Fachhochschule Wedel, Fachbereich Technische Informatik
Labor für Robotik, Bildverarbeitung und CIM-Anwendungen

Diplomarbeit

MARVIN

Mobiler **A**utonomer **R**oboter **V**on **I**Is **N**euentwickelt

Aufbau eines rechnergesteuerten Fahrzeuges
Mechanik und Antrieb

Henry G. Kleta
ii6582

Betreuer: Prof. Dr. Uelzmann

2.7.1997

Gewidmet all jenen,
die mich auf meinem bisherigen Weg
begleitet und unterstützt haben.

Seid bedankt...

Eidesstattliche Erklärung

Hiermit versichere ich, Henry G. Kleta, geboren am 12.4.71 in Hamburg, an Eides statt, diese Diplomarbeit selbständig und ohne fremde Hilfe durchgeführt zu haben.

Hamburg, den 2.7.1997

Henry G. Kleta

Inhaltsverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] Sven Heine, Diplomarbeit "Implementierung eines Modulkonzeptes für ein rechnergesteuertes Fahrzeug", FH-Wedel, 6/97

[2] Matthias Wittern, Diplomarbeit "Implementierung optischer Sensor- und Anzeigeelemente für ein rechnergesteuertes Fahrzeug", FH-Wedel, 6/97

[3] Fabian Schnell, Diplomarbeit "Analyse- und Entwicklungsprogramm für ein rechnergesteuertes Fahrzeug", FH-Wedel, 6/97

[4] Prof. Dr. Stenzel, Vorlesung "Regelungstechnik 1", FH-Wedel, WS 96/97

[5] Prof. Dr. Stenzel, Vorlesung "Regelungstechnik 2", FH-Wedel, WS 96/97

[6] Firma Faulhaber Motoren, 96'er Katalog, Schönaich, 1996

[7] Hans-Jörg Himmeröder, "Raubkatze Einplatinenrechner KAT-Ce 68332, Teil 1: Hardware", ELRAD Heft3, 1994

[8] Otmar Feger, MC-Tools 3: Bausteine und Applikationen, Feger+Reith, 1990, Traunstein

[9] KSC Software Systems, System51 User Manual, Release 3.1, DK-Ballerup

1 Einleitung

Ein Gebiet der Robotik ist der Bereich der rechnergesteuerten Fahrzeuge. Bei ihnen handelt es sich um kleine, autonome Systeme, die in der Lage sind, die unterschiedlichsten Aufgaben zu erfüllen.

Es geht primär darum, die Steuerung eines Fahrzeuges aus den Händen des Menschen heraus in die Verantwortung eines Rechners zu legen. Dabei soll der Mensch entlastet werden, und kann somit für komplexere Aufgaben zur Verfügung stehen.

Die Steuerung von Fahrzeugen hat meist keinen hohen Anforderungscharakter, da diese in den meisten Fällen nach einem sich wiederholenden Schema geschieht. Deswegen soll nach Möglichkeit ein Rechner mit dieser Aufgabe betraut werden. Ein Rechner reagiert wesentlich schneller auf Umwelteinflüsse als der Mensch. Ausserdem bietet ein Rechner eine konstante Leistung an. Er ermüdet nicht, und ist absolut frei von menschlichen "Schwächen" (er muss z.B. nie auf die Toilette...). Die Reaktion eines Rechners auf gleiche Umwelteinflüsse ist immer gleich. Er lässt sich nicht durch z.B. eine schöne Landschaft, Werbebotschaften, oder sogar Gefühle ablenken, was beim Menschen immer wieder zu beobachten ist.

Der grösste Vorteil des Menschen als Steuereinheit hingegen ist seine Flexibilität. Er verfügt über eine relativ hohe Intelligenz, sowie über die flexibelsten Werkzeuge die es gibt - seine Hände - welche es ihm erlauben, die unterschiedlichsten Aufgaben zu erfüllen. Im besonderen in Bezug auf die Fortbewegung ist der Mensch im Vorteil. Er kann in den unterschiedlichsten Geschwindigkeiten gehen, springen, schwimmen, klettern, um nur ein paar der Fortbewegungsmöglichkeiten zu nennen. Durch diese Flexibilität ist der Mensch nicht auf einen Einsatzort festgelegt.

Der grosse Nachteil des Menschen ist aber, dass er als Steuereinheit sehr anspruchsvoll ist, was den Einsatzort angeht. Ein Rechner als Steuereinheit kann genau für den zukünftigen Einsatzort des Fahrzeuges gebaut werden. Es ist mit entsprechendem Aufwand möglich, ein Fahrzeug für extreme Temperaturen, Drücke und Atmosphären zu entwickeln. Um einen Menschen an solchen, für ihn oft tödlichen Umgebungen am Leben zu erhalten, ist ein weitaus grösserer Aufwand notwendig. Bei dem Einsatz eines Menschen als Steuereinheit ist auch die Grösse des Menschen zu betrachten. Ein Rechner lässt sich sehr klein gestalten, woraus resultiert, dass auch das Fahrzeug kleiner, und somit leichter sein kann.

Ein zusätzliches Problem bei dem Einsatz von Menschen als Steuereinheit ist der Aspekt des sozialen Umfeldes. So macht z.B. Einsamkeit über Monate oder sogar Jahre dem Rechner nichts aus. Manch ein Mensch ist unter solchen Bedingungen schon Amok gelaufen, wodurch er nicht gerade dafür prädestiniert ist, solch eine Mission als Steuereinheit zu begleiten.

Gerade auf dem Gebiet der Exploration, also der Erkundung von unbekannten Gegenden, werden verstärkt rechnergesteuerte Fahrzeuge eingesetzt. Bei der Exploration geht es hauptsächlich um das Sammeln von Daten. Diese können von dem Rechner nach einem bestimmten Algorithmus gesammelt und z.B. über Funk an nahezu jeden Ort übertragen werden, wo diese dann von dem Menschen ausgewertet werden können. Speziell in der Raumfahrt finden unbemannte Sonden Anwendung, da gerade im All extrem lebensfeindliche Bedingungen vorzufinden sind.

Ein rechnergesteuertes Fahrzeug besteht aus mehreren Komponenten. Das Kernstück eines solchen Fahrzeuges ist die Steuereinheit, welche die Aufgaben der Kommunikation zwischen den einzelnen Komponenten, deren Ansteuerung sowie die Auswertung (eventuell auch Speicherung und / oder Übertragung) der Daten von den einzelnen Komponenten übernimmt.

Eine weitere, nicht minder wichtige Komponente ist die Antriebseinheit. Von ihrer Gestaltung hängt es ab, wie sich das Fahrzeug bewegen kann. Die Möglichkeiten der Gestaltung sind nahezu unbegrenzt, ist jedoch eine Form gegeben, unterliegt das Fahrzeug gewissen Einschränkungen. So ist z.B. ein schwimmendes Fahrzeug auf trockenem Untergrund wortwörtlich auf dem Trockenen und somit auf ziemlich verlorenem Posten.

Damit Daten von der Umwelt erfasst werden können, ist eine Sensorikeinheit notwendig, die so gestaltet werden kann, dass nur die relevanten Daten erfasst werden. Prinzipiell sind auch der Gestaltung der Sensorikeinheit kaum Grenzen gesetzt.

Damit das Fahrzeug mit der für den Betrieb notwendigen Energie versorgt wird, ist eine Stromversorgung notwendig. Oft reicht eine externe Stromversorgung aus, die über Kabel das Fahrzeug mit der notwendigen Energie versorgt. Um eine absolute Unabhängigkeit von Kabelverbindungen zu erreichen, muss eine Energieversorgung auf dem Fahrzeug selbst vorgesehen werden. Hierzu bieten sich Batterien an, die jedoch geladen werden müssen, damit das Fahrzeug nicht nur einen Entladezyklus der Batterien als Betriebszeit hat. Gerade in der Raumfahrt werden zum Laden der Batterien meist Solarzellen verwendet, da im All ausreichend viel Sonnenenergie zur Verfügung steht.

1.1 Konzept von MARVIN

Für das Labor für Robotik, Bildverarbeitung und CIM-Anwendungen des Fachbereiches Technische Informatik der Fachhochschule Wedel soll ein rechnergesteuertes Fahrzeug entwickelt werden.

Bei der Erstellung des Fahrzeuges sollen die einzelnen, für das Fahrzeug notwendigen Komponenten, sinnvoll entwickelt und aufgebaut werden. Das fertige Fahrzeug soll als Grundlage für eine Kleinserie dienen. Die fertigen Fahrzeuge dieser Kleinserie sollen für Praktikumszwecke an der Fachhochschule Wedel eingesetzt werden. Im Rahmen dieser Praktika wird zukünftigen Ingenieuren die Möglichkeit geboten, die unterschiedlichsten Konzepte der Robotik, der Bildverarbeitung und der Prozessprogrammierung am praktischen Beispiel zu erlernen. Damit diese Praktika flexibel gestaltet werden können, muss auch das Fahrzeug anpassbar sein. Aus diesem Grunde ist das Fahrzeug modular aufzubauen.

1.2 Übersicht Gesamtprojekt

Die Implementierung des Prototypen des Fahrzeuges wurde im Rahmen von vier Diplomarbeiten umgesetzt, die sich mit jeweils einer Komponente des Fahrzeuges beschäftigten.

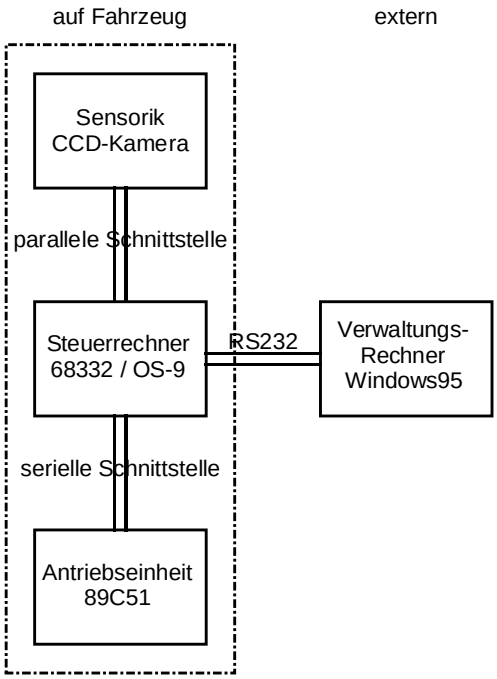


Abbildung 1.1: Übersicht Gesamtprojekt

Als Testaufgabe, mit der die gemeinsame Funktion der Komponenten nachgewiesen wird, soll in einer homogenen, weissen Umgebung, ein senkrechter, schwarzer Stab gefunden werden. Hierzu wird im Rahmen der Diplomarbeit, welche sich mit der Sensorik beschäftigt, ein Programm zur Ansteuerung des Fahrzeuges erstellt.

1.2.1 Steuerrechner

Der Steuerrechner übernimmt die Aufgabe der Ansteuerung der einzelnen Komponenten, sowie die Kommunikation zwischen diesen. Zusätzlich wird auf dem Steuerrechner ein Modulkonzept implementiert, welches als Betriebssystem für das Fahrzeug angesehen werden kann.

Die Aufgabe des Steuerrechners übernimmt ein vorgefertigter Einplatinenrechner auf Basis eines 68332 Prozessors mit dem Betriebssystem OS-9.

Die Lösung dieser Teilaufgabe wurde in der Diplomarbeit "Implementierung eines Modulkonzeptes für ein rechnergesteuertes Fahrzeug" (siehe [1]) bearbeitet.

1.2.2 Mechanik und Antrieb

Der mechanische Grundaufbau des Fahrzeuges entscheidet über spätere Eigenschaften des Fahrzeuges. Er muss deswegen dem zukünftigen Einsatzgebiet des Fahrzeuges angepasst sein. Ebenso verhält es sich mit dem Aufbau der Antriebseinheit.

Um dem geforderten, modularen Konzept gerecht zu werden, soll die Antriebseinheit über eine eigene, lokale Intelligenz verfügen, die die Ansteuerung der Antriebe übernimmt.

Die Antriebseinheit wird mit einem 8051 kompatiblen Mikrocontroller ausgestattet, der über eine serielle Schnittstelle mit dem Steuerrechner verbunden ist. Die Antriebseinheit erhält Befehle von dem Steuerrechner, die decodiert, quittiert und dann ausgeführt werden. Desweiteren übernimmt die Antriebseinheit die Aufgaben der Ansteuerung und Regelung der Antriebe.

Die Lösung dieser Teilaufgabe wurde in dieser Diplomarbeit bearbeitet.

1.2.3 Sensorik und Anzeigeelemente

Die Sensorikeinheit ist ein notwendiges Modul, um der geforderten Autonomie des Fahrzeuges gerecht zu werden. Die Sensorik befähigt das Fahrzeug auf Umwelteinflüsse reagieren zu können. Dieses ist immer dann notwendig, wenn der Ablauf des Fahrzeugprogrammes von externen Faktoren abhängig sein soll. Die Anzeigeelemente dienen dazu, gesammelte Daten zu visualisieren. Im Laborbetrieb können die Anzeigeelemente ausserdem noch als Debug-Hilfe verwendet werden.

Die Sensorik des Fahrzeuges besteht aus einer CCD-Kamera, die über eine parallele Schnittstelle mit dem Steuerrechner verbunden ist. Als Anzeigeelement steht ein LC-Display zur Verfügung, welches Daten in Textform ausgeben kann und ebenfalls von dem Steuerrechner angesteuert wird.

Die Lösung dieser Teilaufgabe wurde in der Diplomarbeit "Implementierung optischer Sensor- und Anzeigeelemente für ein rechnergesteuertes Fahrzeug" (siehe [2]) bearbeitet.

1.2.4 Analyse- und Verwaltungsrechner

Damit die für den Betrieb des Fahrzeuges notwendigen Programme sinnvoll erstellt und verwaltet werden können, sind dafür vorgesehene Tools notwendig. Zusätzlich wird noch eine Einheit zur Analyse der Fahrzeugdaten benötigt.

Diese Aufgaben sollen durch ein Programm mit grafischer Bedieneroberfläche auf einem externen Rechner gelöst werden. Als Betriebssystem für den externen PC steht Windows 95 zur Verfügung. Der Rechner wird über serielle Schnittstellen mit dem Fahrzeug verbunden.

Die Lösung dieser Teilaufgabe wurde in der Diplomarbeit "Analyse- und Entwicklungsprogramm für ein rechnergesteuertes Fahrzeug" (siehe [3]) bearbeitet.

1.3 Übersicht Diplomarbeit

Diese Diplomarbeit beschäftigt sich mit dem Aufbau der mechanischen Komponenten des Fahrzeuges, der Ansteuerung der Antriebe sowie mit der Kommunikation mit dem Steuerrechner. Folgende Teile sollen bearbeitet werden:

- mechanische Entwicklung und Aufbau des Fahrzeuges
- Entwicklung und Aufbau der Elektronik zur Ansteuerung der Motoren
- Entwicklung und Aufbau der Elektronik zur Geschwindigkeitsmessung
- Implementierung der Drehzahlregelung der Motoren
- Implementierung des Protokolls zur Kommunikation mit dem Steuerrechner
- Erstellung der Software für den Mikrocontroller

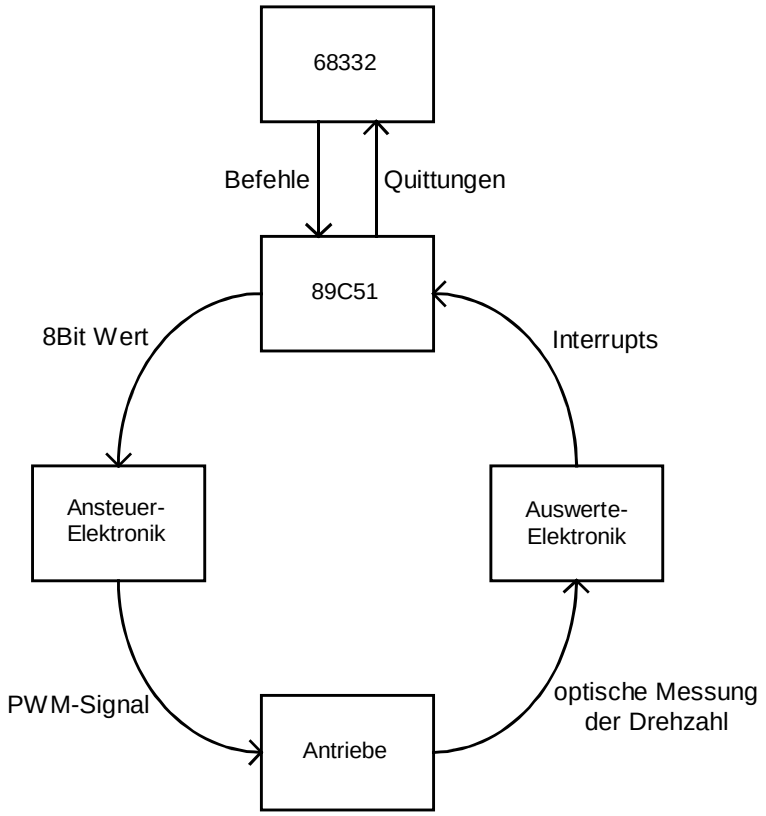


Abbildung 1.2: Übersicht Diplomarbeit

2 Regelung

Mit dem Begriff "Regeln" ist die Herstellung bzw. Erhaltung wünschenswerter Verhältnisse verbunden. Der Begriff stammt von dem lateinischen "regula", zu deutsch: Massstab, Richtschnur, ab.

Die Ausgangsgrösse (Istwert) eines Systems soll durch eine veränderliche Eingangsgrösse an einen gewünschten Wert (Sollwert) angepasst werden. Dieser Vorgang des Anpassens soll möglichst genau (keine Soll-Ist-Abweichung) und dynamisch (schnelle Reaktion) erfolgen. Das zu regelnde System wird Strecke genannt, die regelnde Einheit Regler.

Es ist davon auszugehen, dass die Strecke Störungen unterliegt. Der Regler sorgt dafür, dass die Ausgangsgrösse durch Vorgabe der Eingangsgrösse entgegen dem Einfluss der Störgrösse an ein Sollverhalten angeglichen wird.

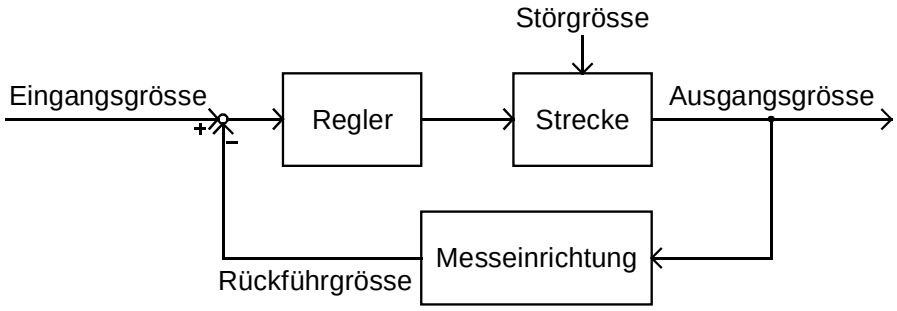


Abbildung 2.1: Prinzip einer Regelung

(siehe hierzu [4])

Die Ausgangsgrösse (Istwert) wird von einer Messeinrichtung aufgenommen und in eine Rückführgrösse umgewandelt, die mit der Eingangsgrösse (Sollwert) mittels eines Differenzbildners (Subtrahierer) verglichen wird. Diese Soll-Ist-Abweichung bildet den Eingangswert für den Regler. Der Regler generiert aus diesem Wert ein Steuersignal, mit dem die Strecke so beeinflusst werden soll, dass die Ausgangsgrösse der Strecke möglichst gut der Eingangsgrösse der Regelung folgt.

2.1 Regelungskonzept

Bei der Ansteuerung der Antriebe ist davon auszugehen, dass diese toleranzbehaftet sind und Störungen unterliegen. Um die Effekte der Toleranzen und Störungen weitestgehend zu kompensieren, ist eine Regelung notwendig. Besonders beim Geradeausfahren würden sich die Unterschiede der beiden Antriebe bemerkbar machen, da hier beide Antriebe die gleiche Drehzahl haben müssen.

Bei den zu regelnden Antrieben handelt es sich um Strecken mit PT1-Verhalten. Als einfach zu realisierende Regler stehen sowohl Proportional-Regler (P-Regler) als auch Integral-Regler (I-Regler) zur Verfügung.

Aus der Regelungstechnik (siehe hierzu [4]) ist bekannt, dass ein P-Regler an einer PT1-Strecke schnell aber ungenau ist, wohingegen ein I-Regler an einer PT1-Strecke genau aber langsam ist. Durch Kombination der beiden Regler erhält man einen PI-Regler, der sowohl schnell als auch genau ist.

Die Messeinrichtung hat meist ein proportionales Verhalten mit der Verstärkung $V = 1$, und kann somit vernachlässigt werden.

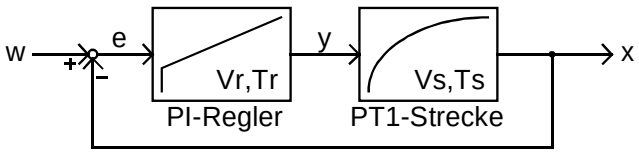


Abbildung 2.2: PI-Regler an PT1-Strecke

Übertragungsfunktionen:

PT1-Strecke: $G_S = \frac{V_S}{(1+T_S)}$

PI-Regler: $G_R = V_R \times \frac{(1+T_R)}{T_R}$

(V: Verstärkung, T: Zeitkonstante; Index R: Regler, Index S: Strecke)

Als Übertragungsfunktion des offenen Kreises ergibt sich:

$$G_O = V_R \times \frac{(1+T_R)}{T_R} \times \frac{V_S}{(1+T_S)}$$

Diese Funktion lässt sich vereinfachen indem $T_R = T_S$ und $V_R = \frac{1}{V_S}$ gewählt wird. Hieraus ergibt sich folgende vereinfachte Übertragungsfunktion:

$$G_O = \frac{1}{T_R}$$

Daraus ergibt sich als Übertragungsfunktion des geschlossenen Kreises:

$$G_W = \frac{1}{(1+T_R)}$$

Dieses entspricht der Übertragungsfunktion eines PT1-Gliedes mit $V = 1$. Dieses Verhalten ist erwünscht, da ein PT1-Verhalten relativ sachte einschwingt und keine Überschwinger aufweist.

Mit der einfachen Regelung der einzelnen Antriebe laufen beide Antriebe mit der gewünschten Drehzahl, jedoch unabhängig voneinander. Beim Geradeausfahren müssen die beiden Antriebe aber abhängig voneinander sein, damit sich eventuelle Störungen nicht nur auf einen Antrieb auswirken. Wenn sich eine Störung (z.B. Unebenheit des Untergrundes) nur auf einen Antrieb auswirkt, fährt das Fahrzeug nicht mehr geradeaus.

Eine einfache Möglichkeit der Geradeausregelung ist die Messung der zurückgelegten Strecken der einzelnen Antriebe und die Rückkopplung des Laufleistungsunterschiedes.

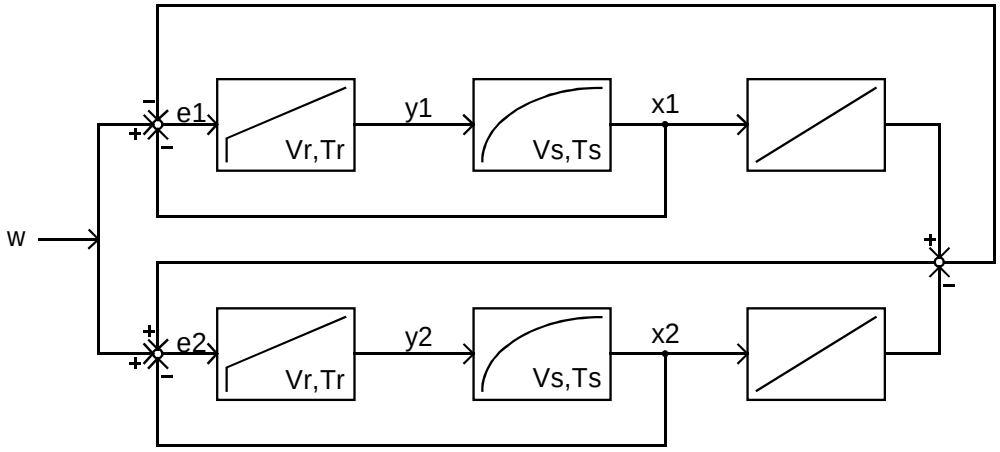


Abbildung 2.3: Prinzip der Aufschaltung des Laufleistungsunterschiedes

Legt nun ein Rad aufgrund einer Störung eine grössere Strecke als das andere Rad zurück, wird die Reglereingangsgrösse des anderen Antriebes grösser, woraus resultiert, dass der Antrieb den Streckenunterschied aufholt. Gleichzeitig wird die Reglereingangsgrösse des gestörten Antriebes kleiner, woraus resultiert, dass dieser zurückfällt. Sind die beiden, zurückgelegten Strecken wieder gleich gross, werden beide Antriebe wieder normal geregelt, da der Laufleistungsunterschied gleich Null ist.

Um sich drehende Antriebe zu bremsen, muss ein Gegenmoment erzeugt werden. Dieses Gegenmoment kann durch ein Signal erzeugt werden, dessen Wirkung der aktuellen Drehung entgegengesetzt ist. Fehlt die Information über die Drehrichtung, kann kein gezieltes Gegenmoment erzeugt werden. Daraus resultiert, dass ein aktives Bremsen nicht möglich ist. Um die Antriebe zu bremsen, müssen diese abgeschaltet werden. Daraus wiederum resultiert ein unkontrolliertes Ausrollen des Fahrzeuges, was vor allem eine Positionsbestimmung unmöglich macht.

Aufgrund eines Planungsfehlers wurde die Messung der Drehrichtung der Antriebe bei MARVIN nicht implementiert. Wegen der geringen, zur Verfügung stehenden Zeit war es leider nicht mehr möglich, die notwendige Hard- und Software nachzurüsten. Siehe hierzu die Abschnitte "Abschlussbetrachtung" und "Verbesserungsvorschläge".

2.2 Digitaler PI-Regler

Normalerweise würde ein Regler mit relativ wenig Aufwand analog aufgebaut werden können. Da ein Mikrocontroller zur Verfügung steht macht es jedoch Sinn, den Regler digital aufzubauen.

Der grösste Unterschied zwischen einem analogen und einem digitalen Regler ist der, dass der digitale Regler in diskreten Zeitschritten regelt und nicht kontinuierlich, das heisst, dass erst nach Ablauf einer bestimmten Zeit der Ausgangswert des Reglers neu berechnet wird. (siehe hierzu [5])

Diese Zeit muss so gewählt werden, dass der Regler genügend Dynamik aufweist, also schnell genug auf die Änderung des Eingangssignales reagieren kann. Die Zeit ist also entsprechend der grössten möglichen Eingangsfrequenz zu wählen.

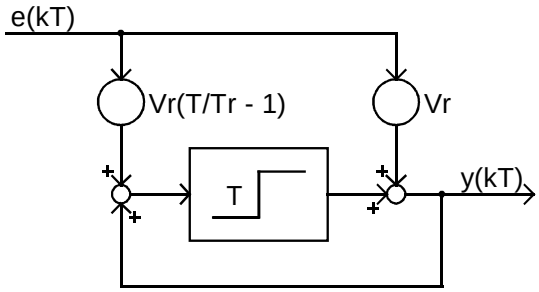


Abbildung 2.4: Prinzip eines digitalen PI-Reglers

Die Reglerausgangsgrösse ergibt sich aus folgender Gleichung:

$$y(kT) = V_R \times \left(e(kT) + e((k-1)T) \times \left(\frac{T}{T_R} - 1 \right) \right) + y((k-1)T)$$

wobei (kT) der aktuelle Zeitpunkt ist, und $((k-1)T)$ der um die Zeit T zurückliegende Zeitpunkt.

3 Hardware

In diesem Abschnitt werden alle Überlegungen dargestellt, welche zu der Wahl der verwendeten Hardwarekomponenten geführt haben.

3.1 Antrieb

Bei der Entwicklung von MARVIN stand von Anfang an fest, dass das Fahrzeug durch Elektromotoren angetrieben werden soll. Deshalb wurde die Betrachtung der verschiedenen Möglichkeiten der Kraftentwicklung bei dem Fahrzeug ausser Acht gelassen.

3.1.1 Antriebskonzepte

Um ein Fahrzeug anzutreiben, gibt es diverse Möglichkeiten, deren Anzahl nur durch die Phantasie des Entwicklers begrenzt ist. Punkte, die vor allem berücksichtigt werden müssen sind folgende:

- Art der Räder
- Anzahl und Lage der Räder

Die Art der Räder ist vor allem relevant, wenn es darum geht, wo sich das Fahrzeug bewegen soll.

Soll das Fahrzeug vor allem in unwegsamen Gelände eingesetzt werden, so ist eventuell ein Raupenantrieb in Betracht zu ziehen.

Handelt es sich bei dem zu befahrenden Gelände um eine sensible Landschaft, so sind grosse, unter geringem Druck stehende Reifen auf mehreren Rädern von Vorteil, um den Druck des Fahrzeuges so weit als möglich zu verteilen. Eine andere Variante wären Beine anstelle von Rädern, um so wenig Berührungen wie möglich mit dem Boden zu erreichen.

Sind sehr grosse Hindernisse zu erwarten, würde sich ein Springen des Fahrzeuges als vorteilhaft erweisen.

Ist eine sehr nasse, sumpfige Gegend der Einsatzort, so ist vielleicht ein Luftkissenfahrzeug die beste Variante oder aber ein komplett schwimmendes Gefährt bei einer noch nasseren Gegend.

Bei Gegenden die extrem schwer befahrbar sind (grosse Löcher, Risse, Hervorhebungen, Hindernisse) ist es vielleicht am besten den Boden ganz zu verlassen und auf ein fliegendes Fahrzeug zurückzugreifen.

Da MARVIN nur für eine Laborumgebung mit ebenem Untergrund entwickelt wurde, fiel die Wahl auf konventionelle Räder.

Die Anzahl der Räder ist relevant, wenn der Aspekt des Gewichtes des Fahrzeuges betrachtet wird. Zusätzlich muss auch hier der zu befahrende Untergrund betrachtet werden, da ein fester Untergrund mehr Druck verkraften kann als ein weicher Untergrund.

Ein schweres Fahrzeug benötigt, um die Kräfte gleichmässig zu verteilen, eine grössere Anzahl von Rädern, als ein vergleichsweise leichtes Fahrzeug.

Die Lage der Räder ist vor allem bei der Überlegung, wie sich das Fahrzeug bewegen soll, zu betrachten.

So sind z.B. mit vielen, hintereinander liegenden Rädern extrem enge Kurven nicht sehr leicht zu befahren. Mit zwei Rädern die auf der Mittelachse liegen, sind hingegen sogar Punktdrehungen möglich.

Mit MARVIN soll ein leichtes, sehr wendiges Fahrzeug verwirklicht werden. Aus diesem Grunde wurden zwei Antriebsräder gewählt, welche auf der Mittelachse des Fahrzeuges liegen. Zur Stabilisierung des Fahrzeuges sind zusätzlich zwei "Hilfsräder" vorne und hinten montiert. Der Einfachheit halber sind die Hilfsräder als Schleifer umgesetzt.

3.1.2 Lenkung

Ein Fahrzeug sollte auch in der Lage sein Kurven zu fahren. Deshalb muss eine Lenkung vorgesehen werden. Hierbei gibt es die Unterscheidung zwischen einer aktiven und einer passiven Lenkung.

Bei einer aktiven Lenkung wird, wie bei einem Auto, mindestens ein Rad gegenüber der Längsachse des Fahrzeuges gedreht. Der Vorteil ist hierbei, dass nur ein Rad angetrieben werden muss, der Nachteil ist jedoch die zusätzlich notwendige Mechanik, um die Drehbewegung des Rades zu ermöglichen.

Bei der passiven Lenkung wird, wie bei einem Panzer, die Lenkung durch unterschiedliche Drehzahlen der linken und rechten Antriebe realisiert. Der Vorteil ist die einfache Verwirklichung der Lenkung, der Nachteil ist, gerade bei mehreren Rädern, die erhöhte Reibung auf dem Untergrund.

MARVIN ist nur mit zwei Rädern versehen. Dadurch lässt sich eine einfache, reibungsarme, passive Lenkung realisieren. Durch die angetriebenen Räder auf der Mittelachse des Fahrzeuges sind Punktdrehungen möglich.

Da sich das Projekt MARVIN noch in den Kinderschuhen befindet, ist damit zu rechnen, dass eines Tages eine andere, aktiv lenkende Antriebseinheit getestet wird.

Um es dem Anwender so einfach wie möglich zu machen, und um eine möglichst hohe Kompatibilität zu eventuellen späteren, aktiv lenkenden Antriebseinheiten zu erreichen, wurde ein Befehlsformat gewählt (siehe Abschnitt Software), bei dem der Lenkradeinschlag Verwendung findet.

Es wurde ein Lenkradeinschlag in Stufen von 2° gewählt, da sich somit ein maximaler Wert von 180 für den Lenkradeinschlag ergibt, was ohne weiteres in einem Byte zu codieren ist.

Um den Lenkradeinschlag auf der momentanen, passiv lenkenden Antriebseinheit verwenden zu können, muss der Lenkradeinschlag in Leistungen der Motoren umgerechnet werden. Je nach Lenkradeinschlag wird die Leistung des zum Kurvenmittelpunkt näheren Rades verringert.

Diese Umrechnung erfolgt mittels einer Tabelle, in die für jeden Lenkradeinschlag die prozentuale Leistung der Motoren eingetragen ist. Durch die Verwendung einer Tabelle ergibt sich eine grosse Flexibilität, da die Reaktion des Fahrzeuges auf den Wert des Lenkradeinschlages durch simples Editieren der Tabelleneinträge geändert werden kann.

Die Reaktion des Fahrzeuges auf unterschiedliche Lenkradeinschläge lässt sich anhand folgender Zeichnung erklären.

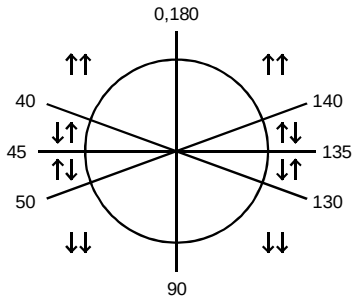


Abbildung 3.1: Lenkkreis

Die Pfeile entsprechen der Drehrichtung des jeweiligen Rades.

Lenkradeinschlag	Fahrtrichtung	Reaktion des Fahrzeuges
0	vorwärts	geradeaus fahren
1..39	vorwärts	Linkskurve
40	vorwärts	Drehung um linkes Rad
41..44	vorwärts	enge Linkskurve
45	vorwärts	Punktdrehung nach links
46..49	rückwärts	enge Linkskurve
50	rückwärts	Drehung um linkes Rad
51..89	rückwärts	Linkskurve
90	rückwärts	geradeaus fahren
91..129	rückwärts	Rechtskurve
130	rückwärts	Drehung um rechtes Rad
131..134	rückwärts	enge Rechtskurve
135	vorwärts	Punktdrehung nach rechts
136..139	vorwärts	enge Rechtskurve
140	vorwärts	Drehung um rechtes Rad
141..179	vorwärts	Rechtskurve
180	vorwärts	geradeaus fahren

3.1.3 Motoren

Bei der Wahl von Motoren spielten vor allem die Gesichtspunkte Wirkungsgrad, Kraft und Bauform ein Rolle.

Der Wirkungsgrad eines Motors beeinflusst den Stromverbrauch. Je höher der Wirkungsgrad, desto weniger Strom wird verbraucht, um eine bestimmte Leistung abzugeben. Vor allem bei batteriebetriebenen Fahrzeugen ist es wichtig, wenig Strom zu verbrauchen, da die mögliche Einsatzzeit des Fahrzeuges direkt mit dem Stromverbrauch zusammenhängt.

Die Kraft des Motors ist so zu wählen, dass das Fahrzeug dynamisch genug bewegt werden kann. Eine zu geringe Kraft des Motors lässt das Fahrzeug zu langsam und unbeweglich werden, eine zu grosse Kraft ist jedoch Verschwendung, die sich normalerweise mit einem zu hohen Stromverbrauch bemerkbar macht.

Die Bauform ist entscheidend, sobald der vorhandene Raum für das Fahrzeug eine Rolle spielt. Steht nur geringer Raum zur Verfügung, sollten kleine Motoren eingesetzt werden. Ein weiterer Aspekt der Bauform ist das Gewicht des Motors, das sich letztlich auch auf den Stromverbrauch auswirkt.

Bei der Wahl der Motoren für MARVIN spielten alle eben genannten Punkte eine Rolle. Der Wirkungsgrad war wichtig, da MARVIN als reines Batteriefahrzeug konzipiert ist.

Der Gesichtspunkt Kraft trug dazu bei, dass Motoren mit Getrieben eingesetzt wurden, die ein genügend grosses Drehmoment garantieren.

Die Bauform war entscheidend, da auf den vorgesehenen Platinen nur bedingt Platz vorhanden war.

Die technischen Daten der Motoren und Getriebe sind dem Anhang zu entnehmen.

3.2 Mechanischer Aufbau des Fahrzeuges

Durch die mechanische Konstruktion wird entscheidend das zukünftige Verhalten des Fahrzeuges beeinflusst. Hierbei spielen vor allem die Faktoren Grösse und Gewicht eine Rolle.

Die Konstruktionszeichnungen der mechanischen Teile sind dem Anhang zu entnehmen.

3.2.1 Grundplatine

Die Grundplatine stellt die Basis der Antriebseinheit dar. Auf ihr wird die Antriebseinheit aufgebaut, sowie alle anderen Elemente des Fahrzeuges montiert. Durch die Wahl der Grundplatine wird über die Grösse und zum Teil auch über die Stabilität der Gesamtkonstruktion entschieden. Soll an Grundfläche gespart werden, so bieten sich mehrlagige Platinen an, die jedoch den Nachteil der komplexen Herstellung haben. Gerade in der Entwicklungsphase bieten sich jedoch simple Lochrasterplatinen an, da auf diesen sehr schnell und einfach Änderungen durchgeführt werden können. Um trotz dessen die Grundfläche gering zu halten, können mehrere Platinen übereinander montiert werden. Die Kontaktierung kann durch Steckverbindungen hergestellt werden. Hierdurch bietet sich auch die Möglichkeit, vorgefertigte Komponenten mit in die Entwicklung einzubeziehen.

Im Falle von MARVIN war der Steuerrechner als fertige Platine im Europaformat vorgegeben.

Damit noch ausreichend Platz auf der tragenden Antriebseinheit für die notwendige Elektronik zur Ansteuerung der Antriebe, sowie für die mechanischen Teile der Antriebe verbleibt, wurde als Grundplatine eine Lochrasterplatine im doppelten Europaformat gewählt.

Da es sich bei der Grundplatine um eine Lochrasterplatine handelt, und diese auf der Leiterbahnseite, wie in der Platinenentwicklung üblich, mit Kupferlackdraht verdrahtet wurde, ergab sich eine relativ sensible Unterseite des Fahrzeuges. Um diese vor Beschädigungen zu schützen, wurde ein Unterfahrschutz konstruiert, der die gleichen Aussenabmessungen und Aussparungen wie die Grundplatine aufweist, und unter diese montiert wird.

3.2.2 Rahmen

Bei rechnergesteuerten Fahrzeugen kann es gerade in der Entwicklungs- und Testphase dazu kommen, dass das Fahrzeug unkontrolliert gegen eventuelle Hindernisse stösst. Damit keine Beschädigungen am Fahrzeug auftreten, ist eine Rahmenkonstruktion erforderlich, um die Stabilität der Grundplatine zu sichern.

Um den Aspekt des möglichst geringen Gewichtes zu berücksichtigen, wurde eine einfache Rahmenkonstruktion aus Aluminiumwinkeln gewählt, die durch weitere, kleinere Aluminiumhaltewinkel miteinander verbunden werden. In diesem Rahmen wird die Grundplatine befestigt.

3.2.3 Antriebshalterungen

Die Lage der Antriebshalterungen sind der gewünschten Antriebsform entsprechend zu wählen. Sie entscheidet unter anderem über die Höhe des Schwerpunktes des Fahrzeuges. Sind die Halterungen unter dem Fahrzeug befestigt, ergibt sich zwar eine grössere Bodenfreiheit, aber auch ein höherer Schwerpunkt.

Da MARVIN für eine Laborumgebung mit glattem Untergrund konzipiert wurde, ist die Bodenfreiheit kein wesentlicher Faktor. Deshalb wurden Antriebshalterungen gewählt, die auf dem Fahrzeug montiert werden.

Die Bauhöhe, die auf der Grundplatine zur Verfügung steht, ist wegen der Montage des Steuerrechners auf der Grundplatine begrenzt. Aus diesem Grund wurden die Motorhalterungen so konstruiert, dass die Motoren flach auf der Grundplatine aufliegen.

Um die Stabilität der Konstruktion zu erhöhen, wurde ein Blech konstruiert, welches auf der Unterseite des Fahrzeuges mit den Motorhalterungen verschraubt wird.

Die Achsen der Motoren müssen möglichst frei von Belastungen sein. Deswegen wurden Halterungen konstruiert, in die Kugellager eingepresst wurden, die das freie Ende der Motorachsen aufnehmen.

3.2.4 Räder

Bei der Wahl der Räder ist vor allem die Betrachtung des zu befahrenden Untergrundes wichtig. Bei einem sensiblen Untergrund sind z.B. relativ breite Räder mit entsprechender Bereifung notwendig.

Da MARVIN für eine Laborumgebung mit festem Untergrund konzipiert wurde, war es möglich, schmale Räder einzusetzen. Hierdurch konnten die Räder direkt auf den Achsen der Motoren montiert werden, woraus eine platzsparende Konstruktion resultiert. Als Bereifung boten sich O-Ringe aus Gummi an, da diese in nahezu jeder Grösse erhältlich sind.

Als Hilfsräder, welche für die stabile Lage des Fahrzeuges notwendig sind, wurden der Einfachheit halber Drahtschleifer eingesetzt.

Der Durchmesser der Räder entscheidet bei vorgegebenen Motoren und Getrieben über die zu erwartenden Geschwindigkeiten. Der Durchmesser der Räder wurde auf ca. 100mm festgelegt, um eine maximale Geschwindigkeit von ca. 1,6km/h zu erreichen, was für die zu befahrende Laborumgebung ausreicht.

3.3 Elektronik

In diesem Abschnitt werden die elektronischen Komponenten der Antriebseinheit von MARVIN erläutert.

3.3.1 Mikrocontroller

Um dem modularen Konzept von MARVIN gerecht zu werden, soll die Antriebseinheit über eine eigene, lokale Intelligenz verfügen. Hierdurch wird gewährleistet, dass sowohl der Steuerrechner als auch die Antriebseinheit austauschbar sind.

Als lokaler Prozessor soll ein Mikrocontroller verwendet werden, da dieser mit geringer, externer Beschaltung betrieben werden kann. Auf dem Markt sind Mikrocontroller in den verschiedensten Varianten verfügbar. Es sollte ein Typ verwendet werden, für den bereits Entwicklungstools vorhanden sind. Deshalb fiel die Wahl auf einen Mikrocontroller mit 8051 kompatiblen Kern. Weiterhin sollte der Mikrocontroller über internes RAM und EEPROM verfügen, da dieses relativ einfach programmierbar ist. Es wurde darüber nachgedacht einen Typ zu verwenden, der on-board über die integrierte Schnittstelle programmiert werden kann. Dieses würde ein Umprogrammieren sehr einfach machen. Der Chip müsste dazu nicht aus der Schaltung entfernt werden, und mit einem entsprechenden Gerät programmiert werden.

Solch eine Prozessor ist noch nicht in einer fehlerfreien Version auf dem Markt verfügbar. Daher wurde eine Mikrocontroller vom Typ 89C51 von der Firma ATMEL eingesetzt.

3.3.2 Ansteuerung des Antriebes

Bei der Ansteuerung eines Motors gibt es prinzipiell zwei Konzepte, mit denen der zur Ansteuerung bereitstehende 8Bit-Wert (von dem Mikrocontroller) in ein Ansteuersignal umgewandelt werden kann.

Zum einen kann der Wert in einen analogen Spannungspegel umgewandelt werden, mit dem dann der Motor angesteuert wird. Zum anderen kann der Wert in ein digital moduliertes Signal umgewandelt werden.

Um aus einem digitalen Eingangswert einen analogen Ausgangswert zu machen, wird ein Digital-Analog-Wandler (DA-Wandler) benötigt. Dieser lässt sich mit einem Summierverstärker aufbauen, bei dem die Wichtung der einzelnen Eingangsbits über die jeweilige Verstärkung (Verhältnis des Rückkoppelwiderstandes zum Eingangswiderstand) des Eingangsbits realisiert wird. Diese Wichtung wird mit einem R2R-Netzwerk (Widerstandswert jeweils doppelt so gross wie der vorherige) am Eingang des Verstärkers erreicht.

Der Nachteil dieses Konzeptes ist die relativ hohe Verlustleistung an den Widerständen und vor allem am Ausgangstransistor des benötigten Leistungstreibers für den Motor.

Bei der digitalen Modulation bietet sich die Puls-Weiten-Modulation (PWM) an. Bei der PWM wird das Ausgangssignal in einem festen Zeitraster sehr schnell zwischen den Pegeln Gnd (0V) und V_{cc} (Versorgungsspannung) hin- und hergeschaltet, wobei je nach Eingangssignal das Verhältnis zwischen den Zeiten, zu denen der jeweilige Ausgangspegel anliegt, (Puls-Pausen-Verhältnis) variiert. Von dem festen Zeitraster hängt die Frequenz der PWM ab, die so hoch sein sollte, dass der Motor das Signal glättet, um ruhig zu laufen. Desweiteren sinkt bei einer höheren Frequenz die Verlustleistung des Motors. Durch die hohe Frequenz der PWM kann der Motor durchaus mit einem V_{cc} versorgt werden, welches über der zulässigen Spannung für den Motor liegt, da die Überlastung zu kurz ist, um Schaden anzurichten.

Für die Umwandlung des digitalen Eingangswertes in ein PWM-Signal wird ein Zähler benötigt, der in einem festen Zeitraster (also getaktet) von Null bis zu einem definierten Endwert zählt. Als Endwert bietet sich der Maximalwert des Zählers an, da der Zähler bei einem Überlauf automatisch wieder bei Null zu zählen beginnt, und somit nicht neu geladen werden muss. Die Frequenz der PWM ergibt sich aus der Zeit, die der Zähler für einen Durchlauf von Null bis zu seinem Überlauf benötigt.

Zusätzlich wird ein Vergleicher benötigt, der den Eingangswert mit dem jeweiligen Zählerstand vergleicht. Ist der Eingangswert grösser als der Zählerstand, wird das Ausgangssignal auf V_{cc} geschaltet; ist der Eingangswert kleiner oder gleich dem Zählerstand, so wird das Ausgangssignal auf Gnd geschaltet.

Da das Ausgangssignal nur die beiden Werte Gnd oder V_{cc} annimmt, arbeitet der Ausgangstransistor der Treiberstufe für den Motor als Schalter, was die Verlustleistung im Vergleich zum analogen Betrieb senkt.

Bei der Erzeugung eines PWM-Signales durch einen Mikrocontroller gibt es wiederum zwei Möglichkeiten. Eine Software- und eine Hardwarelösung.

Bei der Softwarelösung werden der Zähler und Vergleicher intern mittels Software gebildet. Der Vorteil besteht darin, dass kein zusätzlicher Schaltungsaufwand anfällt. Jedoch hat diese Variante gerade bei Mikrocontrollern mit geringer Rechenleistung den Nachteil, dass zu viel Rechenleistung für die Erzeugung der PWM verbraucht wird.

Bei der Hardwarelösung, die auch bei MARVIN Anwendung findet, werden Zähler und Vergleicher extern aufgebaut. Der Prozessor wird dadurch extrem entlastet, da er für die PWM nur noch den Eingangswert liefern muss, was mit der Zuweisung einer einzigen Variablen erledigt ist. Allerdings ergibt sich hierbei der Nachteil des erhöhten Schaltungsaufwandes, woraus auch gerade bei hohen Frequenzen ein höherer Stromverbrauch resultiert. Dieser lässt sich durch Verwendung eines stromsparenden, programmierbaren Logikbausteines vermindern.

3.3.3 Drehzahlmessung

Für die Drehzahlregelung des Motors ist es notwendig die aktuelle Drehzahl (Istwert) zu kennen. Hierfür muss eine Messung durchgeführt werden. Um eine berührungslose, also einflussfreie Messung zu erreichen, bietet sich eine optische Messung mit Lichtschranken an. Zur Verfügung stehen zwei verschiedene Typen von Lichtschranken. Zum einen Gabellichtschranken, zum anderen Reflexlichtschranken.

Gabellichtschranken durchleuchten Löcher in dem zu messenden Objekt. Dieses hat den Vorteil, dass der Empfänger der Lichtschranke ein relativ grosses Eingangssignal bekommt, da der Sender direkt auf den Empfänger einstrahlt, oder aber der Empfänger komplett abgedeckt ist. Der Nachteil ist die relativ komplexe Herstellung der Lochscheibe. Es gibt vorgefertigte Lochscheiben, die direkt an der Antriebsachse befestigt werden, was im Falle von MARVIN jedoch aus Platzgründen nicht möglich war.

Reflexlichtschranken beleuchten das zu messende Objekt. Der Empfänger arbeitet mit der vom Objekt reflektierten Strahlung. Um eine optimale Reflexion zu erhalten wird eine Reflexscheibe mit Schwarz-Weiss Übergängen auf dem Objekt befestigt. Der Nachteil ist das geringere Signal, welches der Empfänger zur Verfügung hat. Der Vorteil ist die wesentlich einfachere Herstellung der Reflexscheibe (sie kann mit einem handelsüblichen CAD-Programm erstellt werden, ausgedruckt und einfach auf das zu messende Objekt geklebt werden) sowie die simple Montage der Lichtschranke.

Um das Ausgangssignal optimal mit dem Mikrocontroller verarbeiten zu können, muss es in ein Rechtecksignal mit den Pegeln Gnd und V_{cc} umgewandelt werden. Hierfür wird das Ausgangssignal der Lichtschranke, welches prinzipielle Ähnlichkeit mit einer Sinusschwingung hat, mit einem einstellbaren Spannungspegel verglichen. Dieses wird mittels eines Komparators (OP, der nicht zurückgekoppelt ist) erreicht, der je nach Eingangswert den Ausgang auf Gnd oder V_{cc} schaltet.

Da die Lichtschranke auch einen Wert liefern kann, der gleich dem festen Vergleichspegel ist, kann es bei dem Ausgangssignal des Komparators zu Schwingungen kommen, die gefiltert werden müssen, da sonst ein Messfehler auftritt.

Für die Filterung bietet sich ein einfacher RC-Tiefpass an, der so dimensioniert wird, dass nur die Frequenzen durchgelassen werden, die auch real auftreten können. Um aus dem Ausgangssignal des Filters wieder ein sauberes Rechtecksignal zu machen, wird das Signal nochmals mit einem Schmitt-Trigger gefiltert.

Das so erhaltene Signal lässt zu, dass die Frequenz des Signales mit dem Mikrocontroller gemessen werden kann. Hierbei gibt es zwei Möglichkeiten. Entweder wird die Anzahl der Impulse pro Zeiteinheit gemessen, oder aber die Zeit zwischen den Impulsen, also die Frequenz des Signales.

Die erste Möglichkeit ist sehr einfach zu implementieren, birgt jedoch den Nachteil des festen Zeitrasters. Das heisst, dass bei jeder gemessenen Geschwindigkeit ein Zeitfenster gewartet werden muss, um einen aktuellen Wert zu erhalten.

Die zweite Möglichkeit, bei der immer der aktuelle Wert gemessen wird, ist wesentlich dynamischer.

Aufgrund des Umstandes, dass die Impulse zu unbestimmten Zeiten auftreten, können sie nicht an normalen Eingängen des Mikrocontrollers gemessen werden. Da aber jeder Impuls zu der Zeit, zu der er auftritt gemessen werden muss, bietet sich an, mit den Impulsen Interrupts auszulösen. In der jeweiligen Interrupt-Service-Routine wird die Zeit gemessen und in einer globalen Variable abgespeichert, damit der Wert dann im normalen Programmablauf zur Verfügung steht.

4 Software

In diesem Abschnitt wird die Kommunikation zwischen der Antriebseinheit und dem Steuerrechner dargestellt.

4.1 Schnittstelle zum Steuerrechner

Die Kommunikation mit dem Steuerrechner findet über die, vom Mikrocontroller zur Verfügung gestellte, serielle Schnittstelle statt.

Bei der seriellen Schnittstelle handelt es sich um eine Dreidraht-Verbindung, die vollduplexfähig und interruptgesteuert ist. Es stehen nur die Leitungen Gnd (Massebezug), RxD (Empfangen) und TxD (Senden) zur Verfügung. Steuerleitungen für ein Hardwareprotokoll sind nicht vorhanden. Damit sichergestellt werden kann, dass kein Datenverlust bei der Übertragung durch eventuelle Überlastung des Empfängers auftritt, muss ein Protokoll verwendet werden.

Hier bietet sich das Softwareprotokoll Xon/Xoff an. Bei diesem Protokoll wird gesendet, bis der Empfänger durch Senden des Zeichens Xoff signalisiert, dass er überlastet ist, und weitere Daten nicht mehr verarbeiten kann. Dieses würde bei einem weiteren Senden zu Datenverlust führen.

Der Sender wartet nun solange mit dem Senden, bis er das Zeichen Xon empfängt. Hiermit signalisiert der Empfänger, dass er wieder empfangsbereit ist.

Der Vorteil bei dem verwendeten Protokoll ist, dass die Schnittstelle mit drei Leitungen auskommt. Der Nachteil ist, dass die Zeichen Xon und Xoff nicht im Wortschatz der gesendeten Daten auftreten dürfen, da diese immer als Steuerzeichen interpretiert werden. Dadurch wird der verfügbare Wortschatz kleiner. Die Zeichen müssen aus dem Wortschatz ausgeschlossen werden, um zu garantieren, dass sie nicht aus Versehen gesendet werden.

Auf eine Fehlerkorrektur kann verzichtet werden, da die Länge der verwendeten Schnittstelle im Zentimeterbereich liegt.

4.2 Protokoll der Datenübertragung

Bei der Übertragung von Daten zwischen Rechnern muss ein vorher definiertes Protokoll eingehalten werden, um eine fehlerfreie Bearbeitung der Daten zu gewährleisten.

Prinzipieller Unterschied zwischen Protokollen ist die Datensatzlänge. Es gibt variable und feste Datensatzlängen.

Ein Protokoll mit variabler Datensatzlänge ist sehr flexibel. Es ist aber eine Absprache zwischen Sender und Empfänger, die aktuelle Datensatzlänge betreffend, notwendig.

Bei einem Protokoll mit fester Datensatzlänge ist keine Absprache zwischen Sender und Empfänger notwendig. Es wird immer die gleiche Datensatzlänge verwendet. Dieses erspart die Absprache, ist aber weniger flexibel.

Da die Kommunikation zwischen der Antriebseinheit und dem Steuerrechner immer gleichlange Befehle beinhaltet, wurde ein Protokoll mit konstanter Datensatzlänge gewählt. Desweiteren ist die Datensatzlänge so gering (vier Byte), dass eine Codierung der Datensatzlänge nicht mehr gerechtfertigt wäre.

4.2.1 Steuerrechner -> Antriebseinheit

Der Steuerrechner sendet Fahrbefehle, die von der Antriebseinheit decodiert, quittiert, und bei korrekten Daten ausgeführt werden. Ein Fahrbefehl setzt sich aus folgenden Daten zusammen:

- Distance: zu fahrende Entfernung
- Speed: zu fahrende Geschwindigkeit
- Angle: Lenkwinkel
- Number: Nummer des aktuellen Befehls

4.2.2 Antriebseinheit -> Steuerrechner

Die Antriebseinheit sendet auf jeden empfangenen Datensatz eine Antwort. Eine Antwort setzt sich aus folgenden Daten zusammen:

- Code: Art der Antwort
- Parameter1: zur Antwort gehörende Daten
- Parameter2: zur Antwort gehörende Daten
- Number: Nummer des Befehls, auf den geantwortet wird

Die Einstellungen der seriellen Schnittstelle und das Protokoll sind dem Anhang zu entnehmen.

5 Abschlussbetrachtung

Diese Diplomarbeit beschäftigte sich, wie in der Einleitung bereits erwähnt, mit dem Aufbau der mechanischen Komponenten des Fahrzeuges, der Ansteuerung der Antriebe sowie mit der Kommunikation mit dem Steuerrechner.

Die mechanische Entwicklung und der Aufbau des Fahrzeuges wurden vollständig bearbeitet.

Während der Entwicklungsphase kam es zu diversen Fehlkonstruktionen. So wurden z.B. Motorhalterungen konstruiert, bei denen die Schrauben zur Befestigung komplett von dem Motor abgedeckt wurden, somit also nicht erreichbar waren. Durch nochmalige Konstruktion wurden schliesslich fehlerfreie, montierbare Teile hergestellt, die in dem vorliegenden Fahrzeug Verwendung finden.

Die Entwicklung und der Aufbau der Elektronik zur Ansteuerung der Motoren wurden vollständig bearbeitet.

Bei den Versuchen mit der Erzeugung des PWM-Signales hat sich gezeigt, dass eine externe Elektronik für die Erzeugung des PWM-Signales notwendig war, da der verwendete Mikrocontroller für eine interne Erzeugung des PWM-Signales zu langsam war. Der gewählte Mikrocontroller fand trotz dessen Anwendung, da ein anderer aufgrund der fehlenden Entwicklungstools nicht in Betracht kam. Die externe Elektronik wurde mit einem programmierbaren Baustein vom Typ Mach210 verwirklicht. Bei der Entwicklung der externen Elektronik wurde vor allem auf einen möglichst geringen Stromverbrauch geachtet.

Die Entwicklung und der Aufbau der Elektronik für die Geschwindigkeitsmessung wurde nur teilweise bearbeitet.

Der Betrag der Geschwindigkeit wird gemessen, aufgrund eines Planungsfehlers jedoch nicht die Richtung der Geschwindigkeit. Hieraus resultieren vor allem Probleme für die Regelung.

Probleme bei der Geschwindigkeitsmessung resultierten nur aus den Schwingungen, welche die Lichtschranken bei den Schwarz-Weiss Übergängen der Reflexscheibe erzeugten. Durch geeignete Filtermassnahmen wurde dieses Problem gelöst.

Die Implementierung der Drehzahlregelung der Motoren wurde nur teilweise bearbeitet.

Hierbei ergab sich folgendes Problem: Es fehlten Informationen über die Richtung, in die sich die Antriebe drehen. Dadurch ist kein aktives Bremsen der Antriebe möglich. Bei der vorliegenden Version müssen die Antriebe zum Bremsen abgeschaltet werden, woraus ein undefiniertes Ausrollen des Fahrzeuges resultiert, was z.B. eine Positionsbestimmung unmöglich macht. Desweiteren ist das aktive Halten der Fahrzeugposition nicht möglich. Das Fahrzeug kann zwar messen, dass es rollt, aber nicht, wohin es rollt. Aus diesem Grund ist keine Gegenmassnahme möglich. Versuche, bei denen mit der Richtungsinformation des letzten Fahrbefehles gearbeitet wurde, gingen wortwörtlich nach hinten los. Wurde das Fahrzeug in die der vorherigen Fahrtrichtung entgegengesetzten Richtung angestossen, hat es in die selbe Richtung beschleunigt, was dazu führte, dass das Fahrzeug letztlich unkontrolliert mit der maximal möglichen Geschwindigkeit davon fuhr.

Die Implementierung des Protokolls zur Kommunikation mit dem Steuerrechner wurde vollständig bearbeitet.

Nachdem eine Einigung mit dem Autor der sich mit dem Steuerrechner befassenden Diplomarbeit erzielt wurde, konnte das Protokoll ohne grössere Probleme implementiert werden. Kleinere Probleme gab es nur bei der exakten Festlegung der von seiten des Steuerrechners benutzten Pins an den Anschlussleisten.

Die Erstellung der Software für den Mikrocontroller wurde vollständig bearbeitet.

Nachdem PAS51 als Programmiersprache für den Mikrocontroller festgelegt wurde, traten nur kleine, einfach zu beseitigende Probleme auf. Ein Problem war die Tatsache, dass Zahlen vom Typ Real von dem Mikrocontroller nur mit externem Speicher verarbeitet werden können, dieser aber nicht vorgesehen war. Gelöst wurde das Problem durch Multiplikation und anschliessende Division des Wertes. Diese Lösung führte allerdings zum nächsten Problem, welches sich als das grösste entpuppte: Es war die 16Bit-Multiplikation, die aufgrund eines freigegebenen Interrupts (Timer1) über einen relativ langen Zeitraum nicht zur Funktion zu bringen war. Nachdem der "Fehler" lokalisiert war, konnte ohne weitere Probleme weiter gearbeitet werden.

Die technischen Details der Antriebseinheit sind dem Anhang zu entnehmen.

6 Verbesserungsvorschläge

Während der Verwirklichung und der Testphase von MARVIN sind einige Punkte aufgefallen, die in späteren Versionen verbessert werden könnten. Diese Punkte sind in diesem Abschnitt dargestellt.

Die wichtigste Erweiterung der Antriebseinheit ist die Implementierung der richtungsabhängigen Geschwindigkeitsmessung. Hierzu sind andere Lichtschranken notwendig, welche die Räder an zwei Stellen abtasten, und über einen Phasendiskriminator die Drehrichtung feststellen. Der implementierte Regelalgorithmus ist prinzipiell bereits in der Lage sowohl positive als auch negative Werte zu verarbeiten.

Eine weitere sinnvolle Erweiterung der Drehzahlregelung wären Reflexscheiben mit mehr Impulsen pro Umdrehung. Daraus würde eine, gerade bei geringen Geschwindigkeiten, dynamischere Regelung resultieren, da die gemessenen Zeiten zwischen den Impulsen kleiner wären. Bei anderen Reflexscheiben müssten ebenfalls andere Lichtschranken eingesetzt werden, da die momentanen Lichtschranken mit den 90 Impulsen pro Umdrehung bereits an der Grenze ihres Auflösungsvermögens sind.

Um die Unabhängigkeit des Fahrzeuges zu erhöhen, war eine auf dem Fahrzeug aufgebaute Stromversorgung vorgesehen. Aufgrund der geringen zur Verfügung stehenden Zeit wurde diese jedoch nicht realisiert. Die für das Fahrzeug vorgesehenen Akkumulatoren, sowie Platz für den Aufbau der Stromversorgung sind vorhanden. Wenn das Fahrzeug batteriebetrieben ist, sollten die Motoren direkt von den Batterien gespeist werden, um die wesentlich stabilere Spannung der Batterien nutzen zu können, und um die Dynamik der Motoren durch die höhere Spannung zu vergrössern.

Eine weitere Erhöhung der Unabhängigkeit des Fahrzeuges lässt sich erzielen, wenn die serielle Verbindung zum Verwaltungsrechner als Funkstrecke aufgebaut wird. Hierdurch würde das Kabel der momentan bestehenden seriellen Verbindung entfallen.

Der momentan in der Antriebseinheit verwendete Mikrocontroller vom Typ 89C51 ist nahezu ausgereizt. Die Kapazität des internen Speichers wird fast vollständig genutzt. Eine Verbesserungsmöglichkeit ist die Verwendung eines 8052 kompatiblen Typs, der mehr internen Speicher aufweist. Eine andere Möglichkeit ist eventuell die Implementierung des Programmes in Assembler. Dadurch würde allerdings die gute Lesbarkeit einer Hochsprache entfallen.

Bei der Wahl eines anderen Mikrocontrollers, sollte ein Typ gewählt werden, der direkt on-board über die integrierte Schnittstelle programmiert werden kann. Dieses würde die Änderung der Software wesentlich einfacher machen.

Eine weitere, sinnvolle Eigenschaft eines neuen Mikrocontrollers, wären integrierte Schaltungen zur Erzeugung der PWM-Signale. Ein Teil der externen Schaltung würde somit wegfallen, woraus sich ein noch geringerer Stromverbrauch ergäbe.

Wenn das Fahrzeug so weit entwickelt ist, dass eine Version für die vorgesehenen Praktikumszwecke in Serie hergestellt werden soll, wäre es angebracht die Grundplatine des Fahrzeuges nicht mehr per Hand zu verdrahten, sondern sie in den dafür vorgesehenen Einrichtungen der FH-Wedel zu entwickeln und zu ätzen. Dadurch würde man eine Platine erhalten, die sowohl mechanisch als auch elektrisch wesentlich stabiler ist. Die mechanische Stabilität würde daraus resultieren, dass die Leiterbahnen nicht mehr abreissen könnten, sobald die Platine unglücklich angefasst wird. Eventuell kann dann auf den Unterfahrschutz verzichtet werden. Die elektrische Stabilität würde aus der besseren Führung der Leiterbahnen resultieren. Desweiteren würden einige Verbindungen entfallen, die in der momentanen Version durch Kabelbrücken gelegt sind.

Eine weitere, sinnvolle Erweiterung, wäre eine zusätzliche Sensorikebene für die Antriebseinheit. Die Sensoren sollten vor allem für den Schutz des Fahrzeuges eingesetzt werden. So sind Kontaktschalter, sogenannte Bumper, die eine Kollision des Fahrzeuges mit Hindernissen melden, denkbar. Die Bumper würden dafür sorgen, dass die Antriebseinheit reagieren kann, wenn das Fahrzeug "aneckt".

Zusammenfassend lässt sich sagen, dass mit der vorliegenden Version von MARVIN der Grundstein für ein selbstfahrendes Fahrzeug an der FH-Wedel gelegt wurde. Das Fahrzeug erfüllt die Erwartungen, die an es gestellt wurden, und ist in der Lage die Testaufgabe - Stabsuche - zu erfüllen.

Aufbauend auf dieser geschaffenen Basis, lässt sich mit geringen Änderungen an den Einzelkomponenten die gewünschte "Serienreife" erzielen.

Wegen der Modularität des Fahrzeuges ist es auch denkbar, Komponenten auszutauschen. So sind z.B. andere Sensoren, eine andere Antriebseinheit oder auch ein anderer Steuerrechner denkbar. Das Hinzufügen neuer Komponenten ist ebenso möglich. Somit ist die Erweiterung des Fahrzeuges um z.B. ein künstliches, neuronales Netz möglich.

7 Anhang

In diesem Abschnitt sind die technischen Details der Mechanik, Elektronik und Software der Antriebseinheit aufgeführt.

7.1 Regelung

Für die Dimensionierung der Regelung ist es notwendig das Verhalten der zu regelnden Strecken zu kennen. Hierzu wird eine Prozessidentifikation durchgeführt, bei der ein definiertes Eingangssignal auf die Strecke geschaltet wird, und das Verhalten der Strecke über die Zeit betrachtet wird. Die Daten der Motorenhersteller können nicht verwendet werden, da diese sich nur auf den Leerlaufbetrieb beziehen, die Motoren jedoch normalerweise unter Last betrieben werden.

Im Falle der Antriebe wurden die Zeitabstände zwischen den von den Lichtschranken gemessenen Impulsen gemessen.

Aus den erhaltenen Daten haben sich folgende Werte ergeben:

Verstärkung der Strecke: ca. 1
Nacheilzeitkonstante: ca. 220ms

Wie im Abschnitt Regelung beschrieben, wurden die Parameter der Regelung entsprechend den Werten der Strecken gewählt:

Verstärkung des Reglers: 1
Reglerzeitkonstante: 250ms

Die Zeitkonstante des Reglers wurde geringfügig grösser als die der Strecke gewählt, da nur ganzzahlige Werte mit dem Mikrocontroller verarbeitet werden konnten.

Die Formel zur Berechnung der Ausgangsgrösse des Reglers ist, wie im Abschnitt "Regelung" angegeben:

$$y(kT) = V_R \times \left(e(kT) + e((k-1)T) \times \left(\frac{T}{T_R} - 1 \right) \right) + y((k-1)T)$$

Die Zeit T wurde auf $T = \frac{1}{20}\,\text{sec}$ eingestellt. Hieraus folgt für den Term $\frac{T}{T_R} - 1$ ein Wert von -0.8. Da der Mikrocontroller ohne externen Speicher keine Realzahlen verarbeiten kann, wurde der Ausdruck als $\frac{V_i}{D_i} = \frac{4}{5}$ implementiert. Das Vorzeichen wurde vorgezogen, da so die Konstanten V_I und D_I vom Typ Byte gewählt werden konnten. Als Formel, welche implementiert wurde, ergibt sich somit:

$$y(kT) = V_R \times \left(e(kT) - e((k-1)T) \times \frac{V_I}{D_I} \right) + y((k-1)T)$$

7.2 Antriebe

Bei den verwendeten Motoren handelt es sich um Elektromotoren der Firma Faulhaber vom Typ 1624T 003S. Die Nennspannung der Motoren beträgt 3Volt. Die benutzten Getriebe der Serie 16/5 haben eine Übersetzung von 141:1.

Für die Räder war ein Bereifung notwendig, um eine grössere Bodenhaftung zu gewährleisten. Als Bereifung wurden O-Ringe aus Gummi von der Firma Gehrkens aus Pinneberg, mit einem Innendurchmesser von 98mm und einer Stärke von 3mm verwendet.

Die technischen Daten der Motoren sind dem Katalog des Herstellers Faulhaber entnommen (siehe hierzu [6]).

Die zur Messung der Drehzahl verwendeten Messscheiben erzeugen 90 Impulse pro Umdrehung. Dadurch entspricht eine Geschwindigkeitsangabe von 90 genau einer gefahrenen Geschwindigkeit von 1Umdrehung/sec der Räder, was bei den gegebenen Rädern ca. 33cm/sec, also 1,2km/h entspricht.

Als minimal fahrbare Geschwindigkeit hat sich 10, also 3,7cm/sec (0,13km/h) herausgestellt. Die maximal zugelassene Geschwindigkeit beträgt 120, also 44cm/sec (1,6km/h).

Durch die Festlegung der Anzahl der Impulse/Umdrehung wird auch die messbare, zurückgelegte Entfernung festgelegt. Sie entspricht bei den gegebenen Werten ca. 4mm. Dieser Wert war zu klein als Schrittweite. Deshalb wurde ein Entfernungsfaktor von 5 eingeführt, der somit die kleinste fahrbare Entfernung auf ca. 2cm festlegt.

Da die Motoren nur eine Nennspannung von 3V haben, dürfen diese auch nur mit einem PWM-Signal angesteuert werden, welches maximal 3V entspricht. Der maximal zulässige Eingangswert der PWM lässt sich wie folgt berechnen:

$$\text{MaxSignal} = \frac{U_{MAX}}{\text{MaxCount}}$$

wobei MaxCount der Maximalwert des PWM-Zählers ist.

7.3 Mechanik

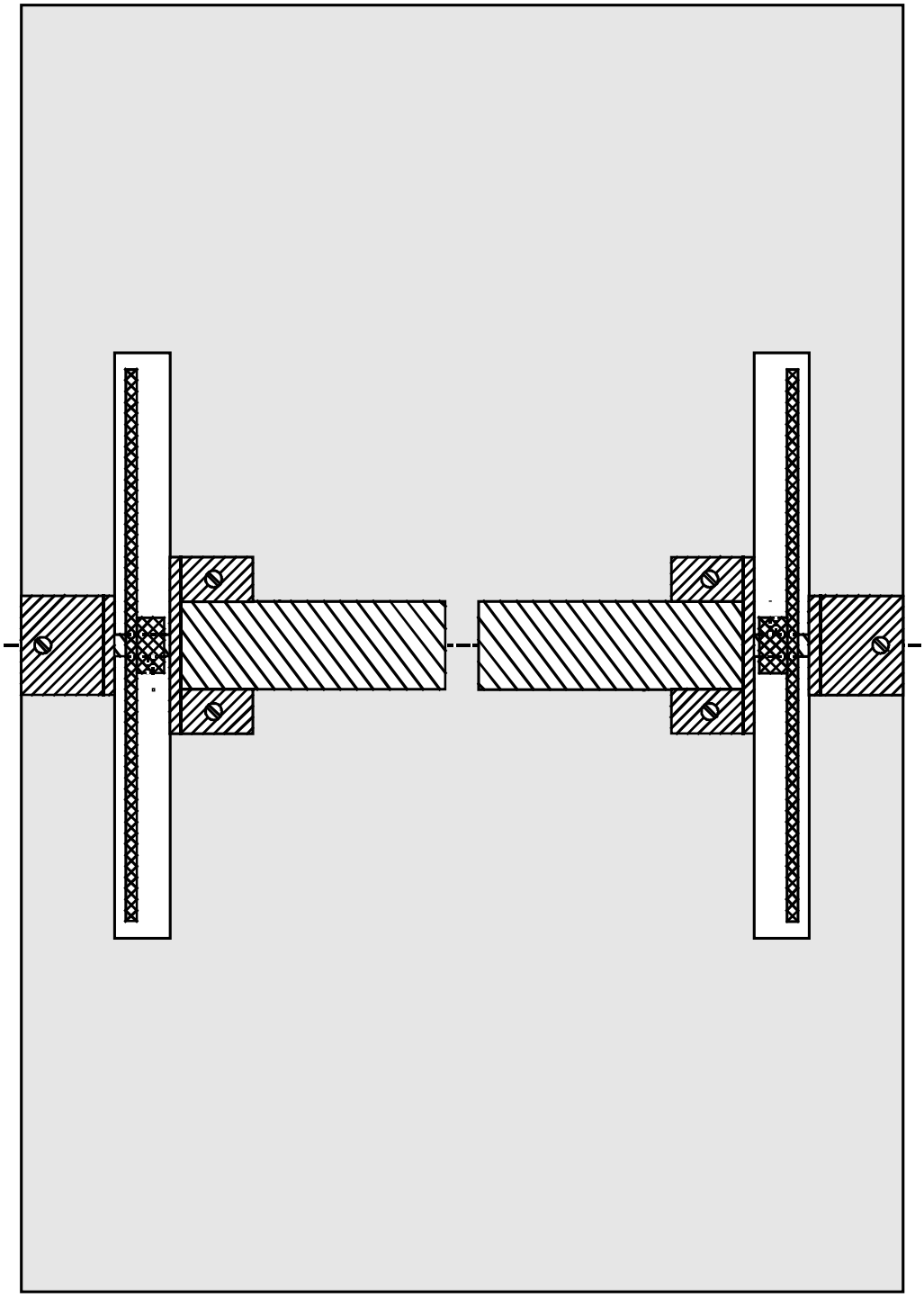


Abbildung 7.1: Antriebseinheit, Draufsicht

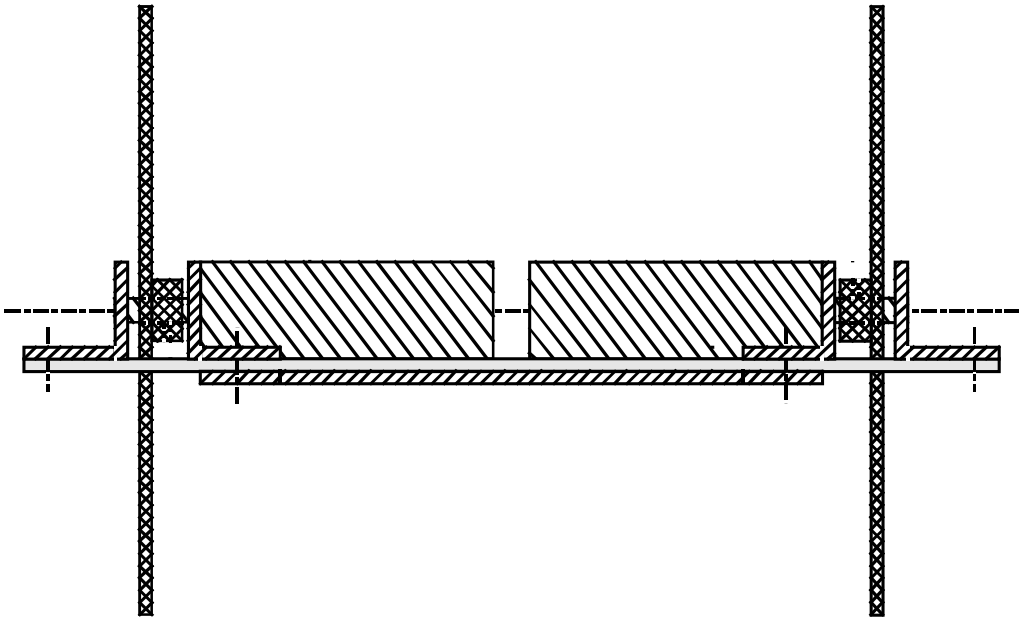


Abbildung 7.2: Antriebseinheit: Frontansicht

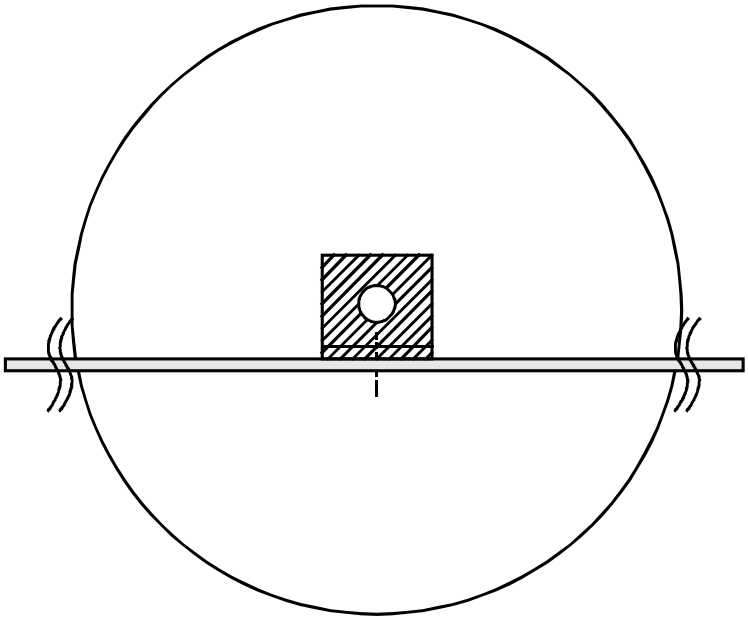


Abbildung 7.3: Antriebseinheit: Seitenansicht

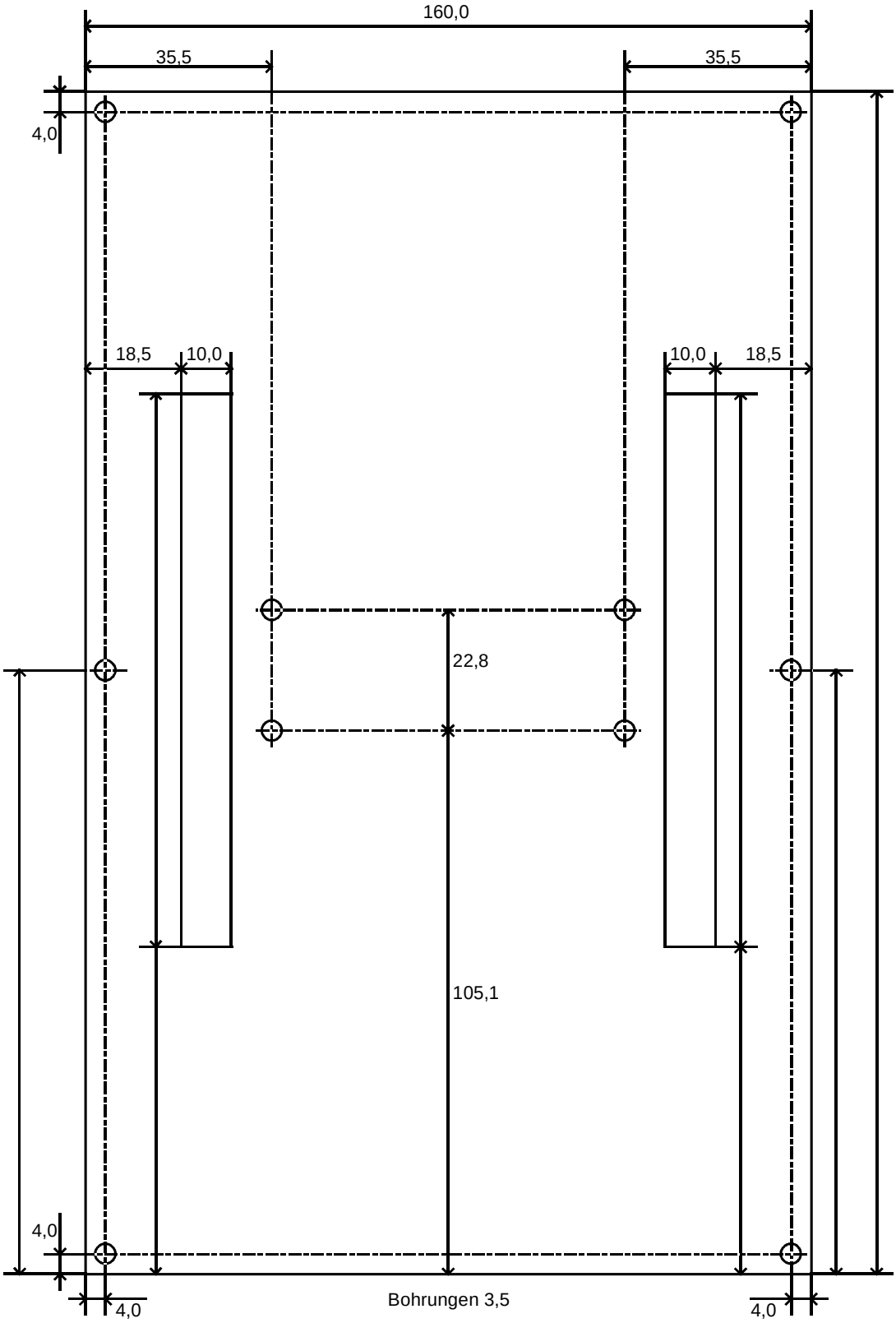


Abbildung 7.4: Konstruktionszeichnung: Grundplatine

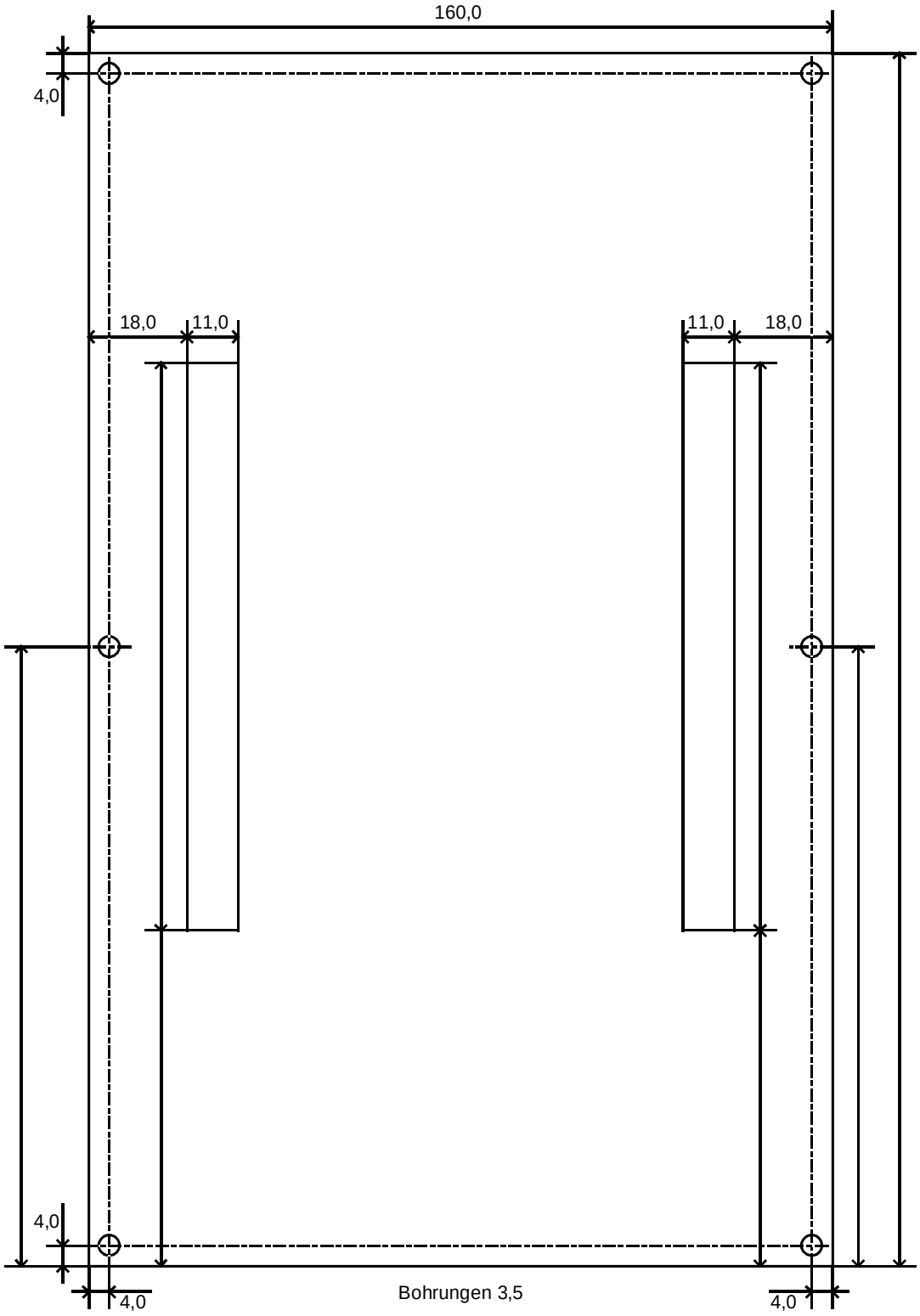


Abbildung 7.5: Konstruktionszeichnung: Unterfahrschutz

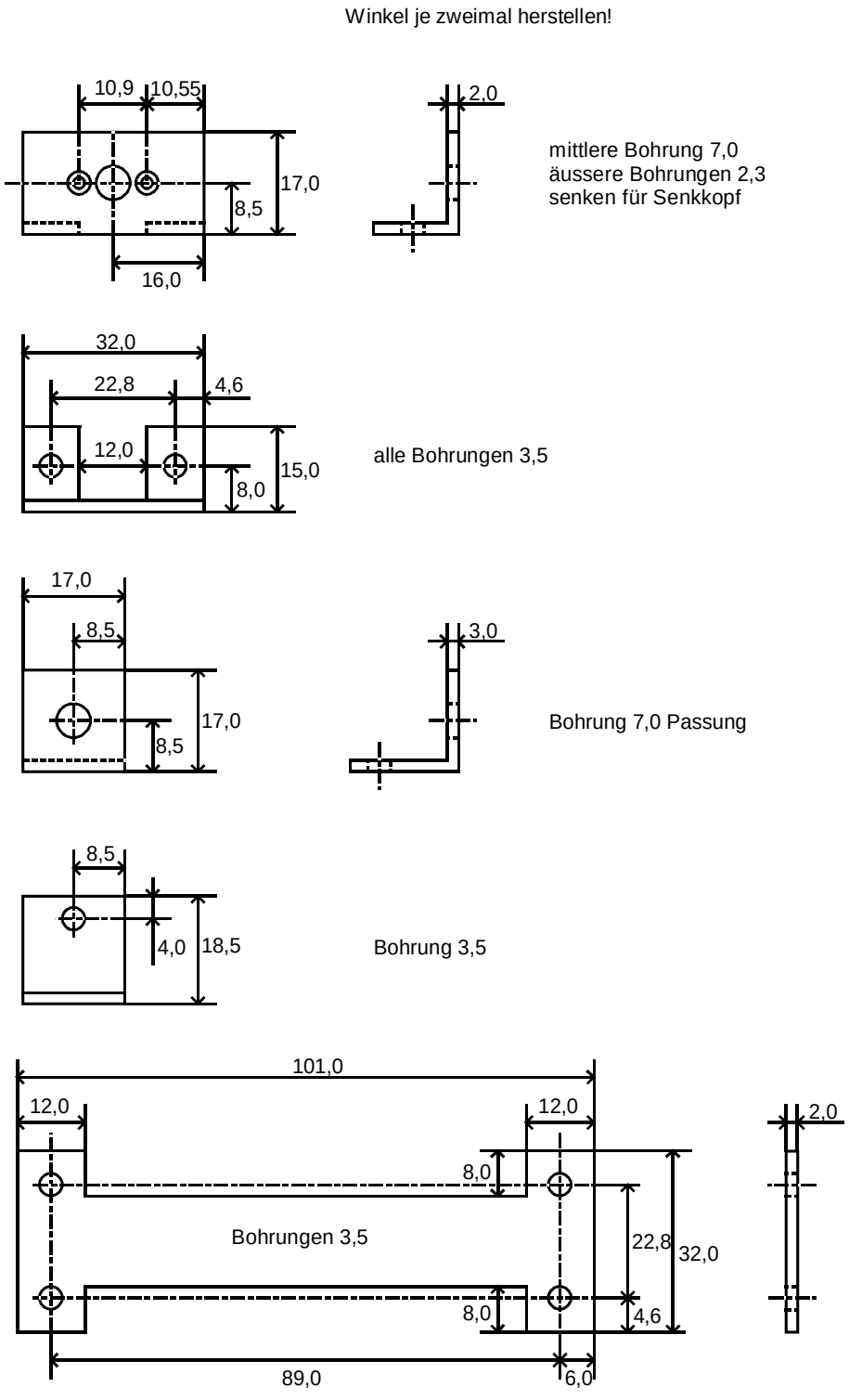
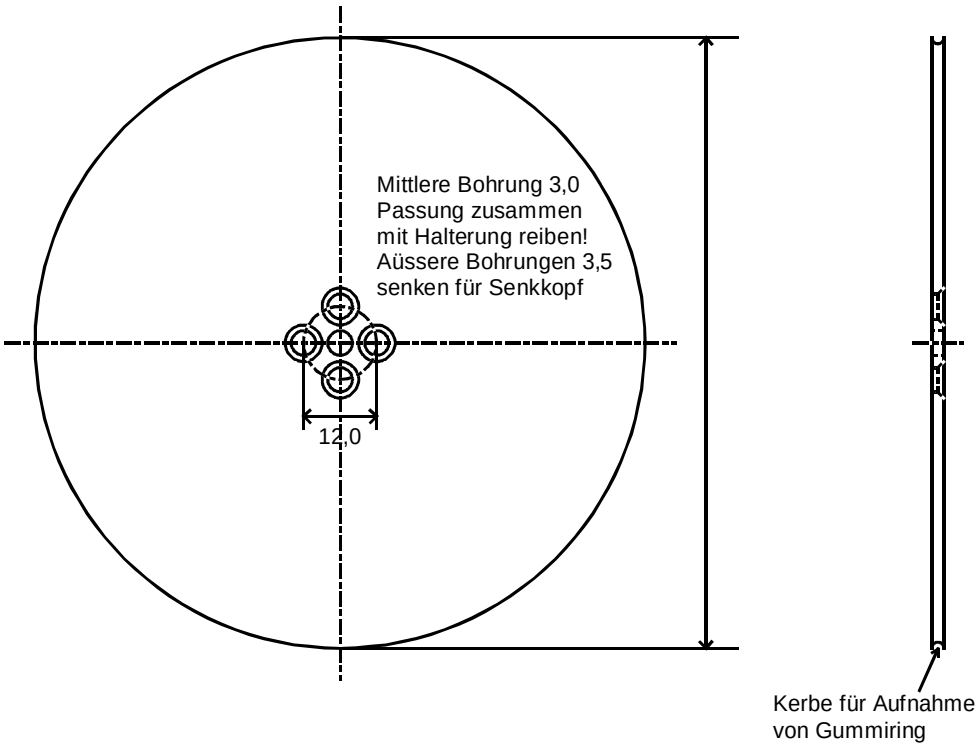
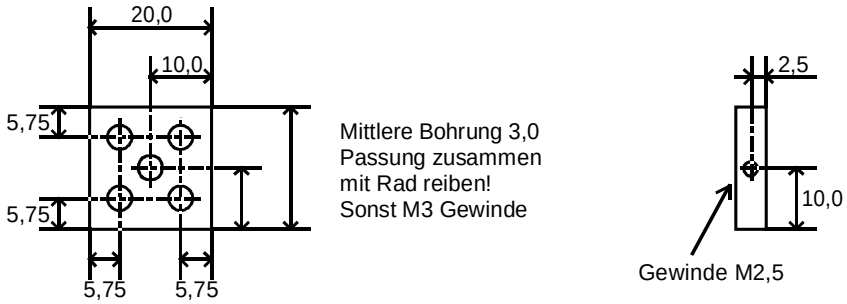


Abbildung 7.6: Konstruktionszeichnung: Motorhalterungen





Dicke der Materialien:

Rad: 2mm
Halterung: 5mm
zusammen jedoch nicht mehr als 7mm

Jedes Teil zweimal herstellen!

Abbildung 7.7: Konstruktionszeichnung: Räder

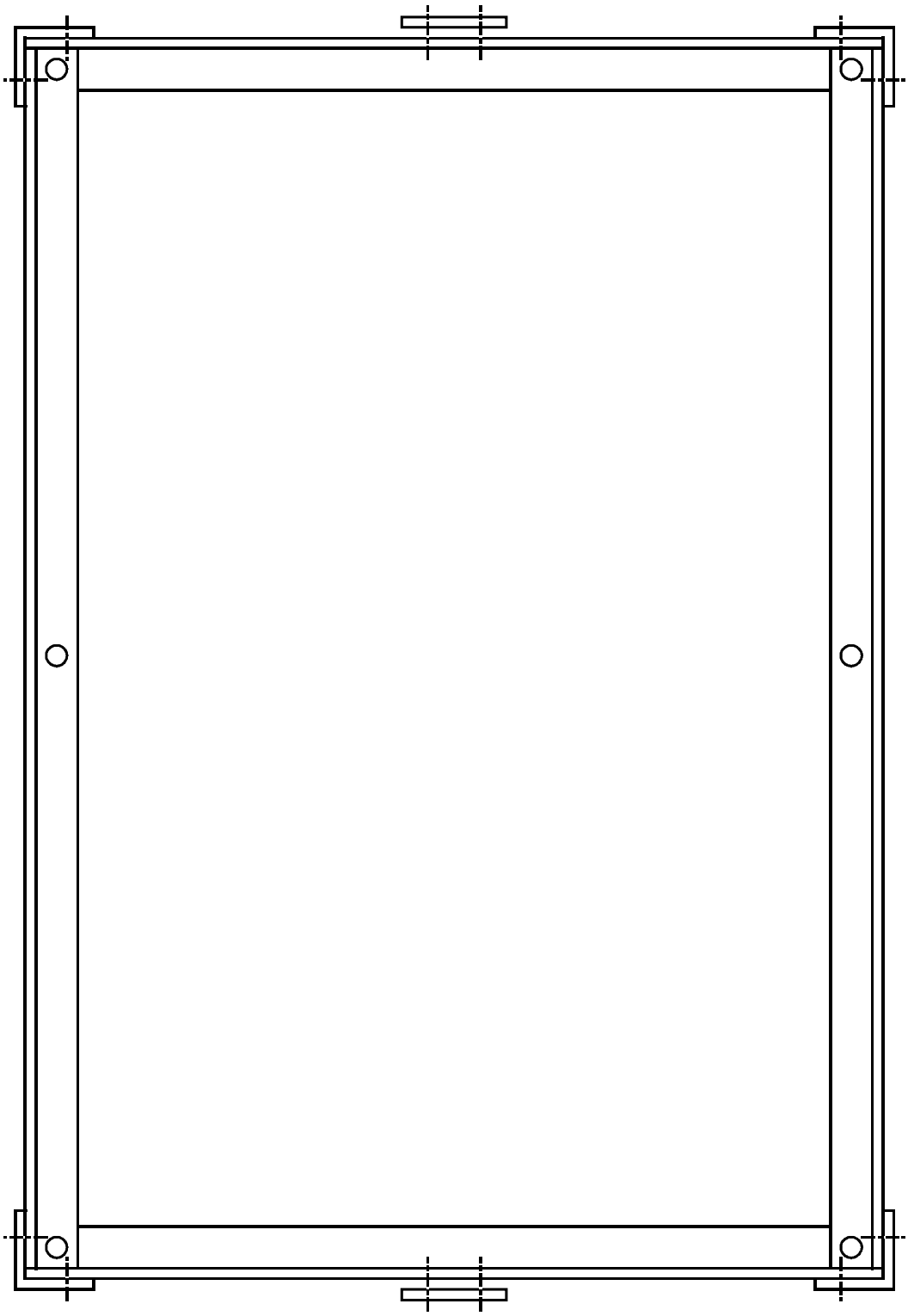


Abbildung 7.8: Antriebseinheit, Rahmen

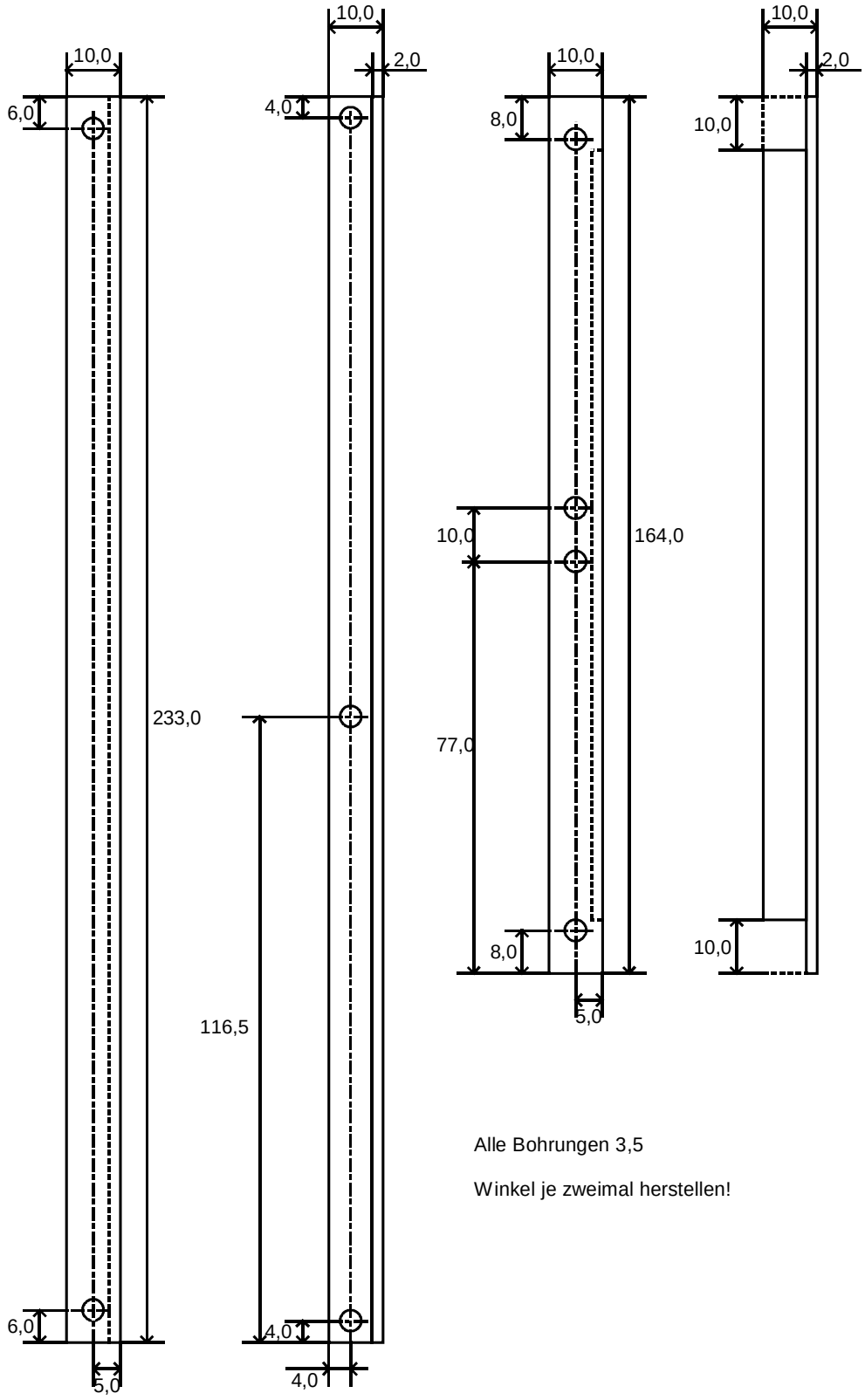
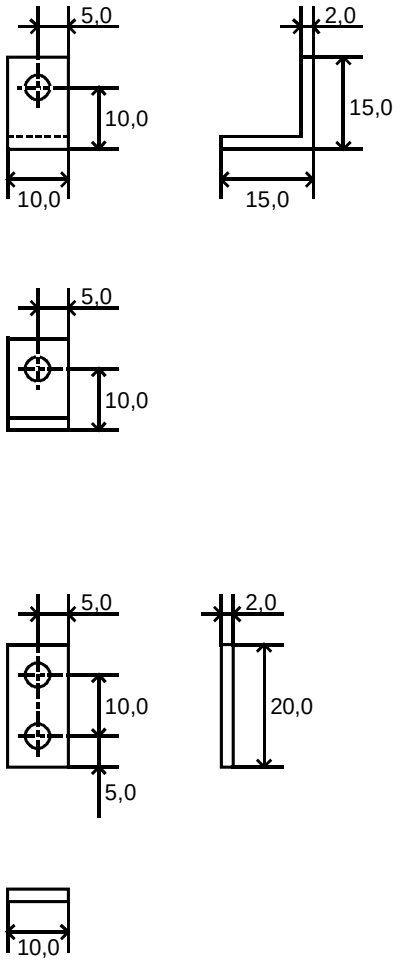


Abbildung 7.9: Konstruktionszeichnung: Rahmen



Bohrungen 3,5

Winkel viermal herstellen!

Bohrungen 3,5

Platten zweimal herstellen!

Abbildung 7.10: Konstruktionszeichnung: Halterungen

7.4 Elektronik

7.4.1 Schaltpläne

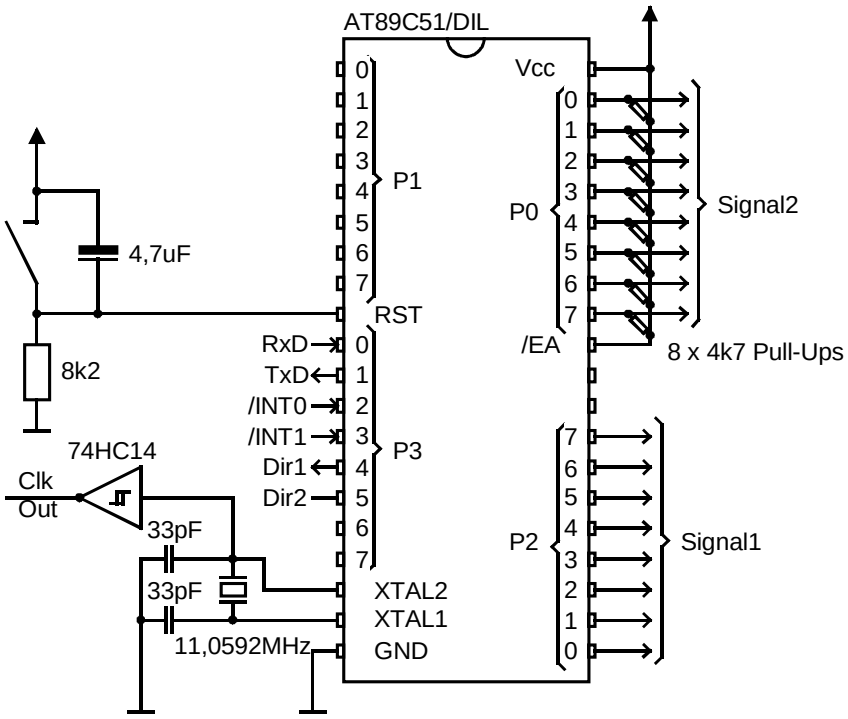


Abbildung 7.11: Beschaltung des Mikrocontrollers

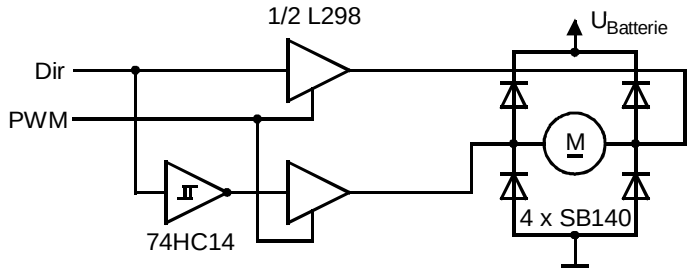


Abbildung 7.12: Schaltplan Motoransteuerung

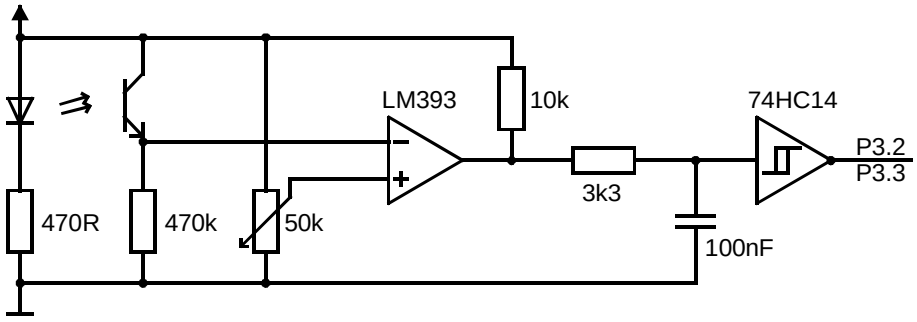


Abbildung 7.13: Schaltplan Drehzahlmessung

Der RC-Tiefpassfilter, bestehend aus dem 3k3 Widerstand und dem 100nF Kondensator wird nach folgender Formel dimensioniert:

$$f_G = \frac{1}{2\pi RC}$$

Die Grenzfrequenz sollte dabei bei ca. 500Hz liegen, und liegt bei der verwendeten Dimensionierung bei 482Hz.

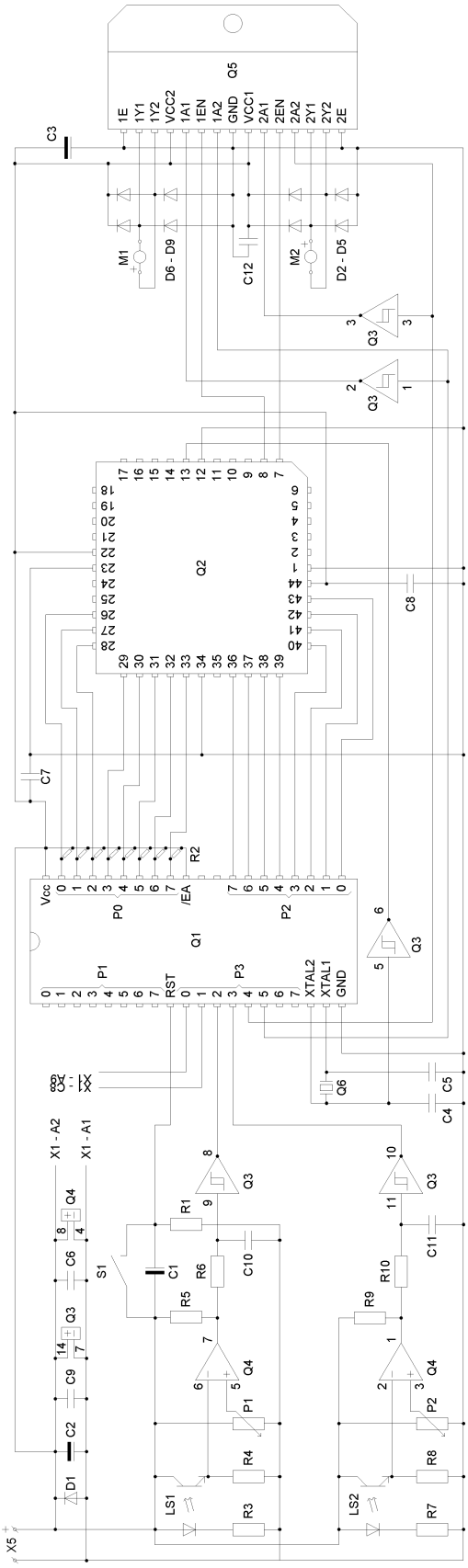


Abbildung 7.14: Gesamtschaltplan der Antriebseinheit

7.4.2 Bestückungsplan

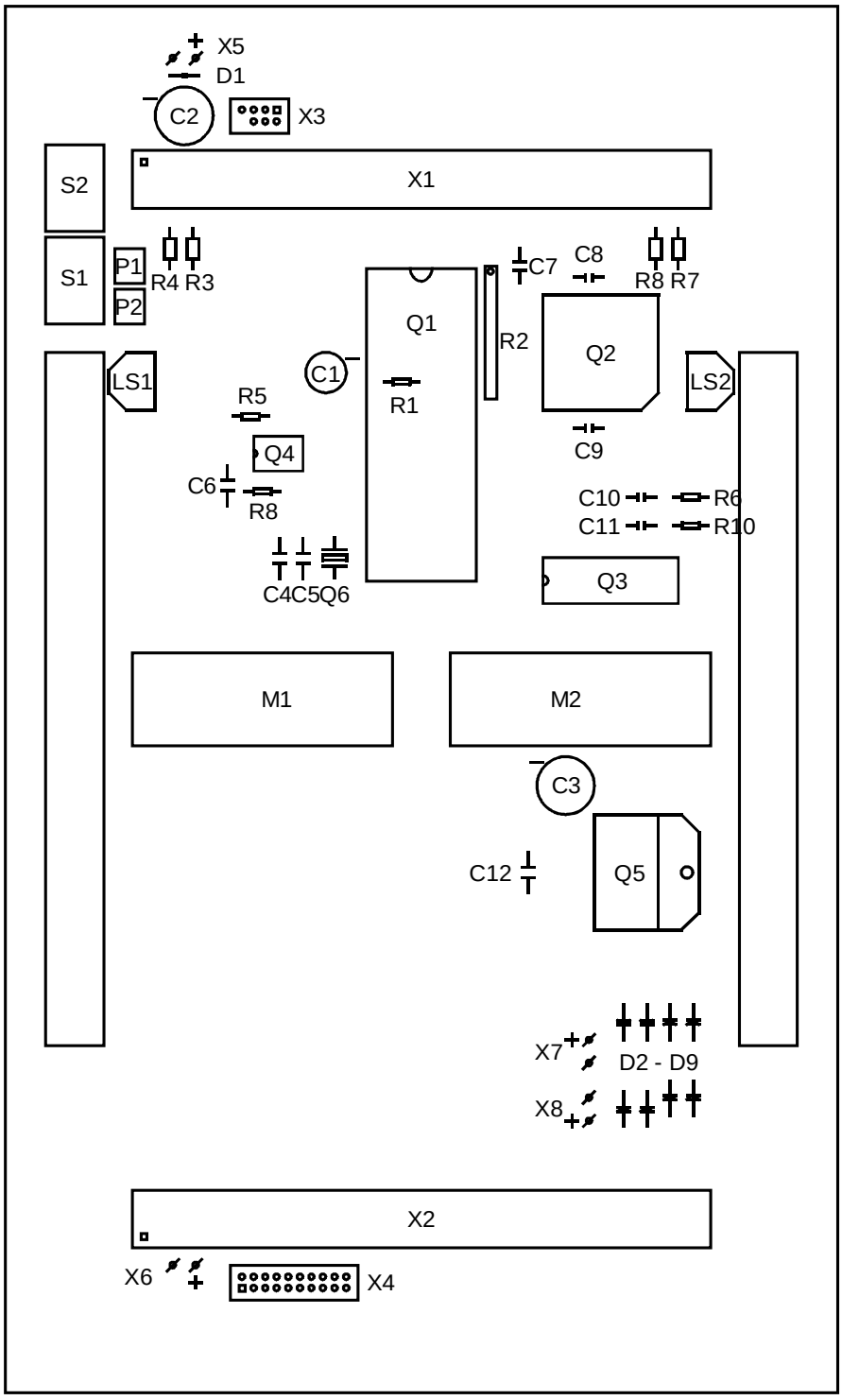


Abbildung 7.15: Bestückungsplan der Antriebseinheit

7.4.3 Stückliste

X1	96’er Stiftleiste
X2	64’er Buchsenleiste
X3	7’er Stiftleiste
X4	20’er Stiftleiste
X5, X6	2’er Stiftleiste, stehend
X7, X8	2’er Stiftleiste, liegend
S1, S2	Taster (Schliesser)
LS1, LS2	Reflexlichtschranke
M1, M2	Motor (Faulhaber 1624T 003S incl. Getriebe 16/5)
Q1	ATMEL 89C51 / DIL
Q2	AMD Mach 210AQ-15JC
Q3	74HC14
Q4	LM393N
Q5	L298N
Q6	11,0592 MHz Quarz
P1, P2	50k Wendel-Potentiometer, stehend
D1	1N4001
D2 - D9	SB140
C1	4,7uF Elko, stehend
C2	470uF Elko, stehend
C3	220uF Elko, stehend
C4, C5	33pF
C6 - C12	100nF
R1	8k2
R2	8 x 4k7 Netzwerk
R3, R7	470R
R4,R8	470k
R5,R9	10k
R6, R10	3k3

7.4.4 Pinbelegungen - ICs

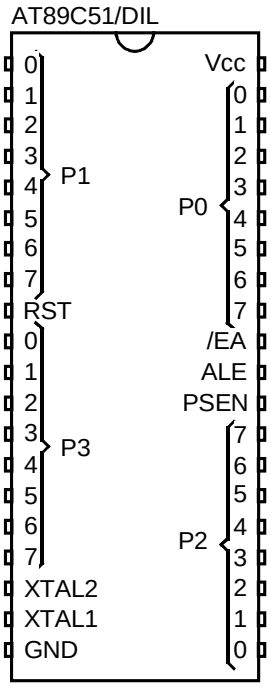


Abbildung 7.16: Pinbelegung Mikrocontroller 89C51

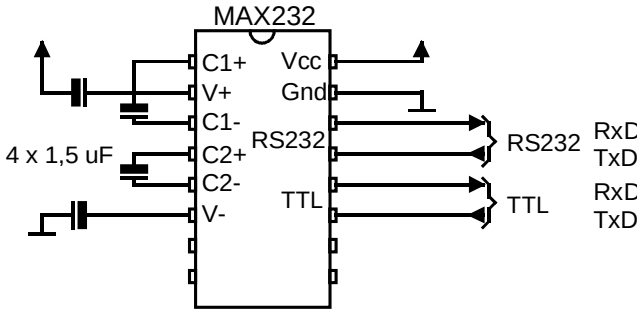


Abbildung 7.17: Pinbelegung Pegelumsetzer MAX232

Der Baustein MAX232 wird benötigt, wenn eine Verbindung zwischen dem Mikrocontroller und einem PC hergestellt werden soll. Während der Entwicklungsphase wurde diese Möglichkeit genutzt, um die Antriebseinheit und das serielle Protokoll zu testen.

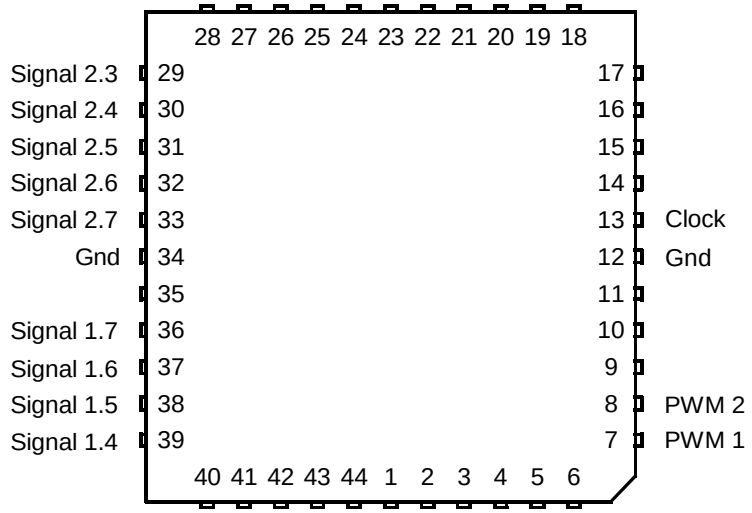


Abbildung 7.18: Pinbelegung Mach210

Abbildung 7.19: Pinbelegung Mach210 Sockel, Leiterbahnseite

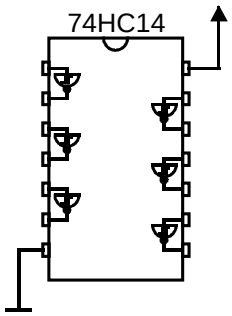


Abbildung 7.20: Pinbelegung Schmitt-Trigger 74HC14

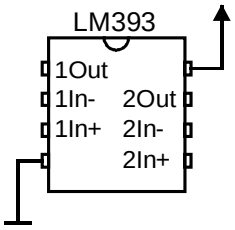


Abbildung 7.21: Pinbelegung Komparator LM393

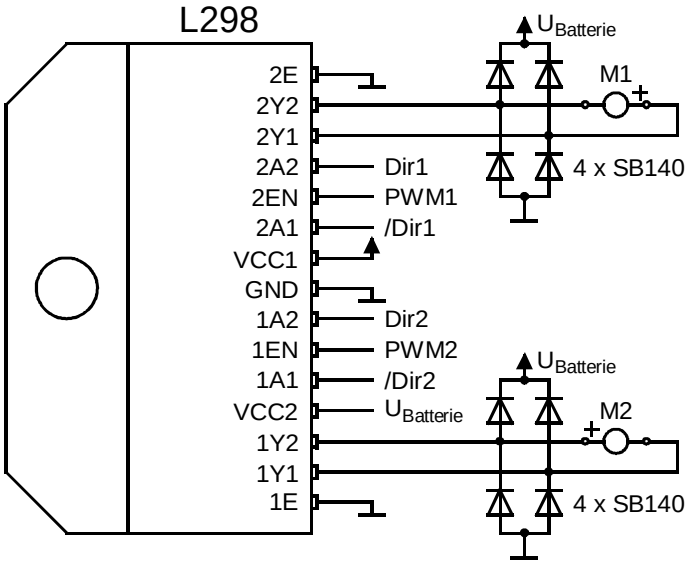


Abbildung 7.22: Pinbelegung Leistungstreiber L298

7.4.5 Pinbelegungen - Steckverbindungen

	A	B	C
1	GND	GND	GND
2	+5V	+5V	+5V
3	-	/ResIn	-
4	Tx0	Rx0	-
5	Tx1	Rx1	-
≈			
8	-	-	Rx2
9	Tx2	-	-
≈			
32	GND	GND	GND

Abbildung 7.23: Pinbelegung 96’er Stiftleiste

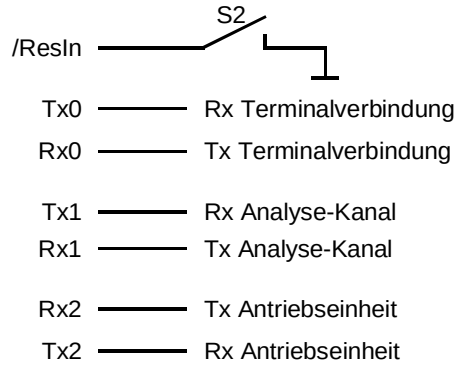


Abbildung 7.24: Signalplan 96’er Stiftleiste

	A	B/C
1	GND	GND
2	+5V	+5V
3	-	CS8
4	-	CS9
5	-	CS10
≈		
12	D8	D15
13	D9	D14
14	D10	D13
15	D11	D12
≈		
22	R/W	/Reset
≈		
25	A3	A4
26	A1	A2

Abbildung 7.25: Pinbelegung 64’er Buchsenleiste

Die Pinbelegungen der Steckverbindungen X1 (96’er Stiftleiste) und X2 (64’er Buchsenleiste) sind der Beschreibung des Steuerrechners entnommen (siehe [7]).

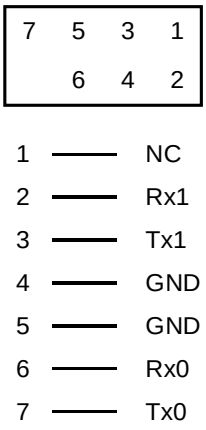


Abbildung 7.26: Pinbelegung der Steckverbindung zum Verwaltungsrechner

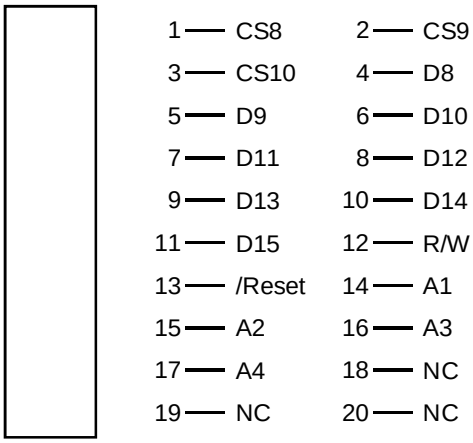


Abbildung 7.27: Pinbelegung der Steckverbindung zur Sensorikplatine

7.5 Software

7.5.1 Mikrocontroller 89C51

Die nun folgenden Einstellungen und Tabellen entstammen dem Buch MC-Tools2 von Otmar Feger, erschienen bei Feger+Reith, Traunstein (siehe [8]).

7.5.1.1 Special-Function-Register

Für die Initialisierung des Mikrocontrollers sind Einstellungen notwendig, die durchgeführt werden, indem bestimmte Werte in dafür vorhergesehene, sogenannte Special-Function-Register (SFR) geladen werden.

TMOD (Timer Modus Register) LSB

Timer1				Timer0			
Gate	C//T	M1	M0	Gate	C//T	M1	M0

Gate: externe Freigabe
C//T: 0 = intern; 1 = extern
M1: Modus
M0: Modus

Beide Timer werden im Modus Auto-Reload betrieben, das heisst, dass nach einem Überlauf ein Interrupt ausgelöst wird, und automatisch ein definierter Wert aus dem jeweiligen Auto-Reload-Register (TH1 bzw. TH0) in den Zähler des Timers geladen wird. Hierdurch wird ein gleichmässiges Zeitverhalten erreicht. Um diese Funktion zu erreichen, muss der Wert 22_{Hex} nach TMOD geladen werden.

TCON (Timer Condition Register) LSB

X	TR1	X	TR0	X	IT1	X	IT0

X: nicht verwendet
TR1: Timer 1 einschalten
TR0: Timer 0 einschalten
IT1: 1 = negative Flanke an externem Interrupteingang 1 löst Interrupt aus
0 = negativer Pegel an externem Interrupteingang 1 löst Interrupt aus
IT0: 1 = negative Flanke an externem Interrupteingang 0 löst Interrupt aus
0 = negativer Pegel an externem Interrupteingang 0 löst Interrupt aus

Beide Timer werden benutzt. Ausserdem soll jeweils eine negative Flanke an den externen Interrupteingängen einen Interrupt auslösen. Um die gewünschten Funktionen zu erhalten, müssen die Bits TR1, TR0, IT1 und IT0 auf TRUE gesetzt werden.

SCON (Serial-Port Condition Register) LSB

SM0	SM1	SM2	REN	X	X	TI	RI

X: nicht verwendet
SM0: Modus
SM1: Modus
SM2: 1 = Empfangsinterrupt wird nur bei gültigem Stoppbit gesetzt
0 = Empfangsinterrupt wird bei jedem Empfang gesetzt
REN: Empfang einschalten
TI: Sendeinterrupt, wird nach Senden eines Datums gesetzt,
muss vom Anwender per Software zurückgesetzt werden
RI: Empfangsinterrupt, wird nach Empfang eines Datums gesetzt,
muss vom Anwender per Software zurückgesetzt werden

Die serielle Schnittstelle soll als 8Bit UART mit einstellbarer Baudrate mit Timer 1 als Baudratengenerator arbeiten. Ausserdem soll nur bei Empfang eines gültigen Stoppbits ein Empfangsinterrupt ausgelöst werden. Diese Einstellungen werden erreicht, indem SM1, SM2 und REN auf TRUE, und SM0, RI und TI auf FALSE gesetzt werden.

Für die Einstellung der Baudrate ist zusätzlich notwendig, das Bit SMOD im Special-Function-Register PCON auf TRUE zu setzen (alle weiteren Bits von PCON werden nicht verwendet). Hierdurch wird eine Verdoppelung der Baudrate erreicht. Siehe hierzu die Formel zur Einstellung der Baudrate später in diesem Abschnitt.

IE (Interrupt Enable Register) LSB

EA	X	X	ES	ET1	EX1	ET0	EX0

X: nicht verwendet
EA: Freigabe aller individuell freigegebener Interrupts
ES: Freigabe Interrupt serielle Schnittstelle
ET1: Freigabe Interrupt Timer 1
EX1: Freigabe externer Interrupt 1
ET0: Freigabe Interrupt Timer 0
EX0: Freigabe externer Interrupt 0

Beide externen Interrupts werden benutzt, müssen also durch Setzten des entsprechenden Bits auf TRUE freigegeben werden.

Beide Timer werden benutzt, jedoch darf nur ET0 auf TRUE gesetzt werden. Timer 1 wird als Baudratengenerator genutzt. Während der Erstellung dieser Arbeit hat sich gezeigt, dass es zu Fehlern bei der 16Bit Multiplikation (das High-Byte des Ergebnisses wird nicht gesetzt) kommt, sobald Timer 1 als Baudratengenerator genutzt wird, und der Interrupt für Timer 1 freigegeben ist. Deshalb muss ET1 auf FALSE gesetzt werden.

Die serielle Schnittstelle wird genutzt und soll interruptfähig sein. Deshalb muss auch ES auf TRUE gesetzt werden.

Um alle individuell freigegebenen Interrupts nutzen zu können, müssen sie noch generell durch Setzten von EA auf TRUE freigegeben werden.

7.5.1.2 Serielle Schnittstelle und Timer

Die serielle Schnittstelle wird im Modus 1 als 8Bit UART mit variabler Baudrate mit Timer 1 als Baudratengenerator betrieben.

Es werden folgende Bits übertragen:

- 1 Startbit (Low)
- 8 Datenbits (LSB zuerst)
- 1 Stoppbit (High)

Die Baudrate der seriellen Schnittstelle wird nach der folgenden Formel berechnet:

$$Bd - Rate = \frac{1 + SMOD}{32} \times \frac{f_{OSZ}}{12 \times (256 - TH1)}$$

Die Baudrate war mit 19200Bd von dem Steuerrechner vorgegeben. Um diesen Wert zu erreichen, musste ein Quarz mit der Frequenz f_{osz} = 11,0592 MHz eingesetzt werden. Als Auto-Reload-Wert für Timer 1 ergibt sich dann TH1 = 253.

Timer 0 dient als interne Zeit. Im Auto-Reload-Modus wird mit in festen Zeitabständen, die von der Oszillatorfrequenz und dem Auto-Reload-Wert abhängen, ein Interrupt ausgelöst. Durch inkrementieren eines Zählers in der Interrupt-Service-Routine des Timers lässt sich so durch Abfrage des Zählerstandes ein Zeitbezug herstellen.

Die Anzahl der Interrupts pro Sekunde lässt sich wie folgt berechnen:

$$\frac{IRs}{sec} = \frac{f_{OSZ}}{12 \times (256 - TH0)}$$

Es war wegen der einfachen Teilbarkeit ein möglichst gerader Wert gewünscht. Gewählt wurde ein Wert von ca. 4 IRs / msec.

Nach der oben genannten Formel ergibt sich bei f_{osz} = 11,0592 MHz somit ein Wert von 25,6 für TH0. Da der Wert ganzzahlig sein muss, wurde der Wert TH0 = 26 gewählt, woraus sich eine Interruptrate von 4007 IRs / sec ergibt.

7.5.2 Programmierbarer Baustein Mach210

Für die Erzeugung der PWM-Signale war externe Elektronik notwendig. Um den Aufwand so gering wie möglich zu halten, und um so viel Strom wie möglich zu sparen, wurde auf ein programmierbaren Baustein des Typs Mach210 zurückgegriffen.

Der grosse Vorteil bei der Verwendung solch eines Bausteines ist, gerade bei der Erstellung eines Prototypen, die einfache Änderbarkeit durch simples Umprogrammieren.

Das Programm für den Zähler und für die Vergleicher wurde mit DSL von MIcroSim7.1 erstellt.

```
PROCEDURE PWM( INPUT clk, Signal1[7..0] , Signal2[7..0], rst;
               OUTPUT PWM1, PWM2 clocked_by clk reset_by rst);

node count[7..0] clocked_by clk reset_by rst;


count[7..0] = count[7..0] .+. 00000001b;

if count[7..0] < Signal1[7..0] then PWM1 = 1;
                               else PWM1 = 0;
end if;
if count[7..0] < Signal2[7..0] then PWM2 = 1;
                               else PWM2 = 0;
end if;

END PWM;
```

clk ist der Takteingang, rst der Reseteingang. Signal1 und Signal2 sind die 8Bit breiten Eingangswerte, PWM1 und PWM2 die Ausgänge. Der interne Knoten count ist der 8Bit Zähler, der mit clk getaktet wird, und durch rst zurückgesetzt wird. Die für die PWM notwendigen Vergleicher sind durch jeweils eine simple if-Abfrage realisiert.

Durch das Rücksetzen der Ausgangs-Flip-Flops mittels rst, ergibt sich automatisch ein Motorschutz, da das Ausgangssignal bei einem Reset auf Low gesetzt wird. Diese entspricht nicht der Reaktion der Ausgänge des Mikrocontrollers, welche bei einem Reset auf High gesetzt werden.

Um eine definierte Pinbelegung zu erhalten, musste die gewünschte Pinbelegung der Entwicklungssoftware (MicroSim 7.1) mitgeteilt werden. Diese geschieht mit Hilfe des sogenannten "Physical Information File". In dieser Datei werden unter anderem die Ein- und Ausgänge den gewünschten Pins zugeteilt. Die Software versucht dann beim Fitten des Bausteines die Wünsche des Entwicklers umzusetzen.

```
" PLSyn Physical Information file
" Optimization parmeters:

{
 MAX_SYMBOLS          20,
 MAX_PTERMS           8,
 POLARITY_CONTROL     TRUE,
 MAX_XOR_PTERMS       0,
 XOR_POLARITY_CONTROL FALSE
};


device
{mach_utilization 100}
target 'template mach210 jlcc-44-std';


"inputs

signal10:43;
signal11:42;
signal12:41;
signal13:40;
signal14:39;
signal15:38;
signal16:37;
signal17:36;

signal20:26;
signal21:27;
signal22:28;
signal23:29;
signal24:30;
signal25:31;
signal26:32;
signal27:33;


"outputs

pwm1:7;
pwm2:8;

end device;
```

7.5.3 Protokoll der Datenübertragung

Der Mikrocontroller erhält Datensätze von dem Steuerrechner, auf die jeweils, dem Datensatz entsprechend, geantwortet wird. Ein Datensatz besteht immer aus vier Byte, und wird immer mit einem ebenfalls aus vier Byte bestehenden Datensatz beantwortet.

Neben den normalen Fahrbefehlen, die sich aus den Daten Distance (zu fahrende Strecke), Speed (zu fahrende Geschwindigkeit), Angle (Lenkradeinschlag) und Number (Befehlsnummer) zusammensetzen, gibt es noch zwei besondere Datensätze.
Zum einen die Statusabfrage, mit welcher der Steuerrechner die Einsatzbereitschaft der Antriebseinheit abfragt. Die Antriebseinheit testet ihre Einsatzbereitschaft beim Hochlaufen durch einen Selbsttest. Die Statusabfrage ist durch viermal FF_{hex} codiert.
Zum anderen den Stoppbefehl, mit dem die Antriebseinheit sofort zum Anhalten gebracht werden soll. Der Stoppbefehl ist durch dreimal 00_{Hex}, gefolgt von einem irrelevanten Byte, codiert.

Ansonsten sind nur folgende Einschränkungen bei der Gültigkeit der Daten zu beachten: Speed muss zwischen 10 und 120 liegen, Angle muss zwischen 0 und 180 liegen.

Im ersten Byte des Antwortdatensatzes ist immer die Art der Antwort codiert. Das zweite und dritte Byte dient der Übertragung von Parametern der Antwort, wobei eventuell nur ein Parameter notwendig ist. In diesem Falle wird eine 0 als zweiter Parameter gesendet. Das letzte Byte ist immer die Nummer des Befehls, auf den geantwortet wird.

Im folgenden ist das Protokoll mit der Bedeutung der jeweiligen Daten aufgelistet:

1.Byte	Bedeutung	
0:	Antwort auf Statusabfrage	
	1. Parameter:	10_{Hex}: Status: OK
		11_{Hex}: Status: Fehler
	2. Parameter:	nicht benötigt
	Nummer:	Nummer des Befehles
1:	Antwort: Fehler	
	1. Parameter:	1: fehlerhafte Daten erhalten
	2. Parameter:	nicht benötigt
	Nummer:	Nummer des Befehles
2:	Antwort auf Fahrbefehl	
	1. Parameter:	zurückgelegte Strecke, linkes Rad
	2. Parameter:	zurückgelegte Strecke, rechtes Rad
	Nummer:	Nummer des vorherigen Befehles
3:	Antwort bei Erreichen der zu fahrenden Strecke	
	1. Parameter:	zurückgelegte Strecke, linkes Rad
	2. Parameter:	zurückgelegte Strecke, rechtes Rad
	Nummer:	Nummer des Befehles

Da die Zeichen 11_{Hex} und 13_{Hex} für das Softwareprotokoll (Xon/Xoff) benötigt werden, wird zu allen zu sendenden Bytes ein Codeoffset von 30_{Hex} addiert. Der gleiche Codeoffset wird bei der Decodierung der empfangenen Daten von diesen subtrahiert.

Dieses Verschieben aller genutzten Codewörter kann geschehen, da genügend Zeichen vorhanden sind. Wenn mehr Zeichen als Wortschatz benötigt werden, können die Zeichen Xon und Xoff durch Umcodierung ausgeschlossen werden.

7.6 Programm des Mikrocontrollers

Bei dem verwendeten Mikrocontroller handelt es sich um einen Typen, der ohne externen Speicher arbeiten kann. Das hat den Vorteil, dass er mit geringer, externer Beschaltung arbeiten kann. Der Nachteil ist jedoch die sehr begrenzte Speicherkapazität.

Im internen RAM sind alle Register (auch die SFRs), alle Variablen sowie der Stack untergebracht. Um diesen sowenig wie möglich nutzen zu müssen, wurden alle Variablen global angelegt. Ausserdem werden Unterprogramme nur in der Initialisierungsphase oder als Interrupt-Service-Routinen verwendet.

7.6.1 Programmablaufplan

Initialisiere Ausgänge
Initialisiere Variable
Initialisiere Timer
Initialisiere serielle Schnittstelle
Initialisiere Flags
Initialisiere Interrupts
Führe Antriebstest aus

Wiederhole

Wenn Daten in Sendepuffer
dann: Sende Daten
Wenn zu fahrende Entfernung erreicht
dann: Sollgeschwindigkeit = 0
Wenn Regelungszeitfenster abgelaufen
dann: Reglerausgangsgrössen neu berechnen
Wenn Datensatz komplett empfangen
dann: Dekodiere Datensatz
Quittiere Datensatz

für immer

7.6.2 Programmlisting

Die Beschreibung der verwendeten Programmiersprache PAS51 ist dem Manual des Herstellers (siehe [9]) zu entnehmen.

```
{$M 0 -1}               { nur interner Speicher            }

{*************** Compiler Directive, muss am Anfang stehen ***************}


{************************************************************************}
{                                                                        }
{                    Programm zur Ansteuerung von MARVIN                 }
{                                                                        }
{                          Mobiler                                       }
{                            Autonomer                                   }
{                              Roboter                                   }
{                                Von                                     }
{                                  IIs                                   }
{                                    Neuentwickelt                       }
{                                                                        }
{************************************************************************}
{                                                                        }
{                 Diplomarbeit an der Fachhochschule Wedel               }
{                    Fachbereich Technische Informatik                   }
{                                                                        }
{************************************************************************}
{                                                                        }
{    Betreuer: Prof. Dr. Uelzmann                                        }
{                                                                        }
{    Autor:    Henry G. Kleta                                            }
{              ii6582                                                    }
{                                                                        }
{    Datum:    2.7.1997                                                  }
{                                                                        }
{************************************************************************}


{****************************** Variable *********************************}

VAR

{Output}
    Signal1    : byte at P2;           { linker Antrieb    }
    Direction1 : boolean at P3.4;
    Signal2    : byte at P0;           { rechter Antrieb   }
    Direction2 : boolean at P3.5;

{interne Zeit}
    Time  : word;       { Zeit fuer Regelung (I-Anteil)    }
    Time1 : word;       { Zeit fuer Frequenzmessung links  }
    Time2 : word;       { Zeit fuer Frequenzmessung rechts }

{gemessene Zeiten zwischen Interrupts}
    T1 : word;          { 1/Frequenz linker Antrieb        }
    T2 : word;          { 1/Frequenz rechter Antrieb       }

{Wegstreckenzaehler}
    Distance1 : word;   { Impulsanzahl links               }
    Distance2 : word;   { Impulsanzahl rechts              }

{Regelungsgroessen}
    w1    : byte;       { Soll-Wert                        }
    w2    : byte;
    e1    : integer;    { Reglereingangsgroesse            }
    e2    : integer;
    e1old : integer;    { Zwischenspeicher fuer I-Anteil   }
    e2old : integer;
    y1    : integer;    { Reglerausgangsgroesse            }
    y2    : integer;
    y1old : integer;    { Zwischenspeicher fuer I-Anteil   }
    y2old : integer;
    x1    : byte;       { Ist-Wert                         }
    x2    : byte;
    delta : integer;    { Laufleistungsunterschied         }

{Sendepuffer}
```

```
    BufferOut : array[1..4] of byte;         { Sendepuffer }

{Inputs (via serial Port)}
    Distance  : byte;   { zurueckzulegende Strecke         }
    Speed     : byte;   { zu fahrende Geschwindigkeit      }
    Angle     : byte;   { Lenkradeinschlag                 }
    Number    : byte;   { Befehlsnummer                    }
    OldNumber : byte;   { alte Befehlsnummer               }

{Hilfsvariable}
    Help      : word;   { Hilfsvariable fuer Berechnungen  }

{Flags}
    ImOk          : boolean;   { Status von Marvin         }
    BytesReceived : byte;      { empfangene Byteanzahl     }
    DataValid     : boolean;   { Datensatz komplett        }
    DataToSend    : boolean;   { Daten zu Senden           }
    ReadyToSend   : boolean;   { Sendebereit ja/nein       }
    XoffReceived  : boolean;   { Empfangsbereit ja/nein    }


{******************************* Konstante ******************************}


CONST

    Xon         : byte = $11;  { serielles Protokoll       }
    Xoff        : byte = $13;

    CodeOffset  : byte = $30;  { Codeoffset                }

    State       : byte = $00;  { Antwort   : Status        }
    Error       : byte = $01;  { Antwort   : Fehler        }
    Answer      : byte = $02;  { Antwort   : normal        }
    Complete    : byte = $03;  { Antwort   : Befehl fertig }
    DataInvalid : byte = $01;  { Parameter : falsche Daten }
    Ok          : byte = $10;  { Parameter : Ok            }
    NotOk       : byte = $11;  { Parameter : nicht Ok      }

    MinSpeed    : byte = 10;   { minimal fahrbare          }
                               { Geschwindigkeit           }
    MaxSpeed    : byte = 120;  { maximal fahrbare          }
                               { Geschwindigkeit           }
    MaxAngle    : byte = 180;  { max Lenkradeinschlag      }

    MaxSignal   : byte = 150;  { Wert zum Motorschutz      }
                               { wie folgt zu berechnen:   }
                               { Umax / Vcc * 256          }
    TestSpeed   : byte = 50;   { Geschwindigkeit fuer Test }

    DFactor     : byte = 5;    { min moegliche Entfernung  }

    IntsPerSec  : word = 4007; { Interrupts pro Sekunde    }
    ControlTime : word = 200;  { Intervall fuer I-Regler   }
    TestTime    : word = 1002; { Zeit fuer Antriebstest    }

    Vp          : integer = 1; { Verstaerkung P-Anteil     }
    Vi          : integer = 4; { Parameter I-Anteil        }
    Di          : integer = 5; { Vi / Di = ( T / Tn ) - 1  }


{ Tabelle zur Umrechnung Lenkradeinschlag -> Leistungen der Motoren
  256 entspricht 100%; Vorzeichen entspricht Fahrtrichtung }

    factor : array[0..180] of integer = ( 256,
                                          250, 243, 237, 230,
                                          224, 218, 211, 205,
                                          198, 192, 186, 179,
                                          173, 166, 160, 154,
                                          147, 141, 134, 128,
                                          122, 115, 109, 102,
                                           96,  90,  83,  77,
                                           70,  64,  58,  51,
                                           45,  38,  32,  26,
                                           19,  13,   6,   0,
                                          -51,-102,-154,-205,
                                         -256,
                                          205, 154, 102,  51,
                                            0,  -6, -13, -19,
                                          -26, -32, -38, -45,
                                          -51, -58, -64, -70,
                                          -77, -83, -90, -96,
                                         -102,-109,-115,-122,
                                         -128,-134,-141,-147,
                                         -154,-160,-166,-173,
                                         -179,-186,-192,-198,
                                         -205,-211,-218,-224,
```

```
                                        -230,-237,-243,-250,
                                        -256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                        -256,-256,-256,-256,
                                         256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256, 256, 256, 256,
                                         256);

{************************ Initialisierungs Prozeduren ********************}


PROCEDURE InitTimer;
{Initialisiert Timer0 : interne Zeit
               Timer1 : Baudratengenerator}
BEGIN
  TMOD := $22;          { Modus Auto-Reload                }
  TH0  := 26;           { ca. 4 Interrupts pro msec        }
  TR0  := true;         { Anschalten Timer0                }
  TH1  := 253;          { 19200Bd bei SMOD=1, f=11,0592MHz }
  TR1  := true;         { Anschalten Timer1                }
END;{of procedure}

PROCEDURE InitSerial;
{Initialisiert serielle Schnittstelle}
BEGIN
  SM0  := false;
  SM1  := true;         { 8bit UART / variable Baudrate    }
  SM2  := true;         { IR nur bei gueltigem Stoppbit    }
  RI   := false;        { Receive-Interrupt loeschen       }
  TI   := false;        { Transmit-Interrupt loeschen      }
  REN  := true;         { Empfang einschalten              }
  PCON := PCON or $80;  { SMOD = 1                         }
END;{of procedure}

PROCEDURE InitInterrupts;
{Initialisiert Interrupts}
BEGIN
  IT0 := true;          { IR bei neg. Flanke an P3.2       }
  EX0 := true;          { Freigabe externer Interrupt 0    }
  IT1 := true;          { IR bei neg. Flanke an P3.3       }
  EX1 := true;          { Freigabe externer Interrupt 1    }
  ET0 := true;          { Freigabe Interrupt Timer0        }
  ET1 := false;         { Sperrung Interrupt Timer1        }
  ES  := true;          { Freigabe Interrupt serial Port   }
  EA  := true;          { Interrupts freigeben             }
END;{of procedure}

PROCEDURE InitPorts;
{Initialisiert Ausgaenge}
BEGIN
  P1         := $00;
  P3         := $FF;    { P3 Alternativfunktionen setzen   }
  Signal1    := 0;      { linken Antrieb initialisieren    }
  Direction1 := true;
  Signal2    := 0;      { rechten Antrieb initialisieren   }
  Direction2 := true;
END;{of procedure}

PROCEDURE InitFlags;
{Initialisiert Flags}
BEGIN
  ImOk          := false;
  BytesReceived := 0;
  DataValid     := false;
  DataToSend    := false;
  ReadyToSend   := true;
  XoffReceived  := false;
```

```
END;{of procedure}

PROCEDURE InitVars;
{Initialisiert Variable}
BEGIN
  Help      := 0;
  Time      := 0;
  Time1     := 0;
  Time2     := 0;
  Distance1 := 0;
  Distance2 := 0;
  T1        := 0;
  T2        := 0;
  w1        := 0;
  w2        := 0;
  e1        := 0;
  e2        := 0;
  e1old     := 0;
  e2old     := 0;
  y1        := 0;
  y2        := 0;
  y1old     := 0;
  y2old     := 0;
  x1        := 0;
  x2        := 0;
  delta     := 0;
  Distance  := 255;
  Speed     := 0;
  Angle     := 0;
  Number    := 0;
  OldNumber := 0;
END;{of procedure}

PROCEDURE TestDrives;
{Flag IamOk wird gesetzt, wenn Antriebe reagieren}
BEGIN
  Direction1 := false;            { Punktdrehung links     }
  Direction2 := true;
  Signal1    := TestSpeed;
  Signal2    := TestSpeed;
  Time       := 0;
  repeat until Time >= TestTime;  { Zeitverzoegerung       }
  Signal1    := 0;                { Stop                   }
  Signal2    := 0;
  Time       := 0;
  repeat until Time >= TestTime;  { Zeitverzoegerung       }
  Direction1 := true;             { Punktdrehung rechts    }
  Direction2 := false;
  Signal1    := TestSpeed;
  Signal2    := TestSpeed;
  Time       := 0;
  repeat until Time >= TestTime;  { Zeitverzoegerung       }
  Signal1    := 0;                { Stop                   }
  Signal2    := 0;
  Time       := 0;
  repeat until Time >= TestTime;  { Zeitverzoegerung       }
  ImOk       := ((Distance1 > 0) and (Distance2 > 0));
  Distance1  := 0;                { Zaehler zuruecksetzen  }
  Distance2  := 0;
  Time       := 0;
END;{of procedure}


{************************ Interrupt Service Routinen ********************}


PROCEDURE interrupt timer0;
{IR - Routine
 interne Zeiten werden inkrementiert}
BEGIN
  inc(Time);
  inc(Time1);
  inc(Time2);
END;{of procedure}

PROCEDURE interrupt int0;
{IR - Routine
 Messung der Zeit zwischen zwei Interrupts (Frequenz)
 und Zaehlen der Interrupts (gefahrene Strecke) linker Antrieb }
BEGIN
  inc(Distance1);
  T1    := Time1;
  Time1 := 0;
END;

PROCEDURE interrupt int1;
{IR - Routine
```

```
 Messung der Zeit zwischen zwei Interrupts (Frequenz)
 und Zaehlen der Interrupts (gefahrene Strecke) rechter Antrieb }
BEGIN
  inc(Distance2);
  T2    := Time2;
  Time2 := 0;
END;

PROCEDURE interrupt serial;
{IR - Routine
 Speichern der empfangenen Bytes in den entsprechenden Variablen }
BEGIN
  if RI then            { Byte empfangen }
          BEGIN
            if SBUF = Xoff then XoffReceived := true
               else if SBUF = Xon then XoffReceived := false
               else BEGIN
                      case BytesReceived of
                           0 : BEGIN
                                 DataValid := false;
                                 Distance  := SBUF;
                                 inc(BytesReceived);
                               END;{of 0}
                           1 : BEGIN
                                 Speed := SBUF;
                                 inc(BytesReceived);
                               END;{of 1}
                           2 : BEGIN
                                 Angle := SBUF;
                                 inc(BytesReceived);
                               END;{of 2}
                           3 : BEGIN
                                 OldNumber := Number;
                                 Number := SBUF;
                                 inc(BytesReceived);
                               END;{of 3}
                      END;{of case}
                    END;{of else}
            RI := false;
          END;{of if}
  if TI then            { Byte gesendet }
          BEGIN
            ReadyToSend := true;
            TI := false;
          END;{of if}
END;{of procedure}


{***************************** Hauptprogramm ******************************}


BEGIN


{************************** Initialisierung ***************************}

  InitPorts;
  InitVars;
  InitTimer;
  InitSerial;
  InitFlags;
  InitInterrupts;
  TestDrives;


{************************ Hauptprogrammschleife ************************}

  WHILE true DO
        BEGIN


{***** Senden von Daten aus dem Sendepuffer wenn vorhanden und bereit *****}

          if (DataToSend and ReadyToSend and (not XoffReceived)) then
             BEGIN
               for Help := 1 to 4 do
                   BEGIN
                     ReadyToSend := false;
                     SBUF := BufferOut[Help];
                     repeat until ReadyToSend;
                   END;{of for}
               DataToSend := false;
             END;{of if}


{***** Anhalten und Meldung bei Erreichen der zu fahrenden Entfernung *****}
```

```
          if (DataValid and
              (((Distance1 div DFactor) > Distance) or
               ((Distance2 div DFactor) > Distance))) then
             BEGIN
               w1 := 0;
               w2 := 0;
               BufferOut[1] := Complete + CodeOffset;
               BufferOut[2] := LO(Distance1 div DFactor) + CodeOffset;
               Distance1  := 0;
               BufferOut[3] := LO(Distance2 div DFactor) + CodeOffset;
               Distance2  := 0;
               BufferOut[4] := Number;
               DataToSend := true;
               Distance   := $FF;
             END;{of if}


{******************** Regelung (digitaler PI-Regler) **********************}

          if Time >= ControlTime
             then BEGIN
{ I-Anteil }        if (w1 <> 0)
                       then BEGIN
                              y1old := y1;
                              e1old := e1;
                            END{of then}
                       else BEGIN
                              y1old := 0;
                              e1old := 0;
                              Distance1 := 0;
                            END;{of else}
                    if (w2 <> 0)
                       then BEGIN
                              y2old := y2;
                              e2old := e2;
                            END{of then}
                       else BEGIN
                              y2old := 0;
                              e2old := 0;
                              Distance2 := 0;
                            END;{of else}

{ Ist-Wert }        if ((T1 = 0) or (T1 > IntsPerSec) or (Time1 > IntsPerSec))
                       then x1 := 0
                       else x1 := LO(word(IntsPerSec div T1));
                    if ((T2 = 0) or (T2 > IntsPerSec) or (Time2 > IntsPerSec))
                       then x2 := 0
                       else x2 := LO(word(IntsPerSec div T2));

{ delta Distance }  if ((w1 > 0) and (w1 = w2))
                       then delta := integer(Distance1 - Distance2)
                       else delta := 0;

{ Regler Input }    e1 := integer(w1) - integer(x1) - delta;
                    e2 := integer(w2) - integer(x2) + delta;

{ Regler Output }   y1 := Vp * (e1 - (Vi * e1old div Di)) + y1old;
                    y2 := Vp * (e2 - (Vi * e2old div Di)) + y2old;

{ Ausgangssignal }  if y1 > integer(MaxSignal)
                       then BEGIN
                              y1 := integer(MaxSignal);
                              Signal1 := MaxSignal;
                            END{of then}
                       else if y1 < 0
                               then Signal1 := 0
                               else Signal1 := LO(word(y1));
                    if y2 > integer(MaxSignal)
                       then BEGIN
                              y2 := integer(MaxSignal);
                              Signal2 := MaxSignal;
                            END{of then}
                       else if y2 < 0
                               then Signal2 := 0
                               else Signal2 := LO(word(y2));

{ Reset Time }      Time := 0;
                  END;{of if}


{************* Decodieren und Quittieren der empfangenen Daten ************}

          if (BytesReceived = 4) then
             BEGIN
               case Speed of
{ Statusabfrage }   $FF : BEGIN
                            if ((Distance = $FF) and
```

```
                                (Angle = $FF)    and
                                (Number = $FF))
{ Meldung Ok }                 then BEGIN
                                       w1 := 0;
                                       w2 := 0;
                                       BufferOut[1] := State + CodeOffset;
                                       if ImOk
                                          then BufferOut[2] := Ok + Codeoffset
                                          else BufferOut[2] := NotOk + CodeOffset;
                                       BufferOut[3] := Codeoffset;
                                       BufferOut[4] := Number;
                                     END{of if}
{ Meldung falsche Daten }      else BEGIN
                                       BufferOut[1] := Error + CodeOffset;
                                       BufferOut[2] := DataInvalid + CodeOffset;
                                       BufferOut[3] := CodeOffset;
                                       BufferOut[4] := Number;
                                     END;{of else}
                           END;{of $FF}
{ Stop }     CodeOffset : BEGIN
                            if ((Distance = CodeOffset) and
                                (Angle    = CodeOffset)) then
                               BEGIN
                                 w1 := 0;
                                 w2 := 0;
                                 BufferOut[1] := Complete + CodeOffset;
                                 BufferOut[2] := LO(Distance1 div DFactor) + CodeOffset;
                                 Distance1    := 0;
                                 BufferOut[3] := LO(Distance2 div DFactor) + CodeOffset;
                                 Distance2    := 0;
                                 BufferOut[4] := OldNumber;
                               END;{of if}
                           END;{of CodeOffset}
{ Fahrbefehl }      else  BEGIN
                            Distance     := Distance - CodeOffset;
                            Speed        := Speed - CodeOffset;
                            Angle        := Angle - CodeOffset;
                            if ((Speed > MaxSpeed) or
                                (Speed < MinSpeed) or
                                (Angle > MaxAngle))
{ Meldung falsche Daten }       then BEGIN
                                       BufferOut[1] := Error + CodeOffset;
                                       BufferOut[2] := DataInvalid + CodeOffset;
                                       BufferOut[3] := CodeOffset;
                                       BufferOut[4] := Number;
                                     END{of then}
{ Dekodieren }                  else BEGIN
                                       Direction1   := (factor[Angle] > 0);
                                       Help         := word(abs(factor[Angle]));
                                       Help         := Help * word(Speed);
                                       w1           := HI(Help);
                                       Direction2   := (factor[180-Angle] > 0);
                                       Help         := word(abs(factor[180-Angle]));
                                       Help         := Help * word(Speed);
                                       w2           := HI(Help);
                                       BufferOut[1] := Answer + CodeOffset;
                                       BufferOut[2] := LO(Distance1 div DFactor) + CodeOffset;
                                       Distance1    := 0;
                                       BufferOut[3] := LO(Distance2 div DFactor) + CodeOffset;
                                       Distance2    := 0;
                                       BufferOut[4] := OldNumber;
                                       DataValid    := true;
                                     END;{of else}
                           END;{of else}
              END;{of case}
              BytesReceived := 0;
              DataToSend := true;
            END;{of if}
        END;{of while}
END.{of program}

{***************************** Programmende ******************************}
```