

**VORLÄUFIGE ARBEITSKOPIE!**

**ÜBERSETZEN VON**

**SCHRITTMOTORPROTOKOLLEN**

**Entwurf eines Hardwareübersetzers**

**Praxisbericht**

im Fachgebiet Mess- und Sensortechnik



vorgelegt von: Johannes Dielmann  
Studienbereich: Technik  
Matrikelnummer: 515956  
Erstgutachter: Prof. Dr. Carstens-Behrens

© 2012

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>IV</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Vorstellung der vorhandenen Hardware</b>	<b>2</b>
2.1. Computer	2
2.2. 3D-Laserscanner VI-900	3
2.2.1. Lasertriangulator Prinzip	3
2.3. Drehtisch und Ansteuerung	4
2.3.1. Drehtisch	4
2.3.2. Spannungsversorgung	4
2.3.3. Schrittmotoren	5
2.3.4. Schrittmotorkarten	5
2.3.5. Motorverkabelung	5
2.3.6. Endschalter	6
2.4. Mikrocontroller	6
2.4.1. Entwicklerboard STK500	7
2.4.2. AVRISP mkII	8
2.4.3. MAX232	8
<b>3. Vorstellung der vorhandenen Software</b>	<b>9</b>
3.1. RapidForm2004	9
3.2. Entwicklungsumgebung	9
3.3. Terminalprogramme	9
<b>4. Zeitlicher Arbeitsablauf</b>	<b>10</b>
4.1. Bereitstellen grundlegender Funktionalitäten	11
4.1.1. Taster	11
4.1.2. LEDs	12
4.1.3. LCD ansteuern	13

4.1.4. RS-232-Schnittstelle . . . . .	14
4.2. Befehlssätze . . . . .	16
4.3. Kommunikation mit der Schrittmotorsteuerung . . . . .	17
4.3.1. Befehle senden . . . . .	17
4.3.2. Antworten Empfangen und speichern . . . . .	19
4.3.3. Antworten auswerten . . . . .	20
4.4. Verbesserungen an der vorhandenen Hardware . . . . .	22
4.4.1. Netzteil . . . . .	22
4.4.2. Zweite Schrittmotorkarte . . . . .	22
4.4.3. Motor- und Endschalerverkabelung . . . . .	23
4.4.4. Endschalter . . . . .	24
4.4.5. Zweite serielle Schnittstelle . . . . .	25
4.5. Kommunikation mit RapidForm2004 . . . . .	26
4.5.1. Befehle empfangen . . . . .	26
4.5.1.1. Automatische Auswahl eines Befehlssatzes . . . . .	27
4.6. Auswerte-Funktionen . . . . .	29
4.6.1. Auswerte-Funktion für Isel Motoren . . . . .	29
4.6.1.1. Initialisierung . . . . .	30
4.6.1.2. Statusabfrage . . . . .	30
4.6.1.3. Bewegung . . . . .	30
<b>5. Probleme und Lösungen</b>	<b>33</b>
5.1. Entwicklungsumgebungen . . . . .	33
5.1.1. AVR Studio 5 . . . . .	33
5.1.2. Eclipse . . . . .	33
5.2. Interrupts . . . . .	33
5.2.1. Endschalter . . . . .	34
5.2.2. Watchdog . . . . .	34
5.3. Fuses . . . . .	35
5.4. Platinenlayout . . . . .	36
5.5. 19"-Einschub . . . . .	36
<b>6. Fazit und Zukunft</b>	<b>38</b>
6.1. Fazit . . . . .	38
<b>Eidesstattliche Erklärung</b>	<b>41</b>

<b>A. Anhang</b>	<b>i</b>
A.1. Schritt für Schritt Anleitung . . . . .	ii
A.2. Protokoll der Schrittmotorkarte . . . . .	x
A.3. Protokolle aus RapidForm2004 . . . . .	xi
A.4. Technische Daten VI-910 . . . . .	xiv
A.5. Verwendete Hardware . . . . .	xvi
A.6. Verwendete Software . . . . .	xvi

## Abkürzungsverzeichnis

ADC .....	Analog Digital Convertor
ASCII .....	American Standard Code for Information Interchange
AVRISP .....	AVR in System Programmer
CAD .....	Computer Aided Design
CF .....	Compact Flash
CPU .....	Central Processing Unit
DAC .....	Digital Analog Convertor
DIL .....	Dual in Line Package
IC .....	Integrated Cuircuit
ISR .....	Interrupt Service Routine
LCD .....	Liquid Crystal Display
MHz .....	Megahertz
RS .....	Recommended Standard
SCSI .....	Small Computer System Interface
USART .....	Universal Asynchronous Receiver Transmitter
USB .....	Universal Serial Bus
V .....	Volt

## Abbildungsverzeichnis

2.1. Blick auf den Arbeitsaufbau . . . . .	2
2.2. VI-900 - 3D-Scanner . . . . .	3
2.3. Prinzip: Laser-Triangulation . . . . .	4
2.4. Drehtisch . . . . .	5
2.5. Ansteuerung im 19"-Rack . . . . .	6
2.6. Block Diagram: Mikrocontroller . . . . .	7
2.7. Schema: STK500 . . . . .	8
4.1. Stromverbinder - Y-Kabel? . . . . .	22
4.2. Motor- und Endschalterverkabelung . . . . .	23
4.3. Motor- und Endschalterverkabelung . . . . .	24
4.4. Schema: MAX232 . . . . .	25
5.1. Platinenlayout . . . . .	37

# Tabellenverzeichnis

2.1. Aufbau . . . . .	2
4.1. Motor- und Endschalterverkabelung . . . . .	23
5.1. Fuses . . . . .	36
A.1. Schritt für Schritt Anleitung . . . . .	ii
A.2. ASCII Befehlssatz R+S Schrittmotorsteuerung . . . . .	x
A.3. Technische Daten - VI-910 . . . . .	xiv

## Codeverzeichnis

4.1. Taster . . . . .	11
4.2. LEDs . . . . .	12
4.3. lcd.h (Auszug) . . . . .	13
4.4. RS-232 . . . . .	14
4.5. Befehlssatz aus Rapidform: Isel . . . . .	17
4.6. Menü . . . . .	18
4.7. Menü Baum . . . . .	18
4.8. RS-232 Empfang . . . . .	20
4.9. FindStringInArray() . . . . .	20
4.10. switchStepper() . . . . .	21
4.11. RS-232 Empfang - RapidForm2004 . . . . .	26
4.12. Funktion: uartrx() . . . . .	27
4.13. Funktion: switchMotor . . . . .	28
4.14. Übersetzungs Logik: Isel . . . . .	30
5.1. ISR: Endschalter . . . . .	34
5.2. Watchdog . . . . .	35
A.1. RapidForm2004 Protokolle Empfang . . . . .	xi



# 1. Einleitung

Ein 3D-Laserscanner bietet vielfältige Möglichkeiten und Einsatzgebiete. Die Haupteinsatzgebiete finden sich in der Bauteilprüfung, zur Erstellung von Finite-Elemente-Daten in Verbindung mit Bauteilanalyse, zur Erstellung von 3D-Daten, zur Kontrolle von Zubehörteilen, zur Archivierung in Museen und Forschungseinrichtungen, zum Reverse-Engineering und in vielen weiteren Gebieten.

Im Besitz der Fachhochschule befand sich ein komplettes 3D-Lasererfassungssystem. Dazu gehörten eine Erfassungssoftware, ein 3D-Laserscanner und ein Drehtisch. Bisher mussten, um eine Aufnahme zu tätigen, alle Komponenten zueinander passen. Der Drehtisch in diesem System war jedoch ein Eigenbau der Fachhochschule und die darin verbaute Drehtischsteuerung nicht kompatibel zu denen von der Erfassungssoftware unterstützten Drehtischsteuerungen.

Mittels eines Mikrocontrollers sollte der vorhandene Aufbau so erweitert werden, dass der Drehtisch von der Software angesteuert werden konnte und so der volle Umfang des Systems nutzbar gemacht werden. Dabei waren folgende Vorgaben zu realisieren. Die Höhenverstellung des Drehtisches sollte genutzt werden können und die verbauten Endschalter ihre vorhergesehene Funktion erfüllen. Der Mikrocontroller sollte sich mit mehreren Tastern bedienen lassen und über ein LC-Display verfügen, welches den aktuellen Status anzeigt. Mit einer Schritt-für-Schritt-Anleitung sollte es auch für Studenten und Mitarbeiter der Fachhochschule möglich sein, schnell und einfach eine Aufnahme durchzuführen. Die Daten dieser Aufnahme sollten exportiert und in z.B. CAD-Software genutzt werden können.

Der Aufbau der Arbeit gliedert sich im Wesentlichen in die Vorstellung der Vorhanden Hard- und Software, den chronologischen Arbeitsablauf während des Projektes, ein Kapitel das Probleme und deren Lösungen aufzeigt, in ein Fazit und mögliche zukünftige Verbesserungen. Im Anhang befindet sich eine Schritt-für-Schritt-Anleitung die es Laien ermöglicht 3D-Modelle aufzunehmen und zu exportieren.

## 2. Vorstellung der vorhandenen Hardware

Die Hardware besteht im wesentlichen aus den Komponenten in Abbildung 2.1.  
**(TODO: NUMMERN ODER FARBEN IN BILD!)**

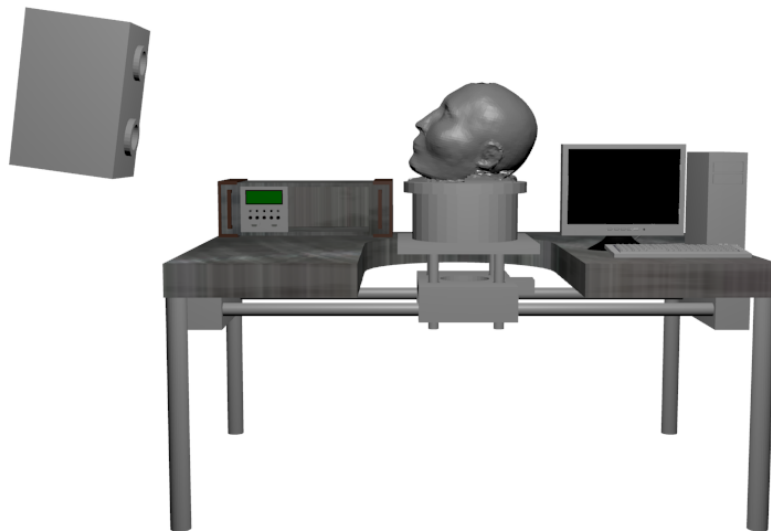


Abbildung 2.1.: Blick auf den Aufbau

1	<a href="#">2.1</a> Computer
2	<a href="#">2.2</a> 3D-Laserscanner VI-900
3	<a href="#">2.3</a> Ansteuerung für den Drehtisch
4	<a href="#">2.4</a> Mikrocontroller

Tabelle 2.1.: Aufbau

### 2.1. Computer

Zur Verfügung steht ein IBM kompatibler x86 Standard PC mit einer *SCSI*- und einer *RS-232-Schnittstelle*. Auf diesem ist die Erfassungssoftware *RapidForm2004*

## 2. Vorstellung der vorhandenen Hardware

---

installiert. Die SCSI Schnittstelle wird zur Kommunikation mit dem 3D-Laserscanner und die RS-232-Schnittstelle zur Kommunikation mit einer Schrittmotorsteuerung genutzt.

### 2.2. 3D-Laserscanner VI-900

Der 3D-Laserscanner *VI-900* der Firma Minolta besteht, wie auf Abbildung 2.2 zu sehen, aus einem Lasertriangulator (unten) und einer Kamera (oben). Das System lässt sich über eine SCSI-Schnittstelle ansprechen und konfigurieren. Zur mobilen Nutzung kann das Gerät auch auf der Rückseite bedient werden. Aufgenommene Daten können auf einer CF-Karte gespeichert werden. Im Projekt wurde jedoch lediglich die direkte Ansteuerung via SCSI genutzt.

Der VI-900 digitalisiert Objekte durch ein Laser-Lichtschnittverfahren. Das vom Objekt reflektierte Licht wird von einer CCD-Flächenkamera erfasst, nach Ermittlung der Distanzwerte (Z-Achse) mittels Laser-Triangulation werden die 3D-Daten erstellt. Der Laserstrahl wird mit Hilfe eines hochpräzisen galvanischen Spiegels über das Objekt projiziert, pro Scan werden 640 x 480 Einzelpunkte erfasst. Minolta [2012] Die Technischen Daten befinden sich im Anhang in Tabelle A.3



Abbildung 2.2.: VI-900 - 3D-Scanner

#### 2.2.1. Lasertriangulator Prinzip

Ein Lasertriangulator besteht, wie in Abbildung 2.3 zu sehen, aus einem Laser, einem Linsensystem und im einfachsten Fall, aus einer Pixeldetektorzeile. Der Laser strahlt auf ein Objekt und je nach Entfernung des Objektes wird das Streulicht unter

## 2. Vorstellung der vorhandenen Hardware

---

einem anderen Winkel zurückgestrahlt. Das Streulicht wird durch die Linsen auf den Pixeldetektor abgebildet. Über die Position des Laserspots auf dem Pixeldetektor lässt sich auf die Entfernung des Objektes schließen.

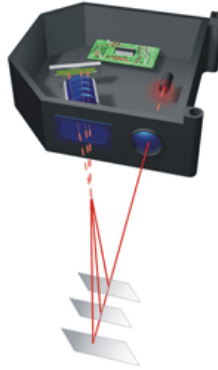


Abbildung 2.3.: Prinzip: Laser-Triangulation

## 2.3. Drehtisch und Ansteuerung

### 2.3.1. Drehtisch

Der Tisch in dem der Drehtisch verbaut ist, ist eine Eigenkonstruktion der Werkstatt des RheinAhrCampus. Er besteht aus einer massiven Edelstahl-Arbeitsplatte, welche auf 4 Füßen ruht. Aus dieser ist ein Rechteck mit aufgesetztem Halbkreis ausgeschnitten. In diesem Ausschnitt befindet sich der Drehtisch (siehe Abbildung 2.4). Er ist auf einem Schienensystem gelagert. Mit dem Schienensystem kann der Drehtisch in der Vertikalen positioniert werden. Mit einem Schrittmotor lässt sich der Drehtisch zusätzlich in der Höhe verstellen. Die Höhenverstellung wird mit einem **Schneckengetriebe** realisiert. Ein weiterer Schrittmotor ist für die Drehung des Tisches zuständig. Der Tisch ist über ein **Harmonic-Drive-Getriebe** mit dem Schrittmotor verbunden. Das Übersetzungsverhältnis des Getriebes beträgt 1:50.

### 2.3.2. Spannungsversorgung

Die Schrittmotorkarten werden von einem PC-Netzteil gespeist. Die **Logikbausteine** werden mit 5V gespeist, zusätzlich werden die Schrittmotorkarten mit 12V für die Schrittmotoren gespeist. Die Kabel sind direkt an die Verbindungsleisten gelötet. Dies verhindert das einfache Ausbauen der Spannungsversorgung und die einfache Erweiterung um neue Einschubkarten.

## 2. Vorstellung der vorhandenen Hardware

---

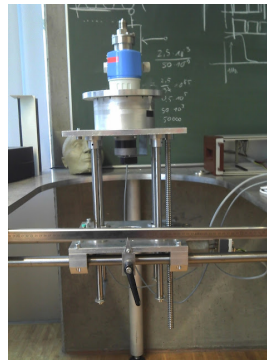


Abbildung 2.4.: Drehtisch

### 2.3.3. Schrittmotoren

(TODO: MOTOREN BESCHREIBEN! TECHNISCHE DATEN!)

(TODO: SCHRITTE, SPANNUNGEN. VERDRAHTUNG.)

### 2.3.4. Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19"-Einschübe realisiert, siehe Abbildung 2.5. Für jeden Schrittmotor wird ein Einschub benötigt. Die Einschübe sind Produkte der Firma R+S. Mittels RS-232 Schnittstelle lassen sich die Karten konfigurieren und ansteuern. Die Konfiguration und Ansteuerung erfolgt über einen vorgegeben ASCII<sup>1</sup> Befehlssatz. Der Befehlssatz befindet sich im Kapitel A.2. Es können zwei oder mehr Karten als Daisy-Chain<sup>2</sup> in Reihe geschaltet werden.

### 2.3.5. Motorverkabelung

Die Schrittmotoren benötigen ein mindestens 4-adriges Kabel. Das Kabel für den Schrittmotor, der für die Rotation zuständig ist, war bereits gefertigt. Ein Kabel zwischen Schrittmotor und Schrittmotorkarte zur Höhenverstellung und für die Endschalter ist nicht vorhanden.

---

<sup>1</sup>Der American Standard Code for Information Interchange (ASCII, alternativ US-ASCII, oft [æski] ausgesprochen) ist eine 7-Bit-Zeichenkodierung [Wikipedia \[2012a\]](#)

<sup>2</sup>Als Daisy Chain (englisch, wörtlich „Gänseblümchenkette“) bezeichnet man eine Anzahl von Hardware-Komponenten, welche in Serie miteinander verbunden sind (meist in sogenannten Bussystemen in der Automatisierungstechnik). [Wikipedia \[2012b\]](#)

## 2. Vorstellung der vorhandenen Hardware

---

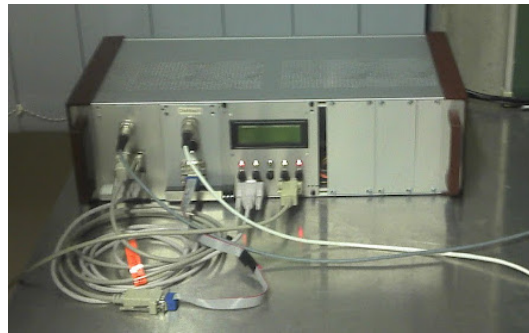


Abbildung 2.5.: Ansteuerung im 19"-Rack

### 2.3.6. Endschalter

Die Schrittmotorkarten unterstützen das Abschalten der Motoren wenn ein sogenannter Endschalter ausgelöst wird. Dies sind im allgemeinen mechanische Schalter die ausgelöst werden wenn der Tisch sich dem Ende des Arbeitsbereiches nähert. Dies verhindert eine Beschädigung des Aufbaus.

Im Aufbau sind bereits induktive Endschalter der Firma *Pepperl+Fuchs* verbaut. Diese werden durch einen Metallstutzen ausgelöst. Dieser ist jedoch schlecht positioniert oder ungenügend lang. Würde der Drehtisch über seine Grenzen hinaus in der Höhe verstellt werden, würden die Endschalter nicht rechtzeitig ausgelöst werden und der Aufbau würde beschädigt werden.

## 2.4. Mikrocontroller

Ein **Mikrocontroller** vereint, in einem IC, die wichtigsten Komponenten um komplexe technische Probleme leicht lösen zu können. Dazu gehören z.B. CPU, Flash-Speicher, Arbeitsspeicher, Register, Ports, ADC, DAC und mehr. Einen schematischen Überblick über die Komponenten eines Mikrocontrollers bietet das Blockdiagramm in Abbildung 2.6.

Für unterschiedliche Aufgaben sind unterschiedliche Mikrocontroller geeignet.

Es steht ein ATmega8515 [Corporation \[2012b\]](#) im DIL-Gehäuse zur Verfügung. Dieser hat 8 Kbyte Flash, drei externe Interrupts, eine serielle Schnittstelle und kann mit bis zu 16 MHz betrieben werden. Dieser ist geeignet um sich in die Programmierung mit **C** einzufinden und eine serielle Schnittstelle anzusteuern.

**(TODO: PORTS, REGISTER ERKLÄREN)**

## 2. Vorstellung der vorhandenen Hardware

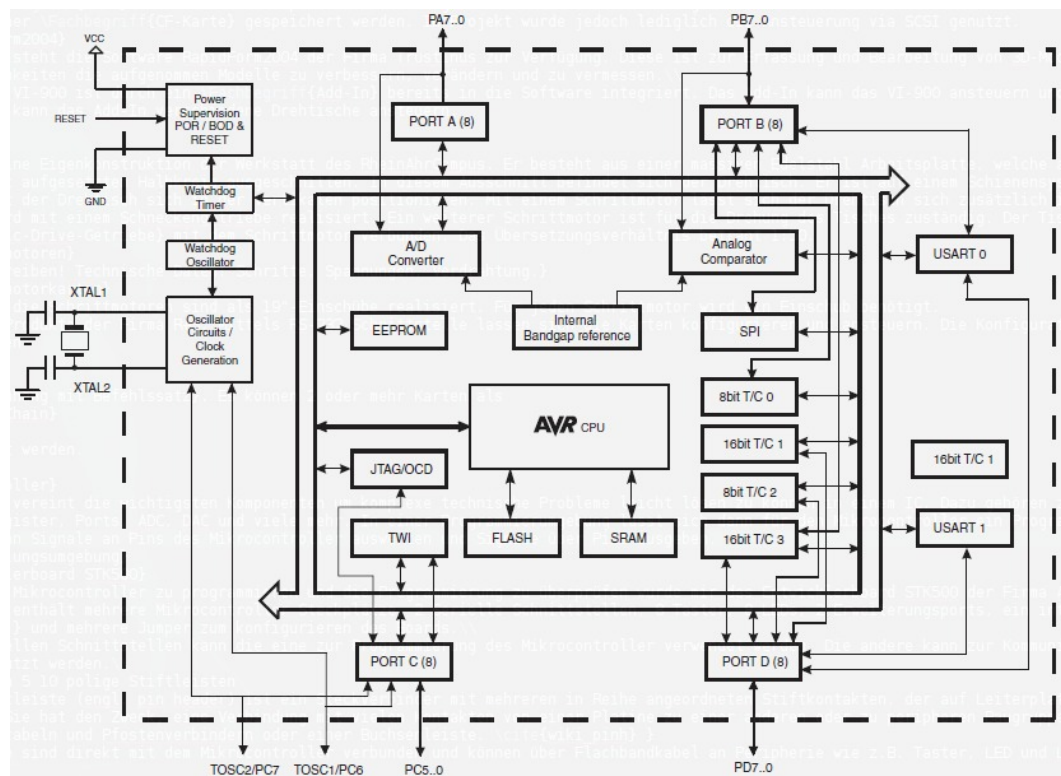


Abbildung 2.6.: Block Diagram: Mikrocontroller  
[Atm 2011]

### 2.4.1. Entwicklerboard STK500

Um den Mikrocontroller zu programmieren und die Programmierung zu überprüfen, wird das **Entwicklerboard STK500** (siehe Abbildung 2.7) der Firma Atmel verwendet. Das Board enthält mehrere Mikrocontroller-Steckplätze, 2 serielle Schnittstellen, 8 Taster, 8 LEDs, 2 Erweiterungsports, eine **ISP**<sup>3</sup> Programmierschnittstelle und mehrere Jumper zum Konfigurieren des Boards.

Von den beiden seriellen Schnittstellen kann die eine zur Programmierung des Mikrocontrollers verwendet werden. Die andere kann zur Kommunikation mit dem Mikrocontroller genutzt werden.

Auf dem Board stehen fünf 10 polige Stiftleisten zur Verfügung. Diese sind direkt mit den Ein- und Ausgabe Pins, den sogenannten *Ports*, des Mikrocontroller verbunden und können über Flachbandkabel mit **(TODO: PERIPHERIE)** wie z.B. Taster, LED, LC-Displays oder seriellen Schnittstellen verbunden werden.

<sup>3</sup>In System Programmer

## 2. Vorstellung der vorhandenen Hardware

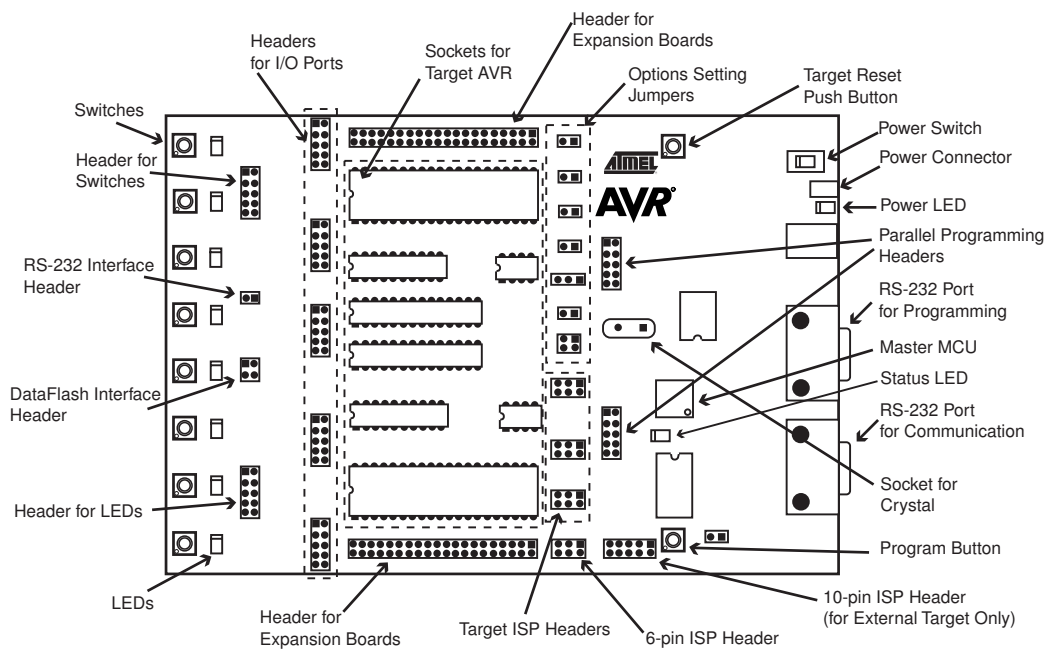


Abbildung 2.7.: Schema: STK500  
[Atm 2003]

### 2.4.2. AVRISP mkII

Das AVRISP mkII ist ein USB-basierter **In-System-Programmer**. Dieser kann anstelle des RS-232 basierten Programmiersystem des STK500 verwendet werden. Die Übertragungsgeschwindigkeit des AVRISP mkII ist wesentlich höher als die der Seriellen Schnittstelle. Der AVRISP mkII lässt sich einfach an den Programmierport, eine 6-Polige Stiftleiste, des STK500 anschließen.

### 2.4.3. MAX232

Um die Serielle Schnittstelle am Mikrocontroller nutzen zu können müssen die Spannungspegel auf die des RS-232 Standard gewandelt werden. Dazu befindet sich auf dem STK500 der **Pegelwandler** MAX232. Dieser wandelt die Spannungspegel des Mikrocontroller (typ. 0 V – 5 V **TTL**<sup>4</sup>) auf die Spannungspegel des RS-232 Standards (typ. -12 V – +12 V).



## 3. Vorstellung der vorhandenen Software

### 3.1. RapidForm2004

Zur Erfassung von 3D-Modellen am PC steht die Software *RapidForm2004* der Firma *INUS Technology Inc.* zur Verfügung. Diese ist zur Erfassung und Bearbeitung von 3D-Modellen gedacht. Sie bietet umfangreiche Möglichkeiten die aufgenommenen Modelle zu verbessern, zu verändern, zu vermessen und in verschiedene Formate zu exportieren.

Mittels eines **Add-In** kann der VI-900 angesteuert und aufgenommenen Daten ausgelesen werden. Weiterhin kann das Add-In über eine RS-232-Schnittstelle verschiedene Drehtische ansteuern.

### 3.2. Entwicklungsumgebung

Die von Atmel bereitgestellte Entwicklungsumgebung *AVR Studio 5* besteht aus einem Editor, einem Compiler und einer Programmiersoftware. Der Editor bietet Komfortfunktionen wie **Syntaxhighlighting**, Autovervollständigung und Projektmanagement. Der Compiler übersetzt den Quelltext in einen maschinenlesbaren Code und die Programmiersoftware kann diesen auf einen Mikrocontroller spielen.

### 3.3. Terminalprogramme

Zur Kommunikation über die RS-232-Schnittstelle steht das Programm *Hypterminal* der Firma Microsoft zur Verfügung.

## 4. Zeitlicher Arbeitsablauf

Dieses Kapitel spiegelt den chronologischen Ablauf des Projektes wieder und zeigt die Schritte auf, die notwendig waren um den Mikrocontroller so zu programmieren, dass dieser die Kommunikation zwischen RapidForm2004 und der vorhandenen Schrittmotorkarte ermöglicht. Er sozusagen als Übersetzer für die unterschiedlichen Befehlssätze von RapidForm2004 und dem der Schrittmotorkarte fungiert.

### Hinweis

Die Codelistings in diesem Kapitel sind thematisch zusammen gefasst und gekürzt um die Lesbarkeit und das Verständnis zu gewährleisten. Ein komplettes Codelisting des Hauptprogramms befindet sich im Anhang ???. Der komplette Code, mit allen Bibliotheken, liegt dem Praxisbericht als CD oder Archiv bei.

Kapitel 4.1 beschreibt die Programmierung, des Mikrocontrollers, welche die notwendigen Grundvoraussetzungen für dieses Projekt schafft. Das Ziel dieser Programmierung besteht darin die geforderten Komponenten, LEDs, LC-Display, Taster und serielle Schnittstellen im Mikrocontroller nutzbar zu machen.

Kapitel 4.2 beschreibt die Erarbeitung der Befehlssätze die die Software RapidForm2004 enthält um mit den von ihr unterstützten Schrittmotorkarten zu kommunizieren. Auch der Befehlssatz zur Kommunikation zwischen dem Mikrocontroller und der Schrittmotorkarte wird beschrieben.

Kapitel 4.3 beschreibt wie der Mikrocontroller diesen Befehlssatz für die Kommunikation mit der vorhandenen Schrittmotorkarte nutzt.

Kapitel 4.4 gibt die Schritte zur Entwicklung und Verbesserung der Hardware, um diese so zu erweitern, dass sie den Vorgaben entspricht, wieder.

Kapitel 4.5 erläutert die Kommunikation zwischen dem Mikrocontroller und RapidForm2004.

Kapitel 4.6 beschreibt, wie die vorherigen Kapitel zusammenspielen, sodass eine reibungslose Kommunikation zwischen RapidForm2004 und der vorhandenen Schrittmotorkarte möglich ist.

#### 4. Zeitlicher Arbeitsablauf

Kapitel ?? beschreibt dann das erstellen des Platinenlayouts und das Fertigen des Einschubs.

### 4.1. Bereitstellen grundlegender Funktionalitäten

Im ersten Schritt ging es darum, den Mikrocontroller so zu programmieren, dass dieser die für dieses Projekt grundlegenden Funktionalitäten bereitstellen kann.

Der Mikrocontroller befand sich vorerst auf dem `STK500` (siehe Kapitel 2.4.1). Um dessen Komponenten im Mikrocontroller nutzen zu können, müssen dafür Register initialisiert werden oder Funktionalitäten wie z.B. Bibliotheken für das LC-Display bereit gestellt werden.

Die folgenden Kapitel beschreiben den Programmcode, der notwendig ist um diese Funktionalitäten bereit zu stellen.

#### 4.1.1. Taster

Um die Taster des STK500 im Mikrocontroller nutzen zu können müssen diese mit dem Mikrocontroller verbunden und entprellt<sup>5</sup> werden.

Dazu muss die Stiftleiste von PortA mit der Stiftleiste für die Taster verbunden werden. Das Entprellen der Taster wird softwareseitig realisiert. Dies bietet sich bei einem Mikrocontroller an. Dazu gibt es vorgefertigte Bibliotheken die genutzt werden können. Im Projekt wurde die Bibliothek `Debounce.h` [Dannegger \[2012\]](#) von Peter Dannegger genutzt. Sie ist sehr komfortabel und funktionsreich und basiert auf `Timer-Interrupts`. Um sie zu nutzen wird die Datei `Debounce.h` in das Projektverzeichnis kopiert und mit Zeile 1 des Codelisting 4.1 in das Programm eingebunden. Die Zeilen 3-10 spiegeln die Funktion zum Initialisieren der Bibliothek wieder. Diese Zeilen müssen auf den jeweils verwendeten Mikrocontroller angepasst werden.

Durch die Verwendung der Bibliothek war es möglich Funktionen wie z.B. `get_key_press()` zu nutzen um den Status der Taster prellfrei auszulesen und diese Information für Entscheidungen im Programm zu verwenden.

Listing 4.1: Taster

```
1 #include "Debounce.h"
2
3 void debounce_init (void) {
4     KEY_DDR &= ~ALL_KEYS; // configure key port for input
```

<sup>5</sup>Als `Prellen` bezeichnet man das ungewollte mehrfache Schalten eines mechanischen Schalters bei einem einzelnen Schaltvorgang.

#### 4. Zeitlicher Arbeitsablauf

```
5     KEY_PORT |= ALL_KEYS; // and turn on pull up resistors
6     TCCR0B = (1 << CS02) | (1 << CS00); // divide by 1024
7     TCNT0 = (uint8_t) (int16_t) -(F_CPU / 1024 * 10 * 10e-3 + 0.5); // preload for 10
        ms
8     TIMSK0 |= 1 << TOIE0; // enable timer interrupt
9     sei();
10 }
11
12 if (get_key_press(1 << KEY0) || get_key_rpt(1 << KEY0)){
13     lcd_puts("Betrete Menue!\n");
14     menu_enter(&menu_context, &menu_main);
15 }
```

##### 4.1.2. LEDs

LEDs sollen im Programmablauf genutzt werden können, um z.B. Fehler zu signalisieren.

Dazu muss zuerst die Stiftleiste von *PortB* mit der LED Stiftleiste des STK500 verbunden werden. Um LEDs an *PortB* betreiben zu können müssen die entsprechenden Pins im Register `DDRB` als Ausgänge definiert werden. Dies geschieht in Zeile 3 des Codelisting 4.2. Die Bibliothek zum Entprellen der Taster nutzt die Variablen `LED_DDR` und `LED_PORT`. Diese Variablen werden auch hier genutzt um auf die Register zuzugreifen. Dies gewährleistet eine bessere Übersicht. Die Werte im 8-Bit Register `LED_PORT` spiegeln die Spannungen an den Pins des `PortB` am Mikrocontroller wieder. Da die LEDs auf dem STK500 mit `active-low-Logik` betrieben werden, muss das jeweilige Bit gelöscht, also auf "0", gesetzt werden damit die LED leuchtet. Um alle Bits in einem Register zu verändern kann das Register mit einem 2-stelligen Hex-Wert(8-Bit) oder einem 8 stelligen binären Bitmuster beschrieben werden. In Zeile 4 werden alle Bits mit dem Hex-Wert `0xFF` auf "1" gesetzt und somit alle LEDs ausgeschaltet. Um ein einzelnes Bit zu verändern, können die Anweisungen in den Zeilen 5 und 6 verwendet werden. Dabei steht das "X" in *PBX* für die x-te Stelle im Register die gesetzt oder gelöscht werden soll.

Es ist damit möglich im Programmablauf einzelne LEDs anzusteuern.

Listing 4.2: LEDs

```
1 #define LED_DDR DDRB
2 #define LED_PORT PORTB
3 LED_DDR = 0xFF; // LED Port Richtung definieren (Ausgang)
4 LED_PORT = 0xFF; // LEDs ausschalten
5 LED_PORT &= ~(1 << PBX); // loescht Bit an PortB – LED an
```

#### 4. Zeitlicher Arbeitsablauf

```
6 LED_PORT |= ((1 << PBX)); // setzt Bit an PortB – LED aus
```

### 4.1.3. LCD ansteuern

Um den aktuellen Status des Motor komfortabel in Textform anzeigen zu können und die Schrittmotorkarte *Menü basiert* ansteuern zu können wird ein LC-Display verwendet. Das verwendete Display ist alpha numerisch und kann 4x20 Zeichen anzeigen.

Die meisten LC-Displays werden auf die gleiche Weise angesteuert. Hier gibt es fertige Bibliotheken die frei genutzt werden können. Im Projekt wurde die von Peter Fleury [Fleury \[2012\]](#) verwendet. Die Bibliothek wird heruntergeladen und die Dateien `lcd.c` und `lcd.h` in das Projektverzeichnis entpackt. Die Bibliothek wird mit `#include "lcd.h"` eingebunden. In der `lcd.h` müssen noch die Daten des Displays eingegeben werden (siehe Codelisting 4.3 Zeilen 2–9).

Danach kann das Display im Programm mit den Befehlen aus Zeile 11–20 angesteuert werden.

Listing 4.3: lcd.h (Auszug)

```
1 #define LCD_CONTROLLER_KS0073 0 /**< Use 0 for HD44780 controller, 1 for KS0073  
   controller */  
2 #define LCD_LINES 4 /**< number of visible lines of the display */  
3 #define LCD_DISP_LENGTH 19 /**< visibles characters per line of the display */  
4 #define LCD_LINE_LENGTH 0x40 /**< internal line length of the display */  
5 #define LCD_START_LINE1 0x00 /**< DDRAM address of first char of line 1 */  
6 #define LCD_START_LINE2 0x40 /**< DDRAM address of first char of line 2 */  
7 #define LCD_START_LINE3 0x14 /**< DDRAM address of first char of line 3 */  
8 #define LCD_START_LINE4 0x54 /**< DDRAM address of first char of line 4 */  
9 #define LCD_WRAP_LINES 1 /**< 0: no wrap, 1: wrap at end of visibile line */  
10  
11 extern void lcd_init(uint8_t dispAttr);  
12 extern void lcd_clrscr(void);  
13 extern void lcd_home(void);  
14 extern void lcd_gotoxy(uint8_t x, uint8_t y);  
15 extern void lcd_putc(char c);  
16 extern void lcd_puts(const char *s);  
17 extern void lcd_puts_p(const char *progmem_s);  
18 extern void lcd_command(uint8_t cmd);  
19 extern void lcd_data(uint8_t data);  
20 #define lcd_puts_P(__s) lcd_puts_p(PSTR(__s))
```

#### 4.1.4. RS-232-Schnittstelle

RS-232 ist der Name der am häufigsten verwendeten seriellen Schnittstelle um Daten zwischen zwei elektronischen Geräten hin und her zu senden. [Mikrocontroller.net \[2012\]](#)

Auf dem STK500 ist bereits eine serielle Schnittstelle vorbereitet. Um diese nutzen zu können, müssen die Pins 3 und 4 des *PortC* (erster UART) des Mikrocontrollers mit der Stiftleiste *Rx/Tx* auf dem STK500 verbunden werden. Eine weitere Schnittstelle wurde auf einem Steckbrett aufgebaut. Diese wurde mit den Pins 1 und 2 des *PortC* (zweiter UART) des Mikrocontrollers verbunden. Um die Schnittstellen im Mikrocontroller nutzen zu können müssen diese noch durch setzen von Bits in den entsprechenden Registern des Mikrocontrollers aktiviert werden.

Das Codelisting 4.4 teilt sich in 4 wesentliche Bereiche:

- Zeilen 1 – 2: Setzen der Baudrate und einbinden der benötigten Bibliotheken.
- Zeilen 4 – 16: Initialisieren der Schnittstellen durch setzen der richtigen Bits in den entsprechenden Registern.
- Zeilen 17 – 32: Funktionen zum Senden von Daten
- Zeilen 34 – 65: Funktionen zum Empfangen von Daten

Listing 4.4: RS-232

```
1 #define BAUD 9600
2 #include <util/setbaud.h>
3
4 void    uart_init          () { // Initialisierung der Schnittstellen
5     UBRR0H = UBRRH_VALUE; // UART 0 – IN (Rapidform Software/Terminal)
6     UBRR0L = UBRL_VALUE;
7     UCSR0C = (3 << UCSZ00);
8     UCSR0B |= (1 << TXEN0); //Transmitter Enabled
9     UCSR0B |= (1 << RXEN0); // UART RX einschalten
10
11     UBRR1H = UBRRH_VALUE; // UART 1 – OUT (Stepper Karte/Drehtisch)
12     UBRR1L = UBRL_VALUE;
13     UCSR1C = (3 << UCSZ00);
14     UCSR1B |= (1 << TXEN1); //Transmitter Enabled
15     UCSR1B |= (1 << RXEN1); // UART RX einschalten
16 }
17 void    uart_put_charater  (unsigned char c, int dir) { // Versenden von einzelnen
    Zeichen
```

## 4. Zeitlicher Arbeitsablauf

```

18     if (dir == D_RapidForm) { // To Rapidform
19         while (!(UCSR0A & (1 << UDRE0))) {} //warten bis Senden moeglich
20         UDR0 = c; // sende Zeichen
21     }
22     else { // To Stepper
23         while (!(UCSR1A & (1 << UDRE1))) {} //warten bis Senden moeglich
24         UDR1 = c; // sende Zeichen
25     }
26 }
27 void uart_put_string (char *s, int dir) { //Versenden von ganzen Strings
28     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)"
29     {
30         uart_put_charater(*s, dir);
31         s++;
32     }
33 }
34 int uart_get_character (int dir) { // Empfang einzelner Zeichen
35     if (dir == D_RapidForm) {
36         while (!(UCSR0A & (1 << RXC0)))
37             // warten bis Zeichen verfuegbar
38             ;
39         return UDR0; // Zeichen aus UDR an Aufrufer zurueckgeben
40     }
41     if (dir == D_Stepper) {
42         while (!(UCSR1A & (1 << RXC1)))
43             // warten bis Zeichen verfuegbar
44             ;
45         return UDR1; // Zeichen aus UDR an Aufrufer zurueckgeben
46     }
47     return -1;
48 }
49 void uart_get_string (char * string_in, int dir) { // Empfang von ganzen Strings
50     char c;
51     int i = 0;
52     do { // Schleife zum zusammenbauen einzelner Zeichen zu einem String
53         c = uart_get_character(dir); // Einzelnes Zeichen holen
54         if (c != '\r') { // Wenn kein Zeilenende Zeichen
55             *string_in = c; // Aktuelles Zeichen im String = Zeichen
56             string_in += 1; // Position im String hochzaehlen
57             i++; // Schleifenzaehler hoch zaehlen.
58         }
59     } while (i < 100 && c != '\r' && c != '\n'); // Maximal bis 100 Zeichen oder
        Zeilenende oder Zeilenvorschub
60     *string_in = '\0'; // Stringende Null-Terminieren
    
```

#### 4. Zeitlicher Arbeitsablauf

```
61     if (dir == D_Stepper)           // Dateneingangs LEDs ausschalten
62         LED_PORT |= ( 1 << LED3 );
63     else
64         LED_PORT |= ( 1 << LED2 );
65 }
```

Damit stehen die essentiellen Funktionen `uart_put_string(dir)` und `uart_get_string(dir)` zur Verfügung. Mit diesen kann der Mikrocontroller über die serielle Schnittstelle Strings senden und empfangen. Der Parameter `dir` gibt dabei die Schnittstelle an über die gesendet oder empfangen werden soll.

### 4.2. Befehlssätze

Das zu erreichende Ziel bestand darin, dass RapidForm2004 mit dem Mikrocontroller und dieser mit der Schrittmotorkarte kommunizieren können sollte. Die Kommunikation läuft dabei über Befehle ab, die über die serielle Schnittstelle gesendet werden. Jede Schrittmotorkarte verwendet eigenen Befehle. Alle Befehle für eine Schrittmotorkarte werden im Folgenden als Befehlssatz bezeichnet. Die Software RapidForm2004 kennt mehrere Befehlssätze um verschiedene Schrittmotorkarten anzusteuern. Der Befehlssatz der vorhandenen Schrittmotorkarten zum ansteuern der Motoren des Drehtisches ist jedoch nicht in RapidForm2004 vorhanden.

Nun soll der Mikrocontroller sowohl mit RapidForm2004 als auch mit der ersten der vorhandenen Schrittmotorkarten kommunizieren. Befehle an die zweite Schrittmotorkarte werden über die erste gesendet. Um mit beiden Seiten kommunizieren zu können muss der Mikrocontroller den Befehlssatz der vorhanden Schrittmotorkarten und zumindest einen der Befehlssätze aus RapidForm2004 kennen. Außerdem muss er wissen welche Antwort RapidForm2004 auf einen gesendeten Befehl erwartet.

In der ersten Phase wurde die Software *Free Serial Port Monitor* verwendet um die Kommunikation zwischen RapidForm2004 und dem Mikrocontroller abzuhören. Dies hatte jedoch den Nachteil, das RapidForm2004 erst dann den nächsten Befehl sendete, wenn der erste mit der erwarteten Antwort quittiert wurde. Die Befehle die RapidForm erwartete, konnten zwar teilweise aus den Betriebsanleitungen der Schrittmotorsteuerungen entnommen werden, dieses Vorgehen war jedoch sehr mühsam. Nach einiger Zeit kam ich auf die Idee **Reverse-Engineering** zu verwenden und war dadurch in der Lage alle Befehlssätze und erwartete Antworten aus einer ausführbaren Datei von RapidForm2004 auszulesen.

Das Codelisting 4.5 zeigt einen Auszug für den Befehlssatz eines Isel Schrittmotors. Im Anhang befinden sich die Befehlssätze aller Schrittmotorkarten. Somit stehen



4. Zeitlicher Arbeitsablauf

die Befehlssätze aller Schrittmotorsteuerungen zur Verfügung. Diese wurden in einer Textdatei gespeichert und werden später im Programm verwendet. Dadurch sind alle Befehlssätze und die Antworten die RapidForm2004 auf einen daraus ausgesendeten Befehl erwartet, bekannt.

**Hinweis**

In Codelistings und im Quelltext wird teilweise noch die Bezeichnung *Protokolle* statt *Befehlssätze* verwendet. Diese ist gleichbedeutend.

Listing 4.5: Befehlssatz aus Rapidform: Isel

```

1 model "isel(RF-1)"
2 port "9600" "n81h"
3 init "@01\r" "0"
4 finish "@0M0\054+600\r" "0"
5 arot "@0M%d\054+600\r" "0"
6 stop "" "0"
7 home "@0M0\054+600\r" "0"
8 step "-0.0173076" "-8000000" "8000000"
9 timeout "60"
10 firsttimeout "10"

```

## 4.3. Kommunikation mit der Schrittmotorsteuerung

### 4.3.1. Befehle senden

Im nächsten Schritt geht es darum, Befehle an die Schrittmotorkarte zu versenden. Da es nicht möglich ist, für jeden Befehl eine eigene Taste zu verwenden, wird eine Menü basierte Steuerung mittels des LC-Display verwendet. Im Menü lässt sich mit den Tasten *Hoch* *Runter* *Ok* und *Zurück* navigieren.

Analog wie beim LC-Display und bei den Tastern wird hier eine vorhandene Bibliothek genutzt. Um die Bibliothek verwenden zu können musste die Menüstruktur den Bedürfnissen des Projekts angepasst werden und die Funktionen zum Ausgeben von Text auf dem LC-Display und zum Versenden von Befehlen über die RS-232-Schnittstelle, aus den vorangegangenen Kapiteln, bekannt gemacht werden. Dies geschieht in der Datei `tinymenu/tinymenu.h`.

Die Zeilen 1–6 des Codelisting 4.6 dienen zum Einbinden der benötigten Bibliotheken. Die Zeilen 8-16 zeigen eine vereinfachte Struktur des Hauptprogramms. Wird

#### 4. Zeitlicher Arbeitsablauf

ein Taster gedrückt, wird dies durch die `get_key_press()`-Funktion, bekannt aus Kapitel 4.1.1, erkannt und die entsprechende Menü Funktion aufgerufen.

Listing 4.6: Menü

```
1 #define MCU_CLK F_CPU
2 #include "tinymenu/spin_delay.h"
3 #define CONFIG_TINYMENU_USE_CLEAR
4 #include "tinymenu/tinymenu.h"
5 #include "tinymenu/tinymenu_hw.h"
6 #include "mymenu.h"
7
8 int main(void) {
9     while (1) {
10         if (get_key_press(1 << KEY0)) menu_enter(&menu_context, &menu_main);
11         if (get_key_press(1 << KEY1)) menu_prev_entry(&menu_context);
12         if (get_key_press(1 << KEY2)) menu_next_entry(&menu_context);
13         if (get_key_press(1 << KEY4)) menu_select(&menu_context);
14         if (get_key_press(1 << KEY4)) menu_exit(&menu_context);
15     }
16 }
17
18 void menu_puts (void *arg, char *name) {
19     uart_put_string(arg, D_Stepper);
20     lcd_clrscr();
21     lcd_puts("Send: ");
22     lcd_puts(arg);
23     lcd_puts("\n");
24     ms_spin(100);
25     //if ((UCSR1A & (1 << RXC1)))
26     uart_rx(D_Stepper);
27     ms_spin(1000);
28 }
```

Folgende Menüpunkte wurden realisiert:

Listing 4.7: Menü Baum

```
1 Main Menu
2   Bewegen – Rotation
3     +90
4     -90
5     +10.000 Schritte
6     -10.000 Schritte
7     Gehe zum Ursprung
8   Bewegen – Hoehe
9     +500000
```

#### 4. Zeitlicher Arbeitsablauf

```
10  -500000
11  +1000000
12  -1000000
13  Gehe zum Ursprung
14  Konfigurieren
15  Motorstatus
16  Setze Ursprung
17  Write to EEPROM
18  Newline 1
19  Parameter Auslesen
```

Wird einer der Menüpunkte aufgerufen, wird die im Menüpunkt hinterlegte Funktion mit dem hinterlegten Parameter aufgerufen. Wird beispielsweise der Befehl `+90` ausgewählt, wird die hinterlegte Funktion `menu_puts(arg, name)` (Codelisting 4.6 Zeile 18-28) mit dem hinterlegten Wert aufgerufen. Diese sendet dann mit der aus Kapitel 4.1.4 bekannten Funktion `uart_puts(arg, dir)` einen Befehl an die Schrittmotorsteuerung.

Es ist nun somit möglich, mit Tastern, vordefinierte Befehle aus dem Menü auszuwählen und an die Schrittmotorsteuerung zu senden.

#### 4.3.2. Antworten Empfangen und speichern

Die Schrittmotorsteuerung antwortet auf Befehle mit einem `String`. In diesem Arbeitsschritt wird die Funktionalität zum Empfangen von Antworten der Schrittmotorsteuerung auf Befehle des Mikrocontrollers hergestellt. Diese Antworten sollen in einem String gespeichert und im nächsten Schritt an eine *Auswerte-Funktion* weiter gegeben werden.

Dazu wird in der Hauptschleife des Programms ständig das Eingangsregister der ersten seriellen Schnittstelle abgefragt (siehe Codelisting 4.8 Zeile 10–13). Dieses Vorgehen bezeichnet man als `Polling`. Sind Daten im Register vorhanden, wird `LED3` eingeschaltet und die Funktion `uart_rx(int dir)` mit dem Parameter `D_Stepper` aufgerufen. Der übergebene Parameter gibt an, dass der Befehl von der, für die Schrittmotorkarte zuständigen Schnittstelle empfangen wurde. Dadurch wird festgelegt, dass der empfangene `String` aus dem richtigen Datenempfangsregister ausgelesen wird und wie er weiterverarbeitet wird. Die Funktion `uart_rx(dir)` liest dann das Empfangsregister mit der aus Kapitel 4.1.4 bekannten Funktion `uart_get_string(str_rx, dir)` aus und schreibt den empfangenen String in die Variable `str_rx` (Codelisting 4.8, Zeile 7). In einer `if-Abfrage` wird entschieden, von welcher Schnittstelle der empfangene Befehl kam. Da `D_Stepper` übergeben wurde, wird der `if-Teil` der Abfrage

#### 4. Zeitlicher Arbeitsablauf

ausgeführt. In dieser wird der empfangene String an die *Auswerte-Funktion* für die Schrittmotorkarte (Codelisting 4.8, Zeile 15-45) übergeben. Durch diesen Teil des Programms ist es nun möglich Antworten der Schrittmotorkarte zu empfangen, in dem String `str_rx` zu speichern und an die Auswerte-Funktion `switch_Stepper(str_rx)` zu übergeben.

Listing 4.8: RS-232 Empfang

```
1 if ((UCSR1A & (1 << RXC1))) {
2     LED_PORT &= ( 1 << LED3 );
3     uart_rx(D_Stepper);
4 }
5
6 void    uart_rx                (int dir) {
7     uart_get_string(str_rx, dir);
8     if (dir == D_Stepper)
9         switch_Stepper(str_rx);
10    else {
11        ...
12    }
13 }
```

#### 4.3.3. Antworten auswerten

Die Funktion zum Auswerten empfangener Strings spielt eine zentrale Rolle im Projekt. Diese Funktion ermöglicht es, ankommende Strings im Mikrocontroller gegen die bekannten Antworten zu prüfen und eine entsprechende Reaktion auszuführen.

In der Auswerte-Funktion wird der übergebene String mittels der Funktion `FindStringInArray(str_rx, pOptions, length)` (Codelisting 4.9) gegen ein `Array` (Codelisting 4.10, Zeile 2) mit bekannten Befehlen geprüft. Ist der String in diesem Array vorhanden, wird die Position des Strings im Array zurückgegeben. Ansonsten "99". In einer anschließenden `switch/case-Struktur` wird dann der Position im Array ein bestimmtes Verhalten des Mikrocontrollers zugeordnet. Wird beispielsweise der String `#` empfangen, wird Position `0` zurück gegeben und auf dem LC-Display wird *Erfolgreich* ausgegeben.

Durch diese Funktion kann nun auf Strings reagiert werden und eine entsprechende Reaktion seitens des Mikrocontrollers erfolgen.

Listing 4.9: FindStringInArray()

```
1 int    FindStringInArray      (const char* pInput, const char* pOptions[], int cmp_length) {
2     int n = -1;
```

4. Zeitlicher Arbeitsablauf

```
3 while (pOptions[++n]) { //Array durchlaufen bis 0 terminiert
4     //Wenn pInput == pOptions dann gib Array Position zurueck
5     if (!strcmp(pInput, pOptions[n], cmp_length)) return n;
6 }
7 return 99;
8 }
```

Listing 4.10: switchStepper()

```
1
2
3 void switch_Stepper (char * str_rx) { //Auswerte-Funktion fuer die
  Schrittmotorkarte
4     const char* pOptions[] = { // Array mit moeglichen Antworten gegen das geprueft
      wird
5         "#", // 0 - Stepper Karte Befehl erkannt
6         "E", // 1 - Error
7         "!CLS", // 2 - Clear Screen
8         "Test", // 3 - Interner Test zum Debuggen
9         0 };
10    switch (FindStringInArray(str_rx, pOptions, 1)) {
11    case 0: // 0 - Stepper Karte Befehl erkannt
12        lcd_puts("Erfolgreich\n"); // "Erfolgreich" auf dem Display anzeigen
13        //uart_put_string("0\n\r", D_RapidForm);
14        break;
15    case 1: // 1 - Error
16        lcd_puts("Error\n"); // "Error" auf dem Display anzeigen
17        uart_put_string("1\r\n", D_RapidForm); // "1" an RapidForm senden um
          einen Fehler zu melden
18        break;
19    case 2: // 2 - Clear Screen
20        lcd_clrscr(); // Debug: Loescht das Display
21        break;
22    case 3: // 3 - Test
23        lcd_puts("Test bestanden\n"); // Debug: gibt "Test bestanden" auf dem
          Display aus.
24        //uart_put_string("Test bestanden\n\r", D_RapidForm);
25        //uart_put_string("Test bestanden\n\r", D_Stepper);
26        break;
27    default: // Standardmaessig warte kurz.
28        ms_spin(10);
29        //lcd_puts("Stepper: "); // Debugging
30        //lcd_puts(str_rx);
31        //lcd_puts("! \n");
32    }
```

33 }

## 4.4. Verbesserungen an der vorhandenen Hardware

### 4.4.1. Netzteil

Ziel dieses Arbeitsschrittes war es, die festen Lötverbindungen zwischen dem PC-Netzteil und den einzelnen Karteneinschüben im 19"-Rack durch Steckverbindungen zu ersetzen und dadurch leicht erweiterbar zu machen.

Die festen Lötverbindungen am Einschub für die Schrittmotorkarte wurden durch Standard PC-Netzteil Stecker ersetzt. Die Logikbausteine der Schrittmotorkarte wird mit 5V gespeist. Die Schrittmotorkarte wird zusätzlich mit 12V für den Schrittmotor gespeist. Der Stecker lässt sich nun einfach mit einer Buchse des Standard PC-Netzteils verbinden und es ist nicht mehr notwendig zu löten wenn das Netzteil ausgebaut wird. Mittels eines Y-Kabels (siehe Abbildung 4.1) können nun leicht weitere Buchsen hinzugefügt werden.

Dadurch kann das Netzteil nun einfach ein- und ausgebaut werden, bzw. das System leicht um neue Einschubkarten erweitert werden. (TODO: [http://www.kosatec.](http://www.kosatec.de/prod_images/kc/640x480/100539.jpg)

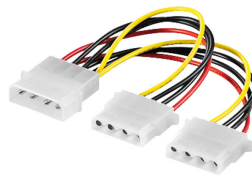


Abbildung 4.1.: Stromverbinder - Y-Kabel?

[de/prod\\_images/kc/640x480/100539.jpg](http://www.kosatec.de/prod_images/kc/640x480/100539.jpg))

### 4.4.2. Zweite Schrittmotorkarte

Zu Anfang war nur eine Schrittmotorkarte für die Rotation des Drehtisches vorbereitet. Mit einem zweiten Schrittmotor konnte der Tisch in der Höhe verstellt werden. Für diesen fehlte jedoch noch eine zweite Schrittmotorkarte. Diese musste noch vorbereitet und mit der ersten verbunden werden.

Dazu wurde, wie in Kapitel 4.4.1 beschrieben, ein weiterer Einschubplatz für die Schrittmotorkarte vorbereitet. Die Karte wurde mit einer Frontblende versehen und auf dieser eine Buchse für die Motorverkabelung und je eine Buchse und einen Stecker

#### 4. Zeitlicher Arbeitsablauf

für die seriellen Schnittstellen verbaut. Diese wurden mit den entsprechenden Anschlüssen auf der Schrittmotorkarte verlötet. Die Karte wird in den Einschubplatz geschoben und mit einem seriellen Kabel als **Daisy-Chain** mit der ersten Schrittmotorkarte verbunden. Dadurch kann die zweite Schrittmotorkarte über die erste angesteuert werden.

Somit steht eine Baugleiche Schrittmotorkarte zur Verfügung. Diese kann nun den Schrittmotor für die Höhenverstellung ansteuern. Befehle an diese Schrittmotorkarte werden an die erste Karte geschickt, jedoch mit dem Prefix 2. Dieser weist die erste Karte an den Befehl an die zweite Karte weiter zu senden. So kann das System um weitere Karten erweitert werden. Für jede weitere Karte muss der Prefix um eins erhöht werden.

##### 4.4.3. Motor- und Endschalterverkabelung

Zwischen der zweiten Schrittmotorkarte und dem zugehörigen Schrittmotor, der für die Höhenverstellung zuständig ist, war noch kein Kabel vorhanden. Dieses musste noch gefertigt und um 3 Leitungen für die Endschalter erweitert werden.

Dafür wurde in der Werkstatt ein 7 adriges Kabel (Abbildung 4.3) besorgt und die passenden Endstecker bestellt. Die Belegung wurde gleich zum Kabel für den ersten Schrittmotor gewählt, jedoch um die 3 Adern für die beiden Endschalter erweitert. Tabelle 4.1 gibt die Belegung des Kabels wieder.

Somit stand ein Kabel zur Verfügung mit dem sowohl der Schrittmotor gesteuert, als auch der Status der Endschalter an die Schrittmotorkarte übermittelt werden konnte. **(TODO: BELEGUNG ÜBERPRÜFEN!)**

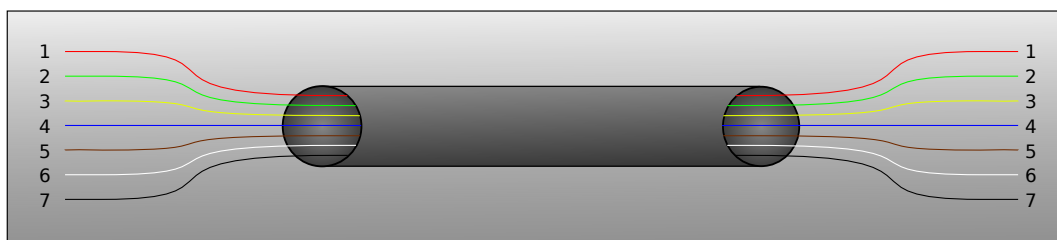


Abbildung 4.2.: Motor- und Endschalterverkabelung

Tabelle 4.1.: Motor- und Endschalterverkabelung

1	Phase A
2	Phase B

#### 4. Zeitlicher Arbeitsablauf

3	Phase C
4	Phase D
5	Endschalter oben
6	Endschalter unten
7	Endschalter Masse

#### 4.4.4. Endschalter

Nun geht es darum, die vorgegeben induktiven Endschalter mit der Schrittmotorkarte und dem Mikrocontroller zu verbinden. Dadurch soll gewährleistet werden, dass der Drehtisch nicht über den Arbeitsbereich hinaus bewegt werden kann. Zusätzlich soll das Erreichen der Endpositionen auf dem LC-Display angezeigt werden.

Da die Schrittmotorkarte nur mechanische Endschalter unterstützt, ließen sich die induktiven Endschalter nicht ohne weiteres nutzen. Um die induktiven Endschalter nutzen zu können, musste die Spannung über einen Spannungsteiler heruntergesetzt werden und die standardmäßigen Eingänge für die mechanischen Endschalter umgangen werden. Die induktiven Endschalter werden direkt an den Optokoppler angeschlossen, welcher für die mechanischen Endschalter zuständig ist. Dadurch wurden die Signale der Endschalter für die Schrittmotorkarte nutzbar.

Ein weiteres Problem bestand darin, dass, wenn der Tisch sich bereits in der End-

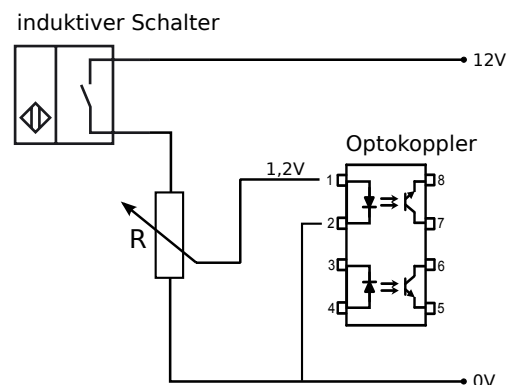


Abbildung 4.3.: Motor- und Endschalterverkabelung

position befand, die Endschalter noch nicht aktiviert wurden. Dies lag daran, dass der Metallstutzen, der die Endschalter auslösen sollte, sich nicht im Schaltbereich der induktiven Schalter befand. Zur Abhilfe wurde ein längerer Metallstutzen von der Werkstatt angefertigt.



#### 4. Zeitlicher Arbeitsablauf

Wenn der Tisch sich in der Endposition befindet, soll dies auch auf dem LC-Display angezeigt werden. Die Signale der Endschalter liegen auf der Rückseite der Schrittmotorkarte am Verbindungsstecker an. Um die Signale dem Mikrocontroller zugänglich zu machen wurde eine Brücke zwischen den Verbindungssteckern der Schrittmotorkarte und der Mikrocontroller-Platine gelötet. Auf der Mikrocontroller-Platine sind diese beiden Pins mit je einem Pin des Mikrocontrollers verbunden. Diese beiden Pins werden im Mikrocontroller als **Interrupts** definiert. Die **Interrupt-Service-Routine** zum Anzeigen der Nachricht auf dem LC-Display wird in Kapitel 5.2.1 beschrieben. Da die Signale der Endschalter nun an der Schrittmotorkarte anliegen, stoppt diese den Motor wenn die Endschalterpositionen erreicht wird. Zusätzlich liegen die Signale am Mikrocontroller an. Dieser gibt dadurch auf dem Display die Meldung *Endschalterposition erreicht!* aus.

#### 4.4.5. Zweite serielle Schnittstelle

Das STK500 bietet nur eine serielle Schnittstelle. Um zusätzlich zur Schrittmotorkarte auch mit RapidForm2004 kommunizieren zu können, wird eine zweite RS-232-Schnittstelle benötigt.

Dafür wurde vorerst auf einem Steckbrett eine zweite serielle Schnittstelle nach dem Schaltplan in Abbildung 4.4 aufgebaut. Später wird diese Schnittstelle direkt auf der Mikrocontroller-Platine vorgesehen. Dadurch ist es möglich mit dem Mikrocontroller über zwei RS-232-Schnittstellen gleichzeitig zu kommunizieren.

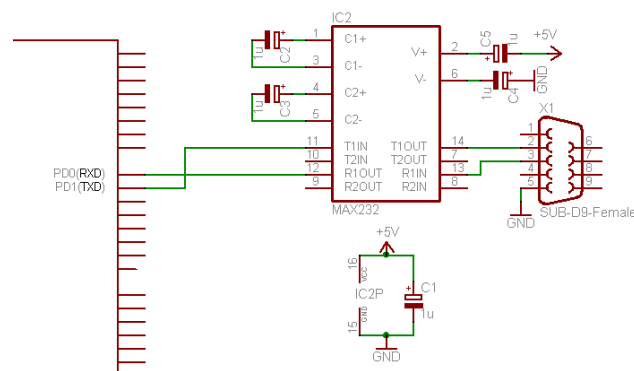


Abbildung 4.4.: Schema: MAX232  
[Mikrocontroller.net 2012]

## 4.5. Kommunikation mit RapidForm2004

RapidForm2004 sendet Befehle die für die Drehtischsteuerung bestimmt sind an den Mikrocontroller. Diese sollen dort empfangen, ausgewertet und in verständlicher Form an die Drehtischsteuerung weiter gegeben werden. RapidForm2004 verwendet dabei verschiedene Befehlssätze für verschiedene Schrittmotorsteuerungen. Für jeden dieser Befehlssätze wird eine eigene Auswerte-Funktion geschrieben. Im folgenden Kapitel wird nun das Empfangen der Befehle beschrieben und eine erste Auswertung, die den empfangenden Befehl dem Befehlssatz einer Schrittmotorsteuerung zuordnet.

Nach dem ein Befehl in der richtigen Auswerte-Funktionen erkannt wurde, soll ein entsprechender Befehl an die Drehtischsteuerung gesendet und die Antwort der Drehtischsteuerung ausgewertet werden. Abschließend soll eine entsprechende Antwort an RapidForm2004 zurück gesendet werden.

Die Kommunikation mit RapidForm2004 ist ähnlich zu der mit der Schrittmotorsteuerung. Diese wurde bereits in Kapitel 4.3 ausführlich beschrieben. Daher wird die Kommunikation hier etwas oberflächlicher beschrieben. **(TODO: TOLLE ZEICHNUNG!)**

### 4.5.1. Befehle empfangen

Zuerst sollen nun die Befehle von RapidForm2004 an den Mikrocontroller, gespeichert werden. Anschließend wird die automatische Auswahl des Befehlssatzes beschrieben. Um anstehende Befehle zu empfangen wird, ähnlich wie in Kapitel 4.3.2, eine Funktion die ständig das Eingangsregister der ersten seriellen Schnittstelle abfragt verwendet (siehe Codelisting 4.11). Auch hier wird die Funktion `uart_rx(dir)` aufgerufen, jedoch mit dem Parameter `D_RapidForm`. Der empfangenen String wird auch hier in die Variable `str_rx` gespeichert. Somit können nun auch Strings von RapidForm2004 empfangen und in der Variablen `str_rx` gespeichert werden.

Listing 4.11: RS-232 Empfang - RapidForm2004

```
1 if ((UCSR0A & (1 << RXC0))){  
2     LED_PORT &= ( 1 << LED2 );  
3     uart_rx(D_RapidForm);  
4 }
```

#### 4.5.1.1. Automatische Auswahl eines Befehlssatzes

Nun geht es darum, dass der Mikrocontroller anhand eines ersten Befehls der empfangen wird, festlegt, mit welchem Befehlssatz fortan kommuniziert werden soll. Die Kennung für den Befehlssatz wird in einer globalen Variable gespeichert und alle weiteren Befehle werden an die entsprechende Auswerte-Funktion für diesen Befehlssatz übergeben.

In der Funktion `uart_rx(dir)` (Codelisting 4.12) wird nun in der ersten *if-Abfrage* entscheiden, von welcher Schnittstelle der empfangene Befehl kam. Diese verzweigt nun, da `D_RapidForm` übergeben wurde, in den *else-Teil*. In diesem wird mit mehreren *if-Abfragen* überprüft, ob bereits der Befehlssatz für einen bestimmten Motor ausgewählt wurde. Ist dies nicht der Fall, wird der empfangende String an die Funktion `switch_Motor(str_rx)` (Codelisting 4.13) übergeben. Diese prüfte mit der aus Kapitel 4.3.3 bekannten Funktion `FindStringInArray(str_rx, pOptions, 3)`, den angekommenen String gegen die Initialisierungsbefehle der einzelnen Befehlssätze. Die Initialisierungsbefehle sind die ersten Befehle die RapidForm2004 an eine Schrittmotorsteuerung sendet um zu prüfen ob diese vorhanden ist. In diesem ersten Schritt wird der String nur zur Identifizierung des von RapidForm2004 verwendeten Befehlssatzes verwendet. Das Antworten auf einen String wird erst in den Nachfolgenden Kapiteln beschrieben. Die Funktion `switch_Motor(str_rx)` gibt einen numerischen Wert zurück. Jede Zahl entspricht dabei dem Befehlssatz für eine Schrittmotorsteuerung. Die Zahlenwerte werden dabei mittels Makro-Definitionen(Codelisting 4.13 Zeile 1-6) durch lesbare Text-Variablen ersetzt. Dies erhöhte die Lesbarkeit und das Verständnis ungemein. War dieser Schritt erfolgreich, wird in den folgenden *if-Abfragen* die richtige Auswerte-Funktion aufgerufen. Konnte die Funktion `switch_Motor(str_rx)` den empfangen Befehl nicht zuordnen, gibt sie `M_UNK` zurück und es wird auf dem Display *Unbekannter Motor!* ausgegeben.

Somit ist es möglich Befehle von RapidForm2004 zu empfangen und an die richtige Auswerte-Funktionen zu übergeben. Zusätzlich wird die Programmierung dadurch wesentlich robuster, da unbekannte Befehle ignoriert werden.

Der Nachteil dieses Vorgehens besteht darin, dass für eine wechseln des Befehlssatzes der Mikrocontroller neu gestartet werden muss. Ein Beheben dieses Nachteils wäre nicht ohne weiteres möglich gewesen.

Listing 4.12: Funktion: `uartrx()`

```
1 void    uart_rx                (int dir) {  
2     uart_get_string(str_rx, dir);  
3     if (dir == D_Stepper)
```

4. Zeitlicher Arbeitsablauf

```
4         switch_Stepper(str_rx);
5     else {
6         if ( Initialized == M_UNK){
7             lcd_puts("Unbekannter Motor!\n");
8             //lcd_puts(str_rx);
9             Initialized = M_NOTI;
10        }
11        if ( Initialized == M_NOTI){
12            Initialized = switch_Motor(str_rx);
13        }
14        if ( Initialized == M_ISEL)
15            switch_Isel(str_rx);
16        if ( Initialized == M_CSG)
17            switch_csg(str_rx);
18        if ( Initialized == M_ZETA)
19            switch_Zeta(str_rx);
20        if ( Initialized == M_TERMINAL)
21            switch_Terminal(str_rx);
22    }
23 }
```

Listing 4.13: Funktion: switchMotor

```
1 #define M_UNK      -2
2 #define M_NOTI     -1
3 #define M_ISEL      0
4 #define M_CSG       1
5 #define M_ZETA      2
6 #define M_TERMINAL 3
7
8 int      Initialized = M_NOTI;
9
10 int      switch_Motor      (char * str_rx) {
11     const char* pOptions[] = {
12         "@01",             // 0 - Isel
13         "Q:",               // 1 - CSG
14         "ECHO0",           // 2 - Zeta
15         "!Terminal",       // 3 - Terminal ansteuerung!
16         0 };
17     switch (FindStringInArray(str_rx, pOptions, 3)) {
18     case 0:                 // 0 - ISEL
19         return M_ISEL;
20         break;
21     case 1:                 // 1 - CSG
22         return M_CSG;
```

#### 4. Zeitlicher Arbeitsablauf

```
23         break;
24     case 2:                // 2 – Zeta
25         return M_ZETA;
26         break;
27     case 3:                // 3 – Terminal ansteuerung
28         return M_TERMINAL;
29         break;
30     default:
31         return M_UNK;
32     }
33 }
```

## 4.6. Auswerte-Funktionen

Die Auswerte-Funktionen sind das Herzstück des Programms. Es geht darum für jedes Protokoll eine eigene Auswerte-Funktion zu schreiben. Diese sollen die von RapidForm2004 kommenden Strings verstehen können und in einen, für die vorhandene Schrittmotorkarte, verständlichen Befehl übersetzen können. Die Funktionen sollen weiterhin prüfen, ob der Befehl von der Schrittmotorkarte erkannt wurde und den Status der Schrittmotorkarte zurück an RapidForm2004 melden.

Alle bisherigen Arbeitsschritte hatten zum Ziel, die Kommunikation zwischen RapidForm2004 und der ersten Schrittmotorkarte zu ermöglichen. Nun fehlte nur noch der Teil des Programms der die ankommenden Befehle auswertet und in verständlicher Form an die Schrittmotorkarte weitergibt.

Im folgenden Kapitel wird dieser Ablauf nun exemplarisch für den Befehlssatz eines Isel-Motors erklärt.

### 4.6.1. Auswerte-Funktion für Isel Motoren

Nun soll die Kommunikation zwischen RapidForm2004 und der Schrittmotorkarte beschrieben werden.

Wurde wie in Kapitel ?? beschrieben, der Befehlssatz für einen Isel Motor erkannt, wird der empfangene String, an die Auswerte-Funktion `switch_Isel(str_rx)` übergeben. Der Ablauf dieser Funktion ist ähnlich aufgebaut wie bei der Kommunikation mit der Schrittmotorkarte(Kapitel 4.3) und bei der Automatischen Auswahl des Befehlssatzes(Kapitel ??). In der Funktion `switch_Isel(str_rx)` sind in dem Array `pOptions` alle benötigten Befehle des Isel-Befehlssatzes hinterlegt. Mit der aus Kapitel ?? bekannten Funktion `FindStringInArray(str_rx, pOptions)` wird `str_rx` gegen diese Befehle geprüft. Wird der Befehl im Array gefunden gibt die

#### 4. Zeitlicher Arbeitsablauf

Funktion `FindStringInArray()` die Position des Strings im Array zurück. Mittels der *switch-case-Struktur* lässt sich nun so für jeden Befehl eine entsprechende Reaktion ausführen.

##### 4.6.1.1. Initialisierung

##### 4.6.1.2. Statusabfrage

Kommt z.B. der String `@0R` an, wird der Codeblock von *case 4* ausgeführt. Dies ist der Codeblock für eine Statusabfrage. Auf dem LC-Display wird *Statusabfrage*: ausgegeben. Danach wird der entsprechende Befehl für eine Statusabfrage an die Schrittmotorkarte gesendet. Nach einer kurzen Pause von 50ms, um die Verarbeitung auf der Schrittmotorkarte zu gewährleisten, wird mit einer if-Anweisung geprüft ob sich Daten im Schrittmotorkarten seitigen Empfangsregister befinden. Sprich, die Schrittmotorkarte reagiert hat. Ist dies der Fall, wird der Ablauf, bekannt aus Kapitel 4.3, durchlaufen. Während diesem Ablauf wird die entsprechende Antwort der Schrittmotorkarte auf dem LC-Display ausgegeben. In einer weiteren if-Anweisung wird überprüft ob der angekommene String erfolgreich war. Wenn ja, wird dies an RapidForm2004 gemeldet. Andernfalls zeigt das Display *Fehlgeschlagen* an und sendet eine 1 an RapidForm2004.

##### 4.6.1.3. Bewegung

Listing 4.14: Übersetzungs Logik: Isel

```
1 void switch_Isel(char * str_rx) {
2     const char* pOptions[] = {
3         "XXXXXXX", // 0 - Reserve
4         "!CLS",    // 1 - LC-Display loeschen
5         "Test",    // 2 - Test
6         "@01",     // 3 - Achse auswaehlen
7         "@0R",     // 4 - Status abfrage
8         "@0M",     // 5 - Gehe zu Position MX , +600
9         0 };
10
11     int Ret_Val = FindStringInArray(str_rx, pOptions, 3);
12     switch (Ret_Val) {
13     case 0:        // 0 - Reserve
14     case 1:        // 1 - LC-Display loeschen
15     case 2:        // 2 - Test
16     case 3:        // 3 - Achse auswaehlen
```

4. Zeitlicher Arbeitsablauf

```
17         ms_spin(10);
18         lcd_puts("Init");
19         uart_put_string("0\r\n", D_RapidForm);
20         break;
21     case 4:                // 4 – Status abfrage
22         lcd_puts("Statusabfrage:  \n");
23         uart_put_string("A\n", D_Stepper);
24         ms_spin(50);
25         if ((UCSR1A & (1 << RXC1)))
26             uart_rx(D_Stepper);
27         if (!strcmp(str_rx, "0#"))
28             uart_put_string("0\r\n", D_RapidForm);
29         else {
30             lcd_puts("Fehlgeschlagen  \n");
31             uart_put_string("1\r\n", D_RapidForm);
32         }
33         break;
34     case 5:                // 5 – Gehe zu Position MX , +600
35         ms_spin(10);
36         char Position [33], Winkel[6];
37         memset(Position, '\0', 33);
38         memset(Winkel, '\0', 6);
39         String_zerlegen_Isel(str_rx, Position, Winkel);
40         char Move_To[40];
41         memset(Move_To, '\0', 40);
42         Move_To[0] = 'M';
43         Move_To[1] = 'A';
44         Move_To[2] = ' ';
45         Move_To[3] = '\0';
46         strcat(Move_To, Position);
47         strcat(Move_To, "\n");
48         lcd_puts("Pos:");
49         lcd_puts(Move_To);
50
51         uart_put_string(Move_To, D_Stepper);
52         ms_spin(50);
53         if ((UCSR1A & (1 << RXC1)))
54             uart_rx(D_Stepper);
55         else {
56             break;
57         }
58
59         uart_put_string("A\n", D_Stepper);
60         ms_spin(50);
61         if ((UCSR1A & (1 << RXC1)))
```

4. Zeitlicher Arbeitsablauf

```
62         uart_rx(D_Stepper);
63     else {
64         lcd_puts("Keine Bewegung!\n");
65     }
66
67     while (!strcmp(str_rx,"1#")){
68         uart_put_string("A\n", D_Stepper);
69         ms_spin(50);
70         if ((UCSR1A & (1 << RXC1))){
71             uart_rx(D_Stepper);
72             lcd_clrscr();
73             lcd_puts("Gehe zu Winkel: ");
74             lcd_puts(Winkel);
75             lcd_puts("\n");
76         }
77         else {
78             lcd_puts("Keine Antwort\n");
79         }
80         wdt_reset();
81     }
82     lcd_puts("Winkel: ");
83     lcd_puts(Winkel);
84     lcd_puts(" Erreicht\n");
85     uart_put_string("0\r\n", D_RapidForm);
86     break;
87 default:
88     lcd_puts(str_rx);
89 }
90 }
```



## 5. Probleme und Lösungen

### 5.1. Entwicklungsumgebungen

#### 5.1.1. AVR Studio 5

Die von Atmel AVR Studio 5 ist eine von Atmel bereitgestellte Entwicklungsumgebung. Diese scheint jedoch eine fehlerhafte Bibliothek zu enthalten. Die Kombination aus Mikrocontroller ATmega324A und AVR Studio 5 erzeugte nicht nachvollziehbare Probleme. Bei dem selbem Programm und einem anderem Mikrocontroller oder einer anderen Entwicklungsumgebung tauchten keine Fehler auf. In der Entwicklungsumgebung Eclipse lies sich der Fehler reproduzieren wenn der Pfad der Atmel Bibliotheken eingestellt wurde. Die WinAVR Bibliotheken und eine selbst kompilierte **Toolchain** unter Linux zeigten keine Probleme.

Daher wechselte ich zur **Open Source** Entwicklungsumgebung Eclipse. Erst dadurch wurde es möglich erfolgreich zu arbeiten. Außerdem wurde das Projekt dadurch plattformunabhängig und ich nutzte bis auf RapidForm2004 nur noch freie Open Source Software.

#### 5.1.2. Eclipse

Eclipse ist eine in Java programmierte freie Open Source Entwicklungsumgebung für Java. Sie lässt sich durch **Plugins** leicht für viele Sprachen erweitern.

Mit dem CDT-Plugin, dem AVR-Plugin und einer Bibliothek wie z.B. WinAVR für Windows ist Eclipse eine vollwertige Entwicklungsumgebung für Atmel Mikrocontroller. Ergänzt wird diese durch die Programmiersoftware AVR-Dude.

### 5.2. Interrupts

Viele Mikrocontroller bieten die Möglichkeit Eventbasierte Subroutinen auszuführen. Wenn einer der Interrupts ausgelöst wird, wird das Hauptprogramm unterbrochen

und die Entsprechende Interrupt-Service-Routine, kurz ISR, ausgeführt. Nach Beendigung der ISR wird das Hauptprogramm an der vorherigen Stelle wieder aufgenommen.

ISR dürfen nur sehr wenige Befehle enthalten und müssen innerhalb weniger Clock-Cycles abgeschlossen sein.

Interrupts können z.B. der Überlauf eines internen Timer sein, oder ein externens Signal an einem Pin.

Im Projekt werden externe Interrupts, Timer-Überlauf Interrupts und der Watchdog Interrupt genutzt.

### 5.2.1. Endschalter

Die Endschalter sind über die Schrittmotorkarten und eine Brücke in der Steuerung mit der Mikrocontroller Platine Verbunden. Dort sind sie an 2 Interrupt Pins angeschlossen. **(TODO: PINS RAUS SUCHEN!)** Bei einem Flanken Wechsel an den Pins wird ein Interrupt ausgelöst.

Das Code-Listing 5.1 zeigt die ISR für die Endschalter.

Listing 5.1: ISR: Endschalter

```
1 PCMSK3 |= ( 1 << PCINT28 ); // Interrupts definieren PD4 als Interrupt zulassen
2 PCICR |= ( 1 << PCIE3 ); // Pin Change Interrupt Control Register – PCIE3 setzen fuer
   PCINT30
3 ISR(PCINT3_vect){ // Endschalter Position erreicht
4   lcd_puts("Positive Endschalter Position Erreicht!");
5   LED_PORT ^= (1 << LED3);
6 }
7 ISR(PCINT2_vect){ // Endschalter Position erreicht
8   lcd_puts("Negative Endschalter Position Erreicht!");
9   LED_PORT ^= (1 << LED3);
10 }
```

### 5.2.2. Watchdog

Der Watchdog ist eine Sicherungseinrichtung des Mikrocontroller. In regelmäßigen Abständen wird überprüft ob das Watchdog Bit gesetzt ist und anschließend zurück gesetzt. Das Bit muss innerhalb der voreingestellten Zeit immer wieder neu gesetzt werden. Dies kann mit der wdt\_reset() Funktion realisiert werden. Ist das Bit nicht gesetzt, wird der Mikrocontroller zurückgesetzt. **(TODO: INVERSE LOGIK?)** Dies geschieht z.B. bei nicht geplanten Endlosschleifen.

Wahlweise kann kurz vor dem Reset noch die Watchdog-ISR durchlaufen werden.

Im Projekt wird in der ISR die Fehler LED eingeschaltet und eine Meldung auf dem LC-Display ausgegeben. Siehe hierzu auch Listing 5.2 Zeilen 12-15.

Listing 5.2: Watchdog

```
1 #include <avr/wdt.h>
2
3 void init_WDT(void) {
4     cli();
5     wdt_reset();
6     WDTCSR |= (1 << WDCE) | (1 << WDE);
7     WDTCSR = (1 << WDE) | (1 << WDIE) | (1 << WDP3) | (1 << WDP0); //Watchdog 8s
8     //WDTCSR = 0x0F; //Watchdog Off
9     sei();
10 }
11
12 ISR(WDT_vect){ // Watchdog ISR
13     LED_PORT &= ~(1 << LED4); // LED5 einschalten
14     lcd_puts("Something went \nterribly wrong!\nRebooting!");
15 }
```

### 5.3. Fuses

Als Fuses werden Register bezeichnet mit denen sich, auf Hardwareebene, das Verhalten des Mikrocontrollers verändern lässt.

Im Projekt wurden folgende Fuses problematisch.

- **JTAGEN** - Ist dieses Fusebit gesetzt, werden 4 Pins des PortB genutzt um den Mikrocontroller zu debuggen und können nicht anders genutzt werden. Hardware Debugging bietet viele Vorteile. Diese wurden im Projekt jedoch nicht genutzt da PortB für die LEDs genutzt wurde.
- **WDTON** - Ist dieses Fusebit gesetzt läuft der Watchdog Timer immer mit. Wird der Watchdog dann nicht regelmäßig zurückgesetzt startet der Mikrocontroller ständig neu.
- **CKDIV8** - Teilt den Systemtakt des Mikrocontroller durch 8. Dies ist Energiesparender. Der geringere Takt muss in F\_CPU angepasst werden da sonst zeitkritische Prozesse mit der falschen Geschwindigkeit ablaufen.
- **CKOUT** - An PortB wird an einem Pin der Systemtakt ausgegeben. Dieser kann dann leicht mit einem Frequenz-Messgerät überprüft werden. Der Pin kann dann jedoch nicht anderweitig genutzt werden.

- **CKSELX** - Über diese 4 Bits kann der Systemtakt eingestellt werden.

Tabelle 5.1.: Fuses

OCDEN	On Chip Debugging
JTAGEN	Hardware Debugging
SPIEN	Serial Program and Data Downloading
WDTON	Watchdog Timer always on
EESAVE	EEPROM memory is preserved through the Chip Erase
BOOTSZ1	Select Boot Size
BOOTSZ0	Select Boot Size
BOOTRST	Select Reset Vector
CKDIV8	Divide clock by 8
CKOUT	Clock output
SUT1	Select start-up time
SUT0	Select start-up time
CKSEL3	Select Clock source
CKSEL2	Select Clock source
CKSEL1	Select Clock source
CKSEL0	Select Clock source

## 5.4. Platinenlayout

Für den Mikrocontroller und seine Peripherie entwickelte ich ein Platinenlayout in der Open Source Software KiCad.

Dazu wurden die Schaltungen wie auf dem STK500 in den Schaltplan übernommen und dort das Layout entwickelt. **(TODO: SCHALTPLAN UND EINBINDEN.)**

## 5.5. 19"-Einschub

Die Platine für den Mikrocontroller konstruierte ich als 19"-Einschub. Über den rückwärtigen Steckverbinder verband ich die Platine mit der Spannungsversorgung. Zusätzlich kommen hier auch die Signale der Endschalter an. An der Vorderseite befestigte ich eine Blende. Auf der Blende befinden sich das LC-Display, fünf Taster, 5 LEDs und 2 serielle Schnittstellen. Alle Bauteile sind steckbar mit der Platine verbunden. **(TODO: BILD DES EINSCHUB)**

### 5. Probleme und Lösungen

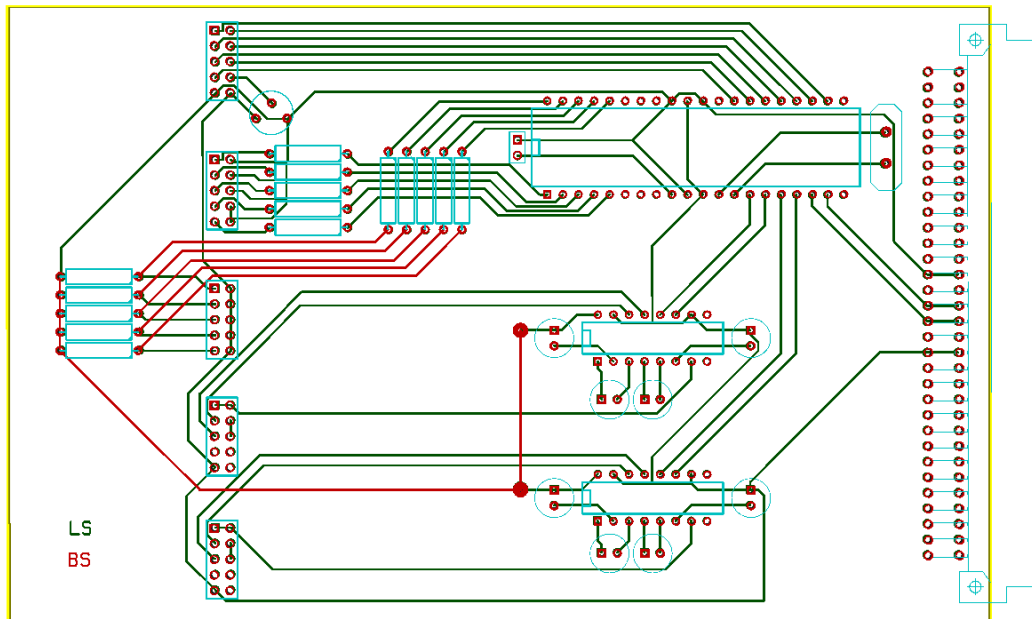


Abbildung 5.1.: Platinenlayout

(TODO: UMSTELLUNGSPROBLEME)

## 6. Fazit und Zukunft

### 6.1. Fazit

**(TODO: FAZIT SCHREIBEN!)**

## Literaturverzeichnis

### **Atm 2003**

ATMEL CORPORATION (Hrsg.): *AVR STK500 User Guide*. San Jose, CA 95131, USA: Atmel Corporation, 06 2003 [2.7](#)

### **Atm 2011**

ATMEL CORPORATION (Hrsg.): *ATmega164A/PA/324A/PA/644A/PA/1284/P Complete*. San Jose, CA 95131, USA: Atmel Corporation, 06 2011 [2.6](#)

### **Corporation 2012a**

CORPORATION, Atmel: *ATmega324A- Atmel Corporation*. <http://www.atmel.com/devices/ATMEGA324A.aspx>. Version: 2012. – [Online; Stand 11. Februar 2012]

### **Corporation 2012b**

CORPORATION, Atmel: *ATmega8515- Atmel Corporation*. <http://www.atmel.com/devices/ATMEGA8515.aspx>. Version: 2012. – [Online; Stand 11. Februar 2012] [2.4](#)

### **Dannegger 2012**

DANNEGGER, Peter: *Entprellung - Mikrocontroller.net*. <http://www.mikrocontroller.net/articles/Entprellung>. Version: 2012. – [Online; Stand 11. Februar 2012] [4.1.1](#)

### **Fleury 2012**

FLEURY, Peter: *Peter Fleury's Home Page*. <http://jump.to/fleury>. Version: 2012. – [Online; Stand 11. Februar 2012] [4.1.3](#)

### **Mikrocontroller.net 2012**

MIKROCONTROLLER.NET: *RS-232 - Mikrocontroller.net*. <http://www.mikrocontroller.net/articles/RS-232>. Version: 2012. – [Online; Stand 11. Februar 2012] [4.1.4](#), [4.4](#)

### **Minolta 2012**

MINOLTA: *Funktionen - VI-910 / KONICA MINOLTA*. <http://>

[//www.konicaminolta.eu/de/messinstrumente/produkte/3d-messtechnik/beruehrungsloser-3d-scanner/vi-910/funktionen.html](http://www.konicaminolta.eu/de/messinstrumente/produkte/3d-messtechnik/beruehrungsloser-3d-scanner/vi-910/funktionen.html). Version: 2012. – [Online; Stand 11. Februar 2012] 2.2

**V9141 2001**

RS (Hrsg.): *Schrittmotor-Platine mit integriertem Treiber*. Mörfelden-Walldorf: RS, 03 2001 A.2

**Twillman 2011**

TWILLMAN, Tymm: *AVR Freaks*. <http://www.avrfreaks.net/>. Version: 2011. – [Online; Stand 01. November 2011]

**Wikipedia 2012a**

WIKIPEDIA: *American Standard Code for Information Interchange* — *Wikipedia, Die freie Enzyklopädie*. [http://de.wikipedia.org/w/index.php?title=American\\_Standard\\_Code\\_for\\_Information\\_Interchange&oldid=99892678](http://de.wikipedia.org/w/index.php?title=American_Standard_Code_for_Information_Interchange&oldid=99892678). Version: 2012. – [Online; Stand 23. Februar 2012] 1

**Wikipedia 2012b**

WIKIPEDIA: *Daisy chain* — *Wikipedia, Die freie Enzyklopädie*. [http://de.wikipedia.org/w/index.php?title=Daisy\\_chain&oldid=98475104](http://de.wikipedia.org/w/index.php?title=Daisy_chain&oldid=98475104). Version: 2012. – [Online; Stand 11. Februar 2012] 2

**Wikipedia 2012c**

WIKIPEDIA: *Kommunikationsprotokoll* — *Wikipedia, Die freie Enzyklopädie*. <http://de.wikipedia.org/w/index.php?title=Kommunikationsprotokoll&oldid=99325271>. Version: 2012. – [Online; Stand 25. Februar 2012]



## Eidesstattliche Erklärung

Hiermit versichere ich, dass ich den vorliegenden Bericht:

*Übersetzen von Schrittmotorprotokollen*  
*Entwurf eines Hardwareübersetzers*

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

Remagen, den 7. März 2012



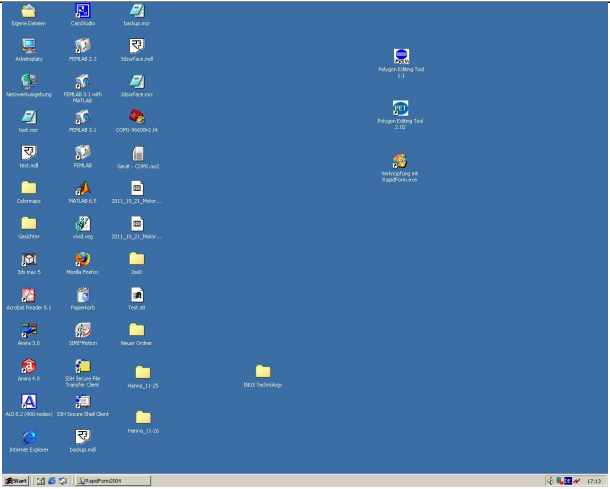
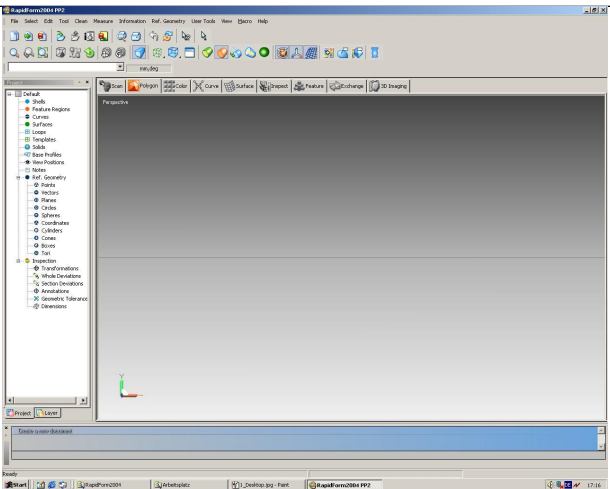
JOHANNES DIELMANN

## A. Anhang

## A.1. Schritt für Schritt Anleitung

Eine Schritt für Schritt Anleitung zum vollständigen Scannen und exportieren eines 3D-Objektes.

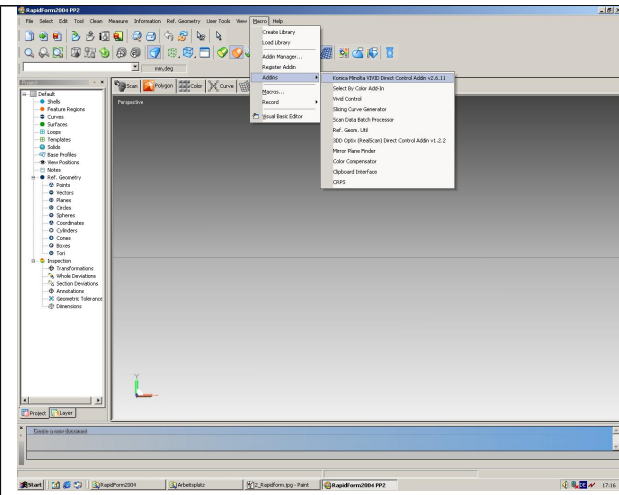
Tabelle A.1.: Schritt für Schritt Anleitung

Schritt für Schritt Anleitung	
Schritt 1 - Starten von RapidForm2004	
Auf dem Desktop doppelt auf das RapidForm Icon klicken.	
Schritt 2 - Oberfläche von RapidForm2004	
Die Oberfläche unterteilt sich in Menü, Werkzeugleisten, Projektbaum und ???. Je nach dem welche Ansicht in ??? gewählt ist, verändern sich auch das Menü und die Werkzeugleisten.	

## Fortsetzung

### Schritt 3 - Starten des "ADD-IN"

In der Menüzeile auf **Macro**  
-> **Addins** -> **Konica Mi-**  
**nolta VIVID Direct Con-**  
**trol Addin v2.6.11** klicken.  
**(TODO: SCHRITTMOTOR**  
**VERBINDEN!)**



### Schritt 4 - Kalibrieren vorbereiten

#### Hinweis

Für ein erfolgreiches Zu-  
sammenführen der ein-  
zelnen Aufnahmen ist die  
Kalibrierung unerlässlich!

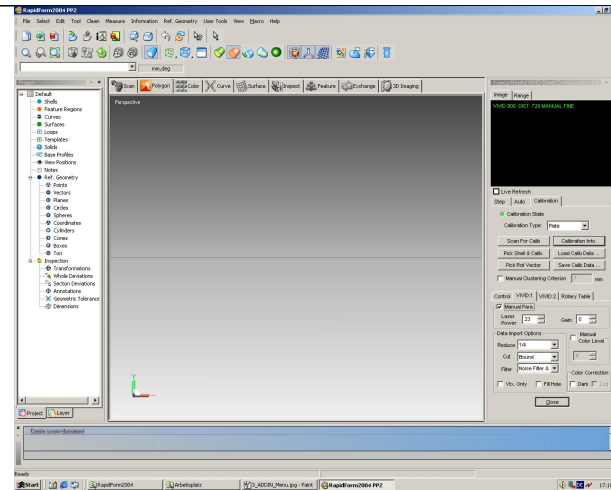
Auf dem Add-In Panel, un-  
ter dem Vorschau Fenster,  
auf **Live-Preview** klicken.  
Das Kalibrierungsblech auf  
dem Drehtisch positionieren.  
Dabei muss der Noppen an  
der Unterseite des Kalibrie-  
rungsblechs in das mittlere  
Loch des Drehtisches gesteckt  
werden. Die abgeklebte Seite  
muss zum VI-900 zeigen.

Bild?

## Fortsetzung

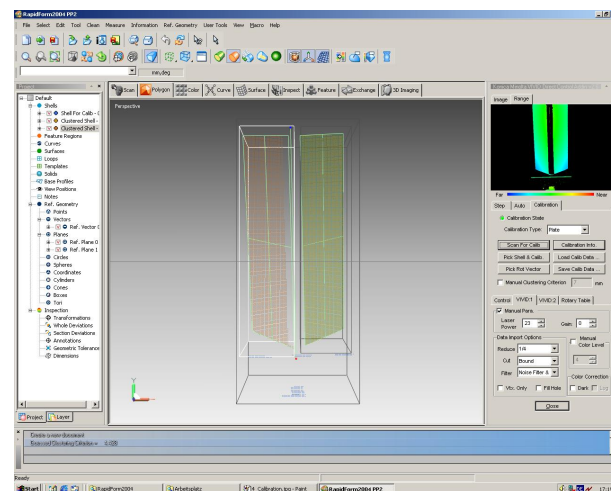
### Schritt 5 - Kalibrieren

Den Reiter **VI-VID: 1** auswählen.  
Bei **Manual Para.** ein Häkchen setzen.  
Im Feld **Laser Power** "23" eintragen.  
Auf **Scan for Calib** klicken.



### Schritt 6 - Kalibrierungsergebnis

Das Ergebnis sollte ähnlich zu dem in der rechten Abbildung sein.  
Falls das Add-In einen Fehler ausgibt, muss das Kalibrierungsblech eventuell anders positioniert werden, der Wert im Feld **Laser Power** verändert werden oder der Fokus manuell eingestellt werden.  
War die Kalibrierung erfolgreich können die *Kalibrationsebenen* im *Projektbaum* ausgeblendet werden.



Fortsetzung

**Schritt 7 - StepScan**

Bei **Manual Para.**

das Häkchen entfernen.

Zum Reiter **Step** wechseln.

Bei **Angle Tag** und **Rotate Table to next Scan position** Häkchen setzen.

Bei **Init. Align in RF**

**Using Rotary In-**  
**fo.** und **Auto Accept**

die Häkchen entfernen.

Bei **Rotation Step**

die gewünschte Dre-

hung in Grad eingeben.  
"45", "60" und "90" sind gu-

**Schritt 8 - AutoFocus**

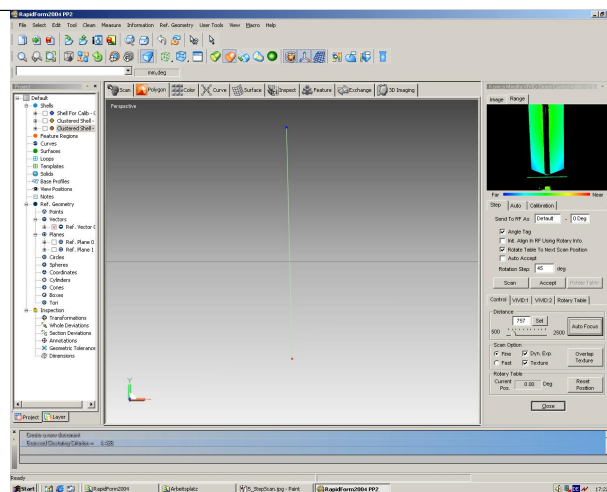
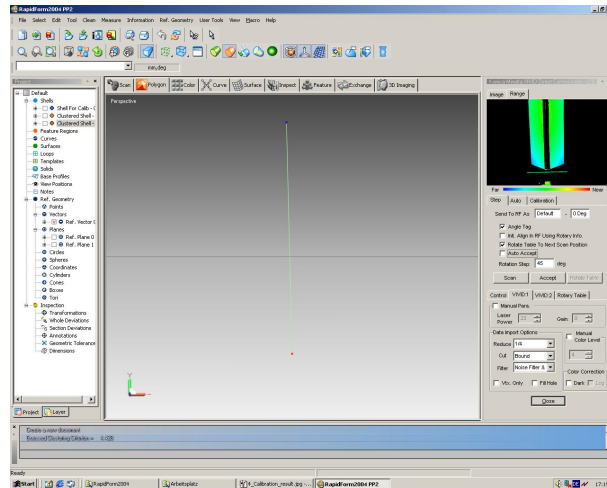
Das *Kalibrationsblech* ent-

fernen und durch das zu

scannende Objekt ersetzen.

Zum Reiter **Con-**  
**trol** wechseln.

Auf **Autofocus** klicken.



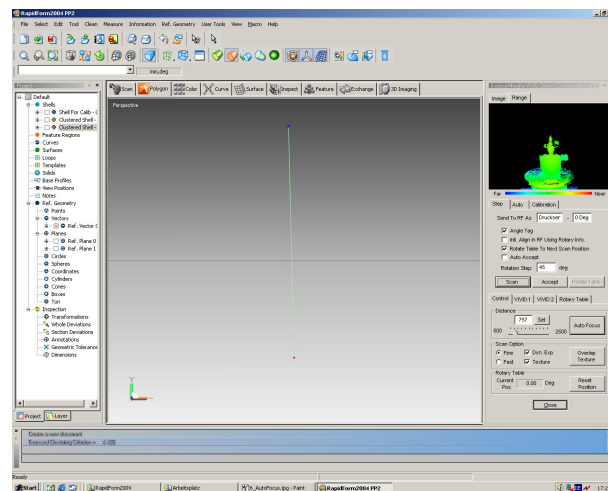
## Fortsetzung

### Schritt 9 - Scan

Auf **Scan** klicken.  
Das Objekt sollte möglichst schon zu erkennen sein und die Farben sich im Mittleren Bereich bewegen.  
Ansonsten muss mit den Parametern **Focus** und **Laser-Power** gespielt werden.

#### Hinweis

Die Position des VI-900 darf nach der Kalibrierung nicht mehr verändert werden!

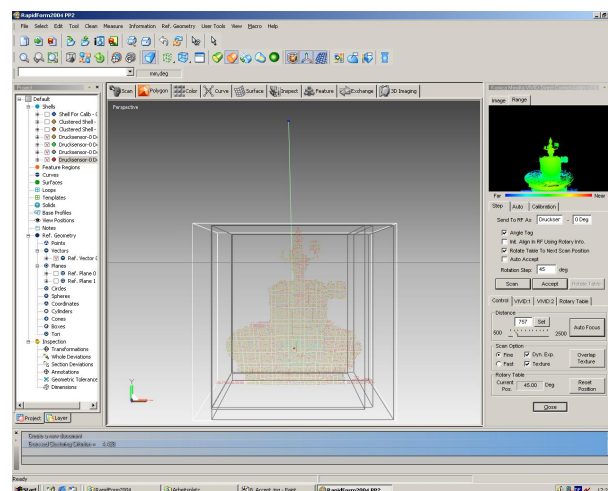


### Schritt 10 - Akzeptieren

Wenn das Objekt gut zu erkennen ist, werden mit **Accept** die Daten an RapidForm2004 gesendet. Der Drehtisch sollte sich nun automatisch um den eingestellten Winkel drehen.

#### Hinweis

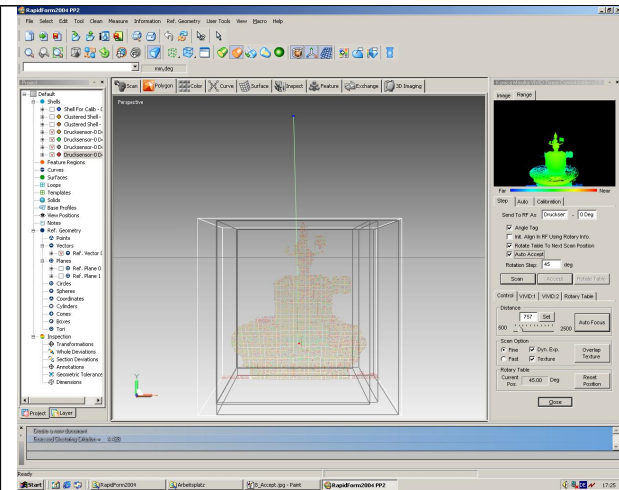
Bei **AutoAccept** kann nun ein Häkchen gesetzt werden.



## Fortsetzung

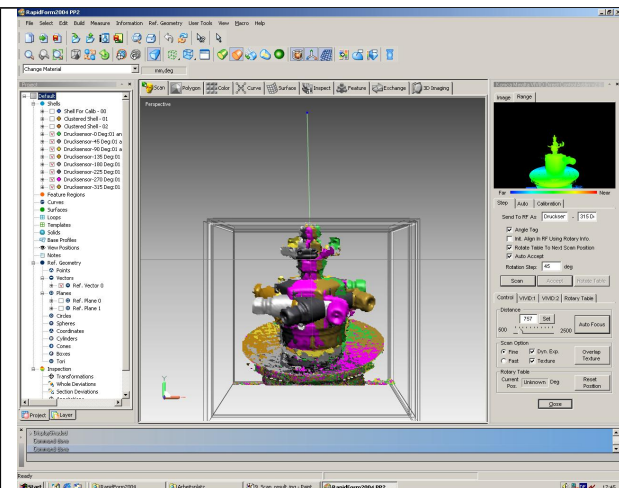
### Schritt 11 - Scannen

Auf **Scan** klicken.  
Diesen Schritt wieder-  
holen bis alle Aufnah-  
men abgeschlossen sind.



### Schritt 12 - Ergebnis der Scans

Nach Abschluss aller  
Scans dreht der Tisch sich  
automatisch in die Ur-  
sprungsposition zurück.  
Im Arbeitsbereich sollten sich  
nun alle Scans befinden.

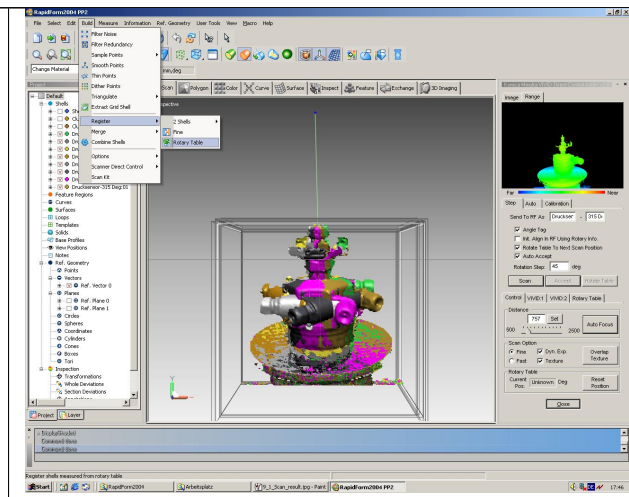




### Fortsetzung

#### Schritt 13 - Drehen und Zusammenführen der Scans

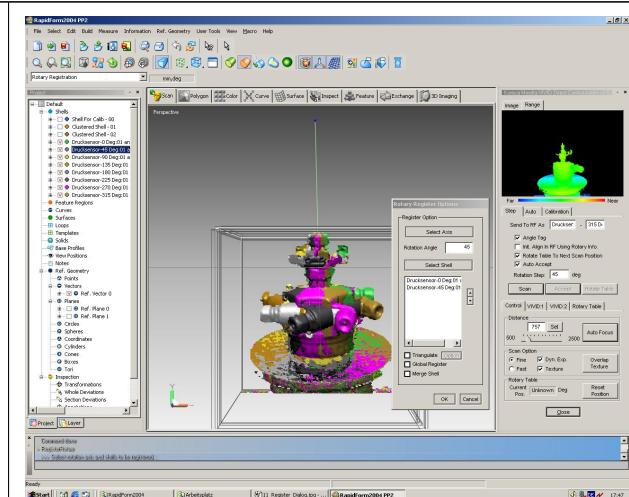
In der Menüzeile auf **Build -> Register -> Rotary Table** klicken.



#### Schritt 14 - Registrieren

Im Dialog auf **Select Axis** klicken.

Im Darstellungsbereich auf die Achse aus dem Kalibrierungsscan klicken. Bei **Rotation-Angle** den Winkel eines Scans eintragen. Im **Projektbaum** den Entsprechenden Scan auswählen. Die letzten beiden Schritte mit allen Scans wiederholen. Den Dialog mit **Ok** verlassen.



[illegible]

## A.2. Protokoll der Schrittmotorkarte

Tabelle A.2 zeigt den ASCII Befehlssatz der Schrittmotorkarte.

_A	Motorstatus liefern
_C n	konstante Geschwindigkeit einstellen
_D n	Bezugswert definieren
_E n	Motorstrom einstellen
_F	Standardeinstellungen aktivieren
_H	Sanfter stop
_I	4-Bit-Eingang lesen
_J jdss	Joystickparameter einstellen
_L n	lokalen Modus aktivieren/beenden
_M n	n Schritte ausführen
_MA n	zu n bewegen
_MC n	mit konstanter Geschwindigkeit bewegen
_MCA n	MA mit konstanter Geschwindigkeit
_MCL n	MC zu Endschalterposition
_ML n	zur Endschalterposition bewegen
_N n	Zeilenvorschub (LF, hex. 0A) einfügen/löschen
_O n	n an 4-Bit-Ausgang senden
_P nnnn	Motorparameter einstellen
_Q	Parameter in EEROM speichern
_R n	Mikroschritteilung einstellen
_RL	Endschalterwerte lesen
_RS	verbleibende Schritte lesen
_S	Nothalt
_T n	Eingang n auslösen
_W	Position anfordern

Tabelle A.2.: ASCII Befehlssatz R+S Schrittmotorsteuerung

V9141 [2001] Der ”\_” wird mit der anzusteuernenden Kartennummer ersetzt. Dabei wird von 1 aufwärts gezählt. Bei der ersten Karte kann die Nummer weggelassen werden.

### A.3. Protokolle aus RapidForm2004

Listing A.1: RapidForm2004 Protokolle Empfang

```
1 model "CSG-602R(Ver.2.0)"
2 port "9600" "n81h"
3 init "D:2S500F5000R200S500F5000R200\r\nH:1-\r\n" ""
4 finish "L:1\r\nH:1-\r\n" ""
5 arrot "M:1+P%d\r\nG:\r\n" ""
6 stop "L:E\r\n" ""
7 home "L:1\r\nH:1-\r\n" ""
8 step "-0.0025" "-99999999" "99999999"
9 timeout "60"
10 firsttimeout "10"
11
12 model "CSG-602R(Ver.1.0)"
13 port "9600" "n81h"
14 init "D:2S500F5000R200S500F5000R200\r\nH:1-\r\n" ""
15 finish "L:1\r\nH:1-\r\n" ""
16 arrot "M:1+P%d\r\nG:\r\n" ""
17 stop "L:E\r\n" "" home "L:1\r\nH:1-\r\n" ""
18 step "-0.005" "-99999999" "99999999"
19 timeout "60"
20 firsttimeout "10"
21
22 model "Mark-202"
23 port "9600" "n81h"
24 init "S:180\r\nD:1S20000F200000R200\r\n" "OK\r\nOK\r\n"
25 finish "L:1\r\nH:1\r\n" "OK\r\nOK\r\n"
26 arrot "M:1+P%d\r\nG:\r\n" "OK\r\nOK\r\n"
27 stop "L:E\r\n" "OK\r\n"
28 home "L:1\r\nH:1\r\n" "OK\r\nOK\r\n"
29 step "-0.0000625" "-99999999" "99999999"
30 timeout "60" firsttimeout "10"
31
32 model "Mark-102" port "9600" "n81h"
33 init "D:WS500F5000R200S500F5000R200\r\n" "OK\r\nOK\r\n"
34 finish "H:1-\r\n" "OK\r\n"
35 arrot "M:1+P%d\r\nG:\r\n" "OK\r\nOK\r\n"
36 stop "L:E\r\n" "OK\r\n"
37 home "H:1-\r\n" "OK\r\n"
38 step "-0.0025" "-99999999" "99999999"
39 timeout "60"
40 firsttimeout "10"
41
```



*A. Anhang*

---

```
84 firsttimeout "10"
85
86 model "MMC-2"
87 port "9600" "n81h"
88 init "F:XP3\r\nS:X1\r\n" "\r\n\r\n"
89 finish "L:X\r\nA:XP0\r\nW:\r\n" "\r\n\r\n"
90 arot "A:XP%d\r\nW:\r\n" "\r\n\r\n"
91 stop "E:\r\n" "\r\n"
92 home "A:XP0\r\nW:\r\n" "\r\n"
93 step "0.005" "-99999999" "99999999"
94 timeout "60"
95 firsttimeout "10"
```

## A.4. Technische Daten VI-910

Die Technischen Daten beziehen sich auf den VI-910. Dies ist das Nachfolgemodell. Die meisten Daten sollten jedoch ähnlich sein.

Tabelle A.3.: Technische Daten - VI-910

Modellbezeichnung	Optischer 3D-Scanner VI-910
Messverfahren	Triangulation durch Lichtschnittverfahren
Autofokus	Autofokus auf Objektoberfläche (Kontrastverfahren); aktiver AF
Objektive (wechselbar)	TELE Brennweite f=25mm MITTEL: Brennweite f=14 mm WEIT: Brennweite f=8mm
Messabstand	0,6 bis 2,5m (2m für WIDE-Objektiv)
Optimaler Messabstand	0,6 bis 1,2m
Laserklasse	Class 2 (IEC60825-1), Class 1 (FDA)
Laser-Scanverfahren	Galvanisch-angetriebener Drehspiegel
Messbereich in X-Richtung (anhängig vom Anstand)	111 bis 463mm (TELE), 198 bis 823mm (MITTEL), 359 bis 1.196mm (WEIT)
Messbereich in Y-Richtung (abhängig vom Abstand)	83 bis 347mm (TELE), 148 bis 618mm (MITTEL), 269 bis 897mm (WEIT)
Messbereich in Z-Richtung (abhängig vom Abstand)	40 bis 500mm (TELE), 70 bis 800mm (MITTEL), 110 bis 750mm (WEIT/Modus FINE)
Genauigkeit	X: $\pm 0,22$ mm, Y: $\pm 0,16$ mm, Z: $\pm 0,10$ mm zur Z-Referenzebene (Bedingungen: TELE/Modus FINE , Konica Minolta Standard)
Aufnahmezeit	0,3s (Modus FAST), 2,5s (Modus FINE), 0,5s (COLOR)
Übertragungszeit zum Host-Computer	ca. 1s (Modus FAST) oder 1,5s (Modus FINE)
Scanumgebung, Beleuchtungsbedingungen	500 lx oder geringer

A. Anhang

Aufnahmeeinheit	3D-Daten: 1/3CCD-Bildsensor (340.000 Pixel) Farbdaten: Zusammen mit 3D-Daten (Farbtrennung durch Drehfilter)
Anzahl aufgenommener Punkte	3D-Daten: 307.000 (Modus FINE), 76.800 (Modus FAST) Farbdaten: $640 \times 480 \times 24$ Bit Farbtiefe
Ausgabeformat	3D-Daten: Konica Minolta Format, (STL, DXF, OBJ, ASCII-Punkte, VRML; Konvertierung in 3D-Daten durch Polygon Editing-Software / Standardzubehör) Farbdaten: RGB 24-Bit Rasterscan-Daten
Speichermedium	Compact Flash Memory Card (128MB)
Dateigrößen	3D- und Farbdaten (kombiniert): 1,6MB (Modus FAST) pro Datensatz, 3,6MB (Modus FINE) pro Datensatz
Monitor	5,7LCD ( $320 \times 240$ Pixel)
Datenschnittstelle	SCSI II (DMA-Synchronübertragung)
Stromversorgung	Normale Wechselstromversorgung, 100V bis 240 V (50 oder 60 Hz), Nennstrom 0,6 A (bei 100 V)
Abmessungen (B x H x T)	$213 \times 413 \times 271$ mm
Gewicht	ca. 11kg
Zulässige Umgebungsbedingungen (Betrieb)	10 bis 40°C; relative Luftfeuchtigkeit 65% oder niedriger (keine Kondensation)
Zulässige Umgebungsbedingungen (Lagerung)	-10 bis 50°C, relative Luftfeuchtigkeit 85% oder niedriger (bei 35°C, keine Kondensation)



## A.5. Verwendete Hardware

- **VI-900**

Konica Minolta Sensing Europe, B.V.

Website: <http://www.konicaminolta.eu/de/messinstrumente/produkte/3d-messtechnik/beruehrungsloser-3d-scanner/vi-910/einfuehrung.html>

- **ATMega 324A**

Atmel Corporation

Website: <http://www.atmel.com/devices/ATMEGA324A.aspx>

- **STK500**

Atmel Corporation

Website: <http://www.atmel.com/tools/STK500.aspx>

- **AVRISP mkII**

Atmel Corporation

Website: <http://www.atmel.com/tools/AVRISPMKII.aspx>

- **Induktiver Endschalter**

Pepperl+Fuchs

Website: [http://www.pepperl-fuchs.de/germany/de/classid\\_143.htm?view=productdetails&productid=3012](http://www.pepperl-fuchs.de/germany/de/classid_143.htm?view=productdetails&productid=3012)

- **MAX232**

Texas Instruments Incorporated

Website: <http://www.ti.com/product/max232>

## A.6. Verwendete Software

Hier ist die verwendete Software aufgelistet. Soweit es möglich war, wurden Open-Source-Programme eingesetzt. **(TODO: ÜBERARBEITEN!!!)**

- **RapidForm2004** (Closed Source)

INUS Technology, Inc.

Website: <http://www.rapidform.com>

- **AVRStudio 5** (Closed Source)

Atmel Corporation

Website: <http://www.atmel.com/>

*A. Anhang*

---

- **Eclipse** mit CDT und AVRPlugin  
The Eclipse Foundation Website: <http://www.eclipse.org>  
Website: <http://www.eclipse.org/>
- **AVRDude**  
Prorammer
- **Blender**
- **Texmaker**
- **LaTeX**
- **GIT**
- **Inkscape**
- **Hyperterminal** (Closed Source)

**(TODO: WEITERE?!)**