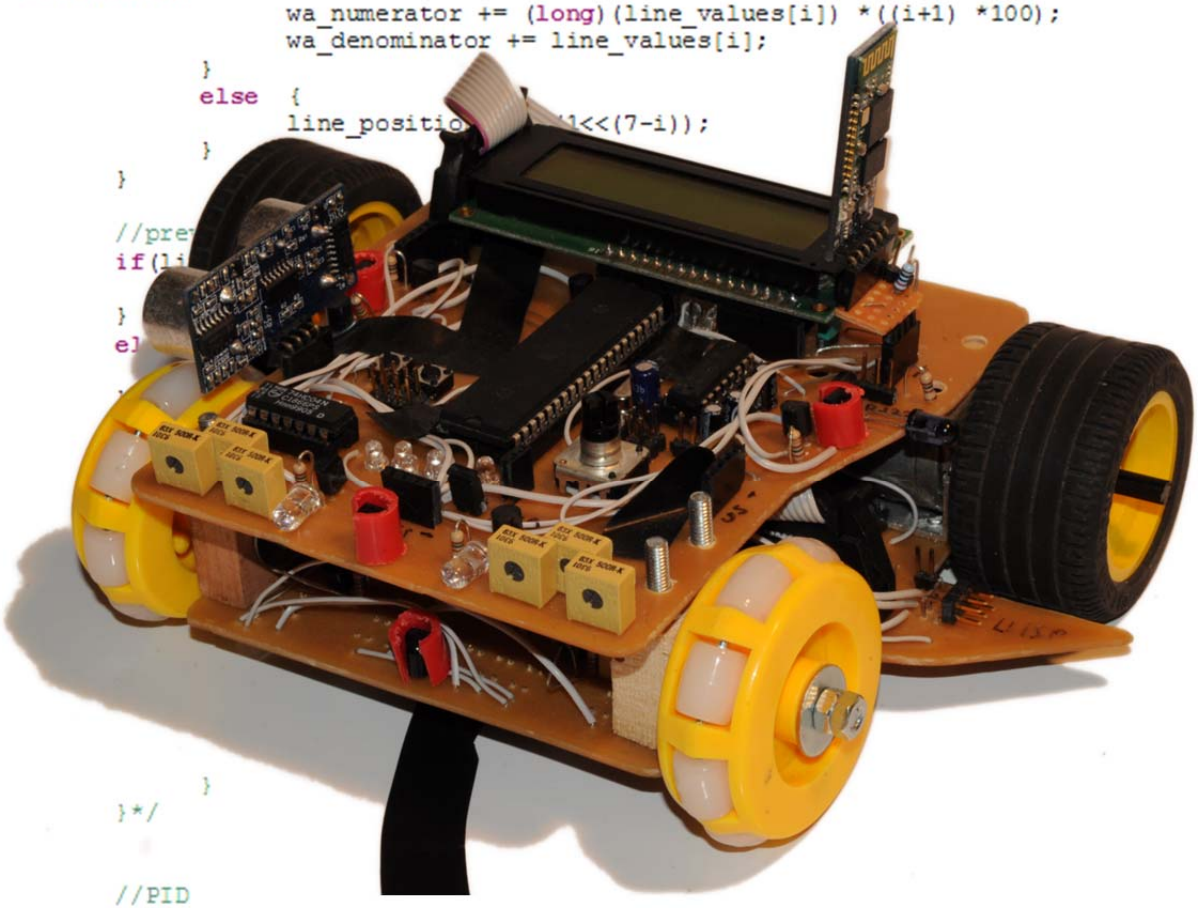
Konstruktion und Programmierung eines autonomen Roboters für den Bewerb RoboCup Junior Rescue



Alexander Kargl

BRG Graz Keplerstrasse 1

Konstruktion und Programmierung eines autonomen Roboters für den Bewerb RoboCup Junior Rescue

Fachbereichsarbeit aus Angewandter Informationstechnologie

Betreuer: Leander Brandl

Alexander Kargl, 8a

Graz, am 23.02.2012

Inhaltsverzeichnis

1 Vorwort

In der 8. Schulstufe wurde meine Faszination an der Robotik durch die Teilnahme am Wettbewerb Robocup Junior geweckt. Seitdem nehme ich nun mit meinem Team an den jährlichen Wettbewerben teil und wir versuchen unsere Roboter jedes Jahr aufs Neue zu verbessern.

Dabei kam mir die Idee die Entstehung eines neuen Roboters zu dokumentieren und darüber eine Fachbereichsarbeit zu verfassen.

In den neuen Roboter sollten einerseits die Erkenntnisse und Erfahrungen der vergangenen Jahre einfließen, und andererseits sollte im Rahmen dieser Arbeit auch neue Ideen zu getestet und umgesetzt werden. Während der Recherchen und der Entwicklungsarbeit konnte ich meinen Wissensstand in vielen Bereichen, besonders in der Programmierung und Elektronik, immens erweitern.

Mein Dank gilt besonders Herr Prof. Leander Brandl, der mich im Verlauf dieser Arbeit immer unterstützt und betreut hat. Außerdem leitet er gemeinsam mit Frau Prof. Nicole Bizijak und Herr Prof. Siegfried Patz das Freifach Robotik, wo sie immer mit Rat und Tat zur Seite standen, weswegen ich mich auch für ihr Engagement bedanken möchte.

Graz, am 23.02.2012 Alexander Kargl

2 Einleitung

Roboter lösen bei Menschen jedes Alters eine Faszination aus. Bereits heute übernehmen sie verschiedene, mehr oder weniger komplexe Arbeitsschritte in vielen Bereichen. Doch selbst für einfache Aufgaben sind komplizierte Systeme vonnöten. Die Robotik vereint viele Fachgebiete, allen voran Informatik gefolgt von Elektronik und Maschinenbau und wird ein immer bedeutenderer Forschungszweig.

Um die Forschung in diesem Gebiet weiter anzutreiben wurde 1997 die Robocup Federation gegründet, die sich zum Ziel gesetzt hat bis 2050 ein Team von Fußballrobotern zu entwickeln, das die amtierenden menschlichen Weltmeister schlägt. Dazu wurde ein Wettbewerb mit mehreren Ligen geschaffen, in denen verschiedene Universitäten und Forschungs-einrichtungen gegeneinander antreten.

Eine dieser Ligen ist Robocup Junior, die sich speziell an Jugendliche unter 19 Jahren richtet. In zwei Altersklassen konkurrieren verschiedene Teams in den Bereichen Soccer, Rescue sowie Dance miteinander. Im Vordergrund stehen jedoch der Wissenserwerb und der Austausch zwischen den Teams.

3 Aufgabenstellung

Ziel dieser Fachbereichsarbeit war die Entwicklung, Konstruktion und Programmierung eines Roboters, der im Stande sein soll die Aufgaben des Robocup Junior Bewerbes „Rescue“ nach den neusten Vorgaben und Regeln[1] des Robocup Junior Komitees zu meistern. Das ganze Projekt wurde im Zuge dieser FBA dokumentiert.

3.1 Robocup Rescue

Die Rescue-Liga hat ein Katastrophenszenario als Vorbild, in dem Roboter z.B. in eingestürzten Gebäuden Menschen finden und retten müssen.

Die Angaben beziehen sich auf die zuletzt herausgegebene Version der Regeln vom 7. Juni 2011.

Das Spielfeld besteht aus einer Arena (Abb. 3-2) mit drei Modulen, die durch Gänge sowie eine Rampe, mit einer maximalen Steigung von 25°, miteinander verbunden sind. Die Module umfassen ca. 120 x 90 cm und sind von mindestens 10 cm hohen Wänden begrenzt.

Der Roboter muss einer aufgeklebten, schwarzen Linie mit 1-2 cm Breite, die durch die Arena bis zum dritten Modul führt, folgen. Die Linie enthält Kurven mit Winkeln bis zu 90° und kann auch für bis zu 20 cm unterbrochen sein.

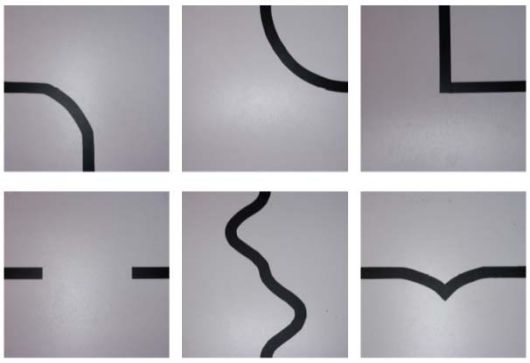


Abb. 3-1 - mögliche Linienverläufe (aus[2])

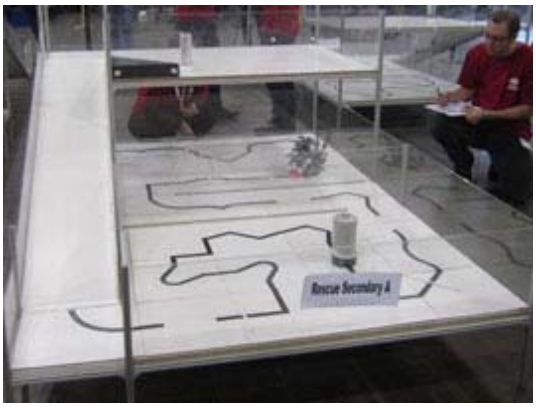


Abb. 3-2 - Rescue Arena (aus[3])

Um den Linienverlauf schnell (z.B. zwischen versch. Durchgängen) ändern zu können, sind die Linienabschnitte auf Kacheln mit 30 *30 cm Kantenlänge geklebt (Abb. 3-1). So können die Anforderungen der Bahn, und somit auch der Schwierigkeitsgrad schnell variiert werden.

Auf dem Weg durch die Arena sind verschiedene Hindernisse aufgestellt, die das Weiterkommen erschweren oder den Weg blockieren. Diese bestehen aus Getränkeflaschen, Ziegelsteinen oder anderen schweren Objekten, ausgestreuten kleinen Holzstäbchen (z.B.

[1] RoboCup Junior Technical Comittee (2011): RoboCupJunior Rescue A Rules. Online im Internet: URL: http://rcj.robocup.org/rcj2011/rescueA_2011.pdf [Stand: 21.11.2011]

[2] Organisationsteam Robocup German Open (2011): mögliche Fliesen. Online im Internet: URL: http://rcj-orga.robocupgermanopen.de/content/pdfs/2011_Rescue_A_Fliesen.pdf [Stand: 21.2.2012]

[3] Organisationsteam Robocup German Open (2011): Rescue Arena. Online im Internet: URL: http://rcj-orga.robocupgermanopen.de/content/images/rescue_komplett_arena.jpg [Stand: 21.2.2012]

Zahnstocher) sowie den so genannten „Speed Bumps“, halbierte Holzzylinder mit einer maximalen Höhe von 1 cm, die quer zur Linie aufgeklebt werden.

Im letzten Modul gibt es keine leitende Linie mehr und es muss ein „Opfer“ gefunden und gerettet werden. Das „Opfer“ wird durch eine 330 ml Getränkedose, die mit Alufolie umhüllt ist, dargestellt. Diese wird zufällig an einer beliebigen Stelle, mit mindestens 10 cm Entfernung zur Wand, positioniert. Der Roboter muss das „Opfer“ finden und in die Evakuierungszone bringen. Diese besteht aus einem rechtwinkeligen Dreieck, das sich in einer Höhe von 6 cm in einer Ecke des Modules befindet.

Das Ziel des Bewerbes ist es, eine möglichst hohe Punktezahl zu erreichen. Punkte werden für das Wiederauffinden der Linie nach einer Lücke, das Umfahren von Hindernissen (jeweils 10 Punkte) und das Bewältigen einer „Speed Bump“ (5 Punkte) vergeben. Außerdem gibt es für jeden Raum, der vom Eingang bis zum Ausgang fehlerfrei durchfahren wird, 50 Punkte und auch für die Rampe werden 20 Punkte verliehen. Das Finden und Aufnehmen des „Opfers“ wird mit 20 und das erfolgreiche Absetzten in der Evakuierungszone mit 50 Punkten belohnt.

Der Roboter muss die gesamte Arena komplett autonom und ohne Fremdeinwirkung absolvieren. Bei Fehlern muss er an den Beginn des entsprechenden Modules zurückgesetzt werden und es werden 15 Punkte pro Eingriff abgezogen.

4 Zielsetzung

Noch bevor mit der Planung des Roboters begonnen wurde, wurden einige Ziele fixiert, die der Roboter erfüllen sollte:

- **fehlerfreie Bewältigung des Parcours**: Für einen Sieg bei einem Wettbewerb ist es mittlerweile beinahe unabdingbar, dass die maximal erreichbare Punktezahl erzielt wird.
- **möglichst schnell:** Da bei einem Punktegleichstand auch die Laufzeiten in Betracht gezogen werden, ist es wichtig, schnell zu fahren. Allerdings steigt mit zunehmender Geschwindigkeit auch die Fehleranfälligkeit, sodass ein Kompromiss zwischen Geschwindigkeit und Genauigkeit gefunden werden muss.
- **einfacher und schneller Auf- und Abbau**: In Falle eines Fehlers oder Defekts ist es wichtig, dass alle Bauteile und Komponenten möglichst rasch ausgetauscht werden können, da man während eines Bewerbes unter ständigem Zeitdruck steht und auch sonst viel Zeit mit Auf- und Abbau verloren geht. Alle Teile sollten zudem von mir selbst, mit zur Verfügung stehenden Werkzeugen angefertigt werden können um einerseits Kosten zu sparen und um auch schnell viele Prototypen entwickeln zu können.
- **billig:** Da der gesamte Roboter größtenteils von mir selbst finanziert wird und ich somit nur ein sehr begrenztes Budget zur Verfügung hatte, sollten alle Teile möglichst kostengünstig sein. So werden z.B. statt teuren Fertigmodulen eigene Entwicklungen eingesetzt und die Platinen selbst hergestellt.

5 Hardware

Bevor mit der Programmierung der Software begonnen werden konnte, musste zunächst die Hardware entwickelt und gebaut werden.

5.1 Grundlegender Aufbau

Anfangs erwog ich einen Aufbau auf Sperrholzplatten, auf denen dann die einzelnen Elektronikplatinen montiert werden sollten. Dies wurde allerdings schnell zu Gunsten eines anderen Konzeptes verworfen: Die gesamte Elektronik wird auf zwei Platinen untergebracht, die gleichzeitig auch als Basisplatten dienen, an der die Motoren und der Hebemechanismus für das „Opfer“ befestigt sind. Dadurch wird einerseits das Gewicht gesenkt, was der Geschwindigkeit und der Beschleunigung zu Gute kommt, und außerdem die Verwendung von kleineren Motoren ermöglicht, die weniger Strom verbrauchen und somit auch die Akkulaufzeit erhöhen. Andererseits ermöglicht es auch einen kompakten Aufbau und eine unübersichtliche Kabelführung zwischen den Platinen kann vermieden werden.

Durch die Regeln wird die Größe des Roboters zwar nicht limitiert, dennoch wird die maximale Höhe und Breite mit 250 mm * 250 mm durch die Größe der Durchgänge zwischen den einzelnen Räumen vorgegeben.

5.2 Antriebsarten

Für Roboter gibt es unterschiedliche Antriebskonzepte mit verschiedenen Vor- und Nachteilen. Ungewöhnliche Varianten, wie z.B. ein laufender Roboter auf Beinen, bieten sich hier jedoch nicht an, da sie zu langsam und darüber hinaus auch sehr komplex zu konstruieren sind. Bei einem herkömmlichen Antrieb mit Rädern gibt es verschiedene Varianten:

- Die bekannteste ist wohl die **Ackermann-Lenkung**, die in Autos verwendet wird.
  Auch. der **Dreiradantrieb** funktioniert nach demselben Prinzip.
  Die beiden Antriebsräder befinden sich auf einer gemeinsamen Achse und werden von einem Motor angetrieben, während die Fahrtrichtung durch die schwenkbaren Vorderräder bestimmt wird. Ein Nachteil ist, dass durch eine derartige Lenkung die Mobilität des Roboters eingeschränkt wird, da z.B. eine Drehung auf der Stelle nicht möglich ist.

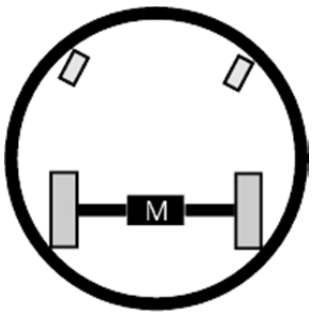


Abb. 5-1 - Schema Ackermann-Lenkung (aus [1])

- Eine der beliebtesten Antriebskonzepte bei mobilen Robotern ist der **Differentialantrieb**, bei dem zwei Räder getrennt voneinander von zwei Motoren angetrieben werden. Dadurch ist der Roboter „flink“ und wendig. Die Ansteuerung der Motoren ist sehr einfach, da keine aufwendigen Berechnungen nötig sind und die Drehung der Räder nicht überwacht werden muss.

  Zusätzlich werden, je nach Gewichtsverteilung, ein oder zwei Stützräder an den Auflagepunkten benötigt um die Balance zu wahren. Diese sollten sich sowohl nach vorne als auch zur Seite bewegen lassen und zusätzlich leichtgängig sein, um Reibungsverluste und die Bremswirkung zu minimieren. Oft eingesetzt wird hier eine sog. Freilaufkugel, wie sie in Abb. 5-3 zu sehen ist.
  Ein Problem des Differentialantriebes ist, dass durch Fertigungstoleranzen und ungleichmäßige Gewichts-verteilung innerhalb des Roboters, sich die Motoren bei gleicher anliegender Spannung mit unterschiedlichen Geschwindigkeiten drehen. Dadurch driftet der Roboter

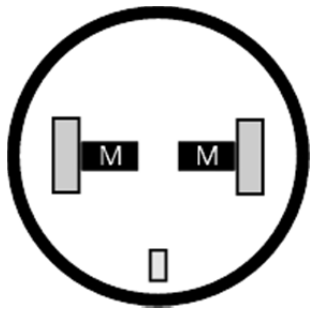


Abb. 5-2 - Schema Differentialantrieb (aus [2])

Abb. 5-3 – Freilaufkugel (aus [3])

[1] Technische Universität Chemnitz (2007): RoboKing 2008 Technische Dokumentation. Online im Internet: URL: http://www.tu-chemnitz.de/etit/proaut/rk/fileadmin/user_upload/RK2008/Downloads/roboking2008_technischedokumentation.pdf [Stand: 21.11.2011]

[2] Technische Universität Chemnitz (2007): RoboKing 2008 Technische Dokumentation. Online im Internet: URL: http://www.tu-chemnitz.de/etit/proaut/rk/fileadmin/user_upload/RK2008/Downloads/roboking2008_technischedokumentation.pdf [Stand: 21.11.2011]

[3] RobotShop: Pololu Ball Caster. Online im Internet: URL: http://www.robotshop.com/eu/productinfo.aspx?pc=RB-pol-96&lang=en-US

dann mehr oder weniger stark zur Seite ab, was eine Geradeausfahrt erschwert. Um dem entgegenzuwirken, muss eine Regelung die Motorgeschwindigkeiten korrigieren. Dazu werden oft Encoder an den Rädern angebracht, die die aktuelle Drehzahl messen. Aber auch Sensoren, die externe Bezugsquellen verwenden, wie z.B. ein Kompass oder ein Entfernungsmesser zu Wänden, können eingesetzt werden.

- Sehr dem Differentialantrieb ähnlich ist auch der **Panzerantrieb**, mittels zweier Ketten. Vorteile sind die hohe Geländegängigkeit und, dass kein zusätzliches Stützrad benötigt wird. Allerdings stehen dem eine komplexere Konstruktion, höheres Gewicht sowie mehr Schlupf als bei Rädern gegenüber.
  Ein Kettenantrieb hätte zwar bei der Bewältigung der „Speed Bumps“ Vorteile, jedoch besteht die Gefahr, dass die Ketten auf den lose verteilten Zahnstochern durchdrehen und der Roboter nicht mehr voran kommt.

Nach Abwägen der Vor- und Nachteile der verschiedenen Konzepte, sticht klar der Differentialantrieb hervor. Für meinen Einsatzzweck sind Schnelligkeit und eine hohe Wendigkeit erforderlich, wohingegen die Nachteile nicht so schwer wiegen. Da die meiste Zeit der leitenden Linie nachgefahren wird, muss ohnedies ständig die Fahrtrichtung geändert werden. Während der Lücken und in der letzten Zone, wo keine Linie mehr vorhanden ist, ist eine genaue Geradeausfahrt trotzdem vonnöten, weswegen gegebenenfalls Drehzahlencoder eingebaut werden müssen.

Bei der Wahl des Stützrades habe ich mich gegen die oft verwendete Freilaufkugel entschieden, da die Gefahr besteht, dass sie an den „Speed Bumps“ hängen bleibt. Stattdessen werden zwei Allseitenräder (Abb. 5-4) verwendet. Aufgrund der kreisförmig angeordneten Rollen können sie sich nicht nur vorwärts, sondern auch gleichzeitig zur Seite bewegen.

Abb. 5-4 – Allseitenrad

5.3 Räder

Bei der Wahl der Antriebsräder ist es besonders wichtig darauf zu achten, dass sie auf dem Untergrund der Arena, der meistens aus weiß lackierten MDF-Platten besteht, gut haften. Dies ist vor allem auf der Rampe wichtig, da der Roboter sonst leicht hinunterrutschen und dabei unter Umständen beschädigt werden kann. Außerdem sollten die Räder mit wenig Aufwand fest und sicher an der Motorachse zu befestigen sein.

Beides trifft auf Räder der Firma Lego zu. Es gibt sie in unterschiedlichsten Varianten und Größen und durch das einheitliche Stecksystem können sie zu Testzwecken schnell gegen andere getauscht werden.

Philippe E. Hurbain hat in einer Testreihe die Reibungskoeffizienten verschiedener Lego-Reifen auf diversen Oberflächen bestimmt[1]. Derzeit ist der Roboter mit den Reifen (Produktnummer 6594) (Abb. 5-5) bestückt. Im oben angeführten Vergleichtest sind diese zwar nicht die Besten, ihre Haftung ist aber trotzdem mehr als ausreichend.

Abb. 5-5 - Lego Reifen

5.4 Motoren

Vor der Auswahl von Motoren sollte zumindest ungefähr die benötigte Motorleistung berechnet werden.

Der Durchmesser der Räder d ist mit 5,6 cm gegeben. Für das Masse m wird 0,5 kg angenommen. Die Motoren sollten den Roboter relativ schnell auf $v = 0{,}5\ {}^{\mathrm{m}}/_{\mathrm{s}}$ beschleunigen können, was bereits eine akzeptable Geschwindigkeit ist.

Der Umfang U des Reifen ergibt sich aus: $U = d * \pi = 5{,}6\mathrm{cm} * 3{,}14 \cong 17{,}59\mathrm{cm}$.

Nun muss die Drehzahl n des Motors berechnet werden, die benötigt wird um auf die geforderte Geschwindigkeit v zu kommen.

$$n = \frac{v}{U}\,\frac{1}{t} = \frac{0{,}5\ {}^{\mathrm{m}}/_{\mathrm{s}}}{0{,}175\mathrm{m}}\,\frac{1}{\mathrm{s}} = 2{,}84\ {}^{1}/_{\mathrm{s}} \qquad \text{Formel 5-1}$$

Das heißt, die Motorachse muss sich mindestens 2,84-mal in der Sekunde drehen, das sind ca. 170 U/min.

Bevor die Motorkraft berechnet werden kann, muss auch noch die auftretende Reibung, vor allem die Rollreibung, einkalkuliert werden. Die Luftreibung kann bei solch niedrigen Geschwindigkeiten ignoriert werden und auch die Haftreibung ist vernachlässigbar.

In einem Nachschlagwerk [2] wird der Rollreibungskoeffizient c_r von Hartgummi auf Stahl mit 0,0077 angegeben. Als Koeffizient für die Räder wird daher 0,005 geschätzt.

Die Rollreibung wird folgendermaßen berechnet:

$$\begin{aligned} F_r &= c_r * m * g = 0{,}005 * 0{,}5\mathrm{kg} * 9{,}81\ {}^{\mathrm{m}}/_{\mathrm{s}^2} \\ &= 0{,}024\mathrm{N} \end{aligned} \qquad \text{Formel 5-2}$$

Die größte auftretende Kraft wird benötigt um den Roboter auf v zu beschleunigen. Diese Geschwindigkeit soll er in 1 s erreichen. Die Beschleunigung a errechnet sich folgendermaßen:

$$v = a * t \;\rightarrow\; a = \frac{v}{t} = \frac{0{,}5\ {}^{\mathrm{m}}/_{\mathrm{s}}}{1\mathrm{s}} = 0{,}5\ {}^{\mathrm{m}}/_{\mathrm{s}^2} \qquad \text{Formel 5-3}$$

[1] HURBAIN, Philippe : Wheels, Tyres and Traction. Online im Internet: URL: http://www.philohome.com/traction/traction.htm [Stand: 28.12.2011]

[2] BEARDMORE, Roy: Coefficients of Friction. Online im Internet: URL: http://www.roymech.co.uk/Useful_Tables/Tribology/co_of_frict.htm#Rolling [Stand: 27.2.2012]

Die Kraft beträgt somit:

$$F_B = m * a = 0{,}5\text{kg} * 0{,}5\frac{\text{m}}{\text{s}^2} = 0{,}25\text{N}$$ Formel 5-4

Insgesamt ist also folgende Kraft aufzubringen:

$$F_{\text{ges}} = F_r + F_B = 0{,}024\text{N} + 0{,}25\text{N} \cong 0{,}27\text{N}$$ Formel 5-5

Da die Last gleichmäßig auf die zwei Antriebsmotoren verteilt wird, muss jeder Motor

$$F_{Motor} = \frac{F_{\text{ges}}}{2} = \frac{0{,}27\text{N}}{2} \cong 0{,}13\text{N}$$ Formel 5-6

aufbringen.

Das Drehmoment M, das an der Achse jedes Motors anliegt:

$$M = F_{Motor} * \frac{d}{2} = 0{,}13\text{N} * 0{,}028\text{m} = 0{,}00364\text{Nm} = 3{,}6\text{mNm}$$ Formel 5-7

Abb. 5-6 – Motoren (aus[1])

Qualitätsmotoren, die zugleich klein, leicht und leistungsfähig sind, sind leider auch sehr teuer und liegen deshalb weit außerhalb meines Budget-Rahmens. In einem Internet-Auktionshaus wurde ich dann fündig und ersteigerte 2 „No-Name"-Motoren aus Asien, zu denen aber nicht viele Details preisgegeben wurden. Das Getriebe hat eine Untersetzung im Verhältnis 1:20, die Drehzahl wurde mit 300 U/min und das Spitzen-Drehmoment mit 0,39 Nm angegeben.

Selbst wenn von einem Wirkungsgrad des Getriebes von nur 50% ausgegangen wird und für das unbekannte Dauerdrehmoment ein Drittel des Spitzendrehmoments angenommen wird, ist das Drehmoment des Motors bei weitem ausreichend und auch die Rampe sollte ohne Probleme zu bewältigen sein:

$$M_{\text{eff}} = M_{\text{Motor}} * \text{Eff} = 0{,}13\text{Nm} * 50\% = 0{,}065\text{Nm} = 65\text{mNm}$$ Formel 5-8

Die Berechnung der effektiven Geschwindigkeit zeigt, dass auch die anfangs gesetzte Sollgeschwindigkeit von $0{,}5\ ^{\text{m}}/_{\text{s}}$ überschritten werden kann:

$$V_{\text{eff}} = U * n_{\text{Motor}} = 0{,}175\text{m} * 5\frac{1}{\text{s}} = 0{,}875\ ^{\text{m}}/_{\text{s}}$$ Formel 5-9

(vgl. [2],[3],[4])

[1] VirtualVillage: Gearmotor 300rpm. Online im Internet: URL: http://images.villageorigin.com/003604-001/001.jpg [Stand 23.1.2012]
[2] RN-Wissen: Motorkraft berechnen. Online im Internet: URL: http://www.rn-wissen.de/index.php/Motorkraft_berechnen [Stand: 22.12.2011]
[3] HTWG-Konstanz: Berechnung zur Auslegung der Motoren. Online im Internet: URL: http://www.robotik.in.htwg-konstanz.de/htdocs/components/com_mambowiki/index.php/Berechnung_zur_Auslegung_der_Motoren [Stand: 22.12.2011]
[4] Technische Universität Chemnitz (2007): RoboKing 2008 Technische Dokumentation. Online im Internet: URL: http:/www.tu-

5.5 CAD

Nachdem die einzelnen mechanischen Komponenten ausgesucht wurden und auch der grundsätzliche Aufbau festgelegt wurde, konnten die Komponenten in einem CAD- Programm modelliert werden.

Meine Wahl fiel hier auf die Software Inventor von Autodesk, da ich damit bereits bei der Konstruktion vorheriger Roboter Erfahrung gesammelt habe. Außerdem wird eine kostenlose und uneingeschränkte Version für Schüler und Studenten bereitgestellt.

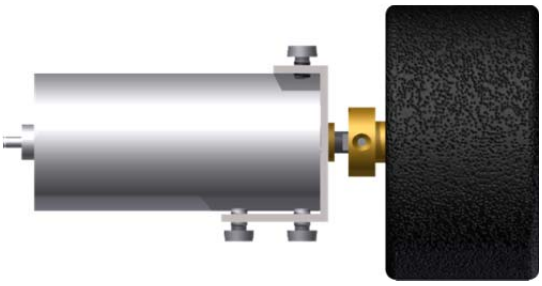


Abb. 5-7 - CAD Modell Antrieb

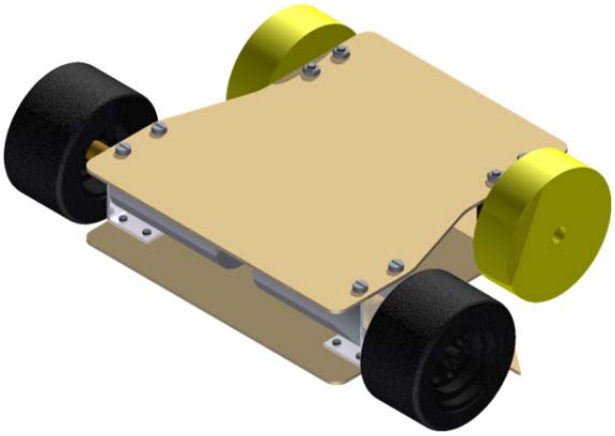


Abb. 5-8 - CAD Modell

5.6 Greifer

Aufgrund von Zeitmangel konnte der Greifer bis zur Abgabe der Fachbereichsarbeit nicht fertiggestellt werden.

chemnitz.de/etit/proaut/rk/fileadmin/user_upload/RK2008/Downloads/roboking2008_technischedokumentation.pdf [Stand: 21.11.2011]

6 Elektronik

Wie in Abb. 6-1 ersichtlich, wird eine Vielzahl an Sensoren und dazugehöriger Elektronik benötigt, um die Aufgaben zu bewältigen. Um die in Kapitel 4 definierten Zielvorgaben möglichst gut umzusetzen, werden hauptsächlich Sensoren verwendet, die aus wenigen Teilen bestehen und im besten Fall mehrere verschiedene Aufgaben erfüllen können.

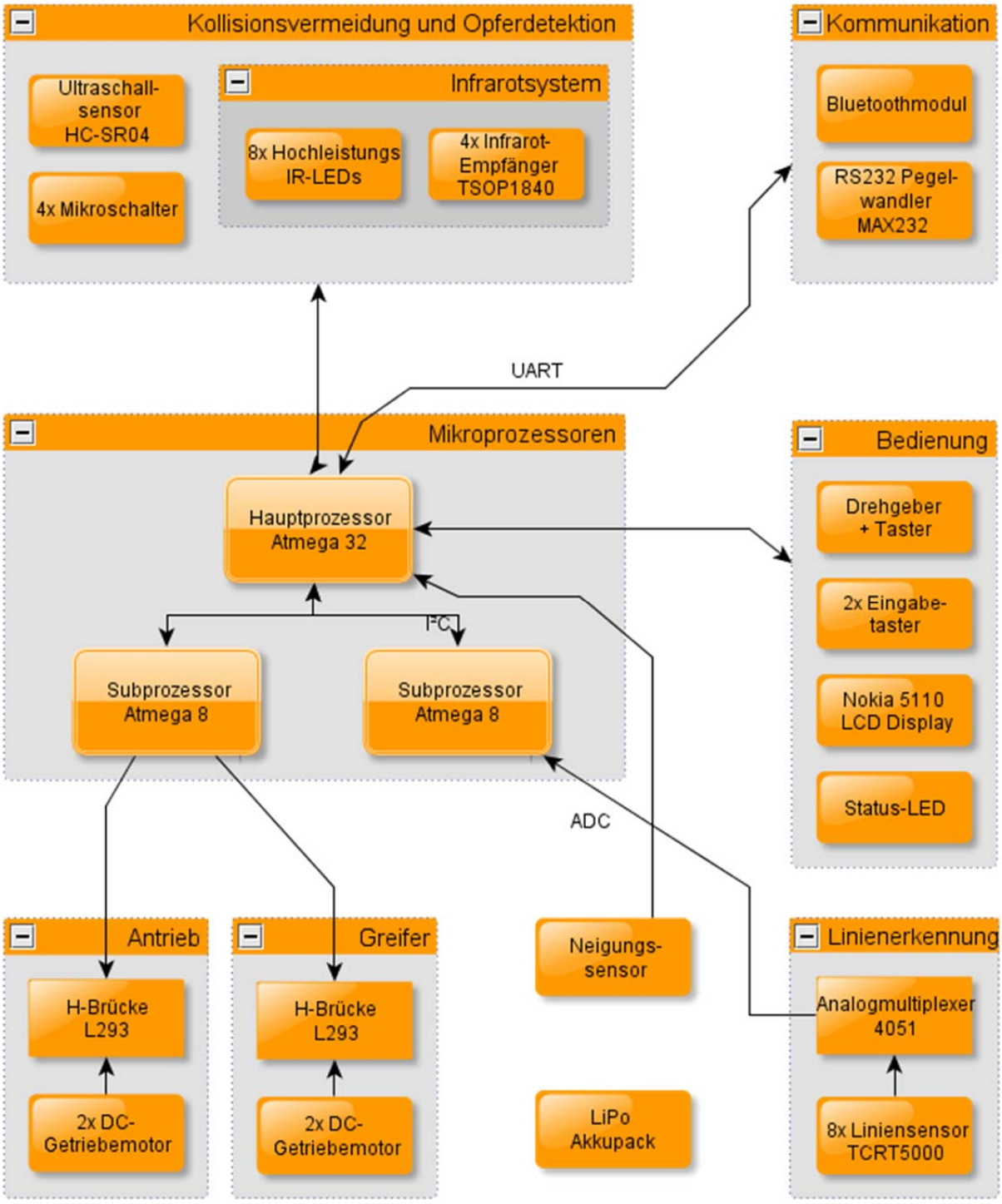


Abb. 6-1 - verwendete Komponenten

6.1 Schaltungsdesign und Platinenlayout

Zuerst wurde für die einzelnen Sensoren und Elektronikkomponenten jeweils ein Schaltplan (siehe Anhang A-1) mithilfe der PCB-Entwicklungssoftware Eagle entworfen und dann auf einem Steckbrett aufgebaut. So konnten verschiedene Konzepte auf ihre Tauglichkeit geprüft und die Funktionsfähigkeit der Schaltungen oft noch erheblich optimiert werden.

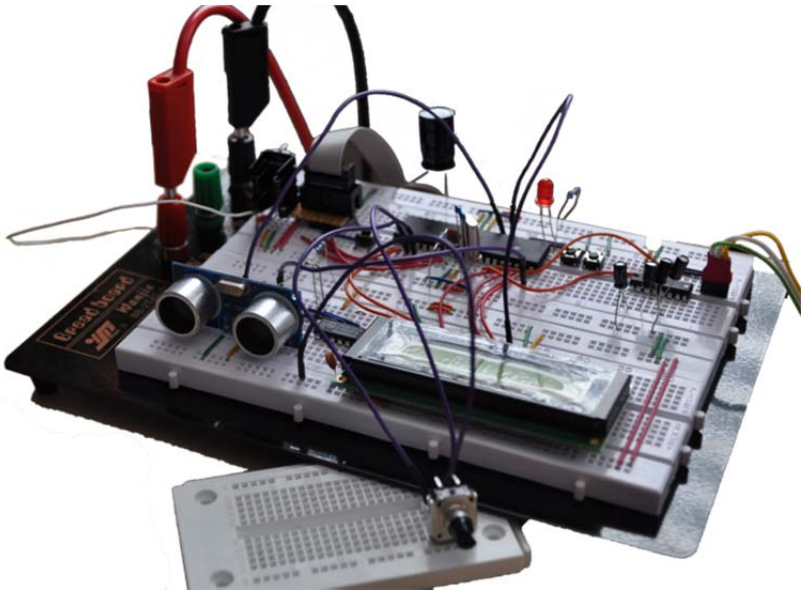


Abb. 6-2 - Schaltungsentwurf und -test auf einem Steckbrett

Nachdem die Tests der einzelnen Elemente zufriedenstellend abgeschlossen wurden, war der nächste Schritt das Erstellen des Platinenlayouts was wiederum mit Eagle erfolgte. Zuerst wurde der Umriss der Basisplatten aus der CAD-Zeichnung nach Eagle importiert und wichtige Elemente, wie die Motoren und die dazugehörigen Halterungen eingezeichnet. Dann wurden Komponenten mit fixer Platzierung, wie z.B. die Liniensensoren unten vorne oder die Infrarotemitter und –empfänger, positioniert. Auch bei der Anordnung der anderen Teile musste darauf geachtet werden, dass Stecker und Buchsen, sowie Taster frei zugänglich sind. Die Platinen sollen später von mir selbst hergestellt werden, deshalb verzichtete ich auf eine doppelseitige Platine mit Kupferflächen auf Ober- und Unterseite. Verbindungen auf der Oberseite werden durch Kabel- oder Drahtbrücken gebildet.

Um Platz zu sparen werden Kleinteile wie Widerstände und Kondensatoren fast vollständig in SMD-Ausführung verbaut. Dabei werden die Bauteile, nicht wie üblich an Drähten, die auf der jeweils anderen Seite verlötet werden, befestigt, sondern direkt auf der Oberfläche verlötet (Abb. 6-3). Bei den Mikroprozessoren und Integrierten Schaltungen (z.B. Motortreiber, Multiplexer) wurden trotzdem die um einiges größeren DIP-Gehäuse und Sockel in Durchstecktechnik

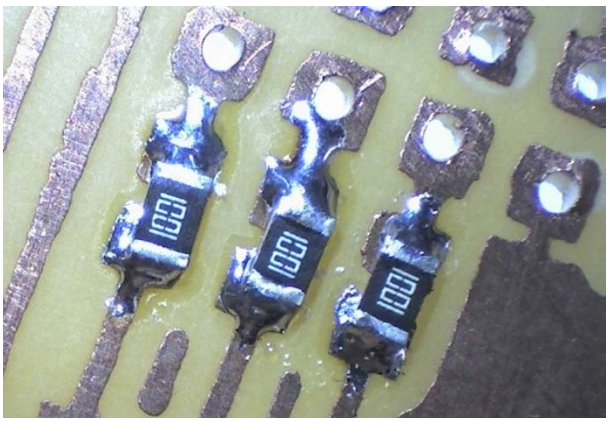


Abb. 6-3 - SMD-Widerstände unter Mikroskop.
Größenvergleich: Die Leiterbahn links ist ca. 0,5mm breit

verwendet, um diese Bauteile im Falle eines Defektes schnell und ohne Löten wechseln zu können.

Ungenutzte Kupferflächen werden als Massefläche genutzt, die Potenzialunterschiede minimiert und die Leiterbahnen gegenüber induzierten Störungen abschirmt. Gleichzeitig dient sie auch zur Kühlung von Leistungsbauteilen, wie den Motortreibern oder dem Spannungsregler, da Kupfer eine sehr gute Wärmeleitfähigkeit besitzt und die Abwärme durch die große Fläche effektiv an die Umgebung abgegeben werden kann.

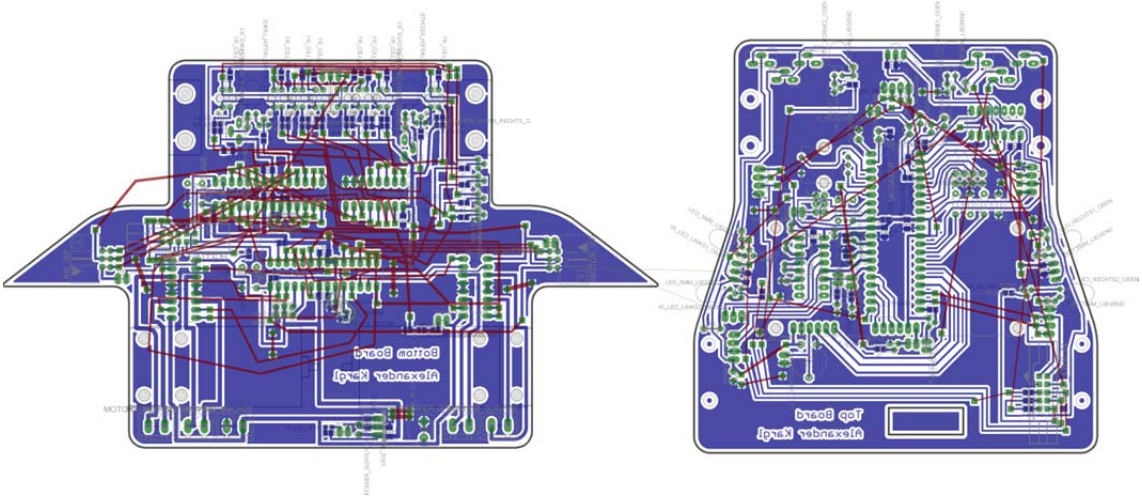


Abb. 6-4 - Layout der beiden Platinen

6.2 Herstellung der Platinen

Da eine professionelle Herstellung der Leiterplatten, vor allem in kleinen Stückzahlen, sehr teuer ist, wurden die Platinen von mir selbst angefertigt. Dabei ging ich nach dem Tonertransferverfahren vor. Zuerst wird das fertige Layout gespiegelt mit einem Laserdrucker ausgedruckt. Das verwendete Papier beeinflusst die Qualität des Ergebnisses dabei maßgeblich. Es soll den Toner nur wenig aufsaugen, deswegen sind hochglänzende Papiersorten wie z.B. aus Werbeprospekten oder Magazinen oder auch Transparentfolie gut geeignet. Die Kupferseite der Platine muss, z.B. mit Aceton, gründlich gereinigt werden. Dann wird der Ausdruck auf der Platine ausgerichtet und bei mittlerer Temperatureinstellung mit einem Bügeleisen aufgebügelt. Dabei ist die richtige Dosierung des Drucks wichtig, denn bei zu wenig wird der Toner nicht

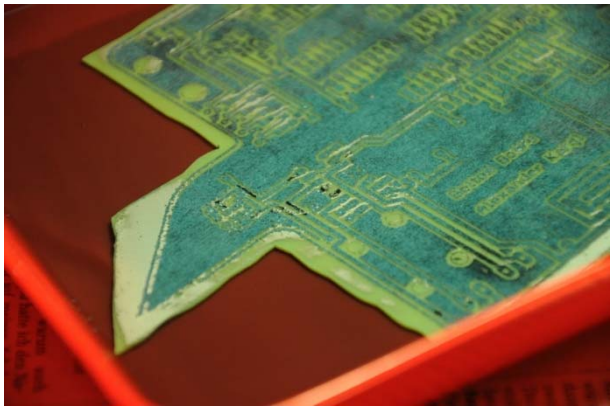


Abb. 6-6 - Platine während des Ätzvorganges

Abb. 6-5 - Fertige Platine nach Ätzen, reinigen und zuschneiden

übertragen, bei zu viel Druck verläuft er. Anschließend wird die abgekühlte Platine in Seifenwasser gelegt, worin sich das Papier auflöst und Papierreste einfach mit dem Finger abgerubbelt werden können.

Der nächste Schritt ist der eigentliche Ätzvorgang. Als Ätzmittel wird Natriumpersulfat (chemische Formel $Na_2S_2O_8$) im Verhältnis 1:5 in Wasser aufgelöst. Um den Ätzvorgang zu beschleunigen, wird das Ätzbad auf ca. 40 – 50 °C erwärmt. Die Tonerschicht auf der Platine schützt dabei die Leiterbahnen, die nicht weggeätzt werden sollen. Abb. 6-6 zeigt die Platine während des Ätzvorganges, es sind nur mehr Reste des Kupfers vorhanden, der Rest wurde bereits aufgelöst.

Nach dem Ätzen wird die Tonerschicht mit Aceton und einer Bürste entfernt und die Leiterplatte in die richtige Form gesägt und gefeilt. Außerdem müssen die Löcher für die einzelnen Bauteile gebohrt werden (Abb. 6-7).

Dann muss mit einem Multimeter und einer Lupe nach Kurzschlüssen und nach Unterbrechungen in den Bahnen gesucht werden und, falls vorhanden, müssen diese mit einem Skalpell durchtrennt bzw. mit Draht repariert werden.

Zuletzt kommt die eigentliche Bestückung der Platine mit Bauteilen. Angefangen wird mit den kleinen SMD-Teilen, dann die anderen Teile, gereiht nach Höhe und zum Schluss die Verkabelung und Drahtbrücken.

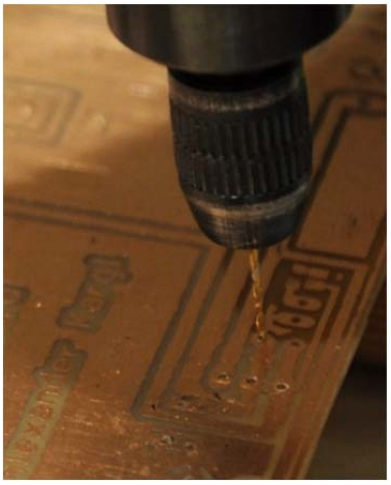


Abb. 6-7 - Bohren der Platine

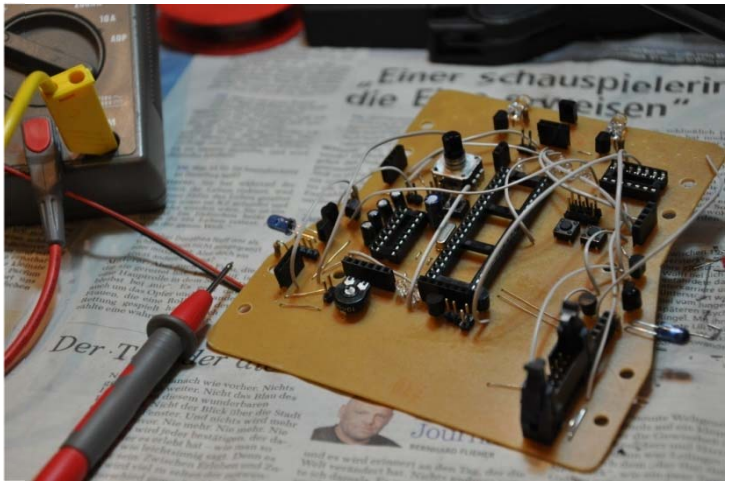


Abb. 6-8 - bestückte Platine

6.3 Mikroprozessoren

Für die Steuerung werden drei 8bit Prozessoren aus der Atmega-Reihe von Atmel eingesetzt, da ich mit diesen bereits aus vorherigen Projekten Erfahrung habe. Sie sind bei Hobbyisten sehr beliebt und auch weit verbreitet, weswegen es viele gute Artikel und Beispielcodes gibt.

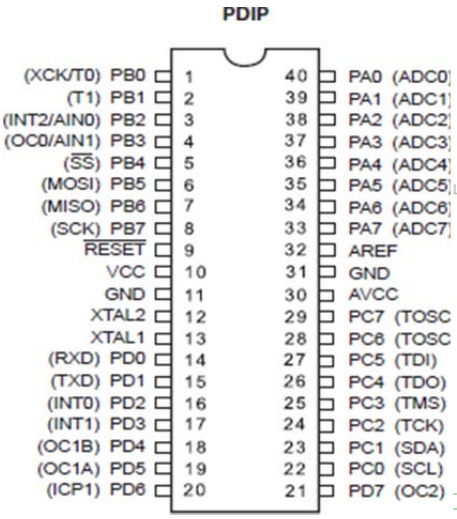


Abb. 6-9 - Pinout Atmega32

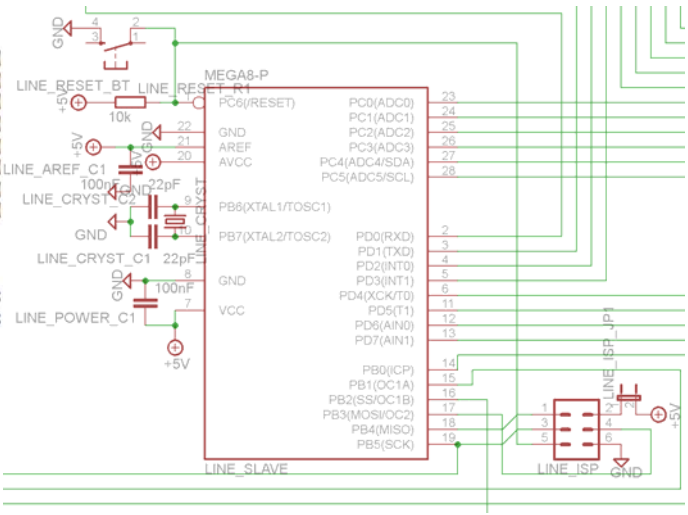


Abb. 6-10 - Basisbeschaltung Atmega

Allerdings wäre ein einziger Mikrocontroller mit der Vielzahl an Sensoren und Aktoren überfordert. Dies könnte man zwar mithilfe von Multiplexern, die die vorhandenen Ein- und Ausgänge erweitern, umgehen was aber zu erheblichen Performance-Einbrüchen führen würde. Deswegen werden stattdessen drei Controller eingesetzt, die untereinander über den I²C-Bus kommunizieren und für jeweils eigene Aufgaben zuständig sind.

Als Hauptprozessor wird ein Atmega32 verwendet, der mit 16 MHz getaktet ist. Seine Aufgaben sind die Kommunikation mit dem Benutzer, Steuerung von Teilen der Infrarot Kollisionsvermeidung, die Ansteuerung des Ultraschallsensors, Abfragen der Daten von den Slave-Kontrollern, sowie die Navigation durch die Arena an sich.

Als Slave-Kontroller kommen zwei Atmega8 zum Einsatz, etwas kleinere Varianten des Atmega32, die auch mit 16 MHz getaktet sind.

Einer ist für das Auslesen und Verarbeiten (Kalibrierung, Normalisierung,…) der Liniensensoren und das Schalten der einzelnen IR-Leds des Abstandssystems zuständig.

Der andere übernimmt die Ansteuerung des Antriebs und des Greifers.

6.4 Infrarot-Kollisionsvermeidung

Oft werden zur Hinderniserkennung normale Taster verwendet. Diese haben jedoch den Nachteil, dass der Roboter mit dem Hindernis bereits kollidiert sein muss, damit diese ansprechen. Eine elegantere Lösung sind Sensoren, die bereits ab einer bestimmten Entfernung ansprechen.

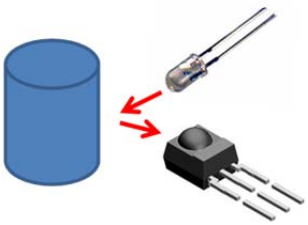


Abb. 6-11 - Funktionsprinzip IR-Sensoren

Das von einer Infrarot-LED ausgesandte Licht wird ab einer gewissen Entfernung reflektiert und von einem IR-Empfänger

wieder empfangen. Das ganze ließe sich bereits mit einer simplen IR-LED und einem IR-Phototransistor umsetzten. Hierbei besteht jedoch das Problem, dass durch wechselnde Umgebungen auch die Lichtverhältnisse variieren. So enthält z.B. Sonnen- und Glühlampenlicht einen hohen Infrarotanteil.

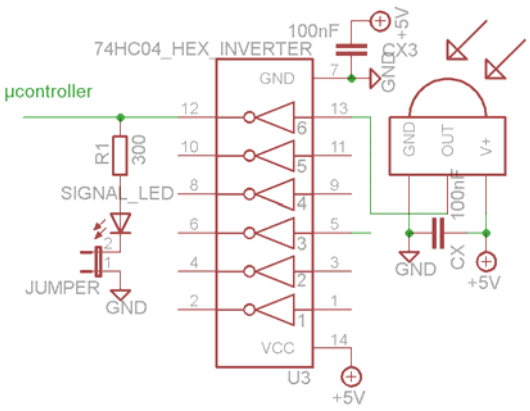


Abb. 6-12 - Schaltung IR-Receiver

Um eine umständliche Kalibrierung an die jeweiligen Bedingungen zu vermeiden, wird das IR-Signal mit einer bestimmten Frequenz moduliert. Es existieren bereits fertige IR-Empfänger, die diese Signale dann herausfiltern.

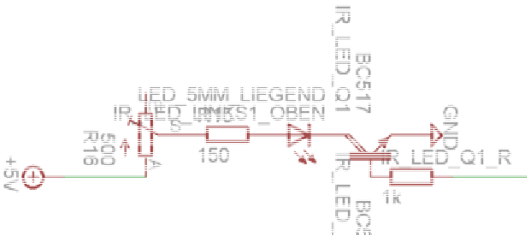


Abb. 6-13 - Schaltung IR Transmitter

Im Roboter befinden sich auf den beiden Seiten, sowie vorne oben und unten jeweils ein IR-Empfänger vom Typ TSOP4838, die auf mit 38 kHz modulierte Signale ausgelegt sind. Beiderseits der Empfänger ist jeweils eine IR-Led angebracht, die individuell geschaltet werden kann. So kann durch abwechselndes Schalten und Kombinieren der Transmitter auch eine grobe Richtungsbestimmung bzw. eine Reichweitenerhöhung (beide Transmitter ein) erzielt werden.

Die Reichweite der Sensoren wird durch die Leuchtstärke der Led bestimmt, die wiederum jeweils mit einem Potentiometer stufenlos geregelt werden kann. Bei einem Strom von 20 mA (das derzeitige Maximum) wird ein Hindernis ab 10 cm Entfernung erkannt, bei 5 mA erst ab 2 cm.

Da der Ausgang der IR-Empfänger „active-low“ ist, wird das Signal mit einem IC invertiert der auch gleich Leds schaltet, die den aktuellen Status der einzelnen Empfänger anzeigen.

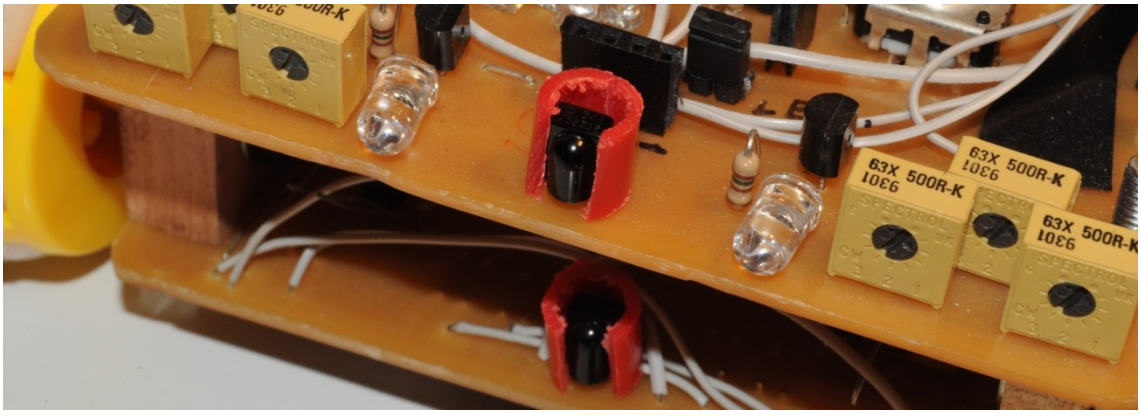


Abb. 6-14 - IR-Anti-Kollisonssystem

Ein Problem, das auftreten kann ist, dass das Licht je nach Oberflächenbeschaffenheit unterschiedlich reflektiert wird und somit auch der Auslösepunkt, ab dem ein Hinderniss erkannt wird, verschoben wird.

6.5 Ultraschallsensor

Im Gegensatz zu den binären Infrarot-Abstandssensoren (Kapitel 6.4) liefert der Ultraschallsensor vom Typ HC-SR04 eine sehr genaue Entfernungsmessung mit einer Auflösung von ca. ±1 cm im Bereich von 2 bis (theoretisch) 500 cm. Dies ist vor allem bei der Erkennung und beim Aufnehmen des „Opfers“ wichtig.

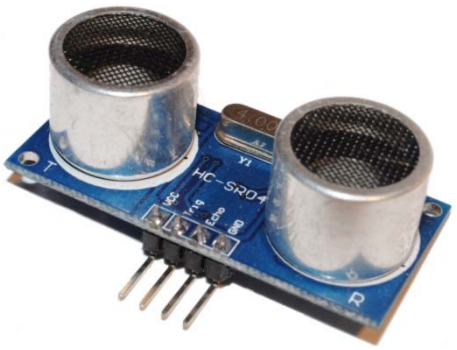


Abb. 6-15 - Ultraschallsensor HC-SR04

Das Funktionsprinzip ist relativ einfach: Um eine Messung zu starten wird der Trigger-Pin für ca. 10 µs auf High-Pegel gelegt. Dadurch wird ein Ultraschallsignal ausgesendet und der Echo-Pin geht auf High. Der ausgesendete Ultraschallimpuls wird von einem etwaigen Hindernis zurückgeworfen und sobald er wieder empfangen wird, geht der Echo-Pin wieder auf Low. Dadurch entsteht ein Impuls der direkt proportional zur Entfernung ist (siehe Abb. 6-16). Die Entfernung D errechnet sich aus der Hälfte der Dauer t des Impulses multipliziert mit der Schallgeschwindigkeit (344m/s in Luft bei 21 °C).

$$D[\mathrm{m}] = \frac{344\,{}^{\mathrm{m}}/_{\mathrm{s}} * t[\mathrm{s}]}{2} \rightarrow D[\mathrm{cm}] = \frac{172 * t[\mu\mathrm{s}]}{10000} \cong \frac{t[\mu\mathrm{s}]}{58}$$

Formel 6-1

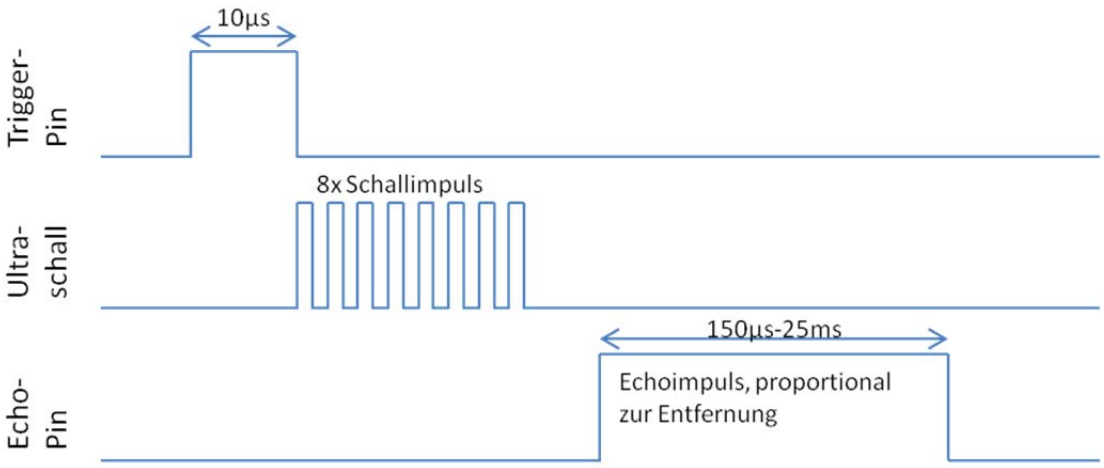


Abb. 6-16 - Schema Ultraschallsensor

Da sich die Schallwellen kegelförmig mit einem Öffnungswinkel von 15 °C ausbreiten, kann nicht genau bestimmt werden wo genau das Objekt sich vor dem US-Sensor befindet. Der so induzierte Ortungsfehler vergrößert sich mit zunehmender Entfernung.

6.6 Liniensensoren

Zur Linienerkennung werden sog. Reflexkoppler vom Typ TCRT5000 eingesetzt. Diese bestehen aus einer IR-LED (Abb. 6-19) sowie einem Phototransistor. Je nach Farbe wirft der Untergrund unterschiedlich viel Licht zurück und die Lichtintensität kann durch den Phototransistor gemessen werden.

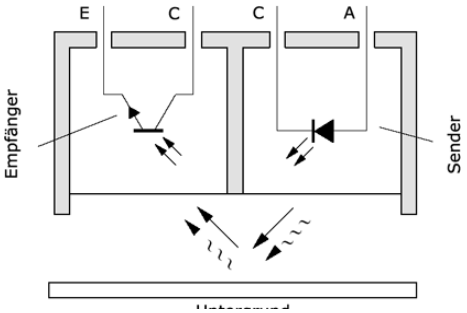


Abb. 6-17 - Funktion Reflexkoppler

Um sie über den Analog-Digital-Konverter des Mikrocontrollers einlesen zu können, wird durch den Phototransistor und den Pullup-Widerstand eine Art Spannungsteiler gebildet. Da der Atmega8 nicht über genug Eingänge mit einem ADC besitzt, werden die Signale der 8 Lichtsensoren über einen Multiplexer der Reihe nach an den Mikrocontroller weitergeleitet. Dieser kann durch drei Steuerleitungen bestimmen, welcher Eingang durchgeschaltet wird.

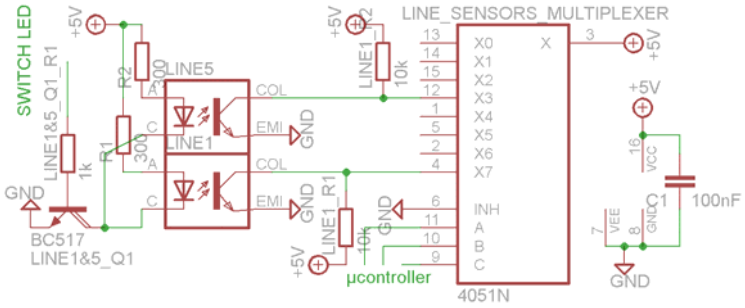


Abb. 6-18 - Schaltplan Lichtsensor

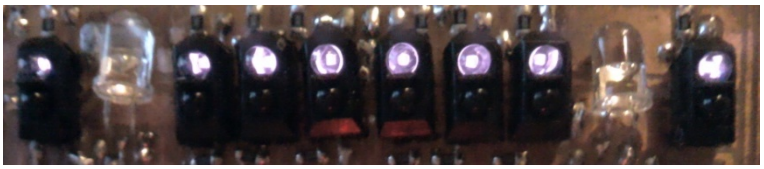


Abb. 6-19 - aktivierte IR-Led der Lichtsensoren

6.7 Motoransteuerung

Da Motoren im Normalfall einen hohen Strom benötigen, können sie nicht direkt an den Pins des Mikrocontrollers betrieben werden und müssen mit einem geeigneten Bauteil (z.B. Relais, Transistor) geschaltet werden. Außerdem soll auch die Drehrichtung änderbar, sowie das Bremsen des Motors möglich sein. Dazu wird häufig eine sog. H-Brücke (Abb. 6-20) eingesetzt. Durch Schließen und Öffnen der vier Schalter S_n, wird die Betriebsart des Motors bestimmt.

Dabei muss beachtet werden, dass in keinem Fall S1 und S2 bzw. S3 und S4 gleichzeitig geschlossen sind, da daraus ein Kurzschluss der Versorgungsspannung resultieren würde.

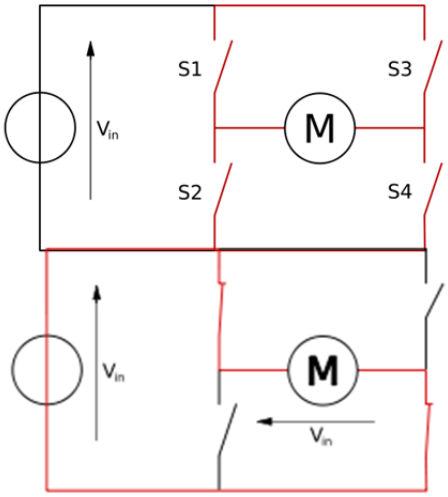


Abb. 6-20 – vereinfachte Darstellung H-Brücke (aus[1],[2])

Relais sind bauartbedingt groß, verschleißen und sind träge und laut, weshalb als Schalter meist Transistoren bzw. MosFETs verwendet werden. Da Motoren induktive Lasten darstellen, müssen die empfindlichen Halbleiter durch eine Freilaufdiode vor Spannungsspitzen geschützt werden. Diese wird antiparallel zur Stromflussrichtung zum Motor angeschlossen[3].

Es gibt bereits viele fertige, integrierte Motortreiberschaltkreise, weswegen gegen einen Aufbau mit diskreten Komponenten entschieden wurde, da diese mehr Platz verbrauchen und im Normalfall auch teurer sind. Bei der Auswahl des Motortreibers muss darauf geachtet werden, dass er sowohl den Strom des Motors im normalen Betrieb als auch die Spitzenströme beim Starten oder bei einer Richtungsänderung schalten kann, innerhalb der vorgesehenen Betriebsspannung funktioniert und, dass der Innenwiderstand und der daraus resultierende Spannungsabfall und die Erwärmung möglichst niedrig sind.

Verbaut wurde schlussendlich ein Motortreiber vom Typ L293D. In ihm sind zwei H-Brücken zur Ansteuerung von zwei Motoren sowie diverse Schutzfunktionen enthalten. Entscheidend für die Auswahl dieses Bauteiles war der geringe Preis, die Verfügbarkeit im sockelbaren DIP-Gehäuse, das einen schnellen Austausch möglich macht, sowie, das keine externen Bauteile benötigt werden. Die Motoren benötigen durchschnittlich um die 170 mA, was noch innerhalb der 600 mA liegt, die der Motortreiber maximal, längerfristig schalten kann. Um Spannungseinbrüche zu verhindern ist ein „low-ESR“ Pufferkondensator mit 2000 µF verbaut.

---

[1] BUTTAY Cyril: Structure of an H-bridge. Online im Internet: URL: http://upload.wikimedia.org/wikipedia/commons/thumb/d/d4/H_bridge.svg/500px-H_bridge.svg.png [Stand: 5.2.2012]

[2] BUTTAY Cyril: The two basic states of a H-bridge. Online im Internet. URL: http://upload.wikimedia.org/wikipedia/commons/thumb/f/f2/H_bridge_operating.svg/500px-H_bridge_operating.svg.png [Stand: 5.2.2012]

[3] RN-Wissen: Diode. Online im Internet: URL: http://www.rn-wissen.de/index.php/Diode#Freilaufdiode [Stand 15.01.2012]

Um die entstehende Wärme abzuführen sind die Massezuleitungen des Motortreibers extra großflächig ausgeführt (siehe 6.1).

Da nicht immer mit voller Geschwindigkeit der Motoren gefahren werden soll und z.B. für Kurvenfahrten eine Differenzierung der Geschwindigkeiten nötig ist, erfolgt die Anpassung der Drehzahl über PWM (Pulsweitenmodulation). Dabei wird die Stromversorgung des Motors durch ein variierendes Rechtecksignal ständig sehr schnell ein- und ausgeschalten. Das Tastverhältnis $DC = \frac{t_{ein}}{t_{ein}+t_{aus}}$ bestimmt dabei die Geschwindigkeit. Durch die hohen Schaltfrequenzen und die Trägheit des Motors erscheint das Signal für den Motor wie eine Gleichspannung $U_m = U_{ein} * DC$ (vgl. [1]).

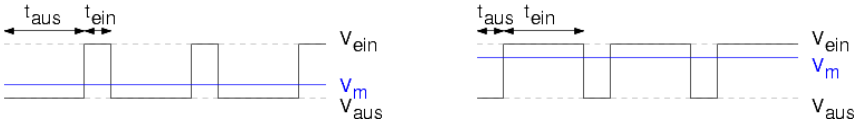


Abb. 6-21 - Schema PWM-Signale (aus [2])

[1] Mikrocontroller.net: Pulsweitenmodulation. Online im Internet: URL: http://www.mikrocontroller.net/articles/Pulsweitenmodulation [Stand: 15.1.2012]
[2] SCHWARZER Andreas (2004): Pulsweitenmodulation. Online im Internet: URL: http://www.mikrocontroller.net/articles/Datei:Pwm1.png

6.8 Kommunikation

Ein wichtiges Hilfsmittel während der Entwicklung ist die Kommunikation mit dem Computer über die RS232 Schnittstelle. In einem Terminal-Programm lassen sich so Variablen- und Sensorwerte zur Laufzeit anzeigen und zur späteren Analyse abspeichern. So können Fehler schneller gefunden und Algorithmen optimiert werden.

Die TTL-Pegel des Mikrocontrollers müssen allerdings mit einem Transreceiver-IC auf -12 V bzw. +12 V konvertiert werden um den RS232-Spezifikationen zu entsprechen.

Außerdem ist zusätzlich noch ein Bluetoothmodul verbaut, das als Kabelersatz eingesetzt wird. Hierbei entfällt nicht nur die Umwandlung der Pegel, sondern auch die Kabelverbindung, wodurch Daten auch z.B. während der Fahrt gesendet werden können.

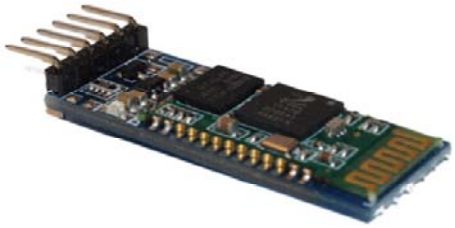


Abb. 6-22 - Bluetoothmodul

6.9 Bedienung

Zur Kommunikation mit dem Nutzer verfügt der Roboter über ein LCD-Display mit 2 Zeilen á 16 Zeichen, sowie über einen Drehgeber und zwei Taster. Auf dem Display kann z.B. ein Menü angezeigt werden, in dem verschiedene Einstellungen im Programm getätigt und Werte verändert werden können, ohne den Code neu kompilieren zu müssen.

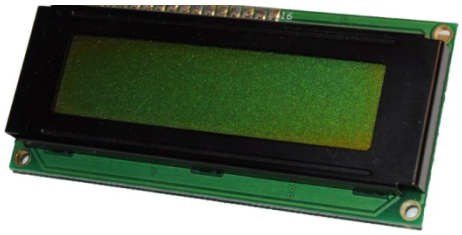


Abb. 6-23 - LCD-Display

6.10 Stromversorgung

Als Energiequelle wird ein Lithium Polymer-Akku mit 7,4 V Nennspannung und einer Kapazität von 900 mAh verwendet. Diese Technologie hat sich im Modellbau schon länger gegenüber anderen Typen durchgesetzt. Vorteile von Lithium-Polymer-Akkus sind die kleine Bauweise, das leichte Gewicht und, dass sie (kurzfristig) sehr hohe Ströme bereitstellen können. Auch tritt bei ihnen kein Memory Effekt auf, d.h. sie können unabhängig vom Ladezustand ohne Kapazitätsverlust geladen werden.

Bei der Handhabung ist jedoch einiges zu beachten! der Akku darf niemals kurzgeschlossen werden und die Zellen dürfen zu keiner Zeit den vorgesehenen Spannungsbereich über- bzw. unterschreiten, da sie sonst explodieren können.

Die Versorgungsspannung von 5 V für die Elektronik wird mit dem Linearregler 7805 erzeugt. Die Spannungsdifferenz fällt am Regler ab und wird als Wärme abgegeben weswegen ein Kühlkörper benötigt wird.

Abb. 6-24 - LiPo-Akku

7 Software

Die Programmierung der Mikrocontroller erfolgt in C, da ich damit bereits Erfahrung habe und die Entwicklungswerkzeuge kostenlos zur Verfügung stehen. Als Entwicklungsumgebung verwende ich die bekannte IDE Eclipse mit dem Avr-Plugin und dem avr-gcc-Compiler. Im Gegensatz zu einem normalen Editor, bietet Eclipse Komfortfunktionen wie Syntaxhighlighting und Code-Vervollständigung und eine umfassende Projekt- und Dateiverwaltung.

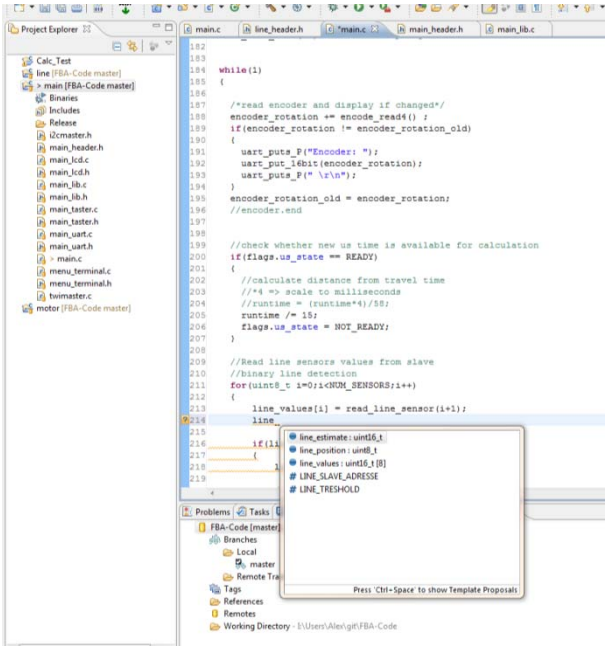


Abb. 7-1 - Eclipse

Um jederzeit einen Überblick über gemachte Änderungen zu haben und diese gegebenen falls zu einem späteren Zeitpunkt wieder rückgängig machen zu können, wurde das Versionsverwaltungssystem Git verwendet.

```
commit 36cb3745c294124130367f5fec08a65a2b95e291
Author: Alexander Kargl <alexanderkargl@gmail.com>
Date:   Sat Feb 11 16:04:09 2012 +0100

    LCD is now working

commit 9c532dbcc41d39ccc69a55c770f3538392723fd8
Author: Alexander Kargl <alexanderkargl@gmail.com>
Date:   Fri Feb 10 23:49:36 2012 +0100

    line: saving and reading calibration constants in/from eeprom works now
    :) Normalizing still not working correctly tough (manual calculation is
    correct)

    main: changed menu
```

Abb. 7-2 - Versionsverwaltung mit Git

Der gesamte Quellcode ist im Anhang zu finden. Dabei wurde versucht, diesen möglichst umfassend zu kommentieren und auch auf Formatierung und selbsterklärende Variablen- und Funktionsnamen wurde Wert gelegt um den Quellcode auch für Laien einigermaßen verständlich zu machen. Aus diesen Gründen werden hier auch nur einige interessante Teile herausgegriffen und genauer erläutert.

7.1 Aufbau

Das Programm ist auf verschiedene Dateien verteilt, um den Code übersichtlich zu halten. So gibt es z.B. eine Bibliothek mit Funktionen für die I2C-Kommunikation, eine zur Ansteuerung des Displays, zur Kommunikation über die serielle Schnittstelle und zum Auslesen der einzelnen Sensoren. Außerdem werden auch alle Konstanten und viele globale Variablen in externen Dateien ausgelagert. Diese werden dann alle in einer zentralen Hauptdatei eingebunden und gebündelt.

7.1.1 Liniensubprozessor

Das Hauptprogramm läuft in einer Endlosschleife, während die I2C-Kommunikation im Hintergrund über Interrupts abgewickelt wird. Je nach Inhalt, der vom Master beschriebenen Register, liest es die Liniensensoren ein und verarbeitet die Daten gegebenenfalls weiter (siehe 7.2). Außerdem werden noch die IR-Leds für das Infrarot-Abstands-System ein- und ausgeschaltet.

7.1.2 Motorensubprozessor

Hier werden die vom Master empfangenen Befehle mit Geschwindigkeit und Drehrichtung der Motoren validiert und umgesetzt.

7.1.3 Hauptprozessor

Das Hauptprogramm wird am besten durch ein Flussdiagramm dargestellt:

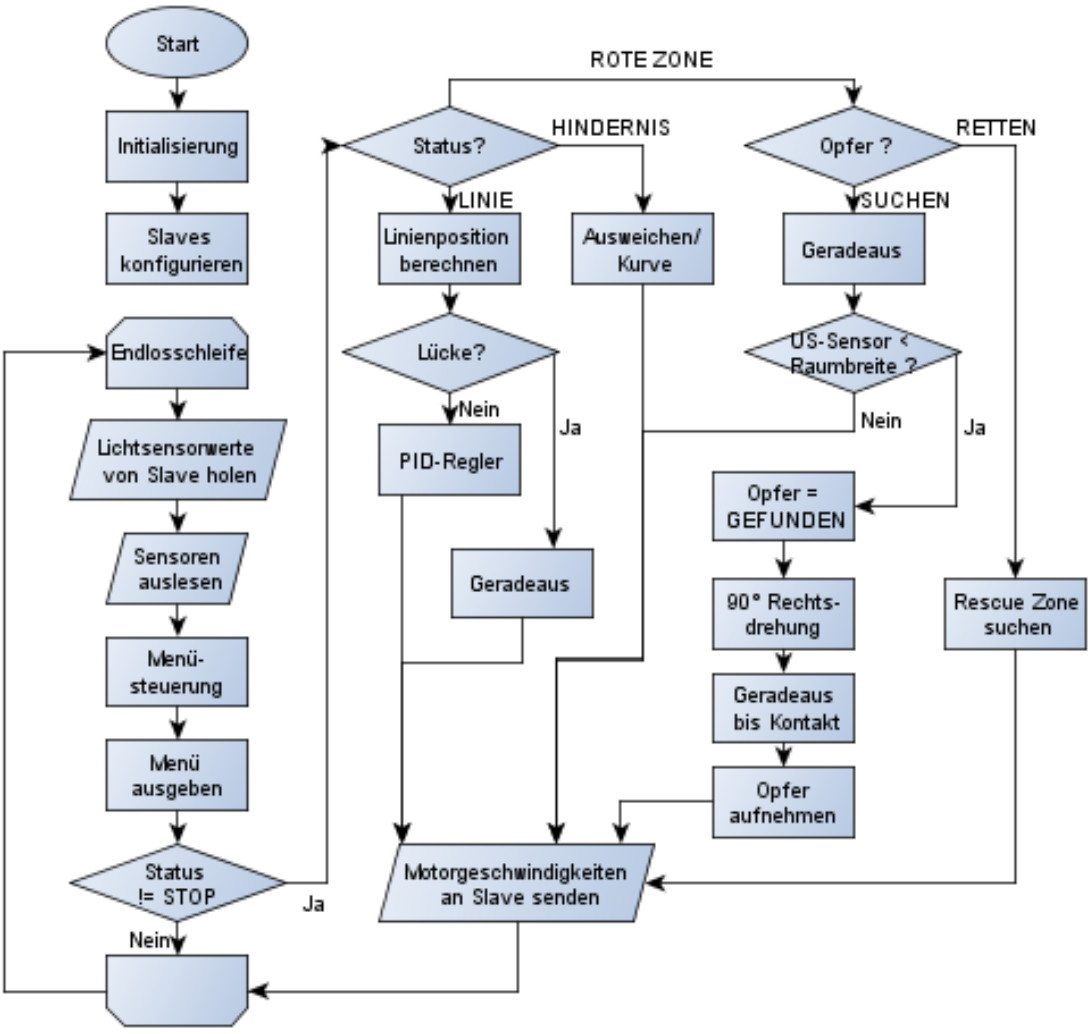


Abb. 7-3 - Flussdiagramm Hauptprogramm

7.2 Linienverarbeitung

7.2.1 Normalisierung

Aufgrund von Fertigungstoleranzen der Phototransistoren und Widerstände und da auch der Abstand der Liniensensoren zum Boden nicht immer exakt gleich ist, variieren die Messwerte der einzelnen Sensoren bei gleichem Untergrund doch erheblich. Deshalb müssen die Messwerte vor einer Weiterverarbeitung zuerst normalisiert werden. Dazu wird für jeden Sensor der jeweilige Wert auf der hellsten (weißer Boden) und dunkelsten (schwarze Linie) Oberfläche der Bahn gemessen und im Eeprom (nicht flüchtiger Speicher) abgelegt, um auf die Konstanten auch nach einer Unterbrechung der Stromversorgung zugreifen zu können und das aufwendige manuelle Ermitteln der Werte bei nicht bei jedem Start durchführen zu müssen.

Dann werden die Werte jedes Sensors nach folgender Formel normalisiert, wobei $K_{\min}$ dem kleinsten gemessenen Wert und $K_{\max}$ dem höchsten entspricht[1]:

$$\text{Ergebnis} = \frac{(\text{Rohwert} - K_{\min}) * 1000}{K_{\max} - K_{\min}}$$

Dadurch werden die Messwerte immer so skaliert, dass sie über der Linie annähernd 1000 entsprechen und über dem weißen Boden nahe null sind.

7.2.2 Berechnung der Linienposition

Um die Position der Linie zu ermitteln, werden die Sensorwerte je nachdem ob sie unter bzw. über einem definierten Grenzwert liegen, als 0 oder 1 gespeichert. So erhält man z.B. folgende Informationen:

- 00000001 – Linie ist unter äußerstem rechten Sensor
- 01100000 – Linie ist zwischen den Sensoren 2 und 3
- 00000000 – keine Linie vorhanden

Um eine größere Genauigkeit zu erhalten und die Informationen der Sensoren komplett auszureizen, ist jedoch eine andere Methode erforderlich. Su et al[2] berechnen mithilfe eines gewichteten Mittelwertes die Position der Linie, wobei x_{0-7} den Koordinaten (Gewicht) der Sensoren und y_{0-7} dem jeweiligen Messwert entsprechen:

$$x = \frac{\sum_{i=0}^{7} x_i y_i}{\sum_{i=0}^{7} y_i}$$ [2]

Als Gewichtungswert wird 1000 * der Sensorposition verwendet, z.B. für den ersten Sensor 1000 und für den achten 8000. In der Berechnung werden allerdings nur Werte über einem bestimmten Grenzwert berücksichtigt um Störungen zu minimieren.

---

[1] LEE, Chyi-Shyong et al (2008): A hands-on laboratory for autonomous mobile robot design courses. Online im Internet: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/1158.pdf [Stand: 17.01.2012]

[2] SU, Juing-Huei et al (2010): An intelligent line-following robot project for introductory robot courses. Online im Internet: http://www.wiete.com.au/journals/WTE&TE/Pages/Vol.8,%20No.4%20(2010)/9-15-SU-J-H.pdf [Stand: 18.1.2012]

7.2.3 PID-Regler

Der folgende Abschnitt orientiert sich an einem Artikel der „Chicago Area Robotics Group" [1].

Bei niedrigen Geschwindigkeiten fällt ein Programm für einen Linienverfolger relativ simpel aus. Nach folgendem Schema können schon recht passable Ergebnisse erzielt werden:

```
[...]

while(1)  {
      [...] //Sensoren einlesen

      //Linie auf linkem Sensor
      if(sensor1 == 1)  {
        drehung_nach_rechts();
      }
      //Linie auf mittlerem Sensor
      if(sensor2 == 1)  {
        fahre geradeaus();
      }
      //Linie auf rechtem Sensor
      if(sensor3 == 1)  {
        drehung_nach_links();
      }
}
```

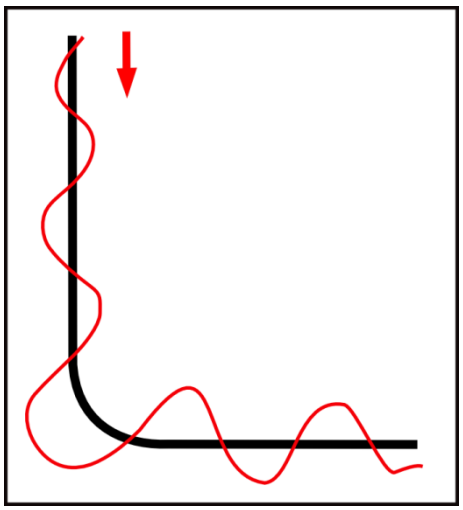


Abb. 7-4 - Verhalten simpler Linienfolger

Allerdings wird sich der Roboter so wie in Abb. Abb. 7-4 verhalten und dabei stark oszillieren, wobei nicht nur wertvolle Zeit verloren geht, sondern auch die ständige Gefahr besteht, dass die Linie nicht mehr aufgefunden wird. Außerdem funktioniert diese Methode nur unterhalb einer bestimmten Geschwindigkeit.

Deswegen werden bestimmte Regelungstechniken angewandt um die Bewegungen des Roboters genau zu kontrollieren. Besonders beliebt ist hierbei der sogenannte PID (**P**roportional-**I**ntegral-**D**ifferenzial)-Regler. Die oben gezeigte Methode zieht nur die Position der Linie in Betracht, der PID-Regler berücksichtigt auch wie schnell der Roboter sich von Seite zu Seite bewegt und wie lange er nicht über der Linie zentriert ist.

Erklärung einiger Begriffe, die für die Implementierung eines PID-Reglers wichtig sind:

- **Ziel** – Das gewünschte Ergebnis, hier die Positionierung der Linie mittig unter dem Roboter.
- **Aktuelle Position** – die derzeitige, gemessene, Ausrichtung des Roboters zur Linie.
- **Fehler –** die Differenz zwischen **Ziel** und **aktueller Position**. Kann sowohl positiv, negativ als auch 0 sein.
- **Proportional –** misst die Entfernung des Roboters zur Linie. Das P-Glied ist der Grundstock um die Position des Roboters anhand von Sensordaten zu erfassen.
- **Integral -** misst den, sich mit der Zeit ansammelnden **Fehler**. Während der Roboter nicht in Zielposition ist, erhöht sich das I-Glied, je länger desto größer wird es.
- **Differenzial –** misst die Frequenz, mit der der Roboter sich zwischen rechts-links bzw. links-rechts bewegt. Je schneller der Roboter sich von Seite zu Seite bewegt, desto größer ist auch das D-Glied.

[1] Chicago Area Robotics Group (2007): PID for Line Following. Online im Internet: http://www.chibots.org/index.php?q=node/339 [Stand: 20.1.2012]

K_p, K_d und K_i sind Konstanten, die benutzt werden um die Auswirkungen des jeweiligen Gliedes zu erhöhen.

Die Umsetzung in ein Programm zum Folgen der Linie sieht so aus:

```
while(1)  {

[...] //read Sensors etc.

int16_t proportional = ((int)line_estimate) - SET_POINT;  //should be 0 when
over line
int16_t derivative = proportional - last_proportional;
integral += proportional;
//Remember last position
last_proportional = proportional;

//Difference between motor speeds. if positive -> turn right, if negative
turn left.
//Magnitude determines sharpness of turn
error_value = proportional / Kp + integral / Ki + derivative * Kd;

//drive motors
const int max = 600; //maxium speed

if(error_value > max)  {
      error_value = max;
}
if(error_value < -max)  {
      error_value = -max;
}

if(error_value < 0)  {
      motor1_speed = max + error_value;
      motor2_speed = max;
}
else  {
      motor1_speed = max;
      motor2_speed = max - error_value;
      }

[...] //drive motors etc.

}
```

Der wichtigste Schritt ist nun die Konstanten K_p, K_d und K_i an den jeweiligen Roboter anzupassen. Dies geschieht in einem langwierigen „Trial& Error“ Verfahren.

8 Literaturverzeichnis

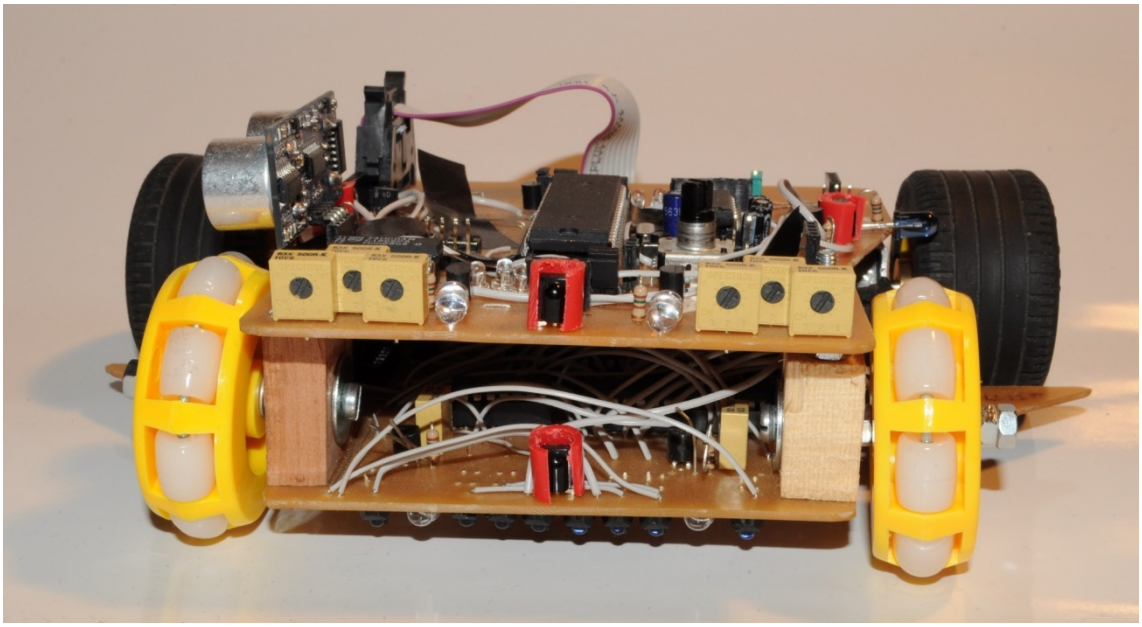
- BEARDMORE, Roy: Coefficients of Friction. Online im Internet: URL: http://www.roymech.co.uk/Useful_Tables/Tribology/co_of_frict.htm#Rolling [Stand: 27.2.2012]
- Chicago Area Robotics Group (2007): PID for Line Following. Online im Internet: http://www.chibots.org/index.php?q=node/339 [Stand: 20.1.2012]
- HTWG-Konstanz: Berechnung zur Auslegung der Motoren. Online im Internet: URL: http://www.robotik.in.htwg-konstanz.de/htdocs/components/com_mambowiki/index.php/Berechnung_zur_Auslegung_der_Motoren [Stand: 22.12.2011]
- HURBAIN, PhilippePhilippe E. Hurbain: Wheels, Tyres and Traction. Online im Internet: URL: http://www.philohome.com/traction/traction.htm [Stand: 28.12.2011]
- LEE, Chyi-Shyong et al (2008): A hands-on laboratory for autonomous mobile robot design courses. Online im Internet: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/1158.pdf [Stand: 17.01.2012]
- Mikrocontroller.net: Pulsweitenmodulation. Online im Internet: URL: http://www.mikrocontroller.net/articles/Pulsweitenmodulation [Stand: 15.1.2012]
- RN-Wissen: Diode. Online im Internet: URL: http://www.rn-wissen.de/index.php/Diode#Freilaufdiode [Stand 15.01.2012]
- RN-Wissen: Motorkraft berechnen. Online im Internet: URL: http://www.rn-wissen.de/index.php/Motorkraft_berechnen [Stand: 22.12.2011]
- RoboCup Junior Technical Comittee (2011): RoboCupJunior Rescue A Rules. Online im Internet: URL: http://rcj.robocup.org/rcj2011/rescueA_2011.pdf [Stand: 21.11.2011]
- SU, Juing-Huei et al (2010): An intelligent line-following robot project for introductory robot courses. Online im Internet: http://www.wiete.com.au/journals/WTE&TE/Pages/Vol.8,%20No.4%20(2010)/9-15-SU-J-H.pdf [Stand: 18.1.2012]
- Technische Universität Chemnitz (2007): RoboKing 2008 Technische Dokumentation. Online im Internet: URL: http://www.tu-chemnitz.de/etit/proaut/rk/fileadmin/user_upload/RK2008/Downloads/roboking2008_technischedokumentation.pdf [Stand: 21.11.2011]

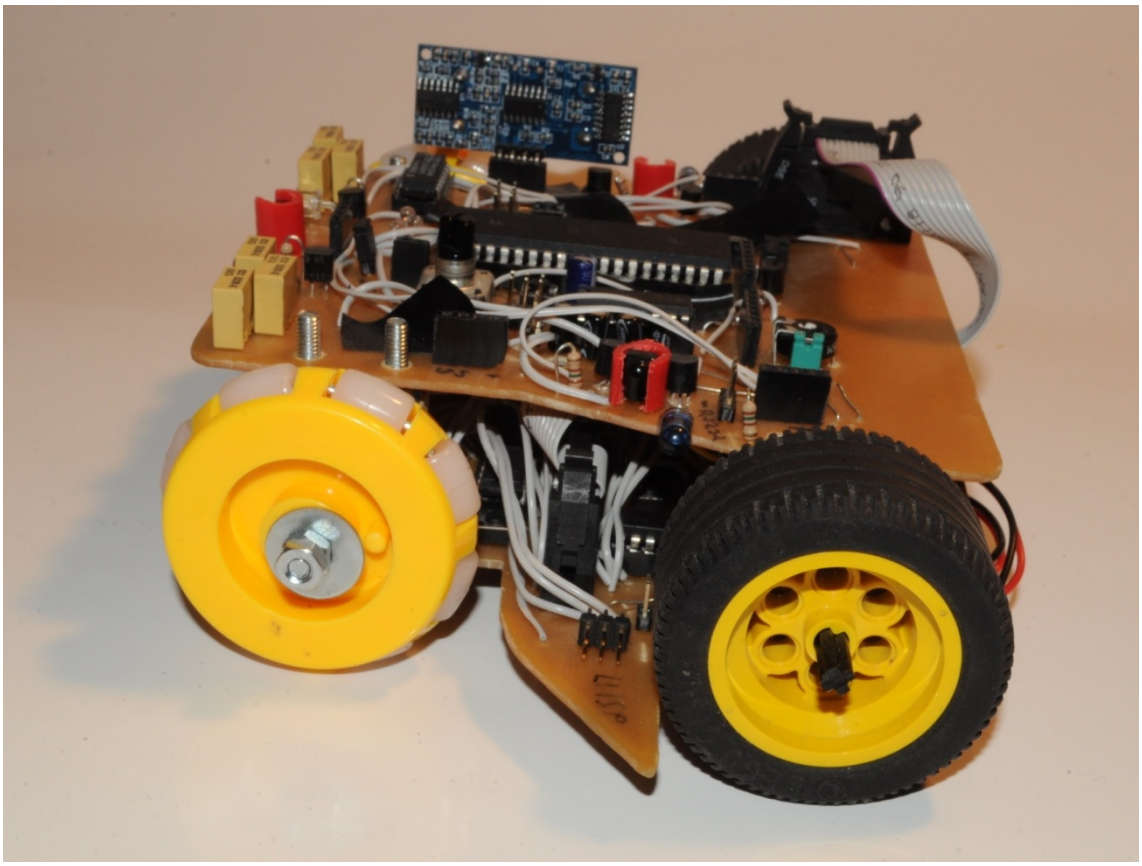
A) Anhang

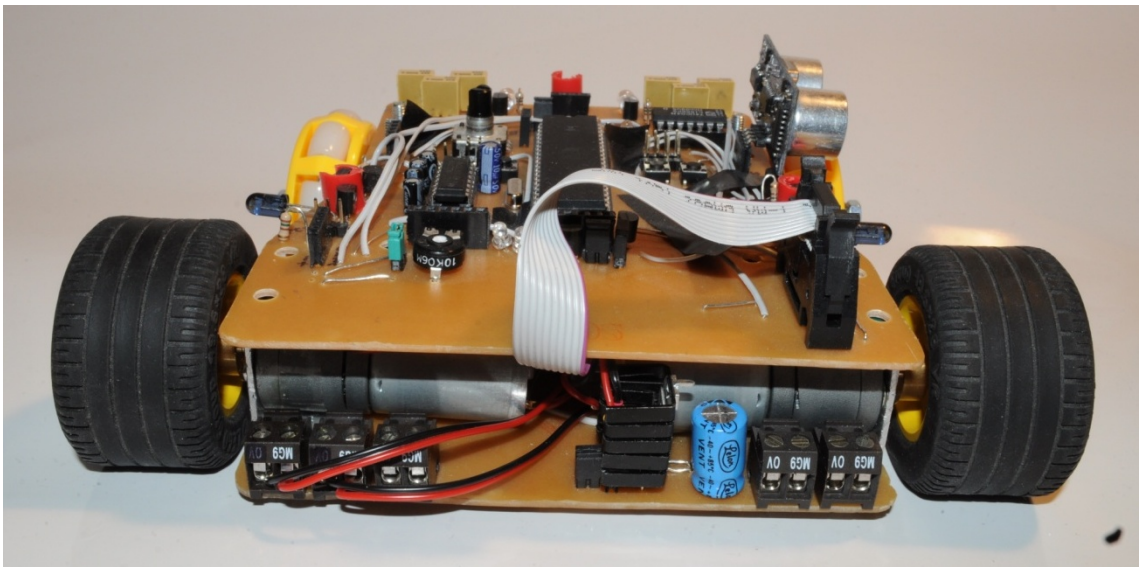
Der Anhang ist folgendermaßen gegliedert:

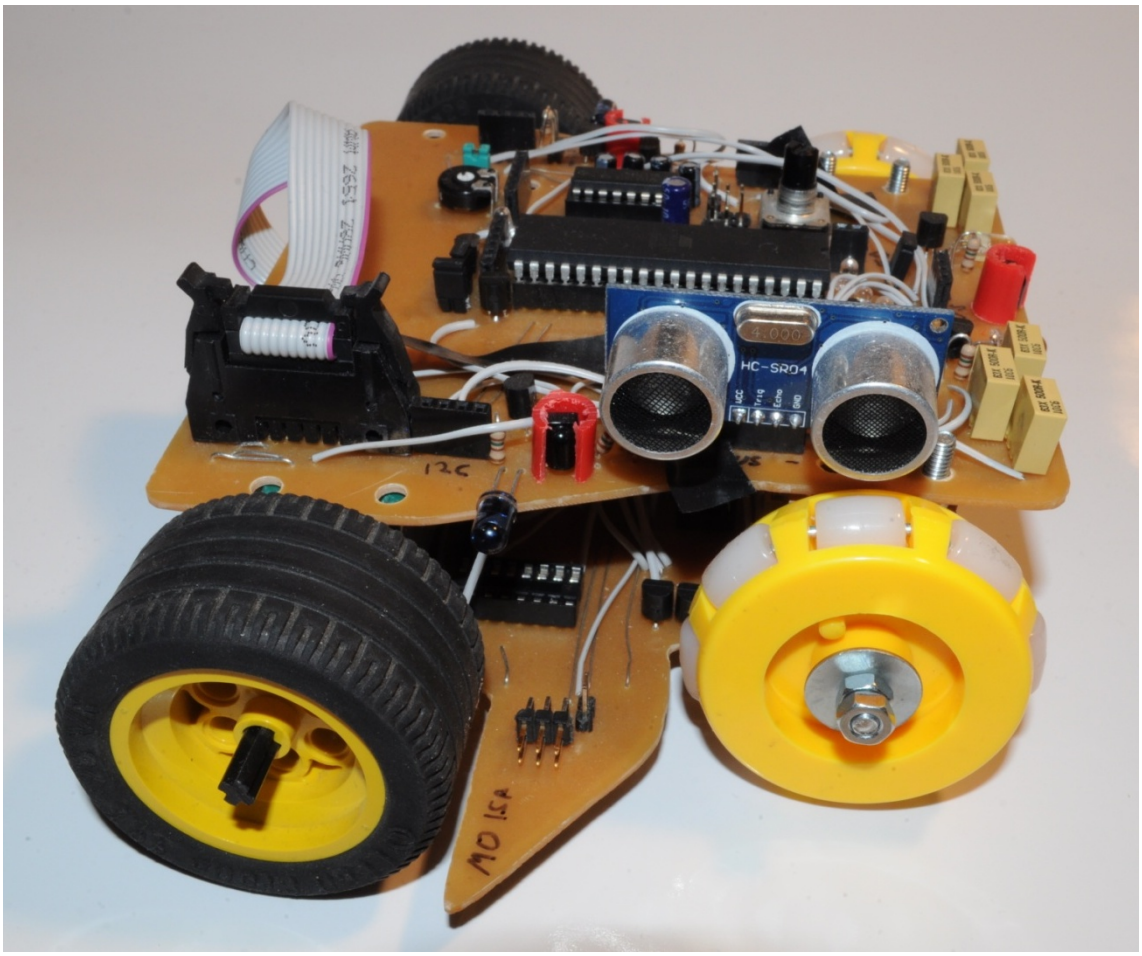
A. Großaufnahmen des Roboters
B. Schaltpläne
C. Quellcode

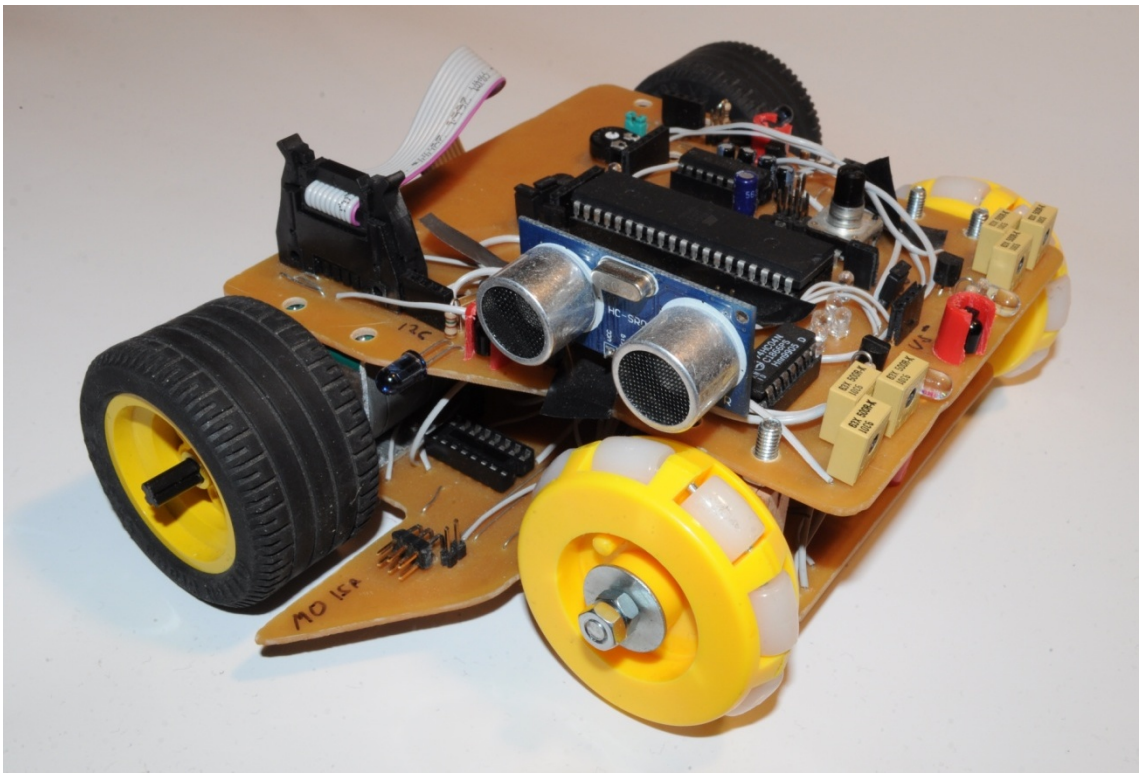
a) Großaufnahmen des Roboters

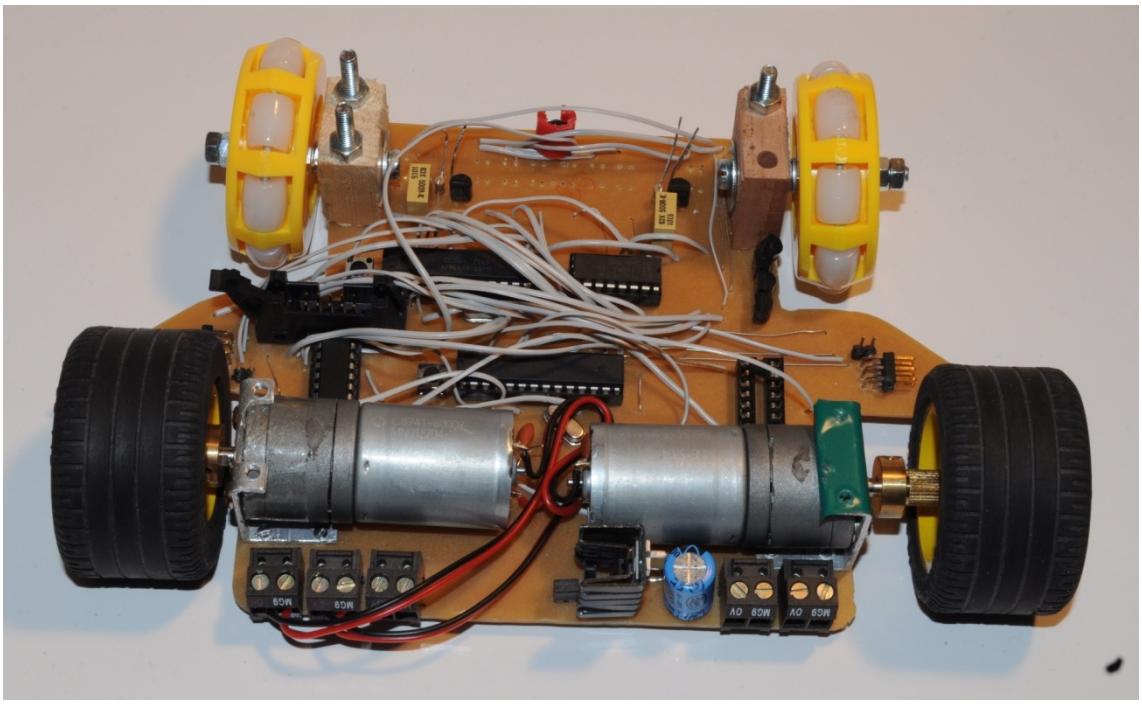




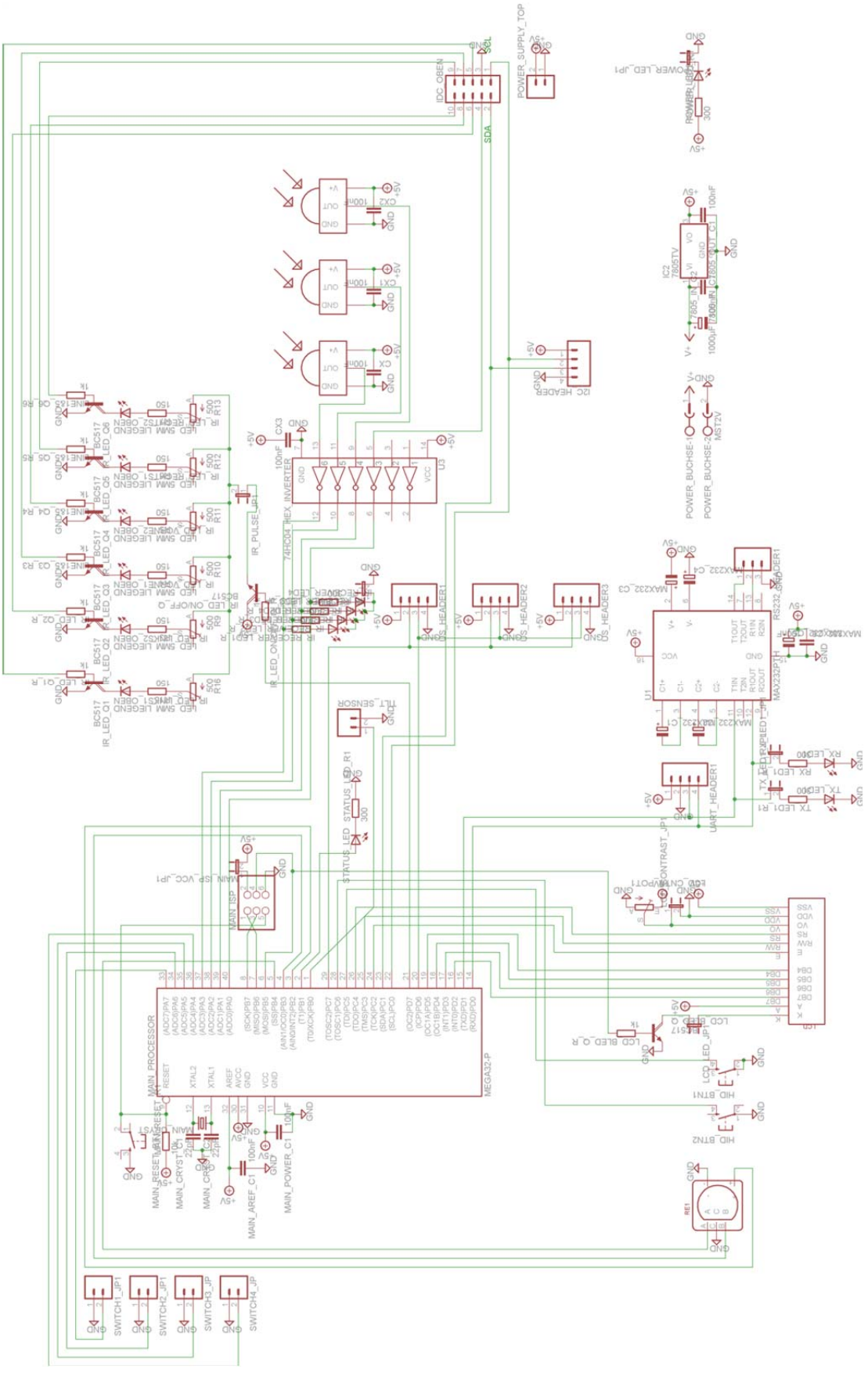



HC-SR04

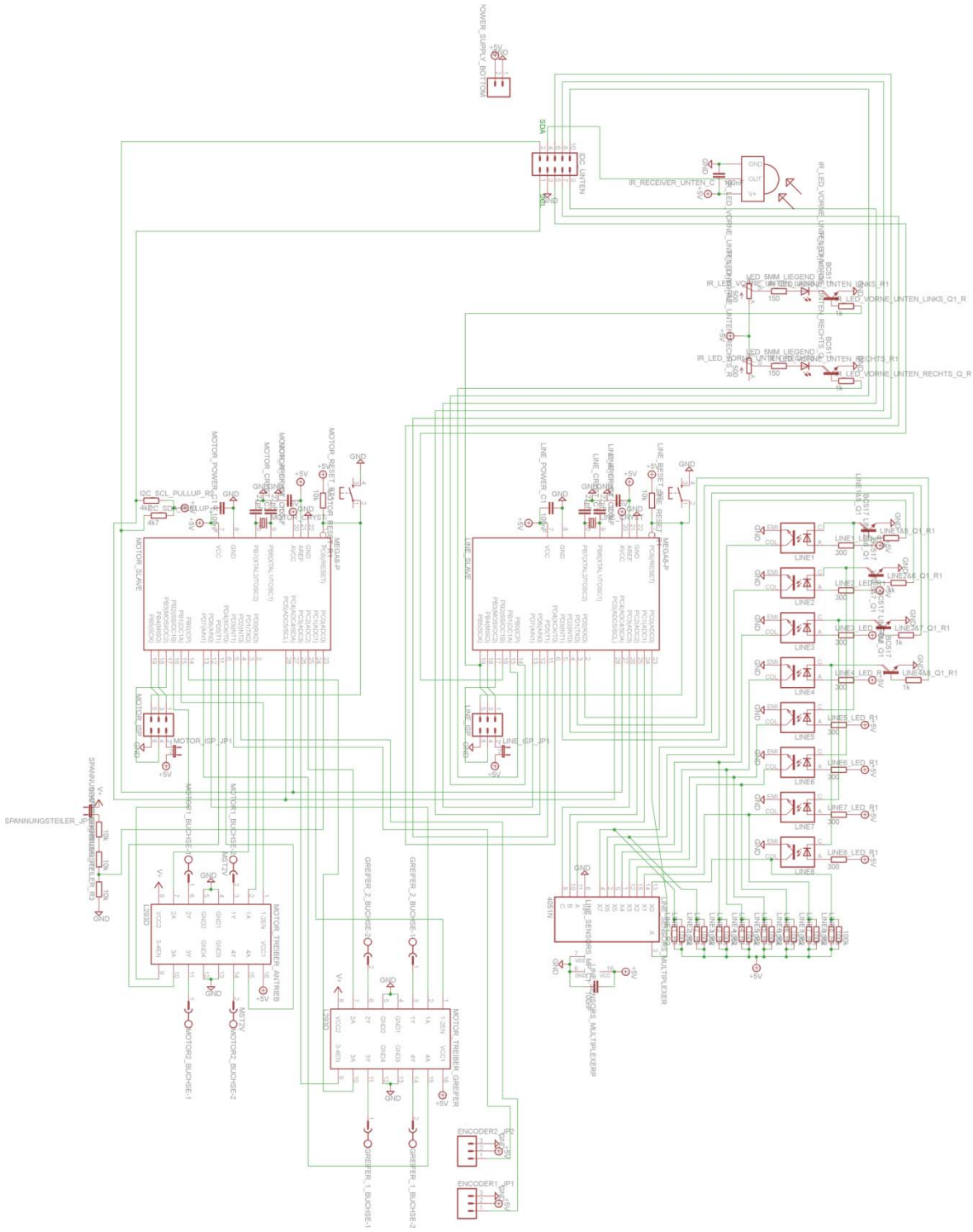




b) Schaltpläne



A-1 - Schaltplan der oberen Platine



A-2 - Schaltplan der unteren Platine

c) Quellcode

main/main.c

```
/*INCLUDES*/

//Standard C-Libs
#include <stdlib.h>


//Standard AVR-Libs
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>


//Libs
#include "main_taster.h"
#include "main_uart.h"
#include "i2cmaster.h"
#include "main_lcd.h"

#include "main_header.h"
#include "main_lib.h"
#include "menu_terminal.h"



int16_t motor1_speed = 0;
int16_t motor2_speed = 0;

volatile uint16_t timestamp_last=0;
volatile uint16_t runtime=0;



//called every 1,024ms
ISR(TIMER0_OVF_vect)
{
   static uint8_t millisec_count;


   millisec_count++;

   //led toggle
   if((millisec_count % 100) == 0)
   {
     //switch LED on/off
       status_led(flags.led_state);
       //invert LED state
       if(flags.led_state)
         flags.led_state = 0;
       else
         flags.led_state = 1;
   }

   //Button debounce
   if((millisec_count % 10) == 0)
   {
```

```
      get_taster (BUTTON_1, (PINC & (1<<PIN_BUTTON1)));
      get_taster (BUTTON_2, (PINC & (1<<PIN_BUTTON2)));
        get_taster (BUTTON_ENCODER, (PINB & (1<<PIN_ENCODER_BUTTON)));
    }

    //Encoder
    if((millisec_count % 2) == 0)
    {
      int8_t new, diff;
      new = 0;
      if( PHASE_A )
        new = 3;
      if( PHASE_B )
        new ^= 1;                    // convert gray to binary
      diff = last - new;                 // difference last - new
      if( diff & 1 )  // bit 0 = value (1)
      {
        last = new;                  // store new as next last
        enc_delta += (diff & 2) - 1;        // bit 1 = direction (+/-)
      }
    }

    if((millisec_count % 50) == 0)
    {
      trig_us();
    }

}

/*US Messung*/
ISR(TIMER1_CAPT_vect)
{
  //Wenn steigende Flanke
  if(TCCR1B & (1<<ICES1))
  {
    //Flankenerkennung auf fallend
      TCCR1B ^= (1<<ICES1);
      //Aktuellen timer-wert speichern
      timestamp_last = ICR1;
  }
  //fallende Flanke
  else
  {
    //Flankenerkennung auf steigend
      TCCR1B ^= (1<<ICES1);
      //Laufzeit = aktueller timerwert - vorheriger timerwert
      runtime = ICR1 - timestamp_last;
      flags.us_state = READY;
  }

}



int main(void)
{


  int16_t encoder_rotation = 0;
  int16_t encoder_rotation_old = 0;

  uint8_t line_position = 0;
  uint16_t line_estimate = 0;
```

```
  uint32_t wa_numerator = 0;
  uint16_t wa_denominator = 0;

  uint16_t line_values[8];

  //PID stuff
  //int16_t proportional = 0;
  int16_t last_proportional = 0;
  int16_t integral = 0;
  //int16_t derivative = 0;
  int16_t error_value = 0;
  uint16_t Kp = 0;
  uint16_t Kd = 0;
  uint16_t Ki = 0;

  //Menu
  uint16_t menu=0;


  for(uint8_t i=0;i<8;i++)
           line_values[i] = 0;

  io_init();


  lcd_init(LCD_DISP_ON);

  lcd_bl(1);


  /*TIMER CONFIG*/

  /*System Tick*/
  // Prescaler auf 64
  TCCR0|=(1<<CS01)|(1<<CS00);

  //Overflow Interrupt aktivieren
  TIMSK|=(1<<TOIE0);

  /*US Timer*/
  //Counter initialisieren
  TCNT0=0;

  //Timer konfigurieren
  TCCR1A = 0;                       // normal mode, keine PWM Ausgänge
  //Noise Canceler aktivieren, Flankenerkennung auf steigende,
Prescaler auf 64
  TCCR1B |= (1<<ICNC1) |       (1<<ICES1) |
(1<<CS11) |(1<<CS10);

  //ICP Interrupt aktivieren
  TIMSK |= (1<<TICIE1);

  /*IR Timer*/
  OCR2 = 240;

  TCCR2 |= (1<<CS20) | (1<<WGM21) | (1<<COM20);

  uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );

  _delay_ms(1000);  //wait for slaves to initialize properly

  i2c_init();         // init I2C interface
```

```
  encode_init();

  tasten[0].mode = TM_LONG;
  tasten[1].mode = TM_LONG;
  tasten[2].mode = TM_LONG;

  sei();


  send_line_mode(10);  //start sampling


  while(1)
  {

      /*read encoder and display if changed*/
      encoder_rotation = 0;
      encoder_rotation += encode_read4() ;
      if(encoder_rotation != encoder_rotation_old)  //has changed
since last time
      {
            switch (menu) {
                  case 2:
                        Kp += encoder_rotation;
                        break;
                  case 3:
                        Kd += encoder_rotation;
                        break;
                  case 4:
                        Ki += encoder_rotation;
                        break;
                  case 7:
                        motor1_speed += (encoder_rotation * 10);
                        motor2_speed = motor1_speed;
                        break;
                  default:
                        break;
            }
      }
      encoder_rotation_old = encoder_rotation;
      //encoder.end


      //check whether new us time is available for calculation
      if(flags.us_state == READY)
      {
        //calculate distance from travel time
        //*4 => scale to milliseconds
        //runtime = (runtime*4)/58;
        runtime /= 15;
        flags.us_state = NOT_READY;
      }

      //Read line sensors values from slave
      //binary line detection
      for(uint8_t i=0;i<NUM_SENSORS;i++)
      {
            line_values[i] = read_line_sensor(i+1);

            if(line_values[i]>LINE_TRESHOLD)
            {
                  line_position |= (1<<(7-i));
```

```
                  //values for line estimation via weighted mean
                  //only values above threshold (e.g. black) are
considered in calculation
                  wa_numerator += (long)(line_values[i]) *((i+1) *100);
                  wa_denominator += line_values[i];
            }
            else  {
                  line_position &= ~(1<<(7-i));
            }
      }

      //prevent division through zero
      if(line_position==0)  {
            line_estimate = 0;
      }
      else  {
            line_estimate = wa_numerator/wa_denominator;
      }

      //reset values
      wa_numerator = 0;
      wa_denominator = 0;

      /*
      for(uint8_t i=0;i<NUM_SENSORS;i++)
      {
            if(line_position & (1<<(7-i)))  {
                  lcd_putc('1');
            }
            else  {
                  lcd_putc('0');
            }
      }*/


      //PID
      int16_t proportional = ((int)line_estimate) - SET_POINT;
//should be 0 when over line
      int16_t derivative = proportional - last_proportional;
      integral += proportional;
      last_proportional = proportional;
      //Difference between motor speeds. if positive -> turn right, if
negative turn left
      error_value = proportional / Kp + integral / Ki + derivative *
Kd;

      //drive motors

      const int max = 600;
      if(error_value > max)  {
            error_value = max;
      }
      if(error_value < -max)  {
            error_value = -max;
      }

      if(error_value < 0)  {
            motor1_speed = max + error_value;
            motor2_speed = max;
      }
      else  {
            motor1_speed = max;
            motor2_speed = max - error_value;
```

```
}


signed char tast = taster;

switch(tast)
{
  case NO_TASTER:
      break;

  case BUTTON_1:
        menu++;
      break;

  case BUTTON_1 + TASTER_LONG:
      break;

  case BUTTON_2:
        if(menu)
              menu--;
      break;

  case BUTTON_2 + TASTER_LONG:
      break;

  case BUTTON_ENCODER:
        if(menu==5)  {
            if (!flags.run_start) {
                  flags.run_start = 1;
            }
            else  {
                  flags.run_start = 0;
            }
        }
        if(menu==7)  {
              if (flags.motor_start) {
                    flags.motor_start = 0;
              }
              else  {
                    flags.motor_start = 1;
              }
        }
      break;

  case BUTTON_ENCODER + TASTER_LONG:
      break;
}
if (tast != NO_TASTER)
      taster = NO_TASTER;

switch (menu) {
      case 0:
            lcd_clrscr();
            lcd_gotoxy(0,LCD_LINE1);
            lcd_puts_P("Alexander Kargl");
            lcd_gotoxy(0,LCD_LINE2);
            lcd_puts_P("Btn1++  Btn2--");
            break;
      case 1:
            lcd_clrscr();
            lcd_gotoxy(0,LCD_LINE1);
            lcd_puts_P("Line Position");
            lcd_gotoxy(0,LCD_LINE2);
```

```
		lcd_put_uint16(line_estimate);
		break;
	case 2:
		lcd_clrscr();
		lcd_gotoxy(0,LCD_LINE1);
		lcd_puts_P("Kp");
		lcd_gotoxy(0,LCD_LINE2);
		lcd_put_uint16(Kp);
		break;
	case 3:
		lcd_clrscr();
		lcd_gotoxy(0,LCD_LINE1);
		lcd_puts_P("Kd");
		lcd_gotoxy(0,LCD_LINE2);
		lcd_put_uint16(Kd);
		break;
	case 4:
		lcd_clrscr();
		lcd_gotoxy(0,LCD_LINE1);
		lcd_puts_P("Ki");
		lcd_gotoxy(0,LCD_LINE2);
		lcd_put_uint16(Ki);
		break;
	case 5:
		lcd_clrscr();
		lcd_gotoxy(0,LCD_LINE1);

		if (!flags.run_start) {
			lcd_puts_P("Start Program");
		}
		else  {
			lcd_puts_P("Stop Program");
		}
		lcd_gotoxy(0,LCD_LINE2);
		lcd_put_uint8(flags.run_start);
		break;
	case 6:
		lcd_clrscr();
		lcd_gotoxy(0,LCD_LINE1);
		lcd_puts_P("P");
		lcd_put_int16(proportional);
		lcd_gotoxy(9,LCD_LINE1);
		lcd_puts_P("D");
		lcd_put_int16(derivative);
		lcd_gotoxy(0,LCD_LINE2);
		lcd_puts_P("I");
		lcd_put_int16(integral);
		lcd_gotoxy(9,LCD_LINE2);
		lcd_puts_P("E");
		lcd_put_int16(error_value);
		break;
	case 7:
		lcd_clrscr();
		lcd_gotoxy(0,LCD_LINE1);
		lcd_puts_P("M1 ");
		lcd_put_int16(motor1_speed);
		lcd_gotoxy(15,LCD_LINE1);
		lcd_put_uint8(flags.run_start);
		lcd_gotoxy(0,LCD_LINE2);
		lcd_puts_P("M2 ");
		lcd_put_int16(motor2_speed);
		break;
	default:
```

```
                break;
        }


        if(flags.motor_start)
        {
                send_motor1_speed(motor1_speed);
                send_motor2_speed(motor2_speed);
        }
        else
        {
                send_motor1_speed(0);
                send_motor2_speed(0);
        }


    }//while.end

   return 0;
}//main.end
```

main/main_header.h

```
#ifndef MAIN_HEADER_H
#define MAIN_HEADER_H

/**********************
FILE: main_header.h
DATE: 2.11.2011
**********************/

/* DEFINES */
#define NUM_SENSORS          8

//LCD
#define LCD_LINE1 0
#define LCD_LINE2 1

//line detection
#define LINE_TRESHOLD 150
#define SET_POINT 450

#define UART_BAUD_RATE      9600
//i2c adresses
#define MOTOR_SLAVE_ADRESSE 0x50
#define LINE_SLAVE_ADRESSE 0x40

//motor float
#define FLOAT 2000

//encoders
#define PHASE_A     (PINB & 1<<PIN_ENCODER_A) //Encoder
#define PHASE_B     (PINB & 1<<PIN_ENCODER_B)

//buttons
#define BUTTON_1        0
#define BUTTON_2        1
#define BUTTON_ENCODER  2

#define READY           1
#define NOT_READY 0

/*IR Leds*/
#define IR_LED_UNTEN_VORNE1   1
#define IR_LED_UNTEN_VORNE2   2
#define IR_LED_OBEN_LINKS2          3
#define IR_LED_OBEN_LINKS1          4
#define IR_LED_OBEN_VORNE1          5
#define IR_LED_OBEN_VORNE2          6
#define IR_LED_OBEN_RECHTS1         7
#define IR_LED_OBEN_RECHTS2         8

#define IR_LED_ALL                        0
#define IR_LED_UNTEN_VORNE_ALL      12
#define IR_LED_OBEN_VORNE_ALL       56
#define IR_LED_OBEN_LINKS_ALL       34
#define IR_LED_OBEN_RECHTS_ALL      78

/*MACROS*/
#define uniq(LOW,HEIGHT)      ((HEIGHT << 8)|LOW)                 // 2x
8Bit  --> 16Bit
#define LOW_BYTE(x)           (x & 0xff)                          //
16Bit       --> 8Bit
```

```
#define HIGH_BYTE(x)          ((x >> 8) & 0xff)            // 16Bit
      --> 8Bit



/*Global Variables*/
//Flags
struct {
   unsigned led_state:1;
   volatile unsigned us_state:1;
   unsigned run_start:1;
   unsigned motor_start:1;
} flags;




//Encoder
volatile int8_t enc_delta;          // -128 ... 127
static int8_t last;




/* PINBELEGUNG MAIN

PORTA
  PA0 - IR Receiver vorne unten
  PA1 - IR Receiver links oben
  PA2 - IR Receiver vorne oben
  PA3 - IR Receiver rechts oben
  PA4 - Switch 4
  PA5 - Switch 3
  PA6 - Switch 2
  PA7 - Switch 1

PORTB
  PB0 - Tilt Sensor
  PB1 - Status Led
  PB2 - Encoder Button
  PB3 - Encoder B
  PB4 - Encoder A
  PB5 - LCD Backlight / ISP MOSI
  PB6 - ISP MISO
  PB7 - ISP SCK

PORTC
  PC0 - I2C SCL
  PC1 - I2C SDA
  PC2 - LCD Enable
  PC3 - LCD R/W
  PC4 - LCD RS
  PC5 - Button 1
  PC6 - Button 2
  PC7 - US Trigger

PORTD
  PD0 - UART RX
  PD1 - UART TX
  PD2 - LCD DB7
  PD3 - LCD DB6
```

```
  PD4 - LCD DB5
  PD5 - LCD DB4
  PD6 - US Echo
  PD7 - IR Pulse

*/



/*Ports*/
#define PORT_IR_RECEIVER  PORTA
#define PORT_SWITCH       PORTA

#define PORT_TILT_SENSOR  PORTB
#define PORT_STATUS_LED   PORTB
#define PORT_ENCODER      PORTB
#define PORT_LCD_BL       PORTB

#define PORT_LCD_E        PORTC
#define PORT_LCD_RW       PORTC
#define PORT_LCD_RS       PORTC
#define PORT_BUTTON       PORTC
#define PORT_US_TRIGGER   PORTC

#define PORT_LCD_DATA     PORTD
#define PORT_US_ECHO      PORTD
#define PORT_IR_PULSE     PORTD

/*DDRx*/

#define DDR_IR_RECEIVER  DDRA
#define DDR_SWITCH       DDRA

#define DDR_TILT_SENSOR  DDRB
#define DDR_STATUS_LED   DDRB
#define DDR_ENCODER      DDRB
#define DDR_LCD_BL       DDRB

#define DDR_LCD_E        DDRC
#define DDR_LCD_RW       DDRC
#define DDR_LCD_RS       DDRC
#define DDR_BUTTON       DDRC
#define DDR_US_TRIGGER   DDRC

#define DDR_LCD_DATA     DDRD
#define DDR_US_ECHO      DDRD
#define DDR_IR_PULSE     DDRD

/*Pins*/
#define PIN_IR_RECEIVER_VORNE_UNTEN PA0
#define PIN_IR_RECEIVER_LINKS_OBEN  PA1
#define PIN_IR_RECEIVER_VORNE_OBEN  PA2
#define PIN_IR_RECEIVER_RECHTS_OBEN PA3
#define PIN_SWITCH4                 PA4
#define PIN_SWITCH3                 PA5
#define PIN_SWITCH2                 PA6
#define PIN_SWITCH1                 PA7

#define PIN_TILT_SENSOR             PB0
#define PIN_STATUS_LED              PB1
#define PIN_ENCODER_BUTTON          PB2
#define PIN_ENCODER_B               PB3
#define PIN_ENCODER_A               PB4
```

```
#define PIN_LCD_BL                  PB5

#define PIN_LCD_E                           PC2
#define PIN_LCD_RW                          PC3
#define PIN_LCD_RS                          PC4
#define PIN_BUTTON1                         PC5
#define PIN_BUTTON2                 PC6
#define PIN_US_TRIGGER              PC7

#define PIN_LCD_DB7                         PD2
#define PIN_LCD_DB6                         PD3
#define PIN_LCD_DB5                         PD4
#define PIN_LCD_DB4                         PD5
#define PIN_US_ECHO                         PD6
#define PIN_IR_PULSE                PD7


#endif //MAIN_HEADER_H
```

main/main_lib.c

```
#ifndef MAIN_HEADER_C
#define MAIN_HEADER_C

#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "main_uart.h"
#include "main_lcd.h"
#include "i2cmaster.h"
#include "main_header.h"


void io_init (void)
{
  /* Ein-/Ausgänge festlegen*/
  DDR_LCD_BL |= (1<< PIN_LCD_BL);
  DDR_STATUS_LED |= (1<<PIN_STATUS_LED);
  DDR_US_TRIGGER |= (1<<PIN_US_TRIGGER);
  DDR_IR_PULSE |= (1<<PIN_IR_PULSE);

  DDR_BUTTON &= ~(1<<PIN_BUTTON1);
  DDR_BUTTON &= ~(1<<PIN_BUTTON2);

  DDR_ENCODER &= ~(1<<PIN_ENCODER_BUTTON);
  DDR_ENCODER &= ~(1<<PIN_ENCODER_A);
  DDR_ENCODER &= ~(1<<PIN_ENCODER_B);

  /*Ausgänge ein-/auschalten*/
  PORT_LCD_BL &= ~(1<< PIN_LCD_BL);
  PORT_STATUS_LED &= ~(1<<PIN_STATUS_LED);
  PORT_US_TRIGGER &= ~(1<<PIN_US_TRIGGER);
  PORT_IR_PULSE &= ~(1<<PIN_IR_PULSE);

  /*Pullups ein-/auschalten*/
  PORT_BUTTON |= (1<<PIN_BUTTON1);
  PORT_BUTTON |= (1<<PIN_BUTTON2);

  PORT_ENCODER |= (1<<PIN_ENCODER_BUTTON);
  PORT_ENCODER |= (1<<PIN_ENCODER_A);
  PORT_ENCODER |= (1<<PIN_ENCODER_B);
}


//LCD Hintergrundbeleuchtung an/aus
//1=an, 0=aus
void lcd_bl (uint8_t state)
{
  if(state)
    PORT_LCD_BL |= (1<<PIN_LCD_BL); //LCD LED an
  else
    PORT_LCD_BL &= ~(1<<PIN_LCD_BL); //LCD LED aus
}


//Status LED an/aus
//1=an, 0=aus
void status_led (uint8_t state)
{
```

```
  if(state==1)
    PORT_STATUS_LED |= (1<<PIN_STATUS_LED); //StatusLED an
  else if (state==0)
    PORT_STATUS_LED &= ~(1<<PIN_STATUS_LED); //Status LED aus
}

//IR Entfernung ein/aus
void ir_pulse(uint8_t state)
{
  if(state==1)
    DDR_IR_PULSE |= (1<<PIN_IR_PULSE); //StatusLED an
  else if (state==0)
    DDR_IR_PULSE &= ~(1<<PIN_IR_PULSE); //Status LED aus
}

//start US-measurement
void trig_us(void)
{
  PORT_US_TRIGGER |= (1<<PIN_US_TRIGGER);//Trig high
  _delay_us(12);
  PORT_US_TRIGGER &= ~(1<<PIN_US_TRIGGER);//TRIG auf low
}


/*MOTOR SLAVE*/

//Sends motor speeds for both motors to the motor-slave
uint8_t send_motor12_speed(int16_t motor1_speed, int16_t motor2_speed)
{
  //Slave ready?
  if(!(i2c_start(MOTOR_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(0);  //buffer startadress
        i2c_write( LOW_BYTE(motor1_speed) );
        i2c_write( HIGH_BYTE(motor1_speed) );
        i2c_write( LOW_BYTE(motor2_speed) );
        i2c_write( HIGH_BYTE(motor2_speed) );
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}

//Sends motor speeds for motor 1 to the motor-slave
uint8_t send_motor1_speed(int16_t motor1_speed)
{
  //Slave ready?
  if(!(i2c_start(MOTOR_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(0);  //buffer startadress
        i2c_write( LOW_BYTE(motor1_speed) );
        i2c_write( HIGH_BYTE(motor1_speed) );
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}

//Sends motor speeds for motor 2 to the motor-slave
```

```
uint8_t send_motor2_speed(int16_t motor2_speed)
{
  //Slave ready?
  if(!(i2c_start(MOTOR_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(2);  //buffer startadress
        i2c_write( LOW_BYTE(motor2_speed) );
        i2c_write( HIGH_BYTE(motor2_speed) );
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}


//Send motor1 float command
uint8_t send_motor1_float(void)
{
  //Slave ready?
  if(!(i2c_start(MOTOR_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(0);  //buffer startadress
        i2c_write( LOW_BYTE(FLOAT) );
        i2c_write( HIGH_BYTE(FLOAT) );
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}


//Send motor2 float command
uint8_t send_motor2_float(void)
{
  //Slave ready?
  if(!(i2c_start(MOTOR_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(2);  //buffer startadress
        i2c_write( LOW_BYTE(FLOAT) );
        i2c_write( HIGH_BYTE(FLOAT) );
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}

/*Line Slave*/

//Set line sensor sampling mode
//mode: 0 stop sampling; 1-8 sample sensor; 10 sample continuesly
uint8_t send_line_mode(uint8_t mode)
{
  //Slave ready?
  if(!(i2c_start(LINE_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(16);  //buffer startadress
        i2c_write(mode);
```

```
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}

//get value of specified sensor
uint16_t read_line_sensor(uint8_t sensor)
{
  uint8_t values[2];
  values[0] = 0;
  values[1] = 0;

  if(sensor>8) return 0;

  if(!(i2c_start(LINE_SLAVE_ADRESSE+I2C_WRITE))) //Slave bereit zum
lesen?
  {
    i2c_write((sensor*2)-2); //Buffer Startadresse zum Auslesen
    i2c_rep_start(LINE_SLAVE_ADRESSE+I2C_READ); //Lesen beginnen
    values[0] = i2c_readAck(); // Bytes lesen...
    values[1] = i2c_readNak(); // letztes Byte lesen, darum kein ACK
    i2c_stop();           // Zugriff beenden
  }

  return uniq(values[0],values[1]);
}

/* 0    =>no calibration
   1    => start calibration; look for highest and lowest values until
2
   2    => stop fetching values and save max. and min. in eeprom; set
to 0 when finished
*/
uint8_t send_calibration_mode(uint8_t mode)
{
  //Slave ready?
  if(!(i2c_start(LINE_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(19);  //buffer startadress
        i2c_write(mode);
        i2c_stop();       //stop

        return 0;
  }
  else
    return 1;
}


//switch ir led on off
//modes defined in line_header.h
uint8_t send_ir_led(uint8_t led, uint8_t state)
{
  if(state>1) return 2; //state must be 1 or 0

  if(!(i2c_start(LINE_SLAVE_ADRESSE+I2C_WRITE)))
  {
        i2c_write(17);  //buffer startadress
        i2c_write(led);
        i2c_write(state);
```

```
        i2c_stop();       //stop

        return 0; //transmission sucessful
  }
  else
    return 1;  //slave not ready/responding
}



/*int to string*/

void uart_put_int(int8_t var)
{
  char buffer[20];
  uart_puts(itoa(var, buffer, 10));
}

void uart_put_uint(uint8_t var)
{
  char buffer[20];
  uart_puts(utoa(var, buffer, 10));
}

void uart_put_u16bit( uint16_t value )
{
    unsigned char digit;

    digit = '0';

    while( value >= 10000 )                // Still larger than 1000 ?
        {
            digit++;                          // Increment first digit
            value -= 10000;
        }
      uart_putc( digit );

    while( value >= 1000 )                 // Still larger than 100 ?
    {
        digit++;                           // Increment first digit
        value -= 1000;
    }
      uart_putc( digit );

      digit = '0';
    while( value >= 100 )                  // Still larger than 100 ?
    {
        digit++;                           // Increment first digit
        value -= 100;
    }

   uart_putc( digit );                    // Send first digit

    digit = '0';
    while( value >= 10 )                   // Still larger than 10 ?
    {
        digit++;                           // Increment second digit
        value -= 10;
    }

    uart_putc( digit );                    // Send second digit

    uart_putc( '0' + value );              // Send third digit
```

```
}

void uart_put_16bit( int16_t value )
{
    unsigned char digit;

      if(value<0)
      {
        uart_putc('-');
        value *= -1;
      }

      digit = '0';
    while( value >= 1000 )                 // Still larger than 100 ?
    {
        digit++;                           // Increment first digit
        value -= 1000;
    }
      uart_putc( digit );

      digit = '0';
    while( value >= 100 )                  // Still larger than 100 ?
    {
        digit++;                           // Increment first digit
        value -= 100;
    }

   uart_putc( digit );                    // Send first digit

    digit = '0';
    while( value >= 10 )                   // Still larger than 10 ?
    {
        digit++;                           // Increment second digit
        value -= 10;
    }

    uart_putc( digit );                    // Send second digit

    uart_putc( '0' + value );              // Send third digit
}

/*LCD*/

void lcd_put_int8(int8_t var)
{
  char buffer[20];
  itoa(var, buffer, 10);
  lcd_puts(buffer);
}

void lcd_put_uint8(uint8_t var)
{
  char buffer[20];
  utoa(var, buffer, 10);
  lcd_puts(buffer);
}

void lcd_put_int16(int16_t var)
{
  char buffer[20];
  itoa(var, buffer, 10);
  lcd_puts(buffer);
}
```

```
void lcd_put_uint16(uint16_t var)
{
  char buffer[30];
  utoa(var, buffer, 10);
  lcd_puts(buffer);
}


/*ENCODER*/

void encode_init( void )
{
  int8_t new;

  new = 0;
  if( PHASE_A )
    new = 3;
  if( PHASE_B )
    new ^= 1;                    // convert gray to binary
  last = new;                    // power on state
  enc_delta = 0;

}

int8_t encode_read4( void )         // read four step encoders
{
  int8_t val;

  cli();
  val = enc_delta;
  enc_delta = val & 3;
  sei();
  return val >> 2;
}

#endif //MAIN_HEADER_C
```

main/main_lib.h

```
#ifndef MAIN_LIB_H
#define MAIN_LIB_H

void io_init (void);

void lcd_bl (uint8_t state);

void status_led (uint8_t state);

void trig_us(void);

/*IR Sensors*/
void ir_pulse(uint8_t state);
uint8_t send_ir_led(uint8_t led, uint8_t state);

/*Line Slave*/
uint8_t send_line_mode(uint8_t mode);
uint16_t read_line_sensor(uint8_t sensor);
uint8_t send_calibration_mode(uint8_t mode);

/*MOTOR CONTROL*/
uint8_t send_motor12_speed(int16_t motor1_speed, int16_t
motor2_speed);
uint8_t send_motor1_speed(int16_t motor1_speed);
uint8_t send_motor2_speed(int16_t motor2_speed);

uint8_t send_motor1_float(void);
uint8_t send_motor2_float(void);

/*INT TO STRING*/
void uart_put_int(int8_t var);
void uart_put_uint(uint8_t var);
void uart_put_u16bit( uint16_t value );
void uart_put_16bit( int16_t value );

/*LCD*/
//Displays a variable at cursor position
void lcd_put_int8(int8_t var);
void lcd_put_uint8(uint8_t var);
void lcd_put_int16(int16_t var);
void lcd_put_uint16(uint16_t var);


/*ENCODER*/
void encode_init(void);
int8_t encode_read4(void);

#endif //MAIN_LIB_H
```

line/main.c

```
/*INCLUDES*/

//Standard C-Libs
#include <stdlib.h>
#include <stdint.h>

//Standard AVR-Libs
#include <util/twi.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <avr/eeprom.h>


//Libs
#include "line_header.h"
#include "line_lib.h"
#include "twislave.h"



int main(void)
{
	/*Variables*/
	uint16_t ambient_light = 0, light_value = 0;

	struct light_sensor
	{
		uint16_t cal_min;
		uint16_t cal_max;
	} light_sensors[NUM_SENSORS];


	init_io();  //Set up IO-Ports
	init_adc(); //Set up ADC
	init_twi_slave(SLAVE_ADRESSE);	//I2C-Slave

	set_line_led(LINE_LED_ALL,OFF);

	for(uint8_t i=0;i<buffer_size;i++)
	{
		//fill up I2C-Buffers
		rxbuffer[i]=0;
		txbuffer[i]=0;
	}

	for(uint8_t i=0;i<NUM_SENSORS;i++)
	{
		light_sensors[i].cal_min = 1023;
		light_sensors[i].cal_max = 0;
	}

	sei();  //activate global interrupts

	//read calibration min max from eeprom (TODO: add some sanity
check whether values are really saved and haven't been
erased/corrupted )

	for(uint8_t i=0;i<NUM_SENSORS;i++)
```

```
	{
		light_sensors[i].cal_min = eeprom_read_word ((uint16_t *)
(2*i ));
	}

	for(uint8_t i=0;i<NUM_SENSORS;i++)
	{
		light_sensors[i].cal_max = eeprom_read_word ((uint16_t *) (
16 + 2*i));
	}


	while(1)
	{
		/*Calibration*/
		while(rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]==1)
//start looking for calibration values
		{
			for(uint8_t i=0;i<NUM_SENSORS;i++)  //get max. and
min. values for each sensor
			{
				set_line_mux_channel(i);
				//_delay_ms(10);  //allow multiplexer enough
time to change channel, can probably be reduced/omitted

				/*switch on respective leds*/
				if((i==0)||(i==4))
					set_line_led(LINE_LED_15,ON);

				if((i==1)||(i==5))
					set_line_led(LINE_LED_26,ON);

				if((i==2)||(i==6))
					set_line_led(LINE_LED_37,ON);

				if((i==3)||(i==7))
					set_line_led(LINE_LED_48,ON);

				//look for max. and min. value and store
				light_value = read_adc_avg(0,10);

				if(light_value > light_sensors[i].cal_max)
					light_sensors[i].cal_max = light_value;

				if(light_value < light_sensors[i].cal_min)
					light_sensors[i].cal_min = light_value;

				set_line_led(LINE_LED_ALL,OFF);
			}
		}

		if(rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]==2)  //save
calibration values to eeprom
		{
			for(uint8_t i=0;i<NUM_SENSORS;i++)
			{
				eeprom_write_word ((uint16_t *) (2*i ),
light_sensors[i].cal_min);
			}

			for(uint8_t i=0;i<NUM_SENSORS;i++)
			{
```

```
                        eeprom_write_word ((uint16_t *) ( 16 + 2*i),
light_sensors[i].cal_max);
                  }
                  rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]=0;
            }

            if(rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]==3)  //prepare
min. calibration values for transmission
            {
                  //fill txbuffer with min. calibration values
                  for(uint8_t i=0;i<NUM_SENSORS;i++)
                  {
                        txbuffer[2*i] =
LOW_BYTE(light_sensors[i].cal_min);
                        txbuffer[2*i+1] =
HIGH_BYTE(light_sensors[i].cal_min);
                  }
                  rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]=0;
            }

            if(rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]==4)  //prepare
max. calibration values for transmission
            {
                  //fill txbuffer with max. calibration values
                  for(uint8_t i=0;i<NUM_SENSORS;i++)
                  {
                        txbuffer[2*i] =
LOW_BYTE(light_sensors[i].cal_max);
                        txbuffer[2*i+1] =
HIGH_BYTE(light_sensors[i].cal_max);
                  }
                  rxbuffer[LINE_SENSOR_CALIBRATION_COMMAND]=0;
            }

            /*Calibration END*/

            /*Sampling and processing*/

            if(rxbuffer[LINE_SENSOR_COMMAND]!=0) //0 -> no sampling
            {
                  for(uint8_t i=0; i<NUM_SENSORS; i++) //sweep through
sensors
                  {
                        set_line_mux_channel(i);
                        _delay_ms(10); //TODO: reduce/omit delay

                        if(rxbuffer[LINE_SENSOR_COMMAND]==10) //no
calibration
                        {
                              /*switch on respective leds*/
                              if((i==0)||(i==4))
                                    set_line_led(LINE_LED_15,ON);

                              if((i==1)||(i==5))
                                    set_line_led(LINE_LED_26,ON);

                              if((i==2)||(i==6))
                                    set_line_led(LINE_LED_37,ON);

                              if((i==3)||(i==7))
                                    set_line_led(LINE_LED_48,ON);

                              light_value = read_adc_avg(0,10);
```

```
                        }
                        else if(rxbuffer[LINE_SENSOR_COMMAND]==11)
//sample with ambient light filtering
                        {
                              ambient_light = read_adc_avg(0,10);
//take measurement without led activated -> get ambient light

                              /*switch on respective leds*/
                              if((i==0)||(i==4))
                                    set_line_led(LINE_LED_15,ON);

                              if((i==1)||(i==5))
                                    set_line_led(LINE_LED_26,ON);

                              if((i==2)||(i==6))
                                    set_line_led(LINE_LED_37,ON);

                              if((i==3)||(i==7))
                                    set_line_led(LINE_LED_48,ON);

                              //_delay_ms(10);

                              light_value = read_adc_avg(0,10);

                              if(ambient_light < light_value)
                                    light_value=0;  //prevent underflow
                              else
                                    light_value = ambient_light -
light_value;  //eliminate ambient light
                        }
                        else if(rxbuffer[LINE_SENSOR_COMMAND]==12)
//normalize values
                        {
                              /*switch on respective leds*/
                              if((i==0)||(i==4))
                                    set_line_led(LINE_LED_15,ON);

                              if((i==1)||(i==5))
                                    set_line_led(LINE_LED_26,ON);

                              if((i==2)||(i==6))
                                    set_line_led(LINE_LED_37,ON);

                              if((i==3)||(i==7))
                                    set_line_led(LINE_LED_48,ON);

                              light_value = read_adc_avg(0,10);

                              uint16_t temp;

                              temp = ((light_value -
light_sensors[i].cal_min) * 1000 )/ ( light_sensors[i].cal_max -
light_sensors[i].cal_min );  //normalize line values
                              //TODO: Values always stay low WTF?

                              if(temp<0)  //prevent underflow
                                    temp=0;

                              if(temp>1023)  //prevent overflow
                                    temp=1023;

                              light_value = temp;
                        }
```

```
                        //fill  transmission buffer with light values
                        txbuffer[2*i] = LOW_BYTE(light_value);
                        txbuffer[2*i+1] = HIGH_BYTE(light_value);

                        set_line_led(LINE_LED_ALL,OFF);
                    }
                }
                /*Sampling and processing END*/

                set_ir_led(rxbuffer[17],rxbuffer[18]);  //switch Distance
IR-Leds
            }

            return 0;
}
```

line/line_header.c

```
/**********************
FILE: motor_header.h
DATE: 2.11.2011
**********************/
/* DEFINES */
#define SLAVE_ADRESSE         0x40  //I2C Slave Adresse

#define ON  1
#define OFF 0

#define NUM_SENSORS           8

/*I2C-Protocol*/
#define LINE_SENSOR_COMMAND         16
#define IR_LED_COMMAND              17
#define LINE_SENSOR_CALIBRATION_COMMAND 19


/*IR Leds*/
#define IR_LED_UNTEN_VORNE1   1
#define IR_LED_UNTEN_VORNE2   2
#define IR_LED_OBEN_LINKS2          3
#define IR_LED_OBEN_LINKS1          4
#define IR_LED_OBEN_VORNE1          5
#define IR_LED_OBEN_VORNE2          6
#define IR_LED_OBEN_RECHTS1         7
#define IR_LED_OBEN_RECHTS2         8

#define IR_LED_ALL                        0
#define IR_LED_UNTEN_VORNE_ALL      12
#define IR_LED_OBEN_VORNE_ALL       56
#define IR_LED_OBEN_LINKS_ALL       34
#define IR_LED_OBEN_RECHTS_ALL      78

/*Line Leds*/
#define LINE_LED_ALL                0
#define LINE_LED_48                 48
#define LINE_LED_37                 37
#define LINE_LED_26                 26
#define LINE_LED_15                 15



/*MACROS*/
#define uniq(LOW,HEIGHT)      ((HEIGHT << 8)|LOW)                 // 2x
8Bit to 16Bit
#define LOW_BYTE(x)           (x & 0xff)                          //
16Bit to 8Bit
#define HIGH_BYTE(x)          ((x >> 8) & 0xff)             // 16Bit to
8Bit



/* PINBELEGUNG MOTOR

PORTB
  PB0 - IR_LED_OBEN_RECHTS2
  PB1 - IR_LED_UNTEN_VORNE1
  PB2 - IR_LED_OBEN_LINKS1
```

```
  PB3 - MOSI
  PB4 - MISO
  PB5 - SCK/ IR_LED_UNTEN_VORNE2

PORTC
  PC0 - Liniensensor Multiplexer Output
  PC1 - Liniensensor Multiplexer A
  PC2 - Liniensensor Multiplexer B
  PC3 - Liniensensor Multiplexer C
  PC4 - SDA
  PC5 - SCL
  PC6 - Reset

PORTD
  PD0 - Liniensensor LED 4 und 8 Ein/Aus
  PD1 - Liniensensor LED 3 und 7 Ein/Aus
  PD2 - Liniensensor LED 2 und 6 Ein/Aus
  PD3 - Liniensensor LED 1 und 5 Ein/Aus
  PD4 - IR_LED_OBEN_VORNE2
  PD5 - IR_LED_OBEN_LINKS2
  PD6 - IR_LED_OBEN_VORNE1
  PD7 - IR_LED_OBEN_RECHTS1

*/



/*Ports*/
#define PORT_IR_LED_OBEN_RECHTS2    PORTB
#define PORT_IR_LED_UNTEN_VORNE1    PORTB
#define PORT_IR_LED_UNTEN_VORNE2    PORTB
#define PORT_IR_LED_OBEN_LINKS1           PORTB

#define PORT_LINE_MUX                     PORTC

#define PORT_LINE_LED                     PORTD
#define PORT_IR_LED_OBEN_VORNE1           PORTD
#define PORT_IR_LED_OBEN_LINKS2           PORTD
#define PORT_IR_LED_OBEN_VORNE2           PORTD
#define PORT_IR_LED_OBEN_RECHTS1    PORTD



/*DDRx*/
#define DDR_IR_LED_OBEN_RECHTS2           DDRB
#define DDR_IR_LED_UNTEN_VORNE1           DDRB
#define DDR_IR_LED_UNTEN_VORNE2           DDRB
#define DDR_IR_LED_OBEN_LINKS1            DDRB

#define DDR_LINE_MUX                      DDRC

#define DDR_LINE_LED                      DDRD
#define DDR_IR_LED_OBEN_VORNE1            DDRD
#define DDR_IR_LED_OBEN_LINKS2            DDRD
#define DDR_IR_LED_OBEN_VORNE2            DDRD
#define DDR_IR_LED_OBEN_RECHTS1           DDRD


/*Pins*/
#define PIN_IR_LED_OBEN_RECHTS2     PB0
#define PIN_IR_LED_UNTEN_VORNE1     PB1
#define PIN_IR_LED_OBEN_LINKS1            PB2
#define PIN_IR_LED_UNTEN_VORNE2           PB5
```

```
#define PIN_LINE_MUX_OUT                PC0
#define PIN_LINE_MUX_A                  PC1
#define PIN_LINE_MUX_B                  PC2
#define PIN_LINE_MUX_C                  PC3

#define PIN_LINE_LED_15                 PD0
#define PIN_LINE_LED_26                 PD1
#define PIN_LINE_LED_37                 PD2
#define PIN_LINE_LED_48                 PD3
#define PIN_IR_LED_OBEN_VORNE2          PD4
#define PIN_IR_LED_OBEN_LINKS2          PD5
#define PIN_IR_LED_OBEN_VORNE1          PD6
#define PIN_IR_LED_OBEN_RECHTS1         PD7
```

line/line_lib.c

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

#include "twislave.h"
#include "line_header.h"


void init_io (void)
{
	/* Set In- Outputs*/
	DDR_IR_LED_OBEN_RECHTS2 |= (1<<PIN_IR_LED_OBEN_RECHTS2);
	DDR_IR_LED_UNTEN_VORNE1 |= (1<<PIN_IR_LED_UNTEN_VORNE1);
	DDR_IR_LED_UNTEN_VORNE2 |= (1<<PIN_IR_LED_UNTEN_VORNE2);
	DDR_IR_LED_OBEN_LINKS1  |= (1<<PIN_IR_LED_OBEN_LINKS1);
	DDR_IR_LED_OBEN_VORNE1  |= (1<<PIN_IR_LED_OBEN_VORNE1);
	DDR_IR_LED_OBEN_LINKS2  |= (1<<PIN_IR_LED_OBEN_LINKS2);
	DDR_IR_LED_OBEN_VORNE2  |= (1<<PIN_IR_LED_OBEN_VORNE2);
	DDR_IR_LED_OBEN_RECHTS1 |= (1<<PIN_IR_LED_OBEN_RECHTS1);

	DDR_LINE_MUX |= (1<<PIN_LINE_MUX_A) | (1<<PIN_LINE_MUX_B) |
(1<<PIN_LINE_MUX_C);
	DDR_LINE_MUX &= ~(1<<PIN_LINE_MUX_OUT);

	DDR_LINE_LED |= (1<<PIN_LINE_LED_48) |(1<<PIN_LINE_LED_37)
|(1<<PIN_LINE_LED_26) |(1<<PIN_LINE_LED_15);

	/*switch Outputs on/off*/
	PORT_IR_LED_OBEN_RECHTS2 &= ~(1<<PIN_IR_LED_OBEN_RECHTS2);
	PORT_IR_LED_UNTEN_VORNE1 &= ~(1<<PIN_IR_LED_UNTEN_VORNE1);
	PORT_IR_LED_UNTEN_VORNE2 &= ~(1<<PIN_IR_LED_UNTEN_VORNE2);
	PORT_IR_LED_OBEN_LINKS1  &= ~(1<<PIN_IR_LED_OBEN_LINKS1);
	PORT_IR_LED_OBEN_VORNE1  &= ~(1<<PIN_IR_LED_OBEN_VORNE1);
	PORT_IR_LED_OBEN_LINKS2  &= ~(1<<PIN_IR_LED_OBEN_LINKS2);
	PORT_IR_LED_OBEN_VORNE2  &= ~(1<<PIN_IR_LED_OBEN_VORNE2);
	PORT_IR_LED_OBEN_RECHTS1 &= ~(1<<PIN_IR_LED_OBEN_RECHTS1);

	PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_A) | (1<<PIN_LINE_MUX_B) |
(1<<PIN_LINE_MUX_C);

	PORT_LINE_LED &= ~(1<<PIN_LINE_LED_48) | (1<<PIN_LINE_LED_37) |
(1<<PIN_LINE_LED_26) | (1<<PIN_LINE_LED_15);
	/*(de)activate Pullups*/
	PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_OUT);

}


/*ADC*/

void init_adc(void)
{
	uint16_t result;

	//internal reference
	ADMUX = (1<<REFS1) | (1<<REFS0);
	//prescaler
	ADCSRA = (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
	//activate ADC
	ADCSRA |= (1<<ADEN);

	//make a dummy measurement
```

```
	ADCSRA |= (1<<ADSC);  //single conversion
	while (ADCSRA & (1<<ADSC) ); //wait until finished

	result = ADCW;  //read measurement
}


uint16_t read_adc(uint8_t channel)
{
	//select channel
	ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F);
	ADCSRA |= (1<<ADSC);            //single conversion
	while (ADCSRA & (1<<ADSC) );  // wait until finished

	return ADCW;                  // read and return the result
}


uint16_t read_adc_avg( uint8_t channel, uint8_t average )
{
	uint16_t result = 0;

	//make samples
	for(uint8_t i = 0; i < average; ++i )
		result += read_adc(channel);

	return ( result / average );  //calculate average
}


//Activate channel in line sensor multiplexer
void set_line_mux_channel(uint8_t channel)
{
	//extract lower(?) three bits from channel-byte NOT WORKING!
	//PORT_LINE_MUX |= ((channel & (1 << 0))<<PIN_LINE_MUX_A) |
((channel & (1 << 1))<<PIN_LINE_MUX_B) | ((channel & (1 <<
2))<<PIN_LINE_MUX_C);

	switch (channel)
	{
		case 0:
		{
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_A);
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_B);
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_C);
			break;
		}
		case 1:
		{
			PORT_LINE_MUX |= (1<<PIN_LINE_MUX_A);
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_B);
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_C);
			break;
		}
		case 2:
		{
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_A);
			PORT_LINE_MUX |= (1<<PIN_LINE_MUX_B);
			PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_C);
			break;
		}
		case 3:
		{
```

```
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_A);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_B);
                  PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_C);
                  break;
            }
            case 4:
            {
                  PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_A);
                  PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_B);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_C);
                  break;
            }
            case 5:
            {
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_A);
                  PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_B);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_C);
                  break;
            }
            case 6:
            {
                  PORT_LINE_MUX &= ~(1<<PIN_LINE_MUX_A);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_B);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_C);
                  break;
            }
            case 7:
            {
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_A);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_B);
                  PORT_LINE_MUX |= (1<<PIN_LINE_MUX_C);
                  break;
            }
      }


}


//Switch specified IR led on/off
void set_ir_led(uint8_t led,uint8_t state)
{
      switch (led)
      {
            case IR_LED_ALL:
            {

                  PORT_IR_LED_UNTEN_VORNE1 |= (state <<
PIN_IR_LED_UNTEN_VORNE1);
                  PORT_IR_LED_UNTEN_VORNE2 |= (state <<
PIN_IR_LED_UNTEN_VORNE2);
                  PORT_IR_LED_OBEN_LINKS1  |= (state <<
PIN_IR_LED_OBEN_LINKS1);
                  PORT_IR_LED_OBEN_LINKS2  |= (state <<
PIN_IR_LED_OBEN_LINKS2);
                  PORT_IR_LED_OBEN_VORNE1  |= (state <<
PIN_IR_LED_OBEN_VORNE1);
                  PORT_IR_LED_OBEN_VORNE2  |= (state <<
PIN_IR_LED_OBEN_VORNE2);
                  PORT_IR_LED_OBEN_RECHTS1 |= (state <<
PIN_IR_LED_OBEN_RECHTS1);
                  PORT_IR_LED_OBEN_RECHTS2 |= (state <<
PIN_IR_LED_OBEN_RECHTS2);
```

```
                  break;
            }

            case IR_LED_UNTEN_VORNE1:
            {
                  PORT_IR_LED_UNTEN_VORNE1 |= (state <<
PIN_IR_LED_UNTEN_VORNE1);
                  break;
            }

            case IR_LED_UNTEN_VORNE2:
            {
                  PORT_IR_LED_UNTEN_VORNE2 |= (state <<
PIN_IR_LED_UNTEN_VORNE2);
                  break;
            }

            case IR_LED_OBEN_LINKS1:
            {
                  PORT_IR_LED_OBEN_LINKS1 |= (state <<
PIN_IR_LED_OBEN_LINKS1);
                  break;
            }

            case IR_LED_OBEN_LINKS2:
            {
                  PORT_IR_LED_OBEN_LINKS2 |= (state <<
PIN_IR_LED_OBEN_LINKS2);
                  break;
            }

            case IR_LED_OBEN_VORNE1:
            {
                  PORT_IR_LED_OBEN_VORNE1 |= (state <<
PIN_IR_LED_OBEN_VORNE1);
                  break;
            }

            case IR_LED_OBEN_VORNE2:
            {
                  PORT_IR_LED_OBEN_VORNE2 |= (state <<
PIN_IR_LED_OBEN_VORNE2);
                  break;
            }

            case IR_LED_OBEN_RECHTS1:
            {
                  PORT_IR_LED_OBEN_RECHTS1 |= (state <<
PIN_IR_LED_OBEN_RECHTS1);
                  break;
            }

            case IR_LED_OBEN_RECHTS2:
            {
                  PORT_IR_LED_OBEN_RECHTS2 |= (state <<
PIN_IR_LED_OBEN_RECHTS2);
                  break;
            }
      }
}

//switch line led
void set_line_led(uint8_t led, uint8_t state)
```

```
{
	if(state==ON)
	{
		switch (led)
		{
			case LINE_LED_ALL:
			{
				PORT_LINE_LED |= (1 << PIN_LINE_LED_48) | (1 <<
PIN_LINE_LED_37) | (1 << PIN_LINE_LED_26) | (1 << PIN_LINE_LED_15);
				break;
			}

			case LINE_LED_48:
			{
				PORT_LINE_LED |= (1 << PIN_LINE_LED_48);
				break;
			}

			case LINE_LED_37:
			{
				PORT_LINE_LED |= (1 << PIN_LINE_LED_37);
				break;
			}

			case LINE_LED_26:
			{
				PORT_LINE_LED |= (1 << PIN_LINE_LED_26);
				break;
			}

			case LINE_LED_15:
			{
				PORT_LINE_LED |= (1 << PIN_LINE_LED_15);
				break;
			}
		}
	}
	else
	{
		switch (led)
		{
			case LINE_LED_ALL:
			{
				PORT_LINE_LED &= ~(1 << PIN_LINE_LED_48) & ~(1
<< PIN_LINE_LED_37) & ~(1 << PIN_LINE_LED_26) & ~(1 <<
PIN_LINE_LED_15);
				break;
			}

			case LINE_LED_48:
			{
				PORT_LINE_LED &= ~(1 << PIN_LINE_LED_48);
				break;
			}

			case LINE_LED_37:
			{
				PORT_LINE_LED &= ~(1 << PIN_LINE_LED_37);
				break;
			}

			case LINE_LED_26:
			{
```

```
                        PORT_LINE_LED &= ~(1 << PIN_LINE_LED_26);
                        break;
                    }

                    case LINE_LED_15:
                    {
                        PORT_LINE_LED &= ~(1 << PIN_LINE_LED_15);
                        break;
                    }
                }
            }

}
```

line/line_lib.h

```
void init_io (void);

void init_adc(void);
uint16_t read_adc( uint8_t channel );
uint16_t read_adc_avg( uint8_t channel, uint8_t average );

void set_line_mux_channel(uint8_t channel);

void set_ir_led(uint8_t led,uint8_t state);
void set_line_led(uint8_t led, uint8_t state);
```

motor/main.c

```
/*INCLUDES*/

//Standard C-Libs
#include <stdlib.h>
#include <stdint.h>

//Standard AVR-Libs
#include <util/twi.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>


//Libs
#include "motor_header.h"
#include "motor_lib.h"
#include "twislave.h"


int main(void)
{
  //_delay_ms(500);     //boot up delay
  io_init();            //set in/outputs, pullups etc.
  pwm_init();           //set pwm-mode, frequency etc.
  init_twi_slave(SLAVE_ADRESSE);   //I2C-Slave

  for(uint8_t i=0;i<i2c_buffer_size;i++)  i2cdata[i]=0;  //fill up
I2C-Buffer

  sei();  //activate global Interrupts

  while(1)
  {
    drive_motors();
  }

  return 0;
}
```

motor/motor_header.h

```
/**********************
FILE: motor_header.h
DATE: 2.11.2011
**********************/
/* DEFINES */
#define MAX_PWM_VALUE         1023
#define SLAVE_ADRESSE         0x50  //I2C Slave Adresse

#define FLOAT 2000

/*MACROS*/
#define uniq(LOW,HEIGHT)      ((HEIGHT << 8)|LOW)                 // 2x
8Bit to 16Bit
#define LOW_BYTE(x)           (x & 0xff)                          //
16Bit to 8Bit
#define HIGH_BYTE(x)          ((x >> 8) & 0xff)             // 16Bit to
8Bit



/* PINBELEGUNG MOTOR

PORTB
  PB0 - Greifer 2 A
  PB1 - Motor 1 EN
  PB2 - Motor 2 EN
  PB3 - MOSI
  PB4 - MISO
  PB5 - SCK

PORTC
  PC0 - Batterie Spannung
  PC1 - Greifer 1 EN
  PC2 - Greifer 1 B
  PC3 - Greifer 2 EN
  PC4 - SDA
  PC5 - SCL
  PC6 - Reset

PORTD
  PD0 - Motor 1 A
  PD1 - Motor 1 B
  PD2 - Encoder 1
  PD3 - Encoder 2
  PD4 - Motor 2 B
  PD5 - Motor 2 A
  PD6 - Greifer 1 A
  PD7 - Greifer 2 B

*/



/*Ports*/
#define PORT_GRIPPER2_A       PORTB
#define PORT_MOTOR1_EN        PORTB
#define PORT_MOTOR2_EN        PORTB

#define PORT_BATTERY_U        PORTC
```

```
#define PORT_GRIPPER1_EN      PORTC
#define PORT_GRIPPER1_B       PORTC
#define PORT_GRIPPER2_EN      PORTC

#define PORT_MOTOR1_A         PORTD
#define PORT_MOTOR1_B         PORTD
#define PORT_ENCODER1         PORTD
#define PORT_ENCODER2         PORTD
#define PORT_MOTOR2_B         PORTD
#define PORT_MOTOR2_A         PORTD
#define PORT_GRIPPER1_A       PORTD
#define PORT_GRIPPER2_B       PORTD


/*DDRx*/
#define DDR_GRIPPER2_A        DDRB
#define DDR_MOTOR1_EN         DDRB
#define DDR_MOTOR2_EN         DDRB

#define DDR_BATTERY_U         DDRC
#define DDR_GRIPPER1_EN       DDRC
#define DDR_GRIPPER1_B        DDRC
#define DDR_GRIPPER2_EN       DDRC

#define DDR_MOTOR1_A          DDRD
#define DDR_MOTOR1_B          DDRD
#define DDR_ENCODER1          DDRD
#define DDR_ENCODER2          DDRD
#define DDR_MOTOR2_B          DDRD
#define DDR_MOTOR2_A          DDRD
#define DDR_GRIPPER1_A        DDRD
#define DDR_GRIPPER2_B        DDRD


/*Pins*/
#define PIN_GRIPPER2_A        PB0
#define PIN_MOTOR1_EN         PB1
#define PIN_MOTOR2_EN         PB2

#define PIN_BATTERY_U         PC0
#define PIN_GRIPPER1_EN       PC1
#define PIN_GRIPPER1_B        PC2
#define PIN_GRIPPER2_EN       PC3

#define PIN_MOTOR1_A          PD0
#define PIN_MOTOR1_B          PD1
#define PIN_ENCODER1          PD2
#define PIN_ENCODER2          PD3
#define PIN_MOTOR2_B          PD4
#define PIN_MOTOR2_A          PD5
#define PIN_GRIPPER1_A        PD6
#define PIN_GRIPPER2_B        PD7


/************/
#define OCR_MOTOR1                OCR1A
#define OCR_MOTOR2                OCR1B
```

motor/motor_lib.c

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

#include "twislave.h"
#include "motor_header.h"


void io_init (void)
{
  /* Ein-/Ausgänge festlegen*/
 //Greifermotor 1
 DDR_GRIPPER1_A |= (1<< PIN_GRIPPER1_A);
 DDR_GRIPPER1_B |= (1<< PIN_GRIPPER1_B);
 DDR_GRIPPER1_EN |= (1<< PIN_GRIPPER1_EN);
 //Greifermotor 2
 DDR_GRIPPER2_A |= (1<< PIN_GRIPPER2_A);
 DDR_GRIPPER2_B |= (1<< PIN_GRIPPER2_B);
 DDR_GRIPPER2_EN |= (1<< PIN_GRIPPER2_EN);
 //Motor 1
 DDR_MOTOR1_A |= (1<< PIN_MOTOR1_A);
 DDR_MOTOR1_B |= (1<< PIN_MOTOR1_B);
 DDR_MOTOR1_EN |= (1<< PIN_MOTOR1_EN);
 //Motor 2
 DDR_MOTOR2_A |= (1<< PIN_MOTOR2_A);
 DDR_MOTOR2_B |= (1<< PIN_MOTOR2_B);
 DDR_MOTOR2_EN |= (1<< PIN_MOTOR2_EN);

 //Batterie Spannung
 DDR_BATTERY_U &= ~(1<< PIN_BATTERY_U);
 //Motorenencoder
 DDR_ENCODER1 &= ~(1<< PIN_ENCODER1);
 DDR_ENCODER2 &= ~(1<< PIN_ENCODER2);

  /*Ausgänge ein-/auschalten*/

 PORT_GRIPPER1_A &= ~(1<< PIN_GRIPPER1_A);
 PORT_GRIPPER1_B &= ~(1<< PIN_GRIPPER1_B);
 PORT_GRIPPER1_EN &= ~(1<< PIN_GRIPPER1_EN);

 PORT_GRIPPER2_A &= ~(1<< PIN_GRIPPER2_A);
 PORT_GRIPPER2_B &= ~(1<< PIN_GRIPPER2_B);
 PORT_GRIPPER2_EN &= ~(1<< PIN_GRIPPER2_EN);

 PORT_MOTOR1_A &= ~(1<< PIN_MOTOR1_A);
 PORT_MOTOR1_B &= ~(1<< PIN_MOTOR1_B);
 PORT_MOTOR1_EN &= ~(1<< PIN_MOTOR1_EN);

 PORT_MOTOR2_A &= ~(1<< PIN_MOTOR2_A);
 PORT_MOTOR2_B &= ~(1<< PIN_MOTOR2_B);
 PORT_MOTOR2_EN &= ~(1<< PIN_MOTOR2_EN);

 /*Eingänge Pullups ein/aus*/

 PORT_BATTERY_U &= ~(1<< PIN_BATTERY_U);
 PORT_ENCODER1 |= (1<< PIN_ENCODER1);
 PORT_ENCODER2 |= (1<< PIN_ENCODER2);


}

//PWM initialisieren
```

```
void pwm_init(void)
{
  //nicht invertierender pwm modus
  TCCR1A |= (1 << COM1A1) | (1 << COM1B1);
  //10-bit "phase corrected" pwm
  TCCR1A |= (1 << WGM11) | (1 << WGM10);
  //set prescaler to 8
  TCCR1B |= (1 << CS11);

  //beide Motoren ausschalten
  OCR_MOTOR1 = 0;
  OCR_MOTOR2 = 0;
}

//"Falsche" PWM_Eingaben korrigieren
int16_t correct_pwm(int16_t value)
{
  if(value> MAX_PWM_VALUE)
    return MAX_PWM_VALUE;
  else if(value<(-MAX_PWM_VALUE))
    return -MAX_PWM_VALUE;
  else
    return value;
}

//Motor 1 steurern
//-1023 <= speed <= 1023
//speed 0 => Brake
void motor1_speed (int16_t speed)
{
  speed = correct_pwm(speed);

  if(speed>0)
  {
    PORT_MOTOR1_A |= (1<< PIN_MOTOR1_A);
    PORT_MOTOR1_B &= ~(1<< PIN_MOTOR1_B);
      OCR_MOTOR1 = speed;
  }
  else if(speed==0) //BRAKE
  {
    PORT_MOTOR1_A |= (1<< PIN_MOTOR1_A);
    PORT_MOTOR1_B |= (1<< PIN_MOTOR1_B);
   OCR_MOTOR1 = speed;
  }
  else if(speed<0)
  {
    PORT_MOTOR1_A &= ~(1<< PIN_MOTOR1_A);
    PORT_MOTOR1_B |= (1<< PIN_MOTOR1_B);
      OCR_MOTOR1 = speed *-1;
  }
}


//Motor 2 steurern
//-1023 <= speed <= 1023
//speed 0 => Brake
void motor2_speed (int16_t speed)
{
  speed = correct_pwm(speed);

  if(speed>0)
  {
    PORT_MOTOR2_A |= (1<< PIN_MOTOR2_A);
```

```
    PORT_MOTOR2_B &= ~(1<< PIN_MOTOR2_B);
      OCR_MOTOR2 = speed;
  }
  else if(speed==0) //BRAKE
  {
    PORT_MOTOR2_A |= (1<< PIN_MOTOR2_A);
    PORT_MOTOR2_B |= (1<< PIN_MOTOR2_B);
   OCR_MOTOR2 = speed;
  }
  else if(speed<0)
  {
    PORT_MOTOR2_A &= ~(1<< PIN_MOTOR2_A);
    PORT_MOTOR2_B |= (1<< PIN_MOTOR2_B);
      OCR_MOTOR2 = speed *-1;
  }
}

//Disconnect motor1
void motor1_float(void)
{
  PORT_MOTOR1_A &= ~(1<< PIN_MOTOR1_A);
  PORT_MOTOR1_B &= ~(1<< PIN_MOTOR1_B);
  OCR_MOTOR1 = 0;
}

//Disconnect motor2
void motor2_float(void)
{
  PORT_MOTOR2_A &= ~(1<< PIN_MOTOR2_A);
  PORT_MOTOR2_B &= ~(1<< PIN_MOTOR2_B);
  OCR_MOTOR2 = 0;
}


//read motor commands from i2c-buffer and drive motors
void drive_motors(void)
{
    //get the motor speed values from I2C-Buffer
      //convert 2 bytes to 16bit value
      int16_t _motor1_speed = uniq(i2cdata[0],i2cdata[1]);
      int16_t _motor2_speed = uniq(i2cdata[2],i2cdata[3]);

      //Float Motor?
      if(_motor1_speed == FLOAT)
        motor1_float();
      else
        motor1_speed(correct_pwm(_motor1_speed));  //Drive motor1

      //Float Motor?
      if(_motor2_speed == FLOAT)
        motor2_float();
      else
        motor2_speed(correct_pwm(_motor2_speed));  //Drive motor2
}
```

motor/motor_lib.h

```
void io_init (void);
void pwm_init(void);

int16_t correct_pwm(int16_t value);

void motor1_speed (int16_t value);
void motor2_speed (int16_t value);

void motor1_float(void);
void motor2_float(void);

void drive_motors(void);
```