Reithofer, Gerhard

Drehwinkelsteuerung mit Mikrocontroller und I^2C-Schnittstelle

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA (FH)

UNIVERSITY OF APPLIED SCIENCES

Informationstechnik und Elektrotechnik

Mittweida, 2006

Erstprüfer: Prof. Dr.-Ing. Olaf Hagenbruch

Zweitprüfer: Dipl.-Ing. (FH) Heiko Polster

vorgelegte Arbeit wurde verteidigt am: 13. Oktober 2006

Überarbeitung am: 22. November 2006

Bibliographische Beschreibung:

Reithofer Gerhard:

Drehwinkelsteuerung mit Mikrocontroller und I^2C-Schnittstelle - 2006 - 88 S.

Mittweida, Hochschule Mittweida (FH), Fachbereich Informationstechnik und Elektrotechnik, Diplomarbeit, 2006

Referat:

Das Ziel dieser Diplomarbeit ist es, mittels eines Mikrocontrollers unter Verwendung von kostengünstigen Bauteilen, eine intelligente Motorsteuerung zu entwickeln und zu implementieren. Die zusätzliche Forderung nach hoher Kontroll-Qualität, die üblicherweise durch Verwendung teurer Hardware-Komponenten erzielt wird, ist ein weiterer wesentlicher Punkt der aufgezeigten Lösung.

Ein zusätzlicher Aspekt, der für eine intelligente Lösung spricht, liegt in der großen Flexibilität der Softwarekomponente, bei der - im Extremfall - die Implementierung neuer Funktionen auf die Entwicklung einer neuen Software reduziert werden kann. Das Weiterverwenden der Hardware-Komponenten vermindert in diesem Fall die Entwicklungskosten und -zeiten.

Die Verwendung von Standardschnittstellen wiederum führt zu einer Modularisierung der Steuerungslösung, bei der die einzelnen Hardwarekomponenten ausgetauscht werden können, ohne den Rest der Steuerung zu beeinflussen. Auch die Softwarekomponente soll möglichst weitgehend modularisiert werden, um einerseits eine universell verwendbare Basisfunktionalität zur Verfügung zu stellen (zB. GUI Komponenten, virtuelles I/O-Konzept, usw.), um auch diese möglichst universell einsetzen oder leicht durch andere Module ersetzen zu können.

Als Ergebnis soll eine Methode zur Verfügung stehen, die Entwicklungkosten und Zeiten minimieren, die Anforderung an Flexibilität und Langzeit-Ersatzteil-Versorgungssicherheit sicherstellen und dennoch hohe Kontrollanforderungen erfüllt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die Idee zur Umsetzung dieser Diplomarbeit ergab sich bei einer Projektarbeit an der TU-Graz, die eine Erweiterung der Fähigkeiten einer bestehenden Versuchsanlage zum Ziel hatte. Ursprünglich war geplant, ein Verfahren zu entwickeln, das es ermöglicht, die Dichte von Filterstaubschichten zu messen. Dieses Projekt führte leider zu keinen verwertbaren Ergebnissen, um eine Diplomarbeit zu rechtfertigen, jedoch kam man in dieser Untersuchung zur Einsicht, dass in jedem Fall eine funktionelle Erweiterung der Anlage für die geplanten Versuchsreihen notwendig sei. Dies führte letztendlich zur Aufgabenstellung der in dieser Diplomarbeit behandelten Entwicklung.

Dieses Kapitel liefert einen Überblick über die bestehende Versuchsanlage, deren Einsatzweck und Funktionalität und über die grundlegenden „neuen” Anforderungen an die Anlage, deren Umsetzung diese Arbeit beschreibt.

1.1. Die Versuchsanlage

Das Institut für „Apparatebau, Mechanische Verfahrenstechnik und Feuerungstechnik” AMFT in der Infeldgasse 21b an der TU-Graz verwendet diese Versuchsanlage zur Grundlagenforschung im Bereich des Verhaltens von Stofffiltern bei der Reinigung von Gasen. Die Anlage wurde 2004 konzipiert und umgesetzt.

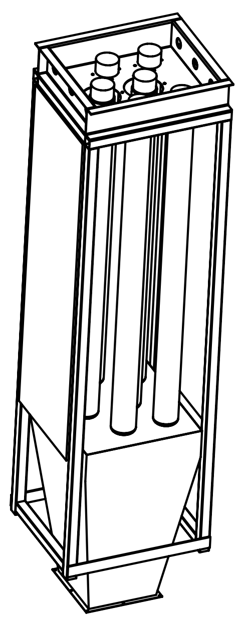


Abbildung 1.1.: 3D-Darstellung der Filter-Versuchsanlage

Im Prinzip besteht die Versuchanlage aus einer hermetisch dichten Filterkammer, in die mehrere Stofffilterschläuche eingesetzt werden können (Abb 1.1). Die Vorderseite ist mit einer Plexiglasplatte gedichtet und ermöglicht die optische Kontrolle der Anlage und der darin befindlichen Komponenten (Staubfilter).

Ein Umluftsystem erzeugt mittels eines Kompressors eine Druckdifferenz zwischen Kammereingang und Kammerausgang, um industrielle Abgasreinigungssysteme, wie sie z. B. in thermischen Kraftwerken eingesetzt werden, zu simulieren. Auf der Eingangsseite wird Staubmaterial eingebracht, das sich auf den Stofffiltern ablagert. In regelmäßigen Abständen werden die Filterschläuche auf der Gas-Ausgangseite über Druckluftimpulse beaufschlagt, um die, auf den Filtern abgelagerten Staubmengen „abzusprengen”. Auch dieses Reinigungsverfahren wird in der industriellen Abgasreinigung verwendet.

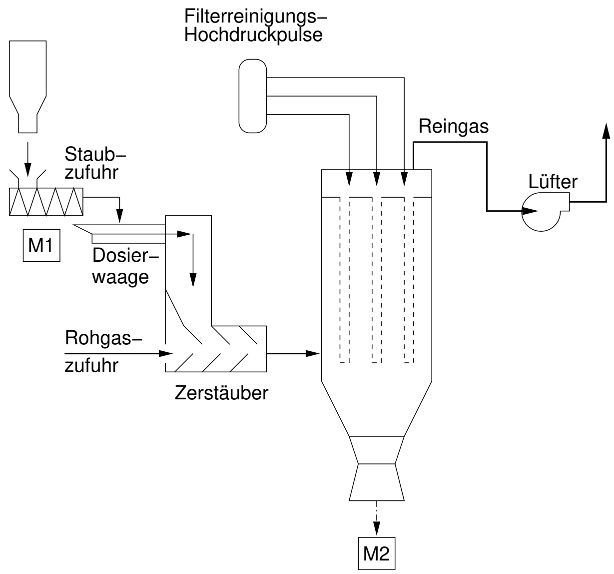


Abbildung 1.2.: Versuchsanlage - vereinfachtes Schema

Die Tatsache, dass einerseits die abgestoßene Staubmenge nicht 100%ig der zugeführten Menge entspricht und weiters die Eigenschaft, dass Teile der „abgesprengten” Staubmengen auf benachbarte Filterflächen „geworfen” werden, sorgt für ein Sinken der Filterqualität im laufenden Betrieb, weshalb in gewissen Abständen die Filteranlagen stillgelegt und überholt werden müssen.

Die Filterservice-Zyklen werden auf Grund eines globalen Betriebszustandes (Differenzdruck zwischen Ein- und Ausgang) über eine große Anzahl von Einzelfiltern ermittelt, man „misst” daher nur eine „mittlere” Filtereffizienz, obwohl einzelne Filter noch wesentlich länger verwendet werden könnten und man durch Austausch wieder anderer Einzelfilter die notwendige Filtereffizienz für längere Zeit erhalten könnte.

Stillstände, verminderte Filterwirkungsgrade und Filterwechsel stellen einen wesentlichen betriebswirtschaftlichen Faktor dar, was dazu führt, dass in die Erforschung des Verhaltens von Feststofffiltern maßgebliche Summen investiert werden.

1.2. Messverfahren

Die Anlage besteht aus der Filterkammer, in die bis zu 6 Feststofffilterschläuche mit einer Länge von bis zu 2m und einem Durchmesser von 120mm eingesetzt werden können. An der „Außenseite” der Filter werden exakt dosierte Mengen an Filterstaub eingebracht, auf der Abgasseite wird eine Unterdrucksituation erzeugt, die die Filterung des Gases und die Ablagerung von Staub auf den Filtern bewirkt. Wie im Schema in Abb. 1.2 dargestellt, werden über hochpräzise Waagen Massenbilanzen erstellt, die den Feststoffanteil in den Filtern exakt beschreiben.

$$M_{filter} = M_1 - M_2$$

Dies bedingt jedoch, dass kein Staub die Anlage „verlassen" darf. Daher wurde die Anlage auch so konzipiert, dass sie in jedem Fall im Rahmen einer Versuchsreihe hermetisch dicht sein muss.

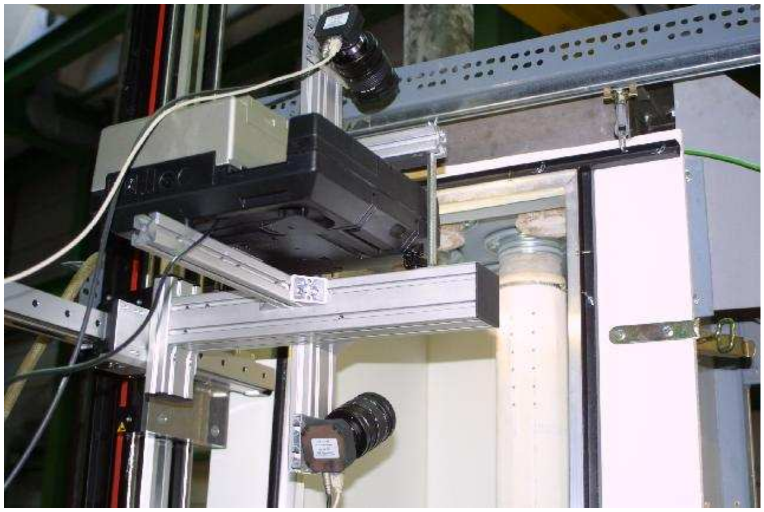


Abbildung 1.3.: Optisches Messsystem

Was die Eigenschaften der Filter betrifft, ist die Dicke der Staubschicht auf dem Filter (auch als „Filterkuchen" bezeichnet) eine der signifikanten Größen, die das Verhalten beschreiben. Ein optisches Messsystem (Abb. 1.3) ermittelt über 2 Kameras die Oberfläche in Form eines 3-dimensionalen Modells der Filterbasisgeometrie. Ein weiteres Modell, das nach Durchlaufen eines Filterdurchzyklus erstellt wird, liefert in Form einer Differenzrechnung das 3-dimensionale Bild des Filterkuchens.

2. Anforderungsdefinition

In diesem Kapitel werden die Anforderungen der geplanten Anlagenerweiterung behandelt, für die eine technische Lösung zu konzipieren und konkret umzusetzen ist. Die geplante Erweiterung besteht aus dem mechanischen Umbau, der nicht Inhalt dieser Diplomarbeit ist, und einem steuerungstechnischen Anteil, der konkret in dieser Arbeit abgehandelt wird.

2.1. Neue mechanische Anforderungen

Das in Absch. 1.2 beschriebene Verfahren zur Messung der Filterstaub-Schichtdicke bedingt die optische Zugänglichkeit der zu messenden Filterfläche.

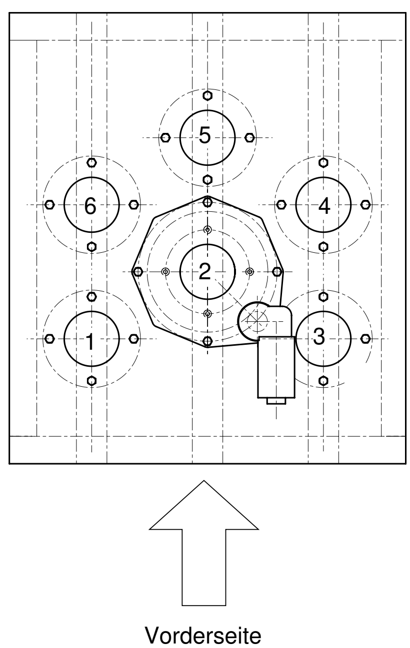


Abbildung 2.1.: Filteranordnung (Draufsicht)

In Abb. 2.1 ist zu erkennen, dass nur ein geringer Teil des mittleren Filters 2 eingesehen und damit optisch „vermessen" werden kann. Die Abbildung stellt die Filteranlage im Grundriss dar, wobei die neue Drehvorrichtung bereits angedeutet ist.

Des Weiteren ist die Oberfläche dieses Filters von besonderer Bedeutung, da diese durch die Reinigungszyklen der umliegenden Filter am stärksten beeinflusst wird - siehe dazu auch die Ausführungen aus Abschnitt 1.1.

Zu Beginn des Jahres 2005 wurde daher beschlossen, die Anlage dahingehend zu erweitern, dass Filter 2 in der geschlossenen und betriebsbereiten Anlage gedreht werden kann (Insitu-Funktion). Damit wurde ermöglicht, dass dieser Filter innerhalb einer Versuchsreihe rundum eingesehen und optisch gemessen werden kann.

2.1.1. Betriebstechnische Anforderungen

Die Montagezeichnung in Abb. 2.2 macht die beengten Platzverhältnisse deutlich, die bei der Konzeption der Steuerungskomponenten berücksichtigt werden mssten.

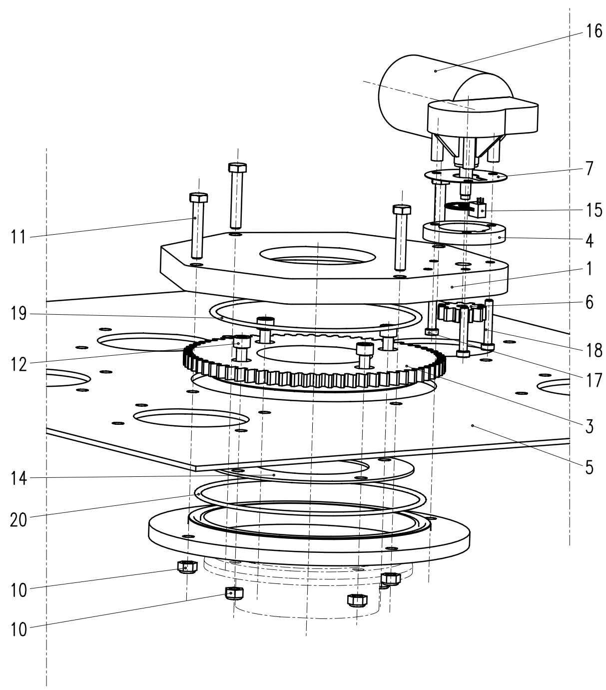


Abbildung 2.2.: Drehvorrichtung - Montagezeichnung

Die in Absch. 2.1 definierten Zusatzanforderungen dürfen jedoch die bereits bestehenden Fähigkeiten der Anlage (siehe Absch. 1.2) nicht einschränken.

Neue zusätzliche Anforderungen entstanden durch den Umbau und den damit verbundenen neuen Fähigkeiten der Versuchanlage.

Dies führt zu folgendem Anforderungsprofil an die umgebaute Anlage:

1. Die hermetische Dichtheit muss auch nach dem Umbau bestehen bleiben. Die Drehbarkeit des Filters darf nicht zu einem stetigen Verlust von Filtermaterial durch Leckage führen.

2. Bei der Drehvorrichtung muss berücksichtigt werden, dass durch die Bewegung Vibrationen entstehen können, die ein „Abbröckeln” des Filterkuchens bewirken und damit zu einer Verfälschung der Messergebnisse führen können.

3. Die Drehbewegungen müssen eine definierte Wiederholgenauigkeit aufweisen und auch nach dem Abschalten und Wiederanfahren der Anlage müssen Positionen reproduzierbar wiederhergestellt werden können.

4. Der Platzbedarf der Anlage soll nicht vergrößert werden, da dies umfangreiche Umbauarbeiten in den Halterungen und Befestigungten nach sich ziehen würde. Der Umbau sollte möglichst wenige Komponenten betreffen, um den bestehenden Betrieb nicht für längere Zeit unterbrechen zu müssen.

Diese Definition musste sowohl bei den Steuerungsanforderungen, als auch beim mechanischen Umbau berücksichtigt werden.

2.1.2. Maschinbautechnische Erweiterungen

Der erste Schritt der Umbauarbeiten bestand in der Konstruktion einer Drehvorrichtung, die in der Lage ist, den mittleren Filter um 360° zu drehen.

Diese Konstruktion wurde von Februar bis Juli 2005 vom Autor dieser Diplomarbeit ausgeführt und der daraus resultierende Umbau an der Techn. Universität Graz konkret umgesetzt, wobei die „alte” Anlage ungestört weiter betrieben werden musste. Dies wurde dadurch erreicht, dass alle Änderungen an der Anlage auf die Änderung einer einzigen Zwischenplatte, an der die Halterungen der Staubfilter angebracht waren, beschränkt werden konnte. Diese Platte wurde mit allen notwendigen Erweiterungen neu konzipiert und als selbständige Einheit neu zusammengebaut. Err.: 4.10.06 Staufilter

Damit beschränkte sich die gesamte Umbauphase für den Betrieb aus dem Austauschen der bestehenden Platte durch die bereits fertig vormontierte neue Version (Abb. 2.3).

Abbildung 2.3.: Innenansicht vor und nach dem Umbau

2.2. Hardwareanforderungen aus Sicht der Anlagensteuerung

Für die, in den vorhergehenden Abschnitten festgelegten Anforderungen wird nun ein Anforderungsprofil für den steuerungstechnischen Teil des Umbaus festgelegt. Dieses stellt die Grundlage für alle weiteren Untersuchungen und Entscheidungen dar.

1. Als ein wesentliches Merkmal der Anlage muss festgehalten werden, dass es sich um eine Einzelanlage handelt. Jede Komponente wird nur für diese Einzelanlage angeschafft, gefertigt oder abgeändert.

2. Ein Mikrocontroller als zentrale Steuereinheit wurde vorgegeben, um vorab eine hohe Flexibilität sicherzustellen. Diese erlaubt weitere Steuerungsfunktionen zu implementieren ohne große Umbau- oder Erweiterungsarbeiten durchführen zu müssen. Die Anzahl der zu steuernden Komponenten wird nach aktueller Planung noch zunehmen.

3. Im Bereich der Elektronik sollen möglichst verbreitete Standardkomponenten verwendet werden, um die Ersatzteilversorgung für die mehrjährige Laufzeit des Projektes sicherzustellen.

4. Die Anlage soll möglichst modular aufgebaut werden, wobei für die Kommunikation zwischen den Bauteilen nach Möglichkeit Standardprotokolle und Busse verwendet werden sollen, um Module leichter durch andere ersetzen zu können.
5. Als Antriebstechnologie für die Drehwinkelsteuerung wird ein elektrischer Antrieb vorgegeben, für Hydraulik oder Pneumatik müssten die notwendigen Aggregate erst angeschafft werden.

3. Stand der Technik

Dieses Kapitel setzt sich mit den einzelnen Komponenten der geplanten Anlage auseinander und gibt einen Überblick über den Stand der Technik möglicher Lösungen und Komponenten. Diese Betrachtung wird zu Entscheidungskriterien führen um die geforderten Funktionen unter Berücksichtigung der technischen und wirtschaftlichen Gegebenheiten zu gewährleisten.

3.1. Grundlegende Festlegungen

Eine Recherche unter Berücksichtigung der Randbedingungen aus Kapitel 2 ergab, dass es keine fertigen Lösungen für diesen Fall gab.

Die Umsetzung der Anforderungen verlangt aber in jedem Fall einen Positionierantrieb und je nach Art der Einheit müssen die weiteren Kriterien beurteilt werden.

> Das Steuern ist der Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen, andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichnend für das Steuern ist der offene Wirkungsablauf, die *Steuerkette* [1].

Dieser Positionierantrieb muss nummerisch angesteuert werden, um quantitative Eingangsgrößen für die Versuchsanlage zu erhalten.

Eine zusätzliche Festlegung vom Auftraggeber fordert, dass, unabhängig von der Art der Steuerung, ein Wegmesssystem verwendet werden muss, um die Positions-Eingangsgröße eindeutig auch als Systemrückmeldung (Istwert) zu erhalten.

> Das Regeln ist ein Vorgang, bei dem die zu regelnde Größe fortlaufend erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und abhängig vom Ergebnis dieses Vergleichs im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Der sich dabei ergebende Wirkungsablauf findet in einem geschlossenen Kreis, dem *Regelkreis*, statt [1].

Mit Verwendung dieses Erfassungsgliedes ist damit ein Regelkreis definiert, auch wenn dieser redundante Informationen liefern würde. Siehe dazu auch 3.2.1.

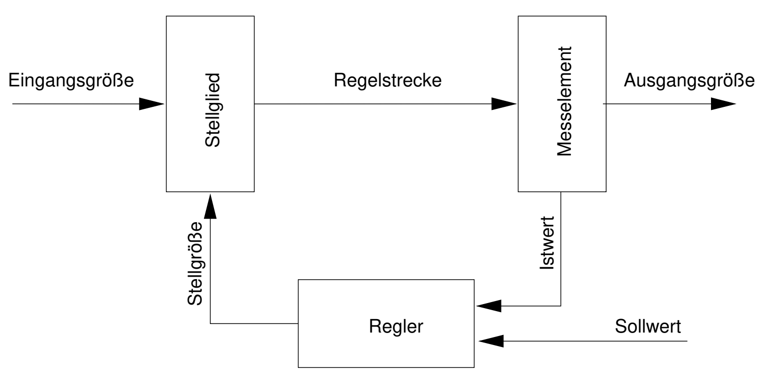


Abbildung 3.1.: Komponenten des Regelkreises

Für die in Abb. 3.1 dargestellten maßgeblichen Funktionen müssen nun die entsprechenden technischen Komponenten nach dem Stand der Technik einzeln betrachtet und bewertet werden.

3.2. Positionierantriebe

Die Entscheidung, einen günstigen elektrischen Positionierantrieb zu verwenden, wurde lt. Absch. 2.2 vorgegeben, dies schränkt die Antriebseinheit auf folgende 3 häufig verwendete Antriebsarten ein.

In Tabelle 3.1 (Zusammenfassung aus [2]) sind die Vor- und Nachteile der einzelnen Technologieen noch einmal zusammengefasst.

3.2.1. Schrittmotor

Winkelgenaue Drehvorrichtungen werden in der Industrie meist mit Schrittmotoren umgesetzt, wenn der Antrieb mit der Bewegungseinheit schlupffrei verbunden ist.

Der Vorteil dieser Methode ist die einfache Ansteuerung, da mit dem Steuerimpuls bereits die Bewegungsgröße mitgegeben wird. Ein zusätzliches Messsystem und ein damit verbundenes Regelsystem ist nicht zwingend notwendig.

3.2.2. Gleichstrommotor (Asynchronmotor)

Eine andere einfache Methode besteht in einem Asynchronmotor, der über ein Wegmesssystem und einen Regler an die entsprechende Position „gefahren" wird. Diese Antriebe benötigen zwingend Wegmesssysteme und Regelkreise, allerdings sind diese Motoren im geforderten Drehmomentbereich wesentlich billiger als Schritt- oder Servomotoren mit vergleichbarer Leistung.

Tabelle 3.1.: Vergleich der Eigenschaften von Positionierantrieben

	Schrittmotor	Geichstrommotor	Servomotor
Steuerelektronik	einfach	komplex	komplex
Treiber-elektronik	komplex	einfach	komplex
Wartung	keine (Bürstenlos)	regelmäßig	keine
Erwärmung, Stromverbrauch	hoch (Haltestrom)	gering (nur in Bewegung)	gering
Drehmomenten-verlauf	max. bei geringer Drehzahl	flach (auch hoch bei hoher Drehzahl)	nahezu linear
Dynamik, Beschleunigung	gering	mittel	gut
Resonanz, Schwingen	Eigenfrequenz	sehr ruhige Bewegung	weitgehend ruhig
Zielposition, Regelung	einfach/offener Regelkreis	schwierig/geschl. Regelkreis	optimal/PID Regler notwendig

3.2.3. Servomotor (Drehstrom-Synchronmaschine)

Ein technisch sehr anspruchsvolle Lösung stellt der Servomotor dar. Dieser ist mit einer Rückführung versehen, die es erlaubt, die aktuelle Drehposition des Rotors zu bestimmen, diese Rückmeldung erfolgt meist über einen Drehwinkelgeber. Eine elektronische Steuerung vergleicht das Signal des Drehgebers mit einem vorgegebenen Sollwert und korrigiert, wenn dies nicht der Fall ist, in beiden Richtungen.

3.2.4. Antriebsmotor, Entscheidung

Auf Grund der Anforderungen aus 2.1, 2.2 und der in Tab. 3.1 angeführten Eigenschaften wurde beschlossen einen Gleichstrommotor als Antriebseinheit zu verwenden, wobei die folgenden Gründe ausschlaggebend waren.

1. Auf Grund der Entscheidung, einen Mikrocontroller einzusetzen, wird die Komplexität der Steuerelektronik bereits großteils abgedeckt, wenn der Controller die wichtigsten Steuerungskomponenten mitbringt.
2. Die Treiberelektronik - also die Komponenten, die auf Grund der „Hardwarenähe" nicht allgemein verwendbar sind - umfassen wenige einfache Komponenten.
3. Die Wartungsüberlegungen haben geringere Priorität, da nur geringe Motor-Betriebszeiten erwartet werden. Ein Messzyklus zieht sich oft über Stunden hin.
4. Bedenken bezüglich Stromverbrauch und Erwärmung entfallen ebenfalls aus den selben Überlegungen, im Stillstand wird, im Gegensatz zum Schrittmotor, kein Strom verbraucht.
5. Dynamik, Drehmomentenverlauf, Beschleunigung, sind nicht relevant, da nur sehr langsame Bewegungen und minimale Beschleunigungen gefordert sind.
6. Resonanz und Schwingungsverhalten sind bei dieser Lösung prinzipbedingt sehr gut (siehe Tabelle 3.1).

Einzig die hohe Komplexität der Regelung spricht gegen diese Lösung, sie wird in Form umfangreicher Controllerprogrammierung umgesetzt, wodurch dieser Teil der Lösung wiederum eine sehr hohe Flexibilität erreicht.

Nach Festlegung dieser Randbedingung werden die Steuerungsmöglichkeiten von Gleichstrommotoren im Detail betrachtet.

3.3. Motorsteuerungen

Nach Festlegung auf einen kostengünstigen Gleichstrommotor, werden die Grundlagen und Technologien, die die Eigenschaften dieser Antriebstechnik beschreiben, näher betrachtet. Diese Informationen sind notwendig, um die einzelnen Module für die geforderten Funktionen beurteilen, auszulegen und auswählen zu können.

3.3.1. Steuerungsarten, Grundlagen

Auf einen Ladungsträger, der sich quer zu einem Magnetfeld bewegt, wirkt die so genannte Lorentzkraft F.

$$\vec{F} = q(\vec{v} \times \vec{B}) \text{ bzw. } \vec{F} = I(\vec{l} \times \vec{B})$$

q ... elektrische Ladung, in $[C], [As]$

$\vec{v}$ ... Geschwindigkeit der Bewegung normal zu der Magnetfeldlinien $[ms^{-1}]$

B ... magnetische Flussdichte $[Vsm^{-2}], [NA^{-1}m^{-1}]$

I ... Stromstärke $[A]$

$\vec{l}$ ... Länge der Windungen normal zu der Magnetfeldlinien $[m]$

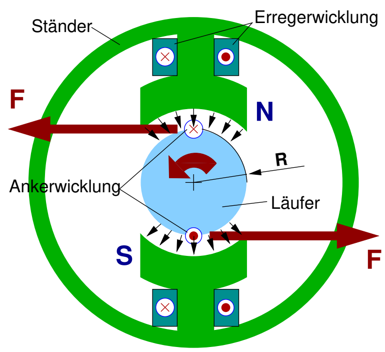


Abbildung 3.2.: Prinzip des Gleichstrommotors

Dies ergibt ein Drehmoment nach der Formel:

$$M = 2FR$$

M ... Drehmoment $[Nm]$

F ... Lorenzkraft $[N]$

R ... Radius der Ankerwindungen $[m]$

Bedingung 1 für die Erhöhung der Drehzahl bei einem definierten Ankerstrom I_A lautet:

$$M \propto I_A$$

Nach dem Induktionsgesetz entsteht jedoch wiederum eine Induktionsspannung, da die Windungen des Rotors die Magnetfeldlinien schneiden, die der Ursache entgegenwirkt (Lenzsche Regel). Die Größe der Gegenspannung U_0 hängt vom Magnetfeld Φ und der Bewegungsgeschwindigkeit, somit von der Drehzahl n ab.

Bedingung 2 für die Verringerung der Drehzahl lautet somit:

$$U_0 \propto n$$

Das Gleichgewicht dieser beiden Effekte bestimmt die Drehzahl eines Gleichstrommotors.

Schlussfolgerung:
Die einzige Möglichkeit, die Drehzahl von Gleichstrommotoren zu beeinflussen, besteht in der Änderung des Anker- oder Ständerstromes und ist somit proportional zur Klemmenspannung U_{KL}.

3.3.2. Steuerung der Drehzahl

In der Praxis werden Gleichstrommotoren jedoch nicht mit konstantem Strom versorgt, sondern mit Impulsen einer definierten Breite, die in Summe dem erwarteten Effektivwert entsprechen. Damit ist ein „sanftes" Anfahren möglich, da ein „hoher" Stromimpuls kurzzeitig ein hohes Drehmoment bewirkt und damit leichter die Haftreibung überwindet.

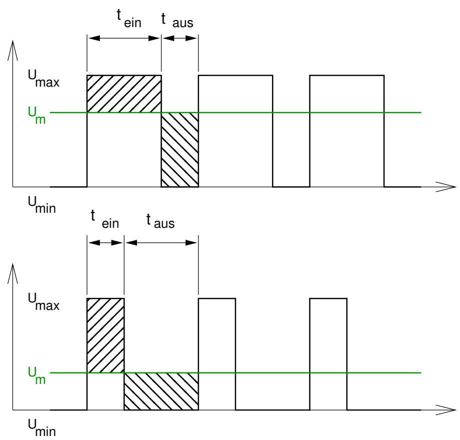


Abbildung 3.3.: Pulsbreiten-Modulation

Die Steuerung mittels Impulsform (siehe Abb. 3.3) wird Pulsbreiten-Modulation oder PWM (**P**ulse **W**idth **M**odulation) genannt.

Der Mittelwert der Spannung ermittelt sich nach der Formel:

$$U_m = U_{min} + (U_{max} - U_{min}) \frac{t_{ein}}{t_{ein} + t_{aus}}$$

bzw. bei $U_{min} = 0$ ergibt sich eine Mittelspannung von:

$$U_{m=}U_{max} \frac{t_{ein}}{t_{ein} + t_{aus}}$$

Somit ist diese Art der Steuerung ausschließlich von der Impulsform abhängig. Das bedeutet, dass man quasi nur einen schnellen Ein-/Ausschalter benötigt, was zu sehr einfachen und effektiven Reglern führt, wie dies im nächsten Abschnitt näher ausgeführt wird.

3.3.3. Steuerung der Drehrichtung

Als weiteres Kritierium muss berücksichtigt werden, ob der Motor nur in einer Drehrichtung oder in beide Richtungen angesteuert werden muss.

Bei nur einer Drehrichtung reicht eine einfache Transistorschaltung aus (Abb. 3.4 zeigt das Prinzip einer Highside-Schaltung auf MOS-FET Basis), um die Klemmenspannung und damit die Drehzahl in beliebigen Bereichen festzulegen.

Muss der Motor jedoch in beiden Richtungen betrieben werden, muss die Spannung umgepolt werden. Dies verursacht bei elektronischen Schaltungen einen wesentlich erhöhten Schaltungsaufwand, wie bereits an einem Prinzipschaltbild der H-Bridge-Schaltung in Abb. 3.5 auf der nächsten Seite zu erkennen ist. Die Aufwände für Schutzbeschaltungen etc. lassen die Kosten noch weiter steigen.

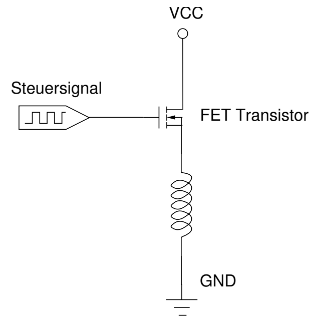


Abbildung 3.4.: Highside-Schaltung

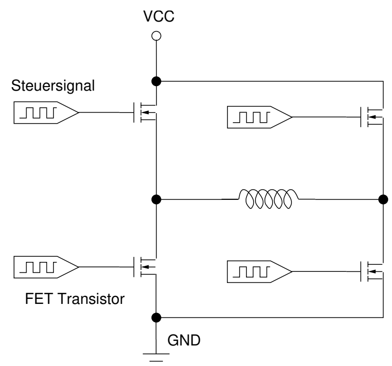


Abbildung 3.5.: Prinzip der H-Bridge Schaltung

3.3.4. Steuerungsverhalten

Ein weiteres wichtiges Kriterium für die Auswahl des Antriebs stellt das Steuerverhalten von Motoren dar. Diese Eigenschaft beschreibt, wie der Antrieb von einem Betriebszustand in einen anderen übergeht und muss ebenfalls mit der Gesamtanforderung abgestimmt werden.

Die 4 möglichen Betriebszustände werden den Quadranten eines Kreises zugeordnet, die „Quadranten" beziehen sich dabei auf die vier Bereiche eines Drehmoment/Drehzahl-Diagramms (Abb. 3.6).

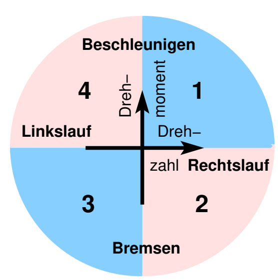


Abbildung 3.6.: Quadrantenregelung

Die Klassifizierung dieser Eigenschaften wird als Quadrantenkontrolle bezeichnet, wobei gilt:

- Eine positive Drehzahl stellt eine Rechtsdrehung dar, eine negative eine Linksdrehung.
- Ein positives Drehmoment stellt eine Beschleunigung dar, ein negatives ein Abbremsen.

In der Praxis werden 3 Reglerklassen angeboten.

Die **1-Quadranten-Regelung** funktioniert in einer einzigen Drehrichtung, ohne die Möglichkeit des Abbremsens. Bei zu hoher Drehzahl unterbricht der Regler die Stromzufuhr, sodass der Motor durch die Last gebremst wird.

Für die **2-Quadranten-Regelung** gilt das gleiche, jedoch in zwei Drehrichtungen. Ein Umschalten der Polung der Antriebsspannung ist nicht erforderlich, da dies innerhalb der Regelung bei Bedarf automatisch erfolgt.

Eine **4-Quadranten-Regelung** funktioniert ebenfalls in zwei Drehrichtungen, ermöglicht aber darüber hinaus auch ein Abbremsen. Bei zu hoher Drehzahl unterstützt der Motor den Bremsvorgang, sodass das System rasch seine Drehzahl verringert. Die 4 Betriebszustände (Rechts- und Linkslauf, Geschwindigkeitszu- und -abnahme) entsprechen dabei einer genau definierten Position im 4-Quadranten Diagramm.

3.4. Wegmesssysteme

Die Anforderung an eine Drehwinkelsteuerung legt fest, dass beliebige Winkel im Rahmen der Positioniergenauigkeit wiederholbar angefahren werden können. Das einzige notwendige Kriterium stellt in der geplanten Anlage die Winkellage dar, deshalb werden im Folgenden nur Winkelmesssysteme betrachtet.

Winkelmesssysteme messen grundsätzlich Anteile eines Kreisumfangs in der Einheit Winkelgrad, wobei Auflösungen von 1 bis ca. 10^6 Winkelschritte pro Umdrehung erhältlich sind. Als Messtechnologien kommen magnetische, kapazitive, optische oder auf Halleffekt beruhende Messwertaufnehmer in Frage.

In der Technik zur Messung der Winkelschritte muss man 2 grundsätzliche Arten unterscheiden, die verschiedene Anforderungen an die Verwaltung der Messdaten stellen.

Absolutwertmessung Der Messwertaufnehmer liefert direkt einen Winkel in Form eines nummerischen Wertes ab. Systeme mit diesem Verhalten werden auch als **Kodiermesssysteme** bezeichnet.

Relativwertmessung Bei diesem Messsystem wird ein Zähler zur Ermittlung des Drehwinkels verwendet, der die Anzahl der Schritte von der Winkelposition 1 zu Position 2 zählt. Man bezeichnet diese Systeme auch als **Schrittzählsysteme**.

Während Kodiermesssysteme auf Grund der Absolutwertmessung in der Lage sind den gemessenen Wert direkt als Winkel abzuliefern, muss der gemessene Winkel eines Relativmesssystems über folgende Formel errechnet werden:

$$\Phi = \alpha_0 + \frac{z_2 - z_1}{R} \cdot 360°$$

$\Phi \ldots$ gemessener Drehwinkel $[Winkelgrad]$
$\alpha_o \ldots$ ursprüngliche Winkelposition $[Winkelgrad]$
$z_1 \ldots$ Anfangsschrittwert
$z_2 \ldots$ Endschrittwert
$R \ldots$ Messauflösung $[Schritte/Umdrehung]$

Bei einer geforderten Winkel-Endposition ermittelt sich deren Wert nach der Formel:

$$\alpha_e = \alpha_0 + \Phi$$

$\alpha_e \ldots$ Winkel-Endlage
$\alpha_0 \ldots$ ursprüngliche Winkelposition

Dies muss bei einer Verwendung eines Relativwertmesssystems berücksichtigt werden, da die „zuletzt” gemessene Position „gespeichert” und beim Neustart wieder eingelesen werden muss. Ist dies nicht sichergestellt, muss eine automatische oder manuelle Kalibrierung der Winkellage durchgeführt werden bevor absolute Winkelwerte bestimmt werden können.

3.5. Kommunikation zwischen den Komponenten

Zum Zeitpunkt der Konzeption der Elektronik war noch nicht klar, ob die Motorsteuerung direkt am Motor (also innerhalb des hermetisch dichten Anlagengehäuses) oder außerhalb montiert werden sollte, ebensowenig war geklärt, von wo und wie die Steuerung tatsächlich bedient werden sollte. Diese undefinierte Situation führte zur Entscheidung, „fernwirkbare" Komponenten zur Erzeugung der Steuerspannung einzusetzen und auch andere Signale über größere Signalstrecken transportieren zu können.

Einige verbreitete Schnittstellen aus dem industriellen Bereich werden nun näher betrachtet.

3.5.1. Industriebus RS485

Der von der Electronics Industry Association (EIA) standardisierte Schnittstelle kann als differentieller 2- oder 4-Draht Bus ausgeführt werden. Mit max. 32 Empfängern und Sendern ist der unidirektionale Bus (1 Sender und mehrere Empfänger zugleich) für ein Netzwerk kleiner bis mittlerer Größe geeignet. Mit bis zu 100KBit bei einer max. Leitungslänge bis ca. 1.2 km spezifiziert, zählt er zu den eher einfacheren „Weitverkehrsbussen".

Dieser Bus wird auch als „Party Line" bezeichnet. Er stellt die Basis für weitere Busspezifikationen, wie z.B. für den InterBus oder ProfiBus, dar.

3.5.2. CAN/ICAN Bus

Die ursprünglich von Bosch entwickelte, busfähige Industrieschnittstelle CAN (**C**ontroller **A**rea **N**etwork) wird mittlerweile vorwiegend als ICAN (Independent CAN) im Automotive Bereich eingesetzt. Als differentieller serieller 2-Draht Bus erfüllt er bereits konzeptionell hohe Ansprüche an Störsicherheit und Robustheit. Non Return to Zero Codierung (NRZ) sorgt für kompakte Informationspakete, Routingfähigkeit, Multicasting, Fehlererkennung und Signalisierung, Arbitrierung, etc. sorgen in kritischen Umgebungen für eine einwandfreie Kontrolle und möglichst fehlerfreie Informationsübermittlung.

Die max. Leitungslänge ist mit 1000m spezifiziert.

CAN ist genormt in der DIN ISO11898 und verwendet wie Ethernet den CSMA/CD-Mechanismus zur Bussteuerung, die Spezifikation ist in [4] nachzulesen.

3.5.3. I^2C-Bus

Dieses bereits 1992 von Philips spezifizierte Übertragungsverfahren stellt eine weitere stabile busfähige Kommunikationsmethode dar. Ebenfalls als 2-Draht Bus (TWI, **T**wo **W**ire **I**nterface) ausgeführt, gilt das vorhin gesagte zum Thema Stör- und Übertragungssicherheit. Als bidirektionaler Bus verwendet er eine serielle Taktleitung und eine Datenleitung, womit Übertragungsraten bis 100KBit, bzw. bis 400KBit im Synchronmodus erreicht werden. Der Multi-Master-Slave-fähige Bus ist sehr einfach aufgebaut und ist in zahlreiche einfache Elektronikmodule integriert erhältlich.

Der I^2C-Bus ist in der Lage, 128 Geräte zu adressieren, die Spezifikation ist in [3] zu finden.

4. Anforderungskriterien

Dieses Kapitel fasst die definierten Anforderungskriterien noch einmal kurz zusammen oder präzisiert die Anforderungen, die dann später zu konkreten Produktentscheidungen führen werden. Die Randbedingungen ergeben sich einerseits aus technischen Kriterien, die im vorigen Kapitel erläutert wurden, und aus wirtschaftlichen Überlegungen und weiteren Forderungen, die von der Auftraggeberseite vorgegeben wurden.

4.1. Mikrocontroller

Bezüglich der Auswahl des Controllers gab es seitens des Auftraggebers keine Präferenzen, sodass die Anforderung nach rein technischen, kostenrelevanten und weiteren Faktoren festgelegt werden konnte.

4.2. Antriebsmotor

Die Entscheidung, einen preisgünstigen Asynchron-Gleichstrommotor zu verwenden, wurde bereits in Abschnitt 3.2.4 abgehandelt.

Auf Grund der zu erwartenden hohen Reibungsverluste der mechanischen Konstruktion wurde als notwendige technische Forderung ein Anzugsdrehmoment von $20Nm$ spezifiziert.

Um ein „abbröckeln" des Filterkuchens zu verhindern, darf die Drehbewegung nicht „ruckartig" einsetzen und muss „langsam" abgebremst werden. Die maximale Drehgeschwindigkeit wird mit ca. 5 Umdrehungen pro Minute festgelegt.

Err.: 4.10.06
0.2 Umdreh.

4.3. Steuerung des Motors

Bei der Beurteilung von Elektromotoren für Steuerungsaufgaben müssen nun die diversen Kriterien betrachtet und im Detail festgelegt werden, die dann zu den Entscheidungen für die entsprechenden Komponenten führen.

4.3.1. Steuerungsart

Wie in Abschnitt 3.3.1 beschrieben ist die Auswahl der Komponenten für die Steuerung von Gleichstrommotoren sehr begrenzt. Eine kostengünstige PWM-Steuerung, um ein möglichst ruckfreies „Anfahren" sicherzustellen, wird auf Grund der technischen Anforderungen als ausreichend angesehen. Schwingungen, die sich auf die Filter übertragen können, werden dadurch weitgehend vermieden.

4.3.2. Richtungssteuerung

Eine Drehrichtungsumkehr tritt selten auf und erfolgt immer aus dem Ruhezustand, sodass ein Wechsel der Drehrichtung durch Umschalten der Stromrichtung über ein Relais bewerkstelligt werden kann.

Eine längere Diskussion über die technische Ausführung der Konstruktion und über mögliche Einflüsse der geplanten Komponenten auf die Messergebnisse führte zu folgender Festlegung:

> **Festlegung:**
> Um alle Einflüsse auf die Messwerte auf ein Minimum zu beschränken, den Aufwand bezüglich der Umsetzung zu minimieren und auf Grund der kritischen Terminsituation wird Anlage vorerst nur im Ein-Richtungsbetrieb verwendet.
> Die technische Möglichkeit für den Zwei-Richtungsbetrieb wird jedoch vorgesehen. Eine Entscheidung über den Einsatz eines Zwei-Richtungsbetriebes wird auf eine späteren Zeitpunkt verschoben und bedingt die vorherige Abklärung aller Einflussfaktoren auf die Versuchsergebnisse.

4.3.3. Ansteuerverhalten

Bei der Festlegung des Steuerungsverhaltens (siehe dazu auch 3.3.4) und auf Grund der vorhin definierten Randbedingungen kann eine einfache 1-Quadrantenreglung verwendet werden. Ein Abbremsen der Bewegung ist nicht notwendig, da durch die Dichtungsflächen hohe Reibungskoeffizienten auftreten, die beim Wegfall des Antriebs die Drehbewegung rasch zum Stillstand bringen.

4.4. Drehwinkelgeber

Die beengten Platzverhältnisse innerhalb der Drehvorrichtung schränken die Auswahl der am Markt erhältlichen Messsysteme schon von vornherein stark ein.

Die bestehende Software zur Auswertung der optischen Messergebnisse ermöglicht eine softwaremäßige Korrektur des optischen Systems (Alignment correction - siehe 1.2) bis zu einer Abweichung von $\pm 0.5mm$, woraus eine Winkelmessgenauigkeit von $1mm$ an der Filteroberfläche gefordert wird.

4.5. Kommunikationseinheit

Zur Kommunikation zwischen Mikrocontroller und Motorsteuerungseinheit sollte auf eine technisch robuste und ausgereifte Schnittstelle zurückgegriffen werden.

Um eine konkrete Entscheidung für das Bindeglied zwischen Mikrocontroller und „restlicher" Hardware treffen zu können, sollten erst die einzelnen Komponenten betrachtet werden, und danach kann die Entscheidung für die Kommunikationseinheit nach technischen Kriterien gefällt werden.

Eine Präferenz seitens des Auftraggebers gab nicht.

5. Ausgewählte Komponenten

Zahlreiche allgemeine technische Betrachtungen und Festlegungen wurden bereits im vorhergehenden Kapitel angeführt, in diesem Kapitel geht es nun um die konkrete technische Umsetzung der Anforderungen aus dem vorigen Kapitel bzw. um die konkrete Produktwahl.

Alle diese Komponenten werden nun im Rahmen dieses Kapitels detailliert beschrieben, deren Einsatzbereich festgelegt und die Begründung für die Entscheidung wird dargelegt.

5.1. Mikrocontroller

Als zentrale Steuereinheit wurde ein fertiges Trainings- und Entwicklungssystem auf Basis des Mikrocontrollers ATMega8 von ATMEL ausgewählt, das unter dem Handelsnamen **halvedDISC** erhältlich ist und am **Application Center Microcontroller** (ACMC) der Hochschule Mittweida entwickelt wurde. Die maßgeblichen Gründe für diese Systementscheidung werden angeführt.

1. Die große Anzahl an bereits in der Grundausstattung vorhandenen Ein- und Ausgabeschnittstellen, im Speziellen der integrierten I^2C-Hardware.
2. Die moderne RISC-Prozessorarchitektur der ATMEL Controllerfamilie mit den verschiedenen Speichermöglichkeiten (Flash-RAM als Programmspeicher, SRAM als Arbeitsspeicher und EEPROM als persistenter Langzeitspeicher).
3. Die offene und gut dokumentierte Struktur des Systems und das daraus resultierende verfügbare Knowhow in der Universitäts-Entwicklung und Zugang zu freier Software.
4. Die Tatsache, dass dieses System vom Autor bereits in einem Programmierpraktikum verwendet wurde und damit gut kennengelernt und getestet werden konnte.
5. Der günstige Preis des Entwicklungssystems.

Dem muss gegenübergestellt werden, dass es sich beim Trainingssystem um keine für die Anforderung optimierte Entwicklung handelt und man Teile „mitschleppt”, die für die geplante Funktion nicht benötigt werden, wie RTC (**R**eal-**T**ime **C**lock), Lautsprecher, Helligkeits- und Temperatursensor.

Die technischen Daten des Controllers sind in Tab. A.1 auf Seite 91 angeführt.

5.2. Antriebsmotor

Die Entscheidung fiel auf einen handelsüblichen Getriebe-Gleichstrommotor (Abb. 5.1) der Marke Fa. Bosch mit den technischen Daten aus Tab. A.2 auf Seite 92.

Abbildung 5.1.: Getriebe-Antriebsmotor (Quelle: Graf Elektromotoren)

Dieser Motor erfüllt die in Absch. 4.2 definierten Anforderungen.

5.3. Motorregelung

Für die Motorsteuerungs-Karte wurde auf ein kostengünstiges Standardmodul mit 1-Quadrantenregelung und PWM-Steuerung auf MOSFET-Basis zurückgegriffen (Absch. 3.3.4), das auch vom Lieferanten des Motors empfohlen wurde. Das fertige Produkt mit der Bezeichnung **GS24S** wurde in das Gehäuse der Mikrocontroller-Bedieneinheit (siehe Abb. 6.2 auf Seite 31) eingebaut. Die Steuerung des Motors geschieht über einen Steuerspannungseingang im Bereich 0. . . 10V.

Err.: 4.10.06 Abschn. 6.2

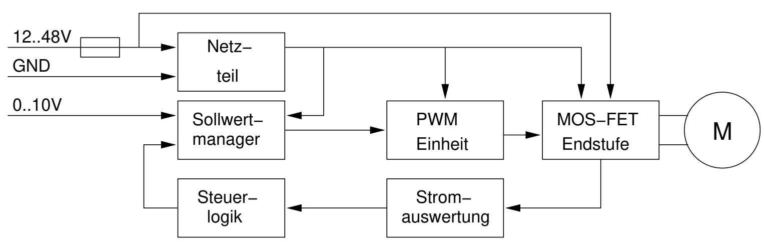


Abbildung 5.2.: Blockschaltbild der PWM-Steuerung

Die technischen Daten der Steuerung finden sich in Tabelle A.3 auf Seite 92, Abb. 5.2 stellt das Prinzipschaltbild dieses „Gleichstromstellers” dar.

5.4. Richtungssteuerung

Um die Drehrichtung vorgeben zu können, wurde ein einfaches Reed-Relais mit Umschaltkontakten angeschafft.

Grundsätzlich verfügt die **halvedDISC** über ein Relais, das diese Umschaltung ausführen könnte, allerdings ist der zulässige Strom lt. halvedDISC Manual auf max. 1A beschränkt. Aus diesem Grund wurde ein eigenes Leistungsrelais zur Umschaltung der Drehrichtung direkt in die Ankerleitungen des Antriebsmotors eingebaut. Das Relais wurde auf der Platine der in Absch. 5.6 beschriebenen I^2C-Steuerungskarte integriert.

Die Wahl fiel auf ein Leistungsrelais der Firma Schrack, die technischen Daten sind in Tabelle A.4 auf Seite 93 nachzulesen.

5.5. Winkelmessung

Die Wahl des Mikrocontrollers (Absch. 5.1) ermöglicht die Langzeitspeicherung von Werten auch bei ausgeschalteter Anlage (EEPROM-Speicher). Dadurch ist eine Winkelmessung mit einem Inkrementalgeber möglich, da durch die Speicherung der „letzten Position” immer der Bezug zu einem vordefinierten bzw. programmierbaren Bezugswinkel herstellbar ist. Zu berücksichtigen ist aber, dass in allen geplanten Betriebsarten (Automatikbetrieb, Handbetrieb, ...) zuverlässig alle Schritte mitgezählt werden und die Endlage der Drehbewegung im Permanentspeicher abgelegt wird.

Man entschied sich für eine einfache Gabellichtschranke mit optischer Taktscheibe der Firma Sharp (Abb. 5.3).

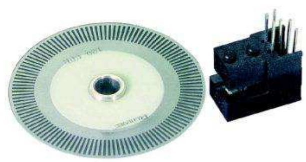


Abbildung 5.3.: Lichtschranken/Drehwinkelgeber (Quelle: Conrad Electronic)

- Dieses Modul enthält 2 versetzte Photosensoren, wodurch es grundsätzlich auch möglich wäre, Geschwindigkeiten und Drehrichtungen zu ermitteln und zusätzlich die Signalauflösung zu erhöhen.
- Den Sensoren ist eine Kontroll-Logik nachgeschaltet, die zu einer Verbesserung der Signalqualität führt.
- Das Modul wird mit einer Taktscheibe für 120 Takte pro Umdrehung ausgeliefert.

Die technischen Daten des Drehwinkelgebers sind aus Tab. A.5 auf Seite 93 ersichtlich.

In Abbildung 2.2 auf Seite 11 ist zu sehen, dass das Winkelmesssystem (Pos. 15) die einzige direkt in die Mechanik integrierte Komponente darstellt. Die Koppelung des Elektromotors an die Drehvorrichtung geschieht über die Abtriebswelle des Motorgetriebes. Diese treibt über zwei Zahnräder direkt die Filterhalterung der Drehvorrichtung an.

Die beiden Zahnräder weisen eine Zähnezahl von 11 Zähnen (Ritzel) und 66 Zähnen (Abtrieb) auf.

Daraus ergibt sich eine Antriebsübersetzung von:

$$N = \frac{z1}{z2} = \frac{11}{66} = \frac{1}{6} \approx 0.166$$

Die Taktscheibe ist direkt auf der Abtriebswelle des Motorgetriebes montiert und führt somit 6 Umdrehungen während einer Filterumdrehung aus. Bei 120 Takten pro Getriebewellenumdrehung errechnet sich die effektive Messauflösung in Winkelgrad über die Formel:

$$\frac{1}{120} \cdot N = \frac{1}{120} \cdot \frac{1}{6} \mathrel{\hat{=}} \frac{1}{720} \cdot 360° = 0.5°$$

Auf die Filteroberfläche umgerechnet ergibt dies nach Abb. 5.4 bei einem Filterkorb-Durchmesser von $d = 120mm$ eine definierte Wegstrecke auf der Filteroberfläche und somit die Messgenauigkeit bezogen auf den Messtakt.

$$L = \frac{1}{720} \cdot d \cdot \pi = \frac{120mm \cdot \pi}{720} \approx 0.524mm$$

Damit ist die geforderte Messgenauigkeit von $1mm$ aus Absch. 4.4 erfüllt.

> Hinweis:
> Diese Komponente ist als einziger elektronischer Bauteil in die Anlage integriert und erfordert umfangreiche Montagearbeiten bei Änderungen, Reparaturen, ... (Abb. 5.5).

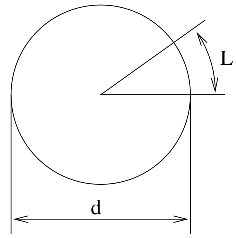


Abbildung 5.4.: Winkel-Messgenauigkeit

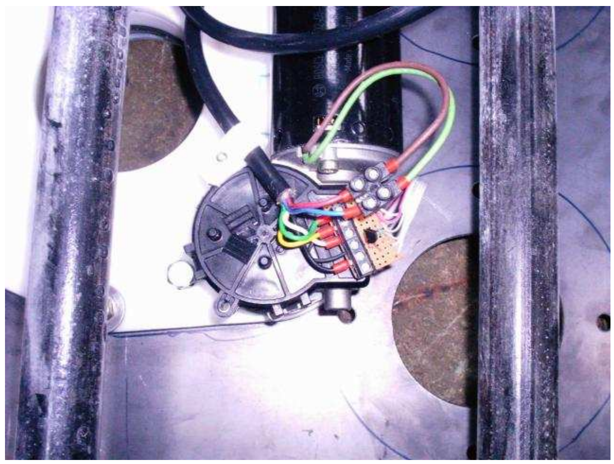


Abbildung 5.5.: Antriebsmotor und Drehwinkelmessmodul

5.6. I^2C-Schnittstellenkarte

Um die Sollwertvorgabe von 0...10V für die Motorsteuerung (siehe dazu auch 5.3) zu generieren, muss ein Digital/Analog-Wandler eingesetzt werden, da die **halvedDISC** über keinen Analogausgang verfügt.

Wie in Abschnitt 5.1 bereits beschrieben, verfügt die halvedDISC bereits in der Grundausstattung über die industrieerprobte I^2C-Bus Schnittstelle (siehe auch 3.5.3). Da mit dem Baustein PCF8591 ein sehr preisgünstiger Analog/Digital- und Digital/Analogwandler auf Basis der I^2C-Schnittstelle existiert, wurde dieser IC als Kommunikationsbaustein gewählt [5].

Als Standard-TTL Chip arbeitet der PCF8591 mit einem Signalpegel von 5V (dies legt auch den oberen Grenzwert der D/A-Ausgangsspannung fest), der Motorregler benötigt jedoch eine Sollwertspannung im Bereich von 0...10V (siehe 5.3). Um diesen Spannungs-Pegel zu wandeln wurde die einfache Operationsverstärker-Schaltung auf Basis des OP LM741C eingesetzt [6].

Die technischen Daten des I^2C-D/A Wandlers PCF8591 sind in Tab. A.6 auf Seite 94 nachzulesen, die des Operationsverstärkers LM741C in Tab. A.7.

5.7. Hardwarekonzept

Die Anforderungen aus 2.1 und 2.2 führten als Ergebnis zum Hardwarekonzept nach Abb. 5.6, für das die Komponenten im Rahmen dieses Kapitels detailliert erläutert und ausgelegt wurden.

Zum Teil konnten die Bauteile direkt in die Anlage integriert werden, andere Komponenten mussten erst entwickelt und technisch umgesetzt werden.

Mit der konkreten technischen Umsetzung beschäftigt sich das nächste Kapitel.

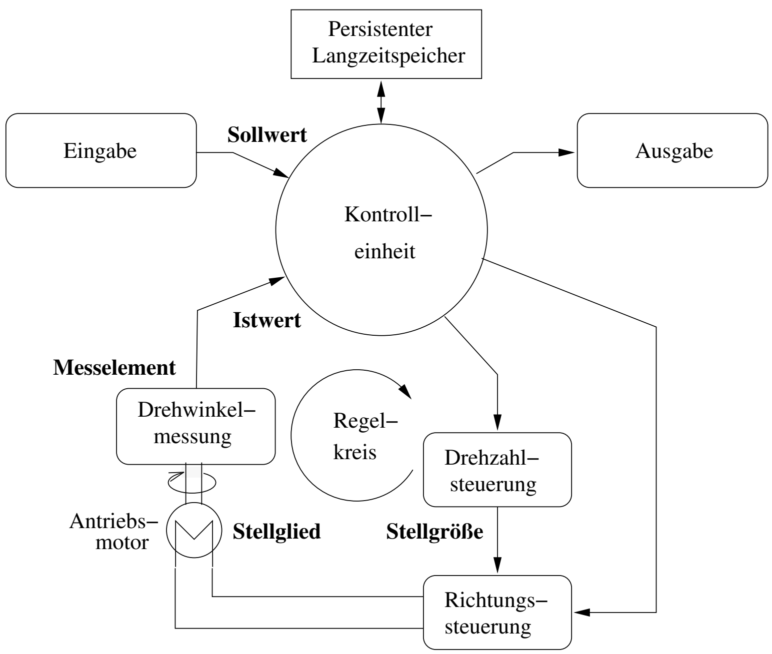


Abbildung 5.6.: Hardwarekonzept der Drehwinkelsteuerung

6. Technische Umsetzung

Nach Auswahl aller funktionswichtigen Komponenten wurden diese Teile in die Gesamtlösung integriert. Zum Teil konnten sie direkt verwendet werden, andere Elektronikkomponenten mussten erst entwickelt und in Module umgesetzt werden. Weitere typische Entwicklungsaufgaben liegen im Bereich der mechanischen Integration, der elektrischen Verkabelung und auch anderer infrastruktureller Maßnahmen, wie die Integration der Benutzerschnittstellen oder die Stromversorgung.

In diesem Kapitel werden die einzelnen Schritte der Hardware-Entwicklung und deren Umsetzung sowie die Integration in die bestehende Anlage unter Berücksichtigung der technischen Randbedingungen und Vorort-Gegebenheiten beschrieben.

6.1. I^2C-Schnittstellenplatine

Eine der Komponenten, die erst entwickelt werden mussten, stellt das I^2C-Schnittstellenmodul dar, die benötigten Haupt-Komponenten sind in Absch. 5.6 beschrieben. Zusätzlich wurde das Leistungsrelais zur Umschaltung der Drehrichtung auf dieses Modul integriert (Absch. 5.4) und als „zentrale Versorgungskomponente" werden alle Verkabelungen (Stromversorgung, Drehwinkel-Messignale, Motor-Steuersignale und Motor-Drehrichtungsumschaltung) auf dieser Karte zusammengeführt.

Daten zum Schaltungslayout finden sich im Anhang B.1 auf Seite 96, das Schaltungsschema ist im Anhang B.2 dargestellt und Details zur Verkabelung sind in Anhang B.3 zu finden.

6.1.1. Leiterplattenfertigung

Ein spezielles Verfahren zur Printplattenfertigung wurde verwendet, um bei der geringen Stückzahl der Steuerungsplatinen geringe Plattenfertigungskosten zu erzielen. Das Verfahren verwendet eine nummerisch gesteuerte Fräse, die die Außenkonturen der Leiterbahnen freifräst und damit ein Minimum an Kupfer und überhaupt keinerlei chemische Verfahren benötigt, in Abb. 6.1 ist die Leiterbahnseite der Schnittstellenkarte abgebildet.

6.1.2. Leiterbahnen CAM-Präprozessing

Zur Ansteuerung der Fräse wird das von HP zur Ansteuerung von Stiftplottern entwickelte Format HPGL (Hewlett Packard Graphic Language) verwendet.

Da die verwendete Layoutsoftware EAGLE (7.4.2.1) nicht in der Lage war, direkt HPGL zu erstellen, waren für die Erstellung der HPGL-Daten weitere Schritte notwendig.

1. Zuerst wurde eine Grafik-Datei im PNG-Format durch einen Datenexport aus der Layoutsoftware erzeugt.
2. Mit der Open Source Software AutoTrace (http://autotrace.sourceforge.net) wurden aus dem Bitmapformat die Leiterbahn-Außenkonturen errechnet und im DXF-Format ausgegeben.
3. Die DXF-Datei wurde in die CAD-Applikation VariCAD (http://www.varicad.com) eingelesen und dann im HPGL-Format geplottet (durch Ausgabe der Plottdaten in eine Datei).

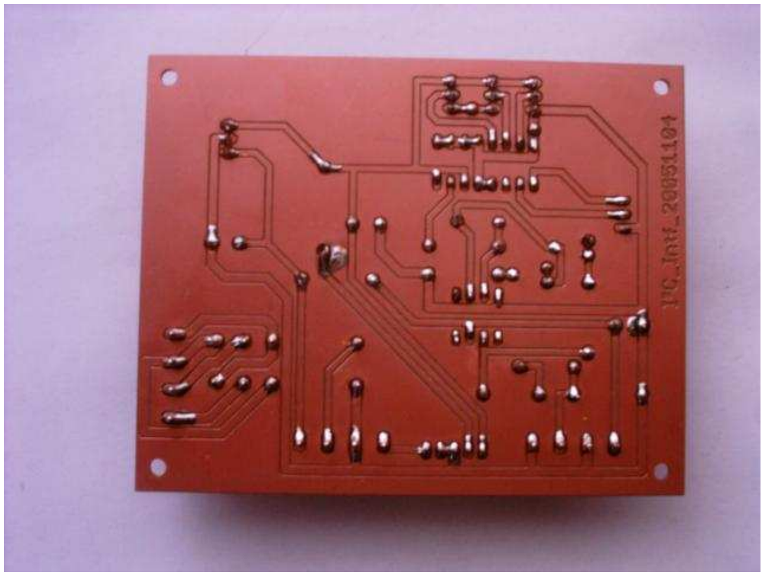


Abbildung 6.1.: I²C-Steuerungskarte, Unterseite

Die Aufrufparameter der Open Source Tracingsoftware AutoTrace sind im Anhang E.1 dokumentiert, in Anhang C.1 ist das gesamte Shellprogramm abgedruckt, das die Vektorisierung der Layoutdaten durchführt.

> Hinweise:
> Die Programme **pngtopnm** und **pnmtopng** konvertieren das PNG-Format ins PNM-Format und wieder zurück, **pnmflip** spiegelt die Grafikdatei, da der Export aus EAGLE grundsätzlich in der Draufsicht (Top) erfolgt. Diese Programme sind Teil der mächtigen Open Source NetPBM-Tools (http://netpbm.sourceforge.net) von Jef Poskanzer.

6.1.3. Bohrbild CAM-Präprozessing

Für die Bohrungen musste eine weitere Datenkonvertierung vorgenommen werden. Zur Erstellung des Bohrbildes wird ebenfalls das Dateiformat HPGL verwendet, wobei festgelegt ist, dass für jeden Bohrdurchmesser eine eigene HPGL-Datei erstellt werden muss.

Ein selbstgeschriebenes Programm wertet dazu mit Hilfe des Unix-Tools **awk** die Gerberdateien (Drill Data) aus der PCB-Software aus und erstellt pro Bohrdurchmesser eine eigene Koordinatendatei mit den Koordinaten in der Einheit Millimeter.

```
gerhardr@hpnb:~/eagle/rot_facility>./drd2dat.awk I2C_Intf_V7.drd
Tool change (01): 1-0.0315
Tool change (02): 2-0.0320
Tool change (03): 3-0.0360
Tool change (04): 4-0.0394
Tool change (05): 5-0.0400
Tool change (06): 6-0.0440
Tool change (07): 7-0.0512
Tool change (08): 8-0.0520
Tool change (09): 9-0.1102
```

Im angeführten Beispiel wurden 9 Koordinatendateien nach einer fixen Namenskonvention erstellt. Jede Datei beginnt mit der Vorsilbe „tool-“, daran angehängt wird der Bohrungsdurchmesser in Zoll und die Dateiendung lautet „.dat“, z.B. **tool-0.0315.dat**

Der Sourcecode dieses Präprozessors ist dem Anhang C.2 entnehmen.

Beispiel einer Ergebnis-Koordinatendatei:

```
gerhardr@hpnb:~/eagle/rot_facility>cat tool-0.1102.dat
95.9223 4.53762
95.9223 75.6576
4.48232 75.6576
4.48232 4.53762
```

> Bemerkung:
> In diesem Beispiel handelt es sich um die 4 Befestigungsbohrungen der Platine.

Um den Ablauf weiter automatisieren zu können, wurden diese Dateien über ein einfaches Shellscript in Makrodateien umgewandelt, die vom CAD-System eingelesen wurden um die notwendigen „Linienzüge” mit den entsprechenden Attributen (Layer) im CAD-System zu erstellen und sie gleich in Dateien zu „plotten”, die dann letztendlich an die Layoutfirma geliefert wurden.

Das gesamte Shellscript ist im Anhang C.3 nachzulesen.

6.2. Bedienung - Eingabe/Ausgabe

Die Anlage muss bedient werden (programmieren, kalibrieren, steuern und die „normale” Messtätigkeit) und die Informationen über den Betriebszustand müssen vorort „ablesbar” sein. Das dazu notwendige User-Interface wurde mit den standardmäßigen Ein-/Ausgabeeinheiten der **halvedDISC** realisiert.

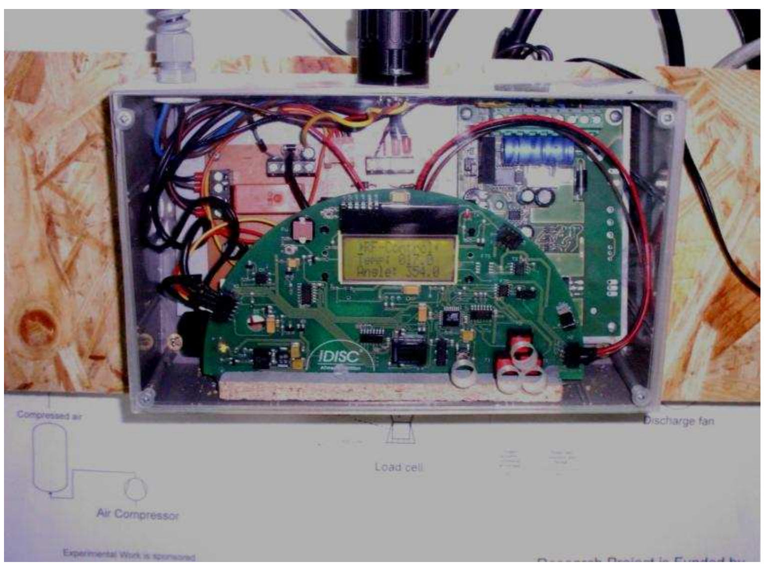


Abbildung 6.2.: Kontrolleinheit, Motorsteuerkarte und Schnittstellenkarte

Als Eingabe steht die Funktionstastatur, die aus 3 Tasten besteht, zur Verfügung, das I^2C-gesteuerte LCD-Display und die rote Leuchtdiode werden als Ausgabeeinheiten und Funktions-Feedback eingesetzt.

Die Mikrocontroller-Einheit wurde zusammen mit der I^2C-Schnittstellenkarte und der Motorcontrollerkarte in ein Gehäuse eingebaut (Abb. 6.2), das mit einem Plexiglasdeckel verschlossen ist und Öffnungen zur Bedienung der 3 Funktionstasten und zum Betätigen des Resettasters aufweist.

Die technischen Daten des halvedDISC Displays sind aus Tab. A.8 auf Seite 95 ersichtlich, die Daten der **halvedDISC** sind in Tab. A.1 auf Seite 91 angegeben und die technischen Daten zur Motorkontrollerkarte sind in Tab. A.3 nachzulesen.

6.3. Stromversorgung

Folgende Komponenten benötigen - zum Teil unterschiedliche - Versorgungsspannungen.

- Der Mikrocontroller wird mit 9 V betrieben, wobei ein eigenes Netzgerät mit ausgeliefert wird.
- Der Elektromotor ist für eine Betriebspannung von 24 V ausgelegt, ein max. Ankerstrom von 11 A ist angegeben.
- Die Motorregelung kann im Bereich von 24 V. . . 48 V betrieben werden.
- Das Richtungs-Umschaltrelais benötigt eine Spulenspannung von 24 V.
- Der OP als Spannungswandler ist für eine symmetrische Spannung von ±18 V ausgelegt.
- Der I^2C-Schnittstellenchip kann im Bereich von 2.6 V. . . 6 V betrieben werden und ist damit TTL-tauglich.
- Der Winkelschrittgeber kann von 4.5 V. . . 5.5 V eingesetzt werden, d.h. ebenfalls TTL-Pegel.

Im Fundus der TU-Graz befand sich ein unstabilisiertes Netzgerät mit einer geglätteten Nennspannung von 24 V und einer Stromstärke von 10 A, das aus einem ausgemusterten Messgerät ausgebaut worden war. Das Gerät kann durch unterschiedliche Primärwindungsanschlüsse auf ca. 10 % Über- bzw. Unterspannung umgestellt werden. Das Netzgerät wurde mit der unteren Spannungsbeschaltung mit einer Leerlaufspannung von 25.5 V letztendlich eingesetzt.

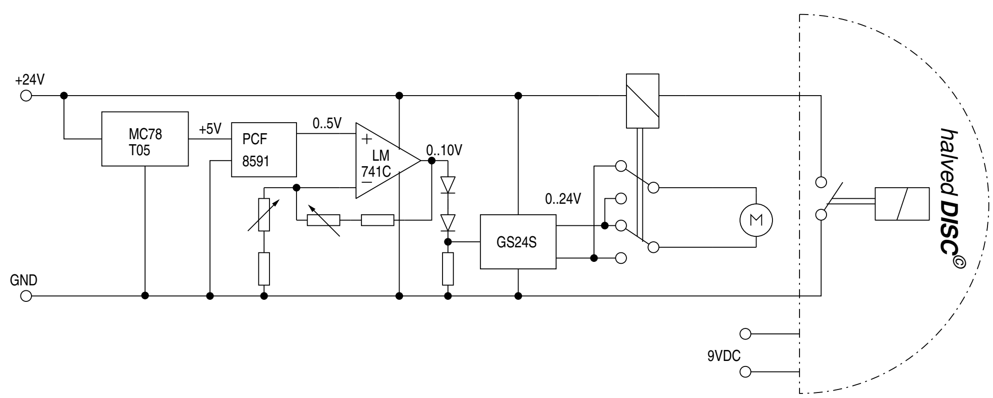


Abbildung 6.3.: Stromversorgung, vereinfachtes Schema

Um dieses Netzgerät verwenden zu können, wurde die Versorgungssituation nach Abb. 6.3 festgelegt.

6.3.1. Asymmetrische OP-Verstärkung

Wie aus Grafik 6.3 ersichtlich, wird der Eingang des Operationsverstärkers im Bereich von 0. . . 5 V verwendet, die Spannungsversorgung liegt im Bereich 0 bis +24 V. Typische OPs sind jedoch ausgelegt, um im Bereich der Mittelspannung U_m betrieben zu werden - ein typisches Beispiel mit symmetrischer Spannungsversorgung könnte folgendermaßen aussehen:

$$U_m = \frac{U_{max} + U_{min}}{2} = \frac{12V + (-12V)}{2} = 0V$$

Was wiederum umgelegt auf die aktuelle Versorgungssituation folgendes bedeuten würde:

$$U_m = \frac{U_{max} + U_{min}}{2} = \frac{24V + 0V}{2} = 12V$$

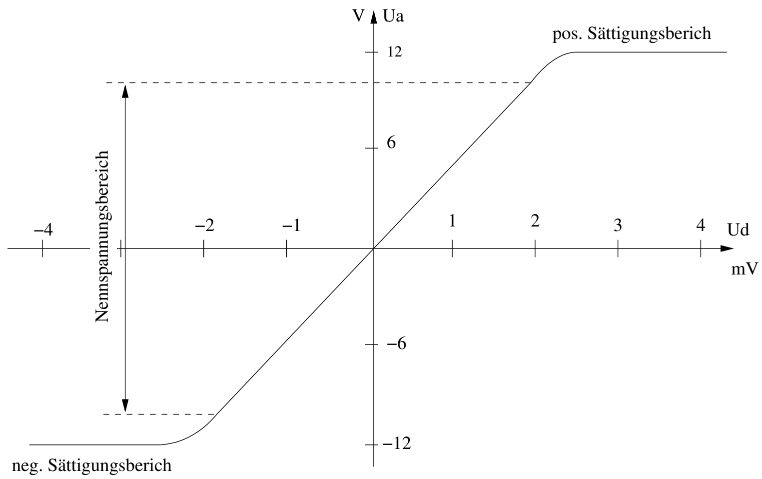


Abbildung 6.4.: Orationsverstärker-Kennlinie vereinfacht

Wie wirkt sich dies nun auf die vorliegende Situation aus?

Der Verlauf einer typischen Kennlinie eines Operationsverstärkers erreicht in den beiden Endbereichen jeweils einen Sättigungsbereich, in dem keine lineare Verstärkung mehr stattfindet (Abb. 6.4). Im vorliegenden Konzept wird der OP im Eingangspannungsbereich von 0. . . 5 V betrieben, wobei der negative Eingang direkt auf Masse liegt und am positiven Eingang die Ausgangsspannung des D/A-Wandlers anliegt, der OP wird somit nahe am unteren Sättigungsbereich betrieben. Die durch diesen Effekt entstehende Offsetspannung (siehe Abb. 6.5) wurde durch 2 Dioden in der Ausgangsleitung wiederum gegen 0 V verschoben (Abb. 6.3).

6.3.2. Kennlinienkorrektur

Ebenfalls wurde die Festlegung getroffen, dass die „Steigung” der Kennlinie einstellbar sein soll, da man das Verhalten der Motorsteuerung in der produktiven Anlage nicht genau voraussagen konnte. Dies wurde durch jeweils 1 Potentiometer im OP-Rückkopplungszweig und eines in der Massestrecke des negativen Eingangs erreicht. Mit diesen Reglern kann die durch die Offset-Dioden nach unten korrigierte Kennlinie wiederum so angepasst werden kann, dass am oberen Ende der Kennlinie die geforderten 10 V auftreten.

Durch Ausnutzung der beiden Endpositionen der Potis ergibt sich nun eine minimale und maximale Verstärkung, die den Einstellbereich der Kennlinie beschreiben.

$$V_{min} = \frac{U_{a1}}{U_e} = \left(1 + \frac{100k\Omega}{200k\Omega}\right) = 1.5 \;\; V_{max} = \frac{U_{a2}}{U_e} = \left(1 + \frac{200k\Omega}{100k\Omega}\right) = 3$$

Damit bewegt sich die einstellbare Ausgangsspannung U_a bei einer Eingangspannung von $U_e = 5V$ im Bereich:

$$7.5V \geq U_a \geq 15V$$

Das komplette Schaltungschema der Schnittstellenkarte ist in Anhang B.2 zu finden.

Diese Situation wurde mittels einer einfachen Spice-Simulation (Abb. 6.5) untersucht, wobei folgende Kriterien analysiert werden sollten:

1. Der Ausgangs-Spannungsverlauf nahe an der Spannung 0 V

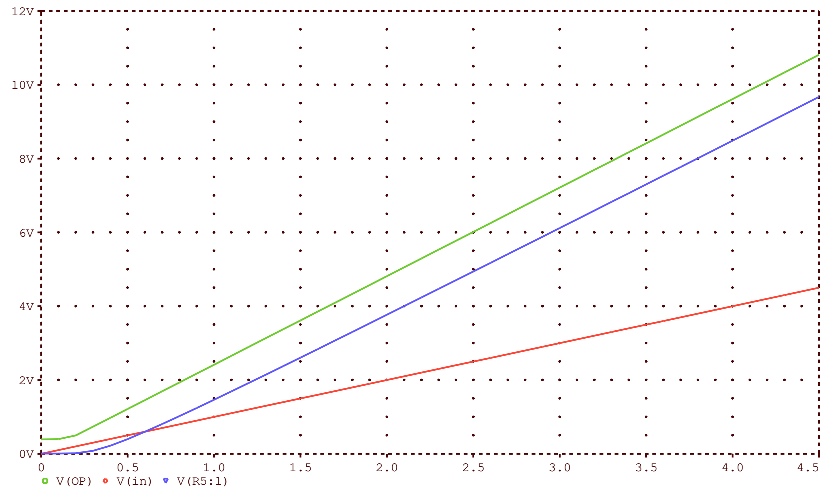


Abbildung 6.5.: Operationsverstärker-Kennlininenkorrektur

2. Die Auswirkung der Kennlinienkorrektur-Potentiometer
3. Die Auswirkung der Spannungsoffset-Dioden

Mangels vorliegender exakter Modellbeschreibung des verwendeten OPs konnte nur das prinzipielle Verhalten der Spannungsversorgung überprüft werden, das genaue Verhalten der Schaltung konnte erst nach Aufbau der Platine untersucht werden.

Im praktischen Aufbau stellte sich dabei eine abweichende Situation ein.

- Der Nullspannungsoffset lag mit ca. +2.5 V wesentlich höher als im Simulationsmodell ermittelt (0.5 V).
- Der Spannungsabfall an den beiden Offsetdioden betrug die erwarteten 1.4 V womit eine Vorspannung von ca. 1 V bei einer Eingangs-Steuerspannung 0 V auftrat.

Diese Tatsache wurde mit den Betreibern der Anlage diskutiert und man kam gemeinsam zur Ansicht, dass einerseits diese kleine Vorspannung zu einem Haltemoment in Richtung der vorgegebenen Drehbewegung führt und somit ein „zurückfedern” aus der Endposition verhindert oder zumindest vermindert wird, andererseits weit genug von der eigentlichen Spannung, bei der die Drehbewegung einsetzt, entfernt ist.

Tests beim Systemaufbau ergaben, dass dieser „Losfahrwert” bei ca. 50 % der Maximalspannung liegt, der bei 0 V Steuerspannung auftretende „Vorspannungswert” kann somit mit ca. 10 % der Maximalsteuerspannung als unkritisch angesehen werden.

6.4. Wirtschaftliche Betrachtung

Nach Festlegung dieser Randbedingungen soll auch noch die Preiskomponente an Hand einer einfachen Kalkulation betrachtet werden.

In dieser Beispiel-Kalkulation wird der in der Leistungsklasse vergleichbare Schrittmotor ST11018M6504 der Firma Nanotec GmbH und Co KG. (http://www.nanotec.de) inkl. des dazugehörigen Endstufentreibers SMC46 aus Tabelle 6.1 mit der im Kapitel 5 beschriebenen Gleichstrommotorlösung mit Leistungsregler und Winkelmesseinrichtung aus Tabelle 6.2 verglichen.

Tabelle 6.1.: Kalkulation Schrittmotorantrieb

Komponente	Preis [EUR]	Bemerkung
Schrittmotor	267.40	Haltemoment 20 Nm, Schrittwinkel 1.8°
Endstufe	279.00	24-48 V, 2.5-6 A
Summe	546.40	

Tabelle 6.2.: Kalkulation Gleichstrommotorantrieb

Komponente	Preis [EUR]	Bemerkung
Gleichstrommotor	114.00	siehe 5.2
Motorsteuerung	58.40	siehe 5.3
Winkelschrittgeber	23.15	siehe 5.5
I^2C-Schnittstelle	47.98	nur Materialkosten berücksichtigt
Platinenfertigung	60.00	I^2C-Schnittstellenplatine
Summe	303.53	

Damit ist eine weitere Begründung für die Entscheidung, den „einfachen" Gleichstromaktuator zu verwenden, gegeben. Weitere Randbedingungen, wie der z.B. der Platzbedarf (Abb. 6.6) und Ansteuermöglichkeiten des Schrittmotors sind hier nicht berücksichtigt und müssten gegebenenfalls gesondert evaluiert werden.

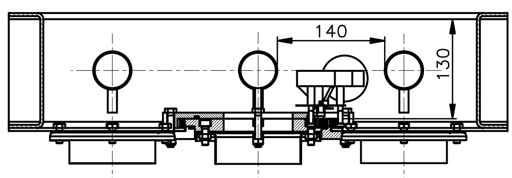


Abbildung 6.6.: Motor-Einbausituation

6.5. Hardwareumsetzung und Module

Mit diesen gewählten Komponenten wurde nun das Hardwarekonzept aus Abb. 5.6 in Module eingeteilt, die in Abbildung 6.7 dargestellt sind. Err.: 4.10.06 Absch. 5.6

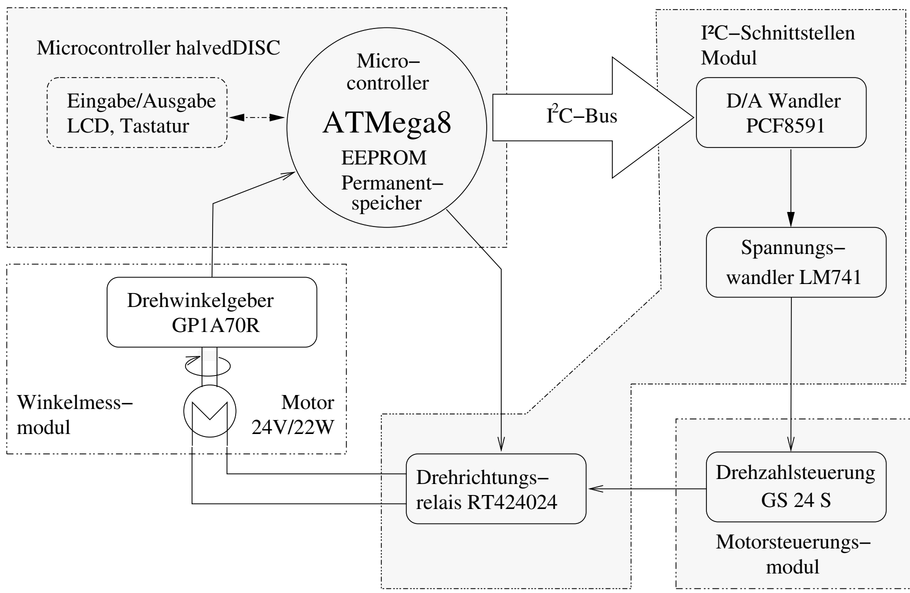


Abbildung 6.7.: Hardware-Modulkonzept

Die einzelnen Platinen, auf denen die Module konkret untergebracht wurden, sind in dieser Grafik über zusammenhängende Bereiche dargestellt.

Die gesamte Verkabelung läuft über das I^2C-Schnittstellenmodul nach Anhang B.3, in Abbildung 6.2 auf Seite 31 sind die einzelnen Platinen in das Gehäuse eingebaut dargestellt. Einzig allein das Winkelmessmodul ist direkt am Gleichstrommotor angebracht (siehe Abb. 5.5 auf Seite 27) und nicht im Steuergehäuse untergebracht.

7. Steuerungssoftware

Dieses Kapitel ist der Steuerungssoftware gewidmet. Da die Software einen zentralen und bestimmenden Faktor im vorgestellten Gesamtkonzept darstellt, wird diese sehr umfangreich behandelt. Es werden aus dem Anforderungsprofil die einzelnen Software-Schichten abgeleitet, der Software-Aufbau aus Sicht der funktionalen Gliederung als auch aus organisatorischer Sicht erläutert. Die „Methodik” dieser Projektumsetzung stellt einen weiteren Schwerpunkt dieser Arbeit dar und soll auch bei Entwicklung ähnlich gearteter Projekte eine Hilfe darstellen.

7.1. Softwareanforderungen

Die Anforderungen, die in den vorhergehenden Kapiteln bereits detailliert behandelt wurden, und weitere Vorgaben von der TU Graz, führten zur Festlegung einer Liste von geforderten Funktionen:

1. Anfahren beliebiger Winkel mit einer Genauigkeit von 1 Winkelgrad
2. Speichern von bis zu 16 vorgegebenen Winkellagen
3. Handsteuerung in Form von direkten Steuerspannungsvorgaben
4. Handsteuerung in Form von beliebigen Winkelvorgaben
5. „Sanftes” Anfahren und Niederfahren der Drehbewegung
6. Kalibrierung zur Festlegung eines beliebigen Bezugswinkels

7.2. Softwarekomponenten

Das Zusammenspiel der einzelnen Software- und Hardware-Komponenten ist in Abb. 7.1 dargestellt und stellt die Zusammenhänge zwischen den einzelnen Komponenten und die Informationspfade dar.

In den folgenden Abschnitten werden die einzelnen logischen Komponenten behandelt und detailliert dargestellt.

7.2.1. Benutzerkommunikations-Modul

Diese sehr wichtige Komponente ist dafür verantwortlich, wie effizient die Anlage bedient werden kann, bestimmt vorwiegend die Anwenderakzeptanz und stellt im weitesten Sinne das „Aushängeschild” des Produktes dar, da die Funktionalität dem Anwender in dieser Schicht „präsentiert” wird. Die „inneren” Mechanismen - die Steuerungslogik - werden selten bewusst wahrgenommen und bewertet, da diese „einfach nur funktionieren” müssen.

Das GUI (**G**raphical **U**ser **I**nterface) verwendet extensiv Ein- und Ausgabefunktionen. Dies sind einerseits die benutzerspezifischen Eingaben und andererseits Rückmeldungen über den Hardwarestatus, wie die aktuelle Winkellage, Außentemperatur, „Geschwindigkeits-Cursor” (dazu später mehr in Absch. 7.6.2), usw. und der Benutzer muss dazu in einer einheitlichen und konsistenten Art durch das GUI quasi „geführt” werden, um ihn von der Komplexität der Steuerungslogik zu entlasten (siehe Absch. 7.6.3).

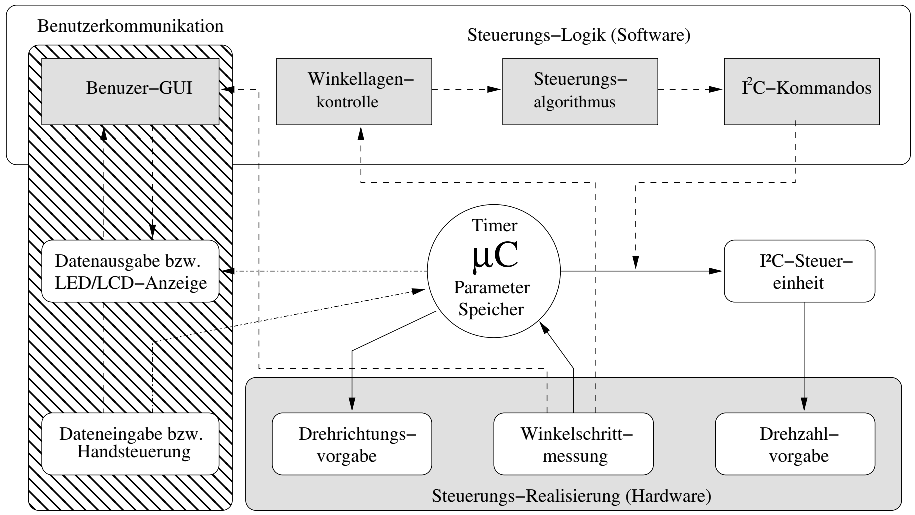


Abbildung 7.1.: Softwarekonzept und Modulzusammenhänge

7.2.2. Steuerungs-Logik

Das Hauptziel der Anlage (Kap. 2) liegt in der Anforderung, die Drehbarkeit eines Filterkorbes zu gewährleisten, um Messungen außerhalb des „normal sichtbaren” Bereiches durchführen zu können. Dies bedingt, dass sich die Winkellage dieses Teiles stufenlos (im Rahmen der Steuerungausauflösung) einstellen lässt.

Üblicherweise wird eine Position nach der anderen in einem vorgegebenen Schema angefahren. Dies muss soweit wie möglich automatisiert geschehen, wobei die technischen Randbedingungen des Messverfahrens (definiert in Absch. 2.1.1) in der Anlagensteuerung berücksichtigt werden müssen. Das „ruckfreie Anfahren” aus dem Stillstand und das „sanfte Herunterfahren” in der Endposition muss die Anlage selbständig durchführen.

Trotz dieser Automatismen, muss der Anwender in der Lage sein, weitere Positionen „händisch” anzusteuern und damit zusätzliche Parameter im „Teach-In” Verfahren festzulegen, bzw. bereits gespeicherte parametrisierte Positionen wieder zu löschen.

Und letztendlich ist vorgesehen, dass beliebige Winkelpositionen manuell definiert werden können, um diese mit der automatischen Positionierroutine im Bedarfsfall „anzufahren”.

7.2.3. Steuerungs-Realisierung

Um die vorhin geforderten Eigenschaften zu erzielen, müssen - unter Verwendung der der GUI-Komponente und der Steuerungslogik - die physischen Eigenschaften gemäß des in Abb. 5.6 auf Seite 28 dargestellten Regelkreises gemessen und voreingestellt werden. Dieser Teil wird direkt in Motorsteuervorgaben umgesetzt, bzw. als Rückmeldung direkt vom Drehwinkelgeber an der Motorwelle entnommen.

- Die Sollwertvorgabe ergibt sich aus dem errechneten äquivalenten Spannungswert für die Motorsteuerung, die über die I^2C-Schnittstellenkarte an die Motorsteuerkarte übertragen wird. Zusätzlich wird die Drehrichtung über ein Mikrocontrollerport vorgegeben, welches das Leistungsrelais zur Richtungsumschaltung betätigt.
- Die Rückmeldung erfolgt direkt über den Drehwinkelgeber, der einen selbständigen Zähler des Mikrocontrollers triggert und somit grundsätzlich ohne Interruptanforderung auskommt (dazu mehr in Absch. 7.7.3.4).
- Die Steuerung wird in der Mikrocontroller-Programmlogik abgebildet und in den entsprechende Phasen durch die notwendigen Sollwertvorgaben umgesetzt. In bestimmten Situationen muss die

Drehwinkellage direkt gelesen und darauf reagiert werden (z.B. bei Erreichen einer Endlage). In dieser Phase wird deshalb der Eingabewert im „Polling-Verfahren” (weitere Details dazu in Absch. 7.7.3) ständig abgetastet, um die notwendigen Reaktionen im entsprechenden Augenblick einleiten zu können.

7.3. Modularität, Basissystem, Systemaufbau

Die Beschränkung auf den Programm-Speicherbereich von 8 Kb Flash RAM, die Tatsache, dass Assemblerprogrammierung eine maximale Optimierung ermöglicht und der Versuch, eine Methode zur effizienten Umsetzung größerer Projekte mittels Assembler zu erarbeiten, haben dazu geführt, dass die gesamte Software in ATMEL Assembler umgesetzt wurde.

Um nun Software in dieser relativ komplexen Anforderung effizient umsetzten zu können, ist es notwendig, die Programmstruktur modular zu halten und gewisse Abstraktionsschichten einzuführen, um, einerseits die Entwicklung Problemkreis bezogen durchführen zu können und andererseits möglichst „einfache” und überschaubare Programmmodule zur Verfügung zu haben. Damit ist zusätzlich auch ein gewisses Maß an Portabilität zu erreichen, falls eine ähnliche Anforderung mit einer anderen Hardware umgesetzt werden soll.

Abb. 7.2 zeigt, dass das Basissystem - wie auch bei Betriebssystemen üblich - in einzelne Schichten unterteilt wurde [7] S. 175. Die Grundidee, die dahinter steckt, ist, dass eine Schicht bestimmte „Services” liefert, die von der darüber liegenden Schicht ohne Berücksichtigung der darunter liegenden Mechanismen verwendet können.

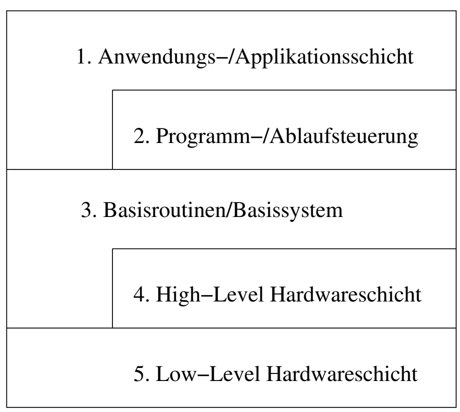


Abbildung 7.2.: Applikationsaufbau

Hinweis:
Diese Gliederung darf nicht mit dem ISO-OSI Schichten-Modell verwechselt werden, das vorwiegend für den Kommunikationsbereich definiert wurde, sondern die Schichten wurden hier ausschließlich nach definierten, aus den Gegebenheiten für sinnvoll erachteten Grenzen festgelegt.

- Zusätzlich wurde durch extensive Nutzung von „symbolischen” Konstanten und durch die Verwendung vom Assembler-Makros versucht, die Lesbarkeit des Source-Codes zu verbessern um damit einen wesentlichen Nachteil gegenüber einer Hochsprachenumsetzung wieder Wett zu machen.

1. Im vorliegenden Fall besteht die Applikationsschicht aus dem Benutzerinterface (GUI) und der Menüsteuerung der obersten Menüebene. Diese besteht aus einem Menüauswahl Event-Loop, um eine der Top-Level Funktionen und damit den Funktionsmodus festzulegen (siehe dazu auch 7.6.1). Wie im folgenden Punkt ersichtlich, wurde diese Schicht vorwiegend aus software-organisatorischen

Überlegungen eingeführt, sie beinhaltet keine funktionellen Komponenten, die in Schicht 2 ausgeführt werden, eine Änderung der Benutzeroberfläche (wie z.B. die Implementierung von Zeit- und Datumsanzeigen) würde in diesen Bereich fallen.

2. Die Programmsteuerung ist in der Art implementiert, dass für jeden Betriebsmodus entweder die notwendigen Untermenüpunkte oder die entsprechenden Programmfunktionen als Aufruf eines Unterprogramms umgesetzt werden. Diese Schicht verwendet ebenfalls die Mechanismen, wie sie in Schicht 1 beschrieben sind und setzen auf die Basisroutinen der darunterliegenden Schicht auf. Die Trennung in Applikationsschicht und Basissoftware wurde vorwiegend auf Grund der in Absch. 7.5 vorgegebenen Code-Organisation eingeführt. Für jeden Hauptmenüpunkt (Modus) existiert genau eine Sourcedatei, die die einzelnen Funktionen umsetzt.

3. An Basisfunktionen stehen Funktionen zur Ein- und Ausgabe von Menü-Interaktionen, LCD-Texten, das Lesen und Schreiben von gespeicherten Werten, die Anzeige von Stati sowie spezielle Ausgabeobjekte wie „Speedcursor" oder „Menüselection" (siehe dazu auch 7.6.2) zur Verfügung. Diese Schicht zeichnet sich dadurch aus, dass die zugeordneten Routinen nicht direkt auf die Hardware zugreifen, sondern Routinen ansprechen, die quasi als „High-Level" Routinen zur Verfügung gestellt werden. In dieser Ebene sind auch die Software-Timer angesiedelt, die nur mittels Schleifen Zeitverzögerungen bewirken oder die Pseudo-Scheduler Schnittstelle (siehe 7.7.3.3), die Interrupt gesteuerte Ereignisse (wie z.B. die Aktualisierung der Anzeige in Schleifen) verwaltet.

4. Die High-Level Hardwareebene greift auf die Ressourcen der halvedDISC zu, verwendet jedoch die darunter liegende Low-Level Abstraktionsschicht, die dann endgültig die Ports ansprechen, Timings berücksichtigen, etc. In dieser Schicht sind Routinen zur Ausgabe typspezifischer Werte (Zahlen im Dezimal-, BCD-, Binär- oder Hexadezimalformat, Winkelanzeige, Hex-Byte, Hex-Word Ausgaben, ASCII-Text) oder die Tastaturbearbeitung angesiedelt. Diese benötigen dazu die Low-Level Routinen, wie z.B. Byte weises Lesen und Schreiben von I^2C-Komponenten oder Port-I/O Funktionen.

5. Die Low-Level Schicht stellt die direkte Verbindung zur Hardware her. Routinen zur Ansteuerung von I^2C-Komponenten, wie dem PCF8591, Einlesen oder Schreiben eines Temperatur-, Zeit- oder Datumswertes werden in dieser Ebene umgesetzt. Diese Softwareebene ist bis zur Adressierung des I^2C-Busses Hardware spezifisch, und muss in fast jedem Fall bei einer Änderung von Hardwarekomponenten mitgeändert werden. Da die Low-Level Routinen in jedem Mikrocontroller-Projekt vorhanden sein müssen, wurde in diesem Bereich großteils auf bestehenden Code (z.B. die IIC-Lib oder LCD-Lib von Heiko Polster, ACMC oder die Timing-Routinen Harald Knapp, www.comlogic.at) zurückgegriffen.

7.4. Verwendete Werkzeuge und Hilfsmittel

Für die Entwicklung eines Produktes werden diverse Werkzeuge benötigt, die die Arbeit mehr oder weniger gut unterstützen. Während professionelle Werkzeuge mehrere zehntausend Euro kosten können, gibt es Produkte, die wesentlich günstiger oder kostenlos zu haben sind, die z.T. eine beachtliche Leistungsfähigkeit aufweisen (siehe z.B. die „Usage-Meldung" des ISP-Programmers in Anhang E.3 auf Seite 107). Diese Werkzeuge stammen üblicherweise aus den universitären Bereich oder aus Open Source Entwicklungen an denen oft eine große Anzahl an Entwicklern beteiligt sind und stellen daher „öffentlich zugängliches Wissen" dar.

Dieser Abschnitt beschäftigt sich mit den Werkzeugen, die für die Entwicklung dieses Produktes eingesetzt oder erst entwickelt wurden. Weiters umfasst dieser Abschnitt Arbeitsweisen und Methoden für bestimmte z. T. automatisierbare oder zumindest unterstützende Tätigkeiten.

7.4.1. Gesellschaftspolitisches

Der Autor dieser Arbeit beschäftigt sich bereits seit Jahren mit dem Thema „Open Source", freier Software und freien Softwarelizenzen (GPL, LGPL, BSD License . . .) [8]. Mit diesem Projekt war damit auch die Möglichkeit gegeben, zu testen, ob und inwieweit der gesamte Entwicklungsprozess mit freier Software abgedeckt werden kann. Speziell für Einzelpersonen, im Ausbildungsbereich oder für kleine Firmen besteht damit eine Möglichkeit kostengünstig Entwicklungen durchzuführen, die sonst nur mit sehr teuren professionellen Produkten durchgeführt werden können.

Zusätzlich liegt es durchaus im Interesse des Auftraggebers der Anlage, im universitären Bereich entwickelte oder frei verfügbare Werkzeuge vorzufinden, die einerseits im Unterrichtsbereich verwendet werden können und weiters eine gute Zugänglichkeit der Werkzeuge ermöglicht und damit einen Langzeitsicherheitsaspekt für die Weiterbetreuung der Anlage darstellt.

7.4.2. Verwendete Entwicklungswerkzeuge

Die gesamte Entwicklung fand unter Linux statt [9].

Linux ist ein Unix ähnliches Betriebssystem, das 1991 vom Schweden Linus Torvalds erstmals der Öffentlichkeit vorgestellt wurde. Der Kern des Betriebssystems, sowie auch die meisten anderen Teile, stehen unter der freien Softwarelizenz GPL (GNU General Public License), die 1991 von der Free Software Foundation, Inc. veröffentlicht wurde, zur Verfügung [10].

Die GPL verfolgt - sehr vereinfacht ausgedrückt - folgende Zielsetzung:

> Für, unter der freien Lizenz GPL veröffentlichte Software muss der Sourcecode der Software öffentlich zugänglich gemacht werden. Die Software darf von jedem (der die GPL akzeptiert) uneingeschränkt verwendet werden. Wird ein GPL-Produkt aber abgeändert (zB. erweitert oder verbessert), muss das „neue” Produkt wieder unter den Bedingungen der GPL veröffentlicht werden und somit wieder Zugang zum Sourcecode ermöglicht werden.

7.4.2.1. Schaltplanentwicklung

Die Schaltplanentwicklung wurde nicht mit Open-Source Software durchgeführt.

Der Hauptgrund für diese Maßnahme liegt in der Tatsache, dass ein früher Prototyp der I^2C-Schnittstellenkarte bereits mit dieser Software für ein Projekt an der FH Mittweida umgesetzt wurde und somit eine Änderung bzw. Erweiterung dieser Entwicklung darstellt.

Der Schaltplanentwurf und das Layouting wurde mit der freien Version der Layout Software „EAGLE” der CadSoft Computer GmbH (http://www.cadsoft.de) durchgeführt. Diese Software steht für MS-Windows, Linux und Mac OSX zur Verfügung und kann als eingeschränkte Version im nicht-kommerziellen Bereich frei genutzt werden. Für dieses CAD Programm stehen ebenfalls freie Teilebibliotheken, Zusatzapplikationen (User Language Programs) und zahlreiche Informationen im Internet zur Verfügung.

7.4.2.2. Assembler

Als Assembler wurde AVRA (http://avra.sourceforge.net) „advanced AVR macro assembler” Version 1.0.1 Build 113 (12. February 2004), Copyright (C) 1998-2004 Jon Anders Haugum, Tobias Weber eingesetzt. Dieser fast vollständig zum ATMEL Assembler AVRASM32 kompatible Assembler ist unter allen 32-bit MS Windows (95/98/NT/2000/XP) und allen POSIX (Linux/BSD/UNIX-like OSes) lauffähig und enthält einige Verbesserungen im Bereich der Makro-Verwendungen. Makros wurden sehr extensiv im Projekt genutzt.

Der GPL Assembler wurde im Sourcecode heruntergeladen und unter diversen Linux-Versionen kompiliert.

Im Anhang E.2 sind weitere Informationen in Form der „Usage-Meldung” verfügbar.

7.4.2.3. ISP-Programmierung

Der Mikrocontroller ATMega8 verwendet zum Laden des fertigen Assemblercodes in den Mikrocontroller ein Programmiergerät, das an die ISP-Schnittstelle (**I**n **C**ircuit **P**rogrammer) angeschlossen ist und in der vorliegenden Version über die serielle Schnittstelle des PCs angesprochen wurde.

Das GPL-Programm UISP (http://savannah.nongnu.org/projects/uisp) in der Version 20050207 von Marek Michalkiewicz wurde zum Programmieren des Controllers eingesetzt.

In Anhang E.3 ist die beeindruckende Leistungsfähigkeit dieser Software ersichtlich.

Die Software wurde zum Teil heruntergeladen und kompiliert, für einige Linux-Distributionen werden fertig kompilierte Pakete angeboten, die direkt über das Internet installiert wurden.

7.4.2.4. LCD Editor/Simulator

Aus Erfahrungen mit anderen Projekten war bekannt, dass für die Erstellung von Handbüchern viele „Dialog-Bildschirme” benötigt werden. Weiters ist die Erstellung von Assembler-Defines für LCD-Displayausgaben sehr aufwändig, da man in der Regel umständlich in Code-Tabellen nachschlagen muss, da für die 7-Bit Philosophie der Anzeigeeinheit EA7123-I2C mit 2 Zeichensätzen die richtige „Seite” gewählt werden muss, nicht-alphanummerische Zeichen oft auf unüblichen Positionen zu finden sind und es auch für die zahlreichen LCD-Grafikzeichen keine ASCII-Repräsentation gibt und diese daher optisch nicht über Textzeichen „dargestellt” werden können.

Dies führte zur Entwicklung eines Programms mit dem Namen „LCD-Edit” (siehe Abb. 7.3), das folgende Features aufweist:

1. Grafische Darstellung aller möglichen Zeichen der LCD-Punktmatrixanzeige
2. Eingabemöglichkeit für die 3 LCD-Zeilen im ASCII und Hexadezimalmodus
3. Das zeichenweise Setzen oder Löschen des Bit 7 (markieren im ASCII-Fenster und betätigen der Schalter „Set” oder „Clr” Bit7).
4. Das Speichern des LCD-Inhaltes in Dateien und das Wiedereinlesen derselben
5. Die Ausgabe des Inhaltes des Grafikfensters als Postscriptdatei zum Drucken oder Einfügen als Grafik in Dokumente
6. Die Ausgabe von Assembler Define-Statements für alle LCD-Zeilen, z.B.

   ```
   LCD0: .db $2e,$e8,$e1,$ec,$f6,$e5,$e4,$c4,$c9,$d3,$c3,$2f,$0,$0
   ; .halvedDISC/
   ```

7. Die Darstellung aller LCD-Zeichen in einer grafischen Tabelle zur bequemen Vorschau und Eingabe von unüblichen Codes im Hexcode
8. Beliebige Definitionen von Hintergrund- und Zeichenfarbe des Grafikbereiches von LCD-Edit

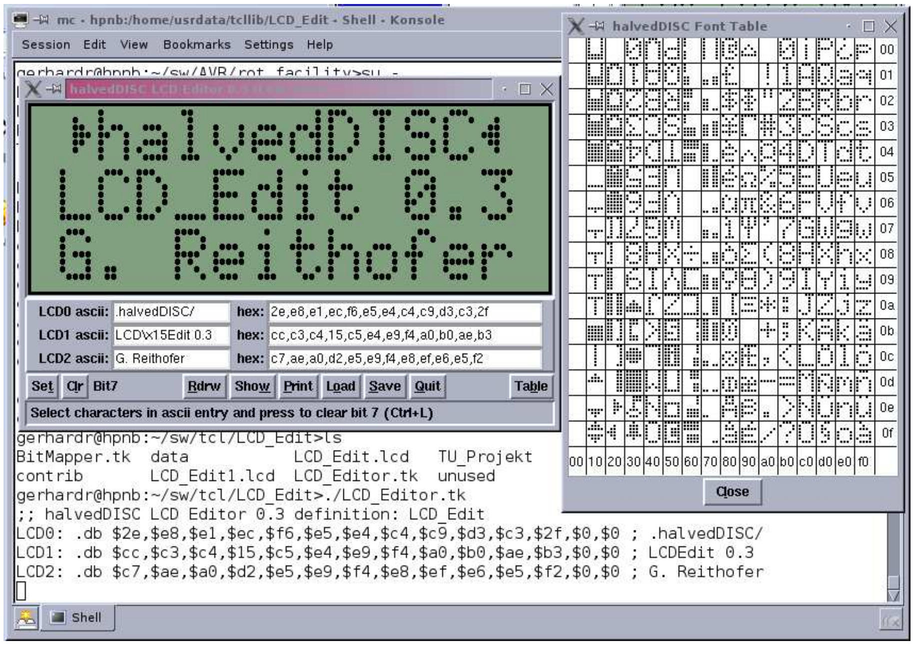


Abbildung 7.3.: LCD-Editor

Das Programm wurde in der Scriptsprache Tcl/Tk (http://www.tcl.tk) programmiert und ist somit auf allen Plattformen lauffähig, für die Tcl/Tk existiert. Der **T**ool **C**ommand **L**anguge Interpreter (TCL) und das grafische **T**ool**K**it (TK) sind ebenfalls Open Source Software und stehen unter der Tcl/Tk Lizenz zur Verfügung [11].

Tcl/Tk wurde mittlerweile auf 18 Plattformen portiert, darunter Windows-9x, Windows 32-Bit, Mac OS X und OS 9, alle Linux-Arten incl. PowerPC Architektur, die meisten Unix Arten, OS/390 MVS/OpenEdition OSF und zahlreiche PDA Plattformen.

LCD-Edit wird in absehbarer Zeit ebenfalls als GPL Programm veröffentlicht werden und gegebenenfalls weiterentwickelt und verbessert.

7.4.2.5. Image Processing Software

Um die Wiederholgenauigkeit der Drehvorrichtung nachzuweisen, wurden mit den bestehenden Messkameras Bilderserien erstellt (siehe Absch. 9.5). Um diese Daten quantitativ auswerten zu können müssen 2 Bilder, die nacheinander aus der gleichen Winkelposition nach einer oder mehreren Umdrehungen der Vorrichtung erstellt wurden, miteinander verglichen werden.

Zum Auswerten dieser Bilder wurde ein weiteres Programm erstellt, das es ermöglicht, 2 „Images” pixelweise zu vergleichen und Abweichungen optisch darzustellen.

Dieses Programm, mit dem Namen „Simple Image Processor”, weist eine Reihe von Features auf, mit denen 2 Bilder „quantitativ” verglichen werden können.

1. einlesen von 2 Bitmap-Dateien und speichern des Ergebnisses als Bitmap-Datei
2. direkter optischer Vergleich, durch „Umschalten” zwischen Bild 1 und Bild 2
3. Anzeige der Grauwerte, bzw. RGB-Komponenten auf Bildpunktbasis
4. Distanzmessung zwischen 2 Bildpunkten (in mm und Bildpunkten)
5. Beschneidung des Bildes auf einen beliebigen rechteckigen Bereich (Cropping)

6. Beliebige Vergrößerung und Verkleinerung um den Faktor 2 (Pixel Duplicating und Pixel Subsampling)
7. „Überlagern”, Löschen und Verschieben von Markierungen (Image-Marker) zum lokalen optischen Vergleich in Form von einzelnen waagrechten und senkrechten Linien, Ellipsen, Kreisen und einem Linienraster
8. Bildmanipulation durch Verschieben der Graustufen (abdunkeln oder aufhellen), Erhöhung und Verminderung des Kontrastes, obere und untere Schwellwert-Isolierung und Invertierung des Grauwertes
9. Berechnung eines „Differenzbildes” auf Pixelebene, wobei die Grauwert-Abweichungen in Helligkeits- oder Farbstufen umgerechnet werden

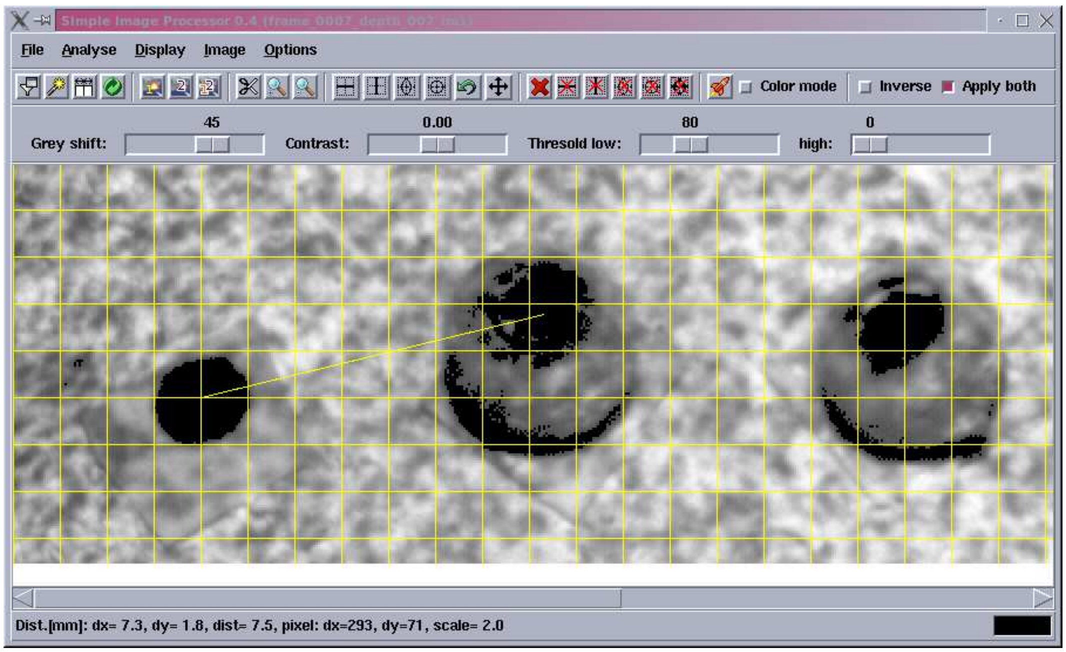


Abbildung 7.4.: Simple Image Processor

Die Abbildungen werden grundsätzlich in dem, für Bildschirme üblichen RGB-Farbraum mit 8-Bit Farbtiefe behandelt [12] S. 585. Da das Messsystem nur Schwarzweißbilder liefert, sind bei den verwendeten Graustufenbildern die Rot-, Grün- und Blauanteile jeweils gleich. Die „Image-Marker” werden farbig dargestellt um sie leichter von den Nutzdaten unterscheiden zu können.

Die Bildmanipulations-Operationen lassen sich am einfachsten mathematisch beschreiben.

Grey shifting Zu jedem Farbwert eines Bildpunktes wird ein konstanter Wert hinzugezählt, der Bereich dieser Korrekturwerte reicht von -254...254.
Formel: $g_1 = g_0 + c$

Contrast stretching Der Grauwert jedes Bildpunktes wird mit einer exponentiellen Konstanten multipliziert, der Wert der Konstanten reicht von -3...3.
Formel: $g_1 = g_0 * 2^c$

Threshold low Jeder Bildpunkt, dessen Grauwert einen definierten Grenzwert unterschreitet, wird auf 0 (=Schwarz) gesetzt (unterer Schwellwert). Dieser Bereich kann von 0...254 festgelegt werden.
Formel: $if\,(g_0 < c) \Rightarrow g_1 = 0$

Threshold high Jeder Bildpunkt, dessen Grauwert einen definierten Grenzwert überschreitet, wird auf 255 (=Weiß) gesetzt (oberer Schwellwert), diese Werte liegen ebenfalls im Bereich von 0...254.
Formel: $if\,(g_0 > c) \Rightarrow g_1 = 255$

Color Inverting Jeder Bildpunkt wird im Farbbereich invertiert.
Formel: $g_1 = 255 - g_0$

Es bedeuten:

$g_0 \ldots$ Grauwert des ursprünglichen Bildpunktes

$g_1 \ldots$ Grauwert des Ergebnis-Bildpunktes

$c \ldots$ Konstanter Wert, der über die „Schieberegler” der Software eingestellt wird

Zusätzlich gilt für jede Bildpunktoperation, dass bei Unterschreiten des Minimalwertes der Grauwert mit 0 und beim Überschreiten des Maximalwertes der Grauwert mit 255 begrenzt wird.

Mit den so „vorbereiteten” Eingabebildern kann dann ein „Differenzbild” errechnet und dargestellt werden, wobei unterschiedliche Algorithmen zur Berechnung eines Ergebnispunktes gewählt werden können. Das errechnete Ergebnisbild kann wiederum als Bitmapdatei gespeichert werden.

Als grundlegender Mechanismus bei der Berechnung wird pro Pixel der Differenzwert der Graukomponenten der beiden „vorbearbeiteten” Eingabedateien ermittelt und das Ergebnis stellt den neuen „Grauwert” dar. Nachdem diese Differenz auch negativ werden kann, wird ein negatives Ergebnis als Differenz vom Maximalwert und dem ermittelten Differenz-Grauwert umgesetzt (siehe auch die Operation „Color Inverting” bei den „Bildmanipulations-Operationen” oben), dies führt bei einem Grauwert-Nulldurchgang (z.B. von +1. . . -1) zu einem abrupten Helligkeitsübergang. Der Differenzwert 0 (d.h. die Grauwerte sind identisch) wird in den verschiedenen Modi unterschiedlich behandelt.

> Bemerkung:
> Das bedeutet aber auch, dass beim Vertauschen der beiden Eingabebilder ein anderes Ergebnis erzielt wird, da die Subtraktion als mathematische Operation nicht kommutativ ist.

Zur Visualisierung des „Differenz-Bildes” können mehrere Darstellungsmodi verwendet werden.

Monochrom-Normal Modus Diese „normale” Berechnung ermittelt den Differenzwert über den vorhin beschriebenen Mechanismus. Der Differenzwert 0 (beide Eingabe-Bildpunkt haben den selben Grauwert) wird immer als weiß (RGB 0xffffff) dargestellt.

Monochrom-Invers Modus Alle Punkte werden nach derselben Methode errechnet und anschließend invertiert ($k = 255 - k$). Der Differenzwert 0 bleibt weiß dargestellt.

Farb-Normal Modus Dieser Modus errechnet die Grauwert-Differenz und setzt die Differenz in Farbwerte um. Negative Differenzen werden gelb, positive Differenzen blau, Null-Differenzwerte werden schwarz dargestellt. Damit ist es möglich, die 3 Zustände, positive, negative Differenzen und identische Grauwerte darzustellen.

Farb-Invers Modus In diesem Modus erfolgt die Berechnung analog zum Normal-Modus, negative Differenzen werden blau, positive gelb und Nullwerte weiß dargestellt, wobei die Farbkomponenten (abgesehen von der 0-Differenz) für die Farbwertdefinition invertiert werden. Auch hier sind die 3 Zustände (Punkt 1 ist heller, Punkt 2 ist heller oder beide Bildpunkte sind gleich hell) unterscheidbar.

Der „Simple Image Processor” wurde ebenfalls in der Programmiersprache Tcl/Tk (siehe 7.4.2.4) geschrieben und verwendet die freie Tcl/Tk Widget-Bibliothek BWidgets[13].

7.5. Software/Code Organisation

Dass der Assembler-Code einigen Umfang annehmen wird (ca. 250 Kb dokumentierten Assemblercode) wurde schon sehr früh erkannt, daher wurden auch die Programmteile bald in einzelne Dateien und Verzeichnisse aufgeteilt. Fixe Regeln beim Verwenden dieser Strukturen, festgelegte Namenskonventionen, die strukturierte Verwendung der Assemblerdateien, die extensive Verwendung von Assembler-Makros und die definierte Verwendung der zahlreichen Register der ATMEL Prozessorfamilie (32 meist gleichberechtigte

Universalregister) sorgen dafür, dass die Entwicklung in Assembler eine vergleichbare Effizienz mit Hochsprachen aufweist, jedoch ohne auf die Vorteile der Assemblerprogrammierung verzichten zu müssen.

Dieser Abschnitt widmet sich diesem Aspekt und erläutert die organisatorischen Regeln und Konventionen mit Begründung und den zu Grunde liegenden Intentionen und beschäftigt sich mit dem Bereich Debugging und Fehleranalyse.

7.5.1. Code- und Abhängigkeitsverwaltung

Zur Verwaltung der Abhängigkeiten, Unterstützung von inkrementellem Assemblieren, etc. wurde der unter Unix übliche Make-Mechanismus verwendet.

Make liest ein so genanntes Makefile ein, in dem die Abhängigkeiten der einzelnen Dateien definiert sind. Der Übersetzungsprozess von Programmen erfolgt unter Beachtung dieser Abhängigkeiten, wobei der Änderungszeitstempel jeder Ergebnisdatei (Target) mit dem Zeitstempel der abhängigen Dateien verglichen wird. Weichen die beiden Werte ab, weil eine Datei inzwischen geändert wurde, wird sie automatisch neu übersetzt.

Mit dem Make-Mechanismus lassen sich auch andere Entwicklungsschritte automatisieren, wie das „Hochladen” des kompilierten Programms in den Controller oder das „Bereinigen” nicht mehr benötigter Zwischendateien, etc.

Im Anhang D.1 ist der komplette Inhalt des Makefiles aufgelistet.

7.5.2. Modulspezifische Initialisierungen

Zahlreiche Schnittstellen, Ports, Variable, etc. müssen vor der ersten Verwendung mit geeigneten Startwerten initialisiert werden. Um auch im Initialisierungsbereich eine transparente Situation vorzufinden wurde eine Namenskonvention eingeführt, um ein einheitliches Vorgehen bei Initialisierungen sicherzustellen.

Es wurde festgelegt, dass in jedem Funktions- und Bibliotheksmodul eine Initialisierungsroutine vorkommen muss, die gleich heißt wie das Modul selber (identisch mit dem Dateinamen der Assemblerdatei ohne Dateierweiterung), dem das Postfix „**init_**” angehängt wird.

Err. 15.11.06: Präfix „init_” vorangestellt

Diese Routine wird im Hauptprogramm im Initialisierungsabschnitt als Unterprogramm aufgerufen. Ist keine spezifische Initialisierung notwendige, besteht diese Routinen nur aus dem Rücksprung in das Hauptprogramm.

Beispiele:

```
; File:    RF_Control.asm
; Title:   main program for RFac-library functions
  ...
  rcall mainmenu_init ; init menu management
  rcall rotation_init ; init rotation preferences
  rcall set_posn_init ; init manual pos setting
...
; File:    mainmenu.asm
; Title:   central menu and cursor management module
; init menu mode
; ------------------------------------------------------------------
; initialize menu structures
  ...
mainmenu_init:
  STSB  glb_menu_mode, menumode_idle
  ret
...
; File:   rotation.lib
; Title:  rotation functions
  ...
```

```
rotation_init:
  LDIWR REG_Z,0x0                      ; initial reset value
  STSWR rot_angle_preset, REG_Z        ; reset anlge preset value
  STSWR rot_contrl_minimum, REG_Z      ; minimum speed value
  STSB  var_valdirection, ValueDirNone ; default rotation is NONE
  STSB  rotation_mode, ModeSpeedNone   ; rotation mode is NONE
  STSB  rot_speed_value, 0x00          ; rotation speed is 0
  ret
...
; File    : set_posn.asm
; Title   : manual setting of predefined positions
  ...
; manual control unit initialization
; ----------------------------------------------
set_posn_init:
  ret
```

7.5.3. Source-Code Organisation

Das gesamte Projekt besteht aus der Hauptdatei **RF_Control.asm**, in die über die Assemblerdirektive „.INCLUDE" alle notwendigen Dateien eingebunden werden.

Es wurde festgelegt, dass nur in dieser Datei Include-Anweisungen enthalten sein dürfen, was zur transparenten Situation führt, dass alle verwendeten Dateien sofort identifiziert werden können, wenn man die .INCLUDE Direktiven in der Hauptdatei analysiert. In dieser Datei werden auch alle notwendigen Systeminitialisierungen durchgeführt, wie im vorigen Abschnitt bereits erläutert wurde.

Die Hauptroutine besteht - abgesehen von diversen Initialisierungsroutinen - nur aus dem Hauptmenü Event-Loop.

Auszug aus RF_Control.asm:

```
main_loop:
  rcall select_menu    ; wait for a mainmenu selection
; ------------------------------------------------------------------
  lds   temp1, glb_func_mode
  cpi   temp1, funcmode_none
  breq  main_loop      ; no specific menu entry selected...
  rcall menu_select_feedback ; visual feedback
; ------------------------------------------------------------------
; main function begin...
; ------------------------------------------------------------------
  rcall wait_key_release ; wait until all keys are released
  rcall main_function ; indirect function call (funcmode_calltab)
; ------------------------------------------------------------------
; main fuction end...
; ------------------------------------------------------------------
  STSB  glb_func_mode, funcmode_none ; reset function mode to NONE
rjmp main_loop
```

Die gesamte Verzeichnisstruktur inkl. der beabsichtigten Verwendung der Verzeichnisse in der endgültigen Version ist in der Tabelle im Anhang D.2 beschrieben.

7.5.4. Verwendung von Assemblermakros

Unterprogrammaufrufe verursachen einen relativ hohen Prozess-Overhead, da meist zahlreiche Register auf den Stack gesichert werden und beim Rücksprung wiederhergestellt werden müssen. Ein Near-Subroutine Aufruf (Assembler-Mnemonic **rcall**) benötigt 3 Prozessortakte um Programmcounter zu sichern und 4 Takte um den „alten" Zustand wiederherzustellen, des Weiteren pro zu sicherndes Register 2 Takte und dann nochmal dasselbe zur Wiederherstellung. Deshalb sind oft 10-20 Takte notwendig um ein einfaches Unterprogramm aufzurufen, das evtl. sogar weniger Takte benötigt um die geforderte Funktion auszuführen.

Ein weiteres Manko der Assemblersyntax ist, dass es nicht möglich ist, die in Hochsprachen übliche Schreibweise von Funktionsaufrufen mit Parametern zu verwenden, sondern man ist gezwungen, Parameter entweder in bestimmte Register zu laden oder in Speichervariablen abzulegen, auf die im Unterprogramm zugegriffen werden kann. Die zuletzt erwähnte Methode ist dabei weniger effizient, da für das Zwischenspeichern und Auslesen der Variablen ebenfalls wertvolle Prozessorzeit verbraucht wird.

Eine Möglichkeit, eine Hochsprachen ähnliche Syntax zu verwenden, existiert mit der Verwendung von Assembler-Makros, die als Platzhalter „Pseudoparameter" verwenden können.

> Beispiel:
> Makros für die Ausgabe beliebiger, im Codesegment definierter ASCII Text-Strings.

Makrodefinitionen:

```
; Parameter: @0 .. REG[LH]
;            @1 .. flash_address
;
; affected: none
.macro FLASH2REG
  ldi   @0H,high(@1<<1)
  ldi   @0L,low(@1<<1)      ;init indirect register-pointer
.endmacro
; -------------------------------------------------------------------
; write string function using a label
; label text MUST be terminated by a 0x0 byte
;
; Parameter: @0 .. SRAM label (must be unique!)
;
; affected: REG_Z
.macro LCDINIT_WRITE_ASTRING
  FLASH2REG REG_Z,@0
  rcall LCDINIT_SEND_ASTRING
.endmacro
```

Makroverwendung:

```
COPYRIGHT_AUTHOR:  .db "G. Reithofer",$0,$0
;
; send string to LCD
LCDINIT_WRITE_ASTRING COPYRIGHT_AUTHOR
```

Der damit erzeugte Makrocode wird vom Assembler in syntaktisch korrekten Assemblercode umgesetzt und die Ausgabeanweisung vom obigen Beispiel wird dann in folgende Assembleranweisungen übersetzt.

```
ldi   ZH,high(COPYRIGHT_AUTHOR<<1)
ldi   ZL,low(COPYRIGHT_AUTHOR<<1)
rcall  LCDINIT_SEND_ASTRING
```

Dabei ist zu beachten, dass in dieser Situation ein Registerpaar (nämlich Z, bzw. R30 und R31) für den Aufruf verwendet wird. Ist der vorherige Inhalt des verwendeten Registers später von Belang, muss dessen Inhalt zwischengespeichert werden (z.B. auf den Stack) und nach Abarbeitung des Makros wieder hergestellt werden. Allerdings tritt u. U. wieder die vorhin erwähnte Prozessorzeitverschwendung zu Tage, wenn diese Vorgangsweise bei jedem Aufruf eines Makros durchgeführt werden würde, das „Sichern" im Einzelfall aber nicht notwendig gewesen wäre. In diesem Fall ist man bei der Verwendung von Makros flexibler als bei einem Unterprogrammaufruf, der in jedem Fall einen Aufruf-Overhead verursacht.

Einige Makros werden hauptsächlich zur Verbesserung der Lesbarkeit des Codes verwendet, obwohl sie Unterprogamm ähnliche Strukturen verwenden, wie das Sichern aller Register. Dies führt dann zu sehr kompaktem und überschaubarem Code, kann sich aber bei unkontrollierter Verwendung auch nachteilig auswirken.

Definitionen:

```
; -------------------------------------------------------------------
; set LCD cursor to specific location
; includes II2C init
; Parameter: @0 .. lcd display column number (0..11)
;            @1 .. lcd display line number (0..2)
; affected: flags
.macro LCDINIT_GOTO_XY
  push  temp1
  push  temp2
  ldi   temp1,@0
  ldi   temp2,@1
  rcall LCDINIT_GOTO_POS
  pop   temp2
  pop   temp1
.endmacro
.macro DRAW_MENU
  PUSHW Z ; used in *STRING macros...
  rcall LCDINIT_CLEAR_DISP
  FLASH2REG Z,@0
  rcall LCDINIT_SEND_ASTRING
  LCDINIT_GOTO_XY 0,1
  FLASH2REG Z,@1
  rcall LCDINIT_SEND_RSTRING
  LCDINIT_GOTO_XY 0,2
  FLASH2REG Z,@2
  rcall LCDINIT_SEND_ASTRING
  POPW  Z
.endmacro
```

Verwendung:

```
; -------------------------------------------------------------------
goto_menumode:
  DRAW_MENU MNU_FUNC_GOTO,MNU_LEVEL_CHNG,MNU_FUNC_MCTL
  ret
```

Hinweis:
Dieser Aufruf gibt ein komplettes Benutzermenü aus, das alle 3 LCD-Zeilen umfasst (siehe dazu auch Absch. 7.6.2.1).

Als Nachteil der Makroverwendung tritt ein erhöhter Speicherverbrauch auf, da jeder Makroaufruf in eigene Assembleranweisungen umgesetzt wird, wohin gegen der Code einer Subroutine nur einmal im Programmspeicher abgelegt ist.

Als Beispiel sei hier der Aufruf des oben angeführten **DRAW_MENU** Makros betrachtet, das in 30 Assembleranweisungen mündet und nicht mehr als effizienter Assemblercode bezeichnet werden kann.

```
goto_menumode:
  push  ZH
  push  ZL
  rcall LCDINIT_CLEAR_DISP
  ldi   ZH,high(MNU_FUNC_GOTO<<1)
  ldi   ZL,low(MNU_FUNC_GOTO<<1)
  rcall LCDINIT_SEND_ASTRING
  push  temp1
  push  temp2
  ldi   temp1,0
  ldi   temp2,1
  rcall LCDINIT_GOTO_POS
  pop   temp2
  pop   temp1
  ldi   ZH,high(MNU_LEVEL_CHNG<<1)
  ldi   ZL,low(MNU_LEVEL_CHNG<<1)
  rcall LCDINIT_SEND_RSTRING
  push  temp1
  push  temp2
  ldi   temp1,0
```

```
    ldi   temp2,2
    rcall LCDINIT_GOTO_POS
    pop   temp2
    pop   temp1
    ldi   ZH,high(MNU_FUNC_MCTL<<1)
    ldi   ZL,low(MNU_FUNC_MCTL<<1)
    rcall LCDINIT_SEND_ASTRING
    pop   ZL
    pop   ZH
    ret
```

Dies bedeutet, dass der Programmierer sehr konsequent bei der Verwendung von Makros (aber auch bei Unterprogrammaufrufen) vorgehen muss, da sonst leicht Fehler durch „Seiteneffekte" auftreten können oder der Programmspeicher nicht optimal genutzt wird. Es ist auf jeden Fall ratsam, alle Register, Variablen, ... die sich in einem Unterprogramm oder Makro verändern können, konsequent in der entsprechenden Routine bzw. Makrodefinition zu dokumentieren.

Da der ATMEL Prozessor ATmega8 grundsätzlich ein 8-Bit Prozessor ist, für das Projekt aber sehr häufig 16-Bit Operationen benötigt werden, wurde eine 16-Bit Arithmetikbibliothek in Form einer Makrosammlung implementiert.

Um die Makros leicht verständlich zu halten wurden sie namentlich an die Mnemonik der 8-Bit Arithmetikbefehle angelehnt. Die Makros erhielten den selben Namen wie die entsprechenden 8-Bit Operationen, jedoch mit dem Postfix „**WR**" (Abkürzung für **W**ord **R**egister). Auf das Thema Namenskonventionen wird später in Absch. 7.5.6 noch einmal detailliert eingegangen.

Beispiele:

```
ADD R1, R2         ; 8-bit addition
ADDWR REG_A, REG_B ; 16-bit addition
CPI R16, 0x0f      ; immediate compare with 8-bit register
CPIWR Z, 0x00ff    ; immediate compare with 16-bit register
```

Die 16-Bit Makros erfordern eine bestimmte Namenskonvention in der Bezeichnung der Register, nämlich die Postfix-Zeichen H und L für den jeweiligen High-Byte und Low-Byte Anteil der 16-Bit Werte, auch darauf wird im Abschnitt 7.5.5 noch einmal eingegangen.

Alle produktiven Makrodefinitionen sind in der Datei **inc/macros.inc** definiert.

7.5.5. Bestimmungsgemäße Verwendung der Register

Ein weiterer Ansatz, das Problem des unbeabsichtigten Überschreibens bestehender Registerinhalte (Seiteneffekte) in den Griff zu bekommen, ohne die Register ständig zu sichern und wieder herstellen zu müssen, ist das Festlegen definierter Register auf bestimmte Verwendungen.

In diesem Projekt wurden alle 32 Register mit symbolischen Namen versehen, um damit eine bestimmte Verwendung zu bewirken und zu dokumentieren, weiters wurden mehrere Register mit den vorhin beschriebenen High/Low Prefixen versehen, um sie als 16-Bit Operanden für die 16-Bit Makrobibliothek verwenden zu können.

Beispiele:

```
.def sched_temp      = r12 ; sheduler temporary usage
.def reserved_reg    = r13 ; reserved (PUSHF, POPF macros)
;...
; ------------------------------------------------------------
; upper temporary immeadiate registers (usually not saved)
; should be used only inside of functions or 'visible' area
.def temp1           = r16 ; usuallay subroutine call input
.def temp2           = r17 ; usuallay subroutine call input
; ------------------------------------------------------------
;;; common used return values
```

```
.def return1          = r18 ; reserved return value for calls
.def return2          = r19 ; reserved return value for calls
; -------------------------------------------------------
;;; i/o definitions
.def iic_receivebyte  = r20 ; i2c receive buffer
.def iic_sendbyte     = r21 ; i2c send buffer
; -------------------------------------------------------
;;; registers for word arithmetics
.def REG_AL           = r22 ; operand low  byte for 16-bit operations
.def REG_AH           = r23 ; operand high byte for 16-bit operations
```

Mit Konventionen dieser Art wird die Verwendung der Register vergleichbar mit der Verwendung von Speicher-Variablen, jedoch werden die Operationen wesentlich schneller ausgeführt (auf Grund der ATmel RISC-Prozessorarchitektur in einem Prozessortakt) da keine Speicheradressierungen und Transfers benötigt werden.

Alle Register-Definitionen dieses Projektes sind in der Datei **inc/reg_defs.inc** definiert.

7.5.6. Namenskonventionen und aussagekräftige Identifier

Wie schon in den vorhergehenden Unterabschnitten dieses Bereiches mehrfach erwähnt, stellen Konventionen eine gute Methode dar, um Fehlerhäufigkeit oder Verwechslungen beim Programmierer oder Programmnutzer zu vermindern. Dies wurde auch in der Namensvergabe von Unterprogrammen und Makros konsequent eingesetzt. Als Beispiel seien hier die LCD-Ausgaberoutinen angeführt, analoge Mechanismen wurden auch in anderen Bereichen verwendet.

Die Ausgabe von Informationen auf das LCD-Display ist, bezogen auf die Prozessorzeit, eine sehr aufwändige Operation. Es ist somit sinnvoll, mehrere Zeichen in einer I^2C-Ausgaberoutine auszugeben, ohne den Bus neu initialisieren zu müssen. Es gibt jedoch auch zahlreiche Situationen, in denen man um eine Bus-Initialisierung nicht herumkommt.

Daher sind diese „High-Level" Ausgabefunktionen alle zwei mal vorhanden,

- einmal als reine I^2C-Ausgabe ohne Businitialisierung und
- einmal mit vorheriger Bus-Initialisierung und nachträglicher Busfreigabe

Beide Aufrufarten sind identisch, mit der Ausnahme, dass die Funktionen der 1. Art mit „LCD_" beginnen und in der 2. Version „LCD_INIT_" vorangestellt bekommen.

```
; ==================================================================
; display Z-value as floating point number - without LCD_INIT
; float format: NNN.F - in current implementation the fractional
; float part F can only be 0 or 5 (0.5 degrees),
; -------------------------------------------------------------------
; input: REG_Z (word) in 0.5 degrees (=value*2)
; output: none
; affected: iic_sendbyte
;
LCD_SEND_CFLOAT:
  push  temp1
  push  ltmp1
  PUSHW REG_Z
... assembler code ...
; do clean up ...
  POPW  REG_Z
  pop   ltmp1
  pop   temp1
  ret
; -------------------------------------------------------------------
; display Z-value as floating point number - with LCD_INIT
; ...
LCDINIT_SEND_CFLOAT:
```

```
    rcall LCD_START                 ; start bus transfer
    rcall LCD_SEND_CFLOAT           ; send float to LCD
    rcall IIC_STOP                  ; stop bus transfer
    ret
```

Weitere häufig verwendete Konventionen bestehen darin, dass in zahlreichen Subroutines ein Label mit dem Namen des Unterprogramms aufscheint, dem die Nachsilbe „**_ret**" nachgestellt ist. Dies ist jenes Label, nach dem nur mehr Stack-Wiederherstellungen ausgeführt werden und das somit jederzeit gefahrlos zum Verlassen des Unterprogramms angesprungen werden kann, bzw. soll.

```
; --------------------------------------------------------------------
; user selectable preset rotation direction
; -------------------------------------------------------------------
; input:  none
; output: temp1 - rotation step counter increment (-1,0,+1)
; affected: flags
; -------------------------------------------------------------------
user_select_direction:
  PUSHW REG_Z
... assembler code ...
dir_none:
  ldi   temp1, ValueDirNone ; no rotation?
  rjmp  user_select_direction_ret
dir_left:
  ldi   temp1, ValueDirDecr ; reverse direction
  rjmp  user_select_direction_ret
dir_right:
  ldi   temp1, ValueDirIncr ; normal direction
; rjmp  user_select_direction_ret
user_select_direction_ret:
  POPW  REG_Z
  tst   temp1
  ret
```

Weiters enden alle Hauptfunktionen der obersten Menüebene (Module) mit der Nachsilbe „**_main**" und können daher leicht als Einsprungspunkte der Hauptfunktionen identifiziert werden.

Die häufige Verwendung der 16-Bit Makros führt zwangsweise auch dazu, dass zwei 8-Bit Register gemeinsam auf den Stack gesichert und vor allem in der richtigen Reihenfolge wiederhergestellt werden müssen.

Dies erledigen die beiden Makros **PUSHW** und **POPW**, die sich immer auf einen High- und Low-Registeranteil beziehen und somit auf alle Register angewendet werden können, die das H- und L-Postfix verwenden.

Diese „Typkennzeichnung" wird aber auch bei 8-Bit Makros verwendet, so gibt das Makro **OUTB** ein Byte in ein 8-Bit Port unter Verwendung eines Hilfsregisters aus.

```
OUTB TIMSK,(1<<OCIE1A)|(1<<TOIE0)
```

7.5.7. Fehleranalyse/Debugging

Ein grundsätzliches Problem bei Hardware naher Softwareentwicklung stellt das Debugging und die Programmkontrolle dar.

Es existieren auch für die ATMEL Prozessoren Software-Emulatoren, die Prozessoren und Register, sowie Ein-/Ausgebeports nachbilden können. Diese Werkzeuge stellen eine gute Möglichkeit zum Kennen lernen der Mikrocontrollerhardware oder für Unterrichtszwecke dar, doch eine Reaktion von Hardwarebausteinen, wie z.B. ein Zählsignal am Timer/Counter1 Eingang, ist mit Emulatoren nicht oder nur sehr umständlich abbildbar.

Dennoch kann es oft notwendig sein, dass in der Entwickungsphase an beliebiger Stelle des Programmablaufs eine Systemrückmeldung zur Programmkontrolle erforderlich ist, die von einem kurzen Aufleuchten einer Leuchtdiode bis zur Klartext-Fehlermeldung reichen kann. Diese Systemreaktionen müssen zudem noch einfach ein- und ausschaltbar sein.

Eine kleine Bibliothek von Makros und Unterprogrammen wurde geschaffen, die über eine Assemblerdirektive während der Entwicklungsphase zu Testzwecken bedingt in die Hauptdatei **RF_Control.asm** eingebunden werden kann.

```
; ------------------------------------------------------------------
; debug functions
; ------------------------------------------------------------------
.ifdef DEBUG
.include "debug_fun.inc"   ; debug functions (conditional include)
.endif
```

Um debug-kritische Situationen austesten zu können, wurden in einer weiteren Include-Datei (**testfunc.asm**) entsprechende Testfunktionen angelegt, die Initialisierungen vornahmen und Unterprogramme zu Testzwecken ausführten.

> Die beiden Hilfsdateien befinden sich laut Anhang D.2 im Hauptverzeichnis der Entwicklungsumgebung (siehe auch Absch. 7.5.3).

Makros zur Ausgabe des Inhaltes eines Registers oder Registerpaares wurden erstellt, die keine Seiteneffekte aufweisen und an jeder Stelle des Programms aufgerufen werden können, ohne den Programmablauf (abgesehen vom Status der LCD-Anzeige, in der die Werte ausgegeben werden) sonst irgendwie zu beeinflussen.

```
; ------------------------------------------------------------------
; display a character as prefix and the
; hexadecimal value of a register
;
; affected: none (not even the flags)
; Parameter: @0 .. prefix char
;            @1 .. register
.macro DEBUG_REGISTER
  PUSHF                  ; save flags
  push  iic_sendbyte
  push  @1               ; put register value onto stack
  ldi   iic_sendbyte,@0  ; write out ascii character
  rcall LCDINIT_SEND_ACHAR
  pop   iic_sendbyte
  rcall LCDINIT_SEND_HEX ; write out hex value
  pop   iic_sendbyte
  POPF                   ; restore flags
.endmacro
```

Verwendung:

```
DEBUG_REGISTER 'R',return2
```

Ein Beispiel das den Einsatz demonstriert.

In Abschnitt 7.8.1.3, der sich mit der 5-Phasensteuerung beschäftigt, wird darauf hingewiesen, dass der in der Phase I (S. 73) gemessene „Losfahrwert”, als Zielwert für die Phase III (S. 74) angesteuert wird. Außerdem stellte dieser Rotations-Startwert einen Anhaltspunkt für die Festlegung eines sinnvollen Steuerungs-Startwertes für die Phase I dar. Ein Startwert von 0 würde nur unnötige Wartezeit bewirken, da die Drehbewegung erst bei einem wesentlich höheren Startwert (ca. 130) einsetzt.

Um diese konkreten Werte zu ermitteln, wurde in einer früheren Code-Version das „DEBUG_REGISTER” Makro eingesetzt.

```
phase_speed_down:
  push  temp1
  push  temp2
  push  ltmp2
  STSB  rotation_mode, ModeSpeedDown ; rotation mode is go down
```

```
  lds   temp1, rot_speed_value       ; get current rotation speed value
  lds   ltmp2, rot_contrl_minimum    ; get value of rotation startup value
LCDINIT_ERASE_LINE 0
DEBUG_REGISTER 's',temp1
DEBUG_REGISTER 'm',ltmp2
BLINK_RED_LED
decrease_speed:                      ; decreasing speed until
  in    temp2, TIMSK                 ; read TIMSK port
  sbrc  temp2, OCIE1B                ; is bit OCIE1B set?
  rjmp  speed_slowdown_fast          ; if yes, back to loop
```

Die Display-Zeile 0 ist üblicherweise mit einem fixen Text versehen und „verdeckt” durch die Debug-Ausgabe keine betriebsrelevanten Informationen, das Makro „LCDINIT_ERASE_LINE” sichert ebenfalls alle Register und verändert damit keine Register, Flags sind an dieser Stelle nicht relevant. Dies führt zu einer Anzeige, wie sie in Abb. 7.5 beispielhaft dargestellt ist.

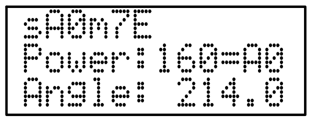


Abbildung 7.5.: LCD-Anzeige mit Debug-Ausgabe

Wie werden diese Werte nun interpretiert?

- Zu diesem Zeitpunkt (Start der Steuerungspahse III) betrug die „Power” den Wert 0xA0 („s” wie Speed), die Steuerung ging somit vor Erreichen des Maximalwertes in die „Geschwindigkeits-Reduzierphase” über (siehe Abb. 7.14 auf Seite 70).
- Des Weiteren betrug der „Losfahrwert” (Variable **rot_control_minimum**) den Wert 0x7E (Dez. 126, „m” wie Minimum). Dies ist zugleich der Zielwert der „Geschwindigkeits-Reduzierphase”, der in der Phase I ermittelt wurde (siehe dazu auch 7.8.1.3).

Eine Wartezeit von einer Sekunde mit dem Aufblinken der roten LED informiert den Anwender mit dem Makro „BLINK_RED_LED” über diese Debug-Phase.

Sollten kompliziertere Zustände untersucht werden, die umfangreicheren Code benötigen, wurden diese Routinen als Unterprogramme ebenfalls in die Debug-Datei mit aufgenommen und in Abhängigkeit von der DEBUG-Direktive bedingt über ein Makro aufgerufen.

```
; -------------------------------------------------------------------
; conditional DEBUG call
; this is executed if DEBUG directive is set...
; the subroutine should be between .ifdef DEBUG and .endif
; to avoid assemblation of the debug code if DEBUG
; directive is not set
; -------------------------------------------------------------------
; Parameter: @0 .. debug subroutine
.macro CALL_DEBUG
.ifdef DEBUG
rcall @0
.endif
.endmacro
```

Verwendung:

```
CALL_DEBUG show_preproc_state
CALL_DEBUG debug_rotation_mode
CALL_DEBUG debug_show_angle
```

Diese Aufrufe verzweigen in Unterprogramme, die umfangreichere Statusinformationen ausgeben, wenn die DEBUG-Direktive gesetzt ist, wenn nicht, werden die Anweisungen ignoriert.

Als Beispiel wird die Funktion **show_preproc_state** betrachtet, die in Version 0.8 der Steuerungssoftware noch vorhanden war und am Ende der Routine **rotation_preproc** aufgerufen wurde.

```
show_prepoc_state:
  push  temp1
  PUSHW REG_Z
  PUSHW REG_B
  LCDINIT_ERASE_LINE 0
  INWR  REG_B, TCNT1           ; get current value currpos(REG_B)
  LCDINIT_WRITE_HEXWORD REG_B
  LCDINIT_WRITE_CHARACTER '-'
  LCDINIT_WRITE_HEXWORD REG_A
  LCDINIT_ERASE_LINE 1
  LCDINIT_WRITE_CHARACTER '='
  LCDINIT_WRITE_SRAM rot_angle_godown+1
  LCDINIT_WRITE_SRAM rot_angle_godown
  LCDINIT_ERASE_LINE 2
  in    iic_sendbyte, TIMSK
  rcall LCDINIT_SEND_BINARY
  rcall wait_key_confirm
  POPW  REG_B
  POPW  REG_Z
  pop   temp1
  ret
```

Bemerkungen:
Das Unterprogramm **rotation_preproc** ermittelt unter anderem den Rotations-Endwinkel (REG_A), in Registerpaar REG_B wurde der aktuelle Winkel (Inhalt des Zählers TCNT1) eingelesen.
Das Makro „LCDINIT_WRITE_SRAM” gibt den Inhalt einer Speicheradresse am LCD aus, im vorliegenden Fall einen 16-Bit Wert im LSB-Format, der in der Speicherstelle (Variablen) **rot_angle_godown** und der Folgeadresse abgelegt ist.

Das Unterprogramm zeigt dabei folgende Daten an (Abb. 7.6):

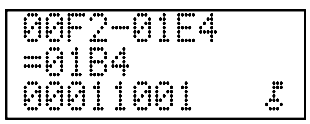


Abbildung 7.6.: LCD-Anzeige mit Debug Inhalt

1. In Zeile 1 wird der aktuelle Rotationwinkel (0x0F2=121°, Inhalt des Zählers TCNT1) und der vorberechnete Endwinkel (0x1E4=242°) angezeigt.
2. Der vorberechnete „SpeedDown” Wert (0x1B4=218°), der Beginn der Steuerungsphase III, wird nach dem „=” Zeichen dargestellt (genaueres dazu in Absch. 7.8.1.2).
3. Die binäre Darstellung des Inhaltes des Timer-Interrupt Mask Registers TIMSK wird in Zeile 3 ausgegeben. Das „Schlüsselsymbol” rechts unten erscheint im DEBUG-Modus, wenn auf einen Tastendruck zum Fortfahren des Programms gewartet wird.

7.6. Applikationsschicht/User-Interface

Das User-Interface soll einfach und intuitiv gehalten sein. Eine konsistente Menüstruktur und die entsprechenden Meldungen am 3-zeiligen Display (User-Feedback) müssen alle notwendigen Informationen enthalten um die Anlage zu bedienen, Fehlersituationen zu erkennen und alle notwendige Betriebszustände ablesen zu können.

Dieser Abschnitt erläutert ausführlich die Aspekte der Benutzerführung und die Interaktion zwischen Mensch und Maschine. Menüstrukturen, Benutzerführung und semantische Festlegungen sorgen für eine effiziente und fehlerarme Bedienung der Anlage und sollen zu einer hohen Anwenderakzeptanz führen.

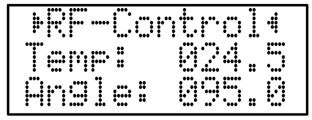


Abbildung 7.7.: Controller Hauptbildschirm (Idle Mode)

7.6.1. Menüstruktur

Wie bereits in Absch. 7.3 erwähnt, besteht das „Betriebssystem" aus dem Hauptbildschirm (Abb. 7.7) und den Hauptmenüpunkten (Funktionsmodi). Err.: 4.10.06 Abb. ersetzt

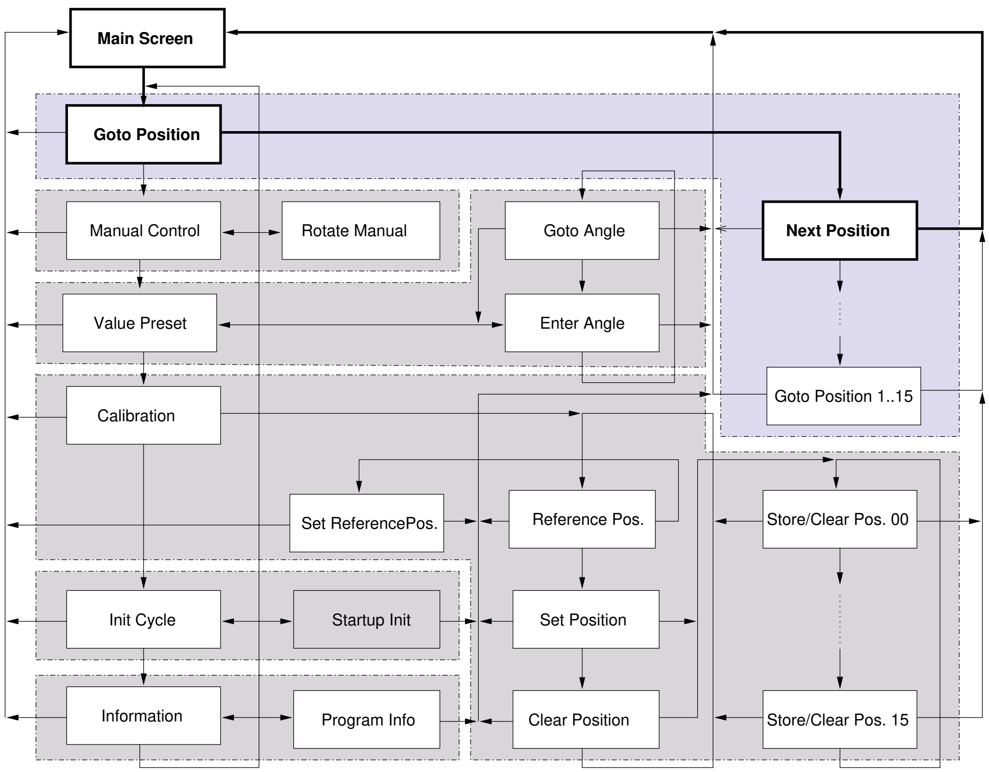


Abbildung 7.8.: Userinterface und Menüstruktur

Die gesamte Menüstruktur ist in der Abbildung 7.8 dargestellt, alle zu einem Funktionsmodus gehörenden Menüpunkte sind in dieser Abbildung in zusammen hängenden grafischen Bereichen dargestellt. Im Anhang F auf Seite 110 existiert eine ausführlichere Grafik mit Kontext abhängigen LCD-Darstellungen.

Diese oberste Menüebene (Hauptmenü) dient nur der Auswahl eines der 6 Arbeitsmodi, die in Abb. 7.8 ganz links dargestellt sind, oder dem Verlassen des Hauptmenüs zurück in den Hauptbildschirm (in Abb. 7.8 ganz oben).

7.6.2. LCD-Anzeigeninhalte und deren Bedeutungen

Um eine sichere Anlagenkontrolle zu gewährleisten, wurde eine einfache, in allen Menüauswahl-Situationen gleich dargestellte und bedienbare Bildschirmstruktur definiert. Drei spezielle „Aktionen" werden in Form von „Cursor"-Pfeilen dargestellt und mit den Namen „*Next*" (Pfeil nach unten), „*Back*" (Pfeil links) und

„*Enter*” (Pfeil rechts) bezeichnet sind. Diese drei „Eingabeobjekte” werden eindeutig den Funktionstasten zugeordnet (siehe 7.6.3) und dienen somit in jeder Auswahlsituation als optische Hilfestellung zu den möglichen Auswahlsituationen.

Die Ausgabe besteht aus der 3-zeiligen LCD-Anzeige mit einer Zeilenlänge von 12 Zeichen und wird in der gesamten Programmlogik immer in einem der drei definiertenen Kontexte verwendet.

1. Werte-, Menüauswahl oder Bestätigung
2. Eingabe oder Änderung eines nummerischen Wertes
3. System-Feedback oder Informations-Anzeige

Während Punkt 3 in allen Situationen verwendet wird, in denen keine Benutzereingabe vorgesehen ist, es handelt sich also um eine reine „Anlagen-Rückmeldung”, erfordern die Situationen 1 und 2 eine konkrete Aktion vom Anwender. Er muss entweder aus einer diskreten Liste von Werten auswählen (z.B. einen Betriebsmodus auswählen oder eine Aktion bestätigen bzw. verweigern) oder er kann einen bestimmten Wert über die Tasten „einstellen”.

7.6.2.1. Menü- oder Werteauswahl-Modus

Eine Menüauswahl präsentiert sich dem Anwender ausnahmslos in folgender Darstellung, wobei jede Displayzeile eine genau definierte Information darstellt:

1. Die oberste Zeile zeigt immer den aktuellen Modus oder den entsprechend bezeichneten aktuellen Menüpunkt an.
2. Die mittlere Zeile stellt immer die folgenden Menüauswahl-Funktionen dar:
   a) Verlassen des aktuellen Menüs (mit dem Text „*Back*” gekennzeichnet und den links zeigenden „Cursor”)
   b) die Anwahl des aktuellen Menüpunktes, also die Bestätigung des aktuellen Menüs (mit dem Text „*Enter*” und einem „Cursor” rechts)
   c) und, wenn es in der Menüstruktur vorgesehen ist, den nach unten zeigenden „*Next*”-Cursor mit dem Folgemenü oder der Beschreibung der nächsten Aktion
3. In der untersten Zeile erscheint die Bezeichnung des Folgemenüs oder des nächsten zu wählenden Wertes, wenn der „*Next*”-Cursor sichtbar ist. Ist dies nicht der Fall, kann nur der aktuelle Wert mit „*Enter*” bestätigt oder mit dem „*Back*” Kommando die Auswahl wieder verlassen werden.

Abbildung 7.9 zeigt einige „typische” Situationen an, in denen notwendige Eingaben vom Anwender erfolgen müssen (v.l.n.r.).

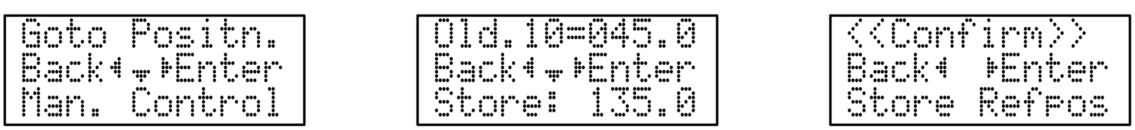


Abbildung 7.9.: Beispiele für Werte-/Menüauswahl-Dialoge

1. Ein Hauptmenüpunkt mit Folgemenü. Der oben dargestellt Funktionsmodus (Goto Position Mode) kann mit „*Enter*” bestätigt und somit aktiviert werden, mit „*Next*” kann zum nächsten Menüpunkt (Manual Control Mode) weitergeschaltet werden und mit „*Back*” wird das Hauptmenü wieder verlassen und der „Idle-Screen” (Abb. 7.7) wieder aktiviert.

2. Funktionsmenü mit Werteauswahl. Der „alte” Wert, der mit dieser Funktion überschrieben werden kann, wird in der obersten Zeile angezeigt, der „aktuelle” Wert - dies ist jener Wert, der mit dieser Funktion gespeichert werden kann - wird in der unteren Zeile dargestellt und kann mit „*Enter*” gespeichert werden. Mit „*Next*” wird auf den nächstmöglichen Wert (Inhalt der Speicherzelle) „weitergeschaltet”, dessen Inhalt wird dann wieder in der oberen Displayzeile angezeigt. Mit „*Back*” wird dieses Untermenü verlassen.

3. Ein Bestätigungsmenü. Die unten dargestellte Aktion (Speichern der aktuellen Position als „Referenz-Position”) kann über „*Enter*” bestätigt, d. h. ausgeführt werden, oder der Modus kann über „*Back*” wieder verlassen werden. Die „*Next*” Taste hat hier keine Funktion und wird ignoriert.

7.6.2.2. Werteingabe und Wertänderung-Modus

Falls gefordert ist, einen bestimmten nummerischen Wert einzugeben, werden die Cursortasten zur „Dekrementier-” (*Decrease*), bzw. „Inkrementier”-Tasten (*Increase*), die mittlere Taste wird zur „*Exit*”-Taste umfunktioniert, mit der die Wertedefinition beendet werden kann.

Im Gegensatz zur „linearen” Menüauswahl (d.h. für jedem Menüschritt ist genau ein Tastendruck zugeordnet), reagiert die Werteingabe „exponentiell”, in dem sich die Geschwindigkeit mit der Zeitdauer, in der eine Taste niedergehalten wird, pro Zeiteinheit erhöht.

Als Beispiel für eine Anzeige einer Werteänderung wird die „Power”-Kontrolle (Abb. 7.10) der manuellen Steuerung (siehe Absch. 7.8.2) angeführt.

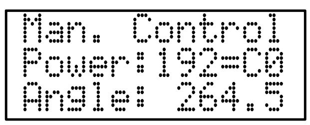


Abbildung 7.10.: Werteänderungs-Dialog

In diesem Dialog erscheint ebenfalls in der Titelzeile (oberste LCD-Zeile) der Betriebmodus, in der mittleren Zeile wird der aktuell einzustellende Wert dargestellt, die untere Zeile ist die Statuszeile für den aktuellen Winkel, sie zeigt also die tatsächliche Auswirkung des eingestellten Power-Wertes an.

Der einzustellende Wert erhöht sich pro Tastendruck auf die „*Increase*”-Taste, bzw. vermindert sich bei Betätigung der „*Decrease*”-Taste. Wird eine dieser Tasten längere Zeit niedergehalten, wird im Auto-Repeat Modus der Power-Wert ca. 2 mal pro Sekunde erhöht oder vermindert, wobei sich die Geschwindigkeit der Änderung mit der Tasten-Niederhaltezeit erhöht.

Der Werteingabemodus wird mit der „*Exit*”-Taste beendet.

Um die nummerische Wertedefinition beispielhaft darzustellen, wird die Winkelvorgabe (Abb. 7.11) für den „Angle Preset Mode” (siehe Absch. 7.8.3) konkret betrachtet.

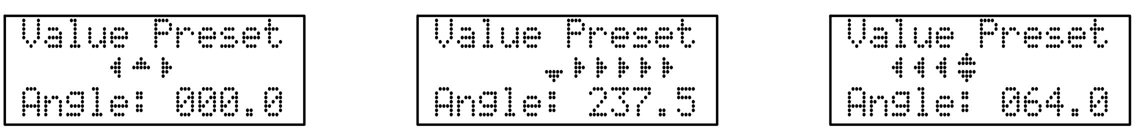


Abbildung 7.11.: Wertedefinitions-Dialog

In dieser Werteeinstellfunktion wurde ein spezieller „Speed-Cursor” implementiert, der in der mittleren Anzeigenzeile Information über folgende Situationen liefert:

1. Status der Geschwindigkeitskontrolle - Werteänderung findet statt oder nicht

   a) Die beiden äußeren Pfeile zeigen in beide Richtungen - keine Änderung

b) ein oder mehrere Pfeile in einer Richtung zeigen die Zunahme oder Abnahme an

2. Mögliche Geschwindigkeitsänderung in der aktuellen Situation
   a) Mittlerer Pfeil zeigt nach oben - nur Zunahme möglich, d. h. Bewegungsstart
   b) Pfeil zeigt nach unten - nur Abnahme möglich, d. h. Maximalwert erreicht
   c) In der Mitte zeigt der Doppelpfeil nach unten und nach oben - Zunahme und Abnahme des Wertes möglich
3. Richtung der Änderung
   a) Zunahme des Wertes - ein oder mehrere Pfeile zeigen nach rechts
   b) Abnahme des Wertes - ein oder mehrere Pfeile zeigen nach links
4. Änderungsgeschwindigkeit in den Stufen 1-5 (dargestellt durch 1...5 Pfeilspitzen in Richtung der Geschwindigkeitsänderung), wobei sich die Änderung der Geschwindigkeit pro Stufe nach folgender Formel errechnet:

$$\Delta v = 2^f$$

Δv ... Änderungsgeschwindigkeit pro Zeiteinheit
f ... Änderungsfaktor (1...5)

Daraus resultiert eine maximale Geschwindigkeitsänderung pro Zeiteinheit.

$$\Delta v_{max} = 2^5 = 32$$

7.6.3. Tastatursemantik und Benutzereingaben

Die oben erwähnten drei logischen Funktionen *Enter*, *Back* und *Next* (1) bzw. *Increase*, *Decrease* und *Exit* (2) werden gemäß Abb. 7.12 ihren geometrischen Positionen entsprechend den drei Funktionstasten des Mikrocontrollers zugeordnet.

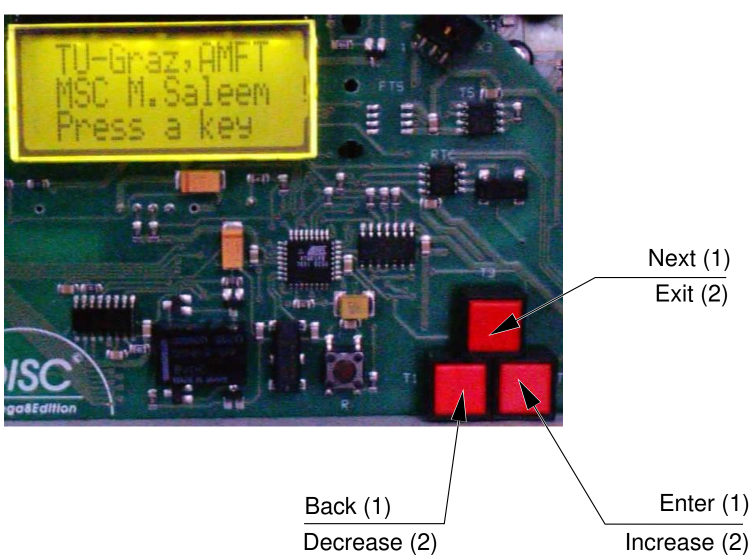


Abbildung 7.12.: Mikrocontroller Tastatursemantik

Die komplette Liste der Funktionen der Drehwinkelsteuerung ist in [14] nachzulesen.

7.7. Ein-/Ausgabemodule

Dieser Abschnitt beschäftigt sich mit allen Informationen, die in irgend einer Weise an den Mikrocontroller „übergeben" oder von diesem nach außen „abgeliefert" werden.

Das in Abb. 6.7 auf Seite 36 dargestellte detaillierte Hardware-Modulkonzept zeigt alle notwendigen Ein- und Ausgabeeinheiten auf, die in diesem Abschnitt einzeln behandelt werden, darin enthalten ist auch die Ansteuerung des Gleichstrommotors und die Permanentspeicherung der letzten Winkelposition und aller vordefinierten Winkellagen.

Err.: 4.10.06 Abschn. 6.7

Die unterste Schicht für die notwendigen Benutzerinteraktionen und zur Steuerung der Anlage stellt den grundsätzlichen Hardware-Bezug her. Diese Module sind speziell an die bestehende Hardware angepasst und dienen der grundsätzlichen Kommunikation zwischen den einzelnen Hardwarekomponenten und letztendlich auch zur Kommunikation zwischen der Steuerung und dem Bediener der Anlage.

Software technisch wird die Ein-/Ausgabe wiederum in zwei Ebenen unterteilt, einerseits die Lowlevel-Routinen, die direkt mit den Hardwarekomponenten kommunizieren, und andererseits die Highlevel Schicht, die allgemeinere Funktionen (z. B. die Ausgabe eines nummerischen Wertes im Hexadezimalformat) zur Verfügung stellen und somit auch nach Austauschen der Hardwarekomponenten beibehalten werden können. In Absch. 7.3 auf Seite 39 ist das „Schichtenmodell" detailliert behandelt.

7.7.1. Ausgabeeinheiten

Der Großteil der Ausgabe geschieht - aus Sicht des Mikrocontrollers - über die Onboard-I^2C-Schnittstelle. Dies betrifft die LCD-Matrix Anzeige genau so wie den I^2C-gesteuerten D/A-Wandler PCF-8591, nur einige wenige Komponenten werden direkt über Ausgabeports angesprochen.

7.7.1.1. Port gesteuerte Ausgabe

Als Port-gesteuerter Ausgang wird die Richtungs-Steuerung über das in die **halvedDISC** integrierte Relais umgesetzt, ebenso werden die Hintergrundbeleuchtung des LCD-Displays und die rote Leuchtdiode, die als User-Feedback eingesetzt wird, über ein Ausgabeport aus- bzw. eingeschaltet. Das Onboard-Relais betätigt dabei das Leistungsrelais, das sich auf der I^2C-Schnittstellenkarte befindet, das direkt den Ankerstrom des Elektromotors umpolt, siehe dazu auch Absch. 6.3. Als weitere direkte I/O ist das Lesen und Schreiben des EEPROM-Bereiches notwendig, darauf wird in den folgenden Abschnitten noch näher eingegangen.

7.7.1.2. Motorsteuerung

Wie in Abb. 6.7 auf Seite 36 ersichtlich, wird der Elektromotor über einen Regler gesteuert. Dieser MOSFET PWM-Regler (siehe 5.3), benötigt eine Leitspannung im Bereich von 0. . . 10 V.

Die Steuerspannung wird in dieser Anlage von einem D/A-Wandler generiert, der über den I^2C- Bus kontrolliert wird. Da der D/A-Wandler PCF8591 als TTL-Baustein jedoch nur eine Ausgangsspannung im Bereich von ca. 0 V. . . 5 V erzeugen kann, wird diese Steuerspannung von einem nachgeschalteten OP in den geforderten Nutzbereich transformiert.

Der Code zur Ansteuerung des PCF8591 ist im Großen und Ganzen identisch mit den Low-Level-Routinen zur Ansteuerung des LCD-Displays, da es sich auch „nur" um eine I^2C-Komponente handelt. Über die Jumper 1. . . 3 kann auf der I^2C-Schnittstellenplatine die I^2C-Hardware-Adresse des D/A-Wandlers im Bereich von 0x90. . . 0x9e eingestellt werden, technische Details dazu gibt es im Anhang B.1.

> Die Softwareadresse des PCF8591 sowie auch die aller anderen I^2C-Komponenten, sind in der Datei **inc/contants.inc** festgelegt, die High-Level Routine zur Ansteuerung des PCF8591 ist in **lib/pcf8591.lib** abgelegt.

7.7.1.3. Permanentspeicherung der Winkelwerte

Die Forderung, Winkelpositionen auch nach dem Abschalten der Anlage wieder herzustellen, wurde im vorliegenden Fall durch Verwendung des Permanentspeichers der **halvedDISC** in Form des internen EEP-ROMs realisiert, um einen einfachen inkrementellen Winkelschrittzähler zur Messung des Drehwinkels verwenden zu können.

Dieser Speicher wird einerseits zum Speichern der „letzten” Position verwendet und andererseits zum Speichern von bis zu 16 Winkelpositionen, die im Rahmen der automatischen Positionierung angefahren werden können.

Die Speicheradresse im EEPROM Modul wird zuerst über eine I/O-Adresse im 16-Bit Register EEAR(H/L) (**EE**PROM **A**ddress **R**egister) adressiert, danach wird der zu speichernde Wert mit einem „OUT-Befehl” in das Datenregister EEDR (**EE**PROM **D**ata **R**egister) und damit in den Speicher geschrieben.

> Alle Routinen, die mit dem EEPROM der halvedDISC zu tun haben, befinden sich in der Datei **lib/eepromio.lib**.

7.7.2. Eingabeeinheiten

An Port-Input wird in der konkreten Anlage nur die Bedientastatur und der über einen Lichtschranken getriggerte Hardwarezähler für den internen 16-Bit Timer/Counter1 des ATmega8 verwendet. Eingaben, die über den I^2C-Bus durchgeführt werden, sind das Auslesen der aktuellen Temperatur und das Lesen und Schreiben der Echtzeituhr (RTC, **R**eal **T**ime **C**lock)[1]. Das Lesen der EEPROM-Werte wird, analog zum vorigen Abschnitt, mit einem Port „IN-Befehl” über das Datenregister EEDR, nach Adressierung des EEPROM-Speichers über EEAR, in ein internes Register eingelesen.

7.7.3. Hardwaresteuerung und Interrupts

Port-Eingabeaktionen sind im vorliegenden Fall immer extern getriggerte Ereignisse. Sowohl eine Benutzerinteraktion (Tastatureingabe) als auch das Zählen der Drehwinkelimpulse können zu jedem Zeitpunkt auftreten und werden deshalb sinnvoller Weise von Interrupt-Behandlungsroutinen (ISR, **I**nterrupt **S**ervice **R**outine) abgearbeitet.

Da jedoch auch Interrupts von weiteren Interrupts unterbrochen werden können und diese dann gemäß einer vorgegebenen Priorität abgearbeitet werden, kann es im Extremfall zu Verlusten von externen Ereignissen kommen. Deshalb liegt es nahe die ISRs möglichst kurz zu halten, bzw. das Abarbeiten weiterer ISRs zeitweilig zu unterbinden, wenn die Gefahr besteht, dass eine zwischenzeitlich auftretende Unterbrechung zu Informationsverlusten führen kann.

Da aber auf keinen Fall Zählschritte verloren gehen dürfen, wurden folgende Eingabebehandlungen und Zeit abhängige bzw. Zeit gesteuerte Mechanismen festgelegt:

1. Unkritische Warteschleifen, in denen keine unerwarteten Zustände eintreten können, werden als einfache Software-Loops (z.B. Tastatur-Wiederholrate bei niedergedrückten Tasten) ausgeführt. Dies beeinträchtigt nicht die Interrupt-Behandlung, verzögert aber die abhängigen Zyklen in vorhersehbarer weise.
2. Bei einer Tastatureingabe werden von einer minimalen ISR die notwendigen Informationen zur späteren Auswertung der Tastaturbetätigung in Variablen abgelegt.
3. Timer gesteuerte Interrupts (über den 8-Bit Timer/Counter0 ausgelöst) werden als „Pseudo-Scheduler” zur Steuerung zeitabhängiger Ereignisse verwendet.

[1] Die Zeitfunktionen wurden in der endgültigen Version entfernt, die Codeteile sind jedoch noch zum Teil vorhanden, werden aber in der aktuellen Implementierung nicht genutzt.

4. Das Zählen der Winkelschritte soll unabhängig passieren, sodass nicht einmal etwaige auftretende Interrupts dies beeinflussen können.
5. Das Anfahren einer Endposition wird im Pollingmodus durchgeführt, was auf Grund der geringen Dreh-Geschwindigkeit zu keinen Verlusten führen kann (siehe 7.8.1.3).

Diese Mechanismen werden nun in den folgenden Abschnitten im Einzelnen beschrieben.

7.7.3.1. Software-gesteuerte Zeitschleifen

In Situationen, in denen Wartezeiten als unkritisch angesehen werden können (wie. z.B. Zeitschleifen für die Tasten-Wiederholfunktion), wurden „normale" Software gesteuerte Zeitschleifen programmiert, die jederzeit kurzzeitig durch Interrupts unterbrochen werden können.

Es existieren Unterprogramme mit fixen Verzögerungen im Bereich von $200\,\mu s$ bis zu 1 Sekunde, sowie Zeitschleifen mit variablen Parametern. Diese können kombiniert werden um alle benötigten Verzögerungen zu erzielen.

```
; -----------------------------------------------------------------
; fixed time delay for approx. 50 msec
; -----------------------------------------------------------------
; input: none
; output: none
; affected: flags
;
TIME_50MS:           ; 3 - rcall assumed
  push  temp2        ; 2
  ldi   temp2, 0x00  ; 1
TIME_50MS_L:
  rcall TIME200US    ; 779*256
  dec   temp2        ; 1*256
  brne  TIME_50MS_L  ; 255*2
                     ; 1 - loop true value
  pop   temp2        ; 2
  ret                ; 4
;       3+2+1+779*256+1*256+2*255+1+2+4=200203
; 200203/4000000*1000=50.05075 msec
```

Aufruf-Beispiel einer parametrisierten Zeitverzögerung (Parameter der Zeitverzögerung in Millisekunden im Register **temp2**):

```
...
  ldi   temp2, Speedup_MSecs   ; loop time for go up phase
  rcall wait_msecs             ; wait Speedup_MSecs millisecs
  rjmp  increase_speed         ; ... increase speed
```

Alle Software-Timer Funktionen sind in der Datei **lib/delay.lib** definiert.

7.7.3.2. Behandlung der Tastatureingaben

Eine Tastatur-ISR sichert den Tastencode der zuletzt gedrückten Taste in der Variablen **curr_key_state**, der Status der einzelnen Tasten wird in den Variablen **key_state0...2** abgelegt.

```
; =================================================================
;        keyboard interrupt service routine
; this subroutine just stores the actual keyboard in
; $curr_key_state, also taking into account, that key0
; only sets bit 0b0000100 of port D. Pattern 0b00 is stored if
; key 0 is pressed, 0b10, 0b01 otherwise.
; -----------------------------------------------------------------
keyboard_isr:
  PUSHF                       ; save flags
```

```
  push  temp1
  push  temp2
  in    temp1, pinD
  andi  temp1, key_mask0      ; mask b0+b1
  sts   curr_key_state, temp1 ; store current key state
; we also set the state of the key to 'press_state'
  ldi   temp2, press_state
  dec   temp1
  brne  isr_not_key1          ; it was not key1
  sts   key_state1, temp2     ; it was key1
  rjmp  keyboard_isr_return
isr_not_key1:
  dec   temp1
  brne  isr_not_key2          ; it was not key2
  sts   key_state2, temp2     ; it was key2
  rjmp  keyboard_isr_return
isr_not_key2:
  sts   key_state0, temp2     ; it was key0
keyboard_isr_return:          ; as user feedback
  RED_LED_ON                  ; we switch red led on
  pop   temp2
  pop   temp1
  POPF                        ; restore flags
  reti         ; end of interrupt service routine
```

Der Tastaturinterrupt wird nur direkt beim Tastendruck ausgelöst, ein längeres Niederhalten einer Taste erzeugt keinen weiteren Interrupt. Um diesen Umstand berücksichtigen zu können, wurde eine Highlevel Tastaturbehandlung eingeführt (Keyboard-Postprocessing), die die Stati der Tasten, die Tasten-Niederhaltedauer im speziellen und die Tastaturwiederholungen behandelt.

Dazu wird pro Taste eine Statusvariable mit Statuswerten verwendet, dies sind die Variablen **key_state0**, **key_state1** und **key_state2**, die die in Tabelle 7.1 festgelegten Status-Werte annehmen können.

Tabelle 7.1.: Tastatur Statuswerte

Status	Wert	Bemerkung
inact_state	0x00	Taste ist und wurde nicht gedrückt bzw. alle Keyboard Events sind bereits abgearbeitet
activ_state	0x01	Taste wurde gedrückt, jedoch noch nicht „verarbeitet" oder sie wird noch niedergehalten
press_state	0x02	Taste wurde gedrückt, jedoch noch nicht „verarbeitet" Dieser Wert wird von der Tastatur-ISR gesetzt
	>0x02	Beim längeren Niederhalten wird dieser Wert bei jedem Postprocessing-Durchlauf erhöht

Die Statusvariablen sind nach der im Abschnitt 7.5 festgelegten Struktur in der Datei **inc/variables.inc** abgelegt.

Die Statuswerte sind als Konstante in der Datei **inc/constants.inc** festgelegt, wie auch die „symbolischen" Tastenbezeichnungen **key_mask0**, **key_next** und **key_middle** für die mittlere Funktionstaste, **key_mask1**, **key_back** und **key_left** für die linke Funktionstaste und **key_mask2**, **key_enter** und **key_right** für die rechte Taste.

Damit wird eine gute Lesbarkeit und bessere Wartbarkeit des Source-Codes erreicht.

Zusätzlich zu den „normalen" Tastaturbehandlungs-Routinen, die den Status einer Taste oder die „Niederhaltezeit" liefern, existieren High-Level Funktionen mit einem hohen Abstraktionsgrad, sodass diese Funktionen auch bei einer Änderung der Hardware weiter verwendet werden könnten. Einige Beispiele sind in Tab. 7.2 angeführt.

Die sehr mächtige Werte-Eingabefunktion Funktion **keyproc_varvalue_preset** stellt die Basis aller Werteingaben der Drehwinkelsteuerung dar und ermöglicht die Kontrolle eines beliebigen nummerischen 16-Bit

Tabelle 7.2.: Highlevel Tastaturbehandlungen

Subroutine	Beschreibung
wait_key_confirm	Ausgabe einer Textmeldung und Warten auf eine Taste
wait_key_pressed	Wartet auf einen beliebigen Tastendruck
wait_key_release	Wartet bis keine Taste mehr gedrückt ist
check_keypress	Prüft ob und welche Taste betätigt wurde
keyproc_varvalue_preset	Komplexe Tastatur Eingaberoutine - siehe unten

Wertes, wobei über Parameter das zu ändernde Register und der Maximalwert festgelegt wird (Tabelle 7.3).

Tabelle 7.3.: Highlevel Werte-Eingabefunktion

Eingabe:	LREG_A	16-Bit Wert, der über die Tastatur geändert wird
	MREG_A	16-Bit Wert als oberer Grenzwert, die untere Grenze ist immer 0
Ausgabe:	LREG_A	Interaktiv ausgewählter 16-Bit Wert
Verändert:	var_valdirection	intern verwendete Speichervariable, die die Änderungsrichtung (Zunahme, Abnahme) steuert

Die nähere Beschreibung dieser Funktion aus Sicht der Anwendung im GUI kann im Abschnitt 7.6.2.2 nachgelesen werden.

Alle Tastaturroutinen sind in der Bibliotheksdatei **lib/keyboard.lib** zusammen gefasst.

7.7.3.3. Pseudo-Scheduling Funktion zur Ausgabesteuerung

Eine hohe Anforderung an ein User-Interface ist jene, dass eine möglichst regelmäßige Aktualisierung der Anzeigen stattfinden soll. In vielen Situation ändern sich quasi zugleich 2 Zeilen der LCD-Anzeige ständig, was nach einer hohen „Refresh-Rate” verlangt.

Ein zu häufiges Ansteuern der Anzeige, die sehr oft mit Display- und Businitialisierungen verbunden ist, kann allerdings dazu führen, dass auf Grund der „Trägheit” des LCD-Displays die Anzeige noch nicht fertig aufgebaut ist, wenn der neue, eventuell geänderte Wert bereits wieder ausgegeben wird. Dies führt zu einem „Verblassen” der LCD-Anzeige bzw. zu einem unruhigen „Flackern” das die Lesbarkeit deutlich vermindert.

Um dies zu verhindern, wurde eine einfache Zeitsteuerung über den 8-Bit Timer/Counter0 realisiert (Scheduling [7] S. 188). Dieser niedrigst-priorisierte Timer-Interrupt wird dazu folgendermassen „programmiert”:

```
; ... from inc/constants.inc
; 4000000/256=15625 ~ 16kHz/TCNT0
.equ Timer0Div        = 0b00000100 ; cpuClk/256
.equ Timer0Val        = -128       ; 15625/128~120Hz
; ...
; level0_value = 4000000/256/128 ~ 8.2msec
; ----------------------------------------------------------
 .equ schedLevel1Val  = 0x10  ;  16*level0_value ~ 7.5Hz
 .equ schedLevel2Val  = 0x80  ; 128*level0_value ~ 0.9Hz
; ... from lib/systimer.lib
; Timer0 clock(io)/64
  OUTB  TCCR0, Timer0Div       ; prescaler value
; Timer0 oreset value (-128)
  OUTB  TCNT0, Timer0Val       ; set new init value
```

Der daraus resultierende Grundtakt (z. Zt. ca. 120 Hz) wird nun in drei Schedulerringen mit unterschiedlichen Teilern verwendet, damit ist es möglich, drei unabhängige Zeit-Zyklen (Ringe) zu verwalten.

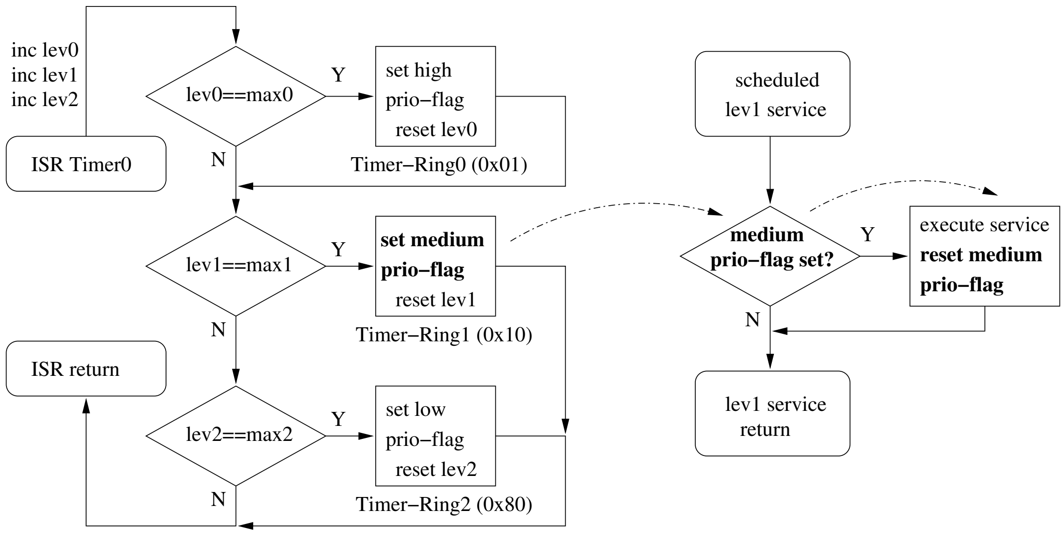


Abbildung 7.13.: Scheduler und „scheduled Service” Beispiel

Die konkrete Implementierung verwendet dazu die **Timer/Counter0 Overflow I**nterrupt **S**ervice **R**outine, die drei Schalter (Flags) verwendet, die im jeweiligen Ring nach Erreichen eines vordefinierten Timerwertes auf „aktiv” gesetzt werden.

Jede Service-Routine, die einen dieser Ringe verwendet, muss das Flag auswerten und die Abarbeitung abbrechen wenn der Wert „inaktiv” ist. Falls das Flag aber „aktiv” ist, wird diese Serviceroutine ausgeführt und am Ende der Routine wird das Flag von der Service-Routine wieder auf „inaktiv” gesetzt, danach beginnt der Zyklus wieder von vorne.

> **Anmerkung:**
> Im Gegensatz zum herkömmlichen Scheduling Begriff, der den einzelnen Services eine garantierte Ausführungszeit zusichern soll, zielt diese Methode darauf ab, für ein Service eine kürzestmögliche Wiederholzeit festzulegen, unabhängig davon, wie oft der Programmteil aufgerufen wird.

Abb. 7.13 zeigt ein konkretes Beispiel, das die Steuerung der Anzeigenhäufigkeit (**auto_update_display**) im automatischen Positioniermodus (siehe 7.8.1) kontrolliert. Dieses Service wird mit mittlerer Priorität ausgeführt.

```
; -----------------------------------------------------------------
; scheduler controlled display update (can be called always)
; -----------------------------------------------------------------
auto_update_display:                ; 3 rcall overhead
  lds   sched_temp, level1_waiting  ; 2 memory access
  tst   sched_temp                  ; 1 test function
  brne  auto_update                 ; 1 (2 if true) branch
  ret                               ; 4 - sum=11 (2.75 usecs)
; ...
; do display update ...
auto_update:
  PUSHW REG_Z
  push  ltmp2
  push  temp1
  rcall powr_funcmode_1 ; update power value display
  rcall angl_funcmode_2 ; update angle value display
  pop   temp1
  pop   ltmp2
  POPW  REG_Z
; -----------------------------------------------------------------
; reset scheduler values for a new timer cycle
```

```
clr   sched_temp
sts   level1_waiting, sched_temp ; reset scheduler flag
ret
```

In diesem Projekt wird Ring 1 zur Steuerung der Anzeigenhäufigkeit im automatischen Positioniermodus (siehe 7.8.1) und Ring 2 zur Aktualisierung der Temperaturanzeige im „Idle” Modus (in der Datei **lib/main-menu.lib**) verwendet, Ring 0 wird in der aktuellen Implementierung nicht benutzt.

Alle Timer-Initialiserungen und Timer-Services sind in **lib/systimer.lib** definiert.

7.7.3.4. Winkelschrittzählung durch Zählerprogrammierung des ATMega8

Der 16-Bit Zähler **Timer/Counter1** ist in der Lage vollkommen unabhängig von externen Interrupts Signale am Takt-Eingang **T1** zuverlässig zu zählen, wobei verschiedene Modi unterstützt werden. Im gegenständlichen Fall reicht ein automatisiertes Hochzählen des Counters, bis ein bestimmter Wert erreicht ist (der Winkelmaximalwert), danach wird der Wert automatisch von der Controllerhardware wieder zurückgesetzt und der Zählzyklus beginnt von vorne.

Die notwendige Initialisierung der ATMega8-Register wird nun im Sourcecode von **lib/systimer.lib** dargestellt.

```
; -----------------------------------------------------------------
; Counter1 settings
; -----------------------------------------------------------------
; TCCR1A Timer/Counter 1 Control Register A
; Mode CTC (ATmega manual p. 97)
; ...
; bits 7   6   5   4   3   2   1   0
;    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  OUTB TCCR1A,0b00000000 ; normal mode (no PWM,WGM,...)
; TCCR1B Timer/Counter 1 Control Register B
; bits 7   6   5   4   3   2   1   0
;    | X | X | X | 0 | 1 | X | X | X |
; 0b00001000 - WGM13
; 0b00000100 - WGM12 = CTC *** USED!
; CTC mode (Clear Time on Compare)
; WGM13:0 WGM12:1 - WGM11:0 WGM10:0 = 01 - 00
; see also TCCR1A settings above WGM10, WGM11
; TCCR1B Clock Select bits
;    | X | X | X | X | X | 1 | 1 | 1 |
; 0b00000110 - ext. clock pin T1 - falling edge
; 0b00000111 - ext. clock pin T1 - rising edge *** USED!
;
; 0b000xxxxx                       - no ICNC1, ICES1
; 0bxxx01xxx|xxxxx00 TCCR1B|TCCR1A - mode 4 = CTC mode
  OUTB TCCR1B,0b00001111
; see Table 39 ATmega8 manual p. 97
; -----------------------------------------------------------------
; TIMSK Timer/Counter Interrupt Mask Register
; bits 7   6   5   4   3   2   1   0
;    | X | X | X | 1 | 0 | X | X | 1 |
; 0b00010000 - OCIE1A Timer/Counter1, Output Capture A Match Enable
; 0b00001000 - OCIE1B Timer/Counter1, Output Capture B Match Enable
; 0b00000001 - TOIE0  Timer/Counter0, Overflow Interrupt Enable
  OUTB  TIMSK,(1<<OCIE1A)|(1<<TOIE0)
; -----------------------------------------------------------------
; set timer 1 output compare register A value to angle max. value
  LDIWR REG_A, RotationOver-1
  OUTWR OCR1A, REG_A
```

Hinweis:
Bei den Assembleranweisungen LDIWR und OUTWR handelt es sich um Assemblermakros, die ein 16-Bit Register mit einem 16-Bit IMMEDIATE Wert laden (siehe dazu auch Absch. 7.5.6 zum Thema Namenskonventionen), bzw. den Inhalt eines 16-Bit Registers in ein Port

schreiben, was nach einer definierten Reihenfolge beim Schreiben der beiden 8-Bit Werte verlangt [15] S. 77. Das OUTB Makro schreibt ein Byte in ein Ausgaberegister.

Der verwendete CTC-Modus (**C**lear **T**imer on **C**ompare Mode) inkrementiert das Zählregister TCNT1(H/L) des Timer/Counter 1 und stellt den Zähler wiederum beim Erreichen des Vergleichregisters OCR1A(H/L) (**O**utput **C**ompare **R**egister **1A**) auf 0 zurück und sorgt somit automatisch für den korrekten Zählerüberlauf.

Ein Zwei-Richtungsbetrieb ist mit dieser einfachen Timerprogrammierung allerdings nicht möglich, da bei einer Richtungsänderung auch der Zähler umprogrammiert werden müsste, damit er „negativ” zählt.

Mit dieser Maßnahme erfolgt das Zählen unabhängig von anderen Ereignissen und Interrupts.

7.7.3.5. Positionskontrolle über Interrupts

Im Absch. 7.8.1 über das automatische Positioniermodul wird beschrieben, wie die Steuerungslogik das Erreichen einzelner Steuerungsphasen kontrollieren muss. Wenn eine Endposition der Drehbewegung im vorhinein bekannt ist, kann eben dieser hier verwendete Zähler zum Kontrollieren der bestimmten Position verwendet werden.

Durch Programmierung des OCR1B-Registers (**O**utput **C**ompare **R**egister **1B**) kann eine Position definiert werden, bei deren Erreichung der Interrupt 0x08 (Output Compare1B Interrupt Vector Address) ausgelöst wird. Dazu muss im Kontrollregister TIMSK (**T**imer **I**nterrupt **M**ask) das Bit OCIE1B (**O**utput **C**ompare **I**nterrupt **E**nable **1B**) gesetzt werden.

Das Setzen dieses Registers erfolgt im Unterprogramm **setup_counter_value**.

```
; -------------------------------------------------------------------
; set the next counter interrupt value
; -------------------------------------------------------------------
; input:    REG_A   - initial rotation angle
; output:   none
; affected: none
; -------------------------------------------------------------------
setup_counter_value:
  push  temp2                 ; used in macro SBIP
  OUTWR OCR1B, REG_A          ; write new value to OCR1B
  SBIP  TIMSK, (1<<OCIE1B)    ; OCIE1B bit in TIMSK will be
  pop   temp2                 ; reset if OCIE1B value (REG_A)
ret                           ; value is reached
```

Diese Routine aktiviert den o. g. Interrupt und setzt den erwarteten „Endwert” des Zählers TCNT1, der direkt über den Winkelschrittgeber (Absch. 5.5) getriggert wird, in dem sie das 16-Bit Register OCR1B „programmiert”.

Die dazugehörige Interrupt-Service-Routine macht nichts anderes, als den Interrupt wieder zu deaktivieren, indem das entsprechenden Bit im Timer Interrupt Mask Register (TIMSK) wieder gelöscht wird.

```
; -------------------------------------------------------------------
; rotation angle is reached, we disable the interrupt
; -------------------------------------------------------------------
T1_step_isr:
  PUSHF
  push  temp2               ; used in macro CBIP
  CBIP  TIMSK, (1<<OCIE1B)
  pop   temp2
  POPF
  reti                      ; return from interrupt...
```

Da für die Steuerung nur die Information über das Erreichen der Endposition des jeweiligen Winkels von Interesse ist, kann dieser Umstand direkt aus dem TIMSK-Bit abgelesen werden. Diese Abfrage erfolgt im Rahmen der Assemblermakros „AFTER_POSITION” und „BEFORE_POSITION”, diese werden auf Seite 73 im Abschnitt 7.8.1.3 noch ausführlicher erläutert.

Die Timer-ISR Routinen sind in **lib/systimer.inc** abgelegt, die Makros „AFTER_POSITION” und „BEFORE_POSITION” sind in **inc/macros.inc** definiert.

7.8. Programmmodule

Dieses Unterkapitel beschreibt die einzelnen Programmmodule, die letztendlich die tatsächliche Funktionalität der Anlage beschreiben. Es wird hier versucht, die einzelnen Module aus der Sicht des Anwenders und der Bedienung zu beschreiben, auch wenn intern zahlreiche Querverbindungen zwischen den einzelnen Programmmodulen bestehen.

Weiters soll dem Leser auch die methodische Verwendung des Sourcecodes und dessen Aufbaus anhand der konkreten Beispiele nähergebracht werden und ihm die Möglichkeit geben, die Methodik des Programmcodes, stellvertretend auch für die anderen Module, kennen zu lernen, besser zu verstehen und eventuell ähnliche Projekte selber effizient mit diesen Methoden umsetzen zu können. Damit werden auch die in Abschnitt 7.1 beschriebenen Mechanismen an konkreten Beispielen demonstriert.

7.8.1. Automatisches Positioniermodul

Aus der primären Zielsetzung der Anlage heraus, besteht der Großteil der Tätigkeiten in der Messung der Staubschichtdicken auf den Stofffiltern.

Mit der Funktionserweiterung der Drehbarkeit eines Filterkorbes, ist nun die Möglichkeit vorhanden, dass auch bisher nicht sichtbare Bereiche in den Messbereich der optischen Dickenmessung „gedreht" und damit vermessen werden können (siehe dazu auch Absch. 2.1). Diese Tätigkeit, die auch in der Menüstruktur (Abb. 7.8 auf Seite 56) optisch hervorgehoben ist, stellt die „normale" Handhabung der Anlage dar und besteht aus dem automatischen Anfahren bereits vordefinierter Positionen.

Diese meist verwendete Funktion - das Anfahren der „nächsten" Winkelposition - wird über maximal drei Tastendrücke erreicht, wenn sich die Anlage im „Ruhezustand" (Idle-Mode) befindet.

Als weitere Vorgabe muss die Drehbewegung möglichst ruhig erfolgen, um die Messungen nicht durch „Abbröckeln" von Filterkuchenanteilen zu verfälschen. Dieses „ruckfreie Anfahren" und das „sanfte Herunterfahren" in der Endposition muss die Steuerung ohne Eingriff einer Benutzeraktion automatisch ausführen.

7.8.1.1. Der Positionier-Algorithmus

Die Drehrichtung des Motors ist fixiert, um Ungenauigkeiten resultierend aus dem Umkehrspiel der Zahnräder auszuschließen (siehe auch 4.3.2). Der Berechnungsalgorithmus zur Steuerung der Winkellage ergibt sich aus folgender Formel.

$$\Phi_d = \alpha_2 - \alpha_1$$

Φ_d … Differenzwinkel, um den gedreht werden muss (Winkelzählschritte)
α_1 … aktueller Winkel, bzw. Ausgangslage (aktueller Zählerstand)
α_2 … anzusteuernder Winkel, bzw. Endlage (Zählerstand in der Endlage)

Aus der Nebenbedingung, dass nur eine Drehrichtung verwendet wird, erhält man einen negativen Differenz-Winkel Φ_d, wenn der Endwinkel innerhalb des Winkelbereiches $0° \ldots 360°$ kleiner als der Anfangswinkel ist. In diesem Fall wird der negative Differenz-Winkel über die folgende Formel in einen positiven Wert Φ_p umgewandelt.

$$\Phi_p = \Phi_d + 360°$$

Φ_p … positiver Absolutwert der Endlage (kann $\geq 360°$ sein)

Die Messung der Winkellage erfolgt mit einem inkrementierenden Signalzähler, der bei Überschreiten von 360° auf 0° zurückgesetzt wird (siehe Absch. 7.7.3.4). Da Φ_p aber Werte annehmen kann, die größer als 360° sind, würde dieser Wert niemals erreicht werden.

Zur Lösung dieses Problems wurde die 5-Phasen Steuerung erarbeitet, bei der nur „Phasen-Endpunkte" angefahren werden, wodurch die Anzahl der konkreten Winkelschritte quasi „nebensächlich" wird.

Dieser Positionier-Mechanismus wird nun im Detail behandelt.

7.8.1.2. Konzept der 5-Phasen Steuerung

Der „normale" Positionierzyklus durchläuft bei der automatisierten Positionierung fünf Phasen, wobei nicht zwingend alle Phasen auftreten müssen, wenn sie der Steuerungslogik widersprechen. Im Folgenden werden die einzelnen Phasen kurz beschrieben und beispielhaft LCD-Anzeigeninhalte zu jeder Phase dargestellt.

```
Auto Control
Power:069=45
Angle: 355.0
```

1. Initialisierung und Startphase:
   Diese Phase definiert den Zeitraum in dem noch keine Bewegung eintritt (bevor das Drehmoment aus dem Steuerungs-Vorgabewert den Wert der Haftreibung der Drehvorrichtung überwindet).

```
GoUp Mode>>>
Power:188=BC
Angle: 002.0
```

2. Geschwindigkeit-Anstiegsphase:
   In dieser Phase wird die Geschwindigkeit stetig erhöht, bis der Maximal-Steuerungs-Vorgabewert von 255 (Hex 0xFF) erreicht ist.

```
High Mode---
Power:255=FF
Angle: 005.5
```

3. Die Maximal-Geschwindigkeitsphase:
   Nach erreichen der Höchstgeschwindigkeit bleibt die Drehgeschwindigkeit konstant, bis die Geschwindigkeits-Reduzierphase eintritt.

```
Down Mode<<<
Power:201=C9
Angle: 170.0
```

4. Geschwindigkeits-Reduzierphase:
   Um an einer vordefinierten Stelle anzuhalten, müssen die letzten Schritte mit einer sehr geringen Schrittgeschwindigkeit durchgeführt werden, um im richtigen Zeitpunkt die Spannung abzuschalten damit die Anlage an der gewünschten Position stoppt. Diese Phase verringert die Geschwindigkeit stetig, bis die „Positionier-Geschwindigkeitsphase" eintritt.

```
Slow Mode <
Power:120=78
Angle: 214.0
```

5. Positionier-Geschwindigkeitsphase:
   Dies ist die eigentliche Positions-Kontrollphase. Mit einer vordefinierten Minimalgeschwindigkeit wird die vorprogrammierte Endposition angesteuert, ein Schritt vor Erreichen dieser Position wird die Steuerspannung auf 0 geschaltet und die Trägheit bewegt den Filterkorb in die Endlage.

Diese fünf Zyklen stellen die Basis der automatischen Positionierung dar, jedoch kann es passieren, dass nicht alle fünf Phasen durchlaufen werden.

Unter welchen Bedingungen tritt diese Situation nun ein?

Die einzelnen Positionierphasen werden nun im Geschwindigkeits-Zeit Diagramm in Abb. 7.14, unter Berücksichtigung der Phasen-Zykluszeiten, näher betrachtet.

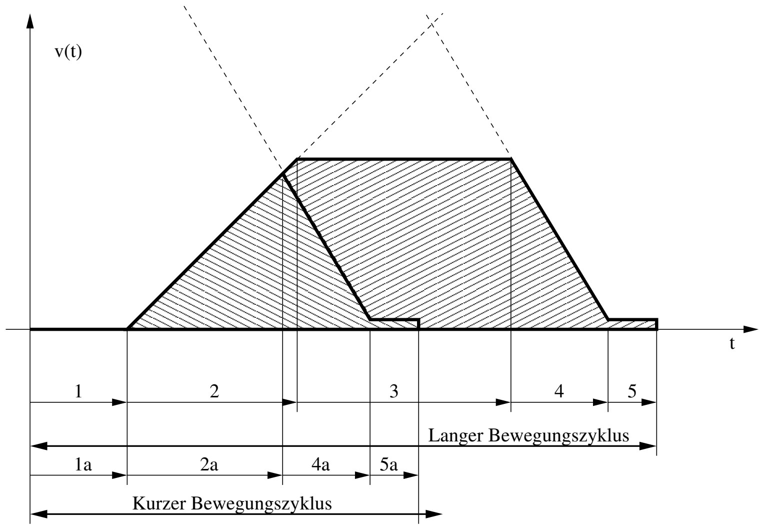


Abbildung 7.14.: Positionierphasen-Überschneidung

- Die Phase 1 hängt vorwiegend von den physischen Gegebenheiten der Anlage (Reibungwerte, Montagebedingungen, ...) ab und ändert sich kaum im Laufe der Zeit.
- Die Phasen 2 und 4 sind in der Software parametrisiert und können somit ebenfalls als konstant angenommen werden und hängen, wie Phase 1 auch von den Anlagengegebenheiten ab. Es ist zwar möglich, dass diese Phasen kürzer ausfallen, jedoch hat diese Situation keine Relevanz für diese Betrachtung.
- Die Phase 5 ist auch primär parametrisiert, da für sie eine fixe Anzahl von Bewegungsschritten reserviert ist, und kann daher ebenfalls als fix angesehen werden.

Daraus schließend kann nur Phase drei (die Maximal-Geschwindigkeitsphase) als nicht vorherbestimmbare Phase - im Sinne der Laufzeit - diesen Zustand verursachen.

Mathematisch ausgedrückt kann nun folgendes geschlossen werden:

$$t_{Cycle} = t_1 + t_2 + t_3 + t_4 + t_5$$

$$t_3 = t_{Cycle} - t_1 - t_2 - t_4 - t_5$$

Dies führt zur Forderung nach einer minimalen Zykluszeit, bei deren Unterschreitung die Phase III entfallen muss:

$$t_{Cycle} \geq t_1 + t_2 + t_4 + t_5 \Rightarrow t_{3_{min}} \geq 0$$

Es bedeuten:

t_{Cycle} ... der gesamte Steuerungszyklus

t_1, t_2, t_3, t_4, t_5 ... die Zykluszeiten der Phasen 1...5.

$t_{3_{min}}$ ... die minimale Zykluszeit für Phase 3

Aus dieser Betrachtung werden folgende Rückschlüsse abgeleitet:

1. Falls der Positionierzyklus kürzer oder gleich der Summe der Zyklen aus 1, 2, 4 und 5 ausfällt, entfällt die Maximalgeschwindigkeitsphase.
2. Bei Unterschreiten der minimalen Positionierzyklus-Zeit können ebenfalls die Anstiegsphase und Reduzierphase entsprechend verringert werden (siehe „Kurzer Bewegungszyklus" in Abb. 7.14).

Wie dieser Algorithmus tatsächlich in Assembler umgesetzt wurde, wird im folgenden Abschnitt dargestellt.

7.8.1.3. Umsetzung der 5-Phasensteuerung

Die Funktion wird im Modul **Goto Position** (siehe Menüstruktur in Abb. 7.8 auf Seite 56), entsprechend der Assemblerdatei **mod/got_posn.asm** direkt gestartet und benötigt außer der Definition des Zielwinkels keine weiteren Eingaben mehr.

Die Routine beginnt mit einem Assembler-Label nach der Namenskonvention aus Absch. 7.5.6, nämlich mit **select_position_main**.

```
; ======================================================================
; goto positon main entry point
; ----------------------------------------------------------------------
; directly called from mainmenu 'Goto Positn.'
; 3 options:
;   1. (Enter) execute select next position
;   2. (Back) leave select_position_main
;   3. (Next) goto position select menu
; ----------------------------------------------------------------------
; input: none
; output: none
; affected: flags
; ----------------------------------------------------------------------
select_position_main:
  push  temp1
; ......................................................................
select_position_id:
  rcall next_notempty_angle
  rcall goto_pos_main_menu
  WAIT_MENU select_position_main
  rcall wait_key_release
  cpi   temp1, key_next
  breq  select_position_id
; ......................................................................
  rcall wait_key_release
; set preset position (end value)
  lds   ltmp1, last_position
  rcall restore_angle_positon
  MOVWR REG_A, REG_Z              ; REG_A is the required "next" position
  rcall select_position_enter     ; accept "enter next position"
; rjmp  select_position_main_ret
; ......................................................................
select_position_main_ret:
  STSB  glb_menu_mode, menumode_idle
  pop   temp1
  ret
```

Dieser Programmteil dient nur dazu, eine der max. 16 gespeicherten Winkelpositionen auszuwählen.

Bemerkungen:
Im Makro „WAIT_MENU" wird bereits auf das Verlassen der Funktion über die „*Back*"-Taste (siehe 7.6.2.1) reagiert, dieses Makro verzweigt zum standardmäßig bezeichneten Exit-Label **select_position_main_ret** (Name des Assembler Einsprung-Labels mit der Nachsilbe „_ret") wenn die „*Back*"-Taste betätigt wurde.

```
; ------------------------------------------------------------------
; wait for a menu seelction
; ------------------------------------------------------------------
;
; Parameter: @0 .. label for select loop and exit label
;
; affected: Z-flag, temp1
.macro WAIT_MENU
  rcall wait_key_release
@0_loop:
  rcall check_keypress
  breq  @0_loop
  cpi   temp1, key_back
  breq  @0_ret
  WAIT_LOOP 1
.endmacro
```

Nur durch Anwahl dieser Funktion über die „*Enter*" Taste wird das Unterprogramm **select_position_enter** angesprungen. In dieser Routine werden die fünf Steuerungsphasen der automatischen Positionskontrolle durchlaufen.

Bemerkung: Diese fünf Phasen sind jedoch nicht identisch mit den fünf Phasen aus Absch. 7.8.1.2, die die Anwendersicht darstellen. Diese Betrachtung der 5-Phasen-Steuerung spiegelt die programminterne Strukturierung und Algorithmik wieder, daher werden sie bewusst als „Steuerungsphasen" im Gegensatz den den vom Anwender wahrgenommenen „Positionierphasen" bezeichnet.

Beispielsweise finden sich die beiden Positionierphasen „Initialisierung und Startphase" und „Geschwindigkeits-Anstiegsphase" in der Programmlogik in einem einzelnen Programmteil, nämlich die kontinuierliche Steigerung der Sollwertvorgabe von einem Minimum bis zum Maximum, wieder. Andererseits benötigen der Übergang von der Minimalgeschwindigkeits- in die Stopp-Phase programmtechnisch eigene steuerungstechnische Maßnahme (z. B. Speichern des letzten Winkelwertes im EEPROM), wohingegen vom Anwender die „Nichtbewegung" nicht als eigene Positionierphase wahrgenommen wird.

Die unterschiedliche Nummerierungsart soll auch diesem Umstand Rechnung zu tragen.

```
; ==================================================================
; MAIN rotation movement and control end position using 4 steps
; ------------------------------------------------------------------
; I.   phase_speed_goup - rotation_mode=ModeSpeedGoUp
; II.  phase_speed_high - rotation_mode=ModeSpeedHigh
; III. phase_speed_down - rotation_mode=ModeSpeedDown
; IV.  phase_speed_slow - rotation_mode=ModeSpeedSlow
; V.   phase_speed_none - rotation_mode=ModeSpeedNone
;
; the rot_angle_godown position is controlled by OCIE1B which
; has been programmed in rotation_startup subroutine
;
; the rot_angle_preset position is controlled by OCIE1B which
; will be programmed in rotation_high, rotation_down or
; rotation_slow
;
; ------------------------------------------------------------------
; goto position main function
; move facility to a specific angle position
; -------------------------------------------------------------------
; input:  REG_A - requested rotation end position
; output: none
; affected: flags
; -------------------------------------------------------------------
select_position_enter:
  push  temp1
  push  ltmp1
  PUSHW REG_Z
  PUSHW REG_A
```

```
  rcall auto_funcmode            ; display angle information
; ..............................................................
; set rotation direction - currently always positive direction!!!
  ldi   temp1, ValueDirIncr      ; direction flag (-1,0,+1)
  rcall set_direction
; precompute rotation values     ; noof steps returned in REG_B
; ..............................................................
; Preparation phase: data preprocessing...
  rcall rotation_preproc         ; rot_angle_godown and rot_angle_preset
                                 ; values stored
  LDSWR REG_A, rot_angle_godown  ; store req. position (with overflow)
  rcall setup_counter_value      ; write slow_down value to OCR1B
                                 ; and start steps counting process
; ..............................................................
; PHASE I: rotation speed goup
  rcall phase_speed_goup         ; input: REG_B - no of steps to endpos
  lds   temp1, rotation_fail_flag ; check the failure flag...
  tst   temp1
  brne  failure_occurred         ; if failure occurred ...
  AFTER_POSITION skip_speed_high ; uses ltmp1
; continue on hisped?            ; ... we skip hispeed phase ...
```

Steuerungsphase I (SpeedGoUp) Diese Startphase muss folgende Aufgaben abdecken:

1. Initialisierung der Rotationsfunktion (Richtungs-Steuerung)
2. Das Präprozessing, also die Vorberechnung des Zählerwertes, an dem die Geschwindigkeit in die „SpeedDown" Phase III übergeht.
3. Die „Programmierung" des **Ouput Compare 1B** (OCR1B) Interrupt Registers mit diesem Wert (`rcall setup_counter_value`).
4. Kontinuierliche Erhöhung die Steuerspannung, bis die Drehbewegung einsetzt. Dieser „Startwert" wird als Variable **rot_contrl_minimum** gespeichert und dient in der „SpeedDown"-Phase III als Zielwert für die Geschwindigkeitsreduzierung.
5. Überprüfen, ob die Rotation beim Maximalwert tatsächlich begonnen hat, wenn nicht, wird das Programm abgebrochen (siehe Schlagwort **InitCycle** im Absch. 7.8.5).
6. Überprüfen, ob die „SpeedDown"-Phase bereits erreicht ist. Ist dies der Fall, wird die „SpeedHigh"-Phase übersprungen.

Zur Verbesserung der Lesbarkeit des Codes wurden die Makros „BEFORE_POSITION" und „AFTER_-POSITION" eingeführt, die das Erreichen oder „noch nicht" Erreichen des vorberechneten Endwerts überprüfen, indem das entsprechende Bit im Timer Interrupt Mask Register (TIMSK) kontrolliert wird. Das Bit wird von der ISR des Timer1B (Output Compare1B Interrupt Vector Address) kontrolliert.

```
; ==================================================================
; interrupt checks
; ------------------------------------------------------------------
; check if a value is reached (TIMSK bit is set)
; ------------------------------------------------------------------
;
; Parameter: @0 .. label to jump if reached
.macro AFTER_POSITION
  in    ltmp1, TIMSK      ; read TIMSK port, is bit OCIE1B set?
  sbrs  ltmp1, OCIE1B     ; position value is reached?
  rjmp  @0                ; if yes (bit is reset), we jump to...
.endmacro
```

Anwendungsbeispiele sind in den Beschreibungen der Phasen II und III zu finden.

Der Mechanismus zum Setzten und Rücksetzten des verwendeten **Output Compare1B Interrupts** wurde bereits in Abschnitt 7.7.3.5 beschrieben.

Steuerungsphase II (SpeedHigh)

```
; ..............................................................
; PHASE II: continue on hisped
  rcall phase_speed_high          ; drive on full hispeed otherwise
; continue on slowdown?           ; ... we skip slowdown phase ...
```

Die Hochgeschwindigkeits Phase macht nichts anderes, als auf den vorprogrammierten Endwert (OCR1B) zu „warten” und besteht im Prinzip nur aus der Warteschleife, die das Makro „BEFORE_POSITION” ausführt und die Zeit gesteuerte LCD-Aktualisierung „auto_update_display” ausführt.

```
; ------------------------------------------------------------------
; high speed rotation part
; until reaching the rot_angle_godown value we turn at full speed
; the rot_angle_godown position is controlled by OCIE1B
; ------------------------------------------------------------------
; input:    none
; output:   none
; affected: flags
; ------------------------------------------------------------------
phase_speed_high:
  STSB  rotation_mode, ModeSpeedHigh ; rotation mode is hispeed
  rcall auto_funcmode
; ..............................................................
hispeed_continue:                    ; loop until rot_angle_godown
  rcall auto_update_display          ; scheduled display update
  BEFORE_POSITION hispeed_continue   ; uses ltmp1
  ret
```

Die Implementierung von **auto_update_display** ist im Abschnitt 7.7.3.3 beschrieben, der sich mit Scheduler Mechanismus beschäftigt.

Die Mechanismen zur Verwendung des **Output Compare1B Interrupts** ist im Abschnitt 7.7.3.5 beschrieben, der sich mit der Verwendung des Interrupts zur Positionskontrolle beschäftigt.

Steuerungsphase III (SpeedDown) Diese Phase beginnt mit der „Programmierung” der Drehwinkel-Endposition im **Ouput Compare Register 1B** (OCR1B). Danach erfolgt die konstante Geschwindigkeitsabnahme, bis der in der „SpeedGoUp” Phase ermittelte Wert **rot_contrl_minimum** erreicht ist. Der Mechanismus ist identisch mit dem in Phase I beschriebenen, nur, dass hier die Geschwindigkeit abnimmt, bis der o. g. Endwert „rot_contrl_minimum” erreicht ist.

Da der Anstieg der Geschwindigkeit in der Phase I (SpeedGoUp) auf Grund der notwendigen Massenbeschleunigung und Haftreibung länger dauert als die Geschwindigkeitsabnahme in der Phase III (SpeedDown), verwenden die beiden Phasen unterschiedliche „Beschleunigungswerte”. Diese Schleifen-Verzögerungswerte für die Geschwindigkeitsänderung sind in den Konstanten **Speedup_MSecs** und **SlowdownMSecs** in der Datei **inc/constants.inc** definiert.

Mit dieser Maßnahme soll sicher gestellt werden, dass die „SpeedDown” Phase nicht länger dauert als die „GoUp” Phase und damit quasi „zu spät” (d.h. nach Erreichen der Endposition) in die „SpeedSlow” Phase übergehen würde. Eine zusätzlich Sicherheitsabfrage nach der „SpeedDown” Phase sorgt im Extremfall dafür, dass die „SpeedSlow” Phase übersprungen und eine Art „Notabschaltung” eingeleitet wird.

```
; ..............................................................
; PHASE III: speed slowdown
skip_speed_high:
  LDSWR REG_A, rot_angle_preset   ; store req. position (with overflow)
  SBIWR REG_A, 1                  ; decr. by 1
  brcc  store_counter
  ADIWR REG_A, RotationOver       ; correction if underflow
; ..............................................................
store_counter:                    ; write end position value to OCR1B
  rcall setup_counter_value       ; and start steps counting process
  rcall phase_speed_down          ; lower speed until rot_contrl_minimum
                                  ; value is reached
```

```
  AFTER_POSITION skip_speed_slow  ; emergency exit - should never occur
; slow rotation until target      ; move in slow speed to target position
```

Steuerungsphase IV (SpeedSlow) Diese letzte Bewegungphase ist die eigentliche exakte Positionierphase. In einer Schleife (getriggert von der Konstanten **RotationCheck**) wird überprüft, ob seit der „letzten” Positionsbestimmung eine Winkeländerung stattgefunden hat. Ist dies nicht der Fall, wird der Steuerwert erhöht, gab es eine Winkeländerung, wird die Sollwertvorgabe vermindert, alle 50 Millisekunden erfolgt zusätzlich die Überprüfung auf das Erreichen der Endposition, wodurch diese Steuerungsphase verlassen wird.

```
; ....................................................................
; PHASE IV: speed slow movement
skip_speed_down:                  ; ... and slow down to 0
  rcall phase_speed_slow          ; we continue in slow(est) speed
skip_speed_slow:                  ; until rot_angle_preset is reached
```

Steuerungsphase V (SpeedNone) Die letzte Phase der Winkelpositionierung besteht aus dem „Abstellen” des Drehvorganges durch Zurücksetzen des Steuerungs-Sollwertes auf 0 und der Speicherung dieser Position im Permanentspeicher des Mikrocontrollers als „letzte Position”.

```
; ....................................................................
; PHASE V: no rotation phase begins
  rcall stop_rotation             ; set speed to 0x00 and store angle

; ... from lib/rotation.lib
; ==================================================================
; rotation control stopping function
; ------------------------------------------------------------------
; we ALWAYS use this function to stop rotation, because it
; stores the "last value" in EEPROM
; ------------------------------------------------------------------
stop_rotation:
  push  temp1
  push  temp2
  clr   temp1                          ; stop rotation...
  rcall PCF_SEND_VALUE                 ; send STOP to pcf
  ldi   temp2, 5                       ; wait a rather long
  rcall wait_loop                      ; time... (0.5secs)
  rcall store_current_angle            ; store angle value in eeprom
  STSB  rotation_mode, ModeSpeedNone   ; set rotation mode to 'none'
  STSB  var_valdirection, ValueDirNone ; set rotation flag to 'none'
  pop   temp2
  pop   temp1
  ret
```

Der Modul-Code für den Auto-Positionier-Mechanismus befindet in **mod/goto_pos.asm**.

7.8.2. Hand-Steuerungsmodul

Dieses recht einfache Modul bietet die Möglichkeit die Steuerspannung zur Steuerung des Motors von 0 V bis zum Maximalwert 10 V im Rahmen des 8-Bit DA-Wandlers (von 0…255) über die Eingabetastatur zu definieren. Es besteht nur aus einer Schleife, in der die eigentliche „Tastatur-Steuerroutine” immer wieder mit einer 0.3 Sek. Verzögerung (WAIT_LOOP 3) aufgerufen wird, bis das Modul verlassen wird.

```
; ====================================================================
; manual control main program
; --------------------------------------------------------------------
; direct manual control of the PCF 8591 by controlling the "Power"
; --------------------------------------------------------------------
manual_control_main:
```

```
  rcall mod_manctrl_init         ; initialize manual control env.
...
; manual control loop begin...
mcntrl_loop:
  WAIT_LOOP 3                    ; software timer
  rcall speed_control            ; key controlled speed of rotation
  rcall check_keypress           ; manage keyboard events...
  breq  mcntrl_loop              ; ...nozzzin pressed...
  cpi   temp1, key_mask0         ; has "stop" been pressed?
  brne  mcntrl_loop              ; if not, we loooooooooooop...
; manual control loop end...
; --------------------------------------------------------------------
  rcall stop_rotation            ; we NEVER send stop directly to PCF
  rcall mctl_funcmode            ; do some screen io...
manual_control_main_ret:
  ret
```

Dabei wird der Mechanismus der nummerischen Werteingabe (siehe Absch. 7.6.2.2) verwendet um dem „Power”-Wert über die Tastatur zu kontrollieren, dieser wird dann zum D/A-Wandler PCF8591 geschickt und in der LCD-Anzeige zusammen mit dem aktuellen Drehwinkel ständig aktualisiert.

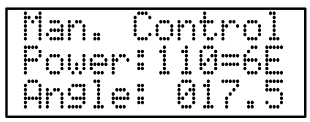


Abbildung 7.15.: Manuelle Winkelsteuerung

Die eigentliche Tastatursteuerung geschieht in der Subroutine **speed_control**. Das High-Level Tastaturkontroll-Unterprogramm **keyproc_varvalue_preset** führt anhand der aktuellen Tastenstati die konkrete Tasten-Wertumsetzung, unter Berücksichtigung der Tastendynamik, durch. Die Geschwindigkeit der Wertänderung nimmt daher mit der Tasten-Haltezeit, wie in Absch. 7.6.2.2 beschrieben, zu.

```
; ====================================================================
; manual control subroutine for speed control
; --------------------------------------------------------------------
; input: none
; output: none
; affected: temp1
; --------------------------------------------------------------------
speed_control:
  LDIWR REG_A, RotSpeedMax
  MOVWR MREG_A, REG_A            ; inp: maxmum 8-bit value
  LDSWR LREG_A, rot_speed_value  ; inp: value to modify
  rcall keyproc_varvalue_preset  ; call speed keyboard function
  STSWR rot_speed_value, LREG_A  ; storing the "new" rot_speed_value
  rcall mctl_funcmode            ; screen io...
  lds   temp1, rot_speed_value
  rcall PCF_SEND_VALUE           ; send "new" speed value to pcf
  ret
```

Der Modul-Code befindet sich (lt. Code-Organisation in Absch. 7.5.3) in der Datei **mod/-man_ctrl.asm**.

7.8.3. Winkel-Vorgabemodul

Dieses Modul erlaubt eine beliebige Winkelvorgabe über die Funktionstastatur und das „direkte” Anfahren dieser Position. Damit ist die Möglichkeit gegeben, dass auch jede Winkelposition, die nicht in einer der 16 Speicherstellen abgelegt ist, im Auto-Positioniermodus (7.8.1) exakt angefahren werden kann.

Dieses Modul verwendet den so genannten „Speed-Cursor” zur Unterstützung der interaktiven Eingabe, mit der man den gewünschten Endwinkel vorwählt. Dieses GUI-Element zeigt die Größe der Geschwindigkeitsänderung, die Änderungsrichtung und zeigt auch an, ob der Eingabewert im Moment noch weiter erhöht oder vermindert werden kann. Der „Speed-Cursor” wurde bereits in Absch. 7.6.2 ausführlich behandelt.

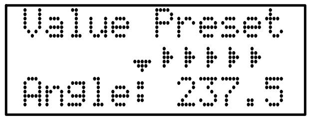


Abbildung 7.16.: Speed-Cursor als Benutzer-Interaktionselement

Nach Vorwahl der gewünschten Winkelposition, kann der Wert in diesem Modus auch über den Auto-Positioniermechanismus konkret „angefahren” und danach im Kalibriermodus (Absch. 7.8.4) als vordefinierte Position gespeichert werden.

Der Programmcode für dieses Modul befindet sich in **mod/set_posn.asm**.

7.8.4. Werte-Definitionsmodul (Kalibrierung)

Die Festlegung des Positions-Referenzwinkels und der vordefinierten Winkelwerte, die zur raschen Positionierung verwendet werden können, oder das Löschen bereits gespeicherter Winkelpositionen wird über dieses Modul kontrolliert.

Die Hauptroutine besteht aus dem Auswahlmenü für die gewünschte Position (0…15), in dieser wird der aktuell vorhandene Winkel als zukünftiger Positionierwinkel für den automatischen Positioniermodus (Absch. 7.8.1 auf Seite 68) gespeichert.

Jeder bereits gespeicherte Wert kann auch wieder gelöscht werden. Dieses Löschen geschieht programmtechnisch durch Speichern des reservierten Wertes **0xFFFF**, der in allen Winkelanzeigen als „**none**” aufscheint.

Eine wichtige und regelmäßig verwendete Unterfunktion ist das Neusetzen des Referenzwinkels, also jener Winkelposition, die als Winkel 0.0° definiert ist, alle anderen Winkelpositionen beziehen sich auf diesen Referenzwinkel, der im GUI als „**RefPos**” bezeichnet wird.

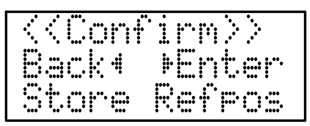


Abbildung 7.17.: Referenzposition neu setzen

Programmtechnisch wird die Referenzposition über die reservierte Position 0 festgelegt.

Da aber auch in diesem Positionsspeicher ein Wert abweichend von 0.0° gespeichert werden kann, erhält man dadurch die Möglichkeit, einen „Referenz-Offset” Winkel zu definieren, der zwar als „Referenz-Position” bezeichnet wird, aber tatsächlich einem von 0.0 verschiedenen Absolutwinkel entspricht.

Der Programmcode für die Kalibrier-Funktionen befindet sich in **mod/calibrate.asm**.

7.8.5. Zurücksetzen/Initialisierung

Während der Entwicklung der Anlage kam es zur Situation, dass die Anlage „festfuhr” und keine Drehbewegung einsetzte. Dies führte nach einiger Zeit zur Zerstörung eines Bauteils, der einen Kurzschluss verursachte. Dieser Kurzschluss bewirkte offensichtlich eine Induktionsspitze, die den Mikrocontroller zerstörte.

Die vermuteten Gründe liegen bei folgenden Hauptursachen, von denen vermutlich einer oder mehrere zugleich aufgetreten sind.

1. Fehlerhafte Montage des mechanischen Teiles (zu hohes oder unregelmäßiges Schrauben-Anzugsmoment)
2. Zu hohe Haftreibung aufgrund der aktuellen Versuchsumgebung (Temperatur, Luftfeuchtigkeit, . . .)
3. Zu hoch eingestellter Laststrom in der Motorsteuerkarte (siehe 5.3)
4. evtl. weitere Ursachen

Dieses Ereignis führte zum Entschluss, softwaretechnische Absicherungen zu implementieren.

Softwaresicherung Analog zu einer elektrischen Sicherung, die beim längeren Überschreiten eines zulässigen Stromes die Stromzufuhr unterbricht, wurde in der automatischen Positionier-Routine ein Mechanismus implementiert, der die Steuerung abschaltet, wenn sich spätestens bei Erreichen des Maximal-Vorgebewertes keine Drehbewegung einstellt (siehe Absch. 7.8.1.2).

Initialisierung und Losreiß-Funktion Wenn der vorhin genannte Zustand eintritt, gibt es nur mehr die Möglichkeit, die Mechanik „händisch” in Bewegung zu setzen oder, dies ist aber vorerst nur eine Vermutung, mittels Software technischer Methoden, die kurze und „schlagartige” Steuerimpulse in verschiedene Richtungen einbringen.

Da für diese Software-Umsetzung weder die Nachvollziehbarkeit gegeben war, noch der Projektplan längere Untersuchungen zuließ, blieb dieser 2. Teil der Programmierung unerledigt.

Lediglich das GUI wurde implementiert, das den Anwender über das Fehlen dieses Features informiert.

Die Programmteile für die Initialisierung sind in der Datei **mod/initcycl.asm** abgelegt.

7.8.6. Informationsmodul

Um Status, Softwarestand, etc. zu hinterlegen, wurde ein Informationsmodul implementiert, das in verschiedenen Bildschirminhalten über Produktversion, Programmierer, Built-Time und Anlagenbetreuer informiert. Diese Inhalte erscheinen zyklisch auf verschiedenen Displayseiten, der Modus kann mit einem beliebigen Tastendruck beendet werden.

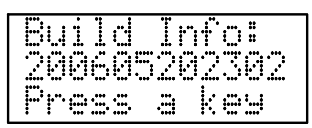


Abbildung 7.18.: Informationsmodus Anzeige

Die Inhalte dieser Texte sind, wie auch (fast) alle anderen Textinformationen, in Form von **.DB Labels** in der Datei **inc/cseg_data.inc** abgelegt. Nur die Built-Time Information wird über ein Assemblermakro aus der aktuellen Systemzeit dynamisch erzeugt.

Der Programmcode für den Info-Modus ist in **mod/info_mod.asm** definiert.

Auszug aus **inc/cseg_data.inc**:

```
; ================================================================
;;; Copyright and author information (info mode)
;;;                  123456789012
COPYRIGHT_TITLE:
  .db $2e,$d2,$c6,$ad,$c3,$ef,$ee,$f4,$f2,$ef,$ec,$2f,$0,$0
  ; >RF-Control<
COPYRIGHT_VERSION: .db "Version 1.1 ",$0,$0
COPYRIGHT_AUTHOR:  .db "G. Reithofer",$0,$0
; ================================================================
```

```
; Serial information
; -----------------------------------------------------------------------
SOFTWARE_BUILD: .db "Build Info: ",$0,$0
SOFTWARE_BUILT: .db "%YEAR%%MONTH%%DAY%%HOUR%%MINUTE%",0x0,0x0
```

7.9. Spezialfunktionen und Fuse-Bits

Mehrere, bei der Implementierung und bei den Tests festgestellte „Unbequemlichkeiten” und vorgebrachte Wünsche des Auftraggebers verlangten nach Zusatzfunktionen, die bis zu diesem Zeitpunkt noch nicht eingeplant waren. Dieser Abschnitt erläutert die Gründe, warum und in welcher Art diverse „Goodies” wie Testfunktionen und Unterstützungen implementiert wurden.

Verursacher für die außerplanmäßigen Erweiterungen waren Punkte, die erst zum Ende der Entwicklung auffielen bzw. als Wunsch zu einem sehr späten Zeitpunkt eingebracht wurden.

1. Bereits während der Entwicklung stellte sich heraus, dass das ständige „Neuprogrammieren” des Controllers mit dem anschließenden Abspeichern der Positionen zum Testen sehr zeitaufwändig ist. Dies würde auch alle Software-Updates betreffen, wenn die Anlage einmal im Betrieb ist und man Updates einspielen müsste.

2. Durch den speziellen „Referenz-Offset” Modus (siehe Absch. 7.8.4) entsteht eine Situation, dass die „normale” Bildschirmausgabe unter Umständen nicht eindeutig genug ist und zu Verwechslungen führen könnte.

3. Schon bei Vorgesprächen zur Implementierung der Anlage an der TU-Graz wurde der Wunsch geäußert, dass die Programmierung so genannter Standardwerte unterstützt werden soll, um einen einheitlichen Start-Zustand mit wenig Aufwand wiederherstellen zu können.

4. Durch die Festlegung auf den Single-Direction Betrieb (siehe 4.3.2) kann nur eine Drehrichtung zum Positionieren verwendet werden. Bei der Handsteuerung (siehe Absch. 7.8.2) kann es jedoch leicht vorkommen, dass man unbeabsichtigt „zu weit” fährt, was eine weitere komplette Rotation benötigt um die „korrekte” Position wieder zu erreichen.

Diese Anforderungen wurden teilweise in Form von Spezialprogrammen implementiert, die man über Tastenbetätigungen (sog. Cheat-Keys [16], in Anlehnung an Computerspiele) während der Startphase der Mikrocontroller-Software bewusst aktivierenen muss, oder über das Setzen von Konfigurations-Registern im Controller umgesetzt.

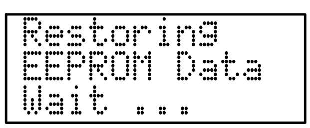


Abbildung 7.19.: Boot Informationsanzeige

Während des Hochfahrens der Steuerung erscheint für ca. zwei Sekunden der in Abb. 7.19 dargestellte Dialog am LCD-Display.

Wird während dieser Bootphase, eine der Funktionstasten gedrückt, wird jeweils ein spezieller Modus aktiviert, der eine der unten beschrieben Aktionen ausführt.

7.9.1. EEPROM-Schutz beim Software-Upload

Der ATMega8 ist auch, was die Programmierung der diversen Speicherbereiche (SRAM, Flash Speicher, EEPROM) betrifft, sehr flexibel und über die so genannten Fuse-Bits konfigurierbar. Dies wurde dazu verwendet, um den EEPROM-Bereich vor dem Beschreiben beim „Hochladen” der Software zu schützen. Das EESAVE Bit des Fuse-High-Byte musste dazu auf 0x0 gesetzt werden [15] Tab. 87, S. 219.

Dieses wurde ebenfalls mit der UISP-Programmer Software (siehe 7.4.2.3) bewerkstelligt und führte zu einer Erleichterung bei der SW-Entwicklung und beseitigte die Notwendigkeit, bereits gespeicherte Werte bei einem Software-Update neu „programmieren” zu müssen. Damit wurde die erste Forderung zur Gänze ohne „Programmierung” im herkömmlichen Sinne erfüllt.

7.9.2. Speicheranzeige Modus

Um dem Anlagenbediener bereits beim Start die Möglichkeit zu geben, sich über die gespeicherten Werte zu informieren, ohne die Werte umständlich über die Funktionstastatur anwählen zu müssen, wurde diese Funktion implementiert.

Alle 16 gespeicherten Winkelpositionen werden am Display kurz angezeigt, danach kann die der Anlage wieder wie gewohnt bedient werden.

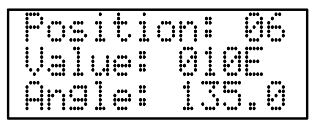


Abbildung 7.20.: Speicheranzeige Modus

Diese Funktion wird aktiviert, wenn während der oben genannten Anzeige die linke Taste betätigt wird.

7.9.3. Automatische Standardinitialisierung

Zur einfachen Vorbelegung der gespeicherten Drehwinkel mit Standardwerten wurde folgender Mechanismus implementiert.

Die Winkelspeicher-Initialisierung belegt die ersten 8 Speicher mit den jeweiligen 45° Schritten des Kreisquadranten, die Werte 8-15 werden gelöscht und die aktuelle Winkelposition wird als Referenzposition abgespeichert. Am Bildschirm wird die Nummer jeder Position, der geschriebene und wieder eingelesene Wert kurz angezeigt (Abb. 7.21).

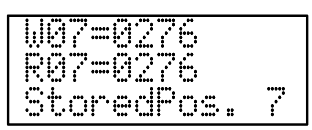


Abbildung 7.21.: Standard-Initialisierung

Dieser Mechanismus wird aktiviert, wenn während der Boot-Up Phase die mittlere Taste gedrückt wird.

7.9.4. Aktivierung des Zwei-Richtungsbetriebes

Wie vorhin unter Punkt 4. erwähnt, ist die händischen Kontrolle der Anlage im Zwei-Richtungsbetrieb in der Regel einfacher zu bewerkstelligen, wenn man die Bewegung zu spät gestoppt hat und man dann ein paar Schritte „zurückfahren” kann.

Mit Drücken der rechten Taste während der o. a. Boot-Anzeige, wird dieser Modus steuerungsmäßig aktiviert. Damit ist die händische Steuerung über die Funktionstasten (Absch. 7.8.2 Handsteuermodul) in beiden Richtungen möglich.

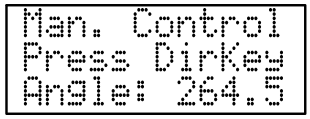


Abbildung 7.22.: Manueller Zwei-Richtungsbetrieb

Nach Aktivierung dieses Modus, erfolgt jedesmal zu Beginn des Handsteuermodus die Abfrage nach der gewünschten Richtung (Abb. 7.22), die mit der „Rechts”-Taste (normale Drehrichtung) oder „Links”-Taste (umgekehrte Drehrichtung) vorgewählt werden kann.

> **Wichtige Bemerkung:**
> Die Software ist nicht dafür ausgelegt, in beiden Richtungen korrekt zu zählen, daher ist in diesem Modus eine „nummerische Winkelpositionierung” nicht möglich, da die Winkelschritte immer nur in einer Richtung gezählt werden. Nach einer händischen Positionierung in der „Gegenrichtung” muss deshalb die Referenzposition wieder neu gesetzt werden.

8. Systemintegration

Dieses Kapitel beschreibt in kurzen Worten, was notwendig war, um die Anlage vom „Schreibtisch" des Autors in das „Labor" der TU-Graz zu bekommen und welche Hürden dafür überwunden werden mussten.

8.1. Basisarbeiten und Controllerprogrammierung

Da der mechanische Teil der des Anlagenumbaus bereits 2005 umgesetzt wurde (siehe Absch. 2.1.2) war die gesamte „mechanische Hardware" bereits zur Konzepterstellung der Steuerung fertig. Der elektrische und elektronische Teil der Anlage wurde nach den im Kapitel 5 diskutierten Kriterien bald festgelegt und die Komponenten mussten nur noch beschafft und gefertigt werden.

Die ersten Programmteile (das sind die Low-Level und Basis-Routinen) waren bald fertig gestellt, einzig die Programmierung der Tastaturfunktionen nahm einen längeren Zeitraum in Anspruch, da diese sehr umfangreich ausgefallen sind (siehe dazu auch Absch. 7.6.2).

Danach wurde bald klar, dass die weitere Entwicklung nur stattfinden kann, wenn man direkt mit den Aktuatoren und den Messsignalen arbeiten kann.

8.2. Hardwareintegration

In diesem Zeitraum war die I^2C-Schnittstellenkarte bereits in einer ersten Version fertig und der Motor mit dem Winkelschrittgeber in die Mechanik eingebaut. Der gesamte Aufbau war auf einer einzigen 4mm starken Stahlplatte der Versuchanlage möglich (siehe Absch. 2.1.2), die die Abmessungen 740x650mm hatte und mit der gesamten Mechanik ca. 15 kg wiegt.

Diese Platte inkl. Aufbau konnte in das eingene Büro transportiert werden, wo der Rest der Softwareentwicklung quasi „am Schreibtisch" stattfand (Abb. 8.1).

Abbildung 8.1.: Entwicklung am Schreibtisch

8.2.1. I^2C-Controllerplatine

Die Entwicklung der I^2C-Schnittstellenkarte durchlief mehrere Zyklen.

In der ersten Version sollte die Spannungsversorgung des D/A-Wandlers noch über eine Zenerdiode sicher gestellt und der Operationsverstärker sollte nahe der Mittelspannung der 24 V Versorgung betrieben werden, was zu einem linearen Verhalten der Steuerspannung geführt hätte. Die Annahme, dass es sich beim Steuersignal-Eingang der Motorkontrollerkarte um einen galvanisch von der Stromversorgung getrennten Differenzeingang handelt, wurde beim ersten Test widerlegt. Es stellte sich heraus, dass der Steuerpannungs-Eingang der Motorkontrollerkarte dieselbe Masseleitung verwendet wie die Stromversorgung des Motors selbst. Diese Situation und die Festlegung, das bereits vorhandene 24 V Netzteil zu verwenden, führte zu den Überlegungen für die Stromversorgung, nach Absch. 6.3. Die Möglichkeit, den gesamten 5 V-Teil mit einer Zenerdiode auf einen höheren Spannungspegel zu heben wurde diskutiert, aber auf Grund der Tatsache, dass die bestehende Platine nicht mit wenig Aufwand auf die neue Anforderung umgebaut hätte werden können, wurde diese Vorgangsweise wieder verworfen, um den Fertigstellungstermin nicht noch weiter nach hinten verschieben zu müssen.

Dies führte letztendlich zur Umsetzung der in Absch. 6.3 vorgestellten Lösung für die Stromversorgung.

Mit ein paar Lötbrücken und Leitungsunterbrechungen konnte die bestehende Karte für die Entwicklung weiter verwendet werden, parallel dazu wurde die neue Platine produziert und nach Fertigstellung in der weiteren Entwicklung eingesetzt.

8.2.2. Implementierung der Winkelmessung

Ein weiteres Integrationsproblem ergab sich daraus, dass der Winkelschrittgeber nicht die geforderten und in den Datenblättern angegebene Stromstärke vom 20m A lieferte. Vor Einbau des dritten Drehwinkelgebers wurde an einem Testaufbau endgültig nachgewiesen, dass der Ausgang des Lichtschranken zwar die exakten Spannungspegel 0 V und 5 V aufwies, jedoch nur eine max. Stromstärke von 0.5 mA lieferte. Damit erreichte dieser Bauteil nicht die Signalstromstärke zum „Treiben" der Optokopplereinganges der **halvedDISC**.

Um die notwendige Signalstärke zu erreichen, wurde dem Drehwinkelgeber die in Abb. 8.2 dargestellte einfache Transistorverstärkerstufe nachgeschaltet, was auch dieses Problem zuverlässig löste.

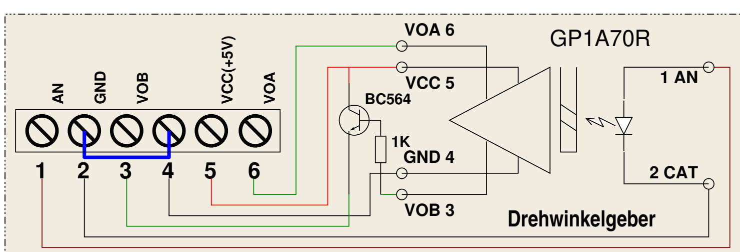


Abbildung 8.2.: Korrekturschaltung zur Pegelanpassung

Der Transistor mit Widerstand wurden direkt auf der Anschlussplatine des Drehwinkelgebers am Elektromotor untergebracht (siehe Abb. 5.5 auf Seite 27).

8.2.3. Weitere Defekte und Störfälle

Die Zerstörung des Mikrocontrollers auf Grund einer Überspannung (siehe Absch. 7.8.5) führte kurz vor der geplanten Inbetriebnahme zur einer zusätzlichen Verzögerung von ca. zwei Wochen, es musste eine neue **halvedDISC** angeschafft werden.

Letztendlich fiel auch noch die Motorsteuerkarte aus, was den Projekt-Fertigstellungstermin eine weitere Woche nach hinten verlagerte. Der Austausch erfolgte als Garantieleistung kostenlos.

9. Inbetriebnahme und Test

Nach Fertigstellung der Software und des Hardware-Aufbaus musste die gesamte Elektronik in die bestehende Anlage integriert werden.

Dieses Kapitel befasst sich mit den notwendigen Erweiterungen und Tätigkeiten, um die „neue" Hardware zu implementieren und die geforderten Funktionen sicherzustellen. In diesen Bereich fallen auch die Einführungsprobleme und Störfälle, die die „normale" Funktionalität negativ beeinflussten und damit unbrauchbar machten oder zumindest unakzeptabel einschränkten. In solchen Fällen war eine Lösung zu suchen und diese umzusetzen.

9.1. Mechanische Umbauten

Die mechanische Inbetriebnahme bestand aus dem Abbau des Anlagenkopfes der bestehenden Anlage, die großteils als Schweißkonstruktion ausgeführt war. Die Demontage der bestehenden Anlage benötigte ca. zwei Tage, das Einbauen der neuen Zwischenplatte und die Fertigstellung, die dieses Mal als mechanische Schraubkonstruktion ausgeführt ist, war nach ca. einem Arbeitstag erledigt. Dieser Anlagenaufbau stellt damit auch eine Erleichterung für etwaige zukünftige Umbauten dar.

Die nach dem Umbau zu Beginn auftretende Leckage zeigte allerdings, dass sehr gründliche Abdichtung aller Komponenten erforderlich ist, um die geforderte Dichtheit zu erreichen. An der Drehvorrichtung wurden bisher keine Dichtheitsprobleme festgestellt.

9.2. Verkabelung und Bedienkonsole

Mit dem Einbau der Steuerungs-Elektronik und Antriebselektrik wurde auch die in Absch. 6.2 beschriebene Verkabelung montiert. Um die Komponenten möglichst einfach wieder demontieren zu können, wurde ein 7-poliger Modulstecker (in Abb. 9.1, oben in der Mitte des Steuergehäuses sichtbar) verwendet, der, abgesehen von den Stromversorgungen, die gesamte „Steuerkonsole" von der restlichen Anlage trennt.

9.3. Störsignale und deren Eliminierung

Bei der Inbetriebnahme der Anlage trat plötzlich ein Messfehler auf, der dazu führte, dass die Drehbewegung immer etwas vor der Endposition stoppte. Die Größenordnung des Fehlers lag bei ca. 4° bezogen auf eine Gesamtumdrehung und war bereits mit freiem Auge sichtbar.

Die Analyse des Messsignals des Drehwinkelgebers mit einem Oszilloskop ließ rasch die Ursache erkennen.

Am Verkabelungsplan im Anhang B.3 ist ersichtlich, dass sowohl die Signalleitungen (Drehwinkelgeber) als auch die Motorsteuerleitungen in einem Kabelstrang eingebunden sind. Diese Situation und wahrscheinlich auch die Nähe der Kabel innerhalb des Bediengehäuses führten in der fertig montierten Anlage zum dem Effekt, dass die 18-kHz PWM-Signale (siehe dazu Abschnitt 5.3) in die Messimpulse des Drehwinkelgebers einstrahlten, die in der Größenordnung unter 100 Hz liegen. Dies führte dazu, dass beim Übergang

Abbildung 9.1.: Verkabelung, Demontagemöglichkeit

vom High- in den Low-Pegel immer wieder einzelne Steuerimpulse mitgezählt wurden, was zum vorhin beschriebenen Fehler führte.

Das Parallelschalten eines Elektrolytkondensators mit der Kapazität von $1\mu F$ in der Messignalleitung behob auch dieses Problem zuverlässig.

9.4. Spannungsregler und I^2C-Schnittstellenplatine

Nach ca. zwei Wochen Betrieb kam es abermals zu einem Zählfehler, aber diesmal mit dem gegenteiligen Effekt, es gingen Zählimpulse verloren, der Filterkorb kam damit „zu spät" zum Stillstand. Als weiterer Effekt trat zu Tage, dass die Fehlerhäufigkeit mit der Anlagenlaufzeit zunahm, nach einem Anlagestillstand wurde kurzzeitig wieder korrekt gezählt. Im Fehlerfall trat sich die Abweichung nach kurzer Zeit (< 1 Std.) so deutlich zu Tage, dass man sie auch mit freiem Auge erkennen konnte.

Abermals wurde versucht mit dem Oszilloskop den Fehler zu lokalisieren und dabei stellte sich heraus, dass die Impulse immer wieder in unregelmäßigen Abständen kurzzeitig ausfielen.

Der anfängliche Verdacht, dass der Winkelgeber defekt sei, stellte sich als nicht richtig heraus, denn während einer längeren Beobachtungsphase fiel die Winkelmessung komplett aus. Beim Ausbauen der Platinen aus dem Controllergehäuse wurde die Fehlerursache sichtbar.

Der Spannungsregler auf der I^2C-Schnittstellenplatine hatte sich gelockert, was vorerst nur zu einem Haarriss in einer Leiterbahn führte, später wurde diese Leiterbahn komplett unterbrochen und führte zum beschriebenen Totalausfall.

Die Ursache für diesen Effekt lag in der mechanischen Ausführung der im Gehäuse eingebauten Komponenten. Die Mikrocontrollerplatine ist räumlich über der Schnittstellenplatine angeordnet und drückte auf den Kühlkörper des darunterliegenden Spannungsreglers. Mit jedem Tastendruck bei der Bedienung der Drehwinkelsteuerung wurde immer eine leichte Kraft auf den Kühlkörper übertragen und führte nach längerer Bedienung zu der dargestellten Situation.

Der Spannungsregler wurde von der Platine entfernt und an anderer Stelle im Bediengehäuse montiert und über flexible Litzenkabel mit der Steuerungsplatine verbunden.

9.5. Überprüfung der Positioniergenauigkeit

Dieser Abschnitt beschäftigt sich mit dem Nachweis, dass die Drehwinkelsteuerung tatsächlich die geforderte Positionier- und Wiederholgenauigkeit (siehe 2.2) erreicht. Je nach Entwicklungsstand wurden dafür unterschiedliche Methoden angewendet.

9.5.1. Entwicklungsstadium

Während der Entwicklung „im Büro” konnte die Hardware mit den rotierenden Teilen direkt eingesehen werden (siehe Abb. 5.5 auf Seite 27).

Die primäre Forderung nach der Wiederholbarkeit von Positionen konnte an Hand von angebrachten Markierungen am rotierenden und feststehenden Teil der Drehvorrichtung visuell kontrolliert werden (siehe dazu auch die Vorgaben zur Genauigkeit in Absch. 4.4 auf Seite 23).

9.5.2. Inbetriebnahmetest

Da die Versuchsanlage prinzipbedingt „optisch zugänglich” ist, ist ebenfalls die Wiederholgenauigkeit visuell kontrollierbar. Da normale Markierungen jedoch rasch vom Staub der Versuchsanlage verdeckt werden würden, wurden 2 Stahlstifte (Nägel) im rechten Winkel so angebracht, dass eine visuelle Kontrolle möglich war (siehe Abb. 9.2).

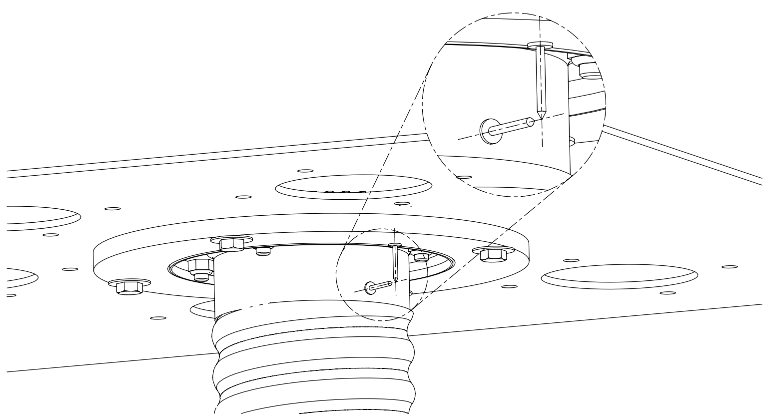


Abbildung 9.2.: Markierungen an der Drehvorrichtung

9.5.3. Versuchs-Messreihe

Um „quantifizierbare” Ergebnisse zu erzielen, wurden mit den bestehenden Messkameras (siehe Anlagenbeschreibung in Absch. 1.2) Aufnahmen gemacht, die digital ausgewertet wurden.

Im Anhang G sind die 14 Abbildungen der in Tabelle G.1 angegebenen Kamera-Images abgebildet, wobei jeweils ein Bild der unteren Kamera 1 neben dem der oberen Kamera 2 dargestellt ist. In Tabelle 9.1 sind die Einstellwerte der Image-Processing Software (siehe Absch. 7.4.2.5) angegeben, mit denen die beiden Abbildungen aufbereitet wurden.

Zuerst wurden die Grauwerte um 45 Punkte erhöht (aufgehellt), der untere Threshold Wert (Grau-Schwellwert) wurde auf 80 eingestellt, was dazu führt, dass alle Bildpunkte, die den Grauwert 80 unterschreiten schwarz dargestellt werden. Dies führt zu dem abrupten Übergang, der in der Differenz-Berechnung (Abb. 9.3 rechts), der als „scharfer” Helligkeitsübergang aufscheint.

Tabelle 9.1.: Image-Processor Einstellungen

Parameter	Wert
Image name	frame_0006_depth_006_im1 frame_0007_depth_007_im1
Size	263x145
ID	imag27,image28
Inverse result mode	No
Grey shift value	45
Contrast stretch	0
Low threshold value	80
High threshold value	0

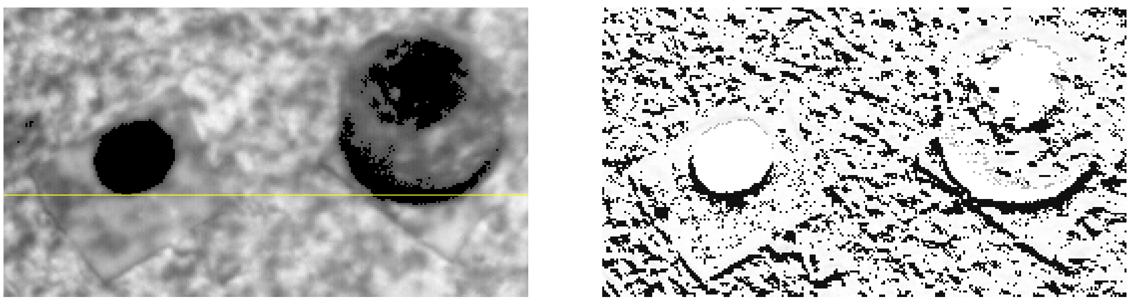


Abbildung 9.3.: Preprocessed Image und Image difference processing

Die optische Beurteilung dieses Differenzbildes zeigt, dass zwar eine vertikale Lageverschiebung von ca. 4-6 Bildpunkten auftritt, waagrecht ist keine Abweichung erkennbar. Die vertikale Abweichung stammt von der Ungenauigkeit der vertikalen Positionierung der Messkameras, die für jedes Bildpaar neu positioniert wurden, eine horizontale Abweichung (in Bewegungsrichtung der Drehvorrichtung) würde die zu untersuchende Winkelabweichung darstellen.

Da die Größe eines Bildpunktes auf der Filteroberfläche ca. 1/40 mm entspricht ist die Genauigkeitsvorgabe aus Abschnitt 4.4 auf Seite 23 mit hinreichender Sicherheit erfüllt.

Weiteren optische Analysen ergaben ebenfalls, dass keine Messabweichung mit einem Fehler von mehr als 1…4 Pixeläquivalenten ($\leq$0.1 mm) festgestellt werden konnte.

10. Zusammenfassung und Ausblick

Dieses Kapitel soll ein Resümee ziehen über das in dieser Diplomarbeit beschriebene Projekt und Denkanstöße geben, wie die bestehende Entwicklung besser und in anderen Bereichen eingesetzt werden könnte oder welches Weiterentwicklungs-Potenzial im bestehenden Produkt steckt.

Welche weitere Funktionen könnten mit geringem Aufwand in Form zusätzlicher Programmierung implementiert werden?

10.1. Ungenutzte bereits konzipierte Features

Die Anlage wurde bereits im Hinblick auf absehbare Erweiterungen konzipiert, außerdem ist die offene Struktur auch gut dazu geeignet, die Anlage um neue Funktionen mit wenig Implementierungs-Aufwand zu erweitern.

Durch die bisher freiwillig auferlegten Beschränkungen (z. B. Ein-Richtungsbetrieb) sind prinzipbedingt bereits einige Erweiterungen mit definierten Funktionen eingeplant, die nur mehr softwaremäßig implementiert werden müssen.

Im Konkreten sind dir folgende Funktionen „unfertig" geblieben:

1. Zwei-Richtungsbetrieb in der gesamten Steuerungslogik (siehe dazu auch 4.3.2)
2. Verwendung des Helligkeits-Sensors zur Steuerung der LCD-Hintergrundbeleuchtung
3. Verwendung der Echtzeituhr (RTC) zur Datums- und Uhrzeitanzeige

10.2. Ausnutzung der verwendeten Hardware

Weitere noch in der Hardware vorhandene Fähigkeiten wurden bisher überhaupt nicht beachtet, wie z. B. Verwendung der Stromsparfunktionen des ATMega8.

Der verwendete I^2C-Schnittstellenbaustein PCF8591 wird nur als Digital-Analogwandler betrieben. Die vier im Chip integrierten Analog-Digitalwandler könnten ebenfalls verwendet werden, um die tatsächlich vorgegebenen Steuerspannungen oder Betriebszustände an der Anlage wieder zu messen und an den Controller, quasi zur „Kontrolle" wieder zurück zu liefern. Dieser Aspekt müsste auf jeden Fall berücksichtigt werden, wenn es sich um eine sicherheitskritische Anwendung handelt.

In der aktuellen Lösung ist die Implementierung weiterer Funktionen prinzipiell vorgesehen, die Controllerhardware ist noch lange nicht „ausgelastet" und man wird bei den Entwicklungen an der Versuchsanlage überlegen, ob und wieweit die bestehende Hardware sinnvoll genutzt werden kann.

10.3. Softwareoptimierung

Selbstverständlich steckt, wie in jeder Software, noch Verbesserungs-Potenzial, das bei einer Weiterentwicklung in jedem Fall genutzt werden muss.

So ist der Programmspeicher bereits jetzt schon ziemlich ausgeschöpft und würde beim Implementieren weiterer Funktionen die 8Kb-Grenze des Programmspeichers bald überschreiten. Ein Assembleraufruf stellt z. Zt. folgende Situation dar.

```
gerhardr@hpnb:~/sw/AVR/rot_facility>make
avra --define DEBUG=1 --listfile RF_Control.list --includedir ../inc --includedir inc
 --includedir lib  --includedir mod RF_Control.asm
AVRA: advanced AVR macro assembler Version 1.0.1 Build 113 (12. February 2004)
... deleted ...
Pass 1...
Pass 2...
done
Segment usage:
   Code      :      3798 words (7596 bytes)
   Data      :        44 bytes
   EEPROM    :        34 bytes
Assembly complete with no errors.
```

Optimierungspotenzial steckt noch im Bereich der Überarbeitung der Makroverwendungen (siehe dazu Absch. 7.5.4) und in der Zentralisierung mehrfach verwendeter Codeteile.

Aus Erfahrung muss außerdem gesagt werden, dass ein generelles Code-Reviewing immer zur Verbesserung und Optimierung von Sourcecode beiträgt, allerdings können seriöse Aussagen über konkretes Verbesserungspotenzial erst nach diesem Review getätigt werden.

A. Technische Daten

In diesem Teil des Anhangs sind die technischen Daten der verwendeten Komponenten aufgelistet, mit Hersteller und Lieferantenangaben, soweit bekannt.

Detaillierte Hersteller- und Lieferantenangaben werden pro Firma jeweils nur bei ihrem ersten Vorkommen aufgeführt, danach wird nur mehr der Firmenname angegeben.

A.1. halvedDISC Mikrocontroller

Tabelle A.1.: halvedDISC

Technische Daten		
Prozessor	ATMEL	ATMega8
Taktfrequenz	4	MHz
ISP-Flash (Prog.)	8	kByte
SRAM (Daten)	1	kByte
ISP-EEPROM	512	Byte
Datenanzeige	LCD	12x3
Signalanzeigen	LED	2x
Timer/Counter	8/16-Bit	1x8, 2x16
Bedienelemente	Tasten	3
Optokoppler	Eingang	1
I^2C-Bus	Ein-/Aus	1x
Relais/Umschaltbar	2-fach	1x

Vertrieb:
IMM Elektronik GmbH, Mittweida
Leipziger Strasse 32
D-09648 Mittweida

IMM Homepage: [http://www.imm-gruppe.de]

Application Center Microcontroller: [http://www.htwm.de/acmc]

HalvedDisc Homepage: [http://halveddisc.htwm.de]

A.2. Gleichstrom-Getriebemotor

Tabelle A.2.: Gleichstrommotor

Technische Daten		
Nennspannung	24	V
Nennleistung	12	W
Nennstrom	2.5	A
max. Stromaufnahme	11	A
Nenndrehzahl	29	min^{-1}
Nenndrehmoment	4	Nm
Anzugsdrehmomment	22	Nm
Übersetzung	1:55	
Gewicht	1.4	kg

Hersteller:
Fa. Bosch
Lieferant:
Wolfgang Graf - Elektromotoren
Albertgasse 11
A - 1080 Wien

Wolfgang Graf [http://www.elektromotor.at]

A.3. 1-Quadranten PWM-Regler

Tabelle A.3.: Gleichstromsteller GS24S

Technische Daten		
Versorgungsspannung	12-48	V
Ankerspannung	0-32	VDC
max. Ankerstrom	10	A
max. Leistung	ca. 160	W
Taktfrequenz	18	kHz
Formfaktor	<1.05	
Drehzahl-Sollwertvorgabe	0-10	V

Hersteller:
EPH Elektronik
Produktions- und Handelsgesellschaft mbH
Rudolf-Diesel-Straße 18
D-74354 Ottmarsheim
Lieferant:
Wolfgang Graf - Elektromotoren

EHP Elektronik Homepage [http://www.eph-elektronik.:de]

A.4. Richtungsumschalt-Leistungsrelais

Tabelle A.4.: Leistungsrelais RT424024

Technische Daten		
Spulenkontakte	2	
Spulenspannung	24	VDC
Schaltstrom	8	A
Wechselkontakte	2	
Printausführung	ja	

Hersteller:
Schrack Energietechnik GmbH
Lieferant:
Conrad Electronic
Kärntner Straße 228
A-8053 Graz

Schrack Österreich [http://www.schrack.at]

Conrad Electronic [http://www.conrad.at]

A.5. Lichtschranken-Drehwinkelgeber

Tabelle A.5.: Drehlwinkelgeber GP1A71R

Technische Daten			Bemerkung
Forwardspannung	1.2	V	Eingang (1,2)
Forwardstrom	20	mA	
Betriebsspannung	4.5…5.5	V	
High Ausgangsstrom	20	mA	wurde nie erreicht - siehe Kap. 8
Low Ausgangsstrom	8	mA	wurde nie erreicht - siehe Kap. 8
max. Transferfrequenz	10	kHz	

Hersteller:
SHARP Microelectronics
Lieferant:
Conrad Electronic

Sharp Corporation [http://sharp-world.com]

A.6. I²C-Digital/Analogwandler

Tabelle A.6.: I²C-D/A Wandler PCF8591

Technische Daten			Bemerkung
Betriebsspannung	2.6…6.0	V	Single Power
Hardwareadress Bits	3		0x90…0x9e
Abtastrate	I²C-Busgeschw.		
Analogeingänge	4		
Analogausgang	1		
A/D Konverterbereich	8	bit	0…255

Hersteller:
Philips Semiconductors
Lieferant:
Conrad Electronic

Philips Semiconductors [http://www.semiconductors.philips.com]

A.7. Operationsverstärker

Tabelle A.7.: Operaktionsverstärker LM741

Technische Daten		
Betriebsspannung	±18	V
Leistungsaufnahme	500	mW
Differential-Eingangsspng.	±30	V
Eingangsspannung	±15	V
Eingangsspannungsoffset	2	mV
Eingangsoffset Korrektur	±15	mV
Flakenanstiegsrate	0.5	V/µs
Eingangswiderstand	2	$M\Omega$

Hersteller:
National Semiconductor
Lieferant:
Conrad Electronic

National Semiconductor [http://www.national.com]

A.8. Mikrocontroller-Display

Tabelle A.8.: LCD-Display EA7123-I2C

Technische Daten		
Anzeigengröße	3x12	
Zeichenmatrix	5x7	
Controllerchip	PCF21165	on Board
Zeichenhöhe	4.7	mm
Sichtfenster	32x17	mm

Hersteller:
ELECTRONIC ASSEMBLY GmbH
Lochhamer Schlag 17
D-82166 Gräfelfing bei München

In halvedDISC integriert

Electronic Assembly [http://www.lcd-module.de]

A.9. Temperatursensor

Tabelle A.9.: I²C-Temperatursensor LM75

Technische Daten		
Betriebsspannung	$\pm 3.0 - 5.5$	V
Stromaufnahme (typ.)	250	µA
Stromaufnahme (max.)	1	mA
Max. Abweichung $-55 - 125$°C	± 3	°C
Max. Abweichung $-25 - 100$°C	± 2	°C
Temperaturformat	9	Bit

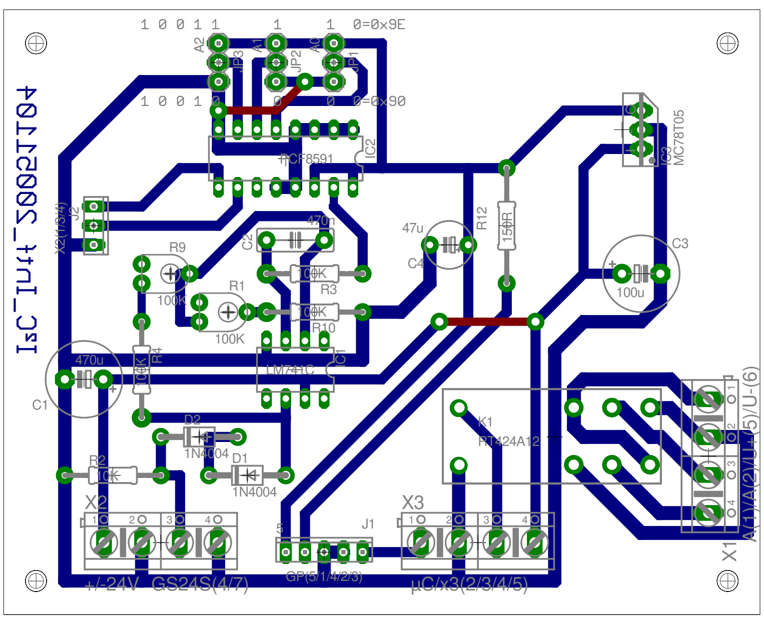
Hersteller:
National Semiconductor

In halvedDISC integriert

B. I^2C-Interface Platine

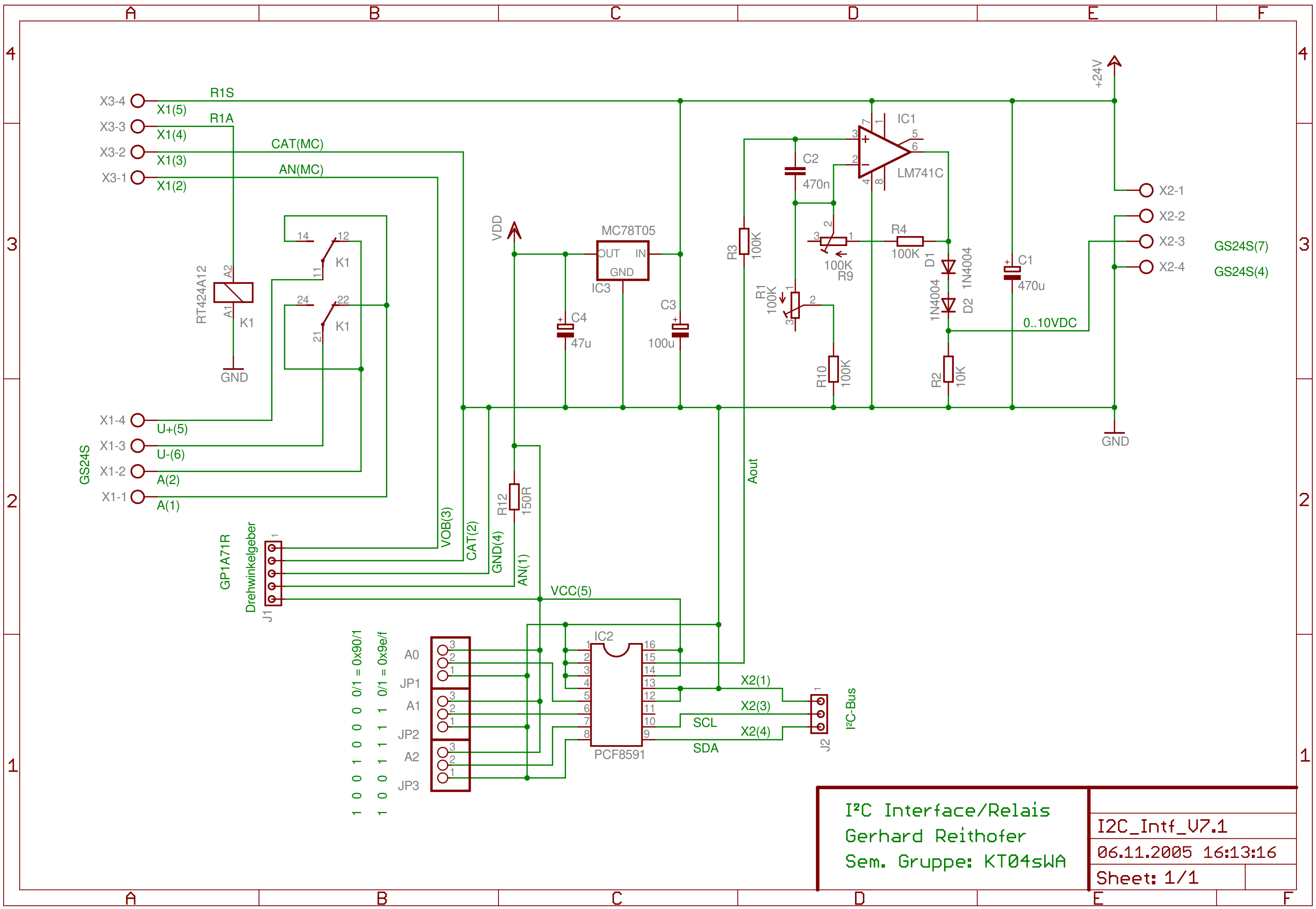
In diesem Teil des Dokumentes sind die Entwicklungsdaten der I^2C-Schnittstelleplatine aufgeführt. Diese bestehen im Detail aus dem Schaltungsschema, dem Platinenlayout (einseitig) und aus der zentralen Modulverkabelung.

B.1. Layout



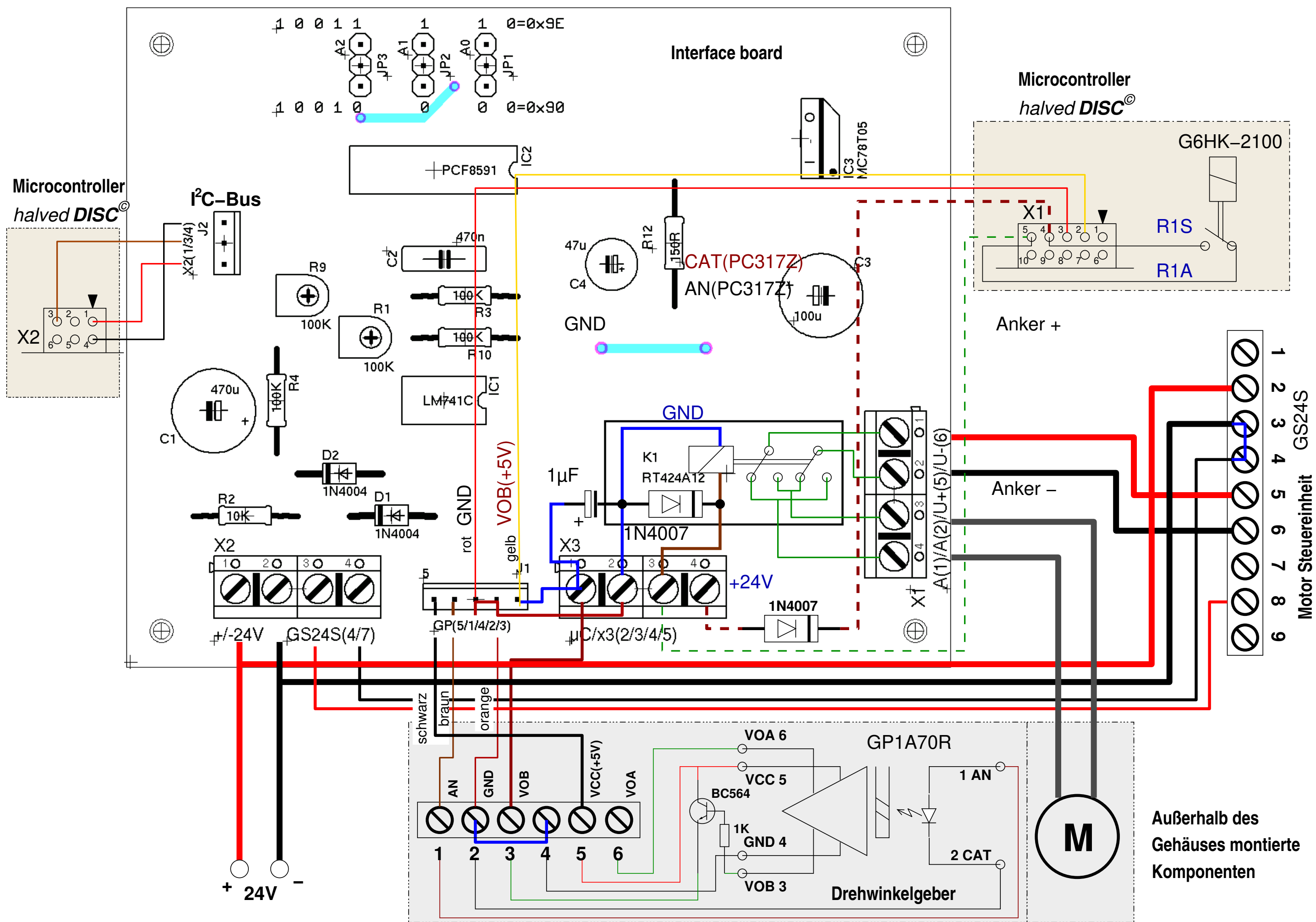
Layout Bauteilseite mit Bestückung

B.2. Schema



I²C Interface/Relais
Gerhard Reithofer
Sem. Gruppe: KT04sWA
I2C_Intf_V7.1
06.11.2005 16:13:16
Sheet: 1/1
X3-4
X3-3
X3-2
X3-1
X1(5)
X1(4)
X1(3)
X1(2)
R1S
R1A
CAT(MC)
AN(MC)
RT424A12
K1
GND
X1-4
X1-3
X1-2
X1-1
U+(5)
U-(6)
A(2)
A(1)
GS24S
GP1A71R
Drehwinkelgeber
J1
VOB(3)
CAT(2)
GND(4)
AN(1)
VCC(5)
R12
150R
VDD
MC78T05
IC3
C4
47u
C3
100u
R3
100K
R1
100K
C2
470n
IC1
LM741C
R9
100K
R4
100K
R10
100K
D1
1N4004
D2
1N4004
R2
10K
C1
470u
+24V
X2-1
X2-2
X2-3
X2-4
GS24S(7)
GS24S(4)
0..10VDC
Aout
IC2
PCF8591
A0
A1
A2
JP1
JP2
JP3
0/1 = 0x90/1
0/1 = 0x9e/f
X2(1)
X2(3)
X2(4)
SCL
SDA
J2
I²C-Bus

B.3. Verkabelungsplan



C. Software Listings

In diesem Anhangteil werden die einfachen Programmlistings abgedruckt, die Assemblerroutinen, die einen Umfang von über 7000 Zeilen annahmen und die Zusatzprogramme LCDEdit und ImageProc sind nur auf Datenträger verfügbar.

C.1. Tracingsoftware Shell

```
#!/bin/sh
#
# AutoTracer startup script
#
IMG_INPUT=I2C_Intf_V7.png
IMGMIRROR=I2C_Intf_V7m.png
DXFOUTPUT=I2C_Intf_V7.dxf
echo "Creating mirrored image..."
pngtopnm $IMG_INPUT|pnmflip -leftright|pnmtopng>$IMGMIRROR
echo "Tracing mirrored image..."
autotrace -report-progress \
          -output-format=dxf \
          -corner-surround=1 \
          -filter-iterations=1 \
          -background-color=000000 \
          -corner-always-threshold=30 \
          -output-file=$DXFOUTPUT \
          -dpi=600 $IMGMIRROR
rm $IMGMIRROR
```

C.2. Drill-Data CAM Präprozessor

```
#!/bin/sh
#
# Simple AWK script to convert EAGLE drill data files
# into metric macro input files for VariCAD.
#  (c) Gerhard Reithofer, 2005-08-11
#
LANG=C           # for correct decimal delimiters,
MIRROR=1         # usually we get the top view whereas
                 # drill data may com from bottom view
if [ $MIRROR = 1 ]
then SCALE_X=-0.00254
     OFFSETX=100.474
else SCALE_X=0.00254
     OFFSETX=-0.16
fi
SCALE_Y=0.00254
OFFSETY=-0.169
if [ -z "$1" ] ; then
  echo "Usage: `basename $0` drd-file"
  exit 1
fi
INP_FILE=$1
if [ ! -f "$INP_FILE" ] ; then
  echo "drd-file '$INP_FILE' does not exist."
  exit 2
```

```
fi
rm -f tool*.dat
awk -v scx=$SCALE_X -v scy=$SCALE_Y \
    -v ofx=$OFFSETX -v ofy=$OFFSETY \
'BEGIN{
  mode="";
  glb_tool_cnt=0;
  cur_tool_idx=0;
  max_tool_idx=0;
}
function get_tindex(idx) {
  for (i=1;i<=max_tool_idx;i++) {
    if (glb_tool_num[i] == idx) break;
  }
  return i;
}
{
  iso=toupper($1);
  if (iso == "") next;
  mode=substr(iso,1,1);
  if (mode == "T") {
    idx=substr(iso,2,2);
    c_pos=index(iso,"C");
    if (c_pos) {
      siz=substr(iso,5);
      cur_tool_idx=get_tindex(idx);
      glb_tool_siz[cur_tool_idx]=siz;
      glb_tool_num[cur_tool_idx]=idx;
    } else {
      cur_tool_idx=get_tindex(idx);
      cur_tool_siz=glb_tool_siz[cur_tool_idx];
print "Tool change (" idx "): " cur_tool_idx "-" cur_tool_siz;
      out_file = "tool-" cur_tool_siz ".dat";
      system("rm " out_file " 2>/dev/null");
      system("touch " out_file);
    }
    if (cur_tool_idx>max_tool_idx)
      max_tool_idx=cur_tool_idx;
  } else if (mode == "X") {
    d_len = length(iso);
    y_pos = index(iso,"Y");
    x_len = y_pos-x_pos-2;
    cur_x = substr(iso,2,x_len);
    cur_y = substr(iso,y_pos+1);
    # Exchgange X and Y values because the drill data
    # was created in mirror mode
    print scx*cur_x+ofx " " scy*cur_y+ofy >> out_file;
  } else if (mode == "G") {
    g_code = substr(iso,2);
  } else if (mode == "M") {
    m_code = substr(iso,2);
  } else if (mode == "%") {
    comment = substr(iso,2);
  } else {
    print "# Unexpected line: " iso;
  }
}' $INP_FILE
```

C.3. CAD Import und Plotmacro Erstellung

```
#!/bin/ksh
#
###
### necessary scale factor: (1/dpi*254) (eg. 1/300*25.4=.08458)
###
```

```
# MODE=layer
# MODE=draw
# MODE=plot
MODE=""
if [ -z "$1" ] ; then
  echo "No working mode specified!"
  echo "Usage: `basename $0` mode"
  echo "Valid modes are: layer, draw, plot or all."
fi
MODE=$1
DATA_DIR=~/eagle/rot_facility
MACRODIR=/cad/VariCAD/Macros
# ==========================================================
#                macro header
# ==========================================================
macro_header()
{
echo "Creating VariCAD macro '$MACROFIL'...\c"
echo "# macro $MACRONAM
# # of values;ortho;step;aperture;x step; y step;X win;Y win
Set;7;1;0;4;1;1;869;626
# command: als
CmdN;+;394
#  Taste ; Aus
Panel;+;0;1;1;;
#  Taste II ; OK
Panel;+;0;3;1;;" >> $MACROFIL
}
# ==========================================================
#                layer macro
# ----------------------------------------------------------
macro_layr()
{
    echo "
# included command: lay
ICmdN;+;226
#  Taste II ; Neu
Panel;+;0;2;1;;
#  Taste ; $LAYER_CNT.
Panel;+;1;$LAYER_CNT;1;;
>Text;1;l$DIMENS
#  Taste II ; OK
Panel;+;1;18;1;;
>Text;1;l$DIMENS
#  Taste II ; Zuruck
Panel;+;0;7;1;;" >> $MACROFIL
  LAYER_CNT=`expr $LAYER_CNT + 1`
}
# ==========================================================
#                data import macro
# ----------------------------------------------------------
macro_draw()
{
echo "
# included command: lay
ICmdN;+;226
#  Textwahl
Panel;+;0;1;$LAYER_CNT;l$DIMENS
# command: gra
CmdN;+;2
File;+;/home/gerhardr/eagle/rot_facility/$DATA
BS;File" >> $MACROFIL
  LAYER_CNT=`expr $LAYER_CNT + 1`
}
# ==========================================================
#                plot macro
# ----------------------------------------------------------
```

```
macro_plot()
{
    echo "
# command: bla
CmdN;+;32
#  Objekte zum Ausblenden wahlen
SegR;+;o;101.504;35.6148
Region;+;104.311;35.9976;104.439;36.2528
RET;SegR
ESC;SegR
# command: ubl
CmdN;+;33
Menu;+;2
#  Textwahl
Panel;+;0;1;$LAYER_CNT;l$DIMENS
Y-N;+;1
#  Taste II ; Zuruck
Panel;+;0;2;1;;
BS;Menu
# command: prn
CmdN;+;420
#  Textwahl ; (Kompatiblen) Drucker wahlen:
Panel;+;1;1;8;HPGL Plotter (pen plotter)
>Text;4;;
#  Taste ; Ausgabe un Datei
Panel;+;1;3;1;;
>Text;4;lpr %s
#  Taste II ; OK
Panel;+;1;5;1;;
>Text;4;/home/gerhardr/eagle/rot_facility/$TOOL_N.hpg
#  Taste II ; Drucken
Panel;+;3;20;1;;
>Text;8;1
>Text;13;297
>Text;14;420" >> $MACROFIL
  LAYER_CNT=`expr $LAYER_CNT + 1`
}
# ===========================================================
#               MAIN function begin
# ===========================================================
cd $DATA_DIR
LAYER_CNT=3
case $MODE in
  l* ) MACROFIL=$MACRODIR/mk_layr.mcr ;;
  d* ) MACROFIL=$MACRODIR/mk_draw.mcr ;;
  p* ) MACROFIL=$MACRODIR/mk_plot.mcr ;;
  a* ) $0 layer
       $0 draw
       $0 plot
       exit 0;;
  *  ) echo "Invalid mode '$MODE', valid modes are: layer, draw, plot or all."
       exit 1 ;;
esac
rm -f $MACROFIL
macro_header
for DATA in tool-?.????.dat
do DIMENS=`echo $DATA|cut -c8-11`
   TOOL_N=`echo $DATA|cut -c1-11`
  case $MODE in
    l* ) macro_layr ;;
    d* ) macro_draw ;;
    p* ) macro_plot ;;
  esac
done
echo " created."
```

D. Software Organisation

Dieser Teil des Anhangs ist den software-organisatorischen Inhalten gewidmet und listet Werkzeuge und Strukturen des Projektes auf.

D.1. Makefile

```
#
# Makefile for RF_Control.asm
#
MODULE=RF_Control
### MODULE=RF_Test
OUTPUT=$(MODULE).hex
SYSINCL = ../inc
INC_DIR = inc
LIBRARY = lib
MODULES = mod
ASSEMBLER = avra
# comment out the following for removing debug code
DEFINES = --listfile $(MODULE).list
# DEFINES = --listfile $(MODULE).list --define DEBUG=1
SOURCES = inc/*.inc lib/*.lib mod/*.asm
DEBUG_F = testfunc.asm debug_fun.inc
ASM_OPT = $(DEFINES)
ASM_OPT+= --includedir $(SYSINCL)
ASM_OPT+= --includedir $(INC_DIR)
ASM_OPT+= --includedir $(LIBRARY)
ASM_OPT+= --includedir $(MODULES)
UISP = uisp
ISP_OPT = -v=2
ISP_OPT+= -dprog=stk500
ISP_OPT+= -dserial=/dev/modem
ISP_OPT+= -dspeed=115200
ISP_OPT+= -dpart=ATmega8
### Firmware Version: 1.18
### Atmel AVR ATmega8 is found.
### Firmware Version: 1.18
###
### Fuse Low Byte      = 0xdd
### Fuse High Byte     = 0xd9
### Cannot read extended fuse bits on this device. Returning 0xff
### Fuse Extended Byte = 0xff
### Calibration Byte   = 0x99  --  Read Only
### Lock Bits          = 0xff
###     BLB12 -> 1
###     BLB11 -> 1
###     BLB02 -> 1
###     BLB01 -> 1
###       LB2 -> 1
###       LB1 -> 1
###
#  --wr_fuse_h  Write fuse high byte
#  --wr_fuse_e  Write fuse extended byte
#                                ----V---
#                                76543210
# Fuse High Byte normal = 0xd9 = 11011001
# EEPROM preserved      = 0xd1 = 11010001
```

```
#
# Fuse High Byte
# +-------------+---+----------------------+------------------------+
# | Fuse High   |Bit| Description          | Default Value          |
# |   Byte      |No.|                      |                        |
# +-------------+---+----------------------+------------------------+
# | RSTDISBL(4) | 7 |  Select if PC6 is I/O | 1 (unprogrammed, PC6   |
# |             |   |  pin or RESET pin    |   is RESET-pin)        |
# +-------------+---+----------------------+------------------------+
# | WDTON       | 6 |  WDT always on       | 1 (unprogrammed, WDT   |
# |             |   |                      |    enabled by WDTCR)   |
# +-------------+---+----------------------+------------------------+
# | SPIEN(1)    | 5 | Enable Serial Program | 0 (programmed, SPI     |
# |             |   | and Data Downloading |   prog. enabled)       |
# +-------------+---+----------------------+------------------------+
# | CKOPT(2)    | 4 | Oscillator options   | 1 (unprogrammed)       |
# +-------------+---+----------------------+------------------------+
# | EESAVE      | 3 | EEPROM memory is     | 1 (unprogrammed, EEPROM |
# |             |   | preserved through    |   not preserved)       |
# |             |   | the Chip Erase       |                        |
# +-------------+---+----------------------+------------------------+
# | BOOTSZ1     | 2 | Select Boot Size     | 0 (programmed)(3)      |
# +-------------+---+----------------------+------------------------+
# | BOOTSZ0     | 1 | Select Boot Size     | 0 (programmed)(3)      |
# +-------------+---+----------------------+------------------------+
# | BOOTRST     | 0 | Select Reset Vector  | 1 (unprogrammed)       |
# +-------------+---+----------------------+------------------------+
#
### Not preserve EEPROM memory call
# uisp -v=2 -dprog=stk500 -dserial=/dev/modem -dspeed=115200\
#      -dpart=ATmega8 --wr_fuse_h=0xd9
### Preserve EEPROM memory call
# uisp -v=2 -dprog=stk500 -dserial=/dev/modem -dspeed=115200\
#      -dpart=ATmega8 --wr_fuse_h=0xd1
#
$(OUTPUT):      $(MODULE).asm $(SOURCES) $(DEBUG_F)
        $(ASSEMBLER) $(ASM_OPT) $(MODULE).asm
all:    $(OUTPUT)
up:             $(OUTPUT)
        $(UISP) $(ISP_OPT) --erase --upload if=$(OUTPUT)
clean:
        rm -f $(MODULE).hex $(MODULE).list $(MODULE).map $(MODULE).obj\
              $(MODULE).cof $(MODULE).eep.hex
```

D.2. Verzeichnisstruktur

Verzeichnis	Dateiname	Bemerkung
./	Makefile	Make-Steuerdatei
./	RF_Control.asm	Hauptprogramm
./	debug_fun.inc	DEBUG-Hilfsfunktionen
./	testfunc.asm	Testfunktionen (produktiv nicht benötigt)
./inc/		Include-Dateien
./inc	constants.inc	Konstantendefinitionen
./inc	cseg_data.inc	Code-Segment Daten (Text-Defines)
./inc	intrupts.inc	Interrupttabelle und Stack-Init
./inc	ioconfig.inc	Port I/O Definitionen
./inc	macros.inc	Makrosammlung
./inc	reg_defs.inc	Register-Defines (symbolische Namen)
./inc	variables.inc	Datasegement, Variablendefinitionen
./lib/		Assemblercode-Bibliotheksroutinen
./lib	datetime.lib	RTC-Highlevel-Funktionen (nicht verw.)
./lib	delay.lib	Software-Timer Routinen
./lib	ds1307.lib	RTC-Lowlevel Funktionen (nicht. verw.)
./lib	eepromio.lib	EEPROM Ein-/Ausgabe
./lib	iic.lib	I2C Lowlevel Funktionen
./lib	keyboard.lib	Keyboard Low/Highlevel Funktionen
./lib	lcd_func.lib	LCD-Lowlevel Funktionen
./lib	lm75.lib	Temparatursensor Lowlevel Funktionen
./lib	mainmenu.lib	Hauptmenü und allg. Menüsteuerung
./lib	num_disp.lib	nummerische typspez. Ausgaberoutinen
./lib	pcf8591.lib	A/D-D/A Lowlevel Ausgaberoutinen
./lib	rotation.lib	Drehsteuerung Highlevel Funktionen
./lib	systimer.lib	Hardware Timer und ISRs
./lib	temperat.lib	Highlevel Temperaturroutinen
./mod/		Moduleverzeichnis/Progammmodi
./mod	calibrat.asm	Calibrate: Winkelspeicherung und Ref.Pos.
./mod	goto_pos.asm	GotPosn: automat. Positionssteuerung
./mod	info_mod.asm	Information: Info-Anzeigen/Programminfo
./mod	initcycl.asm	InitCycle: Initialisierung (teilweise impl.)
./mod	man_ctrl.asm	ManControl: manuelle Steuerung
./mod	set_posn.asm	SetPosn: manuelle Winkelvorgabe

E. Verwendete Werkzeuge

In diesem Bereich werden die „Usage-Meldungen" der verwendeten Werkzeuge angegeben, die die möglichen Ausführungsoptionen des jeweiligen Programms beschreiben und damit auch ein Beurteilungskriterium für den Funktionsumfang darstellen.

E.1. Tracing Software

```
Usage: autotrace [options] <input_name>.
Options:<input_name> should be a supported image.
  You can use '--' or '-' to start an option.
  You can use any unambiguous abbreviation for an option name.
  You can separate option names and values with '=' or ' '.
background-color <hexadezimal>: the color of the background that
  should be ignored, for example FFFFFF;
  default is no background color.
centerline: trace a character's centerline, rather than its outline.
color-count <unsigned>: number of colors a color bitmap is reduced to,
  it does not work on grayscale, allowed are 1..256;
  default is 0, that means not color reduction is done.
corner-always-threshold <angle-in-degrees>: if the angle at a pixel is
  less than this, it is considered a corner, even if it is within
  'corner-surround' pixels of another corner; default is 60.
corner-surround <unsigned>: number of pixels on either side of a
  point to consider when determining if that point is a corner;
  default is 4.
corner-threshold <angle-in-degrees>: if a pixel, its predecessor(s),
  and its successor(s) meet at an angle smaller than this, it's a
  corner; default is 100.
despeckle-level <unsigned>: 0..20; default is no despeckling.
despeckle-tightness <real>: 0.0..8.0; default is 2.0.
dpi <unsigned>: The dots per inch value in the input image, affects scaling
  of mif output image
error-threshold <real>: subdivide fitted curves that are off by
  more pixels than this; default is 2.0.
filter-iterations <unsigned>: smooth the curve this many times
  before fitting; default is 4.
input-format:  PNG, TGA, PBM, PNM, PGM, PPM, 8BIM or BMP.
help: print this message.
line-reversion-threshold <real>: if a spline is closer to a straight
  line than this, weighted by the square of the curve length, keep it a
  straight line even if it is a list with curves; default is .01.
line-threshold <real>: if the spline is not more than this far away
  from the straight line defined by its endpoints,
  then output a straight line; default is 1.
list-output-formats: print a list of support output formats to stderr.
list-input-formats:  print a list of support input formats to stderr.
log: write detailed progress reports to <input_name>.log.
output-file <filename>: write to <filename>
output-format <format>: use format <format> for the output file
  eps, ai, p2e, sk, svg, fig, emf, mif, er, dxf, epd, pdf, cgm, ..., magick,
  idraw, obj, tgif, gnuplot, svm, tk, gschem, pcbfill, pcb, pcb, pcbi, hpgl,
  pic, xml, noixml, tex, latex2e, m, mma, asy, mp, mpost, txt, text, kil,
  java2, java, java1, rpl, rib, lwo, meta, gmfa, meta, gmfb, plot, pnm,
  plot-pnm, pcl, plot-pcl, hpgl, plot-hpgl, tek, plot-tek or dr2d can be used.
preserve-width: whether to preserve line width prior to thinning.
remove-adjacent-corners: remove corners that are adjacent.
```

```
tangent-surround <unsigned>: number of points on either side of a
  point to consider when computing the tangent at that point; default is 3.
report-progress: report tracing status in real time.
debug-arch: print the type of cpu.
debug-bitmap: dump loaded bitmap to <input_name>.bitmap.
version: print the version number of this program.
width-weight-factor <real>: weight factor for fitting the linewidth.
You can get the source code of autotrace from
http://autotrace.sourceforge.net
```

E.2. Assembler

```
AVRA: advanced AVR macro assembler Version 1.0.1 Build 113 (12. February 2004)
Copyright (C) 1998-2004 Jon Anders Haugum, Tobias Weber
   AVRA is an open source assembler for Atmel AVR microcontroller family
   It can be used as a replacement of 'AVRASM32.EXE' the original assembler
   shipped with AVR Studio. We do not guarantee full compatibility for avra.
   AVRA comes with NO WARRANTY, to the extent permitted by law.
   You may redistribute copies of avra under the terms
   of the GNU General Public License.
   For more information about these matters, see the files named COPYING.
usage: AVRA [-f][O|M|I|G] output file type
            [-o <filename>] output file name
            [-l <filename>] generate list file
            [-m <mapfile>] generate map file
[--define <symbol>[=<value>]]  [--includedir <dir>] [--listmac]
            [--max_errors <number>] [--devices] [--version]
            [-h] [--help] general help
                        <file to assemble>
   --listfile    -l : Create list file
   --mapfile     -m : Create map file
   --define      -D : Define symbol.
   --includedir  -I : Additional include dirs.
   --listmac        : List macro expansion in listfile.
   --max_errors     : Maximum number of errors before exit
                      (default: 10)
   --devices        : List out supported devices.
   --version        : Version information.
   --help, -h       : This help text.
Just replace the AVRASM32.EXE with AVRA.EXE in your
AVRStudio directories to avra's binary.
Report bugs to tobiw@suprafluid.com
            or jonah@omegav.ntnu.no
```

E.3. USIP Programmiersoftware

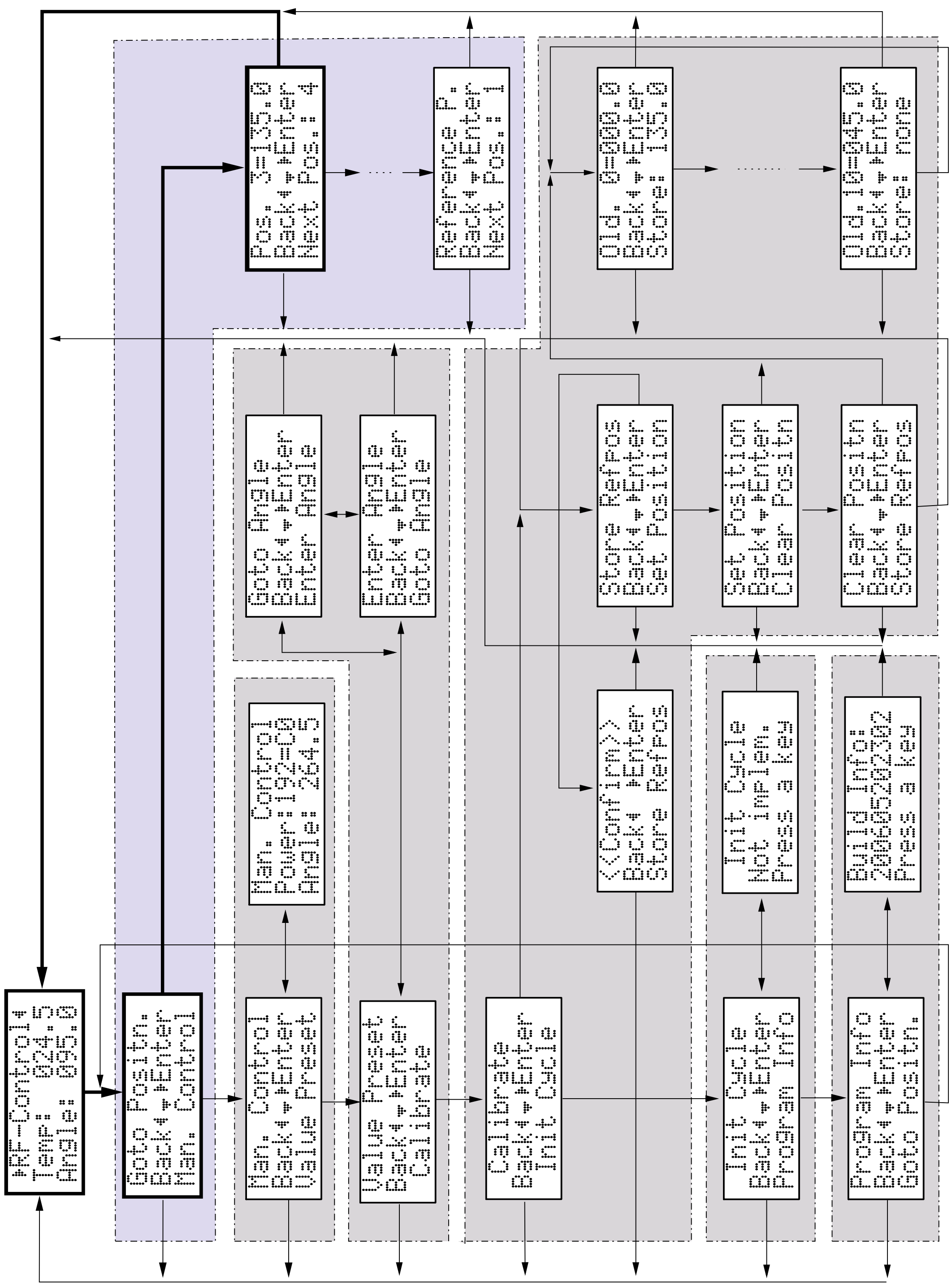
```
uisp version 20050207
(C) 1997-1999 Uros Platise, 2000-2003 Marek Michalkiewicz
uisp is free software, covered by the GNU General Public License.
You are welcome to change it and/or distribute copies of it under
the conditions of the GNU General Public License.
Syntax: uisp [-v{=level}] [-h] [--help] [--version] [--hash=perbytes]
             [-dprog=avr910|pavr|stk500] [-dprog=type]
             [-dlpt=address|/dev/parportX] [-dno-poll] [-dno-retry]
             [-dvoltage=...] [-dt_sck=time] [-dt_wd_{flash|eeprom}=time]
             [-dt_reset=time] [-dinvert=sck,mosi,miso,reset]
             [-dserial=device] [-dpart=name|no]
             [-dspeed=1200|2400|4800|9600|19200|38400|57600|115200]
             [--upload] [--verify] [--erase] [if=input_file]
             [--download] [--size=bytes] [of=output_file]
             [--segment=flash|eeprom|fuse] [--terminal]
             [--rd_fuses] [--wr_fuse_l=byte] [--wr_fuse_h=byte]
             [--wr_fuse_e=byte] [--wr_lock=byte]
```

```
Programming Methods:
  -dprog=avr910    Standard Atmel Serial Programmer/Atmel Low Cost Programmer
         pavr      http://www.avr1.org/pavr/pavr.html
         stk500    Atmel STK500
         avrisp    Atmel AVRISP (uses stk500 protocol, but can only perform
                   serial programming using SPI)
  -dprog=dapa|stk200|abb|atdh|avrisp|bsd|fbprg|dt006|maxi|xil|dasa|dasa2|dasa3
       Programmer type:
         dapa      Direct AVR Parallel Access
         stk200    Parallel Starter Kit STK200, STK300
         abb       Altera ByteBlasterMV Parallel Port Download Cable
         atdh      Atmel-ISP Download Cable
         paravrisp [Unknown origin] Parallel port programmer
         bsd       http://www.bsdhome.com/avrdude/ (parallel)
         fbprg     http://ln.com.ua/~real/avreal/adapters.html (parallel)
         dt006     http://www.dontronics.com/dt006.html (parallel)
         maxi      Investment Technologies Maxi (parallel)
         xil       Xilinx HW-JTAG-PC Cable (parallel)
         ett       ETT AVR Programmer V2.0 [from Futurlec] (parallel)
         dm04      Datamekatronik 2004 (parallel)
         dasa      serial (RESET=RTS SCK=DTR MOSI=TXD MISO=CTS)
         dasa2     serial (RESET=!TXD SCK=RTS MOSI=DTR MISO=CTS)
         dasa3     serial (RESET=!DTR SCK=RTS MOSI=TXD MISO=CTS)
Target Device Selection:
  -dpart       Set  target abbreviated name or number. For some programmers, if
               -dpart is not given programmer's supported devices  are  listed.
               Set  -dpart=auto for auto-select. Auto-select does not work with
               all programmers, so it is recommended to always specify a target
               device explicitly.
Parallel Device Settings:
  -dlpt=       specify device name (Linux ppdev, FreeBSD ppi, serial)
               or direct I/O parallel port address (0x378, 0x278, 0x3BC)
  -dno-poll    Program without data polling (a little slower)
  -dno-retry   Disable retries of program enable command
  -dvoltage    Set timing specs according to the power supply voltage in [V]
               (default 3.0)
  -dt_sck      Set minimum SCK high/low time in micro-seconds (default 5)
  -dt_wd_flash Set FLASH maximum write delay time in micro-seconds
  -dt_wd_eeprom Set EEPROM maximum write delay time in micro-seconds
  -dt_reset    Set reset inactive (high) time in micro-seconds
  -dinvert=... Invert specified lines
               Use -v=3 option to see current settings.
  -d89         Allow parallel programming of AT89S51 and AT89S52 devices.
Atmel Low Cost Programmer Serial Device Settings:
  -dserial     Set serial interface as /dev/ttyS* (default /dev/avr)
  -dspeed      Set speed of the serial interface (default 19200)
Stk500 specific options:
  -dhiv        Use Hi-Voltage programming instead of SPI (default is SPI)
  --rd_aref    Read the ARef Voltage. Note that due to a bug in the
               stk500 firmware, the read value is sometimes off by 0.1
               from the actual value measured with a volt meter.
  --wr_aref    Set the ARef Voltage. Valid values are 0.0 to 6.0 volts in
               0.1 volt increments. Value can not be larger than the
               VTarget value.
  --rd_vtg     Read the Vtarget Voltage. Note that due to a bug in the
               stk500 firmware, the read value is sometimes off by 0.1
               from the actual value measured with a volt meter.
  --wr_vtg     Set the VTarget Voltage. Valid values are 0.0 to 6.0 volts in
               0.1 volt increments. Value can not be smaller than the
               ARef value.
  --rd_osc     Read the oscillator frequency in Hertz.
  --wr_osc     Set the oscillator frequency in Hertz, from 14.06 to 3686400.
Functions:
  --upload     Upload "input_file" to the AVR memory.
  --verify     Verify "input_file" (processed after the --upload opt.)
  --download   Download AVR memory to "output_file" or stdout.
  --erase      Erase device.
```

```
  --segment    Set active segment (auto-select for AVA Motorola output)
  --size       Set segment size for download (default entire segment)
Fuse/Lock Bit Operations:
  --rd_fuses   Read all fuses and print values to stdout
  --wr_fuse_l  Write fuse low byte
  --wr_fuse_h  Write fuse high byte
  --wr_fuse_e  Write fuse extended byte
  --wr_lock    Write lock bits. Argument is a byte where each bit is:
                   Bit5 -> blb12
                   Bit4 -> blb11
                   Bit3 -> blb02
                   Bit2 -> blb01
                   Bit1 -> lb2
                   Bit0 -> lb1
  --lock       Write lock bits [old method; deprecated].
Files:
  if           Input file for the --upload and --verify functions in
               Motorola S-records (S1 or S2) or 16 bit Intel format
  of           Output file for the --download function in
               Motorola S-records format, default is standard output
Other Options:
  -v           Set verbose level (-v equals -v=2, min/max: 0/4, default 1)
  --hash       Print hash (default is 32 bytes)
  --help -h    Help
  --version    Print version information
  --terminal   Invoke shell-like terminal
Report bugs to: Maintainers <uisp-dev@nongnu.org>
Updates:        http://savannah.nongnu.org/projects/uisp
```

F. Menüstruktur

Err.: 4.10.06
Abb. ersetzt



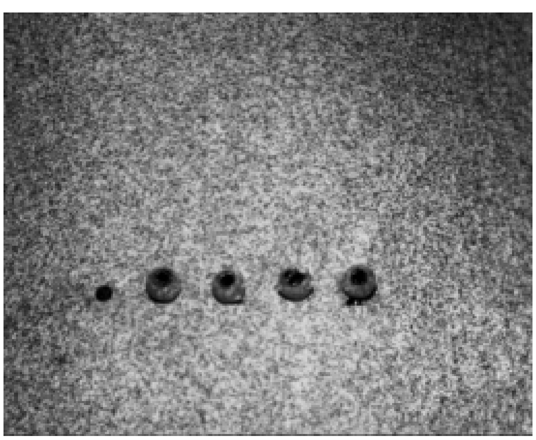
G. Messdatenreihe

Messdatenreihe zur Genauigkeitsbestimmung der Drehwinkelsteuerung.

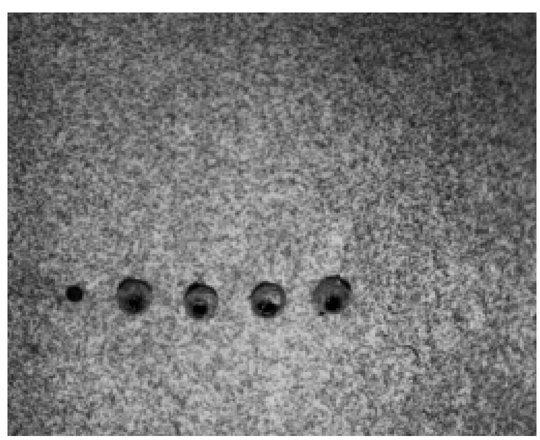
Tabelle G.1.: Messreihe, Abbildungen

Bild	Winkel	Kam.	Abbildung	Bemerkung
1	315	1	frame_0001_depth_001_im1.gif	Markierungen
2		2	frame_0001_depth_001_im2.gif	sichtbar
3	45	1	frame_0002_depth_002_im1.gif	
4		2	frame_0002_depth_002_im2.gif	
5		1	frame_0003_depth_003_im1.gif	
6		2	frame_0003_depth_003_im2.gif	
	225			keine Aufnahme
7	315	1	frame_0004_depth_004_im1.gif	Markierungen
8		2	frame_0004_depth_004_im2.gif	sichtbar
9	225	1	frame_0005_depth_005_im1.gif	
10		2	frame_0005_depth_005_im2.gif	
11	315	1	frame_0006_depth_006_im1.gif	Markierungen
12		2	frame_0006_depth_006_im2.gif	sichtbar
	180			keine Aufnahme
13	315	1	frame_0007_depth_007_im1.gif	Markierungen
14		2	frame_0007_depth_007_im2.gif	sichtbar

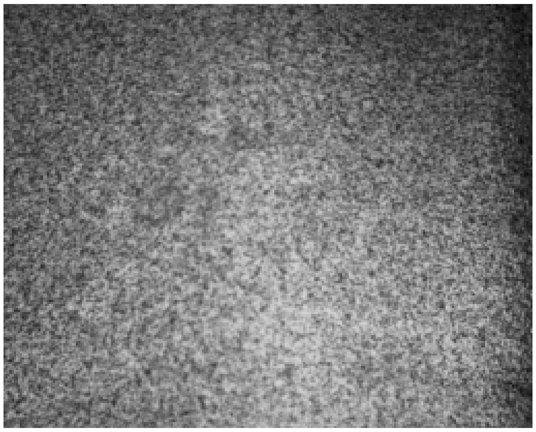
frame_0001_depth_001_im1.gif



frame_0001_depth_001_im2.gif

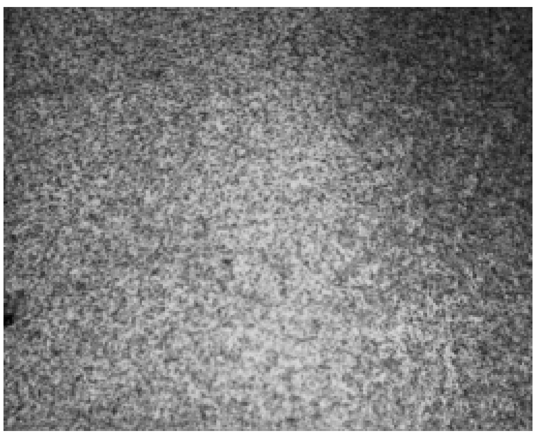


frame_0002_depth_002_im1.gif

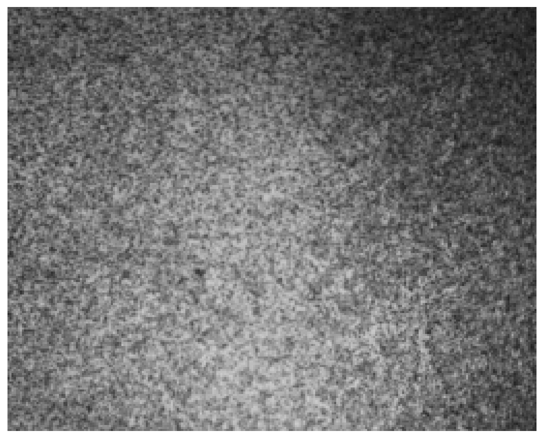


frame_0002_depth_002_im2.gif

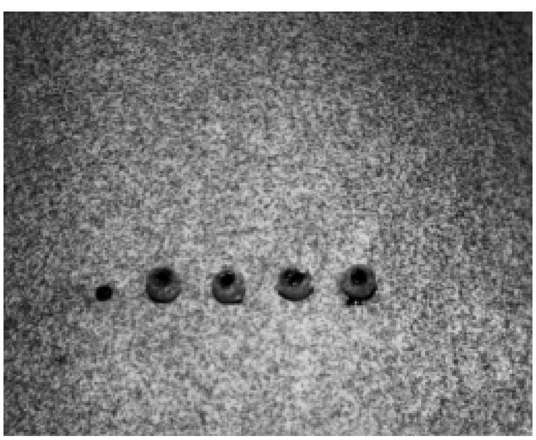
frame_0003_depth_003_im1.gif



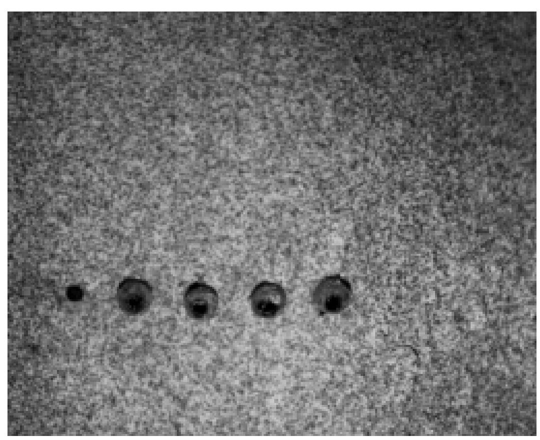
frame_0003_depth_003_im2.gif



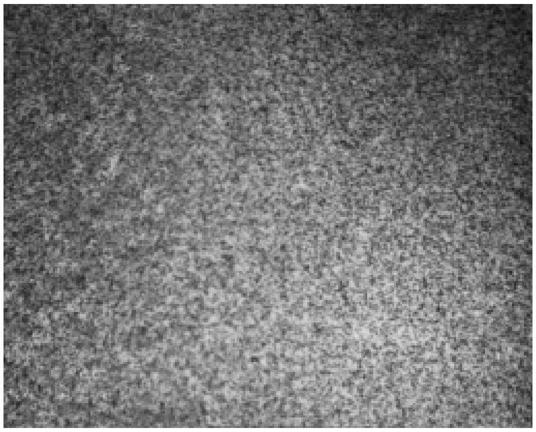
frame_0004_depth_004_im1.gif



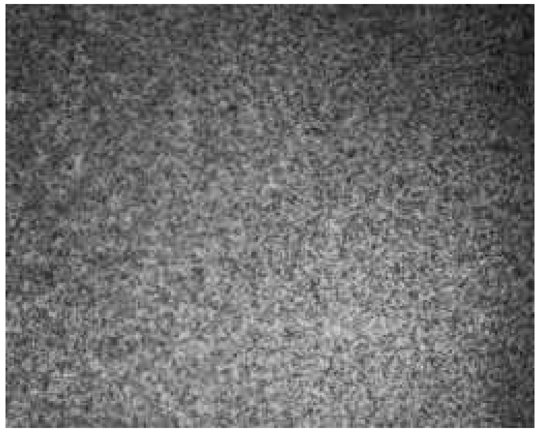
frame_0004_depth_004_im2.gif



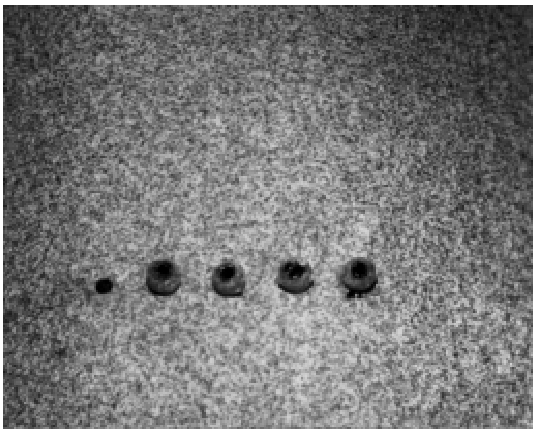
frame_0005_depth_005_im1.gif



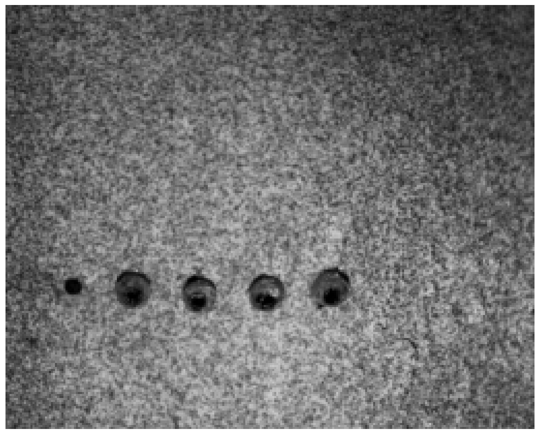
frame_0005_depth_005_im2.gif



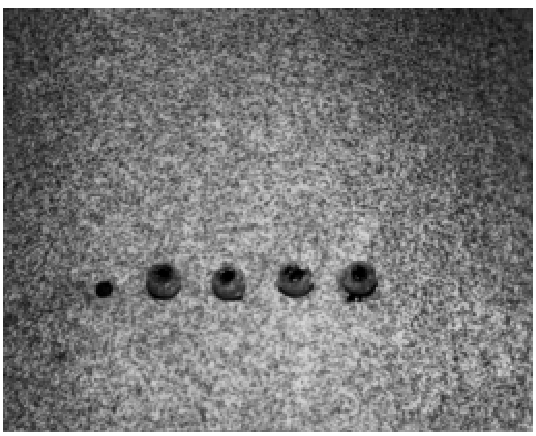
frame_0006_depth_006_im1.gif



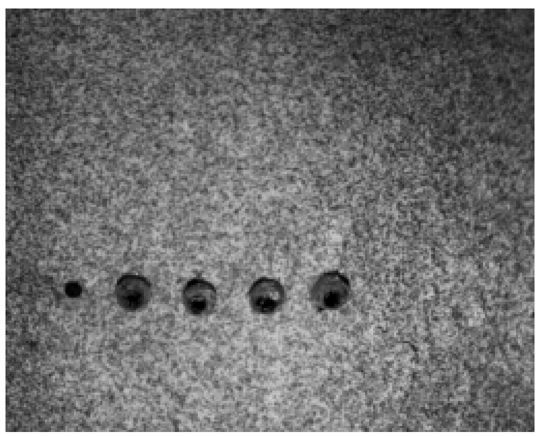
frame_0006_depth_006_im2.gif



frame_0007_depth_007_im1.gif



frame_0007_depth_007_im2.gif



Literaturverzeichnis

[1] Norm DIN 19226

[2] Beckhoff: Antriebsmotoren
In: PC-Control - Firmenmagazin Beckhoff GmbH. - 2005/3 - S.8, Online: http://www.pc-control.net/pdf/032005/pcc0305_title_d.pdf, 2006-07-22

[3] Philips Semiconductors - I^2C-BUS SPECIFICATION - VERSION 2.1 - JANUARY 2000 - Doc. Order No: 9398 393 40011

[4] Rober Bosch GmbH - CAN Specification - Version 2.0 - 1991 - Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart

[5] Philps Semiconductors - PCF8591 8-bit A/D and D/A converter - 2003 Jan 27 - Doc. Order No: 9397 750 10464

[6] National Seminconductor - LM741 Operational Amplifier - Nov. 1994 - Doc. RRD-B30M115

[7] Schneider, Uwe: Werner, Dieter: Taschenbuch der Informatik - 5. Auflage - Leipzig: Carl Hanser, 2004

[8] Open Source Initiative osi@opensource.org: Open Source Homepage. URL: http://www.opensource.org, 2006-07-22

[9] Linux Online Inc. webmaster@linux.org: Linux Homepage, URL: http://www.linux.org, 2006-06-22

[10] Free Software Foundation, Inc. gnu@gnu.org: GNU General Puplic License, URL: http://www.gnu.org/copyleft/gpl.html, 2006-06-22

[11] Tcl Developer Xchange webmaster@tcl.tk: Tcl/Tk License Terms, URL: http://www.tcl.tk/software/tcltk/license.html, 2006-06-22

[12] Foley, James D...: Computer Graphics: Principles and Practice. - Massachussets: Addison-Wesely, 1993

[13] TCL-Lib andreas_kupries@users.sourceforge.net..: Sourceforge project page: http://sourceforge.net/projects/tcllib/, 2006-08-13

[14] Reithofer, Gerhard: Bag rotation facility manual - Internal document AMFT-7-2005 Rev. II - TU-Graz

[15] ATMEL - 8-bit AVR with 8-K Bytes, In-System Pragrammable Flash ATmega8, ATmega8L - Rev. 2486J-AVR-02/03 - 2003

[16] Wikimedia Foundation, Inc. board@wikimedia.org: Cheat Codes, URL: http://en.wikipedia.org/wiki/Cheat_codes, 2006-06-22

Danksagung

Ich möchte mich herzlich bei allen, die mich bei dieser Arbeit und dem vorausgehenden Studium so tatkräftig unterstützt haben, bedanken, ohne deren Hilfe hätte diese Arbeit nicht entstehen können.

Im Besonderen gilt dies für alle Mitglieder meiner Familie, die wirkliches tiefes Verständnis dafür aufgebracht, alle meine Launen und Stimmungen tapfer ertragen haben und auch bei zahlreichen Recherchen und Diskussionen wertvolle Beiträge einbrachten.

Weiters gilt mein Dank dem Initiator dieser Diplomarbeit, Univ.-Prof. Dipl. Ing, Dr. techn. Gernot Krammer von der TU Graz, später Prof. an der NTNU (Norges teknisk-naturvitenskapelige universitet) Trondheim, der mich in allen Dingen der Ausführung der Arbeit unterstützt hat, MSC. Mahmood Saleem, als Hauptanwender der Anlage, vom Institut AMFT an der TU Graz, der mir in allen Angelegenheiten für den Um- und Aufbau der neuen Anlage zur Seite gestanden ist, Ing. Klaus Kutschi, für seine immer kompetente hilfreiche Unterstützung in allen Hardwarebelangen und allen beteiligten Mitarbeitern der AMFT-Crew, sowie O.Univ.-Prof. Dipl.-Ing. Dr.techn. Gernot Staudinger als Leiter dieses Institutes.

Abschließend möchte ich mich noch bei Prof. Dr.-Ing. Olaf Hagenbruch und Dipl. Ing. (FH) Heiko Polster vom ACMC, Fachbereich IT&ET an der HTWM für die Betreuung dieser Diplomarbeit und ihre fachliche und organisatorische Unterstützung recht herzlich bedanken.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Graz, am 8. August 2006 Unterschrift