Entwicklung einer Motorsteuerung für zwei Getriebemotoren

T. K.

19. Februar 2008

Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Studienarbeit

Entwicklung einer Motorsteuerung für zwei Getriebemotoren

von
cand. Dipl.-Ing. T. K.

Aufgabenstellung und Betreuung:
Prof. Dr.-Ing. L. Wolf und Dipl.-Ing. D. Brökelmann

Braunschweig, den 19. Februar 2008

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 19. Februar 2008

Kurzfassung

Diese Studienarbeit umfasst die Entwicklung einer Motorsteuerung für zwei Getriebemotoren. Ziel dieser Arbeit ist eine Mikrocontroller-gesteuerte Fahrplattform, welche hauptsächlich für ein Team Projekt am Institut für Betriebssysteme und Rechnerverbund (IBR) eingesetzt wird. Der Hauptbestandteil des Fahrmoduls ist ein Atmel AVR Mikrocontroller, der die beiden Getriebemotoren über einen Treiber ansteuert und die resultierende Bewegung über zwei Inkrementalgeber überwacht. Dadurch lassen sich die Geschwindigkeit und die zurückgelegte Strecke ermitteln, um somit das Fahrzeug präzise bewegen zu können. Bei der Implementierung wurde speziell darauf Wert gelegt, dass das Fahrzeug in der Lage ist, geradeaus zu fahren. Dem Benutzer stehen einfache Steuerbefehle zur Verfügung, die wahlweise über EIA-232 oder I^2C übertragen werden können. Das dazugehörige Protokoll ist ebenfalls Bestandteil dieser Arbeit.

Abstract

This paper is about the development of a motor control unit with two standard dc motors. The goal is a microcontroller circuit for a team project at the IBR at the Technical University of Braunschweig. Main part is a Atmel AVR Microcontroller which controls the dc motors through a driver. The resulting movement is fed back with two rotary encoders and you can determine the driven way and speed to do precise moves. Especially a high value was set on driving straight ahead. A small set of control commands are available to the user, which could be sent over an I^2C bus oder a simple EIA-232 interface. The corresponding protocol is also element of this paper.

Technische Universität Braunschweig
Institut für
Betriebssysteme und Rechnerverbund
Prof. Dr.-Ing. L. Wolf, Prof. Dr.-Ing. M. Beigl

Braunschweig, 20. Nov. 2007

Aufgabenstellung für die Studienarbeit

„Entwicklung einer Motorsteuerung für zwei Getriebemotoren“

vergeben an
Herrn cand. Dipl.-Ing. T. K.

Matr.-Nr. xxxxxxx, E-Mail: klingeb@ibr.cs.tu-bs.de
Studiengang: Informations–Systemtechnik

Aufgabenstellung

Im Mikroprozessorlabor des Institutes für Betriebssysteme und Rechnerverbund wird das Teamprojekt „Programmierung verteilter eingebetteter Systeme“ durchgeführt. Im Laufe des Teamprojektes wird eine Fahrplattform eingesetzt, mit dessen Hilfe Sensorik und Aktorik demonstriebar wird. Für diese Fahrplattform soll eine Motorsteuereinheit entwickelt und implementiert werden. Die Fahrplattform enthält zwei Gleichstrom-Getriebemotoren welche jeweils mit zwei Hallsensoren zur Drehwinkelbestimmung bestückt sind. Die Auflösung der Drehwinkelbestimmung beträgt 360 Impulse pro Umdrehung.

Es ist eine Schaltung zu entwickeln, deren zentrales Element durch einen Mikrocontroller realisiert wird. Desweiteren soll die Schaltung eine serielle Schnittstelle und einen I2C-Bus zur Übermittlung von Befehlen und zur Ausgabe von Kontroll- und Statusmeldungen enthalten. Zur autonomen Ausgabe ist die Ansteuerung eines LC-Displays zu realisieren. Als Ergebnis sind die Motoren mittels Puls-Weiten-Modulation in Richtung und Geschwindigkeit anzusteuern.

Im Rahmen dieser Studienarbeit ist diese Schaltung als gedruckte Schaltung zu entwerfen, aufzubauen und in Betrieb zu nehmen. Zur Kommunikation über den I2C-Bus bzw. über die serielle Schnittstelle ist ein Protokoll für die Befehlsübermittlung und zum Informationsaustausch zu entwerfen und zu implementieren. Bei der Umsetzung der Befehle in Drehbewegungen der Räder ist besonderer Wert auf eine Geradeausfahrt der Fahrplattform zu legen.

Laufzeit: 3 Monate

Die Hinweise zur Durchführung von Studien- und Diplomarbeiten am IBR sind zu beachten (siehe http://www.ibr.cs.tu-bs.de/kb/arbeiten.html).

Aufgabenstellung und Betreuung:

Prof. Dr.-Ing. L. Wolf _____

Dipl.-Ing. Dieter Brökelmann _____

Bearbeitung:

cand. Dipl.-Ing. T. K. _____

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Studienarbeit beschreibt eine Fahrplattform, die für Arbeiten am Institut für Betriebssysteme und Rechnerverbund eingesetzt werden soll.

Die Plattform unterscheidet sich von bereits vorher eingesetzten Fahrmodulen durch die Art des Antriebs. Zuvor genutzte Plattformen arbeiteten meistens mit modifizierten Servomotoren, welche nicht sehr präzise zu steuern waren. Diese neue Plattform wurde daher mit einfachen Gleichstrom-Getriebemotoren ausgestattet, die mit einem Inkrementalgeber gekoppelt sind. Dadurch lässt sich die Bewegung der Motoren genau kontrollieren. Besonders wurde in dieser Arbeit Wert auf das Geradeausfahren der Plattform gelegt, was nur durch eine genaue Regelung der Geschwindigkeit möglich ist.

Das Kernstück der Arbeit ist eine Platine mit einem Atmel AVR Mikrocontroller mega2561, der sowohl die Motorregelung als auch die Kommunikation mit einer Hauptplatine koordiniert. Darüber hinaus ist es möglich, Debugausgaben zu erhalten um die Funktion der Platine einfacher zu überprüfen. Zu diesem Zweck kann wahlweise auch ein LC-Display angeschlossen werden.

Zur Steuerung der Fahrplattform wird dem Benutzer ein einfaches Protokoll zur Verfügung gestellt, mit dem sich alle Bewegungen realisieren lassen, ohne genaueres Wissen über die Ansteuerung eines Motors zu benötigen. Weiterhin lassen sich Werte wie die Geschwindigkeit oder die gefahrene Strecke abrufen, um somit die Position kontrollieren zu können.

Kapitel 2 erläutert zunächst einige Hardwareaspekte wie die Funktion der Inkrementalgeber oder die Belegung der Steckverbinder. In Kapitel 3 wird danach näher auf die Software im Mikrocontroller eingegaben. Kapitel 4 setzt den Benutzer über das Protokoll in Kenntnis, um das Fahrmodul bedienen zu können.

2 Hardware

In diesem Abschnitt werden einige grundlegende Hardwareaspekte der Studienarbeit beschrieben. Dabei wird speziell auf die drei Elemente des Regelkreises eingegangen, die Motoren, die Inkrementalgeber und den Mikrocontroller. Darüber hinaus werden kurz die übrigen Elemente der Schaltung erläutert, um dem Leser Kenntnisse über Steckverbindungen etc. zu vermitteln.

2.1 Drehzahlsteuerung bei Gleichstrommotoren

Für die Realisierung eines Fahrmoduls muss die Steuerungselektronik in der Lage sein, die Drehzahl der angeschlossenen Motoren zu verändern. Für unterschiedliche Arten von Motoren gibt es dafür auch unterschiedliche Möglichkeiten. Hier soll jedoch nur auf die Steuerung von Gleichstrommotoren eingegangen werden, die im Rahmen dieser Arbeit mit der einfachen Pulsweitenmodulation, im folgenden *PWM* genannt, durchgeführt wird.

Ein PWM-Signal ist ein einfaches Rechtecksignal mit gleichbleibender Frequenz, bei dem jedoch das Puls-Pausen-Verhältnis (meist auch als Tast-Verhältnis oder *Duty Cycle* bezeichnet) veränderlich ist.

Abbildung 2.1 zeigt eine solches PWM-Signal, bei dem zwei verschiedene Tast-Verhältnisse verwendet werden. Trotz dieser Änderung sieht man, dass sich die Periodendauer nicht verändert hat.

Wird mit solch einem Signal ein Motor angesteuert, so ändert sich in Abhängigkeit zum Tast-Verhältniss die Durchschnittsspannung, woraus eine Drehzahländerung resultiert. Für die Durchschnittsspannung gilt

$$\overline{U_{\mathrm{pwm}}} = U_{\mathrm{in}} \cdot \frac{t_{\mathrm{on}}}{t_{\mathrm{on}} + t_{\mathrm{off}}} = U_{\mathrm{in}} \cdot t_{\mathrm{on}} \cdot f_{\mathrm{pwm}},$$

wobei U_{in} die Eingangsspannung, t_{on} die Pulsdauer und t_{off} die Pausendauer ist. Alternativ lässt sich das Ergebnis auch aus Eingangsspannung, Pulsdauer und der Frequenz

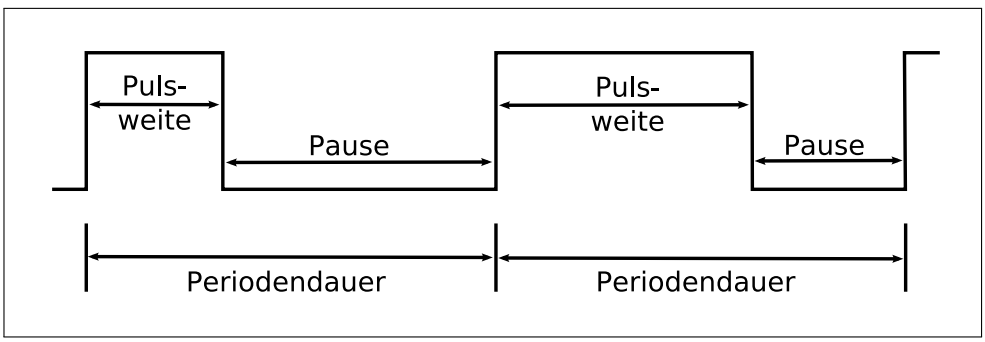

Abbildung 2.1: Schematische Darstellung einer Pulsweitenmodulation

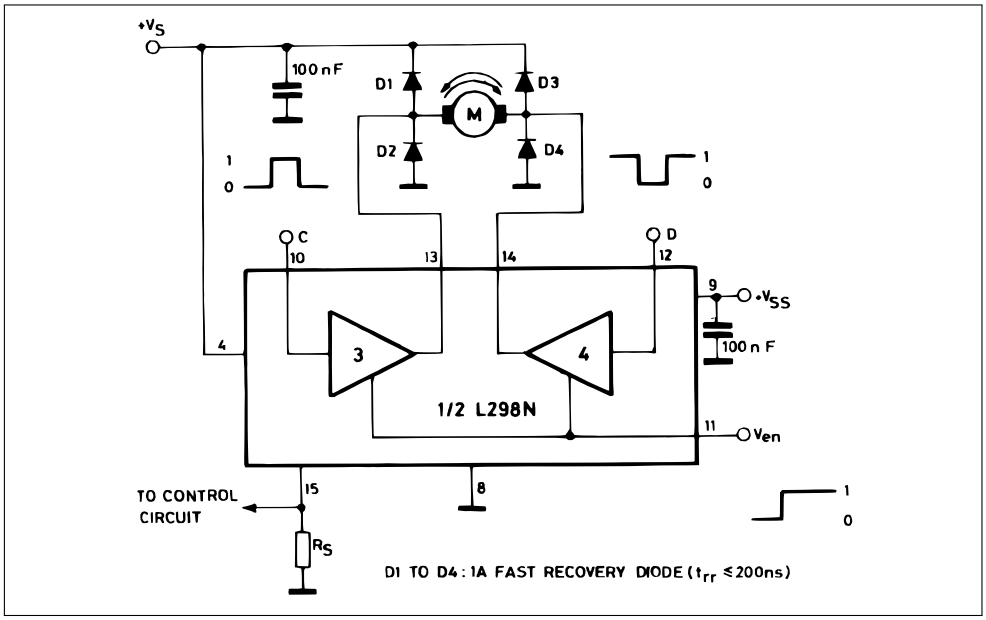

Abbildung 2.2: Motortreiber mit Motor und Schutzdioden[5]

f_{pwm} bestimmen, was die Pulsweitenmodulation charakterisiert. Um nicht nur die Geschwindigkeit zu variieren, sondern auch die Richtung umzukehren, muss lediglich die Polarität umgedreht werden.

Die Wahl der Frequenz ist je nach Verbraucher verschieden, sollte jedoch nicht zu hoch gewählt werden, um auftretende Schaltverluste gering zu halten. Die hier verwendetet Frequenz liegt etwa bei $f_{\mathrm{pwm}} = 500\mathrm{Hz}$. Da dies innerhalb des hörbaren Bereichs liegt, kann man bei Ansteuerung des Motors deutlich einen $500\mathrm{Hz}$ Ton wahrnehmen.

Da ein Mikrocontroller nicht die von den Motoren benötigte Leistung aufbringen kann, wird ein geeigneter Treiberbaustein dazwischen geschaltet. Der hier verwendete Motortreiber *L298* der Firma *ST*[5] besitzt für jeden Motorausgang drei Eingänge, von denen einer direkt mit dem PWM-Signal verbunden wird. Die anderen Eingänge dienen zur Einstellung der Drehrichtung. Um den Motortreiber vor hohen Spannungsspitzen zu schützen, die vom Motor in der t_{off}-Phase induziert werden können, ist der Anschluss von Schutzdioden unumgänglich, die diese Spitzen in die Versorgungsleitung abführen. Abbildung 2.2 zeigt einen Ausschnitt aus dem Datenblatt des Treibers. Daraus wird ersichtlich, wie die Dioden angeschlossen werden müssen. Wegen der höheren Frequenzen werden hier schnelle Schottky-Dioden verwendet, die für die auftretenden Ströme ausreichend groß dimensioniert sein müssen.

Abbildungen 2.3 und 2.4 zeigen grob den internen Aufbau des Motortreibers für einen Kanal. Die H-Brücke aus vier Transistoren ist in der Mitte zu sehen. Über die vier AND-Gatter, deren Ausgänge die Basis der Transistoren steuern, wird das Enable-Signal zugeschaltet. Ist Enable auf Low, sind alle Gatter geschlossen und der Motor ist von der Stromversorgung getrennt.

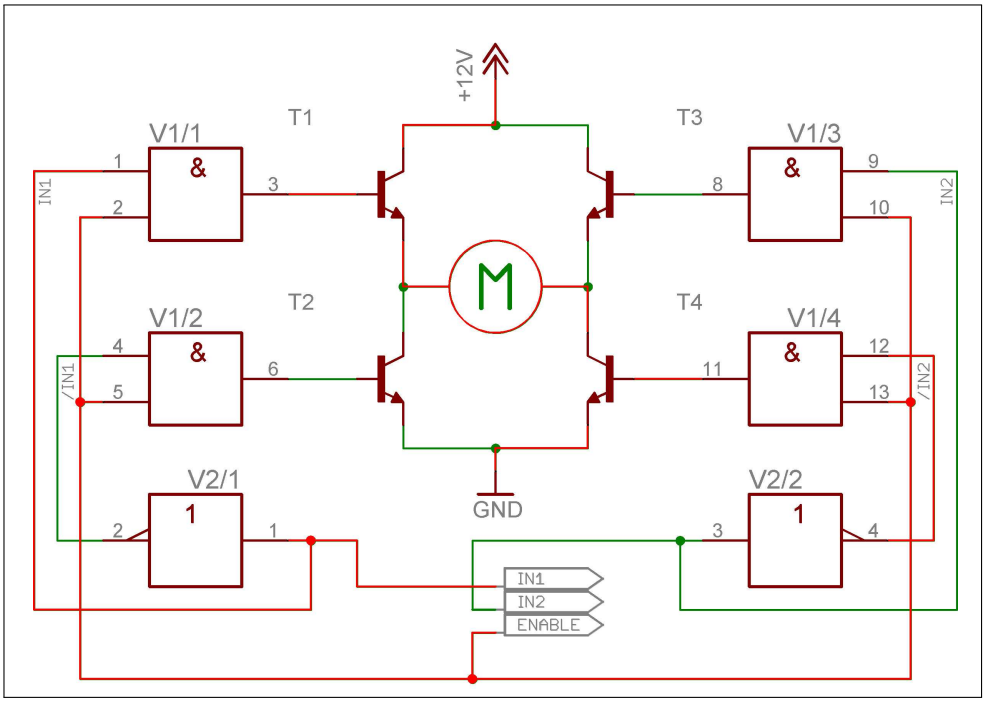

Abbildung 2.3: Strompfad bei Linksdrehung

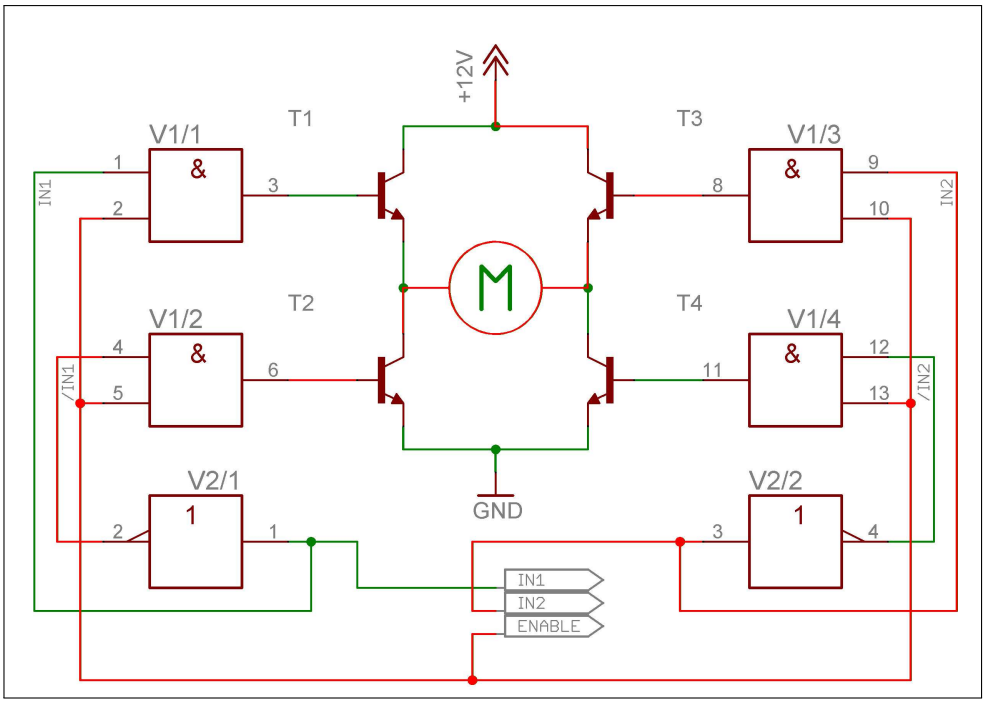

Abbildung 2.4: Strompfad bei Rechtsdrehung

In Abbildung 2.3 stehen Enable und der Eingang IN1 auf High, Eingang IN2 hingegen auf Low. (Erkennbar durch die Farbgebung rot = High, grün = Low) Dadurch ist das AND-Gatter links oben geöffnet und das darunter liegende wegen des Inverters geschlossen. Auf der gegenüberliegenden Seite, die von IN2 gesteuert wird, ist es genau umgekehrt. Der Strom fliesst also von der Quelle durch den Transistor T1, anschließend von links nach rechts durch den Motor, um dann über T4 mit der Masse verbunden zu werden. In der Abbildung 2.4 liegt an IN1 nun der Low-Pegel, an IN2 der High-Pegel. Die Transistoren T2 und T3 sind nun leitend, während T1 und T4 gesperrt sind, was einen Stromfluss von rechts nach links durch den Motor bewirkt. Durch den Einsatz der Inverter wird die H-Brücke zusätzlich geschützt, da es nicht dazu kommen kann, dass die Transistoren auf einer Seite der Brücke gleichzeitig leitend sind. Dadurch würde der Strom auf einer Seite am Motor vorbei fließen weil die Transistoren auf dieser Seite beide öffnen und einen Kurzschluss der Versorgungsspannung verursachen.

2.2 Inkrementalgeber

Um einen Gleichstrommotor in seiner Geschwindigkeit nicht nur steuern sondern auch regeln zu können, bedarf es einer Rückmeldung vom Motor zur Steuerung. Dazu wird die Achse des Motors mit einem Inkrementalgeber verbunden, dessen Signal zum Beispiel mit Hilfe eines Mikrocontrollers ausgewertet wird. Wie die Auswertung im Rahmen dieser Studienarbeit konkret aussieht, wird in Abschnitt 3.2 näher erläutert. Hier sollen zunächst nur der Inkrementalgeber und das Signal beschrieben werden. Der hier verwendete Inkrementalgeber ist fest mit der Rückseite des Motors verbunden. Er besteht lediglich aus einem auf der Motorachse befestigten Dauermagneten und zwei um 90 Grad versetzten Hall-Sensoren (siehe Abbildung 2.5).

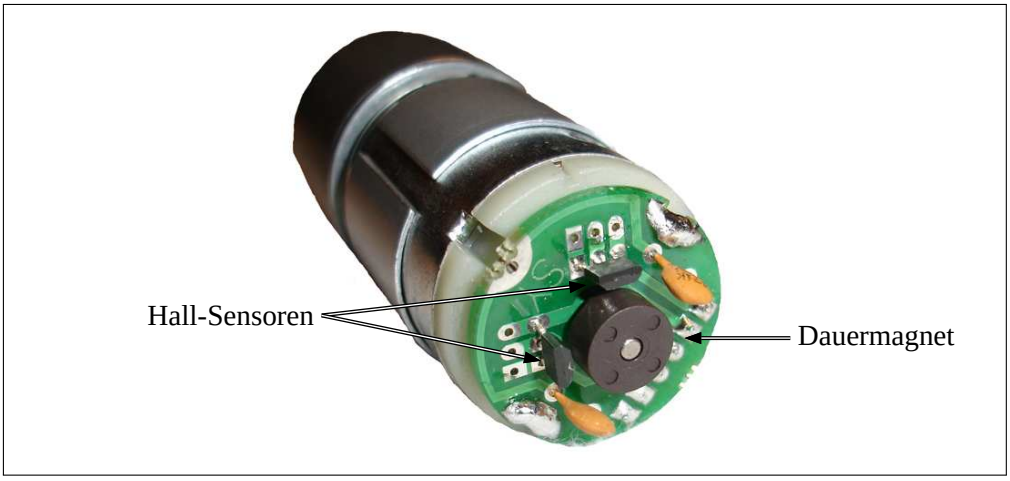

Abbildung 2.5: Inkrementalgeber auf Motorrückseite

Wenn der Magnet durch die Drehung des Motors bewegt wird, ändert sich das magnetische Feld an den Hall-Sensoren, und zwar in Form einer Sinusschwingung. Da beide Sensoren um 90 Grad versetzt um den Magneten angeordnet sind, sind auch die

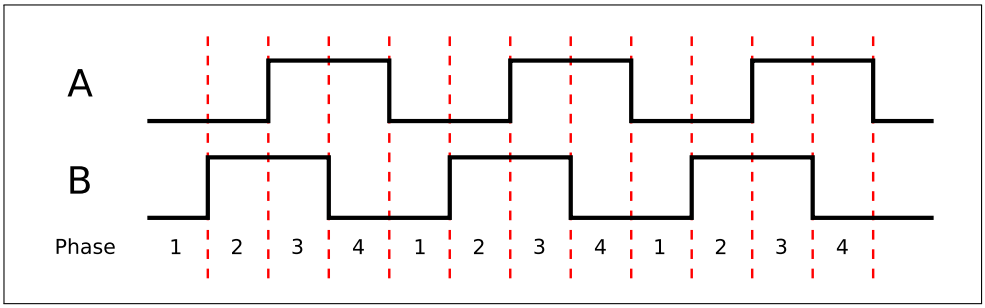

Abbildung 2.6: Quadraturausgang des Inkrementalgebers

Phase	A	B
1	0	0
2	0	1
3	1	1
4	1	0

Tabelle 2.1: Kombinationen des Quadraturausgangs

Ausgangssignale um 90 Grad phasenverschoben. Wegen der etwas schwierigeren Verarbeitung eines solchen analogen Signals mit einem Mikrocontroller - es würden zwei Analog-Digital-Wandler benötigt - ist in den hier gezeigten Hall-Sensoren zusätzlich noch ein Komparator integriert, welcher das Sinussignal in ein Rechteck umwandelt. Das resultierende Ausgangssignal beider Sensoren ist in Abbildung 2.6 dargestellt.

Unter den Signalen ist zusätzlich noch die Phase angegeben. In Tabelle 2.1 werden genau die vier möglichen Kombinationen der beiden Ausgänge gezeigt, die daher auch als *Quadraturausgang* bezeichnet werden.

Mit Hilfe dieses Signals lassen sich nun sowohl die Geschwindigkeit des Motors als auch dessen Drehrichtung auswerten. Für den Benutzer ist hierbei nur die resultierende Geschwindigkeit hinter dem Getriebe interessant. Daher ist die Kenntnis über das Übersetzungsverhältnis irrelevant. Lediglich die Information, dass aus einer Radumdrehung 360 Ticks am Ausgang des Inkrementalgebers resultieren, wird zur Berechnung benötigt. Werden beispielsweise innerhalb einer Sekunde 540 Ticks gezählt, müssen diese durch Division mit 360 in Umdrehungen umgerechnet werden. Nach einer weiteren Multiplikation mit dem Faktor 60 ergibt sich eine Geschwindigkeit von 90 Umdrehungen pro Minute. Diese Bestimmung der Geschwindigkeit ist auch mit nur einem Ausgang des Inkrementalgebers möglich. In diesem Fall würden sich dabei die Anzahl von Ticks pro Umdrehung bzw. die gezählten Ticks innerhalb einer Sekunde halbieren. Zur Erkennung der Drehrichtung werden hingegen beide Signale benötigt. Je nach Richtung treten die möglichen Ausgangskombinationen in unterschiedlicher Reihenfolge auf. Bei einer Drehung im Uhrzeigersinn ergibt sich der Phasenverlauf

$$1 \to 2 \to 3 \to 4 \to 1 \to 2 \to \cdots,$$

bei der Drehung gegen den Uhrzeigersinn ist der Phasenverlauf

$$1 \to 4 \to 3 \to 2 \to 1 \to 4 \to \cdots.$$

Die Drehrichtung lässt sich also aus den Zustandsübergängen ableiten.

2.3 Mikrocontroller

Der Mittelpunkt des gesamten Fahrmoduls dieser Studienarbeit ist ein Mikrocontroller. Erste 'Gehversuche' bei der Entwicklung basierten auf dem Einsatz mehrerer Controller, um zeitkritische Aufgaben, wie zum Beispiel die Auswertung der Inkrementalgeber-Signale, mit Hilfe von schnellen Assembler Programmen zu bewältigen. Das Hauptprogramm sollte auf einem zentralen Controller in der Sprache *C* implementiert werden. Jedoch zeigte sich sehr bald, dass die als zeitkritisch angenommenen Signale problemlos mit einem Mikrocontroller erfasst werden konnten. Zusätzlich hätte die Kommunikation der Controller untereinander zu weiteren Problemen geführt, die Wahl fiel daher nun auf den *ATmega2561* der Firma *ATMEL* (Abbildung 2.7).

Abbildung 2.7: Mikrocontroller ATmega2561

Dieser Mikrocontroller verfügt über einen 8-Bit RISC Kern, der in der Lage ist, 1 MIPS pro MHz zu erreichen. Die gesamte *ATmegaXXX* Reihe von *ATMEL* hat unter anderem diverse Vorteile:
Ein zur Entwicklung benötigter C-Compiler basiert auf dem bekannten GCC, der im Internet frei verfügbar ist. Die Assembler Entwicklungsumgebung von *ATMEL* lässt sich ebenfalls frei im Internet beschaffen. Die Mikrocontroller besitzen alle einen ISP-Port, und ein passender Programmieradapter lässt sich mit wenigen Bauteilen aus der Bastelkiste anfertigen. Darüber hinaus lassen sich die meisten Controller mit einem Systemtakt von bis zu 16 MHz betreiben, was für ausreichende Geschwindigkeit sorgt. Bei dem hier verwendeten *ATmega2561* existieren 256 Kilobyte Flash-Speicher für das Programm und 8 Kilobyte interner SRAM. Die weiteren benötigten Funktionen des Controllers sind:

- **Timer**
  Ein Timer ist ein im Hintergrund unabhängig laufender Zähler mit 8 oder 16 Bit. Die Geschwindigkeit lässt sich über einen Teiler relativ zum Systemtakt anpassen. Das Programm hat jederzeit die Möglichkeit, auf den Wert des Timers zuzugreifen. Es lassen sich Werte festlegen, die von der Hardware permanent mit dem Zähler verglichen werden und bei Übereinstimmung einen Interrupt auslösen.
  In dieser Arbeit werden zwei Zähler benötigt. Der erste liefert eine genaue Zeitbasis, welche im Kapitel 3, Abschnitt 3.2 näher beschrieben wird. Der zweite Timer arbeitet im PWM-Modus. Hierzu wird ein 8-Bit Timer über seinen Teiler zunächst so konfiguriert, dass ein Überlauf (Übergang von 255 nach 0) etwa 500 Mal in der Sekunde auftritt. Zu dieser Grundfrequenz kommt zusätzlich ein Vergleichswert, der direkt mit einem Ausgangspin gekoppelt wird. Beim Zählerüberlauf wird der Ausgangspin auf 0 geschaltet, stimmt der Vergleichswert mit dem Zähler überein, schaltet der Ausgang auf 1. Über diese Funkti-

onsweise erhält man an dem Ausgang eine Pulsweitenmodulation, deren Tastverhältnis über den Vergleichswert gesteuert werden kann.

- **USART**
  USART steht für Universal Serial Asynchronous Receiver Transmitter und stellt eine serielle Schnittstelle zu Verfügung, die für das EIA-232 Interface genutzt werden kann. Auch hierfür lässt sich ein Interrupt konfigurieren, der bei ankommenden Zeichen auslöst und ein ständiges Pollen von Daten überflüssig macht.

- **TWI**
  Das Two-Wire-Interface ist eine weitere hardwaregesteuerte Schnittstelle, die mit dem bekannten I^2C kompatibel ist und genau für diesen Zweck zum Einsatz kommt. Der Controller stellt eine Reihe von Statuscodes zur Verfügung über die die Kommunikation überwacht werden kann. Der Vorteil einer solchen Schnittstelle ist wie bei der USART, dass sich der Benutzer nicht um die Übertragung einzelner Bits und deren Timing kümmern muss, da dieses vollständig von der Hardware übernommen wird.

- **8-Bit I/Os**
  Der Controller verfügt über eine Vielzahl von digitalen Pins, die sowohl als Eingang als auch als Ausgang genutzt werden können. Bei der Motorsteuerung werden diese I/Os für verschiedene Dinge benötigt, zu denen beispielsweise das LC-Display, die DIP-Schalter, die Drehgebereingänge, etc. gehören.

2.4 Schaltungsbeschreibung

2.4.1 Blockschaltbild

Das Blockschaltbild 2.8 zeigt grundlegende Teile der Motorsteuerung. Das zentrale Element, welches die verschiedenen Ein- und Ausgänge miteinander verbindet, ist der Mikrocontroller. Die beiden Kommunikationsschnittstellen, I^2C und EIA-232, die DIP-Schalter für verschiedene Konfigurationen sowie das optionale LC-Display sind direkt an den Controller angeschlossen und können so über die Software entsprechend kontrolliert werden. Der Regelkreis ist in diesem Blockdiagram besonders hervorgehoben, da er das Kernstück dieser Studienarbeit bildet. Er besteht auf dem Mikrocontroller, den über die Motorendstufe verbundenen Motoren M1 und M2 und den zur Schließung des Kreises notwendigen Inkrementalgebern. Die unterbrochene Linie zwischen den Motoren und den Inkrementalgebern deutet darauf hin, dass hier eine ausschließlich mechanische Verbindung besteht. Alle anderen Komponenten sind elektrisch über Leiterbahnen und Leitungen miteinander verbunden. Ein detailliertere Beschreibung des Regelkreises befindet sich in Kapitel 3, Abschnitt 3.4. In dem Blockdiagram nicht dargestellt ist die Stromversorgung. Diese erzeugt aus einem Bleiakkumulator eine geregelte +5V-Spannung, welche über einen Standard-Linearregler, LM7805, erzeugt wird und hauptsächlich zum Betrieb des Mikrocontrollers benötigt wird. Die ungeregelte Ausgangsspannung des Akkumulators, ca. +12V, wird vom Motortreiber direkt zur Ansteuerung der Motoren benutzt und lediglich über einen Elektrolytkondensator gestützt, um etwaige Stromspitzen auszugleichen.

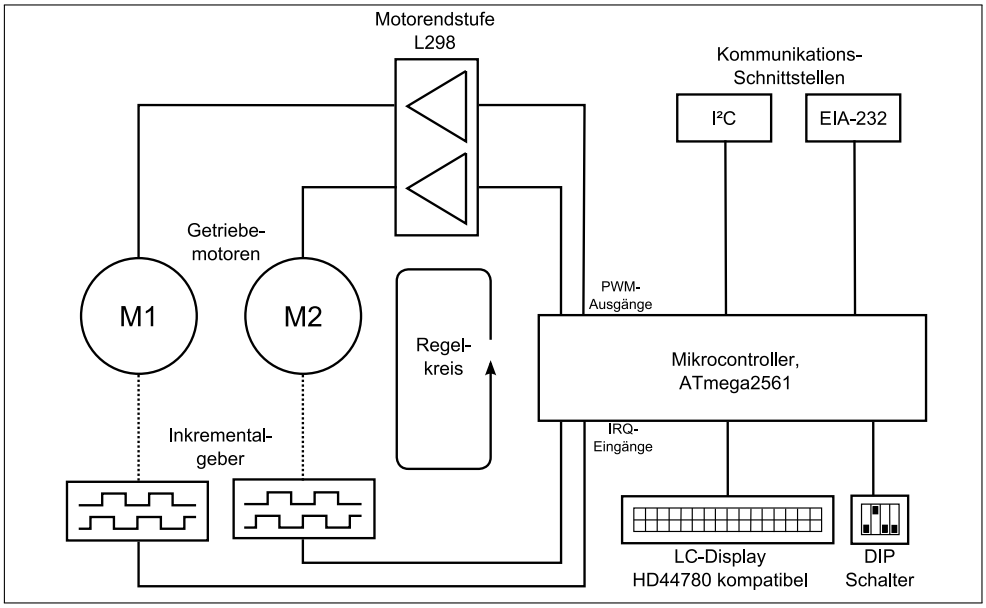

Abbildung 2.8: Blockschaltbild des Fahrmoduls

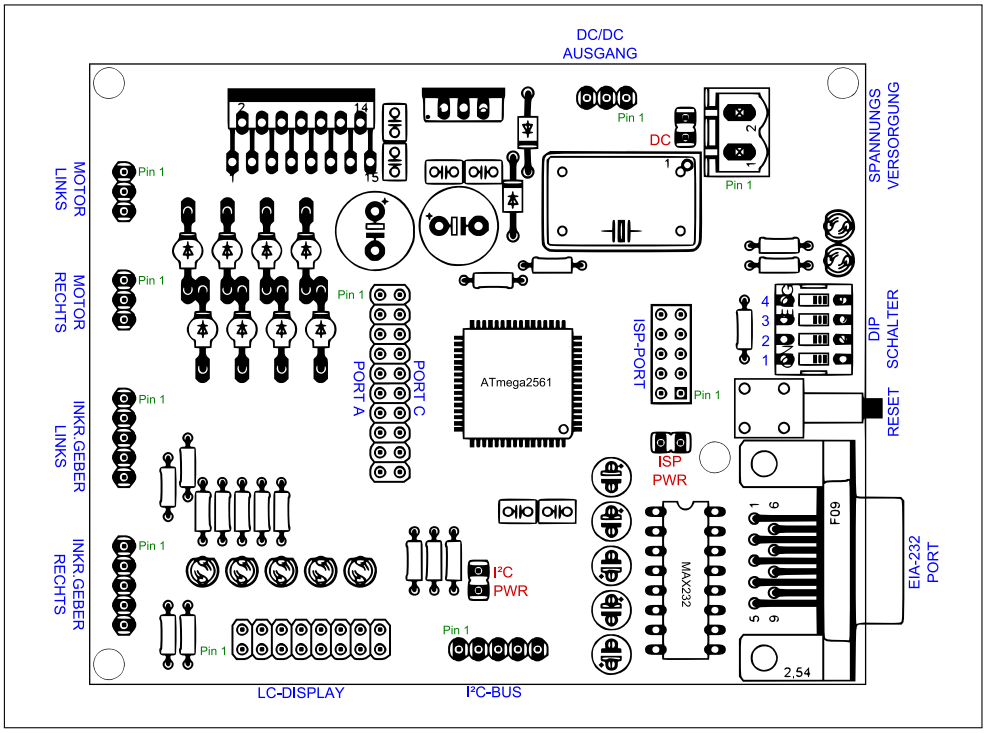

Abbildung 2.9: Fahrmodul, Bestückungsansicht

2.4.2 Steckverbinder

Um die Platine des Fahrmoduls mit den Motoren, den Inkrementalgebers, der Spannungsversorgung etc. zu verbinden, sind unterschiedliche Anschlüsse vorgesehen, de-

ren Funktion und Pinbelegungen hier erläutert werden soll. Eine Übersicht über alle Steckverbinder und deren Position auf der Platine stellt Abbildung 2.9 dar, wobei die Namen der Anschlüsse blau, der jeweilige *Pin 1* grün und die Jumper in roter Schrift gekennzeichnet sind.

- **MOTOR LINKS**
  Anschluss für den linken Motor der Fahrmoduls.

Pin	Funktion
1	Positiver Ausgang, linker Motor
2	Negativer Ausgang, linker Motor
3	*nicht belegt*

Tabelle 2.2: Pinbelegung, MOTOR LINKS

- **MOTOR RECHTS**
  Anschluss für den rechten Motor der Fahrmoduls.

Pin	Funktion
1	Positiver Ausgang, rechter Motor
2	Negativer Ausgang, rechter Motor
3	*nicht belegt*

Tabelle 2.3: Pinbelegung, MOTOR RECHTS

- **INKREMENTALGEBER LINKS**
  Anschluss für den Inkrementalgeber des linken Motors.

Pin	Funktion
1	+5V Versorgungsspannung
2	Masse
3	Ausgang A
4	Ausgang B
5	*nicht belegt*

Tabelle 2.4: Pinbelegung, INKREMENTALGEBERG LINKS

- **INKREMENTALGEBER RECHTS**
  Anschluss für den Inkrementalgeber des rechten Motors.

Pin	Funktion
1	+5V Versorgungsspannung
2	Masse
3	Ausgang A
4	Ausgang B
5	*nicht belegt*

Tabelle 2.5: Pinbelegung, INKREMENTALGEBER RECHTS

- **LC-DISPLAY**
  Optionaler Anschluss eines HD44780-kompatiblen LC-Displays.

Funktion	Pin	Pin	Funktion
Masse	1	2	+5V Versorgungssp.
Kontrast	3	4	Reg Select
R/W	5	6	Enable
nicht belegt	7	8	*nicht belegt*
nicht belegt	9	10	*nicht belegt*
D4	11	12	D5
D6	13	14	D7
LED A, Hintergrundbel.	15	16	LED K, Hintergrundbel.

Tabelle 2.6: Pinbelegung, LC DISPLAY

- **I^{2}C BUS**
  Anschluss für den I^{2}C-Bus.

Pin	Funktion
1	Masse
2	SDA
3	SCL
4	*nicht belegt*
5	+5V (zuschaltbar durch Jumper `I2C PWR`)

Tabelle 2.7: Pinbelegung, I^{2}C BUS

- **EIA-232 PORT**
  Serieller EIA-232 Port.

Pin	Funktion
1, 4, 6, 9	*nicht belegt*
2	TX (aus Sicht des Fahrmoduls)
3	RX
5	Masse
7	*verbunden mit Pin 8*
8	*verbunden mit Pin 7*

Tabelle 2.8: Pinbelegung, EIA-232 PORT

- **SPANNUNGSVERSORGUNG**
  Anschluss für die Spannungsversorgung des Blei-Akkumulators.

Pin	Funktion
1	+12V
2	Masse

Tabelle 2.9: Pinbelegung, SPANNUNGSVERSORGUNG

- **DC/DC AUSGANG**
  Optionaler Anschluss für einen DC/DC Wandler. Bei Nichtverwendung muss der Jumper `DC` gesetzt werden.

Pin	Funktion
1	+12V Ausgang
2	Masse
3	+7V Eingang (mindestens)

Tabelle 2.10: Pinbelegung, DC/DC AUSGANG

- **ISP-Port**
  Anschlussmöglichkeit für In-System-Programmer zur Programmierung des Mikrocontrollers. Wird der Jumper `ISP POWER` gesetzt, kann die +5V-Versorgung über diesen Port erfolgen, sodass keine weitere Spannungsquelle zur Programmierung notwendig ist.

Funktion	Pin	Pin	Funktion
MOSI	1	2	+5V
GND	3	4	GND
Reset	5	6	GND
SCK	7	8	GND
MISO	9	10	GND

Tabelle 2.11: Pinbelegung, ISP-Port

- **PORT A, PORT C**
  Diese beiden Ports werden für die Motorsteuerung nicht benötigt und stehen für mögliche Programmerweiterungen zur Verfügung.

Funktion	Pin	Pin	Funktion
GND	1	2	GND
PA0	3	4	PC0
PA1	5	6	PC1
PA2	7	8	PC2
PA3	9	10	PC3
PA4	11	12	PC4
PA5	13	14	PC5
PA6	15	16	PC6
PA7	17	18	PC7
+5V	19	20	+5V

Tabelle 2.12: Pinbelegung, PORT A und PORT C

- **DIP SCHALTER**
  Die Einstellungen, die über die DIP-Schalter verändert werden können, sind in Kapitel 4, Abschnitt 4.1 aufgelistet.

3 Software

Nachdem im vorherigen Kapitel auf den Aufbau des Fahrmoduls eingegangen wurde, soll sich dieses Kapitel nun der im Mikrocontroller implementierten Software widmen. Vor einigen Jahren war es noch üblich, einfache Funktionen mit Hilfe diskreter Logik direkt in der Hardware aufzubauen. Inzwischen sind Mikrocontroller jedoch so günstig, dass auch im Rahmen dieser Studienarbeit ein solcher Controller die komplette Steuerung übernehmen kann. Im Folgenden werden Funktionen des Programms erläutert, jedoch ohne dabei ins Detail des Quelltextes zu gehen.

3.1 Hauptschleife

Die Hauptschleife, dargestellt in Abbildung 3.1, bestimmt den Grundablauf des Programms. Sie läuft permanent weiter und beginnt nach jedem Durchlauf wieder erneut. Er werden nun die einzelnen Blöcke in ihrem Sinn und ihrer Funktion erläutert.

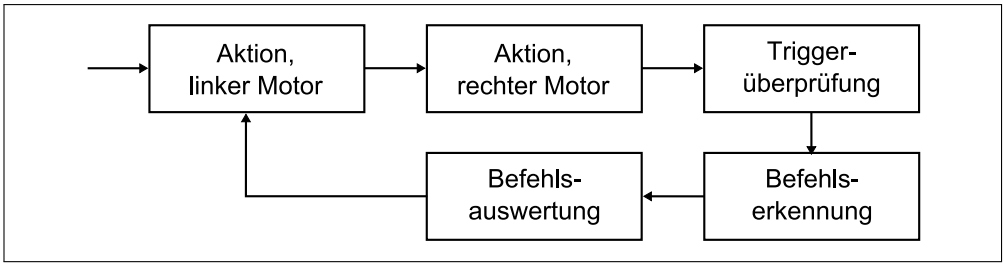

Abbildung 3.1: Schematische Darstellung der Hauptschleife

Aktion, linker und rechter Motor

Dieser stets zu Anfang der Schleife auftretende Block hat die Aufgabe, die Motoren zu steuern. Jeder einzelne Motor hat 3 grundlegende Betriebsarten. Im *Freilauf* wird die Spannungsversorgung ausgeschaltet, der Motor lässt sich frei drehen. Zum *Bremsen* werden die Anschlussleitungen kurzgeschlossen und für die *„normale Fahrt“* steuert man den Motor mit einer Puls-Weiten-Modulation an. Um eine dieser Betriebsarten einzustellen, reicht das einmaliges Ausführen des Blockes, da die Grundarten nur ein- oder ausgeschaltet werden müssen. Für zwei weitere Betriebsarten, die auf den Grundarten aufbauen, ist das nicht ausreichend, da sie in jedem Durchlauf eine erneute Kontrolle bzw. Berechnung benötigen.

- Aktives Bremsen
  Bereits erste Fahrversuche zeigten, dass der normales Bremsmodus nicht ausreichend ist, um ein Rad vollständig anzuhalten. Viel mehr wurde dieses, falls sich das andere Rad in Bewegung befand, mitgezogen. Daher mußte eine *aktive*

Bremse über die Software realisiert werden. Die Funktion ist dabei relativ simpel. Soll ein Rad an einer bestimmten Stelle festgehalten werden, so speichert man einfach die dazugehörige Position, die dann in jedem Schleifendurchlauf mit der aktuellen Position verglichen wird. Erkennt das Programm nun eine Abweichung, wird der Motor entsprechend angesteuert um die gespeicherte Position wieder zu erreichen.

- PID-Fahrt
  Durch diesen Modus wird die eigentliche Motorregelung realisiert. PID-Fahrt bedeutet, dass es sich hierbei um das Fahren mit Hilfe eines PID-Reglers handelt, in dem **p**roportionale, **i**ntegrale und **d**ifferentielle Anteile eine Rolle spielen und in die Berechnung eingehen. Nach der Ermittelung der Geschwindigkeit, der Abweichung zum Sollwert und der Berechnung des neuen Puls-Pausen-Verhältnisses wird hier der neue Wert für die PWM eingestellt. Eine genauere Beschreibung des PID-Reglers befindet sich in Abschnitt 3.4.

Im Programm ist die derzeit aktive Betriebsart in einer Variablen gespeichert. Mit Hilfe eines *switch*-Blocks wird dann der zuständige Programmteil angesprungen. Für die drei Grundarten, die wie bereits erwähnt nur einmalig ein- oder ausgeschaltet werden müssen, hat man die Möglichkeit, diesen Block zu überspringen.

Setzt man nun diese Betriebsarten zusammen, lassen dich daraus prinzipiell alle benötigten Bewegungen zusammensetzen. Zur Veranschaulichung dient die (unvollständige) Zusammenfassung in der Tabelle 3.1.

Modus, links	Tempo	Modus, rechts	Tempo	Resultat
Aktives Bremsen	0	Aktives Bremsen	0	Anhalten
PID-Fahrt	20	PID-Fahrt	20	Fahren, geradeaus, Tempo 20
PID-Fahrt	40	PID-Fahrt	20	Leichte Rechtskurve
PID-Fahrt	40	Aktives Bremsen	0	Drehung um rechtes Rad
PID-Fahrt	-20	PID-Fahrt	+20	Drehung um Achsmittelpunkt

Tabelle 3.1: Kombinationen der Betriebsarten

Triggerüberprüfung

Dieses Fahrmodul ist mit einer Befehlskette ausgestattet, die es dem Benutzer erlaubt, verschiedene Fahrkommandos unmittelbar hintereinander zu übertragen. Diese Funktion setzt voraus, dass es eine Bedingung geben muss, wann ein Fahrkommando fertig ist, um erst dann das nächste Kommando in der Befehlskette auszuführen. Diese Bedingung wird mit Hilfe der Trigger realisiert, ein einfaches *Fahre geradeaus*-Kommando wird durch ein *Fahre geradeaus bis...*-Kommando erweitert. Der Benutzer hat bei den Fahrkommandos für jedes Rad die Möglichkeit, drei verschiedene Abbruchbedingungen einzustellen.

- Zeittrigger
  Bei dieser Abbruchbedingung wird in Unabhängigkeit des Betriebsmodus ein Timer gestartet. Ist dieser abgelaufen, gilt das aktuelle Kommando als erledigt. Denkbare Kommandos sind beispielsweise *Fahre 5 Sekunden geradeaus* oder *Bleibe 2 Sekunden stehen*.

- Positionstrigger
  Diese Abbruchbedingung überwacht während der Fahrt die aktuelle mit einer eingestellten Position des Rades. Ist die Position erreicht, gilt das Kommando als erledigt und der zugehörige Motor wird aktiv gebremst. Für Kommandos wie *Fahre 1000 Ticks und dann...* ist dieser Trigger zwingend notwendig.

- kein Trigger
  Zusätzlich besteht die Möglichkeit, für ein Rad gar keinen Trigger zu verwenden. Ein Fahrkommando ohne Trigger wird ausgeführt und gilt sofort als erledigt. Dadurch lässt sich zum Beispiel nur ein Trigger für das andere Rad festlegen und es sind Kommandos wie *Drehe auf der Stelle, bis ein neues Kommando kommt* möglich.

In dem Block *Triggerüberprüfung* in Abbildung 3.1 werden die gerade beschriebenen Trigger behandelt. Zunächst wird festgestellt, ob bzw. welcher der beiden Trigger aktiv ist, und falls ja wird überprüft, ob die Abbruchbedingung bereits erreicht ist.

Befehlserkennung

Damit ein Benutzer das Fahrmodul steuern kann, muss dieses in der Lage sein, Befehle entgegenzunehmen. In jedem Schleifendurchgang wird deshalb überprüft, ob der Benutzer ein neues Kommando an das Fahrmodul gesendet hat. Der Block *Befehlserkennung* testet zusätzlich, ob es sich um ein gültiges Kommando handelt, wobei zwischen zwei Arten von Befehlen unterschieden wird.

- Ein-Byte-Kommandos
  Ein Ein-Byte-Kommando ist, wie aus dem Namen hervorgeht, 1 Byte groß. Genutzt werden diese Kommandos für einfach Steuerbefehle wie zum Beispiel *Reset*, *Anhalten*, *Queue löschen*, etc. Diese einfachen Anweisungen werden ohne Umwege über die Befehlskette unmittelbar ausgeführt. Weitere, noch in der Queue vorhandenen Befehle, werden verworfen.

- Fahr-Kommandos
  Das Fahr-Kommando besteht aus einer ganzen Zeichenkette und beinhaltet einen Fahrbefehl. Ohne zusätzliche Ein-Byte-Kommandos wird ein neues Fahr-Kommando am Ende der Befehlskette eingetragen und erst ausgeführt, wenn alle vorherigen Befehle beendet sind.

Die genaue Form und Syntax der Kommandos wird im Abschnitt 3.3 genau beschrieben.

Befehlsauswertung

Im vorherigen Block *Befehlserkennung* wurde beschrieben, wie ein Fahr-Kommando in die Befehlskette gelangt. Dieser Block ist nun dafür zuständig, dass die Befehle der Kette ausgeführt werden. Zunächst wird überprüft, ob noch ein Befehl aktiv ist, also derzeit ein Fahrkommando läuft, von dem mindestens ein Trigger noch nicht ausglöst hat. Ist dieses nicht der Fall, wird das nächste Kommando aus der Befehlskette übernommen. Nach der Auswertung werden gegebenenfalls die Betriebsmodi verändert und die Parameter für die Trigger gesetzt. Kommandos in der Kette, die keine direkten Fahrbefehle sind, sondern lediglich einen Parameter einstellen und daher keine Option für die Triggereinstellung besitzen, werden wie Fahrkommandos ohne Trigger behandelt und sofort als erledigt markiert. Im nächsten Schleifendurchlauf wird dann anschließend das nächste Kommando ausgeführt, da kein Trigger aktiv ist.

3.2 Interrupts

Der vorherige Abschnitt zeigte den Grundablauf des Programms, der stets in der gleichen Reihenfolge nacheinander ausgeführt wird. Eine Besonderheit dieses Fahrmoduls ist, dass es permanent die Drehung der Räder überwacht, wodurch eine Regelung überhaupt möglich wird. Würde diese Überprüfung in einem Block der Hauptschleife stattfinden, gingen mit großer Wahrscheinlichkeit einige Impulse des Inkrementalgeberausgangs verloren, da die Hauptschleife sich möglichweise gerade in einem anderen Block befindet. Das zweite Problem ist die Tatsache, dass manche Funktionen, beispielsweise die PID-Regelung, eine feste Zeitbasis für ihre Berechnung benötigen. Da die Hauptschleife jedoch keine feste Ausführungszeit benötigt, muss an anderer Stelle im Hintergrund für einen solchen festen Zeittakt gesorgt werden.

Um diese Probleme in den Griff zu bekommen, bedient man sich der in der Mikrocontrollerhardware fest eingebauten Interrupts. Ein Interrupt (zu deutsch: Unterbrechung) hat nun die Möglichkeit, aufgrund verschiedener, einstellbarer Ereignisse das Programm an beliebiger Stelle zu unterbrechen und eine eigene Funktion auszuführen. Ist diese beendet, springt das Programm an die Stelle der Unterbrechung zurück und setzt seinen Ablauf fort. Die Software dieses Fahrmoduls verwendet verschiedene Interrupts, von denen hier die zwei für die Regelschleife wichtigen beschrieben werden sollen.

3.2.1 Externer Interrupt

Der externe Interrupt überwacht die einstellbaren Eingangspins des Mikrocontrollers auf eine Pegeländerung hin, wobei gewählt werden kann, welche Art von Änderung zu einem Interrupt führen soll. Genutzt wird diese Funktion, um das Quadratursignal der Inkrementalgeber auszuwerten, bei dem sowohl die fallende als auch die steigende Flanke benötigt wird.

Im Hardwarekapitel über die Inkrementalgeber wurde bereits erläutert, dass die Drehrichtung aus den Zustandsübergängen erkannt werden kann. Mit Hilfe der Interrupts funktioniert diese Erkennung hier in etwas abgewandelter Form: Beide Ausgänge eines Inkrementalgebers sind je mit einem Interrupt Eingang des Mikrocontrollers verbunden. Daher ist der Software bekannt, bei welchem Signal die Pegeländerung aufge-

treten ist. Diese Information, zusätzlich zu den beiden Signalpegeln, lässt eine eindeutige Identifizierung der Drehrichtung zu. Der vorherige Zustand wird nicht zusätzlich gespeichert, sondern ist gewissermassen im Interrupt vorhanden. Die folgende Tabelle 3.2 zeigt, wie sich die Drehrichtung aus Signalpegeln und Interrupt zusammensetzt. Zum besseren Verständnis wird hier das Signal des Inkrementalgebers aus Kapitel 2.2 nochmals dargestellt.

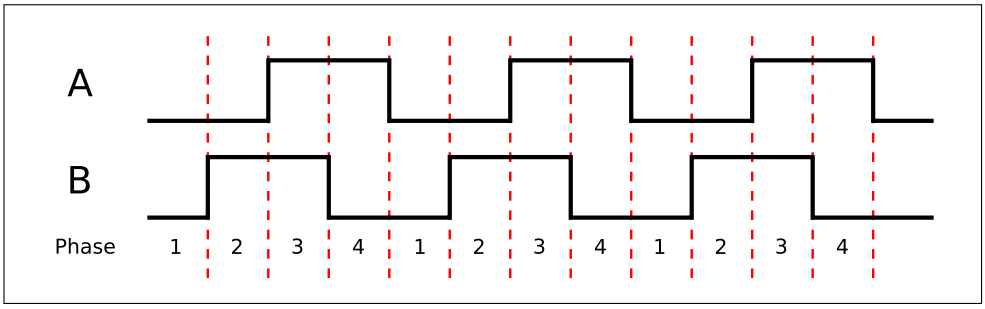

Abbildung 3.2: Wiederholung, Quadraturausgang

Interrupt von Signal	Pegel A	Pegel B	Phasenübergang	Drehrichtung
A	1	1	$2 \rightarrow 3$	rechts
A	0	0	$4 \rightarrow 1$	rechts
A	1	0	$1 \rightarrow 4$	links
A	0	1	$3 \rightarrow 2$	links
B	0	1	$1 \rightarrow 2$	rechts
B	1	0	$3 \rightarrow 4$	rechts
B	1	1	$4 \rightarrow 3$	links
B	0	0	$2 \rightarrow 1$	links

Tabelle 3.2: Drehrichtungserkennung mit Interrupts

Listing 3.1 zeigt einen Quelltextausschnitt, an dem die Drehrichtungserkennung für ein Rad deutlich wird. Das Schlüsselwort `ISR` kennzeichnet dabei eine Interrupt Service Routine dessen Interruptvektor in diesem Fall `INT4_vect` bzw. `INT5_vect` heißt. Eine Flanke des Ausgangs A erzeugt einen externen Interrupt 4, eine Flanke von B entsprechend den Interrupt 5, und es wird im Programm an die entsprechende Stelle gesprungen. Innerhalb der Funktion werden nun die relevanten Bits, welche die Pegel von A und B repräsentieren, aus dem Eingang herausgefiltert und anschließend mit einer Switch-Anweisung verzweigt. Die Möglichkeiten, die bei den Cases auftreten können, passen zu den in Tabelle 3.2 gezeigten Kombinationen.

```
ISR(INT4_vect) // Interrupt Service Routine für Drehgeber 1-A
{
  switch (IRQ_PIN & 0x30) // Bit 4 und 5 filtern
    {
    case ((0 << IRQ_A0) | (0 << IRQ_B0)): // IRQ A, A:0, B:0
    case ((1 << IRQ_A0) | (1 << IRQ_B0)): // IRQ A, A:1, B:1
      // Rechtsdrehung, Rad 1
      {...}
      break;

    case ((0 << IRQ_A0) | (1 << IRQ_B0)): // IRQ A, A:0, B:1
    case ((1 << IRQ_A0) | (0 << IRQ_B0)): // IRQ A, A:1, B:0
      // Linksdrehung, Rad 1
      {...}
      break;
    }
}

ISR(INT5_vect) // Interrupt Service Routine für Drehgeber 1-B
{
  switch (IRQ_PIN & 0x30) // Bit 4 und 5 filtern
    {
    case ((0 << IRQ_A0) | (0 << IRQ_B0)): // IRQ B, A:0, B:0
    case ((1 << IRQ_A0) | (1 << IRQ_B0)): // IRQ B, A:1, B:1
      // Linksdrehung, Rad 1
      {...}
      break;

    case ((0 << IRQ_A0) | (1 << IRQ_B0)): // IRQ B, A:0, B:1
    case ((1 << IRQ_A0) | (0 << IRQ_B0)): // IRQ B, A:1, B:0
      // Rechtsdrehung, Rad 1
      {...}
      break;
    }
}
```

Listing 3.1: Quelltextausschnitt, Externe Interrupts

3.2.2 Timer-Interrupt

Der Timer-Interrupt wird benötigt, um dem Programm eine feste Zeitbasis zur Verfügung zu stellen. Zu diesem Zweck kommt ein 16-Bit Timer zum Einsatz, der nach Einstellung des Vorteilers auf 64 sein Zählregister alle 4μs um einen Wert erhöht.

$$\mathrm{clk}_{\mathrm{timer}} = \frac{\mathrm{clk}_{\mathrm{I/O}}}{\mathrm{prescacler}} = \frac{16\mathrm{MHz}}{64} = 250\mathrm{kHz}$$

$$\mathrm{t}_{\mathrm{interval}} = \frac{1}{\mathrm{clk}_{\mathrm{timer}}} = 4\mu s$$

- $\mathrm{clk}_{\mathrm{timer}}$
  Mit diesem Takt inkrementiert der Timer sein Zählregister
- $\mathrm{clk}_{\mathrm{I/O}}$
  Systemtakt des Mikrocontrollers $= 16\mathrm{MHz}$
- $\mathrm{t}_{\mathrm{interval}}$
  Eingestellte kleinste Zeitauflösung des Timers

Mit Hilfe dieser eingestellten kleinsten Zeitauflösung werden nun drei Zeitinterrupts erzeugt. Der Timer besitzt dafür drei Compare-Register, die jeweils einen Interrupt auslösen, wenn der Timer den entsprechenden Compare-Wert erreicht. Dieser Vergleich läuft komplett in der Hardware ab. Die Werte für die Vergleichsregister werden so gewählt, dass alle 1ms, 10ms und 100ms ein Interrupt ausgelöst wird.

$$\mathrm{compare}_a = 250 \longrightarrow 1ms = 250 \cdot 4\mu s$$

$$\mathrm{compare}_b = 2500 \longrightarrow 10ms = 2500 \cdot 4\mu s$$

$$\mathrm{compare}_c = 25000 \longrightarrow 100ms = 25000 \cdot 4\mu s$$

In der entsprechenden Interrupt Service Routine wird jedesmal der Comparewert aufaddiert um das nächste Zeitintervall einzustellen. Da sowohl die Vergleichsregister als auch das Zählregister des Timers jeweils 16 Bit breit sind, ergibt sich bei einem Überlauf automatisch eine Berechnung $\mathrm{mod}\, 2^{16} = 65536$.

Zusätzlich wird bei Überlauf des Zählregisters ein weiterer Interrupt generiert, der ein weiters, ca. 262ms langes Interval erzeugt.

$$\frac{\mathrm{clk}_{\mathrm{I/O}}(= 16\mathrm{MHz})}{64 \cdot 2^{16}} \approx 3.81\mathrm{Hz}$$

$$\mathrm{t}_{\mathrm{overflow}} = \frac{1}{3.81\mathrm{Hz}} \approx 262ms$$

3.3 Kommunikation

Damit ein Benutzer dem Fahrmodul Anweisungen übermitteln kann, benötigt das Fahrmodul eine Schnittstelle, über die Befehle entgegengenommen und Werte zurückgegeben werden können. Das hier entwickelte Modul besitzt gleich zwei unterschiedliche Schnittstellen zur Kommunikation, zum einen EIA-232, zum anderen I^2C, die in diesem Abschnitt näher erläutert werden.

3.3.1 EIA-232

Diese Schnittstelle ist dazu da, um Befehle an das Fahrmodul zu schicken. Weiterhin werden mögliche Debug-Ausgaben, falls diese aktiviert sind, über dieses Interface ausgegeben. Zur Kommunikation müssen bei der Gegenseite folgende Parameter eingestellt werden:

```
115200 Baud, 8 Datenbits, keine Parität, 1 Stoppbit
```

Anschließend kann mit der Übertragung von Kommandos begonnen werden, wobei zwischen zwei Modi unterschieden wird.

- Direkter Modus
  Im direkten Modus wird jedes gesendete Zeichen sofort als möglicher Ein-Byte-Befehl ausgewertet und, falls der Befehl existiert, ausgeführt. Unbekannte Befehle werden ignoriert. Eine Liste der Ein-Byte-Befehle befindet sich im Kapitel 4 Anwendung im Abschnitt 4.3.5.

- Befehls-Modus
  Der Befehls-Modus wird mit dem Zeichen `'>' (0x3e)` eingeleitet. Anschließend hat der Benutzer die Möglichkeit, eine komplette Befehls-Zeichenkette einzugeben, die durch Drücken der Enter-Taste `(0x0c)` abgeschlossen wird. Bei vorzeitigem Abbruch mit Escape `(0x27)` wird das bisher eingegebene verworfen und vorzeitig in den direkten Modus zurück gewechselt. Selbiges passiert bei Eingabe einer ungültigen Zeichenkette. Auch das Format einer solchen Befehls-Zeichenkette wird in Kapitel 4 Anwendung, Abschnitt 4.3.1 genau beschrieben.

Alle sinnvollen Eingaben, mit Ausnahme von Enter und Escape, befinden sich im *lesbaren* ASCII-Bereich, also ASCII-Code `0x20` bis `0x7f`. Werden drei Escape Zeichen hintereinander empfangen, wird unabhängig vom aktuellen Eingabemodus die Ausführung aller Fahrkommandos gestoppt und der Mikrocontroller zurückgesetzt.

3.3.2 I^2C

Die Kommunikation über das I^2C-Interface ist ein wenig komplizierter als die über EIA-232. Da es sich bei I^2C um einen Bus handelt, an dem mehrere Teilnehmer angeschlossen werden können, muss hier zusätzlich eine Adressierung erfolgen. Weitere Informationen zu I^2C lassen sich im Internet[1][2] nachlesen. Das Fahrmodul arbeitet im Slave Modus mit der Adresse `84` bis zu einer Datenrate von 400 kBit/s. Im Gegensatz zur EIA-232 Schnittstelle werden über I^2C keine Debug-Meldungen ausgegeben, dafür hat der Benutzer jedoch die Möglichkeit, gezielt Parameter oder Zählerwerte abzufragen. Alle eingehenden Zeichen im Bereich von `0x00` bis `0xef` werden zunächst als mögliche Ein-Byte-Kommandos interpretiert und dementsprechend ausgewertet. Der Bereich `0xf0` bis `0xff` ist für spezielle Steuercodes reserviert und wird in Abschnitt 4.3 genau erklärt. Es gibt darüber hinaus noch zwei weitere Modi: den zur Befehlszeichenketten-Eingabe und einen zur Bestimmung des als nächstes zu übertragenden Registers. Wurden diese Modi über einen Steuercode angewählt, werden die

nächsten Eingaben entsprechend verwendet. Außerhalb des Ein-Byte-Modus funktionieren die Steuercodes nicht. Auch zur Datenübertragung vom Fahrmodul zum Benutzer werden verschiedene Betriebsarten benötigt. In der Grundbetriebsart bewirkt ein Lesebefehl die Übertragung der aktuellen Registerposition, wobei diese anschließend um eins erhöht wird. Ein sich laufend wiederholender Lesebefehl bewirkt demnach die Übertragung aller Register von 0 bis 255, anschließend beginnt der Zähler erneut bei 0. Durch diese Funktion ist es möglich, die Registerinhalte mit geringem Aufwand komplett zu übertragen, ohne jedesmal die neue Adresse zu übermitteln. Zur gezielten Übertragung lässt sich die Registerposition über einen Steuercode anwählen. Die Liste der benutzen Register befindet sich in Tabelle 4.1. Die anderen Lesemodi dienen zur Übertragung von Befehlszeichenketten aus der Queue und werden in Abschnitt 4.3.6 beschrieben.

3.4 PID-Regler

Das folgende Kapitel benutzt in den Beschreibungen teilweise Ausdrücke der Webseite über Reglungstechnik des Roboternetzes[3]. Dort kann eine detaillierte Darstellung über kontinuierliche und diskrete Regler nachgelesen werden. Hier werden nun die für das Fahrmodul wichtigen Reglerbestandteile erläutert.

Ein Regler dient dazu, eine Ausgangsgröße über ein Stellglied trotz des Einflusses einer Störgöße auf einem Niveau zu halten. Bei dem hier entwickelten Fahrmodul ist die Ausgangsgröße die Fahrgeschwindigkeit und das Stellglied das variable Puls-Pausen-Verhältnis der Pulsweitenmodulation. Die Störgröße ist zum Beispiel eine unterschiedliche Steigung des Fahruntergrundes oder ein verändertes Fahrzeuggewicht durch zusätzliche Aufbauten.

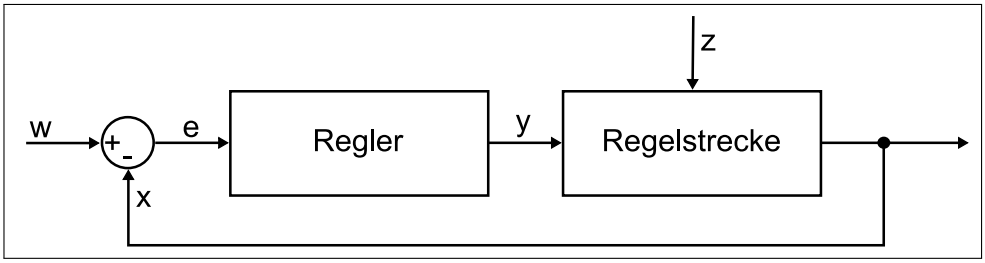

Abbildung 3.3: Blockschaltbild des Regelkreises

Abbildung 3.3 zeigt ein vereinfachtes Blockschaltbild eines Regelkreises. Der dargestellte **Regler** ist der Teil des Regelkreises, der unter Berücksichtigung der dynamischen Eigenschaften der Regelstrecke aus der Regelabweichung die Korrekturmaßnahmen zum Ausregeln ergreift, hier also der in der Software implementierte PID-Regler. Die **Regelstrecke**, der vom Regler auszuregelnde Teil des Kreises, ist bei diesem Fahrmodul der von der PWM angesteuerte Motor mit dem zugehörigen Inkrementalgeber. Über die Führungsgröße bzw. den **Sollwert** w wird die Geschwindigkeit vorgegeben. Die **Regelgröße** bzw. der **Ist-Wert** x stellt die gemessene Geschindigkeit dar und wird zusammen mit w zu $e = w - x$ verrechnet, wobei e **für die Regelabweichung** bzw. der Fehler steht. Die **Stellgröße** y ist der Ausgang des Reglers und repräsentiert mit

einem Wert zwischen 0 und 255 das Puls-Pausenverhältnis für die PWM. Der letzte Parameter z **ist die Störgröße**, die oben bereits erläutert wurde.

Um eine Störung zu erkennen, benötigt der Regelalgorithmus mindestens zwei Eingabewerte. Den Soll-Wert, hier die Geschwindigkeit, die das Fahrzeug haben soll, und den Ist-Wert, die gemessene Geschwindigkeit des Fahrzeugs. Daraus lässt sich durch Subtraktion der momentane Fehler e_k berechnen.

$$e_k = \text{Soll-Wert} - \text{Ist-Wert}$$

Im Folgenden werden nun drei Reglertypen vorgestellt, die zusammengefasst den hier eingesetzten PID-Regler bilden. Da dieser Regler in der Software realisiert wird und zeitdiskret, also nur zu bestimmten Abtastzeitpunkten arbeitet, werden die einzelnen Typen ebenfalls nur diskret betrachtet.

3.4.1 P-Regler, proportional wirkender Regler

Der proportional wirkende P-Regler multipliziert die Regelabweichung (den Fehler) mit seinem Verstärkungsfaktor K_p und gibt das Ergebnis unverzögert an den Ausgang weiter.

$$y_p = K_p \cdot e_k$$

Das Problem bei dem einfachen P-Regler besteht darin, dass der Ausgang bei einer nicht-vorhandenen Regelabweichung ebenfalls 0 wird. Bei dem Fahrzeug würde das bedeuten, dass sich die Motoren beim Erreichen der Soll-Geschwindigkeit abschalten, was wiederum zu einer größer werdenden Regelabweichung führt, die der Regler auszugleichen versucht. Bei einem passend gewählten Zeitintervall für den Regler äußert sich dieses Verhalten darin, dass sich die Ist-Geschwindigkeit meist deutlich unterhalb der Soll-Geschwindigkeit einschwingt und eine Rest-Regelabweichung übrig bleibt, die auch nicht über einen anderen Verstärkungsfaktor eliminiert werden kann. Da der P-Regler sehr schnell arbeitet, ist er bei dem Einsatz im Fahrmodul speziell zum Geradeausfahren sehr gut geeignet. Da kleine Geschwindigkeitsabweichungen schnell zu einer Kurvenfahrt führen, müssen diese unmittelbar ausgeglichen werden. Im Test funktionierte dieses so gut, dass das Fahrzeug auch ohne den in Kapitel 3.5 beschriebenen Differenzenausgleich schon annähernd geradeaus fahren konnte, wenn es auf einem gleichmäßigen Untergrund lief.

3.4.2 I-Regler, integral wirkender Regler

Der integral wirkende Regler summiert die Regelabweichung über die Zeit auf und multipliziert die Summe (bzw. das Integral) mit dem Faktor K_i.

$$y_i = K_i \cdot T_a \cdot \sum_{i=0}^{k} e_i$$

Je länger also eine Regelabweichung besteht, desto größer wird ihr Einfluß auf die Stellgröße. Der I-Regler funktioniert im Vergleich zum P-Regler relativ langsam. Der Einsatz des I-Reglers beim Fahrmodul ist zwingend notwendig, da bei ihm die Rest-Regelabweichung vollständig verschwindet. Das langsame Regelverhalten kann dazu verwendet werden, um eine einfache Anfahrrampe zu realisieren.

3.4.3 PD-Regler, P-Regler mit differentiell wirkendem Anteil

Bei diesem proportional differentiell wirkenden Regler wird der oben bereits erwähnte P-Regler um einen differentiellen Anteil erweitert. Der D-Anteil bewertet die Änderung der Regelabweichung (er „differenziert“) und berechnet so deren Änderungsgeschwindigkeit. Diese wird dann mit dem Faktor K_d multipliziert und auf den P-Anteil addiert.

$$y_p = K_p \cdot e_k$$
$$y_d = \frac{K_d}{T_a} \cdot (e_k - e_{k-1})$$
$$y = y_p + y_d$$

Über die Variable T_a fliesst hier zusätzlich die Abtastgeschwindigkeit mit in die Berechnung ein. Durch sein Verhalten reagiert der PD-Regler bereits auf Ankündigungen von Änderungen, was ein Vorhalten in der Regelung bewirkt. Der differentielle Anteil dieses Reglers ist für das Fahrmodul nicht unbedingt notwendig, dient jedoch mit einem niedrig gewählten Faktor zusätzlich zur Stabilisierung des Regelverhaltens.

3.4.4 P + I + D

Beim PID-Regler sind alle drei Regleranteile kombiniert, wobei jeder Anteil mit einem eigenen Faktor gewichtet wird. Am Ausgang liegt dann die Summe der Anteile an.

$$y = \text{P-Anteil} + \text{I-Anteil} + \text{D-Anteil}$$

$$y = K_p \cdot e_k + K_i \cdot T_a \cdot \sum_{i=0}^{k} e_i + \frac{K_d}{T_a} \cdot (e_k - e_{k-1})$$

Bei der Implementation in Software muss darüber hinaus noch auf eine sinnvolle Begrenzung der Werte geachtet werden. Wenn über einen längeren Zeitraum eine Regelabweichung besteht, wird diese auch permanent zur Fehlersumme addiert. Verschwindet diese Abweichung nun relativ schnell, kann ein unbegrenzter I-Anteil trotzdem sehr lange brauchen, bis die Fehlersumme wieder abgebaut wird. Die Begrenzung bei der Fehlersumme hängt von dem Faktor des I-Anteils ab, und sollte etwa so gewählt werden, dass der I-Anteil annähernd in der Lage ist, den Ausgang vollständig auszusteuern.

Zur Bestimmung der optimalen Werte für K_p, K_i und K_d gibt es unterschiedliche Verfahren, die jedoch den Rahmen dieser Studienarbeit überschreiten würden. Die hier eingesetzten Werte wurden einfach durch Ausprobieren (Empirisches Einstellen) ermittelt. Ein akzeptales Fahrverhalten des Fahrzeugs zeigte sich bei den folgenden Faktoren:

$$K_p = 80$$
$$K_i = 20$$
$$K_d = 10$$

Die Fehlersumme wird dabei bei dem Wert 500 begrenzt.

Zur Realisierung einer Anfahrrampe funktionierten die Werte

$$K_p = 0$$

$$K_i = 5$$

$$K_d = 0$$

$$\text{Fehlersumme} \leq 2000$$

zufriedenstellend. Dem Benutzer steht es natürlich frei, die Parameter optimal anzupassen.

3.5 Geradeausfahrt

In dem Praktikum, für das dieses Fahrmodul in erster Linie entwickelt wurde, wurden vorher andere Plattformen eingesetzt, die aufgrund der Art des Antriebs häufig nicht in der Lage waren, längere Strecken geradeaus zu fahren. Grund dafür waren unter anderem das spannungsabhängige Verhalten der umgebauten Servo-Motoren und deren Ansteuerung. Das im Rahmen dieser Arbeit aufgebaute Fahrmodul benutzt zwei identische Gleichtrom-Getriebemotoren, die beide mit gleichfrequenter Pulsweitenmodulation gesteuert werden. Dennoch treten Gleichlauf-Probleme auf, die durch kleine Abweichungen in der Herstellung oder durch unterschiedliche Belastung entstehen können.

3.5.1 Differenzwert

Das Problem der Geradeausfahrt wird hier in der Software gelöst. Wie bereits erwähnt, sind beide Motoren mit einem Inkrementalgeber ausgestattet, der 360 Impulse pro Umdrehung an den Mikrocontroller schickt, bei dem dadurch ein Interrupt ausgelöst wird. Im Programm existiert nun eine Variable, die bei Vorwärtsdrehung des linken Rades inkrementiert und bei Vorwärtsdrehung des rechten Rades dekrementiert wird (Bei Rückwärtsdrehung werden Inkrementierung und Dekrementierung ebenso vertauscht). Diese Variable repräsentiert also die Differenz von Impulsen zwischen den beiden Rädern. Läuft beispielsweise das linke Rad eine halbe Umdrehung weiter als das rechte, wird der Differenzwert um 180 erhöht. Wenn sich das Fahrzeug nun geradeaus bewegt, wofür sich beide Räder genau gleich schnell drehen müssen, ist der Differenzwert demnach null. Ein von Null verschiedener Wert deutet also auf eine – möglicherweise ungewollte – Kurve hin.

3.5.2 Differenzwert und PID-Regler

Um das Fahrzeug nun doch geradeaus fahren zu lassen, wird ein einfacher Trick angewandt. Im vorherigen Abschnitt wurde der PID-Regler erklärt, welcher aus Soll- und Ist-Geschwindigkeit das korrekte Puls-Pausen-Verhältnis der PWM bestimmt. Der Differenzwert wird für diese Regelung einfach mit der Ist-Geschwindigkeit verrechnet, wobei jeweils die Hälfte der Differenz für eines der beiden Räder benutzt wird.

$$\text{Links:} \quad speed_{diff,left} = speed_{left} + \frac{difference}{2}$$

$$\text{Rechts:} \quad speed_{diff,right} = speed_{right} - \frac{difference}{2}$$

$speed_{diff,l/r}$ ist also die neu berechnete Ist-Geschwindigkeit für den Regelalgorithmus. Am Beispiel wird diese Vorhergehensweise noch einmal deutlich: Das Fahrzeug fährt mit gleicher Soll-Geschwindigkeit für beide Motoren vorwärts. Aus irgendeinem Grund dreht sich das linke Rad minimal schneller und erzeugt mit der daraus resultierenden Rechtskurve einen positiven Differenzwert. Dieser Wert wird nun auf die tatsächliche Geschwindigkeit des Rades addiert und an den Regler weitergegeben (zur Vereinfachung wird die Differenz nur mit einem Rad verrechnet). Aus den übermittelten Werten erkennt der Regelalgorithmus, dass sich das linke Rad nun scheinbar mit zu hoher Geschwindigkeit dreht und verringert deshalb das Puls-Pausen-Verhältnis der Pulsweitenmodulation, um den Motor wieder auf die Soll-Geschwindigkeit zu bringen. Hat der Differenzwert nach diesem Ausgleich wieder den Wert Null erreicht, läuft die Fahrt geradeaus weiter.
Trotz der sehr einfachen Implementierung dieses Verfahrens verliefen die Testfahrten erstaunlich gut. Speziell bei sehr kleinen Abweichungen vom Null-Wert der Differenz ist das Korrekturverhalten bei mittlerer Geschwindigkeit kaum sichtbar. Bei sehr langsamen Tempo bemerkt man meist ruckartige Bewegungen des Fahrzeugs, eine große Differenz zwischen den Radpositionen führt zu einer Kreisfahrt. Da das Fahrzeug mit den eingesetzten Getriebemotoren jedoch selbst ohne Berücksichtigung der Differenz annährend geradeaus fährt und somit nur geringe Abweichungen entstehen, ist der Algorithmus in diesem Fall gut geeignet.

$\longrightarrow$ Zur Verwendung des Fahrzeuges ist dem Benutzer noch mitzuteilen, dass stets darauf geachtet werden muss, den Differenzwert vor Beginn einer Geradeausfahrt manuell auf Null zu stellen, da dieses nicht automatisch geschieht. Ein Vergessen dieser Rücksetzung führt dazu, dass das Fahrzeug nach einer gewollten Kurve mit anschließender Geradeausfahrt sich zunächst in die Richtung drehen wird, die es vor Beginn der Kurve hatte, um die bei der Kurve entstandene Differenz auszugelichen.

4 Anwendung

4.1 DIP-Schalter

Vor der Inbetriebnahme des Fahrmoduls muß über die DIP-Schalter (siehe Abbildung 2.9) eine entsprechende Konfiguration eingestellt werden, wobei dem Benutzer die folgenden Optionen zur Verfügung stehen.

- **DIP 1 - ON**, Steuerung über TWI
  In der ON-Position von DIP-Schalter 1 ist die Kommunikation über das Two-Wire-Interface (I^2C-Bus) aktiviert. Ankommende Daten an der EIA-232 Schnittstelle werden nicht mehr ausgewertet.

- **DIP 1 - OFF**, Steuerung über UART
  Mit dieser Einstellung ist das TWI inaktiv, und die Steuerung funktioniert über die serielle UART-Schnittstelle.

- **DIP 2 - ON**, Debugausgaben über UART
  Bei eingeschaltetem DIP 2 werden über die UART-Schnittstelle Debug-Ausgaben übertragen.

- **DIP 2 - OFF**, Debugausgaben ausgeschaltet
  Keine Debug-Ausgaben über EIA-232.

- **DIP 3 - ON**, LCD angeschlossen
  Ist ein LC-Display mit der Plantine verbunden, so kann es über diese Schnittstelle aktiviert werden.

- **DIP 3 - OFF**, kein LCD angeschlossen
  Bei nicht angeschossenem LC-Display muß dieser Schalter in der OFF-Position sein. Die Motorsteuerung arbeitet bidirektional mit dem Displaycontroller und fragt dessen Busy-Flag ab. Ist kein Display vorhanden, wartet die Software endlos lange auf eine Antwort des Displays und das Programm kann nicht weiterlaufen.

4.2 Rückgabewerte

Der Benutzer hat über das I^2C-Interface die Möglichkeit, verschiedene Werte abzurufen, um eine Rückmeldung vom Fahrmodul zu erhalten. Über welche Kommandos dies funktioniert, wird in Abschnitt 4.3.6 und Abschnitt 3.3.2 erklärt. Hier folgt nun eine Tabelle über den Inhalt der Register. Werden nicht genutzte Register ausgelesen, ist der Rückgabewert `0xff`.

Register	Inhalt
0	In diesem Register steht stets der Wert 0
1	High-Byte (Bit 15-8) der Position des linken Rades
2	Low-Byte (Bit 7-0) der Position des linken Rades
3	High-Byte der Position des rechten Rades
4	Low-Byte der Position des rechten Rades
5	Aktuelle Geschwindigkeit links
6	Aktuelle Geschwindigkeit rechts
7	High-Byte des Differenzwertes
8	Low-Byte des Differenzwertes
9	Betriebsmodus links
10	Betriebsmodus rechts
11	Anzahl der Einträge in der Queue
12	High-Byte des Tachowertes, links
13	Low-Byte des Tachowertes, links
14	High-Byte, Tachowert rechts
15	Low-Byte, Tachowert rechts
16	High-Byte, Tachodifferenz
17	Low-Byte, Tachodifferenz

Tabelle 4.1: Registerübersicht

4.3 Protokoll

Im Folgenden werden die Befehle und deren Syntax beschrieben, die benötigt werden um das Fahrmodul zu steuern. Unterschieden wird zwischen Ein-Byte-Befehlen und Befehlszeichenketten. Beide Arten können sowohl über EIA-232 als auch über I^2C verwendet werden, wobei bei I^2C zusätzlich die Möglichkeit besteht, verschiedene Zähler oder Parameter abzufragen.

4.3.1 Befehlszeichenkette

Eine Befehlszeichenkette setzt sich aus einem bis drei Buchstaben und bis zu vier mit Komma getrennten Parametern zusammen.

```
XYZ,a,b,c,d
```

Die Buchstaben, von denen `Y` und `Z` optional sind, sind in der Regel Großbuchstaben und stellen den Fahrbefehl dar, die optionalen Parameter `a` bis `d` können Zahlen von -32768 bis 32767 sein. Eine Befehlszeichenkette ohne einen Fahrbefehl ist ungültig. Ebenso darf sie nicht mit einem Komma enden. Ist der Wert eines Parameters außerhalb des erlaubten Bereiches, können ungewollte Fehlfunktionen auftreten.

4.3.2 Fahrkommando `D`

Syntax: `DYZ,a,b,c,d`
Das Fahrkommando `D` ist ein grundlegender Befehl, um dem Fahrzeug Bewegungs-

kommandos mitzuteilen. Es lassen sich gleichzeitig für beide Räder die Geschwindigkeit sowie die Trigger einstellen. Durch dieses Kommando erfolgt ein Eintrag in der Befehlskette. Das Kommando wird also erst ausgeführt, wenn vorherige Fahranweisungen abgearbeitet wurden.

- ***Y*: Einstellung für Trigger des linken Rades**
  An der Stelle *Y* können drei verschiedene Buchstaben stehen, die den Trigger des linken Rades steuern. Eine Übersicht findet man in Tabelle 4.2.

- ***Z*: Einstellung für Trigger des rechten Rades**
  Z stellt wie *Y* den Trigger des rechten Rades ein. Es gilt die gleiche Übersichtstabelle 4.2.

Y bzw. Z	Funktion
N	Trigger deaktivieren
T	Zeit-Trigger aktivieren
P	Positions-Trigger aktivieren

Tabelle 4.2: Einstellungen für Trigger

- ***a* bis *d*: Geschwindigkeit und Trigger-Parameter**
  Die bei *a* bis *d* anzugebenen Zahlenwerte dienen zur Einstellung der Geschwindigkeit und der Trigger. Die Funktion und der gültige Wertebereich lassen sich in Tabelle 4.3 ablesen.

Parameter	Beschreibung	Wertebereich
a	Geschwindigkeit, linkes Rad	-127 bis 128
b	Geschwindigkeit, rechtes Rad	-127 bis 128
c	Parameter, Trigger links	siehe Tabelle 4.4
d	Parameter, Trigger rechts	siehe Tabelle 4.4

Tabelle 4.3: Bedeutung der Parameter *a* bis *d*

Die Geschwindigkeit gibt hierbei die Soll-Geschwindigkeit für das jeweilige Rad an. Bei dem Wert 0 wird das Rad aktiv gebremst. Die Funktion des Triggerparameters ist wiederum abhängig vom eingestellten Trigger. Aufgelistet ist dies in folgender Tabelle.

Eingestellter Trigger	Funktion des Parameters c bzw. d
N - kein Trigger	Parameter wird ignoriert
T - Zeit-Trigger	Zeitangabe in 100 Millisekunden, Wertebereich 1 bis 255
P - Positions-Trigger	Angabe der zu fahrenden Ticks, Wertebereich -32768 bis 32768

Tabelle 4.4: Bedeutung der Trigger-Parameter

Bei der Nutzung des Positions-Triggers muss darauf geachtet werden, dass beim Rückwärtsfahren auch eine negative Anzahl von Ticks angegeben werden muss, damit der Trigger ordnungsgemäß funktionieren kann.

4.3.3 Fahrkommando S

Syntax: S*Y*, *a*, *c*
Das Kommando S dient wie das Kommando D zur Fortbewegung des Fahrzeuges, jedoch wird bei S ein spezieller Fahrmodus gewählt, in dem zusätzlich die Differenz zwischen den beiden Rädern berücksichtigt und ausgeglichen wird. Dieser Befehl ist also für die in Abschnitt 3.5 beschriebene Geradeausfahrt da.

Die Möglichkeiten für *Y* unterscheiden sich geringfügig von denen des Fahrkommandos D und sind in Tabelle 4.5 erklärt.

Y	Funktion
N	Trigger deaktivieren
T	Zeit-Trigger aktivieren
P	Positions-Trigger aktivieren
D	Differenzwert einstellen (siehe Text)

Tabelle 4.5: Einstellungen für Trigger beim Kommando S

Für die ersten drei Möglichkeiten, N, T und P ist die Funktion wie beim Kommando D, wobei der Parameter *a* die Geschwindigkeit beider Räder angibt und Parameter *c* den Trigger-Parameter darstellt. Bei der Geradeausfahrt arbeitet ausschließlich der linke Trigger, da durch den Differenzenausgleich beide Räder die gleiche Strecke und Zeit laufen.

Verwendet man nun das Kommando SD, *a*, hat man die Möglichkeit, den Differenzwert einzustellen. Der Parameter *a* gibt an, wie die Differenz verändert werden soll. Ein von Null verschiedener Wert wird zum aktuellen Differenzwert addiert. Wird der Wert Null angegeben, wird die Differenz auf Null zurückgesetzt. Ein zusätzlich angegebener Parameter *b* hat keine Bedeutung und wird ignoriert.

4.3.4 Kommando P

Syntax: P*Y*, *a*, *b*, *c*, *d* bzw. P*YZ*, *a*
Um die Parameter des PID-Reglers zu verändern, nutzt man das Kommando P. Auch dieser Befehl wird in die Befehlskette eingetragen und erst zum richtigen Zeitpunkt ausgeführt. Zusätzlich zu den Parametern P, I und D lassen sich die maximale und die aktuelle Fehlersumme verändern. Welche Angaben dazu erforderlich sind, geht aus den weiteren Tabellen hervor.

Angabe	Funktion
Y	L für linkes Rad, R für rechtes Rad
a	Angabe des P-Faktors
b	Angabe des I-Faktors
c	Angabe des D-Faktors
d	Angabe der maximalen Fehlersumme
Z	S um aktuelle Fehlersumme festzulegen
a	Angabe der aktuellen Fehlersumme

Tabelle 4.6: Parameterbeschreibung für das P-Kommando

Bei alleiniger Angabe von *Y* gilt der obere Block der Tabelle 4.6. Wird für *Z* zusätzlich ein S angegeben, ist nur noch der Parameter *a* erforderlich.

4.3.5 Ein-Byte-Befehle

Zeichen	ASCII-Code	Funktion
<SPC>	32	Fahrzeug anhalten
w	119	Vorwärts fahren
s	115	Rückwärts fahren
a	97	Links drehen
d	100	Rechts drehen
p	112	Pause
P	80	Pause aufheben
<RET>	13	Befehl abbrechen
j	106	Jobausführung anhalten
J	74	Jobausführung beginnen
k	107	Befehlskette löschen
t	116	Tacho zurücksetzen

Tabelle 4.7: Befehlsübersicht der Ein-Byte Befehle

Die Tabelle 4.7 zeigt eine Übersicht aller Ein-Byte Befehle. Die Ein-Byte Befehle sind im wesentlichen vorprogrammierte, häufig gebrauchte Befehle, die auch über die bereits erläuterten Befehlszeichenketten eingegeben werden können.

- **<SPC>, ASCII: 32, Fahrzeug anhalten**
  Dieser Befehl löscht die Befehlskette, beendet noch laufende Befehle und hält das Fahrzeug unmittelbar an.

- **w, ASCII: 119, Vorwärts fahren**
  Die Befehlskette wird gelöscht, laufende Befehle werden abgebrochen und die PID-Parameter werden auf die Standardwerte 80, 20, 10, 500 gesetzt. Anschließend wird das Kommando *'DNN,40,40,0,0'* in die Befehlskette geschrieben und die Jobausführung gestartet. Das Kommando bewirkt eine Vorwärtsfahrt mit Tempo 40.

- **s, ASCII: 115, Rückwärts fahren**
  Genau wie **w**, jedoch mit *'DNN,-40,-40,0,0'* als Kommando, was der Rückwärtsfahrt mit Tempo 40 entspricht.

- **a, ASCII: 97, Links drehen**
  Ebenfalls wie **w**, wobei das Kommando *'DNN,-40,40,0,0'* eine Linksdrehung bewirkt.

- **d, ASCII: 100, Rechts drehen**
  Rechtsdrehung durch den Befehl *'DNN,40,-40,0,0'*.

- **p, ASCII: 112, Pause**
  Durch Aufruf dieses Kommandos wird die aktuelle Befehlsausführung angehalten und das Fahrzeug bleibt stehen. Ein möglicherweise aktiver Zeit-Trigger wird ebenfalls gestoppt.

- **P, ASCII: 80, Pause aufheben**
  Nach einem *Pause*-Kommando wird hiermit die Fahrt fortgesetzt. War ein Zeittrigger aktiv, wird dieser ab der unterbrochenen Stelle weiterlaufen.

- **j, ASCII: 106, Jobausführung anhalten**
  Wird die Jobausführung angehalten, führt das Fahrzeug keine weiteren Befehle aus der Queue aus. Ein derzeit laufender Befehl wird zu Ende ausgeführt.

- **J, ASCII: 74, Jobausführung beginnen**
  Nach Beginnen der Jobausführung wird das nächste, sich in der Befehlskette befindende Fahrkommando, ausgeführt. Die Ausführung startet automatisch, falls eines der Fahrkommandos **w**, **s**, **a**, **d** oder <**SPC**> gewählt wurde, da diese intern über die Queue arbeiten und sonst eventuell nicht sofort starten könnten.

- **k, ASCII: 107, Befehlskette löschen**
  Zum Löschen der Queue wird dieser Befehl benötigt. Es werden nur noch nicht bearbeitete Kommandos entfernt, ein derzeit laufendes wird nicht unterbrochen.

- **t, ASCII: 116, Tacho zurücksetzen**
  Der Tickzähler beider Räder und der Differenzwert werden auf null zurückgesetzt. Dieser Reset hat keine Auswirkung auf die intern verwendeten Werte für Strecke und Differenz.

4.3.6 I²C-Steuercodes

Sollen über den I²C-Bus andere als die Ein-Byte-Befehle übertragen werden, sind spezielle Steuerzeichen notwendig. Auch dienen diese dazu, dem Programm mitzuteilen, welche Werte nach einem Lesebefehl gesendet werden sollen. Die Codes liegen im Bereich `0xf0` bis `0xff` und werden nach der Übersicht in Tabelle 4.8 noch einmal genau beschrieben.

Code	Funktion
`0xf0`	Registerabfrage
`0xf1`	Ausgabe des aktuellen Kommandos
`0xf2`	Ausgabe der Queue
`0xf3`	Übertragung eines Kommandos
`0xff`	Controller-Reset

Tabelle 4.8: I^2C-Steuercodes

- **`0xf0`, Registerabfrage**
  Nach `0xf0` muss zusätzlich noch eine Registernummer zwischen `0x00` und `0xff` gesendet werden. Dadurch wird festgelegt, welches Register nach einem Lesebefehl übertragen werden soll. Eine Liste der Registerinhalte zeigt Tabelle 4.1. Unabhängig vom vorherigen Ausgabemodus schaltet dieser Befehl stets auf die Registerausgabe.

  I^2C Startbit → I^2C Adresse + W → **`0xf0`** → *Registernummer* → I^2C Stoppbit

  I^2C Startbit → I^2C Adresse + R → *Daten lesen* → I^2C Stoppbit

- **`0xf1`, Ausgabe des aktuellen Kommandos**
  Welches Kommando zur Zeit aktiv ist, kann über diesen Befehl ausgegeben werden. Dabei bewirkt `0xf1` eine Umschaltung des Ausgabemodus. Wird anschließend des Lesebefehl ausgeführt, erfolgt eine byteweise Übertragung einer lesbaren Zeichenkette, die mit ASCII-Code `0x00` abgeschlossen wird. Danach schaltet der Ausgabemodus wieder auf die Registerausgabe zurück. Sollte kein Kommando aktiv sein, wird der Code `0xee` übertragen.

  I^2C Startbit → I^2C Adresse + W → **`0xf1`** → I^2C Stoppbit

  I^2C Startbit → I^2C Adresse + R → *Kommando lesen* → I^2C Stoppbit

- **`0xf2`, Ausgabe der Queue**
  Ähnlich wie bei `0xf1` bewirkt `0xf2` nun die Ausgabe der gesamten Befehlskette. Nach jedem einzelnen Kommando wird ein `0x0c` gesendet, was dem Benutzer ermöglicht, den Lesevorgang nach einen bestimmten Anzahl von Kommandos abzubrechen. Nach dem letzten Kommando wird auch hier ein `0x00` übertragen, eine leere Queue gibt `0xee` zurück.

  I^2C Startbit → I^2C Adresse + W → **`0xf1`** → I^2C Stoppbit

  I^2C Startbit → I^2C Adresse + R → *Queue lesen* → I^2C Stoppbit

- **`0xf3`, Übertragung eines Kommandos**
  Dieser Code funktioniert analog zum `'>'`-Zeichen bei EIA-232 und schaltet in den Befehls-Modus. Anschließend übertragene Zeichen werden als Bestandteil einer Befehlszeichenkette gesehen und als solche ausgewertet. Die Befehlszei-

chenkette wird durch `0x00` abgeschlossen (also ein Null-terminierter String) und der Modus aufgehoben.

I²C Startbit → I²C Adresse + W → **`0xf3`** → *Befehlszeichenkette übertragen* → I²C Stoppbit

- **`0xff`, Controller-Reset**
  Durch Übertragung von `0xff` wird ein Reset des Mikrocontrollers ausgelöst.

  I²C Startbit → I²C Adresse + W → **`0xff`** → I²C Stoppbit

5 Zusammenfassung und Ausblick

In dieser Studienarbeit wurde ein Fahrmodul entwickelt, welches aus zwei Getriebemotoren besteht, die von einer Steuerplatine geregelt werden. Das Kernelement hierbei ist ein Atmel AVR Mikrocontroller, welcher über einen Motortreiber direkt mit den Motoren verbunden ist und sie mit Hilfe einer Puls-Weiten-Modulation steuert. Die an den Motoren angekoppelten Inkrementalgeber geben dem Mikrocontroller die nötige Rückmeldung des Motorverhaltens, um einen abgeschlossenen Regelkreis zu erhalten.

Zur Befehlsübertragung von einer möglichen Hauptplatine ist ein I^2C Interface implementiert. Die Befehle sind so aufgebaut, dass ein Benutzer ohne spezielle Kenntnisse der Regelungstechnik das Fahrzeug lenken kann. Statusausgaben sind entweder über ein direkt anschließbares LC-Display oder eine weitere EIA-232 Schnittstelle möglich. Über einen 4fach DIP-Schalter kann die gewünschte Betriebsart gewählt werden. Zur Stromversorgung besitzt das Fahrmodul einen 12V Bleiakku, dessen Spannung auf der Platine zum Betrieb des Mikrocontrollers heruntergeregelt wird.

Ein wesentliches Merkmal dieser Fahrplattform ist die Möglichkeit, besonders genau geradeaus zu fahren. Hierzu wird in der Software die Differenz zwischen beiden Rädern ermittelt und damit die Geschwindigkeit der Motoren angepasst.

Zur Weiterentwicklung bietet das Fahrmodul sehr gute Grundlagen. Von dem verfügbaren 256k-Flash-Programmspeicher des Mikrocontrollers sind derzeit fünf Prozent belegt, Variablen belegen 25 Prozent des 2k-SRAM. Zusätzlich zu den verwendeten Anschlüssen sind 2 8 Bit Ports des Controllers nach draußen geführt, die frei programmiert werden können.

Ideen für weiterführende Entwicklung sind beispielsweise fest implementierte Fahrmuster, die über einfache Kommandos direkt abgerufen werden können. Über die freien Ports könnten Sensoren angeschlossen werden, die dem Fahrmodul schnellere Reaktionen auf bestimmte Ereignisse liefern. Auch die Möglichkeit weiterer Sensoren zur Lagebestimmung wäre denkbar, um das Fahrzeug noch präziser lenken zu können.

Literaturverzeichnis

[1] I^2C. In: Wikipedia, Die freie Enzyklopädie.
Bearbeitungsstand 21. Jan. 2008, 12:39
http://de.wikipedia.org/wiki/I2C
(Abgerufen: 4. Feb. 2008, 11:53)

[2] Philips Semiconductors: The I^2C-Bus Specification. Version 2.1, January 2000
http://www.nxp.com/acrobat/literature/9398/39340011.pdf
(Abgerufen: 31. Jan. 2008, 13:17)

[3] Regelungstechnik. In: Roboter Netz-Wissen, ein MediaWiki
Bearbeitungsstand 20. Aug. 2007, 13:17
http://www.roboternetz.de/wissen/index.php/Regelungstechnik
(Abgerufen: 31. Jan. 2008, 15:33)

[4] Atmel: 8-bit AVR Microcontroller with 64k/128k/256k Bytes
In-System Progammable Flash. Revision 2549L-08/07
http://www.atmel.com/dyn/resources/prod documents/doc2549.pdf
(Abgerufen: 31. Jan. 2008, 15:59)

[5] ST Microelectronics: L298 Dual Full-Bridge Driver. January 2000
URL: http://www.st.com/stonline/products/literature/ds/1773.pdf
(Abgerufen: 4. Feb. 2008, 11:42)

A Anhang

A.1 Schaltplan

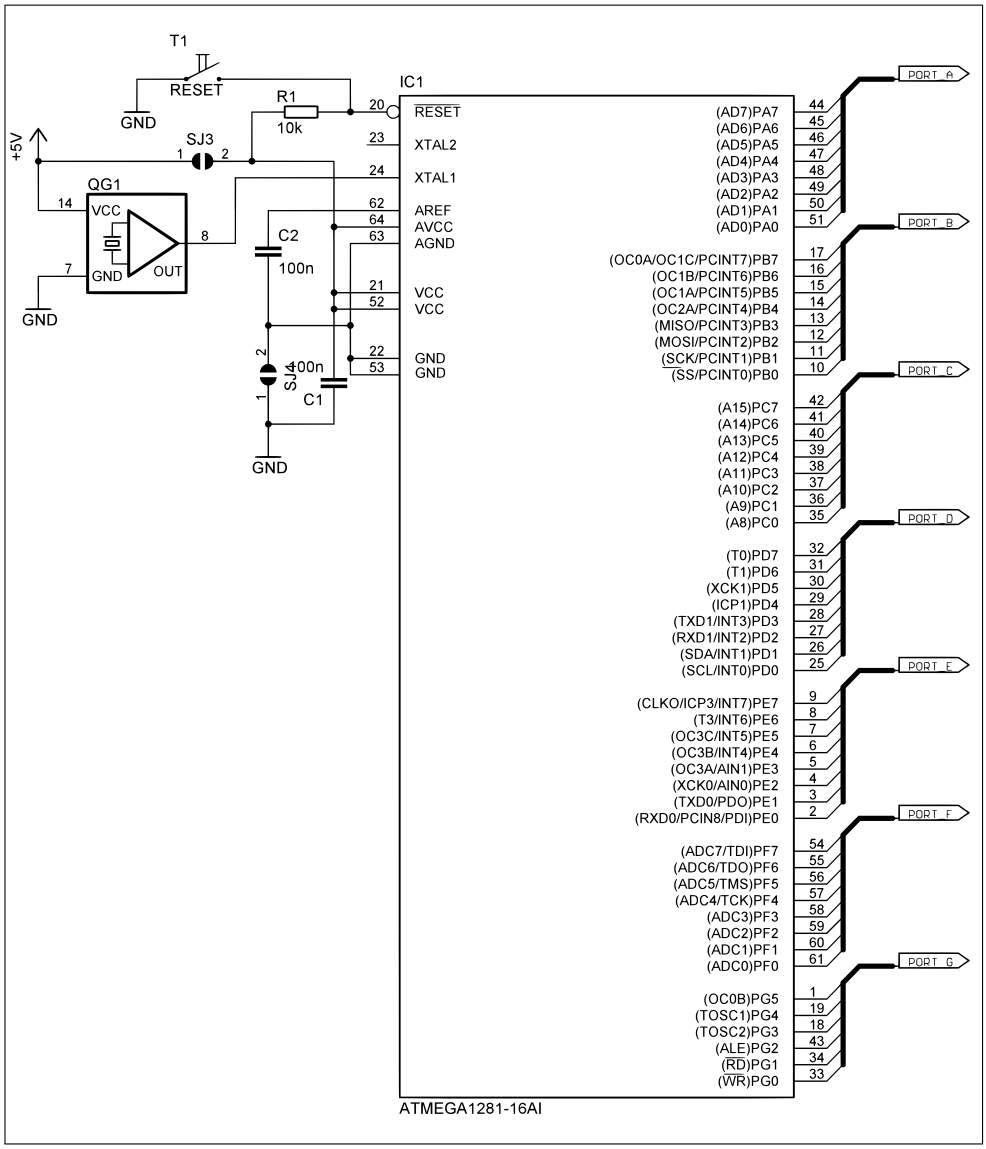

Abbildung A.1: Schaltplan, Mikrocontroller

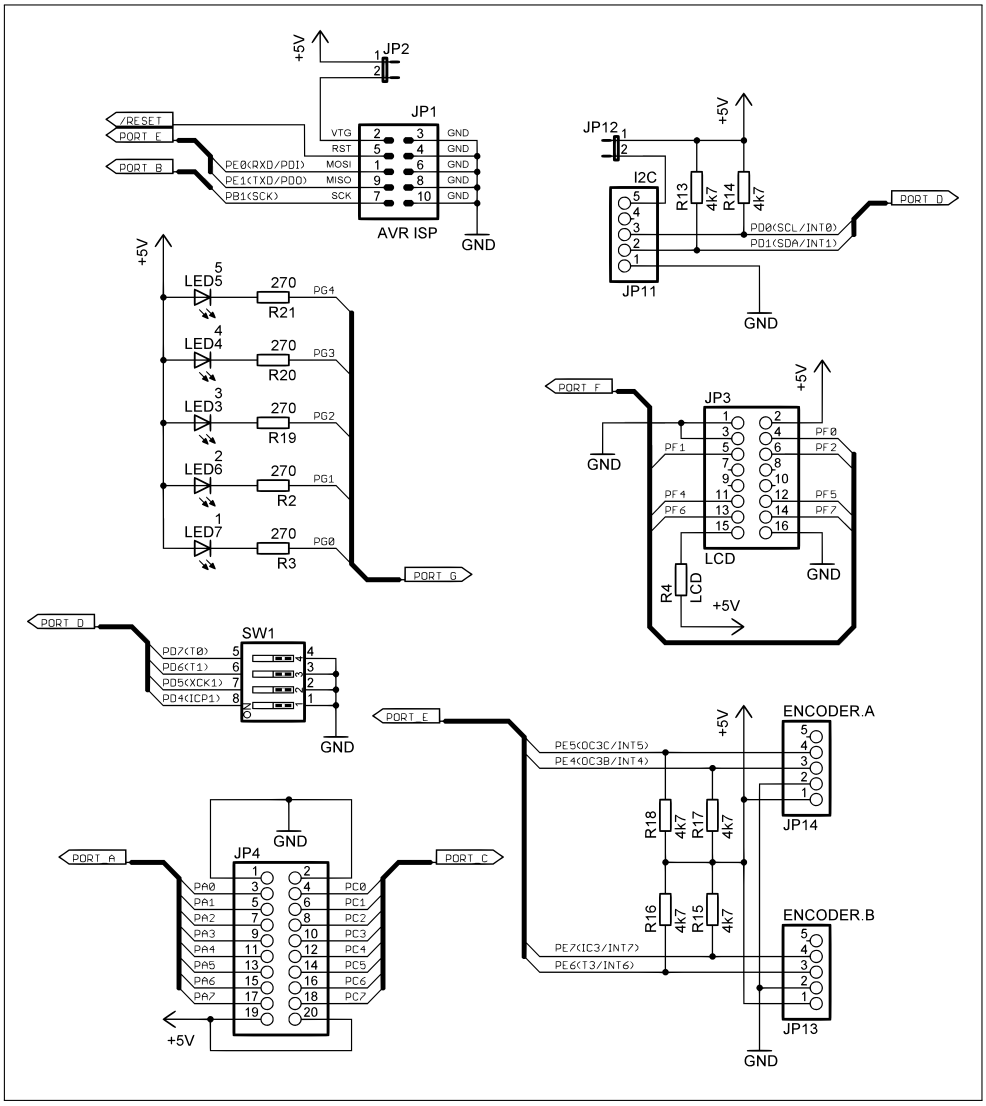

Abbildung A.2: Schaltplan, Verbindungen

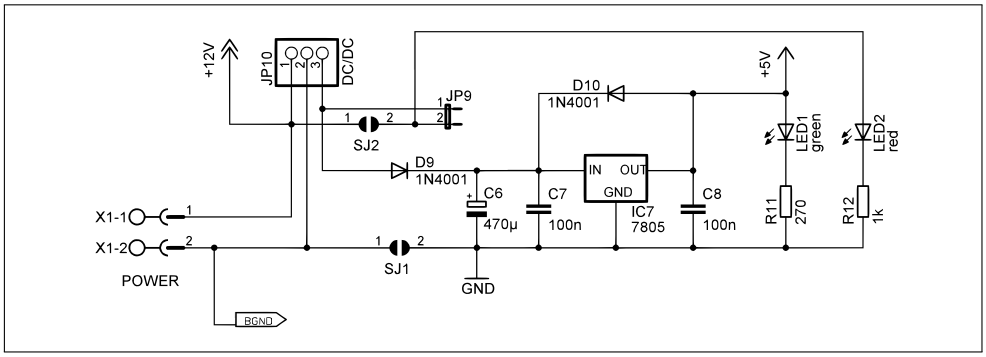

Abbildung A.3: Schaltplan, Stromversorgung

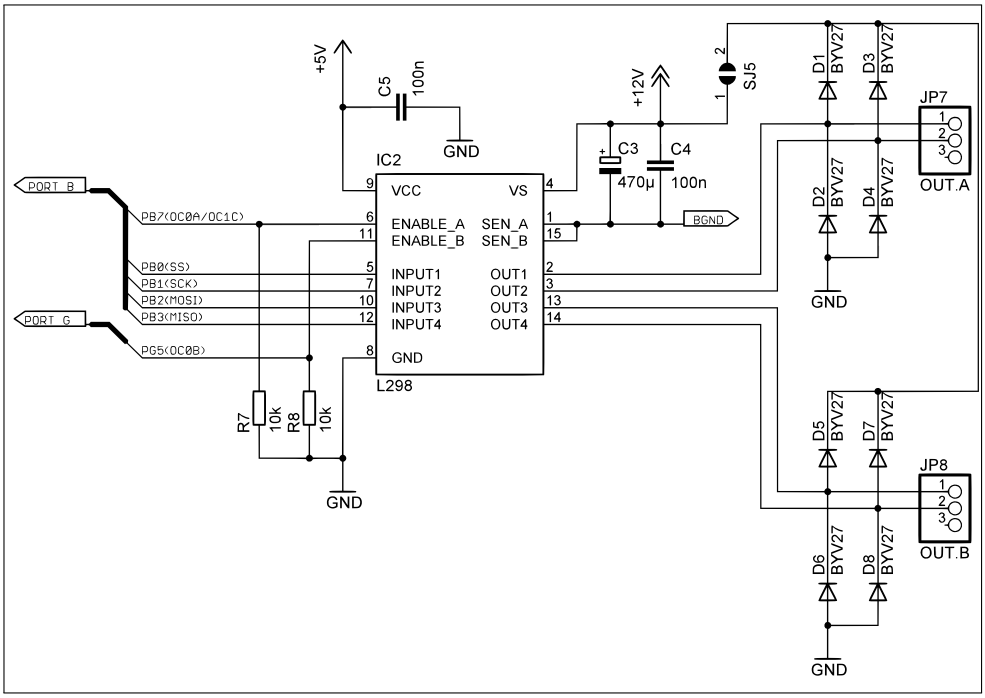

Abbildung A.4: Schaltplan, Motortreiber

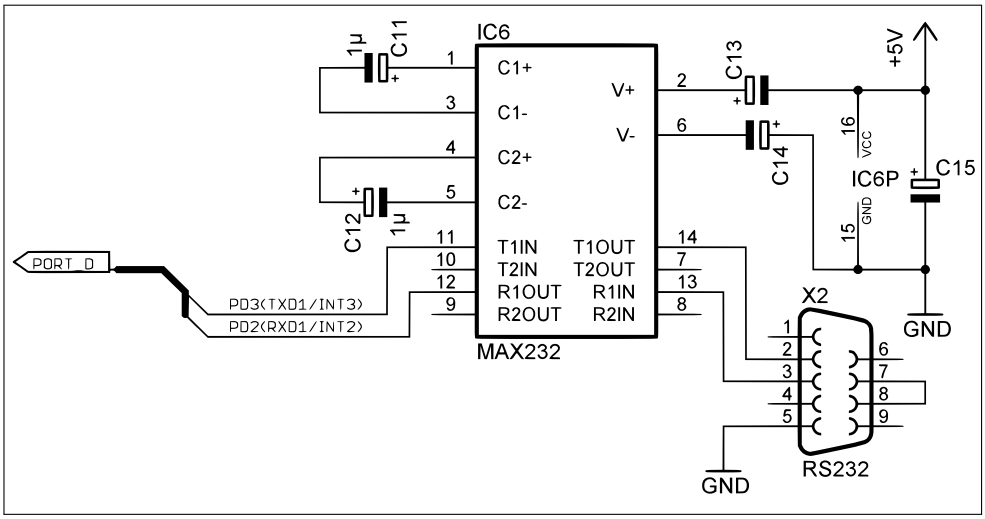

Abbildung A.5: Schaltplan, Pegelumsetzer

A.2 Layout

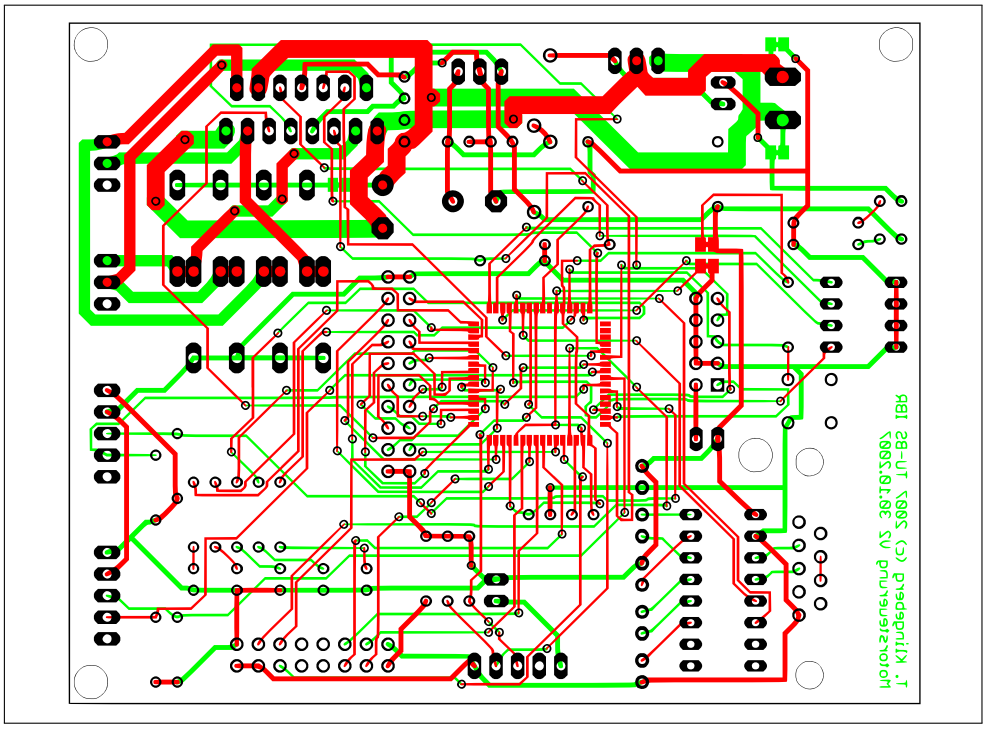

Abbildung A.6: Löt- und Bestückungsseite

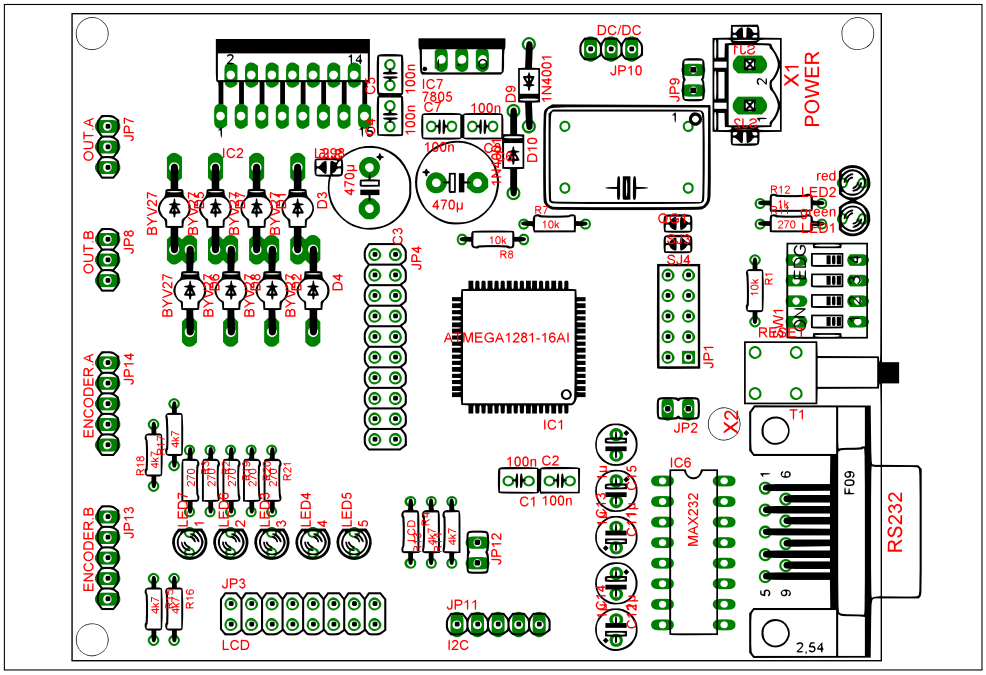

Abbildung A.7: Bauteile-Ansicht

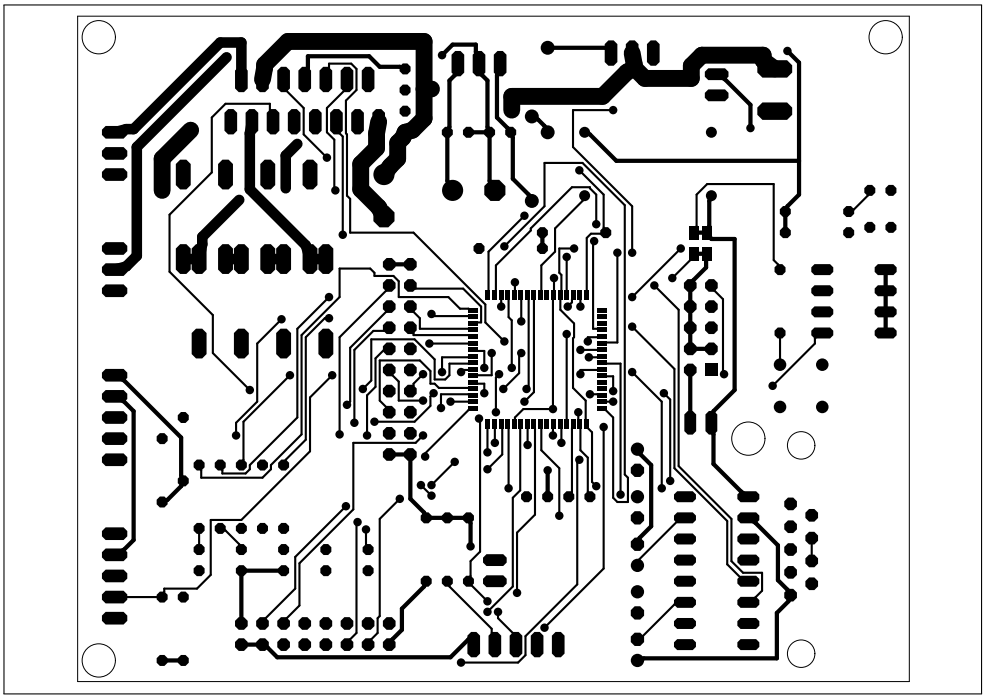

Abbildung A.8: Bestückungsseite

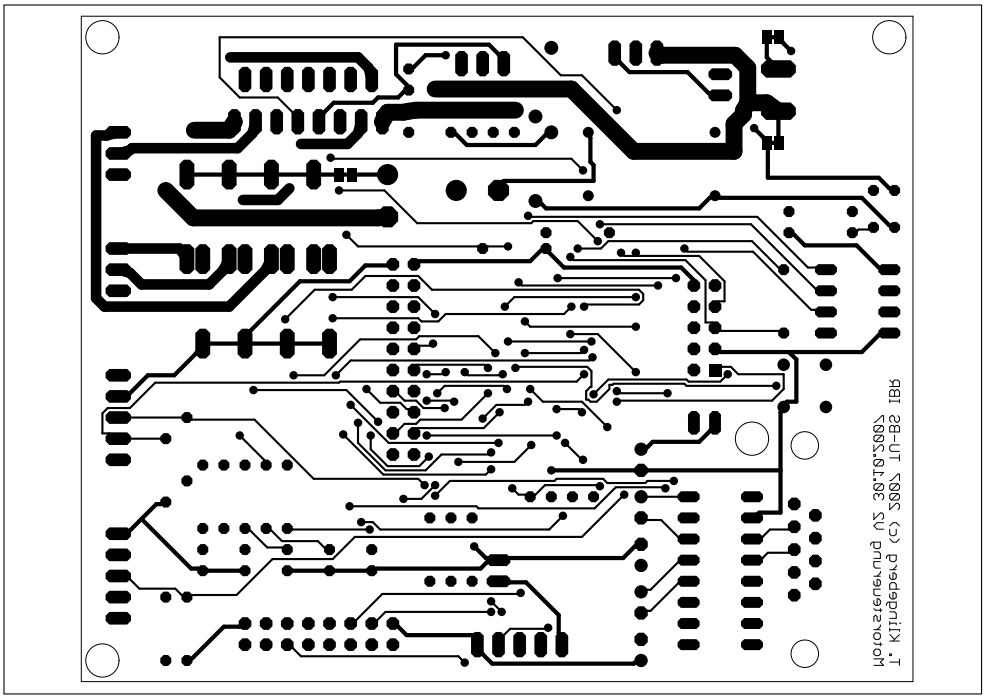

Abbildung A.9: Lötseite

A.3 Teileliste

Kondensatoren		
C1	100nF	Keramik-Kondensator, Vielschicht, 10%
C2	470μF	Elko, radial, 16V
C3	100nF	Keramik-Kondensator, Vielschicht, 10%
C4	100nF	Keramik-Kondensator, Vielschicht, 10%
C5	100nF	Keramik-Kondensator, Vielschicht, 10%
C6	470μF	Elko, radial, 16V
C7	100nF	Keramik-Kondensator, Vielschicht, 10%
C8	100nF	Keramik-Kondensator, Vielschicht, 10%
C9	1μF	Elko, radial, 16V
C10	1μF	Elko, radial, 16V
C11	1μF	Elko, radial, 16V
C12	1μF	Elko, radial, 16V
C13	1μF	Elko, radial, 16V
Dioden		
D1	1N5158	Schottky Diode
D2	1N5158	Schottky Diode
D3	1N5158	Schottky Diode
D4	1N5158	Schottky Diode
D5	1N5158	Schottky Diode
D6	1N5158	Schottky Diode
D7	1N5158	Schottky Diode
D8	1N5158	Schottky Diode
D9	1N4001	Standard Diode
D10	1N4001	Standard Diode
ICs		
IC1	L298	Motortreiber
IC2	ATmega2561	Atmel AVR Mikrocontroller
IC3	L7805	Spannungsregler, 1A positiv, TO-220
IC4	MAX232	Pegelwandler, DIL-16
Quartz		
QG1	16MHz	Quartz Oszillator, DIL-14
Schalter, Taster		
SW1	DIP-Switch	4pol DIP Schalter
T1	Reset	Reset Taster, liegend

Tabelle A.1: Stückliste, Teil 1

Anschlüsse und Jumper		
JP1	OUT.A	1x3 Pins, 2.54mm
JP2	ISP POWER	Jumper
JP3	AVR-ISP-10	2x5 Pins, 2.54mm
JP4	OUT.B	1x3 Pins, 2.54mm
JP5	I2C POWER	Jumper
JP6	I2C	1x5 Pins, 2.54mm
JP7	DC/DC	1x3 Pins, 2.54mm
JP8	DC	Jumper
JP9	ENCODER.A	1x5 Pins, 2.54mm
JP10	LCD	2x8 Pins, 2.54mm
JP11	PORT A C	2x10 Pins, 2.54mm
JP12	ENCODER.B	1x5 Pins, 2.54mm
X1	POWER	2pol Steckverbinder, Phoenix
X2	RS232	9pol SUB-D Buchse, gewinkelt
Leuchtdioden		
LED1	blau	Standard LED, 3mm
LED2	grün	Standard LED, 3mm
LED3	gelb	Standard LED, 3mm
LED4	rot	Standard LED, 3mm
LED5	rot	Standard LED, 3mm
LED6	grün	Standard LED, 3mm
LED7	rot	Standard LED, 3mm
Widerstände		
R1	10kΩ	1/4W, 5%
R2	10kΩ	1/4W, 5%
R3	10kΩ	1/4W, 5%
R4	270Ω	1/4W, 5%
R5	270Ω	1/4W, 5%
R6	270Ω	1/4W, 5%
R7	4.7kΩ	1/4W, 5%
R8	4.7kΩ	1/4W, 5%
R9	270Ω	1/4W, 5%
R10	270Ω	1/4W, 5%
R11	270Ω	1/4W, 5%
R12	1kΩ	1/4W, 5%
R13	4.7kΩ	1/4W, 5%
R14	4.7kΩ	1/4W, 5%
R15	4.7kΩ	1/4W, 5%
R16	4.7Ω	1/4W, 5%
R17	10Ω	1/4W, 5%, Abhängig vom LC-Display

Tabelle A.2: Stückliste, Teil 2