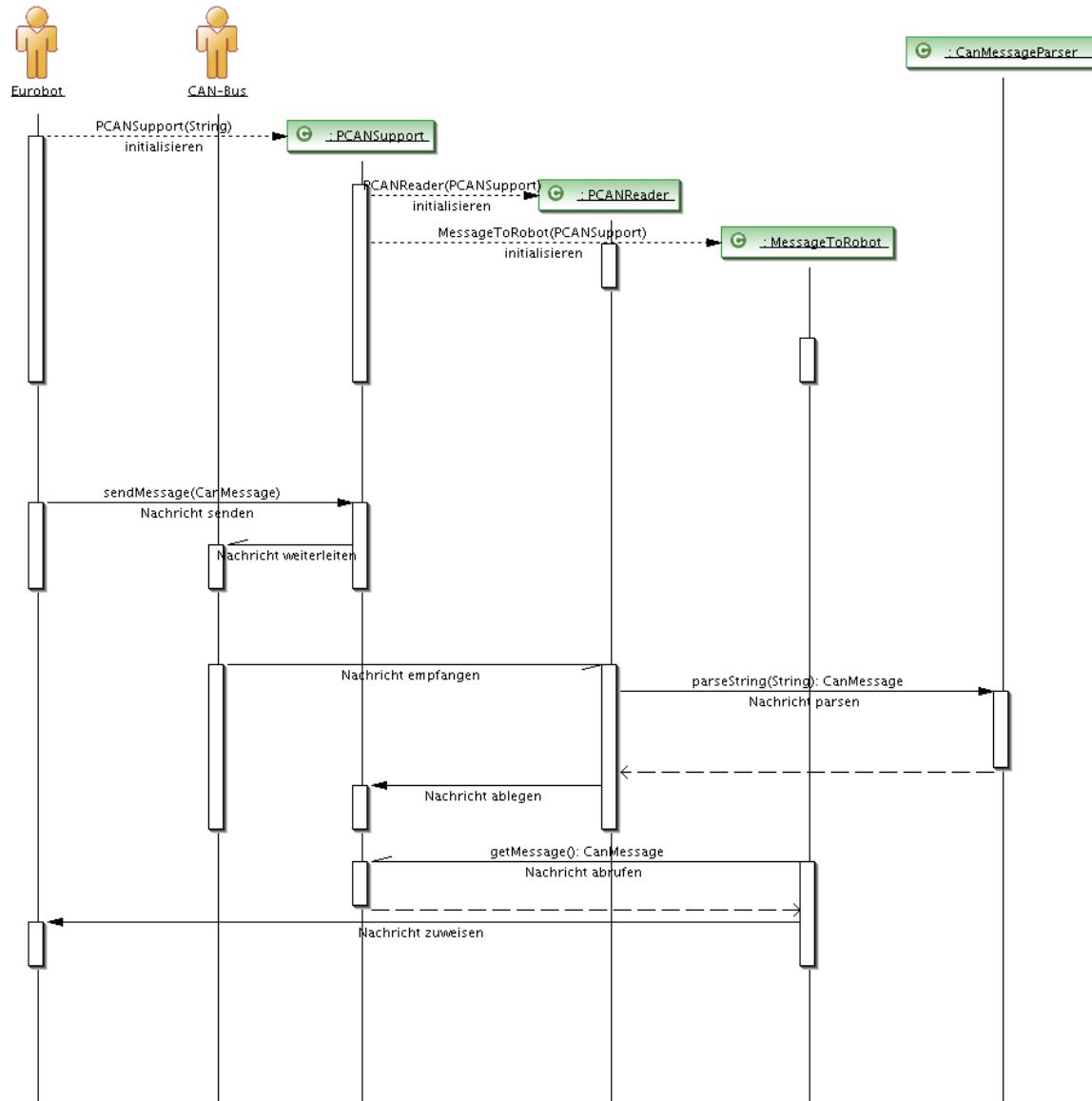


Abbildung 5.7.: Sequenzdiagramm CAN-Bus

5.3.1.1. Initialisieren

Zu Beginn erstellt das Eurobot Objekt ein PCANSupport (5.4.3.8) Objekt. Dabei wird dem Konstruktor als Attribut den Namen des PCAN-Devices übergeben. Das PCANSupport Ob-

jekt erstellt danach ein PCANReader Objekt welches anschliessend einen Thread startet, welcher das PCAN-Device überwacht und eingehende Nachrichten entgegen nimmt.

Weiter wird ein MessageToRobot Objekt erstellt, welches selber einen Thread startet, der die inputMessageQueue des PCANSupport überwacht.

5.3.1.2. Nachricht senden

Nachrichten, welche gesendet werden, gelangen an die PCANSupport Methode sendMessage(CanMessage). Diese legt die Nachricht in der outputMessageQueue ab. Die outputMessageQueue wird vom PCANSupport-Thread periodisch abgearbeitet. Dabei werden die Nachrichtenobjekte in Strings umgewandelt und in das PCAN-Device geschrieben. Sofern sich die Motoren im Schritt-Modus befinden, werden dynamische Fahrbefehle nicht gesendet, sondern verworfen. Weiter werden doppelte Kompasslesebefehle verworfen und dynamische Motorenfahrbefehle überschreiben alte Befehle, welche sich nicht in der outputMessageQueue befinden.

5.3.1.3. Nachricht empfangen

Wie bereits angetönt, überwacht das PCANReader (5.4.3.7) Objekt das PCAN-Device. Bei eingehenden Nachrichten, vom PCAN-Device, werden die Nachrichten-Strings mit der Methode CanMessageParser.parseString(String) in ein CanMessage Objekt umgewandelt. Dieses Objekt wird beim erfolgreichen Umwandeln anschliessend in der inputMessageQueue des PCANSupport abgelegt.

Die inputMessageQueue wird vom MessageToRobot Thread periodisch überwacht. Bei neuen Nachrichten verarbeitet der Thread diese. So wird ein ankommender neuer Kompasswert dem DirectionSensor Objekt des Roboters zugewiesen.

5.3.2. Routing

Wenn der Roboter dynamisch mit Routing (5.4.18) fährt, dann wird mit der Methode RoutingHelper.getPath() gearbeitet. Darin befindet sich eine Schleife, welche erst beim Erreichen des Zielpunktes verlassen wird.

In dieser Schleife wird zuerst mit dem schnellen und zuverlässigen Matrixrouting mit der Methode Routing.findPathByMatrixRouting(Point,Point) gerechnet. Diese rechnet die Koordinaten in ein eigenes System um. Dabei wird die Präzision reduziert, so dass die zu berechnende Matrix kleiner wird. Nach der Pfadberechnung wird dieser erste Pfad optimiert. Nach der Optimierung werden die Koordinaten dieses Pfades in das mm-Koordinatennetz des Roboters umgerechnet.

Wenn der Pfad des Matrixroutings aus mehr als zwei Punkten besteht, wird davon ausgegangen, dass ein Pfad um den Mitbewerber herum errechnet wurde. In diesem Fall wird mit der Methode Routing.findPathByTangentRouting(Point,Point) versucht einen für die Odometrie optimierten Pfad zu errechnen. Wenn dies nicht gelingt, wird mit dem Pfad des Matrixroutings weitergerechnet.

Sofern das Matrixrouting fehlgeschlagen ist, wird ein weiterer Routingversuch mit dem alternativen Vektorrouting in der Methode Routing.findPathByVektorRouting(Point,Point)

unternommen. Diese Methode versucht rekursiv den Weg zu finden, indem zuerst die möglichen nächsten Punkte bestimmt werden und anschliessend der Beste gewählt wird. Vom neuen Punkt aus wird dann die Fortsetzung des Weges gesucht.

Sofern beide Routingmethoden erfolglos blieben, wird nach mehreren aufeinanderfolgenden Fehlern versucht vom Hindernis weg zu fahren. Dabei wird kontrolliert ob bei den Routingfehlern der Methoden ein Kollisionspunkt übergeben wurde. In diesem Fall fährt der Roboter einige Schritte in entgegengesetzter Richtung von diesem Punkt weg.

Sofern mit den Routingmethoden erfolgreich ein Pfad festgelegt werden konnte, werden die Motorenengeschwindigkeiten dementsprechend berechnet und den Motoren zugewiesen oder aus dem Pfad Odometriebefehle (Command (5.4.10.2)) erstellt, welche danach abgearbeitet werden.

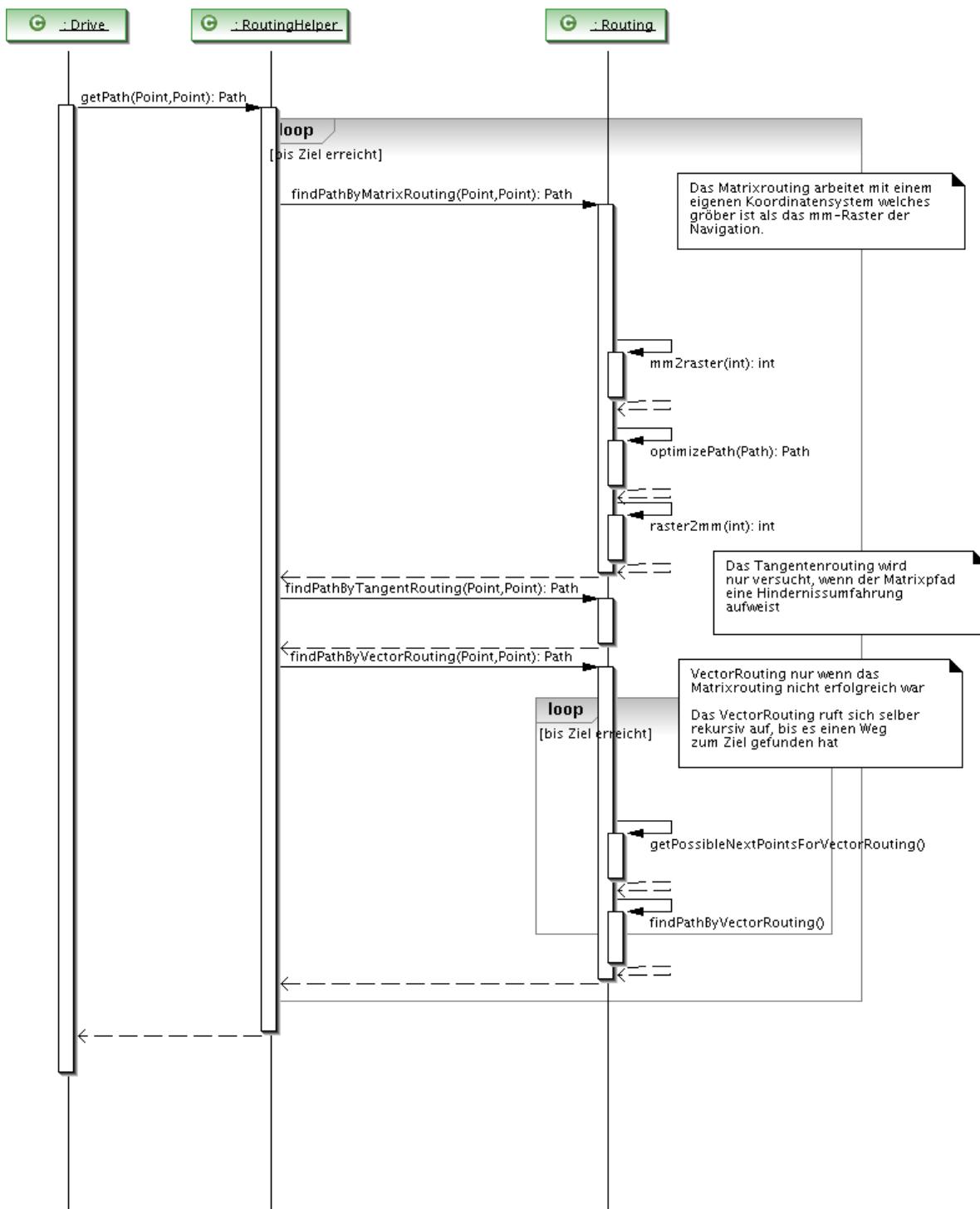


Abbildung 5.8.: Sequenzdiagramm Routing

5.3.3. BatchStrategy

Die BatchStrategy (siehe Bild 5.9) arbeitet der Reihe nach einen Befehl nach dem anderen ab.

Zu Beginn wird auf den Spielstart gewartet, anschliessend wird der Timer (TimeOut (5.4.1.7)) gestartet und die der Spielerfarbe entsprechenden Dispenser- und EjectContainer-Positionen werden festgelegt.

Anschliessend nimmt der Roboter an den Dispensern die Bälle auf und gibt sie darauf folgend beim EjectContainer aus.

Danach werden die Einstellungen so geändert, dass nun 3 farbige und 2 weisse Bälle aufgenommen werden sollten.

Darauf hin werden wieder Bälle aufgenommen (nicht mehr im Bild). Die Ballanzahl wird aber je nach verbleibendem Zeitbudget reduziert.

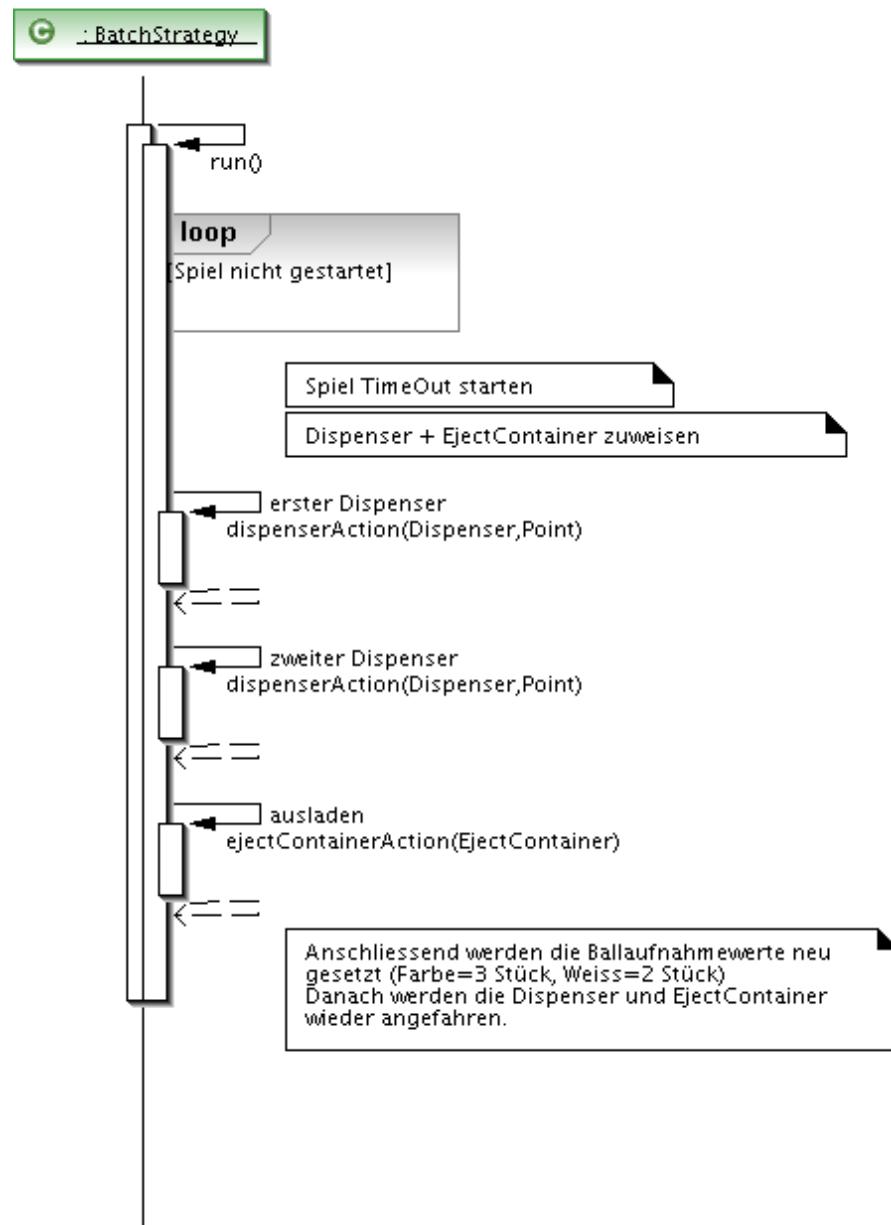


Abbildung 5.9.: Sequenzdiagramm BatchStrategy

5.3.4. BatchStrategy.dispenserAction()

Die `BatchStrategy.dispenserAction()` Methode beinhaltet das Anfahren des Dispensers und die dortige Ballaufnahme (Bild 5.10).

Als erstes wird der Bandeinzug ganz geöffnet, so dass bei Kollisionen während der Fahrt weniger Schaden entstehen kann und dass direkt an den Dispenser heran gefahren werden kann.

Nach dem Öffnen fährt der Roboter mit der Methode `moveToObjectNew()` der `AimTarget` Klasse zum Dispenser.

Am Dispenser angekommen nimmt der Roboter die gewünschte Anzahl Bälle auf indem die nun folgenden Schritte wiederholt werden:

- **Storage.turnToFreePos()**: Der Trommel wird der Befehl gegeben zu einer freien Position zu drehen.
- **LidServo.up()**: Die Klappe vor der Trommel wird geöffnet.
- **Band.close()**: Der Bandeinzug wird geschlossen. Dabei wird der unterste Ball vom Dispenser eingeklemmt.
- **Drum.waitWhileTurning()**: Es wird gewartet bis die Trommel still steht und somit für einen Ball aufnahmefähig ist.
- **Band.collect()**: Der Bandeinzug wird gestartet, so dass der eingeklemmte Ball in die freie Trommelposition gelegt wird.
- **Band.stop()**: Anschliessend wird der Bandeinzug angehalten.
- **Band.open()**: Damit der nächste Ball vom Dispenser nachrutschen kann wird der Befehl zum Öffnen des Bandeinzuges gegeben.
- **Storage.addBall()**: Dem Speicher wird mitgeteilt, dass ein weiterer Ball aufgenommen wurde.

Nach dem letzten Ball wird die Klappe vor der Trommel geschlossen und der Roboter fährt ein Stück vom Dispenser weg.

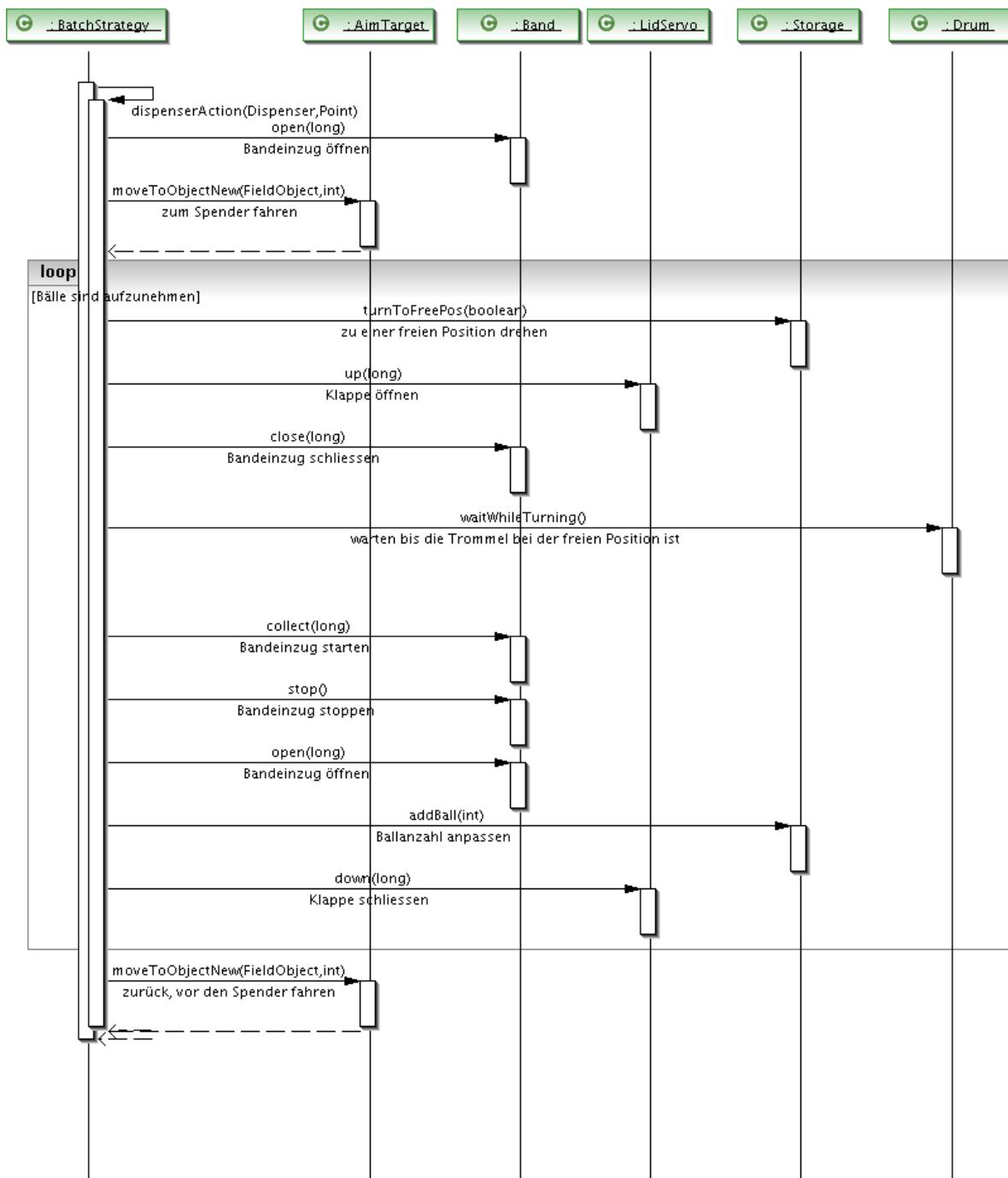


Abbildung 5.10.: Sequenzdiagramm BatchStrategy.dispenserAction()

Die `ejectContainerAction()` wird nicht explizit beschrieben, da diese eine vergleichbar mit der `dispenserAction()` arbeitet.

5.4. Klassendiagramme

5.4.1. eurobot

5.4.1.1. eurobot.CmdLineParser

Ermöglicht den einfachen Umgang mit Kommandozeilenoptionen. Diese Klasse wird von eurobot.Run (5.4.1.6) verwendet um die Startoptionen zu bestimmen.

5.4.1.2. eurobot.DataRequestThread

Sendet laufend und vom Spielzustand abhängig Daten an RS232 und CAN (5.4.17.1).

5.4.1.3. eurobot.Eurobot

Von dieser Klasse darf und kann nur ein Objekt erstellt werden. Sie beinhaltet den Kern des Roboters. Über die entsprechenden Attribute und Methoden kann von hier aus auf alle wichtigen Objekte (Motoren, Routing,...) zugegriffen werden. Mit Eurobot.getInstance() kann nach dem Initialisieren auf das Objekt zugegriffen werden.

5.4.1.4. eurobot.EurobotVoice

In dieser Klasse befinden sich die Methoden für die Text-to-Speech Funktionalität.

5.4.1.5. eurobot.IEurobot

Dieses Interface definiert Methoden für die Fernsteuerung via Remote-GUI

5.4.1.6. eurobot.Run

Im Normalbetrieb wird der Roboter über diese Klasse gestartet. Ein Objekt dieses Klasse liest mittels CmdLineParser (5.4.1.1) die Kommandozeilenargumente ein und konfiguriert und initialisiert den Roboter entsprechend. Danach wird auf das Startsignal gewartet.

5.4.1.7. eurobot.TimeOut

Diese Klasse beinhaltet einen Timer welcher nach Ablauf der Zeit einen Software-Notstop ausführt und den Roboter anhält. Die Strategie eurobot.play.BatchStrategy (5.4.12.1) kann den aktuellen Timerstand abfragen um zu entscheiden wie gehandelt werden soll.

5.4.1.8. eurobot.VirtualBot

Der VirtualBot soll den Roboter und sein Verhalten nachbilden. Insbesondere werden hier die Motoren simuliert, damit auch ohne „echte“ Hardware herumgefahren werden kann. Diese Klasse wurde nur zu Beginn der Entwicklung betreut, so dass Sie mit dem aktuellen Stand der Software sehr wahrscheinlich nicht mehr korrekt arbeitet.

5.4.2. eurobot.analyse

Das Analyse-GUI wurde programmiert um die unübersichtliche Log-Datenbank zu Visualisieren. Das GUI wurde so weit programmiert, dass die gegnerische und die eigene Route über ein Spiel angezeigt werden kann. Ein ausgewähltes Spiel kann vorwärts und rückwärts durchgestepppt werden. Ein automatischer Vorlauf ist ebenfalls implementiert. Das GUI könnte erweitert werden mit dem Anzeigen detaillierten RobotermodeLLs. Die Darstellung von Trommel, Bändern und Klappen ist vorgesehen. Zurzeit werden die abgespeicherten Log Informationen rechts neben dem Spielfeld ohne Veränderung angezeigt.

Das GUI wurde nicht weiterentwickelt, da wir das Logen von Spielen abschalten mussten, weil die Solid State Disk eine sehr langsame Schreibgeschwindigkeit hat.

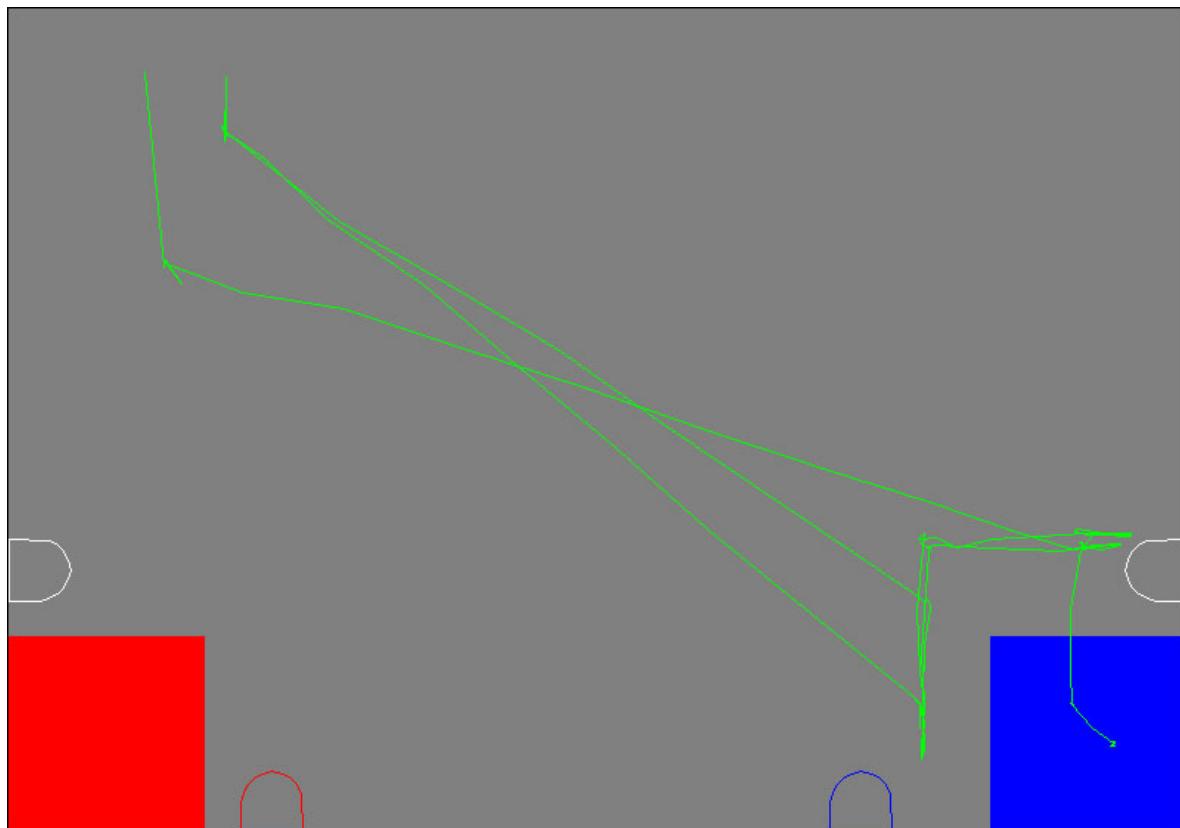


Abbildung 5.11.: Analyse GUI gegnerischer Pfad

5.4.2.1. eurobot.analyse.AddList

AddList stellt uns eine Liste zur Verfügung. Die Liste ist gedacht um die Details während einem Spiel anzuzeigen. Unterhalb der List wird das RobotermodeLl anzeigen.

5.4.2.2. eurobot.analyse.AnalyseGui

Von hier aus wird das GUI zum Auswerten unserer Spielstrategie gestartet.

5.4.2.3. eurobot.analyse.ButtonsPanel

Das ButtonsPanel stellt die Buttons zum Steuern zur Verfügung. Die Buttons erlauben das Navigieren durch die Zeitachse des gewählten Spiels. Mittels eines Timers werden die Spieldaten automatisch alle 250ms geladen. Somit ist ein automatisches Betrachten des Spielablaufs möglich.

5.4.2.4. eurobot.analyse.ControlPanel

Klasse die den Navigation-Slider zur Verfügung stellt.

5.4.2.5. eurobot.analyse.Field

Die Klasse Field zeichnet das Spielfeld.

5.4.2.6. eurobot.analyse.GameDetails

In dieser Klasse werden alle Koordinaten und nützliche Eigenschaften eines Spiels gespeichert.

5.4.2.7. eurobot.analyse.MessageList

Diese Klasse dient zum Speichern der Nachrichten von der Datenbank. Dadurch ist eine Darstellung und Auswertung der Nachricht einfacher.

5.4.2.8. eurobot.analyse.RobotModel

Teile des Roboters werden mit dieser Klasse dargestellt.

5.4.2.9. eurobot.analyse.RoutePoint

Die Klasse dient als Struktur um Koordinaten der Roboter und ihre Zeit zu speichern.

5.4.2.10. eurobot.analyse.SelectGamePabel

Klasse zum Laden des gewünschten Spiels.

5.4.3. eurobot.can

5.4.3.1. eurobot.can.CanBusException

Klasse zum Mitteilen von Fehlern im Zusammenhang mit dem CAN-Bus.

5.4.3.2. eurobot.can.CanConst

In dieser Klasse werden die verwendeten CAN-IDs als Konstanten definiert, so dass anstelle der abstrakten Nummern aussagekräftige Namen verwendet werden können.

5.4.3.3. eurobot.can.CanMessage

Objekte dieser Klasse repräsentieren CAN-Nachrichten die Empfangen wurde oder gesendet werden sollen. Die set-Methoden sind so realisiert, dass die Nachricht nach Angabe von ID und Nachrichtendaten bereits gültig ist. Die fehlenden Werte werden automatisch gesetzt.

5.4.3.4. eurobot.can.CanMessageParserException

Mit dieser Klasse werden Fehler beim Parsen umwandeln von Strings in CAN-Nachrichten übergeben.

5.4.3.5. eurobot.can.CanMessageParser

Wandelt Strings von CAN-Bus in CanMessage-Objekte um.

5.4.3.6. eurobot.can.MessageToRobot

Weist empfangene Sensordaten der Hardwareabstraktion zu (Details siehe Threadbeschreibung 5.4.17.4)

5.4.3.7. eurobot.can.PCANReader

Überwacht den CAN-Bus und liest Daten vom Bus (Details siehe Threadbeschreibung 5.4.17.3)

5.4.3.8. eurobot.can.PCANSupport

Öffnet die CAN-Verbindung und ermöglicht das Senden von Nachrichten über eine intelligente outputMessageQueue (Details siehe Threadbeschreibung 5.4.17.2)

5. UML Dokumentation

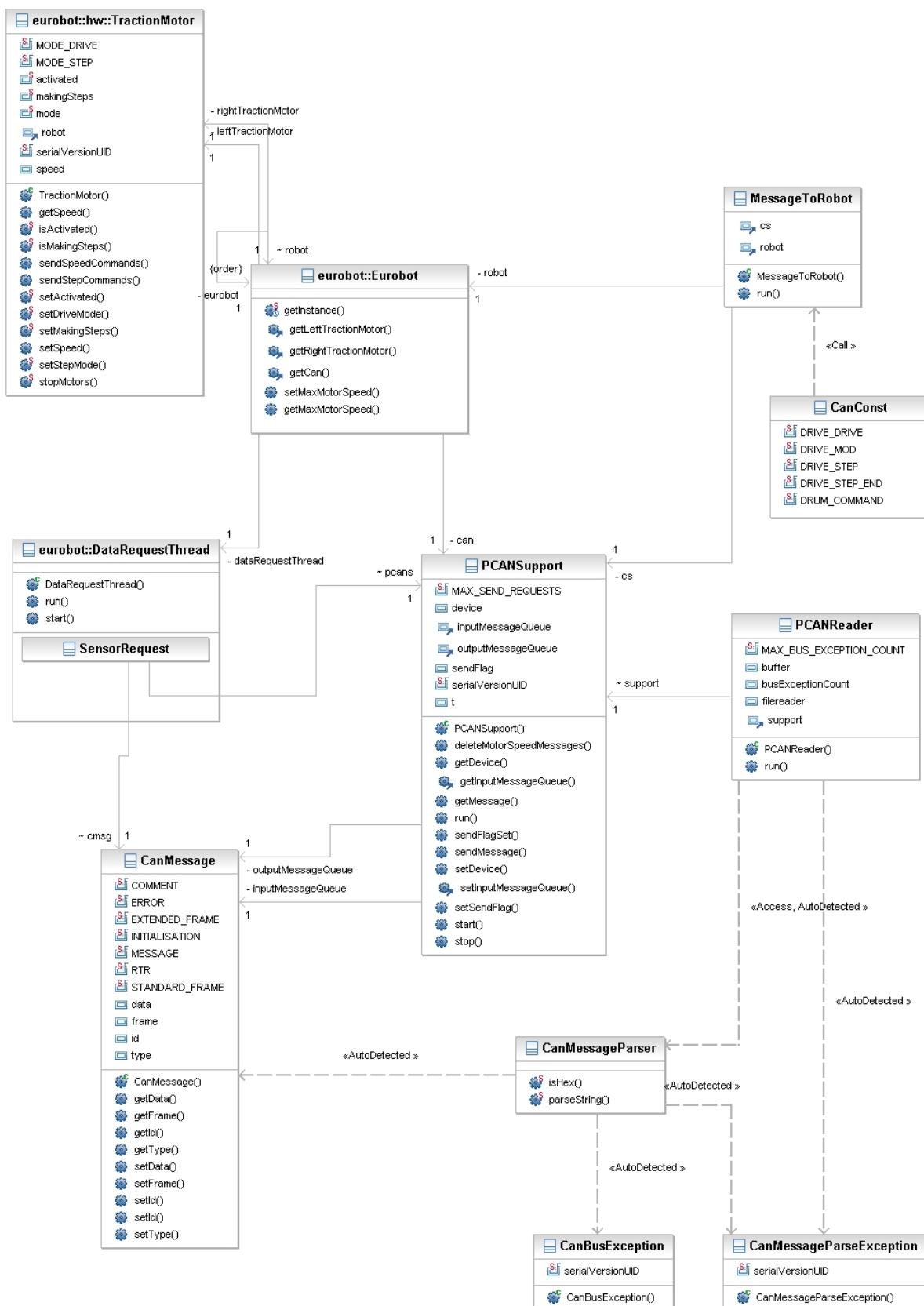


Abbildung 5.12.: Klassendiagramm eurobot.can

5.4.4. eurobot.field

5.4.4.1. eurobot.field.Dispenser

Diese Klasse stellt einen Dispenser dar.

5.4.4.2. eurobot.field.EjectContainer

Damit wird die Auswurfposition dargestellt.

5.4.4.3. eurobot.field.Field

Diese Klasse stellt das Spielfeld dar. Sie beinhaltet unter anderem den Roboter der Mitbewerber.

5.4.4.4. eurobot.field.FieldObject

Diese Klasse stellt Objekte auf dem Feld dar. Sie haben eine Position und eine Grösse. So sind unser Roboter oder auch die Dispenser Feldobjekte.

5.4.4.5. eurobot.field.Obstacle

Ein Obstacle-Objekt ist ein Hindernis auf dem Spielfeld.

5.4.5. eurobot.hw

5.4.5.1. eurobot.hw.Band

Hiermit werden die Einzugsbänder angesteuert.

5.4.5.2. eurobot.hw.CollectionMotor

Der CollectionMotor bietet die Schnittstelle zu den Bandmotoren an.

5.4.5.3. eurobot.hw.Device

Device ist ein Interface für alle Hardwareklassen.

5.4.5.4. eurobot.hw.Drum

Die Trommel wird durch die Klasse Drum dargestellt.

5.4.5.5. eurobot.hw.Eject

Die Klasse Eject bietet Zugriff auf die Auswurfklappe an.

5.4.5.6. eurobot.hw.LidServo

Die Verschlussklappe bei den Bändern kann über die LidServo Klasse angesteuert werden.

5.4.5.7. eurobot.hw.Storage

Der Ballspeicher wird mit der Storage Klasse angesteuert.

5.4.5.8. eurobot.hw.TractionMotor

Die Fahrmotoren können über die Klasse TractionMotor angesteuert werden.

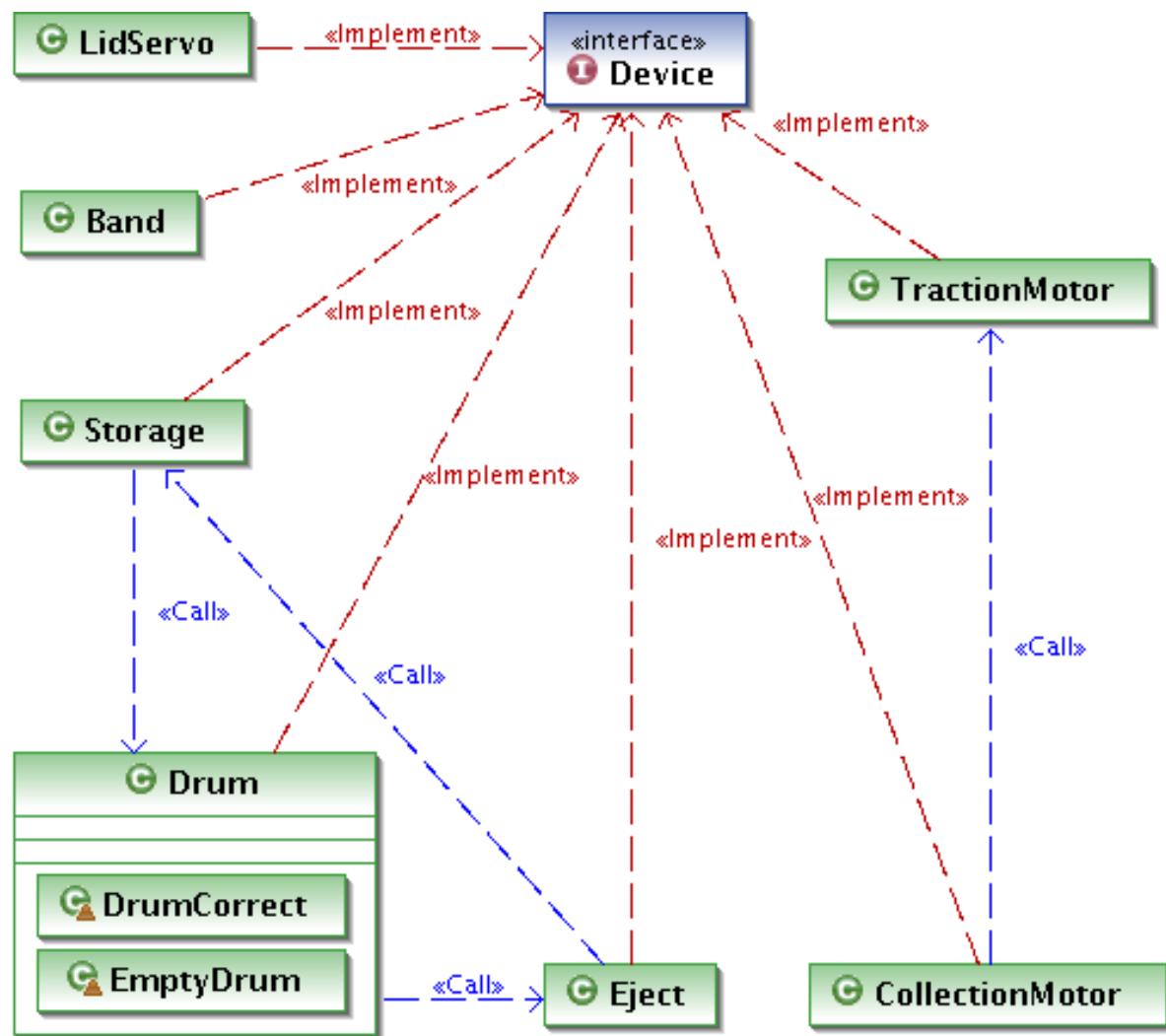


Abbildung 5.13.: Klassendiagramm eurobot.hw

5.4.6. eurobot.hw.sensor

5.4.6.1. eurobot.hw.sensor.ColorSensor

Die Schnittstelle zum Farbsensor. (noch nicht implementiert)

5.4.6.2. eurobot.hw.sensor.DirectionSensor

Der DirectionSensor ist der Kompass.

5.4.6.3. eurobot.hw.sensor.DistanceSensor

Die Distanzsensordaten können über die DistanceSensor Klasse ausgelesen werden.

5.4.6.4. eurobot.hw.sensor.DrumPositionSensor

Die Trommelposition kann über die Klasse DrumPositionSensor abgefragt werden.

5.4.6.5. eurobot.hw.sensor.Sensor

Die Sensor Klasse ist die Basis für alle Sensoren.

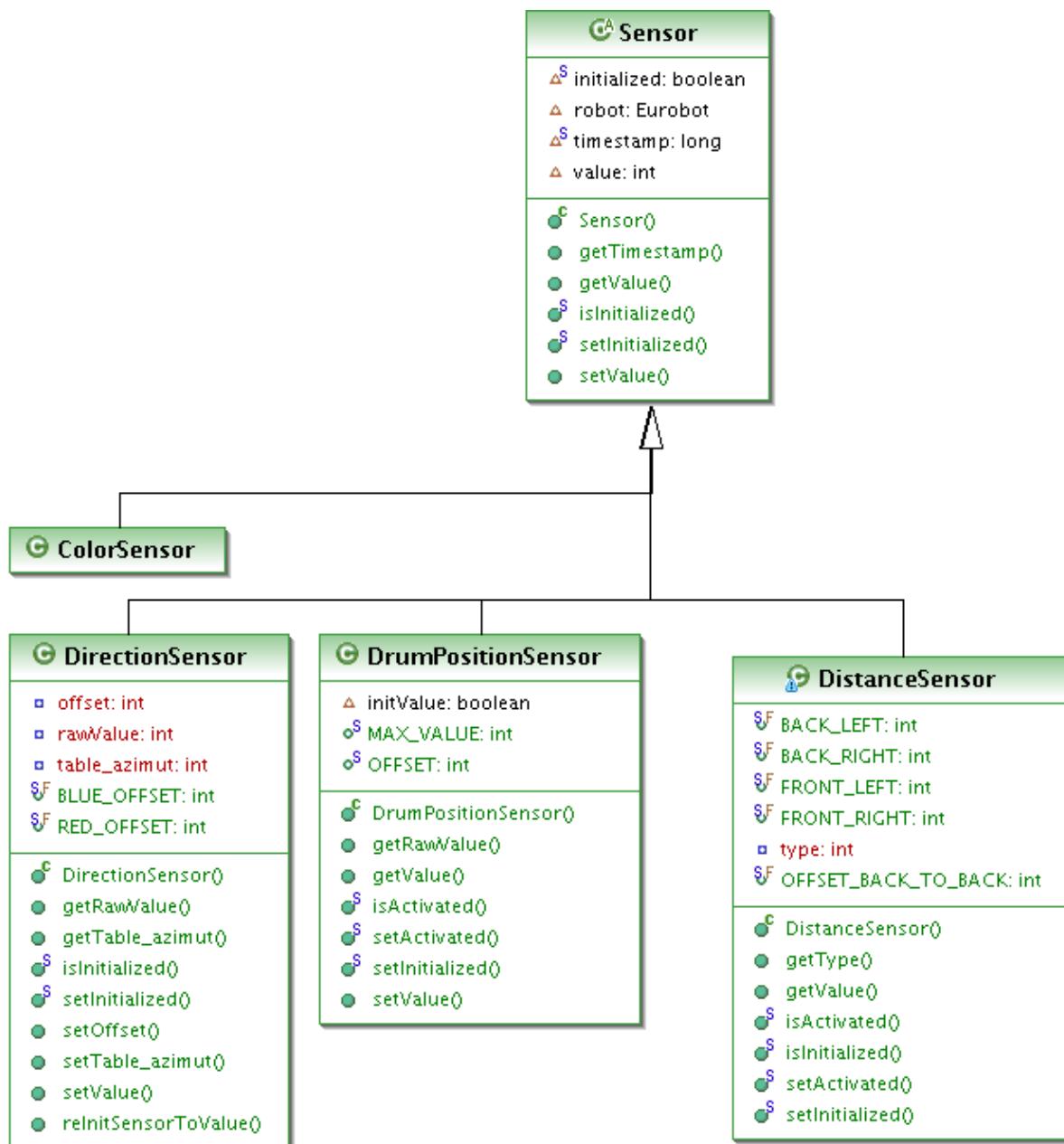


Abbildung 5.14.: Klassendiagramm eurobot.hw.sensor

5.4.7. eurobot.imp

Im Wahlmodul Maschinelles Sehen und Bildverarbeitung erarbeiteten Simon Eggimann und Simon Kunz eine Machbarkeitsstudie ob man Bildverarbeitung in Java einsetzen könnte.

5.4.7.1. eurobot.imp.AngleDistance

Mathematische Beschreibung wie man aus den Bildkoordinaten Winkeländerungen und Distanzen zu den Bällen ausrechnen könnte.

5.4.7.2. eurobot.imp.AreaSearch

AreaSearch beinhaltet die Algorithmen um die Flächen zu separieren des Binärbildes.

5.4.7.3. eurobot.imp.BallDetection

Hauptteil der Bilderkennung welcher Bälle in einem Bild sucht.

5.4.7.4. eurobot.imp.BallDetectionThread

Thread der so in die Spielstrategie eingefügt werden kann. Es wird laufend geprüft ob Bälle im Sichtfeld der Kamera sind.

5.4.7.5. eurobot.imp.BallPoint

Diese Klasse stellt einen Ball aus Sicht der Kamera dar.

5.4.7.6. eurobot.imp.FindDistanceAngle

Methode die anhand bekannten Winkel- und Distanzabweichungen die Korrektur berechnet.

5.4.7.7. eurobot.imp.ImageComponent

Zur Visualisierung von Bildern nötig.

5.4.7.8. eurobot.imp.ImageHelper

Stellt eine Menge von Methoden zur Bildverarbeitung zur Verfügung

5.4.7.9. eurobot.imp.PolarPoint

Diese Klasse stellt einen Ball aus der Sicht des Roboters dar.

5.4.7.10. eurobot.imp.ShowImage

Zur Visualisierung von Bildern nötig.

5.4.7.11. eurobot.imp.TestImageProcessing

Testklasse zur Bilderkennung. Detektiert der Roboter ein oder mehrere Bälle fährt er hinzu und nimmt sie auf.

5.4.8. eurobot.move

5.4.8.1. eurobot.move.CollisionDetection

Die Kollisionserkennung kann erkennen, ob der Roboter an einem gewählten Punkt mit dem Roboter der Mitbewerber kollidieren wird.

5.4.8.2. eurobot.move.Drive

Die Drive Klasse ist für das Fahren und Drehen des Roboters verantwortlich. Sie ruft gegebenfalls Methoden des Routings auf um die Fahrbefehle umzusetzen. Weiter werden aktuelle oder gerechnete Positions- und Richtungsdaten der Sensoren verarbeitet und unter Umständen korrigiert.

5.4.8.3. eurobot.move.Path

Ein Pfad verfügt über eine Referenz auf das erste und das letzte Element.

5.4.8.4. eurobot.move.PathElement

Ein Pfadelement ist ein Teil eines Pfades. Dieser Teil hat eine Koordinate sowie einen Verweis auf das vorherige und nächste Pfadelement.

5.4.8.5. eurobot.move.Point

Ein Punkt hat eine x- und eine y- Koordinate in mm.

5.4.8.6. eurobot.move.Routing

Diese Klasse kann mittels verschiedener Algorithmen einen Pfad zwischen zwei Punkten unter Einbeziehung der Mitbewerberposition errechnen. Weiter verfügt die Klasse über diverse Methoden, welche für die Pfadberechnung hilfreich sind. Das Routing ist unter (5.4.18) genauer beschrieben.

5.4.8.7. eurobot.move.RoutingHelper

Die RoutingHelper Klasse verfügt nur über die Methode getPath(Point,Point). Diese versucht die Pfadfindung mit den Methoden der Routingklasse durchzuführen.

5.4.9. eurobot.move.exception

5.4.9.1. eurobot.move.exception.CollisionException

Bei dieser Exception wird der Punkt mit der errechneten Kollision mitgegeben.

5.4.9.2. eurobot.move.exception.CommandUnknownException

Diese Exception dient zum Mitteilen, dass ein Odometriefahrbefehl von eurobot.move.odometrie.OdometrieDrive (5.4.10.6) nicht verarbeitet werden kann.

5.4.9.3. eurobot.move.exception.DestinationReachedException

Diese erfreuliche Exception teilt mit, dass der Zielpunkt erreicht wurde.

5.4.9.4. eurobot.move.exception.IntersectionNotCalculateableException

Diese Exception dient zum Signalisieren, dass der Schnittpunkt zweier Geraden nicht errechnet werden konnte. (Wird in eurobot.move.Routing (5.4.8.6) verwendet)

5.4.9.5. eurobot.move.exception.IntersectionOutOfField

Wenn der Schnittpunkt zweier Geraden nicht im Spielfeld liegt wird diese Exception geworfen.

5.4.9.6. eurobot.move.exception.NotTurnedException

Mit dieser Exception wird mitgeteilt, dass sich der Roboter nicht wie gewünscht drehen konnte.

5.4.9.7. eurobot.move.exception.PathTooShortException

Wenn ein errechneter Pfad zum weiterverarbeiten zu kurz ist, wird diese Exception geworfen.

5.4.9.8. eurobot.move.exception.UnroutablePathException

Infolge einer zum Beispiel ungünstigen Position könnte geschehen, dass kein Pfad errechnet werden kann, in diesem Fall würde diese Exception geworfen werden.

5.4.10. eurobot.move.odometrie

5.4.10.1. eurobot.move.odometrie.CollisionAvoidanceThread

Erkennt Kollisionen beim Fahren mit Odometrie. (Details siehe Threadbeschreibung 5.4.17.7).

5.4.10.2. eurobot.move.odometrie.Command

Diese abstrakte Klasse stellt einen Odometriefahrbefehl dar.

5.4.10.3. eurobot.move.odometrie.CommandSteps

Diese Klasse beinhaltet einen Fahrbefehl mit einer zu fahrenden Distanz und den gewünschten Zielkoordinaten.

5.4.10.4. eurobot.move.odometrie.CommandTurn

Diese Klasse beinhaltet den Befehl um einen bestimmten Winkel zu drehen.

5.4.10.5. eurobot.move.odometrie.CommandTurnTo

Diese Klasse beinhaltet den Befehl um **zu** einem bestimmten Winkel zu drehen.

5.4.10.6. eurobot.move.odometrie.OdometrieDrive

Mit Hilfe dieser Klasse kann ein Punkt mit Routing unter Verwendung von Odometrie angefahren werden. Dabei wird zuerst konventionell ein Pfad errechnet, welcher anschliessend in Odometriefahrbefehle umgerechnet wird. Diese Fahrbefehle werden anschliessend in einer Liste einer nach dem anderen abgearbeitet. Dabei wird laufend beobachtet ob eurobot.move.odometrie.CollisionAvoidanceThread (5.4.10.1) eine Kollision befürchtet. In diesem Fall wird ein Ausweichmanöver eingeleitet.

5.4.11. eurobot.move.place

5.4.11.1. eurobot.move.place.AimTarget

Diese Klasse beinhaltet Methoden zum Anfahren einer gewünschten Position.

5.4.11.2. eurobot.move.place.Place

Ein Place ist eine anzufahrende Position. Die Position wird mit einem eurobot.field-.FieldObject (5.4.4.4) definiert. Weiter können unter anderem Toleranzinformationen übergeben werden.

5.4.11.3. eurobot.move.place.PlaceError

Bei Fehlern beim Anfahren einer Position kann mit PlaceError weitergegeben werden was genau für einen Fehler geschehen ist, wie z.B. eine x-Koordinatenabweichung oder eine Winkelabweichung.

5.4.12. eurobot.play

5.4.12.1. eurobot.play.BatchStrategy

Dies ist die aktuelle verwendete Strategie. Hier werden die einzelnen Befehle (Ball aufnehmen, ablegen...) nacheinander abgearbeitet. Die Befehle sind im Programmcode nacheinander angeordnet und werden nicht über eine Liste übergeben. Vor den Befehlen wird mit den aktuellen Wert von eurobot.TimeOut (5.4.1.7) entschieden ob die Zeit dazu noch reicht.

5.4.12.2. eurobot.play.ComplexStrategy

Eine alte iterative Strategie, welche die Aufgaben zuerst in eine Liste abfüllte und diese dann nacheinander ausführte.

5.4.12.3. eurobot.play.SimpleStrategy

In dieser Klasse könnte eine weitere iterative Strategie implementiert werden.

5.4.12.4. eurobot.play.Strategy

Die Strategien bauen auf dieser abstrakten Klasse auf.

5.4.12.5. eurobot.play.StrategyBatchA

Abstrakte Klasse für Batch Strategien.

5.4.12.6. eurobot.play.StrategyIterA

Iterative Strategien verwenden diese abstrakte Klasse.

5.4.12.7. eurobot.play.StrategyProperty

Diese Klasse kann Einstellungen von Strategien festhalten. Dies wäre insbesondere wichtig, wenn man während dem Spiel die Strategie wechseln würde. Dann könnte man die Einstellungen damit übermitteln.

5.4.13. eurobot.remote

5.4.13.1. eurobot.remote.Client

Der Netzwerkclient für den Remotezugriff.

5.4.13.2. eurobot.remote.DBHelper

Diese Klasse bietet Methoden für den Datenbankzugriff an.

5.4.13.3. eurobot.remote.Server

Der Server für den Remotezugriff.

5.4.13.4. eurobot.remote.Toolbox

Implementiert alle Methoden, die Remote gebraucht werden, aber nicht direkt per serialisierte Klasse abgerufen werden können (gewisse Klassen können auf Grund ihrer Komplexität nicht serialisiert werden).

5.4.14. eurobot.remote.gui

5.4.14.1. eurobot.remote.gui.ClientClassList

Diese Klasse stellt eine Liste aller verfügbaren Klassen dar.

5.4.14.2. eurobot.remote.gui.ClientGUI

Die Clientanwendung wird über die Klasse ClientGUI gestartet.

5.4.14.3. eurobot.remote.gui.ContextFrame

Die Klasse ContextFrame erstellt das Kontrollfenster, welches ausgewählte Daten vom Roboter anzeigt und das Ausführen einiger Methoden ermöglicht.

5.4.14.4. eurobot.remote.gui.DBLog

Klasse zur Visualisierung der Datenbank.

5.4.14.5. eurobot.remote.gui.SettingsFrame

Die Verbindungseinstellungen für die Netzwerkverbindung werden über die Klasse SettingsFrame verwaltet.

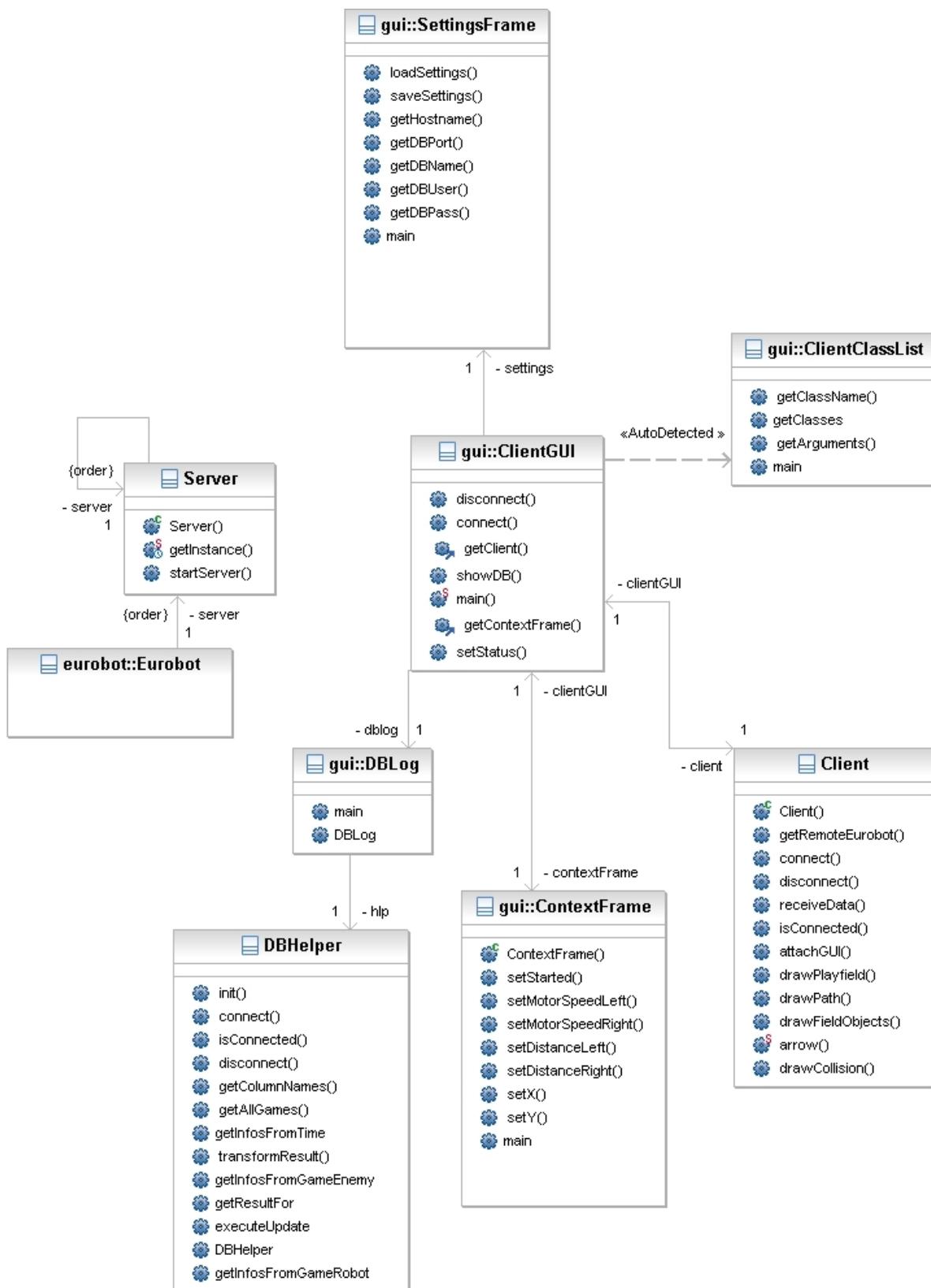


Abbildung 5.15.: Klassendiagramm `eurobot.remote`

5.4.15. eurobot.rs232

5.4.15.1. eurobot.rs232.RS232Support

Öffnet die serielle Schnittstelle und ermöglicht das Schreiben auf die Schnittstelle.

5.4.15.2. eurobot.rs232.RS232Reader

Liest Daten von der Schnittstelle und schreibt Sie in die `inputMessageQueue` vom RS232Support (5.4.17.5).

5.4.15.3. eurobot.rs232.RS232Message

Stellt eine RS232-Nachricht dar.

5.4.15.4. eurobot.rs232.MessageToRobot

Liest Positions-Daten aus der `inputMessageQueue` und weist diese den entsprechenden Spielfeldobjekten zu.

5.4.16. Hardwareabstraktion

Mit dem Hardwareabstraktionspaket `eurobot.hw` (5.4.5) soll die Hardware mit Ihren Funktionen in der Software abgebildet werden.

5.4.16.1. Motoren

Die Motoren sind so implementiert, dass Sie selbstständig CAN-Nachrichten versenden, wenn man die Werte wie Geschwindigkeit (Fahrmotoren) oder die gewünschte Position (Speichertrommel) ändert.

5.4.16.2. Sensoren

Die Sensorwerte werden laufend vom `DataRequestThread` (5.4.17.1) abgefragt. Die Antworten werden von `MessageToRobot` (5.4.17.4) den Sensorobjekten zugewiesen, so dass diese wiederum von anderen Objekten abgefragt werden können.

5.4.17. Threads

Normalerweise laufen mehrere Threads, welche sich um einzelne Teilbereiche des Roboters kümmern.

5.4.17.1. DataRequestThread

Der `eurobot.DataRequestThread` (5.4.1.2) fragt laufend via CAN und RS232 nach Sensorsordaten und nach den Positionen des Roboters und des Gegners.

5.4.17.2. PCANSupport

Damit der CAN-Bus nicht unnötig überbelastet wird kontrolliert der eurobot.can.PCANSupport (5.4.3.8)-Thread das Senden der Nachrichten indem neuen zu sendende Nachrichten in der outputMessageQueue abgelegt werden. Diese Warteschlange wird laufend abgearbeitet. Beim Hinzufügen von neuen Nachrichten wird bei Fahrbefehlen und Kompassleseaufforderungen verhindert, dass es unnötige Duplikate gibt indem bereits existierende Nachrichten gelöscht werden. Weiter werden Befehle für das dynamische Fahren durch Stoppbefehle ersetzt, wenn die Motoren angehalten sind.

5.4.17.3. PCANReader

Der eurobot.can.PCANReader (5.4.3.7)-Thread hört auf dem CAN-Bus und liest dort kommende Daten ein. Diese werden anschliessend mit dem eurobot.can.CanMessageParser (5.4.3.5) in eurobot.can.CanMessage (5.4.3.3)-Objekte umgewandelt und in der inputMessageQueue des eurobot.can.PCANSupport (5.4.3.8) abgelegt.

5.4.17.4. MessageToRobot (CAN)

Die inputMessageQueue wird von eurobot.can.MessageToRobot (5.4.3.6)-Thread laufend ausgelesen. Neue Nachrichten werden analysiert und die Sensorwerte werden der Hardwareabstraktion (5.4.5) (5.4.16) zugewiesen.

5.4.17.5. RS232Reader

Wie beim CAN-Bus gibt es auch bei der seriellen Schnittstelle einen Thread welcher auf Daten wartet und diese in die inputMessageQueue des eurobot.rs232.RS232Support (5.4.15.1) schreibt.

5.4.17.6. MessageToRobot (RS232)

Auch bei der seriellen Schnittstelle werden die Nachrichten der inputMessageQueue verarbeitet. Sie werden hier als Position den Spielfeldobjekten (eurobot.field.FieldObject (5.4.4.4)) Roboter und Gegner zugewiesen.

5.4.17.7. CollisionAvoidanceThread

Da beim Fahren mit Odometrie der Pfad nicht laufen mit Einbezug der Mitbewerberposition neu berechnet wird, gibt es eine Klasse, welche parallel zum Fahren laufend überwacht ob in der Fahrrichtung mit einer Kollision zu rechnen ist. Im Falle einer erwarteten Kollision wird ein Flag gesetzt, welches von eurobot.move.odometrie.OdometrieDrive (5.4.10.6) regelmässig abgefragt wird. Sobald die Kollisionsgefahr nicht mehr besteht, wird das Flag wieder zurückgesetzt.

5.4.17.8. Strategy

Die Strategie versucht der ändernden Umgebung gerecht zu werden und Entscheidungen zum aktuellen Spielgeschehen zu Fällen, die eine optimale Punktzahl zur Folge haben.

5.4.18. Routing

Mit dem Routing wird ein Pfad zwischen zwei Punkten auf dem Spielfeld festgelegt. Dabei werden statische wie auch dynamische Hindernisse beachtet.

Zuerst wurde nur mit dem Vektorrouting gearbeitet, da dieses keine vollständige Abbildung des Spielfeldes als Matrix benötigt und nicht alle Punkte davon initialisiert werden müssen. Bei Simulationen und Praxistests stellte sich heraus, dass dieser Routingmechanismus in ungünstigen Fällen (Roboter und Ziel am Rand des Spielfeldes mit dazwischenliegendem Hindernis) sehr viele Rekursionsschritte benötigt und teilweise keinen Weg findet oder die Wegfindung zu lange dauert. Auf der Suche nach alternativen Routingmethoden verwies uns Ivo Oesch auf ein Verfahren, welches beim Routen von Leiterplatten angewendet wird. Ein solches Verfahren haben wir anschliessend als Matrixrouting implementiert und erste Optimierungsschritte unternommen.

Das Vektorrouting wird momentan nur noch in `eurobot.move.Drive.driveTo()` als Backuplösung eingesetzt, für den Fall, dass das Matrixrouting mit einem Fehler abbricht.

Beide Routingvarianten liefern uns momentan im Erfolgsfall nach zwischen 4-12 ms einen möglichen Pfad. Genauere Messungen wurden noch nicht vorgenommen.

Da wir nun nicht mehr mit dynamischen Fahrbefehlen, sondern mit Odometrie fahren, ist ein Routingverfahren nötig, welches einen Pfad mit möglichst wenig Drehpunkten generiert. So sind wir auf das unten erwähnte Tangenten-Routing gestossen, welches nach dem Matrixrouting ausgeführt wird, sofern dem Mitbewerber ausgewichen werden muss. Sofern das Tangenten-Routing einen Pfad finden, wird der Pfad des Matrixroutings damit überschrieben.

5.4.18.1. Vektorrouting `eurobot.move.Routing.findPathByVectorRouting`

Das Vektorrouting arbeitet rekursiv mit einer Rekursionstiefenkontrolle, damit das Routing abgebrochen werden kann, sofern die Methode nicht schnell genug ans Ziel kommt.

Das Verfahren arbeitet so, dass immer ein nächster Punkt in Luftlinie zum Ziel gesucht wird. Sofern das Anfahren dieses Punktes zu einer Kollision mit einem Hindernis führen würde, versucht die Methode Punkte um den idealen Punkt herum zu verwenden. Beim Erfolg wird dieser Punkt an den Pfad angehängt und ein nächster Punkt wird gesucht. Beim Misserfolg mit dem gewünschten oder darum herum liegenden Punkten meldet die Methode dies der aufrufenden Methode mit einer Exception. Diese weiss dann, dass dieser Punkt auch ohne Kollision nicht zu verwenden ist. Weiter werden alle Punkte, bei denen festgestellt wurde, dass Sie zu einer Kollision mit einem Hindernis führen oder dass von ihnen aus nicht ans Ziel geroutet werden kann, in einer Liste gespeichert, so dass diese beim aktuellen Routingversuch nicht mehr berücksichtigt werden.

5.4.18.2. Matrixrouting `eurobot.move.Routing.findPathByMatrixRouting`

Das Matrixrouting legt ein Raster über das Spielfeld und initialisiert dieses. Dabei werden mit Hindernissen belegte Felder markiert. Anschliessend werden rund um den Startpunkt die unbelegten Felder mit aufsteigenden Zahlen ausgefüllt. Das ganze geschieht wie eine Welle, welche sich vom Startpunkt aus ausbreitet. So fliessen die Zahlen um Hindernisse, so dass nach dem Erreichen des Ziels zur Pfadfindung jeweils das umliegende Feld mit der tiefsten Zahl gewählt werden kann.

Dieses Verfahren wurde optimiert, indem das Befüllen mit Zahlen beim Erreichen des Ziels beendet wird. Somit werden keine unnötigen Felder gefüllt. Weiter wählt die anschliessende Pfaderstellung (rückwärts: Ziel→Start) bei mehreren möglichen angrenzenden Feldern mit der kleinsten Zahl, dasjenige Feld welches den kleinsten direkten Abstand zum Startpunkt hat (ohne Hindernisse zu beachten).

Am Ende wird der Pfad optimiert, indem eine Methode beim Startpunkt beginnend versucht auf direktem Weg an einen dem Ziel möglichst nahen Punkt zu gelangen, ohne auf diesem Weg zu kollidieren. Die dabei übersprungenen Punkte werden aus dem Pfad entfernt, so dass sich dieser verkürzt und die zu fahrende Richtung genauer stimmt. Ohne diese Optimierung gibt es beim jeweiligen neu routen während der Fahrt zu grosse Richtungsänderungen.

5.4.18.3. Tangentenrouting `eurobot.move.Routing.findPathByTangentRouting`

Wie weiter oben erklärt, ist für das Fahren mit Odometrie ein Routingverfahren nötig, welches lange gerade Pfadteile mit möglichst wenig Drehpunkten liefert.

Das Tangentenrouting geht davon aus, dass sich der Mitbewerber zwischen Start- und Zielposition befindet. Das Routing rechnet nun mit einem Kreis um den anderen Roboter. An diesen werden vom Start- und Zielpunkt aus Tangenten angelegt. So ergeben sich zwei mögliche Pfade. Wobei davon der kürzere ausgewählt wird.

6. Abschluss der Arbeit

6.1. Schweizermeisterschaft

16. Mai 2008

Am Abend vor der nationalen Ausscheidung beendeten wir unsere Tests rechtzeitig. Unsere Autos waren um 18:30 Uhr bepackt und abfahrtsbereit.

17. Mai 2008

Um 7:00 Uhr fuhren wir in Richtung Rapperswil los. Dort angekommen richteten wir unseren Arbeitsplatz ein und führten unseren Roboter zum ersten Mal dem Schiedsrichterkomitee für die Zulassungsprüfung vor. Nach einigen Optimierungen bei der Erkennung und Umfahrung des Gegners wurde unser Roboter zugelassen.

Zu Beginn des Wettkampfes hatten wir einige Schwierigkeiten beim Start. Einmal brachten wir die Software durch unsere Nervosität zum Absturz. Ein anderes Mal hatten wir auf dem CAN-Bus ein Problem, welches den Start verhinderte. In der vierten und fünften Runde hatten wir mit der Ballspeichertrommel Probleme.

Die Resultate vom Samstag:

MarsRiders	vs	- - -	00:00
MarsRiders	vs	Arc'obot	00:12
MarsRiders	vs	Dombot Rapperswil	13:01
MarsRiders	vs	EcubeMsquare	01:19
MarsRiders	vs	Icare	01:19

Über Nacht wurde an der Ballspeichertrommel ein Sensor angebracht, mit dem nun festgestellt werden konnte, ob die Ballspeichertrommel an der richtigen Position steht. Daneben wurden noch diverse Verbesserungen der Software vorgenommen, welche die Fehleranfälligkeit reduzieren sollten. Als die Sonne wieder am Horizont aufstieg, legten sich auch die letzten Teammitglieder schlafen, wenn auch nicht für lange.

18. Mai 2008

An diesem Morgen gelang uns endlich der Durchbruch. Unser Roboter punktete Match für Match und unserer Arbeit hatte sich ausbezahlt. Beim letzten Match hatte b.i.t.t. wieder einmal Lampenfieber und bewegte sich nicht einmal aus seinem Startfeld.

Wir belegten punktgleich mit dem Team Arc'obot den 8. Platz. Da unser Roboter jedoch mehr Punkte während der Spiele sammelte, qualifizierten wir uns für die Viertelfinals. Danach hatte unser Roboter b.i.t.t. keine Startprobleme mehr. Wir gewannen jeden Match und bewegten uns ohne Umweg direkt zuoberst aufs Podest.

6. Abschluss der Arbeit

Die Resultate vom Sonntag:

MarsRiders	vs	oneTeam	20:14
MarsRiders	vs	Deimos	27:01
MarsRiders	vs	CVRA	27:01
MarsRiders	vs	e-Robot	00:14

Viertelfinal:

1. Phobos vs **8. MarsRiders** 01:23

Halbfinal:

MarsRiders vs Funky Donkey Rapperswil **16:08**

Nach dieser Runde waren wir definitiv für Heidelberg qualifiziert!

Finaleinzugsrunde:

MarsRiders vs Deimos **27:03**

Über die Verliererstrasse qualifizierte sich das Bieler Team Deimos für den Final. Sie setzten sich gegen das Winterthurer Team Viper durch und so lautete der Final gleich wie die Finaleinzugsrunde. Zufälligerweise war dann auch das Resultat das Gleiche.

Final:

MarsRiders vs Deimos **27:03**



Abbildung 6.1.: Siegerfoto

6.2. Weltmeisterschaft

22. Mai 2008

Um 6:58 Uhr starteten die Motoren der 2 BFH Fahrzeuge und wir setzten uns in Bewegung Richtung Heidelberg. In Langenthal luden wir die restlichen Teammitglieder auf.

Schon bald erreichten wir Basel und somit den Zoll. Unserer Ladung wurde keine Beachtung geschenkt und so konnten wir problemlos passieren.

Wir erreichten Heidelberg noch vor dem Mittag und konnten unseren Arbeitsplatz beziehen. Nach dem Mittagessen ging es dann zur Zulassung für b.i.t.t. Der Schiedsrichter simulierte zwar keinen realistischen Gegner, dennoch konnte unser Roboter ausweichen. Wir erhielten die Zulassung beim ersten Versuch.

23. Mai 2008

Noch vor dem ersten Match mussten wir bei der Spielleitung Beschwerde einreichen, weil die Tische nicht Regelkonform waren. Da die Spieltische von der französischen Ausscheidung benutzt wurden und offensichtlich 90% der französischen Teams die Bälle nicht aufnehmen konnten, wurde unter den Spendern eine Ballfixierung montiert. Diese Änderung erschwerte jedoch uns und noch weiteren Teams die Ballaufnahme, insbesondere wenn sich mehrere Bälle in der Nähe des Spenders befanden. Wir erhielten Recht und durften vor unseren Spielen diese Ballfixierungen entfernen.

Der erste Match:

MarsRiders vs Turag (D) 16:10

Leider konnten wir bei diesem Match keine weissen Bälle aufnehmen, weil die Distanz zum Spender nicht stimmte. Deshalb gab es keine Samples und somit auch keine Zusatzpunkte.

Der zweite Match:

MarsRiders vs 1966 (GB) 01:07

Unser Roboter hätte eigentlich einen perfekten Lauf absolviert. Leider stimmte mit dem Sensor, der die eingeladenen Bälle detektiert etwas nicht. So dachte unser Roboter, er hätte keinen Ball eingeladen und drehte deshalb die Trommel nicht. Wenn er denkt er hätte keine Bälle eingeladen, so lädt er auch keine aus. Dieser Fehler des Sensors war noch nie aufgetreten und trat in Folge auch nicht mehr auf. Zur Sicherheit wurde er aber Softwaremässig auch noch überbrückt.

Der dritte Match:

MarsRiders vs Übermachin (SRB) 10:22

Auch in diesem Match machte unser Roboter seine Arbeit grundsätzlich gut. Leider entlud unser Gegner jedoch zuerst seine Bälle bei unserem Container. Unser Roboter entlud dann seine Bälle darüber. Diese zählten so nicht und wir verloren deshalb auch diesem Match äusserst unglücklich.

6. Abschluss der Arbeit

Am Abend fuhren wir mit der Strassenbahn in die Altstadt von Heidelberg. Dort gönnen wir uns zusammen mit den mitgereisten Dozenten R. Weber und F. Bircher im *goldenene Falken* ein vorzügliches Abendessen und feierten somit den Schweizermeistertitel.

24. Mai 2008

Leider sind wir nach dem ersten Tag etwas in der Tabelle zurück gefallen. Bei zwei normalen Spielen wäre es jedoch kein Problem noch ins Final zu kommen, da wir nur 6 Punkte Rückstand auf Platz 16 hatten.

Der vierte Match:

MarsRiders vs RoBUTE (H) 00:01

Der Morgen begann schlecht. Das Navigationssystem verweigerte seinen korrekten Dienst. Entweder waren zwei Baken falsch aufgestellt oder der Gegner verhinderte durch seine ungünstige Startposition den Kontakt zur einen Navigationsbake. Wir drehten folglich nur eine Pirouette und verliessen nicht einmal das Startfeld. Nach diesem Spiel war eigentlich klar, dass wir die Finalrunde nicht erreichen werden.

Der fünfte Match:

MarsRiders vs RallyRobot (RUM) 16:03

Beim letzten Match konnten wir wieder etwas Punkten. Aber auch in diesem Spiel hatte das Pech seine Finger im Spiel. Ein weisser Ball rollte vor den farbigen Spender und somit nahmen wir einen falschen Ball auf. Dies machte unsere Samples kaputt, was gleich eine massive Punkteeinbusse zur Folge hatte.

So ging für uns das Abenteuer Heidelberg schon am Vormittag zu Ende. Am Schluss waren wir auf Rang 24 von 46 zugelassenen Teams. Unser Potential blieb weitgehend ungenutzt, trotzdem waren wir über alles gesehen zufrieden. Wir hatten an den Schweizermeisterschaften mehr erreicht als wir erhofften.

Ein weiteres Highlight war dann das Barbecue am Abend. Die Organisatoren rechneten mit ca. 80 Leuten, jedoch erschienen etwa 200. Das Fleisch wurde schon bald knapp. Weiter ging eine Gasflasche für den einen Grill bereits nach 10 Minuten aus und es hatte keine Reserveflasche.

Unser Team hatte einen Grill dabei und leinte diesen den Grillmeistern - nachdem wir unser Fleisch selber gegrillt hatten und somit die lange Warteschlange umgingen!.

Irgendwann hatte man dann einen Holzgrill organisiert. Einige wollten diesen mit ein wenig Papier und Kohle in Betrieb nehmen. Nachdem wir diesem lustigen Treiben ein wenig zugeschaut hatten, beschlossen unsere zwei Pfadfinder den armen Leuten zu helfen.

Nach kurzer Zeit war dann der Grill in Betrieb und unser Team unterstützte das Eurobot-Grillteam. Wir betreuten den ganzen Abend den Holzkohlegrill.



Abbildung 6.2.: Grillhilfe

Wir wurden dafür mit reichlich Bier entschädigt. Der Abend wurde ziemlich lustig. Nach interessanten Bürostuhlwettrennen gegen andere Eurobot-Teams ging dann der Abend langsam zu Ende. Wir machten uns auf den Weg in Richtung Unterkunft und legten uns schlafen.

25. Mai 2008

Nachdem alle schlecht oder noch schlechter in der Turnhalle geschlafen hatten, ging es nun nach Hause. Um ca. 8:45 Uhr starteten wir in Heidelberg. Die Fahrt verlief sehr ruhig. Am Zoll wollte wieder keiner wissen, was wir transportieren. Um ca. 13.00 Uhr waren wir dann zurück in Burgdorf.

6.3. Fazit Projektarbeit 1

6.3.1. Arbeitsvorgehen

Unsere Zeitplanung hat sich bis Weihnachten bewährt. Zu Beginn des Projekt lernten sich das Team erst mal kennen. Eigenheiten und Spezialfähigkeiten der einzelnen Teammitglieder wurden während der Konzeptphase bekannt. Bis zur Konzeptpräsentation wurde sehr viel gesprochen. Das theoretische Modell in unseren Köpfen wurde durch das LEGO-Modell des Einzuges und des Kartonmodels der Trommel viel praktischer und wir konnten unser Konzept überzeugt präsentieren.

Danach wurde auf verschiedenen Ebenen gearbeitet. Parallel wurden mechanische Zeichnungen gemacht, Akteure und Sensoren evaluiert und eine Softwareumgebung programmiert. Nebenbei liefen Abklärungen mit Sponsoren und das Erstellen unserer Team-Homepage. Kurz vor Weihnachten wurde klar, dass einige Probleme nicht gerechnet werden konnten. Der Vorgänger schloss das Navigationsmodul nicht rechtzeitig ab. Das Zusammenführen der Einzelarbeiten nahm mehr Zeit in Anspruch als angenommen. Mit viel zusätzlichem Auf-

6. Abschluss der Arbeit

wand im neuen Jahr gelang es uns in der ersten Schulwoche erfolgreich mit den Roboter zu Fahren.

6.3.2. Soll / Ist

Unser Roboter erfüllt alle Anforderungen der Zulassungsprüfung, ausser das wir ein Punkt noch nicht in neunzig Sekunden erreichen. Der Notschalter ist angeschlossen und die Start-schnur implementiert. Das aufnehmen, speichern und ausgeben des Unihockeyballes funktioniert gut. Fehler können durch eine Anpassung des Aufnahmemechanismus noch ausgeschlossen werden.

6.4. Ausblick auf Projektarbeit 2

Nachdem wir nun ein Semester lang motiviert und erfolgreich an diesem Projekt gearbeitet haben, freuen wir uns auf das nächste Semester und wollen dabei neue Herausforderungen angehen und möglichst erfolgreich am Wettbewerb teilnehmen.

6.4.1. Fertigstellung angefangener Teilprojekte

Zuerst wollen wir begonnene Teilprojekte abschliessen. Diese umfassen den fertigen Anbau und die Ansteuerung der noch nicht in Betrieb genommenen Sensoren: Den Farbsensor zur Erkennung der Farbe des aufgenommenen Balls und die Distanzsensoren hinter dem Roboter, welche ein präzises Anfahren der Abladestelle ermöglichen sollte. Weiter ist in der Software die Spiellogik fertig zu implementieren, so dass der Roboter die Wettbewerbsaufgaben selbstständig lösen kann.

6.4.2. Optimierung

Ebenfalls ein wichtiger Bestandteil der Projektweiterführung ist die Optimierung des aktuellen Roboters. Darunter fallen die Verbesserung der Motorenansteuerung, so dass wir schneller fahren können, sowie die Optimierung der Steuersoftware. Bei der Software sind konkret die Erhöhung der Geschwindigkeit sowie die Verbesserung der strategischen Abläufe auf der Tätigkeitsliste.

6.4.3. Erweiterung

6.4.3.1. Batteriespannungsüberwachung

Damit der Roboter nicht plötzlich mit leeren Batterien nach 30 s auf dem Spielfeld stehen bleibt, möchten wir die Batteriespannung überwachen können, so dass wir die Akku's rechtzeitig wechseln.

6.4.3.2. Bilderkennung

Um nach dem Leeren der Dispenser weitere Bälle aufzunehmen, planen wir die Erweiterung des Roboters mit einer WebCam. Zudem wird die Software angepasst, so dass wir die Position von Bällen vor dem Roboter kennen.

6.4.3.3. Wireless-CAN-Debugger

Während der Entwicklung stellten wir fest, dass es praktisch wäre, die Nachrichten auf dem CAN-Bus laufend mit verfolgen zu können. Da sich das mit einem CAN-Dongle und Computer als unpraktisch erwies, ist die Idee eines Wireless-CAN-Debuggers entstanden.

- kleines Gerät für den Anschluss am Bus
- Zugriff über bekannte Technologie (TCP/IP, WLAN)
- Komfortable Oberfläche am PC
 - Nachrichten empfangen und sortiert darstellen
 - Nachrichten definieren und per Klick oder mit Timer versenden
 - Makros erstellen
 - * Folgen von Nachrichten definieren → per Klick mehrere nacheinander senden
 - * Reaktionen auf Nachrichten definieren → Nachricht beantworten

6.4.3.4. intelligentes Routing

Eine weitere mögliche Erweiterung wäre ein intelligentes Routing, welches den gegnerischen Roboter nicht statisch betrachtet, sondern dessen Fahrtrichtung in die Pfadfindung einbezieht.

6.4.4. Mechanik

Die Halterungen für die Positionssensoren müssen noch gefertigt werden. Alle Peripherieanschlüsse für den Industrie-PC und der Not-Aus werden noch auf dem Roboter integriert. Zudem sollte unser Roboter noch eine Schutzverschalung aus Plexiglas erhalten. Eventuell muss der Antrieb noch angepasst werden.

6.5. Was wir dem nächsten Team weitergeben möchten

Während dem ganzen Projekt haben wir sehr viele Erfahrungen gesammelt, die wir unserem Nachfolgerteam gerne weitergeben möchten, um deren Arbeit zu erleichtern.

6.5.1. Hardware

- Der Antrieb ist besser mit DC-Motoren zu realisieren, welche über eigene Odometrierräder verfügen sollten. Um während den 90 Sekunden vernünftig zu punkten ist es wichtig, dass der Roboter mindestens 0.8 - 1.0 m/s schnell fahren kann. Mit dem ersten Antrieb den wir realisiert haben, ist der Roboter lediglich 0.3 m/s gefahren und das war eindeutig zu langsam.
- Im Halbschrittbetrieb läuft ein Schrittmotor „sanfter“, hat aber weniger Moment. Trotzdem bewährte sich der Halbschrittbetrieb, da so mehr Grip erreicht werden konnte.
- Ein Schneckengetriebe hat einen sehr schlechten Wirkungsgrad.
- Es ist wichtig, die Spielaufgabe gründlich zu studieren um herauszufinden wie am meisten Punkte erzielt werden können. Unser Roboter konnte zum Beispiel nur Bälle im Standard-Behälter ablegen. Er war nicht in der Lage Bälle zu werfen, oder auf dem Spielfeld welche einzusammeln. Diese Strategie hatte beim Wettkampf grosse Vorteile, da wir eine der zuverlässigsten und schnellsten Ballaufnahmen hatten.
- Die Antriebsräder sollten wenn möglich innerhalb des Gehäuses sein, damit nahe an den Banden entlang gefahren werden kann.
- Wir empfehlen den Schrittmotor der Firma "McLennan" bei zukünftigen Projekten nicht mehr zu verwenden. Der Motor ist nicht sehr kraftvoll und das Getriebe hat relativ viel Spiel. Am Wettkampf hatten wir immer wieder Schwierigkeiten, dass der Motor Schritte verloren hat und die Trommel nicht richtig positioniert war. Ein DC-Motor mit Planetengetriebe stellt da die bessere Alternative dar.
- Eine „normale“ Notebook Festplatte ist zu wenig stossresistent. Wir haben 2 davon zerstört. Schlussendlich haben wir eine Flash-Festplatte mit 32GB Speicher eingesetzt. Diese ist aber mit einem schreibenden Zugriff von max. 6MB/s sehr langsam! Dies kann zu „Hängern“ bei der Ausführung von Software führen (bis zu ein paar Sekunden). Eine sinnvolle Investition wäre daher eine teurere SSD-Festplatte (ca. 400 Franken).
- Positionen von beweglichen Teilen sollten von der Software aus mittels Endschaltern oder ähnlichem abgefragt werden können um Fehler zu erkennen.
- Die von uns verwendete Flash-Festplatte ist recht langsam. Dies ist aber nur beim Starten des Betriebssystems und beim Kompilieren der Software wirklich bemerkbar.
- Das Navigationssystem arbeitet sehr zuverlässig, wenn auch recht träge. Pro Sekunde kann man bis zu 10 Positionen abfragen. Das beste Resultat wird erzielt, wenn abwechselnd die eigene und die Gegner-Position abgefragt wird. Bei schneller Fahrt (ca. 1m/s) kann die Position bis zu 0.5 Meter „hinterherhinken“. Das beste Resultat wurde mit 40kHz und Kanal 0 erzielt. Eine Optimierung ist kaum möglich, ausser man wechselt die Mikrokontroller oder das ganze System. Ausfälle hatten wir keine zu verzeichnen, das System reagiert nicht auf andere Ultraschallquellen, weil ein Codierung

stattfindet. Einzig zu bemängeln ist, dass der Stromverbrauch sehr hoch ist. Nach 3-4 Matches müssen die 9V-Batterien ausgetauscht werden. Wirtschaftlicher wäre die Verwendung von Akkus, was getestet werden sollte. Im Nahbereich (bis 50cm) sollten zusätzlich Distanzsensoren (induktiv oder Infrarot) als Gegnererkennung eingesetzt werden!

6.5.2. Software

- Die Software sollte beim Initialisieren Sensordefekte und/oder ungültige Sensorwerte erkennen und dementsprechend ohne diesen Sensor arbeiten können.
- Eine saubere Hardwareabstraktion hat sich bewährt, so kann man sich zum Beispiel bei der Spielstrategie einfach auf die Strategie konzentrieren. Beispiel: `robot.getDrum().turnToFreePosition()` oder `robot.getStorage().getWishedColorToEject()`
- Die Idee, dass der Roboter die aktuelle Softwareversion aus der Versionsverwaltung holen und kompilieren kann, hat sich als sehr komfortabel erwiesen.
- Ein verteiltes Versionierungssystem, dass nicht auf EIN Repository angewiesen ist, würde sich vor allem in einem Umfeld ohne Internet bewähren. SVN ist da zu wenig flexibel. Mercury (hg) sollte evaluiert werden. Eine gute Anleitung dazu findet sich in der c't 12/08 vom 26.5.2008.
- Die Gerüchte betreffend Java, das es sich als langsam erweist, haben sich nicht bestätigt. Wir hatten keine Geschwindigkeitsprobleme. Hänger waren der langsamen Flash-Festplatte zuzuschreiben.
- Unser Java-GUI, welches die Anzeige des aktuellen Roboterstatus ermöglicht, hat sich als sehr praktisch erwiesen!
- Das Aufstarten des Roboters ist das Wichtigste, die Software sollte möglichst jedes menschliche Versagen ausschliessen können. Vertauschte Navigationsbaken oder nicht eingeschaltete Navigationsbaken könnten durch geeignete Test in der Software abgefangen werden. Der Initialisierungsvorgang muss auch ohne angeschlossenes Laptop möglich sein. Ein Hintergrundprozess könnte beispielsweise auf eine Tastenkombination hören oder auf einen GPIO. Eine einmal laufende Software kann per CAN neu gestartet werden. System-Calls sind in Java möglich (`Runtime.getRuntime().exec("shutdown -r")`).

6.5.3. Wettkampfvorbereitung

- Der genaue Wettkampfablauf sollte mehrfach geübt werden. Es ist zu beachten, dass vom Aufstellen auf dem Tisch bis zum eigentlichen Spielstart **einige Minuten vergehen können**. Den Roboter also auch einmal für 30 Minuten initialisiert stehen lassen und dann die Startsehne ziehen!

6.5.4. Wettkampf

6.5.5. Sponsoren

- Sponsorenanfragen lohnen sich frühzeitig. Jede Firma beschliesst zu Beginn des Jahres ihr Sponsoring-Budget. Sponsoren wollen Gegenleistungen sehen. Eine Beschreibung der Gegenleistung wie Plakat, Homepage oder Roboterbanner lohnt sich. Ein regelmässiger Newsletter wird geschätzt.
- Eine Homepage zur Information der Sponsoren ist toll. Am einfachsten ist ein eigener Anbieter über eine BFH-Url zu verlinken. **Das Angebot, dass der Informatik-Service der Schule bietet, ist sehr unkomfortabel.** Die Homepageänderungen können nur auf einem Testserver vorgenommen werden. Somit dauerte die Aktualisierung zeitweise über eine Woche.

6.5.6. Medien

- Medien sind so etwas von unfähig. Damit keine Missverständnisse entstehen empfehlen wir, ein Fact-Sheet zu erstellen und dies jedem, der ähnlich wie ein Moderator, Fotograf oder Reporter aussieht, in die Hand zu drücken. Das Schweizer Fernsehen berichtete in ihrem Beitrag der Tageschau, das Team MarsRiders studiere an der Fachhochschule Rapperswil und musste sich bei mehreren Studenten für den Fehler entschuldigen.

6.5.7. Diverses

- Falls ein Kickboard zur schweizerischen Ausscheidung mitgeführt wird, sollte unbedingt auch ein Helm eingepackt werden. Es könnte sein, dass ein übermüdetes Teammitglied mit dem Kickboard einen Unfall mit einem Tisch hat.
- Menschen aus Dublin mögen keine Simpsons.
- Heidelberger können nicht grillieren.
- Eigener Grill mitnehmen (kann gegen Bier vermietet werden).
- Nicht nur 5l Burgdorfer Bier mitnehmen.
- Therapiematratzen eignen sich nicht zum schlafen.
- Um die französischen Fans zu toppen, muss unbedingt ein Megafon (oder Baseball-Schläger) mitgenommen werden. Von Vorteil ist ein Megafon, dass bereits irgendeine nervige Melodie abspielen kann wie zum Beispiel die Titelmelodie von Titanic.
- Die Spieltische an der EM kommen aus Frankreich, das heisst sie sind nicht regelkonform. Bitte nicht verwundert sein, wenn nicht alles so ist, wie im Labor in Burgdorf.
- Engel sind nicht immer so, wie man sie sich vorstellt ((not so) beautiful Hellsangel).

- Die Eurobotteams für die EM sollten nicht grösser als 2 Personen sein, da sonst der Arbeitsplatz nicht genügend gross ist.
- Es sollte für jedes Teammitglied eine Schweißerbrille mitgeführt werden, falls russische Teams am Start sind.
- Die ausländische Polizei macht auch schöne Fotos. So können bereits vor dem eigentlichen Wettkampf Punkte in Flensburg gesammelt werden.

6.6. Danksagung

Diese Projektarbeit wäre nicht möglich gewesen ohne fremde Hilfe.

Daher möchten wir uns bei allen direkt und indirekt Beteiligten herzlich bedanken.

Besonders intensiv haben uns die Dozenten R. Weber, I. Oesch, D. Lanz und F. Bircher mit Ratschlägen, Empfehlungen und Notfallplänen unterstützt. Wir haben ihre unkomplizierte Art sehr geschätzt.

Weitere grosse Hilfe waren Stucki Erwin und Hersperger Walter, die für uns die Bestellungen übernahmen und uns mit spontanen Ideen die Arbeit erleichterten.

Mit der Fertigung mechanischer Komponenten, hat uns Niklaus Bruno mit seinem Team einen grossen Teil der Arbeit abgenommen.

Zudem danken wir unseren Sponsoren Farnell, PCB-Pool und Graf Endless Belts für ihre grossen finanziellen Beiträge an unsere Arbeit.

Weiter bedanken wir uns bei der Firma IT & Design Solutions GmbH für das Bereitstellen der Versionsverwaltung, Mailingliste und WebDav-Netzlaufwerk.

Für die gute Zusammenarbeit mit dem OneTeam bedanken wir uns herzlich.

Als Projektleiter bedanke ich mich persönlich bei meinen Team-Mitgliedern für ihren grossen Einsatz. Als Privatperson bedanke ich mich herzlich bei meiner Freundin, meiner Familie und meinen WG-Genossen für das entgegengebrachte Verständnis während dieser stressreichen Zeit.

Simon Kunz

6.7. Schlusswort

Bei unserer Projektarbeit hat sich eine gute Teamarbeit entwickelt. Eine gute Zusammenarbeit während des ganzen Semesters war unumgänglich.

Unsere Tätigkeiten waren sehr vielfältig. Der Tätigkeitenbereich bestand aus Lötarbeiten, Testen von Komponenten, Entwicklung von hardwarenaher bis abstrakter Software mit C oder JAVA. Unsere Arbeit war nicht immer einfach, vor allem war die Einarbeitung in die Softwareumgebung für die Atmel-Boards anspruchsvoll.

Bei Schwierigkeiten oder Problemen halfen uns die Dozenten D. Lanz, R. Weber, I. Oesch und F. Bircher rasch.

Erfahrungen konnte das ganze Team mit der Projektorganisation, Motoransteuerung und Robotik sammeln.

MarsRiders

6.8. Selbstständigkeitserklärung

Mit unserer Unterschrift bestätigen wir, die vorliegende Arbeit nach bestem Wissen selbstständig und mit den angegebenen Quellen verfasst zu haben.

Burgdorf, 1. Oktober 2012

Fabian Bürli

Florian Neuhaus

Moritz Kobel

Philipp Haslebacher

Samuel Hubacher

Simon Eggimann

Simon Kunz

A. Pflichtenheft

A.1. Analyse des technischen Prozesses: Ist-Aufnahme

A.1.1. Bestandsaufnahme

Beim Eurobot 2008 handelt es sich um einen Wettbewerb, dessen Ziel es ist, autonome Roboter herzustellen und zu programmieren. Die Berner Fachhochschule in Burgdorf nimmt mit diesem Projekt bereits zum dritten Mal am Wettbewerb teil.

Die Gruppe MarsRiders besteht aus zwei angehenden Maschineningenieuren und fünf angehenden Elektroingenieuren: Der zeitliche Rahmen während der Projektarbeit 1 beträgt 6

Name / Vorname	Klasse	Vertiefung / Bereich
Kunz Simon	E3B	Technische Informatik / Projektleiter
Kobel Moritz	E3B	Technische Informatik / Java
Neuhaus Florian	E3A	Technische Informatik / Java
Eggimann Simon	E3A	Mechatronik
Haslebacher Philipp	E3A	Mechatronik
Hubacher Samuel	M3A	Mechanik
Bürli Fabian	M3A	Mechanik

Tabelle A.1.: Ressorts

Lektionen während 16 Schulwochen (Donnerstag von 07:35 bis 12:55). In vier Sitzungen werden wir von Herrn Felser in die Thematik eingeführt. Um in der Freizeit am Projekt zu arbeiten, können wir beim Abwart einen Schlüssel beziehen.

Arbeitsort ist die BFH in Burgdorf, Gebäude Tiergarten (Labor 211, die Werkstätte oder die Kanzleräume). Um Sitzungen abzuhalten benutzen wir das Sitzungszimmer 008 im Erdgeschoss.

Arbeitsmittel, wie Computer, IT-Infrastruktur, Drucker und Messgeräte können wir aus dem Labor 211 benützen.

Der Roboter [kjū:] der Vorgänger dürfen wir demontieren und die einzelnen Teile weiterverwenden. Der ganze Roboter nur umzaprogrammieren macht keinen Sinn, da die Aufgabenstellung grundlegend geändert hat.

A.1.2. Aufgabenstellung

Die Aufgabenstellung der Projektarbeit ist im Dokument PA1_HS07_Eurobot V1_0.doc festgehalten. Die betreuenden Dozenten sind: Ivo Oesch, Daniel Lanz, Roger Weber und Fritz Bircher (Mechanik).

Die **Aufgabenstellung**¹ des Wettbewerbs findet man online.

Kurz zusammengefasst erstellt unser Team einen fahrbaren Roboter, der auf dem Spieltisch die erforderlichen Aktionen durchführt, um die Zulassung für den Wettbewerb zu erreichen. Auch gehören Projektplanung, sowie Dokumentation dazu.

A.1.3. Anforderungsliste

F: Forderung

W: Wunsch

¹http://www.eurobot.org/de/docs/2008/E2008_Rules_German%20version.pdf

Kat	Anforderungen qualitativ	quantitativ	Bemerkungen
Kernanforderungen			
F	Sortieren von versch. farbigen Unihockeybällen	Umfang \leq 1200mm	
F	Baugrösse ausgeschaltet	Umfang \leq 1400mm	
F	Baugröße in Betrieb	\leq 350mm	Ohne Bake Kontaktaufnahme zu Baken erlaubt
F	Bauhöhe		
F	Eigenständiges Agieren des Roboters		
F	Sicherheitsvorschriften einhalten		www.eurobot.org^a
F	Fairplay beachten		www.eurobot.org^b
F	Zwei Startpositionen		Zufällig vor dem Match zugewiesen
Anschlussbedingungen			
F	Transportierbar	80x80 mm in 430 mm	
F	Bakenhalterung	Höhe	
Betriebsbedingungen			
F	Start mittels Startseilnur, bleibt nicht auf dem Roboter	Min. 500 mm	
F	Notstop auf Oberseite	Min. ø 20mm	
F	Zeitschaltuhr zum selbstständigen Anhalten	90s	
F	Leicht austauschbare Akkus		Nur zugelassene Verwenden
F	Energiequelle		www.eurobot.org^c

Tabelle A.2.: Anforderungsliste

Kat	Anforderungen qualitativ	quantitativ	Bemerkungen
Herstellungsbedingungen			
F	Roboter fahrbar und Zulassung erreicht	17.01.2008	
Entwicklungsbedingungen			
W	Vorhandene Komponenten verwenden		
F	Dokumentation (Poster, Webseite)	Min. DIN A1	
F	Projektarbeit		
Wartungsbedingungen			
F	Gute Zugänglichkeit		
Verwertungsbedingungen			
W	Wiederverwendbarkeit der Komponenten		

A.2. Ressourcen und Infrastruktur

A.2.1. Räume, Dozenten und Personal

Als Arbeitsort dient das Labor 211 am Jlcoweg 1 in Burgdorf. Im Labor haben wir genügend Arbeitsplätze für die ganze Gruppe und um den Tisch aufzustellen. Die Messgeräte und Infrastruktur des Labor dürfen benutzt werden.

Die mechanische Werkstatt darf ebenfalls benutzt werden, sowie die Kanzelräume im ersten Stock. Die Werkstatt ist immer in sauberem und aufgeräumtem Zustand zu hinterlassen. Ansprechperson der Werkstatt ist Werner Reichen.

Die betreuenden Dozenten sind Ivo Oesch, Daniel Lanz, Roger Weber und Fritz Bircher (Mechanik). Während der Projektarbeit 2 betreuen uns zusätzliche Willi Merk und Gerhard Krucker.

Herr Erwin Stucki ist verantwortlich für die Bestellungen, die unser Team betreffen - ihm sind Bestellungen zu übergeben. Herr Bruno Niklaus ist Chef der Werkstatt der mechanischen Abteilung am Standort Gsteig. Er ist dort zuständig für die Benutzung der Werkstatt.

A.2.2. Dokumentation und Datenaustausch

Als Laufwerk zum Datenaustausch ist ein Webdav² eingerichtet. Alle erarbeiteten Unterlagen werden dort abgespeichert und sind für alle am Projekt beteiligten Personen zugänglich.

A.2.2.0.1. Empfohlene Software

- Novell NetDrive³
- Webdav Internet Explorer⁴
- Webdav für Vista⁵

Wir verwenden die Versionsverwaltung Subversion⁶. Für die Dokumentation verwenden wir LATEX. Das Dokument Installationsanleitung.pdf gibt weitere Infos zum einrichten der Umgebung unter Windows.

A.2.2.0.2. Empfohlene Software für Windows

- Miktex⁷
- TeXnicCenter⁸
- TortoiseSVN⁹

²<https://svn.itds.ch/eurobot>

³http://www.tecchannel.de/download/tools_utilities/treiber-tools/109253/novell_netdrive_client

⁴<http://www.uni-tuebingen.de/info-provider/webDAV-faq/webDAV-faq.html>

⁵<http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=17c36612-632e-4c04-9382-987622ed1d64>

⁶<http://subversion.tigris.org>

⁷<http://miktex.org/2.7/Setup.aspx>

⁸<http://www.toolscenter.org/21.html>

⁹<http://tortoisेस्वन.net/downloads>

A.2.2.0.3. Empfohlene Software für Linux

Die folgenden Programme sind als Debian & Ubuntu Pakete verfügbar.

- ViM
- Kile
- svn
- tex-live

Als Arbeitsjournal können wir auf unserer [Homepage](#)¹⁰ Einträge zu unseren Tätigkeiten verfassen.

Um eine einfache Kommunikation zu ermöglichen, verwenden wir eine [Mailingliste](#)¹¹, über die wir alle Teammitglieder erreichen können.

Auf der [Eurobot-Homepage](#)¹² der BFH können wir unsere Sponsoren und Interessierte über unser Geschehen informieren.

A.2.3. Mechanik

Die verwendete CAD-Software ist NX4.

A.2.4. Programmierumgebung Atmel

Für die Programmierung der Atmel-Boards verwenden wir das [AVR Studio 4](#)¹³. Zudem wird der Compiler [WinAVR](#)¹⁴ benötigt.

A.2.5. Programmierumgebung Java mit Eclipse

Als Programmieroberfläche für die Programmierung von Java auf dem Industrierechner, verwenden wir [Eclipse](#)¹⁵ mit dem [SVN Plugin Subclipse](#)¹⁶.

Auf dem Industrierechner läuft ein Skript das die neuste Software aus dem Subversion Repository (SVN) lädt, kompiliert und ausführt.

¹⁰<http://eurobot.itds.ch/>

¹¹<mailto:eurobot@lists.itds.ch>

¹²<http://eurobot.bfh.ch/>

¹³http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

¹⁴<http://sourceforge.net/projects/winavr>

¹⁵<http://www.eclipse.org/downloads/>

¹⁶<http://subclipse.tigris.org/>

A.3. Definition der Aufgaben

A.3.1. Mechanik

Während der Projektdefinitionsphase haben wir uns auf ein mechanisches Konzept geeinigt. Unser Roboter fährt wie letztes Jahr auf zwei Inlineskate-Rädern.

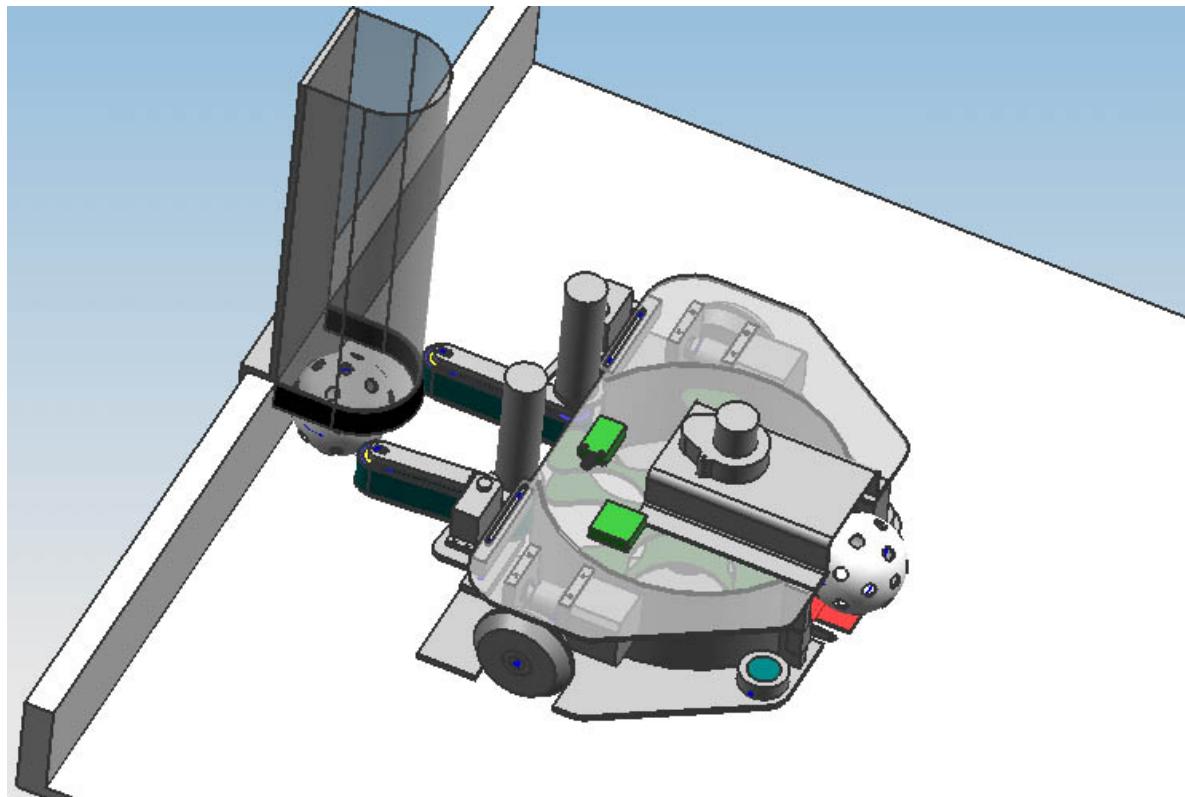


Abbildung A.1.: CAD-Zeichung Roboter

Da unsere Taktik das Aufnehmen von Unihockeybällen aus den vertikalen Spendern bevorzugt, nehmen wir die Bälle mit zwei beweglichen Förderbändern auf. Eine einfache Mechanik verhindert das Verklemmen bei den Spendern.

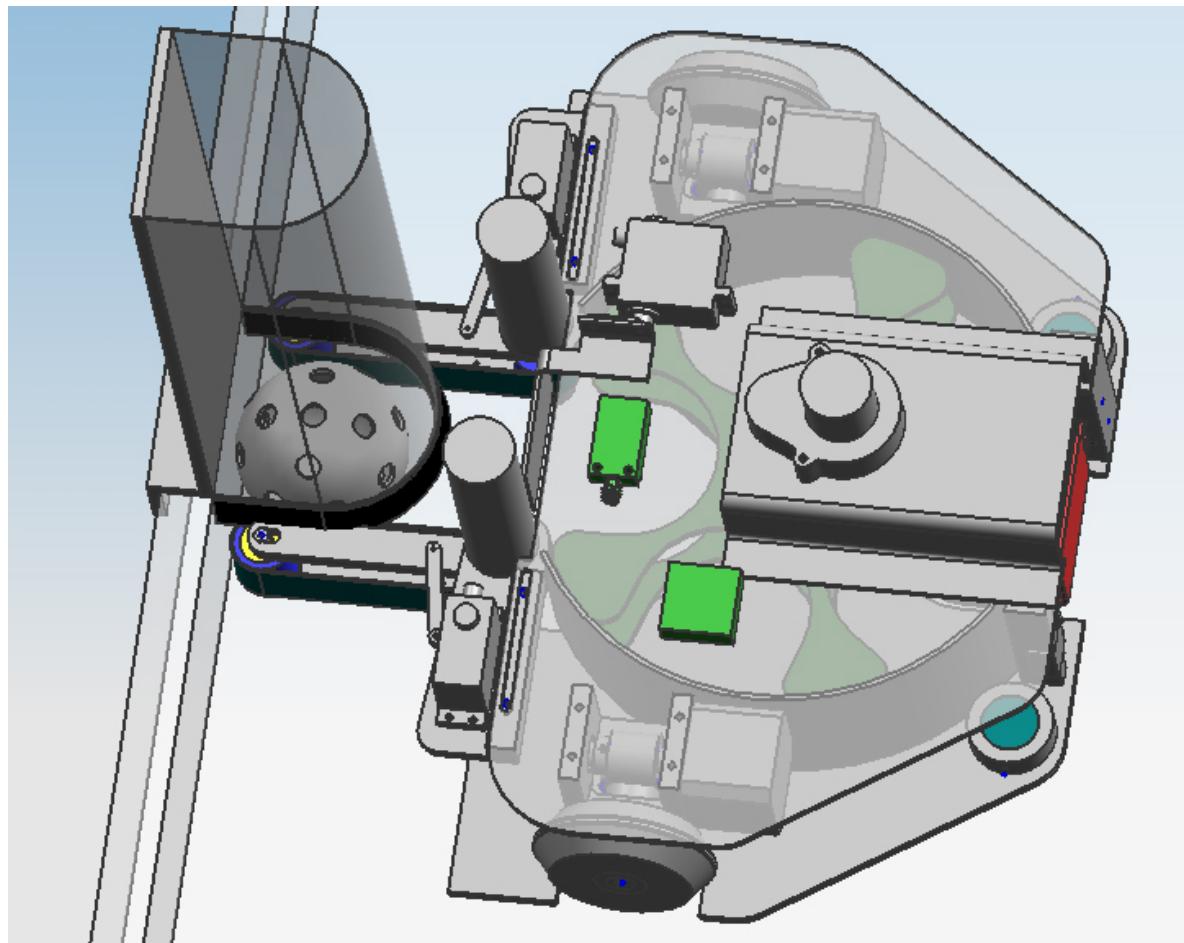


Abbildung A.2.: CAD-Zeichung Ballaufnahme

Bis zu fünf Bälle können wir in der Trommel speichern. Die Trommel hat nebst der Speicherung auch noch die Funktion, die Bälle vom Eingang zum Ausgang zu befördern. Eine Sortierung nach Farben ist ebenfalls vorgesehen.

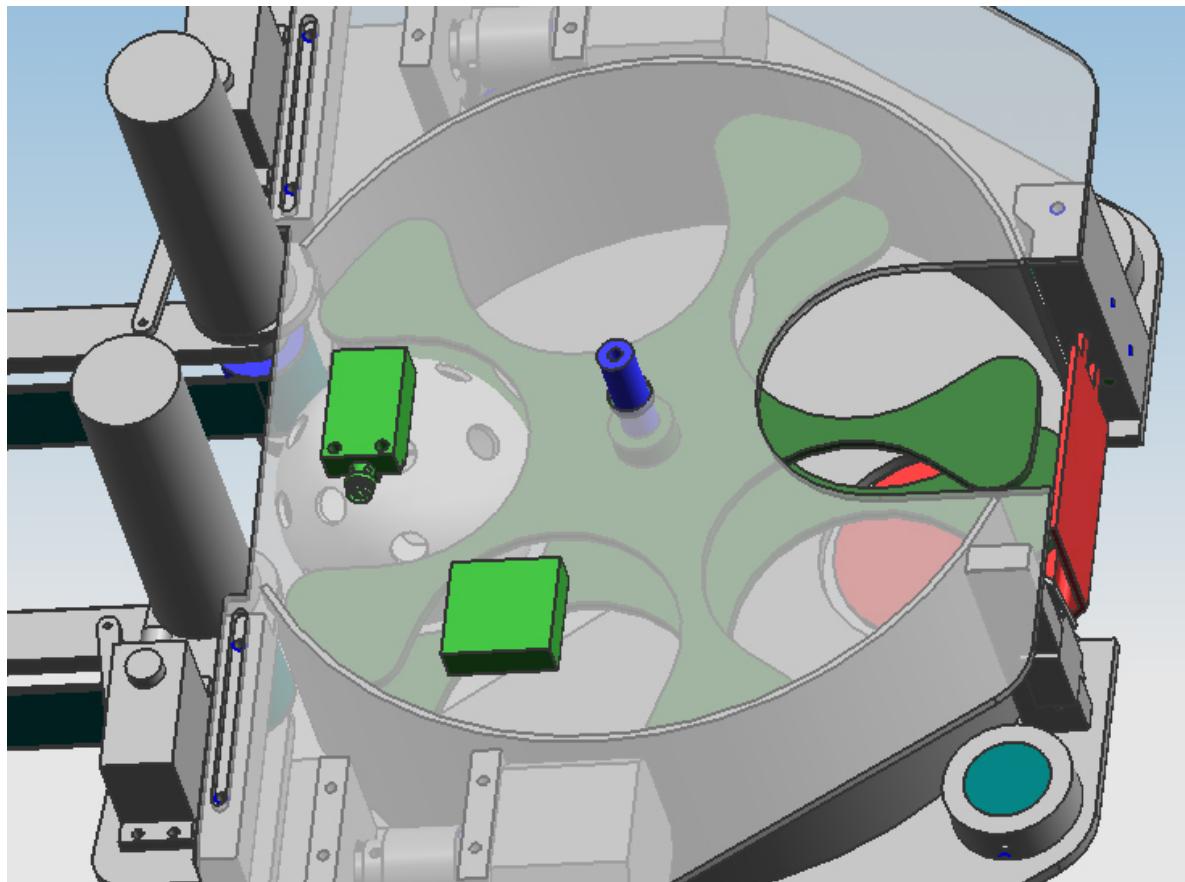


Abbildung A.3.: CAD-Zeichung Trommelförderer

Der Aufwurf des Balles in den Standard-Container erfolgt über eine Klappe. Diese kann bewegt werden. Dadurch wird der Ball um 2cm angehoben und rollt über den Tisch in den Standard-Container.

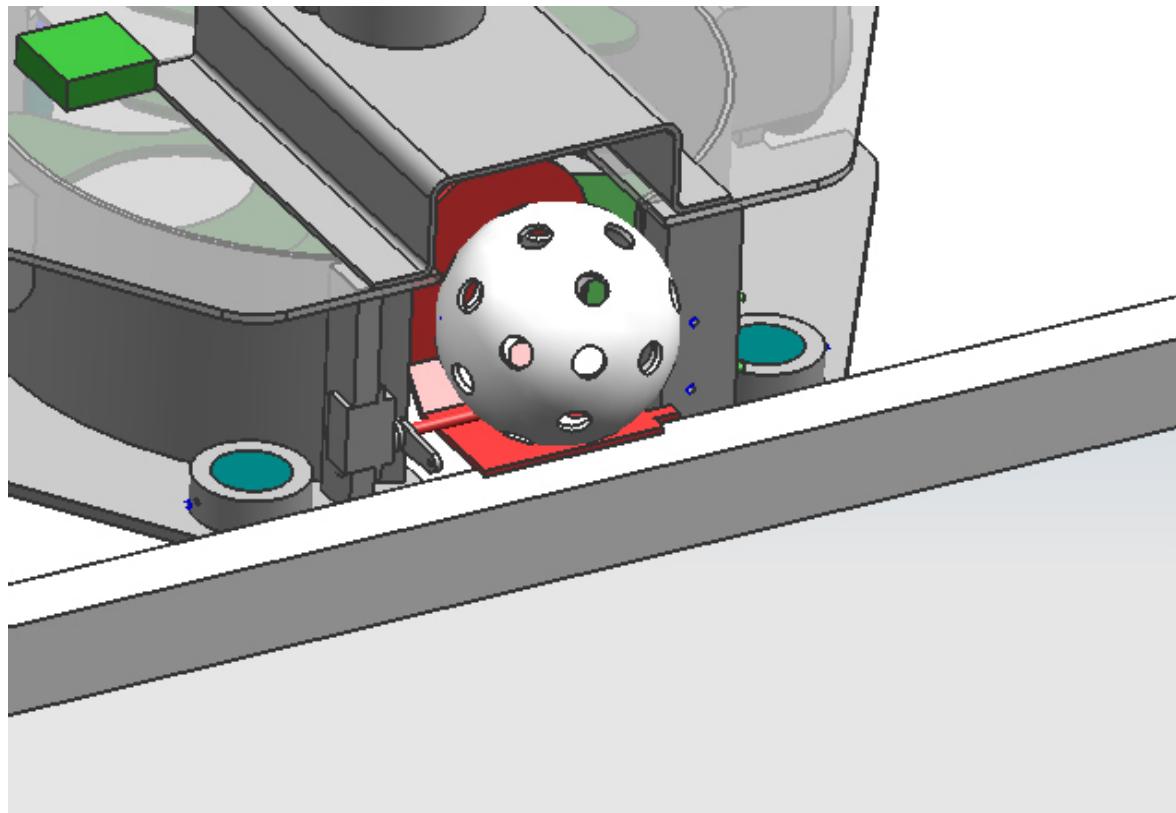


Abbildung A.4.: CAD-Zeichnung Auswurfmechanik

A.3.2. Hardware

A.3.2.0.4. Schematischer Aufbau

Abbildung A.5 zeigt den schematischen Aufbau der Hardwarekomponenten. Wir verwenden eine dezentrale Architektur der Komponenten. Die gesamte Berechnung und Auswertung übernimmt der Industrie-PC. Er kommuniziert über den CAN-Bus mit den zwei angeschlossenen Knoten (Atmel-Boards). Die Knoten setzen um Sensoren und Aktoren zu trennen, sowie eine günstige Arbeitsteilung zu ermöglichen.

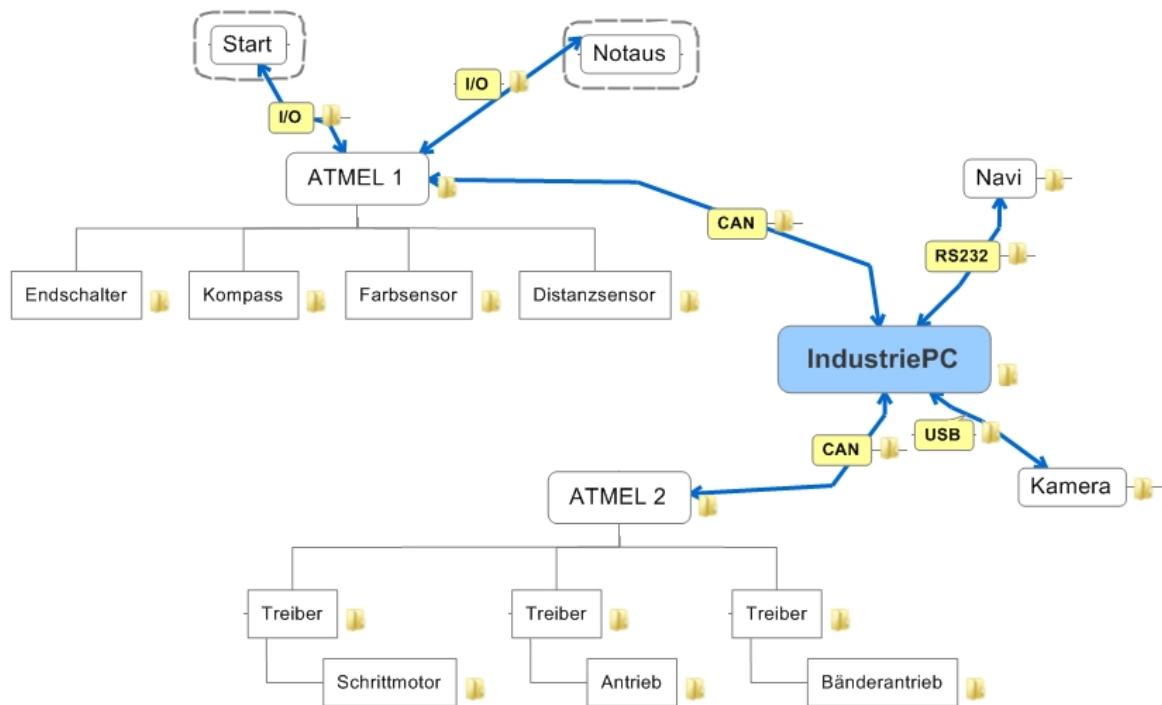


Abbildung A.5.: schematischer Aufbau der Hardware

A. Pflichtenheft

Folgende Bausteine wurden während der Projektphase evaluiert:

Baustein	Details	Bemerkungen	personelle Zuteilung
ATMEL 1 / Microcontroller Endschalter	DVK90CAN1 Entwicklungsboard Contrinex induktiv M4	Erweiterung I/O nötig (ADC, DI)	Haslebacher Philipp
Kompass oder Gyrometer	CMPS03	I2C - Bus	Haslebacher Philipp
Distanzsensor Farbsensor	GP2D12 Sharp Agilent HDJD-S831-QT333	Analogausgänge für R, G und B	Haslebacher Philipp
Startknopf	von Vorgänger übernommen		Haslebacher Philipp
Not-Aus-Schalter	von Vorgänger übernommen		Haslebacher Philipp
ATMEL 2 / Microcontroller	DVK90CAN1 Entwicklungsboard		Eggimann Simon
Antrieb 2 Schrittmotoren	Nanotec ST4118M1404	neu evaluiert	Eggimann Simon
Bänderantrieb 2 Bürstenmotoren	M28x20/S	von Vorgänger übernommen	Eggimann Simon
Trommel Schrittmotor	McLennan M82201-P2SB	von Vorgänger übernommen	Eggimann Simon
(Kamera)		wird in dieser Projektarbeit nicht benötigt.	
IndustriePC	von Vorgänger übernommen		Moritz Kobel / Florian Neuhaus
CAN-Dongle		wird im Labor verwendet	Moritz Kobel / Florian Neuhaus

A.3.2.0.5. Speisungskonzept



Abbildung A.6.: Speisungskonzept

A.3.3. Menschliche Eingriffe

Not-Aus-Schalter	Unterbrechung der Stromversorgung für alle Aktoren.
Start-Leine	Auslösen der Spielsequenz.
TCP / IP über Ether- net	ssh Verbindung als Monitoring für die Software während der Testphase

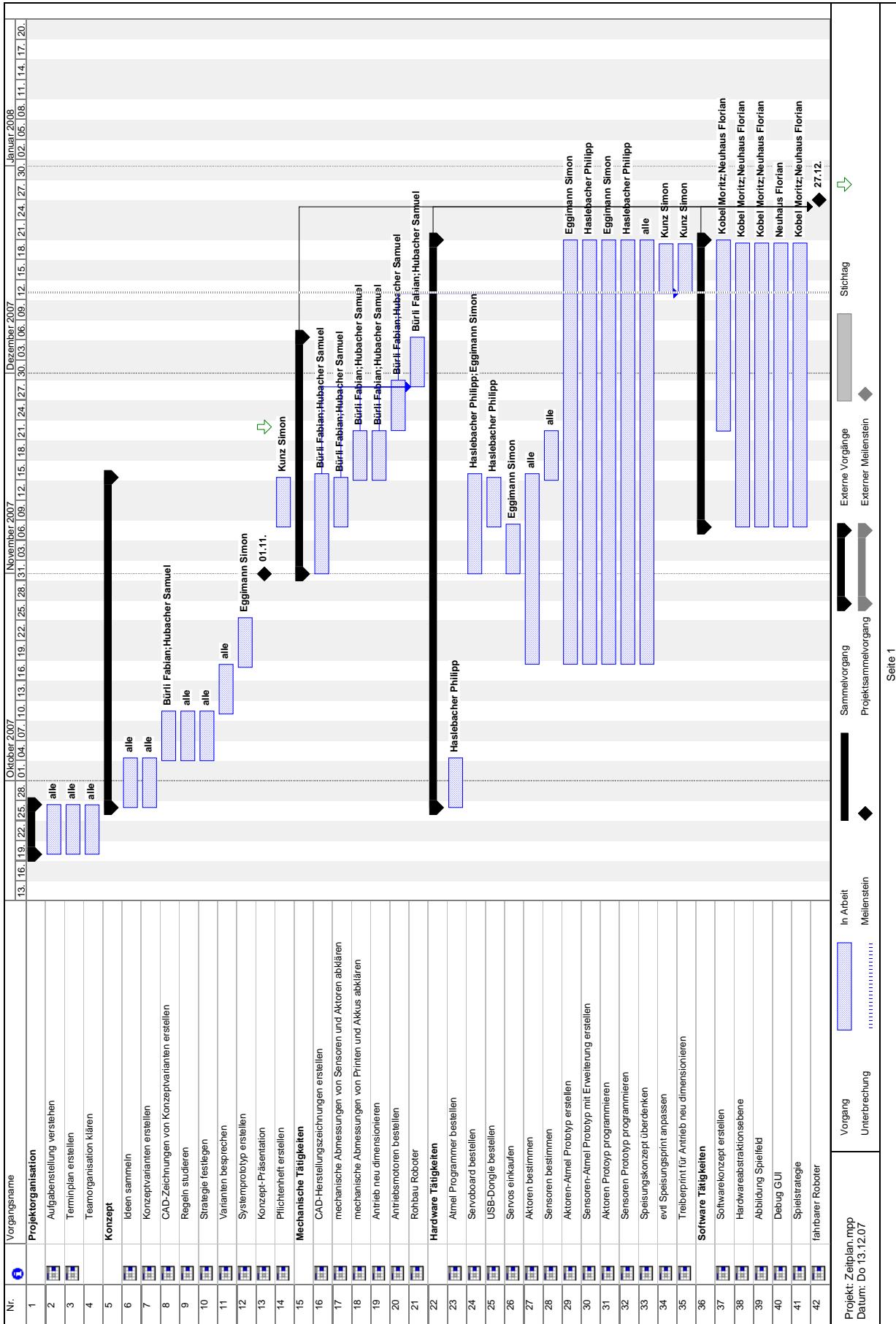
A.4. Umgebungsbedingungen: Betriebsbedingungen

Die Betriebsbedingungen für den Roboter sind einwandfreies Funktionieren bei Raumtemperatur und normaler Feuchtigkeit. Am Wettbewerb ist mit einem hohen Lärmpegel sowie einer starken Bestrahlung durch Lampen zu rechnen. Die Einflüsse von Licht und Schall sollten die Arbeitsweise des Roboters nicht beeinträchtigen.

A.4.1. Angaben zum Projektablauf

A.4.2. Zeitplan

Siehe dazu die folgende Abbildung auf der folgenden Seite.



B. Anleitungen

B.1. AT90CAN128 Debugging on Vista

AT90CAN128 Debugging on Vista

MarsRiders

Im folgenden wird erklärt, wie sich der Mikrokontroller AT90CAN128 von Atmel in Eclipse unter Vista debuggen lässt.

1. AVR-Eclipse installieren

Plugin von <http://avr-eclipse.sourceforge.net/> installieren. Dabei kann bei Eclipse über Help, Software Updates, Find and install gegangen werden. Hier als Remote site folgendes eintragen:

`http://avr-eclipse.sourceforge.net/updatesite/`

2. libusb-win32-device-bin-0.1.12.1.tar.gz herunterladen

libusb für Windows herunterladen, aber nicht installieren. Das File `libusb-win32-device-bin-0.1.12.1.tar.gz` von hier herunterladen und irgendwo entpacken.

http://sourceforge.net/project/showfiles.php?group_id=78138

Das JTAG-Device anschliessen und einschalten. Die Treiber noch nicht installieren!

Den `inf-wizard` aus `libusb-win32-device-bin-0.1.12.1.tar-1\libusb-win32-device-bin-0.1.12.1\bin` starten. Jetzt JTAGICE wählen und die files nach `C:\Windows\inf` speichern.

`libusb0.dll` und `libusb0.sys` nach `C:\Windows\inf` kopieren.

Jetzt die Treiber (im Gerätetypenmanager) für das JTAG-Device aus `C:\Windows\inf` installieren.

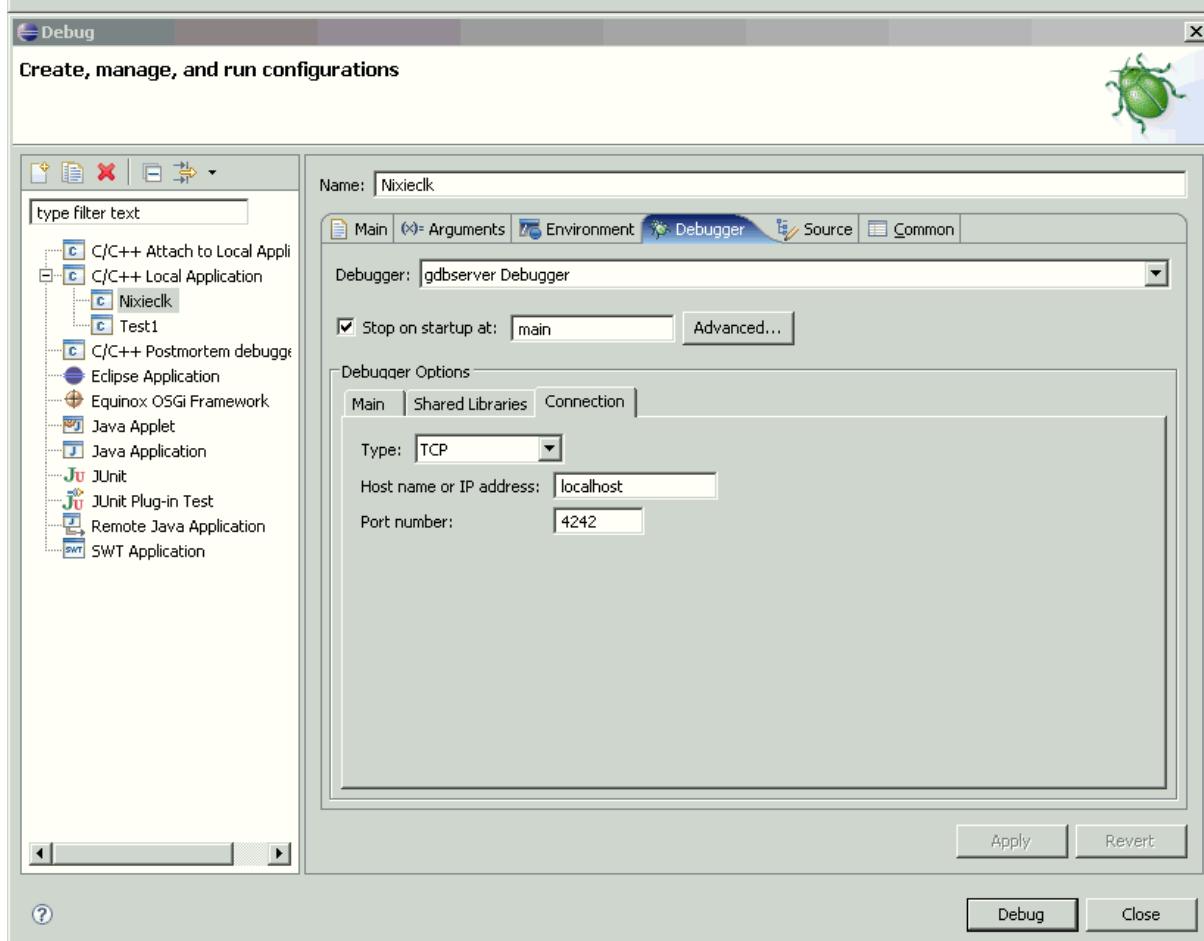
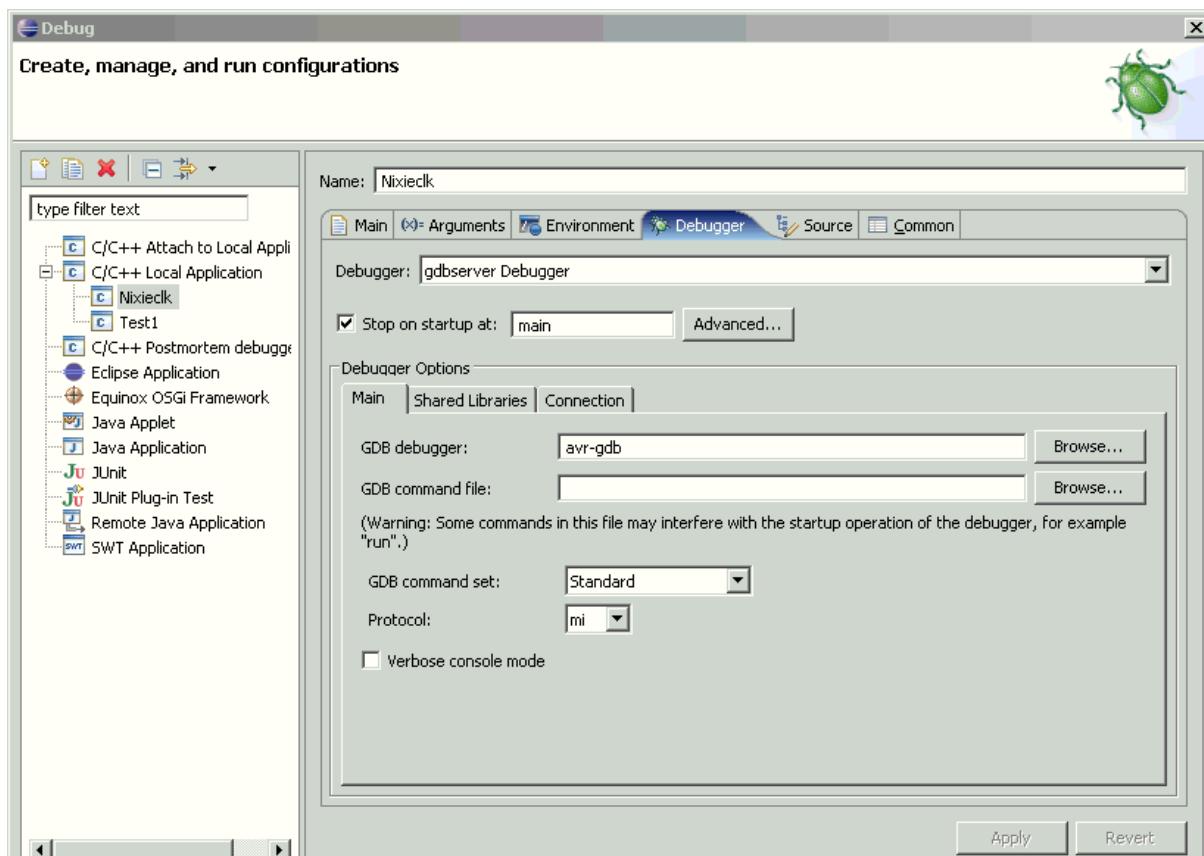
3. Neues Eclipse Projekt erstellen

Am besten in einem neuen Workspace ein C-Projekt erstellen. Dabei die Settings wie hier wählen:

<http://tinkerlog.com/2007/08/01/avr-plugins-for-eclipse/>

Die uCOS-Files ins Stammverzeichnis des Projekts kopieren. Einen neuen Debug-Dialog erstellen (Open Debug Dialog). Die folgenden Settings dafür verwenden (siehe Bilder):

Eurobot 2008



Eurobot 2008

External Tools

Create, manage, and run configurations

Run a program

Name: AVARICE Programming Device USB

Main Refresh Environment Common

Location: C:\Programme\WinAVR-20070525\bin\avarice.exe

Working Directory: \${workspace_loc}\${project_path}/DEBUG

Arguments: -e -p -f \${project_name}.hex -2 -P atmega32 --jtag usb

Note: Enclose an argument containing spaces using double-quotes ("").

Variables... Apply Revert Run Close

External Tools

Create, manage, and run configurations

Run a program

Name: AVARICE Debugging USB

Main Refresh Environment Common

Location: C:\Programme\WinAVR-20070525\bin\avarice.exe

Working Directory: \${workspace_loc};\${project_name}/Debug

Arguments: -2 -P atmega32 --jtag USB :4242

Note: Enclose an argument containing spaces using double-quotes ("").

Variables... Apply Revert Run Close



Zum debuggen muss nun zuerst das Tool „AVARICE Debugging USB“ gestartet werden. Danach kann über den Debug-Dialog mit dem Debugging gestartet werden.

B.2. Bedienungsanleitung Roboter

Bedienungsanleitung Roboter

MarsRiders – B.I.T.T.

1.	Booting	1
2.	Einloggen	1
3.	Starten.....	1
4.	Fehlerquellen.....	2

1. Booting

1. 36V Speisung anschliessen (Akku)
2. Navigationssystem hochfahren (9V Speisung der Baken aktivieren, evtl. Schalter betätigen)
3. 24V Speisung anschliessen (Akku oder Netzgerät)
4. Überprüfen ob alle Lüfter (CPU, Motorentreiber) laufen.
5. Servoboard in Position „OFF“ bringen, danach „ON“.
6. Beide Atmel-Boards resetten.
7. Boxen anschalten ;)

2. Einloggen

Computer entweder per LAN-Kabel oder per WLAN über VPN verbinden. Jetzt per Putty oder ssh auf den Roboter einloggen:

eurobot.kicks-ass.org:22

User: root

Pass: eurobot

3. Starten

1. Alle Bälle entfernen (siehe auch Startoptionen).
2. Trommel auf Einladeposition bringen.
3. Bälle auffüllen.
4. Notstop kontrollieren.
5. Roboter bündig zur unteren Tischkante positionieren.

Neuste Software-Version auschecken. Dazu ins eurobot Verzeichnis wechseln:

```
cd eurobot  
./getAndBuild.sh
```

Java Programm starten. Dies wird über das Script „run.sh“ erledigt. Diesem Script wird die Main-Klasse des Java-Programms übergeben.

Folgende Startoptionen der Java-Klasse sind möglich:

```
Usage: eurobot.Run [OPTION] STRATEGY COLOR [TESTCLASS] \n
Run the robot. Color and strategy always required.

-h, --help           displays this help
-d, --debug          no function at the moment
-v, --virtualbot    run the non real eurobot for testing
                     purpose
-p, --potiEnable     enable the drum-correction with poti
-e, --eject          ejects all balls before starting. Drum
                     has to be in a collect position!
-g, --goHome         try to go home after 90seconds. You have
                     to pull the start-line again.
-t, --tts <voiceName> use another voice for text-to-speech
                     (standard kevin)
-l, --logLevel <debug/info/warn/error/fatal> set the debug level.
                     Option "debug" logs all messages.
-r, --runStrategy   run the strategy after initializing.
-s, --strategy <complex/simple> set the strategy to use. Standard
                     is "complex"
-c, --color <red/blue> set the color of OUR robot
<eurobot.test.TestClassName> test-class to be used. Can be empty.
```

Examples:

```
eurobot.Run -c red -s complex -l info -r
```

Standard-Start (Farbe anpassen):

```
./run.sh „eurobot.Run -c red -s complex -l info -r“
```

4. Fehlerquellen

Symptom: Position wird nicht erkannt. Roboter fährt einfach immer weiter geradeaus. Gegner wird nicht erkannt.

Abhilfe: Kanäle des Navigationssystems überprüfen. Speisung überprüfen. Evtl. Batterie zu schwach.

Symptom: Trommel dreht bei der Initialisierung unkontrolliert. Korrigiert ewig.

Abhilfe: Sensorenboard resetten. Java neu starten. Programm ohne Poti laufen lassen (Option -p weglassen). Dabei aber beachten, dass eine Einladeposition beim Starten gewählt wird.

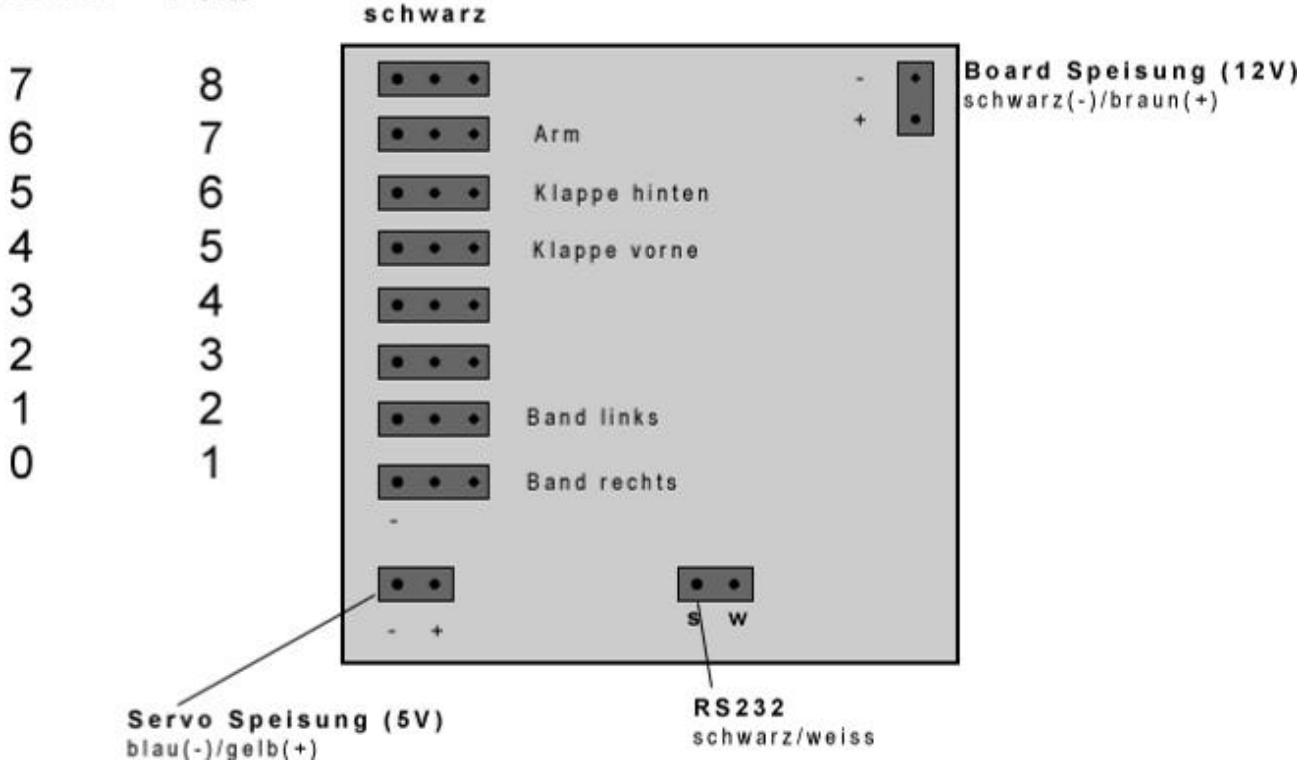
Symptom: Roboter bleibt einfach stehen.

Abhilfe: Log in der Konsole anschauen. Wahrscheinlich ein CAN-Bus Error. Evtl. ist eines der Atmel-Boards abgeschmiert. Das Sensor-Board blinkt kontinuierlich, wenn es läuft. Wenn ein CAN-Error vorliegt, dann leuchtet evtl. ein orangenes Lämpchen beim CAN-Controller auf dem Atmel.

B.3. Anschlussbelegung Servo-Board

Servo Board Pinbelegung Team MarsRiders

RS232 Ports



C. Resultate Swisseurobot

Nr	Team	Ort	Nº Stand	Homologation	Points
1	Arc'obot	Saint-Imier		OK	88
2	CRH	Yverdon		OK	4
3	CVRA	Estavayer-le-lac		OK	54
4	Deimos	Biel		OK	92
5	DOMBOT Rapperswil	Rapperswil		OK	61
6	e-robot	Saint-Imier		OK	34
7	EcubeMsquare	Rapperswil		OK	120
8	ETML NO8	Lausanne		NOT OK	0
9	Funky Donkey Rapperswil	Rapperswil		OK	111
10	Icare	Petit-Lancy		OK	161
11	MarsRiders	Burgdorf		OK	88
12	oneTeam	Burgdorf		OK	99
13	Phobos	Biel		OK	202
14	RCR	Rapperswil		NOT OK	0
15	Robopoly	Lausanne		NOT OK	0
16	Team Trancezistor	Moutier		NOT OK	0
17	Viper	Winterthur		OK	117
18	Zeptox	Satigny		NOT OK	0

Round 01

		red	VS	blue		
start	Table	Team	Points	Team	Points	
14:45	A	EcubeMsquare	5	VS	oneTeam	14
14:50	B	DOMBOT Rapperswil	7	VS	Phobos	31
14:55	A	Icare	17	VS	Deimos	1
15:00	B	Funky Donkey Rapperswil	11	VS	Viper	1
15:05	A	e-robot	3	VS	Arc'obot	17
15:10	B	MarsRiders	0	VS		
15:15	A			VS		
15:20	B			VS		
15:25	A			VS		

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 02

		red	VS		blue	
start	Table	Team	Points	VS	Team	Points
16:00	A	DOMBOT Rapperswil	3	VS	oneTeam	13
16:05	B	Funky Donkey Rapperswil	10	VS	Deimos	14
16:10	A	MarsRiders	0	VS	Arc'obot	12
16:15	B	Viper	10	VS	e-robot	1
16:20	A	Phobos	32	VS	Icare	27
16:25	B	EcubeMsquare	17	VS	CVRA	7
16:30	A			VS		
16:35	B			VS		
16:40	A			VS		

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 03

		red		VS		blue	
start	Table	Team		Points	VS	Team	Points
17:00	A	e-robot		1	VS	CVRA	13
17:05	B	DOMBOT Rapperswil		1	VS	MarsRiders	13
17:10	A	Icare		34	VS	Arc'obot	4
17:15	B	Viper		11	VS	Phobos	37
17:20	A	Deimos		9	VS	EcubeMsquare	18
17:25	B	oneTeam		21	VS	Funky Donkey Rapperswil	25
17:30	A				VS		
17:35	B				VS		
17:40	A				VS		

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 04

		red	VS		blue	
start	Table	Team	Points	VS	Team	Points
19:30	A	DOMBOT Rapperswil	5	VS	Deimos	25
19:35	B	Phobos	31	VS	Arc'obot	1
19:40	A	MarsRiders	0	VS	EcubeMsquare	19
19:45	B	Viper	18	VS	oneTeam	10
19:50	A	Icare	-1	VS	e-robot	1
19:55	B	CVRA	1	VS	Funky Donkey Rapperswil	16
20:00	A			VS		
20:05	B			VS		
20:10	A			VS		

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 05

		red	VS	blue	
start	Table	Team	Points	Team	Points
20:30	A	Icare	29	VS MarsRiders	1
20:35	B	Phobos	1	VS e-robot	12
20:40	A	Deimos	12	VS Viper	7
20:45	B	EcubeMsquare	19	VS Arc'obot	1
20:50	A	CVRA	1	VS oneTeam	9
20:55	B	CRH	0	VS Funky Donkey Rapperswil	11
21:00	A	DOMBOT Rapperswil	1	VS	
21:05	B			VS	
21:10	A			VS	

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 06

		red	VS	blue	
start	Table	Team	Points	Team	Points
09:30	A	Viper	4	Arc'obot	21
09:35	B	Icare	17	Funky Donkey Rapperswil	3
09:40	A	CRH	1	e-robot	1
09:45	B	CVRA	12	Phobos	23
09:50	A	EcubeMsquare	8	DOMBOT Rapperswil	16
09:55	B	MarsRiders	20	oneTeam	14
10:00	A	Deimos	25	VS	
10:05	B			VS	
10:10	A			VS	

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 07

		red	VS	blue	
start	Table	Team	Points	Team	Points
10:10	A	e-robot	1	DOMBOT Rapperswil	14
10:15	B	Icare	17	oneTeam	17
10:20	A	Phobos	1	CRH	1
10:25	B	Deimos	1	MarsRiders	27
10:30	A	Arc'obot	1	CVRA	17
10:35	B	Viper	29	EcubeMsquare	8
10:40	A	Funky Donkey Rapperswil	0	VS	
10:45	B			VS	
10:50	A			VS	

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 08

		red	VS		blue	
start	Table	Team	Points	VS	Team	Points
10:50	A	DOMBOT Rapperswil	13	VS	Viper	19
10:55	B	MarsRiders	27	VS	CVRA	1
11:00	A	Phobos	17	VS	Deimos	5
11:05	B	Arc'obot	14	VS	Funky Donkey Rapperswil	9
11:10	A	oneTeam	0	VS	CRH	1
11:15	B	EcubeMsquare	10	VS	Icare	24
11:20	A	e-robot	0	VS		
11:25	B			VS		
11:30	A			VS		

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

Round 09

		red	VS		blue	
start	Table	Team	Points	VS	Team	Points
11:30	A	Arc'obot	17	VS	oneTeam	1
11:35	B	DOMBOT Rapperswil	1	VS	Icare	-3
11:40	A	CVRA	2	VS	Viper	18
11:45	B	CRH	1	VS	Deimos	0
11:50	A	MarsRiders	0	VS	e-robot	14
11:55	B	EcubeMsquare	16	VS	Funky Donkey Rapperswil	26
12:00	A	Phobos	29	VS		
12:05	B			VS		
12:10	A			VS		

start = heure du début du match, la préparation se fait avant

start = Zeit wenn das Spiel anfängt, ohne Vorbereitungszeit.

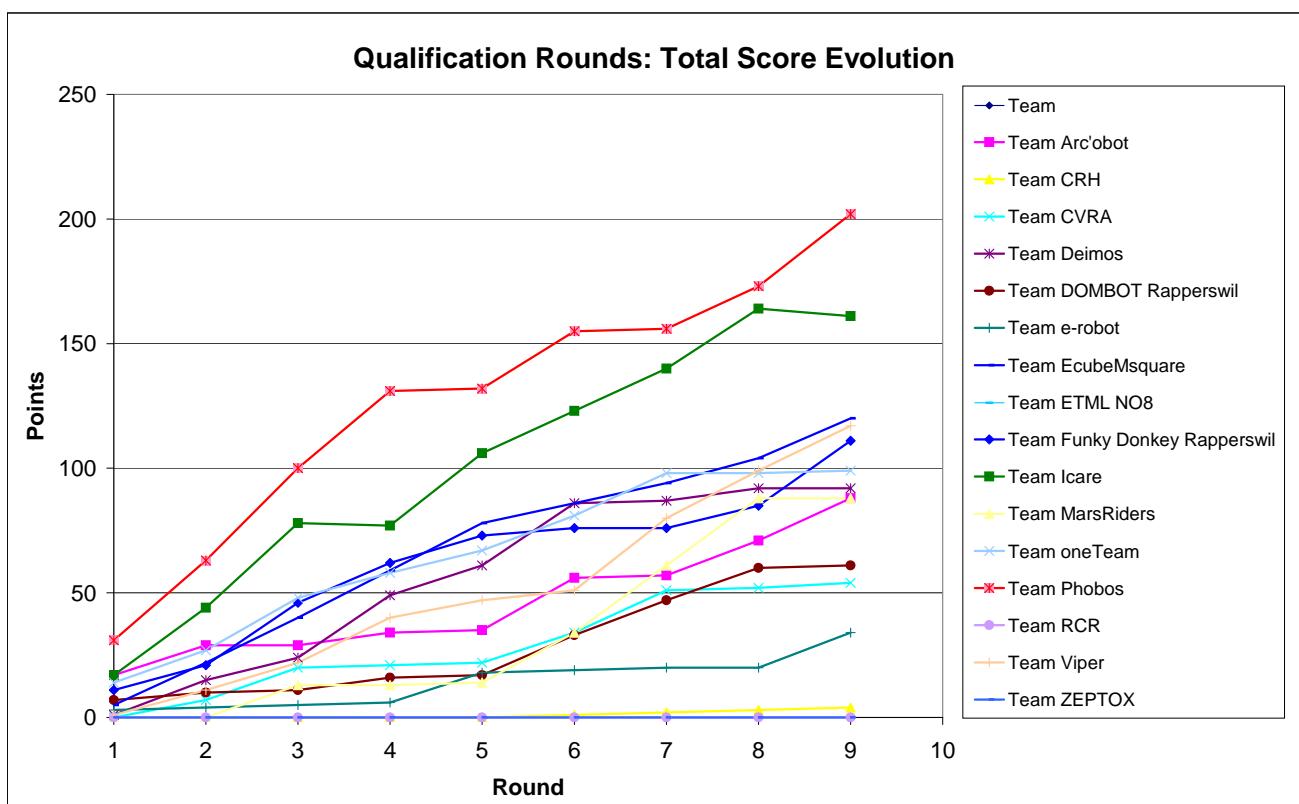
	Arc'obot	CRH	CVRA	Deimos	DOMBOT	e-robot	EcubeMsquare	ETML N08	Funky Donkey	Icare	MarsRiders	one Team	Phobos	RCR	Robopoly	TRANCEZISTOR	Viper	ZEPTOX	TOTAL
Arc'obot	0 0	1 17	0 0	0 0	17 3	1 19	0 0	14 9	4 34	12 0	17 1	1 31	0 0	0 0	0 0	21 4	0 0	88	
CRH		0 0	1 0	0 0	1 1	0 0	0 0	0 11	0 0	0 0	1 0	1 1	0 0	0 0	0 0	0 0	0 0	4	
CVRA	17 1		0 0	0 0	13 1	7 17	0 0	1 16	0 0	1 27	1 9	12 23	0 0	0 0	0 0	2 18	0 0	54	
Deimos		0 1		25 5	0 0	9 18	0 0	14 10	1 17	1 27	0 0	5 18	0 0	0 0	0 0	12 7	0 0	92	
DOMBOT			5 25		14 1	16 8	0 0	0 0	1 -3	1 13	3 13	7 31	0 0	0 0	0 0	0 0	13 19	0 0	61
e-robot	3 17	1 1	1 13		1 14		0 0	0 0	0 1	-1 14	0 0	12 1	0 0	0 0	0 0	0 0	1 10	0 0	34
EcubeMsquare	19 1		17 7	18 9	8 16		0 0	16 26	10 24	19 0	5 14	0 0	0 0	0 0	0 0	0 0	8 29	0 0	120
ETML N08								0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0
Funky Donkey	9 14	11 0	16 1	10 14			26 16			3 17	0 0	25 21	0 0	0 0	0 0	0 0	11 1	0 0	111
Icare	34 4			17 1	-3 1	-1 1	24 10		17 3		29 1	17 17	27 32	0 0	0 0	0 0	0 0	0 0	161
MarsRiders	0 12		27 1	27 1	13 1	0 14	0 19			1 29		20 14	0 0	0 0	0 0	0 0	0 0	0 0	88
one Team	1 17	0 1	9 1		13 3		14 5		21 25	17 17	14 20		0 0	0 0	0 0	0 0	10 18	0 0	99
Phobos	31 1	1 1	23 12	18 5	31 7	1 12				32 27				0 0	0 0	0 0	37 11	0 0	##
RCR																0 0	0 0	0 0	0
Robopoly																0 0	0 0	0 0	0
TRANCEZISTOR																0 0	0 0	0 0	0
Viper	4 21		18 2	7 12	19 13	10 1	29 8		1 11			18 10	11 37						117
ZEPTOX																		0	
Virtual Robot				25	1	0		0		0		29						0	
TOTAL	88	4	54	92	61	34	120	0	111	161	88	99	203	0	0	0	117	0	117

Nr	Team	Homologation	Round 01	Round 02	Round 03	Round 04	Round 05	Round 06	Round 07	Round 08	Round 09	TOTAL
		Points	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points
	Arc'obot	1	17	12	4	1	1	21	1	14	17	88
	CRH	1					0	1	1	1	1	4
	CVRA	1		7	13	1	1	12	17	1	2	54
	Deimos	1	1	14	9	25	12	25	1	5	0	92
	DOMBOT Rapperswil	1	7	3	1	5	1	16	14	13	1	61
	e-robot	1	3	1	1	1	12	1	1	0	14	34
	EcubeMsquare	1	5	17	18	19	19	8	8	10	16	120
	ETML NO8											0
	Funky Donkey Rapperswil	1	11	10	25	16	11	3	0	9	26	111
	Icare	1	17	27	34	-1	29	17	17	24	-3	161
	MarsRiders	1	0	0	13	0	1	20	27	27	0	88
	oneTeam	1	14	13	21	10	9	14	17	0	1	99
	Phobos	1	31	32	37	31	1	23	1	17	29	202
	R2R											0
	Robopoly											0
	TRANCEZISTOR											0
	Viper	1	1	10	11	18	7	4	29	19	18	117
	ZEPTOX											0
	Virtual Robot	0										0
												0
												0
												0
												0

4 pts Sieg
2 pts Gleichstand
1 pt Verloren
0 pt Forfait

1 pt 1 weisser Ball
1 pt 1 Penalti
1 pt bonus

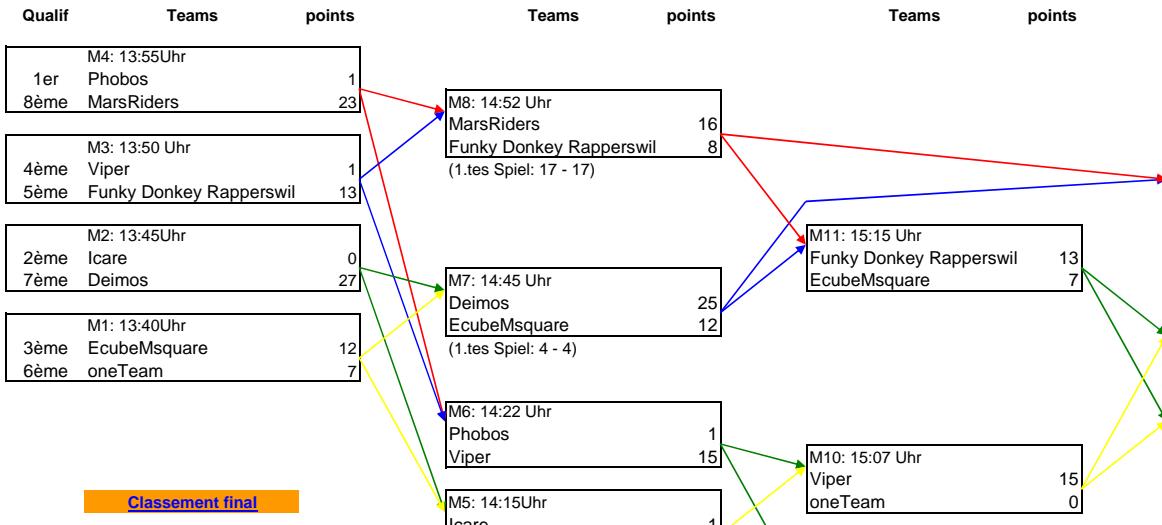
Diagramm: Points in Qualification Rounds



Rang	Nr.	Team	Points
1	13	Phobos	202
2	10	Icare	161
3	7	EcubeMsquare	120
4	17	Viper	117
5	9	Funky Donkey Rapperswil	111
6	12	oneTeam	99
7	4	Deimos	92
8	11	MarsRiders	88
9	1	Arc'obot	88
10	5	DOMBOT Rapperswil	61
11	3	CVRA	54
12	6	e-robot	34
13	2	CRH	4

Classement

Rang	N° équipe	Nom équipe	Points	Coeff
1er	13	Phobos	202	1
2ème	10	Icare	161	2
3ème	7	EcubeMsquare	120	3
4ème	17	Viper	117	4
5ème	9	Funky Donkey Rapperswil	111	5
6ème	12	oneTeam	99	6
7ème	4	Deimos	92	7
8ème	EGALITE	EGALITE	88	8
9ème	EGALITE	EGALITE	88	9
10ème	5	DOMBOT Rapperswil	61	10
11ème	3	CVRA	54	11
12ème	6	e-robot	34	12
13ème	2	CRH	4	13
14ème	EGALITE	EGALITE	0	14
15ème	EGALITE	EGALITE	0	15
16ème	EGALITE	EGALITE	0	16
17ème	EGALITE	EGALITE	0	17
18ème	EGALITE	EGALITE	0	18
19ème	EGALITE	EGALITE	0	19
20ème	EGALITE	EGALITE	0	20
21ème	EGALITE	EGALITE	0	21
22ème	EGALITE	EGALITE	0	22
23ème	EGALITE	EGALITE	0	23



Rahmenprogramm/Speeches:

13:30 Uhr Speech: Heinz Domeisen
 14:00 Uhr Speech: Herr Zibung (Eröffnung Halbfinals)
 14:30 Uhr Speech: Herr Würth

Teams

points

Teams

points

Teams

points

M14: 15:40 Uhr MarsRiders Deimos	27 3
--	---------

M15: 15:50 Uhr Deimos Viper	12 1
-----------------------------------	---------

Finale: 16:00 Uhr MarsRiders Deimos	27 3
---	---------

M13: 15:32 Uhr Funky Donkey Rapperswil Viper	15 30
--	----------

M15: 15:50 Uhr Deimos Viper	12 1
-----------------------------------	---------

M12: 15:25 Uhr EcubeMsquare oneTeam	16 32
---	----------

Rahmenprogramm/Speeches:
 16:00 Uhr Finale
 16:10 Uhr Wettbewerb/Competition Guests
 16:15 Uhr Apero for the Teams, in Pit-Lane

7ème
8ème

EUROBOT Schweizermeisterschaften

Rapperswil

Final rank list

Rank	Team Name	Place	Special
1st	MarsRiders	Burgdorf	--> Eurobot Heidelberg
2nd	Deimos	Biel	--> Eurobot Heidelberg
3rd	Viper	Winterthur	--> Eurobot Heidelberg
4th	Funky Donkey Rapperswil	Rapperswil	
5th	oneTeam	Burgdorf	
6th	EcubeMsquare	Rapperswil	
7th	Phobos	Biel	
8th	Icare	Petit-Lancy	--> Prize: SmartRob
9th	Arc'obot	Saint-Imier	
10th	DOMBOT Rapperswil	Rapperswil	
11th	CVRA	Estavayer-le-lac	
12th	e-robot	Saint-Imier	
13th	CRH	Yverdon	
	ETML NO8	Lausanne	
	RCR	Rapperswil	
	Robopoly	Lausanne	
	Team Trancezistor	Moutier	
	Zeptox	Satigny	

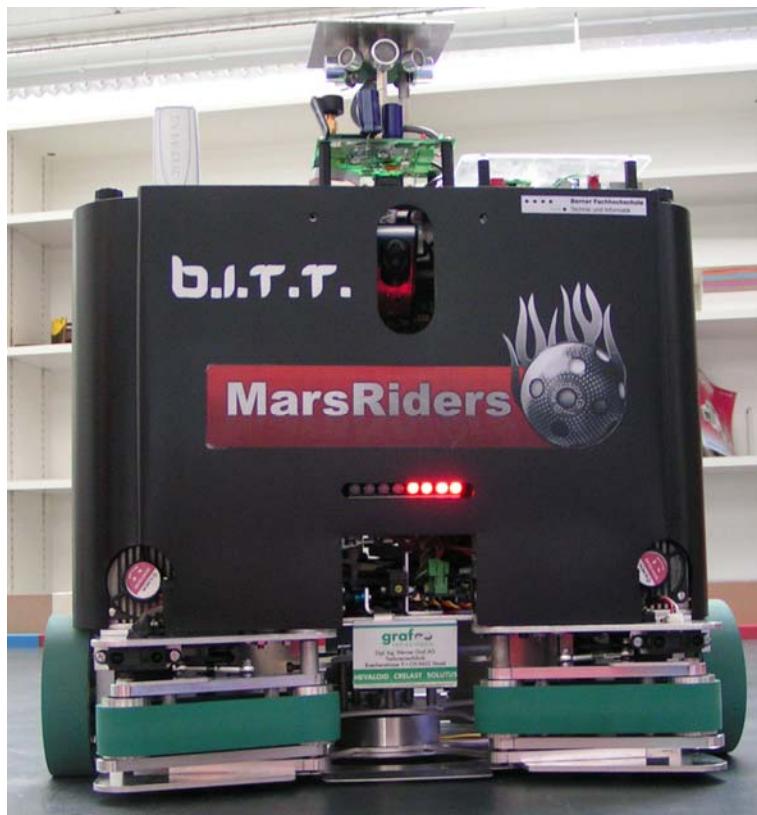
D. Newsletter Team MarsRiders



Newsletter MarsRiders vom 15.Mai 2008

Wie wir im letzten Newsletter angekündigt haben wir unseren neuen Antrieb in Betrieb genommen. Wir erreichen nun eine Geschwindigkeit von 1m/s. Damit rasen wir deutlich schneller über den Tisch! Präzision haben wir dadurch keine verloren. Anstatt des ungenauen Kompasses verwenden wir ein einfaches Gyroskope, das den genauen Winkel mit dem wir zum Tisch stehen bestimmt.

Um die Zulassung zu bestehen müssen wir davon ausgehen das unser B.I.T.T die Farbe der Bälle erkennen kann. Dies haben wir mittels eines RGB-Sensor realisiert. In den letzten Tagen vor dem Wettbewerb testen wir noch die Bälle die auf dem Tisch herumliegen mittels Bilderkennung anzufahren und aufzunehmen. Somit haben wir die Möglichkeit wenn uns der Gegner den Weg verstellt Bälle auf dem Spielfeld zu Sammeln.



Heute Morgen wurde das Schutzgehäuse von den Maschinentechnik-Studenten schwarz gesprayt und mit den Sponsorenklebern verziert.

Zur Zeit verpacken wir unser Ersatzmaterial, Werkzeuge und Messgeräte

Wir möchten uns bei unseren Sponsoren bedanken für ihre Dienstleistung. Unsere Arbeit wurde erheblich erleichtert.

Team Marsriders

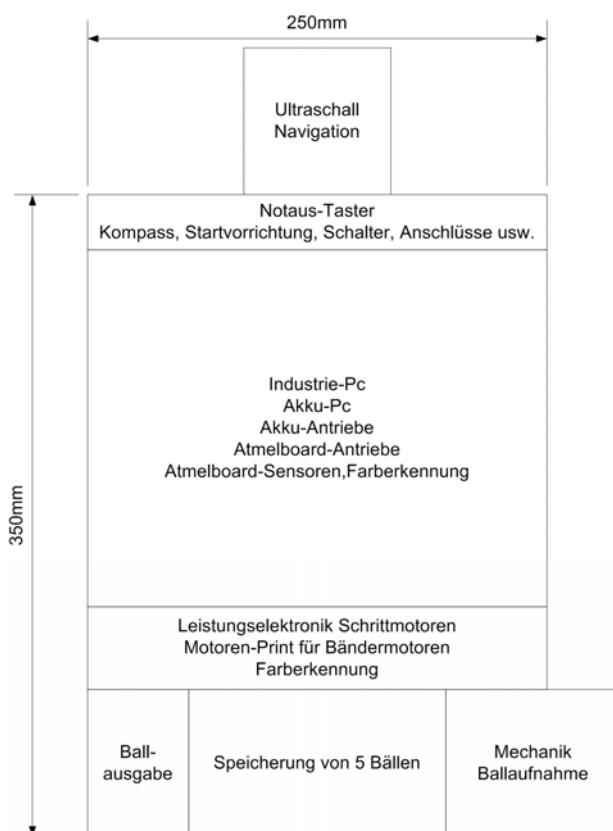
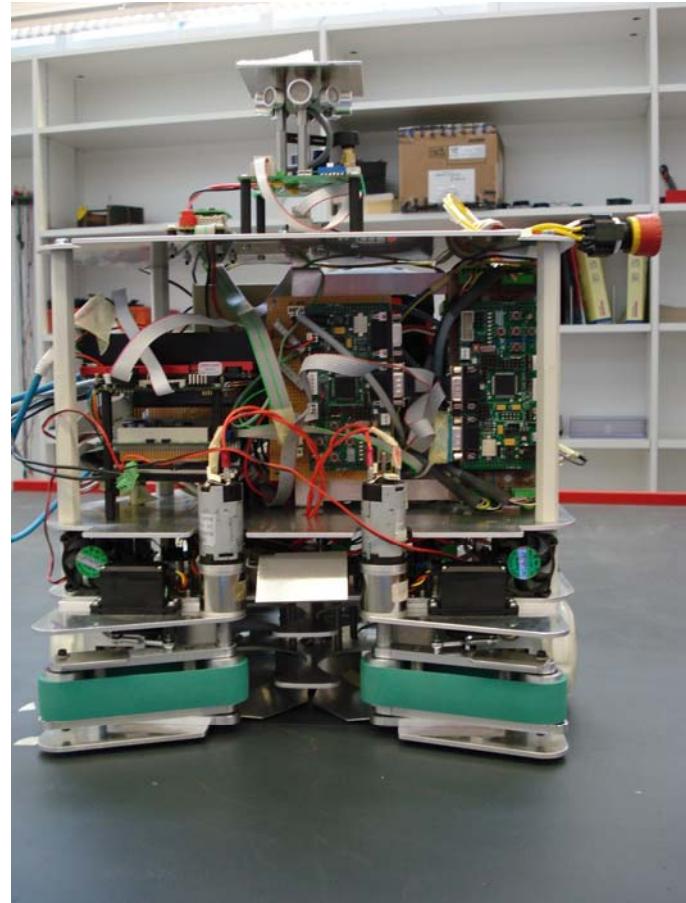
Kunz Simon
Neuhaus Florian
Kobel Moritz
Eggimann Simon

Haslebacher Philipp
Hubacher Samuel
Bürli Fabian

Newsletter MarsRiders vom 22.Januar 2008

Einige Tage nach Sylvester haben wir unsere Projektarbeit fortgesetzt. Denn bereits am 17.Januar 2008 sollte unser Roboter die Zulassungsprüfung erreichen können.

Von jetzt an wird mit Druck auf die erste Testfahrt gearbeitet. Dauern hört man ein lautes Aufheulen der Antriebsmotoren. Mit einem einfachen Befehl am Laptop kann man die Trommel drehen. Zwischendurch dreht sie nicht, das bedeutet Fehler suchen. Mit dem Navigationsystem bestimmt man die eigene Position und die Gegnerposition. Mit diesen Punkten rechnet der Industrie-PC wohin wir fahren wollen. Die erste Testfahrt war erfolgreich. Der Roboter fährt zum Dispenser, nimmt einen Ball auf, fährt zurück und gibt den Ball über die Tischkante aus. Damit ist erst ein Teilziel erreicht...



Es ist die letzte Nacht vor dem Vortrag an dem wir unsere Projektarbeit präsentieren. Der Roboter fährt, aber kann sich nur sehr langsam auf dem Tisch orientieren und fortbewegen. Wir begnügen uns mit diesem Fahrverhalten und bereiten unsere Demonstration vor.

Nun ist es an der Zeit die Dokumentation abzuschliessen für dieses Semester.

Am 18. Februar beginnt das neue Semester. Folgende Punkte befinden sich in unserer Planung:

- Geschwindigkeitsoptimierung der Abläufe
- Spielerfahrung sammeln mit dem OneTeam
- Sicherheitsaspekte beachten um unsere Hardware zu schonen
- Geschwindigkeit des Antriebes erhöhen
- Bilderkennung für Bälle auf dem Tisch

Newsletter MarsRiders vom 14.April 2008

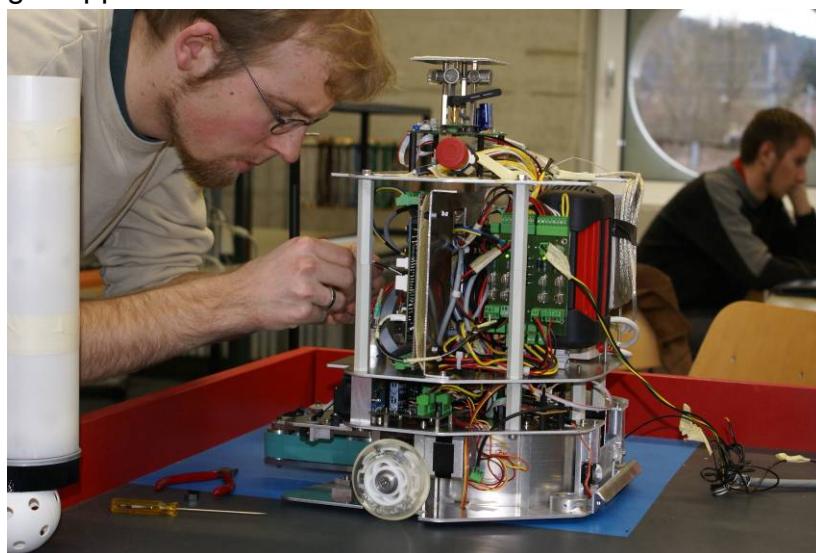
Vor einigen Wochen haben wir unseren Roboter auf den Namen B.I.T.T getauft. Wir haben bereits einige Testfahrten erfolgreich überstanden. Erfolgreich testen konnten wir das Verhalten von B.I.T.T mit und ohne Gegner dank dem Team oneTeam.

B.I.T.T kann zur Zeit 2 farbige und 3 weisse Unihockeybälle aufnehmen, sortieren und ausgeben innerhalb der erlaubten 90 Sekunden. Bei Einwirkung des Gegners umfährt unser Roboter ohne Eingriffe den gegnerischen Roboter.

Seit dem letzten Newsletter haben wir unserem Roboter das Sprechen beigebracht. Dank einer freien Text-To-Speech-Software teilt uns der Roboter seine Handlungen mit.

Schlechte Erfahrungen haben mit dem elektrischen Kompass gemacht und mit einem analogen Kompass überprüft. Die Genauigkeit des Kompasses ist ungenügend. Der Winkel kann sich im Stillstand bis zu 30 Grad verändern. Aus diesem Grund implementiert ein Teil unseres Teams ein einfaches Gyroskop, dass uns über den Drehwinkel informiert.

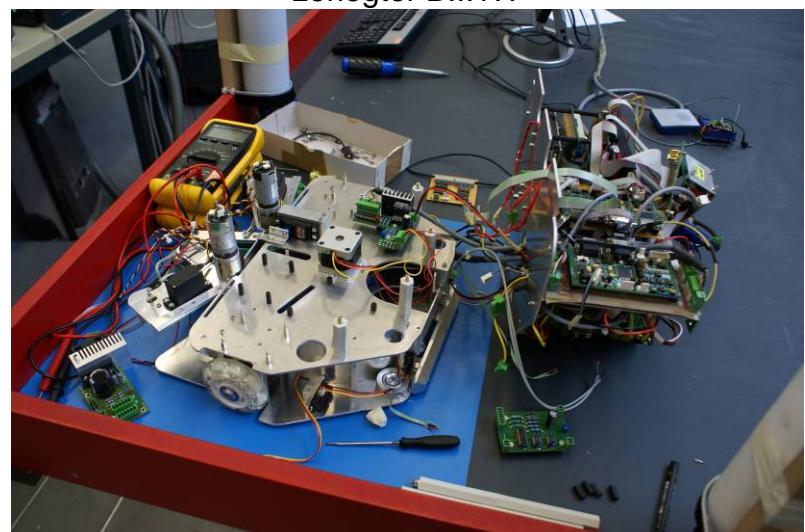
Eine weitere Neuerung trifft noch diese Woche ein: Um uns schneller auf dem Spielfeld zu Bewegen erhalten wir stärkere direktangetriebene Motoren. Dadurch möchten wir eine Fahrgeschwindigkeit von bis zu 1m/s erreichen. Damit sind wir gewappnet für den Wettbewerb.



Ausprobieren, Optimieren, Feinabstimmung

In einem Monat, nämlich am 17.Mai 2008, findet der Wettkampf statt. bis zu diesem Datum werden wir noch viele Testläufe durchführen und manche Codezeile ändern.

zerlegter B.I.T.T





Newsletter MarsRiders vom 15.Mai 2008

Wie wir im letzten Newsletter angekündigt haben wir unseren neuen Antrieb in Betrieb genommen. Wir erreichen nun eine Geschwindigkeit von 1m/s. Damit rasen wir deutlich schneller über den Tisch! Präzision haben wir dadurch keine verloren. Anstatt des ungenauen Kompasses verwenden wir ein einfaches Gyroskope, das den genauen Winkel mit dem wir zum Tisch stehen bestimmt.

Um die Zulassung zu bestehen müssen wir davon ausgehen das unser B.I.T.T die Farbe der Bälle erkennen kann. Dies haben wir mittels eines RGB-Sensor realisiert. In den letzten Tagen vor dem Wettbewerb testen wir noch die Bälle die auf dem Tisch herumliegen mittels Bilderkennung anzufahren und aufzunehmen. Somit haben wir die Möglichkeit wenn uns der Gegner den Weg verstellt Bälle auf dem Spielfeld zu Sammeln.



Heute Morgen wurde das Schutzgehäuse von den Maschinentechnik-Studenten schwarz gesprayt und mit den Sponsorenklebern verziert.

Zur Zeit verpacken wir unser Ersatzmaterial, Werkzeuge und Messgeräte

Wir möchten uns bei unseren Sponsoren bedanken für ihre Dienstleistung. Unsere Arbeit wurde erheblich erleichtert.

Team Marsriders

Kunz Simon
Neuhaus Florian
Kobel Moritz
Eggimann Simon

Haslebacher Philipp
Hubacher Samuel
Bürli Fabian



Nationale Qualifikation 17./18. Mai in Rapperswil

Am Abend vor der nationalen Ausscheidung beendeten wir unsere Tests rechtzeitig. Unsere Autos waren um 18:30 bepackt und abfahrtsbereit.

Um 7:00 fuhren wir in Richtung Rapperswil ab. Dort angekommen richteten wir unseren Arbeitsplatz ein und führten unseren Roboter zum ersten Mal dem Schiedsrichterkomitee für die Zulassungsprüfung vor. Nach einigen Optimierungen bei der Erkennung und Umfahrung des Gegners, wurde unser Roboter zugelassen.

Resultate Samstag:

MarsRiders vs. VirtualBot	00:00
MarsRiders vs. Arc'obot	00:12
MarsRiders vs. DOMBOT Rapperswil	13:01
MarsRiders vs. EcubeMsquare	01:19
MarsRiders vs. Icare	01:19

Zu Beginn des Wettkampfes hatten wir einige Schwierigkeiten beim Start. Einmal brachten wir die Software durch unsere Nervosität zum Absturz. Ein anderes Mal hatten wir auf einem internen Kommunikationsbus ein Problem, welches den Start verhinderte. In der vierten und fünften Runde hatten wir mit der Ballspeicherung Probleme. Am Abend begannen wir damit diese, für uns bisher unbekannten, Probleme zu beseitigen. So versetzten wir über Nacht einen Sensor, mit dem wir nun feststellen konnten, ob die Ballspeichertrommel an der richtigen Position steht. Diese Änderung wurde auch in der Software implementiert und ausgiebig getestet. Daneben gab es noch diverse Verbesserungen der Software welche die Fehleranfälligkeit reduzieren sollen.

Als die Sonne wieder am Horizont aufstieg, legten sich auch die letzten Teammitglieder schlafen, wenn auch nicht für lange.

Resultate Sonntag:

MarsRiders vs. oneTeam	20:14
MarsRiders vs. Deimos	27:01
MarsRiders vs. CVRA	27:01
MarsRiders vs. e-Robot	00:14

An diesem Morgen gelang uns endlich der Durchbruch. Unser Roboter punktete Match für Match und unserer Arbeit hatte sich ausbezahlt. Beim letzten Match hatte unser b.i.t.t. wieder etwas Lampenfieber und bewegte sich nicht einmal aus seinem Startfeld.

Danach belegten wir mit dem Team Arc'obot den Platz acht. Nach einer Diskussion entschied der Hauptschiedsrichter, dass unser Team aufgrund von mehr Punkten während der Matches für den Viertelfinal qualifiziert wurde.

Danach hatte unser Roboter b.i.t.t. keine Startprobleme mehr. Wir gewannen jeden Match und bewegten uns ohne Umweg direkt aufs Podest.

Schlussklassement

1. MarsRiders, Burgdorf
2. Deimos, Biel
3. Vipers, Winterthur

Alle Resultate sind auf der Webseite www.swisseurobot.ch einsehbar.



Internationale Ausscheidung 22. bis 25 Mai 2008 in Heidelberg (DE)

Zulassung

Donnerstags um 06:58 fuhren wir in Burgdorf los. Über den Zoll kamen wir problemlos, unserer Ladung wurde keine Beachtung geschenkt. Noch vor dem Mittag haben wir unseren Arbeitsplatz bezogen und „b.i.t.t.“ Leben eingehaucht.

Nach einer Stärkung bewegte sich „b.i.t.t.“ auf dem Zulassungstisch. Obwohl der Schiedsrichter keine realistische Situation mit dem Test-Gegner simulierte, erhielten wir die Zulassung beim ersten Versuch.

Match 1

Noch vor dem ersten Match mussten wir uns bei der Wettkampf-Leitung beschweren. Da die Tische nicht dem Reglement entsprachen, durften wir vor jedem Match bei den näheren Dispensern die Drähte, die mit Klebeband aufgeklebt waren, entfernen. Diese Hilfe war aufgebracht, weil 90% der französischen Teams sonst die Bälle nicht aufnehmen konnten.

MarsRiders vs. Turag

16:10

Leider konnten wir die weissen Bälle nicht ergattern, so dass keine Samples möglich waren, die uns Zusatzpunkte bringen würden.

Match 2

Vor dem Match 2 gegen den Roboter 1966 passten wir noch die Distanzen an, dass wir wieder weisse Bälle aufnehmen konnten.

MarsRiders vs. 1966

01:07

Obwohl unser Roboter im Prinzip einen perfekten Lauf absolvierte, konnten wir nicht Punkten, da sich die Trommel nicht drehte. Dies weil ein Sensor keinen Ball registrierte. Kein Ball aufgenommen = kein Ball ausladen. Dieses Phänomen konnten wir bei späteren Test und Matches nicht reproduzieren.

Dieser Fehler war noch nie aufgetreten und somit konnten wir ihn erst jetzt softwaremässig umgehen.

Match 3

MarsRiders vs. Übermaschin

10:22

Eigentlich machte unser Roboter seine Arbeit sehr gut und präzise, leider lud aber unser Gegner zu Beginn an unserer Position Bälle aus, so dass diese von uns, welche wir am selben Ort entluden, auf den Bällen des Mitbewerbers zu liegen kamen und somit für uns nicht zählten.

Match 4

MarsRiders vs. RoboBUTE

00:01

Im Match gegen RoboBUTE verloren wir, weil das Navigationssystem seinen Dienst verweigerte. Eventuell waren die Baken falsch aufgestellt oder der Gegner verhinderte mit einer ungünstigen Startposition den Kontakt mit den Navigationsbaken. So drehten wir nur eine Pirouette...



Match 5

MarsRiders vs. RallyRobot

16:03

Der letzte Match war erfolgreicher. Allerdings hatten wir auch hier Pech, da uns beim ersten Dispenser ein weisser Ball die Aufnahme erschwerte. So blieb unser Potential weitgehend ungenutzt.

So beenden wir unsere Odyssee in Heidelberg mit dem 24. Rang. Unter den gegebenen Umständen ein zufriedenstellendes Resultat, wenn man bedenkt, dass 46 Teams gestartet sind.

Detaillierte Information zur internationalen Ausscheidung findet man in unserem Blog unter <http://blog.itds.ch/roller/marsriders/>.

Besten Dank

Das Team MarsRiders möchte sich bei allen Sponsoren und Helfern bedanken, die uns während den 9 Monaten unterstützt haben. Ebenfalls bedanken wir uns bei den Dozenten und Assistenten für ihre kostbare Zeit, welche sie für uns aufgewendet haben.

Wir blicken auf eine strenge, aber sehr lehrreiche Zeit zurück. Es ist nicht auszuschliessen, dass sich das Team MarsRiders in einigen Jahren zurückmeldet.

Mit bestem Dank, Team MarsRiders:

Simon Kunz
Simon Eggimann
Moritz Kobel
Florian Neuhaus
Philipp Haslebacher
Fabian Bürli
Samuel Hubacher



E. Software

E.1. Dokumentation, Präsentationen und Plakate

Siehe dazu das CD-Verzeichnis .\0_Dokumentation\

E.2. Korrespondenz

Siehe dazu das CD-Verzeichnis .\1_Organisation\

E.3. mechanische Zeichnungen

Siehe dazu das CD-Verzeichnis .\2_Mechanik\

E.4. Schemas

Siehe dazu das CD-Verzeichnis .\3_Schemas\

E.5. Datenblätter

Siehe dazu das CD-Verzeichnis .\4_Datenblätter\

E.6. Latex Sourcecode und sonstige Sourcecodes

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\

E.6.1. Dokumentation und Sourcecode der Projektarbeit 1

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\tags\Eurobot-Ende-Projektarbeit1\

E.6.2. Sourcecode Antriebsknoten RTOS

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\trunk\Atmel\AntriebRTOS\

E.6.3. Sourcecode Sensorikknoten RTOS

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\trunk\Atmel\Sensorik\

E.6.4. Sourcecode Rampenberechnung in Java

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\trunk\Atmel\AntriebRTOSRampCalcToolJ\

E.6.5. Sourcecode Java Eurobot

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\trunk\Eurobot\

E.6.6. Javadoc vom Sourcode Java

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\trunk\Eurobot\doc\

E.6.7. Sourcecode diese Dokumentation

Siehe dazu das CD-Verzeichnis .\5_letzte SVN Version\trunk\doc\

E.7. Doxygen Antriebssystem

AntriebRTOS Reference Manual

1.3

Generated by Doxygen 1.5.4

Thu May 29 23:26:29 2008

Contents

1	AntriebRTOS Class Index	1
2	AntriebRTOS File Index	1
3	AntriebRTOS Class Documentation	2
4	AntriebRTOS File Documentation	3

1 AntriebRTOS Class Index

1.1 AntriebRTOS Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MOTOR_PROPERTIES	2
-------------------------	----------

2 AntriebRTOS File Index

2.1 AntriebRTOS File List

Here is a list of all documented files with brief descriptions:

belt.c (Stellt Funktionen zur Ansteuerung der Bänder-Motoren zur Verfügung)	3
belt.h (Stellt Funktionen zur Ansteuerung der Bänder-Motoren zur Verfügung)	5
can.c	??
can.h	??
can_controller.c	??
can_controller.h	??
config.h (Hier werden die Motoren-Parameter und andere Konfigurationsoptionen gesetzt. Gewisse DEFINES werden noch nicht verwendet)	8
initMyPorts.c (Stellt Funktionen zur Initialisierung einiger Ports zur Verfügung)	9
initMyPorts.h (Stellt Funktionen zur Initialisierung einiger Ports zur Verfügung)	10
motor.c (Stellt die Strukturen für die Motoren zur Verfügung. Diese abstrahieren die Motoren-Parameter. Zudem werden hier alle Funktionen bereitgestellt, die zur Beeinflussung der Motoren dienen. Die Regelung findet in motorControl statt)	11
motor.h (Stellt die Strukturen für die Motoren zur Verfügung. Diese abstrahieren die Motoren-Parameter. Zudem werden hier alle Funktionen bereitgestellt, die zur Beeinflussung der Motoren dienen. Die Regelung findet in motorControl statt)	15

rampCalculation.c (Hier werden Funktionen zur Berechnung der Anfahr-/Bremsrampe zur Verfügung gestellt. Diese Funktionen berücksichtigen die aktuellen Motorendaten aus motor.h (p. 15). Die Werte der Rampe werden auf Frequenzen heruntergerechnet, je nachdem was in config.h (p. 8) für Maximal-/Minimalwerte angegeben wurden. Die Rampenwerte stammen aus rampCalculationValues.h (p. 24))	19
rampCalculation.h (Hier werden Funktionen zur Berechnung der Anfahr-/Bremsrampe zur Verfügung gestellt. Diese Funktionen berücksichtigen die aktuellen Motorendaten aus motor.h (p. 15). Die Werte der Rampe werden auf Frequenzen heruntergerechnet, je nachdem was in config.h (p. 8) für Maximal-/Minimalwerte angegeben wurden. Die Rampenwerte stammen aus rampCalculationValues.h (p. 24))	22
rampCalculationValues.h (Enthält die Werte der Anfahr- und Bremsrampe. freqRamp2 bildet eine Sinus-Quadrat Funktion ab)	24
RTOS.c (Von hier aus wird das ganze System initialisiert. Enthält die main Funktion. Erstellt alle Tasks. Definiert die Stack-Größen)	25
taskMsgControl.c (Hier werden Funktionen zur Auswertung der CAN-Nachrichten bereitgestellt. Der Task reagiert auf eine Nachrichtin der main_msgq)	27
taskMsgControl.h (Hier werden Funktionen zur Auswertung der CAN-Nachrichten bereitgestellt. Der Task reagiert auf eine Nachrichtin der main_msgq)	28
taskRS232.c	??
taskRS232.h	??
taskSpeedControl.c (Kontrolliert die Geschwindigkeiten der versch. Schrittmotoren über die Funktion motorControl (in motor.c (p. 11) definiert) und des DC-Motors über die Funktion beltControl (in belt.c (p. 3) definiert))	29
taskSpeedControl.h (Kontrolliert die Geschwindigkeiten der versch. Schrittmotoren über die Funktion motorControl (in motor.c (p. 11) definiert) und des DC-Motors über die Funktion beltControl (in belt.c (p. 3) definiert))	31
timer.c (Implementiert die Interruptroutinen der Timer und stellt Funktionen zum Starten und Stoppen der Timer bereit. Zudem kann der Speed in Hz gesetzt werden)	32
timer.h (Implementiert die Interruptroutinen der Timer und stellt Funktionen zum Starten und Stoppen der Timer bereit. Zudem kann der Speed in Hz gesetzt werden)	37

3 AntriebRTOS Class Documentation

3.1 MOTOR_PROPERTIES Struct Reference

```
#include <motor.h>
```

3.1.1 Detailed Description

VALUES

Definition at line 41 of file motor.h.

The documentation for this struct was generated from the following file:

- motor.h

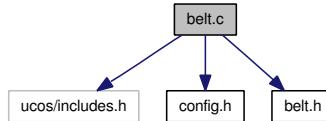
4 AntriebRTOS File Documentation

4.1 belt.c File Reference

Stellt Funktionen zur Ansteuerung der Bänder-Motoren zur Verfügung.

```
#include "ucos/include.h"
#include "config.h"
#include "belt.h"
```

Include dependency graph for belt.c:



Functions

- void **initBelt** (void)

Bänder initialisieren. Bänder ausschalten. Pins "NULL" setzen.

- void **beltControl** (void)

*DC-Motoren drehen vorwärts oder rückwärts. Je nachdem was der TaskMsgControl ihnen aufgetragen hat.
Über die beltStatusFlags kann überprüft werden, ob sich etwas am Status geändert hat.*

- void **beltSettings** (INT8U beltNumber)

Ändert die Drehrichtung und startet oder stoppt die Bänder. Dazu wird die Bändernummer und der beltStatus-struct verwendet.

- void **beltStop** (INT8U beltNumber)

Stoppt die Bänder explizit. Könnte auch etwas mühsamer über beltSettings abgewickelt werden. Reine Komfortfunktion. Aktuell nicht verwendet.

4.1.1 Detailed Description

Stellt Funktionen zur Ansteuerung der Bänder-Motoren zur Verfügung.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **belt.c**.

4.1.2 Function Documentation

4.1.2.1 beltControl (void)

DC-Motoren drehen vorwärts oder rückwärts. Je nachdem was der TaskMsgControl ihnen aufgetragen hat. Über die beltStatusFlags kann überprüft werden, ob sich etwas am Status geändert hat.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 73 of file belt.c.

References beltSettings().

Here is the call graph for this function:



4.1.2.2 beltSettings (INT8U beltNumber)

Ändert die Drehrichtung und startet oder stoppt die Bänder. Dazu wird die Bändernummer und der beltStatus-struct verwendet.

Parameters:

beltNumber Nummer des Bandes.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 113 of file belt.c.

4.1.2.3 beltStop (INT8U *beltNumber*)

Stoppt die Bänder explizit. Könnte auch etwas mühsamer über beltSettings abgewickelt werden. Reine Komfortfunktion. Aktuell nicht verwendet.

Parameters:

beltNumber Nummer des Bandes.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 159 of file belt.c.

4.1.2.4 void initBelt (void)

Bänder initialisieren. Bänder ausschalten. Pins "NULL" setzen.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 39 of file belt.c.

References beltSettings().

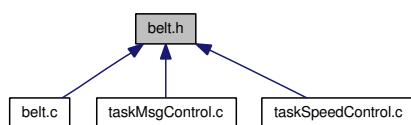
Here is the call graph for this function:



4.2 belt.h File Reference

Stellt Funktionen zur Ansteuerung der Bänder-Motoren zur Verfügung.

This graph shows which files directly or indirectly include this file:



Functions

- **void initBelt ()**
Bänder initialisieren. Bänder ausschalten. Pins "NULL" setzen.

- void **beltSettings** (INT8U beltNumber)
Ändert die Drehrichtung und startet oder stoppt die Bänder. Dazu wird die Bändernummer und der beltStatus-struct verwendet.
- void **beltStop** (INT8U beltNumber)
Stoppt die Bänder explizit. Könnte auch etwas mühsamer über beltSettings abgewickelt werden. Reine Komfortfunktion. Aktuell nicht verwendet.
- void **beltControl** (void)
DC-Motoren drehen vorwärts oder rückwärts. Je nachdem was der TaskMsgControl ihnen aufgetragen hat. Über die beltStatusFlags kann überprüft werden, ob sich etwas am Status geändert hat.

4.2.1 Detailed Description

Stellt Funktionen zur Ansteuerung der Bänder-Motoren zur Verfügung.

Version:

1.2

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications:

LastChangedDate

2008-05-28 17:51:32 +0200 (Mi, 28 Mai 2008)

LastChangedRevision

20

LastChangedBy

neuhf1

Definition in file **belt.h**.

4.2.2 Function Documentation

4.2.2.1 void beltControl (void)

DC-Motoren drehen vorwärts oder rückwärts. Je nachdem was der TaskMsgControl ihnen aufgetragen hat. Über die beltStatusFlags kann überprüft werden, ob sich etwas am Status geändert hat.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 73 of file belt.c.

References beltSettings().

Here is the call graph for this function:



4.2.2.2 void beltSettings (INT8U *beltNumber*)

Ändert die Drehrichtung und startet oder stoppt die Bänder. Dazu wird die Bändernummer und der beltStatus-struct verwendet.

Parameters:

beltNumber Nummer des Bandes.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 113 of file belt.c.

4.2.2.3 void beltStop (INT8U *beltNumber*)

Stoppt die Bänder explizit. Könnte auch etwas mühsamer über beltSettings abgewickelt werden. Reine Komfortfunktion. Aktuell nicht verwendet.

Parameters:

beltNumber Nummer des Bandes.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 159 of file belt.c.

4.2.2.4 void initBelt ()

Bänder initialisieren. Bänder ausschalten. Pins "NULL" setzen.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

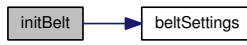
Date:

01.01.2008

Definition at line 39 of file belt.c.

References beltSettings().

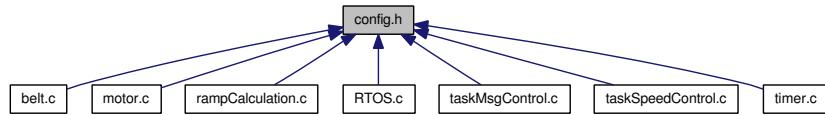
Here is the call graph for this function:



4.3 config.h File Reference

Hier werden die Motoren-Parameter und andere Konfigurationsoptionen gesetzt. Gewisse DEFINES werden noch nicht verwendet.

This graph shows which files directly or indirectly include this file:



4.3.1 Detailed Description

Hier werden die Motoren-Parameter und andere Konfigurationsoptionen gesetzt. Gewisse DEFINES werden noch nicht verwendet.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

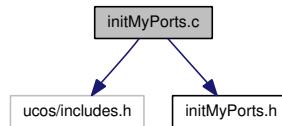
Definition in file **config.h**.

4.4 initMyPorts.c File Reference

Stellt Funktionen zur Initialisierung einiger Ports zur Verfügung.

```
#include "ucos/include.h"
#include "initMyPorts.h"
```

Include dependency graph for initMyPorts.c:



Functions

- void **initMyPorts** (INT8U *perr)
Initialisiere alle Ports (A und B) als Output.

4.4.1 Detailed Description

Stellt Funktionen zur Initialisierung einiger Ports zur Verfügung.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **initMyPorts.c**.

4.4.2 Function Documentation

4.4.2.1 initMyPorts (INT8U * *perr*)

Initialisiere alle Ports (A und B) als Output.

Parameters:

perr Eventuelle Fehlermeldung. Nicht verwendet.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

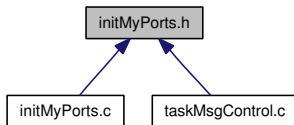
01.01.2008

Definition at line 32 of file initMyPorts.c.

4.5 initMyPorts.h File Reference

Stellt Funktionen zur Initialisierung einiger Ports zur Verfügung.

This graph shows which files directly or indirectly include this file:



Functions

- void **initMyPorts** (INT8U *perr)
Initialisiere alle Ports (A und B) als Output.

4.5.1 Detailed Description

Stellt Funktionen zur Initialisierung einiger Ports zur Verfügung.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **initMyPorts.h**.

4.5.2 Function Documentation

4.5.2.1 void initMyPorts (INT8U * *perr*)

Initialisiere alle Ports (A und B) als Output.

Parameters:

perr Eventuelle Fehlermeldung. Nicht verwendet.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

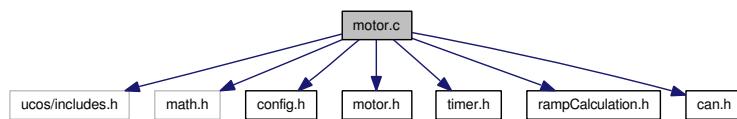
Definition at line 32 of file initMyPorts.c.

4.6 motor.c File Reference

Stellt die Strukturen für die Motoren zur Verfügung. Diese abstrahieren die Motoren-Parameter. Zudem werden hier alle Funktionen bereitgestellt, die zur Beeinflussung der Motoren dienen. Die Regelung findet in motorControl statt.

```
#include "ucos/include.h"
#include <math.h>
#include "config.h"
#include "motor.h"
#include "timer.h"
#include "rampCalculation.h"
#include "can.h"
```

Include dependency graph for motor.c:



Functions

- void **motorControl** (INT8U *perr)

*Kern der Anwendung: Übernimmt die Steuerung der Motoren. Fährt eine Rampe beim Beschleunigen (\sin^2) und Bremsen. Versucht rechtzeitig (punktgenau mit Rampe) zu bremsen, wenn sich die Anzahl zu fahrenden Schritte erreicht sind. Funktioniert nur mit etwas Bastelaufwand. Genaue Berechnungen mit dem selbstgeschriebenen Java-Tool RampCalcToolJ führen leider nicht zum Ziel. Die Steuerung erkennt Richtungswechsel. Die Motorenfrequenzen werden effektiv in **timer.c** (p. 32) gesetzt.*

- void **initMotorControl** (INT8U *perr)

Initialisiert alle Motoren auf einen definierten Zustand. Die Motoren erhalten Haltestrom (65% Strom)! Leert den Akku relativ schnell. Per CAN-Nachricht kann der Haltestrom ausgeschaltet werden. Siehe taskMsgControl.c (p. 27).

- void **motorSettings** (INT8U motorNumber, INT8U *perr)

Setzt enable und Richtung auf Grund der Angaben im Motoren-struct.

- void **stopMotorSlow** (INT8U motorNumber)

Stoppe die Motoren sanft. Also lasse sie von der Steuerung stoppen. Es wird eine Bremsrampe gefahren.

- void **stopMotors** (INT8U motorNumber)
Stoppe die Motoren grob. Es wird keine Bremsrampe gefahren.

Variables

- OS_EVENT * **motor_actual_semaphore** [MOTOR_COUNT]
binary semaphore to access motors
- OS_EVENT * **motor_nominal_semaphore** [MOTOR_COUNT]
binary semaphore to access motors

4.6.1 Detailed Description

Stellt die Strukturen für die Motoren zur Verfügung. Diese abstrahieren die Motoren-Parameter. Zudem werden hier alle Funktionen bereitgestellt, die zur Beeinflussung der Motoren dienen. Die Regelung findet in motorControl statt.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **motor.c**.

4.6.2 Function Documentation

4.6.2.1 initMotorControl (INT8U * *perr*)

Initialisiert alle Motoren auf einen definierten Zustand. Die Motoren erhalten Haltestrom (65% Strom)! Leert den Akku relativ schnell. Per CAN-Nachricht kann der Haltestrom ausgeschaltet werden. Siehe **taskMsgControl.c** (p. 27).

Parameters:

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 268 of file motor.c.

References motor_actual_semaphore, motor_nominal_semaphore, and stopVirtualTimer().

Here is the call graph for this function:

**4.6.2.2 motorControl (INT8U * perr)**

Kern der Anwendung: Übernimmt die Steuerung der Motoren. Fährt eine Rampe beim Beschleunigen (\sin^2) und Bremsen. Versucht rechtzeitig (punktgenau mit Rampe) zu bremsen, wenn sich die Anzahl zu fahrenden Schritte erreicht sind. Funktioniert nur mit etwas Bastelaufwand. Genaue Berechnungen mit dem selbstgeschriebenen Java-Tool RampCalcToolJ führten leider nicht zum Ziel. Die Steuerung erkennt Richtungswechsel. Die Motorenfrequenzen werden effektiv in **timer.c** (p. 32) gesetzt.

Parameters:

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

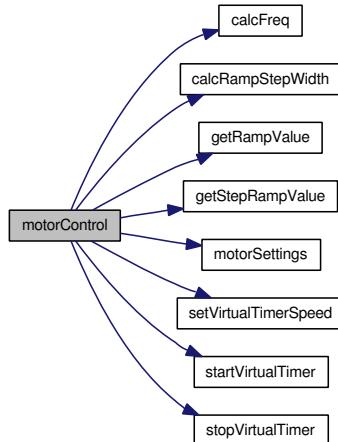
Date:

01.01.2008

Definition at line 56 of file motor.c.

References calcFreq(), calcRampStepWidth(), getRampValue(), getStepRampValue(), motorSettings(), setVirtualTimerSpeed(), startVirtualTimer(), and stopVirtualTimer().

Here is the call graph for this function:



4.6.2.3 motorSettings (INT8U *motorNumber*, INT8U * *perr*)

Setzt enable und Richtung auf Grund der Angaben im Motoren-struct.

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 330 of file motor.c.

4.6.2.4 stopMotors (INT8U *motorNumber*)

Stoppe die Motoren grob. Es wird keine Bremsrampe gefahren.

Signalisiert per CAN dass die gewünschte Anzahl Schritte gefahren wurde oder die Fahrt durch einen Stopp-Befehl unterbrochen wurde. Wenn letzteres der Fall ist, dann wird die Anzahl nicht gefahrenen Schritte zurückgeliefert.

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

stepsLeft100 Anzahl nicht gefahrene Schritte, Hunderter

stepsLeft10 Anzahl nicht gefahrene Schritte, Zehner

stepsLeft1 Anzahl nicht gefahrene Schritte, Einer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 441 of file motor.c.

4.6.2.5 stopMotorSlow (INT8U *motorNumber*)

Stoppe die Motoren sanft. Also lasse sie von der Steuerung stoppen. Es wird eine Bremsrampe gefahren.

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

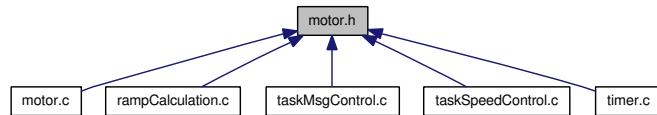
01.01.2008

Definition at line 411 of file **motor.c**.

4.7 motor.h File Reference

Stellt die Strukturen für die Motoren zur Verfügung. Diese abstrahieren die Motoren-Parameter. Zudem werden hier alle Funktionen bereitgestellt, die zur Beeinflussung der Motoren dienen. Die Regelung findet in **motorControl** statt.

This graph shows which files directly or indirectly include this file:



Classes

- struct **MOTOR_PROPERTIES**

Functions

- void **initMotorControl** (INT8U *perr)

Initialisiert alle Motoren auf einen definierten Zustand. Die Motoren erhalten Haltestrom (65% Strom)! Leert den Akku relativ schnell. Per CAN-Nachricht kann der Haltestrom ausgeschaltet werden. Siehe **taskMsgControl.c** (p. 27).
- void **motorSettings** (INT8U *motorNumber*, INT8U *perr)

Setzt enable und Richtung auf Grund der Angaben im Motoren-struct.
- void **stopMotors** (INT8U *motorNumber*)

Stoppe die Motoren grob. Es wird keine Bremsrampe gefahren.
- void **motorControl** (INT8U *perr)

Kern der Anwendung: Übernimmt die Steuerung der Motoren. Fährt eine Rampe beim Beschleunigen (\sin^2) und Bremsen. Versucht rechtzeitig (punktgenau mit Rampe) zu bremsen, wenn sich die Anzahl zu fahrenden Schritte erreicht sind. Funktioniert nur mit etwas Bastelaufwand. Genaue Berechnungen mit dem selbstgeschriebenen Java-Tool RampCalcToolJ führen leider nicht zum Ziel. Die Steuerung erkennt Richtungswechsel. Die Motorenfrequenzen werden effektiv in **timer.c** (p. 32) gesetzt.

- void **stopMotorSlow** (INT8U motorNumber)

Stoppe die Motoren sanft. Also lasse sie von der Steuerung stoppen. Es wird eine Bremsrampe gefahren.

Variables

- OS_EVENT * **motor_actual_semaphore** [MOTOR_COUNT]
binary semaphore to access motors
- OS_EVENT * **motor_nominal_semaphore** [MOTOR_COUNT]
binary semaphore to access motors

4.7.1 Detailed Description

Stellt die Strukturen für die Motoren zur Verfügung. Diese abstrahieren die Motoren-Parameter. Zudem werden hier alle Funktionen bereitgestellt, die zur Beeinflussung der Motoren dienen. Die Regelung findet in motorControl statt.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **motor.h**.

4.7.2 Function Documentation

4.7.2.1 void **initMotorControl** (INT8U * *perr*)

Initialisiert alle Motoren auf einen definierten Zustand. Die Motoren erhalten Haltestrom (65% Strom)! Leert den Akku relativ schnell. Per CAN-Nachricht kann der Haltestrom ausgeschaltet werden. Siehe **taskMsgControl.c** (p. 27).

Parameters:

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 268 of file motor.c.

References motor_actual_semaphore, motor_nominal_semaphore, and stopVirtualTimer().

Here is the call graph for this function:

**4.7.2.2 void motorControl (INT8U * perr)**

Kern der Anwendung: Übernimmt die Steuerung der Motoren. Fährt eine Rampe beim Beschleunigen (\sin^2) und Bremsen. Versucht rechtzeitig (punktgenau mit Rampe) zu bremsen, wenn sich die Anzahl zu fahrenden Schritte erreicht sind. Funktioniert nur mit etwas Bastelaufwand. Genaue Berechnungen mit dem selbstgeschriebenen Java-Tool RampCalcToolJ führten leider nicht zum Ziel. Die Steuerung erkennt Richtungswechsel. Die Motorenfrequenzen werden effektiv in **timer.c** (p. 32) gesetzt.

Parameters:

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

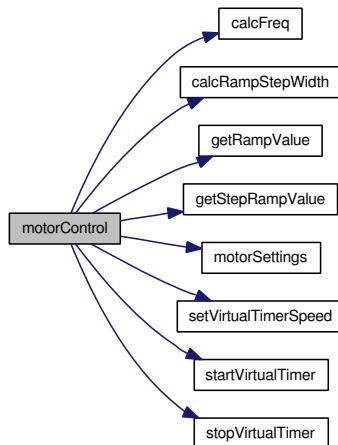
Date:

01.01.2008

Definition at line 56 of file motor.c.

References calcFreq(), calcRampStepWidth(), getRampValue(), getStepRampValue(), motorSettings(), setVirtualTimerSpeed(), startVirtualTimer(), and stopVirtualTimer().

Here is the call graph for this function:



4.7.2.3 void motorSettings (INT8U *motorNumber*, INT8U * *perr*)

Setzt enable und Richtung auf Grund der Angaben im Motoren-struct.

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 330 of file motor.c.

4.7.2.4 void stopMotors (INT8U *motorNumber*)

Stoppe die Motoren grob. Es wird keine Bremsrampe gefahren.

Signalisiert per CAN dass die gewünschte Anzahl Schritte gefahren wurde oder die Fahrt durch einen Stopp-Befehl unterbrochen wurde. Wenn letzteres der Fall ist, dann wird die Anzahl nicht gefahrenen Schritte zurückgeliefert.

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

stepsLeft100 Anzahl nicht gefahrene Schritte, Hunderter

stepsLeft10 Anzahl nicht gefahrene Schritte, Zehner

stepsLeft1 Anzahl nicht gefahrene Schritte, Einer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 441 of file motor.c.

4.7.2.5 void stopMotorSlow (INT8U *motorNumber*)

Stoppe die Motoren sanft. Also lasse sie von der Steuerung stoppen. Es wird eine Bremsrampe gefahren.

Parameters:

motorNumber Motoren-Nummer. Definiert in **motor.h** (p. 15).

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

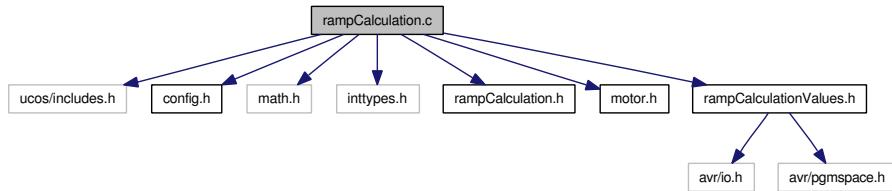
Definition at line 411 of file **motor.c**.

4.8 rampCalculation.c File Reference

Hier werden Funktionen zur Berechnung der Anfahr-/Bremsrampe zur Verfügung gestellt. Diese Funktionen berücksichtigen die aktuellen Motorendaten aus **motor.h** (p. 15). Die Werte der Rampe werden auf Frequenzen heruntergerechnet, je nachdem was in **config.h** (p. 8) für Maximal-/Minimalwerte angegeben wurden. Die Rampenwerte stammen aus **rampCalculationValues.h** (p. 24).

```
#include "ucos/includes.h"
#include "config.h"
#include <math.h>
#include <inttypes.h>
#include "rampCalculation.h"
#include "motor.h"
#include "rampCalculationValues.h"
```

Include dependency graph for **rampCalculation.c**:



Functions

- INT16U **getRampValue** (INT16U index)
einen Wert aus der Rampe lesen
- INT16U **getStepRampValue** (INT16U index)
einen Wert aus der Step-Rampe lesen

- INT16U **calcFreq** (INT16U rampValue, INT16U fStart, INT16U fEnd, INT8U *perr)
Wandle eine Zahl rampValue von einer 0-INT8U_max Rampe in eine entsprechende Frequenz zwischen fStart und fEnd um.
- INT16S **calcRampStepWidth** (INT16U fStart, INT16U fEnd)
Berechne die Schrittweite für die Rampe anhand der gewünschten Start- und Endfrequenz.

4.8.1 Detailed Description

Hier werden Funktionen zur Berechnung der Anfahr-/Bremsrampe zur Verfügung gestellt. Diese Funktionen berücksichtigen die aktuellen Motorendaten aus **motor.h** (p. 15). Die Werte der Rampe werden auf Frequenzen heruntergerechnet, je nachdem was in **config.h** (p. 8) für Maximal-/Minimalwerte angegeben wurden. Die Rampenwerte stammen aus **rampCalculationValues.h** (p. 24).

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **rampCalculation.c**.

4.8.2 Function Documentation

4.8.2.1 calcFreq (INT16U *rampValue*, INT16U *fStart*, INT16U *fEnd*, INT8U * *perr*)

Wandle eine Zahl rampValue von einer 0-INT8U_max Rampe in eine entsprechende Frequenz zwischen fStart und fEnd um.

Parameters:

rampValue Wert aus der Wertetabelle für die Rampe
fStart Startfrequenz der Rampe
fEnd Endfrequenz der Rampe
perr void pointer not used in this example

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 98 of file rampCalculation.c.

4.8.2.2 calcRampStepWidth (INT16U *fStart*, INT16U *fEnd*)

Berechne die Schrittweite für die Rampe anhand der gewünschten Start- und Endfrequenz.

Parameters:

fStart Startfrequenz der Rampe

fEnd Endfrequenz der Rampe

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 136 of file rampCalculation.c.

4.8.2.3 getRampValue (INT16U *index*)

einen Wert aus der Rampe lesen

Parameters:

index Index in der Wertetabelle

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 43 of file rampCalculation.c.

4.8.2.4 getStepRampValue (INT16U *index*)

einen Wert aus der Step-Rampe lesen

Parameters:

index Index in der Wertetabelle

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

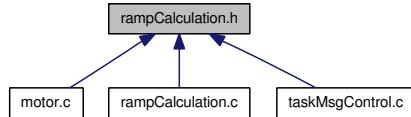
01.01.2008

Definition at line 67 of file rampCalculation.c.

4.9 rampCalculation.h File Reference

Hier werden Funktionen zur Berechnung der Anfahr-/Bremsrampe zur Verfügung gestellt. Diese Funktionen berücksichtigen die aktuellen Motorendaten aus **motor.h** (p. 15). Die Werte der Rampe werden auf Frequenzen heruntergerechnet, je nachdem was in **config.h** (p. 8) für Maximal-/Minimalwerte angegeben wurden. Die Rampenwerte stammen aus **rampCalculationValues.h** (p. 24).

This graph shows which files directly or indirectly include this file:



Functions

- INT16U **getRampValue** (INT16U index)
einen Wert aus der Rampe lesen
- INT16U **getStepRampValue** (INT16U index)
einen Wert aus der Step-Rampe lesen
- INT16U **calcFreq** (INT16U rampValue, INT16U fStart, INT16U fEnd, INT8U *perr)
Wandle eine Zahl rampValue von einer 0-INT8U_max Rampe in eine entsprechende Frequenz zwischen fStart und fEnd um.
- INT16S **calcRampStepWidth** (INT16U fStart, INT16U fEnd)
Berechne die Schrittweite für die Rampe anhand der gewünschten Start- und Endfrequenz.

4.9.1 Detailed Description

Hier werden Funktionen zur Berechnung der Anfahr-/Bremsrampe zur Verfügung gestellt. Diese Funktionen berücksichtigen die aktuellen Motorendaten aus **motor.h** (p. 15). Die Werte der Rampe werden auf Frequenzen heruntergerechnet, je nachdem was in **config.h** (p. 8) für Maximal-/Minimalwerte angegeben wurden. Die Rampenwerte stammen aus **rampCalculationValues.h** (p. 24).

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **rampCalculation.h**.

4.9.2 Function Documentation

4.9.2.1 INT16U calcFreq (INT16U *rampValue*, INT16U *fStart*, INT16U *fEnd*, INT8U * *perr*)

Wandle eine Zahl *rampValue* von einer 0-INT8U_max Rampe in eine entsprechende Frequenz zwischen *fStart* und *fEnd* um.

Parameters:

rampValue Wert aus der Wertetabelle für die Rampe

fStart Startfrequenz der Rampe

fEnd Endfrequenz der Rampe

perr void pointer not used in this example

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 98 of file rampCalculation.c.

4.9.2.2 INT16S calcRampStepWidth (INT16U *fStart*, INT16U *fEnd*)

Berechne die Schrittweite für die Rampe anhand der gewünschten Start- und Endfrequenz.

Parameters:

fStart Startfrequenz der Rampe

fEnd Endfrequenz der Rampe

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 136 of file rampCalculation.c.

4.9.2.3 INT16U getRampValue (INT16U *index*)

einen Wert aus der Rampe lesen

Parameters:

index Index in der Wertetabelle

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 43 of file rampCalculation.c.

4.9.2.4 INT16U getStepRampValue (INT16U *index*)

einen Wert aus der Step-Rampe lesen

Parameters:

index Index in der Wertetabelle

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

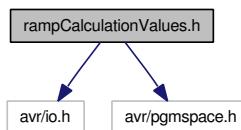
Definition at line 67 of file rampCalculation.c.

4.10 rampCalculationValues.h File Reference

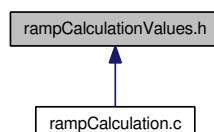
Enthält die Werte der Anfahr- und Bremsrampe. freqRamp2 bildet eine Sinus-Quadrat Funktion ab.

```
#include <avr/io.h>
#include <avr/pgmspace.h>
```

Include dependency graph for rampCalculationValues.h:



This graph shows which files directly or indirectly include this file:

**4.10.1 Detailed Description**

Enthält die Werte der Anfahr- und Bremsrampe. freqRamp2 bildet eine Sinus-Quadrat Funktion ab.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

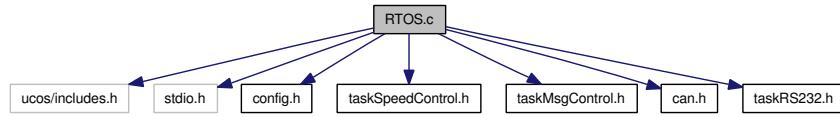
Definition in file **rampCalculationValues.h**.

4.11 RTOS.c File Reference

Von hier aus wird das ganze System initialisiert. Enthält die main Funktion. Erstellt alle Tasks. Definiert die Stack-Größen.

```
#include "ucos/includes.h"
#include <stdio.h>
#include "config.h"
#include "taskSpeedControl.h"
#include "taskMsgControl.h"
#include "can.h"
#include "taskRS232.h"
```

Include dependency graph for RTOS.c:



Functions

- static void **init** (void)

Wird vom TaskStart dazu benutzt um das System zu initialisieren.

- static void **AppCreateTasks** (void)

Erstellt alle Tasks. Dies wären: TaskStart TaskCanReception TaskCanTransmission TaskMsgControl TaskSpeedControl TaskRS232.

- int **main** (void)

Wird als erste Funktion nach einem Reset aufgerufen. Startet das Betriebs- system.

4.11.1 Detailed Description

Von hier aus wird das ganze System initialisiert. Enthält die main Funktion. Erstellt alle Tasks. Definiert die Stack-Größen.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **RTOS.c**.

4.11.2 Function Documentation**4.11.2.1 void AppCreateTasks (void) [static]**

Erstellt alle Tasks. Dies wären: TaskStart TaskCanReception TaskCanTransmission TaskMsgControl TaskSpeedControl TaskRS232.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

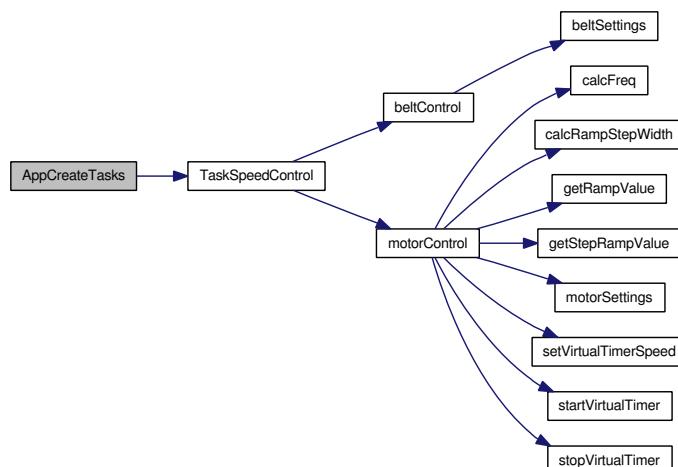
Date:

01.01.2008

Definition at line 151 of file **RTOS.c**.

References **TaskSpeedControl()**.

Here is the call graph for this function:

**4.11.2.2 void init (void) [static]**

Wird vom TaskStart dazu benutzt um das System zu initialisieren.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

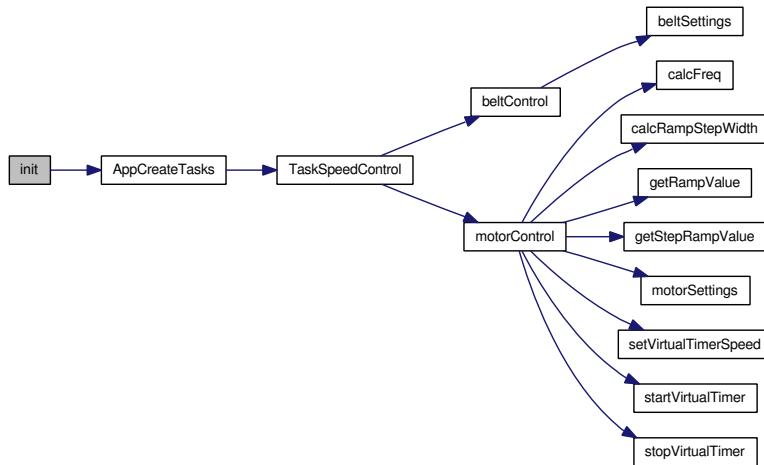
Date:

01.01.2008

Definition at line 120 of file RTOS.c.

References AppCreateTasks().

Here is the call graph for this function:

**4.11.2.3 main (void)**

Wird als erste Funktion nach einem Reset aufgerufen. Startet das Betriebs- system.

Returns:

Gibt 0 ans System zurück. Sollte allerdings NIE passieren.

Author:

Phillip Haslebacher

Date:

01.01.2008

Definition at line 75 of file RTOS.c.

4.12 taskMsgControl.c File Reference

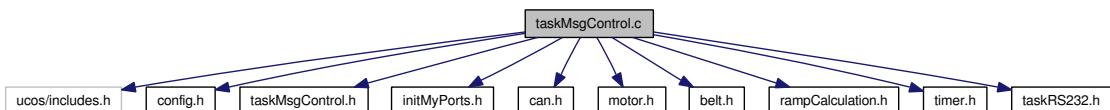
Hier werden Funktionen zur Auswertung der CAN-Nachrichten bereitgestellt. Der Task reagiert auf eine Nachricht in der main_msgq.

```

#include "ucos/includes.h"
#include "config.h"
#include "taskMsgControl.h"
#include "initMyPorts.h"
  
```

```
#include "can.h"
#include "motor.h"
#include "belt.h"
#include "rampCalculation.h"
#include "timer.h"
#include "taskRS232.h"
```

Include dependency graph for taskMsgControl.c:



4.12.1 Detailed Description

Hier werden Funktionen zur Auswertung der CAN-Nachrichten bereitgestellt. Der Task reagiert auf eine Nachrichtin der main_msgq.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

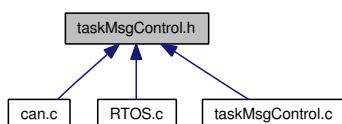
Last Modifications: 28.05.2008

Definition in file **taskMsgControl.c**.

4.13 taskMsgControl.h File Reference

Hier werden Funktionen zur Auswertung der CAN-Nachrichten bereitgestellt. Der Task reagiert auf eine Nachrichtin der main_msgq.

This graph shows which files directly or indirectly include this file:



4.13.1 Detailed Description

Hier werden Funktionen zur Auswertung der CAN-Nachrichten bereitgestellt. Der Task reagiert auf eine Nachrichtin der main_msgq.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

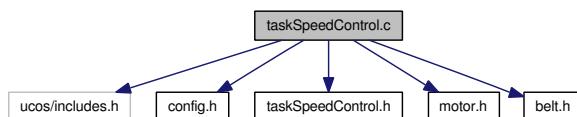
Definition in file **taskMsgControl.h**.

4.14 taskSpeedControl.c File Reference

Kontrolliert die Geschwindigkeiten der versch. Schrittmotoren über die Funktion motorControl (in **motor.c** (p. 11) definiert) und des DC-Motors über die Funktion beltControl (in **belt.c** (p. 3) definiert).

```
#include "ucos/includes.h"
#include "config.h"
#include "taskSpeedControl.h"
#include "motor.h"
#include "belt.h"
```

Include dependency graph for taskSpeedControl.c:



Functions

- void **TaskSpeedControl** (void *pdata)

Ruft periodisch die beiden Motorensteuerungen für Bänder und Fahrmotoren auf.

4.14.1 Detailed Description

Kontrolliert die Geschwindigkeiten der versch. Schrittmotoren über die Funktion motorControl (in **motor.c** (p. 11) definiert) und des DC-Motors über die Funktion beltControl (in **belt.c** (p. 3) definiert).

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **taskSpeedControl.c**.**4.14.2 Function Documentation****4.14.2.1 TaskSpeedControl (void *pdata)**

Ruft periodisch die beiden Motorensteuerungen für Bänder und Fahrmotoren auf.

Parameters:*pdata* Nicht verwendet.**Author:**

Moritz Kobel, Simon Kunz, Florian Neuhaus

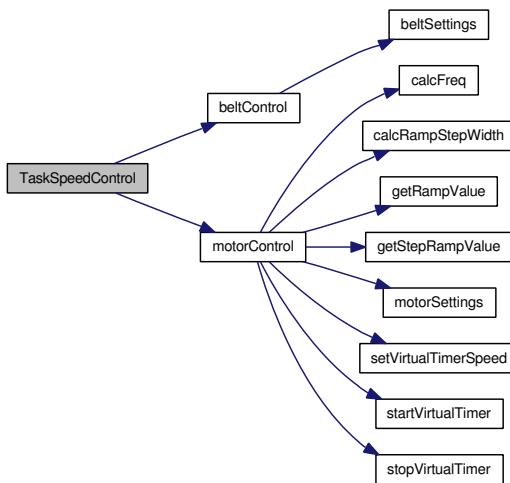
Date:

01.01.2008

Definition at line 40 of file taskSpeedControl.c.

References beltControl(), and motorControl().

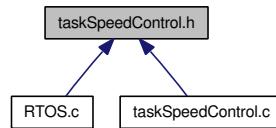
Here is the call graph for this function:



4.15 taskSpeedControl.h File Reference

Kontrolliert die Geschwindigkeiten der versch. Schrittmotoren über die Funktion motorControl (in **motor.c** (p. 11) definiert) und des DC-Motors über die Funktion beltControl (in **belt.c** (p. 3) definiert).

This graph shows which files directly or indirectly include this file:



Functions

- void **TaskSpeedControl** (void *pdata)
Ruft periodisch die beiden Motorensteuerungen für Bänder und Fahrmotoren auf.

4.15.1 Detailed Description

Kontrolliert die Geschwindigkeiten der versch. Schrittmotoren über die Funktion motorControl (in **motor.c** (p. 11) definiert) und des DC-Motors über die Funktion beltControl (in **belt.c** (p. 3) definiert).

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **taskSpeedControl.h**.

4.15.2 Function Documentation

4.15.2.1 void TaskSpeedControl (void *pdata)

Ruft periodisch die beiden Motorensteuerungen für Bänder und Fahrmotoren auf.

Parameters:

pdata Nicht verwendet.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

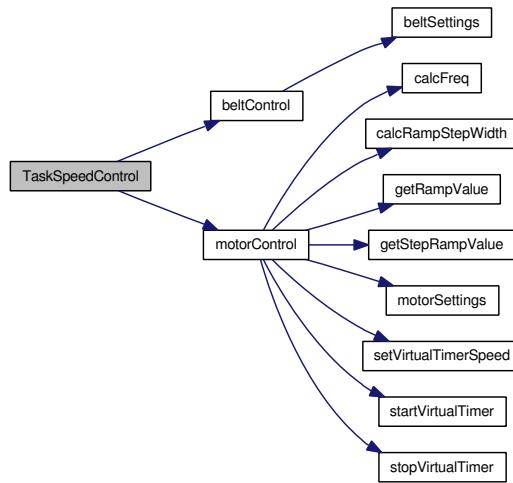
Date:

01.01.2008

Definition at line 40 of file taskSpeedControl.c.

References beltControl(), and motorControl().

Here is the call graph for this function:

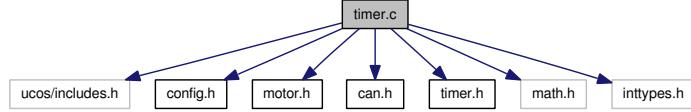
**4.16 timer.c File Reference**

Implementiert die Interruptroutinen der Timer und stellt Funktionen zum Starten und Stoppen der Timer bereit. Zudem kann der Speed in Hz gesetzt werden.

```

#include "ucos/includes.h"
#include "config.h"
#include "motor.h"
#include "can.h"
#include "timer.h"
#include <math.h>
#include "inttypes.h"
  
```

Include dependency graph for timer.c:

**Functions**

- void **initTimer** (INT8U *perr)
Initialisiert alle Timer auf eine Minimalfrequenz.

- void **setVirtualTimerSpeed** (INT16U frequency, INT8U timerNumber)

Die Idee der virtuellen Timer ist folgende: Wir haben einen physikalischen Timer, der 3 Compare-Values und entsprechende Interrupt-Routinen bietet. Jetzt können wir mit dieser Funktion die Compare Values so anpassen, dass wir in der gewünschten Frequenz jeweils Interrupts erhalten. Konkret: | | | | <- Timer1 Overflow | cmpa cmpb cmpc | Damit ein Compare-Value nicht nur einmal pro Timer-Durchgang anspricht, muss er in der Compare-ISR jeweils um seinen eigenen Wert inkrementiert werden. Z.B. für cmpa: | | | | | <- Timer1 Overflow | 1xcmpa 2xcmpa 3xcmpa 4xcmpa 5xcmpa 6xcmpa.

- void **stopVirtualTimer** (INT8U timerNumber)

Stoppe einen virtuellen Timer. Die entsprechende ISR wird nicht mehr ausgeführt.

- void **startVirtualTimer** (INT8U timerNumber)

Aktiviere die Interrupt-Routinen, die zu Timer1 gehören.

- void **setTimerSpeed** (INT16U frequency, INT8U timerNumber)

Setze einen physikalischen Timer (1,2,3) auf eine bestimmte Frequenz. Achtung: Wenn der virtuelle Timer verwendet wird, muss die Frequenz von Timer 1 UNTER der minimalen Frequenz eines beliebigen virtuellen Timers liegen! Siehe auch setVirtualTimerSpeed.

- void **ISR_TIMER2_COMP** (void)

Dieser Timer2 regelt die Geschwindigkeit der Rampe. Es wird jeweils ein Flag in der ISR gesetzt, dass der Funktion motorControl signalisiert, dass der nächste Rampenschritt angewendet werden kann.

- void **ISR_TIMER1_COMPA** (void)

Für TROMMEL-Motor. Gibt das Taktsignal aus. Stoppt die Taktgenerierung, wenn die Anzahl Schritte gefahren wurden.

- void **ISR_TIMER1_COMPC** (void)

RIGHT_MOTOR Gibt das Taktsignal aus. Stoppt die Taktgenerierung, wenn die Anzahl Schritte gefahren wurden.

- void **ISR_TIMER3_COMPA** (void)

Nicht verwendet. ISR muss existieren, wenn auch in interrupts.s definiert!

4.16.1 Detailed Description

Implementiert die Interruptroutinen der Timer und stellt Funktionen zum Starten und Stoppen der Timer bereit. Zudem kann der Speed in Hz gesetzt werden.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **timer.c**.

4.16.2 Function Documentation**4.16.2.1 initTimer (INT8U **perr*)**

Initialisiert alle Timer auf eine Minimalfrequenz.

Parameters:

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

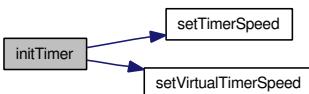
Date:

01.01.2008

Definition at line 76 of file timer.c.

References setTimerSpeed(), setVirtualTimerSpeed(), and virtual_timer_semaphore.

Here is the call graph for this function:

**4.16.2.2 ISR_TIMER1_COMPA (void)**

Für TROMMEL-Motor. Gibt das Taktsignal aus. Stoppt die Taktgenerierung, wenn die Anzahl Schritte gefahren wurden.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

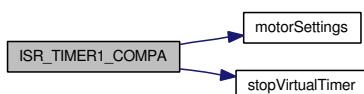
Date:

01.01.2008

Definition at line 394 of file timer.c.

References motorSettings(), and stopVirtualTimer().

Here is the call graph for this function:



4.16.2.3 ISR_TIMER1_COMPC (void)

RIGHT_MOTOR Gibt das Taktsignal aus. Stoppt die Taktgenerierung, wenn die Anzahl Schritte gefahren wurden.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

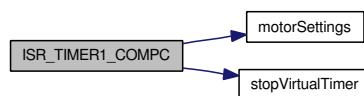
Date:

01.01.2008

Definition at line 485 of file timer.c.

References motorSettings(), and stopVirtualTimer().

Here is the call graph for this function:



4.16.2.4 ISR_TIMER2_COMP (void)

Dieser Timer2 regelt die Geschwindigkeit der Rampe. Es wird jeweils ein Flag in der ISR gesetzt, dass der Funktion motorControl signalisiert, dass der nächste Rampenschritt angewendet werden kann.

LEFT_MOTOR Gibt das Taktsignal aus. Stoppt die Taktgenerierung, wenn die Anzahl Schritte gefahren wurden. Auf Grund historischen Gründen ist TIMER1_COMPB_vect auf PB6 geschaltet, deshalb wird aus Kompatibilitätsgründen der Timer 1B verwendet.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 359 of file timer.c.

4.16.2.5 ISR_TIMER3_COMPA (void)

Nicht verwendet. ISR muss existieren, wenn auch in interrupts.s definiert!

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 527 of file timer.c.

4.16.2.6 setTimerSpeed (INT16U *frequency*, INT8U *timerNumber*)

Setze einen physikalischen Timer (1,2,3) auf eine bestimmte Frequenz. Achtung: Wenn der virtuelle Timer verwendet wird, muss die Frequenz von Timer 1 UNTER der minimalen Frequenz eines beliebigen virtuellen Timers liegen! Siehe auch setVirtualTimerSpeed.

Parameters:

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 328 of file timer.c.

4.16.2.7 setVirtualTimerSpeed (INT16U *frequency*, INT8U *timerNumber*)

Die Idee der virtuellen Timer ist folgende: Wir haben einen physikalischen Timer, der 3 Compare-Values und entsprechende Interrupt-Routinen bietet. Jetzt können wir mit dieser Funktion die Compare Values so anpassen, dass wir in der gewünschten Frequenz jeweils Interrupts erhalten. Konkret: |||||<- Timer1 Overflow | cmpa cmpb cmpe | Damit ein Compare-Value nicht nur einmal pro Timer-Durchgang anspricht, muss er in der Compare-ISR jeweils um seinen eigenen Wert inkrementiert werden. Z.B. für cmpa: ||||| ||||<- Timer1 Overflow | 1xcmpa 2xcmpa 3xcmpa 4xcmpa 5xcmpa 6xcmpa.

Parameters:

frequency Timer-Frequenz in Hz

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 189 of file timer.c.

References virtual_timer_semaphore.

4.16.2.8 startVirtualTimer (INT8U *timerNumber*)

Aktiviere die Interrupt-Routinen, die zu Timer1 gehören.

Parameters:

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 277 of file timer.c.

References virtual_timer_semaphore.

4.16.2.9 stopVirtualTimer (INT8U *timerNumber*)

Stoppe einen virtuellen Timer. Die entsprechende ISR wird nicht mehr ausgeführt.

Parameters:

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

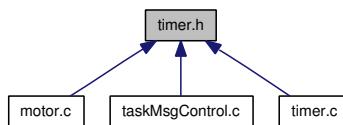
01.01.2008

Definition at line 221 of file timer.c.

4.17 timer.h File Reference

Implementiert die Interruptroutinen der Timer und stellt Funktionen zum Starten und Stoppen der Timer bereit. Zudem kann der Speed in Hz gesetzt werden.

This graph shows which files directly or indirectly include this file:

**Functions**

- void **initTimer** (INT8U *perr)

Initialisiert alle Timer auf eine Minimalfrequenz.

- void **setVirtualTimerSpeed** (INT16U frequency, INT8U timerNumber)

Die Idee der virtuellen Timer ist folgende: Wir haben einen physikalischen Timer, der 3 Compare-Values und entsprechende Interrupt-Routinen bietet. Jetzt können wir mit dieser Funktion die Compare Values so anpassen, dass wir in der gewünschten Frequenz jeweils Interrupts erhalten. Konkret: | | | | <- Timer1 Overflow | cmpa cmpb cmpe | Damit ein Compare-Value nicht nur einmal pro Timer-Durchgang anspricht, muss er in der Compare-ISR jeweils um seinen eigenen Wert inkrementiert werden. Z.B. für cmpa: | | | | | <- Timer1 Overflow | 1xcmpa 2xcmpa 3xcmpa 4xcmpa 5xcmpa 6xcmpa.

- void **setTimerSpeed** (INT16U frequency, INT8U timerNumber)

Setze einen physikalischen Timer (1,2,3) auf eine bestimmte Frequenz. Achtung: Wenn der virtuelle Timer verwendet wird, muss die Frequenz von Timer 1 UNTER der minimalen Frequenz eines beliebigen virtuellen Timers liegen! Siehe auch setVirtualTimerSpeed.

- void **stopVirtualTimer** (INT8U timerNumber)
Stoppe einen virtuellen Timer. Die entsprechende ISR wird nicht mehr ausgeführt.
- void **startVirtualTimer** (INT8U timerNumber)
Aktiviere die Interrupt-Routinen, die zu Timer1 gehören.

Variables

- OS_EVENT * **virtual_timer_semaphore** [MOTOR_COUNT]
binary semaphore to access motors

4.17.1 Detailed Description

Implementiert die Interruptroutinen der Timer und stellt Funktionen zum Starten und Stoppen der Timer bereit. Zudem kann der Speed in Hz gesetzt werden.

Version:

1.3

Date:

18.01.2008

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Remarks:

Last Modifications: 28.05.2008

Definition in file **timer.h**.

4.17.2 Function Documentation

4.17.2.1 void **initTimer** (INT8U * *perr*)

Initialisiert alle Timer auf eine Minimalfrequenz.

Parameters:

perr Eventuelle Fehlermeldung für die höhere Instanz.

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

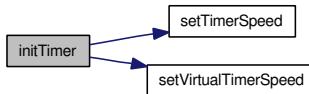
Date:

01.01.2008

Definition at line 76 of file timer.c.

References setTimerSpeed(), setVirtualTimerSpeed(), and virtual_timer_semaphore.

Here is the call graph for this function:

**4.17.2.2 void setTimerSpeed (INT16U frequency, INT8U timerNumber)**

Setze einen physikalischen Timer (1,2,3) auf eine bestimmte Frequenz. Achtung: Wenn der virtuelle Timer verwendet wird, muss die Frequenz von Timer 1 UNTER der minimalen Frequenz eines beliebigen virtuellen Timers liegen! Siehe auch setVirtualTimerSpeed.

Parameters:

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 328 of file timer.c.

4.17.2.3 void setVirtualTimerSpeed (INT16U frequency, INT8U timerNumber)

Die Idee der virtuellen Timer ist folgende: Wir haben einen physikalischen Timer, der 3 Compare-Values und entsprechende Interrupt-Routinen bietet. Jetzt können wir mit dieser Funktion die Compare Values so anpassen, dass wir in der gewünschten Frequenz jeweils Interrupts erhalten. Konkret: ||| |<- Timer1 Overflow | cmpa cmpb cmpc | Damit ein Compare-Value nicht nur einmal pro Timer-Durchgang anspricht, muss er in der Compare-ISR jeweils um seinen eigenen Wert inkrementiert werden. Z.B. für cmpa: ||| | |<- Timer1 Overflow | 1xcmpa 2xcmpa 3xcmpa 4xcmpa 5xcmpa 6xcmpa.

Parameters:

frequency Timer-Frequenz in Hz

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 189 of file timer.c.

References virtual_timer_semaphore.

4.17.2.4 void startVirtualTimer (INT8U *timerNumber*)

Aktiviere die Interrupt-Routinen, die zu Timer1 gehören.

Parameters:

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 277 of file timer.c.

References virtual_timer_semaphore.

4.17.2.5 void stopVirtualTimer (INT8U *timerNumber*)

Stoppe einen virtuellen Timer. Die entsprechende ISR wird nicht mehr ausgeführt.

Parameters:

timerNumber Virtuelle Timer-Nummer

Author:

Moritz Kobel, Simon Kunz, Florian Neuhaus

Date:

01.01.2008

Definition at line 221 of file timer.c.

Index

AppCreateTasks
 RTOS.c, 26

belt.c, 3
 beltControl, 4
 beltSettings, 4
 beltStop, 4
 initBelt, 5

belt.h, 5
 beltControl, 6
 beltSettings, 7
 beltStop, 7
 initBelt, 7

beltControl
 belt.c, 4
 belt.h, 6

beltSettings
 belt.c, 4
 belt.h, 7

beltStop
 belt.c, 4
 belt.h, 7

calcFreq
 rampCalculation.c, 20
 rampCalculation.h, 23

calcRampStepWidth
 rampCalculation.c, 20
 rampCalculation.h, 23

config.h, 8

getRampValue
 rampCalculation.c, 21
 rampCalculation.h, 23

getStepRampValue
 rampCalculation.c, 21
 rampCalculation.h, 23

init
 RTOS.c, 26

initBelt
 belt.c, 5
 belt.h, 7

initMotorControl
 motor.c, 12
 motor.h, 16

initMyPorts
 initMyPorts.c, 9
 initMyPorts.h, 10

initMyPorts.c, 9
 initMyPorts, 9

initMyPorts.h, 10
 initMyPorts, 10

initTimer
 timer.c, 34
 timer.h, 38

ISR_TIMER1_COMPA
 timer.c, 34

ISR_TIMER1_COMPC
 timer.c, 34

ISR_TIMER2_COMP
 timer.c, 35

ISR_TIMER3_COMPA
 timer.c, 35

main
 RTOS.c, 27

motor.c, 11
 initMotorControl, 12
 motorControl, 13
 motorSettings, 13
 stopMotors, 14
 stopMotorSlow, 14

motor.h, 15
 initMotorControl, 16
 motorControl, 17
 motorSettings, 17
 stopMotors, 18
 stopMotorSlow, 18

MOTOR_PROPERTIES, 2

motorControl
 motor.c, 13
 motor.h, 17

motorSettings
 motor.c, 13
 motor.h, 17

rampCalculation.c, 19
 calcFreq, 20
 calcRampStepWidth, 20
 getRampValue, 21
 getStepRampValue, 21

rampCalculation.h, 22
 calcFreq, 23
 calcRampStepWidth, 23
 getRampValue, 23
 getStepRampValue, 23

rampCalculationValues.h, 24

RTOS.c, 25
 AppCreateTasks, 26
 init, 26
 main, 27

setTimerSpeed
 timer.c, 35
 timer.h, 39
setVirtualTimerSpeed
 timer.c, 36
 timer.h, 39
startVirtualTimer
 timer.c, 36
 timer.h, 39
stopMotors
 motor.c, 14
 motor.h, 18
stopMotorSlow
 motor.c, 14
 motor.h, 18
stopVirtualTimer
 timer.c, 37
 timer.h, 40

taskMsgControl.c, 27
taskMsgControl.h, 28
TaskSpeedControl
 taskSpeedControl.c, 30
 taskSpeedControl.h, 31
taskSpeedControl.c, 29
 TaskSpeedControl, 30
taskSpeedControl.h, 31
 TaskSpeedControl, 31
timer.c, 32
 initTimer, 34
 ISR_TIMER1_COMPA, 34
 ISR_TIMER1_COMPC, 34
 ISR_TIMER2_COMP, 35
 ISR_TIMER3_COMPA, 35
 setTimerSpeed, 35
 setVirtualTimerSpeed, 36
 startVirtualTimer, 36
 stopVirtualTimer, 37
timer.h, 37
 initTimer, 38
 setTimerSpeed, 39
 setVirtualTimerSpeed, 39
 startVirtualTimer, 39
 stopVirtualTimer, 40

Index

- Aktoren, 27, 46
- Anforderungsliste, 130
- Anhang, 129
- Anleitungen, 145
 - Anschlussbelegung Servo-Board, 153
 - Atmel Debugging, 145
 - Bedienungsanleitung Roboter, 150
- Anschlussbelegung, 53, 153
- Antriebskonzept, 16
- Antriebssystem, 184
- Anwendungsfall
 - Bälle einladen, 84
 - Gegner ausweichen, 85
 - Position anfahren, 83
 - Roboter kontrollieren, 86
 - Wettbewerb gewinnen, 81
- AT90CAN128, 51
- Aufbau des Roboters, 11
- Aufgabenstellung, 130
- Ausblick auf Projektarbeit 2, 122
- Auswurfsklappe, 49
- Bänder, 46
- Ballschiebevorrichtung, 49
- Batterispannungsueberwachung, 122
- Bestandesaufnahme, 129
- Betriebssystem, 65
- Bilderkennung, 123
- CAN Knoten, 27
- CAN-Messages Aktorknoten, 52
- Danksagung, 127
- Datenbank, 71
- Debian, 65
- Debugging, 70
- Distanzsensor, 31
- Doxygen, 184
- Drehgeber Trommel, 33
- DVK90CAN1, 51
- Einleitung, 1
- Einzugsbänder, 46
- Einzugsklappe, 49
- Endschalter, 34
- Entwicklungsumgebung, 75
- Erfahrungen, 123
- Erweiterung Atmel, 28
- Eurobot 2008, 1
- Farbsensor, 36
- Fazit
 - Projektarbeit 1, 121
- Gyroskop, 35
- Hardware, 65
- IDE, 75
- Ideenfindung, 5
- Industrie PC, 65
- Installation Betriebssystem, 65
- Ist-Aufnahme, 129
- Java-Software, 76
- Kollision, 89
- Kompass, 34
- Komplilation, 70
- Komponenten, 78
- Konstruktion neuer Antriebsräder, 23
- Konzept, 76
- Konzeptphase, 6
- Linux, 65
- Management Summary, i
- Mechanik, 5, 123

Meisterschaft	Software
Schweizermeisterschaft, 117	Anhang, 183
Weltmeisterschaft, 119	Antrieb, 53
Motorenauslegung, 16	Betriebssystem, 65
MySQL, 71	Java, 76
Navigation, 65	Sensorknoten, 38
Navigationssystem, 63	Servoansteuerung, 52
Netzwerkkonfiguration, 71	Speichertrommel, 47
Neuerungen, 19	Speisungsanschlüsse, 62
Newsletter, 175	Sponsoren, 126, 127
Optimierung, 122	Startvorrichtung, 24
PCAN, 69	SVN, 75
Pflichtenheft, 129	Tasks
Quellcodeverwaltung, 75	TaskCanReception, 60
Rechnerische Auslegung des neuen An-	TaskCanTransmission, 60
triebs, 21	TaskMsgControl, 60
Routing, 89, 114	TaskRS232, 61
RS232, 70	TaskSpeedControl, 61
Schiebe-Arm, 25	TaskStart, 60
Schlusswort, 128	Technische Umsetzung, 12
Schrittmotor, 46	Threads, 112
Schweizermeisterschaft, 117	Tools, 65
Sensorboard, 29	uC-OS, 39
Sensoren, 27	Variante 1, 19
Distanzsensor, 31	Variante 2, 20
Drehgeber Trommel, 33	Variantenbewertung, 9
Endschalter, 34	Variantenentscheid, 11
Erweiterung Atmel, 28	Verschalung, 24
Farbsensor, 36	Weltmeisterschaft, 119
Gyroskop, 35	Wettbewerb, 129
Kompass, 34	Wireless-CAN-Debugger, 123
Sensorboard, 29	Zeitplan, 142
Servo-Board, 153	Zusammenfassung, 3