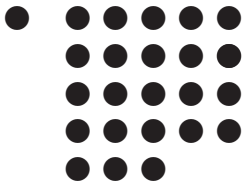


Fachhochschule Köln
University of Applied Sciences Cologne

07 Fakultät für Informations-, Medien- und Elektrotechnik

Institut für Nachrichtentechnik

Diplomarbeit

Hardware- und Softwaremodule für eine exemplarisch aufzubauende Roboterzelle zur bildverarbeitungsbasierten Sortierung von Stanzteilen

Student: **Jürgen Wischer**
Martr.Nr.: 11023875

Referent: Prof. Dr.-Ing. Lothar Thieling
Korreferent: Prof. Dr.-Ing. Georg Hartung

Abgabedatum: 15. Januar 2004

Hiermit versichere ich, dass ich die Diplomarbeit selbständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

(Jürgen Wischer)

Danksagung

Ich danke Herrn Prof. Dr.-Ing. Lothar Thieling für die Betreuung und das interessante Thema sowie Herrn Prof. Dr.-Ing. Georg Hartung für die Unterstützung als Korreferent.
Bei Herrn Dipl. Ing. Peter Pohlig möchte ich mich, stellvertretend für alle im Labor, für die gute Atmosphäre bedanken.
Weiterhin möchte ich mich bei meinen Eltern bedanken, die mir das Studium durch ihre Unterstützung erst ermöglicht haben. Meiner Freundin Jeannette Wittek danke ich für die seelische Unterstützung und für das Korrekturlesen während dieser Zeit.

Inhaltsverzeichnis

Teil I

Hauptteil

1 Einleitung

Dieses Kapitel beinhaltet die Einführung und eine Kapitelübersicht der Diplomarbeit.

1.1 Einführung

Die Grundidee der Diplomarbeit war die Realisation eines Demonstrationsobjektes, um die Fähigkeiten der industriellen Bildverarbeitung aufzuzeigen.

Um diese möglichst interessant darzustellen, war die Aufgabenstellung der Entwurf eines Systems zur Erkennung und Sortierung von Stanzteilen mit Hilfe einer Roboterzelle.

Als Stanzteile wurden der Anschaulichkeit halber Euro-Münzen gewählt.

Dabei gliedert sich das Thema in zwei Arbeitsbereiche auf. Zum einen in die Entwicklung einer Ansteuerungsschaltung und -software für den Roboter, zum anderen in die Entwicklung eines Bildverarbeitungssystems zur Erkennung der Stanzteile.

Das Thema bietet einen interessanten Querschnitt von der Entwicklung der Ansteuerungsschaltung (*VHDL*) und -software (*Visual C++*), über die Koordinatentransformation für den Roboter (Geometrie und *Visual C++*) bis hin zur Erstellung eines Bildverarbeitungssystems (*Ad Oculos* und *C*). Die Diplomarbeit wurde zusammen mit einem zweiten Diplomanden Herrn Andreas Koch erstellt, wobei jeder einzelne Hard- und Softwaremodule entwickelt hat.

Beim Entwurf der Software wurde darauf geachtet, dass die einzelnen Teile möglichst modular aufgebaut sind, um diese im Rahmen nachfolgender Projekte wiederverwenden zu können.

Der Aufbau dieser Arbeit orientiert sich an der Entwicklung des gesamten Systems. Die Übergänge der einzelnen Bereiche gehen teilweise fließend ineinander über. Zuerst wurde eine Ansteuerungsschaltung für den Roboter entwickelt, danach die Ansteuerungssoftware

und die Koordinatentransformation, im Anschluss das Bildverarbeitungssystem. Abschließend erfolgte die Abstimmung der Teilsysteme aufeinander.

Der folgende Abschnitt gibt einen Überblick über die einzelnen Kapitel dieser Diplomarbeit.

1.2 Kapitelübersicht

Kapitel 2 gibt einen Überblick über die für die Aufgabenstellung verwendete Hardware und Software.

Kapitel 3 beschreibt die Schaltung und die einzelnen Module für die Ansteuerungsplatine.

Kapitel 4 beschreibt die Herleitung der Formeln für die Koordinatentransformation und gibt einen Überblick über die Dynamic Link Library, die daraus erstellt wurde.

Kapitel 5 gibt einen kurzen Einblick in die Ansteuerung des Roboters mit dem PC (*Cobradll*), beschreibt den *CobraTESTer*, die *cobra_use* Library und das Ansteuerungsmodul.

Kapitel 6 beschreibt und bewertet das Bildverarbeitungssystem mit den einzelnen Modulen.

Kapitel 7 gibt einen Überblick über das Gesamtsystem, die Voreinstellungen und den Ablauf, sowie einen Ausblick zur Weiterentwicklung.

2 Arbeitsmittel

Dieses Kapitel gibt einen Überblick über die verwendete Hardware und Software, die zur Lösung der Aufgabenstellung verwendet wurde bzw. Bestandteil dieser sind.

2.1 Der Roboter Cobra RS2

Der Cobra RS2 von der Firma Sekuria ist ein Lehrroboter, der einen guten Einstieg in die Welt der Robotik bietet.

Auf den folgenden Seiten findet sich eine genaue Beschreibung des Roboters, seiner Funktionen, der Schnittstelle und ihre Ansteuerung.

2.1.1 Beschreibung des Roboters

Der Cobra RS2 wurde mit dem Ziel entwickelt den Zugang zur Robotertechnik zu erleichtern. Durch seine Auslegung ist er vielseitig nutzbar. Er ist mit 4,5 kg Gewicht in einer Alu-Zellenbauweise sehr kompakt gebaut und vielseitig nutzbar.

Der Roboter ist mittels einer 8 bit Parallelschnittstelle programmierbar (eine genaue Beschreibung der Schnittstelle findet sich in Abschnitt 2.1.2), womit beliebige Bewegungen durchgeführt werden können.

Zum Erreichen der Positionen bietet der Cobra RS2 fünf Freiheitsgrade mit sechs Schrittmotoren/Achsen. Mit diesen kann man den Drehturm, den Oberarm und den Unterarm schrittweise bewegen und den Greifer hoch- ,runterfahren, drehen, öffnen und schließen. Daraus ergibt sich eine Arbeitshalbkugel mit dem Durchmesser von 890 mm, einer mittleren Positionsauflösung von 0,5 mm und einem Positionierungsfehler von ± 2 mm. Es können Objekte bis zu einem Gewicht von 500 g vom Greifer gefasst und bewegt werden.

Während die Drehung des Greifers nicht begrenzt ist, besitzen alle anderen Achsen einen Anschlag, der den Bewegungsbereich mechanisch einschränkt. Der Vorteil des Cobra RS2

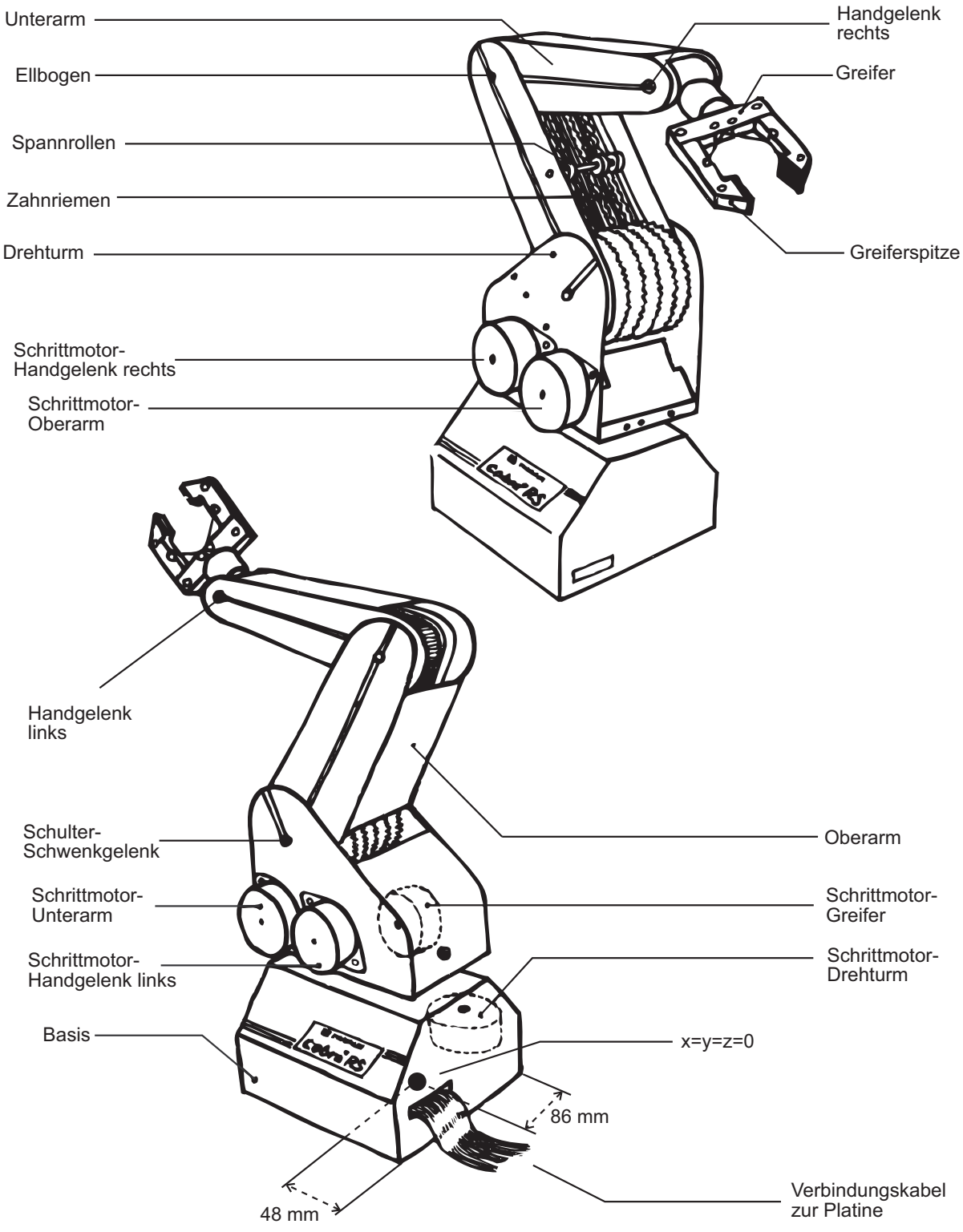


Abbildung 2.1: Grundbegriffe des Cobra RS2

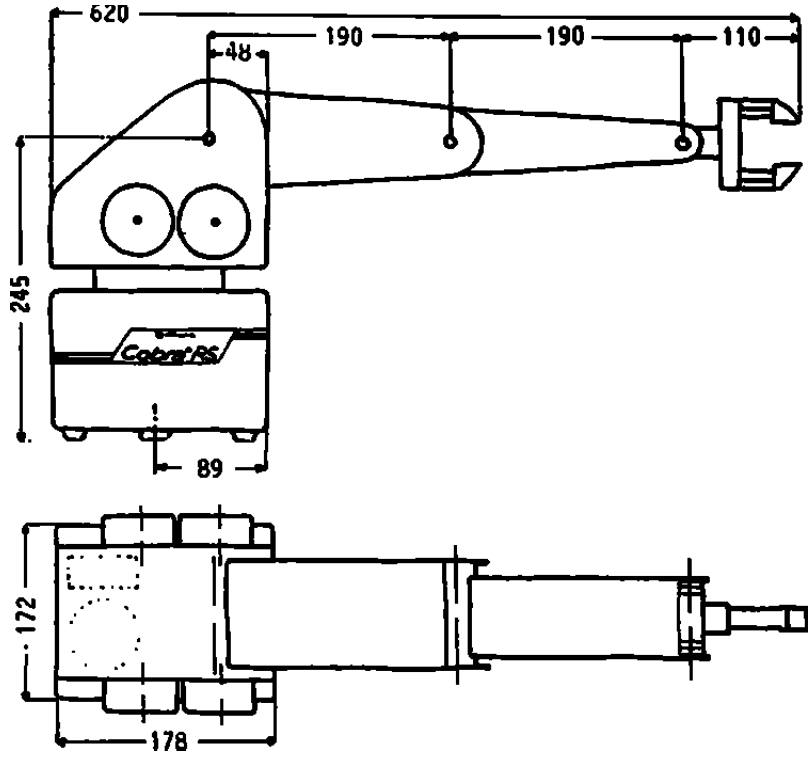


Abbildung 2.2: Ausmaße des Cobra RS

ist, dass jede Achse einzeln bewegt werden kann ohne die Winkel der anderen Achsen zu ändern. Diese Eigenschaft macht die Berechnung der Robotergeometrie einfacher und vereinfacht das Greifen von Objekten, da der Greifer ebenfalls konstant gehalten wird.

Abbildung 2.2 informiert über die Ausmaße des Roboters und Tabelle 2.1 über das Auflösungsvermögen und den Bewegungsbereich jeder einzelnen Achse.

Achsen	Achsen Nr.	Aktions-Bereich	Auflösung/Halbschritt
Drehturm	1	340°	0,115°
Oberarm	2	180°	0,081°
Unterarm	3	180°	0,081°
Hand-Schwenk-Achse	4	180°	0,115°
Hand-Dreh-Achse	5	360°	0,115°
Greifer	6	0-80 mm	0,026 mm

Tabelle 2.1: Aktionsbereiche der einzelnen Achsen

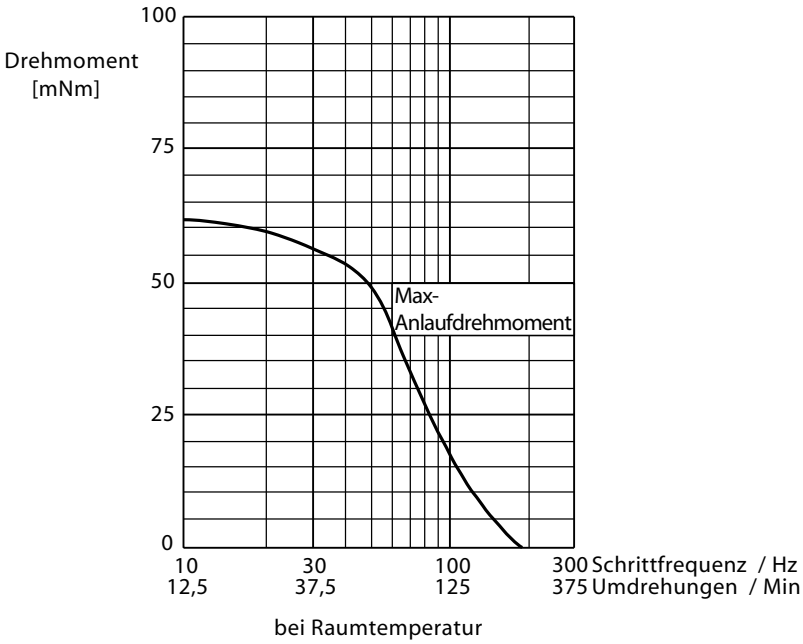


Abbildung 2.3: Anlaufdrehmoment

Bits	Beschreibung
D0	Übertragungsrichtung / Takt
D1..D3	Motor-Adressbits
D4..D7	Schrittmotor-Bits

Tabelle 2.2: Schnittstelle

Die Geschwindigkeit des Cobra RS2 wird durch die Ansteuerungsfrequenz festgelegt und kann durch die Motorschrittwahl (siehe Abschnitt 2.1.2) geringfügig geändert werden.

Die Frequenz bestimmt daneben auch das Drehmoment des Roboters. Bei niedriger Taktung schalten die Schrittmotoren langsamer um und besitzen so für jeden Schritt mehr Kraft. Das Diagramm in Abbildung 2.3 gibt Auskunft über den Zusammenhang von Frequenz und Drehmoment.

2.1.2 Schnittstelle und Ansteuerung

Der Cobra RS2 bietet zur Ansteuerung einen 8 bit Datenport. Eine genaue Übersicht der Belegung der Schnittstelle befindet sich in Tabelle 2.2.

Achse	Bit		
	1	2	3
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
frei	0	0	0
frei	1	1	1

Tabelle 2.3: Adressierung

Die Schnittstelle wurde so konzipiert, dass ein Pseudo-Multiplex hergestellt werden musste, um den Roboter anzusteuern. Das heißt, dass auf den Motor-Adressbits fortlaufend die Adressen ausgegeben werden müssen, die in Tabelle 2.3 dargestellt sind.

Die Ansteuerungsschaltung muss die Schrittmotor-Bits entsprechend der jeweils anliegenden Motoradresse senden.

Die Datenleitung *D0* ist der Takteingang des Roboters. Mit jeder fallenden Flanke werden die Zustände der Bits *D1 – D7* in der Roboterplatine weiterverarbeitet.

Diese Ansteuerung hat den Vorteil, dass beim Zusehen der Eindruck vermittelt wird, dass sich alle Motoren gleichzeitig bewegen können, dies trifft aber nicht zu.

Mit den Schrittmotor-Bits wird der jeweilig adressierte Motor angesteuert. Beim ersten Einschalten des Roboters und der Ansteuerungsschaltung befinden sich die Schrittmotoren in einer beliebigen Position. Deshalb sollten die Motoren zunächst einmal mit einem Schritt initialisiert werden.

Der Zustand *0* (LOW) in den entsprechenden Datenleitungen aktiviert die zugehörige Motorwicklung, *1* (HIGH) schaltet sie frei.

Um die Motoren zu schonen sollte die Adressierung *0000* möglichst vermieden werden, da bei dieser Ansteuerung alle Motorwicklungen mit Strom versorgt werden, der Motor hingegen nicht bewegt wird. Das führt dazu, dass sich der Motor innerhalb kurzer Zeit sehr stark aufheizt.

Zu empfehlen ist deshalb die Adressierung *1111*, da hier alle Wicklungen unversorgt sind.

Schritt	Bit				hex	dez
	7	6	5	4		
1	1	1	0	0	C0	192
1,5	1	1	0	1	D0	208
2	1	0	0	1	90	144
2,5	1	0	1	1	B0	176
3	0	0	1	1	30	48
3,5	0	1	1	1	70	112
4	0	1	1	0	60	96
4,5	1	1	1	0	E0	224

Tabelle 2.4: Halbschrittmuster

Schritt	Bit				hex	dez
	7	6	5	4		
1	1	1	0	0	C0	192
2	1	0	0	1	90	144
3	0	0	1	1	30	48
4	0	1	1	0	60	96

Tabelle 2.5: Vollschrittmuster

Die Motoren können in Voll- oder Halbschritten angesteuert werden, wobei sich die Liste der Vollschritte aus jedem zweitem Halbschritt zusammensetzt.

Grundsätzlich kann man sagen, dass die Motoren bei Halbschrittansteuerung genauer, kraftvoller und ruhiger laufen.

2.2 El Camino DIGILAB 10K10

Die Firma El Camino GmbH gibt unter der Bezeichung DIGILAB 10K10 ein Entwicklungsboard mit integriertem Altera FLEX 10K10 heraus.

Neben der Programmierung des enthaltenen Chips, sind zahlreiche weitere Verwendungsmöglichkeiten vorhanden. Dazu im folgenden Abschnitt mehr.

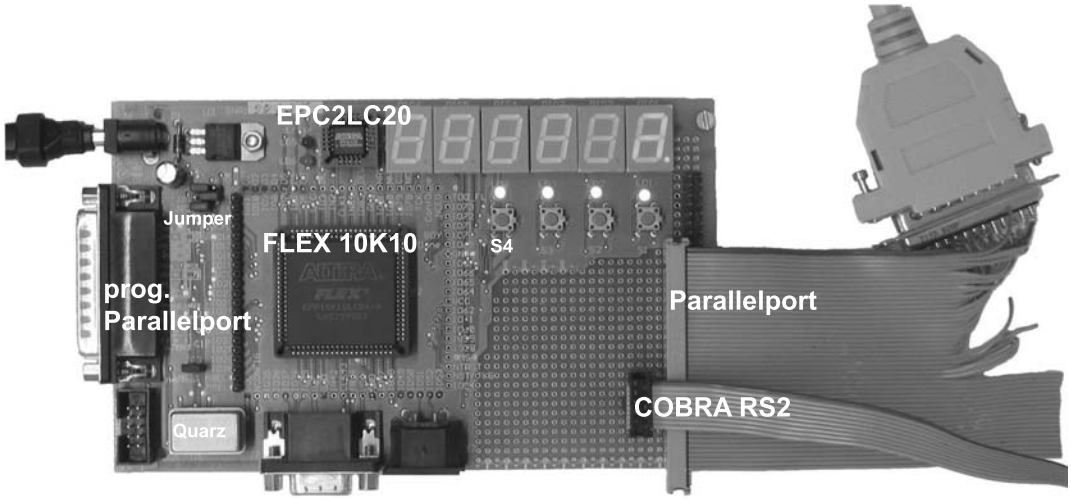


Abbildung 2.4: DIGILAB 10K10 von El Camino

2.2.1 Die Entwicklungsplatine

Die Platine ist das DIGILAB 10K10 von El Camino (siehe Abbildung 2.4) und bietet einen einfachen Einstieg in die Welt der programmierbaren Bausteine. Auf ihr befindet sich in der Mitte der Altera Flex 10K10 CPLD[1], links eine Parallelport-Schnittstelle zur Konfiguration, einen VGA-Stecker, eine mehrstellige Siebensegment-Anzeige und viele weitere Ausgabemöglichkeiten über die Ausgänge des Chips.

Der Chip kann über ein Parallelportkabel und die Altera Software *MAX+plus II* programmiert werden. Mit *MAX+plus II* ist es auf einfache Art und Weise möglich entweder Gatterschaltungen selber zu zeichnen und miteinander zu verbinden oder VHDL-Code zu schreiben, der dann zur Programmierung des Chips entsprechend umgesetzt wird.

Für diese Diplomarbeit wurden alle wichtigen Module zur Ansteuerung des Cobra RS2 mit VHDL beschrieben, in einer graphischen Oberfläche unter *MAX+plus II* verbunden und den Ein-/Ausgängen des FLEX 10K10 zugeordnet.

[1]Complex Programmable Logic Device

Signal	Pin
D0	50
D1	51
D2	52
D3	53
D4	17
D5	18
D6	19
D7	21
STROBE	36
BUSY	25
GND	GND

Tabelle 2.6: Parallelport

Signal	Pin
D0	49
D1	17
D2	18
D3	19
D4	8
D5	7
D6	6
D7	5
GND	GND

Tabelle 2.7: Roboter-Schnittstelle

Zur Ansteuerung des Roboters musste die vorliegende Platine um 2 Schnittstellen erweitert werden:

- Eine für die Kommunikation mit dem PC über die parallele Schnittstelle, da über die schon vorhandene Parallelport-Schnittstelle nur der Baustein programmiert werden kann.
- Eine zweite für die Kommunikation mit dem Roboter.

Die parallele Schnittstelle des PCs benutzt die Datenleitungen *D0 – D7* zur Übertragung der Daten zur Platine, *STROBE* als (asynchronen) Takteingang für das Parallelport-Modul, *GND* als Masse und *BUSY* als Rückmeldung für den PC, ob weitere Daten gesendet werden können.

Die Schnittstelle für den Roboter ist entsprechend belegt; *D0 – D7* als Datenleitungen und *GND* als Masse. [1]

S4 in Abbildung 2.4 bezeichnet den Taster zur Initialisierung der Schaltung. Dieser sollte nach dem Einschalten der Platine einmal kurz betätigt werden.

2.2.2 Altera FLEX 10K10 CPLD

Alteras FLEX 10K Module bestehen aus rekonfigurierbaren CMOS SRAM Bausteinen. Diese sog. FLEX[2] Architektur bietet alle nötigen Möglichkeiten um sogenannte *Megafunctions*[3] zu implementieren.

Der FLEX 10K10 ist vollständig programmierbar und hat eine Kapazität von 10000 programmierbaren Gattern und 83 Ein- und Ausgängen.

Jeder Chip besteht aus einer eingebetteten Matrix und einer Logik-Matrix. Die eingebettete Matrix bietet verschiedene Möglichkeiten, um diverse Speicherfunktionen, komplexe Logikfunktionen wie z.B. DSP[4] oder Microcontroller zu realisieren und zu nutzen.

Die Logik-Matrix wird genutzt, um grundlegende Logikbausteine, wie Zähler, Addierer oder Multiplexer zu verwirklichen.

Die eingebettete Matrix besteht aus einer Anzahl von EABs[5], jedes davon besitzt 2048 Bits, welche zur Realisation von RAM, ROM oder FIFO-Funktionen verwendet werden können. Zur Implementierung von Logikbausteinen bietet jeder EAB 100-600 Gatter. Jedes EAB kann entweder alleinstehend oder zusammen mit anderen genutzt werden, um größere Funktionen zu realisieren. Dadurch sind die FLEX 10K CPLD sehr flexibel einsetzbar.

Die Bausteine können entweder direkt über den PC konfiguriert werden oder über einmalig programmierte Konfigurationsbausteine wie z.B. der EPC2, welcher auch auf der Platine zu finden ist. Diese konfigurieren den FLEX 10K mit einem seriellen Datenstrom sobald die Platine Strom bekommt. [2]

2.3 MAX+plus II

Die *MAX+plus II* Entwicklungssoftware von Altera bietet eine komplette Designumgebung, um beliebige Logikschaltungen zu entwerfen.

[2] **F**lexible **L**ogic **E**lement Matri**X**
[3] komplexer Funktionsblock (eine Art „Black Box“, der mit Gatterschaltungen und anderen Funktionen kombiniert werden kann
[4] digital signal processing
[5] embedded array block

Es wird eine graphische Umgebung zur Verfügung gestellt, womit jede Gatterschaltung einfach realisiert werden kann, desweiteren ein Texteditor z.B. zur Beschreibung von VHDL-Modulen.

Dadurch können verschiedene Module, ob Gatterschaltungen oder andere Dateiformate (zur Auswahl stehen u.a. *VHDL*, *AHDL*, *Verilog HDL* oder *OrCAD*) miteinander kombiniert werden, um auch sehr komplexe Schaltungen zu erstellen.

Daneben gibt es die Möglichkeit fertige Module zu nutzen, z.B. können mit Hilfe eines Assistenten Speicherbausteine und Logikmodule erstellt werden, die mittels der vorhandenen *Megafunctions* vom Aufbau sehr einfach sind.

Ist die Schaltung oder das Modul erst einmal fertig, kann diese im *Simulator* simuliert werden, um direkt das Verhalten zu analysieren. Der Platzbedarf des jeweiligen Projekts auf dem Chip ist jederzeit über den *Floorplan Editor* ersichtlich.

Mit Hilfe von *MAX+plus II* können die erstellten Schaltung auch direkt kompiliert werden. Danach besteht die Möglichkeit den entsprechenden Chip z.B. den FLEX 10K10 auf einer geeigneten Platine zu programmieren. [5]

2.3.1 Programmierung

Um das Digilab 10K10, genauer gesagt den FLEX 10K10 bzw. den EPC2, zu programmieren, bedarf es je nach Chip unterschiedlicher Jumper- und Programmeinstellungen. In Abbildung 2.8 ist eine Übersicht über die richtigen Jumper-Einstellungen für beide Konfigurationen dargestellt. Die unter [1] angegebenen Stellungen für JP4 sind nicht korrekt.

Jumper	FLEX 10K	EPC2
JP1	geschl.	offen
JP2	geschl.	geschl.
JP3	geschl.	geschl.
JP4	A	B

Tabelle 2.8: Jumper-Einstellungen zur Programmierung

Für die Programmierung des EPC2 muss folgender Ablauf eingehalten werden:

1. Schaltung mit *Compiler* kompilieren

2. *Programmer* aufrufen
3. *Options → Select Device...* öffnen und den EPC2 auswählen
4. *Examine* im *Programmer* aufrufen
5. Schaltung erneut mit *Compiler* kompilieren
6. *Program* im *Programmer* aufrufen

Die Programmierung des FLEX 10K10 erfolgt ganz normal.

Auf einem PC mit Windows NT oder Windows 2000 ist es nötig einen Treiber für den *Altera Byteblaster*, der auch Teil der *MAX+plus II* Installation ist, zu installieren. Der *Byteblaster* wird benutzt, um die Konfigurationsdaten über die parallele Schnittstelle des PCs an die JTAG Schnittstelle des FLEX Bausteines zu senden. Diese Installation ist für PCs mit Windows 95 oder 98 nicht nötig.

Eine genaue Installationsanleitung des Treibers findet sich unter [1].

2.4 Aufnahmeeinheit

Die Aufnahmeeinheit besteht aus Kamera und Framegrabber zur direkten Bildaufnahme, außerdem aus einem Gestell an dem Kamera und Roboter befestigt sind.

2.4.1 Kamera JAI CV-S3200 & Framegrabber DFG/LC1

Zur Bildaufnahme wurde die Kamera JAI CV-S3200 und der Framegrabber DFG/LC1 benutzt.

Die Kamera ist eine TV-Farbkamera mit einer maximalen Auflösung von 768 x 562 Pixeln. Außerdem besitzt diese einen Infrarot-Filter, der für die richtige Farberkennung der Münzen sehr wichtig ist. Ohne den Filter sind die Münzen im Farbraum nicht so gut voneinander zu trennen.

Im Menü der Kamera werden verschiedene Einstellungen geboten, u.a. Weißabgleich, Shutter[6] und AGC[7]. Auffallend bei der Kamera ist der Weißabgleich, der auf manuell

[6] elektronische Blende der Kamera
[7] automatic gain control

Abbildung 2.5: Aufbau

gestellt, sich auf die vorherrschende Farbe im Bild justiert und dadurch die Farbraum-Darstellung verändert. [3]

Für die Kamera wurde das 12,5 mm Objektiv C21211 der Firma Cosmicar/Pentax verwendet.

Der Framegrabber DFG/LC1 von *The Imaging Source* wurde zur Aufnahme über den „Video for Windows“-Treiber angesteuert. [4]

2.4.2 Das Gestell

Das Gestell, ersichtlich in Abbildung 2.5, ist eine stabile Befestigung für Kamera und Roboter.

Dieser Punkt ist sehr wichtig, da beide Koordinatensysteme (Kamera und Roboter) miteinander in Beziehung stehen und durch Vibrationen des Cobra RS2 keine zusätzlichen

Ungenauigkeiten in das System gelangen dürfen.

Dadurch wird der Arbeitsraum des Gesamtsystems festgelegt und Bereiche zum Ablegen, der zu erkennenden und sortierenden Münzen, definiert.

Aufgebaut ist das Gestell aus quadratischen Aluminium-Stangen, an denen sich beliebig Winkel zur Befestigung anderer Stangen anbringen lassen.

2.5 DBS Ad Oculos 3.1

Ad Oculos (zu deutsch: „vor Augen führen“) von DBS ist ein Programm zur Bildverarbeitung. Es ermöglicht ‚klassische‘ Bildverarbeitungsfunktionen und ihre Wirkungen auszuprobieren. Auf Grund des modularen Aufbaus der einzelnen Funktionen ist es besonders für den Einstieg geeignet.

Auf einfache Art und Weise können schnell komplexe Signalflusspläne zusammengebaut werden, wobei jedes Modul durch eigene Ausgabebilder schnell überprüft werden kann.

Das Softwarepaket enthält neben dem eigentlichen Programm, welches eher eine graphische Oberfläche ist, eine Reihe von fertigen Bildfunktionen mit Sourcecode.

So können erste Bildverarbeitungsaufgaben erstellt und ausgeführt werden. Die Anzahl der Funktionen läßt sich dabei beliebig erweitern.

Die Möglichkeit das Programm benutzerdefiniert zu erweitern ist der größte Pluspunkt dieser Software. Weitere Funktionen können z.B. mit *C* geschrieben werden. Der Rahmen für neue Plugins ist in den mitgelieferten Sourcen ersichtlich.

Aus neuen Funktionen wird eine DLL erstellt, welche schnell in die Oberfläche eingebunden werden kann.

Zudem bietet die Bildverarbeitungsentwicklungsumgebung des Labors die Möglichkeit für *Ad Oculos* geschriebene Funktionen zu Win32-Anwendungen zu kompilieren, so dass Funktionen auch ohne das graphische Interface ausgeführt werden können.

Von dieser Möglichkeit wurde für die Diplomarbeit auch Gebrauch gemacht. So kann durch den Aufruf einer Batch-Datei die komplette Bildverarbeitung aufgerufen und der Roboter angesteuert werden. [6, 7]

3 Ansteuerungsschaltung

In diesem Kapitel werden die Funktionen aller wichtigen Module zur Ansteuerung des Roboters beschrieben sowie die Funktionsweise der Schaltung erklärt.

3.1 Übersicht

Die Ansteuerungsschaltung (siehe Abbildung 3.1) bildet die Schnittstelle zwischen PC und Roboter. Zur Einführung ein kurzer Überblick über die Schaltung, um die nachfolgende Beschreibung der Module besser verstehen zu können.

Die detaillierte Beschreibung der Gesamtschaltung findet sich in Abschnitt 3.4.

Teilt man die Schaltung in 2 Bereiche auf, so steht auf der einen Seite die Schnittstelle zum PC (Module *PARPORT* und *READY*), auf der anderen die Schnittstelle zum Roboter (Module *MUXER* und *ADRESSE*).

Das *PARPORT*-Modul empfängt die Daten vom PC und programmiert die *COUNTER* jedes einzelnen Motors mit Schritten und stellt die Untersetzung ein.

Nach Programmierung der Schritte gibt das Modul *ADRESSE* die Motoradressen fortlaufend als Ringzähler aus, diese wiederum ist Eingangssignal weiterer Module, da die einzelnen Schritte dem entsprechenden Motor zugeordnet werden müssen.

Die *SCHRITTMOTOR* bekommt durch den *SCHALTER* je nach anliegender Adresse eine Taktflanke und springt mit jeder Flanke um einen Schritt weiter, solange der *COUNTER* nicht auf 0 ist.

Der *COUNTER* bekommt einen, abhängig von der Untersetzung, verlangsamten Takt, zählt die programmierten Schritte herunter und gibt währenddessen die Schritte der *SCHRITTMOTOR* aus.

Im letzten Schritt entscheidet der *MUXER*, je nach anliegender Adresse, welcher Schritt zur angelegten Adresse gehört und schaltet diesen auf die 4 Bits zur Schrittsteuerung der Roboterschnittstelle durch.

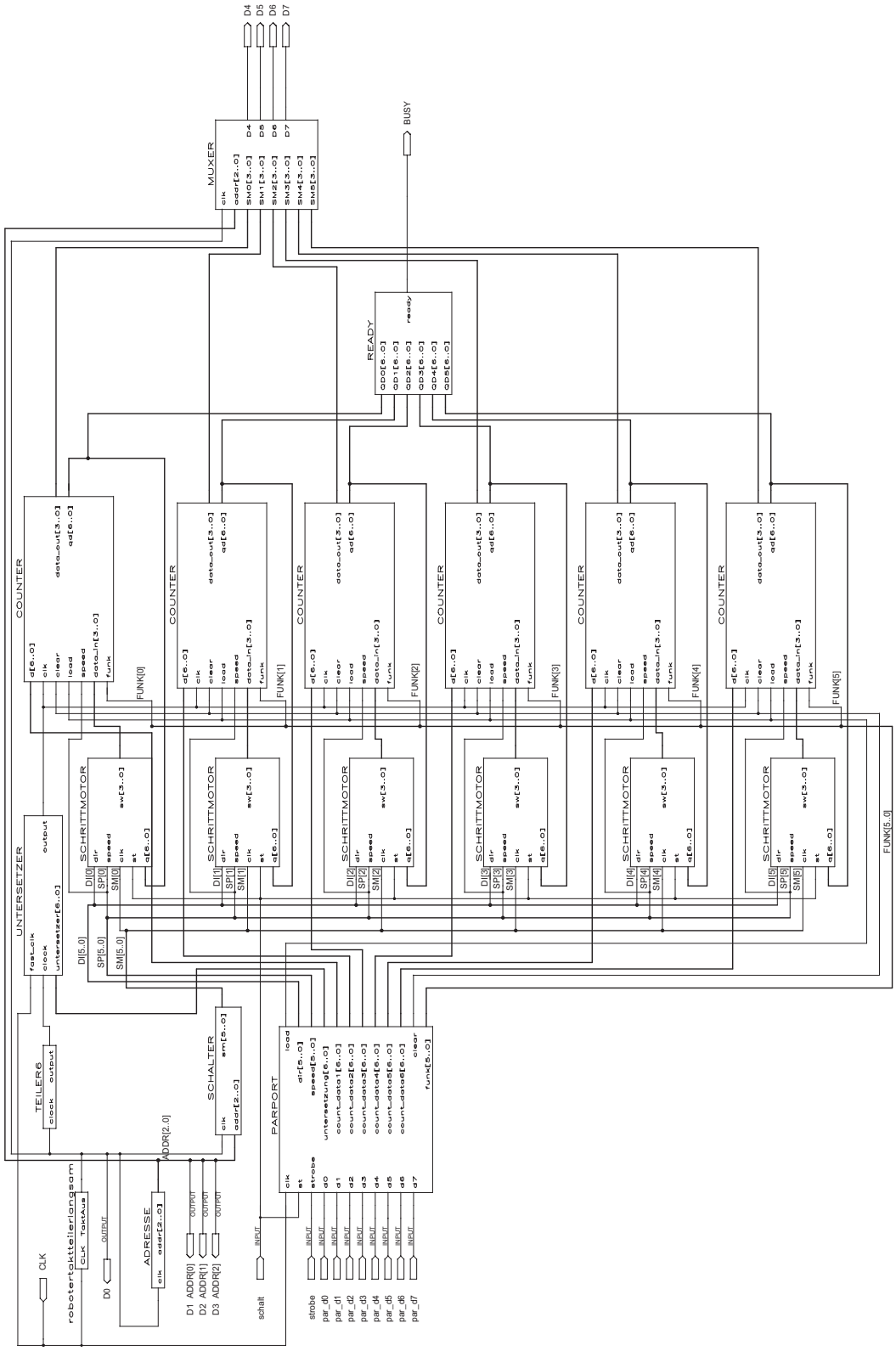


Abbildung 3.1: Ansteuerungsschaltung

3.2 eigene Module

Die folgenden Seiten enthalten eine Aufstellung der verwendeten VHDL-Module zum Aufbau der Ansteuerungsschaltung, die durch meine Arbeit entstanden sind.

3.2.1 Taktteiler

Der *Taktteiler* ist kein VHDL-Modul, da sich dieser mit einer sehr einfachen Logikschaltung realisieren lässt und der VHDL-Entwurf ungleich viel Speicherplatz verbraucht. Er besteht aus der Aneinanderreihung von 15 JK-Flip-Flops.

Die Eingänge J und K der Flip-Flips sind auf *HIGH* geschaltet. Dadurch wird mit jeder ankommenden Taktflanke der Ausgang Q negiert. Die Flip-Flops sind so zusammengeschaltet, das der Ausgang Q jedes einzelnen am Takteingang des nächstfolgenden anliegt. Es ergibt sich ein 15 bit Zähler, dessen Ausgang alle 2^{15} Takte negiert wird.

So wird der Systemtakt der Ansteuerungsplatine (in diesem Fall 10 MHz) auf einen Takt (305 Hz) heruntergesetzt, mit dem der Cobra RS2 angesteuert wird.

$$\begin{aligned} f_{cobra} &= \frac{f_{system}}{2^n} \\ &= \frac{10\,\mathrm{MHz}}{2^{15}} = 305\,\mathrm{Hz} \end{aligned}$$

Um die Arbeitsgeschwindigkeit der Schrittmotoren zu erhalten, muss dieser Takt noch durch den Faktor 6 geteilt werden: $f_{motor} = \frac{305\,\mathrm{Hz}}{6} = 51Hz$.

Die Anzahl der Flip-Flops wurde so gewählt, dass das Drehmoment nicht zu gering für die Oberarmbewegung bei akzeptabeler Geschwindigkeit ist(siehe Abbildung 2.3).

3.2.2 Schrittmotor-Steuerung

Die *Schrittmotor-Steuerung* ist ein steuerbarer Ringzähler mit 3 Steuereingängen: *‚dir‘* für die Richtung der Generierung, *‚speed‘* für die Ausgabe in Voll- oder Halbschritten und *‚q‘* als die Rückmeldung vom Zähler.

Jede steigende Taktflanke am Eingang *‚clk‘* bewirkt, dass der interne Zähler um ein Element in der Schrittliste weiterspringt, solange *‚q‘* (Zählvariable des Zählers) ungleich 0 ist.

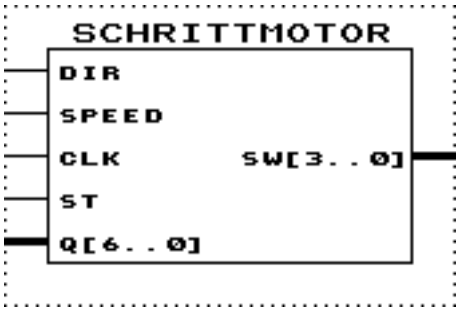


Abbildung 3.2: Modul zur Generierung der Schrittmotor-Schritte

Dadurch wird verhindert, dass die Schrittliste weiter verarbeitet wird und der jeweilige Motor in der Stellung bleibt, wenn der Zählwert 0 beträgt.

Andernfalls könnte sich bei einer erneuten Ansteuerung das Problem ergeben, dass der Schrittmotor in einen anderen Zustand springt, der in der Liste nicht der letzten Schritt-Stellung folgt. Die Bewegung des Roboters könnte somit ungenau werden.

Der Eingang *‚dir‘* bestimmt die Ausgaberichtung der Schrittliste, entweder von oben nach unten oder umgekehrt.

‚speed‘ bestimmt, ob Halbschritte (1) oder Vollschritte (0) ausgegeben werden.

Mit *‚st‘* kann das Modul initialisiert werden, da zu Beginn ein undefinierter Zustand vorliegt. Solange *‚st‘* auf *0* (LOW) ist, werden unabhängig von *‚q‘* neue Schritte generiert.

3.2.3 Zähler

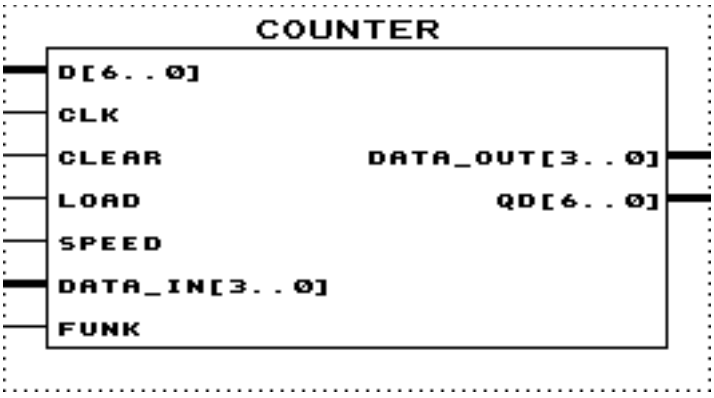


Abbildung 3.3: Zählmodul

Dieses Modul ist ein programmierbarer 7 bit Zähler d.h. ein Bereich von 127 bis 0. Mit jeder steigenden Flanke an *‚clk‘* wird der interne Zählwert niedriger.

Bevor gezählt werden kann muss der *Zähler* auf einen Wert gesetzt werden. Dies geschieht durch den vom Parallelport-Modul anliegenden Zählwert *‚d‘* und dem *‚load‘*-Signal. *‚d‘* wird in den internen Speicher übertragen, sobald an *‚load‘* eine steigende Taktflanke anliegt. Heruntergezählt wird, sobald *‚load‘* wieder auf *0* ist. Dabei wird der Eingang *‚data_in‘* ständig auf *‚data_out‘* ausgeben. Dies geschieht völlig unabhängig vom Takt. Dieser wird nur zum Herunterzählen benötigt.

Daneben gibt es die weiteren Eingänge *‚clear‘*, *‚speed‘* und *‚funk‘*.

Mit dem Anlegen eines High-Levels an *‚clear‘* wird der interne Zählwert sofort auf 0 gesetzt.

‚speed‘ beeinflußt die Größe der Zählschritte und liegt sowohl an der *Schrittmotor-Steuerung* als auch am Zählmodul an. Vom PC werden die zu fahrenden Halbschritte auf die Platine übertragen. Werden von *Schrittmotor-Steuerung* Halbschritte ausgegeben (*‚speed‘* auf 1), so wird der Zählwert immer um 1 vermindert. Sollen Vollschritte gefahren werden (*‚speed‘* auf 0), wird der Zählwert immer um 2 vermindert, da zwei Halbschritte einem Vollschritt entsprechen.

Auf *‚qd‘* wird der jeweilige Stand des internen Zählwertes ausgegeben. Dieser wird von der *Schrittmotor-Steuerung* und vom *Ready*-Modul benötigt.

Liegt an *‚funk‘* ein HIGH-Level an und ist der Zählwert 0, wird auf *‚data_out‘ 1111* ausgegeben, andernfalls *‚data_in‘*. So können einzelne Motoren im jeweiligen Zustand zur Stabilisation gehalten oder freigestellt werden.

3.2.4 Teiler6

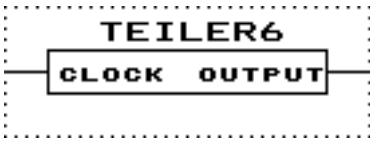


Abbildung 3.4: Modul zur Taktvorverarbeitung für den Untersetzer

Der *Teiler6* ist ein Taktteiler, der den Takt um den Faktor 6 heruntersetzt. So ist die Perdiodenlänge des Ausgangstaktes 6 mal länger als die des Eingangstaktes. 6 deshalb, weil es 6 Motoradressen gibt, die mit jeder Taktflanke wechseln.

Dieses Modul ist eine Vorverarbeitung des Taktes für den *Untersetzer*.

3.2.5 Untersetzer

Abbildung 3.5: Modul zur Generierung der Taktuntersetzung

Der *Untersetzer* ist ein konfigurierbarer Taktteiler, der den an *‚clock‘* anliegenden Takt um einen Faktor heruntersetzt.

Am Eingang *‚fast_ clk‘* liegt ein schneller Takt (Systemtakt) an, damit bei einer Änderung vom *‚untersetzer‘*-Signal, diese möglichst schnell intern übernommen werden kann.

Dieses Modul ist in gewisser Weise auch wieder ein Zähler. Mit jeder steigenden Taktflanke an *‚clock‘* wird der interne Zählwert um 1 erhöht. Ist dieser gleich dem an *‚untersetzer‘* anliegenden Wert, so wird der Status des Ausgangssignals *‚output‘* negiert. D.h. war *‚output‘* bisher *1* wird es *0* und andersherum. Liegt an *‚untersetzer‘* der Wert 0 an, so wird der an *‚clock‘* anliegende Takt unverändert auf *‚output‘* ausgegeben. Bei allen anderen Werten ist die Ausgabe-Taktlänge um den Faktor ‚2 x *untersetzer‘* größer als *‚clock‘*. Ist z.B. *‚untersetzer‘* = 7, dann ist die Taktperiode von *‚output‘* 14 mal länger.

Der *Untersetzer* wird in der Schaltung benötigt, weil die Kapazität des FLEX 10K10 nicht ausreichend ist, um den Zählbereich der *Zähler* so zu erweitern, dass alle maximal möglichen Schritte jedes Schrittmotors ausgegeben werden können.

Andernfalls wären 14 bit statt 7 bit *Zähler* nötig gewesen. Diese würden den zur Verfügung stehenden Speicherplatz der Platine überschreiten.

3.2.6 Ready-Modul

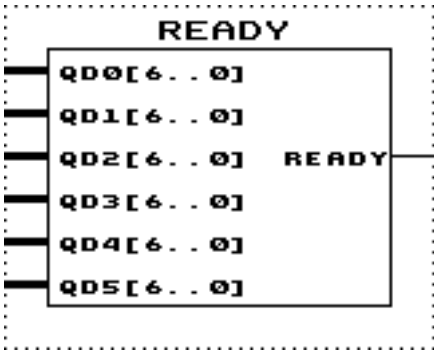


Abbildung 3.6: Modul zur Ausgabe des Busy-Status an den PC

Das *Ready*-Modul liefert den PC über die parallele Schnittstelle (*BUSY*-Signal) eine Rückmeldung, ob die Schaltung gerade den Roboter ansteuert oder bereit für den Empfang von neuen Daten ist.

Das Modul gibt auf *‚ready‘* eine *1* aus, wenn alle Zählereingänge (*‚qd1‘ – ‚qd6‘*) auf 0 sind, andernfalls eine *0* und funktioniert somit wie ein großes *NAND*-Gatter.

3.3 korrespondierende Module

In diesem Abschnitt sind die Module beschrieben, die durch die Arbeit von Andreas Koch entstanden sind. Für eine genauere Beschreibung sei auf [8] verwiesen.

3.3.1 Motoradressen-Generierung

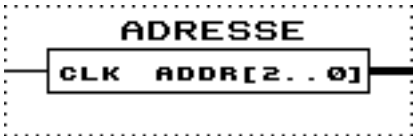


Abbildung 3.7: Modul zur Erzeugung des Adress-Multiplex

Die *Motoradressen-Generierung* ist ein Ringzähler.

Da der Cobra RS2 nicht einfach mit einer Motoradresse und der jeweiligen Schrittmotoreinstellung angesprochen werden kann, muss ein Pseudo-Multiplex erzeugt werden.

Das heißt, dass an der Roboterschnittstelle eine Motoradresse nach der anderen anliegen muss.

Das Modul springt mit jeder steigenden Taktflanke jeweils auf die nächstfolgende Adresse um. Je nach Zählwert wird die dazugehörige Motoradresse aus einer Liste ausgegeben.

3.3.2 Parallelport-Schnittstelle

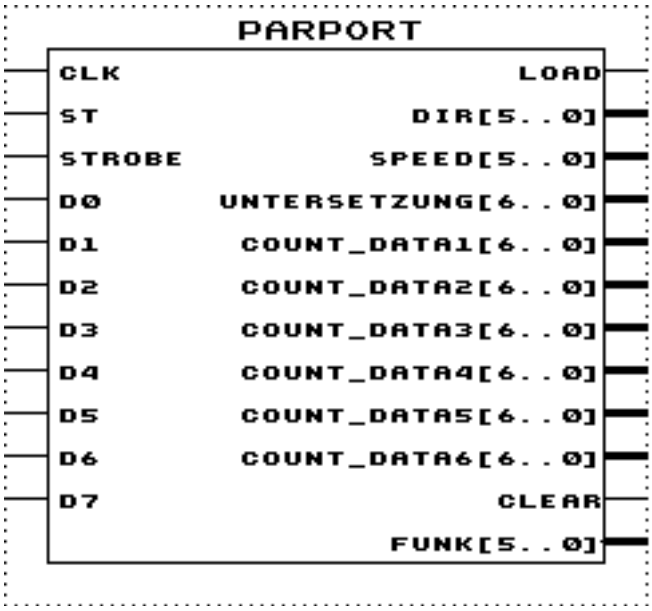


Abbildung 3.8: Modul für die Parallelportschnittstelle des PCs

Dieses Modul ist die direkte Verbindung zur parallelen Schnittstelle des PCs. Hier werden die Ausgänge der parallelen Schnittstelle *STROBE* und *D0 – D7* entsprechend für die Roboteransteuerung umgesetzt und an die entsprechenden Module weitergeleitet.

Das Parport-Modul gibt dabei zum einen die Werte der *Zähler* und des *Untersetzers* aus, zum anderen die Geschwindigkeit und die Richtung für die Generierung der Schrittmotor-Schritte.

3.3.3 Motoradressen-Schalter

Der *Motoradressen-Schalter* ist ein 6-zu-1 Multiplexer.

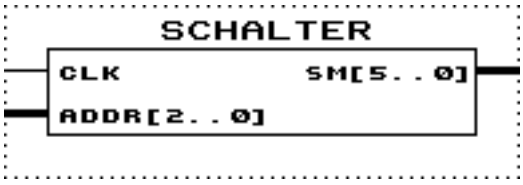


Abbildung 3.9: Modul zur Ansteuerung der Schrittmotoren

Hier wird der Takt für die *Schrittmotor-Steuerung* erzeugt. Je nach anliegender Motoradresse wird der entsprechende Motor angesteuert.

3.3.4 Muxer

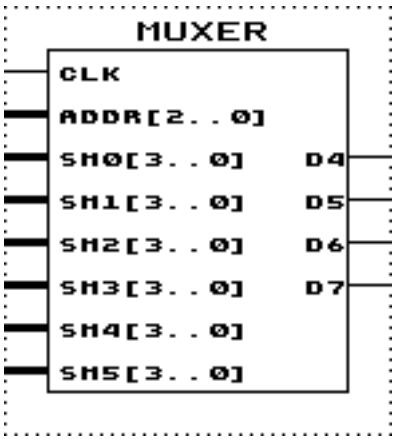


Abbildung 3.10: Modul zur Kombination der Schrittmotor-Schritte

In diesem Modul werden die einzelnen *Zähler*-Ausgänge vereint und mittels des Motoradresseingangs *‚addr‘* entsprechend der angesteuerten Adresse, der zugehörige Schrittmotor-Schritt ausgegeben.

Die 4 Ausgabebits gehen direkt auf die Roboterschnittstelle (*D4 – D7*).

3.4 Funktionsweise der Schaltung

Nachdem alle Module einzeln genau erklärt worden sind, wird nach der kurzen Übersicht in Abschnitt 3.1, detailliert auf die Funktionsweise der Schaltung eingegangen.

Dies soll, wie oben angedeutet, der Übersichtlichkeit halber in zwei Teilbereichen geschehen:

- **Programmierung und Rückgabe**
  Dieser Punkt erläutert die Kommunikation des PCs mit der Platine genauer.
- **Ablauf und Ausgabe**
  Dieser Abschnitt soll den inneren Ablauf zur Generierung der Schritte und deren Ausgabe genauer beschreiben.

3.4.1 Programmierung und Rückgabe

Unter Programmierung der Schaltung ist die Beschaltung der Zählwerteingänge der einzelnen Zähler, der Richtungs- und Geschwindigkeitsleitungen, der Load-,Clear- und Halteleitungen sowie der Untersetzung zu verstehen.

Dazu stehen dem PC zwei verschiedene Datenblöcke zu je 8 bit zur Verfügung. Der erste Datensatz setzt Geschwindigkeit, Richtung und Haltezustand der einzelnen Motoren. Er gibt das Signal zum Setzen und Reset der Zähler (*‚load‘* bzw. *‚clear‘*) und legt die Motoradresse bzw. Untersetzeradresse fest. Der zweite Datensatz beinhaltet den zugehörigen Zähl- bzw. Untersetzungswert.

Die Rückgabe der Schaltung wird durch das sog. *Ready*-Modul realisiert. So ist beim Auslesen des *BUSY*-Signals der parallelen Schnittstelle erkennbar, ob die Schaltung noch Schritte an die Roboterschnittstelle ausgibt.

Diese Rückmeldung ist besonders für die spätere Steuerung des Cobra RS2 wichtig, da erkennbar sein muss, wann die nächste Position ohne Schrittverluste angefahren werden kann.

3.4.2 Ablauf und Ausgabe

Die einzelnen Module der Schaltung sind, nachdem die Schaltung erstmals mit Spannung versorgt wird, nicht initialisiert. Aus diesem Grund muss dieses einmalig manuell gemacht werden.

Um die Initialisierung vorzunehmen, wird auf dem Eingang *‚schalt‘* einmalig eine *0* gesendet und danach auf *1* gehalten. *‚schalt‘* ist auf der Platine mit dem Schalter *S4* verbunden

(siehe Abbildung 2.4), so dass ein kurzer Druck darauf ausreicht.

Der Ablauf soll exemplarisch mit einem Motor (Motor1) dargestellt werden, da die anderen Motoren nach demselben Prinzip gesteuert werden.

Sobald die Initialisierung und Programmierung geschehen und das Load-Signal gesetzt worden ist, beginnt der eigentliche Ablauf der Schaltung.

Die Untersetzung hat nun beispielsweise den Wert 2, d.h. 4-fache Untersetzung des Eingangstaktes. Das *COUNTER*-Modul von Motor 1 hat den Zählwert 19 zugewiesen bekommen.

Der Takt für die Schaltung entsteht durch Heruntersetzen des Systemtaktes durch den *TAKTTEILER*. Dieser Takt wird für den Untersetzer mit dem *TEILER6* vorverarbeitet, da durch die Ansteuerungsart die einzelnen Motoren nur alle 6 Takte angesteuert werden.

Das Modul *ADRESSE* springt mit jeder steigenden Taktflanke durch die Motoradressen und gibt diese aus. Dieser Ausgang wird vom *SCHALTER* und vom *MUXER* benötigt, um jedem *SCHRITTMOTOR*-Modul zur zugehörigen Adresse eine Taktflanke zu geben.

Nun zur *SCHRITTMOTOR*-Modul von Motor 1. Dieses bekommt eine Taktflanke und springt intern, wenn der Zählwert des zugehörigen *COUNTER*-Moduls ungleich 0 ist, entweder einen Halbschritt/Vollschritt (*‚speed‘*) vor oder zurück (*‚dir‘*). Diese werden auf *‚data_out‘* ausgegeben.

Der *COUNTER* zählt währenddessen mit dem *UNTERSETZER*-Takt herunter, je nach *‚speed‘* um 1 oder 2. Dabei werden die Schrittmotor-Schritte von *‚data_in‘* auf *‚data_out‘* ausgegeben. Hat der Zählwert die 0 erreicht so wurden $4 \cdot 19 = 76$ Halbschritte bzw. 38 Vollschritte (1 Vollschritt = 2 Halbschritte) ausgegeben. Danach wird je nach Wert des Eingangs *‚funk‘*, *‚data_in‘* oder *1111* ausgegeben.

Die Zählausgänge der einzelnen *COUNTER* gehen auf den *MUXER*, der entscheidet, wann welcher Schritt an welchen Motor ausgegeben wird. Konkret heißt das: Liegt die Adresse von Motor1 an, so wird die Ausgabe vom *COUNTER* für Motor 1 auf den Bits *D4 – D7* der Roboter-Schnittstelle ausgegeben.

Abbildung 3.11 enthält eine Gegenüberstellung verschiedener Ausgaben der Schaltung an die Roboter-Schnittstelle. Die Roboter-Platine taktet auf die fallenden Flanke von *D0*. *D1 – D3* zeigen die sich ändernden Adressen der einzelnen Motoren.

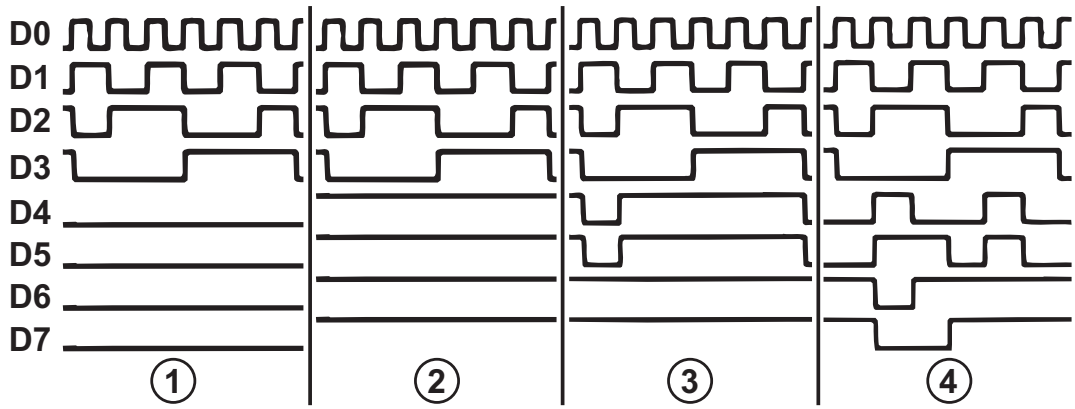


Abbildung 3.11: verschiedene Ausgabezustände der Roboterschnittstelle

Abgebildet sind dabei immer dieselben 6 Takte mit unterschiedlichen Schrittausgaben auf den Bits *D4 – D7*.

1 zeigt die Ausgabe für eine nicht initialisierte Schaltung. Auf *D4 – D7* wird für jeden Motor *0000* ausgegeben.

2 zeigt den Status für eine initialisierte Schaltung, bei der alle Motoren nicht gehalten werden. Also mit *1111* auf frei gesetzt sind.

3 zeigt den Zustand eines Motors mit dem Schritt *0011*. Alle anderen Motoren sind frei.

4 zeigt, dass alle Motoren in einen bestimmten Zustand sind. Dies kann sowohl ein Ausschnitt aus einer laufenden Ausgabe sein, als auch der Haltezustand für alle Motoren.

4 Koordinatentransformation

Die Koordinatentransformation ist neben der Ansteuerungsschaltung und -software ein weiterer zentraler Punkt zur Steuerung des Roboters. Mit Hilfe der Transformation wird das kartesische Koordinatensystem in Roboterkoordinaten (Gelenkwinkel) umgerechnet (siehe Abbildung 4.1).

Zur Berechnung gibt es die Möglichkeit aus zwei Ansätzen zu wählen:

- Transformation nach Denavit-Hartenberg
- geometrischer Ansatz

Bei dem vorliegenden Roboter war der geometrische Ansatz der naheliegendere, da die Transformation nach Denavit-Hartenberg ein relativ komplexes Rechenverfahren ist, bei dem für jedes Gelenk eine Matrix aufgestellt wird, die später miteinander verrechnet werden.
Ein weiterer Pluspunkt der geometrischen Lösung ist, dass diese wesentlich anschaulicher ist.

Im folgenden Kapitel werden zum einen die Formeln für die Rückwärtskinematik (Umrechnung kartesisches Koordinatensystem in Roboterwinkel), als auch für die Vorwärtskinematik (Umrechnung Roboterwinkel in kartesisches Koordinatensystem) hergeleitet.

Die entsprechenden Formeln wurden in einer *Dynamic Link Library* zusammengefasst, um sie für das Ansteuerungsmodul (*Cobradll*) zugänglich zu machen.
Die Beschreibung der einzelnen Funktionen der DLL findet sich ebenfalls in diesem Kapitel.

Der Nullpunkt des kartesischen Koordinatensystem befindet sich auf dem Schnittpunkt der Drehturm-Achse (in Abbildung 4.1 die Z-Achse) mit der Arbeitsfläche, auf der Roboter

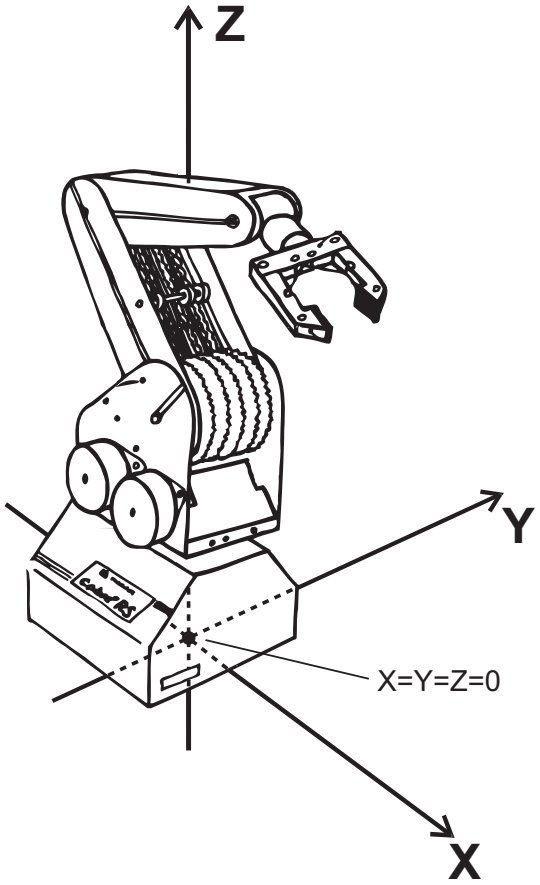


Abbildung 4.1: kartesisches Koordinatensystem des Cobra RS2

steht. In der unteren Darstellung der Abbildung 2.2 ist der Nullpunkt 48 mm vom rechten und 86 mm vom oberen Rand des Gehäuses entfernt.

4.1 Rückwärtskinematik

Die Rückwärtskinematik berechnet die Winkel (α, β, γ) der einzelnen Gelenke aus den x_0, y_0 und z_0-Koordinaten des Punktes E (siehe Abbildung 4.4). Der Punkt E befindet sich im Mittelpunkt zwischen den Greifarmen am unteren Ende des Greifers. Zur Berechnung wird zusätzlich der Winkel des Greifers δ benötigt.

Dieser muss vorgegeben werden, um den Roboter eindeutig positionieren zu können. In Abbildung 4.2 wird deutlich, dass man zum Erreichen einer Position mit dem Punkt E

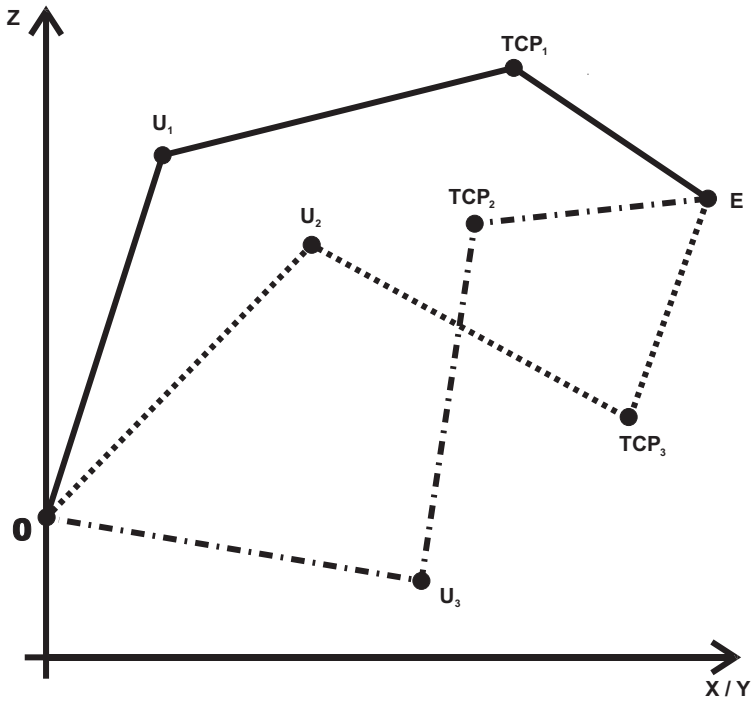


Abbildung 4.2: Mehrdeutigkeit der Anfahrposition

mindestens 3 verschiedene Möglichkeiten hat. Dieser Punkt lässt sich noch mit einer Anzahl von weiteren Armstellungen erreichen.

Durch die Angabe des Greiferwinkels δ wird das Positionierungsproblem eindeutig. Mit Hilfe des Winkels und der Länge des Greifarms kann man nun die Koordinaten der Greiferaufhängung berechnen.

Der erste Schritt zur Lösung der Kinematik ist die Berechnung der Position der Greiferaufhängung (Punkt TCP[1]), beschrieben durch die Koordinaten x_{TCP}, y_{TCP} und z_{TCP}, in Abbildung 4.3 und 4.4.

Berechnung von α

Aus Abbildung 4.3 ergibt sich:

$$\alpha = \arctan \frac{y_0}{x_0} \tag{4.1}$$

[1]Tool Center Point

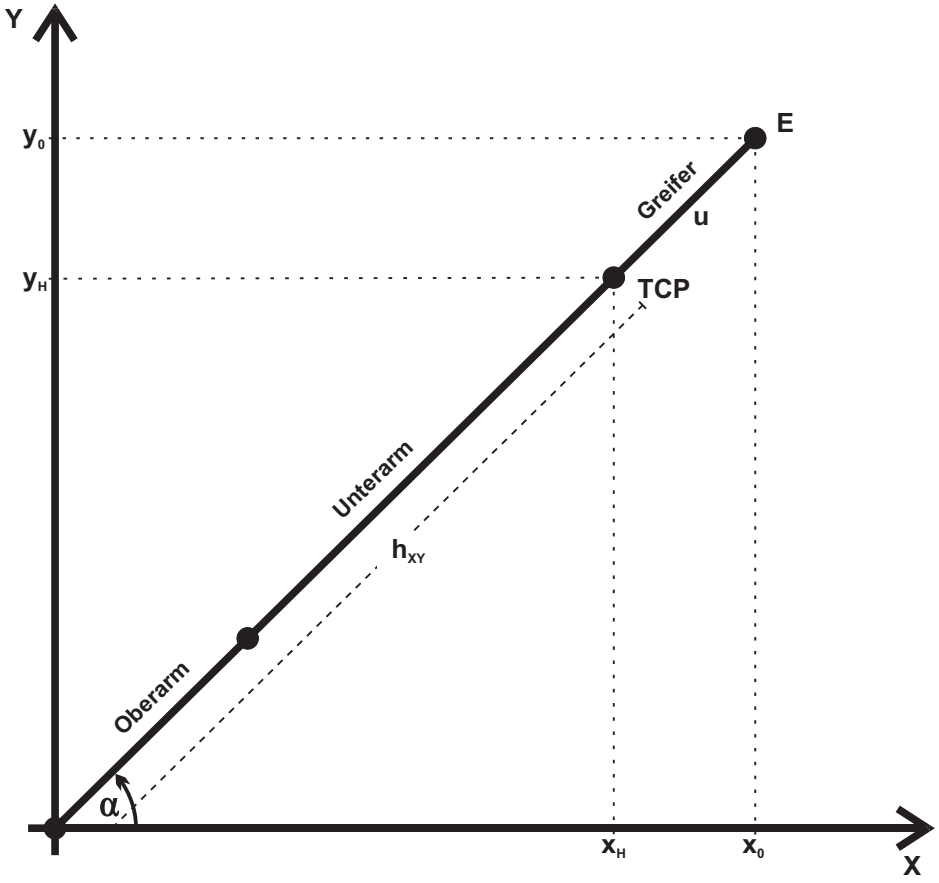


Abbildung 4.3: exemplarische Ansicht des Roboters von oben

Berechnung der Koordinaten des TCPs:

$$
\begin{aligned}
u &= d_4 \cos\delta \qquad \text{(Abb. 4.4)} \\
x_{TCP} &= x_0 - u\cos\alpha \qquad \text{(Abb. 4.3)} \qquad (4.2) \\
y_{TCP} &= y_0 - u\sin\alpha \qquad \text{(Abb. 4.3)} \qquad (4.3) \\
z_{TCP} &= z_0 - d_4 \sin\delta \qquad \text{(Abb. 4.4)} \qquad (4.4)
\end{aligned}
$$

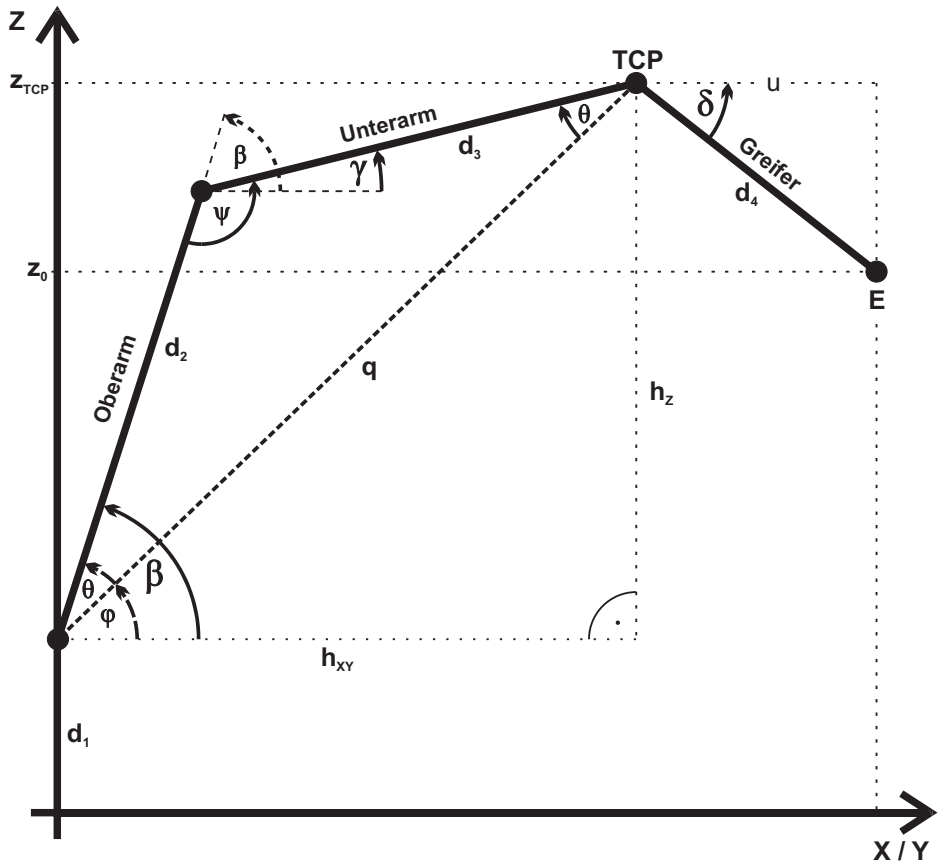


Abbildung 4.4: exemplarische Ansicht des Roboters von der Seite

Berechnung von β

Der nächste Schritt ist die Berechnung des Hilfswinkel φ in Abbildung 4.4.

$$h_{xy} = x_{TCP}\cos\alpha + y_{TCP}\sin\alpha \qquad \text{(Abb. 4.3)} \tag{4.5}$$

$$h_z = z_{TCP} - d_1 \qquad \text{(Abb. 4.4)} \tag{4.6}$$

$$q = \sqrt{h_{xy}^2 + h_z^2} \qquad \text{(Abb. 4.4)} \tag{4.7}$$

$$\varphi = \arctan\frac{h_z}{h_{xy}} \qquad \text{(Abb. 4.4)} \tag{4.8}$$

Mit Hilfe des Kosinussatzes

$$d_3^2 = d_2^2 + q^2 - 2\ d_2\ q\cos\theta$$

und dessen Umformung

$$\begin{aligned} \cos\theta &= \frac{d_2^2 + q^2 - d_3^2}{2\ d_2\ q} \\ &= \frac{q^2}{2\ d_2\ q} \\ &= \frac{q}{2\ d_2} \end{aligned} \tag{4.9}$$

ergibt sich der Hilfswinkel θ in Abbildung 4.4.

$$\theta = \arccos\frac{q}{2d_2} \tag{4.10}$$

Damit erhält man den Oberarmwinkel β.

$$\beta = \varphi + \theta \tag{4.11}$$

Berechnung von γ

Das Dreieck, aufgespannt durch d_2, d_3 und q ist wegen der Konstruktion des Roboters ein gleichschenkliges Dreieck (siehe Abbildung 4.4).

Deshalb gilt

$$\psi = 180° - 2\theta \tag{4.12}$$

In Abbildung 4.4 gilt:

$$\begin{aligned} \beta + \psi - \gamma &= 180° \\ \gamma &= \psi + \beta - 180° \end{aligned} \tag{4.13}$$

(4.14)

Durch Einsetzen der Gleichung (4.12) ergibt sich:

$$\begin{aligned} \gamma &= 180° - 2\theta + \beta - 180° \\ &= \beta - 2\theta \end{aligned} \tag{4.15}$$

Vereinfacht mit Gleichung (4.11) ergibt sich schließlich:

$$\gamma = \varphi + \theta - 2\theta$$

$$\boxed{\gamma = \varphi - \theta} \tag{4.16}$$

4.2 Vorwärtskinematik

Nach Berechnung der Rückwärtskinematik im vorherigen Abschnitt folgt nun die Herleitung der Formeln für die Vorwärtskinematik. Also die Umrechnung von α, β, γ in x_0, y_0, z_0.

Mit der Kinematik können die gegebenen Roboterwinkel in das kartesische Koordinatensystem umgerechnet werden und finden im Ansteuerungsmodul zur Berechnung der Erreichbarkeit der Position Verwendung.

Es besteht die Möglichkeit, dass man eine bestimmte Position ansteuern möchte und diese auf Grund der Mechanik und den Ausmaßen des Cobra RS2 nicht erreichbar ist.

Berechnung von φ und θ

Aus Umformungen der Gleichungen (4.11) und (4.16) ergibt sich:

$$\varphi = \beta - \theta \qquad \text{siehe Gl. (4.11)} \tag{4.17}$$

$$\theta = \varphi - \gamma \qquad \text{siehe Gl. (4.16)} \tag{4.18}$$

Einsetzen der Gleichung (4.18) in (4.17) ergibt den Winkel φ:

$$\begin{aligned} \varphi &= \beta - \varphi + \gamma \\ 2\varphi &= \beta + \gamma \end{aligned}$$

$$\boxed{\varphi = \frac{1}{2}\,(\beta + \gamma)} \tag{4.19}$$

Durch Einsetzen der Gleichung (4.19) in (4.18) ergibt sich der Hilfswinkel θ:

$$\theta = \frac{1}{2}\,(\beta + \gamma) - \gamma$$

$$\boxed{\theta = \frac{1}{2}\,(\beta - \gamma)} \tag{4.20}$$

Berechnung der TCP-Koordinaten

Mit Hilfe von Gleichung (4.9) lassen sich h_{xy} und h_z bestimmen:

$$\begin{aligned} q &= 2d_2 \cos\theta \qquad \text{siehe (4.9)} \\ h_{xy} &= q \cos\varphi \qquad \text{(Abb. 4.4)} \qquad (4.21) \\ h_z &= q \sin\varphi \qquad \text{(Abb. 4.4)} \qquad (4.22) \end{aligned}$$

Hiermit ergeben sich folgende Koordinaten des TCP

$$\begin{aligned} x_{TCP} &= h_{xy} \cos\alpha \qquad \text{(Abb. 4.3)} \qquad (4.23) \\ y_{TCP} &= h_z \sin\alpha \qquad \text{(Abb. 4.4)} \qquad (4.24) \\ z_{TCP} &= h_z + d_1 \qquad \text{(Abb. 4.4)} \qquad (4.25) \end{aligned}$$

und daraus die Koordinaten des Punktes E.

$$u = d_4 \cos\delta \qquad \text{(Abb. 4.4)}$$

$$\begin{aligned} x_0 &= x_{TCP} + u \sin\alpha \qquad \text{(Abb. 4.4)} \qquad (4.26) \\ y_0 &= y_{TCP} + u \cos\alpha \qquad \text{(Abb. 4.4)} \qquad (4.27) \\ z_0 &= z_{TCP} + d_4 \sin\delta \qquad \text{(Abb. 4.4)} \qquad (4.28) \end{aligned}$$

4.3 Dynamic Link Library

Die kinematischen Zusammenhänge aus dem vorherigen Abschnitten wurden in verschiedenen Funktionen abgebildet. Um diese leicht für andere Programme verfügbar zu machen, sind diese in einer Dynamic Link Library (DLL) zusammengefasst worden.

Die DLL wird zusammen mit der *Cobradll* in der *cobra_use* Library verwendet (siehe Kapitel 5), um den Roboter im Koordinatensystem zu bewegen.

Alle verwendeten Winkel (Parameter und Rückgabe) werden in Grad 0..360°angegeben. x, y und z stehen für die 3 Achsen des Roboterkoordinatensystem; alpha, beta, gamma und delta als Robotergelenk-Winkel für Drehturm, Oberarm, Unterarm und Greifer.

4.3.1 interne Funktionen

Dieser Abschnitt enthält eine kurze Beschreibung der sog. internen Funktionen der DLL. Diese sind von anderen Programmen nicht aufrufbar, werden aber für die Berechnungen in den externen Funktionen (siehe Abschnitt 4.3.2) verwendet.

winkel_drehturm (double x, double y, double z, double delta)

Rückgabe

Drehturmgelenk-Winkel alpha (double)

Diese Funktion berechnet aus den angegebenen Koordinaten x, y, z und dem Greifer-Winkel δ den absoluten Winkel der Drehturm-Achse α.

Hierzu werden die Formeln der Rückwärtskinematik (siehe Abschnitt 4.1) benutzt.

winkel_oberarm (double x, double y, double z, double delta)

Rückgabe

Oberarmgelenk-Winkel beta (double)

Diese Funktion berechnet aus den angegebenen Koordinaten x, y, z und dem Greifer-Winkel δ den absoluten Winkel der Oberarm-Achse β.

Hierzu werden die Formeln der Rückwärtskinematik (siehe Abschnitt 4.1) benutzt.

winkel_unterarm (double x, double y, double z, double delta)

Rückgabe

Unterarmgelenk-Winkel gamma (double)

Diese Funktion berechnet aus den angegebenen Koordinaten x, y, z und dem Greifer-Winkel δ den absoluten Winkel der Unterarm-Achse γ.

Hierzu werden die Formeln der Rückwärtskinematik (siehe Abschnitt 4.1) benutzt.

position_x (double alpha, double beta, double gamma, double delta)

Rückgabe

x-Koordinate (double)

Diese Funktion berechnet aus den angegebenen Winkeln α, β, γ und dem Greifer-Winkel δ die zugehörige x-Koordinate im Koordinatensystem.

Hierzu werden die Formeln der Vorwärtskinematik (siehe Abschnitt 4.2) benutzt.

position_y (double alpha, double beta, double gamma, double delta)

Rückgabe

y-Koordinate (double)

Diese Funktion berechnet aus den angegebenen Winkeln α, β, γ und dem Greifer-Winkel δ die zugehörige y-Koordinate im Koordinatensystem.

Hierzu werden die Formeln der Vorwärtskinematik (siehe Abschnitt 4.2) benutzt.

position_z (double alpha, double beta, double gamma, double delta)

Rückgabe

z-Koordinate (double)

Diese Funktion berechnet aus den angegebenen Winkeln α, β, γ und dem Greifer-Winkel δ die zugehörige z-Koordinate im Koordinatensystem.

Hierzu werden die Formeln der Vorwärtskinematik (siehe Abschnitt 4.2) benutzt.

4.3.2 externe Funktionen

Dieser Abschnitt beschreibt die Funktionen die durch die DLL zur Verfügung gestellt werden.

pos_x (double alpha, double beta, double gamma, double delta)

Rückgabe

x-Koordinate (double)

Aufruf der Funktion *position_x* (siehe Abschnitt 4.3.1).

pos_y (double alpha, double beta, double gamma, double delta)

Rückgabe

y-Koordinate (double)

Aufruf der Funktion *position_y* (siehe Abschnitt 4.3.1).

pos_z (double alpha, double beta, double gamma, double delta)

Rückgabe

z-Koordinate (double)

Aufruf der Funktion *position_z* (siehe Abschnitt 4.3.1).

winkel_drehturm_abs (double x,double y, double z, double delta)

Rückgabe

Drehturmgelenk-Winkel alpha (double)

Aufruf der Funktion *winkel_drehturm* (siehe Abschnitt 4.3.1).

winkel_drehturm_rel (double x,double y, double z, double delta, double alpha)

Rückgabe

Drehturmgelenk-Winkel alpha (double)

Aufruf der Funktion *winkel_drehturm* (siehe Abschnitt 4.3.1) und Differenzbildung mit Parameter *double alpha*.

Ausgabe des relativen Drehturmwinkels α.

winkel_drehturm_rel_xyz (double x1, double y1, double z1, double x2, double y2, double z2, double delta1, double delta2)

Rückgabe

Drehturmgelenk-Winkel alpha (double)

Aufruf der Funktion *winkel_drehturm* (siehe Abschnitt 4.3.1) einmal mit den Parametern *x1*, *y1*, *z1* und *delta1* und einmal mit *x2*, *y2*, *z2* und *delta2*. Differenzbildung der beiden berechneten Winkel *alpha1* und *alpha2*.

Ausgabe des relativen Drehturmwinkels α.

winkel_oberarm_abs (double x,double y, double z, double delta)

Rückgabe

Oberarmgelenk-Winkel beta (double)

Aufruf der Funktion *winkel_ oberarm* (siehe Abschnitt 4.3.1).

winkel_oberarm_rel (double x,double y, double z, double delta, double beta)

Rückgabe

Oberarmgelenk-Winkel beta (double)

Aufruf der Funktion *winkel_ oberarm* (siehe Abschnitt 4.3.1) und Differenzbildung mit Parameter *double beta*.

Ausgabe des relativen Oberarmwinkels β.

winkel_oberarm_rel_xyz (double x1, double y1, double z1, double x2, double y2, double z2, double delta1, double delta2)

Rückgabe

Oberarmgelenk-Winkel beta (double)

Aufruf der Funktion *winkel_ oberarm* (siehe Abschnitt 4.3.1) einmal mit den Parametern *x1*, *y1*, *z1* und *delta1* und einmal mit *x2*, *y2*, *z2* und *delta2*. Differenzbildung der beiden berechneten Winkel *beta1* und *beta2*.

Ausgabe des relativen Oberarmwinkels β.

winkel_unterarm_abs (double x, double y, double z, double delta)

Rückgabe

Unterarmgelenk-Winkel gamma (double)

Aufruf der Funktion *winkel_ unterarm* (siehe Abschnitt 4.3.1).

winkel_unterarm_rel (double x, double y, double z, double delta, double gamma)

Rückgabe

Unterarmgelenk-Winkel gamma (double)

Aufruf der Funktion *winkel_unterarm* (siehe Abschnitt 4.3.1) und Differenzbildung mit Parameter *double gamma*.

Ausgabe des relativen Unterarmwinkels γ.

winkel_unterarm_rel_xyz (double x1, double y1, double z1, double x2, double y2, double z2, double delta1, double delta2)

Rückgabe

Unterarmgelenk-Winkel gamma (double)

Aufruf der Funktion *winkel_unterarm* (siehe Abschnitt 4.3.1) einmal mit den Parametern *x1*, *y1*, *z1* und *delta1* und einmal mit *x2*, *y2*, *z2* und *delta2*. Differenzbildung der beiden berechneten Winkel *gamma1* und *gamma2*.

Ausgabe des relativen Unterarmwinkels γ.

pos_check_winkel (double x, double y, double z, double delta)

Rückgabe

Status (int)

Diese Funktion überprüft, ob die durch die Parameter angegebene Position mit dem Roboter erreichbar sind.

Folgende Winkel sind zulässig:

$$-170^\circ \leq \alpha \leq 170^\circ \tag{4.29}$$

$$-30^\circ \leq \beta \leq 150^\circ \tag{4.30}$$

$$\beta - 140 \leq \gamma \leq \beta + 53^\circ \tag{4.31}$$

$$90^\circ \leq \delta - \gamma \leq 136^\circ \tag{4.32}$$

Wenn die Bedingungen (4.29),(4.30) und (4.31) erfüllt sind so ist die angegebene Position grundsätzlich erreichbar. Von Bedingung (4.32) ist es abhängig, ob dies mit dem angegebenen Greiferwinkel δ möglich ist.

Zurückgegeben wird, ob die Position erreichbar (0), nicht erreichbar (1) oder mit einem anderem Greiferwinkel δ erreichbar ist (2).

pos_check_raum (double x, double y, double z, double delta)

Rückgabe

Status (int)

Diese Funktion überprüft, ob die durch die Parameter angegebene Position sich außerhalb der Abmessungen Roboters (siehe Abbildung 2.2) befindet (Rückgabe 0) oder sich darin befindet (Rückgabe 1).

5 Platinenansteuerung

Dieses Kapitel gibt einen Einblick in die Platinenansteuerung mit der *Cobradll*, eine Funktionsbeschreibung des im Rahmen der Diplomarbeit entstandenen *CobraTESTer*, einen Überblick über die *cobra_use* Library und das *Cobra-Steuerung* Modul.

5.1 Die Cobradll Dynamic Link Library

Die *Cobradll* Dynamic Link Library stellt alle zur Ansteuerung des Roboters nötigen Funktionen für andere Programme zur Verfügung.

Darin eingebunden ist ein Treiber für die parallele Schnittstelle des PCs. Für eine detaillierte Funktionsbeschreibung der DLL sei auf [8] verwiesen.

5.2 Das Testprogramm

Für die Entwicklung der *Cobradll* und der DLL für die Koordinatentransformation (siehe Abschnitt 4.3) wurde der *CobraTESTer* mit *Visual Basic 6.0* programmiert.

Dieses, anfangs nur aus einem Bitsender bestehende Programm, bietet mittlerweile alle Möglichkeiten den Roboter Cobra RS2 anzusteuern und zu testen.

Um die bereitgestellten Funktionen besser zu verstehen, nun eine Beschreibung der Oberfläche des *CobraTESTers*, die in Abbildung 5.1 zu sehen ist.

Im Menü des *CobraTESTer* kann der anzusteuernde LPT-Port ausgewählt werden und die Tastatursteuerung gestartet werden. Dazu öffnen sich zwei neue Fenster, mit denen man Untersetzung und Schrittanzahl einstellen sowie über die Tasten Q – Z und A – H den Roboter steuern kann.

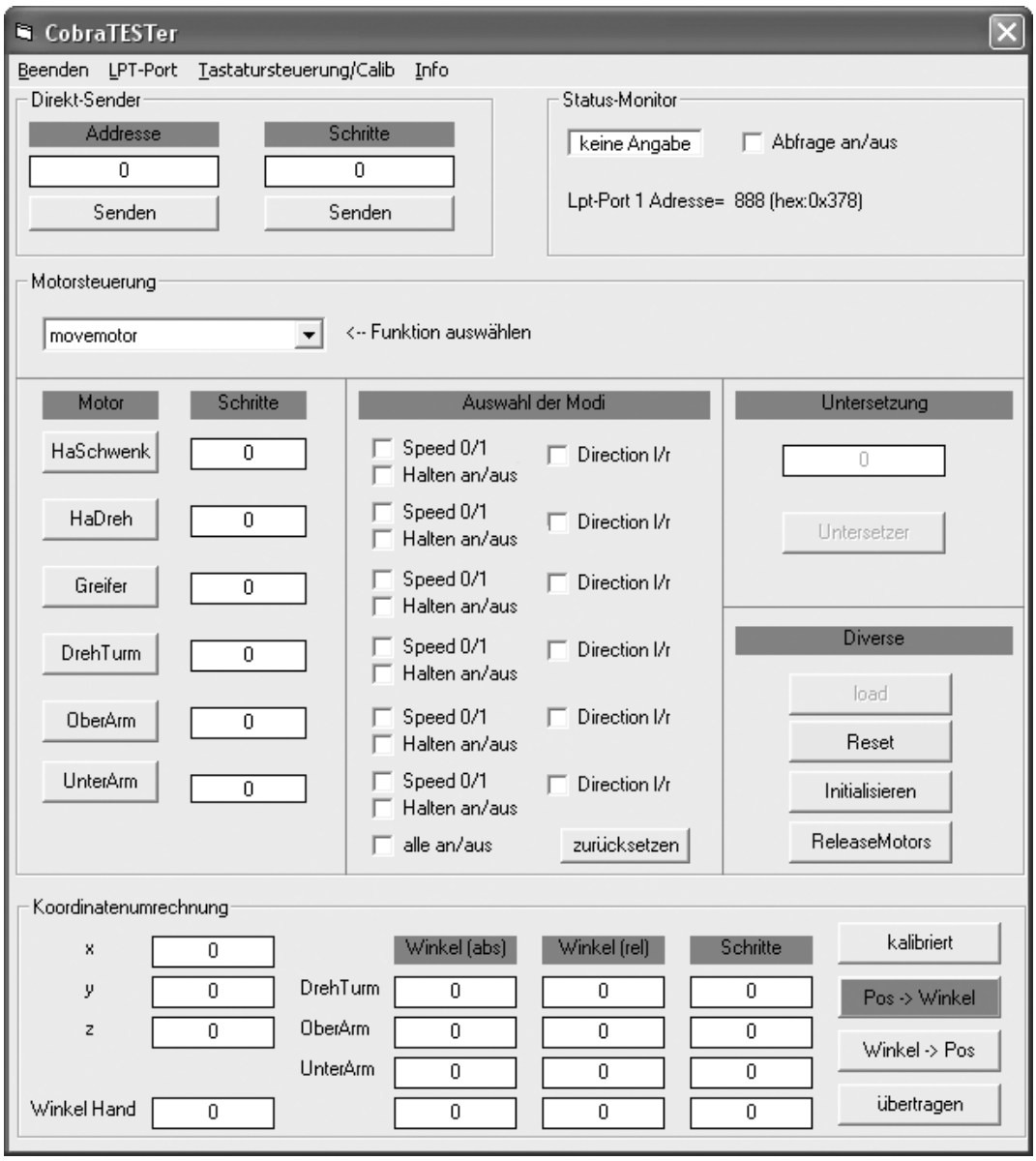


Abbildung 5.1: CobraTESTer Oberfläche

Der erste Teilbereich der Oberfläche ist der *Direkt-Sender*. Hiermit kann eine bestimmte Adresse und/oder Schrittzahl an die Ansteuerungsschaltung (siehe Kapitel 3 und Abschnitt 2.2.1) gesendet werden.

Der nächste Teilbereich ist der *Status-Monitor*. Durch das Kästchen zu aktivieren, zeigt er den jeweiligen Status des *BUSY*-Signals der Parallelport-Schnittstelle an und gibt somit Auskunft, wann die Schaltung für weitere Ansteuerungsbefehle bereit ist.

Die *Motorsteuerung* bietet den direkten Zugriff auf die Funktionen der *Cobradll*. Diese können auch über eine Auswahlliste einzeln ausgewählt werden. Dementsprechend verändert sich die Anzeige der anzusteuernden Motoren und Auswahl der Modi. In Abbildung 5.1 ist z.B. die Funktion *movemotor* ausgewählt.

Den letzten Teilbereich der Oberfläche bildet die *Koordinatenumrechnung*. Diese bietet die Möglichkeit die bereitgestellten Funktionen der Koordinatentransformation-DLL aufzurufen und die berechneten Schritte über die Schaltfläche *Übertragen* direkt in die Felder der *Motor-Steuerung* einzutragen.

5.3 cobra_use Library

Die *cobra_use* Library vereint die Funktionen der *Cobradll*-DLL und der Koordinatentransformation-DLL, um Funktionen bereitzustellen, die eine einfache Steuerung des Cobra RS2 ermöglichen.

Die Funktion *fahre_position* ist dabei zuerst zu nennen. Mit ihr können direkt einzelne Positionen im Roboterkoordinatensystem angefahren werden.

Daneben gibt es zwei weitere Funktionen (*trans_kalib* und *transformation*) mit denen Positionen aus dem Kamera-Weltkoordinatensystem in das Roboterkoordinatensystem umgerechnet werden können. Das bietet den Vorteil einer direkten Verknüpfung mit einem Bildverarbeitungssystem.

writepos (char *posname, double x, double y, double z, double delta)

Parameter

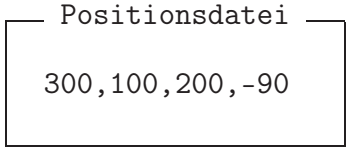
posname	Pfad der Positionsdatei
x	x-Koordinate Position
y	y-Koordinate Position

z	z-Koordinate Position
delta	Handgelenk-Winkel delta

Rückgabe

Null

Diese Funktion schreibt die als Parameter angegebenen Positionsdaten in die Positionsdatei. Damit kann das Cobra Steuerungsmodul auch nach Beendigung auf die aktuellen Positionsdaten zugreifen.



Die Positionsdaten werden durch Kommata getrennt in folgender Form in die Datei geschrieben: *x,y,z,delta*.

getpos (char *posname)

Parameter

posname	Pfad der Positionsdatei

Rückgabe

Positionsvektor (double)

Die Funktion *getpos* liest die in der Positionsdatei gespeicherten Daten aus und übergibt diese als einen Vektor (Array).

trans_kalib(char *kalibname, double *t_ka)

Parameter

kalibname	Pfad der Kalibrationsliste
t_ka	Transformationsvektor

Rückgabe

keine

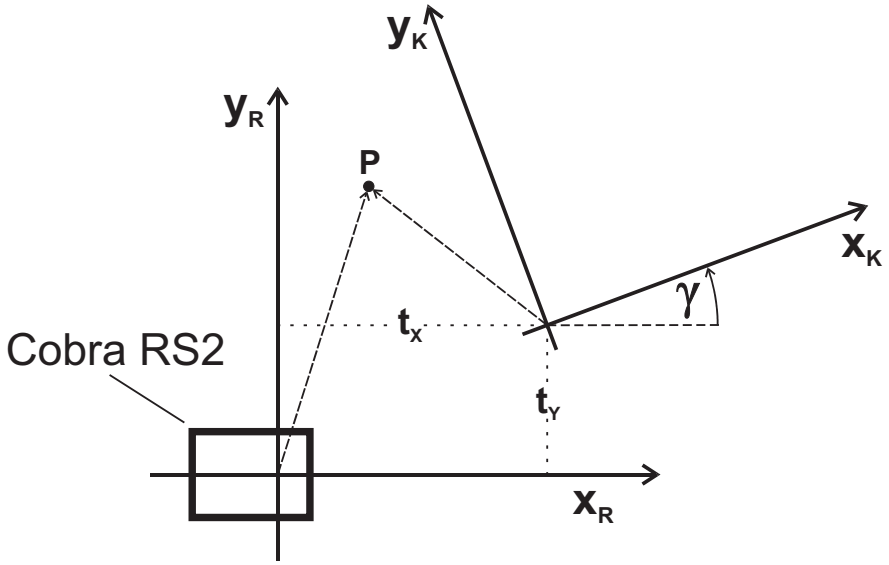


Abbildung 5.2: Translation und Rotation der Koordinatensysteme

Diese Funktion berechnet die Translation und Rotation des Kamerakoordinatensystems zum Roboterkoordinatensystem. Hierbei wird davon ausgegangen, dass die z-Achsen der beiden Koordinatensysteme parallel zueinander sind.

Berechnet werden die Werte t_x, t_y, r und γ aus den gegebenen Werten x_R, y_R, x_K und y_K. Diese werden durch eine Kalibrationsliste angegeben. x_R und y_R sind die Koordinaten eines Punktes P im Roboterkoordinatensystem. x_K und y_K sind die Koordinaten desselben Punktes P im Kamera-Weltkoordinatensystem (siehe Abbildung 5.2).

Die Liste wird dadurch angelegt, dass einzelne Punkte mit dem Roboter angefahren werden. Dort wird eine Münze abgelegt und mit dem Bildverarbeitungssystem die genaue Position im Kamera-Weltkoordinatensystem ermittelt.

Die gewonnenen Daten werden dabei in folgender Form eingetragen: x_R,y_R,x_K,y_K (siehe nachfolgende Beispielliste).

Kalibrationsliste

```
300,100,-30,83
300,0,73,81
250,50,23,133
160,20,53,210
```

Um die Translation und Rotation möglichst genau zu berechnen sollten mehrere Werte aufgenommen werden. Eine Anzahl von 5-10 hat sich dabei als ausreichend herausgestellt.

Für die Berechnung der Transformation gilt dabei folgende Beziehung:

$$\begin{pmatrix} x_R \\ y_R \end{pmatrix} = \underbrace{r}_{Skalierung} \cdot \underbrace{\begin{pmatrix} \sin\gamma & -\cos\gamma \\ \cos\gamma & \sin\gamma \end{pmatrix}}_{Rotation} \begin{pmatrix} x_K \\ y_K \end{pmatrix} + \underbrace{\begin{pmatrix} t_x \\ t_y \end{pmatrix}}_{Translation} \tag{5.1}$$

Dieses Gleichungssystem ist mit einem einzelnen Wertepaar aus der Kalibrationsliste nicht lösbar, da es vier Variablen enthält aber nur zwei Gleichungen.

Da die Kalibrationsliste mehrere Wertepaare enthält, ergibt sich ein überbestimmtes Gleichungssystem, welches mit Hilfe der Gauss'schen Ausgleichsrechnung gelöst werden kann.

Zunächst wurde das Gleichungssystem (5.1) durch folgende Substitutionen vereinfacht:

$$m = r\cos\gamma \tag{5.2}$$

$$n = r\sin\gamma \tag{5.3}$$

Daraus ergibt sich:

$$\begin{pmatrix} x_R \\ y_R \end{pmatrix} = \begin{pmatrix} n & -m \\ n & m \end{pmatrix} \begin{pmatrix} x_K \\ y_K \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{5.4}$$

Mit mehreren Wertepaaren in der Kalibrationsliste ergibt sich ein überbestimmtes Gleichungssystem, für das gilt:

$$\mathbf{A}\vec{x} = \vec{b} \tag{5.5}$$

$\mathbf{A} = (a_{ij})$ ist eine rechteckige Koeffizientenmatrix mit $i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n;$ $m \geq n$. Wenn dieses System nicht lösbar ist, so existiert kein $\vec{x} \in \mathbb{R}^n$, das diese Gleichung erfüllt.

Der Vektor $\vec{x}$ ist genau dann eine Lösung von (5.5), wenn der Restvektor $\vec{r}$ orthogonal zu allen Spalten von $\mathbf{A}$ ist. Das bedeutet:

$$\mathbf{A}^T\vec{r} = \mathbf{A}^T(\mathbf{A}\vec{x} - \vec{b}) = \vec{0} \qquad \text{oder} \qquad \mathbf{A}^T\mathbf{A}\vec{x} = \mathbf{A}^T\vec{b} \tag{5.6}$$

Die Gleichung (5.6) stellt ein lineares Gleichungssystem mit quadratischer Koeffizientenmatrix dar. Es wird als *System der Normalgleichungen* bezeichnet und der Vektor $\vec{x}$ als *Ausgleichslösung*. Seine Dimension ist n. Die Matrix $\mathbf{A}^T\mathbf{A}$ ist symmetrisch. [9]

Gleichung (5.4) muss zunächst in die Form von (5.5) gebracht werden. Damit ergibt sich:

$$\underbrace{\begin{pmatrix} -y_K & x_K & 1 & 0 \\ x_K & y_K & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} m \\ n \\ t_x \\ t_y \end{pmatrix}}_{\vec{x}} = \underbrace{\begin{pmatrix} x_R \\ y_R \\ \vdots \end{pmatrix}}_{\vec{b}} \tag{5.7}$$

Mit allen Werten aus der Kalibrationsliste wird nun die Matrix $\mathbf{A}$ und Vektor $\vec{b}$ aufgestellt, die transformierte Matrix $\mathbf{A}^T$ gebildet und jeweils mit $\mathbf{A}$ und $\vec{b}$ multipliziert (siehe Gleichung (5.7)). Diese sind Parameter der Funktion *gauss*[1].

Nach Aufruf der Funktion erhält man als Ergebnis den Vektor $\vec{x}$. Aus diesem wird der Winkel γ

$$\tan\gamma = \frac{\sin\gamma}{\cos\gamma} = \frac{n}{m}$$

$$\boxed{\gamma = \arctan\frac{n}{m}} \tag{5.8}$$

und die Skalierung r berechnet. Es gilt

$$\cos^2\gamma + \sin^2\gamma = 1 \tag{5.9}$$

[1] gauss.lib stand im Rahmen der BV-Entwicklungsumgebung des Labors zur Verfügung

Quadriert man die Gleichungen (5.2) und (5.3) und formt diese um, ergibt sich:

$$\cos^2\gamma = \frac{m^2}{r^2} \tag{5.10}$$

$$\sin^2\gamma = \frac{n^2}{r^2} \tag{5.11}$$

Setzt man diese beiden Gleichungen nun in Gleichung (5.9) ein, erhält man:

$$\begin{aligned} \frac{m^2}{r^2} + \frac{n^2}{r^2} &= 1 \\ m^2 + n^2 &= r^2 \end{aligned} \tag{5.12}$$

Auf beiden Seiten von Gleichung (5.12) wird nun die Wurzel gezogen und es ergibt sich:

$$r = \sqrt{m^2 + n^2} \tag{5.13}$$

γ, r, t_x und t_y werden zusammen in einem Array übergeben.

transformation (double x, double y, double t_x, double t_y, double gamma, double skalierung)

Parameter

x	x-Position Weltkoordinatensystem in mm
y	y-Position Weltkoordinatensystem in mm
t_x	x-Wert Translationsvektor
t_y	y-Wert Translationsvektor
gamma	Rotationswinkel Weltkoordinatensysten zu Roboterkoordinatensystem
skalierung	Korrekturfaktor r

Rückgabe

Positionsvektor der Roboterkoordinaten (double)

Diese Funktion rechnet mit Hilfe, der aus der Funktion *trans_kalib* berechneten Werte für die Translation und Rotation, Positionen aus dem Kamera-Weltkoordinatensystem in das Roboterkoordinatensystem um.

Dazu wird Gleichung (5.1) verwendet.

fahre_position (double x_neu, double y_neu, double z_neu, double delta_neu, double hand_dreh, double greifer, bool speed, char *posname)

Parameter

x_neu	x-Koordinate der anzufahrenden Position
y_neu	y-Koordinate der anzufahrenden Position
z_neu	z-Koordinate der anzufahrenden Position
delta_neu	Handgelenk-Winkel delta der anzufahrenden Position
hand_dreh	relative Drehung des Greifers in Grad
greifer	Anzahl der Greiferschritte (negativ = zu, positiv = auf)
speed	Halb- oder Vollschritte (true / false)
posname	Pfad der Positionsdatei

Rückgabe

Null

Diese Funktion vereint *Cobradll*-DLL und Koordinatentransformation-DLL und bewegt damit den Roboter im Koordinatensystem.

Dazu werden die zur neuen Position zu fahrenden Schritte berechnet. Die aktuelle Position wird mit Hilfe der Funktion *getpos* ausgelesen und nach erfolgreicher Bewegung mit der neuen aktuellen Position durch die Funktion *writepos* überschrieben.

Vor jedem Start der Bewegung wird überprüft, ob die angegebene Position erreichbar ist.

Es gibt zwei besondere Positionsangaben für die Parameter x, y und z:

0 0 0 Ist die aktuelle Position **0 0 0**, bedeutet dies, dass sich der Roboter in der Kalibrierungsposition befindet (siehe Abbildung 7.1). Dabei werden vor dem Anfahren der neuen Position zunächst einige Schritte auf den Motoren für Drehturm, Oberarm und Unterarm ausgegeben, damit die Gelenke am Anschlag sind.

1 1 1 Werden diese Werte als Parameter übergeben, so fährt der Roboter in die Kalibrierungsposition und es wird automatisch als neue aktuelle Position der Wert **0 0 0** in der Positionsdatei gespeichert.

Diese Position sollte nach jedem Bewegungsablauf zur Zwischenkalibrierung des Roboters angefahren werden, um die Anfahrgenauigkeit des Roboters stabil zu halten.

Das Öffnen, Schließen und Drehen des Greifers ist unabhängig von der Positionierung.

5.4 Cobra-Steuerung Modul

Das *Cobra-Steuerung* Modul steuert unter Zuhilfenahme der *cobra_use* Library (siehe Abschnitt 5.3) den Bewegungsablauf der Münzsortierung.

Voraussetzung für die Sortierung ist die sog. Münzliste, die das Ergebnis des Bildverarbeitungssystem (siehe Kapitel 6) ist. Aus dieser Liste werden die Daten für Position und Art der Münzen benötigt.

Die Funktion *collect_coins* ist die Hauptfunktion zur Münzsortierung. In dieser wird die Funktion *greife_muenzen* aufgerufen, die ihrerseits den Bewegungsablauf für die Sortierung einer Münzart steuert.

collect_coins (char *streamname, char *kalibname, char *posname, bool speed, int hoehe)

Parameter

streamname	Pfad der Münzliste
kalibname	Pfad der Kalibrationsliste
posname	Pfad der Positionsdatei
speed	Halb- oder Vollschritte (true / false)
hoehe	Anfahrhöhe in *mm* auf der z-Achse (siehe Abbildung 4.1)

Rückgabe

Null

Das Modul liest die Münzliste ein und speichert die einzelnen Münzen je nach Wert in acht verschiedenen Arrays ab (pro Münzwert ein Array).

Danach wird für jede Münzart die Funktion *greife_muenzen* gestartet und am Ende die Motoren freigegeben.

greife_muenzen (char *muenz, double muenze[100][2], double t_kal[3], char *posname, bool speed, int hoehe)

Parameter

muenz	Bezeichnung der Münze (z.B. 1eur,1ct...)
muenze	zugehöriges Münzposition-Array
t_kal	Transformationsvektor
posname	Pfad der Positionsdatei
speed	Halb- oder Vollschritte (*true* / *false*)
hoehe	Anfahrhöhe in *mm* auf der z-Achse (siehe Abbildung 4.1)

Rückgabe

Null

Diese Funktion steuert die Bewegungen des Roboters zur Sortierung der Münzen einer Münzart.

Jeder Münzart ist eine Ablageposition und eine entsprechende Anzahl von Greiferschritten zugeordnet.

Die Münzen werden nach dem in Abbildung 5.3 dargestellten Ablauf sortiert.

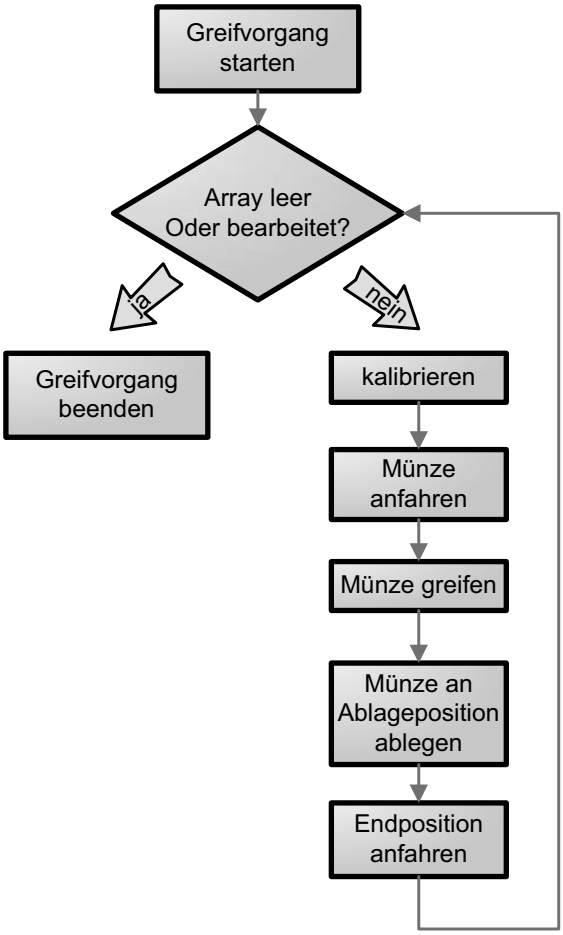


Abbildung 5.3: Cobra-Steuerung Modul

6 Bildverarbeitungsystem

Dieses Kapitel beschreibt das Bildverarbeitungssystem zur Erkennung und Klassifikation der Stanzteile (Münzen). Das System ist, ebenso wie die Ansteuerungsschaltung, modular aufgebaut.

Zunächst ein kurzer Einblick, anschließend eine Beschreibung der Module, um im Abschnitt „Münzerkennung“ das System detailliert zu beschreiben.

In Abschnitt 6.5 erfolgt eine Bewertung des System.

Abbildung 6.1 gibt einen Überblick über den Arbeitsraum des Bildverarbeitungssystem.

Im linken Teil des Bildes ist der Cobra RS2 zu sehen, im rechten die Münzen. Die Referenzmünzen können in einem Bereich von 100 Pixeln Höhe abgelegt werden. Diese werden für die Farberkennung benutzt und müssen in folgender Reihenfolge platziert werden: 1,2€, 10,20,50 ct und 1,2,5 ct (von links).

Darunter können beliebig die zu erkennenden Münzen abgelegt werden.

6.1 Übersicht

Zum besseren Verständnis der einzelnen Module nun ein kurzer Überblick über das Bildverarbeitungssystem. Eine detaillierte Beschreibung basierend auf der Kenntnis der einzelnen Module findet sich in Abschnitt 6.4.

In Abbildung 6.2 befindet sich die Darstellung des zugehörigen Signalflußplans, wie sie auch unter *Ad Oculos* gebräuchlich ist.

Mit dem Modul *vfwcap* wird das aktuelle Kamerabild in die *Ad Oculos*-Umgebung geladen (Bild 1).

Um die Münzen im Bild bestimmen zu können, bedarf es einer weiteren Verarbeitung durch verschiedenene Funktionen. Durch das Modul *rgb_med_hsi* entsteht Bild 8, welches

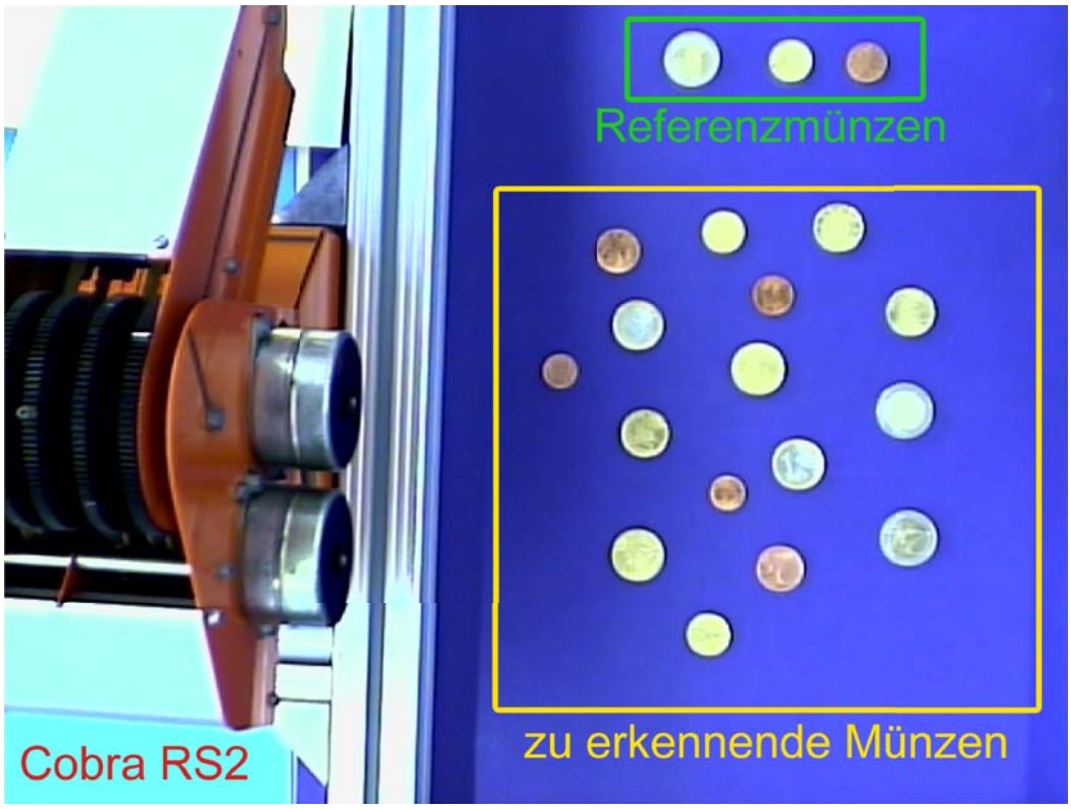


Abbildung 6.1: Arbeitsraum des Bildverarbeitungssystem

Farbinformationen der Münzen enthält. Dieses wird durch das Modul *schwaerzen* beschnitten um Irritationen durch den Roboter im Bild zu vermeiden. Daraus entsteht nun das Bild 9.

convert_to_gray wandelt Bild 1 in Grauwerte um und gibt dieses als Bild 2 aus. Das Modul *segmentierung* erstellt mit Hilfe von Bild 2 und Bild 9 ein binarisiertes Bild 3. In diesem sind die Münzen in Form von weißen Flecken dargestellt.

Die Funktion *vermessung* bestimmt die Fläche der einzelnen Münzen aus Bild 3. Mit Hilfe der Farbinformationen aus Bild 8 und der aus der *vermessung* gewonnenen Daten bestimmt das Modul *muenz_klassifikation* die Münzwerte (1 ct, 2 ct ... 2€).

putlabel beschriftet Bild 1 mit den Ergebnissen aus der *muenz_klassifikation*. Danach wird durch das Modul *cobra_steuerung* die Münzsortierung durch den Roboter gestartet.

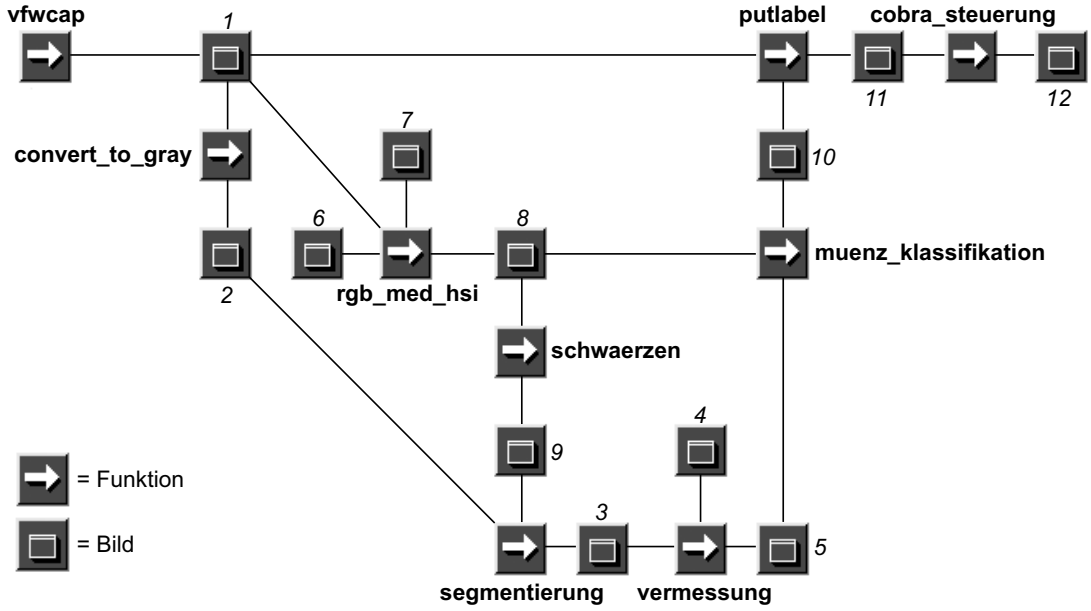


Abbildung 6.2: Signalflußplan

6.2 eigene Module

Auf den folgenden Seiten findet sich eine Aufstellung der verwendeten Bildverarbeitungsmodule, die durch meine Arbeit entstanden sind.

In den einzelnen Abschnitten befinden sich jeweils Übersichten der verwendeten Funktionen. Für eine detaillierte Beschreibung sei auf den Sourcecode (siehe CD-ROM) verwiesen.

Eine Beschreibung der Parameterdateien findet sich in Anhang A.

6.2.1 RGB to Byte

Die Funktion *RGB to Byte* ist Bestandteil des Moduls *RGB med HSI* (siehe Abschnitt 6.2.5). Hiermit wird das 8 bzw. 24 bit RGB-Farbbild in die einzelnen Farbkanäle **R**ot, **G**rün und **B**lau aufgeteilt.

In Abbildung 6.3 ist der RGB-Farbraum in der Form eines Würfels dargestellt. Die jeweilige Pixelfarbe ist dabei ein Punkt in diesem Raum, der aus den unterschiedlichen Längen der Zeiger für Rot, Grün und Blau gebildet wird.

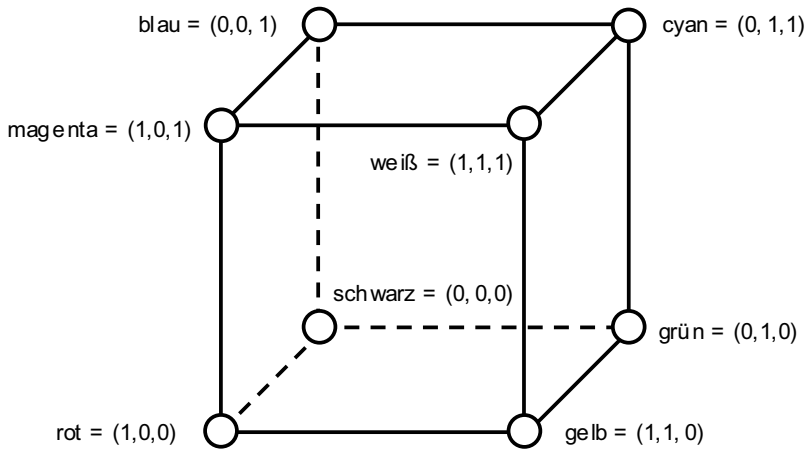


Abbildung 6.3: RGB Farbraum-Würfel

Die Farbkanäle sind in der Adoculos Speicherstruktur schon einzeln vorhanden und werden durch die Funktion in einzelnen Bildern ausgegeben.

func_rgb2byte (PIC *Pic_IN, PIC *Pic_R, PIC *Pic_G, PIC *Pic_B)

Dies ist die Hauptfunktion zur Aufsplittung eines RGB-Bildes (8 bzw. 24 bit) in die drei Farbkanäle.

Es wurde eine Portierung der *Ad Oculos*-Source in das im Labor übliche Format vorgenommen.

Color8ToByte (HPBYTE hpColor, HPBYTE hpRed, HPBYTE hpGreen, HPBYTE hpBlue, PIC *Pic_In)

Diese Funktion splittet ein 8 bit RGB-Bild in drei Byte-Bilder (hpRed, hpGreen und hpBlue) und ein Helligkeitsbild (hpColor) auf.

Es wurde eine Portierung der *Ad Oculos*-Source in das im Labor übliche Format vorgenommen.

Color24ToByte (HPBYTE hpColor, HPBYTE hpRed, HPBYTE hpGreen, HPBYTE hpBlue, PIC *Pic_In)

Diese Funktion splittet ein 24 bit RGB-Bild in drei Byte-Bilder (hpRed, hpGreen und hpBlue) und ein Helligkeitsbild (hpColor) auf.

Es wurde eine Portierung der *Ad Oculos*-Source in das im Labor übliche Format vorgenommen.

6.2.2 Median-Operator

Der Median-Operator hat innerhalb des Bildverarbeitungsystems die Aufgabe, dass Bild zu glätten, d.h. starke Grauwertunterschiede in den einzelnen Farbkanälen werden herausgefiltert. Dieses Modul findet außerdem in *RGB med HSI* Verwendung.

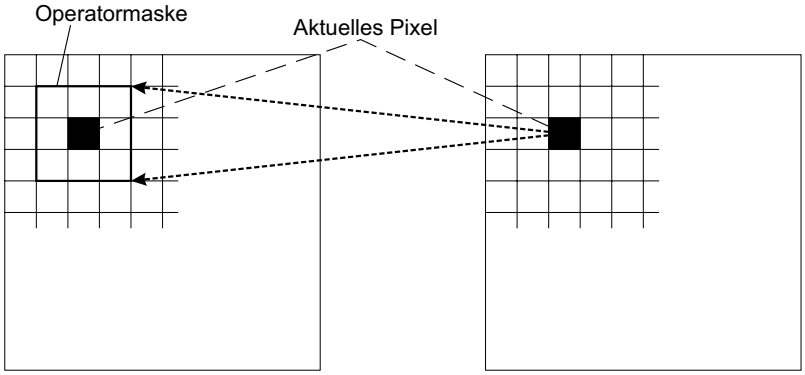


Abbildung 6.4: Operatormaske und aktuelles Pixel

Diese Funktion zählt in der Bildverarbeitung zu den sog. *lokalen Operationen*. Das zu bearbeitende Bild wird pixelweise durchlaufen. Dabei wird um den aktuellen Pixel jeweils eine Operatormaske geöffnet (siehe Abbildung 6.4). Die Größe kann frei gewählt werden.

Das Bild wird von links oben nach rechts unten durchlaufen. Da am Rand die Maske oft nicht vollständig mit Pixeln belegt werden kann, werden die Fehlenden zur Vereinfachung auf den Wert 0 gesetzt.

Für den Median-Operator werden alle Grauwerte in der Maske nach den Beträgen ihrer Grauwerte sortiert. Der mittlere Wert dieser Liste wird an die Position des aktuellen Pixels in das Ergebnisbild geschrieben. In Abbildung 6.5 ist eine beispielhafte Sortierung für eine 3 x 3 Maske dargestellt.

Dadurch werden alle hohen und niedrigen Grauwerte aus dem Bild entfernt, ohne jedoch die Grauwertsprünge abzuflachen.

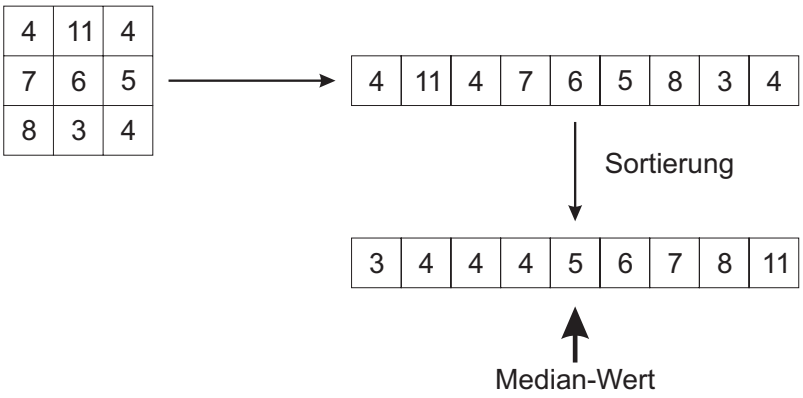


Abbildung 6.5: Ermittlung des Median-Wertes

Der Nachteil dieses Operators ist die relativ hohe Rechenzeit, die durch die Sortierung der Liste entsteht. Diese steigt mit zunehmender Größe des Operatorfensters an. [7]

Median (PIC *Pic_In, PIC *Pic_Out, int matrix)

Die Funktion *Median* realisiert den zuvor beschriebenen Median-Operator. *matrix* gibt dabei die Größe des Operatorfensters an.

Es wurde eine Portierung der *Ad Oculos*-Source in das im Labor übliche Format vorgenommen.

6.2.3 Byte to RGB

Die Funktion *Byte to RGB* ist die inverse Funktion zu *RGB to Byte* (siehe Abschnitt 6.2.1) und gleichfalls Bestandteil der Funktion *RGB med HSI*.

Benötigt werden die einzelnen Farbkanäle Rot, Grün und Blau, die zu einem RGB-Bild zusammengesetzt werden.

func_byte2rgb (PIC *Pic_R ,PIC *Pic_G, PIC *Pic_B, PIC *Pic_Out)

Die Funktion *func_byte2rgb* setzt die einzelnen Farbkanalbilder wieder zu einem 24 bit RGB-Bild zusammen.

Es wurde eine Portierung der *Ad Oculos*-Source in das im Labor übliche Format vorgenommen.

6.2.4 RGB to HSI

RGB to HSI wandelt den RGB-Farbraum in den HSI-Farbraum[1] um.

Zur Berechnung werden dabei folgende Formeln benutzt:

$$\begin{aligned} H &= \frac{1}{2\pi} \arctan\left(\frac{2R-G-B}{\sqrt{3}(G-B)}\right) \\ S &= \sqrt{R^2+G^2+B^2-RG-RB-GB} \\ I &= 0,299R+0,587G+0,114B \end{aligned}$$

H und S sind dabei auf den Bereich 0..1 normiert.

Der Vorteil der HSI-Darstellung in der Bildverarbeitung liegt darin, dass die Helligkeits- und Farbinformation getrennt verarbeitet werden können, während in der RGB-Darstellung die drei Teilbilder meist stark korreliert sind und deshalb viel gemeinsame Informationen beinhalten.

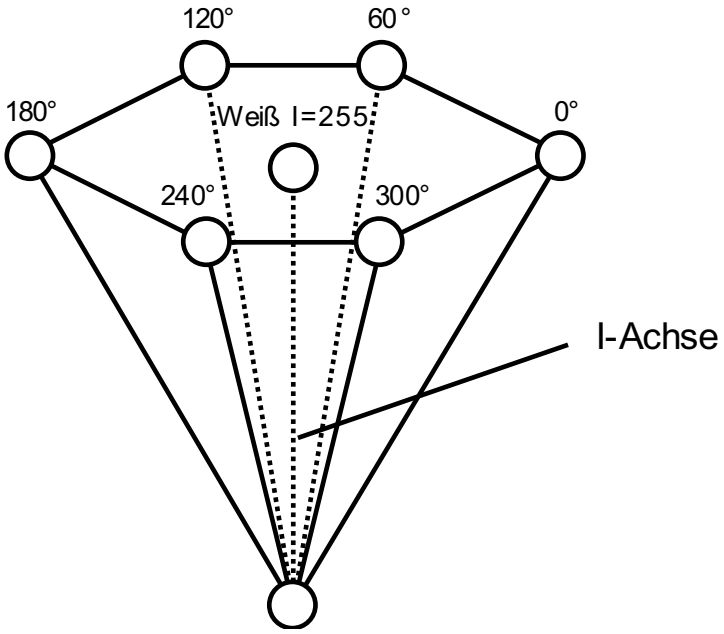


Abbildung 6.6: HSI Farbraum-Pyramide

[1] **H**ue - **S**aturation - **I**ntensity (zu deutsch: Farbe - Sättigung - Intensität)

Abbildung 6.6 zeigt eine exemplarische Ansicht des HSI-Raums. Die Farbe H ist dabei der Winkel im Sechseck. Der Abstand von der I-Achse ist die Sättigung S und die Position auf der I-Achse die Helligkeit I.

RGB_To_HSI_Float_Based (PIC *In, PIC *Out_H, PIC *Out_S, PIC *Out_I)

Diese Funktion erzeugt aus einem RGB-Bild drei Bilder für den HSI-Raum, auf der Basis einer Berechnung mit *double* Werten.

Die Funktion stand als Library in der BV-Entwicklungsumgebung des Labors zur Verfügung.

6.2.5 RGB med HSI

Das Modul *RGB med HSI* fasst, wie schon in den vorherigen Abschnitten angedeutet, die Funktionen *RGB to Byte*, *Median-Operator* und *Byte to RGB* zusammen.

Dabei wird das Eingangsbild zuerst in die einzelnen Farbkanäle aufgeteilt (*RGB to Byte*) und auf jeden Kanal der Median-Operator angewendet.
Anschließend fügt *Byte to RGB* diese wieder zu einem RGB-Bild zusammen, welches mit *RGB to HSI* in den HSI-Raum gebracht wird.

Dadurch erreicht man eine Verbesserung des H-Bildes, welches zur weiteren Verarbeitung benötigt wird.

6.2.6 Münz-Klassifikation

Die *Münz-Klassifikation* verbindet Ergebnisse aus dem Modul *Vermessung* (siehe Abschnitt 6.3.5) mit dem Farbraum der einzelnen Münzen.
Durch die *Vermessung* wird eine Liste ausgegeben. Für jedes Element aus der Liste wird ein Kreis mit dem angegeben Radius und Mittelpunkt im H-Bild des HSI-Farbraums gebildet. Aus den darin befindlichen Grauwerten wird der Mittelwert berechnet.
Für die drei Referenzmünzen, die für die Farbklassifikation nötig sind, wird ebenfalls der Graumittelwert bestimmt.

Es entstehen drei verschiedene Farbbereiche: 1-2€, 10-50 ct und 1-5 ct. Aus den berechneten Werten für die Referenzmünzen werden zwei Grenzen jeweils zwischen den drei Bereichen festgelegt.

Die Münzen werden nun entsprechend ihrer Farbe und anschließend nach gemessener Größe klassifiziert.

Die Vermessungsliste wird um die Information des jeweiligen Münzwerts ergänzt.

farbsegmentierung (PIC *Pic_In, PIC *Pic_Out, char *muenzliste, char *refliste, char *klassliste)

Die Funktion *farbsegmentierung* ist die Hauptfunktion dieses Moduls. In ihr werden die einzelnen Unterfunktionen aufgerufen, die Farbbereichsgrenzen festgelegt und die Münzliste um den mittleren Grauwert der Münze, den mittleren Grauwert der Referenzmünze und den Münzwert (1 ct,2 ct ... 2€) vervollständigt.

Münzliste

```
603,160,-25,108,17,408,125,125,50ct
520,163,-24,161,15,302,132,125,10ct
445,176,-15,209,15,354,170,174,5ct
556,208,4,137,14,252,171,174,2ct
655,221,13,75,17,385,121,125,20ct
459,230,19,199,17,446,67,90,1eur
545,261,39,144,18,466,131,125,50ct
```

Jede Zeile der Münzliste gibt Auskunft über die Daten der jeweiligen Münze. Die Daten sind dabei in folgender Reihenfolge gespeichert:

x-Koordinate (*Pixel*), y-Koordinate (*Pixel*), x-Koordinate (mm), y-Koordinate (mm), Radius (*Pixel*), Fläche (mm^2), mittl. Grauwert, mittl. Grauwert Referenzmünze, Münzwert.

mittelwert (PIC *Pic_In, PIC *Pic_Out, int pos_x, int pos_y, int radius, int min_x, int max_x, int min_y, int max_y, int schwelle)

Die Funktion *mittelwert* berechnet den mittleren Grauwert für einen kreisförmigen Bildausschnitt, der durch die Parameter (*pos_x*, *pos_y*, *radius*, *min_x*, *max_x*, *min_y*, *max_y*) festgelegt wird.

Die einzelnen Beträge der Grauwerte werden in diesem Ausschnitt aufaddiert und durch die Anzahl der Pixel geteilt. Mit *schwelle* kann ein minimaler Grauwert der Pixel für die Berechnung festgelegt werden, um so den Hintergrund herauszufiltern.

klassifikation (int flaeche, char *muenzart, char *klassliste)

Die Funktion *klassifikation* bestimmt mit Angabe von Fläche und Münzart den Münzwert. Der Parameter *muenzart* (`125ct`, `102050ct`, `12eur`) wird über die Farbinformation in der Funktion *farbsegmentierung* bestimmt und übergeben.

Klassifikationsliste

```
1ct=180
2ct=230
5ct=300
10ct=306
20ct=370
50ct=440
1eur=430
2eur=510
```

In der Klassifikationsliste werden die Vergleichsflächen für die einzelnen Münzen festgelegt. Diese können je nach Kamerakalibrierung etwas schwanken und müssen evtl. angepasst werden.

6.2.7 Cobra-Steuerung

Das *Cobra-Steuerung* Modul ist hier zur Vervollständigung des Signalflussplanes aufgeführt. Eine genaue Beschreibung der Funktion findet sich in Abschnitt 5.4.

6.3 korrespondierende Module

In diesem Abschnitt sind die Module beschrieben, die durch die Arbeit von Andreas Koch entstanden sind. Für eine genauere Beschreibung sei auf [8] verwiesen.

6.3.1 vfwcap

Dieses Modul nutzt den *Video for Windows*-Treiber, um ein Kamerabild aus dem Framegrabber zu lesen. Dieses wird damit in die *Ad Oculos* Umgebung übernommen.

6.3.2 Convert to Gray

Convert to Gray wandelt das eingegebene RGB-Bild in ein BYTE-Bild (Grauwertbild) um.

6.3.3 Schwärzen

Das Modul *Schwärzen* setzt den linken Teil des Bildes, in dem normalerweise der Roboter und ein Teil vom Gestell zu sehen wären, auf den Grauwert des Pixel am rechten oberen Bildrand, um Irritation und damit Fehler in der Funktion *Segmentierung* zu vermeiden.

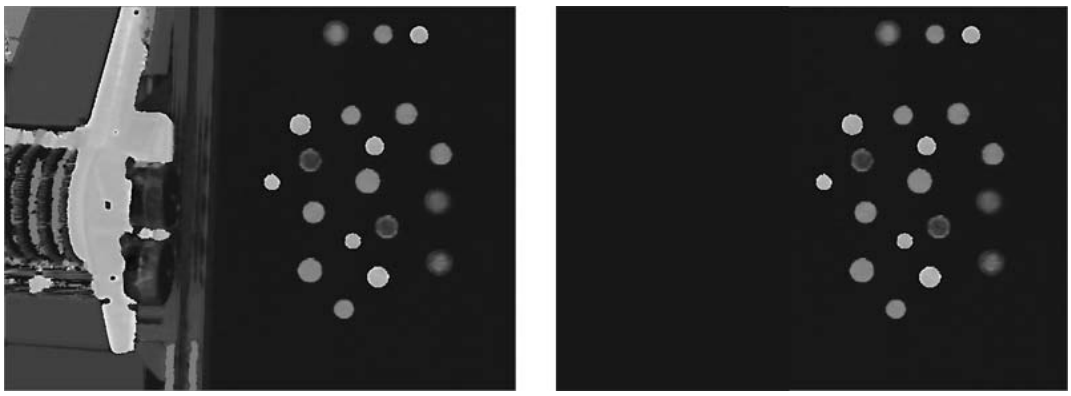


Abbildung 6.7: Schwärzen Modul

In Abbildung 6.7 ist dies dargestellt.

6.3.4 Segmentierung

Die *Segmentierung* ist eine Vorverarbeitung für die *Vermessung* und erstellt ein binarisiertes Bild.

Dazu werden im H-Bild (Bild 8) Konturen gesucht. In Ausschnitten um die Konturen wird im Grauwertbild 2 eine Schwellwertoperation durchgeführt, die eine optimale Binarisierungsschwelle bestimmt.

6.3.5 Vermessung

Dieses Modul bestimmt die Position und die Ellipsenparameter der Münzen.

Dies geschieht mittels einer Ellipsenfindung. Aus den Ellipsenparametern werden Punkte der Ellipse bestimmt, diese mit Hilfe der Kamerakalibrierung in Weltkoordinaten umgerechnet und dann mit der Gauss'chen Ausgleichsrechnung geeignete Kreisparameter berechnet.

Vermessung gibt getrennt für Münzen und Referenzmünzen jeweils eine Liste mit Positionen und Größen der Münzen aus. Diese werden für die *Münz-Klassifikation* benötigt.

6.3.6 Putlabel

Die Funktion *Putlabel* beschriftet das Eingabebild 1 mit den aus der *Münz-Klassifikation* gewonnenen Daten.

Damit ist eine einfache Kontrolle der Ergebnisse der Münzerkennung möglich.

6.4 Münzerkennung

In diesem Abschnitt soll die Münzerkennung, die in Abschnitt 6.1 überblickhaft dargestellt wurde, genauer erläutert werden.

Die Münzerkennung lässt sich dabei in zwei Bereiche unterteilen:

- **Segmentierung und Vermessung**
- **Farberkennung und Klassifikation**

Anhand verschiedener Abbildungen sollen die einzelnen Bildverarbeitungsschritte veranschaulicht werden. Im linken Teil der Abbildung ist das komplette Kamerabild dargestellt, im rechten Teil die Vergrößerung eines Bildbereichs.

6.4.1 Segmentierung und Vermessung

Um die Segmentierung und Vermessung durchzuführen bedarf es zunächst ein wenig an Vorverarbeitung.

Das mittels *vfwcap* zum Eingabebild gewordene Bild (siehe Abbildung 6.8) wird zunächst durch die Funktion *Convert to Gray* in ein Grauwertbild umgerechnet (siehe Abbildung 6.9).

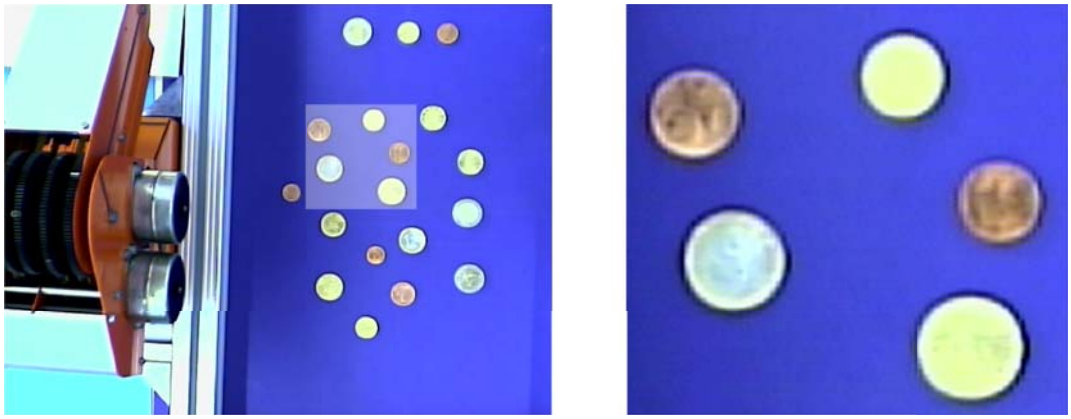


Abbildung 6.8: Eingangsbild

Dort erkennt man, dass sich die Grauwerte von 1 und 2 ct Münze nicht sehr vom Hintergrund abheben, sondern in einem ähnlichen Bereich sind. Die Unterschiede sind in den einzelnen Farbkanälen des RGB-Farbraums auch nicht größer.

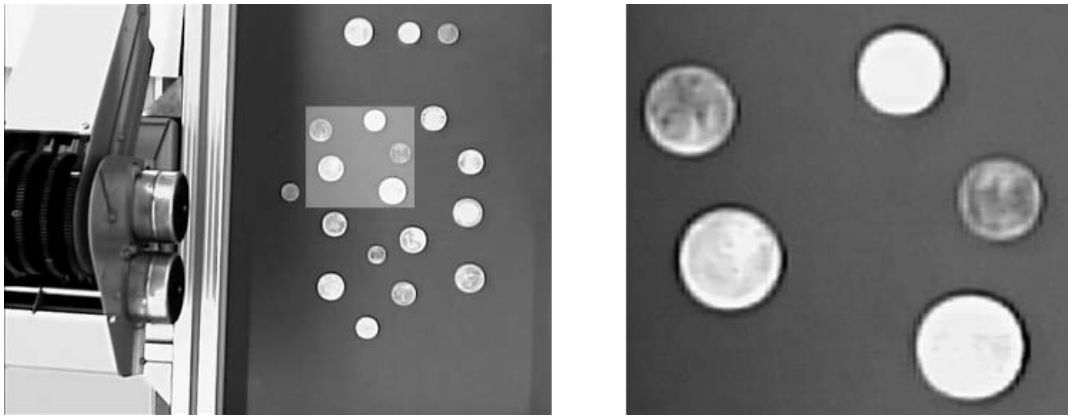


Abbildung 6.9: Convert_to_Gray

Ein weiterer Schritt der Vorverarbeitung ist die Berechnung des H-Bildes des HSI-Farbraums, zu sehen in Abbildung 6.10. Hier sind deutlich die Farbbereiche der einzelnen Münzen zu erkennen.

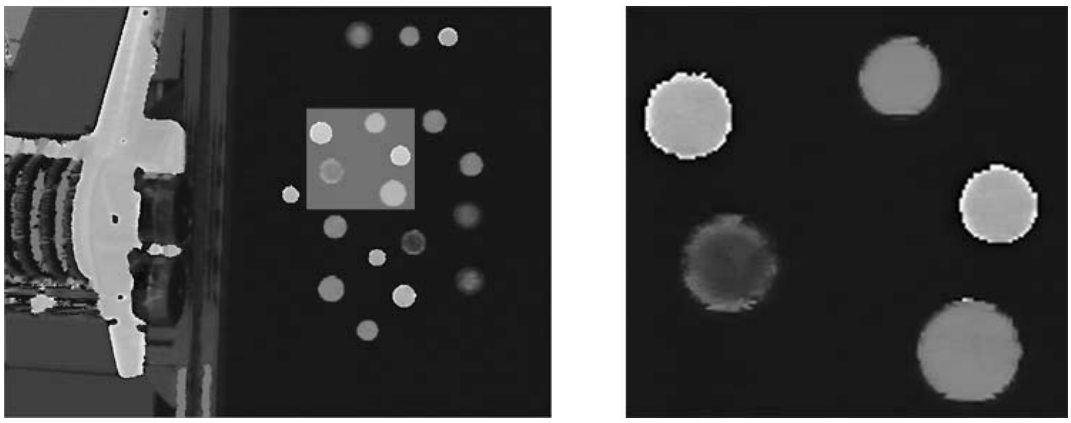


Abbildung 6.10: H-Bild

Der im linken Teil des Bildes sichtbare Roboter wird im nächsten Schritt durch die Funktion *schwaerzen* entfernt, um Irritationen in der *Segmentierung* zu vermeiden (siehe Abbildung 6.11).

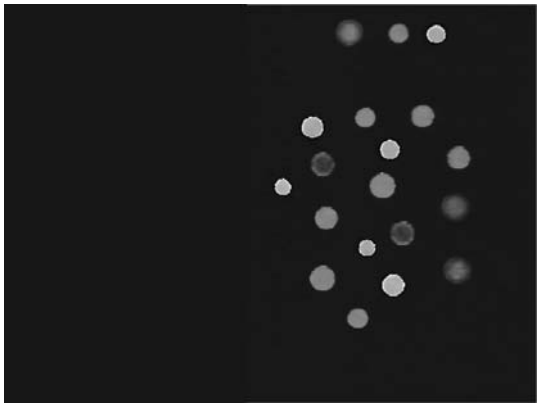


Abbildung 6.11: Schwärzen

Die nun folgende *Segmentierung* sucht im, durch die Funktion *schwaerzen* beschnittenen, H-Bild (siehe Abbildung 6.11) nach Konturen oberhalb einer bestimmten Grauwertschwelle.

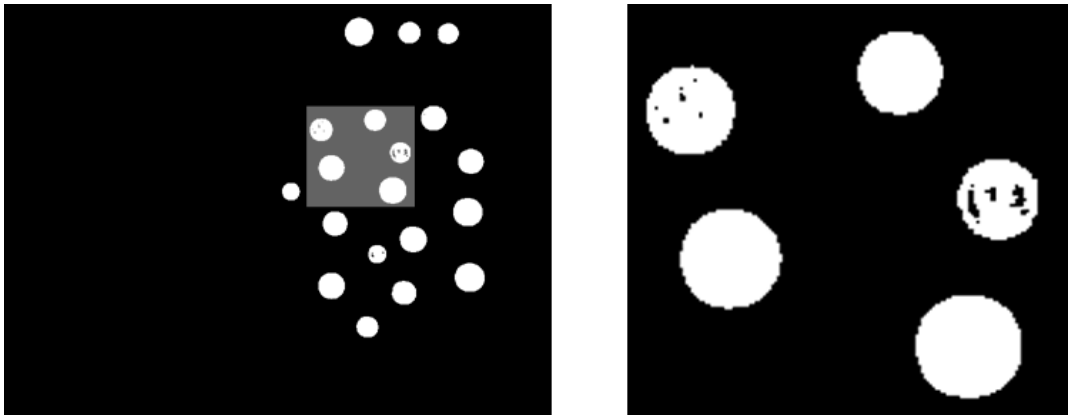


Abbildung 6.12: Segmentierung

Für jede gefundene Kontur bilden die minimalen bzw. maximalen x- und y-Koordinaten einen Ausschnitt. Im Grauwertbild wird nun genau in diesem Ausschnitt eine Schwellwert-Operation durchgeführt, d.h. ein Histogramm wird erstellt, geglättet und die Minima bestimmt. Das unterste Minimum bildet die Binarisierungsschwelle für diesen Ausschnitt.

So erhält man für jeden Ausschnitt eine angepasste Schwelle und optimal im Bild segmentierte Münzen (siehe Abbildung 6.12).

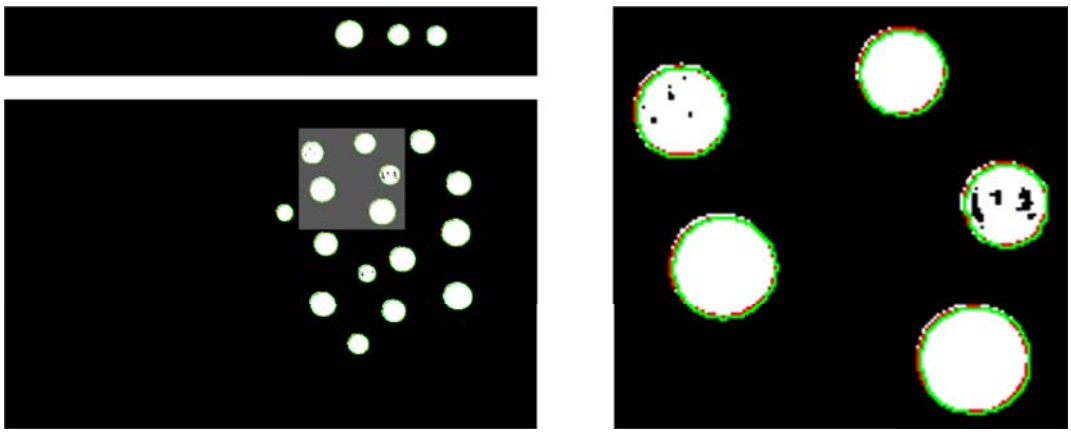


Abbildung 6.13: Vermessung

Das Ausgabebild der *Segmentierung* ist das Eingabebild für das Modul *Vermessung*. In dem binarisierten Bild werden Konturen gesucht und dazu die Ellipsen bestimmt und eingezeichnet (siehe Abbildung 6.13). Mit den Kamerakalibrierungsdaten können die Münzen genau vermessen werden. Es wird sowohl eine Liste für Referenzmünzen als auch für die zu sortierenden Münzen ausgegeben.

6.4.2 Farberkennung und Klassifikation

Die Farberkennung und Klassifikation baut ihrerseits wieder auf dem H-Bild des HSI-Farbraumes (siehe Abbildung 6.10) und den, durch die *Vermessung* erstellten Münzlisten auf.

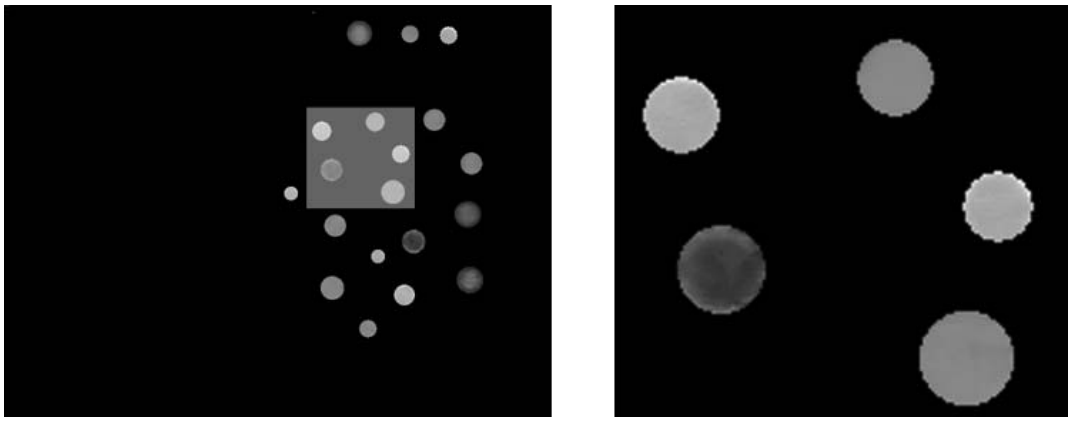


Abbildung 6.14: Münz-Klassifikation

Wie man sieht, lassen sich die einzelnen Münzarten sich klar unterscheiden. Die Kupfermünzen liegen dabei von den Grauwerten am höchsten, im mittleren Bereich finden sich die Messingmünzen, während 1 und 2€ Münzen den Messingmünzen sehr ähnlich sehen können. Bei richtiger Beleuchtung läßt sich bei der 1€ ein dunklerer Innenkreis feststellen, die 2€ Münze erhält hingegen einen dunkleren Rand.

Anhand der in den Listendaten beschriebenen Position und Radius der Münzen wird im H-Bild innerhalb eines Kreises um die Münze der mittlere Grauwert für jede Münze bestimmt.

Das daraus entstandene Bild ist das Ausgabebild des Moduls (siehe Abbildung 6.14).

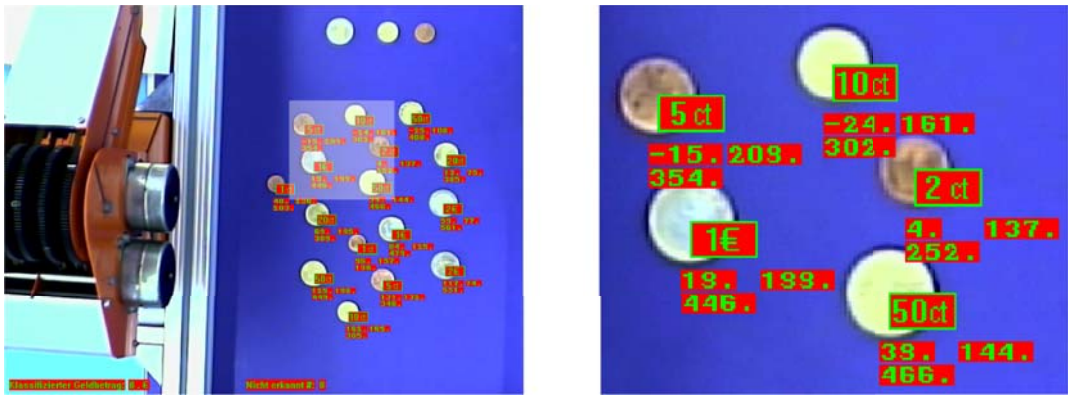


Abbildung 6.15: Ausgabe Putlabel

Der mittlere Grauwert jeder Münze wird mit denen der Referenzmünzen verglichen. Aus dem Vergleich und über die ermittelte Größe (aus dem Modul *Vermessung*) ist eine Bestimmung des Münzwertes möglich. Genauere Informationen hierzu finden sich in Abschnitt 6.2.6.

Mit den Daten aus der *Münz-Klassifikation* beschriftet das Modul *putlabel* nun das Eingabebild. Das Ausgabebild dieses Moduls ist in Abbildung 6.15 zu sehen. Jede Münze wird mit dem erkannten Münzwert beschriftet. Darunter finden sich die Werte für x- und y-Position in mm sowie die Fläche in mm^2.

6.5 Bewertung der Bildverarbeitung

Während zahlreicher Durchgänge der Münzklassifikation sind einige Unregelmäßigkeiten aufgetaucht, die noch genauerer Beschreibung bedürfen.

Zum einen gab es Probleme bei der Farberkennung und Segmentierung der Münzen, zum anderen starke Schwankungen bei der Größenbestimmung der Münzen.

Zur weiteren Untersuchung der Meßfehler wurden jeweils mehrere Aufnahmen von Münzposition auf der Arbeitsfläche gemacht. In der Messung wurden sechs 10 ct Münzen schrittweise mit 64 Ansichten und jeweils vier Aufnahmen über den ganzen Bereich verschoben und so über 1500 Messwerte gesammelt (siehe Abbildung 6.16).

Abbildung 6.16: Münzmatrix

6.5.1 Kamera-Kalibrierung und Rauschen

In einem System zur Bildverarbeitung ist die Erzeugung digitaler Bilder der erste Schritt. Die Aufnahme eines Bildes ist nichts anderes als eine Projektion der betrachteten Szene auf den Sensor der Kamera; 3D Weltpunkte werden auf 2D Bildpunkte abgebildet.

Um nun vom 2D-Kamerakoordinatensystem auf ein 3D-Weltkoordinatensystem schließen zu können, bedarf es einer Zuordnung der entsprechenden Koordinaten miteinander. Eine gebräuchliche Methode dafür ist die Aufnahme eines Kalibrierkörpers, in diesem Fall eine sog. Passpunktplatte. Auf dieser Platte sind in genau bekannten Abständen Messmarken eingebracht worden.

Die Passpunktplatte (siehe Abbildung 6.17) wird in verschiedenen Positionen aufgenommen und schließlich in der Lage, in der sich das Bezugskoordinatensystem befinden soll. Zur Auswertung der Aufnahmen und der Erstellung einer Kameradaten-Datei, die alle inneren und äußeren Kameradaten enthält, sei auf weiterführende Literatur [10, 11, 12] verwiesen.

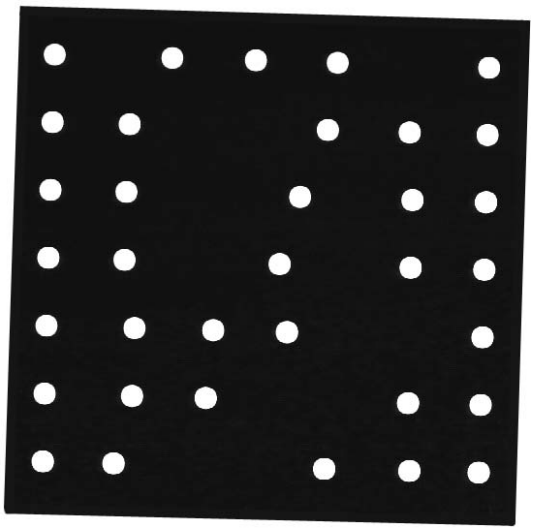


Abbildung 6.17: Passpunktplatte

Die durch die Messreihe erhaltenen Werte wurden in einer Excel-Tabelle zusammengefasst und zur besseren Analyse in *Mathsoft Mathcad 2001* grafisch weiterverarbeitet.

In Abbildung 6.18 ist die Streuung der Münzfläche für eine 10 ct Münze über die Pixel aufgetragen. Die x-Werte beginnen erst ab etwa 400 Pixel, da im linken Bildbereich auf Grund des Roboters keine Münzen platziert werden konnten, somit auch keine Ergebnisse vorliegen.

Die Farben differenzieren dabei die unterschiedlichen Flächenbereiche. Diese sind auf die, durch die Europäische Zentralbank offiziell angegebene [13] Größe einer 10 ct Münze normiert.

Der vorherrschende Wert im Bild entspricht in etwa der Originalfläche. Die dunkleren Flächen sind Bereiche, die flächenmäßig etwa 10% geringer sind, die helleren (je nach Farbe) etwa 10-30% größer.

Wie man sieht werden die Abweichungen im Bereich unten-rechts tendenziell größer.

Somit kann man auf einen grundsätzliche Ungenauigkeit in der Berechnung der Kameradaten schließen.

In Abbildung 6.19 ist deutlich zu sehen, wie stark die einzelnen Flächen an ein und derselben Position streuen.

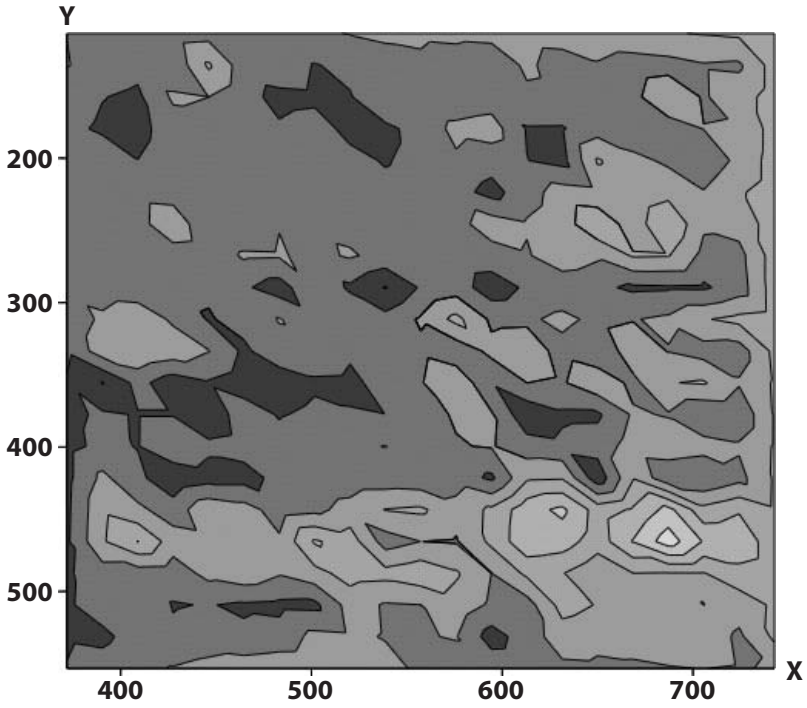


Abbildung 6.18: Streuung der Fläche über das Bild (Messung mit 10ct Münzen)

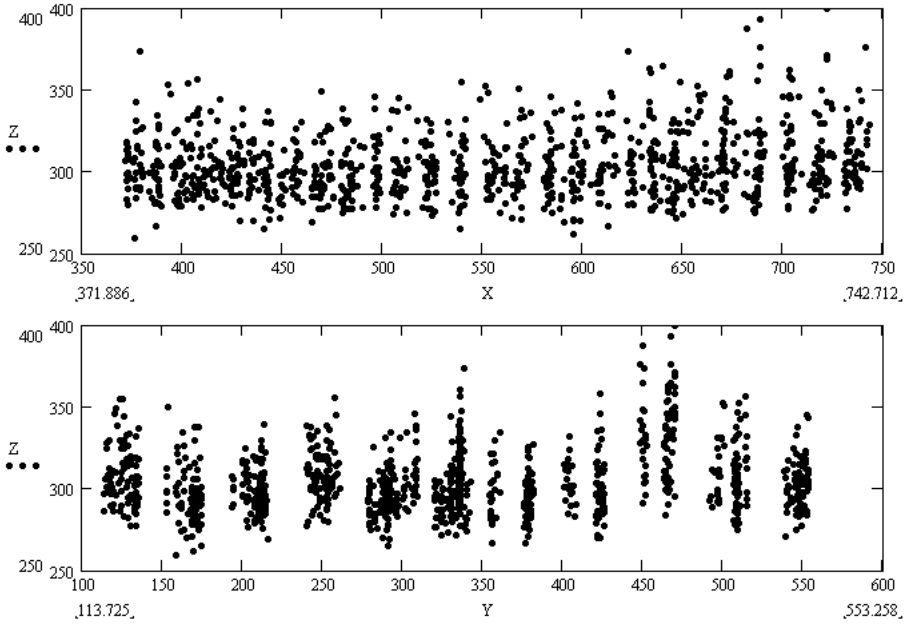


Abbildung 6.19: Streuung der Fläche über die x- bzw. y-Koordinaten

Wie schon oben erwähnt wurden pro Ansicht jeweils vier Aufnahmen gemacht. Um die Ursache dieser extremen Streuung zu finden, wurde von verschiedenen Ansichten jeweils Differenzbilder der scheinbar gleichen Aufnahmen erstellt. Die Originalbilder wurden mit derselben Funktion in Grauwertbilder umgewandelt und danach mit derselben Schwelle binarisiert. Nach Differenzbildung der beiden Bilder zeigen sich die Bilder, welche in Abbildung 6.20 dargestellt sind. Die Bilder wurden jeweils um 90° gegen den UZS[2] gedreht. Bild 1 ist dabei aus dem Bereich oben-links, während Bild 2 von unten-rechts ist.

Abbildung 6.20: Differenzbildung jeweils zweier Aufnahmen von zwei Ansichten

In dieser ist deutlich zu erkennen, dass sich aus Pixeln teilweise Kreise gebildet haben. Diese Pixel befinden sich jeweils am Rand der Münzen.

Die Abweichungen sind rein statistisch und lassen sich durch das Rauschen des Kamerabildes erklären.

Diese Pixelunterschiede erklären auch die unterschiedlichen Flächen an ein und derselben Position. Durch die Schwankungen werden andere Konturen gefunden und damit andere Ellipsenparameter bestimmt. Diese wiederum sorgen für Fehler in der Berechnung der Ausgleichskreisflächen.

Werden Münzfläche in Pixeln mit der Münzfläche in mm^2 miteinander in Beziehung gesetzt, so ergibt sich, dass 1 Pixel Unterschied ca. $2\,mm^2$ in der Fläche ausmachen.

Bei genauerer Betrachtung der Abbildung 6.20 ist zu erkennen, dass sich an einer Münze bis zu 30 Pixel zwischen zwei Aufnahmen ändern. Das entspricht einer Flächenänderung

[2]Uhrzeigersinn

von ca. 60 mm^2, in Extremfällen sogar bis zu 100 mm^2.

Aus diesem Grund ist es leider nicht möglich die *Münz-Klassifikation* allein über die Information aus der *Vermessung* durchzuführen. Die Durchmesser der Münzen unterscheiden sich oftmals nur um 1 mm.

6.5.2 Farberkennung und Segmentierung

Neben Schwierigkeiten der Münzerkennung durch Kamera-Kalibrierung und Bildrauschen, entstehen weitere Probleme durch falsche oder nicht ausreichende Beleuchtung der Arbeitsfläche.

Zuerst sollen dabei Probleme bei der Segmentierung der Münzen (siehe Abschnitt 6.3.4) aufgezeigt werden und schließlich genauer auf den Farbraum der Münzen eingegangen werden.

Segmentierung

Bei der Segmentierung der Münzen können Probleme auf Grund von fehlendem Kontrast zwischen Münzen und Hintergrund auftreten. Mit der Wahl eines anderen Hintergrundes (z.B. schwarz oder weiß) ließen sich diese Probleme vermeiden, allerdings würde sich damit das Farbraumbild sehr stark verschlechtern. Die Verschlechterung resultiert aus dem automatischen Farbabgleich der Kamera (siehe Abschnitt 2.4.1). Diese kalibriert sich auf die vorherrschende Farbe im Bild. Bei schwarzen oder weißen Hintergrund verändert sich der Farbraum so, dass die Farbbereiche der einzelnen Münzarten nicht mehr deutlich genug voneinander getrennt werden können.

Als optimaler Hintergrund hat sich dabei eine dunkelblaue Pappe herausgestellt. Bei entsprechender Beleuchtung und Kameraeinstellungen liefert das Bildverarbeitungssystem sehr gute Ergebnisse.

Anhand von zwei Beispielbildern sollen Probleme im Bereich der Segmentierung aufgezeigt werden. Das linke Bild zeigt dabei immer das Grauwertbild (Ausgabebild des Moduls *Convert_to_Gray*)und das rechte, das bereits durch das Modul *Segmentierung* binarisierte Bild. In einem kleinen Ausschnitt ist die Münze, die nicht segmentiert werden konnte, genauer zu sehen.

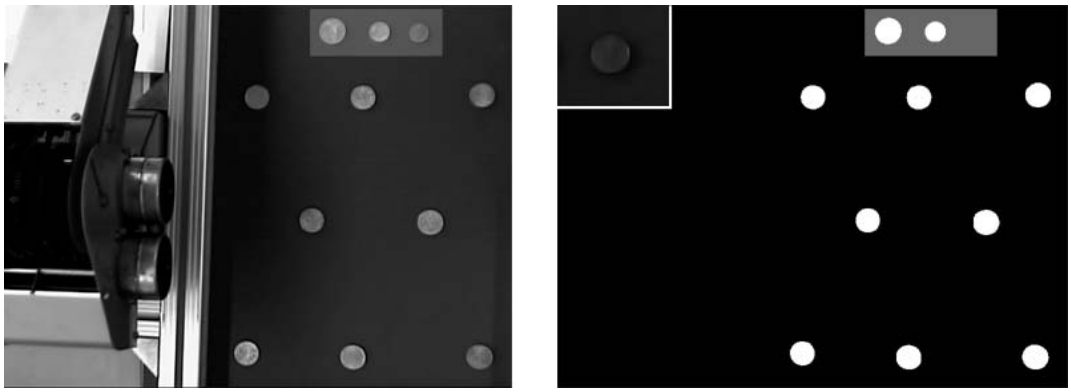


Abbildung 6.21: Segmentierungsfehler 2ct Münze (Shutter 1/250 – Licht an)

In Abbildung 6.21 ist ein Segmentierungsfehler einer 2 ct Referenzmünze dargestellt. In der vergrößerten Darstellung der 2 ct Münze aus dem Grauwertbild sieht man, dass Hintergrund und Münze ähnliche Grauwerte besitzen. Dadurch lässt sich nur schwer eine Binarisierungsschwelle im Histogramm bestimmen.

Das Bild wurde mit dem Shutter künstlich verdunkelt. Die normale Blende der Kamera sollte nach der Kamerakalibrierung nicht mehr verändert werden, da sich dadurch die Kameraparamter ändern und die Umrechnung von Kamerakoordinaten in Weltkoordinaten fehlerhaft wird.

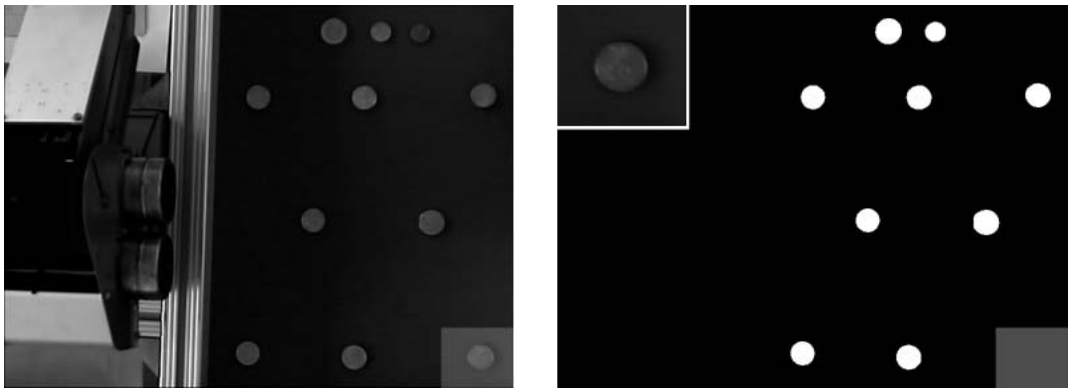


Abbildung 6.22: Segmentierungsfehler 50ct Münze (Shutter 1/125 – Licht aus)

In Abbildung 6.22 sieht man einen Segmentierungsfehler bei einer 50 ct Münze. Diese ist im Grauwertbild ohne eine zusätzliche Beleuchtung zu dunkel geworden, so dass nicht die

richtige Binarisierungsschwelle festgelegt werden konnte.

Grundsätzlich kann man sagen, dass die Segmentierungsfehler bei den 1 , 2 und 5 ct Münzen häufiger auftreten, da diese im Grauwertbild dunkler als die anderen Münzen sind. Die Abbildung 6.22 zeigt, dass auch die 2 ct Münze nicht segmentiert wurde und im Bild fehlt.

Farbraum

Um einen stabilen Farbraum zu erhalten, wird möglichst kaltes diffuses Licht benötigt. Die Schwierigkeit ist jedoch eine stabile gleichmäßige Beleuchtung einzurichten. Die besten Ergebnisse konnten bei Tageslicht mit eingeschalteter Neonröhren-Deckenbeleuchtung erzielt werden.

Die optimalen Shutter-Einstellungen unterscheiden sich je nach Lichteinfall und sollten entsprechend angepasst werden.

In Abbildung 6.23 ist ein Vergleich von Helligkeits- und Farbraumbildern (H-Bild aus HSI-Raum) bei unterschiedlichen Shuttereinstellungen und Beleuchtungen dargestellt. Für den Vergleich wurde immer derselbe Ausschnitt um die Referenzmünzen gewählt.

Nachfolgend eine Beschreibung der einzelnen Bilder.

Shutter 1/125 – Licht an

Mit dieser Einstellung hat sich zum Zeitpunkt der Aufnahme das optimale Bild ergeben.

Im Helligkeitsbild ist ein deutlicher Kontrast zwischen Münzen und Hintergrund festzustellen. Im Farbraumbild sind alle Münzarten deutlich voneinander zu unterscheiden, sogar der silberne Rand der 2€ Münze ist zu erkennen. Bei einer 1€ Münze wäre ein dunklerer Kreis im Münzinneren sichtbar.

Shutter 1/250 – Licht an

Die Beleuchtung ist gleich geblieben, der Shutter wurde um eine Stufe erhöht, die elektronische Verschlusszeit somit halbiert.

Man sieht im Helligkeitsbild eine deutliche Verdunkelung und damit weniger Kontrast zwischen Münzen und Hintergrund. Bei der *Segmentierung* könnten mit diesem Bild schon Probleme auftreten. Das Farbraumbild unterscheidet sich kaum vom vorherigen und ist immer noch optimal.

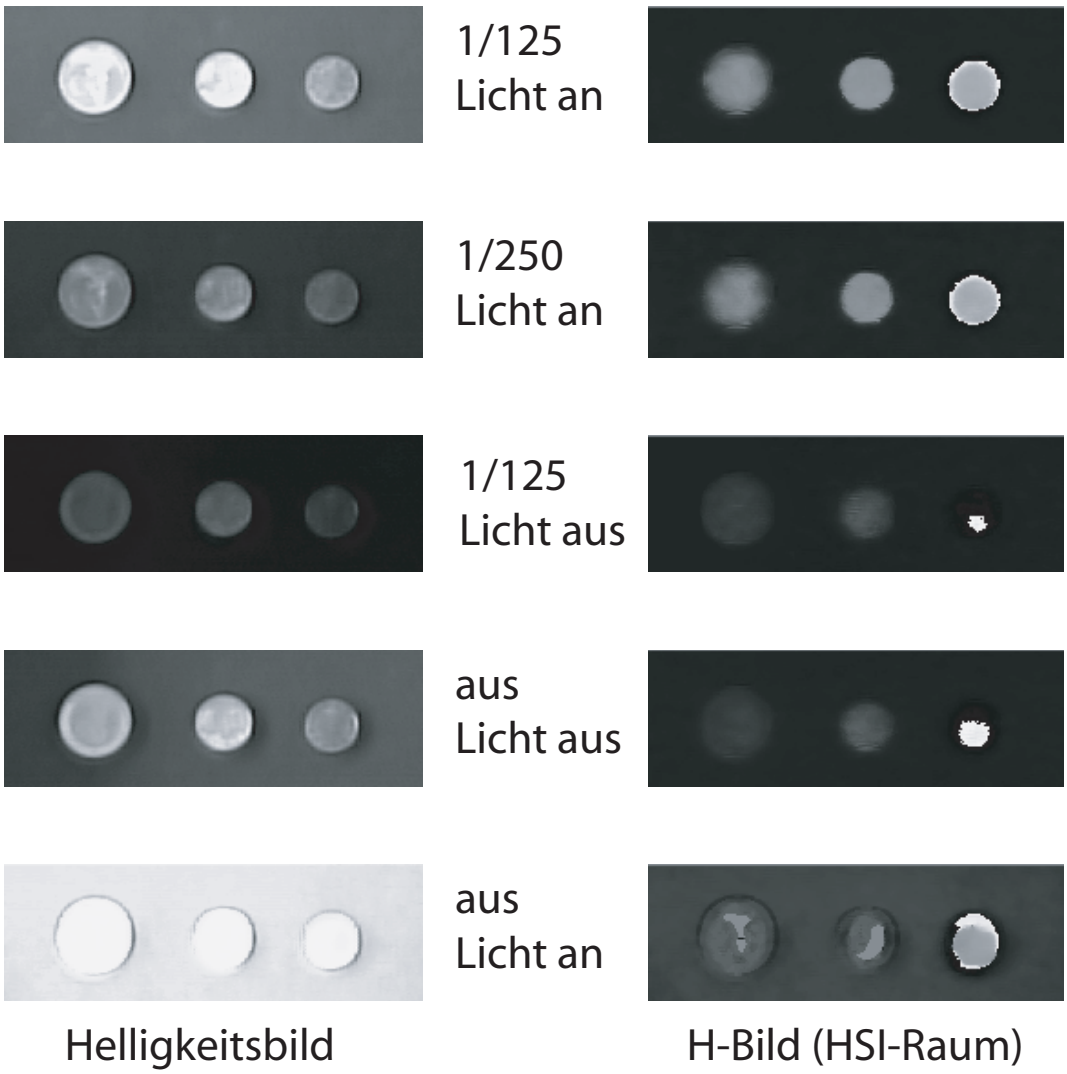


Abbildung 6.23: Farbraumunterschiede

Shutter 1/125 – Licht aus

Mit dieser Einstellung ergibt sich noch ein gutes Kontrastverhältnis im Helligkeitsbild, allerdings wird das Farbraumbild durch die fehlende Beleuchtung sichtbar schlechter. Die 2€ Münze hat nun eine gleichmäßige Färbung und ist der 10 ct Münze sehr ähnlich. Die 2 ct Münze geht größtenteils schon im Hintergrund unter. Der weiße Fleck entsteht durch Reflexionen.

Mit diesen Bildern würde die Münzerkennung wahrscheinlich an dem fehlenden Farbbereich für 1€ & 2€ und den Reflexionen scheitern.

Shutter aus – Licht aus

Der Shutter wurde für die Aufnahme ausgestellt. Das Helligkeitsbild ist dem ersten ähnlich, aber vom Kontrast etwas schlechter. Die 2 ct Münze ähnelt farblich dem Hintergrund. Das Farbraumbild gleicht dem vorherigen. Der weiße Fleck ist auf Grund der größeren Helligkeit und damit vermehrter Reflexionen etwas größer.

Eine korrekte Münzerkennung wird sehr wahrscheinlich nicht möglich sein.

Shutter aus – Licht an

Mit Anschalten der Beleuchtung bei ausgeschalteten Shutter ist nur sehr wenig Kontrast im Helligkeitsbild festzustellen. Damit ist die *Segmentierung* sehr schwierig. Das Farbraumbild besitzt wegen der starken Reflexionen keine Aussagekraft mehr.

Mit dieser Einstellung wird eine Münzerkennung nur in sehr geringem Maße möglich sein, bestenfalls können nur die Ergebnisse aus der *Vermessung* genutzt werden.

Für die richtige Erkennung der 1€ Münze ist die Farbinformation sehr elementar. Ist im Farbraumbild der innere Bereich der Münze nicht in einer anderen Farbe dargestellt, so lässt diese sich nicht mehr von der 50 ct Münze unterscheiden.

Bei der 2€ Münze ist dieses Merkmal nicht so wichtig, da diese sich durch die Größe eindeutig klassifizieren lässt.

Bei zu warmen Licht sind keine deutlichen Unterschiede der Münzen mehr festzustellen. Diffuses Licht ist deshalb sehr wichtig, weil es auf den Münzen wegen ihrer Oberflächenstruktur schnell zu Reflexionen kommt. Diese verursachen ebenfalls eine Verschlechterung des Farbraumbildes.

7 Das System

Dieses Kapitel beschreibt kurz das gesamte System und geht auf Inbetriebnahme und Ablauf ein. Am Ende folgen Abschlussbetrachtung und Ausblick.

7.1 Gesamtbeschreibung

Im Rahmen der Diplomarbeit ist ein modulares System von Hardware- und Softwaremodulen entstanden, um eine Sortierung von Stanzteilen, in diesem Fall Münzen, mit einem Roboter zu realisieren.

Alle Bereiche (Ansteuerungsschaltung und -software, Koordinatentransformation und Bildverarbeitung) sind eng miteinander verbunden. Trotzdem können die meisten Module in anderen Anwendungen eingesetzt werden. So ist z.B. durch die *cobra_use* Library (siehe Abschnitt 5.3) eine Schnittstelle entstanden mit der der Roboter Cobra RS2 angesteuert oder in einem Bildverarbeitungssystem verwendet werden kann.

Einen guten Überblick über das System gibt dabei der Signalflussplan des Bildverarbeitungssystem (siehe Abbildung 6.2) mit der Einschränkung, dass alle Entwicklungen zur Roboteransteuerung im Modul *cobra_steuerung* integriert sind.

Die einzelnen Abläufe und Funktionen sollen hier nicht noch einmal beschrieben werden, hierfür sei auf die detaillierten Beschreibungen in den vorherigen Kapiteln verwiesen.

7.2 Inbetriebnahme und Ablauf

Dieser Abschnitt gibt einen Überblick über die Voreinstellungen vor dem Start des System sowie den Ablauf unter *Ad Oculos* und DOS.

Kamera-Parameter	Einstellung
WHITE BALANCE	3200K
SHUTTER	OFF, 1/125, 1/250 (je nach Licht)
AGC	OFF
BACK LIGHT	OFF
IRIS LEVEL	Mitte
GAMMA	1.0
H. REVERSE	OFF
SENSITIVE	OFF

Tabelle 7.1: optimale Kameramenü-Parameter

7.2.1 Kalibrierung und Voreinstellungen

Vor der Inbetriebnahme des Systems müssen zunächst Roboter und Kamera kalibriert, sowie eine Kalibrierungsliste angelegt werden.

Alle zum Betrieb des Systems nötigen Vorbereitungen werden in diesem Abschnitt genauer erläutert.

1. Initialisierung der Ansteuerungsplatine

Dieser Punkt wurde in Abschnitt 3.4.2 schon erwähnt, soll aber der Vollständigkeit halber hier noch einmal aufgeführt werden.

Die Ansteuerungsplatine muss nach dem Einschalten einmal durch eine kurze Betätigung des Taster *S4* (siehe Abbildung 2.4) initialisiert werden.

2. Kamerakalibrierung und -parameter

Die Kamerakalibrierung (siehe auch Abschnitt 6.5.1) ist durch verschiedene Diplomarbeiten [10, 11, 12] auf einfache Art möglich. Die Ergebnisse werden dann in einer Datei ausgegeben, die im Bildverarbeitungssystem verwendet werden kann.

Damit das Bildverarbeitungssystem möglichst optimale Ergebnisse liefern kann, sollten im Menü der Kamera die Parameter in Tabelle 7.2.1 eingestellt werden.

3. Erstellen der Kalibrationsliste

Die Kalibrationsliste ist ein Eingabeparameter für das *Cobra-Steuerung* Modul bzw. für die *cobra_use* Library.

Die Liste wird benötigt, um einzelne Punkte aus dem Kamera-Weltkoordinatensystem in das Roboterkoordinatensystem umzurechnen (siehe Abschnitt 5.2).

Zur Erstellung wird folgender Ablauf vorgeschlagen:

1. Roboter in Kalibrierungsposition bringen (siehe folgenden Abschnitt)
2. Münze in der Mitte des Greifer befestigen (zu empfehlen ist die 2 ct Münze wegen ihrer Größe)
3. Position auf der Arbeitsfläche mit dem *CobraTESTer* anfahren (siehe Abschnitt 5.2)
4. Münze ablegen
5. Roboter aus dem Kamerabild entfernen und Bildverarbeitungssystem starten
6. Münzablage-Position und aus der Bildverarbeitung bestimmte Position in die Kalibrationsliste mit Kommata getrennt eintragen

Diesen Ablauf mit 5-10 verschiedenen Ablagepositionen wiederholen.

4. Roboterkalibrierung

Die Kalibrierung des Roboter beschränkt sich darauf, diesen in eine bestimmte Position zu bringen, die in Abbildung 7.1 genauer dargestellt ist.

Die Gelenkwinkel des Roboters an dieser Position sind in Tabelle 7.2.1 angegeben. Abgesehen vom Greifer befinden sich alle anderen Winkel in dieser Position im Anschlag.

Gelenk	Winkel	Einstellung
Drehturm	α	170°
Oberarm	β	-30°
Unterarm	γ	20°
Greifer	δ	-90°

Tabelle 7.2: Gelenkwinkel der Kalibrierungsposition Cobra RS2

Abbildung 7.1: Cobra RS2 in Kalibrierungsposition

Konkret heißt das, dass der Drehturm an den linken Anschlag muss, der Oberarm an den unteren Anschlag, der Unterarm an den oberen Anschlag und der Greifer senkrecht nach unten.

7.2.2 Ad Oculos Umgebung

Für die Inbetriebnahme unter der *Ad Oculos* Umgebung muss zunächst ein Signalflussplan (siehe Abbildung 6.2) mit allen Modulen erstellt und die zugehörigen Parameterdateien angepasst werden.

Danach kann das System einfach gestartet werden. Zuerst werden die Münzen durch die Bildverarbeitung erkannt und dann mit dem *Cobra-Steuerung* Modul sortiert.

7.2.3 DOS Umgebung – cobra_batch

Um die Münzerkennung und -sortierung auch ohne der *Ad Oculos* Oberfläche betreiben zu können, wurde jedes Modul zu einer Win32-Anwendung kompiliert.

Genau wie unter *Ad Oculos* benötigt jedes Modul eine Parameterdatei. Um die einzelnen Module zu einem System zu verknüpfen, müssen die Dateinamen für Ein- und Ausgabebilder in den Parameterdateien aufeinander aufbauen.

Das System startet durch den Aufruf einer Batch-Datei, in der zunächst das aktuelle Kamerabild im Bilder-Verzeichnis gespeichert wird. Danach werden die einzelnen Module der Bildverarbeitung in der richtigen Reihenfolge gestartet und schließlich die *Cobra-Steuerung* aufgerufen.

Auf der CD-ROM findet man unter dem Ordner *cobra_ batch* ein Beispiel für die DOS Umgebung.

7.3 Abschlussbetrachtung und Ausblick

Die Aufgabenstellung der Realisierung eines System zur bildverarbeitungsbasierten Sortierung von Stanzteilen mit Hilfe einer exemplarisch aufzubauenden Roboterzelle ist erfüllt worden.

Für die Zukunft wurde eine Roboterschnittstelle entwickelt, die auch neben dieser Diplomarbeit Anwendungen finden kann. Gleiches gilt für einige Module aus dem Bildverarbeitungssystem.

Die Komponenten Cobra RS2 und das Bildverarbeitungssystem bieten noch einige Verbesserungsmöglichkeiten.

Der Roboter Cobra RS2, der für die Diplomarbeit zur Verfügung gestellt wurde, hat mittlerweile ein Alter von 20 Jahren erreicht.

Ein größeres Defizit des Roboters betrifft die Aufhängung des Oberarms. Dieser kann unter einem Winkel von 70 Grad nicht mehr stabil gehalten werden.

Aus diesem Grund musste die Ansteuerungsfrequenz des Roboters heruntergesetzt werden und der Schrittmotor zur Steuerung des Oberarms im jeweiligen Schritt gehalten werden. Andernfalls (bei Freischaltung des Motors) ist es nicht möglich den Arm ohne Schritt- bzw. Genauigkeitsverlust in diesem Bereich zu bewegen.

Der Genauigkeitsverlust nach mehreren Ansteuerungsvorgängen wird durch eine Zwischenkalibrierung möglichst gering gehalten. Dabei fährt der Roboter in die Kalibrierungs-

position und es werden für Drehturm, Ober- und Unterarm einige Schritte gefahren, so dass alle Gelenke im Anschlag sind.

Eine weitere Verbesserung, die schon in der Konstruktion ist, betrifft die Ansteuerung von Objekten auf der Arbeitsfläche.

Da es eine kleine Ansteuerungsungenauigkeit in der Anfahrhöhe gibt, wurde ein Aufsatz für den Greifer entworfen. Dieser soll den Druck auf die Arbeitsfläche durch eine Federung möglichst gering halten, damit sich der Arm des Roboter beim Schließen des Greifers nicht mehr nach vorne drücken kann. Dieses Problem entsteht bei einer zu tiefen Greiferposition. Der Greifer drückt dann oftmals zu stark auf die Arbeitsfläche.

Wie in Abschnitt 6.5 dargestellt, hat das Bildverarbeitungssystem einige Unsicherheiten, die zu Fehlern in der Münzerkennung führen können.

Kurz zu nennen sind dabei das Bild-Rauschen und das Farbraumverhalten bei unterschiedlicher Beleuchtung.

Zur Zeit ist das System auf Grund einer nicht vorhandenen mobilen, diffusen Beleuchtung nur eingeschränkt transportabel. Für die Zukunft als Demonstrationsobjekt ist dieses Problem noch zu lösen.

Eine Möglichkeit wäre, um das ganze System einen Kasten zu bauen und so von allen Umwelteinflüssen abzuschotten. Das ist für Bildverarbeitungssysteme eine gebräuchliche Lösung, allerdings würde in diesem Fall der Sortierungsvorgang nur schlecht zu sehen sein.

Ein weiterer Ansatzpunkt zur Verbesserung der Anfahrgenauigkeit des Roboters ist mit Hilfe der Bildverarbeitung möglich. Bisher ist die Positionsbestimmung des Cobra RS2 im Kamerabild nicht möglich.

Diese könnte mit entsprechenden Messmarken am Roboter und weiteren Funktionen verwirklicht werden.

Teil II

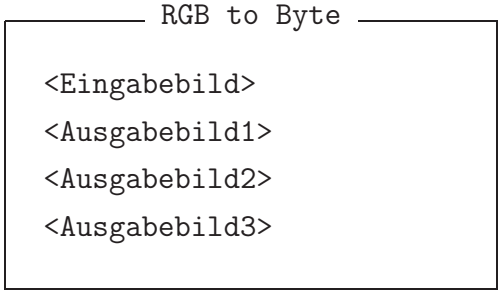
Anhang

A Parameter der Bildverarbeitungsmodule

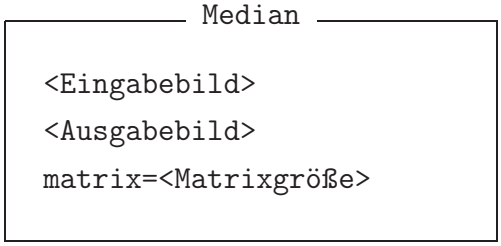
Nachfolgend eine Beschreibung der Parameterdateien für die einzelnen Module der Bildverarbeitung.

Für die Ein- und Ausgabebilder müssen unter der DOS-Umgebung (siehe Abschnitt 7.2.3) Dateiangaben gemacht werden, z.B. `c:\cobra_batch\eingabe.bmp`. Unter Ad Oculos z.B. `#I0` (Eingabebild) und `#O2` (Ausgabebild) je nach Nummer des Bildes.

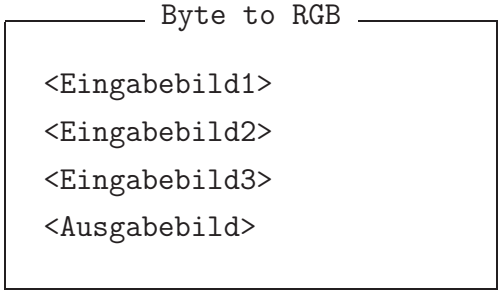
RGB to Byte



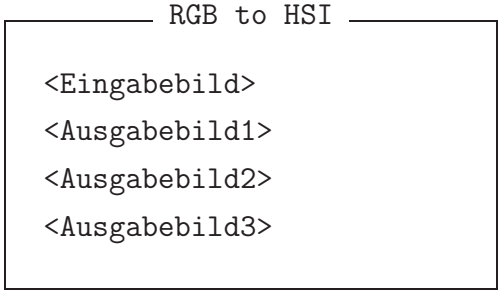
Median



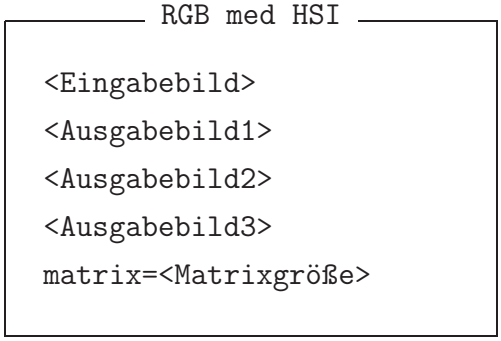
Byte to RGB



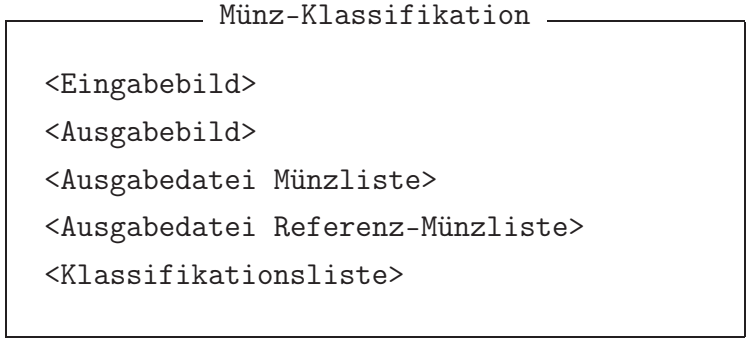
RGB to HSI



RGB med HSI



Münz-Klassifikation



Cobra-Steuerung

Cobra-Steuerung

```
<Eingabebild>
<Ausgabebild>
<Münzliste>
<Kalibrationsliste>
<Ausgabedatei Roboterposition>
speed=<Geschwindigkeit (0 = Vollschritte / 1 = Halbschritte)>
hoehe=<Anfahrhöhe der Münzen in mm>
```

B Inhalt der CD-ROM

Nachfolgend eine Beschreibung des CD-ROM Inhalts.

Bildverarbeitung
: enthält alle im Rahmen der Diplomarbeit verwendeten und erstellten Bildverarbeitungsmodule im Quelltext.

Cobradll
: enthält die Cobradll, sowie Beispielprogramme und Anwendungen.

Diplom
: enthält diese Diplomarbeit als PDF-Dokument.

Doku
: enthält verschiedene Datenblätter, Handbücher und Dokumentationen der verwendeten Hard- und Software.

IBV
: enthält die Bildverarbeitungs- und Ansteuerungsmodule in der vom Labor verwendeten Struktur inklusive aller Parameterdateien.

Koordinatentransformation
: enthält den Quelltext der Dynamic Link Library zur Koordinatentransformation.

Tools
: enthält alle verwendeten SDKs.

VHDL
: enthält alle erstellten VHDL-Module sowie das zugehörige *MAX+plus II* Projekt mit dem Schaltplan.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] El Camino: *DIGILAB 10K10 Handbuch.* URL, `http://www.elca.de/Downloads/Handbuch 10K10.pdf`. Kopie auf CD-ROM.

[2] ALTERA: *FLEX 10K Embedded Programmable Logic Family Data Sheet.* URL, `http://www.altera.com/literature/ds/dsmii.pdf`, Jan 1998. Kopie auf CD-ROM.

[3] JAI: *CV-S3200 1/2,Data Sheet.* URL, `http://www.jai.com/db_datasheet/cv-s3200_3300db.pdf`, Mai 2002. Kopie auf CD-ROM.

[4] The Imaging Source: *DFG/LC1 Users Manual.* URL, `http://www.theimagingsource.com/prod/grab/dfglc1/dfglcx_eng_man.pdf`. Kopie auf CD-ROM.

[5] ALTERA: *MAX+PLUS II Programmable Logic Development System & Software Data Sheet.* URL, `http://www.altera.com/literature/ds/dsf10k.pdf`, Jan 2003. Kopie auf CD-ROM.

[6] DBS: *Ad Oculos Users Manual.* URL, `http://www.theimagingsource.com/prod/soft/adoculos/adoculos_docs.htm`, Jun 2003. Kopie auf CD-ROM.

[7] Bässmann, H.: *Bildverarbeitung Ad Oculos.* Springer Verlag, 3. Auflage, 1998.

[8] Koch, A.: *Hardware- und Softwaremodule für eine exemplarisch aufzubauende Roboterzelle zur bildverarbeitungsbasierten Sortierung von Stanzteilen.* Diplomarbeit, Fachhochschule Köln, Digitaltechnik und digitale Bildverarbeitung, Jan 2004.

[9] Bronstein, Semendjajew, Musiol, Mühlig: *Taschenbuch der Mathematik.* Verlag Harri Deutsch, 5. Auflage, 2000, 2001.

[10] Kakou, G. N.: *Ein Verfahren zur 3-D Vermessung von Werkstücken mittels CCD-Kamera und Projektor.* Diplomarbeit, Fachhochschule Köln, Digitaltechnik und digitale Bildverarbeitung, Jun 2002.

[11] Smolarczyk, M.: *Detektion und grobe Positionsbestimmung von Messmarken in Kamerabildern.* Diplomarbeit, Fachhochschule Köln, Digitaltechnik und digitale Bildverarbeitung, 2004.

[12] Gress, H.: *Subpixelgenaue Positionsbestimmung von Messmarken und darauf aufbauende exemplarische Kamerakalibrierung.* Diplomarbeit, Fachhochschule Köln, Digitaltechnik und digitale Bildverarbeitung, 2004.

[13] Europäische Zentralbank: *Euro-Münzen.* URL, `http://www.euro.ecb.int/de/section/euro0/coins.html`.