# DIPLOMARBEIT

# Montage mit Industrieroboter - Ein Laborversuch

**ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von**

o. Univ.Prof. Dipl.-Ing. Dr. Dr.hc.mult. Peter Kopacek

E 325

Institut für Mechanik und Mechatronik

(Abteilung für Intelligente Handhabungs- und Robotertechnik)

eingereicht an der Technischen Universität Wien

**Fakultät für Maschinenwesen und Betriebswissenschaften**

von

Florian Hochreiter

0025460

Adeldorfer Straße 5, 3143 Pyhra

Wien, am 15.12.2005

# INHALTSVERZEICHNIS

**Abbildungsverzeichnis:**

# Vorwort

Dass ich mich für ein Diplomarbeitsthema im Bereich der Mechatronik entschieden habe ist keineswegs Zufall. Nachdem ich meine HTL-Ausbildung in eben diesem Bereich abgeschlossen hatte, wollte ich eigentlich mein Maschinenbaustudium dafür nutzen, meine Kenntnisse im Bereich der Konstruktion und Projektierung zu vertiefen. Am Beginn meines Studiums hat man mir versichert, dass gerade der klassische Maschinenbau sehr gefragt ist. Während der letzten Jahre musste ich jedoch feststellen, dass heutzutage der Maschinenbau ohne die dazugehörige Steuerungs- und Regelungstechnik nicht mehr denkbar ist. Selbst in den Vorlesungen meines Vertiefungsblocks, der Fördertechnik, wurde ich immer wieder damit konfrontiert, dass die Mechatronik auch hier schon ihren fixen Platz gefunden hatte.

Nun habe ich mich also entschlossen eine Arbeit am Institut für Handhabungsgeräte und Robotertechnik zu schreiben. Dass ich gerade diese Abteilung gewählt habe, liegt wohl daran, dass mich die Robotik schon seit meiner Kindheit fasziniert. Umso mehr habe ich mich darüber gefreut, dass mir Prof. Kopacek eine Arbeit an der laboreigenen Montagezelle, welche mit drei Robotern bestückt war, angeboten hat. Voll Enthusiasmus habe ich mich dann in diese Arbeit gestürzt und war dabei stets felsenfest davon überzeugt, dass dieses komplexe Gerät von mir in Gang gesetzt werden müsste. Allerdings habe ich gerade diese Komplexität unterschätzt. Abgesehen von meinen eigentlichen Aufgaben musste ich mich zum Teil mit sehr detaillierten Beschreibungen von einzelnen Objekten auseinandersetzen, wovon jede für sich wohl schon für eine Diplomarbeit gereicht hätte.

Schlussendlich musste ich mich bei dieser Arbeit geschlagen geben, habe jedoch in den letzen Monaten weit mehr dazugelernt als in den letzten Jahren.

An diesem Punkt möchte ich mich für die großartige Betreuung bei den Herren des Instituts für Handhabungsgeräte und Robotertechnik bedanken, die mir bei all meinen Problemen hilfreich zur Seite standen und zumindest versucht haben mit zum Erfolg zu verhelfen.

F. Hochreiter, Oktober 2005

# 1 Einleitung

Im Labor des Instituts für Robotertechnik und Handhabungsgeräte wurde vor ungefähr zehn Jahren eine Anlage entworfen, welche als Übungsobjekt im Rahmen der Laborübungen genutzt werden sollte. Diese Anlage wurde aus verschiedenen, in der Praxis sehr häufig anzutreffenden Elementen zusammengestellt. Die Aufgabe dieser Anlage ist die Montage eines Malteserkreuzgetriebes.

Eigentlicher Sinn dieser Montagezelle war es, eine neu entwickelte Software zur objektorientierten Steuerung von Fertigungszellen zu testen und die Basis für Weiterentwicklungen zu schaffen.
Im Laufe der Zeit wurden mehrere Veränderungen an der Anlage vorgenommen. Da nun die Gesamtanlage schon seit mehreren Jahren nicht mehr in Betrieb genommen wurde, sollten nun Informationen zum Betrieb gesammelt und die Anlage wieder in betriebsfähigen Zustand versetzet werden.

Es mussten die Funktionen der einzelnen Elemente überprüft und, sofern erforderlich, wieder Instand gesetzt werden. Um die erforderliche Neuprogrammierung des Systems zu ermöglichen, mussten die einzelnen Roboter-Programmiersprachen erlernt werden. Darüber hinaus war es erforderlich die hardwaremäßige Vernetzung der Anlage zu kontrollieren und zu erneuern.
Darüber hinaus sollte versucht werden, das Terminal des PUMA 560 mit dem Steuerrechner zu verbinden, da insbesondere die Speicherung der Programme, welche für die Inbetriebnahme der Anlage von hoher Wichtigkeit ist, mit dem vorhandenen Diskettenlaufwerk nicht mehr ausreichend sicher ist.
Die in der Montagezelle integrierten Roboter sind zudem noch für die Laborübungen eingesetzt worden. Dadurch kamen Studenten, welche zum Teil nicht über das nötige Feingefühl für solche sensible technische Systeme haben, mit dem Gerät in Berührung. Es ist deshalb nicht verwunderlich, dass einzelne ungesicherte Teile beschädigt wurden.

Diese Anlage wurde in den letzten Jahren ständig erweitert, umgebaut und modifiziert. Es wurden mehrere Projektarbeiten an verschiedenen Teilen dieser Montagezelle durchgeführt. Die meisten dieser Arbeiten wurden leider nicht oder nur unzureichend dokumentiert. Dies erschwerte die Arbeit ungemein. Dieses schriftliche Dokument soll als Basis für weiterführende Arbeiten dienen, in dem es den derzeitigen Zustand der einzelnen Komponenten beschreibt. Es soll darüber hinaus Anregungen für Verbesserungen liefern.

# 2 Aufbau der Anlage

Die Anlage im Labor des Instituts ist als Versuchsanlage gedacht und wurde aus unterschiedlichen Modulen zu einer Montagezelle zusammengesetzt. Aufgabe der Anlage ist es, ein montagefreundliches Malteserkreuzgetriebe zusammenzubauen.
Dieses besteht aus:

1) Malteser-Rad (Aluminium-Leg.)
2) Ritzel (Aluminium-Leg.)
3) Gehäuse (aus Thermoplast. Kunststoff)
4) Deckel (aus Acrylglas)

Das ursprüngliche Zellenkonzept sah vor, dass beide Räder (aus Einzelteilen aufgebaut) zuerst in der Anlage vormontiert (Achse in Rad eingepresst) und erst danach in das Gehäuse montiert werden. Dieser Ablauf wurde allerdings aus praktischen Gründen nie umgesetzt. Sowohl Malteser-Rad als auch das Ritzel wurden im vormontierten Zustand magaziniert.

## *2.1 Komponenten der Anlage*

Die Anlage besteht im Wesentlichen aus drei Industrierobotern unterschiedlicher Bauart und einem Förderband, welches die Teile auf speziell für dieses Fördersystem konzipierten Paletten transportiert.
Die einzelnen Elemente der Fertigungszelle werden nun folgend näher beschrieben.
(Zur Vernetzung der einzelnen Elemente wird später ausführlich Stellung genommen)

## Unimation PUMA 560


**Abbildung 1: PUMA-Roboter**

Der Puma 560 von Unimation® ist der älteste in dieser Montagezelle verwendete Industrieroboter. Er ist als Knickarmroboter mit 6 Drehachsen gebaut und wird heute fast nur mehr für Laborzwecke an vielen Technischen Hochschulen in Europa verwendet.

Der Roboter ist mit einer Eigenmasse von 60 kg und einer zulässigen Last von 1,9 kg eher für einfache Aufgaben konzipiert. Sowohl die geringen Abmaße als auch die niedrige Nutzlast sprechen dafür, dass dieser Roboter eher für Handhabungsaufgaben konstruiert wurde. Für die Aufnahme von Teilen ist er mit einem universell einsetzbaren pneumatischen Parallelgreifer ausgestattet, welcher direkt über das Steuerterminal programmiert, bzw. mit dem Handbediengerät angesteuert werden kann.
Über das Bedienerterminal können Programme eingegeben und mit dem integrierten Diskettenlaufwerk gesichert werden.

Aufgabe dieses Roboters in der Fertigungszelle war es, die im Magazin vorbereiteten Gehäuseteile auf Palette zu legen und nach erfolgter Montage das Gehäuse zu applizieren.
Bei der Aufarbeitung dieses Roboters gab es einige Probleme die auf die überalterte Hardware der Steuerung zurückzuführen sind.
Diese Probleme werden im Kapitel 4 näher erläutert.

### 2.1.1 Yaskawa MOTOMAN K6SB

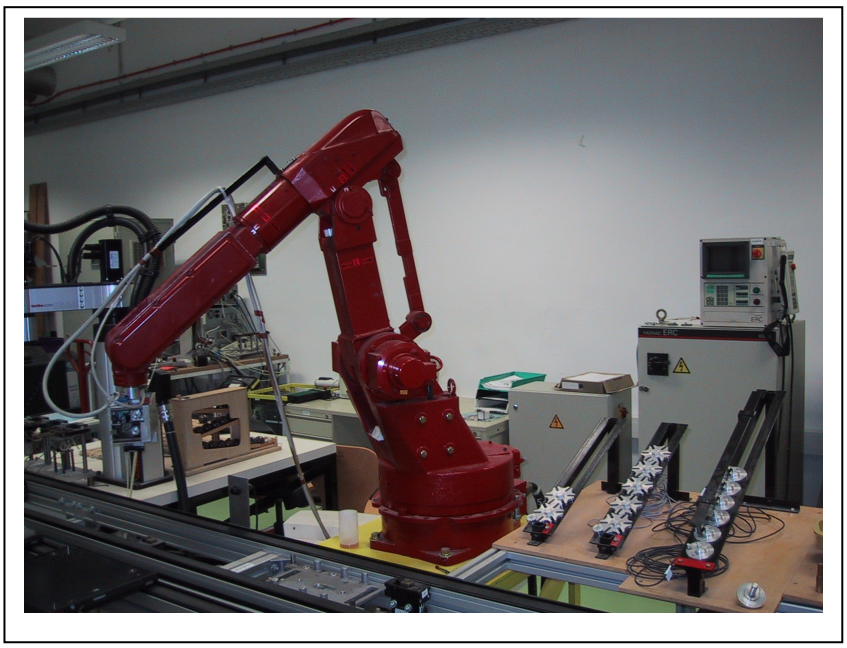


**Abbildung 2: Yaskawa-Roboter**

Dieser in Japan gebaute Industrieroboter besitzt ebenso wie der Puma 6 Drehfreiheitsgrade, hat jedoch einen anderen mechanischen Aufbau. Mit rund 180kg Masse ist er auch fast 3-mal so schwer wie der PUMA 560. Der Motoman ist als Schweißroboter konstruiert, und besitzt, abgesehen von der sehr robusten Konstruktion einige Steuerungselemente und vorprogrammierte Features, welche speziell für das Schweißen ausgelegt wurden (z.B. Pendelbewegung beim Elektro-Schweißen). Er eignet sich aufgrund seiner Bauweise aber ebenso für Montageaufgaben und kann bei Bedarf auch höhere Lasten tragen (bis 6 kg). Die einzelnen Achsen werden mit AC-Motoren angetrieben. Jede Antriebseinheit

besitzt darüber hinaus noch eine Bremseinheit sowie einen Absolut-Encoder als Wegmesseinrichtung.

Für die Montagearbeit ist er mit einem am Institut entwickelten Servogreifer ausgestattet, der die zulässige Nutzlast beträchtlich reduziert.
Der Roboter ist über ein Stromversorgungs- und ein Steuerkabel mit dem Steuerschrank verbunden. Der Steuerrechner selbst wird über einen externen Transformator mit Strom versorgt. Auf dem Steuerschrank befinden sich ein Bildschirmteil und ein Bedienteil mit Funktionstasten. Dieser wird in erster Linie zum Abspielen von Programmen verwendet.
Zum Teachen von Programmen wird üblicherweise das mit dem Bedienteil verbundene Handbediengerät eingesetzt.

Die Aufgabe des Yaskawa in der Montagezelle besteht darin, beide Räder vom Magazin zu nehmen und auf die Palette in speziell dafür vorgesehene Positioniereinrichtungen zu legen.
Alle Einzelheiten zum Roboter sind dem Benutzerhandbuch bzw. den vom Institut für Handhabungsgeräte und Robotertechnik zusammengestellten Arbeitsunterlagen (für Labor-Übungen) zu entnehmen.

### 2.1.2 Bosch SCARA SR60

Der TurboScara SR60 besitzt 4 frei programmierbare Achsen. Durch seine hohe Geschwindigkeit und Traglast ist er für Montage- und Pick&Place-Aufgaben sehr gut geeignet. Der Arbeitsraum ist ringförmig mit relativ geringem Innenradius und einer Reichweite von 600mm.

Der Roboter ist mit der frei programmierbaren Robotersteuerung IQ 140 mit Mehrprozessorsteuerung ausgerüstet. Dieses Multitasking-Betriebssystem ermöglicht die parallele Steuerung von mehreren unabhängigen Bewegungen. Die Programmierung kann zwar auch mit dem Handbediengerät erfolgen, einfacher

und weit aus flexibler in der Handhabung ist jedoch die Möglichkeit, über einen IBM-kompatiblen PC in einem Editor Programme zu erstellen. Diese werden von

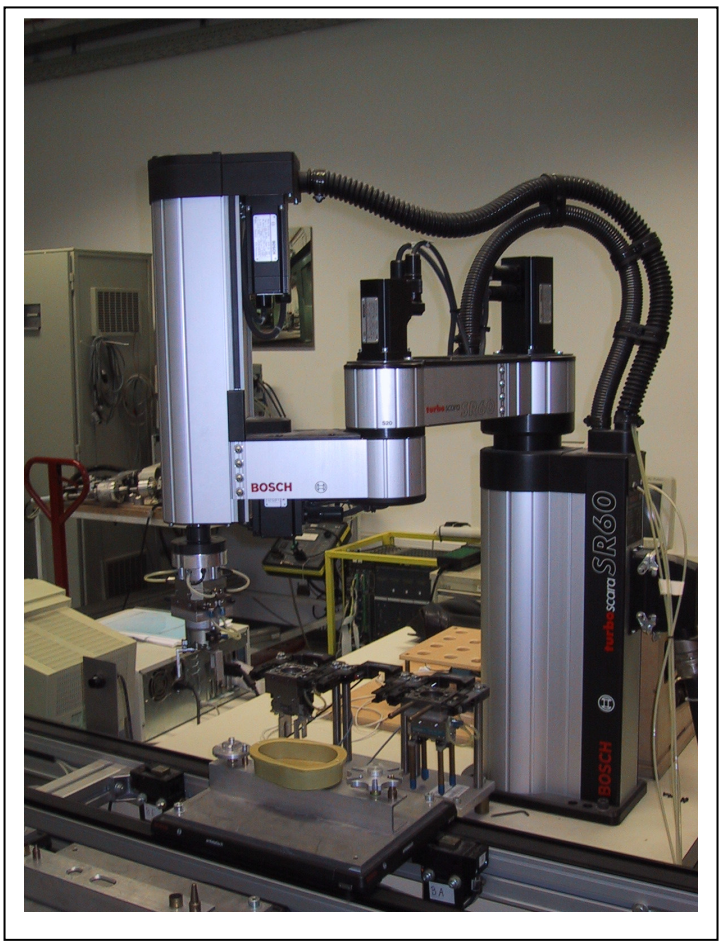



**Abbildung 3: BoschSCARA-Roboter**

einem Übersetzer (BABS) kompiliert und in den für die Steuerung erforderlichen Code umgesetzt. Mit der von BOSCH gelieferten Software lassen sich die Steuerprogramme (max. 10, eines davon ist für die Initialisierungsroutine belegt) direkt über ein serielles Kabel in den steuerinternen Speicher übertragen.

Der SCARA nutzt ein Greiferwechselsystem zum flexiblen Einsatz als Montageroboter. Dieses Greifersystem beinhaltet drei unterschiedliche Greifer, von denen zwei für die Handhabung der beiden Räder verwendet werden. Diese werden in diesem Arbeitsschritt in das Gehäuse fertig montiert.

### 2.1.3 Bosch Fördersystem

Das Förderbandsystem dient als mechanische Schnittstelle zwischen den Bearbeitungsstätten.
Es sorgt für den Transport der Teile, welche auf für dieses System konzipierten Paletten auf dem Förderband laufen. Diese Paletten können je nach Bedarf mit speziellen Halterungen und Montagehilfsmitteln bestückt werden.

Das Förderband selbst besteht aus zwei gegeneinander laufenden Systemen, welche je zwei von einem Getriebemotor angetriebenen Gewebeförderbänder besitzen.
Als Verbindung der beiden unabhängig angesteuerten Förderbänder dienen Querförderer an den Enden. Diese werden, sobald sich eine Palette dem Ende der Förderstrecke nähert, aktiviert und sorgen dafür, dass die Palette die Richtung wechselt. Damit wird der Kreislauf geschlossen.

Die Paletten können mit speziellen Stoppern, die pneumatisch angesteuert werden, an bestimmten Stellen am Förderband angehalten werden, während sich das Förderband kontinuierlich weiterbewegt.
Des weiteren befinden sich an den einzelnen Bearbeitungsstationen Hubtische, die dafür sorgen, dass die Paletten vom Förderband gehoben, und in eine genau definierte Position gebracht werden.

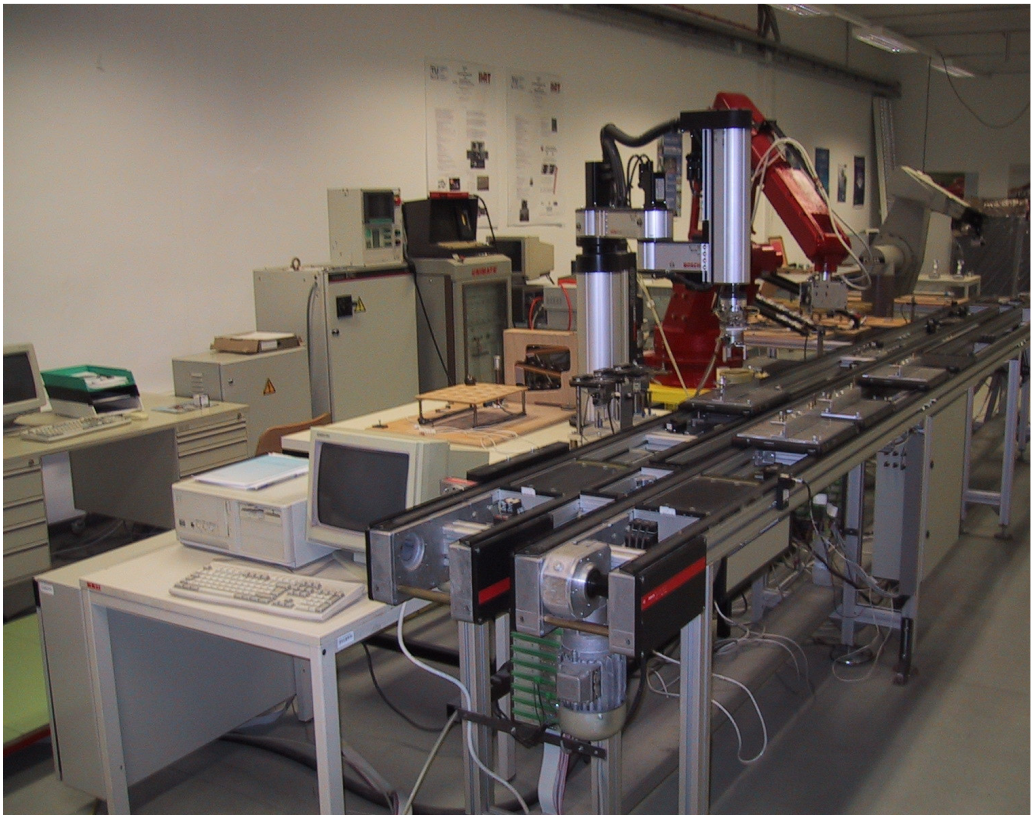


**Abbildung 4: Fördersystem**

### 2.1.4 Steuerrechner

Eigentliches Kernstück der Anlage ist der Steuerrechner, der den gesamten Prozessablauf koordiniert und überwacht. Der Rechner selbst ist ein schon etwas älterer IBM-Standard-PC, welcher mit einer ACL-722-Karte ausgerüstet ist. Diese Karte kann ähnlich wie eine parallele Schnittstelle programmiert werden.

Neben dem PC-Tisch befindet sich ein Steuerschrank, welcher für die Signalaufbereitung verantwortlich ist. Breitbandkabel (24 Kanäle) sorgen für den Informationsaustausch zwischen PC und der Montageanlage.

Die genaue Funktion dieses Systems wird im nächsten Kapitel genauer beschrieben.

# 3 Funktion der Anlage

## 3.1 Allgemein

Die im vorigen Kapitel besprochenen Komponenten werden mittels eines übergeordneten Steuerrechners vernetzt. Die Signale werden mit einer I/O-Karte ACL-722 verarbeitet. In einem direkt angeschlossenen Schaltschrank werden die Signale entsprechend aufbereitet, d.h. auf den für das System erforderlichen Signalpegel von 24 V angehoben. Diese Karte besitzt insgesamt 6 Kanäle (CH0-CH5 mit je 24 Bits) mit je 3 Ports (á 8 Bit). => das heißt, es stehen insgesamt 144 Signalleitungen zur Verfügung.
Die ersten 3 Kanäle (CH0-CH2) werden für Signal-Eingänge, die Kanäle CH3-CH5 als Ausgangsleitungen verwendet (bezogen auf I/O-Karte).

Die detaillierte Belegung der einzelnen Funktionselemente wird im Kapitel 6.2 aufgeführt. Diese Liste stellt den aktuellen Stand der Verdrahtung dar.
Daraus ist zu ersehen, dass CH2 und CH5 noch nicht belegt sind. Diese Kanäle können daher für eventuelle Erweiterungen der Anlage herangezogen werden.
Zur eigentlichen Steuerung der Anlage wird ein Softwarepaket verwendet, welches im Kapitel 3.3 kurz beschrieben wird.

## 3.2 Vernetzung

Hier wird nun die grundsätzliche Vernetzung der Anlage dargestellt. Diese dient lediglich dem Verständnis und ist keineswegs als Schaltplan oder dergleichen zu verstehen. Die Anzahl der Leitungen und die genaue Funktion der Signale sind der Pin-Belegung zu entnehmen (Anhang II). Ebenfalls in diesem Kapitel ist eine graphische Darstellung der genauen Verdrahtung der Förderanlage zu finden.

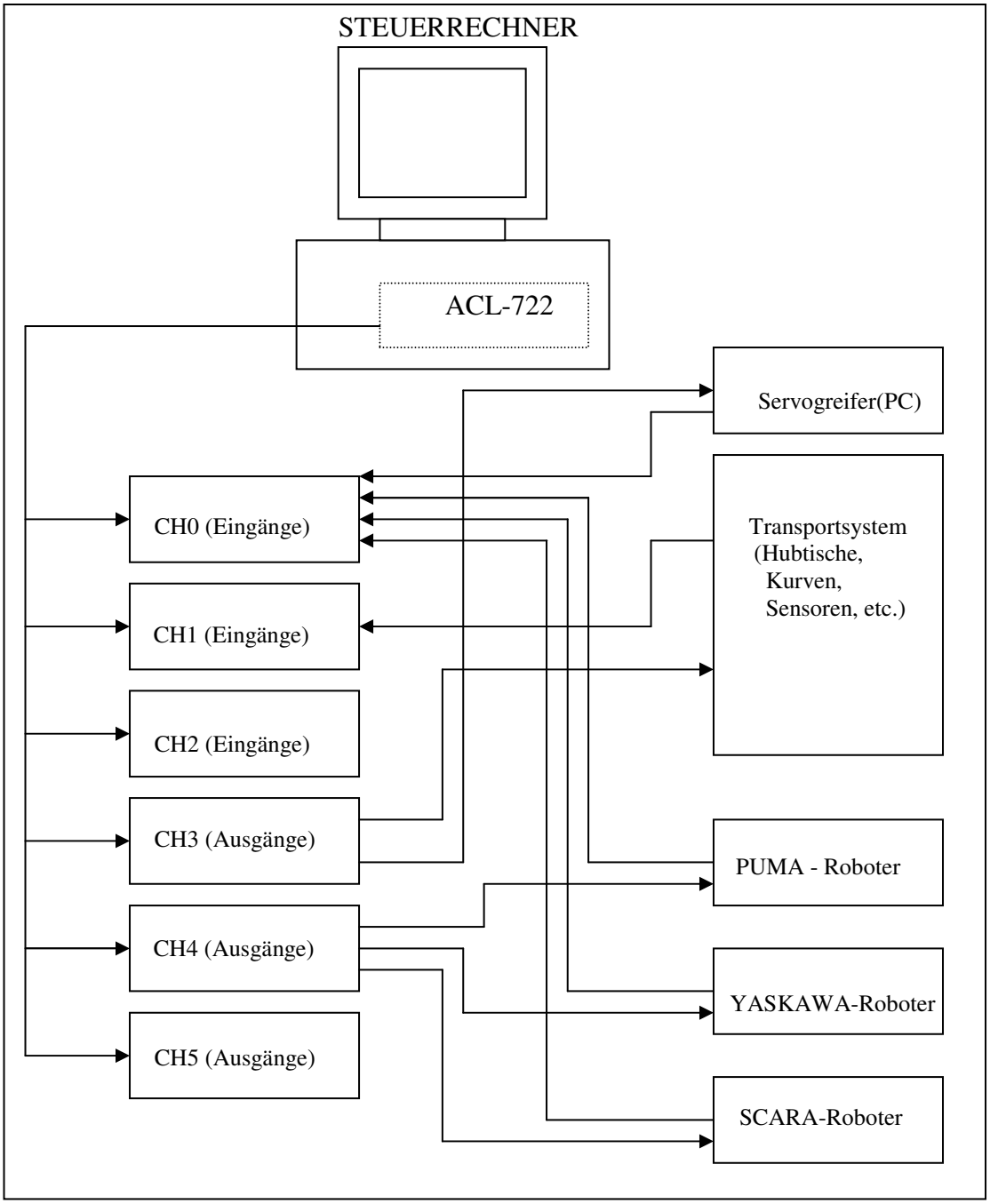



**Abbildung 5: Vernetzung der Anlage**

## 3.3 Einbindung der Industrieroboter in die Fertigungszelle

Die folgende Beschreibung gilt grundsätzlich für alle an der Montagezelle beteiligten Roboter und soll lediglich die Möglichkeiten der Ansteuerung näher erklären.

Um Industrieroboter von einem übergeordneten PC (Host) aus zu steuern, benötigt ein Roboter eine Schnittstelle, mit der er programmgesteuert mit der Peripherie kommunizieren kann.
Die einfachste Möglichkeit ist, mittels digitaler Eingangs/Ausgangs-Karte eines Robotersystems die Anbindung zu realisieren. Solche Schnittstellen arbeiten üblicherweise mit einem genormten Signalpegel (meist 12V od. 24V). Bei den im Labor aufgestellten Robotern liegen die Signalpegel jeweils bei 24V (logisch High) und 0V (Low). Dadurch wird die Vernetzung wesentlich einfacher, da sich dadurch die Signalaufbereitung vereinheitlicht. Die Ein/Ausgangsschnittstellen sind darüber hinaus üblicherweise mittels Optokoppler galvanisch von der Peripherie getrennt.

Diese Schnittstellen lassen sich im Roboterprogramm bearbeiten, d.h. Eingangssignale können verknüpft und in Bedingungen (z.B. IF-Bedingung) eingebunden werden, Ausgangssignale können gesetzt werden um z.B. Informationen über den derzeitigen Stand des Ablaufs zu übermitteln.
In der aufgebauten Montagezelle werden die Eingangssignale zur Auswahl des Roboterprogramms (eigentlich Unterprogramm) verwendet und die Ausgangssignale als Bestätigung dieser Programmnummer bzw. zum Senden des Status (busy/standby) verwendet.

Die Eingangssignale (Programmnummer) werden im Hauptprogramm kontinuierlich abgefragt. Ein weiteres Signal gibt dann Bescheid wenn die Programmnummer vom Roboterprogramm gelesen werden soll (ähnlich wie strobe-Signal bei paralleler Kommunikation). Sobald die Programmnummer eingetroffen ist, wird diese mittel IF-Bedingung verknüpft und das zugehörige

Unterprogramm aufgerufen. Im Unterprogramm wird nun als erstes das busy-Signal auf HIGH gesetzt um dem übergeordneten Steuerrechner mitzuteilen, dass das Programm abgearbeitet wird. Sobald das Programm beendet ist, wird das busy-Signal wieder auf LOW und das standby-Signal auf HIGH gesetzt, um anzugeben, dass die nächste Programmnummer gesendet werden kann.
Detaillierte Informationen zur Anbindung und Programmierung dieser Peripherie-Schnittstellen sind den Handbüchern der jeweiligen Roboter zu entnehmen.

## *3.4 Zellensteuerung mit C_CTRL*

Zur Steuerung der Zelle wurde das am Institut entwickelte Software-Paket C_CTRL herangezogen. Dieses Steuerungsprogramm ist, obwohl noch für den Betrieb unter MS-DOS konzipiert, eine einfache und funktionelle Software, die beim Vernetzen von Fertigungszellen sehr hilfreich ist.
Dieses Softwarepaket besteht aus den Programmen C_SEQU und C_CTRL.
C_SEQU dient zur automatischen Generierung der Anwenderprogramme, d.h. der Zusammenstellung der Bearbeitungs- und Montageoperationen.
C_CTRL stellt den eigentlichen Steuerungs- und Überwachungsmechanismus dar und macht den Rechner (PC) zur übergeordneten Steuerungsinstanz der gesamten Anlage.
Durch den modularen Aufbau der Software und einer strukturierten Anwenderbibliothek ist es möglich, auf einfache Art und Weise Gesamtprogramme zusammenzustellen und Teilprogramme hinzuzufügen, zu löschen und zu modifizieren.
Durch Kombinieren von Subprogrammen können in kurzer Zeit neue Programme erstellt bzw. technische Erweiterungen an der Anlage realisiert werden.
Dieses System ist sehr stark von der am Institut aufgebauten Roboterzelle beeinflusst, wodurch einige Module sehr speziell konzipiert sind. So ist z.B. das Modul „Greifer“ an den, am Yaskawa-Motoman Roboter applizierten Servogreifer angepasst. Auf der anderen Seite gibt es einige Module, die sehr

häufig in Zellen vorkommen (wie z.B. Magazin-Steuerung, Förderband-Schleusen, Pneumatik-Elemente, Robotersteuerung), die dann sehr einfach in eine Fertigungszelle eingebracht werden können. Die benutzerspezifische Benennung d. generierten Funktionen erleichtert sehr wesentlich die Zusammenstellung von Programmen. Für spezielle Aufgaben muss der Source-Code von C_SEQU entsprechend erweitert werden, um die Möglichkeit einer Steuerung mit C_CTRL zu realisieren.
Für genauere Informationen zu diesem Softwarepaket gibt es ein Skriptum, welches im Labor aufliegt.

## 3.5 Inbetriebnahme

Die Anlage ist derzeit in nicht betriebsfähigem Zustand, d.h. dass nur ein Teil der Anlage betrieben werden kann. Zurzeit kann lediglich der SCARA-Roboter seine Aufgabe ausführen. Nämlich die Montage des Malteser-Rades und des Ritzels von der bereits vorbereiteten Palette. Diese muss mit Gehäuse und den beiden Rädern bestückt sein.
Um diesen Teil der Anlage betreiben zu können, muss folgende Routine abgearbeitet werden:

a) Anlage an Strom anschließen (roter Netzstecker)
b) Die Anlage mit Druckluft versorgen (Kompressor an das Labornetz anschließen und Kompressor einschalten)
c) Sperrhahn am Förderband umlegen, damit die Anlage mit Druckluft versorgt wird
d) Bosch-SCARA starten
    a. Hauptschalter am Steuerkasten ein
    b. Warten bis HBG betriebsbereit
    c. Das Zählrad am Steuerkasten auf 9 (Init-Routine) stellen
    d. Starten der Routine mit „Servo ein“ und „Programm starten“ (Umschalthebel muss dazu auf Automatikbetrieb gestellt sein)
    e. Programm Nr. 8 einstellen und „Programm starten“ drücken

e) Steuerrechner starten
f) Nach dem Hochfahren in die Kommandozeile „Malteser“ eingeben
g) Die nun folgende Start-Eingabe mit Enter überspringen
h) Als *.cfg-Konfigurationsdatei „malteser“ wählen
i) Als *.cel-Zellensteuerungsdatei „nurbosch“ wählen
j) Mit ABLAUF/STARTEN wird das System initialisiert
k) Mit F3 wird der Ablauf dann gestartet

# 4 An der Montagezelle ausgeführte Arbeiten

## *4.1 Allgemein*

Die Anlage wie, sie bisher beschrieben wurde, ist bereits aufgebaut und war auch schon in Betrieb. Es ist nun so, dass die Roboterprogramme nicht mehr auf die aktuellen Aufbauabmessungen abgestimmt sind. Also mussten die einzelnen Roboterprogramme neu „geteacht“ werden. Darüber hinaus mussten die Anschlusspläne erneuert und einige defekte elektrische Verbindungen repariert werden. Im folgenden Kapitel werden alle von mir ausgeführten Arbeiten näher beschrieben und erklärt.

## *4.2 Ersatz des Eingabeterminals des PUMA 560 durch einen Steuerrechner*

### 4.2.1 Aufgabenstellung:

Das Terminal des Industrieroboters Puma 560 von Unimation® soll durch einen Standard IBM-Computer ersetzt werden.
Auszuführende Details:
Möglichkeit der Bildschirm-Steuerung (Monitor/Tastatur)
Softwaremäßiger Ersatz des vorhandenen Disketten-Laufwerks (5 ¼“)
Laden der VAL 2-Programmiersoftware in das EPROM des Steuerrechners
Laden der Initialisierung (Calibration-Overlay)
Speichern und Laden von Programmen, Variablen und Koordinatenpunkten

Das Initialisieren und das Laden von VAL-$^{TM}$ ist von besonderer Bedeutung. Der im Steuerrechner vorhandene Speicherbaustein ist ein sog. EPROM und verliert seinen Inhalt, sofern dies nicht rechtzeitig durch das Einschalten des Rechners verhindert wird. Nun ist dieser Roboter nur während der Labor-Übungen im Betrieb. Das heißt, dass jedes Mal vor Inbetriebnahme der Inhalt der Betriebssystem-Diskette in das EPROM geladen werden muss, bevor der Roboter einsetzbar ist. Das entscheidende Problem ist allerdings die begrenzte Lebensdauer dieser Diskette, sowie der dazugehörigen Hardware, welche es erforderlich macht, eine Sicherung auf einem moderneren Speichermedium vorzunehmen.

### 4.2.2 Roboter-Aufbau

Die folgende Beschreibung dient lediglich zum besseren Verständnis des Robotersystems. Um detaillierte Informationen zu erhalten, sollte das Benutzerhandbuch herangezogen werden.

Der Industrieroboter Puma 560 ist ein computergesteuertes Handhabungsgerät, das von UNIMATION® hergestellt wurde. Die Haupteinheit setzt sich aus Handhabungsgerät und Schaltschrank zusammen.

#### 4.2.2.1 Mechanische Komponenten

Der Gelenkarm ist das mechanische Bauteil und hat 6 Bewegungsachsen, welche über je einen Dauermagnet-Gleichstromservomotor gesteuert werden. Der Befestigungsflansch des Gelenks ist für die Aufnahme verschiedenster Werkzeuge geeignet. Im vorliegenden Fall ist dies ein pneumatischer Greifer, der als Sonderausstattung zum Gerät erhältlich ist. Er kann auch direkt vom Handbediengerät aus gesteuert werden.

#### 4.2.2.2 Schaltschrank, Handbediengerät

Der Schaltschrank ist ein 19“ Normgestell mit verschließbarer Glastür. Er beinhaltet Netzteil, Steuermodul, Schalttafel, Verstärkermodul, I/O-Modul.

#### 4.2.2.3 Terminal

Das Terminal dient dem Benutzer als Kommunikations-Schnittstelle zur Steuerung. Die Eingabe und Ausgabe erfolgt über VAL™-Befehle. In dem vorhandenen Terminal ist darüber hinaus noch eine Disketten-Station für 5 ¼“-Disketten integriert. Im Benutzer-Handbuch ist diese Ausführung des Terminals nicht aufgeführt. Zwar gibt es darin Informationen zum VDU-Terminal sowie zur Disketten-Station, jedoch als getrennt ausgeführte Elemente. Deshalb stimmen die Angaben, insbesondere die Anbindungskenngrößen (elektrische Eigenschaften, und Übertragungsprotokolle) nicht mit den Gegebenheiten überein.

#### 4.2.2.4 Software

Das System Puma™ arbeitet mit der Programmiersprache VAL™, mit der das gesamte Robotersteuerungssystem für den Benutzer zugänglich wird. Die Programmierung des Industrieroboters erfolgt durch einfache Eingabe über die manuelle Steuerung oder die Tastatur.
Dabei werden die eingegebenen Daten als Punkte in einem Koordinatensystem gespeichert, das auf den feststehenden Robotersockel bezogen ist, bzw. der eingestellten Kalibrierung. entspricht
Die Sicherung der Daten bzw. der Programme kann mittels Floppy-Diskette ebenfalls über VAL™-Befehle vorgenommen werden.

#### 4.2.2.5 Anbindungsmöglichkeiten

Das Handhabungssystem bietet die Möglichkeit der Anbindung an ein externes Überwachungssystem durch eine eigens dafür zur Verfügung gestellte Schnittstelle (RS-423, NETWORK). Diese wird mittels VAL-2-Protokoll vom Prozessor aus gesteuert. Diese Schnittstelle bietet die Möglichkeit einen PC direkt anzubinden um z.B. Befehle ein/auszugeben oder Programme in das System zu laden (ähnlich wie Disketten-Station).
Diese Anbindung gibt dem Benutzer die Möglichkeiten das Potential der Steuerung, insbesondere hinsichtlich der Größe des Speichers zu erhöhen. Für diese Kommunikation gibt es ein strenges Protokoll, das im Wesentlichen wie ein Dialog zwischen Überwachungssystem und Steuerung aufgebaut ist.
Voraussetzung für die Verwendung ist jedoch, dass VAL™ im EPROM geladen ist, da die gesamte Kommunikation nur mittels VAL2-Befehle ermöglicht wird. Die in Punkt 4.21 erwähnte Aufgabenstellung das VAL2 von externer Seite zu laden, ist auf diese Art und Weise also nicht realisierbar.

#### 4.2.2.6 Schnittstellen zur Peripherie

Der Schaltschrank ist zentrale Einheit des Gesamtsystems. Dort laufen sämtliche Informationen (Steuersignale) zusammen.
Zum Gelenkarm gibt es zwei Kabelverbindungen (Panzerkabel), die zum einen die Ansteuerung der Servomotore (inkl. Bremsen) übernehmen und zum anderen die Informationen vom Gelenkarm zur Steuerung senden (Inkremental-Positionsgeber, Potentiometer etc.).
Des weiteren gibt es drei Anschlüsse mit ähnlichen Übertragungseigenschaften. Es sind dies serielle Schnittstellen zu:

- Handbediengerät
- Terminal
- Diskettenstation

Alle Schnittstellen sind als Vollduplex-Verbindung nach RS-232 ohne Steuerleitung aufgebaut. Das heißt, es gibt immer eine Masse-Leitung und zwei Signalleitungen, die mit 12V-Spannungs-Signalen arbeiten. Wie sich jedoch herausgestellt hat, ist zwar die Versorgungsspannung mit 12V gegeben, die eigentliche Signalspannung beträgt jedoch nur mehr 5,8V. Da diese Spannung innerhalb der für RS-232 normierten Spannungspegel von 3,5-15 V liegt, stellt dies kein weiteres Problem dar. Für den Fall, dass längere Leitungen eingesetzt werden sollen, wäre ein Anheben des Spannungspegels erforderlich. Der anzuschließende PC ist jedoch in unmittelbarer Nähe unterzubringen, da beim Programmieren mittels Handbediengerät eine größere Entfernung zum PC ohnehin nur hinderlich wäre.

### 4.2.3 Anbindung an externen PC

Wie nun schon angesprochen, ist die Anbindung über die NETWORK-Schnittstelle für diese Aufgabe nicht zweckmäßig. Um nun an die Informationen, die zwischen Terminal und Schaltschrank ausgetauscht werden, zu kommen,

wurden die beiden Schnittstellen des VDU-Terminals, sowie der Disketten-Station zur Analyse herangezogen.

### 4.2.3.1 Steckerbelegung

Die beiden Schnittstellen sind als 10-Pin-CANNON-Stecker ausgeführt. Zu der PIN-Belegung war ein techn. Skript vorhanden, das jedoch mit der vorhandenen Verbindung nicht übereinstimmte, da die Schnittstelle aufgrund des anderen Terminal-Typs anders konfiguriert war.
Durch Öffnen der Steckerbuchse konnte die PIN-Belegung wie folgt festgestellt werden.

PIN 5 MASSE
PIN 6 vom Schaltschrank gesendete Signale
PIN 7 vom Terminal gesendete Signale

Die PIN-Belegung ist beim Terminal, sowie bei der Diskettenstation identisch. Bei der Terminal-Leitung sind zusätzlich noch PIN 8 u. 10 belegt. Diese dienen nicht zur Signal-Übertragung sondern lediglich als Spannungsversorgung (5V-Spannung) für den internen Chip-Satz. Bei der Anbindung an den PC werden diese beiden Leitungen deshalb nicht berücksichtigt.

### 4.2.3.2 Übertragungsprotokoll

Die Übertragungsprotokolle der Schnittstellen waren ebenfalls laut Benutzerhandbuch gegeben. Wie bereits bei der Beschreibung des Terminals erwähnt, stimmen die Übertragungsprotokolle nicht mit den im Handbuch angegebenen überein. Nach einer Analyse der Übertragungssignale mittels Oszilloskop konnte die Übertragungsrate aus den Signallaufzeiten mit 9600 baud bestimmt werden. Beim Terminal wurden im Handbuch 7 Datenbits, ungerade Partität und 1 Stop-bit als Übertragungsparameter angegeben. Mit dem Startbit

werden also pro Zeichen 10 bit gesendet. Bei softwaremäßiger Überwachung der Parität wurde jedoch festgestellt, dass die Übertragung meist fehlerhaft war (d.h. Paritätsfehler), obwohl stets die richtigen Zeichen übertragen wurden. Also waren die Übertragungsparameter falsch festgelegt.
Daraufhin wurden die Parameter wie folgt eingestellt:

- 1 Startbit
- 8 bit Zeichen
- keine Paritätsprüfung
- 1 Stop-bit

Mit dieser Einstellung war nun eine fehlerfreie Übertragung möglich. Beim Diskettenlaufwerk wurden diese Einstellungen übernommen.

### 4.2.4 Serielle Schnittstelle

Zur Ansteuerung der Peripherie-Geräte wird, wie schon erwähnt, eine serielle Datenübertragung durchgeführt. Diese funktioniert zum Glück mit den Standardeinstellungen der RS-232-Schnittstelle, welche jeder IBM-kompatible PC besitzt.

Da die Signalpegel mit denen der PC-Schnittstellen kompatibel sind, ist auch keine externe Signalaufbereitung erforderlich. Die Kabel können also direkt verbunden werden.
Zur Datenverarbeitung wurde die Möglichkeit, Assembler-Routinen in C++ einzubinden, genutzt. Das Auslesen der Daten aus dem Empfangsregister bzw. das Schreiben ins Sendehalteregister der PC-Schnittstelle geht sehr viel schneller von statten als über vergleichbare Programm-Befehle, die meistens mittels BIOS-Interrupt 14h arbeiten. Die Initialisierung der Schnittstellen, die jeweils nur am Beginn des C-Programms durchgeführt werden muss, ist softwaremäßig einfacher mit dem BIOS-Interrupt 00h zu bewerkstelligen.

Vermutlich hätte auch der BIOS-Interrupt für diese Aufgaben herangezogen werden können, da aber am Beginn der Arbeit noch nichts Genaues über den

Datenaustausch und die Geschwindigkeit der Auswertung bekannt war, war dies eine Möglichkeit, Fehler bereits im Vorfeld zu vermeiden.

### 4.2.5 Ausführung der Monitor-Steuerung

#### 4.2.5.1 Prinzipielle Vorgehensweise

Die Ein/Ausgabe-Einheit funktioniert nach einem einfachen Prinzip.
Bei Eingabe eines Zeichens über die Tastatur wird dieses Zeichen vom Terminal zum LSI-11 gesendet. Dort wird dieses dann entsprechend bearbeitet. Der Mikrocomputer leitet dazu eine Aktion ein, die dafür sorgt, dass das eben erhaltene Zeichen zum Monitor gesendet wird. D.h., die Eingabe eines Zeichens am Keyboard bewirkt nicht unmittelbar die Ausgabe des Zeichens auf dem Monitor, genauso wie bei einem PC.
Dadurch beschränkt sich die Aufgabe des Computerprogramms, das diese Funktion übernehmen soll, auf die Überprüfung des Zustandes des Tastatur-Puffers, sowie des seriellen Ports.

#### 4.2.5.2 Ausgeführtes Programm

Die Abfrage des Tastatur-Puffers sowie des seriellen Ports erfolgt kontinuierlich. Sobald sich der Zustand geändert hat (Zeichen erhalten), wird die entsprechende Aktion eingeleitet. Das von der Tastatur eingelesene Zeichen wird über die serielle Schnittstelle an LSI-11 geschickt, bzw. das vom LSI-11 erhaltene Zeichen wird am Benutzer-Bildschirm dargestellt.
Diese kontinuierliche Abfrage wird auch noch mit dem zweiten seriellen Port ausgeführt, welches für die Kommunikation mit dem Modul zum Speichern und

Laden von Dateien zuständig ist. (Funktionsweise dieses Moduls wird später ausführlich erklärt).
(Der Programmteil wird im Anhang mit Erklärungen zum Code ausgeführt)

### 4.2.6 Ausführung der Diskettensteuerung

#### 4.2.6.1 Laden von VAL-2

Am Beginn des Kapitels wurde schon über die Wichtigkeit der Systemdiskette gesprochen. Das Laden der Programmiersprache ist im normalen Laborbetrieb dringend notwendig, da Laborübungen üblicherweise nur ein paar Mal im Jahr abgehalten werden. Da zwischen den einzelnen Übungen in der Regel mehr als ein Monat (30 Tage) vergehen, verliert der EPROM jedes Mal wieder seinen gesamten Inhalt und muss mit den Daten der Systemdiskette aktualisiert werden.

Beim ersten Versuch, an den Inhalt der Systemdaten zu gelangen, wurde die gesamte Kommunikation zwischen Disketten-Station und UNIMAT „ausgelesen“. Diese Daten wurden softwaremäßig erfasst und automatisch auf die Festplatte des PC kopiert. Damit die bidirektionale Verbindung mittels RS-232-Schnittstelle am PC erfasst werden kann ist die Anbindung an zwei Schnittstellen erforderlich. Eine dient für die Daten, welche von der Disketten-Station gesendet werden, die andere für die Daten, welche vom Mikrocontroller an das Diskettenlaufwerk geschickt werden.
Dabei ergibt sich eine Datenmenge von ca. 134 kByte. Dieser Wert ist, wie weitere Lese-Versuche gezeigt haben, nicht immer gleich und kann um bis zu +/- 1500 Byte variieren.
Für diese Unterschiede wurden anfangs die Übertragungsleitungen verantwortlich gemacht. Dass dem nicht so ist, wird im Kapitel 6.5 erläutert.
Abgesehen von der „ungenauen“ Datenmenge wurde auch festgestellt, dass das Übertragungsprotokoll nicht dem in den folgenden Kapiteln beschriebenen entspricht.

In einem späteren Versuch wurden die Datensätze der Datei VALII.560 direkt vom externen PC aus gesteuert und von der Disketten-Station ausgelesen. Das heißt, der PC hat in diesem Fall die Aufgabe des Mikrocontrollers übernommen. (Das dazu erforderliche C-Programm ist im Anhang aufgeführt).
Auch hier konnten nicht alle Datensätze gelesen werden, da die Anzahl der Bytes pro Datensatz variierten (sogar sehr stark, der 4. Datensatz hat nur 136 Zeichen). Darüber hinaus wird auch nicht immer der im Kapitel 6.3 angeführte Code 1-R-4 als Leseaufforderung gehandhabt, sondern oft nur durch ein ASCII-4 angegeben.
Dies liegt wohl darin, dass am Beginn des Ladens der Systemdateien der Mikrocontroller vermutlich noch nicht die Informationen über den Datenaustausch mit dem Disk-Laufwerk besitzt, und deshalb auch nicht das für die üblichen Lese-Routinen verwendete Protokoll heranzieht.

#### 4.2.6.2 Initialisierung

Grundsätzlich gibt es ein genau geregeltes Protokoll für den Zugriff auf das Diskettenlaufwerk. Näheres dazu im nächsten Kapitel.
Bei der Aufforderung an das Diskettenlaufwerk, die Datei SN5105.OVE zu laden, gibt das Diskettenlaufwerk 2 Datensätze mit jeweils 258 Zeichen (1. ASCII-2 für Start, 256 Daten-Charakter, 1 Abschlusszeichen für Kontrolle) aus.
(Das dazu verwendete C++-Programm ist im Anhang ausgeführt)
Das Abschlusszeichen wurde hier nicht ausgegeben. Außerdem hat sich gezeigt, dass beim "Abhören" der Leitung nicht 2 Datensätze, sondern nur einer gesendet wurde. Im zweiten Datensatz sind nur mehr 3 Zeichen angegeben:
1. ASCII 2
2. ASCII 6 (wenn 6 unmittelbar nach Leseaufforderung stehen würde, hieße das „END-OF-FILE“)
3. ASCII-Code 255
Dieser zweite Datensatz ist offensichtlich so etwas wie eine Signatur und für das Calibration-Overlay nicht von Bedeutung.

Es musste also beim Programm nur der 1. Datensatz an den Steuerrechner geschrieben werden. Der Abschluss mit ASCII-6 ist nicht erforderlich.
Damit ist die Initialisierung durchgeführt.

### 4.2.6.3 Speichern/Laden von Programmen

Wie bereits erwähnt, kann der Benutzer mit Hilfe der VAL-Befehle Roboter-Programme, Punkte (Locations) und im Programm aufgeführte Variable (REALS) speichern. Diese können separat gespeichert werden oder auch in eine einzige Datei. Dabei gibt es folgende Strukturierung:

.PROGRAMM *(Programmname)*
*(Programmcode)*
.END
.LOCATIONS
*(Punkt-Bezeichnung mit den dazu gehörigen Koordinaten)*
.END
.REALS
(Variablen)
.END

Das Übertragungsprotokoll zum Schreiben oder Lesen einer Datei ist streng festgelegt.

### 4.2.6.4 Schreiben einer Datei

Zum Schreiben einer Datei auf Diskette gehören drei getrennte Vorgänge:

1. Erstellen der Datei
2. Schreiben der Datei in formatierten Sätzen
3. Schließen der Datei

Zu 1.)

Nach dem vom Benutzer mittels Tastatur eingegebenen Befehl STORE sendet der Mikrocomputer LSI-11 folgende Zeichenfolge:

ASCII 1 - ASCII 69 – Dateiname - ASCII 4

☺ E file.v2 ♦

Wenn das Diskettenlaufwerk diese Zeichenfolge erhalten hat, wird eine Datei mit dem Namen angelegt. Dabei wird nicht überprüft, ob es diese Datei bereits gibt. Falls diese Datei existiert, wird sie durch das neue Programm ohne Warnung überschrieben.

Wenn diese Datei angelegt wurde, sendet das Diskettenlaufwerk ein Status-Zeichen zurück. ASCII 16 bedeutet fehlerfreier Vorgang.

Zu 2.)

Nach der Definition der Datei werden die Datensätze an das Diskettenlaufwerk geschickt.

Das Protokoll wird wie folgt ausgeführt:

| LSI-11 | Diskettenantwort |
|---|---|
| 1.) | |
| ASCII 1 – ASCII 87 – ASCII 4 | Status-Code |
| 2.) | |
| ASCII 2 – Datensatz – Prüfsumme | Status-Code |

Diese Datensätze beinhalten 256 Zeichen. Der ASCII kennzeichnet den Beginn der Übertragung. Nach dem der Datensatz geschrieben wurde, folgt noch ein Prüfsummenzeichen. Die Prüfsumme wird durch Aufsummieren der Komplementär-Werte der vorangegangenen 256 Zeichen gebildet. (Die Routine zur Überprüfung diese Summe ist im Anhang aufgeführt).

Die Status-Codes sind dem Benutzerhandbuch zu entnehmen [].

Wenn nun alle Datensätze gesendet wurden, wird die Datei durch LSI-11 mit folgendem Code geschlossen:

| LSI-11 | Diskettenanwort |
|---|---|
| ASCII 1 – ASCII 67 – ASCII 4 | Status-Code |
| ☺ C ♦ | |

### 4.2.6.5 Lesen einer Datei von der Diskette

Dieser Vorgang ist dem des Schreibens einer Datei sehr ähnlich.

1. Öffnen der Datei zum Lesen
2. Schreiben der Datei in formatierten Sätzen

zu 1.)
Das Öffnen der Datei erfolgt mit der gleichen Zeichenfolge wie beim Schreiben. Nur wird hier statt ASCII 69, ASCII 79 gesendet. Damit weiß das Disk-Laufwerk, dass es eine bereits bestehende Datei öffnen muss. Dementsprechend kann nun auch als Code-Antwort ASCII 3 gegeben werden (d.h. Datei nicht vorhanden).

zu 2.)
Die Aufforderung an die Diskette einen Datensatz zu übersenden wird wie folgt ausgeführt:

| LSI-11 | Diskettenantwort |
|---|---|
| ASCII 1 – ASCII 82 – ASCII 4 | Status-Code |

Unmittelbar danach beginnt das Diskettenlaufwerk mit dem Senden der Daten. Dies geschieht mit derselben Signatur wie beim Schreiben.
Diskette:
ASCII 2 – Datensatz – Prüfsumme

Hierbei wird nun die erfolgreiche Datenübertragung vom LSI-11 überprüft.

Ist der letzte Datensatz übertragen worden, wird nach erneuter Aufforderung von LSI-11 zum Lesen eines Datensatzes der Status-Code 6 zurückgegeben. Der Mikro-Computer weiß nun, dass das Ende der Datei erreicht wurde.

Zur erfolgreichen Datenübertragung ist es erforderlich, dass alle Zeichen gesendet werden und das Prüfsummenzeichen richtig ist.
Ansonsten werden Code 12 (Satzprüffehler) bzw. CODE 9 (Kommunikationsfehler) angegeben.
Dies gilt sowohl beim Schreiben als auch beim Lesen einer Datei.

#### 4.2.6.6 Ausführung des Programmteils zum Laden/Speichern von Programmen

Wie bereits erwähnt, wird das Einlesen des zweiten seriellen Ports zur Überwachung der Kommunikation mit dem Speicher-Lade-Modul kontinuierlich ausgeführt.
Wird nun ASCII 1 eingelesen, wird dieser Prüfvorgang unterbrochen und die nachfolgenden Zeichen eingelesen.

| | |
|---|---|
| ☺E Dateiname | Öffnen der Datei zum Schreiben einer Datei |
| ☺O Dateiname | Öffnen der Datei zum Lesen einer Datei |
| ☺W♦ | Aufforderung zum Schreiben eines Datensatzes |
| ☺R♦ | Aufforderung zum Lesen eines Datensatzes |
| ☺C♦ | Schließen einer Datei |

Diese Zeichenabfrage findet im Hauptprogramm statt. Wenn die Zeichenfolge identifiziert ist, wird das entsprechende Unterprogramm aufgerufen.
Die Ausführung dieser Programmteile befindet sich im Anhang.

### 4.2.7 Fehler bei der Kommunikation mit dem Diskettenlaufwerk

Im Zuge der Arbeit traten vermehrt Probleme mit der Übertragung auf. Die ersten lauffähigen Programme waren noch nicht abgesichert. Der Status-Code, der vom LSI gesendet wurde, zuerst ignoriert.
Nachdem eine Überwachung des Statuscodes und der Anzahl der übertragenen Zeichen eingebaut wurde, konnte die Übertragung nicht mehr einwandfrei abgewickelt werden.
Wie sich gezeigt hat, fehlen beim Übertragen des Zeichensatzes einzelne Zeichen. Diese Fehlstellen treten meist unmittelbar nach dem Zeilenvorschub auf.
Nachfolgend ist ein Datensatz angeführt, wie er beim Speichern eines Programms übertragen wird.
*Zum Programm:*
*Dieses Programm mit dem Namen „test“ hat die Aufgabe drei Punkte (p1, p2, p3) nacheinander anzufahren. Es wurden keine Punkte definiert und auch keine Variablen festgelegt. Das Programm wurde ohne Kontrolle der Datensatzgröße gespeichert. Bei dem C-Programm, mit dem diese Speicherung erfolgt ist, wurde einfach die Abfrage der Länge des Datensatzes übersprungen und der Datensatz direkt gespeichert.*

```
.PROGRAM test
MOVE p1
OVE p2
MOVE p3
STOP
.END
.LOCATIONS
.END
.REALS
.END
```

Wie man sieht, fehlt in der dritten Zeile das “M” beim “MOVE”-Befehl. Dieser Fehler tritt nicht systematisch auf, ist aber stets unmittelbar nach einem Zeilenvorschub zu finden.

Zunächst wurde vermutet, dass dieser Fehler auf eine schlecht geschirmte Datenleitung zurückzuführen ist.

Nachdem dieses Problem festgestellt wurde, wurde das Speichern des gleichen Programms noch einmal mit Hilfe des Original-Terminals durchgeführt.
Die Aufforderung zum Speichern mit dem STORE-Befehl wurde mit der Meldung *DATA CHECK ERROR* quittiert.
Damit konnte eine fehlerhafte Übertragung zum Steuerrechner ausgeschlossen werden.

### 4.2.8 Fehleranalyse

Der Mikrocontroller des UNIMAT besteht aus verschiedenen Modulen.
Kernstück des UNIMAT sind die Interfaceplatinen A und B. Interfaceplatine A ist für die Programmiersprache und die Interpretation der eingegebenen Befehle verantwortlich. Die Interfaceplatine B setzt die Interpretation in Steuersignale um, welche dann von der Leistungselektronik in die für die Servomotore erforderliche Spannung umgesetzt werden. In umgekehrter Reihenfolge funktioniert auch die Auswertung der Positionsdaten, welche vom Roboterarm zum UNIMAT übertragen werden.
Die mit der Interfaceplatine A kommunizierende Schnittstellenkarte ist eine serielle Vierfach-Schnittstelle. Diese übernimmt die Kommunikation mit den Peripheriegeräten.

- Handbediengerät
- Terminal (Benutzerein- u. –ausgabe)
- Diskettenstation
- Spezielle NETWORK-Karte

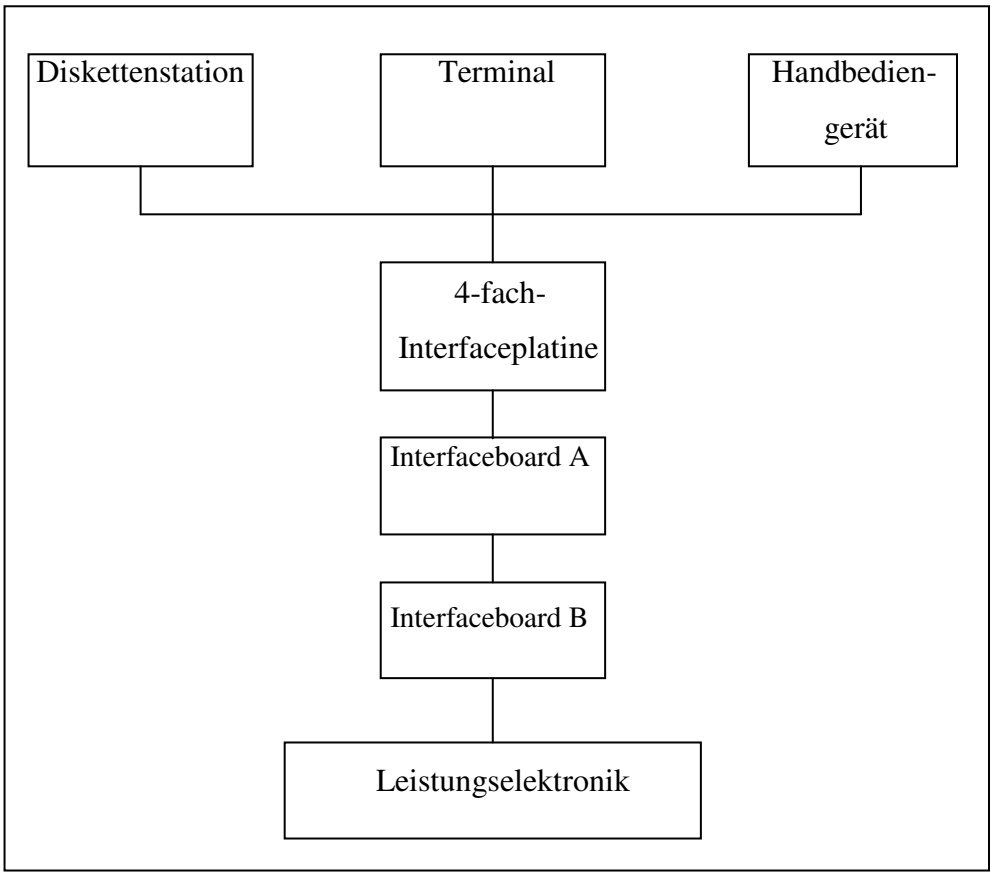



**Abbildung 6: Hardware-Struktur der Puma-Steuerung**

Aufgrund der Tatsache, dass der Fehler zwar nur am Zeilenanfang auftritt, also stets nach einem ASCII 13 (ENTER), nicht jedoch in einer anderen Art und Weise systematisch, d.h. nach einem Datensatztyp (PROGRAMM, LOCATIONS, REALS) auftritt, lässt sich der Fehler wohl auf diese serielle Vierfach-Interface-Schnittstelle zurückführen.
Dies ist lediglich eine Vermutung, jedoch scheint dies die einzig plausible Erklärung für das Auftreten des Fehlers zu sein.

### 4.2.9 Möglichkeiten den PUMA 560 für Labor-Übungen zu nutzen

Der Puma 560 ist mechanisch gesehen noch einwandfrei zu verwenden. Damit kann sowohl mit dem Original-Terminal als auch mit dem neuen externen Steuerrechner kommuniziert werden.

Lediglich das Speichern von Daten und Programmen auf Diskette ist nicht möglich. Daher können auch im Labor keine Protokoll-Daten gesichert werden, und müssten separat mitgeschrieben werden. Zusätzlich bestünde die Möglichkeit, die fehlerhafte Kommunikation mittels C-Programm zu umgehen. Dann könnten die Daten als *.V2-Datei gesichert und in einem TEXT-Editor angezeigt werden.
Wenn der Roboter längere Zeit nicht mehr in Betrieb gewesen ist, ist es erforderlich die Systemdiskette zu laden. Wie bereits erwähnt wurde, lässt sich dies nicht vom Steuerrechner aus bewerkstelligen, da das Datenprotokoll dieser Betriebsdiskette nicht entschlüsselt werden konnte. Darüber hinaus werden bei der Übertragung stets Fehler gemacht, welche das LSI in der Regel ignoriert, bzw. haben diese Fehler bislang keine Auswirkungen auf die Verwendung des Systems gezeigt.

## *4.3 Servo-Greifer*

### 4.3.1 Allgemein

Wie bereits in der Beschreibung der Fertigungszelle erwähnt, besitzt der YASKAWA-MOTOMAN einen 2-Finger Parallel-Servogreifer. Dieser wurde am Institut für Handhabungsgeräte und Robotertechnik der TU-Wien im Jahr 1992 gebaut und in den darauf folgenden Jahren weiterentwickelt.

Das System ist modular aufgebaut und kann durch entsprechende Anordnung der Module als Ein-, Zwei oder Dreifinger-Greifer zusammengestellt werden. Die Greifvorgänge werden auf einem PC, welcher mit einer Positionsregelkarte ausgestattet ist, eingelernt. Auf diese Weise können auch verschiedene Greifvorgänge gespeichert werden. Die dazugehörige Software *servo_ct* wird im MS-DOS-Modus geöffnet, und ist mit einer menü- und maskengesteuerten Oberfläche ausgestattet, welche relativ bedienerfreundlich ist.

Der Vorteil des Servogreifers ist, dass verschiedene Teile (Teile mit unterschiedlichen Greifabmessungen) gehandhabt werden können. Darüber hinaus kann auch die Greifkraft per Steuerungssoftware eingestellt werden. Dadurch können auch besonders empfindliche Teile optimal gegriffen werden.

### 4.3.2 Funktionsweise

Das Servo-Greifer-System besteht im Wesentlichen aus folgenden Komponenten:

a) Parallelgreifer
Dieser besteht aus zwei getrennten Spindelführungen, welche je eine Greifbacke tragen. Im folgenden Bild ist eine dieser Spindelführungen dargestellt.

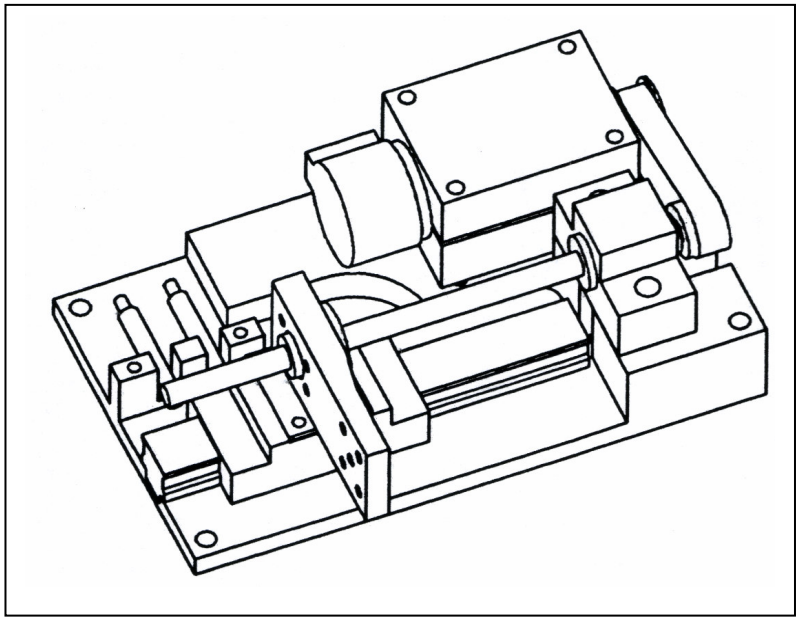


**Abbildung 7: Servogreifermechanik**

b) Schaltkasten

Dieser dient zur Umsetzung der vom Greifer kommenden Signale (Position und Kraft) in zur Weiterverarbeitung mit der Positionsregelkarte erforderliche Signale. Weiters werden die Steuersignale von der Positionsregelkarte in einem speziellen Leistungsteil auf das für die Ansteuerung der Servomotore erforderliche Leistungsniveau gehoben. Dies wird mittels einer H-Brücken-Schaltung realisiert.

Über Breitbandkabel können Signale zur Steuerung der Greifvorgänge herangezogen werden. Diese Eingangs/Ausgangs-Schnittstelle hat eine ähnliche Funktionsweise wie eine I/0-Karte.
Im gegebenen Fall wird diese Schnittstelle vom Steuerrechner, der die Gesamtanlage steuert und überwacht, belegt. Diese Schnittstelle könnte aber auch für die Anbindung an einen Industrieroboter herangezogen werden.

c) Positions-Regelkarte

Als Schnittstelle zum PC wird eine speziell für solche Aufgaben konzipierte Regelkarte vom Typ DMC 630 von GALLI verwendet. Diese ist mittels

direkter Programmierung mit C++ ansteuerbar. Detaillierte Unterlagen zu dieser Karte sind nicht mehr verfügbar.

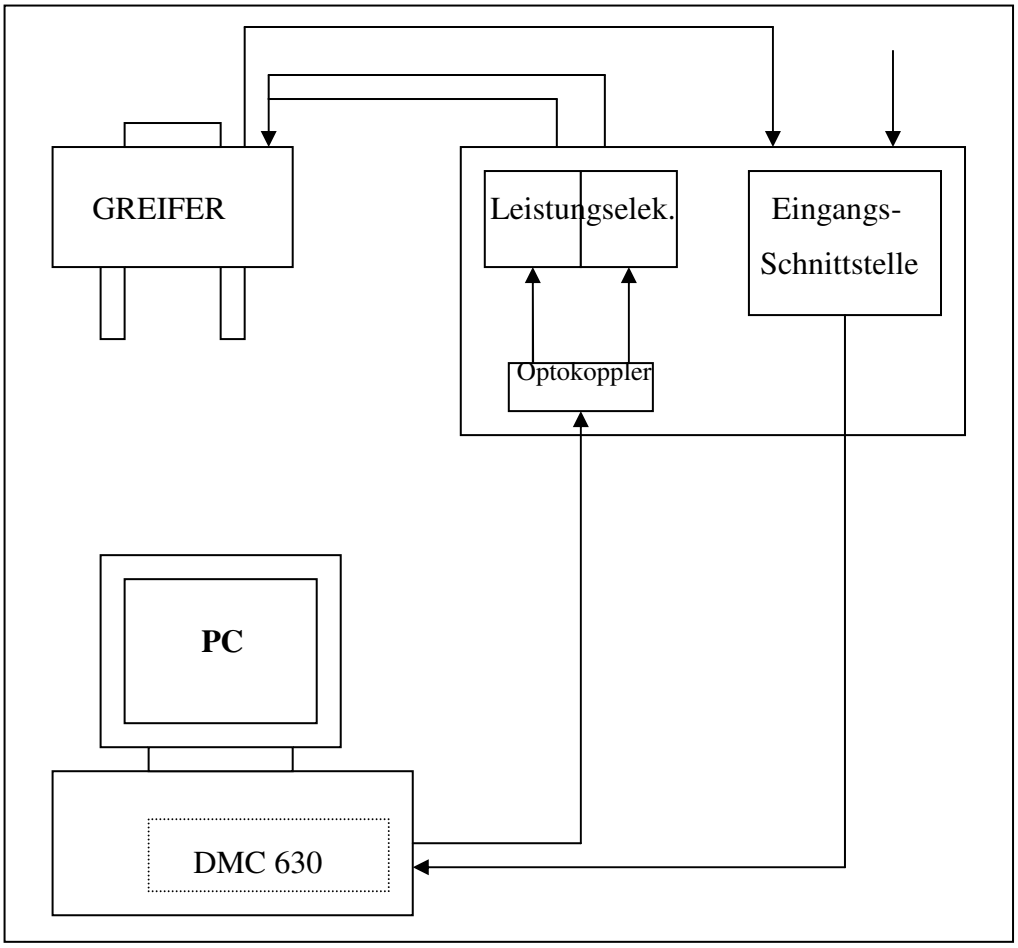



**Abbildung 8: System Servogreifer**

### 4.3.3 Versuch der Instandsetzung und TEACHEN des Greifers

#### 4.3.3.1 Probleme der mechanischen Komponenten

Vor der ersten Inbetriebnahme mussten zuerst die Anschlüsse an einem Servomotor erneuert werden, da diese möglicherweise durch unsachgemäße Behandlung des Greifer bzw. durch die nicht vorhandene Kabelzugentlastung beim Bewegen des Roboter-Arms abgerissen wurden.

Beim ersten Versuch den Servogreifer in Betrieb zu nehmen, wurde ein mechanischer Defekt festgestellt. Bei jeder Umdrehung der Spindel gab es eine Stelle, die nur mit stark erhöhtem Drehmoment zu überwinden war.
Es stellte sich heraus, dass die Kugelspindeln offensichtlich nicht richtig arbeiten konnten, da bereits einige Kugeln aus der Führungsmutter herausgesprungen waren. Dies wurde wohl durch eine geringe Deformation der Spindel, welche möglicherweise auf eine Kollision des Greifers beim Bewegen der Roboterachsen zurückzuführen ist, hervorgerufen.
Es ist auch denkbar, dass dieser Defekt durch einen Bedienungsfehler des Greifers, bei dem die Endlagen überschritten wurden, hervorgerufen wurde.

Das Tauschen der Führungsmutter durch eine andere stellte sich als einzige Möglichkeit heraus, da das Nachfüllen mit Kugeln nur mittels spezieller Vorrichtungen zu bewerkstelligen ist. Selbst das Tauschen der Mutter ist nicht unproblematisch, da die Kugeln unter Spannung stehen und dadurch schon bei geringen Axialkräften auf die Spindelmutter herausspringen. Üblicherweise wird die Gesamteinheit (Spindel und Spindelmutter) getauscht. Diese werden bereits ab Werk zusammengebaut.

Durch diesen Umbau konnte wieder eine kontinuierliche Bewegung realisiert werden, allerdings weist diese Spindel nun ein leicht erhöhtes Drehmoment (aufgrund von Reibungen) auf.

#### 4.3.3.2 Probleme beim TEACHEN von Greifbewegungen

Nachdem dieses mechanische Problem behoben war, wurde mit Hilfe von servo_ct ein Homing-Vorgang durchgeführt.
Anschließend konnte der Greifer vermessen werden. Die Homingposition dient dem Steuerrechner als Referenz-Punkt, von dem aus alle Weg- und Positionsgrößen berechnet werden. Der Abstand der Greifbacken wurde über das

Menü OPTIONEN/GREIFER GRUNDEINSTELLUNGEN mit 62.2 mm eingegeben.

Beim TEACH-Vorgang wird eine Maske geöffnet, die alle Befehle beinhaltet, die zum Speichern von Greif-Vorgängen erforderlich sind. Es können hier nun die Greiferbacken einzeln mittels Curser-Tasten verfahren werden. Damit werden die Greiferbacken an das zu greifende Teil herangefahren. Mit dem Befehl GREIFEN UND POSITION SPEICHERN F6 wird nun die in der vorangegangenen Maske gewählte Backe in Richtung Führungsbacke bewegt. Wenn die Backen das Werkstück gegriffen haben, wird die Greifkraft bis zur vorher eingestellten Kraft erhöht.
Mit F8 wäre es nun möglich, beide Backen wieder zu öffnen.

Nun tritt hier das Problem auf, dass sich die Backe 1 nicht bewegen lässt.
Würde man bei der vorher angesprochenen Eingabe-Maske die andere Backe als Führungsbacke angeben, würde sich diese schon beim GREIFEN-Befehl nicht bewegen, der eigentliche TEACH-Vorgang ist so also nicht möglich.
Wie sich herausstellte, konnte diese defekte Backe händisch weitergedreht werden, und zwar bis zur voll geöffneten Position. Man bemerkt dabei, dass der Motor versucht, in die richtige Richtung zu drehen, jedoch fast kein Drehmoment in diese Richtung aufbringt.

Wenn der Motor von Hand weitergedreht wird, leuchtet eine rote LED im Schaltkasten auf. Es konnte allerdings für diese offensichtlich als Fehler-Meldung gedachte Anzeige keinerlei Information gefunden werden.

Während der Behandlung dieses Problems kam die Vermutung auf, dass dieses „nicht-Öffnen“ möglicherweise beabsichtigt ist, und das Lösen des Drehmoments nur dazu dient, beim Öffnen der anderen Backe die Greifkraft auf Null zu setzen.
Dass dem nicht so ist, bemerkt man, sobald man versucht, eine erneute Bewegung zu vollziehen (z.B. Homing, Öffnen max., Bewegen der einzelnen Backen etc.).

Dann wird nämlich die Fehlermeldung „Letzte Bewegung noch nicht abgeschlossen“ ausgegeben.

#### 4.3.3.3 Analyse des Fehlers

Zuerst wurde ein Fehler bei der Software vermutet, da das Drehmoment ganz offensichtlich auf Null heruntergesetzt wurde.
Im Sourcecode wird bei jener Routine, welche für das Öffnen der Greiferbacke verantwortlich ist, die Einstellung der Greifkraft mit dem Parameter CP_MIN angegeben. Diese ist allerdings im Sourcecode (DEF.H) mit 15 angegeben, also 15% der Maximal-Kraft.
Zum Vergleich: der oben erläuterte Versuch beim TEACHEN der Greifposition wurde mit 5% erfolgreich ausgeführt.

Da die Routinen des Source-Codes für beide Greiferbacken verwendet werden, ist also ein Software-Fehler auszuschließen.
Außerdem wurden die Source-Codes seit 1997 nicht mehr geändert, allerdings noch im Jahr 2002, bei der letzten Programmierung der gesamten Fertigungszelle erfolgreich eingesetzt.

Da nun offensichtlich ein Software-Fehler gänzlich auszuschließen ist, muss der Fehler bei der Hardware liegen.

Hierzu wurden sämtliche Leitungen, die von oder zum Steuerkasten gehen, überprüft und dabei kein Defekt festgestellt.
Also wurde nun der Schaltkasten genauer analysiert.
Dieser besteht, wie am Beginn dieses Kapitels erwähnt, aus einer Leistungseinheit und einer Einheit, welche für die Signalaufbereitung für die Regelkarte verantwortlich ist. Da bei letzterer aufgrund der fehlenden Informationen bezüglich der verwendeten Chips (Gatter) der Fehler nicht lokalisierbar ist, wurde der Fehler vorerst bei der Leistungsseite gesucht.

Die Ansteuerung der Servomotore erfolgt mit einem pulsbreitenmodulierten Signal, welche eine H-Brücke speist.
Mittels Oszilloskop wurde die die Ausgangsspannung der H-Brücke, die Eingänge zur H-Brücke und die Eingänge der vorgeschalteten Optokopler analysiert.
Hier konnte kein Fehler gefunden werden. Das heißt, dass im Bereich außerhalb der Regelkarte das Signal richtig übertragen wird. Aufgrund dieser Tatsache kann der Fehler nur mehr an der DMC-630 Regelkarte liegen.
Da hierfür keinerlei Unterlagen zur Verfügung stehen, kann dieser Teil nicht näher analysiert und der Fehler somit nicht behoben werden.

### 4.3.4 Fazit

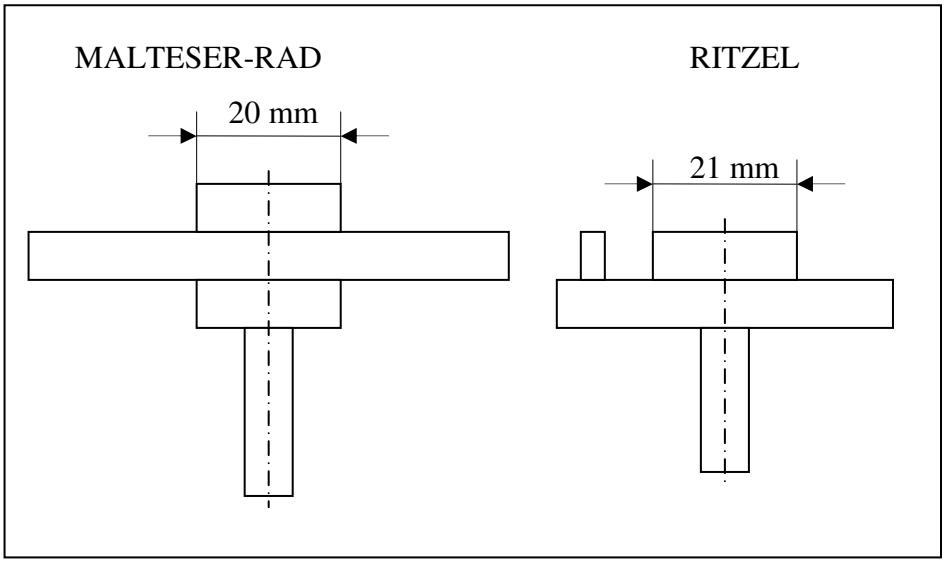
Das Servo-Greifersystem welches am MOTOMAN angebracht ist, ist von der mechanischen Konstruktion gesehen sehr problematisch, da die Führungselemente schon bei geringen Maßabweichungen bei der Fertigung nicht mehr fehlerfrei arbeiten können. Darüber hinaus reagieren sie sehr empfindlich auf mechanische Krafteinwirkung und Verschmutzungen.

Hier wäre eine geschlossene Antriebseinheit, und entsprechende mechanische Überlastsicherungen vorzusehen.
Gerade im Labor-Betrieb, wo ungeübte Studenten dieses Gerät handhaben müssen, können sehr leicht Teile kaputt gehen. Vermutlich hat auch aus diesem Grund das Gerät nun einen Schaden, der nur durch das Tauschen der gesamten Steuereinheit (Hard- und damit auch Software) zu beheben ist.

### 4.3.5 Alternativen zu Servogreifer

Bei der Beschreibung der Fertigungszelle ist erwähnt worden, dass der MOTOMAN mit seinem Greifer ein Malteser-Rad und Ritzel greifen muss. Diese beiden Teile sind nicht besonders empfindlich in der Handhabung.

Darüber hinaus können sie an Stellen gegriffen werden, die vom Durchmesser nur geringfügig abweichen.




Für diesen Verwendungszweck wäre es also auch denkbar, einen pneumatischen Parallel-Greifer einzusetzen. Solche Greifer sind auch für größere Toleranzen noch gut geeignet, da die Greifkraft ja über dem gesamten Hub konstant bleibt.

## 4.4 Neu TEACHEN des Yaskawa

Aufgrund der im vorherigen Kapitel aufgezeigten Probleme ist eine Neuprogrammierung des Yaskawa-Roboter nicht nur sinnlos, sondern nicht einmal möglich. Zum Teachen der Roboter-Koordinaten ist ein simultanes Programmieren des Greifers erforderlich, da beide Koordinatensysteme voneinander abhängen.

Für den Fall, dass der Greifer ersetzt werden sollte, befindet sich im Anhang ein Programm, das die prinzipielle Vorgehensweise bei einer Neuprogrammierung aufzeigt. Programmteile könnten direkt kopiert werden.

Es wäre darüber hinaus auch möglich, den Ablauf etwas zu vereinfachen. Früher musste der Steuerrechner für jede Teilbewegung zwischen den einzelnen Greifoperationen (öffnen/schließen) die jeweils nächste Programmnummer zur Yasnak-Steuerung senden. Wenn man die Ansteuerung des pneumatischen Greifers nicht dem Steuerrechner zuordnet, sondern dem Yaskawa, wäre künftig nur mehr ein Programmaufruf erforderlich. Nachteil dabei ist nur, dass man keine detaillierten feedback-Informationen mehr erhält, da ja nur mehr ein Programmzyklus bestätigt werden muss. Damit erhält der Steuerrechner auch keinerlei Kontroll-Informationen über den Greiferstatus.

### 4.4.1 Ablaufprogramm von C_SEQU für Yaskawa

Folgend wird nun jenes Programm dargestellt, welches in C_SEQU erstellt wurde, um den YASKAWA-Roboter mit dem Servogreifersystem zu betreiben. Da nun, wie bereits erwähnt, dieses nicht mehr einsatzfähig ist, kann dieses Programm nicht mehr verwendet werden. Sollte, wie schon angesprochen, der Yaskawa mit einem pneumatischen Greifer bestückt werden, könnte man dieses Programm kürzen. Dann müsste nur mehr eine Programmnummer gesendet werden, die den gesamten Ablauf startet.

Hier wird zuerst überprüft, ob bereits ein Teil auf der Palette montiert ist. Wenn diese Prüfung positiv ausfällt, müsste die Palette durchgeschleust werden, da der vom Yaskawa ausgeführte Arbeitsschritt am Beginn des Zyklus steht, und daher nur eine leere Palette den Programmstart auslösen darf.

```
Warte auf Palette     //Warte bis Sensorsignal PIN 32
HubtischYASNAK

Pruefe Palette        //Prüfe, ob Teil bereits auf Palette
                      //vorhanden
HubtischCheckY

Hole Palette          //Einschleusen in Bearbeitungsraum und
                      //in Positionbringen des Hubtisches
HubtischYASNAK



TeilnichtDAY          //Wenn Teil nicht vorhanden, dann
                      //Sprung zu Marke "TeilnichtDaY"
IF
HubtischCheckY

Weiter                //Sprung zu Marke "Weiter" wenn
GOTO                  //bereits der montierte Teil vorhanden
                      //ist

TeilnichtDAY          //Sprungmarke "TeilnichtDaY"
LABEL

Öffne Malteserrad (I) //Signal an Servogreifer (öffnen zum
                      //Greifen des Malteserrades)
SERVO-GRIP



Bewegung Pos. Malteserrad  //Starten des Roboterprogramms
                           //(Pos. YASKAWA_KSB6 Malteserrad
                           //im Magazin)

Greife Malteserrad (I)     //Sign. an Servogreifer
                           //(Schließen der SERVO-GRIP
                           //Backen zum Greifen des
                           //Malteserrades)

Bewegung Ins.Pos Malteserrad    //Starten des YASKAWA_KSB6
                           //Roboterprogramms(Einsetzen des
                           //Malteserrades
```

```
Öffne Malteserrad (I) //Sign. an Servogreifer (öffnen der
                      //Backen)
SERVO-GRIP

Bewegung ZwiPos       //Starten des Roboterprogramms
                      //(Bewegung in ein Zwischenposition)
YASKAWA_KSB6

Öffne Ritzel          //Sign. an Servogreifer (öffnen zum
                      //Greifen des Ritzels)
SERVO-GRIP

Bewegung Pos. Ritzel  //Starten des Roboterprogramms (Pos.
                      //Ritzel im YASKAWA_KSB6-Magazin)

Greife Ritzel   //Sign. an Servogreifer (Schließen der
                //Backen zum Greifen des Ritzels)
SERVO-GRIP

Bewegung Ins.Pos Ritzel    //Starten des Roboterprogramms
                           (Einsetzen //des Ritzels)
YASKAWA_KSB6

Öffne Ritzel               //Sign. an Servogreifer (öffnen
                      //des Greifers)
SERVO-GRIP

Bewegung ZwiPos       //Starten des Roboterprogramms
                      //(Bewegung in eine Zwischenposition)
YASKAWA_KSB6

Weiter                //Sprungmarke "Weiter"
LABEL

Palette Ausschleusen  //Absetzen der Palette auf dem
                      //Förderband und Öffnen der Schleuse
HubtischYASNAK
```

## 4.5 Neu TEACHEN des BOSCH-SCARA

Die Aufgabe des SCARA in dieser Montagezelle ist der Einbau des Malteserrades und des Ritzels in das auf der Palette bereits aufgesetzte Gehäuse.
Vor dem eigentlichen Programmieren mussten einige Eingangsleitungen des Bosch erneuert werden, da diese zum Teil ausgerissen bzw. das Kabel gebrochen war.

Um die Montageaufgabe zu bewerkstelligen, mussten die Greiferwechsel-programme funktionieren. Die Koordinaten für die Greiferwechselstation haben sich, wohl aufgrund einer Kollision mit dem Greiferarm oder durch andere mechanische Einwirkungen, geändert.
Deshalb mussten vor Beginn der eigentlichen Arbeit zuerst die neuen Koordinaten geteacht werden und in die Robotersteuerung übernommen werden.

Die eigentliche Programmierarbeit am SCARA-Roboter gestaltete sich eher unproblematisch, da Teile des Programms noch gespeichert waren. Anhand dieser konnte das neue Programm rasch erstellt werden. Die Struktur wurde etwas geändert, da in der ursprünglichen Version die Montage des Ritzels und des Malteserrades jeweils eine Programmnummer in Anspruch genommen haben. Diese sind nun als Unterprogramm in einem einzigen Programm integriert.
Der genaue Code ist im Anhang aufgeführt.

### 4.5.1 Ablaufprogamm von C_SEQU für BOSCH

Das folgende C_SEQU-Programm ist am Steuerrechner in das Zellensteuerungsprogramm (nur_bosch.cel) gespeichert.
Das Programm ist so aufgebaut, dass zuerst überprüft wird, ob bereits das Gehäuse auf der Palette aufgesetzt wurde. Damit würde der Steuerrechner automatisch erkennen, dass auch Malteser-Rad und Ritzel vorbereitet worden sind. Anhand dieses Sensorsignals wird nun der Programmablauf des

BoschSCARA gestartet oder die Palette wieder ausgeschleust. Nachfolgend wird nun der genaue C_SEQU-Code dargestellt.

```
Warte auf Palette //Warte bis Sensorsignal PIN 27
HubtischBOSCH

Pruefe Palette    //Prüfe ob Teil bereits auf Palette vorbereitet
HubtischCheckB

Holen             //Einschleusen in Bearbeitungsraum und in
                  //Positionbringen des Hubtisches
HubtischBOSCH

TeilnichtDAB      //If-Bedingung wenn Prüfvorgang "kein Teil
                  //vorhanden" ergeben hat
IF
HubtischCheckB

Hole Ritzel       //Starten des Roboter-Programms "Hole Ritzel"
TURBOSCARA

Hole Malteserrad  //Starten des Roboter-Programms "Hole
                  //Malteserrad"
TURBOSCARA

TeilnichtDAB      //Sprungmarke für if-Bedingung für "kein Teil
                  //vorhanden"
LABEL

Ausschleusen      //Hubtisch wird gesenkt, Stopper am Ausgang wird
                  //gesenkt
HubtischBOSCH
```

# 5 Zusammenfassung

Das Ergebnis dieser Arbeit ist wohl eher weniger befriedigend. Schließlich konnte die eigentliche Aufgabe - die Inbetriebnahme der gesamten Fertigungszelle – nicht erledigt werden. Die Anlage ist von einer Vielzahl von Studenten, Diplomanden und Assistenten im Laufe einiger Jahre entwickelt worden. Bei der Inbetriebnahme mussten also alle Informationen und Arbeiten, die für diese Anlage vorhanden waren, zuerst ausgewertet werden.

## *5.1 Puma*

Der Puma 560 ist ohne Zweifel ein mechanisch sehr robuster Roboter, dessen Probleme allerdings ganz eindeutig bei der Hardware liegen. Bis auf ein paar nachträglich ersetzte Bauteile besteht die Steuerung noch immer aus den originalen Bauteilen. Sowohl mechanische Einwirkungen (Transport der Gesamtanlage zum Freihaus d. TU) als auch die Versprödung der Kunststoffplatinen und Chips durch die schädliche Einwirkungen von UV-Strahlungen haben dafür gesorgt, dass der Roboter im Laufe der Zeit fehleranfällig geworden ist. Dazu muss gesagt werden, dass der Roboter in den letzten Jahren nur zu Laborzwecken in Betrieb genommen wurde, und wohl in den letzten Monaten weit aus stärker in Betrieb gewesen ist, als in den Jahren zuvor.

Wie in den Ausführungen im 4.2 Kapitel erwähnt, treten beim Speichern von Programmen stets kleinere Fehler auf, die es allerdings unmöglich machen, aufgrund der steuerungsinternen Absicherung Dateien anzulegen und Programme oder Programmteile zu speichern.

Für den Einsatz im Labor wäre es durchaus möglich den Roboter noch einzusetzen, da es während des Betriebes noch möglich ist, Programme zu schreiben (im EPROM). Zur Dokumentation könnten die Dateien dann mit dem von mir geschriebenen Programm, wenn auch nicht fehlerfrei, gesichert werden. Dazu müsste im STORE-Programmteil die Routine zur Zeichenanzahlprüfung

und Prüfsummenzeichenkontrolle durch einen einfachen „goto“-Befehl umgeschrieben werden.
Über kurz oder lang wird es dann wahrscheinlich aufgrund der fehlerhaften Übertragung von Daten zum Mikrocontroller beim Laden der Systemdiskette auch zu Betriebsfehlern kommen. Diese Fehler können, müssen aber nicht unbedingt zu Systemproblemen führen (bei meinen ersten Versuchen, die Übertragungsdaten zu sichern, sind bereits Fehler bei der Übertragung festgestellt worden, jedoch sind bislang keine System-Fehler aufgetaucht).

## 5.2 Servogreifer

Der am Motoman montierte Servogreifer ist, wie im Kapitel 4 bereits erklärt, mit der derzeit eingebauten Positionsregelkarte nicht mehr einsatzfähig. Wie auch schon angesprochen, sollte der Einbau eines einfachen pneumatischen Parallelgreifers in Erwägung gezogen werden, so der Roboter noch für die Montageaufgabe verwendet werden soll. Dazu müsste ein entsprechender Aufnahmeflansch konstruiert werden. Für den Einsatz in der Malteserad-Montage müsste ein Greifer mit einer Greifweite von 20mm in geschlossenem Zustand gewählt werden. Die Abmessungen der gegriffenen Teile weichen nur geringfügig voneinander ab, was den Einsatz eines genau geregelten Greifers überflüssig macht.

## 5.3 Verbesserungen und mögliche Erweiterungen der Anlage

Das ursprüngliche Anlagenkonzept der Montagezelle sah den Einsatz von weiteren Montageeinrichtungen und Magazinüberwachungen vor. Diese wurden in der letzten Konzeption nicht mehr berücksichtigt, bzw. wurden niemals realisiert.

Nun bietet die I/O-Karte des Steuerrechners noch ausreichend Platz für eventuelle Erweiterungen der Anlage.
So könnte zum Beispiel die Füllstandsüberwachung mit den am Magazin bereits installierten Sensoren vernetzt und in den Programmablauf eingebunden werden. Des Weiteren könnten die Montageaufgaben des Puma-Roboters auf die anderen beiden Roboter übertragen werden.
Das Aufsetzen des Gehäuses auf die Palette könnte vom Motoman bewerkstelligt werden. Dazu müsste das Gehäusemagazin entsprechend positioniert werden. Die Montage des Deckels könnte mit einem neuen Greifer im Werkzeugwechsler eventuell vom SCARA übernommen werden.

Wie in der Einleitung bereits erwähnt, wurden einige Teile der Anlage vermutlich während der Laborübungen beschädigt. Es ist klar, dass es sich hierbei nicht um eine Betriebsanlage, sondern um einen Versuchsaufbau handelt, welcher natürlich nicht im ausreichenden Maße gesichert werden kann. Um diese Maschinen sicher betreiben zu können, ist ein gewisses Feingefühl erforderlich. Es wäre ratsam diverse Vorkehrungen zu treffen, um einen Mangel dieses Feingefühls zu kompensieren. Außerdem wäre es empfehlenswert, Schäden sofort zu beheben, da Fehlerursachen später meist viel schwerer erkannt werden.

# Anhang

## I Programmteile der PUMA-Steuerung

**Programmteil zum Einlesen des Tastatur-Puffers und der seriellen Schnittstelle:**

```
//_____________________________________________________________
//Doppelschleife   (LESEN-SCHREIBEN)   als   Prozedur   mit
Rückgabewert b als
//der am Bildschirm dargestellte Wert.
//_____________________________________________________________

char leseschreibe()
{
int i;
char a,b;
b=0;
LESE:
asm{       mov dx,port_tD
     in al,dx
     test al,1d      //Zeichen im Empfangshalteregister
     jz TASTATUR
     mov dx,port_t8
     in al,dx
     mov ah,00h
     mov a,al
    };
     printf("%c",a);


TASTATUR:
i=kbhit();           //Prüfen des Tastaturpuffers
if(i!=0)
{
b=getch();                  //Auslesen des Zeichens wenn im
Tastaturpuffer
                     //Zeichen vorhanden ist.
if(b!=27)                  //ESC: beendet das Programm, das
Zeichen wird nicht gesendet
{
SCHREIBE:
asm{  mov dx,port_tD
     in al,dx
     test al,00100000b
     jz SCHREIBE
     mov dx,port_t8
     mov al,b
     out dx,al
```

```
    };
}
}
if(i==0) {goto ENDE;}
ENDE: return b;          //Return des Tastenzeichens f□r
Abbruchbed. ESC
}
```

Hauptprogramm:

```
//_____________________________________________________________
_____________
//  B E G I N N   D E S   H A U P T P R O G R A M M S
//_____________________________________________________________
_____________

void main()
{
int i=2;
char filechar;
char taste;
char disk_sign;
int init_err;
FELD filename;
filename[0]=65;
filename[1]=58;

textcolor(LIGHTGRAY);
clrscr();

init_err=init_COM();           //Initialisieren   der   COM-
Schnittstellen  0=>Terminal 2=>Disklaufwerk
if(init_err!=1) {printf("\n\rINIT-FEHLER%c\n\r",7);getch();}


do
{i=2;
taste=leseschreibe();
disk_sign=0;
disk_sign=disklese();
if(disk_sign==1)   // " ☺ " lt. IBM-CODE
 {
  do
  {aktion=disklese();}

while(aktion!=69&&aktion!=79&&aktion!=82&&aktion!=87&&aktion
!=67);   //(E/O/R/W/C)


  //Aufruf zum SCHREIBEN eines Datensatzes in ein File
  if(aktion==87)
{disklese_bis(raute);schreiben_eines_datensatzes();}
```

```
  //Aufruf zum LESEN eines Datensatzes
  if(aktion==82)
{disklese_bis(raute);lesen_eines_datensatzes();}


  //SCHLIEßEN einer Datei
  if(aktion==67)   {disklese_bis(raute);DATEI_SCHLIESSEN();}


  //Einlesen des Dateinamen und öffnen der Datei
  if(aktion==69||aktion==79)
  {
   do
      {
       filechar=disklese();

       if(filechar!=127)
        {
         if(filechar!=raute)
            {
             filename[i]=filechar;
             //printf("%c",filechar);
             i++;
            }
         }
      }
      while(filechar!=raute);

      filename[i]=0;   //Abschluss des Dateinamen-Strings
   DATEI_OEFFNEN(filename);
   }


 }
}
while(taste!=27);
}
```

**Routine zum Öffnen der Datei:**

```
//_____________________________________________________________
// ÖFFNEN DER DATEI zum Lesen oder zum Schreiben
//_____________________________________________________________

void DATEI_OEFFNEN(FELD name)
{

  //printf("\n\r%s",name);
  initialise=0;
  if(strcmp(name,sn5105)==0) {initialise=1;}


//_____Öffnen der Datei zum Schreiben/Lesen, je nach
Aktions-Typ (E/O)_____
  if(aktion==79)
  {
    if((datei=fopen(name,"rb"))==NULL)
       {
     //cout<<"\nFEHLER\n";
     diskschreibe(3);
       }
    else diskschreibe(okay);
  }


  if(aktion==69)
  {
     datei=fopen(name,"wb");
     diskschreibe(okay);
  }

}
```

**Routine zum Schließen der Datei:**

```
//_____________________________________________________________
_______________
//   SCHLIEßEN einer Datei
//_____________________________________________________________
_______________
void DATEI_SCHLIESSEN()
{
 fclose(datei);
 diskschreibe(okay);
}
```

**Routine zum Schreiben eines Datensatzes (STORE):**

```
//_____________________________________________________________
_______________
//   SCHREIBEN eines Datensatzes in eine Datei
//_____________________________________________________________
_______________

void schreiben_eines_datensatzes()
{
 int k=0;
 long time_out=0;
 SATZ datensatz;
 char b;
 char satz_okay;


 //Lesen der Daten vom Port
 diskschreibe(okay);
 do
 {
  datensatz[k]=disklese();
  time_out++;
  if(datensatz[k]!=127)
{printf("%c",datensatz[k]);k++;time_out=0;}
 }
 while(k<258&&time_out<200000);

 if(time_out>0) {diskschreibe(9);goto SCHREIBEN_ABBRUCH;}


 //Überprüfen des Datensatzes (Bei Fehler wird Meldung
gleich ausgegeben und
 //Schreiben in Datei □besprungen)

 satz_okay=pruefsumme(datensatz);
 if(satz_okay==0)
{diskschreibe(satzprueffehler);/*printf("SATZ-
Prüffehler");*/}


 //Schreiben des Datensatzes in die Datei
 k=0;
 while(k<258)
 {
   b=datensatz[k];
   fwrite(&b,sizeof(b),1,datei);
   k++;
 }
 diskschreibe(okay);
 SCHREIBEN_ABBRUCH:
}
```

**Routine zum Lesen eines Datensatzes (LOAD)**

```
//_____________________________________________________________
//   LESEN eines Datensatzes aus einer Datei
//_____________________________________________________________

void lesen_eines_datensatzes()
{int j=0;
 char a;
 //printf("LOAD\n\r");

 diskschreibe(okay);


//_______Laden des Calibration Overlay_____________
    if(initialise==1)
     {
      //printf("\n\rCALIBRATION OVERLAY !!\n\r");
      while(fread(&a,sizeof(a),1,datei)>0)
      {
        diskschreibe(a);
      }
      fclose(datei);
     }


//_______Laden einer Datei in Steuerung____________
    if(initialise!=1)
     {
     while(j<258&&fread(&a,sizeof(a),1,datei)>0)
     {
      diskschreibe(a);
      printf("%c",a);
      j++;
      if(j==258) printf("|\n\r");
     }
     //Wenn kein Zeichen mehr gesendet wurde, wird Ende der
     //Datei angegeben (heißt j wurde nicht mehr erhöht, da
     //die while-Schleife nicht mehr durchlaufen wurde.

if(j<=1)
{diskschreibe(end_of_file);printf("ENDE");fclose(datei);}
     }

}
```

**Routine zum Überprüfen des Prüfsummenzeichens:**

```
//_____________________________________________//
//Routine zum Überprüfen des Prüfsummenzeichens//
//_____________________________________________//

//Global deklariert Variablen
typedef char SATZ[258];

//Routine (Datensatz mit 258 Zeichen wird Übergeben)
int pruefsumme(SATZ daten)
{
int zeichen_okay=0;
char pruefzeichen;
int j=1;
long summe=0;
long komplement;
char t;         //t ist das errechnete Prüfzeichen
long und=255;   //Maske für die ersten 8 bit-Stellen

while(j<257)                          //Summe der Daten wird
gebildet
 {
  if(j>=1)
    {
     summe=summe+~daten[j];
    }
  j++;
 }

pruefzeichen=daten[j];  //258. Zeichen ist Prüfzeichen

t=summe&und;            //Und-Verknüpfung

if(t==pruefzeichen) {zeichen_okay=1;}

return zeichen_okay;    //Rückgabewert =1 wenn Summenzeichen
in Ordnung

}
```

**Initialisierung der Ports:**

```
//-------------------------------------------------------------
//INITITIALISIERUNG von COM(port)  (\include\terminal.cpp)
//-------------------------------------------------------------
//port (0,1,2,3); funktion (t...terminal,d...disk)
int init_COM()
{int err=0;

if(port_T==port_D) {goto RET;}

if(port_T!=port_D)
{
asm{  mov ah,00h
      mov dx,port_T
      mov al,11100011b
      int 14h
   }
asm{  mov ah,00h
      mov dx,port_D
      mov al,11100011b
      int 14h
   }

//Portadressen f□r (hex,dez):
//PORT__SENDE/EMPF______STATUS____
//COM1     3F8h=1016  3FDh=1021
//COM2     2F8h=760   2FDh=765
//COM3     3E8h=1000  3EDh=1005
//COM4     2E8h=744   2EDh=749

if(port_T==0) {port_tD=1021;port_t8=1016;}
if(port_T==1) {port_tD=765;port_t8=760;}
if(port_T==2) {port_tD=1005;port_t8=1000;}
if(port_T==3) {port_tD=749;port_t8=744;}

if(port_D==0) {port_dD=1021;port_d8=1016;}
if(port_D==1) {port_dD=765;port_d8=760;}
if(port_D==2) {port_dD=1005;port_d8=1000;}
if(port_D==3) {port_dD=749;port_d8=744;}
err=1;
if(port_tD==0)  {err=0;}  //zur  Überwachung,  ob  auch  ein
richtiger Wert für das Port angegeben wurde (0-3)
if(port_dD==0) {err=0;}    //es  wird  nur  geprüft  ob  die
xxDh_Portadresse einen Wert !=Null hat
                //(Adresse ist so bei globaler Deklaration
                //angegeben worden)
}

RET:
return err;
}
```

**Routinen zum Lesen der Ports für das STORE/LOAD-Modul:**

```
//_________________________
//   DISKLESE-liest Zeichen von DISK-Port
//_________________________

unsigned char disklese()
{unsigned char a=127;
asm{  mov dx,port_dD
      in al,dx
      test al,1d
      jz ENDE
      mov dx,port_d8
      in al,dx
      mov ah,00h
      mov a,al
   };
ENDE:
return a;
}



//_____________________________________________________________
___
//DISKSCHREIBE: schreibt Zeichen d auf DISK-PORT DISK-
>TERMINAL
//_____________________________________________________________
___
void diskschreibe(unsigned char d)
{
SCHREIBE:
asm{  mov dx,port_dD
      in al,dx
      test al,00100000b
      jz SCHREIBE
      mov dx,port_d8
      mov al,d
      out dx,al
    };
}



//_____________________________________________________________
//   liest Disk-Port ein, solange, bis Zeichen e empfangen
wird
//_____________________________________________________________

void disklese_bis(unsigned char e)
{
unsigned char x;
//printf("\n\rwarte auf %c",e);
```

```
do
{
x=disklese();
}
while(x!=e);
//printf("\n\r");
}



//_____________________________________________________________________
//  liest STATUS-CODE ein
//_____________________________________________________________________
char diskstatus()
{

printf("\n\rwarte auf status");
do
{
status=disklese();
}
while(status<2||status>20);
printf("\n\r");
return status;
}
```

**Programm zum satzweisen Lesen von Dateien aus einer Floppy-Disk:**

Für dieses Programm musste die Anbindung an das Terminal abgeändert werden. Das Terminal mit Diskettenlaufwerk wurde mittels eines eigenen Kabels, bei dem die TxD- und RxD- Leitungen vertauscht wurden, an die serielle Schnittstelle des PC angeschlossen.
Das Terminal wurde vom Netzteil des Schaltschrankes aus mit 115V Strom versorgt. Die Datenleitungen wurden jedoch gekappt, sodass nur der peripher angeschlossene PC mit dem Terminal verbunden war.
Das folgende Programm diente zum Lesen der Datei mit dem Calibration Overlay, welches zum Kalibrieren des Systems erforderlich ist. Es kann jedoch durch einfaches Umschreiben des Dateinamens im Programm auch jede andere Datei auslesen.
Das Ganze funktioniert nach dem selben Schema wie unter Punkt 6.3 erklärt, mit dem Unterschied, dass der PC nun die Rolle des LSI-11 übernimmt und das Diskettenlaufwerk zum Lesen einer Datei auffordert.

```
#include<iostream.h>;
#include<conio.h>;
#include<dos.h>;
#include<math.h>;
#include<stdio.h>;
#include<stdlib.h>;
#include<string.h>;
#include<ctype.h>;
#include "a:\include\discfunc.cpp";
FILE *datei;
int no_data;

unsigned char disk_data()
{unsigned char a=0;
no_data=0;
asm{  mov dx,3EDh
      in al,dx
      test al,1d
      jz FIN
      mov dx,3E8h
      in al,dx
      mov ah,00h
      mov a,al
      };
no_data=1;
```

```
FIN:
return a;
}



void lese_aufforderung()
{
diskschreibe(1);
diskschreibe(82); //"W" für write
diskschreibe(4);
}


void main()
{
int i,z,merker;
char aktion=79;   //"O" für Lesen einer Datei
char start=1;
char over=4;
char *filename="SN5105.OVE";
unsigned char charfeld[260];
char err,character;
textcolor(LIGHTGRAY);
clrscr();
//Inititalisierung
asm{  mov ah,00h
      mov dx,02h
      mov al,11100011b
      int 14h
    };
    //Aufforderung Datei SN5105.OVE zu laden
diskschreibe(start);    // 1
diskschreibe(aktion);   // "O"
i=0;
do                      // "DATEINAME"
{
  diskschreibe(filename[i]);
  printf("%c",filename[i]);
  i++;
}
while(i<10);
diskschreibe(over);     // 4

do                      //ist Datei vorhanden?
{err=disk_data();}
while(err<2||err>16);
if(err!=16) {cout<<"ERROR "<<err;}

merker=0;
datei=fopen("A:\MESG.ENG","wb");
```

```
//BEGINN DER LESEROUTINE
do
{
lese_aufforderung();     //1R4
merker=0;

do                       //Abwarten auf 16 oder 6
{err=disk_data();}
while(err<2||err>16);

if(err==16)                  //Wenn Vorgang gestartet werden
kann
{
  i=0;
  do    //IN FELD EINLESEN
  {
     character=disk_data();
     if(no_data==1)
     {
       charfeld[i]=character;
       printf("%c",charfeld[i]);
       i++;
     }
  }
  while(i<258);   //!!!!!!!
   z=i;
  i=0;
  do    //AUS FELD IN DATEI SCHREIBEN
  {
     fwrite(&charfeld[i],sizeof(charfeld[i]),1,datei);
     i++;
  }
  while(i<z);
}

if(err==6) {cout<<"ENDE DER DATEI";merker=1;}


}
while(merker!=1);
cout<<"\n    EINLESEN BEENDET !";
fclose(datei);
getch();
}
```

# II Pin-Belegung der Anlage

Im folgenden Teil wird anhand der im Anhang angeführten *.CFG-Datei die exakte Aufteilung der Steuer- und Signalleitungen aufgelistet.
Diese Auflistung soll zum einen den Ist-Zustand der Verdrahtung wiedergeben, zum anderen können auf dieser Basis Erweiterungen leichter konfiguriert werden.

## Kanal 0

Eingang (bezogen auf PC-Karte)

| CH-0 | Verwendung | Adresse Nr. |
|---|---|---|
| 0 | BOSCH | 0 |
| 1 | BOSCH | 1 |
| 2 | BOSCH | 2 |
| 3 | BOSCH | 3 |
| 4 | BOSCH | 4 |
| 5 | BOSCH | 5 |
| 6 | BOSCH | 6 |
| 7 | YASKAWA | 7 |
| 8 | YASKAWA | 8 |
| 9 | YASKAWA | 9 |
| 10 | YASKAWA | 10 |
| 11 | YASKAWA | 11 |
| 12 | YASKAWA | 12 |
| 13 | YASKAWA | 13 |
| 14 | PUMA | 14 |
| 15 | PUMA | 15 |
| 16 | PUMA | 16 |
| 17 | PUMA | 17 |
| 18 | PUMA | 18 |
| 19 | PUMA | 19 |
| 20 | PUMA | 20 |
| 21 | Servo-Greifer | 21 |
| 22 | Servo-Greifer | 22 |
| 23 | Servo-Greifer | 23 |

## Kanal 1

Eingang (bezogen auf PC-Karte)

| CH-1 | Verwendung | Adresse Nr. |
|---|---|---|
| 0 | Kurve 1 | 0 |
| 1 | Kurve 1 | 1 |
| 2 | Kurve 1 | 2 |
| 3 | BOSCH - Hubtisch | 3 |
| 4 | BOSCH - Hubtisch | 4 |
| 5 | BOSCH - Hubtisch | 5 |
| 6 | BOSCH - Hubtisch | 6 |
| 7 | BOSCH - Hubtisch | 7 |
| 8 | YASKAWA - Hubtisch | 8 |
| 9 | YASKAWA - Hubtisch | 9 |
| 10 | YASKAWA - Hubtisch | 10 |
| 11 | YASKAWA - Hubtisch | 11 |
| 12 | YASKAWA - Hubtisch | 12 |
| 13 | PUMA - Hubtisch | 13 |
| 14 | PUMA - Hubtisch | 14 |
| 15 | PUMA - Hubtisch | 15 |
| 16 | PUMA - Hubtisch | 16 |
| 17 | PUMA - Hubtisch | 17 |
| 18 | Kurve 2 | 18 |
| 19 | Kurve 2 | 19 |
| 20 | Kurve 2 | 20 |
| 21 | | 21 |
| 22 | | 22 |
| 23 | | 23 |

## Kanal 2

Eingang (bezogen auf PC-Karte)

| CH-2 | Verwendung | Adresse Nr. |
|---|---|---|
| 0 | frei | 0 |
| 1 | frei | 1 |
| 2 | frei | 2 |
| 3 | frei | 3 |
| 4 | frei | 4 |
| 5 | frei | 5 |
| 6 | frei | 6 |
| 7 | frei | 7 |
| 8 | frei | 8 |
| 9 | frei | 9 |
| 10 | frei | 10 |
| 11 | frei | 11 |
| 12 | frei | 12 |
| 13 | frei | 13 |
| 14 | frei | 14 |
| 15 | frei | 15 |
| 16 | frei | 16 |
| 17 | frei | 17 |
| 18 | frei | 18 |
| 19 | frei | 19 |
| 20 | frei | 20 |
| 21 | frei | 21 |
| 22 | frei | 22 |
| 23 | frei | 23 |

Kanal 2 wurde ursprünglich für weitere Sensor-Signale eingesetzt. Z.B. können hier künftig Magazin-Füllstandsüberwachungen eingebaut werden.

## Kanal 3

Ausgang (bezogen auf PC-Karte)

| CH-3 | Verwendung | Adresse Nr. |
|---|---|---|
| 0 | Kurve 1 | 0 |
| 1 | Kurve 1 | 1 |
| 2 | Kurve 1 | 2 |
| 3 | BOSCH - Hubtisch | 3 |
| 4 | BOSCH - Hubtisch | 4 |
| 5 | BOSCH - Hubtisch | 5 |
| 6 | YASKAWA - Hubtisch | 6 |
| 7 | YASKAWA - Hubtisch | 7 |
| 8 | YASKAWA - Hubtisch | 8 |
| 9 | PUMA - Hubtisch | 9 |
| 10 | PUMA - Hubtisch | 10 |
| 11 | PUMA - Hubtisch | 11 |
| 12 | Kurve 2 | 12 |
| 13 | Kurve 2 | 13 |
| 14 | Kurve 2 | 14 |
| 15 | Servo-GREIFER | 15 |
| 16 | Servo-GREIFER | 16 |
| 17 | Servo-GREIFER | 17 |
| 18 | Servo-GREIFER | 18 |
| 19 | Servo-GREIFER | 19 |
| 20 | Servo-GREIFER | 20 |
| 21 | Servo-GREIFER | 21 |
| 22 | Servo-GREIFER | 22 |
| 23 | frei | 23 |

## Kanal 4

Ausgang (bezogen auf PC-Karte)

| CH-4 | Verwendung | Adresse Nr. |
|---|---|---|
| 0 | BOSCH - PRG | 0 |
| 1 | BOSCH - PRG | 1 |
| 2 | BOSCH - PRG | 2 |
| 3 | BOSCH - PRG | 3 |
| 4 | BOSCH - PRG | 4 |
| 5 | BOSCH - PRG | 5 |
| 6 | BOSCH - PRG | 6 |
| 7 | BOSCH - PRG | 7 |
| 8 | YASKAWA - PRG | 8 |
| 9 | YASKAWA - PRG | 9 |
| 10 | YASKAWA - PRG | 10 |
| 11 | YASKAWA - PRG | 11 |
| 12 | YASKAWA - PRG | 12 |
| 13 | YASKAWA - PRG | 13 |
| 14 | YASKAWA - PRG | 14 |
| 15 | YASKAWA - PRG | 15 |
| 16 | PUMA - PRG | 16 |
| 17 | PUMA - PRG | 17 |
| 18 | PUMA - PRG | 18 |
| 19 | PUMA - PRG | 19 |
| 20 | PUMA - PRG | 20 |
| 21 | PUMA - PRG | 21 |
| 22 | PUMA - PRG | 22 |
| 23 | PUMA - PRG | 23 |

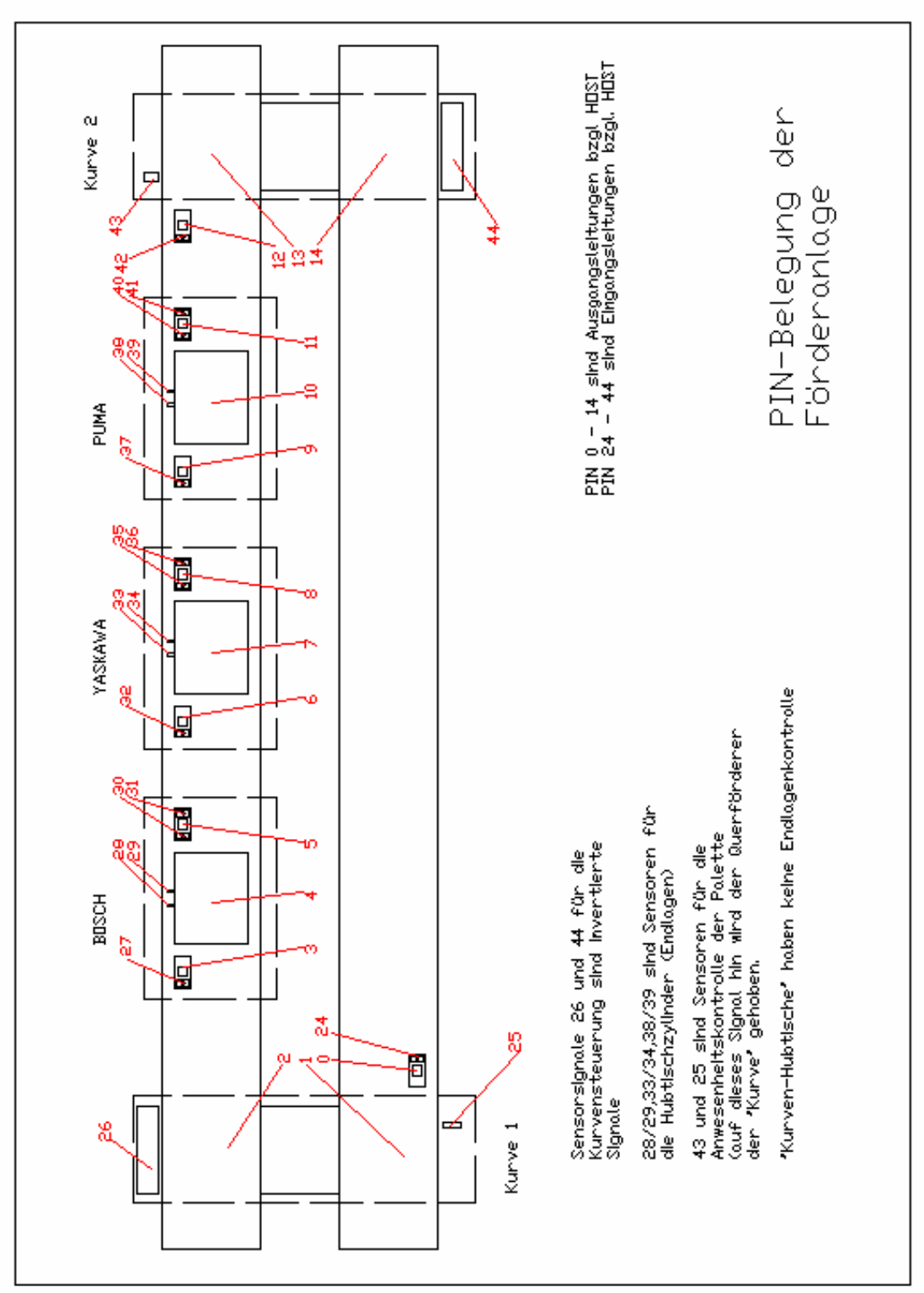
## Kanal 5

Ausgang (bezogen auf PC-Karte)

| CH-5 | Verwendung | Adresse Nr. |
|---|---|---|
| 0 | frei | 0 |
| 1 | frei | 1 |
| 2 | frei | 2 |
| 3 | frei | 3 |
| 4 | frei | 4 |
| 5 | frei | 5 |
| 6 | frei | 6 |
| 7 | frei | 7 |
| 8 | frei | 8 |
| 9 | frei | 9 |
| 10 | frei | 10 |
| 11 | frei | 11 |
| 12 | frei | 12 |
| 13 | frei | 13 |
| 14 | frei | 14 |
| 15 | frei | 15 |
| 16 | frei | 16 |
| 17 | frei | 17 |
| 18 | frei | 18 |
| 19 | frei | 19 |
| 20 | frei | 20 |
| 21 | frei | 21 |
| 22 | frei | 22 |
| 23 | frei | 23 |

Kanal 5 ist gänzlich frei gehalten und könnte für zusätzliche Ausgangs-Signalleitungen herangezogen werden. Möglich sind aktive Magazin-Elemente, wie z.B. Auswerfer. Es wäre auch denkbar zusätzliche Bearbeitungsstationen einzubauen.

## Beschreibung der einzelnen Aus- u. Eingänge der Förderanlage



# III Konfigurationsdatei

Diese Datei wurde aus der bereits bestehenden Anlage übernommen und entsprechend der aktuellen Anschlussdaten abgeändert.
Diese Datei wird in einem DOS-Editor erstellt und von C_CTRL bzw. C_SEQU bei Aufruf kompiliert. Damit keine Fehler auftreten ist es wichtig, die Kommentare mit Strichpunkt von dem eigentlichen Konfigurationstext abzugrenzen. Bei der Benennung der Bauteilgruppen ist darauf zu achten, dass sie keine Leerzeichen enthalten.

```
;************************************************************
***************
;  Cell Definition File CDF
;  Cell Setup:
;     Assembly Line for Geneva Gear
;     Robotics Laboratory IHRT, TU Vienna
;
;
************************************************************
**************
;  Definition of Board

[KARTE]
  NAME = ACL-722
  BASE = 0x2C0
  CHANNELS = 6
  PORT_I/O  =  III,III,III,OOO,OOO,OOO     ;  put  no  spaces
between
;
;  Definition Robots
[ROBOTER]
  NAME = TURBOSCARA        ; Device Identifier
  TYP = BOSCH              ; Name (description)
  PRG_I_ADRESSE = 0        ; Start address input
  PRG_I_NUMMER   =  7           ;  Number  input  lines  (busy,
standby, error signals)
  PRG_O_ADRESSE = 24       ; Start address output
  PRG_O_NUMMER  = 8        ; Number output lines (transfer of
program number)
  WW_I_ADRESSE   = 71        ; Start address input - gripper
changing device (not used)
  WW_I_NUMMER    = 0         ; Number input lines - gripper
changing device
  ERR_I_ADRESSE = 71        ; Start address error lines (not
used)
  ERR_I_NUMMER  = 0        ; Number error lines
```

```
;
[ROBOTER]
  NAME = YASKAWA_KSB6     ; Device Identifier
  TYP = MOTOMAN           ; Name (description)
  PRG_I_ADRESSE = 7       ; Start address input
  PRG_I_NUMMER   = 7          ; Number input lines (busy,
standby, error signals)
  PRG_O_ADRESSE = 32      ; Start address output
  PRG_O_NUMMER  = 8       ; Number output lines (transfer of
program number)
  WW_I_ADRESSE   = 71       ; Start address input - gripper
changing device (not used)
  WW_I_NUMMER    = 0         ; Number input lines - gripper
changing device
  ERR_I_ADRESSE = 71       ; Start address error lines (not
used)
  ERR_I_NUMMER  = 0       ; Number error lines
;
[ROBOTER]
  NAME = PUMA560          ; Device Identifier
  TYP = PUMA              ; Name (description)
  PRG_I_ADRESSE = 14      ; Start address input
  PRG_I_NUMMER   = 7          ; Number input lines (busy,
standby, error signals)
  PRG_O_ADRESSE = 40      ; Start address output
  PRG_O_NUMMER  = 8       ; Number output lines (transfer of
program number)
  WW_I_ADRESSE   = 71       ; Start address input - gripper
changing device (not used)
  WW_I_NUMMER    = 0         ; Number input lines - gripper
changing device
  ERR_I_ADRESSE = 71       ; Start address error lines (not
used)
  ERR_I_NUMMER  = 0       ; Number error lines

;
;  Interface  to  control  system  SERVO_CT  for  controlable
gripper system SERVO-GRIP
;
[GREIFER_PC]
  NAME = SERVO-GRIP           ; Device Identifier
  TYP = Greifstrg             ; Name (description)
  GRF_PC_I_ADRESSE = 21       ; Start address input
  GRF_PC_O_ADRESSE = 15       ; Start address output

;
; Definition of pallet index/lifting tables
;
[HUBTISCH]
  NAME = HubtischBOSCH     ; Device Identifier
  TYP  = HTB               ; Name (description)
  HUBT_I_ADRESSE = 27       ; Start address input  (Number =
5 (fixed); Sensor information)
```

```
  HUBT_O_ADRESSE = 3        ; Start address output (Number =
3 (fixed); Pneumatic elements)
  TIME_OUT = 30              ; Time-out: waiting for sensor
information

[HUBTISCH]
  NAME = HubtischYASNAK    ; Device Identifier
  TYP  = HTY               ; Name (description)
  HUBT_I_ADRESSE = 32       ; Start address input  (Number =
5 (fixed); Sensor information)
  HUBT_O_ADRESSE = 6        ; Start address output (Number =
3 (fixed); Pneumatic elements)
  TIME_OUT = 30              ; Time-out: waiting for sensor
information

[HUBTISCH]
  NAME = HubtischPUMA      ; Device Identifier
  TYP  = HTP               ; Name (description)
  HUBT_I_ADRESSE = 37       ; Start address input  (Number =
5 (fixed); Sensor information)
  HUBT_O_ADRESSE = 9        ; Start address output (Number =
3 (fixed); Pneumatic elements)
  TIME_OUT = 30              ; Time-out: waiting for sensor
information


;
; Definition of pallet reversing devices
;
[KURVE]
  NAME = KURVE01          ; Device Identifier
  TYP  = K1               ; Name (description)
  KURVE_I_ADRESSE = 24    ; Start address input  (Number = 3
(fixed); Sensor information)
  KURVE_O_ADRESSE = 0     ; Start address output (Number = 3
(fixed); Pneumatic elements)
  TIME_OUT = 30             ; Time-out: waiting for sensor
information

[KURVE]
  NAME = KURVE02          ; Device Identifier
  TYP  = K2               ; Name (description)
  KURVE_I_ADRESSE = 42    ; Start address input  (Number = 3
(fixed); Sensor information)
  KURVE_O_ADRESSE = 12    ; Start address output (Number = 3
(fixed); Pneumatic elements)
  TIME_OUT = 30             ; Time-out: waiting for sensor
information


;
; Definition of part feeders
;
```

```
; Number input lines: MAG_I_NR = 1 ... standard feeder or
; MAG_I_NR = 2 ... feeder with level indicator
;
[MAGAZIN]
  NAME = DeckelMag          ; Device Identifier
  TYP  = Bar_Magazine       ; Name (description)
  MAG_I_ADRESSE = 45        ; Start address input
  FUELLSTAND_SENSOR = 1     ; 1 (level indicator present)
  MAG_O_ADRESSE = 23         ; feeding process (Number output
signals = 1)-1 if no feeding
  CHK_DEVICE =               ; Identifier "Check Device" - no
entry if no check device used
  AUSWERFER_SENSORS  = 0
  AUSWERFER_ZEIT     = 4

[MAGAZIN]
  NAME = PinionPre          ; Device Identifier
  TYP  = Bar_Magazine       ; Name (description)
  MAG_I_ADRESSE = 65        ; Start address input
  FUELLSTAND_SENSOR = 1     ; 1 (level indicator present)
  MAG_O_ADRESSE = -1        ; Signal to start feeding process
(Number output signals = 1)
                            ; -1 if no feeding mechanism
  CHK_DEVICE =               ; Identifier "Check Device" - no
entry if no check device used
  AUSWERFER_SENSORS  = 0
  AUSWERFER_ZEIT     = 2

[MAGAZIN]
  NAME = GenevaPre          ; Device Identifier
  TYP  = Bar_Magazine       ; Name (description)
  MAG_I_ADRESSE = 63        ; Start address input
  FUELLSTAND_SENSOR = 1     ; 1 (level indicator present)
  MAG_O_ADRESSE = -1        ; Signal to start feeding process
(Number output signals = 1)
                            ; -1 if no feeding mechanism
  CHK_DEVICE =               ; Identifier "Check Device" - no
entry if no check device used
  AUSWERFER_SENSORS  = 0
  AUSWERFER_ZEIT     = 2


; Check Device
; Anzahl Input-leitungen=1; Output-leitungen=1;
;
[CHK_DEVICE]
  TYP = CheckPuma          ; nur Bezeichnung (kann beliebig
sein)
  NAME = HubtischCheckP ; Ger„t identifikator muá eindeutig
sein
  CHK_I_ADRESSE = 60    ; Input-leitungen begin Anzahl=1
  CHK_O_ADRESSE = 48    ; Output-leitungen begin Anzahl=1
  PUFFER = 1            ; Anzahl Teile bis Magazin
```

```
  WAIT_TIME      = 3    ; Warte Zeit in Sekunden

[CHK_DEVICE]
  TYP = CheckYask         ; nur Bezeichnung (kann beliebig
sein)
  NAME = HubtischCheckY ; Ger„t identifikator muá eindeutig
sein
  CHK_I_ADRESSE = 61    ; Input-leitungen begin Anzahl=1
  CHK_O_ADRESSE = 49    ; Output-leitungen begin Anzahl=1
  PUFFER = 1            ; Anzahl Teile bis Magazin
  WAIT_TIME      = 3    ; Warte Zeit in Sekunden

[CHK_DEVICE]
  TYP = CheckBosch        ; nur Bezeichnung (kann beliebig
sein)
  NAME = HubtischCheckB ; Ger„t identifikator muá eindeutig
sein
  CHK_I_ADRESSE = 62    ; Input-leitungen begin Anzahl=1
  CHK_O_ADRESSE = 50    ; Output-leitungen begin Anzahl=1
  PUFFER = 1            ; Anzahl Teile bis Magazin
  WAIT_TIME      = 3    ; Warte Zeit in Sekunden

;
; ENDE KONFIG - FILE
;*************************************************************
********
```

# *IV Roboter-Programme*

## Roboter-Programm (*.qll)

```
;;achsnamen=a_1,a_2,a_3,a_4
;;koordinaten=x_k,y_k,z_k,c_k

PROGRAMM MALTESER

EXTERN:greifer1,greifer2

AUSGANG :27=greifer
AUSGANG :13=busy
AUSGANG :14=stand_by
AUSGANG :15=sig3
AUSGANG :16=sig4
AUSGANG :17=sig5
AUSGANG :18=sig6
AUSGANG :19=sig7
EINGANG: 29=adr_ok
EINGANG: 22=adr1
EINGANG: 23=adr2

ANFANG
schleife:
      sig3=1
      sig4=1
      sig5=1
      sig6=0
      sig7=0
      busy=0
      stand_by=1

WARTE BIS adr_ok=1
      busy=1
      stand_by=0

WENN (adr1=1) UND (adr2=0) DANN KRANZ
WENN (adr1=0) UND (adr2=1) DANN RITZEL

sprung schleife

PROGRAMM_ENDE



UP RITZEL

PUNKT: zwp,ritzelhol,ritzelab,p3,p4
```

```
ANFANG
zwp=(205,288,195,135)
ritzelhol=(204.47,540.59,135.69,-44.93)
ritzelab=(124.43,540.84,135.73,-44.93)
p3=ritzelhol+(0,0,60,0)
p4=ritzelab+(0,0,60,0)

fahre zwp

greifer1
greifer=1
fahre nach p3
fahre linear nach ritzelhol
greifer=0
warte 0.5
fahre linear nach p3

fahre nach p4
fahre linear nach ritzelab
greifer=1
warte 0.5
fahre nach p4

RSPRUNG
UP_ENDE
```

```
UP KRANZ

PUNKT:zwp1,kranzhol,kranzab,kranz_pos,p1,p2,p5

ANFANG
zwp1=(205,288,195,135)
kranzhol=(-23.47,538.54,138.93,105.77)
kranzab=(78.19,538.54,152.04,120)
kranz_pos=(78.19,538.54,151.48,105)
p1=kranzhol+(0,0,53,0)
p2=kranzab+(0,0,47,0)
p5=kranz_pos+(0,0,0,-25)


fahre zwp1

greifer2
greifer=0
fahre nach p1
fahre linear nach kranzhol
greifer=1
warte 0.5
```

```
fahre linear nach p1

fahre nach p2
fahre linear nach kranzab
greifer=0
warte 0.5
greifer=1
fahre nach kranz_pos
fahre nach p5
fahre nach p2

RSPRUNG
UP_ENDE
```

## Roboter-Programm (Yasnac)

Hauptprogramm:

```
/JOB
//NAME MALTHAUP
//POS
///NPOS 0,0,0,0
//INST
///DATE
NOP
DOUT OT#08 0
DOUT OT#09 0
*ABFRAGE                    //Sprung-Marke
DOUT OT#04 0
WAIT IN#08=1 T=120.00       //Signal IN#08 gibt an ob
Programmnr. Von                        //Hostbereitgestellt
wurde
                            //Nach 120 Sekunden meldet die
                            //Bedienanzeige einen Überlauf,
                            welcher //vom Benutzer quittiert
                            werden muss
DIN B00 IG#01               //Einlesen des Werte von Gruppe 1
                            der //Eingangssignalleitungen
SUB B00 128
CALL JOB:MALT_1 IF B00=1    //Routine zum Anfahren des
                            //Punktes “Malteser-Magazin“
CALL JOB:MALT_2 IF B00=2    //”Malteser-Rad auf Palette“
ablegen
CALL JOB:MALT_3 IF B00=3    //”Ritzel-Magazin”
CALL JOB:MALT_4 IF B00=4    //”Ritzel auf Palette” ablegen
CALL JOB:POS5 IF B00=5      //”Zwischenposition” anfahren
DOUT OT#04 1
DOUT OT#03 0                //”Busy” wird auf Null gesetzt
JUMP *ABFRAGE               //Rücksprung zur Sprungmarke
                            //“Abfrage”
```

```
END                          //Ende des Hauptprogramms
```

## Unterprogramm-Beispiel

```
/JOB
//NAME MALT_1
//POS
///NPOS 3,0,0,0
///TOOL 0
///PULSE
C000=23680,28637,-36806,134,2506,-3791     //Punkt-
C001=32030,26036,-45306,-3384,4318,-3606   //Koordinaten
C002=33361,31356,-49622,-4503,5393,-2917
//INST
///DATE
NOP
DOUT OT#03 1                    //Signal für "Busy"
MOVL C000 V=166.7 PL=2          //Ausführen einer Linear-
                                //Bewegung
MOVL C001 V=116.7 PL=0
MOVL C002 V=8.3 PL=0
TIMER T=3.00                    //Timer halt das Programm
                                //für 3 Sekunden an
RET                             //Rücksprung zu
Hauptprogramm
END
```

Dieses Unterprogramm ist äquivalent zu den anderen im Hauptprogramm aufgerufenen Routinen, es müssen lediglich die Punkte neu geteacht werden.

-

## Literaturverzeichnis:


- Kronreif, Gernot: Robotized assembly of geneva gears – a case study for education in robotics, in Kopacek/Noe: intelligent assembly and disassembly, Bled Slovenia, 1998, S.117-122


Handbücher:


- Bosch GmbH: Handbuch zu BOSCH turboscara SR60 mit Robotersteuerung IQ140, 1992

- Kopacek/Probst/Kronreif: Objektorientierte Roboterzellen Steuerungssoftware / Bediener-Handbuch zu Roboterzellen Programmgenerator „C_SEQU“ und Steuerungssoftware „C_CTRL“ Version 4.0, Insitut f. Robotertechnik u. Handhabungsgeräte TU-Wien, 1998

- Unimation® (Europe) Ltd.: Puma™ 500 MK2 Robotersystem / Technische Anleitung 506.9055, 1982

- Yaskawa Elektronics: Motoman® K6SB mit Yasncac® ERC-Steuerung, 1988