



**FACHHOCHSCHULE KIEL**  
**University of Applied Sciences**

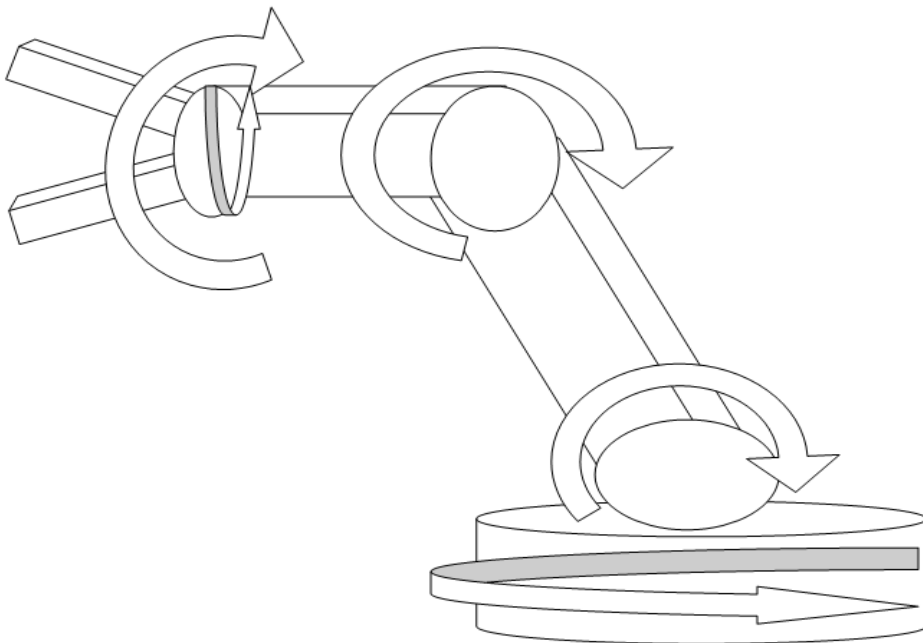
# CRAL

---

Ein Grundkonzept für einen Roboterarm

Erstellt von:

**Christian Kopreit, Lasse Otten**





## Inhalt

1	Anforderungsliste und Kurzbeschreibung .....	5
2	Planung .....	8
2.1	Projektplan .....	8
2.2	Systemübersicht .....	9
2.2.1	Übersicht zur Haupteinheit .....	9
2.2.2	Übersicht zur Motorsteuerung .....	10
2.3	Stückliste .....	11
3	Hauptsteuereinheit und Gehäuse .....	12
3.1	Vereinfachter Verdrahtungsplan Gehäuse .....	14
3.2	Hauptplatine .....	15
3.2.1	Schaltplan Hauptplatine .....	15
3.3	Erläuterung der Wartebedingung .....	19
4	Motorsteuerung .....	21
4.1	Zusammenfassung der Aufgabe .....	21
4.2	Beschreibung der Motorsteuerungssoftware .....	22
4.3	Schaltplan Motorsteuerplatine .....	26
4.3.1	Spannungsteiler und Strombegrenzung .....	27
4.3.2	Steuerung .....	29
4.3.3	Motortreiber .....	30
4.3.4	Klemmen .....	31
4.3.5	TWI & Clock .....	32
5	Kommunikation .....	33
5.1	TWI .....	33
5.2	RS232 .....	34
6	Windowsoberfläche .....	35
6.1	Statusanzeige .....	36
6.2	Befehlseingabe .....	37
6.3	Befehlsdarstellung .....	39
6.4	Verbindungslog .....	41
6.5	RS232-Ports .....	42
6.6	Tabellenbefehle .....	43
6.7	Gespeicherter Bewegungsablauf .....	44
7	Quellenangabe .....	45

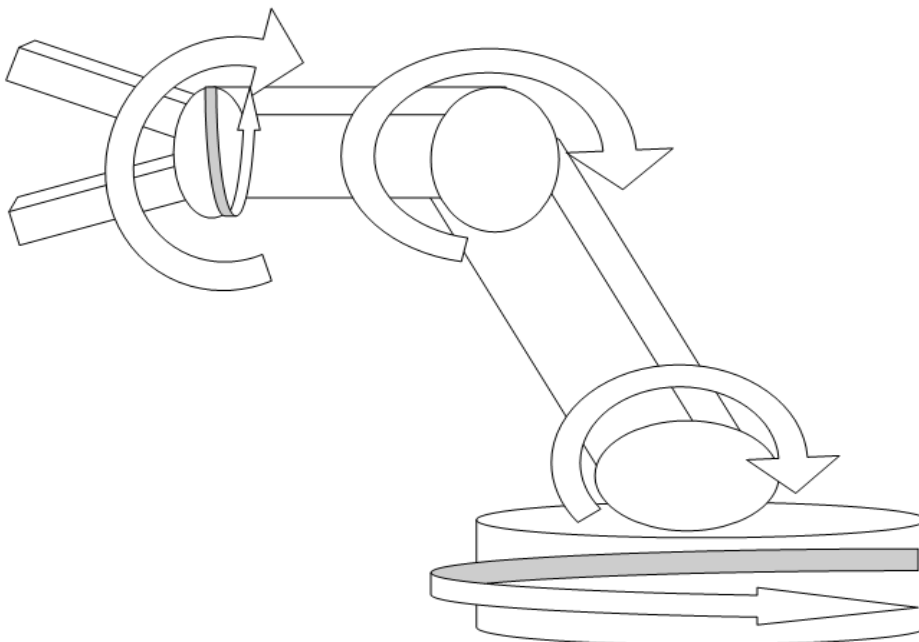
8	Anhang .....	46
---	--------------	----

## 1 Anforderungsliste und Kurzbeschreibung

### Auftrag für ein Semesterprojekt der Robotik AG NorthernStars

#### Anforderungsliste und Kurzbeschreibung:

Ziel ist die Entwicklung und Konstruktion eines funktionsfähigen 5-achsigen Roboterarms, der dem menschlichen Arm in Beweglichkeit und Bewegungsradius ähnelt. Das Operationswerkzeug soll ein einfacher Greifer sein, mit dem Objekte verschiedener Größen gepackt und im Operationsradius bewegt werden können. Der Arm soll einer fest programmierten Bewegungsanweisung folgen. Die folgende Skizze zeigt eine schematische Darstellung des Arms.



Bei Fragen und Anregungen ist Hannes Eilers, 1. Vorsitzender der Hochschulgruppe *NorthernStars*, zu kontaktieren. Im Anhang finden Sie unser Lastenheft.

## **Anforderungsliste**

### **Beschreibung des Ist-Zustands:**

- keine Vorlage, Neuentwicklung

### **Beschreibung des Soll-Zustands:**

- 5-achsiger Roboterarm
  - Standfuß mit horizontaler Achse ca. 360° drehbar
  - Unterste vertikale Achse ca. 120° schwenkbar
  - Oberes Gelenk ca. 120° schwenkbar
  - Handgelenk ca. 180° kippbar und ca. 240° drehbar
- Einfacher Greifer
  - Mit mindestens zwei Fingern
  - Eventuell mit Gelenken und/oder mit Tastsensoren
- Physikalische Begebenheiten
  - Eine Gesamtlänge von 0,5m bis 3m
  - Tragkraft bei voll ausgestrecktem Arm von mindestens 1kg
  - Verwendete Materialien müssen beständig sein
  - Schutzklasse nach DIN40050 Teil 9 min IP40
  - Betrieb an 230V 50Hz
- Sonstiges
  - Ansteuerung soll über einen Mikrocontroller realisiert werden
  - Programmiersprache: C oder C++

### **Beschreibung der Schnittstellen**

- Programmierschnittstellen:
  - Hauptschnittstelle TWI oder RS-232 oder ähnliche

## Anforderungen

- Es soll möglich sein, dem Roboter einen festen Bewegungsablauf einzuprogrammieren, den er dann selbständig ausführt. Des Weiteren soll es die Option geben, dass der Arm mit Hilfe eines Bedienmoduls manuell gesteuert werden kann. Diese Option soll gegebenenfalls in einem weiteren Semester-Projekt entwickelt werden. Der Roboter soll folgende Aufgaben bewältigen können:
  - Greifen eines Gegenstandes bis zu einem 1kg an fester Position
  - Herübertragen des Gegenstandes an eine feste Position, die sich auch in einer anderen Höhe befinden kann
  - So könnte die Aufgabe z.B. lauten: Heben einer Wasserflasche vom Fußboden auf einen nahe gelegenen Tisch, ohne dass sie umkippt

## Lieferumfang und Abnahmekriterien

- Zu liefern ist der Roboter mindestens wie beschrieben:
  - Als komplette Einheit
  - oder zur schnellen Inbetriebnahme durcheinfache Montage von einzelnen Modulen, wenn dies durch fehlende Kompetenzen nicht erbracht werden kann, dann ist der Mindestumfang die elektrotechnische Ausarbeitung.
- Zu liefern ist bis Ende des Wintersemesters, bei einem Semester Konstruktionszeit oder bis zum Ende des Sommersemesters, bei zwei Semestern Konstruktionszeit.
- Außerdem wird eine Unterweisung der Robotik AG *NorthernStars* in die Bedienung und Besonderheiten bei der Programmierung abgegeben.

## 2 Planung

### 2.1 Projektplan

Der erste Schritt eines Projekts sollte immer die Planung sein:

Nr.	Vorgangsr	Vorgangname	Dauer	Anfang	Fertig stellen	Vorgänger
1		<b>0</b>	<b>10 Tage</b>	<b>Mi 17.10.12</b>	<b>Di 30.10.12</b>	
2	✓	Auftrag analysieren	1 Tag	Mi 17.10.12	Mi 17.10.12	
3	✓	Projekt planen	1 Tag	Do 18.10.12	Do 18.10.12	2
4		Konzept erarbeiten	7 Tage	Fr 19.10.12	Mo 29.10.12	3
5		Pflichtenheft erstellen	1 Tag	Di 30.10.12	Di 30.10.12	4
6		Pflichtenheftabnahme	0 Tage	Di 30.10.12	Di 30.10.12	5
7		<b>Entwurf</b>	<b>10 Tage</b>	<b>Mi 31.10.12</b>	<b>Di 13.11.12</b>	<b>1</b>
8		Stückliste erstellen	10 Tage	Mi 31.10.12	Di 13.11.12	6
9		Kosten planen	10 Tage	Mi 31.10.12	Di 13.11.12	6
10		Schaltplan entwerfen	3 Tage	Mi 31.10.12	Fr 02.11.12	6
11		<b>Layout erstellen</b>	<b>7 Tage</b>	<b>Mo 05.11.12</b>	<b>Di 13.11.12</b>	
12		Platinenlayout erstellen	1 Tag	Mo 05.11.12	Mo 05.11.12	10
13		Auftrag für Konstruktionslayout ausschreiben	7 Tage	Mo 05.11.12	Di 13.11.12	10
14		<b>Programmablaufplan erstellen</b>	<b>1 Tag</b>	<b>Mo 05.11.12</b>	<b>Mo 05.11.12</b>	
15		Ablaufplan für Windows-Programm erstellen	1 Tag	Mo 05.11.12	Mo 05.11.12	10
16		Ablaufplan für Microcontroller erstellen	1 Tag	Mo 05.11.12	Mo 05.11.12	10
17		Windowsoberfläche gestalten	1 Tag	Di 06.11.12	Di 06.11.12	15
18		<b>Realisierung eines Prototypen</b>	<b>44 Tage</b>	<b>Mi 14.11.12</b>	<b>Mo 14.01.13</b>	<b>7</b>
19		Teile ordern	7 Tage	Mi 14.11.12	Do 22.11.12	
20		<b>Informationstechnischer Bereich</b>	<b>35 Tage</b>	<b>Di 27.11.12</b>	<b>Mo 14.01.13</b>	<b>36</b>
21		<b>Windows Bereich</b>	<b>35 Tage</b>	<b>Di 27.11.12</b>	<b>Mo 14.01.13</b>	
22		Oberfläche aufbauen	3 Tage	Di 27.11.12	Do 29.11.12	
23		Elemente programmieren	32 Tage	Fr 30.11.12	Mo 14.01.13	22
24		Kommunikationsebene programmieren	32 Tage	Fr 30.11.12	Mo 14.01.13	22
25		<b>Microcontroller Bereich</b>	<b>35 Tage</b>	<b>Di 27.11.12</b>	<b>Mo 14.01.13</b>	
26		<b>Steuerung der Aktoren</b>	<b>35 Tage</b>	<b>Di 27.11.12</b>	<b>Mo 14.01.13</b>	
27		Motoransteuerung programmieren	35 Tage	Di 27.11.12	Mo 14.01.13	
28		Auslesen der Sensoren programmieren	35 Tage	Di 27.11.12	Mo 14.01.13	
29		Kommunikationsebene programmieren	35 Tage	Di 27.11.12	Mo 14.01.13	
30		<b>Mechanischer Bereich</b>	<b>36 Tage</b>	<b>Mi 14.11.12</b>	<b>Mi 02.01.13</b>	
31		Auftrag für Konstruktion vergeben	21 Tage	Mi 14.11.12	Mi 12.12.12	
32		Kontrollieren der mechanischen Elemente	1 Tag	Do 13.12.12	Do 13.12.12	31
33		Elemente zusammensetzen	14 Tage	Fr 14.12.12	Mi 02.01.13	32
34		<b>Elektrischer Bereich</b>	<b>41 Tage</b>	<b>Mi 14.11.12</b>	<b>Mi 09.01.13</b>	
35		Platine ätzen	7 Tage	Mi 14.11.12	Do 22.11.12	
36		Schaltung aufbauen und verlöten	2 Tage	Fr 23.11.12	Mo 26.11.12	19
37		Steuerleitungen verlegen	4 Tage	Do 03.01.13	Di 08.01.13	33
38		Netzanschluss realisieren	1 Tag	Mi 09.01.13	Mi 09.01.13	37
39		<b>Erprobung</b>	<b>14 Tage</b>	<b>Di 15.01.13</b>	<b>Fr 01.02.13</b>	<b>18</b>
40		Analyse verbotener Zustände	7 Tage	Di 15.01.13	Mi 23.01.13	
41		Einbinden der Analysewerte in die Programmierung	7 Tage	Do 24.01.13	Fr 01.02.13	40
42		Einweisung/Übergabe	1 Tag	Mo 04.02.13	Mo 04.02.13	39

Abbildung 1 : Projektplan

Der anfängliche Projektplan konnte nicht gänzlich eingehalten werden. Der mechanische Bereich musste aufgrund eines abgesprungenen Kommilitonen leider gänzlich gestrichen werden. Somit blieb uns mehr Zeit für die Erprobung der verbliebenen Bereiche, die wir auch bis in den Februar hinein fortgesetzt haben.

## 2.2 Systemübersicht

### 2.2.1 Übersicht zur Haupteinheit

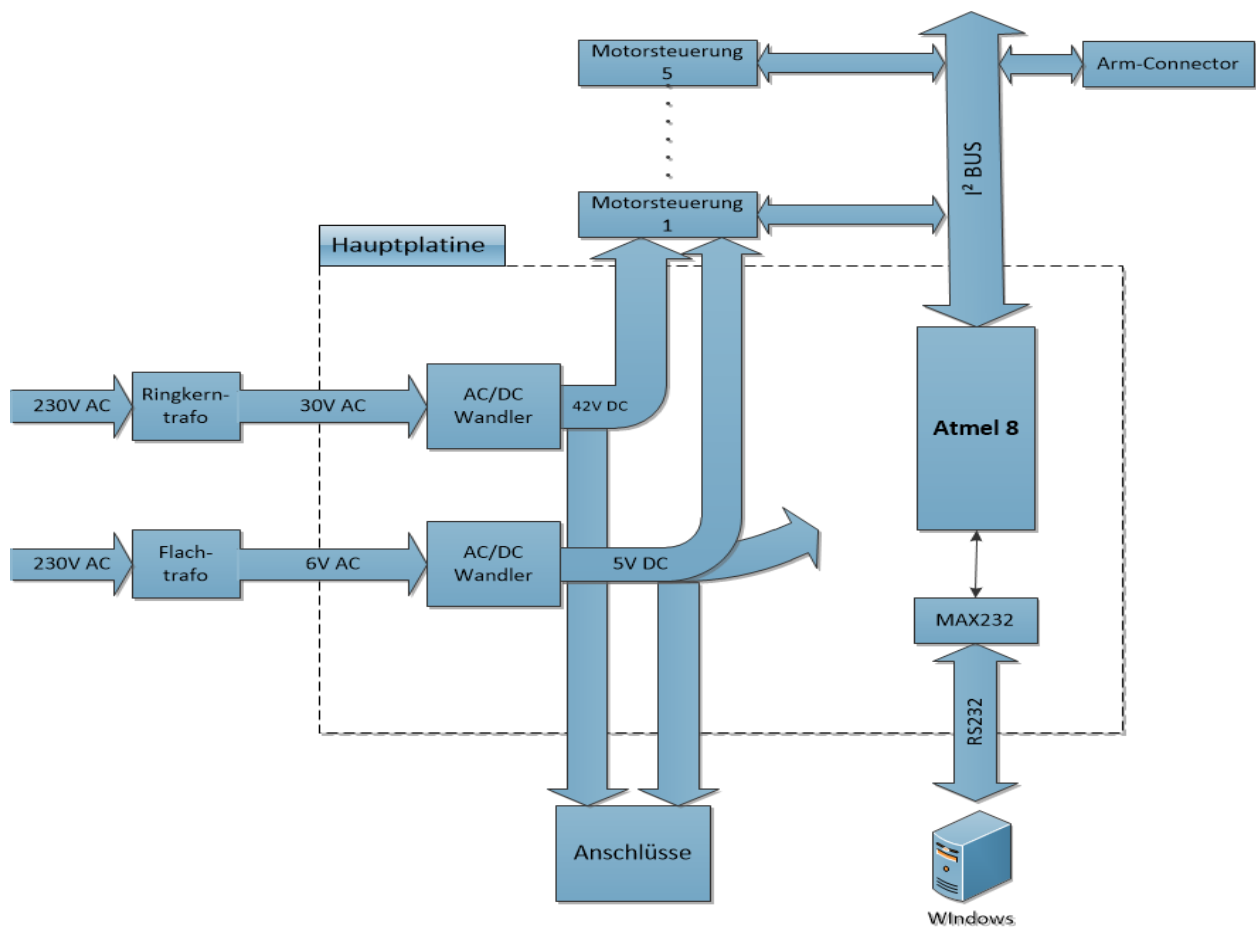


Abbildung 2: Blockschaltbild Haupteinheit

Der Roboterarm soll von einer zentralen Einheit gesteuert werden. Diese Einheit wird gleichzeitig als fester Standfuß dienen und ebenfalls das erste Armelement mit Motor, Ansteuerung und Übersetzung enthalten.

Die Abbildung oben zeigt den schematischen Aufbau dieser Haupteinheit, die hauptsächlich aus einem Netzteil und aus einer Kommunikationseinheit basierend auf dem Atmega 8 und der Weiterleitung von Spannungen und Signalen besteht.

Eine nähere Beschreibung der einzelnen Elemente erfolgt in Kapitel 3.

## 2.2.2 Übersicht zur Motorsteuerung

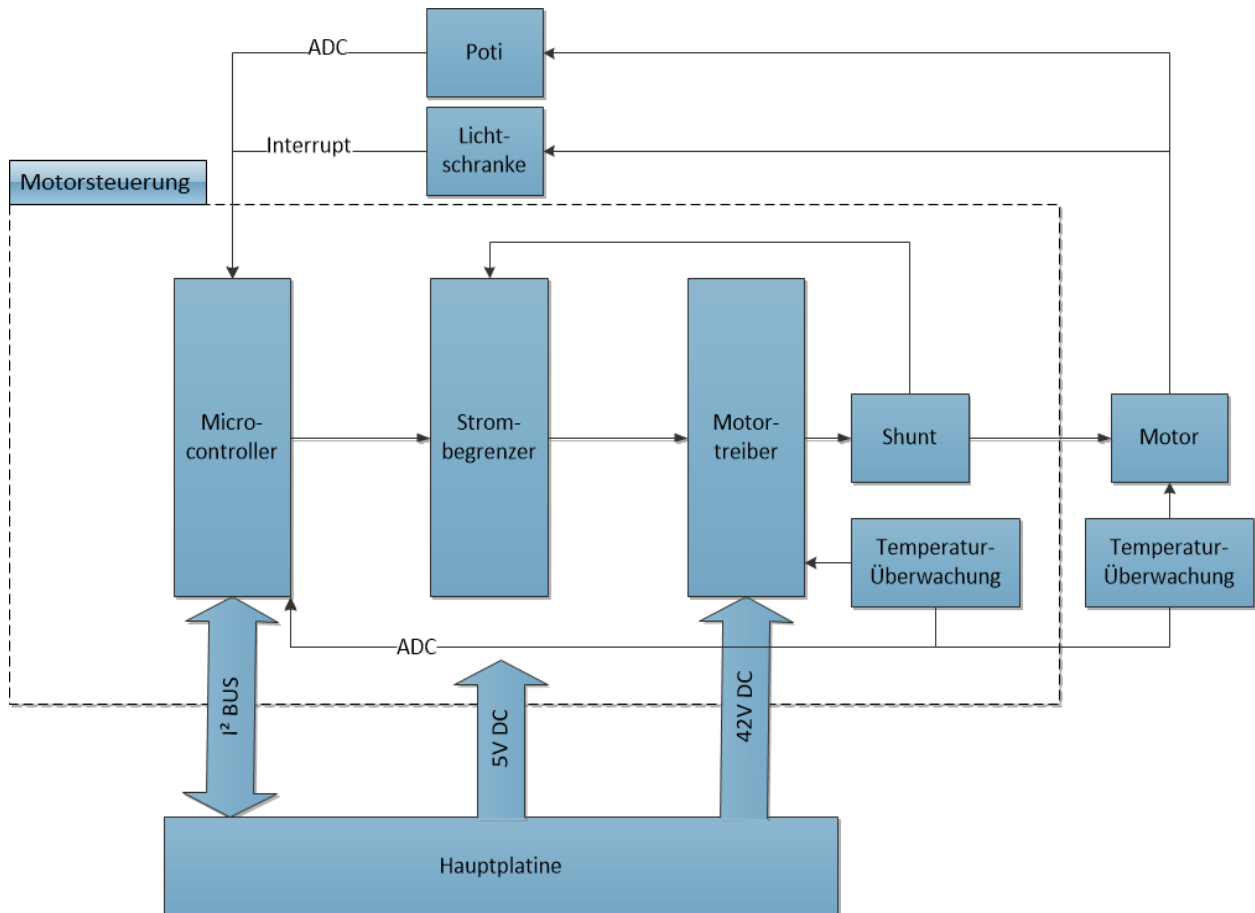


Abbildung 3: Blockschaltbild Motorsteuerung

Die Motorsteuerung kümmert sich um die direkte Ansteuerung eines einzelnen Motors und muss daher in mehrfacher Ausführung, je nach Motoranzahl, angefertigt werden.

Die Befehle werden von der Haupteinheit mittels I<sup>2</sup>C an die einzelnen Motoren übermittelt und dort ebenfalls von einem Atmega 8 verarbeitet. Soll der Motor sich bewegen, muss dank des Einsatzes der Motortreiberbausteine L297 & L298 nur die Frequenz, in der sich der Motor drehen soll und ein Bitmuster für die Bewegungsart ausgegeben werden (z.B.: Halfstep oder Links- und Rechtslauf können so mit dem einfach Setzen von Bits ausgewählt werden). Der L297 ist ebenfalls für die Strombegrenzung verantwortlich und misst den momentan fließenden Strom über Shunt-Widerstände.

Eine Schrittfehlerüberwachung findet mittels Inkrementalgeber statt.

Des Weiteren kann die analoge Position mittels Potentiometer jederzeit eingelesen werden.

Die Temperaturüberwachung der kritischen Bauteile findet mittels zweier NTCs statt.

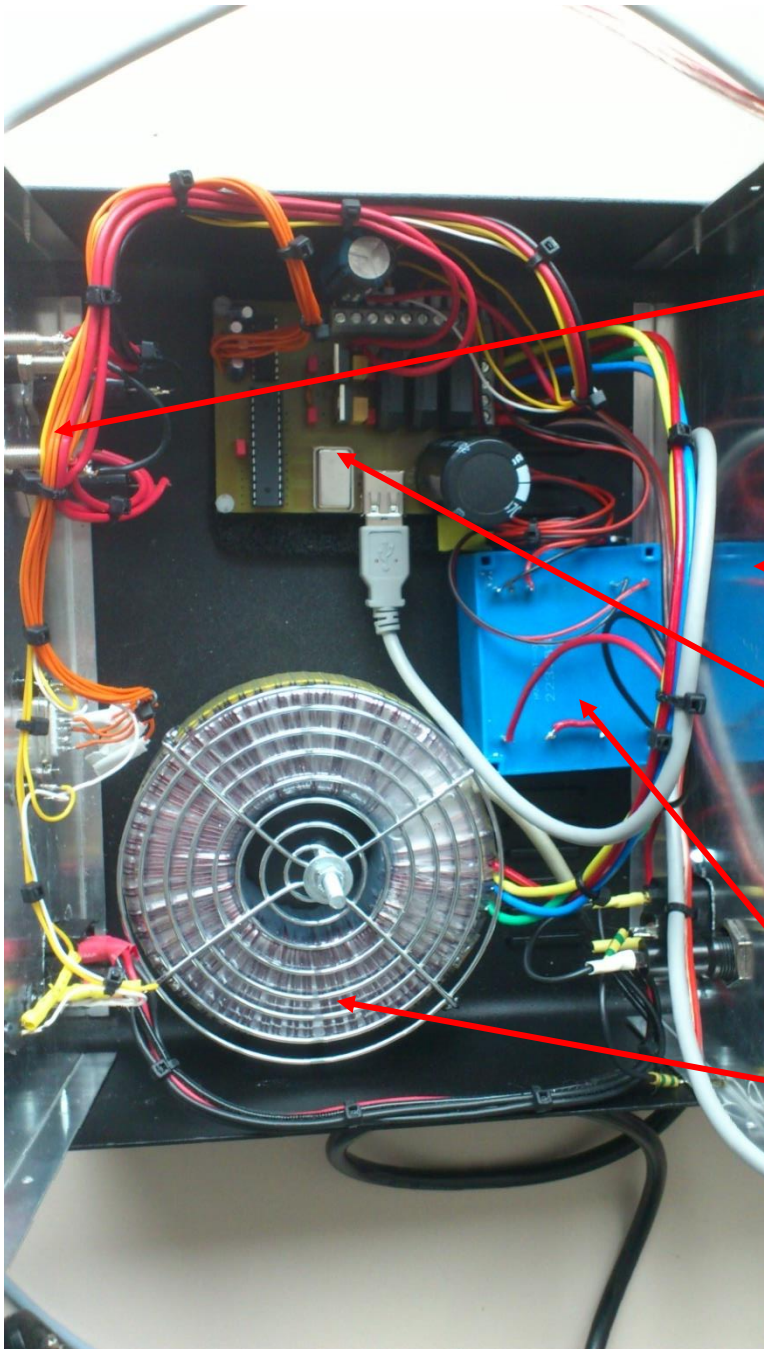
Eine genauere Beschreibung der einzelnen Elemente ist in Kapitel 4 zu finden.

## 2.3 Stückliste

1	Anreihklemmen 4-polig	Reichelt	AKL 055-04	13	0,390	5,07 €
2	Bananenbuchse Rot	Reichelt	BB 4 RT	2	0,300	0,60 €
3	Bananenbuchse Schwarz	Reichelt	BB 4 SW	2	0,300	0,60 €
4	Buchse Floppy	Reichelt	PSG 5	5	0,240	1,20 €
5	Diodennetzwerk	Reichelt	L6210	5	1,650	8,25 €
6	Elko-Kondensator 470µF	Reichelt	RAD 470/63	9	0,140	1,26 €
7	Flachtrafo 30VA	Reichelt	UI39/21 206	1	10,550	10,55 €
8	Gehäuse 250x150x200 BxHxT	Conrad	520497-62	1	25,790	25,79 €
9	Gleichrichter 4A	Reichelt	FBU4A	2	0,440	0,88 €
10	Hybrid-Schrittmotor	Reichelt	QSH5718-51-101	2	44,600	89,20 €
11	Hybrid-Schrittmotor	Reichelt	QSH4218-35-026	2	23,650	47,30 €
12	Hybrid-Schrittmotor	Reichelt	QSH4218-51-049	1	29,950	29,95 €
13	Kaltgeräte Stecker, Einbau	Reichelt	KES 1	1	0,750	0,75 €
14	Kondensator 1µF	Reichelt	RAD 1/63	4	0,040	0,16 €
15	Kondensator 100 nF	Reichelt	MKS-02 100n	21	0,140	2,94 €
16	Kondensator 3,3nF	Reichelt	KERKO 3,3n	7	0,060	0,42 €
17	Kondensator 4.7mF	Reichelt	BSN 4.700/63	1	2,200	2,20 €
18	Kühlkörper SMD IC	Reichelt	V 5619B	2	0,540	1,08 €
19	Lichtschanke TCST 2103	Reichelt	CNY 37	5	0,940	4,70 €
20	Microcontroller	Reichelt	ATMEGA 48a-pu	6	1,300	7,80 €
21	Motor Driver	Reichelt	L297	5	2,350	11,75 €
22	Multiwatt-Treiber	Reichelt	L298	5	2,100	10,50 €
23	NTC 5% 4,7k Seite11	Reichelt	NTC-0,2 4,7K	10	0,360	3,60 €
24	Oszillator	Reichelt	OSZI 8,0000	6	0,860	5,16 €
25	Ringkerntrafo 160VA	Reichelt	RKT16015	1	31,450	31,45 €
26	RS232 Treiber	Reichelt	MAX 232 cpe	1	0,370	0,37 €
27	Schraubsicherung 10A	Reichelt	PL126000	5	0,330	1,65 €
28	Shunt Draht 2,5m	Conrad	429015-62	1	3,070	3,07 €
29	Spannungsregler 7805	Reichelt	µA 7805	4	0,270	1,08 €
30	Stecker Floppy	Reichelt	SVK 5	5	0,220	1,10 €
31	USB Buchse	Reichelt	USB AW Serie a	12	0,170	2,04 €
32	USB Kabel 1m	Reichelt	AK 670/2-1,0	6	0,600	3,60 €
33	Widerstände	Reichelt	METALL 22,0K	100	0,025	2,50 €
34	Widerstände	Reichelt	METALL 100	100	0,025	2,50 €
35	Widerstände	Reichelt	METALL 30,0K	100	0,025	2,50 €
36	Widerstände	Reichelt	METALL 10,0K	100	0,025	2,50 €
37	Widerstände	Reichelt	METALL 1,00K	100	0,025	2,50 €
38	Widerstände	Reichelt	METALL 316	100	0,025	2,50 €
39	Gleichrichter 6A	Reichelt	KBU6B	1	0,460	0,46 €
40	SUB-D Stecker	Reichelt	D-SUB BU 09	1	0,130	0,13 €
41	Kabelschutz 3m	Reichelt	SPIRAL 20-3	1	0,950	0,95 €
42	Kabelbinder	Reichelt	KAB-U 100-2,5	10	0,410	4,10 €
43	IC Sockel	Reichelt	GS 28P-s	1	0,420	0,42 €
44	Potenzimeter	Conrad	452881 - 62	4	0,800	3,20 €

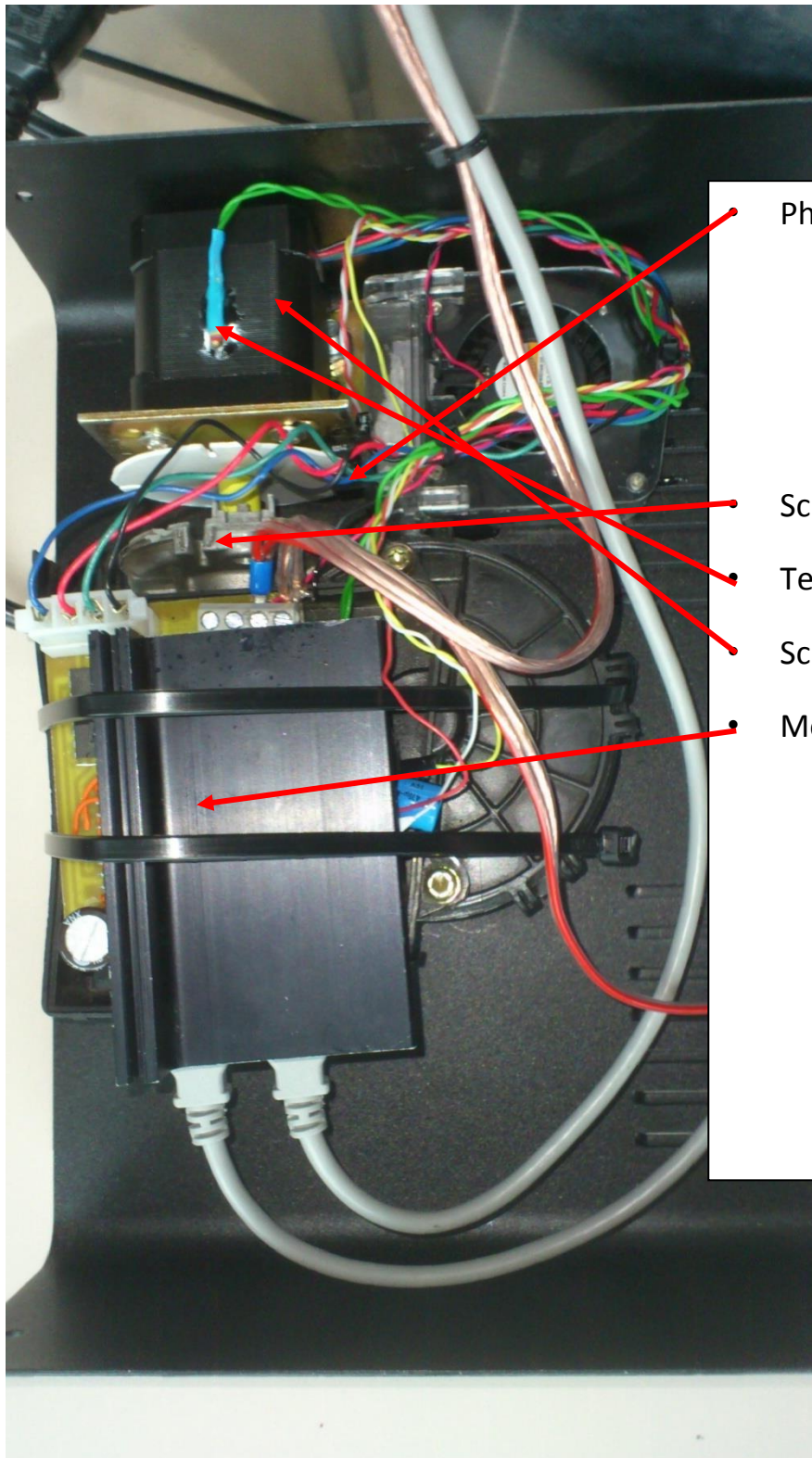
Abbildung 4: Stückliste

### 3 Hauptsteuereinheit und Gehäuse



- Frontpanel
- Bedienelemente, Kontroll-LEDs, Schnittstelle
- Sicherungen
- Rückseite
- Netzanschluss, Netzsicherung
- Hauptsteuerplatine
- 5V-Festspannungsnetzteil, 20V-4BU-Netzteil,
- Steuersegment mit Atmega 8
- Flachtrafo 2x6V AC á 1A
- Ringkerntrafo 2x 18V AC á 6,6A

Abbildung 5: Draufsicht Gehäuse



Photoelektrische Abtastung

Momentane Auflösung von 50 Schritten kann bei Bedarf auf bis zu 2 Schritte genau eingestellt werden

Schneckengetriebe (1:45)

Temperaturfühler PTC-Fühler

Schrittmotorrotor (Größe = Mittel)

Motorsteuereinheit

Nähere Beschreibung siehe Kapitel 4

Abbildung 6: Draufsicht unterstes Gelenk

### 3.1 Vereinfachter Verdrahtungsplan Gehäuse

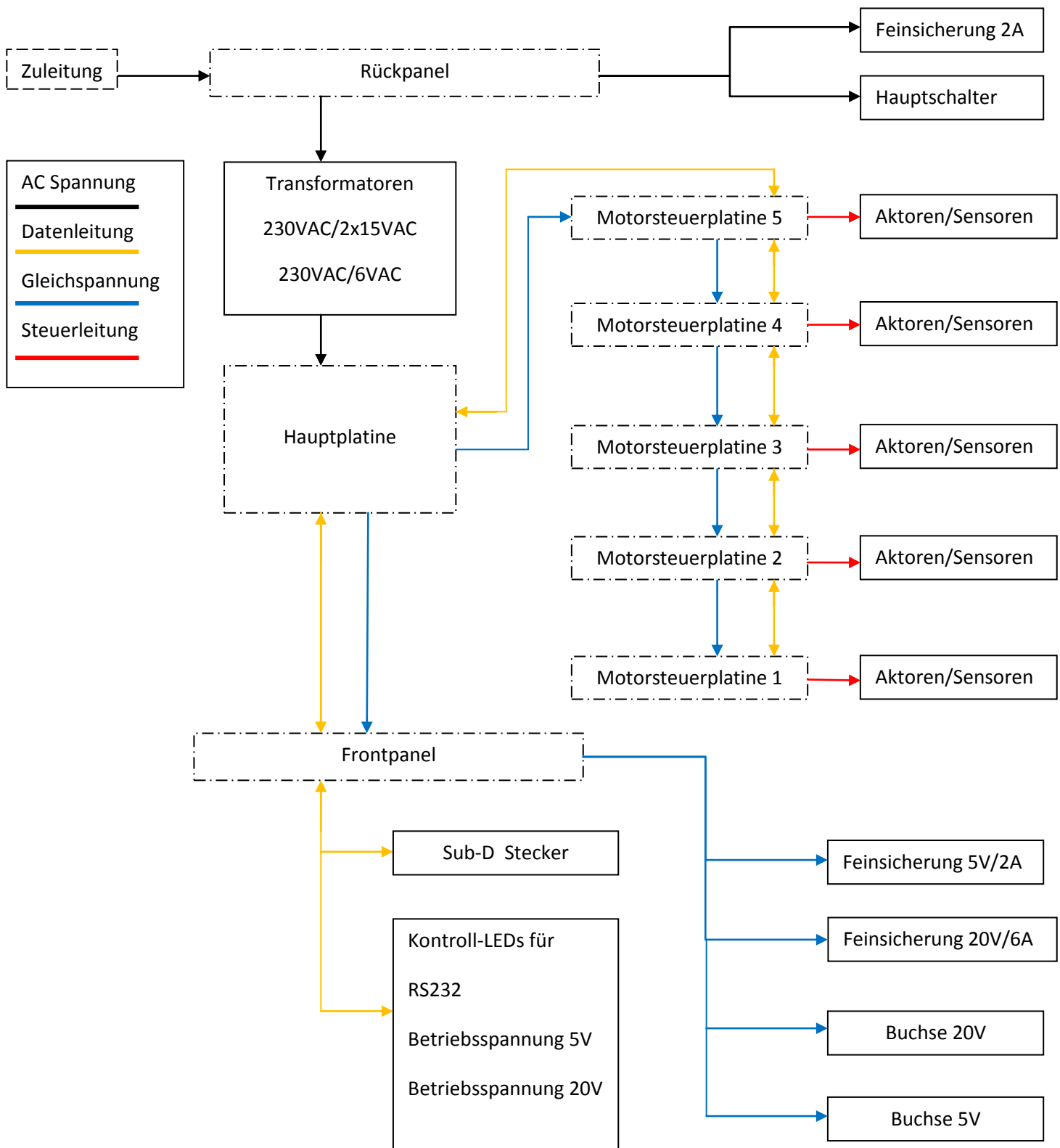


Abbildung 7: Verdrahtungsplan

## 3.2 Hauptplatine

### 3.2.1 Schaltplan Hauptplatine

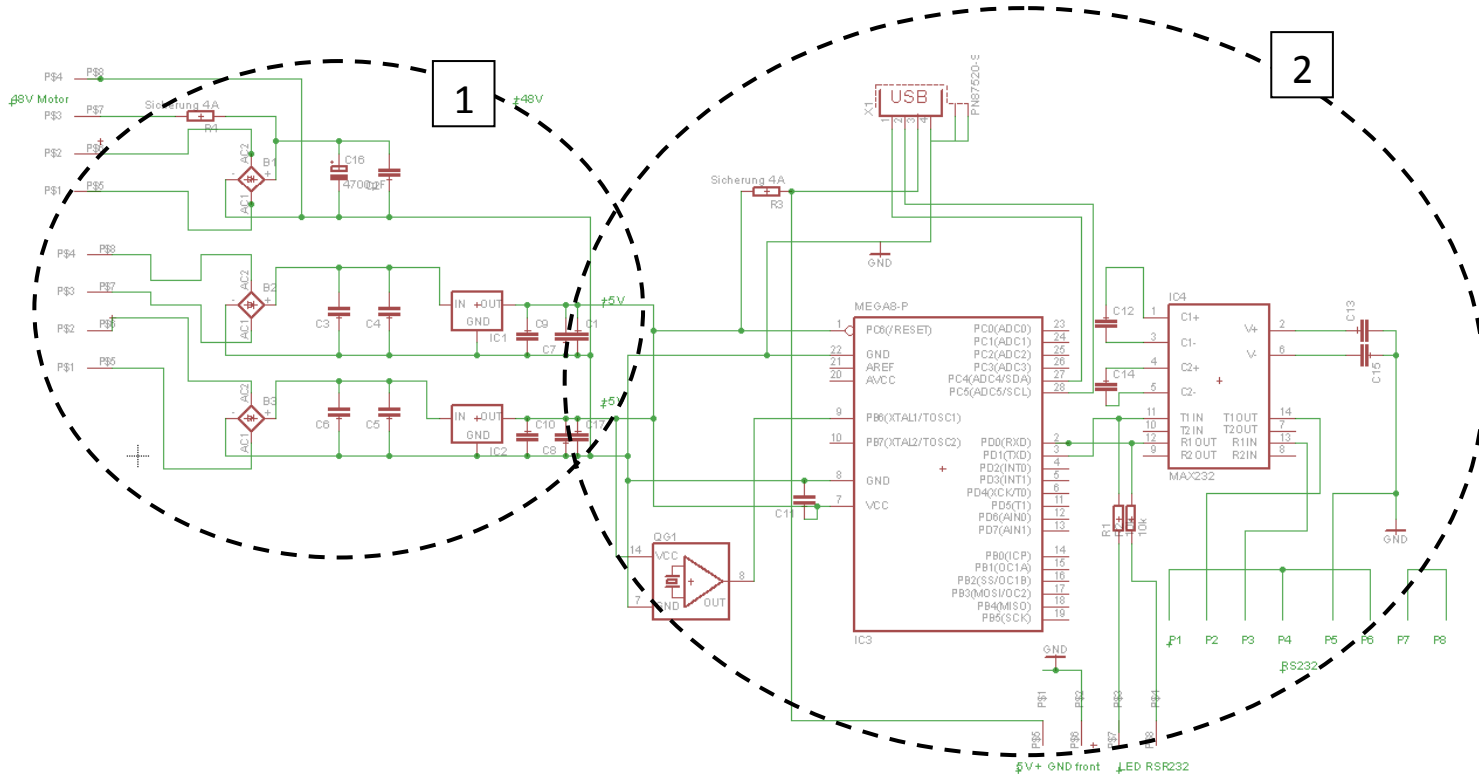


Abbildung 8: Schaltplan Hauptplatine - 1.Spannungsversorgung 2. Steuerbereich

Die Hauptplatine unterteilt sich grob in die beiden Bereiche Spannungsversorgung und Steuerteil. Wie die Namen sagen, kümmert sich das Netzteil um die Spannungsversorgung des gesamten Arms und muss deshalb entsprechend großzügig ausgelegt sein. Der Steuerkreis wird durch einen Atmega 8 gebildet, der mit 8MHz getaktet wird.

Die beiden Bereiche sollen im Folgenden näher betrachtet werden.

### 3.2.1.1 Spannungsversorgung:

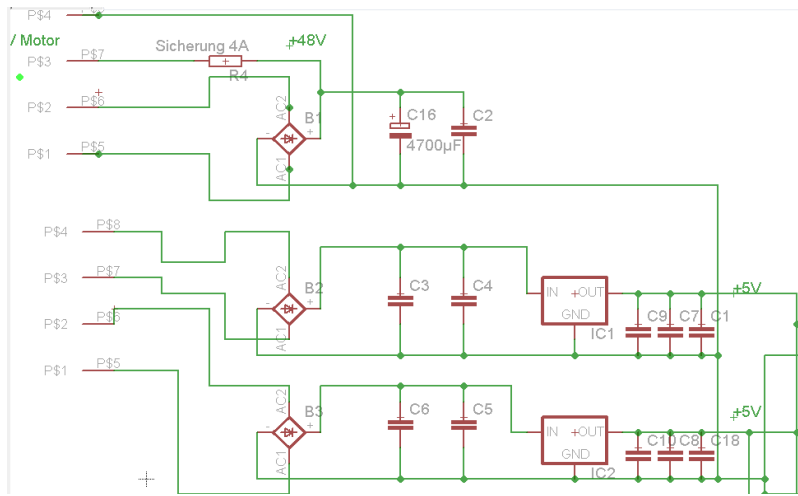


Abbildung 9: Schaltplanausschnitt Hauptplatine - Spannungsversorgung

Hier zu sehen sind drei 4BU-Gleichrichtungen (B1,B2& B3) mit anschließender Glättung durch Kondensatoren. Der 5V-Part ist zusätzlich mit dem Festspannungsregler  $\mu$ A 7805 (IC1 & IC2) bestückt, um eine stabile 5V-Versorgung zu garantieren. Da wir am späteren Prototypen festgestellt haben, dass die Pufferung nicht ausreicht, haben wir zusätzlich einen 2200  $\mu$ F Kondensator hinzugefügt.

Dies ist unter anderem wichtig, da die Analog/Digitalwandlung mit der externen Referenzspannung versorgt wird. So lassen sich Schwankungen in den Messergebnissen vermindern.

Des Weiteren ist der gesamte Stromkreis mit einer Hauptsicherung geschützt und zusätzlich werden die Buchsen zum Abgriff der Spannungen ebenfalls mit einer Feinsicherung gegen zu große Leistungsentnahme abgesichert.

Versorgt wird der 5V-Part mit einem Flachtrafo der 2\*6V und bis zu 30W Leistung liefert. Bei einem Wirkungsgrad von 81% können so 24W bzw. 6V/4A bereitgestellt werden.

Hinweis! Momentan sind nur zwei Verspannungsregler (7805) verbaut. Deshalb können nur 2A abgerufen werden. Dies reicht für alle Steuerplatinen aus (0,5A je Platine), will man aber an den Frontabgriffen weitere Bauelemente anschließen, muss mit weiteren parallelen 7805, oder besser größer dimensionierten, für genügend Leistungsfestigkeit gesorgt werden.

Der Leistungsteil wird mit einem Ringkerntrafo mit 160VA 2\*15V AC parallel versorgt.

$$15V * \sqrt{2} = 21,21V$$

Es ergibt sich also eine Spannung von ca. 21V DC, welche nur mittels Kondensatoren gepuffert wird. Eine Festspannungsregelung ist hier nicht notwendig, da nur die Motoren damit angetrieben werden und ein Spannungsschwanken nur eine Änderung der Beschleunigung und des Drehmoments zur Folge hätten.

### 3.2.1.2 Steuerung

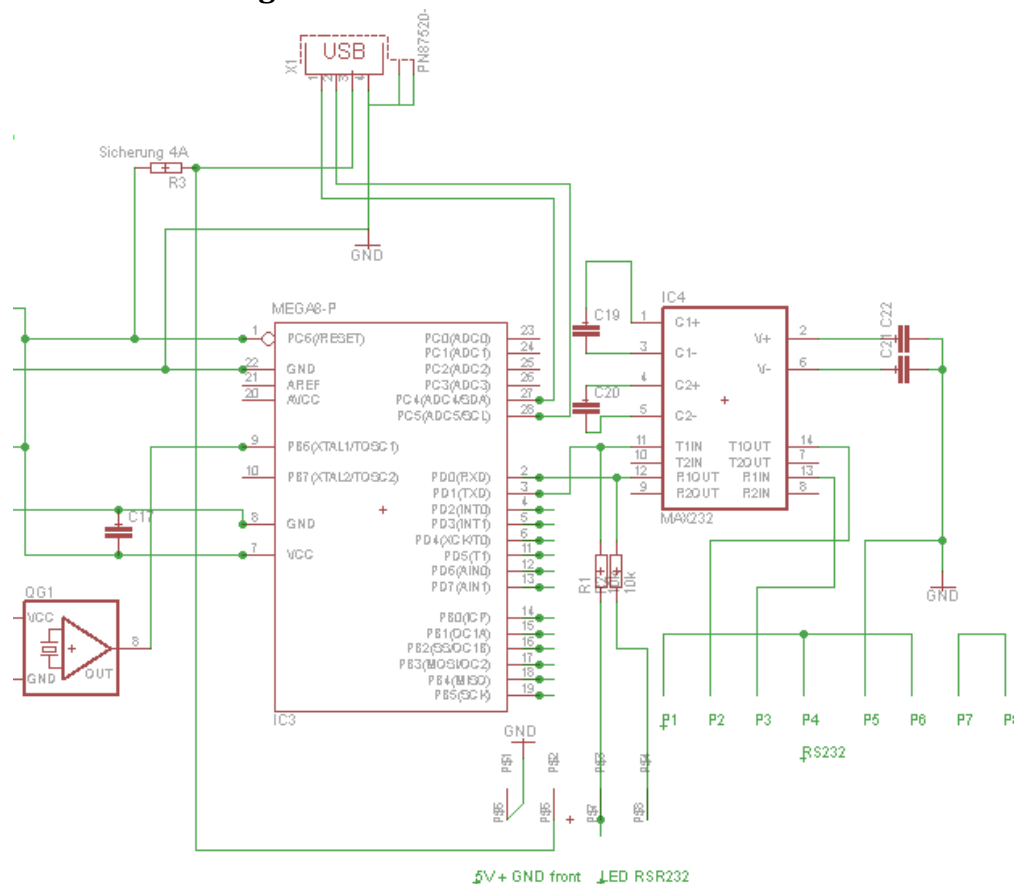


Abbildung 10: Schaltplanausschnitt Hauptplatine - Steuerteil

Dieser Schaltplan zeigt zum einen den Microcontroller, der die Datenverwaltung übernimmt, sowie den Treiber für die RS232-Verbindung (IC4). Bei dem Microcontroller handelt es sich um einen Atmega 8-16PU (IC3). Seinen Takt erhält er von dem Quarz (QG1), der mit einer Frequenz von 8MHz schwingt. Der Atmega 8 nimmt die Daten für die Bewegung vom PC entgegen und wandelt diese intern von ASCII in Hex um. Wird eine Tabelle gestartet, werden diese Daten über einen I<sup>2</sup>-Bus an die entsprechenden Motoren weitergeleitet. Für die I<sup>2</sup>-Schnittstelle haben wir eine USB-Ausführung verwendet, was zum einen den Vorteil hat, dass Steuerleitungen und Buchsen fertig gekauft werden können und zum anderen kann die 5V-Steuerspannung wie üblich übertragen werden.

So können die Signale ohne Probleme an der letzten Motorsteuerplatine abgegriffen werden.

Die Beschaltung des RS232-Treibers wurde nach dem Vorbild aus der Vorlesung von Professor Dr. Weber gestaltet.

Die freien Pins des Microcontrollers sind auf der Platine ausgeführt und können für spätere Erweiterungen genutzt werden. Der Controller sitzt dementsprechend auf einem Sockel.

Nachfolgend dargestellt ist der Programmablaufplan.

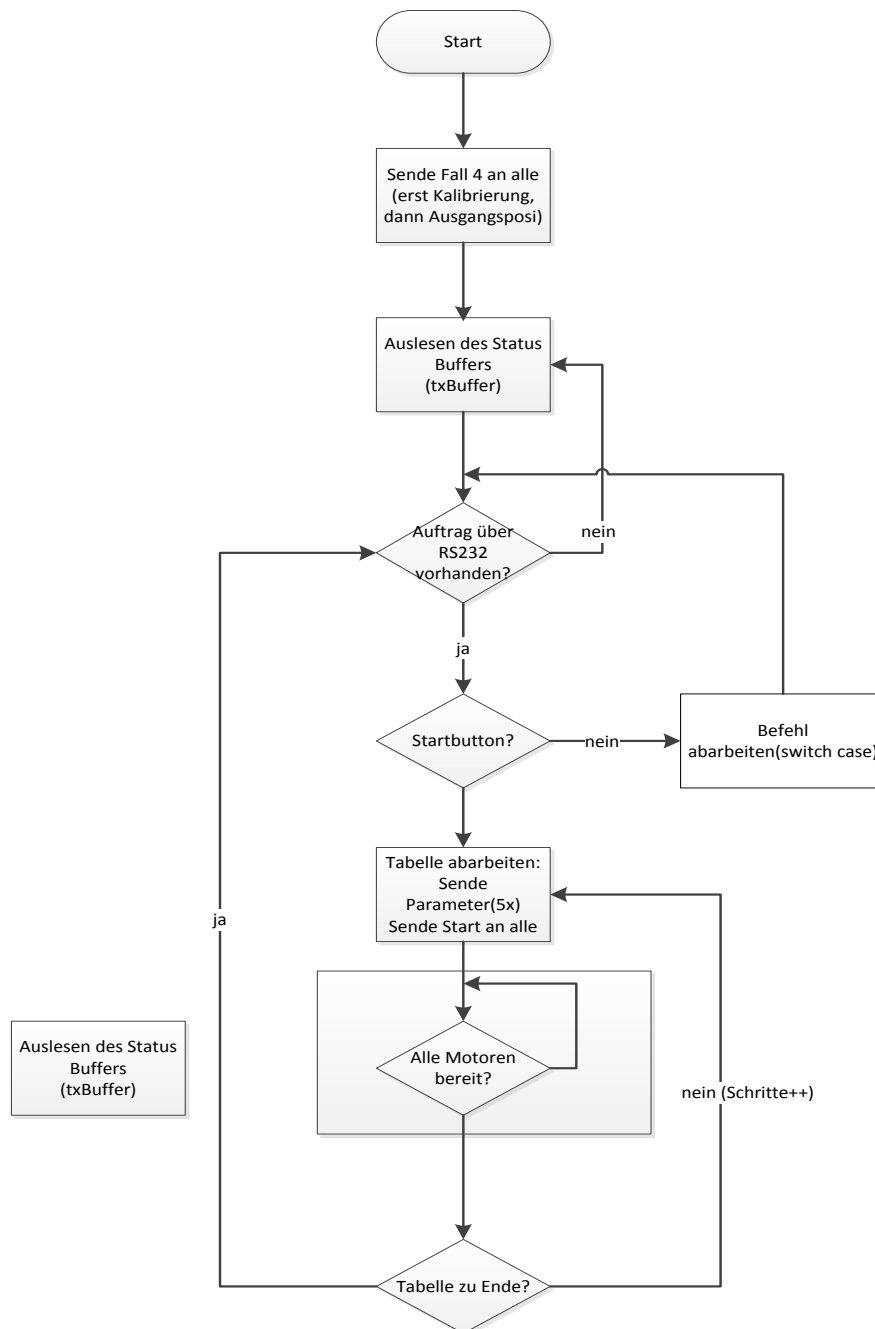


Abbildung 11: Ablaufplan Hauptplatine

Nach dem Einschalten des Gerätes soll eine Kalibrierung erfolgen. Diese ist allerdings erst sinnvoll, nachdem der Arm komplettiert worden ist. In der vorliegenden Form ist dieser Schritt nicht implementiert.

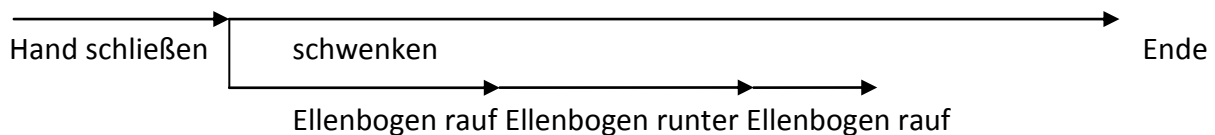
Die Hauptschleife prüft ständig den Status der Motoren und sendet diesen über RS232 an das Frontend. Weiterhin wird bei einer empfangenen Nachricht vom Frontend diese verarbeitet und gegebenenfalls in einer Bewegungstabelle abgespeichert, welche nach dem Befehl „Start“ abgearbeitet wird. Damit die Bewegung sehr variabel gestaltet werden kann, haben wir die Möglichkeit von Wartebedingungen eingefügt.

### 3.3 Erläuterung der Wartebedingung

Damit die Tabelle abgearbeitet wird, muss vor dem Start des nächsten Schrittes eine Wartebedingung erfüllt sein. So soll es ermöglicht werden, verschiedene Aktoren zu unterschiedlichen Zeiten anzusprechen. Ein Beispiel soll den Vorgang verdeutlichen.

Beispiel: Der Arm soll ganz langsam von links nach rechts schwenken, dabei aber vorher zupacken und beim Schwenken den Ellenbogen schütteln.

Beispiel als Ablaufdiagramm:



Beispiel als Wartebedingungen:

- ✓ Warte bis der Start-Befehl kommt, dann schließe die Hand
  - ✓ Warte bis die Hand geschlossen ist, dann schwenke zur Position
  - ✓ Warte bis die Hand geschlossen ist, dann Ellenbogen zur Position schwenken
    - ✓ Warte bis Ellenbogen Position erreicht hat, dann Ellenbogen zu Position

Beispiel als if-Abfrage:

1. if(Start); → Handschließen
2. if(Hand geschlossen); → schwenken & Ellenbogen rauf
3. if(Hand geschlossen); → Ellenbogen runter
4. if(Ellenbogen an Position); → Ellenbogen rauf

Daraus ist abzuleiten, dass die Steuerplatine nur auf die Erfüllung einer oder mehrerer Wartebedingungen wartet, während andere Motoren weiterhin ihre Befehle ausführen. Das erlaubt den Nutzern mehr Flexibilität bei der Programmierung von Bewegungen.

**Quellcode zur Hauptplatine im Anhang: CRAL\_Main\_Atmega8\_Final**

## 4 Motorsteuerung

### 4.1 Zusammenfassung der Aufgabe

- ➔ Die Hauptaufgabe der Steuerplatine besteht darin, die ankommenden Daten entgegen zu nehmen und die ordnungsgemäße Arbeit des Motors und der anderen Komponenten zu überwachen.
- ➔ Eine weitere Aufgabe ist es, die ankommenden Daten zu analysieren und anschließend die auszuführende Bewegung zu ermitteln, dabei mögliche Schrittfehler zu erkennen und gegebenenfalls auszugleichen.
- ➔ Die Motoren werden stufenweise auf die gewünschte Maximalgeschwindigkeit beschleunigt. Dabei wird überwacht, ob die noch zurückzulegende Strecke eine ausreichende Länge besitzt, um die Geschwindigkeit auch wieder negativ zu beschleunigen. Gegebenenfalls wird nur auf eine Geschwindigkeit beschleunigt, welche auch wieder abgebremst werden kann.
- ➔ Die Überwachung der Temperaturen von IC und dem Motor wird über eine AD-Wandlung, um gegebenenfalls den maximalen Strom zu begrenzen.
- ➔ Ausgabe von Rückgabewerten und Statusmeldungen sowie Fehlermeldungen
- ➔ Stufenweise Strombegrenzung
- ➔ Analoge Feststellung der aktuellen Position über ein Poti

#### Einfache Anpassungsmöglichkeit über die Einbindung von Parametern

```
//##### Parameter

#define beschleunigungs_facktor3 //1 ist das schnellste darf nicht 0 sein!
#define min_speed123 //200 Schritte pro sekunde --> 0,3*min_speed -->
//umdrehungen pro min (max ist 900; min 65)
#define max_range_m1360 //in Grad
#define max_fehler175 //größter zulässiger Fehler
#define CPU8000000 //CPU frequenz
#define Motor1 //Motor
#define Motor_anzahl5 //Anzahl der vorhandenen Motoren
#define Befehlsgroesse4 //Anzahl der zu senden/empfangenden char
```

## 4.2 Beschreibung der Motorsteuerungssoftware

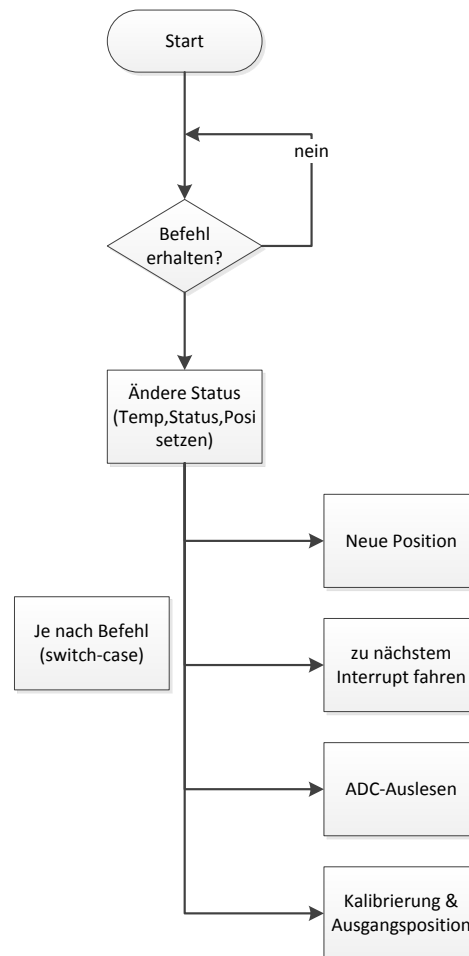


Abbildung 12: Ablaufplan Hauptschleife Motorsteuerung

Nach dem Anlegen der Betriebsspannung befindet die Motorsteuerung sich in einem „Idle-Modus“. Es wird ständig das rxBuffer überprüft, ob neue Befehle eingegangen sind und im txBuffer wird der Status, sprich Temperaturen und eventuelle Fehler, aktualisiert. Wird ein neuer Befehl empfangen, wird dieser entsprechend einer Switch-Case-Tabelle abgearbeitet:

- Case 0x01 → Zurücksetzen: alle Statusmeldungen werden gelöscht, um den Motor im Fehlerfall zurückzusetzen
- Case 0x02 → Neue Position: Normaler Fahrbefehl um eine neue Position einzunehmen
  - wird durch die Funktion *bewegung (PositionHigh, PositionLow, MaxSpeed)* realisiert
    - Ablaufplan siehe Seite 23
    - Fortsetzung siehe Seite 24

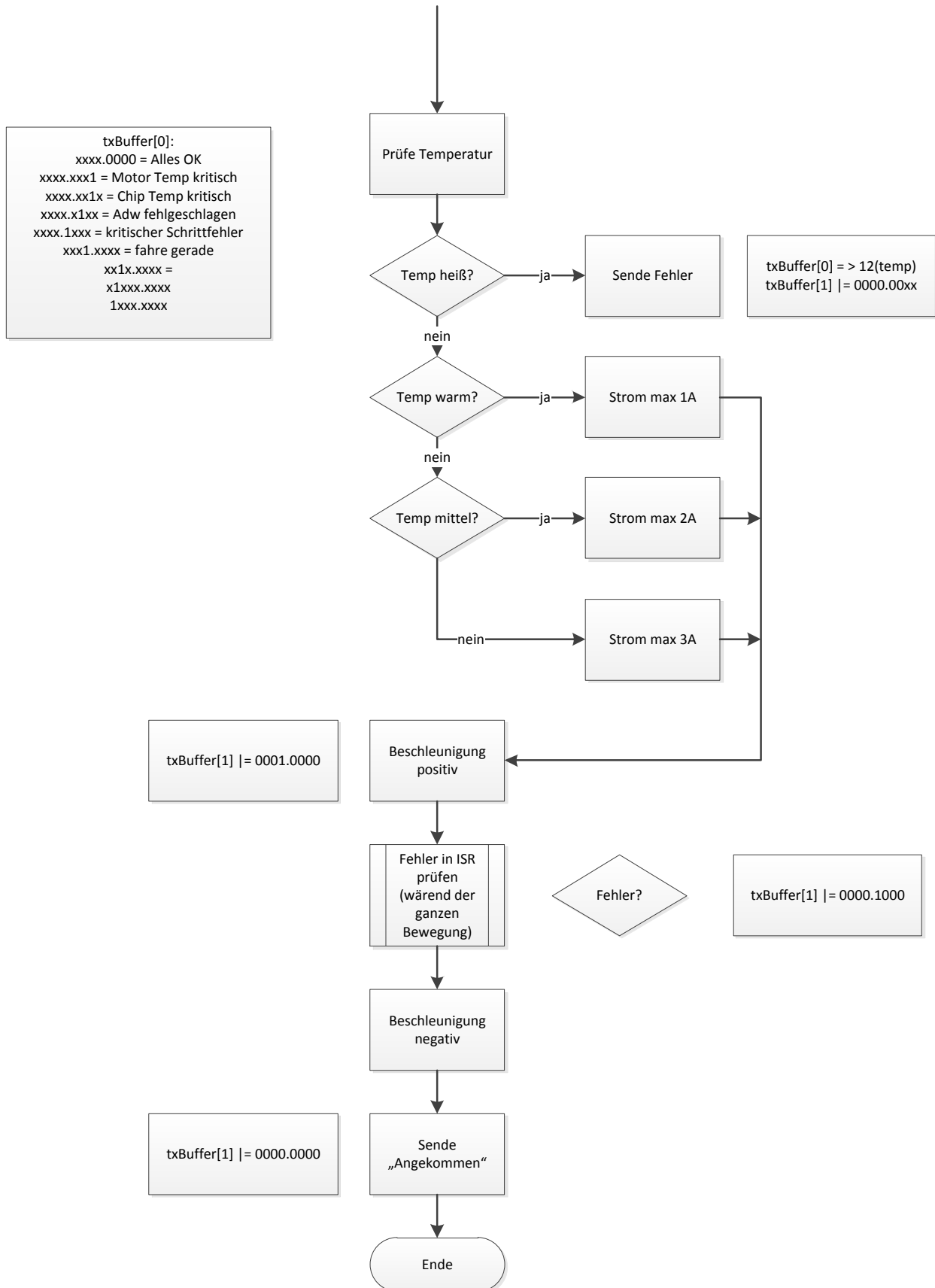


Abbildung 13: Ablaufplan „Neue Position“

### Beschreibung der Bewegungsfunktion:

Vor dem Starten der Bewegung wird die Temperatur überprüft und je nach aktueller Temperatur der Maximalstrom eingestellt (siehe Kapitel 4.3.1).

Nun beginnt die Beschleunigung. Diese ist abhängig vom Inkrementalgeber. Je nach Einstellung wird beim Auslösen des nächsten Inkrementinterrupts die Geschwindigkeit erhöht. Allerdings nur dann, wenn der Weg zum Entschleunigen ausreicht und die Maximalgeschwindigkeit noch nicht erreicht ist.

Auf die Beschleunigungsphase folgt die Geschwindigkeit-halten-Phase, bis der Bremsvorgang gestartet werden muss.

Im Bremsvorgang wird der inverse Vorgang ausgeführt, der beim Beschleunigen benutzt wurde. Ist die Zielposition erreicht, wird der Status wieder auf „Bereit“ gesetzt.

- Case 0x03 -> zum nächsten Interrupt fahren
  - Der Motor fährt mit langsamster Geschwindigkeit, bis der nächste Inkrementalgeber erreicht wurde
- Case 0x04 -> ADC auslesen: Zum Auslesen der analogen Position
  - Wird durch die Funktion *readADC (Channel)* erreicht
- Case 0x05 -> Kalibrierung klein: Macht erst eine ADC-Wandlung um die ungefähre Position zu bestimmen und fährt daraufhin zum nächsten Interrupt und hat nun wieder die genaue Position festgestellt (Case 0x04 & Case 0x03)

- Case 0x06 -> Kalibrierung groß: Führt zusätzlich zur Kalibrierung klein auch zu einer definierten Ausgangsposition

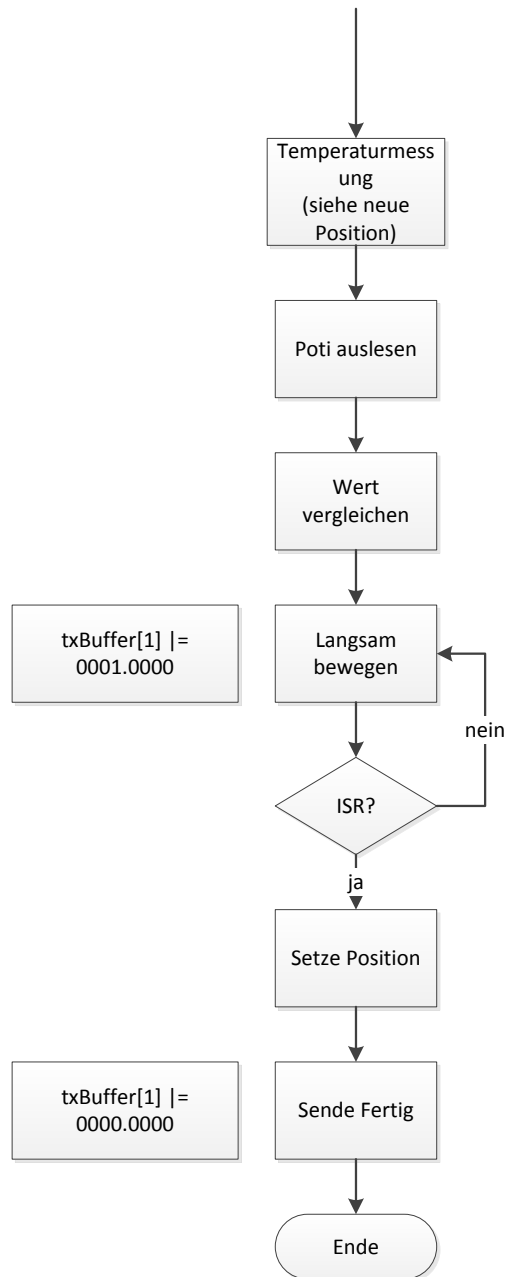
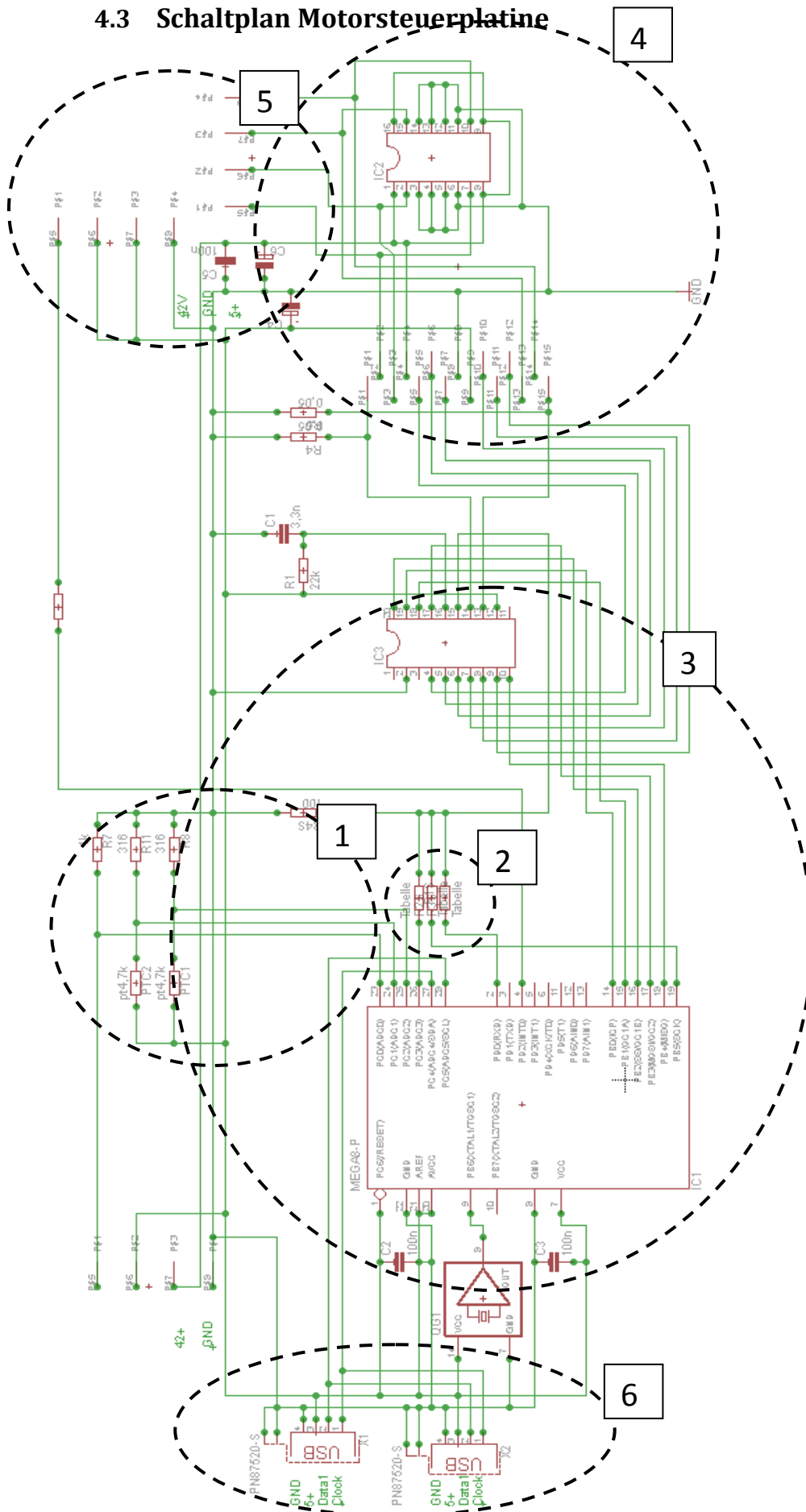


Abbildung 14: Ablaufplan Kalibrierung

Quelltext für die Motorsteuerung siehe Anhang

### 4.3 Schaltplan Motorsteuerplatine



1. Spannungsteiler
2. Strombegrenzung
3. Steuerung
4. Motortreiber
5. Anschlussklemmen
6. TWI & Clock

Abbildung 15: Schaltplan Steuerplatine

### 4.3.1 Spannungsteiler und Strombegrenzung

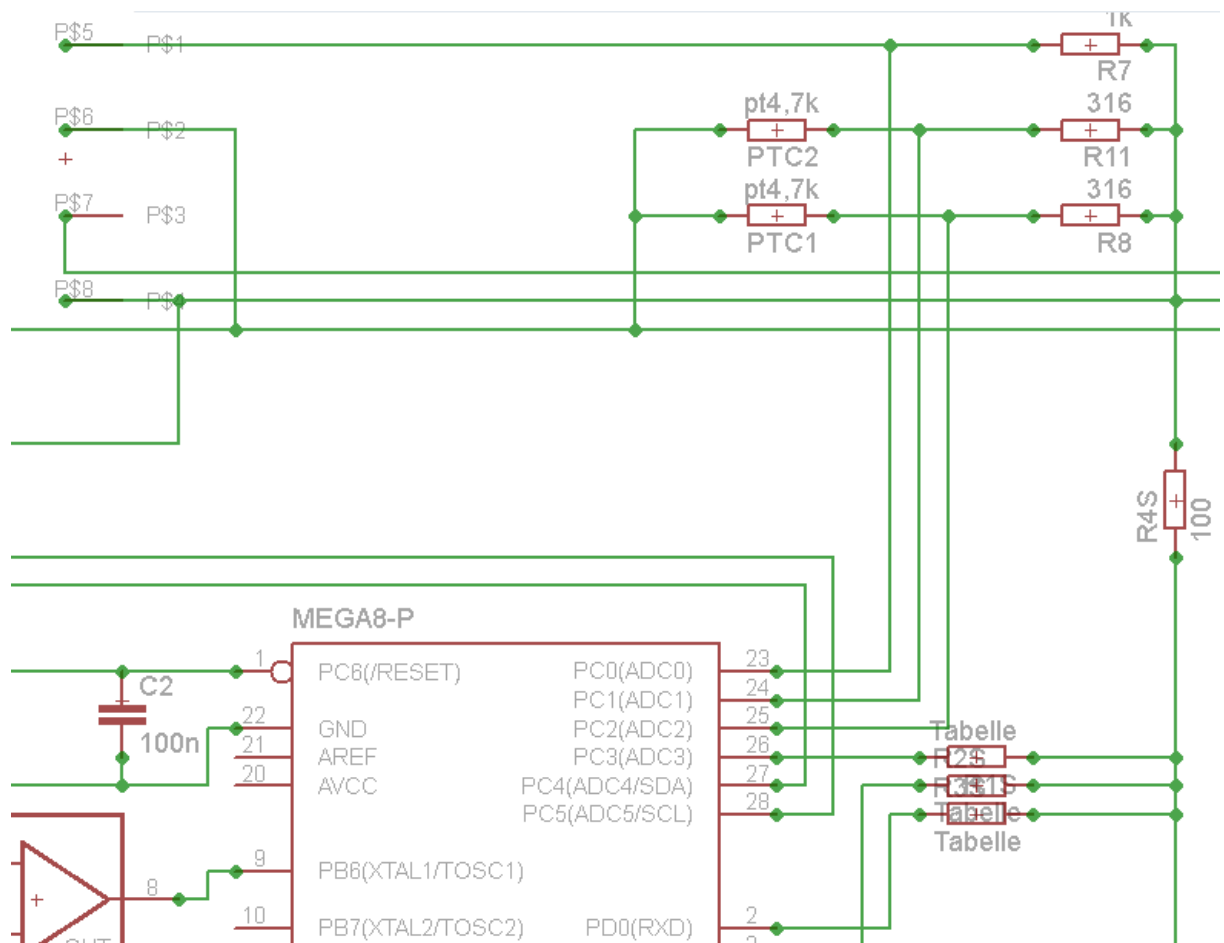


Abbildung 16: Schaltplanausschnitt Spannungsteiler NTC & Stromversorgung

➔ Verhalten des NTCs: B400\_NTC-02\_Serie im Anhang

Werte:

- Betriebstemperatur bis 50°C
- Erhöhte Temperatur zwischen 50°C und 70 °C
- Kritische Temperatur ab 70°C
- 80°C entsprechen der maximalen Gehäusetemperatur des L297

AD-Werte nach Wandlung:  $U_{\text{reff}}/\text{Auflösung} = 5\text{V}/255\text{Werte} = 0,196 \text{ V/Wert}$ . Verwendet werden nur die ersten 4 Bit. Daraus folgt, dass die Auflösung höchstens 16 annehmen kann:  $5\text{V}/16\text{Werte} = 0,3125\text{V/Wert}$  ist.

Im Zusammenhang mit dem Datenblatt ergeben sich folgende Werte für unsere Schwellen:

$$50^{\circ}\text{C} = 1,694\text{k}\Omega; 70^{\circ}\text{C} = 0,824\text{ k}\Omega \Rightarrow$$

$$U_{AD} = U_{CC}/R_g * R_v = 5\text{V}/(3160\Omega + 16940\Omega) * 3160\Omega$$

$$= 0,786\text{V} \Rightarrow \text{Wert} = 0,786\text{V}/0,3125\text{V} = 2,5$$

$$U_{AD} = U_{CC}/R_g * R_v = 5\text{V}/(316\Omega + 824\Omega) * 3160\Omega$$

$$= 1,386\text{V} \Rightarrow \text{Wert} = 1,386\text{V}/0,3125\text{V} = 4,4$$

Wir entschieden uns, die Nachkommastellen abzuschneiden, um einen gewissen Sicherheitsabstand zu den  $T_{\max}$ -Werten zubekommen.

Strombegrenzung:

Die Strombegrenzung wird durch eine Digital-/Analogwandlung mit drei Stufen realisiert. Die Stufen sind identisch mit den Temperaturschwellenwerten.

Normalbetrieb bei ca.  $<50^{\circ}\text{C}$

Eingeschränkter Betrieb zwischen ca.  $50^{\circ}\text{C}$  und  $70^{\circ}\text{C}$

Störbetrieb bei ca.  $>70^{\circ}\text{C}$

Daraus folgt, dass bis zu einem Wert von  $\leq 2$  (Normalbetrieb) 100% Leistung abgerufen werden kann. Ab dem Wert 3 wird nur noch 66% Strom geliefert. Und ab einem Wert von 4 wird der Strom auf maximal 33% begrenzt. Ab dem Wert 5 geht die Motorsteuerung in den Störbetrieb. Das bedeutet, dass der Motor inaktiv und somit ohne Haltemoment ist.

Die Strombegrenzung wird durch das Hinzuschalten oder Abschalten von Widerständen in einer Parallelschaltung realisiert. Der kleinste Widerstandswert ermöglicht den größten Strom. Denn die Spannung über  $R_{45}$  ist die Referenzspannung für den Komparator im L297, der den aufgenommenen Strom des Motors über einen Shunt indirekt misst. Die eigentliche Strombegrenzung erfolgt über das periodische Takten der Spannung, um die Leistungsaufnahme zu drosseln. Der Strom, der sich durch die Trägheit der Induktivität einstellt, ist der Effektivwert des zu dem Zeitpunkt maximal erlaubten Stroms. Die Taktungsfrequenz wird über den Schwingkreis von C1 und R1 bestimmt.

### 4.3.2 Steuerung

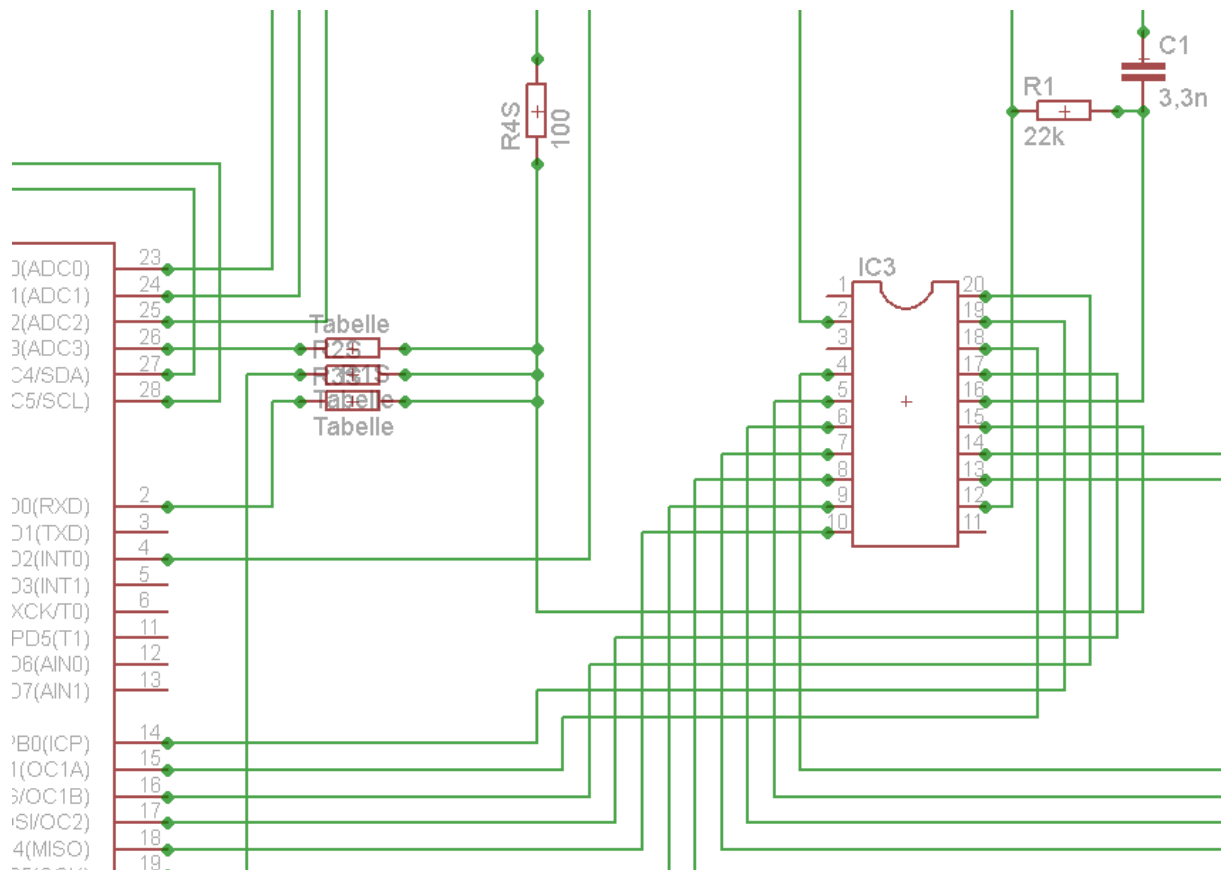


Abbildung 17: Schaltplanausschnitt Steuerung

Die Steuerung wurde hier ebenfalls mit einem Atmega 8-16PU realisiert. Der Microcontroller generiert intern einen variablen Takt, der von der eingestellten Geschwindigkeit abhängt. Der Timer 1 lässt dann OCA1 (Pin 16) in CTC-Modus toggeln. Das IC3 ist ein L297. Er regelt die korrekte Ausgabe der Schrittsignale an den Treiber (ein L298). Bei einer steigenden Flanke vom OCA1 wird ein neuer Schritt generiert. Darüberhinaus muss man über ein Bitmuster einen Half- oder Fullstepmodus, ein Home-Signal (dient als Haltesignal des Schrittmotors) und die Richtung auswählen. Der L297 hat zusätzlich die Aufgabe, den RMS-Strom zu maximieren, die Funktionsweise ist in Kapitel 4.3.1 näher erläutert.

Eine weitere wichtige Aufgabe ist die Schrittfehler-Erkennung und Korrektur. Diese wird mit Hilfe einer Lichtschranke und einer Schlitzscheibe realisiert. Der Ablauf ist folgender: Wenn der Motor sich dreht, dreht sich die Scheibe in gleicher Geschwindigkeit und lässt alle 50 Schritte (Lochabstand) den Fototransistor leitend werden. So wird an PIN 4 der Interrupt INTO ausgelöst und im Programm wird eine ISR

verarbeitet. Somit verfügt man über eine hardwareseitige Schrittfehler-Erkennung und kann nun die Fehler intern überwachen.

Die analoge Position wird durch das Potentiometer am Gelenk des Armgliedes bestimmt. Nun kann man über die Funktion „ACD-Auslesen“ die Werte zu jedem Inkrementalgeberstand auslesen und in eine dafür angelegte Tabelle eintragen. Nachdem diese Werte bekannt sind, kann nun an jeder möglichen Position über eine erneute AD-Wandlung die Nähe zum nächstgelegenen Interrupt festgestellt werden. Mit einer darauffolgenden Bewegung zum ermittelten Loch der Scheibe kann die exakte Position bestimmt werden.

### 4.3.3 Motortreiber

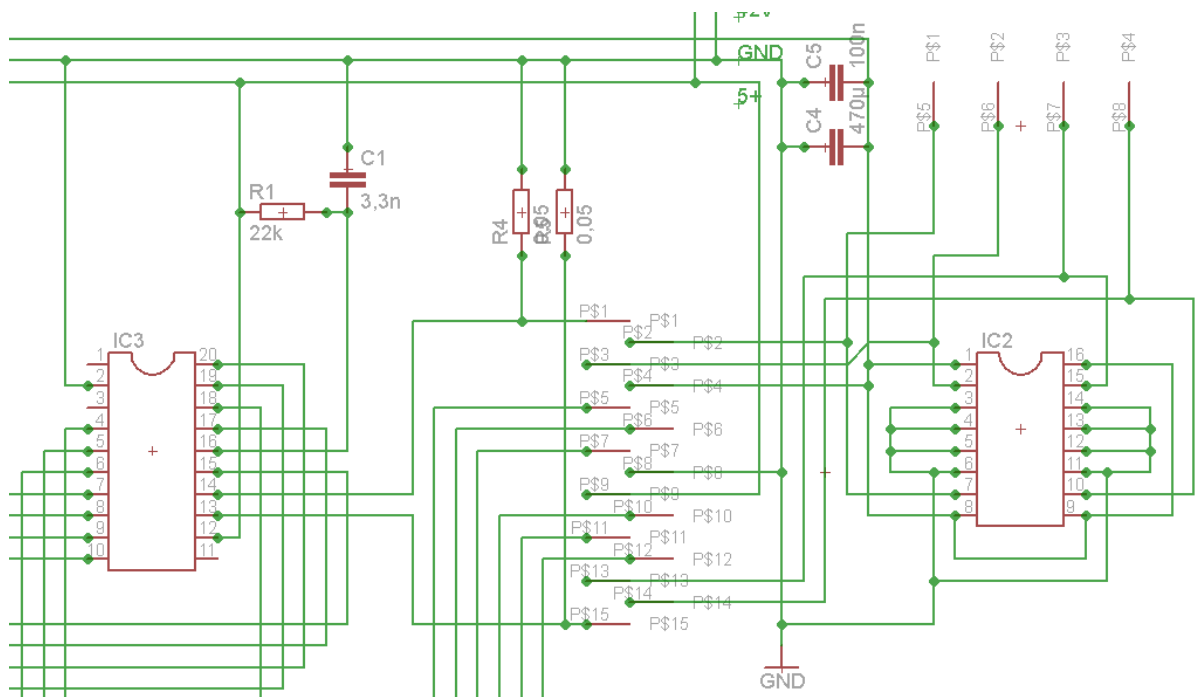


Abbildung 18: Schaltplanausschnitt Motoransteuerung

Die Motoren werden mit dem Leistungstreiber L298 (P\$1-P\$15) angesteuert. Die Steuersignale generiert der L297 (IC3) automatisch in richtiger Reihenfolge und anschließend werden sie im Treiber auf 20V DC verstärkt an den Schrittmotor übertragen. Das IC2 ist ein L6210, in dem sich 8 Schottky-Dioden befinden, die als Freilaufdioden geschaltet sind. Dies ist wichtig, da das Magnetfeld bei einer flüssigen und vor allem schnellen Bewegung zeitnah abgebaut werden muss und damit die Rückspannungen, die beim schnellen Abschalten entstehen, kurzgeschlossen werden.

```

/* PIN BELEGUNGEN & Konfiguration vom L297:
  Halfstep aktiv bei (PB0 = 1),
  pin 14 µC & pin 19 L229Clock (PB1),
  pin 15 µC & Pin 18 L297Reset muss aktiv sein(PB2 = 1);
  pin 16 µC & Pin 20 L297Richtung Wählen für Rechts (PB3 = 1),
  Pin 17 µC & L297Enable chip aktiv bei sonst geht er in die Home-Position (PB4 =
  1); pin 18 µC & Pin 10 L297
*/
PORTB|=(1<<PB2)|(1<<PB3)|(1<<PB4)|(1<<PB0);           //=> 11101
//PORTB &= ~(1<<PB0);
richtung=1;//rechts
//#####Timer Einstellungen
TCCR1B|=(1<<CS10);           //timer starten --> Clk i/o =1 kein Teiler
OCR1A=CPU/(2*min_speed);    //Timer mit dem Gröstenwert laden
  
```

Die beiden Shunt-Widerstände R4 und R5 dienen zur Strombegrenzung und wurden mit Widerstandsdraht realisiert. Die Länge der Drähte ist für die Strombegrenzung entscheidend. Die Werte der Widerstände beziehen sich dabei auf die Innenwiderstände der Motoren, dies verhindert einen zu großen Leistungsabfall an den Widerständen. Die Länge (Widerstandswert) muss dabei um ein vielfaches kleiner sein als der Innenwiderstand. So ergibt sich für die großen Schrittmotoren eine Länge von 2cm und für die kleinen eine Länge von 5 cm, die zugehörigen Widerstände des Spannungsteilers können folgender Tabelle entnommen werden:

	Motor1	Motor2	Motor3	Motor4	Motor5
R1s	100	100	100	100	100
R2s	10k	10k	30k	30k	30k
R3s	10k	10k	30k	30k	30k
R4s	10k	10k	30k	30k	30k

#### 4.3.4 Klemmen

An der unteren Klemmleiste wird der Motor angeschlossen. Der Inkrementalgeber muss zusätzlich zwischen Masse und dem Ausgang des Fototransistors mit einem 4,7kΩ Widerstand beschaltet werden. Ferner muss zwischen der Fotodiode und der Masse zusätzlich ein 50Ω Widerstand eingefügt werden.

Die genaue Belegung der Klemmen kann dem Schaltplan entnommen werden, zu beachten ist jedoch der Wicklungssinn der Schrittmotoren. Diese sind im Datenblatt der Motoren zu finden.

#### 4.3.5 TWI & Clock

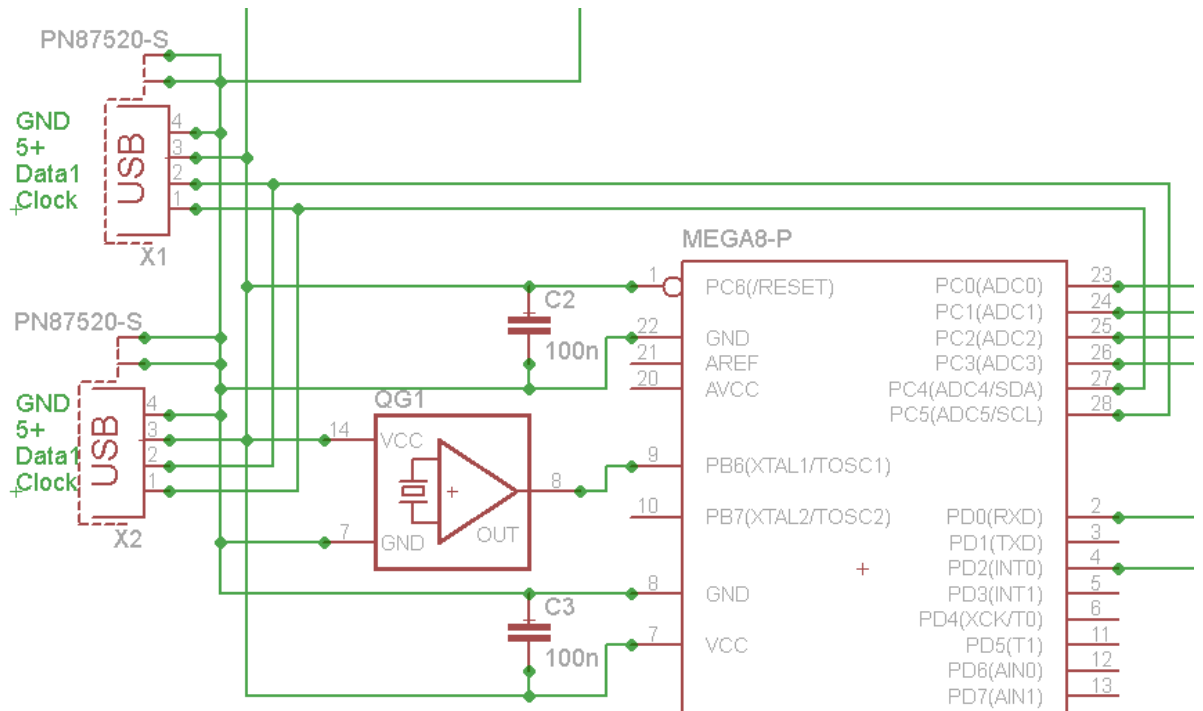


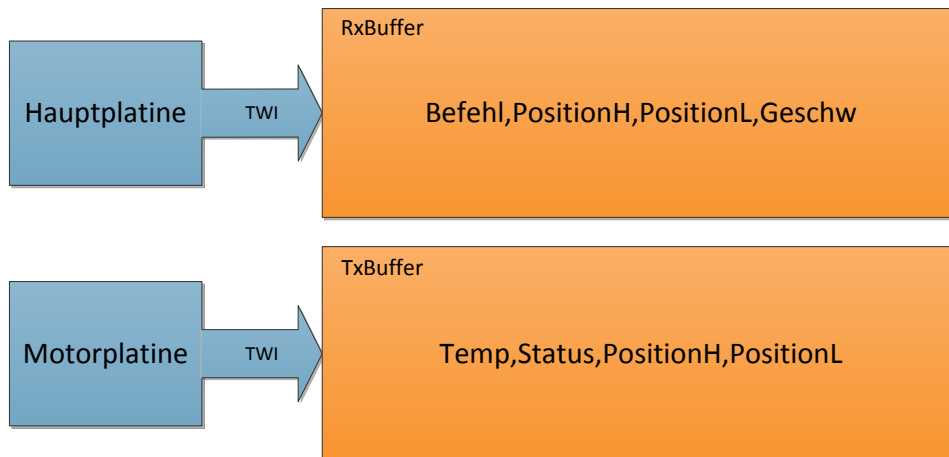
Abbildung 19: Schaltplanausschnitt TWI - Anschluss und Clockgenerierung

Die ankommenden Signale über das TWI werden durchgeschliffen und so an die nachfolgende Steuereinheit weitergeleitet. Im Programm gibt es dann zwei Tabellen, über die kommuniziert werden kann: Eine dient zum Empfangen der Daten von der Hauptplatine und die andere zum Senden von Informationen und Statusmeldungen. Diese Programmabschnitte der TWI sind zum größten Teil der Webseite Mircokontroller.net entnommen und wurden unseren Zwecken angepasst.

Zu sehen ist auch der Crystal-Oszillator (QG1) und Entstör-Kondensatoren. Die Taktfrequenz wird also extern mit 8Mhz generiert.

## 5 Kommunikation

### 5.1 TWI



Die Kommunikation in beide Richtungen wird in zwei Char Arrays (vier Felder pro Motor) gepuffert.

Das RxBuffer enthält die zu den Motoren gesandten Daten (receive).

Das TxBuffer wiederum enthält die zur Hauptplatine gesandten Statusmeldungen (transmit)

Die möglichen Nachrichten können der Abb. 20 unten entnommen werden.

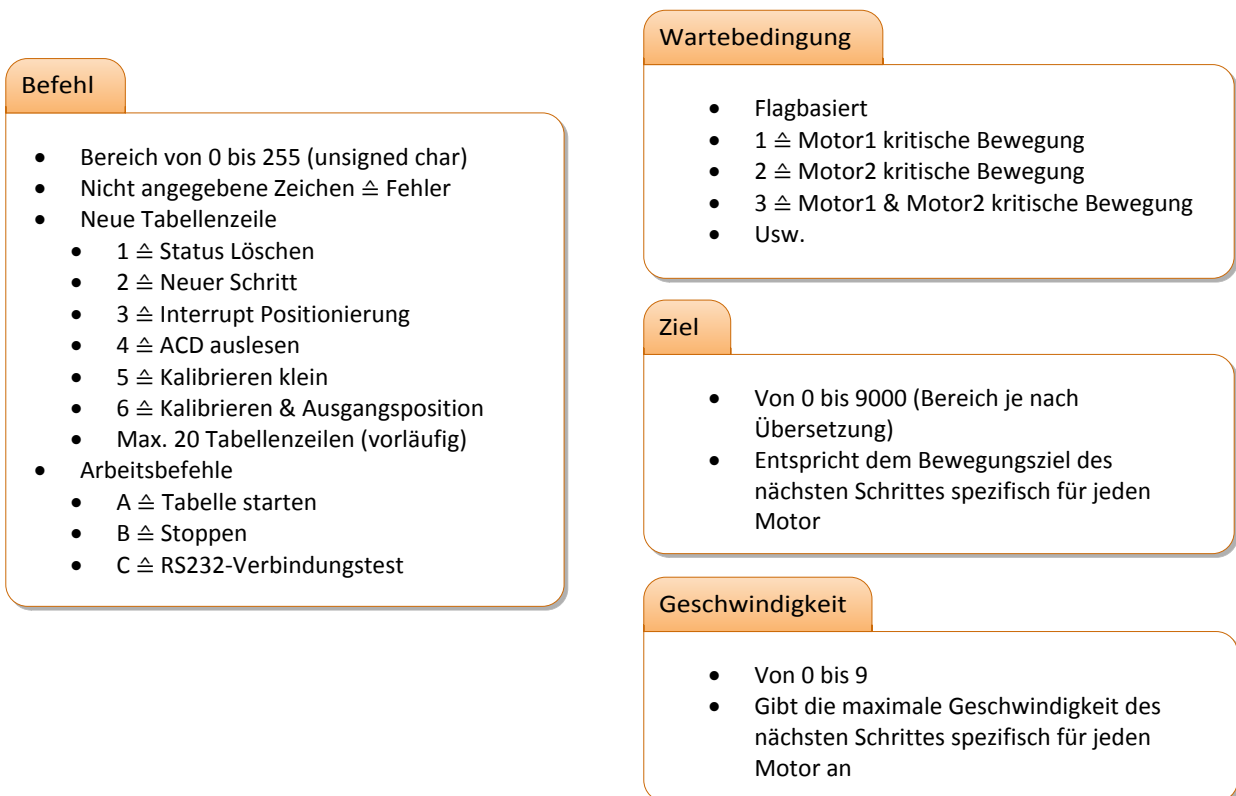
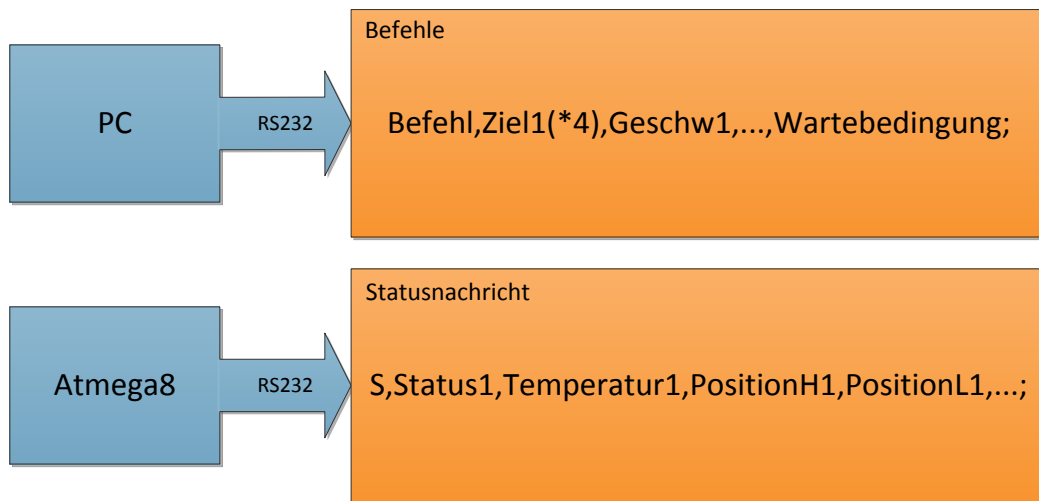


Abbildung 20: Zahlenbereiche

## 5.2 RS232



Für die RS232-Kommunikation haben wir folgende Regeln festgelegt:

- Das erste Zeichen einer Nachricht legt immer den Befehl fest (siehe Abb. 20).
- Die Nachrichtenelemente haben eine feste Länge und somit sind keine Trennungszeichen notwendig.
- Das Ende einer Nachricht wird immer durch ein Semikolon angezeigt.
- „Ziel“, „Geschwindigkeit“, „Status“, „Temperatur“ und „Position“ sind jeweils für alle Motoren zu übertragen. Bereiche für nicht angeschlossene Motoren werden mit Nullen aufgefüllt. (Status für nicht angeschlossene Motoren am besten auf Beschäftigt 0x10 setzen.)
- Das Feld „Geschwindigkeit“ hat während der Übertragung Werte von 0 bis 9, gemeint ist 1 bis 10.  
Dieses Offset wird in der Motorsteuerung wieder addiert.
- Gesendet wird immer ein Befehl pro RS232-Nachricht.

## 6 Windowsoberfläche

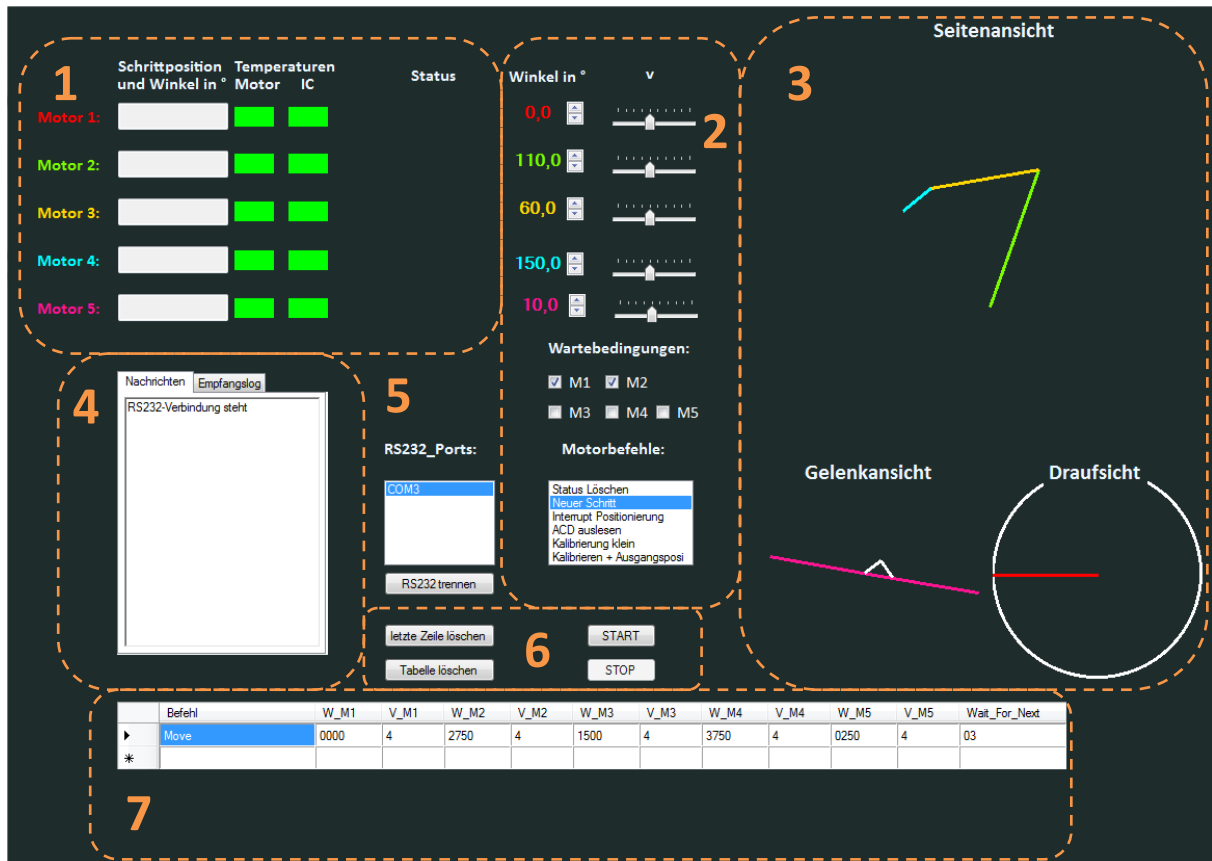


Abbildung 21: Windows Bedienelemente

(1) Statusanzeige

Dient der Anzeige des momentanen Zustands, geordnet nach Motoren.

(2) Befehlseingabe

Hier wird oben der nächste zu speichernde Schritt eingestellt, welcher dann über den Motorbefehl „Neuer Schritt“ an die Hauptplatine gesandt wird.

(3) Befehlsdarstellung

Dient der Darstellung des momentan unter (2) eingestellten Befehls.

(4) Verbindungslog

- Hier werden Verbindungsinformationen angezeigt.
- Der Reiter „Empfangslog“ dient hierbei dem Debugging und enthält die kompletten empfangenen Daten.

(5) RS232 Ports

Zeigt die aktuell möglichen Serial-Ports an:

- Ein Doppelklick versucht einen Verbindungsaufbau.

(6) Tabellenbefehle

Befehle, die direkten Einfluss auf die Tabelle haben.

(7) Gespeicherter Bewegungsablauf

Gespeicherte Abfolge aller unter (2) eingegebenen Befehle.

## 6.1 Statusanzeige

In der Statusanzeige wird detailliert der momentane Status aller angeschlossener Motoren angezeigt. Zur genauen Aufschlüsselung der übertragenen Nachrichten von der Hauptplatine zur Oberfläche bitte in Kapitel 5 nachschlagen.

Die Anzeige ist in drei Bereiche unterteilt:

- Position:
  - o Zeigt den momentanen Schrittwert des Motors an und den daraus berechneten Winkel des zugehörigen Elements
  - o Zugehöriger Quelltext:
 

```
//Umwandeln der Position von HEX-String zu int
int posiM = Convert.ToInt32(message->Substring(5,4),16);
//Ausgabe
aWinkelM1->Text = System.Convert.ToString(posiM);
aWinkelM1->Text += " ";
//Umrechnung der Position in einen Winkel je nach Übersetzung
posiM = (posiM*360)/(uebersetzung * 200);
//Ausgabe
aWinkelM1->Text += System.Convert.ToString(posiM);
```
- Temperaturanzeige:
  - o Die Temperatur wird in vier Bereichen farblich angezeigt:
    - <20C° Dunkelgrün
    - 20C°- 50C° Grün
    - 50C°-70C° Orange
    - >70C° Rot
  - o Zugehöriger Quelltext:
 

```
//Temperaturen
if(* (cMessage+3) <= '1')
    temp1_IC->BackColor = Color::DarkGreen;
elseif(* (cMessage+3) <= '2')
    temp1_IC->BackColor = Color::Lime;
elseif(* (cMessage+3) <= '3')
    temp1_IC->BackColor = Color::Orange;
else
    temp1_IC->BackColor = Color::Red;
if(* (cMessage+2) <= '1')
    temp1_Motor->BackColor = Color::DarkGreen;
elseif(* (cMessage+2) <= '2')
    temp1_Motor->BackColor = Color::Lime;
elseif(* (cMessage+2) <= '3')
    temp1_Motor->BackColor = Color::Orange;
else
    temp1_Motor->BackColor = Color::Red;
```

    - Temperaturbereiche werden auf der Motortreiberplatine vorbereitet
- Statusanzeige
  - o Mögliche Zustände: siehe Kapitel 5
  - o Zugehöriger Quelltext:
 

```
//Senden des Statusbereichs der Message an die StatusTextBox
String^ statusM;
statusM = System.Convert.ToString(message[1]) +
System.Convert.ToString(message[2]);
StatusTextBoxM1->Text = statusM;
```

```
//Senden des Statusbereichs der Message an die StatusTextBox
private: System::Void StatusTextBoxM1_TextChanged(System::Object^ sender,
System::EventArgs^ e)
{
    unsignedchar mStatus;
    //Status einlesen -> Wandeln von Ascii in hex
    if(StatusTextBoxM1->Text[0]<='9')//0-9
        mStatus = (StatusTextBoxM1->Text[0]-48)*16;
    else//A-F
        mStatus = (StatusTextBoxM1->Text[0]-87)*16;
    if(StatusTextBoxM1->Text[1]<='9')
        mStatus += (StatusTextBoxM1->Text[1]-48);
    else
        mStatus += (StatusTextBoxM1->Text[1]-87);

    StatusTextBoxM1->Text = "";//Feld leeren
    //Status verarbeiten
    if(!mStatus)
        StatusTextBoxM1->Text = "Motor bereit\n";
    if(mStatus & 0x01)
        StatusTextBoxM1->Text += "Motor zu heiß\n";
    if(mStatus & 0x02)
        StatusTextBoxM1->Text += "IC zu heiß\n";
    if(mStatus & 0x04)
        StatusTextBoxM1->Text += "ADW fehlgeschlagen\n";
    if(mStatus & 0x08)
        StatusTextBoxM1->Text += "kritischer Schrittfehler\n";
    if(mStatus & 0x10)
        StatusTextBoxM1->Text += "Motor beschaeftigt\n";
}
```

## 6.2 Befehlseingabe

- Zielwinkeleinstellung
  - o Stellt den Winkel für den nächsten zuzusendenden Schritt ein
  - o Auflösung: 0,1 – das entspricht bei unserer Übersetzung von 1:45 einer Auflösung von 2 Schritten
  - o Zugehöriger Quelltext:
 

```
//Umrechnung Winkel -> Schritte
tWinkelM1->Value*uebersetzung*200)/360
```
- Geschwindigkeitsregler
  - o Einstellung der möglichen Höchstgeschwindigkeit
  - o Bereich von 0-9
    - Zur Einsparung einer Stelle wird die Zahl auf der Motorplatine inkrementiert
- Wartebedingungen
  - o Ausgewählte Motoren müssen ihren Befehl vor dem Starten des nächsten Schritts zwingend beendet haben

- Zugehöriger Quelltext:
 

```
//Erstellen des Wait-Feldes
int iWait = 0;
if(checkBoxM1->Checked)
    iWait +=1;
if(checkBoxM2->Checked)
    iWait +=2;
if(checkBoxM3->Checked)
    iWait +=4;
if(checkBoxM4->Checked)
    iWait +=8;
if(checkBoxM5->Checked)
    iWait +=16;
```
- Motorbefehle
  - Doppelklick sendet den ausgewählten Befehl an den Motor, wenn eine RS-232-Verbindung aufgebaut wurde und speichert diesen in der Tabelle
  - Zugehöriger Quelltext:
 

```
if(listBox1->GetSelected(1))//MOVE befehl
{
    //Umwandlung der Winkelfelder in Schrittzahlen, immer 4 Stellig
    int iZiel1 = System::Convert::ToInt32((tWinkelM1->Value*uebersetzung*200)/360);
    int iZiel2 = System::Convert::ToInt32((tWinkelM2->Value*uebersetzung*200)/360);
    int iZiel3 = System::Convert::ToInt32((tWinkelM3->Value*uebersetzung*200)/360);
    int iZiel4 = System::Convert::ToInt32((tWinkelM4->Value*uebersetzung*200)/360);
    int iZiel5 = System::Convert::ToInt32((tWinkelM5->Value*uebersetzung*200)/360);

    dataGridView1->Rows->Add("Move",2,iZiel1.ToString("0000"),
tGeschwM1->Value-1,iZiel2.ToString("0000"),tGeschwM2->Value-1,
iZiel3.ToString("0000"),tGeschwM3->Value-1,
iZiel4.ToString("0000"),tGeschwM4->Value-1,
iZiel5.ToString("0000"),tGeschwM5->Value-1,
iWait.ToString("00"));
}
else//Alle anderen Befehle
    dataGridView1->Rows->Add(listBox1->SelectedItem,
listBox1->SelectedIndex +
1,"0000","0","0000","0","0000","0","0000","0","0000","0",iWait
.ToString("00"));
for(int i=1;i<dataGridView1->ColumnCount;i++)
{
    message += dataGridView1->Rows[dataGridView1->
Rows->Count - 2]->Cells[i]->Value;
}
message+=" ";
serialPort1->WriteLine(message);
```

### 6.3 Befehlsdarstellung

- Arm-Klasse zur einfachen Darstellung der Abhängigkeiten zwischen den zu zeichnenden Elementen

- o Header:

```
#pragma once
#include<cmath>
#define PI 3.14159
refclass Arm
{
private:
int x1,y1,x2,y2,winkel,laenge;
public:
Arm(int laenge, int x1, int y1, int
winkel):laenge(laenge),x1(x1),y1(y1),winkel(winkel)
{
x2=x1-(laenge*cos(winkel*PI/180));
y2=y1-(laenge*sin(winkel*PI/180));
}
int get_laenge(){return laenge;}
int get_x1(){return x1;}
int get_y1(){return y1;}
int get_x2(){return x2;}
int get_y2(){return y2;}
int get_winkel(){return winkel;}
};
```

- Seitenansicht

- o Motoren 2-4
- o Zugehöriger Quelltext:

```
void pictureBox1_Paint( Object^ /*sender*/,
System::Windows::Forms::PaintEventArgs^ e )
{
usingnamespace std;
// Create a local version of the graphics object for the
PictureBox.
Pen^ goldPen = gcnwPen( Color::Gold,3.0f );
Pen^ lawnGreenPen = gcnwPen( Color::LawnGreen,3.0f );
Pen^ cyanPen = gcnwPen( Color::Cyan,3.0f );
//int::Parse() string to int
Arm arm1(140,pictureBox1->Width/2,pictureBox1->Height -
pictureBox1->Height/3,(int)tWinkelM2->Value);
Arm arm2(105,arm1.get_x2(),arm1.get_y2(),(int)tWinkelM3-
>Value+(int)tWinkelM2->Value-180);
Arm arm3(35,arm2.get_x2(),arm2.get_y2(),(int)tWinkelM4-
>Value+(int)tWinkelM2->Value+(int)tWinkelM3->Value);
Graphics^ g = e->Graphics;
// Draw a line in the PictureBox.
g->DrawLine(lawnGreenPen,
arm1.get_x2(),arm1.get_y2(),arm1.get_x1(),arm1.get_y1());
g->DrawLine(goldPen,
arm2.get_x2(),arm2.get_y2(),arm2.get_x1(),arm2.get_y1());
g->DrawLine(cyanPen,
arm3.get_x2(),arm3.get_y2(),arm3.get_x1(),arm3.get_y1());
}
```

- Draufsicht

- Motor 1

- Zugehöriger Quelltext:

```
void pictureBox2_Paint( Object^ /*sender*/,
System::Windows::Forms::PaintEventArgs^ e )
{
    usingnamespace std;
    // Create a local version of the graphics object for the
    PictureBox.
    Pen^ whitePen = gcnewPen( Color::White,3.0f );
    Pen^ redPen = gcnewPen( Color::Red,3.0f );
    Arm arm1(pictureBox2->Width/2,pictureBox2->Width/2,pictureBox2->
    Width/2,(int)tWinkelM1->Value);
    Graphics^ g = e->Graphics;
    // Draw a line in the PictureBox
    g->DrawEllipse(whitePen,0,0,200,200);
    g->DrawLine(redPen,
    arm1.get_x2(),arm1.get_y2(),arm1.get_x1(),arm1.get_y1());
}
```

- Gelenkansicht

- Motor 5

- Zugehöriger Quelltext:

```
void pictureBox3_Paint( Object^ /*sender*/,
System::Windows::Forms::PaintEventArgs^ e )
{
    usingnamespace std;
    // Create a local version of the graphics object for the
    PictureBox.
    Pen^ whitePen = gcnewPen( Color::White,3.0f );
    Pen^ deepPinkPen = gcnewPen( Color::DeepPink,3.0f );
    Arm arm1(pictureBox3->Width/2,pictureBox3->Width/2,pictureBox3->
    Width/2,(int)tWinkelM5->Value);
    Arm arm2(pictureBox3->Width/2,pictureBox3->Width/2,pictureBox3->
    Width/2,(int)tWinkelM5->Value + 180);
    Arm arm3(20,arm1.get_laenge()-10*cos((double)tWinkelM5->
    Value*PI/180),arm1.get_laenge()-10*sin((double)tWinkelM5->
    Value*PI/180),arm1.get_winkel()+135);
    Arm arm4(20,arm3.get_x2(),arm3.get_y2(),arm3.get_winkel()+90);
    Graphics^ g = e->Graphics;
    // Draw a line in the PictureBox.
    g->DrawLine(deepPinkPen,
    arm1.get_x2(),arm1.get_y2(),arm1.get_x1(),arm1.get_y1());
    g->DrawLine(deepPinkPen,
    arm2.get_x2(),arm2.get_y2(),arm2.get_x1(),arm2.get_y1());
    g->DrawLine(whitePen,
    arm3.get_x2(),arm3.get_y2(),arm3.get_x1(),arm3.get_y1());
    g->DrawLine(whitePen,
    arm4.get_x2(),arm4.get_y2(),arm4.get_x1(),arm4.get_y1());
}
```

## 6.4 Verbindungslog

- Nachrichten
  - Zuständig für die Anzeige und Auswertung aller anderer Nachrichten als Statusanzeige
  - Zugehöriger Quelltext:
 

```
if(message == "C") //RS232-Verbindungstets
{
    textRS232->Text = "RS232-Verbindung steht";
    timer2->Enabled = false;
    RS232_buffer->Text = "#";//Ack Signal
elseif(message == "B")//Fahren beendet
{
    textRS232->Text += "\n Fahren beendet\n";
    if(dataGridView1->Rows->Count > 1)
        bStart ->Enabled = true;
}
else
{
    //Wandeln in ein char Array
    System::IntPtr ptr =
    System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi
    (message);
    char* cMessage = static_cast<char*>(ptr.ToPointer());
    if(*cMessage == 'D')
    {
        textRS232->Text += "\nSenden des ";
        textRS232->Text += System::Convert::ToString(*(cMessage+1)-
        48);
        textRS232->Text += System::Convert::ToString(*(cMessage+2)-
        48);
        textRS232->Text += " . Schritts beendet\n";
    }
}
}
```
- Empfangslog
  - Enthält jede empfangene Nachricht für das Debugging
  - Zugehöriger Quelltext:
 

```
richTextBox1->Text += message + "\n";
```

## 6.5 RS232-Ports

- COM-Ports
  - Auswahl des falschen COM-Ports führt möglicherweise zu unerwarteten Fehlern und, in Ausnahmefällen, zum Absturz des Programms
  - Doppelklick führt einen Verbindungsaufbau mit der Hauptplatine aus
    - Message = „C;“ wird als Antwort erwartet
    - Ist nach Ablauf von 5 Sekunden keine Antwort erhalten worden, wird der Versuch abgebrochen
  - Zugehöriger Quelltext:

```

//ConnectionTimer
private: System::Void timer2_Tick(System::Object^ sender,
System::EventArgs^ e)
{
  timer1->Enabled = false;
  serialPort1->Close();
  textRS232->Text = "Verbindung fehlgeschlagen";
  deleteRowButton->Enabled = false;
  deleteRowsButton->Enabled = false;
  timer2->Enabled = false;
}
//ConnectionBuildup
String^ message="C;";
if(!serialPort1->IsOpen)
{
  Try
  {
    serialPort1->PortName =
    System::Convert::ToString(listBoxRS232->SelectedItem);
    serialPort1->Open();
    timer1->Enabled = true;
    serialPort1->WriteLine(message);
    serialPort1->WriteLine(message);
    deleteRowButton->Enabled = false;
    deleteRowsButton->Enabled = false;
    timer2->Enabled = true; //Verbindungstest-Timer
  }
}

```

## 6.6 Tabellenbefehle

- Letzte Zeile löschen
  - Löscht die letzte Zeile aus der Tabelle und sendet einen entsprechenden Befehl an die Hauptplatine
  - Zugehöriger Quelltext:

```
if(dataGridView1->Rows->Count > 1)
{
    dataGridView1->Rows->RemoveAt(dataGridView1->Rows->Count-2);
    serialPort1->WriteLine("D;");
}
if(dataGridView1->Rows->Count == 1)
    bStart->Enabled = false;
```
- Tabelle löschen
  - Löscht die gesamte Tabelle in beiden Systemen
  - Zugehöriger Quelltext:

```
dataGridView1->Rows->Clear();
bStart->Enabled = false;
serialPort1->WriteLine("E;");
```
- START
  - Startet die Abarbeitung der momentan gespeicherten Tabelle
  - Zugehöriger Quelltext:

```
if(dataGridView1->Rows->Count > 1)
{
    serialPort1->WriteLine("A;");
    bStop->Enabled = true;
    bStart->Enabled = false;
}
```
- STOP
  - Stoppt die Abarbeitung der momentan gespeicherten Tabelle
  - Zugehöriger Quelltext:

```
serialPort1->WriteLine("B;");
if(dataGridView1->Rows->Count > 1)
    bStart->Enabled = true;
```

## 6.7 Gespeicherter Bewegungsablauf

- Speichert die an die Hauptplatine gesendeten Befehle zur Übersicht
- Befehl: Speichert den Namen des Befehls
- W\_M1: Speichert den Zielschritt des ersten Motors
  - o Muss immer vierstellig sein
- V\_M1: Speichert die Höchstgeschwindigkeit des ersten Motors
  - o Darf nur eine Stelle belegen
- Wait\_For\_Next: Speichert die Wartebedingung, aufgeschlüsselt in eine zweistellige Zahl
  - o Flag 1: Motor1 muss bereit sein für den nächsten Schritt
  - o Flag 2: Motor2 muss bereit sein für den nächsten Schritt
  - o Flag 4: Motor3 muss bereit sein für den nächsten Schritt
  - o Flag 8: Motor4 muss bereit sein für den nächsten Schritt
  - o Flag 16: Motor5 muss bereit sein für den nächsten Schritt
- Form der Nachricht, siehe Kapitel 5
- Einfügen in die Tabelle, siehe Kapitel 6.2

## 7 Quellenangabe

<http://homepage.hispeed.ch/peterfleury/> → TWI-Header

<http://www.mikrocontroller.net/articles/Schrittmotoren>

<http://www.rn-wissen.de/index.php/Schrittmotoren#L297-L298>

<http://msdn.microsoft.com/en-us/library/w0x726c2.aspx> → .Net 4.5

## 8 Anhang

Auszug aus dem Datenblatt: Widerstandstabelle des NTCs

T <sub>oper</sub> (°C)	R <sub>T</sub> /R <sub>25</sub>	ΔR DUE TO B-TOLERANCE (%)	TC (%/K)	R <sub>25</sub> (kΩ)					
				2381 640 .....; see note 1 at end of tables					
				6.222	6.272	6.332	6.472	6.682	6.103
-5	4.216	1.08	5.24	9.275	11.38	13.91	19.81	28.67	42.16
0	3.255	0.89	5.08	7.162	8.790	10.74	15.30	22.14	32.56
5	2.534	0.70	4.92	5.575	6.842	8.362	11.91	17.23	25.34
10	1.987	0.52	4.78	4.372	5.366	6.558	9.340	13.51	19.87
15	1.570	0.34	4.64	3.454	4.239	5.181	7.378	10.67	15.70
20	1.249	0.17	4.50	2.747	3.372	4.121	5.869	8.492	12.49
25	1.000	0.00	4.37	2.200	2.700	3.300	4.700	6.800	10.00
30	0.8059	0.16	4.25	1.773	2.176	2.660	3.788	5.480	8.059
35	0.6535	0.32	4.13	1.438	1.764	2.156	3.072	4.444	6.535
40	0.5330	0.47	4.02	1.173	1.439	1.759	2.505	3.624	5.330
45	0.4372	0.62	3.91	0.9618	1.180	1.443	2.055	2.972	4.372
50	0.3605	0.77	3.80	0.7932	0.973	1.190	1.694	2.451	3.606
55	0.2989	0.91	3.70	0.6575	0.807	0.9863	1.405	2.032	2.989
60	0.2490	1.05	3.60	0.5478	0.672	0.8217	1.170	1.693	2.490
65	0.2084	1.18	3.51	0.4586	0.562	0.6879	0.9797	1.417	2.084
70	0.1753	1.31	3.42	0.3857	0.473	0.5785	0.8239	1.192	1.753
75	0.1481	1.44	3.33	0.3258	0.399	0.4887	0.6960	1.007	1.481
80	0.1256	1.57	3.25	0.2764	0.339	0.4146	0.5905	0.8544	1.256
85	0.1070	1.69	3.16	0.2355	0.289	0.3532	0.5031	0.7278	1.070
90	0.09154	1.81	3.09	0.2014	0.247	0.3021	0.4303	0.6225	0.9154
95	0.07860	1.93	3.01	0.1729	0.212	0.2594	0.3694	0.5345	0.7860
100	0.06773	2.04	2.94	0.1490	0.182	0.2235	0.3183	0.4607	0.6773
105	0.05858	2.15	2.87	0.1289	0.158	0.1933	0.2753	0.3983	0.5858
110	0.05083	2.26	2.80	0.1118	0.137	0.1677	0.2389	0.3457	0.5083
115	0.04426	2.37	2.73	0.0974	0.1195	0.1461	0.2080	0.3010	0.4426
120	0.03866	2.47	2.67	0.0851	0.1044	0.1276	0.1817	0.2629	0.3866
125	0.03387	2.57	2.61	0.0745	0.0915	0.1118	0.1592	0.2303	0.3387
130	0.02977	2.67	2.55	0.0655	0.0804	0.0982	0.1399	0.2024	0.2977
135	0.02624	2.77	2.49	0.0577	0.0709	0.0866	0.1233	0.1784	0.2624
140	0.02319	2.86	2.43	0.0510	0.0626	0.0765	0.1090	0.1577	0.2319
145	0.02055	2.96	2.38	0.0452	0.0555	0.0678	0.0966	0.1398	0.2055
150	0.01826	3.05	2.33	0.0402	0.0493	0.0603	0.0858	0.1242	0.1826