Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

**Bachelorarbeit**

Alexander Abramson

# Ansteuerung eines Drehtisches und mehrerer Peripheriegeräte zur Objekterfassung, -bearbeitung und -positionierung unter Echtzeitbedingungen

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Alexander Abramson

# Ansteuerung eines Drehtisches und mehrerer Peripheriegeräte zur Objekterfassung, -bearbeitung und -positionierung unter Echtzeitbedingungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Franz Korf
Zweitgutachter: Andreas Meisel

Abgegeben am 26. Oktober 2007

**Alexander Abramson**

**Thema der Bachelorarbeit**

Ansteuerung eines Drehtisches und mehrerer Peripheriegeräte zur Objekterfassung, -bearbeitung und -positionierung unter Echtzeitbedingungen

**Stichworte**

Drehtisch, Gleichstrommotor, Entfernungssensor, Inkrementalgeber, Mikrocontroller, AT90CAN128, Pulsweitenmodulation (PWM), kooperativer Scheduler, Echtzeitsystem, RS232, Übertragungsprotokoll

**Kurzzusammenfassung**

Im Rahmen der vorliegenden Bachelor-Arbeit wird ein mit einem Gleichstrommotor angetriebener Drehtisch zur Rotation von Objekten konstruiert. In den Drehtisch wird ein AVR Controller Board zum Antrieb des Motors und zur Erfassung von Objekten eingebaut. Die rotierenden Objekte werden von einer Hochauflösungskamera erfasst. Die Bilddaten werden an einem PC ausgewertet. Anhand der Ergebnisse der Auswertung können ausgewählte Objekte in bestimmter Position angehalten werden

**Alexander Abramson**

**Title of the paper**

Control of a turn table equipped with several peripheral handling devices – Processing and positioning in real time

**Keywords**

Turn table, DC motor, Distance measuring sensor, Optical kit encoder, Microcontroller, AT90CAN128, Pulse width modulation (PWM), Co-operative scheduler, Real time system, RS232, Message protocol

**Abstract**

The goal of the current bachelor thesis is to build a turn table driven by DC motor on which different types of objects are placed. The motor control as well as the acquisition and processing of the data will be achieved by a micro controller board mounted on the table. While the table is rotating the different devices on the table are filmed by a high resolution camera. The so captured images are transferred to a personal computer for analyse and evaluated. On the basis of the evaluation results specific selected objects at specific position can be intentionally stopped.

# Inhaltsverzeichnis

# Einleitung

## Vorwort

In automatisierten Förder- und Logistikanwendungen werden zum Transport des Materials eine Vielzahl von Bewegungen gesteuert. Dabei spielen unter anderem die rotatorischen Bewegungen über Drehtische eine wesentliche Rolle.

Drehbewegungen erfolgen häufig getaktet(Rundtakt-Tische), das Material wird dabei um eine bestimmte Gradzahl weitergetaktet. Es gibt viele Drehanwendungen, bei denen das Material über den kürzersten Weg zum Ziel gelangen soll oder die Zielposition nur mit einer definierten Drehrichtung angefahren werden darf(Positionierung mit fester Drehrichtung) (SEW-Eurodrive (2007) s.6).

Auch in der Messtechnik, insbesondere im Bereich der industriellen Bildverarbeitung finden die Drehtische ihren Einsatz. Ziel ist meist die Lösung einer ganz speziellen Prüfaufgabe. Bauteile sollen während der Produktion auf ihre Qualität geprüft werden. Da, wo der Mensch wegen der hohen Geschwindigkeit überfordert ist, lässt sich mit der Hilfe der Maschine der Produktionsprozess so steuern, dass möglichst wenig Fehler entstehen. Die automatischen Prüfsysteme sind extrem schnell und präzise, so dass fehlerhafte Teile in Echtzeit erkannt und aussortiert werden können.

Die vorliegende Bachelor-Arbeit beschäftigt sich mit der Konstruktion eines Drehtisches, der zur Objekterfassung, -bearbeitung und -positionierung unter Echtzeitbedingungen verwendet wird. Die auf dem Drehtisch rotierenden Objekte werden mit einer Kamera aufgenommen. Mit Hilfe einer Bildbearbeitungssoftware wird die Farbe der aufgenommenen Objektes festgestellt. Die Objekte in bestimmter Farbe können bei variierbarer Geschwindigkeit des Drehtisches an eine definierte Zielposition gebracht werden.

Der Drehtisch wird durch einen Gleichstrommotor angetrieben. Zur Objekterfassung werden eine Hochauflösungskamera und ein Lichtblitz-Stroboskop eingesetzt. Die Peripheriegeräte(Kamera,Blitz) werden angesteuert, nachdem die Objekte eine am Drehtisch angebrachte Lichtschranke passiert sind. Die Ansteuerung der Hardware erfolgt durch einen im Drehtisch eingebauten Mikrocontroller.

Die Bilddaten der von der Kamera erfassten Objekte werden am PC ausgewertet. Ferner wird am PC eine GUI bereitgestellt, in der die aufgenommenen Objekte angezeigt werden

und von der aus die Benutzer ihre Befehle(wie z.B. „Drehtisch starten“) an die Drehtischsteuerung senden können. Die Ergebnisse der Auswertung und die Benutzerbefehle sendet der PC über die serielle Schnittstelle an den Mikrocontroller.

Die Detektion und die Erfassung der Objekte wird durch die gegenseitige Synchronisation der einzelnen HW-Module realisiert. Durch eine implementierte Zeitsteuerung werden die zeitlichen Abstände zwischen der Detektion der Objekte durch die Lichtschranke und der Erfassung durch die Kamera an die Winkelgeschwindigkeit des Drehtisches angepasst. Die Zeitsteurung garantiert, dass die Sensoren(Lichtschranke, Geschwindigkeitssensor) und die Aktoren(Motor,Kamera) in periodischen Abständen angesteuert werden.

Ferner ist die maximal zulässige Anzahl der Objekte, die auf dem Drehtisch gleichzeitig platziert werden können, festzulegen. Die Anzahl ist sowohl von der maximalen Erfassungsfrequenz der Lichtschranke als auch von der maximalen Aufnahmefrequenz der Kamera abhängig.

## Ziele

Die Bachelor-Arbeit umfasst neben der Konstruktion und dem Aufbau eines mit einem Gleichstrommotor angetriebenen Drehtisches, auch die Programmierung der Antriebsteuerung sowie die Steuerung und die Synchronisierung der Peripheriegeräte.

Folgendes Szenario soll mit dem Drehtisch und den Peripherie-Geräten realisiert werden können:

- **Schritt 1**: Der Benutzer schaltet die Anlage ein und ruft an einem PC die Benutzersteuerung auf. Von da aus stellt er die Peripheriegeräte ein (z.B. Belichtungszeit der Kamera) und wählt die Geschwindigkeit des Rotors aus. Als letztes setzt der Benutzer den Drehtisch vom PC aus in Bewegung.

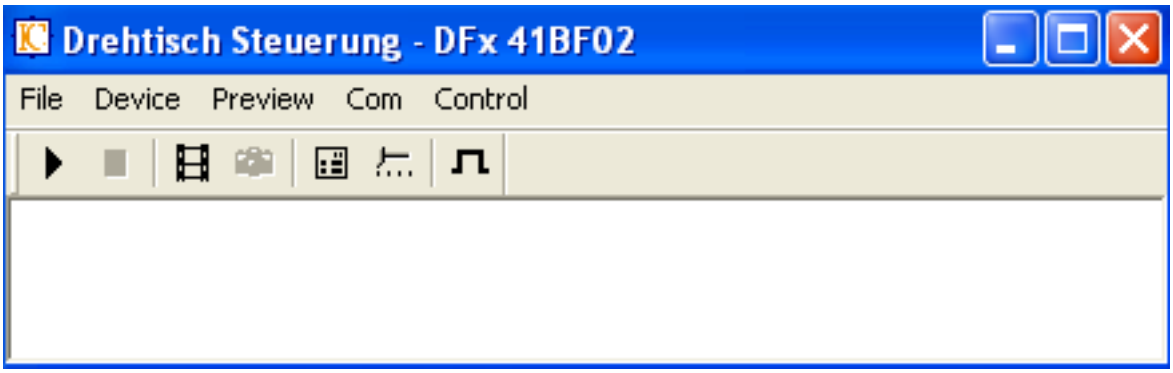



Abbildung 1: Benutzersteuerung

- **Schritt 2**: Über die serielle Datenverbindung kommunizieren PC und Controller mit Hilfe eines Kommunikationsprotokols miteinander. Es werden Pakete fester Länge mit Befehlscodes und Daten ausgetauscht. Vom PC aus kommen die Aufträge ( z.B. vom Benutzer ausgewählte Objektfarbe) bei der Drehtisch-Steuerung an. Zur gleichen Zeit sendet die Kamera die Bilddaten über eine Fire-Wire Schnittstelle an den PC. Die Bilddaten werden von der Bildbearbeitungs-Software ausgewertet. Die Ergebnisse der Auswertung (z.B. das Objekt ist schwarz) werden über die erwähnte Datenverbindung an den Mikrocontroller geleitet.

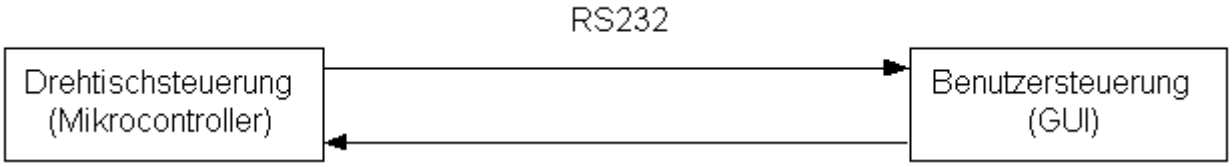



Abbildung 2: Datenverbindung zwischen Drehtisch und PC

- Schritt 3: Stimmt die Farbe des von der Kamera erfassten Objektes mit der, die vom Benutzer ausgewählt wurde, so wird das Objekt in eine Bestimmte Position (Ausgabe-Station) an dem Drehtisch gebracht. Dazu muss der Motorantrieb des Drehtisches so angesteuert werden, so dass der Drehteller genau in der gewünschten Position anhält.

Im Rahmen der Bachelor-Arbeit werden im einzelnen folgende HW- und SW-Module geplant und realisiert:

- Hardware
  - Konstruktion des Drehtisches
  - Auswahl der notwendigen Sensoren (Lichtschranken, Infrarot-Sensoren) und Aktoren (Motor)
  - Steuerung (microC, Motoransteuerung)
  - Verkabelung der einzelnen Einheiten
- Software
  - Ansteuerung des Drehtisches und der Peripheriegeräte (Kamera, Blitz)
  - Kommunikationsschnittstelle zu einem PC zum Senden der erfassten Bilddaten sowie zum Empfang der Ergebnisse der Auswertung (hier z.B.: das Objekt ist schwarz)
  - GUI zur Ansteuerung des Drehtisches durch den Benutzer (z.B. Vorgabe der Geschwindigkeit, Belichtungszeit der Kamera, Auswahl des Objektes)

# Kapitel 1

# Analyse

In diesem Kapitel werden einzelne HW- und SW-Komponenten vorgestellt, die im Rahmen der vorliegenden Bachelor-Arbeit realisiert werden.

## 1.1 Hardware

In den folgenden Kapiteln wird die Hardware beschrieben, die für die Erfüllung der in der Einleitung gestellten Ziele benötigt wird. Es werden die Anforderungen definiert, die bei der Konstruktion bzw. bei der Wahl einzelner Hardware-Bausteine zu beachten sind.

### 1.1.1 Drehtisch

An den Drehtisch ist eine Reihe von Anforderungen bezüglich der Konstruktion, der Größe und des verwendeten Materials zu stellen.

#### 1.1.1.1 Aufbau

Der Drehtisch sollte stabil und präzise sein. Die Drehung hat um einen sauber definierten Mittelpunkt zu erfolgen, d.h. die Achse darf kein merkliches Spiel haben. Die aus der Drehung der Platte entstehende Unwucht, die mit der Zunahme der Rotationsgeschwindigkeit stärker wird, sollte vermieden oder zumindest auf ein Minimum reduziert werden.

Anmerkung: max. Geschwindigkeit, mit der der Drehtisch betrieben werden soll, wird auf 0.5 rps (Rotation pro Sekunde) festgelegt.

Die durchgeführten Tests haben ergeben, dass bei höheren Drehgeschwindigkeiten die Objekte ihre Standfestigkeit bezüglich des Drehtellers verloren haben. Es hat sich bei diesen Versuchen ebenfalls herausgestellt, dass die Motorbelastung durch den Drehteller beim Unterschreiten einer bestimmten Drehgeschwindigkeit (etwa 0.2 rps) zu groß war, um einen

störfreien Betrieb des Drehtisches zu garantieren. Demzufolge wird die minimal zulässige Drehgeschwindigkeit auf 0.2 rps festgesetzt.

#### 1.1.1.2 Größe

Der Drehteller sollte groß genug sein, um die zu erfassenden Objekte darauf platzieren zu können. Die benutzten Testobjekte haben die Form eines Zylinders mit Radius r=5cm und einer Höhe h=10cm. Die Objekte sind auf dem Drehtisch so zu positionieren, dass sie sich im Focus der Kamera sowie des Lichtstrahls des Stroboscopes befinden. Die mögliche Aufstellung der Peripherie-Geräte und der Objekte kann man auf dem folgenden Bild sehen.

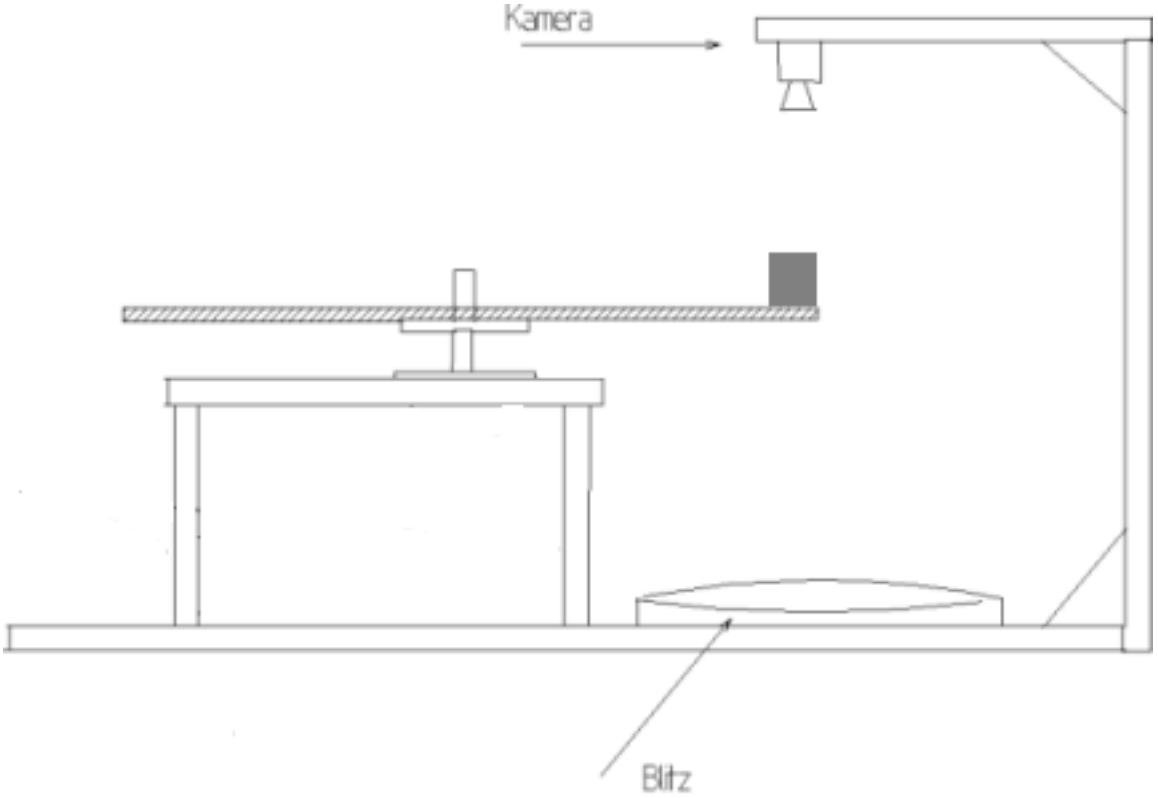



Abbildung 1.1: Peripheriegeräte

Die Größe des Drehtellers wird durch die Position der Peripherie-Geräte und der Objekte bestimmt.

Ferner sollte im Innenraum des Drehtisches ausreichend viel Platz für den Einbau des Motorantriebes und der Steuerungselektronik vorhanden sein.

#### 1.1.1.3 Material

Die Objekte werden von unten mit dem Lichtblitz-Stroboskop beleuchtet und von der oben befestigten Kamera erfasst. Daher sollte der Drehteller aus einem transparenten Material gefertigt werden.

Einige Anforderungen an den Drehtisch werden in der folgenden Tabelle zusammengefasst:

| | | |
|---|---|---|
| **Geschwindigkeit** | min | 0,2rps |
| | max | 0,5rps |
| **Objektgröße** | min | r=5cm,h=10cm |
| | max | r=5cm,h=10cm |
| **Objektanzahl n** | min | 1 |
| | max | $n_{max}$ |

Tabelle 1.1: Anforderungen an den Drehtisch

Die Anzahl der Objekte, die auf dem Drehteller gleichzeitig platziert werden können, ist von der maximalen Erfassungsfrequenz der Lichtschranke sowie von der maximalen Aufnahmefrequenz der Kamera abhängig. Bei den Tests wurden maximal 4 Objekte eingesetzt.

### 1.1.2 Peripheriegeräte

Die beiden in der Bachelor-Arbeit eingesetzten Peripheriegeräte werden zur Aufnahme der auf dem Drehtisch befindlichen Objekte verwendet. Wie man auf der Abbildung 1.1 sehen kann, wird die Kamera über dem Drehteller befestigt und der Stroboskop darunter. Die beiden Geräte sollten einen Triggereingang haben. Bei der Kamera wird er zur Bestimmung des Zeitpunktes der Belichtung eingesetzt und bei dem Lichtblitz zur Bestimmung des Zeitpunktes der Beleuchtung. Die Kamera und der Stroboskop werden miteinander synchronisiert, so dass der Lichtblitz während der Belichtungszeit der Kamera mehrmals ausgelöst werden kann. Die genaue Anzahl der Blitze pro Bildaufnahme soll mit Hilfe von Testreihen bestimmt werden. Es wäre vom Vorteil, wenn man die Belichtungszeit der Kamera variieren könnte. So könnte man die Belichtungszeit an die Blitzfrequenz des Stroboskops anpassen.

Die Bildaufnahmefrequenz der Kamera sollte hoch genug sein, um alle Objekte erfassen zu können. Die maximale Anzahl der Objekte, die sich gleichzeitig auf dem Drehteller befinden können, ist in der Tabelle 1.1 aufgelistet.

### 1.1.3 Motorantrieb

Durch den Motor soll der Drehteller, der mit der Motorachse verbunden wird, zum Rotieren gebracht werden. Die Rotationsgeschwindigkeit muss in einem definierten Bereich variierbar sein. Wird eine bestimmte Geschwindigkeit vom Benutzer vorgegeben, sollte sie konstant gehalten werden können.

Ein wichtiges Kriterium bei der Wahl des Motors ist das abgegebene Drehmoment $D$ gemessen in Newtonmeter (Nm). Das notwendige Drehmoment berechnet sich aus dem Produkt vom Trägheitsmoment $I$ des zu rotierenden Körpers und der Winkelbeschleunigung $\alpha$ (Siehe Formel:1.1).

$$D = I \cdot \alpha \tag{1.1}$$

**Trägheitsmoment $I$** Der Drehteller hat die Form eines Zylinders, der um seine Symmetrieachse rotiert (siehe Abb. 1.2).

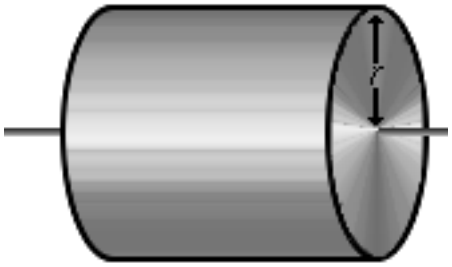



Abbildung 1.2: Ein Vollzylinder, der um seine Symmetrieachse rotiert.

Das Trägheitsmoment eines Vollzylinders lässt sich nach der folgenden Formel berechnen:

$$I = \frac{1}{2} \cdot m \cdot r^2 \tag{1.2}$$

**Winkelbeschleunigung $\alpha$** Die Winkelbeschleunigung $\alpha$ gemessen in $rad/s^2$ ist die zeitliche Änderung der Winkelgeschwindigkeit $\omega$ gemessen in $rad/s$. Zwischen beiden Größen besteht folgender funktionaler Zusammenhang:

$$\alpha = d\omega/dt$$

Die Winkelbeschleunigung des Drehtellers kann man anhand folgenden Beispiels abschätzen:

Angenommen, der Drehteller rotiert mit der Winkelgeschwindigkeit $\omega_1 = 0,4$ rps ( oder umgerechnet $4\pi/5$ rad/s). Er soll auf die Winkelgeschwindigkeit $\omega_2 = 0,5$ rps oder $\pi$ rad/s

beschleunigt werden. Vorausgesetzt, die Winkelbeschleunigung $\omega$ ändert sich linear bezüglich der Änderung der Winkelgeschwindigkeit $\alpha$ und die notwendige Zeit $t_a$ für den Beschleunigungsvorgang zwischen $t_{min} = 0,25s$ und $t_{max} = 0,5s$ liegt, dann muss $\alpha$ im folgenden Bereich liegen (s. Formel 1.3):

$$\alpha_{min} = \frac{\omega_2 - \omega_1}{t_{max}} \leq \alpha \leq \frac{\omega_2 - \omega_1}{t_{min}} = \alpha_{max} \tag{1.3}$$

Setzt man nun in die Formel die dazugehörigen Werte ein, kommen für die Extrema der Winkelbeschleunigung folgende Werte heraus:

$$\alpha_{min} \approx 1,25rad/s^2 \leq \alpha \leq 2,5rad/s^2 \approx \alpha_{max}$$

Zur Berechnung des notwendigen Drehmoments fehlen noch folgende Größen: Masse m und Radius r (siehe die Formeln 1.1 und 1.2) des Drehtellers. Nachdem die fehlenden Größen im Kapitel 2.1.1 festgelegt werden, kann das Drehmoment des einzusetzenden Motors bestimmt werden. Im Kapitel 2.1.3 kann man dann anhand dieses Wertes die Auswahl eines geeigneten Motors vornehmen.

## 1.1.4 Sensoren

### 1.1.4.1 Objekterkennung

Wie bereits im Kapitel 1.1.2 dargestellt wurde, soll die Kamera getriggert werden, wenn die Objekte sich in ihrer Höhe befinden. Synchron zur Kamera wird der Stroboskop angesteuert.

Um festzustellen, dass die Objekte die Kamera erreicht haben, kann man sie kurz davor durch einen Sensor detektieren. Der Sensor (z.B. eine Infrarot-Lichtschranke) wird an der Seite des Drehtellers, in einem Winkel $\varphi$ von der Kamera entfernt, angebracht. Auf der Abbildung 1.3 ist mit A die Position gekennzeichnet, an der die Objekte die Lichtschranke durchlaufen und mit B die Position der Kamera.

Es muss zusätzlich sichergestellt werden, dass durch den Sensor alle auf dem Drehteller befindlichen Objekte erfasst werden. Sind die Einstellungen (Rotationsgeschwindigkeit, Anzahl der Objekte) auf Maximum gesetzt (s. Tabelle. 1.1), darf die Frequenz der Lichtschranke nicht einen bestimmten Wert $f_{min\ Abstast}$ unterschreiten.

$$f_{min\ Abtast} > 2 \cdot Objektanzahl_{max} \cdot \omega_{max} \tag{1.4}$$

Die Multiplikation mit Faktor 2 ist durch das Nyquist-Shannonsche Abtasttheorem bedingt, nach dem die Relation $f_{abtast} > 2 \cdot f_{max}$ erfüllt werden muss. Das Abtasttheorem lautet: *die Abtastrate eines Signals muss mehr als doppelt so hoch sein wie die höchste im Signal vorkommende Frequenz* (Bernstein (2007) s. 72). Setz man die Werte in die obere Formel ein, ergibt sich für $f_{min\ Abtast}$ folgender Wert:

$$f_{min\ Abtast} > 2 \cdot n_{max} Objekte \cdot 0,5rps = n_{max}\ Hz \tag{1.5}$$

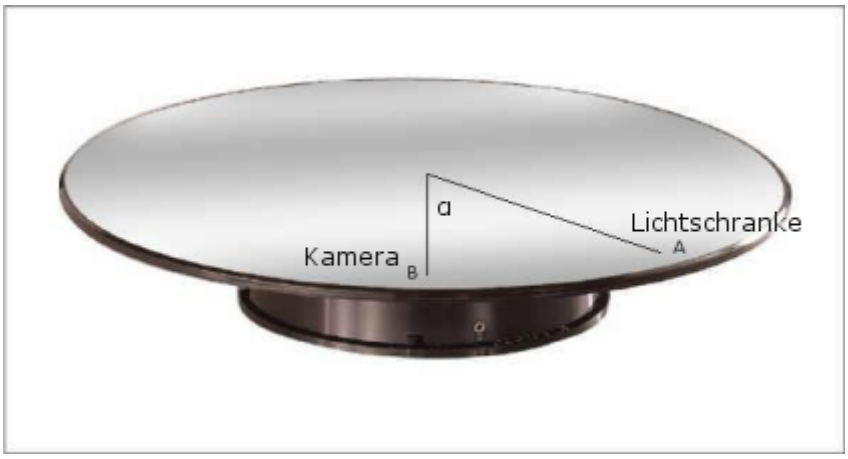



Abbildung 1.3: Objektdetektion

#### 1.1.4.2 Drehzahlmessung

Die aktuelle Rotationsgeschwindigkeit des Drehtellers kann mit Hilfe eines zweiten Sensors (z.B Inkrementalgeber), der an der Motorachse befestigt wird, berechnet werden. Die Drehzahl wird durch die Anzahl der Signale, die der Sensor während einer Umdrehung des Drehtellers an die Steuerung liefert, bestimmt. Genaueres zur Arbeitsweise dieses Sensors findet man im Kapitel 2.1.4.

Der Sensor liefert $n_{Signale}$ pro Umdrehung. Bei der maximalen Rotationsgeschwindigkeit $\omega_{max} = 0,5rps$ kann die minimale Abtastfrequenz $f_{min\ Abtast}$ nach der folgenden Formel berechnet werden:

$$f_{min\ Abtast} > 2 \cdot n_{Signale} \cdot 0,5rps = n_{Signale}\ rps \tag{1.6}$$

### 1.1.5 Benötigte Elektronik, AVR Controller Board

Die in den vorherigen Kapiteln beschriebenen HW-Komponenten werden durch eine in den Drehtisch eingebaute Steuerungseinheit (z.B. ein Mikrocontroller) miteinander vernetzt. Die Steuerung verarbeitet analoge Sensorsignale, sendet Triggersignale an die Peripheriegeräte und steuert den Motorantrieb an. Sie dient dabei auch zur Interaktion der Hardware mit der Außenwelt (hier: PC mit graphischer Benutzeroberfläche 1.2.2). Die Kommunikation läuft über eine definierte Schnittstelle mittels eines Protokolls 1.2.3 ab.

Für diese Zwecke wird in dieser Bachelor-Arbeit ein AVR Controller Board eingesetzt, das von den Mitarbeitern der HAW Hamburg konstruiert und gefertigt wurde. Das Herzstück des AVR Boards ist der AT90CAN128-Mikrocontroller (eine fertige Recheneinheit auf einem Chip mit einem Prozessor, einem Programmspeicher und einem Arbeitsspeicher) des Herstellers ATMEL Atmel.

## 1.2 Software

In diesem Kapitel werden die für die Realisierung des Bachelor-Arbeit notwendigen SW-Komponenten vorgestellt.

### 1.2.1 HW-Steuerung

Um die in der Einleitung definierten Ziele zu erreichen, wird auf dem Mikrocontroller eine SW implementiert, die folgende Funktionen zur Verfügung stellt:

- Ansteuerung des Motors (Start, Stop, Änderung der Drehzahl)
- Verarbeiten der Sensorsignale (Lichtschranke, Inkrementalgeber)
- Triggerung der Peripheriegeräte (Auslösen der Kamera und des Stroboskops)

Im weiteren wird dargestellt, wie mit Hilfe dieser Funktionen, die Ziele dieser Bachelor-Arbeit erreicht werden können.

#### 1.2.1.1 Erfassung der Objekte

Ein Objekt auf dem Drehteller wird erkannt, wenn es die Lichtschranke passiert. In diesem Moment muss festgestellt werden, nach welcher Zeit das Objekt die Kamera und das Stroboskop erreicht, um die Peripheriegeräte rechtzeitig triggern zu können. Der Winkel $\varphi$ in rad, der zwischen der Lichtschranke und der Kamera liegt, ist davon abhängig, wie man den Sensor und die Geräte positioniert.

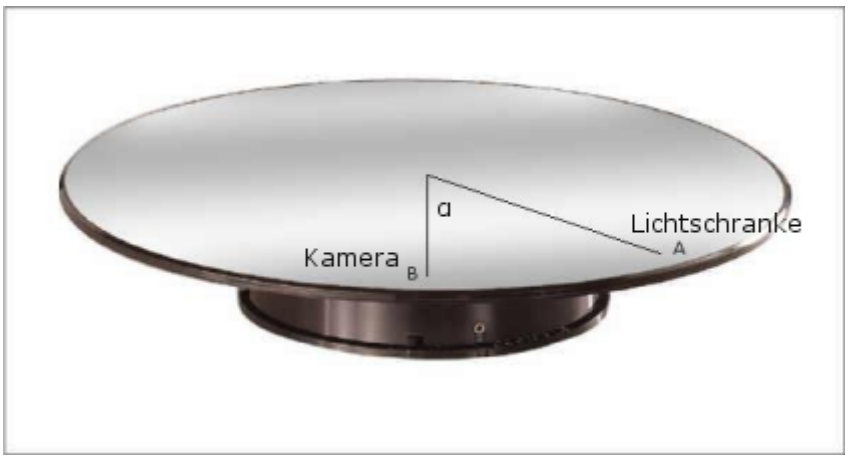



Abbildung 1.4: Erfassung der Objekte

Die aktuelle Winkelgeschwindigkeit $\omega$ kann mit Hilfe des Inkrementalgebers berechnet werden. Die Zeit $t$, die zwischen Detektion des Objektes und seiner Erfassung liegt, beträgt:

$$t = \frac{\varphi}{\omega}$$

Da die Erfassung der Objekte unter Echtzeitbedingungen abläuft , muss die Software garantieren, dass die Berechnung der Zeit $t$ fertiggestellt ist, bevor das Objekt die Kamera erreicht.

Wurde das Objekt aufgenommen, so werden die Bilddaten von der Kamera an den PC weitergeleitet. Der Mikrocontroller teilt der Benutzersteurung mit intergrierter Bildbearbeitung-SW am PC (siehe Kapitel 1.2.2) mit, dass neue Bilddaten zur Verfügung stehen. Nach der Auswertung der Daten empfängt der Mikrocontroller die Ergebnisse der Bildbearbeitung. Die Kommunikation zwischen dem Mikrocontroller und dem PC läuft über ein Kommunikationsprotokol (siehe Kap. 1.2.3) ab.

#### 1.2.1.2 Bearbeitung von Nachrichten

Die weitere Aufgabe der Mikrocontroller-SW ist die Bearbeitung der Nachrichten, die von der Benutzersteuerung gesendet werden. Der Aufbau der Nachrichten wird im Kapitel 1.2.3 beschrieben. Der Benutzer kann Nachrichten mit folgenden Befehlen an die Drehtisch-Steuerung senden:

**Drehtisch starten** - Drehteller wird mit einer Anfangsgeschwindigkeit in Bewegung gesetzt

**Drehtisch anhalten** - Drehteller wird angehalten

**Rotation des Drehtellers beschleunigen bzw. verlangsamen** - Die Motordrehzahl wird erhöht bzw. gesenkt

**Drehtisch in bestimmter Position anhalten** - der Benutzer kann die Mikrocontroller-SW auffordern, den Drehtisch so anzuhalten, dass ein Objekt in einer bestimmten Farbe in der Zielposition (bestimmtes Segment des Drehtellers) stehenbleibt.

### 1.2.2 Benutzersteuerung und Bildbearbeitung

Zur Steueurung des Drehtisches wird ein Programm mit graphischer Oberfläche dem Benutzer zur Verfügung gestellt. Das Programm, im Folgenden als Benutzersteuerung (oder User-Interface) bezeichnet, kann von einem PC oder Notebook ausgeführt werden.

Zur Verarbeitung der von der Kamera empfangenen Bilddaten ist eine weitere SW-Komponente erforderlich, die entweder als eigenständiges Programm oder als Bestandteil der Benutzersteuerung vom PC aus agieren kann. Diese Komponente soll im Rahmen einer anderen Bachelor-Arbeit entwickelt und getestet werden.

### 1.2.3 Kommunikationsschnittstelle

In den letzten beiden Kapiteln wurden zwei wesentliche SW-Bestandteile dieser Bachelor-Arbeit vorgestellt. Das ist zum einen die Software auf dem AVR Controller Board ( siehe Kap. 1.2.1), zu deren Aufgaben die Steuerung des Motorantriebes, der Sensoren und der Peripheriegeräte gehört, und zum anderen die Software auf dem PC, die sowohl dialogbasierte Benutzersteuerung als auch eine Bildbearbeitungssoftware integriert (siehe Kap. 1.2.2). Vom PC aus sollen Benutzerbefehle(z.B.: „Drehtisch starten“ ) an den AVR Controller Board gesendet werden können, und in die umgekehrte Richtung soll die Benutzersteuerung mit Statusmeldungen wie z.B. „ein Objekt passierte die Lichtschranke“ über den aktuellen Stand der Hardware informiert werden.

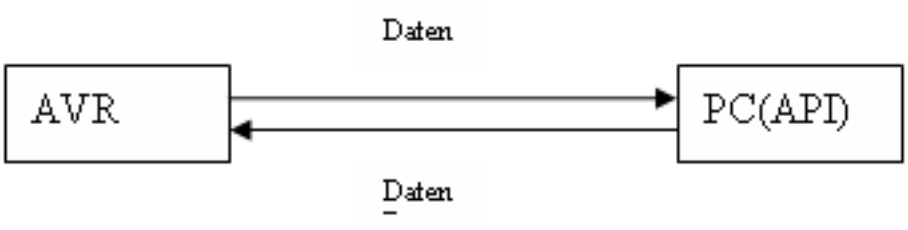



Abbildung 1.5: Kommunikation der Module

Es muss also eine Lösung gefunden werden, wie man eine Datenkommunikation zwischen diesen beiden Einheiten herstellen kann. In weiteren Ausführungen werden Anforderungen an diese Lösung definiert.

Voraussetzung für den Austausch der Daten ist eine physikalische Verbindung (Übertragungsstandard), die von beiden Kommunikationsteilnehmern unterstützt wird.

#### 1.2.3.1 Physikalische Verbindung

Da die beiden Module sowohl AVR Controller Board als auch der PC über serielle Schnittstelle verfügen, also den gleichen Übertragungsstandard unterstützen, erschien die Wahl der seriellen Kommunikation für die Anforderungen dieser Arbeit am sinnvollsten. Ferner ist RS232 der am meisten verwendete Übertragungsstandard, um Daten zwischen einem Mikrocontroller und einem PC hin und her zu schicken Trampert (2003b). Demzufolge existiert eine Reihe von Übertragungsprotokollen, die darauf basieren, und die als Vorlage für die Suche nach einem für die vorliegende Arbeit geeignetem Protokoll dienen können.

In folgender Auflistung sind die Gründe für die Wahl der seriellen Schnittstelle zusammengefasst dargestellt:

- Beide Module (AVR und PC) verfügen über serielle Schnittstelle (d.h. keine Erweiterung der vorhandenen HW notwendig)

- RS232 ist ein oft verwendeter Übertragunsstandard (d.h. es gibt viele Übertragungsprotokolle für die Schnittstelle)

#### 1.2.3.2 Übertragunsprotokoll

Die Vermittlerrolle zwischen dem AVR Controller Board und dem PC übernimmt ein Übertragungsprotokoll. Im folgenden werden Anforderungen an den Aufbau und an die Arbeitsweise des Protokolls definiert. Dabei werden einige allgemeingültigen Eigenschaften eines Übertragungsprotokolls als Referenz herangezogen Netzwerkprotokoll (2007).

**Aufbau eines Datenpakets**
Der im Protokoll beschriebene Aufbau eines Datenpakets enthält für den Datenaustausch wichtige Informationen über das Paket wie beispielsweise:

- dessen Absender und Empfänger
- den Typ des Pakets
- die Paketlänge
- die Prüfsumme

Diese Informationen werden den Nutzdaten als so genannter Header vorangestellt oder als Trailer angehängt .

**Arbeitsweise des Protokolls**

- Die Daten müssen in beide Richtungen fließen können(AVR→PC und PC→AVR). PC sendet an den AVR z.B. ein Befehl zum Starten des Drehtisches und AVR teilt dem PC mit, dass ein neues Bild aufgenommen wurde.
- Stellung der Kommunikationsteilnehmer: Die beiden Module AVR und PC sind untereinander gleichberechtigt, weil in beide Richtungen sowohl Befehlscodes als auch Statusmeldungen versendet werden, d.h. die Kommunikation läuft symmetrisch ab.
- Da das Protokoll eine Ansteuerung des Drehtisches unter Echtzeitbedingungen garantieren muss, sind die einzelnen Anfragen der beiden Module vom Empfänger verzögerungsfrei zu beantworten(synchrone Kommunikation).

**Zeitverhalten des Protokolls**

Nachdem ein Objekt detektiert wird, sendet die Drehtisch-Steuerung eine Nachricht vom Typ „NEW_PICTURE“ an den PC. Nachdem dieses Objekt von der Kamera aufgenommen wurde, sendet die Kamera die Bilddaten an den PC. Damit die Nachrichten den richtigen Bilddaten zugeordnet werden können, muss folgende Bedingung erfüllt werden: die Zeit zwischen der Detektion von zwei aufeinander folgenden Objekten darf nicht kleiner sein, als die notwendige Zeit zum Versenden einer Nachricht.

Bei der maximalen Drehgeschwindigkeit $\omega_{max}$, der maximalen Objektanzahl $n_{max}$ (s. Tabelle 1.1) und der gleichmäßigen Positionierung der Objekte auf dem Drehteller kann der minimale zeitliche Abstand zwischen der Detektion von zwei Objekten nach Formel 1.7 berechnet werden:

$$t_{Det_{min}} = \varphi_{min} / \omega_{max} \tag{1.7}$$

wobei $\varphi_{min} = 1/n_{max}$ ist.

Angenommen, die eingestellte Baudrate der seriellen Schnittstelle liegt bei 9600 Baud oder Bits/s und ein Paket besteht aus 8 Datenbits, einem Startbit und einem Stopbit. Die Länge einer Nachricht beträgt Len Zeichen. Dann kann die Zeit zur Übertragung einer Nachricht nach der Formel 1.8 berechnet werden.

$$t_{message} = Len \cdot 10/9600Baud \tag{1.8}$$

Bei der Implementierung des Übertragungsprotokolls muss also folgende zeitliche Abhängigkeit zwischen der Detektion der Objekte und der Übertragung von Nachrichten berücksichtigt werden:

$$t_{Det_{min}} > t_{message} \tag{1.9}$$

# Kapitel 2

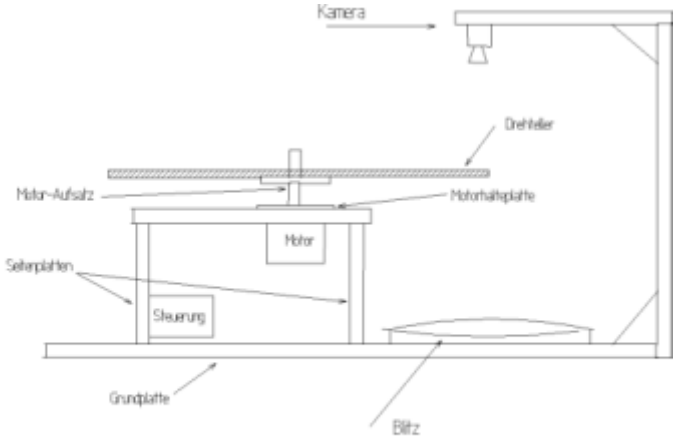
# Design

In folgenden Kapiteln werden Konzepte und Implementierungsdetails zur Umsetzung der im Kapitel 1 formulierten Anforderungen vorgestellt. Im Kapitel 2.1 wird beschrieben, wie die einzelnen HW-Komponenten ausgewählt bzw. konstruiert werden. Im Kapitel 2.2 wird der Aufbau einzelner SW-Module(Drehtisch-Steuerung, Kommunikationsprotokoll, Benutzersteuerung) vorgestellt.

## 2.1 Hardware

### 2.1.1 Drehtisch

Auf der Abbildung 2.1.1 ist eine Skizze des Drehtisches mit dazugehöriger Peripherie dargestellt.



Die Konstruktionspläne einzelner Komponenten (Grundplatte, Seitenplatte, Deckplatte, Motorflansch) befinden sich im Anhang (s. Kap. B).

Bei der Konstruktion werden die im Kapitel Analyse formulierten Anforderungen berücksichtigt. In den folgenden Unterkapiteln wird darauf eingegangen, wie man sie im einzelnen gelöst hat.

#### 2.1.1.1 Aufbau

Um die aus der Drehung entstehende Unwucht zu minimieren, wird der Drehteller durch kugelgelagerte Standfüsse unterstützt.

Abbildung 2.1: kugelgelagerter Standfuß

Zu diesem Zweck wird eine zusätzliche kreisförmige Fläche (in der Abb. 2.2 mit 2 markiert) auf der Deckplatte montiert. Sie wird durch drei zylinderförmige Stützen in $120°$ Abstand befestigt. Die drei Standfüsse (im Bild 2.2 mit 1 markiert) werden ebenfalls in $120°$ Abstand zwischen dieser Fläche und dem Drehteller eingebaut.

#### 2.1.1.2 Größe

Die Größe der einzelnen Komponenten des Drehtisches wird auf die Größe der Objekte und der Peripheriegeräte zugeschnitten. Die genauen Abmessungen der Komponenten sind in den Konstruktionsplänen im Anhang zu finden.

#### 2.1.1.3 Material

Als Material für den Drehteller kam der Kunststoff Makrolon in Frage. Die herausragenden Eigenschaften von Makrolon sind Transparenz, hohe Wärmeformbeständigkeit und hohe Zähigkeit (MaterialScience (2007) s.6).

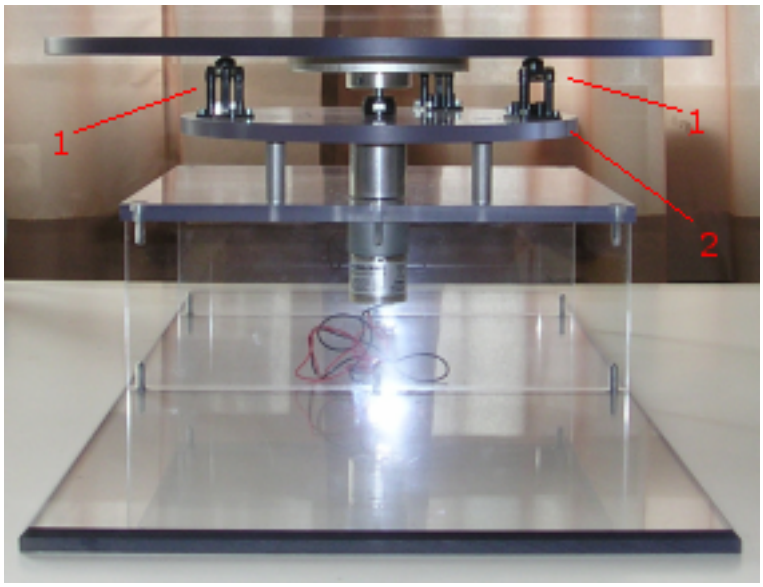



Abbildung 2.2: Einbau der Standfüsse

### 2.1.2 Peripheriegeräte

In der vorliegenden Bachelor-Arbeit werden folgende Peripheriegeräte eingesetzt:

- Fire-Wire Kamera DFK41BF02 2.3
- Lichtblitz-Stroboskop DRELLOSCOP 255 2.4

Die DFK41BF02 ist eine für industrielle Zwecke hergestellte Farb-Kamera TheImaging-Source (2007). Sie kann bis zu 15 Bilder/s aufnehmen. Die Aufnahmefrequenz ist ausreichend, um alle Objekte bei maximal möglichen Einstellungen 1.1 zu erfassen. Man kann das durch die gleiche Berechnung wie im Fall der minimalen Erfassungsfrequenz der Lichtschranke 1.5 nachweisen.

Abbildung 2.3: Kamera

Wie bereits in der Einleitung erwähnt, wird ein Stroboskop eingesetzt, um die rotirenden Objekte mit periodischen Lichtblitzen immer in der gleichen Position zu beleuchten, so dass für den Betrachter ein scheinbar stehendes Bild entsteht.

Das Stroboskop „Lichtblitz-Stroboskop DRELLOSCOP 255“ Drello (2007) besteht aus einem kompakten, robusten Leichtmetall-Druckgußgehäuse mit eingebauter Blitzröhre.


Abbildung 2.4: Stroboskop

Die Beleuchtung der Objekte erfolgt über Lichtleiter mit unterschiedlichen Vorsatzstücken. In dieser Bachelor-Arbeit wird ein Lichtleiter mit einem kreisförmigen Aufsatz verwendet.

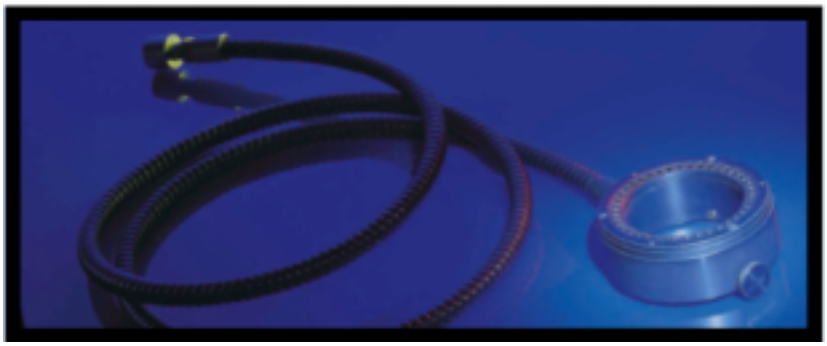


Abbildung 2.5: Der Ringlichtleiter für die Objektbeleuchtung

Sowohl die Kamera als auch das Stroboskop verfügen über einen Triggereingang.

In der Abbildung 2.6 ist dargestellt, wie man den Triggereingang bei der Kamera nutzen kann.

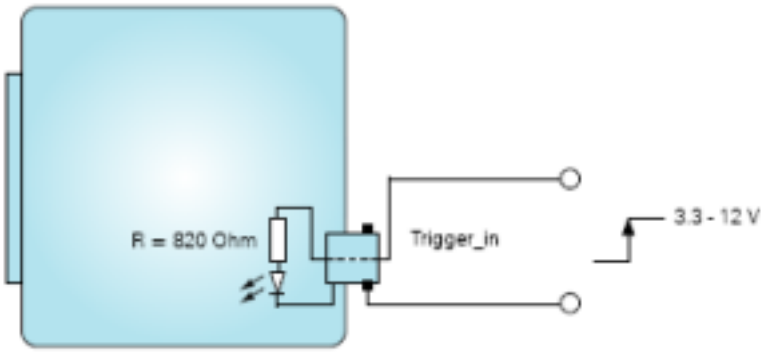



Abbildung 2.6: Nutzung des Triggereingangs

Ein kurzer Trigger-Impuls im Bereich von 3,3 bis 12 Volt setzt die Kamera in Betrieb. In Abbildung 2.7 ist das Verhalten der Kamera im Triggermodus dargestellt.

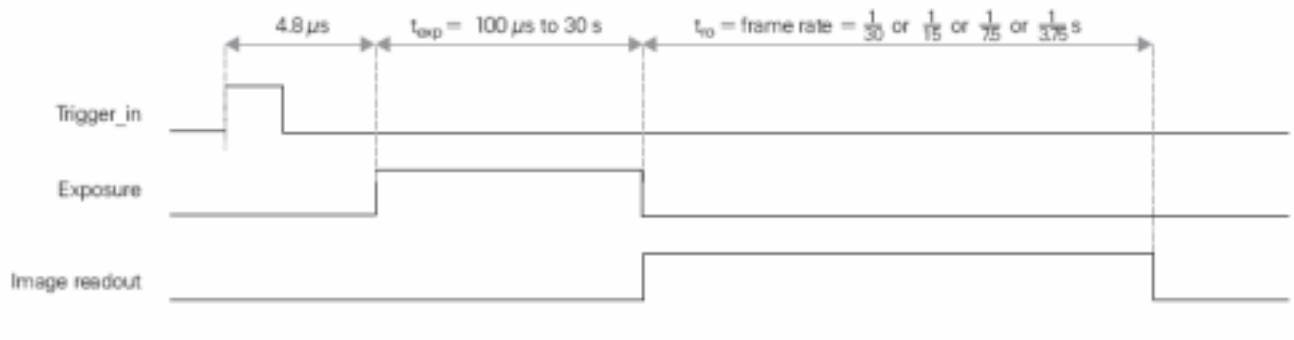



Abbildung 2.7: Verhalten der Kamera im Triggermodus

Die Belichtung (Exposure) beginnt 4,8$\mu$s nach Eintreffen eines Trigger-Impulses. Die Länge der Belichtung ist per Software von 100$\mu$s bis 30s einstellbar. Die Zeitdauer der Bildausgabe (Übertragung der Bilddaten an den PC über Fire-Wire Schnittstelle) bestimmt sich aus dem Kehrwert der jeweils eingestellten Bildrate. Nach dem Abschluss der Bildausgabe akzeptiert die Kamera einen neuerlichen Trigger-Impuls zu einem beliebigen Zeitpunkt.

Angenommen: es befindet sich die maximal zulässige Anzahl der Objekte $n$ auf dem Drehteller und der Drehtisch wird mit maximaler Rotationsgeschwindigkeit ($\omega = 0.5rps$) betrieben 1.1. Der minimale zeitliche Abstand zwischen zwei Objekten (bei gleichmäßiger Po-

sitionierung der Objekte) beträgt somit:

$$t_{distance} = \frac{\frac{1}{n}}{\omega} = \frac{2}{n} sec \tag{2.1}$$

D.h. um die Erfassung aller Objekte sicher zu stellen, darf die Belichtungszeit der Kamera nicht den Wert $t_{distance}$ überschreiten. Bei n=4 liegt die maximale Belichtungszeit bei $2/4 = 0,5s$. Der Triggereingang des Stroboskops reagiert auf elektrische Impulse von 3 bis 12V, die zwischen 30$\mu$s und 1ms dauern. Das Gerät kann bis zu 50 Blitze/s auslösen. Der minimale zeitliche Abstand zwischen zwei Trigger-Impulsen beträgt also:

$$t_{strobo} = \frac{1}{50 Blitze/s} = 20ms/Blitz$$

Die beiden Geräte können im Triggermodus mit folgenden Einstellungen betrieben werden(siehe Tabelle:2.1).

| | |
|---|---|
| **Belichtungszeit der Kamera** | $t_{exposure} = 100ms$ |
| **Abstand zwischen zwei Stroboskopblitzen** | $t_{strobo} = 20ms$ |
| **Anzahl der Stroboskopblitze während einer Belichtungsperiode** | $n_{Blitze/Objekt} = 5$ |

Tabelle 2.1: Einstellungen der Peripherie

Wenn die maximale Anzahl der Objekte den Wert n=20 überschreitet, wird der zeitliche Abstand zwischen zwei Objekten kleiner als 100ms (s. Formel 2.1). Dann müssen die in der Tabelle 2.1 festgelegten Einstellungen geändert werden.

### 2.1.3 Auswahl des Motors

Wie im Kapitel 1.1.3 beschrieben, muss das abgegebene Drehmoment des eingesetzten Motors grösser als das Drehmoment des zu rotierenden Körpers sein. Die für die Berechnung des Drehmoments des Drehtellers notwendigen Maße (Radius $r$ und Masse $m$) können den Konstruktionsplänen B.5 entnommenn werden:

**Radius** $r$ = 20 cm

**Masse** Diese kann durch die Beziehung Masse = Dichte·Volumen oder formal ausgedrückt $m = \rho \cdot V$ berechnet werden

**Volumen** Volumen des Drehtelles $V = \pi \cdot r^2 \cdot h = 1,25 \cdot 10^{-3} m^3$ oder $1,25 \cdot 10^3 cm^3$

**Dichte** Dichte des Makrolons $\rho_{Makrolon} = 1240 kg/m^3$ MaterialScience (2007)

Die Masse $m = 1240 kg/m^3 \cdot 1,25 \cdot 10^{-3} m^3 \approx 1,5 kg$

Mit den berechneten Werten kann nun das Trägheitsmoment nach Formel 1.2 und anschließend das Drehmoment des Drehtellers nach Formeln 1.1 und 1.3 berechnet werden:

$$I = \frac{1}{2} \cdot 1,5kg \cdot 0,2^2 m^2 = 0,03kg \cdot m^2$$

Das Drehmoment liegt dann im folgenden Bereich:

$$I \cdot \alpha_{min}^2 \leq D \leq I \cdot \alpha_{max}^2$$

oder

$$D_{min} = 46,8 \cdot 10^{-3} Nm \leq D \leq 187,5 \cdot 10^{-3} Nm = D_{max}$$

Als Referenzwert zur Auswahl des Motors wird der Mittelwert des Drehmoments zuzüglich 50% als Sicherheit herangezogen:

$$D_{ref} = 3 \cdot (D_{min} + D_{max})/4 \approx 1,8Nm$$

Anhand dieses Referenzwertes wurde folgender Motor für die Bachelor-Arbeit ausgewählt:

Abbildung 2.8: Getriebemotor RB35 1:100

Dieses Model ist mit einem Getriebe von 1:100 untersetzt, das eine Belastung bis zu 18 kg/cm Modelcraft aushält. Die Belastung entspricht einem abgegebenen Drehmoment von 1,8 Nm, das mit dem erforderlichen Drehmoment ($D_{ref}$) des Drehtisches übereinstimmt.

Die Last-Drehzahl des Motors liegt bei 52 U/min oder umgerechnet bei etwa 0,86 (52/60) rps, so dass die höchste Rotationsgeschwindigkeit $\omega = 0,5$ rps (s. Tabelle 1.1) mit diesem Motor ebenfalls realisiert werden kann.

### 2.1.4 Auswahl der Sensoren

Wie bereits im Kapitel 1.1.4 erwähnt wurde, werden in dieser Bachelor-Arbeit zwei Sensoren eingesetzt. Anstatt einer Lichtschranke wird ein Entfernungssensor genommen, der die auf dem Drehteller positionierten Objekte in einem bestimmten Bereich erkennen kann. In der Abbildung 2.9 ist der zu überprüfende Bereich durch das weiß markierte Segment gekennzeichnet. Die vom Sensor erkannten Objekte außerhalb des markierten Bereiches werden nicht beachtet.


Abbildung 2.9: Abtastbereich des Sensors

Der in der Bachelor-Arbeit eingesetzte Entfernungssensor Sharp GP2D12 (s. Abb.2.10) ermöglicht genaue oder relative Abstandsmessungen des Objekts per Infrarot zwischen 10 und 80 cm. Der Sensor kann 20-30 Messungen (s. Timingsimulation in der Abb.3 in Sharp) pro Sekunde durchführen und liefert die gemessene Entfernung als synchrones 8-Bit Datenpaket an den Mikrocontroller zurück.

Abbildung 2.10: Entfernungssensor Sharp GP2D12

Die Entfernung wird als Analogsignal ausgegeben, wobei 10 cm etwa 2,6 V und 80 cm etwa 0,4 V (s. Kennlinie auf Abb.6 in Sharp) entsprechen.

Der in der Abbildung 2.9 markierte Bereich des Drehtellers ist $d$ cm vom Sensor entfernt und $w$ cm breit. Die Werte d und w können variieren. Sind die Werte festgelget, können die Objekte innerhalb des Bereichs detektiert werden. Angenommen: d = 20 cm und w = 10 cm, dann sollen nur die Messungen von der Steuerung ausgewertet werden, die in der Entfernung 20 (etwa 1,4 V) bis 30 cm (etwa 1 V) vom Sensor stattfinden. Damit man aus den analogen Signalen auf die Entfernung der detektierten Objekte schließen kann, müssen die Signale von der Steuerung in digitale Werte übersetzt werden. Aus den Werten kann man dann auf die Entfernung der detektierten Objekte schließen. Der Ablauf der Objektdetektion wird im Kapitel 2.2.1 beschrieben.

Der zweite Sensor ist der Inkrementalgeber E4P Digital, der zur Drehzahlermittlung eingesetzt wird.

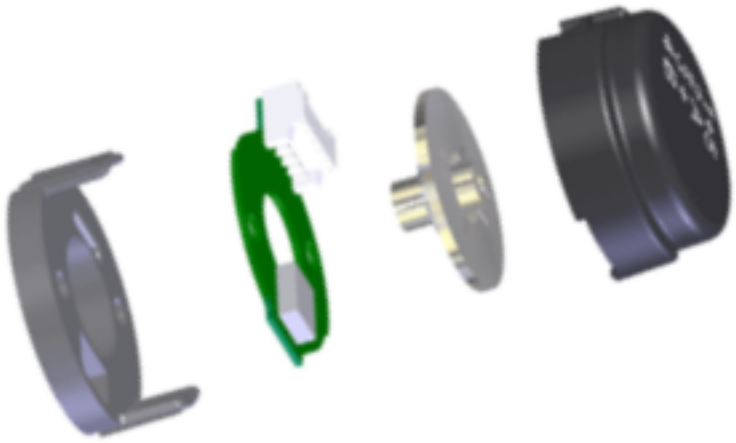


Abbildung 2.11: Inkrementalgeber(Quelle: **?**)

Der Sensor funktioniert nach dem photoelektrischen Prinzip: zwischen einer LED und zwei photoelektrischen Empfangseinheiten ist eine Scheibe mit einem Strichgitter Abb. 2.11 auf die Antriebswelle angebracht. Bei einer Bewegung der Scheibe wird das Licht der LED durch das Strichgitter in seiner Helligkeit moduliert. Es entstehen zwei um 90° versetzte Signale, die von der Empfangseinheit in elektrische Signale umgewandelt werden. Das Strichgitter des verwendeten Sensors ist in 600 feine Abschnitte unterteilt, wobei nach jedem hellen Abschnitt ein dunkler Abschnitt folgt. Demzufolge können pro Umdrehung 600 elektrische Signale (300 Hell-Dunkel Übergänge und 300 Dunkel-Hell Übergänge) generiert werden. Bewegt sich der Drehteller mit der maximalen Rotationsgeschwindigkeit $\omega = 0.5rps$ (s. Tabelle 1.1), so sendet der Inkrementalgeber ein Signal mit der Frequenz $f_{sensor} = 600 \times 0,5rps = 300Hz$ an den Mikrocontroller. Die Frequenz des vom Mikrocontroller zu generierenden Abtastsignals $f_{\mu C}$ sollte doppelt so hoch wie $f_{sensor}$ sein,

also $f_{\mu C} = 2f_{sensor} = 600Hz$, um zu gewährleisten, dass alle Low-High sowie High-Low Übergänge des empfangenen Sensorsignals detektiert werden.

### 2.1.5 Steuerung der HW mit AVR Controller Board

In den folgenden Kapiteln wird beschrieben, wie mit Hilfe des AVR Controller Boards einzelne HW-Komponenten angesteuert werden.

#### 2.1.5.1 Motoransteuerung

Die Betriebsspannung des eingesetzten Motors liegt bei 12 V/DC Modelcraft. Daher kann der Motor nicht direkt an das Controller Board angeschlossen werden, das mit 5 V/DC betrieben wird. Um den Motor durch den Mikrocontroller ansteuern zu können, wird zwischen diesen beiden Einheiten ein zusätzliches Modul geschaltet (s.Abb. 2.12).

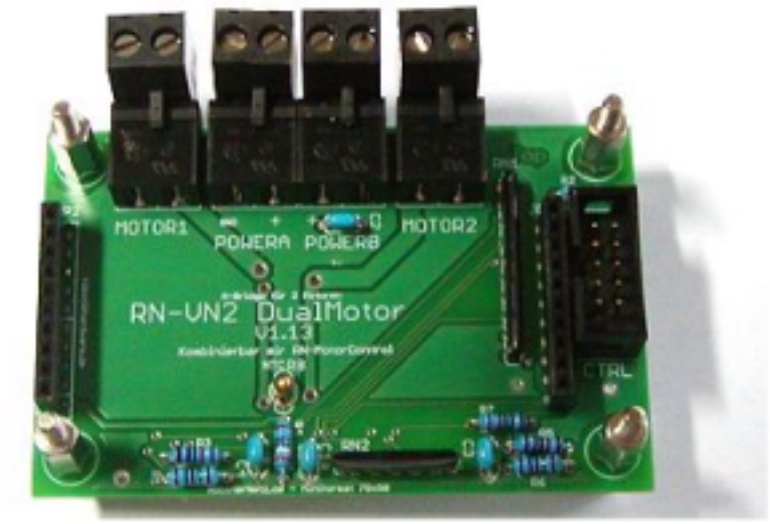



Abbildung 2.12: Motoransteuerung

Das RN-VNH2 Dualmotor Modul RoboterNetz.de bietet die Möglichkeit, über einen Mikrocontroller die Drehzahl und die Drehrichtung von 1 oder 2 Gleichstrommotoren zu regeln. Die Betriebsspannung des Moduls liegt bei ca. 6 bis 16V - ideal ca. 12-13 V. Die Geschwindigkeit ist über einen PWM (Pulsweitenmodulation)-Port, der bis zu 20 kHz liefern darf, regelbar.

#### 2.1.5.2 Ansteuerung der Peripheriegeräte

Zur Triggerung der Kamera und des Stroboskops ist ein Controllerpin pro Gerät notwendig, von dem das Triggersignal zu definierten Zeitpunkten an die Peripheriegeräte gesendet wird. Da durch die Voreinstellung die Controllerpins als Eingänge definiert sind, müssen die beiden Trigger-Pins auf Ausgang geschaltet werden. Um das Triggersignal zu erzeugen, legt man an den Mikrocontroller-Ausgängen ein High-Pegel an. Laut der Dokumentation Atmel S.361 unterschreitet der minimale High-Pegel nicht den Wert von 4,2 V, so dass sowohl die Kamera (notwendiger High-Pegel 3,3 bis 12V) als auch das Stroboskop (3 bis 12V) angesteuert werden können. Wie lange der High-Pegel anliegen soll, damit der Triggersignal empfangen werden kann, ist bei beiden Geräten unterschiedlich. Bei der Kamera muss der High-Pegel mindestens 4,8 $\mu$s am Triggereingang anliegen TheImagingSource, um sie auszulösen. Das Stroboskop benötigt mindestens ein 30$\mu$s langes High-Pegel Drello.

#### 2.1.5.3 Anschluss der Sensoren an AVR Controller Board

Die beiden in der Bachelor-Arbeit verwendeten Sensoren (Entfernungssensor und Inkrementalgeber) werden über Anschlüsse am AVR Controller Board gleichermaßen wie die beiden Peripheriegeräte jeweils mit einem Controllerpin verbunden. Die Auswertung der analogen Eingangssignale des Entfernungssensors übernimmt der im Mikrocontroller integrierter Analog-Digital-Konverter (ADC). Der Inkrementalgeber sendet digitale Signale an den Mikrocontroller.

#### 2.1.5.4 Kommunikation mit der Benutzersteuerung

Zur Kommunikation des AVR Controllers mit der Benutzersteuerung auf dem PC stellt das AVR Controller Board u.a. zwei RS232-Anschlüsse zur Verfügung.

## 2.2 Software

### 2.2.1 Mikrocontrollerprogrammierung

In folgenden Kapiteln ist die Vorgehensweise bei der Implementierung erforderlicher Funktionalitäten auf dem Mikrocontroller (s.Kap. 1.2.1) dargestellt.

#### 2.2.1.1 Motoransteuerung mit PWM

Bei der Motoransteuerung möchte man die Drehzahl des Motors variieren können. Mit AT90CAN128 kann man das mit Hilfe der Pulsweitenmodulation (PWM) lösen. Bei PWM wird die Ein- und Ausschaltzeit eines Rechtecksignals bei fester Grundfrequenz variiert. Das

Verhältnis $t_{ein} / (t_{ein} + t_{aus})$ bezeichnet man als Tastgrad (Trampert (2003a) s.106). In der Abb. 2.13 ist der Tastgrad das Verhältnis $Pulsbreite / Periode$. Durch das RN-VNH2 Dualmo-

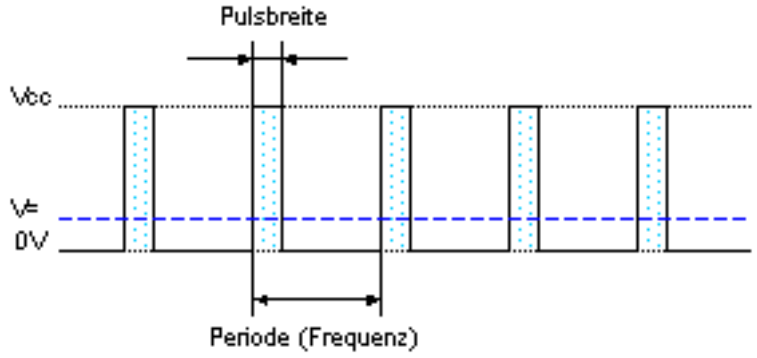



Abbildung 2.13: Pulsweitenmodulation

tor Modul 2.12 wird die pulsierende Ausgangsspannung in eine entsprechende Gleichspannung umgewandelt. Die Dauer der Einschaltzeit des Rechtecksignals (Duty Cycle) bestimmt die Drehzahl des Motors. Um festzulegen, mit welcher Frequenz f das PWM-Signal erzeugt werden soll, sind Testsläufe mit verschiedenen Signal-Frequenzen notwendig. Als Entscheidungskriterium kann die Genauigkeit der Geschwindigkeitsmessung bei verschiedenen Frequenzen dienen. Die Dauer der einzelnen Duty Cycles ist von den zulässigen Tiefst- bzw. Höchstwerten der Rotationsgeschwindigkeit (s.Tabelle 1.1) abhängig.

Die Aufgabe des Mikrocontrollers ist die Generierung eines PWM-Signals mit der Frequenz f, mit dem man die Rotationsgeschwindigkeit des Drehtellers durch die Änderung des Duty Cycles variieren kann.

#### 2.2.1.2 Triggerung von Kamera und Stroboskop

Wie bereits im Kapitel 2.1.5.2 dargestellt, muss an die Kamera und an das Stroboskop ein kurzes Signal gesendet werden, um sie triggern zu können. An die zwei Ausgangspins des AVR Controllers, die jeweils durch eine physikalische Verbindung mit den Triggereingängen der Geräte verbunden sind, legt man für eine Zeitperiode $t_{high}$ ein High-Pegel an.

#### 2.2.1.3 Verarbeitung der Sensorsignale

**Drehzahlmessung mit Inkrementalgeber**

Die Drehzahlmessung findet zwischen zwei Zeitpunkten $t_0$ und $t_1$ statt. In der Zeitperiode $\Delta t = t_1 - t_0$ treffen beim Mikrocontroller durch die Verbindung zum Inkrementalgeber $n$ Signalübergänge ein. Jeder Signalübergang (sowohl High-Low Übergang als auch Low-High

Übergang) bedeutet, dass der Drehteller sich um einen Winkel $\varphi_1 = \pi/600$ (600 - die Anzahl der Signalübergänge pro Umdrehung) bewegt hat. Die Rotationsgeschwindigkeit lässt sich nach der folgenden Formel berechnen:

$$\omega = \frac{n\pi/600}{\Delta t} \tag{2.2}$$

Die minimale Abtastfrequenz der Sensorsignale kann nach der im Kapitel 1.1.4 aufgestellten Formel 1.6 berechnet werden. Bei 600 Signalübergängen pro Umdrehung und maximaler Rotationsgeschwindigkeit $\omega_{max} = 0,5rps$ gilt folgende Ungleichung:

$$f_{min\ Abtast} > 2 \cdot 600 \cdot 0,5rps = 600Hz \tag{2.3}$$

**Objekterkennung mit Entfernungssensor**

Der Entfernungssensor kann bis maximal 30 Messungen 2.1.4 in einer Sekunde durchführen. Der zeitliche Abstand zwischen zwei Objekten kann auf $1/30\ s$ minimiert werden. Bei der maximalen Rotaionsgeschwindigkeit von 0,5 rps (s.Tabelle 1.1) braucht der Drehteller $t_{minUmdrehung} = 1/0,5rps = 2s$ für eine Umdrehung. Die maximale Anzahl der Objekte, die theoretisch detektiert werden kann, beträgt $n_{max} = 2s/1/30s = 60$ Objekte.

Für die Anforderungen dieser Bachelor-Arbeit ($n_{Objekte} = 4$) ist es ausreichend, wenn man nur 2 Messungen des Sensors pro Sekunde bearbeitet, da der minimale zeitliche Abstand zwischen 2 Objekten auf dem Drehteller $t_{distance} = 2/n_{Objekte} = 2/4 = 0,5$s (s. Formel 2.1) beträgt.

Der Entfernungssensor wird direkt an den Eingang des in den AVR eingebauten Analog-Digital-Wandlers (ADC) angeschlossen. Der ADC wird in der Betriebsart „Frei laufend“ (engl. Free Running) betrieben, d.h. der Wandler erfasst permanent die anliegende Spannung und schreibt diese in das ADC Data Register. Der ADC benötigt einen eigenen Takt, welchen er sich selber aus der CPU-Taktfreqenz erzeugt. Der ADC-Takt sollte zwischen 50 und 200kHz liegen www.mikrocontroller.net (2007). Wenn man also $n$ Messungen des Entfernungssensors pro Sekunde haben möchte, liest man $2n$ mal pro Sekunde das Datenregister des ADC aus.

#### 2.2.1.4 Erfassung der Objekte

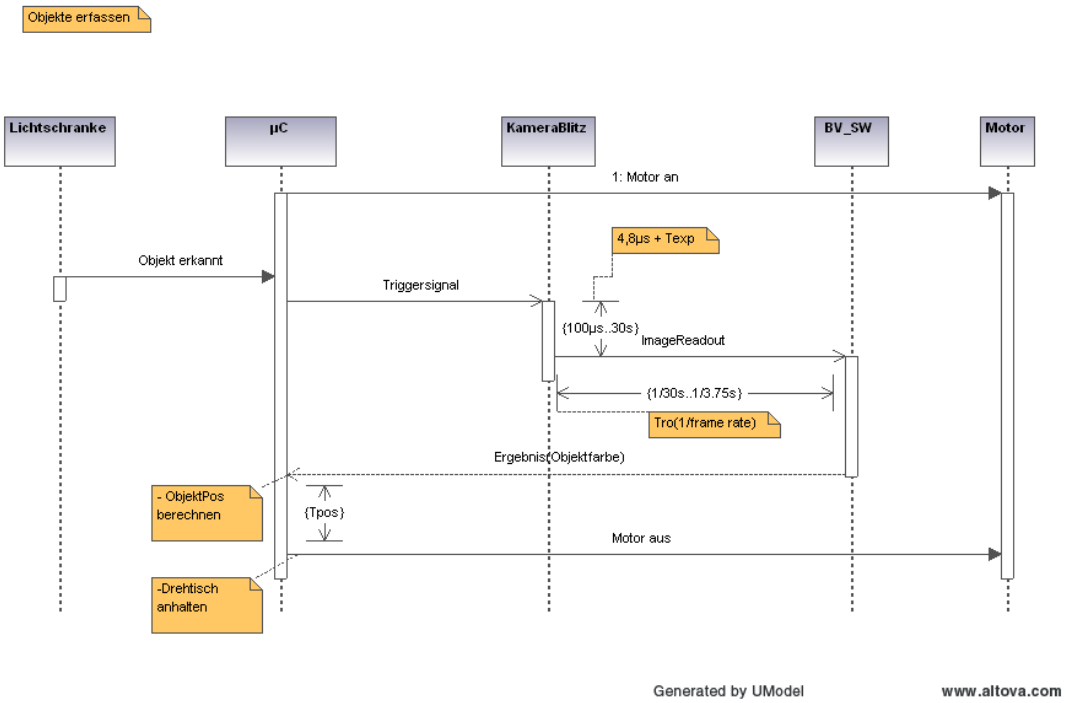
Nachdem die Objekte vom Entfernungssensor im Zeitpunkt $t_0$ erkannt werden, vergeht Zeit $t$, bis sie die Kamera und das Stroboskop erreicht haben. Die Zeit $t$ muss berechnet werden, um den Zeitpunkt der Triggerung der Geräte feststellen zu können. Der Winkel zwischen dem Entfernungssensor und der Kamera ist $\varphi$ rad. Die aktuelle Rotationsgeschwindigkeit $\omega$ kann nach der Formel 2.2 ermittelt werden. Die Zeit t lässt sich folgendermaßen berechnen:

$$t = \varphi/\omega \tag{2.4}$$

#### 2.2.1.5 Bearbeitung und Positionierung der Objekte

Die Bearbeitung der durch die Kamera erfassten Objekte erfolgt durch ein SW-Modul, das als eigenständiges Programm oder als ein Teil der Benutzersteuerung implementiert werden kann. Die Aufgabe des SW-Moduls besteht darin, die Farbe der ankommenden Bilder festzustellen und die Information an die Drehtisch-Steuerung (Mikrocontroller) weiterzugeben.

Die Vorgehensweise bei der Erfassung und der anschließenden Bearbeitung der Objekte wird im folgenden Sequenzdiagramm demonstriert:




Im Sequenzdiagramm kann man erkennen, dass die Zeit $t_{ObjektPosition}$, die zwischen der Detektion eines Objektes und der Bestimmung seiner Farbe vergeht, aus mehreren Zeitabschnitten besteht:

$t_1$ - Der Zeitabschnitt $t_1$ zwischen der Detektion des Objektes und dem Auslösen der Kamera kann nach der Formel 2.4 berechnet werden.

$t_{Exp}$ - Die Belichtungszeit $t_{Exp}$ der Kamera ist zwischen $100\mu$s und 30s einstellbar.

$t_{ro}$ - Die Zeitdauer der Bildausgabe (Image readout) bestimmt sich aus dem Kehrwert der eingestellten Bildrate (s. Abb. 2.7).

$t_{bv}$ - Die Zeit, die die Bildverarbeitungs-SW zur Bestimmung der Farbe eines Objektes benötigt.

$t_{message}$ Die notwendige Zeit zur Übertragung einer Nachricht(hier: eine Nachricht vom Typ „NEW_COLOR_VALUE“).

Sind die einzelnen Zeitabschnitte bekannt, kann durch die Bildung ihrer Summe $t_{ObjektPosition} = t_1 + t_{Exp} + t_{ro} + t_{bv}$ und der Einbeziehung der aktuellen Rotationsgeschwindigkeit die aktuelle Objektposition berechnet werden. In der Abbildung 2.14 ist diese Position mit „C“ gekennzeichnet.

Der Benutzer kann die Drehtisch-Steueurung veranlassen, ein Objekt in der vorgegebenen Position anzuhalten. Dazu wählt er in der Benutzersteuerung über den Menupunkt „SelectColor“ (s. Kap. 3.2) eine Farbe aus. Die Position wird durch einen Bereich des Drehtellers bestimmt. In der Abb. 2.14 ist der Bereich durch den schwarz markierten Sektor dargestellt.

Anm.: der markierte Sektor „bewegt“ sich nicht bei der Rotation des Drehtellers. Die Markierung in der Abbildung dient zur Veranschaulichung der vorgegebenen Position.

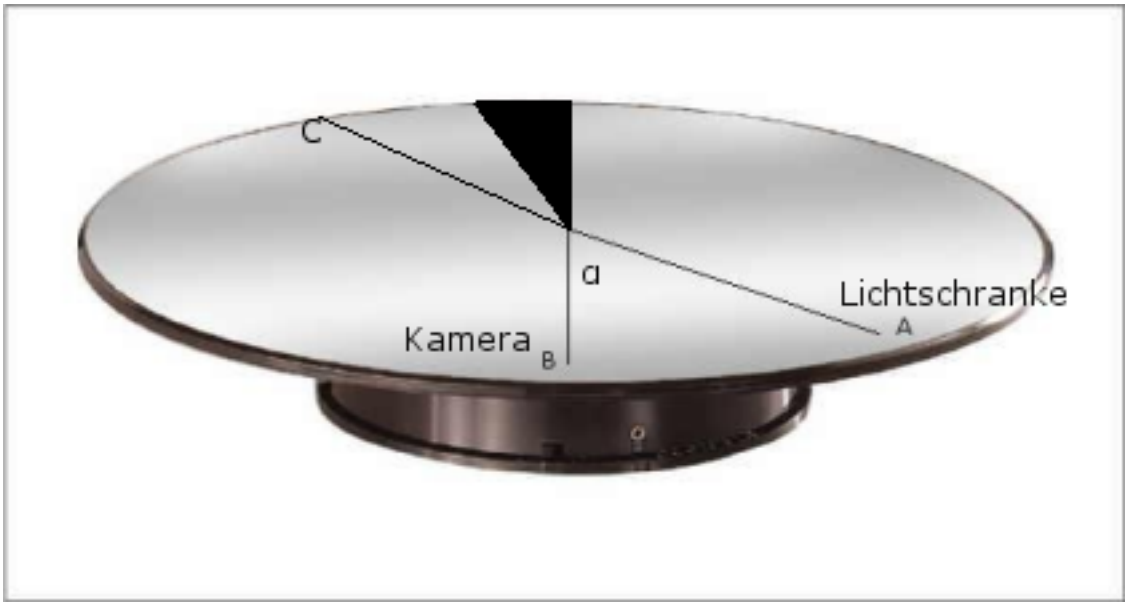



Abbildung 2.14: Positionierung der Objekte

Nach der Berechnung der Position „C“ kann die Zeit ($T_{pos}$ im Sequenzdiagramm 2.2.1.5) festgestellt werden, die das Objekt benötigt, um den Winkel zwischen „C“ und der Zielposition zurückzulegen. Ist die Zeit $T_{pos}$ nach der Bestimmung der Farbe des Objektes abgelaufen und stimmt die Farbe des Objektes mit der Farbe, die der Benutzer ausgewählt hat, überein, hält die Drehtich-Steuerung den Drehteller an. Beim Anhalten des Drehtellers befindet sich das Objekt im markierten Sektor (vorausgesetzt, der Drehteller hat keinen Nachlauf).

### 2.2.2 Entwicklung der Benutzersteuerung

Der Hersteller TheImagingSource (2007) der in dieser Arbeit eingesetzten Kamera stellt eine Demo-Applikation zur Verfügung, mit der man die aufgenommenen Bilder anzeigen kann. Diese Applikation dient als Grundlage der Benutzersteuerung.

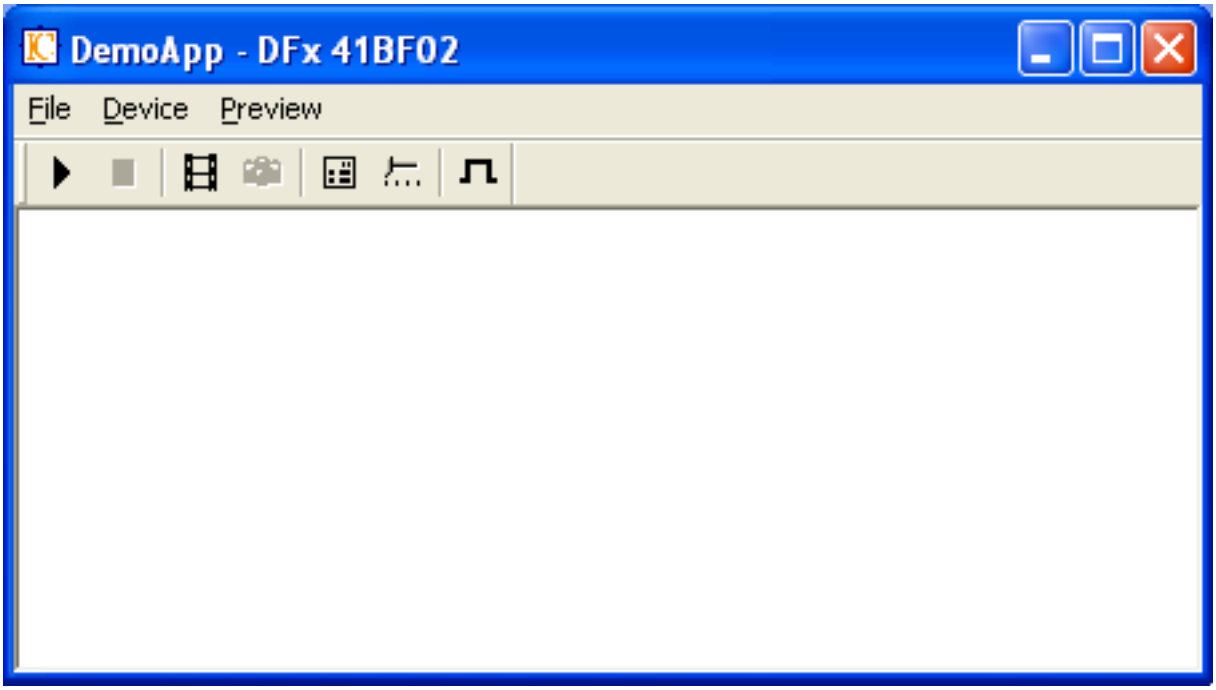



Abbildung 2.15: Demo Applikation

Die Software wird um folgende Funktionalitäten erweitert:

- RS232-Schnittstelle zur Kommunikation mit dem Mikrocontroller
- zusätzliche Menu-Punkte in der GUI zur Eingabe der Benutzerbefehle
- Bildverarbeitungsmodul (wird im Rahmen einer anderen Bachelor-Arbeit implementiert)

### 2.2.3 Kommunikationsprotokoll

In folgenden Ausführungen wird anhand der im Kapitel 1.2.3 definierten Anforderungen ein Übertragungsprotokoll definiert, das die Regelung der Kommunikation zwischen User-Interface (PC) und Drehtisch-Steuerung (AVR Mikrocontroller) übernimmt.

Auf den Abbildungen 2.16 und 2.17 ist ein Beispiel der Kommunikation zwischen PC und AVR in einem Sequenzdiagramm dargestellt. In der Nachricht vom Typ „NEW_PICTURE“

teilt die Drehtisch-Steuerung dem User-Interface mit, dass ein neues Bild aufgenommen wurde. Konnten die von der Kamera übertragenen Bilddaten in der Zeitspanne zwischen dem Erhalt der ersten Nachricht und dem Versand der Antwortnachricht ausgewertet werden (s. Abb. 2.16), werden die Ergebnisse mit der Nachricht vom Typ „NEW_COLOR_VALUE“ an AVR gesendet.

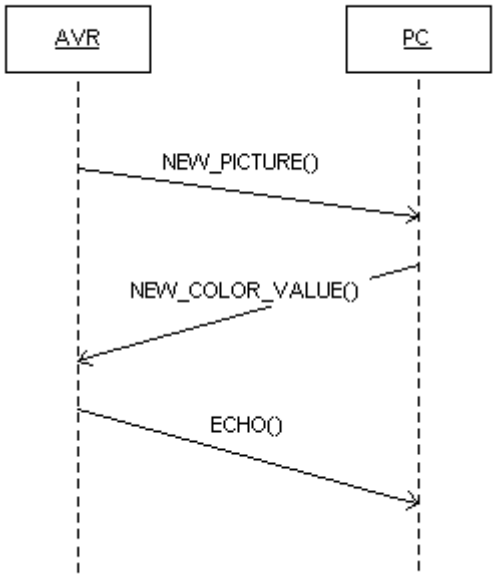



Abbildung 2.16: Kommunikation zwischen AVR und PC Bsp.1

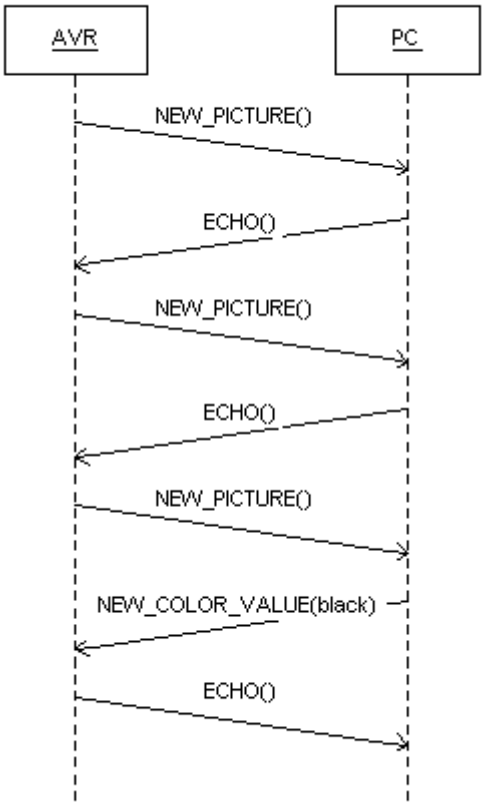



Abbildung 2.17: Kommunikation zwischen AVR und PC Bsp.2

Die Drehtisch-Steuerung bestätigt den Erhalt der Ergebnisse der Auswertung mit einer Nachricht vom Typ „ECHO“.

Im Sequenzdiagramm in der Abbildung 2.17 ist die Situation dargestellt, in der der PC nach dem Erhalt der Nachricht „NEW_PICTURE“ vorerst keine Auswertungsergebnisse mit der Antwortnachricht versenden kann. Auf die Nachricht vom AVR reagiert PC daher mit einer „ECHO“-Nachricht. Da die Drehtisch-Steuerung eine Nachricht vom Typ „NEW_COLOR_VALUE“ erwartet und anstatt dessen eine Antwort vom Typ „ECHO“ erhält,

sendet sie die „NEW_PICTURE“ - Nachricht solange an den PC, bis sie die Auswertungsergebnisse erhält.

Die Anzahl der „NEW_PICTURE“-Nachrichten, die der Mikrocontroller an den PC versenden muss, bis die Auswertungsergebnisse vorliegen, hängt davon ab, wie schnell die Bilddaten von der Kamera an den PC übertragen werden.

#### 2.2.3.1 Architektur

Das Übertragungsrotokoll basiert auf einem für die Kommunikation mit eingebetteten Systemen entwickelten Protokoll Ibrt (2007). Das Protokoll ist nicht darauf ausgelegt, große Datenmengen zu transportieren. Die maximale Länge einer Nachricht beträgt 255 Bytes. Im Datenblock einer Nachricht werden überwiegend Statusmeldungen(wie z.B. aktuelle Drehzahl oder Farbe des Objektes) übertragen.

In jeder Nachricht ist ein Header mit zusätzlichen Informationen (wie z.B. Sender-, Empfänger-ID und Befehlscodes) enthalten. Durch die Befehlscodes, welche im Vorfeld zwischen den Applikationen (Drehtisch-Steueurung und User-Interface) vereinbart werden, werden Aktionen (wie z.B. „Drehtisch starten“) angestoßen.

#### 2.2.3.2 Protokollregeln

Die Kommunikation zwischen beiden Modulen läuft im Halbduplex-Modux ab, d.h. zur gleichen Zeit kann nur in einer Richtung gesendet oder empfangen werden. Beide Module sind gleichberechtigt, jedes Modul kann entweder als ein Sender oder als ein Empfänger agieren. Nach dem Versenden einer Nachricht wartet der Sender auf eine Antwort des Empfängers, bevor er die nächste Nachricht senden kann (synchrone Kommunikation). Das Sicherstellen einer fehlerfreien Übertragung wird durch Überprüfung der Prüfsumme garantiert.

Da es keine große Datenmengen zu übertragen gibt, sondern fast ausschließlich Befehlscodes und Statusmeldungen, werden für den Datenblock einer Nachricht 4 Bytes reserviert. Mit den dazugehörigen Header-Feldern 2.2.3.3 kann eine Nachricht maximal 11 Bytes lang werden.

Die notwendige Zeit zur Übertragung einer Nachricht kann nach der im Kapitel 1.2.3.2 aufgestellten Formel 1.8 berechnet werden. Bei Baudrate von 9600 Bits/s ist $t_{Message} = 11Bytes \cdot 10/9600Baud = 11ms$.

Die zeitliche Anforderung an das Übertragungsprotokoll (s. Formel 1.9) verlangt, dass der zeitliche Abstand zwischen zwei Objekten $t_{Distance}$ nicht den Wert $t_{Message}$ unterschreiten darf. Die Bedingung ist erfüllt, da $t_{Distance}$ minimal $1/30s \approx 33ms$ 2.1.4 betragen kann.

#### 2.2.3.3 Aufbau einer Nachricht

Der Sender und der Empfänger tauschen gegenseitig Nachrichten aus. Eine Nachricht besteht aus einem Header und einem Datenblock.

**Header**
Das erste Feld des Headers ist das STX-Byte, das im Vorfeld zwischen beiden Kommunikationsteilnehmern vereinbart wird. Mit Hilfe dieses Wertes kann der jeweilige Empfänger den Anfang einer Nachricht erkennen. Im nächsten Feld des Headers wird die Länge einer Nachricht übertragen, die davon abhängig ist, wie lang der aktuelle Datenblock der Nachricht ist. Im Gegensatz zum Header, dessen Länge unabhängig vom Typ der einzelnen Nachrichten konstant bleibt, variiert die Länge des Datenblocks.

In den Feldern SRC und DST werden jeweils der Absender und der Empfänger der Nachricht übertragen. Im Feld Cmd steht der Befehlscode (s. Tbl. 2.3) der jeweiligen Nachricht.

**Datenblock**
Der Datenblock einer Nachricht besteht aus 0 bis 5 Bytes. Die Länge des Datenblocks ist vom Typ der einzelnen Nachrichten abhängig. In der Nachricht vom Typ „COLOR_CHOICE“ wird ein Datenbyte zur Übertragung des Farbencodes beansprucht. In der Nachricht vom Typ „NEW_PICTURE“ sendet die Drehtisch-Steuerung den aktuellen Wert der Drehzahlmessung in einem Datenblock von 3 Bytes an das User-Interface.

Jede Nachricht wird durch die Checksumme, die 2 Bytes lang ist, abgeschlossen. In der Tabelle 2.2 ist der Aufbau der Nachrichten dargestellt:

| Feld | **STX** | **Len** | **SRC** | **DST** | **Cmd** | **[Data]** | **CRC** |
|---|---|---|---|---|---|---|---|
| Länge | 1Byte | 1Byte | 1Byte | 1Byte | 1Byte | 0-4 Bytes | 2Byte |

Tabelle 2.2: Aufbau einer Nachricht

**Beschreibung der einzelnen Felder:**

- STX - Anfang einer Nachricht(0x02)
- Len - Länge einer Nachricht
- SRC - Sender-ID
- DST - Empfänger-ID
- Cmd - Befehl(vordefinierter Code)
- [Data] - Datenblock, Länge 0-4 Bytes

- CRC - Checksumme

#### 2.2.3.4 Arbeitsweise des Protokolls

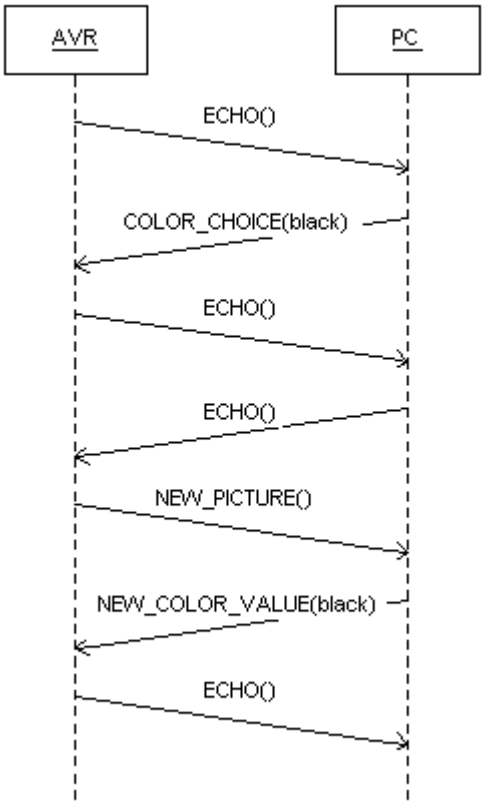
Wird ein bestimmter Befehl und dazugehörige Daten an eins der Module verschickt, werden die Felder „Cmd“ und „[Data]“ vom Sender mit entsprechenden Daten gefüllt. Soll z.B. Echo-Nachricht vom AVR Controller Board an den PC versandt werden, wird in das „Cmd“ - Byte des Sendepuffers der Code für „Echo“ - Befehl (hier 0x00) eingetragen. Die „Echo“-Nachrichten übertragen wie alle anderen Nachrichten, die vom AVR an den PC gehen, den aktuellen Wert der Drehzahlmessung. Daher werden die Daten-Bytes einer „Echo“ - Nachricht immer ausgewertet.

Weitere Befehlscodes, die sowohl von der Hardware Steuerung (AVR Modul) als auch von der Benutzersteuerung (PC Modul) akzeptiert werden müssen, sind mit Bezeichnung und kurzer Beschreibung in folgender Tabelle aufgelistet:

| **command code** | **name** | **meaning** |
|---|---|---|
| 0x00 | ECHO | |
| 0x01 | NEW_PICTURE | Ein Objekt passierte die Lichtschranke $\mu C \rightarrow PC$ |
| 0x02 | NEW_COLOR_VALUE | Farbe des Objektes wurde ermittelt $PC \rightarrow \mu C$ |
| 0x03 | NEW_V_VALUE | Benutzer ändert die Geschwindikeit $PC \rightarrow \mu C$ |
| 0x04 | COLOR_CHOICE | Benutzer wählt eine Farbe aus $PC \rightarrow \mu C$ |
| 0x05 | START | Benutzer startet den Drehtisch $PC \rightarrow \mu C$ |
| 0x06 | STOP | Benutzer stopt den Drehtisch $PC \rightarrow \mu C$ |

Tabelle 2.3: Befehlscodes

Auf der Abbildung 2.18 ist ein Sequenzdiagramm zu sehen, das einen Kommunikationsvorgang zwischen beiden Modulen demonstriert.




In der Benutzeroberfläche wählt man eine Farbe aus. Mit der nächsten Nachricht an den AVR wird der Befehlscode „COLOR_CHOICE“ und der dazugehörige Parameter (Farbe=„black“) versendet.

Ein neues Objekt wird vom $\mu$C detektiert. Die Bilddaten werden über Fire-Wire Verbindung an den PC transportiert. Über das Protokol sendet AVR eine „NEW_PICTURE“ - Nachricht an den PC. Nachdem die Bilddaten ausgewertet wurden, sendet PC eine „NEW_COLOR_VALUE“ - Nachricht mit dem Farbenwert als Parameter an den AVR. Stimmt der Wert mit dem Parameter aus der „COLOR_CHOICE“ - Nachricht überein, wird das aufgenommene Objekt an die Zielposition gebracht.

Abbildung 2.18: Sequenzdiagramm(COLOR_CHOICE)

#### 2.2.3.5 Zusammenfassung

Die Analyse der einzelnen Nachrichten, die AVR und PC untereinander austauschen, ergibt, dass für die Länge des Datenblockes einer Nachricht 4 Bytes ausreichend sind.

# Kapitel 3

# Realisierung

In diesem Kapitel wird beschrieben, wie die einzelnen SW-Module (Drehtisch-Steuerung, User-Interface und das Kommunikationsprotokoll) implementiert werden.

## 3.1 Implementierung der Mikrocontroller-Software

Das Programm zur Steuerung der Hardware wird in der Programmierprache C unter Einsatz der Entwicklungsumgebung AVR Studio 4.12 entwickelt. Der von der Entwicklunsumgebung erzeugte Maschinencode wird über die Programmierschnittstelle JTAG in den Speicher des Mikrocontrollers übertragen.

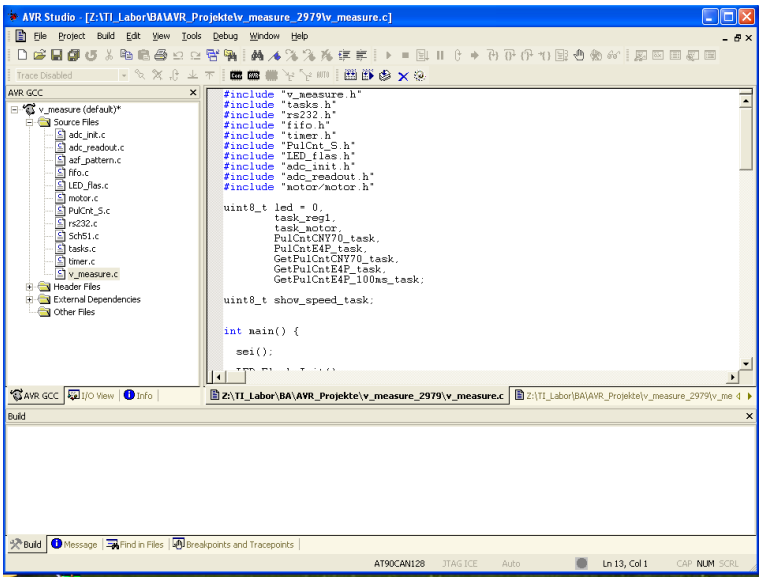



Abbildung 3.1: Screenshot der AVR-Entwicklungsumgebung

Die notwendige Zeitsteuerung der einzelnen Funktionen des Mikrocontrollers wird mit Hilfe eines kooperativen Schedulers implementiert.

### 3.1.1 Kooperativer Scheduler

Ein Scheduler kann als ein Betriebssystem angesehen werden, dass einzelne Funktionen (engl. Tasks) in definierten Zeitabständen oder einmalig aufrufen kann. Er wird in Form einer ISR (Interrupt Service Routine) implementiert, die verschiedene Tasks nacheinander (bei Einprozessorsystemen) ausführen kann. Es muss ein Timer initialisiert werden, um die ISR in bestimmten Zeitabständen (Timer Ticks) anzustoßen (vgl. s. 245ff Pont (2001)).

In dieser Bachelor-Arbeit wird ein kooperativer Scheduler verwendet, der auf ein Einprozessorsystem ausgelegt ist. Der Quellcode des Schedulers ist aus dem Buch von Michael J. Pont übernommen worden Pont (2001).

#### 3.1.1.1 Funktionsweise

Alle Funktionen, die in periodischen Abständen ausgeführt werden sollen, werden als Scheduler-Tasks definiert. Die Tasks werden „gescheduled“, d.h. jeder Task wird ihr Startzeitpunkt und ihre Ausführungsart (periodisch oder einmalig) zugewiesen. Wird der Startzeitpunkt einer Task erreicht, wird sie in die Liste der wartenden Tasks eingetragen. Sobald die CPU frei ist, wird die als erste in die Liste eingetragene Task ausgeführt. Nach der kompletten Ausführung der Task übernimmt der Scheduler wieder die Kontrolle.

#### 3.1.1.2 Implementation

Der Scheduler ist in drei Dateien implementiert (s. Anhang C.2), die ein Teil des AVR Projektes bilden.

Er besteht aus folgenden Funktionen:

**SCH_Add_Task** Fügt in den Scheduler eine neue Task ein

**SCH_Delete_Task** Entfernt eine Task aus dem Scheduler

**SCH_Dispatch_Task** Startet eine Task, sobald sie zur Ausführung bereit ist

**SIGNAL(SIG_OUTPUT_COMPARE0)** Die Interrupt Service Routine dient zur periodischen Aktualisierung des Schedulers. Die Periodizität wird durch die Initialisierung des Timers festgelegt.

Der zeitliche Abstand zwischen zwei Aktualisierungen des Schedulers wird als Scheduler Tick bezeichnet. Alle Tasks, die nach einer Aktualiserung des Schedulers ihr Startzeitpunkt erreicht haben, müssen während des nächsten Scheduler Ticks komplett ausgeführt werden.

### 3.1.2 Implementierung der Mikrocontroller-Software mit Hilfe des kooperativen Schedulers

An die Mikrocontroller-Software werden folgende Aufgaben gestellt, die in regelmäßigen Abständen bearbeitet werden müssen:

- Detektion der Objekte
- Drehzahlmessung
- RS232-Kommunikation

Die einzelnen Aufgaben werden durch periodisch auszuführende Tasks abgearbeitet. Sie werden in definierten Zeitabständen durch den Scheduler gestartet. Müssen in einem Scheduler Tick mehrere Tasks ausgeführt werden, werden sie nach ihrer Position in der Warteliste nacheinander aufgerufen.

#### 3.1.2.1 Timer-Initialisierung

In der Funktion „timer.c“ (s. Anhang C.2) wird die Initialisierung des Timers vorgenommen, durch den die ISR zur Aktualisierung des Schedulers angestossen wird. Die Aktualisierungshäufigkeit des Schedulers hängt davon ab, wie oft die einzelnen Tasks aufgerufen werden müssen.

In der Tabelle 3.1 sind die in der Aufzählung 3.1.2 formulierten Aufgaben mit dazugehörigen Funktionen (Quellcode auf der beigelegten CD), die als Scheduler Tasks implementiert werden, dargestellt.

| **Aufgabe** | **Scheduler Task** | **Periode in ms** |
|---|---:|---:|
| Detektion der Objekte | | |
| | adc_readout() | 16 |
| Drehzahlmessung | | |
| | Poll_Count_E4P() | 1 |
| | Get_Count_E4P() | 1000 |
| RS232 Kommunikation | | |
| | tReg() | 1 |
| | ComLoop() | 11 |

Tabelle 3.1: Häufigkeit der Aufrufe einzelner Scheduler-Tasks

Die Perioden der einzelnen Tasks werden durch die im Kapitel 2.2.1aufgestellten Designanforderungen an die Aufgaben der Mikrocontroller-Software bestimmt.

Die am häufigsten auszuführenden Tasks sind die Funktionen „tReg()“ zum Senden bzw. Empfangen von Daten über RS232-Schnittstelle und die Funktion „Poll_Count_E4P“ zum

Abtasten der Signalflanken des Inkrementalgebers mit Periode P = 1 ms. Demzufolge muss der Timer mindestens jede ms zurückgesetzt werden.

In folgenden Kapiteln werden die einzelnen als Scheduler Tasks implementierten Funktionen beschrieben.

#### 3.1.2.2 Detektion der Objekte

**adc_readout()** Zur Detektion der Objekte wird der Entfernungssensor an den Eingang des Analog-Digital Wandlers (ADC) angeschlossen. Der Sensor liefert 30 Messungen in einer Sekunde 2.2.1.3. Die Funktion „adc_readout()“ zum Auslesen des Datenregisters des ADC wird als eine Scheduler Task definiert. Die Periode der Task wird durch die doppelte Anzahl (s. Nyquist-Shannonsche Abtasttheorem Bernstein (2007) s.72) der Messungen des Sensors bestimmt: $P = 1/2 \cdot 30\ s \approx 16ms$.

#### 3.1.2.3 Drehzahlmessung

Die Messung der Rotationsgeschwindigkeit erfolgt durch die Kombination von zwei Funktionen. Die Funktion „Poll_Count_E4P()“ tastet sowohl die fallenden als auch die steigenden Signalflanken des Inkrementalgebers ab und die Funktion „Get_Count_E4P()“ gibt den Stand des Flankenzählers, der alle 1000 ms erfasst wird, zurück.

**Poll_Count_E4P()** Der Inkrementalgeber wechselt die Signalzustände mit der Frequenz f = 600 Hz 2.3. Die Periode P der Task muss mit Berücksichtigung des o.g. Abtasttheorems demzufolge $P = 1/2 \cdot 600\ s \approx 0,8ms$ betragen. Für den Fall ist der Scheduler Tick von 1ms zu lang. Ein kürzerer Scheduler Tick kann theoretisch mit einem 16 Bit Timer im Gegensatz zum verwendeten 8 Bit Timer realisiert werden. Die Worst-Case-Analyse 3.1.3 zeigt aber, dass der Scheduler Tick nicht kürzer als 1ms sein darf.

Die fehlerfreie periodische Ausführung aller Scheduler Tasks hat höhere Priorität für diese Bachelor-Arbeit als die Tatsache, dass alle Signalflanken des Inkrementalgebers detektiert werden. Die Periode der Task wird daher mit 1 ms und nicht mit 0,8 ms initialisiert.

**Get_Count_E4P()** Die Funktion „Get_Count_E4P()“ wird mit Frequenz f aufgerufen und liefert somit die Anzahl der Signalflanken des Inkrementalgebers, die in P = 1/f s detektiert worden sind. Um die Rotationsgeschwindigkeit rad/s berechnen zu können, wird die Periode P mit 1000 ms initialisiert.

#### 3.1.2.4 RS232-Kommunikation

Das Senden und Empfangen der Daten über die serielle Schnittstelle wird mit Hilfe von zwei folgenden Funktionen realisiert. Die Funktion „tReg()“ dient dazu, um die Daten zeichenweise

in das UART-Schieberegister zu schreiben bzw. daraus zu lesen. Die Funktion „ComLoop()“ ist in Form eines Automaten realisiert (s.Abb. 3.4). In der Funktion werden die vom User-Interface empfangenen Nachrichten verarbeitet und die Antwortnachrichten zusammengestellt.

**tReg()** Bei der Übertragungsrate von 9600 Baud braucht die serielle Schnittstelle nach der Formel 3.1 $\approx 1ms$ zum Empfang bzw. Versand eines Zeichens. Die Funktion „tReg()“ wird demzufolge mit Periode P = 1 ms „gescheduled“.

**ComLoop()** Die Nachrichten, die Drehtisch-Steuerung und das User-Interface sendet bzw. vom User-Interface empfängt, sind 11 Zeichen lang. D.h. bei der im Protokoll implementierten Halb Duplex Kommunikation sendet bzw. empfängt der Mikrocontroller alle 11 ms (SW-Verzögerung des PC nicht eingerechnet) eine Nachricht.

Die Periode der Task zur Bearbeitung von Nachrichten wird mit P = 11 ms initialisiert. Die möglichen Fehler in der Kommunikation zwischen AVR und PC werden im Kapitel 3.3.3 behandelt.

### 3.1.3 Qualitätssicherung

In diesem Kapitel wird die Situation durchgespielt, wenn alle auszuführenden Tasks in einem Scheduler Tick abgearbeitet werden müssen.

Die einzelnen Tasks werden zwar mit unterschiedlichen Perioden „gescheduled“, spätestens aber nach (s. Tbl. 3.1)

$$P_{adc\_readout()} \cdot P_{Poll\_Count\_E4P()} \cdot P_{Get\_Count\_E4P()} \cdot P_{tReg()} \cdot P_{ComLoop()} =$$

$$16 \cdot 1 \cdot 1000 \cdot 1 \cdot 11 = 176000ms \approx 3min$$

trifft die oben beschriebene Situation ein.

Die Messungen der Laufzeit der einzelnen Tasks sind in der Tabelle 3.2 dargestellt:

| **Scheduler Task** | **Laufzeit in $\mu$s** |
|---|---:|
| adc_readout() | 30 |
| Poll_Count_E4P() | 4 |
| Get_Count_E4P() | 518 |
| tReg() | 10 |
| ComLoop() | 310 |
| **Summe** | 872 |

Tabelle 3.2: durchschnittliche Laufzeit der Scheduler-Tasks

Die Summe der Laufzeiten der Tasks überschreitet nicht die Dauer eines Scheduler Ticks:

$$872\mu s = t_{WorstCase} < t_{SchedulerTick} = 1ms$$

Fazit: Alle Scheduler Tasks können beim dargestellten Worst-Case-Szenario komplett ausgeführt werden.

## 3.2 Implementierung des User-Interfaces

Das User-Interface ist in dieser Bachelor-Arbeit, wie bereits erwähnt, ein auf einem PC zu installierendes grafisches Programm zur Ansteuerung des Drehtisches. Der Benutzer kann über die Menu-Punkte in der Oberfläche des Programms Befehle an die Drehtisch-Steuerung auf einem Mikrocontroller senden. Die Kommunikation zwischen dem PC und dem Mikrocontroller läuft über die serielle Schnittstelle.

### 3.2.1 Serielle Kommunikation

Das zur Implementierung des User-Interfaces benutzte SW-Projekt (s. Kap. 2.2.2) wird durch eine Klasse zur seriellen Kommunikation de Klein (2003) erweitert. Die Implementierung der Kommunikation wird im Kapitel 3.3 dargestellt.

### 3.2.2 Eingabe der Benutzerbefehle

Die Eingabe der Benutzerbefehle wird durch die Erweiterung der grafischen Oberfläche realisiert.

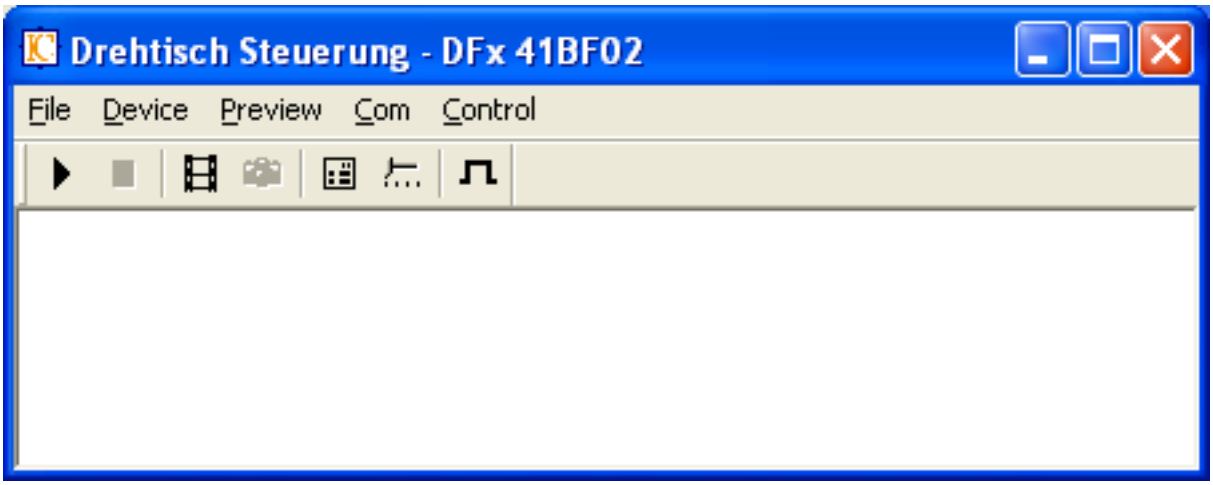



Abbildung 3.2: Benutzersteuerung

Das Menu wird um zwei neue Punkte mit dazugehörigen Unterpunkten erweitert:

- Com

  **Open** serielle Schnittstelle öffnen

  **Close** serielle Schnittstelle schließen

- Control

  **Start** Drehtisch starten

  **VControl** Drehzahl ändern

  **SelectColor** Farbe des zu positionierenden Objektes auswählen

  **Stop** Drehtisch stoppen

## 3.3 Implementierung des Kommunikationsprotokolls

In folgenden Kapiteln wird dargestellt, mit welchen Programmiertechniken der Empfang und der Versand von Nachrichten in der Arbeit implementiert wurden. Man geht darauf ein, welche Kommunikationsfehler passieren und wie sie korrigiert werder können.

### 3.3.1 Serielle Schnittstelle

An beiden Enden der Kommunikation, d.h. im eingebetteten System sowie auf dem PC werden die von der seriellen Schnittstelle empfangenen Daten zeichenweise in einem Zustandsautomaten verarbeitet (Abb.3.4) Ibrt (2007). Die Funktionsweise des Automaten wird im nächsten Kapitel beschrieben.

Der Zustandsautomat wird zwar bei beiden Modulen auf unterschiedliche Art und Weise implementiert, die Funktionsweise des Automaten ändert sich aber nicht.

In der Software im PC wird der Automat als eine eigenständige Funktion implementiert, die beim Eintreten eines von RS232-Schnittstelle getriggerten Events aufgerufen wird (siehe Kapitel 3.2).

Die AVR Software ist in dieser Arbeit nicht interruptgesteuert, wie es üblicherweise der Fall ist, sondern mit Hilfe eines Schedulers(siehe Kapitel 3.1) aufgebaut. Der Zustandsautomat ist demzufolge nicht der Inhalt einer ISR, sondern eine Funktion, die in regelmäßigen Abständen vom Scheduler aufgerufen wird. Bei den folgenden RS232-Einstellungen:

- 9600 Bits/sec
- 8 Data Bit

- Keine Parität
- 1 Stop Bit

braucht serielle Schnittstelle für den Empfang eines Zeichens etwa 1 ms (siehe Formel 3.1).

$$t = \frac{1}{9600 Bits/s} = \frac{1}{9600 Baud} = 1,04 ms \tag{3.1}$$

Also muss die für den Aufruf des Automaten im AVR verantwortliche Scheduler-Task mit einer Periode von T = 1 ms aufgerufen werden, damit keine Daten verloren gehen.

### 3.3.2 Receiving state machine

In diesem Kapitel wird die Funktionsweise des Automaten zum Empfang der Daten von der seriellen Schnittstelle beschrieben.

Der Automat ist dafür zuständig, die einzelnen aus dem über serielle Schnittstelle ankommenden Datenstrom empfangenen Nachrichtenblöcke richtig auszufiltern und im Empfangspuffer zur weiteren Verarbeitung zusammenzusetzen. Die Vorgehensweise soll am folgenden Beispiel demonstriert werden. Der AVR sendet an den PC eine Nachricht mit folgendem Inhalt:

| STX | Len | SRC | DST | Cmd | Data0 | Data1 | Data2 | Data3 | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 0x02 | 0x0A | 0x00 | 0x01 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 2Byte |

Abbildung 3.3: Nachricht in Blöcken

Der Zustandsautomat des Empfängers (siehe Abb. 3.4) ist im Zustand „wait for STX“ (Warten auf den Anfang einer Nachricht). Nach dem Empfang von STX geht der Automat in den Zustand „read Len“ (Auslesen der Nachrichtenlänge) über. Im Beispiel ist $Len = 0x0A$.

Im nächsten Zustand „CountDown“ bleibt der Automat so lange, bis die nächsten Len-2 (abzüglich der Bytes, die STX und Len enthalten) Zeichen empfangen und in den Empfangspuffer eingetragen sind. Erst dann wechselt er in den Zustand „Message received“. Nun ist die Nachricht komplett empfangen und kann vom Programm bearbeitet werden. Der Empfangspuffer wird ausgelesen und für den Empfang der nächsten Nachricht bereitgestellt. Der Automat kehrt in den Anfangszustand zurück.

Sollte der Empfangspuffer überlaufen, während der Automat sich im Zustand „ContDown“ befand, geht er in den Zustand „Buffer full“ (Empfangspuffer ist überlaufen) über. Der Empfangspuffer wird geleert und der Automat geht in den Anfangszustand über.

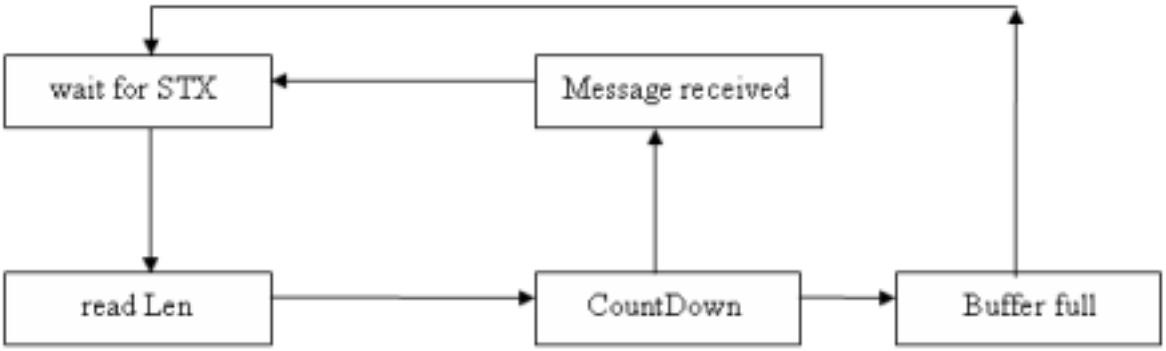



Abbildung 3.4: Zustandsautomat zum Empfang der Daten

### 3.3.3 Empfang von Nachrichten

Ferner verfügen das AVR- und das PC-Modul über eine ähnlich aufgebaute Funktion, die die vom Zustandsautomaten zusammengesetzten Nachrichten bearbeitet. Der Quellcode der Funktionen befindet sich im Anhang: siehe für AVR-Modul den Abschnitt C.1.1 und für PC-Projekt den Abschnitt C.1.2.

#### 3.3.3.1 AVR Modul

Die Funktion „ComLoop()“ verarbeitet die einzelnen Nachrichten. Sie ist wie der Zustandsautomat in Form einer Scheduler Task implementiert. Die Periodizität, mit der die Task aufgerufen werden soll, hängt von der maximalen Länge der gesendeten Nachrichten ab. Da die maximale Nachrichtenlänge nicht die Grenze von 11 Bytes 2.2.3.5 überschreitet und zur Übertragung von einem Byte 1 ms notwendig ist (s. Formel 3.1), ist die Funktion zur Berarbeitung der Nachrichten, abgekürzt mit $t_{message}$, alle 11ms oder noch öfter aufzurufen. Eine höhere Periode $P_{t_{message}} > 11ms$ kann zu Verlust von Nachrichten führen 3.6. In der Abbildung 3.5 ist das zeitliche Verhalten der Tasks dargestellt.

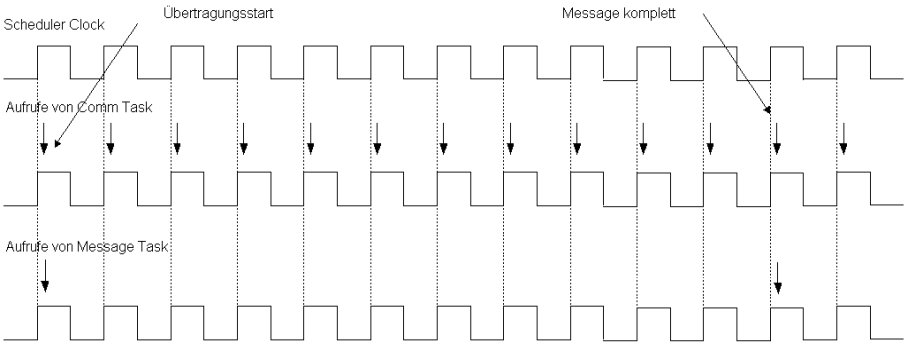



Abbildung 3.5: Zeitverhalten der Scheduler Tasks

**tReg Task** - die Funktion zur Übertragung einzelner Zeichen $P_{t_{comm}} = 1ms$

**ComLoop Task** - die Funktion zur Bearbeitung der Nachrichten $P_{t_{message}} \leq 11ms$

Bei jeder ansteigenden Flanke des Scheduler Clocks werden die aufzurufenden Tasks nacheinander, in der Reihenfolge ihrer Definition, ausgeführt (siehe das Kapitel 3.1). Bedingt durch den zeitlichen Aufwand, der dabei entsteht, werden die beiden Tasks in der Abbildung um $t_{delay}$ versetzt zur ansteigenden Flanke des Scheduler Clocks aufgerufen.

Das nächste Diagramm zeigt einen zeitlichen Verlauf der Tasks, bei dem es zu Verlust von Nachrichten führen kann.

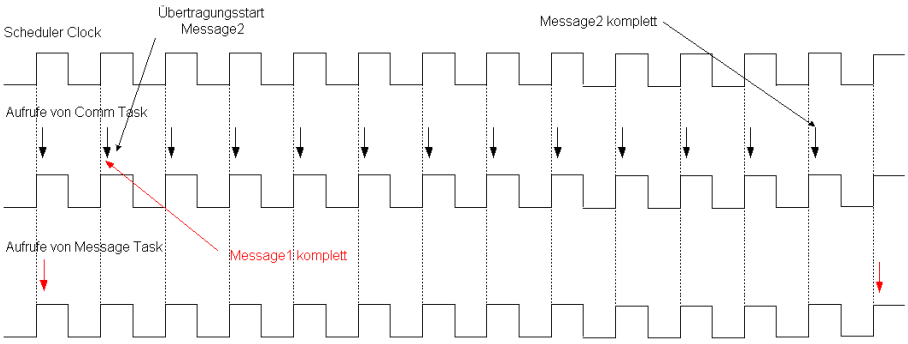



Abbildung 3.6: Verlust von Nachrichten

Angenommen: die Periode der Message Task ist grösser als 11ms ($P_{t_{message}} = 13ms$). Bei der Konstellation in der Abbildung 3.6 würde Message 1 verloren gehen. Beim ersten Aufruf von der Message Task ist Message 1 noch nicht komplett übertragen und beim zweiten befindet sich bereits die Message 2 im Empfangspuffer.

#### 3.3.3.2 PC Modul

Das PC Modul ist ein MFC (Microsoft Foundation Classes) basiertes Projekt 3.2 mit einer grafischen Oberfläche zur Interaktion mit den Benutzern. Für die serielle Kommunikation benutzt man eine lizenzfreie Programmbibliothek de Klein (2003), die ein ereignisgesteuertes Modell der Datenübertragung implementiert. D.h. immer wenn an der seriellen Scnittstelle ein Ereignis eintritt, wird eine Nachricht an das dazugehörige GUI-Fenster verschickt, dieses Ereignis zu bearbeiten.

Das Ereignis „EEventRecv“ tritt ein, wenn neue Daten von der seriellen Schnittstelle empfangen wurden. Dadurch wird die Funktion „OnSerialMsg“ der Fensterklasse „CChildView-Com“ getriggert (siehe Quellcodeauschnitt im Anhang C.1.2), die die empfangenen Daten zeichenweise aus dem Empfangspuffer ausliest und zur weiteren Verarbeitung an den Zustandsautomaten 3.3.2 übergibt.

Der Zustandsautomat ist in der Funktion „RS232::receive_data(char c)“ realisiert. Wird eine Nachricht komplett empfangen, wird die Funktion „RS232::ReadRxBuffer()“ zur Verarbeitung der Nachrichten aufgerufen.

# Kapitel 4

# Zusammenfassung

Ein Ziel dieser Bachelor-Arbeit war die Erfassung der rotierenden Objekte mit Hilfe einer digitalen Kamera. Zu diesem Zweck wurde ein mit einem Gleichstrommotor angetriebener Drehtisch konstruiert. Die auf dem Drehteller des Tisches positionierten Objekte werden während der Rotation durch einen Entfernungssensor detektiert. Die im Drehtisch implementierte Steuerung auf einem AVR-Mikrocontroller empfängt die Signale des Sensors. Zur Erfassung der detektierten Objekte sendet die Drehtisch-Steuerung Triggersignale an die Peripheriegeräte (Kamera, Blitz).

Ein weiteres Ziel war es, die Bilder durch eine Bildbearbeitungs-Software ( die Software wird im Rahmen einer anderen Bachelor-Arbeit erstellt) zu bearbeiten. Die von der Kamera aufgenommenen Objekte werden an einen PC gesendet. Die Ergebnisse der Auswertung ( Farbe des Objektes) werden über die serielle Schnittstelle der Drehtisch-Steuerung zur Verfügung gestellt. Die Kommunikation zwischen dem Mikrocontroller und PC wurde durch die Implementierung eines Übertragungsprotokolls realisiert. Über das Protokoll können Daten in beide Richtungen sowohl vom Mikrocontroller an den PC (wie z.B. Statusmeldungen der Hardware) als auch in die umgekehrte Richtung (wie z.B. Benutzerbefehle) gesendet werden.

Zur Eingabe der Benutzerbefehle wurde am PC eine Benutzersteuerung mit grafischer Oberfläche implementiert. In der Oberfläche kann der Benutzer ebenfalls die von der Kamera aufgenommenen Bilder sehen.

Ferner wurde in dieser Bachelor-Arbeit die Funktionalität implementiert, die vom Benutzer ausgewählze Objekte im bestimmten Bereich des Drehtellers zu positionieren. Dazu kann der Benutzer in der grafische Oberfläche eine Objektfarbe auswählen. Daraufhin wartet die Drehtisch-Steuerung, bis das nächste Objekt in dieser Farbe erfasst wird. Anschließend wird das Objekt in der gewünschten Position angehalten.

# Literaturverzeichnis


[Atmel ] ATMEL: *AT90CAN128 8-bit AVR Microcontroller with 128K Bytes of ISP Flash and CAN Controller*. – Dokumentation des AVR Mikrocontrollers (auf der beiliegenden CD)

[Bernstein 2007] BERNSTEIN, Herbert: *Mechatronik in der Praxis: Sensoren, Busssysteme, Antriebssysteme, Messverfahren*. 1. VDE Verlag GmbH, 2007. – ISBN 978-3-8007-2912-8

[de Klein 2003] DE KLEIN, Ramon: *Serial library for C++*. November 2003. – URL `http://www.codeproject.com/system/serial.asp`

[Digital ] DIGITAL, US: *OEM Miniature Optical Kit Encoder*. – Dokumentation des Inkrementalgebers (auf der beiliegenden CD)

[Drello ] DRELLO: *Lichtblitz-Stroboskop DRELLOSCOP 255*. – Dokumentation des Lichtblitz-Stroboskops (auf der beiliegenden CD)

[Drello 2007] DRELLO: *Lichtblitz Stroboskop 255*. August 2007. – URL `http://www.drello.de/german/strob.html`

[Ibrt 2007] IBRT: *the ibrt short message protocol, Receiving state machine*. August 2007. – URL `http://www.ibrtses.com/embedded/shortmsgprotocol.html`

[MaterialScience 2007] MATERIALSCIENCE, Bayer: *Materialeigenschaften von Makrolon*. August 2007. – URL `http://plastics.bayer.com/plastics/emea/de/docguard/A8043.pdf?docId=2386`

[Modelcraft ] MODELCRAFT: *RV-35C Geared Motor Series*. – Dokumentation des Getriebemotors (auf der beiliegenden CD)

[Netzwerkprotokoll 2007] NETZWERKPROTOKOLL, Wikipedia: *Netzwerkprotokoll*. August 2007. – URL `http://de.wikipedia.org/wiki/Netzwerkprotokoll`

[Pont 2001] PONT, Michael J.: *Patterns for Time-Triggered Embedded Systems*. Addison Wesley, 2001

[RoboterNetz.de ] ROBOTERNETZ.DE: *RN-VNH2 Dualmotor*. – Dokumentation des Moduls zur Motoransteuerung (auf der beiliegenden CD)

[SEW-Eurodrive 2007] SEW-EURODRIVE: *SEW-Eurodrive Handbuch*. August 2007. – URL `http://www.sew-eurodrive.de/download/pdf/11349204.pdf`

[Sharp ] SHARP: *GP2D12/GP2D15 General Purpose Type Distance Measuring Sensors*. – Dokumentation des Entfernungssensors (auf der beiliegenden CD)

[TheImagingSource ] THEIMAGINGSOURCE: *TheImagingSource DFK41BF02*. – Dokumentation der FireWire-Kamera (auf der beiliegenden CD)

[TheImagingSource 2007] THEIMAGINGSOURCE: *DFK 41BF02 - Spezifikation*. August 2007. – URL `http://www.theimagingsource.com/de/products/cameras/firewirecolor/dfk41bf02/specification/`

[Trampert 2003a] TRAMPERT, Wolfgang: *AVR-RISC Mikrocontroller: Architektur, Hardware-Ressourcen, Befehlsvorrat, Programmierung, Applikationen*. 2. Franzis Verlag GmbH, 2003. – ISBN 3-7723-5476-9

[Trampert 2003b] TRAMPERT, Wolfgang: *Messen, Steuern und Regeln mit AVR-Mikrocontrollern*. 2. Franzis Verlag, 2003

[www.mikrocontroller.net 2007] WWW.MIKROCONTROLLER.NET: *AVR-GCC-Tutorial*. September 2007. – URL `http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial`

# Abbildungsverzeichnis

# Anhang A

# Bedienungsanleitung

## A.1 Aufbau der Hardware

### A.1.1 Positionierung des Drehtisches und der Peripheriegeräte

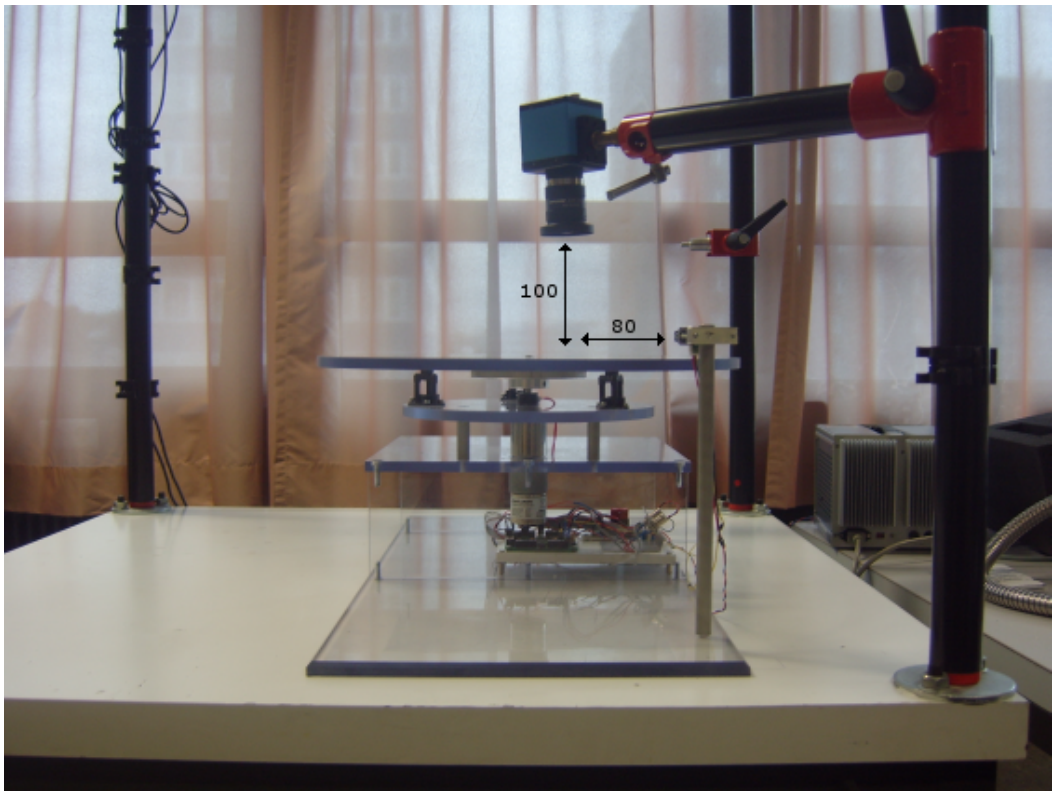



Abbildung A.1: Seitenansicht

Die Kamera ist oberhalb des Drehtellers(s. Abb. A.1) zu positionieren. Wie die optimale Position des Lichtblitz-Stroboskops zu den aufzunemenden Objekten ist, wird im Rahmen einer anderen Bachelor-Arbeit entschieden. Die Größenangaben der eingezeichneten Abstände auf den Bildern sind in [mm].

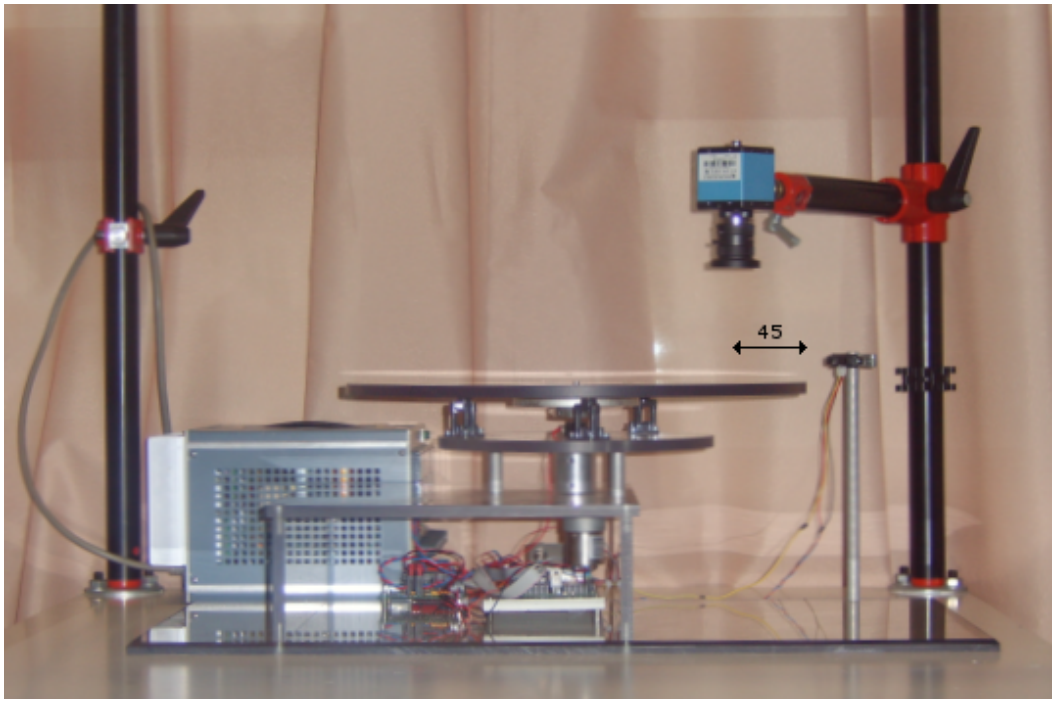



Abbildung A.2: Vorderansicht

Die Objekte sind im Abstand d=40mm (Bei der Objektbreite von 10mm) vom Drehtellerrand entfernt zu platzieren, so dass sich der Mittelpunkt der Objekte im Fokus der Kamera befindet.

### A.1.2 Verkabelung

Auf dem Bild A.4 ist die Innenansicht des Drehtisches zu sehen. Mit roten Markierungen 1 und 2 sind Anschlüsse für die Peripheriegeräte gekennzeichnet:

**1** Kameraanschluß

**2** Stroboskopanschluß

Als Verbindungskabel können Koaxialkabel mit BNC-Steckern auf beiden Seiten(s. Abb. A.3) verwendet werden.

Abbildung A.3: Triggerkabel

Mit 3 ist die RS232-Buchse markiert für die serielle Verbindung über ein Null-Modem-Kabel mit einem PC.

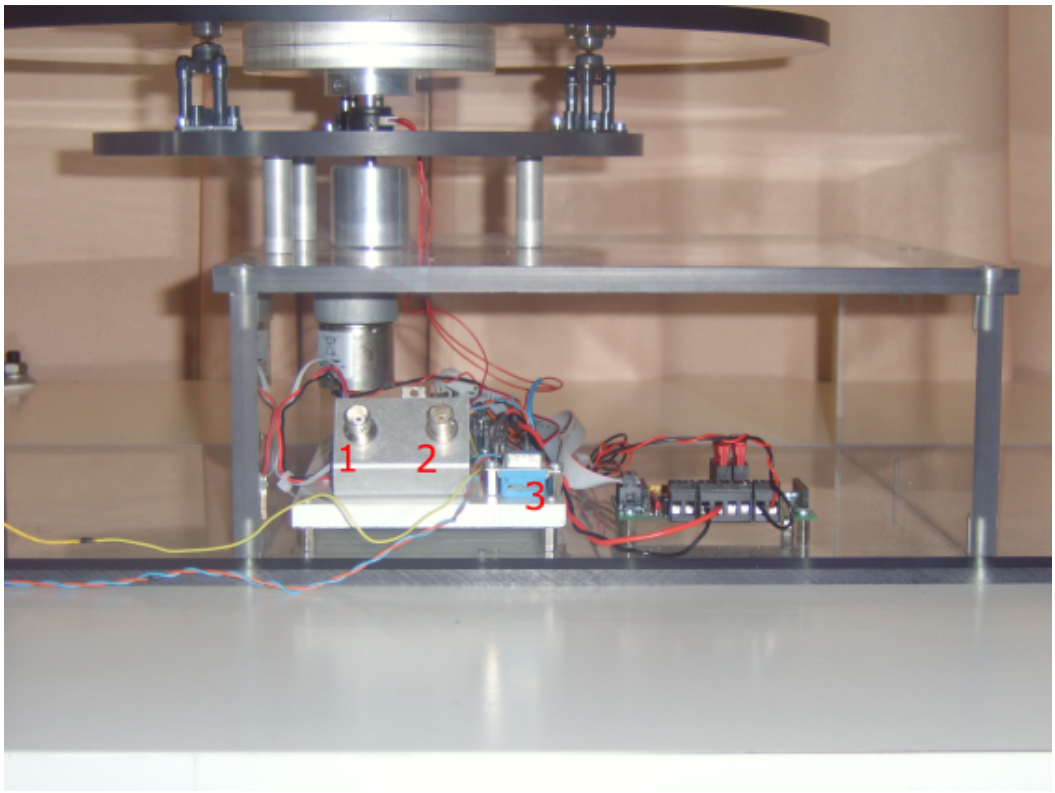



Abbildung A.4: Innenansicht

### A.1.3 Stromversorgung

Auf der Abbildung A.5 sind die Stromanschlüsse des Drehtisches dargestellt.

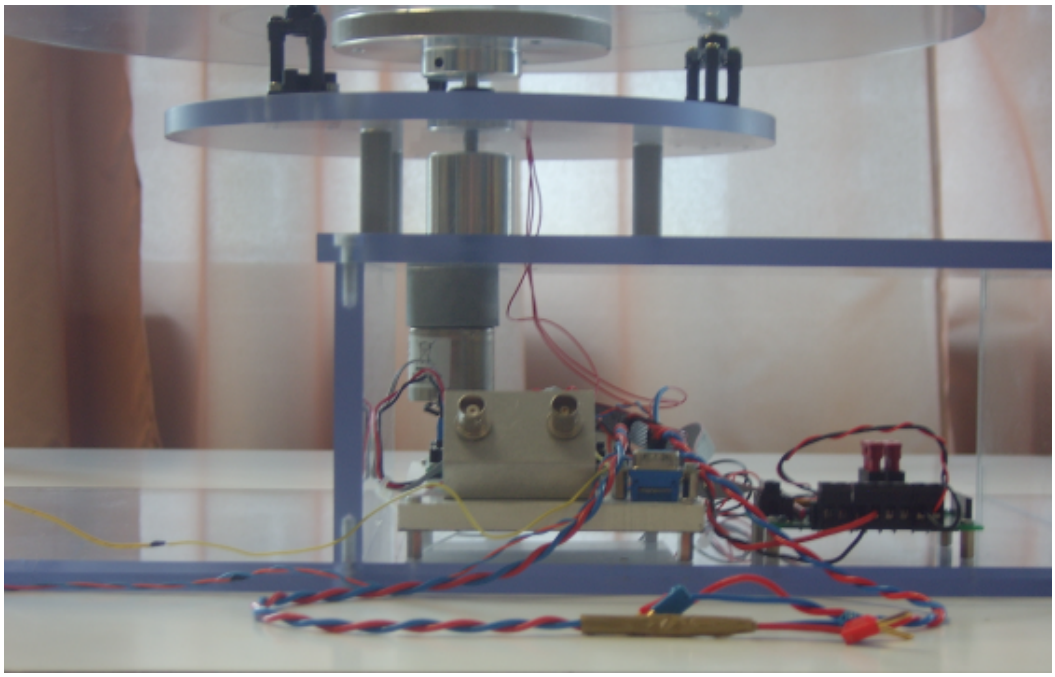


Abbildung A.5: Stromanschlüsse

Beide Anschlüsse bestehen aus zwei Kabeln:

- **Rot** für die Spannung
- **Blau** für die Masse

Der Motor wird mit 12V versorgt und die restliche Elektronik mit 5V:

- **12V** ist der mit Klebeband markierter Anschluss
- **5V** ist der Anschluss ohne Markierung

## A.2 Installation der Benutzersteuerung

### A.2.1 Treiberinstallation

Vor der Treiberinstallation ist die Kamera(hier: Fire-Wire Kamera DFK41BF02 2.3) über ein FireWire-Kabel mit dem PC zu verbinden.

Mit der Ausführung der Datei „drvInstaller.exe“ im Ordner „kamera_treiber“(im Hauptverzeichnis der beiliigenden CD) wird der notwendige Kameratreiber installiert.

### A.2.2 User-Interface

Der Ordner „DrehtischSteuerung“ (im Hauptverzeichnis der beiliigenden CD) ist komplett in ein lokales Verzeichnis zu übertragen, in dem der Benutzer Schreibrecht besitzt.

Mit der Ausführung der Datei „DrehtischSteuerung.exe“ startet die Benutzersteuerung ( s. Abb. A.6).

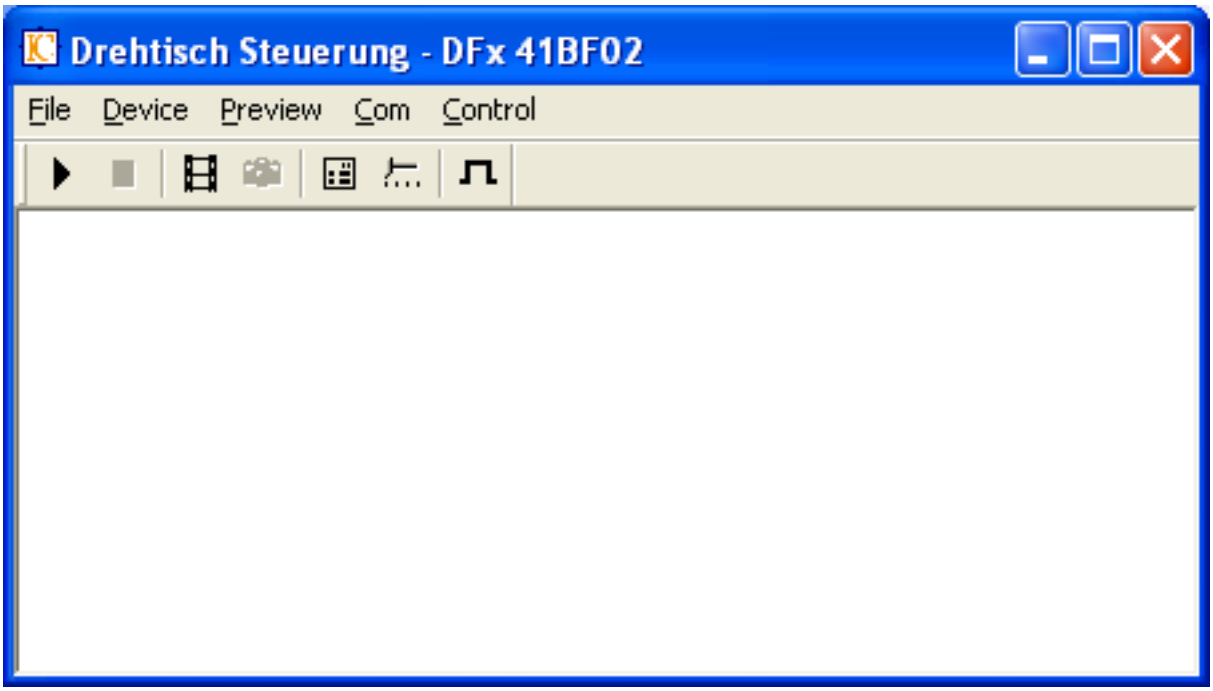



Abbildung A.6: Benutzersteuerung

## A.3 Steuerung des Drehtisches

Bevor der Drehtisch in Betrieb genommen werden kann, müssen folgende Vorbereitungen getroffen werden:

- Peripheriegeräte sind mit dem Drehtisch verbunden und positioniert nach Anleitung im Kapitel A.1
- Kamera ist über FireWire-Anschluss mit PC verbunden
- Treiber und Benutzersteuerung sind auf dem PC installiert (s. Anleitung im Kap. A.2)
- der Drehtisch und der PC sind über ein serielles Kabel miteinander verbunden.

### A.3.1 Verbindungsaufbau

Nach dem Start der Benutzersteuerung ist die Verbindung zum Drehtisch aufzubauen. Durch die Auswahl des Menüpunktes „Com -> Open“ (s. Abb. A.7) erscheint ein Dialog-Fenster zur Auswahl der seriellen Schnittstelle.

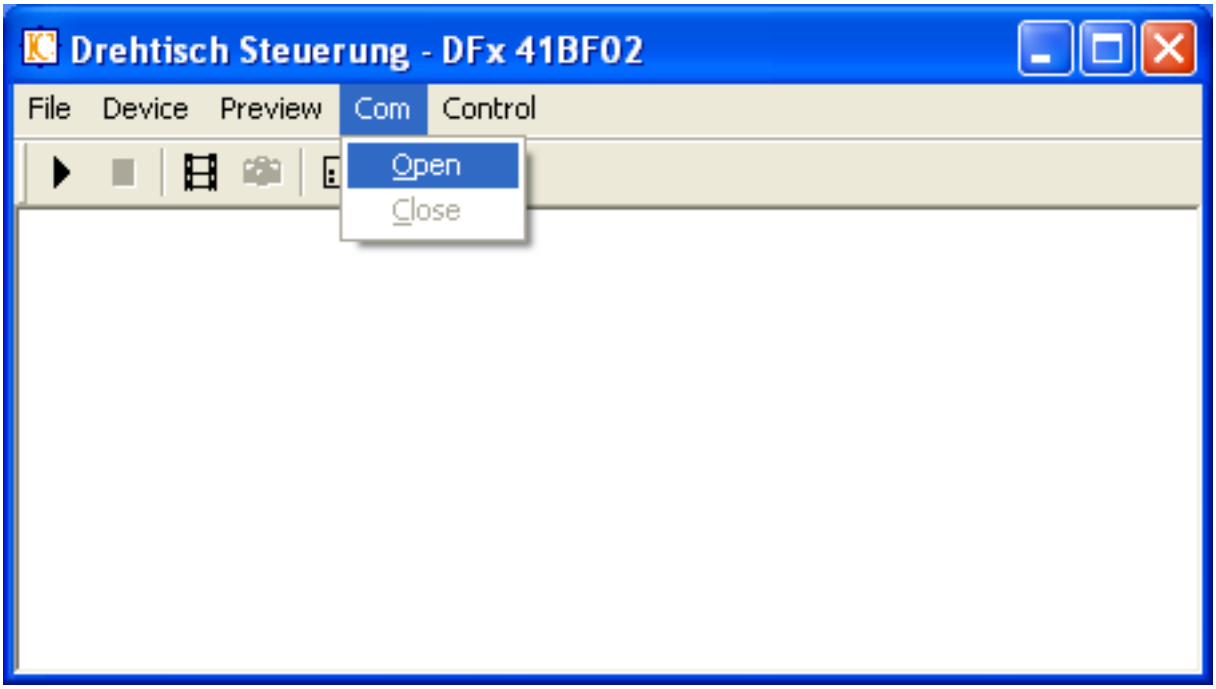
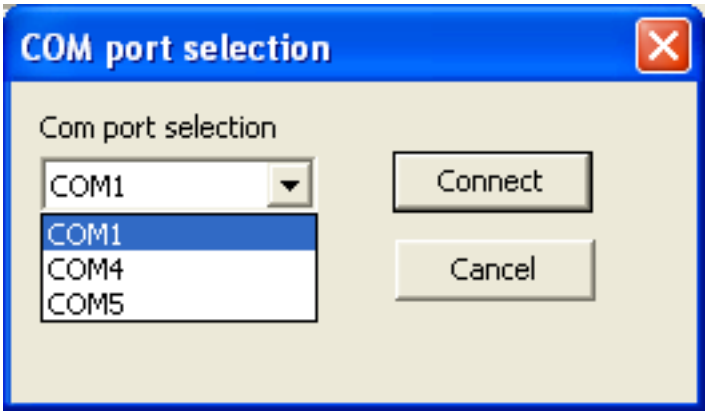



Abbildung A.7: Aufbau der RS232-Verbindung

In dem Dialog-Fenster(s. nächste Abbildung) ist über ein Drop-Down-Feld die serielle Schnittstelle auszuwählen, über die der PC mit dem Drehtisch verbunden ist. Per Button „Connect“ kann die Verbindung aufgebaut werden.



Über den Menüpunkt „Com -> Close“ (s. Abb. A.7) kann die aufgebaute Verbindung geschlossen werden.

## A.3.2 Benutzerführung

### A.3.2.1 Drehtisch starten und anhalten

Über die Menüpunkte „Control->Start“ und „Control->Stop“(s.Abb. A.8) wird der Drehtisch mit der minimalen Drehzahl gestartet bzw. angehalten.

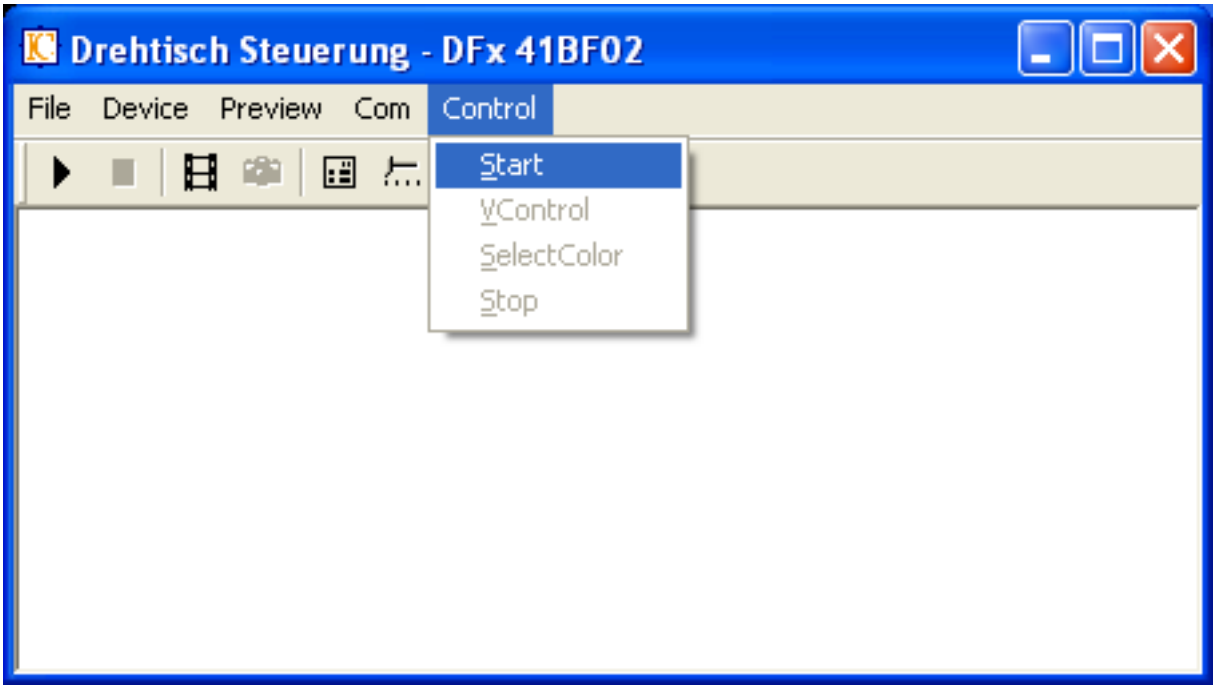



Abbildung A.8: Drehtisch starten

### A.3.2.2 Rotationsgeschwindigkeit ändern

Über den Menüpunkt „Control->VControl“ wird ein Dialog-Fenster(s. Abb. ) aufgerufen, in dem aktuelle Rotationsgeschwindigkeit des Drehtellers angezeigt und geändert werden kann.

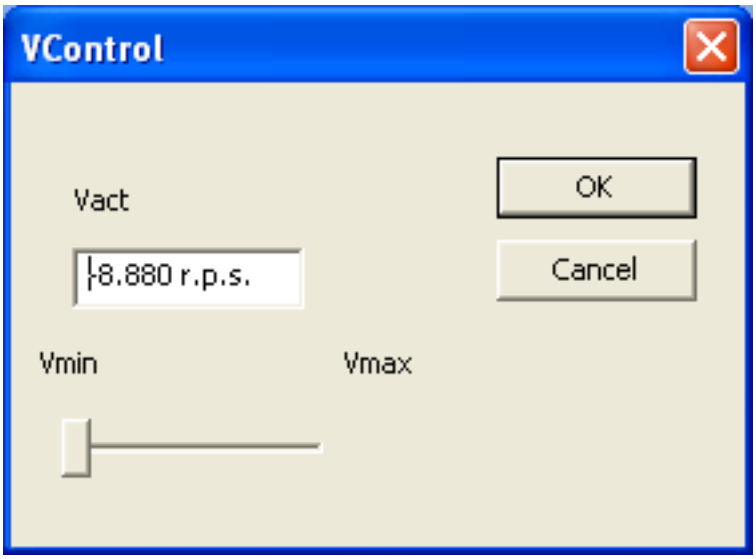



Abbildung A.9: Änderung der Drehzahl

Die Geschwindigkeit wird in Umdrehungen pro Sekunde(rps) angezeigt.

#### A.3.2.3 Farbe des zu positionierenden Objektes wählen

Über den Menüpunkt „Control->SelectColor“ wird ein Dialog-Fenster aufgerufen, in dem der Benutzer die Farbe des Objektes auswählen kann, das in die definierte Position gebracht werden soll.

In der vorliegenden Bachelor-Arbeit wurde ein Testmodul zur Bildbearbeitung eingesetzt, mit dem man schwarze und weisse Objekte unterscheiden konnte. Dementsprechend erscheinen im Dialog-Fenster zwei Farben zur Auswahl(schwarz und weiss).

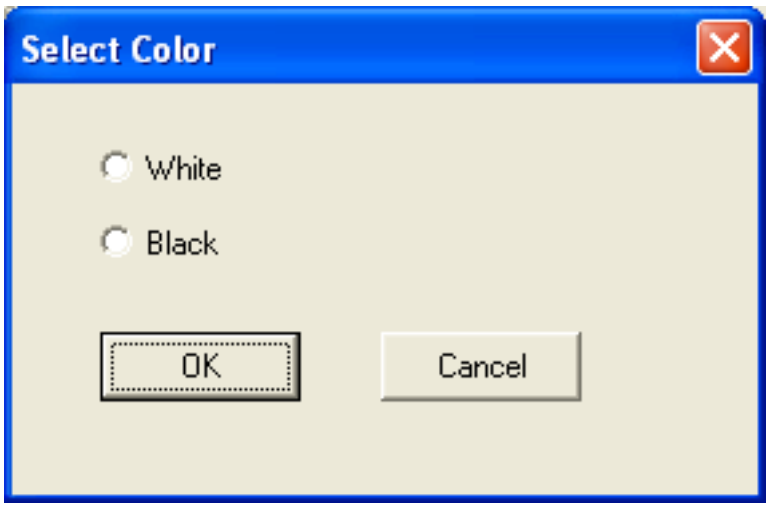
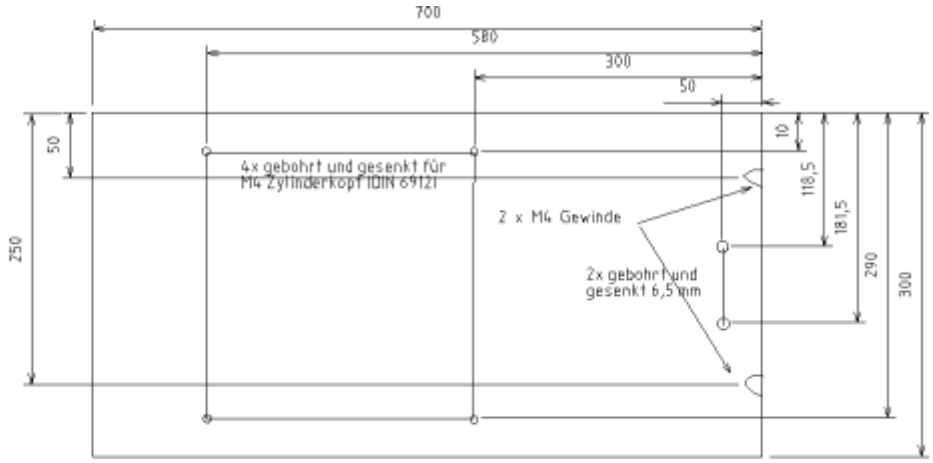



Abbildung A.10: Auswahl der Objektfarbe

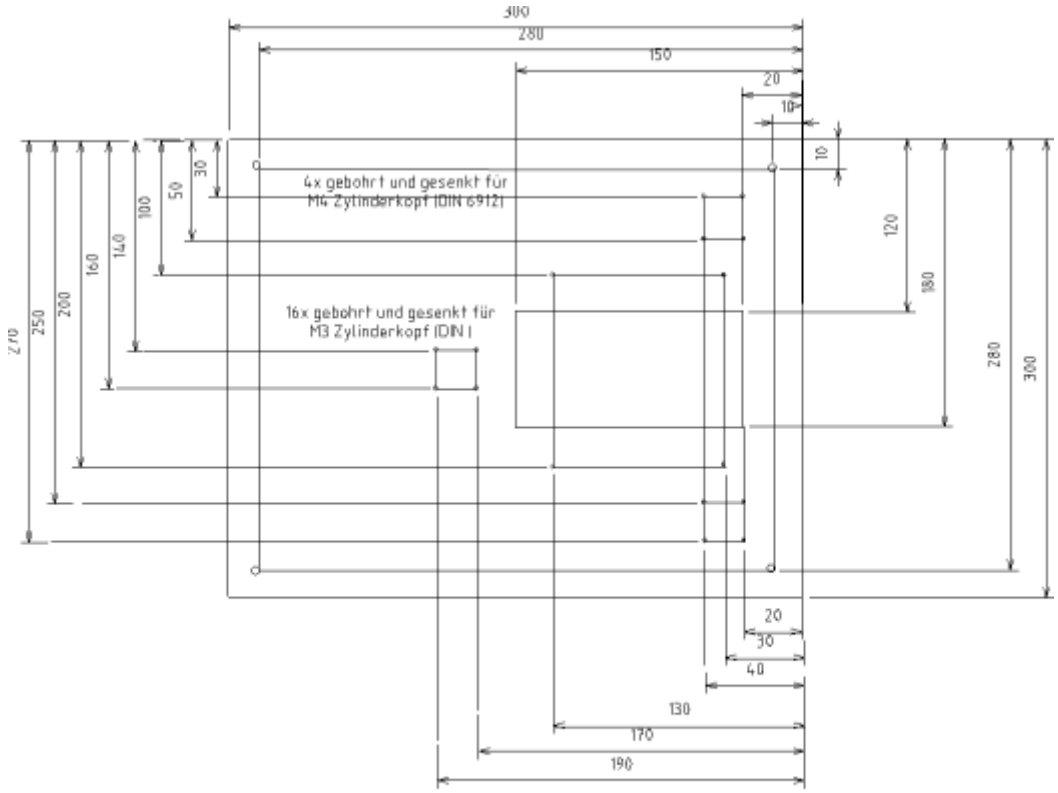
# Anhang B

# Konstruktionspläne




Grundplatte (1 Stück, Material: Makrolon 10 mm dick)

Abbildung B.1: Grundplatte




Deckplatte (1 Stück, Material: Makrolon 10mm dick)

Abbildung B.2: Deckplatte

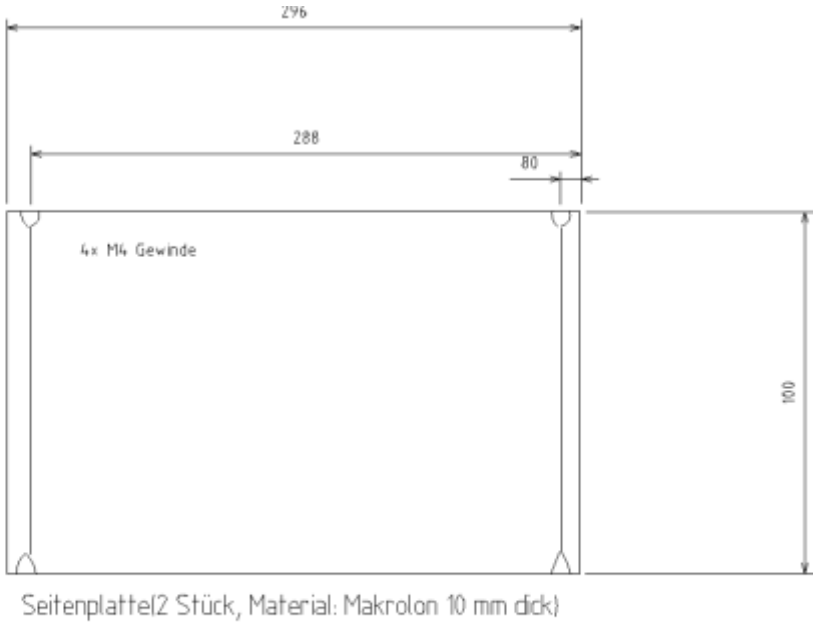



Abbildung B.3: Seitenplatte

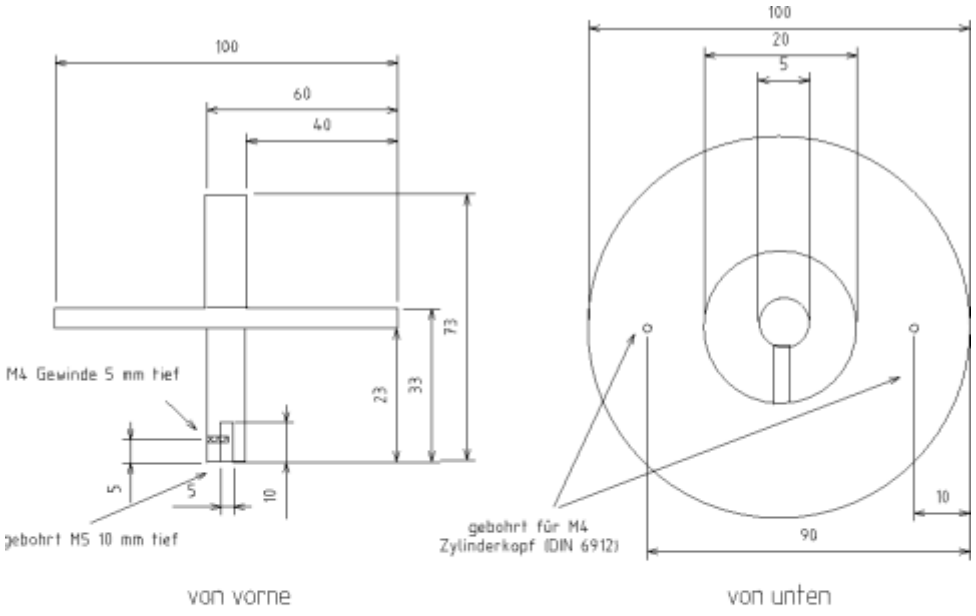



Abbildung B.4: Motorflansch

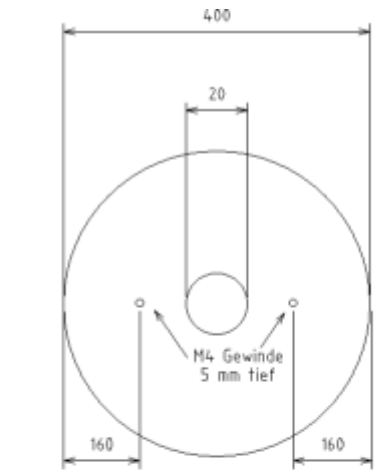



Abbildung B.5: Drehteller

# Anhang C

# Ausschnitte des Quellcodes

## C.1 Kommunikationsprotokoll

### C.1.1 AVR

```
void ComLoop() {

  if (RxState==0x0E) {
    RxState=0;
  } else if (RxState==0x0F) {

    received_message_count++;
if ( (RxBuff[3]==microC_ID) || (RxBuff[3]=0xFF) ) {
      src=RxBuff[2];
      dst=RxBuff[3];
  if(new_picture_flag==TRUE) {
    cmd = NEW_PICTURE;
        new_picture_flag = FALSE;
      }
  else cmd=RxBuff[4];
  data_0=RxBuff[5];
  data_1=RxBuff[6];
      switch (cmd) {
        case ECHO : { // Echo
  send_buffer_to_rs232(TxBuff,TxBufLen,RS232_0);
  break;
}
```

```
case NEW_PICTURE : { // Kamera wurde getriggert
          TxBuff[TxCMD_BYTE] = cmd;
  send_buffer_to_rs232(TxBuff,TxBufLen,RS232_0);

  break;
}

case NEW_COLOR_VALUE : {
          // Bilderdaten wurden ausgewertet
          // cmd zurücksetzen
  if(data_1 == NEW_SELECTED_COLOR_VALUE) {
    // ausgewähltes Objekt positionieren
    drehtisch_anhalten(ziel_position);
  }
  TxBuff[TxCMD_BYTE] = ECHO;
  send_buffer_to_rs232(TxBuff,TxBufLen,RS232_0);

  break;
}
case NEW_V_VALUE : {

  change_speed(data_0);
  TxBuff[TxCMD_BYTE] = ECHO;
  send_buffer_to_rs232(TxBuff,TxBufLen,RS232_0);

  break;
}

case RESTART : {
  drehtisch_starten();
  TxBuff[TxCMD_BYTE] = ECHO;
  send_buffer_to_rs232(TxBuff,TxBufLen,RS232_0);

  break;
}

default : break;
  }
```

```
}
RxState=0;

  }
}
```

### C.1.2 PC

```
LRESULT CChildViewCom::OnSerialMsg (WPARAM wParam)
    {
    CSerial::EEvent eEvent = CSerial::EEvent(LOWORD(wParam));
    CSerial::EError eError = CSerial::EError(HIWORD(wParam));

    if (eEvent & CSerial::EEventRecv)
    {
    DWORD dwRead;
    char szData[20];
    const int nBuflen = sizeof(szData)-1;

    m_serial.Read(szData,nBuflen,&dwRead);

    for (DWORD dwChar=0; dwChar<dwRead; dwChar++)
    {
      m_RS232.receive_data(szData[dwChar]);
        }

      return 0;
      }
    }

      void RS232::receive_data(char c) {
      // receiving state machine (procedure is moreless identical on th
      // embedded device)
      if(RxPtr == RxBufLen) {
        RxState = 0x0E;
      ReadRxBuffer();
      } else {

      switch(RxState) {
        case 0 : {
```

```
        if ((int)c==SYN) RxState = 1; // wait for SYN
    break;
      }
      case 1 : {
        switch((int)c) {              // wait for STX
          case SYN : {
            RxState = 1;
            break;
          }
        case STX : {
        RxBuffer[0] = STX;
        RxState  = 2;
            break;
          }
          default : break;
        }
    break;
      }

      case 2 : {                  // read Len
        RxBuffer[1] = (int)c;
        RxLen = (int)c;
        RxState = 3;
        RxPtr = 2;
        break;
      }

      case 3 : {                  // counting
        RxBuffer[RxPtr] = (int)c;
        RxPtr++;
    if(RxPtr == RxLen) {
      RxState = 0x0F;
      ReadRxBuffer();
    }
        break;
      }
 default : break;
    }
  }
}
```

```
void RS232::ReadRxBuffer() {
   unsigned int src=0,
                dst=0,
           cmd=0;
   if (RxState==0x0E) {
     RxState=0;
   } else if (RxState==0x0F) {

 if ( (RxBuffer[3]==PC_ID) || (RxBuffer[3]=0xFF) ) {
     src=RxBuffer[2];
     dst=RxBuffer[3];
     cmd=RxBuffer[4];

     switch (cmd) {
       case ECHO : { // Echo
         send_data(ECHO);
     break;
   }
   case NEW_PICTURE : {
         // Bilderdaten wurden von der Kamera verschickt
         send_data(NEW_COLOR_VALUE);
   break;
   }

   default : break;
   }
 }
 RxState=0;
   RxPtr=0;

   }

 }

 void RS232::send_data(unsigned int cmd) {
 DWORD pdwWritten;

   if(new_v_value_flag == TRUE) cmd = NEW_V_VALUE;
 if(restartFlag == TRUE) cmd = RESTART;
```

```
switch(cmd) {
  case ECHO : {
      TxBuffer[TxCMD_BYTE]     = ECHO;
  this->pm_serial->Write(TxBuffer, TxBufLen, 0);
  RS232::message_counter++;
  RS232::new_color_flag = FALSE;
  break;
  }
  case NEW_COLOR_VALUE : {
      if(RS232::new_color_flag==TRUE) {
    switch(RS232::new_selected_color_flag) {
      case FALSE : {
      TxBuffer[TxCMD_BYTE]     = NEW_COLOR_VALUE;
      TxBuffer[TxDATA_BYTE_0] = color;
      this->pm_serial->Write(TxBuffer, TxBufLen, 0);
      RS232::new_color_flag = FALSE;
      break;
    }
    case TRUE : {
      TxBuffer[TxCMD_BYTE]     = NEW_COLOR_VALUE;
      TxBuffer[TxDATA_BYTE_0]  = color;
      if(color==selected_color) {
        TxBuffer[TxDATA_BYTE_1]  = NEW_SELECTED_COLOR_VALUE;
        RS232::new_selected_color_flag = FALSE;
      }
        this->pm_serial->Write(TxBuffer, TxBufLen, 0);
        RS232::new_color_flag = FALSE;
        TxBuffer[TxDATA_BYTE_1]  = 0x00; // reset data byte
        break;
      }
        default : break;
      }
    } else {

    TxBuffer[TxCMD_BYTE] = ECHO;
    this->pm_serial->Write(TxBuffer, TxBufLen, 0);
  }
  break;
  }
```

```
      case NEW_V_VALUE : {
          TxBuffer[TxCMD_BYTE]     = NEW_V_VALUE;
      TxBuffer[TxDATA_BYTE_0]  = v_value;
      this->pm_serial->Write(TxBuffer, TxBufLen, 0);
      RS232::new_v_value_flag = FALSE;
      break;
      }

      case RESTART : {
      TxBuffer[TxCMD_BYTE]  = RESTART;
          this->pm_serial->Write(TxBuffer, TxBufLen, 0);
      RS232::restartFlag = FALSE;
      break;
      }

      default : break;

    }

    }
```

## C.2 Kooperativer Scheduler

### C.2.1 Header Datei

```
/*-----------------------------------------------*-

   Sch51.h (v1.00)


-*-----------------------------------------------*/


#ifndef _SCH51_H
#define _SCH51_H

#include "v_measure.h"
#include "port.h"
```

```
// ------ Public function prototypes --------------

// Core scheduler functions
void SCH_tasks_Array_Init(void);
void  SCH_Dispatch_Tasks(void);
tByte SCH_Add_Task(void (*pFunction) (void), const tWord, const tWord);

//bit   SCH_Delete_Task(const tByte);
uint8_t   SCH_Delete_Task(const tByte);
void  SCH_Report_Status(void);


// ------ Public constants ------------------------

// The maximum number of tasks required at any one time
// during the execution of the program
//
// MUST BE ADJUSTED FOR EACH NEW PROJECT
#define SCH_MAX_TASKS 10

// ------ Public data type declarations -----------



// Store in DATA area, if possible, for rapid access
// Total memory per task is 7 bytes
typedef struct  {
  // Pointer to the task (must be a 'void (void)' function)
  void (* pTask)(void);

  // Delay (ticks) until the function will (next) be run
  // - see SCH_Add_Task() for further details
  tWord Delay;

  // Interval (ticks) between subsequent runs.
  // - see SCH_Add_Task() for further details
  tWord Period;

  // Incremented (by scheduler) when task is due to execute
  tByte RunMe;
```

```
} sTask;

// ------ Public variable definitions -------------
// The array of tasks

sTask SCH_tasks_G[SCH_MAX_TASKS];
sTask* pSCH_tasks_G;
// Used to display the error code
// See Main.H for details of error codes
// See Port.H for details of the error port
tByte Error_code_G;
tByte Last_error_code_G;

#endif

/*-------------------------------------------------*-
  ---- END OF FILE ------------------------------
-*-------------------------------------------------*/
```

### C.2.2 Source Datei

```
/*-------------------------------------------------*-

   Sch51.C (v1.00)

-*-------------------------------------------------*/


#include "Sch51.h"



// ------ Private function prototypes ---------------

static void SCH_Go_To_Sleep(void);

// ------ Public variable definitions ---------------

// The array of tasks
```

```
// deklaration im header file
//sTask SCH_tasks_G[SCH_MAX_TASKS];



/*------------------------------------------------------*-

  SCH_Dispatch_Tasks()

  This is the 'dispatcher' function.  When a task (function)
  is due to run, SCH_Dispatch_Tasks() will run it.
  This function must be called (repeatedly) from the main loop.

-*------------------------------------------------------*/
void SCH_Dispatch_Tasks(void) {
  tByte Index;


  // Dispatches (runs) the next task (if one is ready)
  for (Index = 0; Index < SCH_MAX_TASKS; Index++) {

 if (SCH_tasks_G[Index].RunMe > 0) {

       (*SCH_tasks_G[Index].pTask)();  // Run the task


     // Reset / reduce RunMe flag
       SCH_tasks_G[Index].RunMe -= 1;

       // Periodic tasks will automatically run again
       // - if this is a 'one shot' task,
       // remove it from the array
       if (SCH_tasks_G[Index].Period == 0) {
         SCH_Delete_Task(Index);
       }

     }

 //wdt_reset();
```

```
  }


  // The scheduler enters idle mode at this point
  SCH_Go_To_Sleep();
}


/*-------------------------------------------------*-

  SCH_Add_Task()

  Causes a task (function) to be executed at regular
  intervals or after a user-defined delay

  Fn_P   - The name of the function which is to be
    scheduled. NOTE: All scheduled functions
    must be 'void, void' - that is, they must
    take no parameters, and have a void return
    type.

  DELAY  - The interval (TICKS) before the task is
    first executed

  PERIOD - If 'PERIOD' is 0, the function is only
    called once, at the time determined by
    'DELAY'.  If PERIOD is non-zero, then the
    function is called repeatedly at an interval
           determined by the value of PERIOD (see below
           for examples which should help clarify this).


  RETURN VALUE:

  Returns the position in the task array at which the
  task has been added.  If the return value is
  SCH_MAX_TASKS then the task could not be added to the
  array (there was insufficient space).  If the return
  value is < SCH_MAX_TASKS, then the task was added
  successfully.
```

```
  Note: this return value may be required, if a task is
  to be subsequently deleted - see SCH_Delete_Task().

  EXAMPLES:

  Task_ID = SCH_Add_Task(Do_X,1000,0);
  Causes the function Do_X() to be executed once after
  1000 sch ticks.

  Task_ID = SCH_Add_Task(Do_X,0,1000);
  Causes the function Do_X() to be executed regularly,
  every 1000 sch ticks.

  Task_ID = SCH_Add_Task(Do_X,300,1000);
  Causes the function Do_X() to be executed regularly,
  every 1000 ticks.
  Task will be first executed at T = 300 ticks,
  then 1300, 2300, etc.

-*---------------------------------------------------*/

tByte SCH_Add_Task(void (* pFunction)(void), const tWord DELAY,
const tWord PERIOD) {
  tByte Index = 0;

  // First find a gap in the array (if there is one)
  while ((SCH_tasks_G[Index].pTask != 0) && (Index < SCH_MAX_TASKS)) {
    Index++;
  }

  // Have we reached the end of the list?
  if (Index == SCH_MAX_TASKS) {
    // Task list is full
    //
    // Set the global error variable
    Error_code_G = ERROR_SCH_TOO_MANY_TASKS;

    // Also return an error code
    return SCH_MAX_TASKS;
```

```
  }

  // If we're here, there is a space in the task array
  SCH_tasks_G[Index].pTask  = pFunction;

  SCH_tasks_G[Index].Delay  = DELAY;
  SCH_tasks_G[Index].Period = PERIOD;

  SCH_tasks_G[Index].RunMe  = 0;

  return Index; // return position of task (to allow later deletion)
}

/*------------------------------------------------------------*-

  SCH_Delete_Task()

  Removes a task from the scheduler.  Note that this does
  *not* delete the associated function from memory:
  it simply means that it is no longer called by the scheduler.

  TASK_INDEX - The task index.  Provided by SCH_Add_Task().

  RETURN VALUE:  RETURN_ERROR or RETURN_NORMAL

-*------------------------------------------------------------*/
bit SCH_Delete_Task(const tByte TASK_INDEX) {
  bit Return_code;

  if (SCH_tasks_G[TASK_INDEX].pTask == 0) {
    // No task at this location...
    //
    // Set the global error variable
    Error_code_G = ERROR_SCH_CANNOT_DELETE_TASK;

    // ...also return an error code
    Return_code = RETURN_ERROR;
  }
  else {
    Return_code = RETURN_NORMAL;
```

```
  }

  SCH_tasks_G[TASK_INDEX].pTask   = 0x0000;
  SCH_tasks_G[TASK_INDEX].Delay   = 0;
  SCH_tasks_G[TASK_INDEX].Period  = 0;

  SCH_tasks_G[TASK_INDEX].RunMe   = 0;

  return Return_code;       // return status
}
```

### C.2.3 Timer und ISR

```
/*------------------------------------------------------------*-

   timer.c (v1.00)

/*------------------------------------------------------------*-

  SCH_Init_T0()

  Scheduler initialisation function.  Prepares scheduler
  data structures and sets up timer interrupts at required rate.

  You must call this function before using the scheduler.

-*------------------------------------------------------------*/
void SCH_Init_T0(void) {

  // Using Timer 0, 8-bit
  // Compare Match Output Mode, Prescaler 64
  TCCR0A = (1 << COM0A0) | (1 << WGM01) | (1 << CS01) | (1 << CS00);
  // enable interrupt on compare match
  TIMSK0 = (1 << OCIE0A);
  TIFR0 = (1 << OCF0A);

  // new, zwei perioden alternierend ~0,9ms - ~0,1ms
  OCR0A = SHORT_PERIOD;
```

```
}


SIGNAL(SIG_OUTPUT_COMPARE0){
  tByte Index,counterRunMe = 0,tmp;

  LED_Flash_Update();

  tmp = OCR0A;
  if(tmp==LONG_PERIOD) {
  OCR0A = SHORT_PERIOD;

    SCH_tasks_G[0].RunMe += 1;  // Inc. the 'RunMe' flag
    counterRunMe ++;
  } else {

OCR0A = LONG_PERIOD;

  // NOTE: calculations are in *TICKS* (not milliseconds)
  for (Index = 1; Index < SCH_MAX_TASKS; Index++) {
    // Check if there is a task at this location
    if (SCH_tasks_G[Index].pTask) {

      if ( SCH_tasks_G[Index].Delay == 0) {
        // The task is due to run

        // Inc. the 'RunMe' flag
        SCH_tasks_G[Index].RunMe += 1;
        counterRunMe ++;
    if (SCH_tasks_G[Index].Period) {
          // Schedule regular tasks to run again
          SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].Period;
          SCH_tasks_G[Index].Delay -= 1;
        }
      }
      else {
        // Not yet ready to run: just decrement the delay
        SCH_tasks_G[Index].Delay -= 1;
      }
    }
```

```
  }


  }

  if(counterRunMe > 1) {
    Error_code_G = ERROR_SCH_TOO_MANY_TASKS;
  }
}
```

*Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, 26. Oktober 2007 Alexander Abramson