Johann Wolfgang Goethe-Universität
Frankfurt am Main

Fachbereich Informatik (20), Lehrstuhl für Verteilte Systeme und Fehlertoleranz *VSFT*

Robotersteuerung

Vorlesung WS 1987/88

Dr.R. Brause

Inhalt

1.0 Einleitung

Schon seit langem werden Menschen von der Idee fasziniert, einen künstlichen Menschen zu bauen, der alle (unangenehme) Arbeit für sie verrichtet.
Solch ein Humunkulus wurde auch Anfang der 20-ger Jahre in dem Theaterstück *Rossums Universelle Roboter* von dem tschechoslowakischen Autor Karel Capek beschrieben. Das tschechische Wort "robota" heisst dabei soviel wie "hart arbeiten". In den Übersetzungen wurde dieses Wort beibehalten, und so benutzte auch *Isaak Asimov* in seinen bekannten Science-Fiction Romanen seit 1939 diese Bezeichnung. Im Unterschied zu dem Theaterstück, das katastrophal für die Menschen endet, sind für Asimov Roboter gut entworfene, fehlersichere Maschinen, die den folgenden drei Gesetzen gehorchen:

1) Ein Roboter darf keinen Menschen verletzen oder durch Inaktivität dies zulassen.

2) Ein Roboter muß allen Befehlen eines Menschen gehorchen, falls dies nicht in Konflikt zum ersten Gesetz führt .

3) Ein Roboter muß seine eigene Existenz beschützen, falls dies nicht zum Widerspruch mit dem ersten und zweiten Gesetz führt.

Asimov und die zahllosen Science-Fiction Schreiber und Filmautoren nach ihm leisteten durch ihre Robotergeschichten zweifelsohne einen wichtigen Beitrag zur Entwicklung der Roboter. Sie faszinierten und begeisterten viele junge Menschen, die später als Ingenieure und Wissenschaftler sich ernsthaft mit dem Phänomen *Roboter* auseinandersetzten.
Obwohl schon seit den frühen 50-ger Jahren Roboter konstruiert wurden, bekam dieses Thema erst eine reale Bedeutung, als die technische Entwicklung der Mikroprozessortechnik preiswerte Kontrollschaltungen möglich machte und den Robotern damit einen Platz in der Produktionsautomatisierung schaffte.

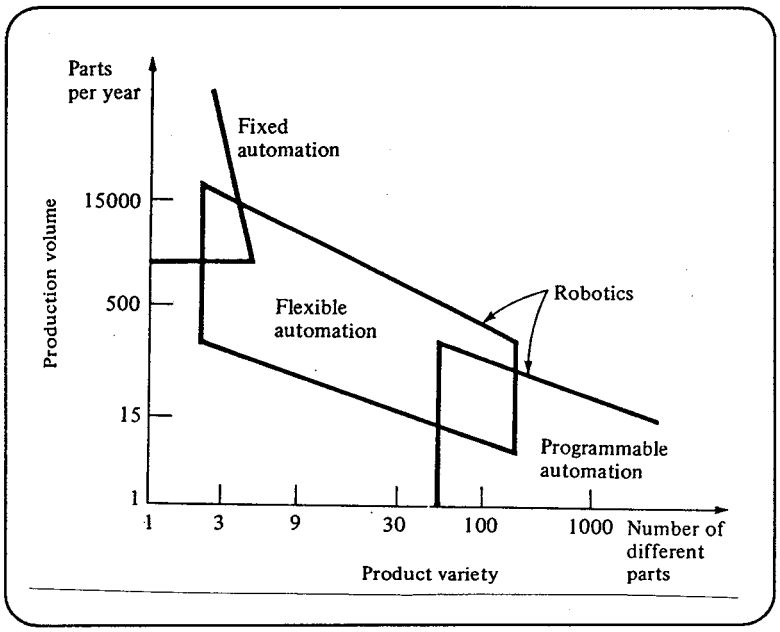


Abb.1a Einordnung der flexiblen Fertigung durch Roboter

In Abb.1a ist die Produktion, charakterisiert durch Produktionsvolumen und Produktvarietät, in drei Bereiche eingeteilt.

Feste Fertigung
Möchte man preiswert Massenprodukte wertigen, so sind dafür am besten Maschinen geeignet, die speziell für diese Produkte gefertigt sind . Ändern sich aber Konsumgewohnheiten und Technologie relativ schnell, so kann dies zum Problem werden, da diese speziellen Maschinen nur schwer geändert werden können.

Programmierte Fertigung
Ganz anders ist dagegen die Situation bei programmierbaren Fertigungsautomaten, z.B. numerisch gesteuerten Fräsmaschinen. Sie verfügen über einen Satz von Werkzeugen, der - für jedes Produkt unterschiedlich - zusammen mit dem kompletten Arbeitsprogramm die Produkttypische Ausrüstung darstellt. Zwar sind die Maschinen universelll programmierbar, aber die Umstellung auf ein anderes Produkt dauert eine gewisse Einrüstzeit, so daß immer nur ein Produkt in Serie gefertigt wird.

Flexible Fertigung
Die Die Lücke zwischen der festen und der programmierten Fertigung wird durch den Einsatz von Robotern bei der flexiblen Fertigung gefüllt. Die Überschneidungen der Bereiche zeigt, daß die flexible Fertigung positive Eigenschaften sowohl von der festen als auch von der programmierten Fertigung aufweist.
Die flexible Fertigung besteht meist aus einer Anzahl von Fertigungszellen, die durch ein Transport- und Pufferspeichersystem verbunden sind. Die Bearbeitungsfolge der Werkteile wird durch ein zentrale Computersystem koordiniert, in dem die Anforderungen an das Endprodukt gespeichert sind. Beispiel dafür ist ein auftragsorientiertes Automobilwerk, das nach den Farb- und Ausstattungswünschen der Kunden die Autos "maßgerecht" fertigt.
Die Roboter unterscheiden sich dabei von der programmierten Fertigung durch den Einsatz von "menschenähnlichen" Teilen wie Arme, Bewegungsapparat, etc, mit denen sie andere Maschinen beladen und entladen können, Schweißarbeiten ausführen (wobei die Schweißtransformatoren etc bereits im Roboter integriert sein können) und Lackierarbeiten, vorzugsweise in Autofabriken.
Die Marktchancen sind dabei ziemlich gut, wie die Verkaufs- und Gesamtinstallationszahlen für die USA im folgenden Diagramm (Abb 1b) zeigen.

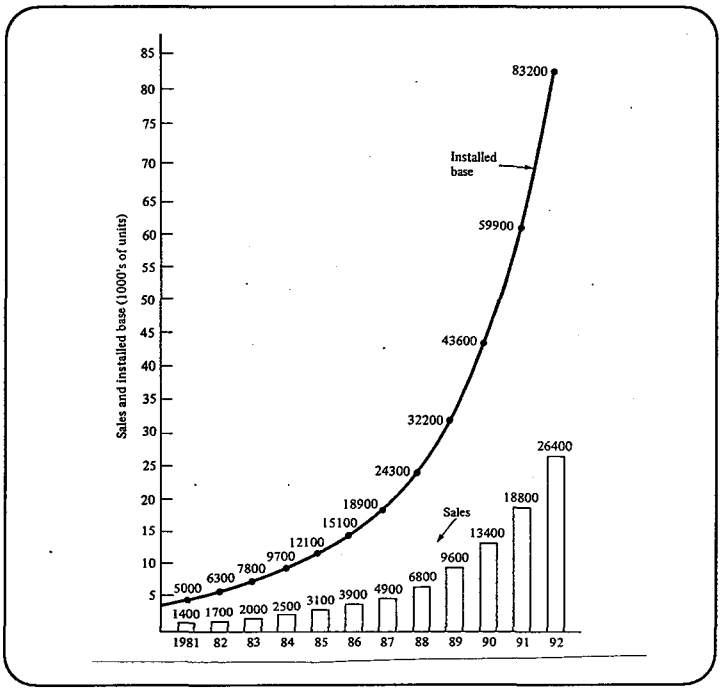


ABB 1b Verkauf und Bestand von Robotern

Mit einer Wachstumsrate von 25% ist der Robotermarkt sicher eine Branche mit Zukunft. Durch die Entwicklung von besseren Benutzerschnittstellen und durch fallende Preise für die Elektronik und damit auch der Roboter selbst werden Roboter auch für kleinere und mittlere Firmen interessant, die meist nur Kleinserien fabrizieren. Damit ist dann ein weiteres Anwachsen der Wachstumsrate zu erwarten, so daß auch bei einer Abschreibungszeit von ca. 7Jahren pro Roboter der Gesamtbestand stark wächst.
Bei allem Enthusiasmus für die Möglichkeiten der Roboter, harte, gefährliche, gesundheitsschädigende und monotone Arbeit anstelle der Menschen zu verrichten, dürfen allerdings die sozialen Probleme durch diese Möglichkeit der Rationalisierung nicht vergessen werden. Aus Abb.1b läßt sich die wahrscheinliche Zahl der menschlichen Arbeitsplätze in den USA, die durch die Installierung von Robotern wegrationalisiert werden, leicht erschließen. Nehmen wir an, daß im Mittel jeder Roboter drei Arbeitsplätze ersetzt, so ergibt sich das Bild von ersetzten bzw. gar nicht erst geschaffenen menschlichen Arbeitsplätzen in Abb.1c.

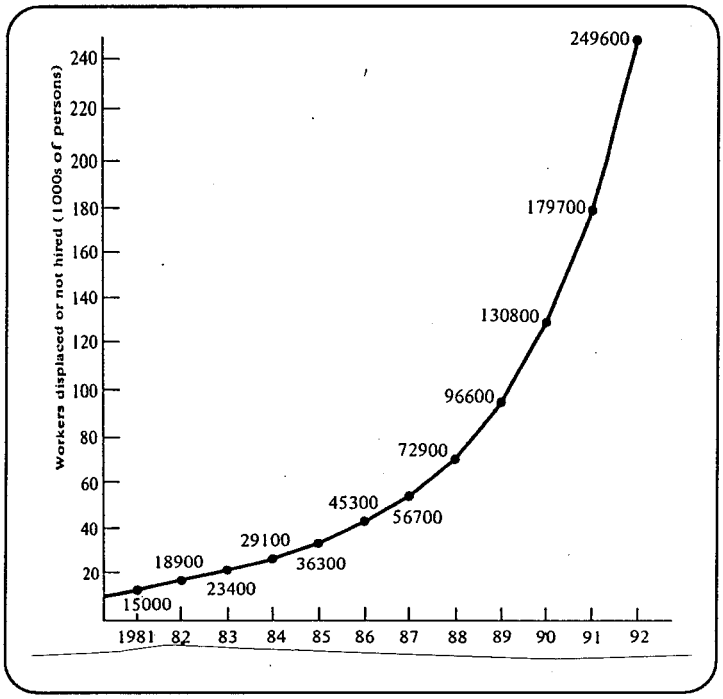


Abb. 1c Arbeitsplatzvernichtung durch Roboter

Die direkte Folge dieser Automatisierung ist ein erhöhter Bedarf an weniger qualifizierten Arbeitern, die Programmierung, Service und Störungsdienst der Roboter übernehmen. Dies bedeutet einen Zwang zur Weiterbildung für die Arbeiter oder die Entlassung.
In manchen Branchen lassen sich Entlassungen dadurch verhindern, daß durch die Automatisierung eine Verbesserung der Produktqualität und Verringerung der Produktpreise möglich wird und mit steigendem Absatz Produktion und Belegschaft ausgeweitet werden. Beispiel dafür sind die Banken, die durch die Computerisierung der Buchungsvorgänge ein Vielfaches an Umsatz bei gleichem Personal erfahren hatten.

Um die voraussichtlichen Konflikte bei der Einführung von Robotern abzuschätzen läßt sich beispielsweise folgende Ckeckliste verwenden:

Item	Points to be Distributed
1. Can workers be openly assured of job retention?	20
2. Can workers displaced, but retained, be placed in equally rated jobs?	15
3. Will the installations benefit the workers in terms of: a. Health? b. Safety? c. Relief from dehumanizing jobs? d. Relief from dirty, overly hot, back-breaking, onerous tasks?	15
4. Is the present union–management climate favorable to open exchange? Disclosure of economic conditions? Labor unrest and frequent grievances? Usually distrustful? (If no union, assign points on similar issues for management–work force relations.)	15
5. Is the present economic condition of the organization sufficiently healthy to guarantee that promises are kept?	5
6. Have manufacturing engineering and other management units shown the ability to establish rapport with workers or does inordinate "social distance" exist?	5
7. Is there management recognition and concern for the dehumanizing aspects of jobs to be performed by a robot? Or is the concern solely economic?	5
8. Is there a plan to select and upgrade workers who will supervise or perform setups for the robot?	5
9. Will workers on incentive rates be penalized by new rates or robot downtime that is not attributable to operator?	5
10. Has management in the past demonstrated respect and regard for the talents, skills, and intelligence of the workers?	3
11. Is the organization willing to share the results of this checklist with the work force and/or union?	3
12. Will robot training be on organization time? Is there willingness to send the workers (if required) to the robot vendor's training school?	2
13. Can workers express their concerns, apprehensions, and fears, without riducule?	2

Scoring:

1. Total Driving Points = _____
2. Total Restraining Points = _____

Net Score (1–2) = _____

Interpretation:

Range of Net Score	*Probability of Acceptance*
80–100	High. Implementation may proceed, assuming management acceptance conditions are equally high rated.
60–80	Proceed with Caution. After examination of the feasibility of changing strength of existing forces.
40–60	Insufficient. Reexamination of forces and management action required to increase probability.
Below 40	Failure More Than Likely. A score in this range indicates a poor probability of even modifying the forces.

2.0 Robotertypen, Greiferarten und Fertigungszellen

In der flexiblen Fertigung stehen Roboter meist nicht allein, sondern sind Teil einer automatisierten Fertigungsanlage. Gemäß der Produktionsanforderung und -umgebung haben sich dabei verschiedenen Robotertypen als besonders günstig herauskristallisiert.

2.1 Robotertypen

Die Grundform der meisten, in der Produktion eingesetzten Roboter ist die eines in allen drei Dimensionen mit drei Freiheitsgraden beweglichen Armes. Verwenden wir das kartesische Koordinatensystem (x,y,z), so benutzt der am Roboter befestigte Greifer drei Translationen, um zu einem gewünschten Punkt (x_1,y_1,z_1) zu gelangen.

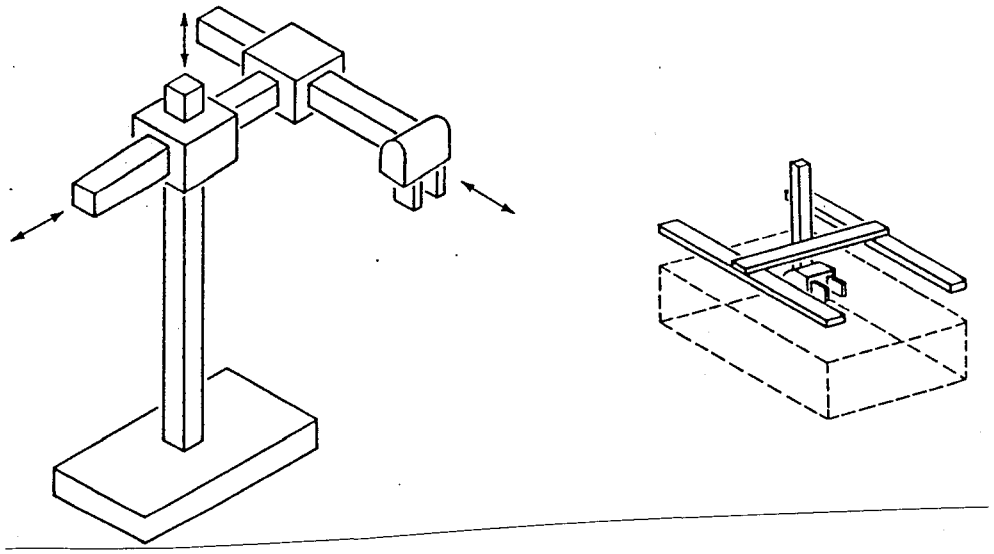


Abb.2.1a Kartesische Roboterkonfiguration und Arbeitsraum

Benutzen wir anstelle der drei linearen Koordinaten nur zwei und dafür einen Winkel ß, so erhalten wir das zylindrische Koordinatensystem (r,ß,z).

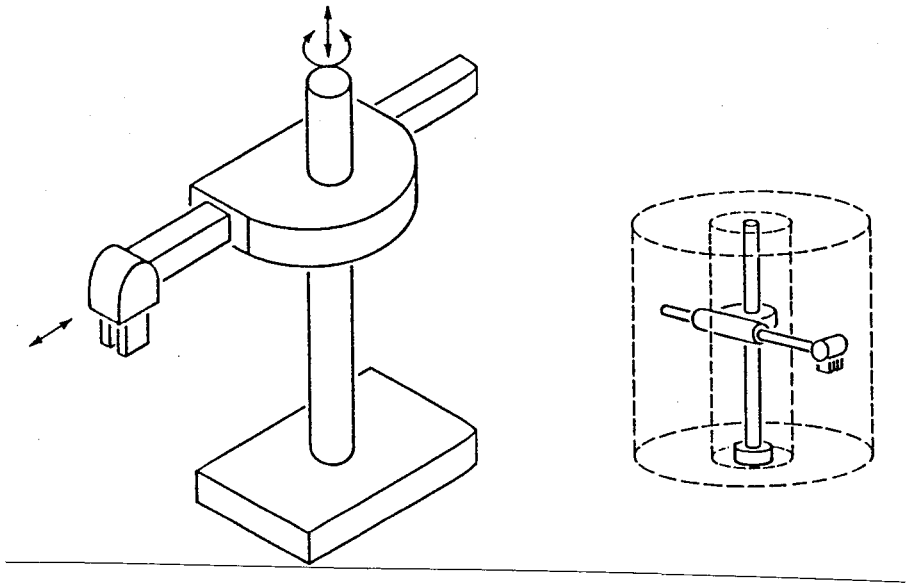


Abb.2.1b zylindrische Roboterkonfiguration und Arbeitsraum

Verwenden wir für die drei Freiheitsgrade nur noch eine lineare Koordinate und zwei Winkel, so resultiert das polare Koordinatensystem (r,ß, Θ).

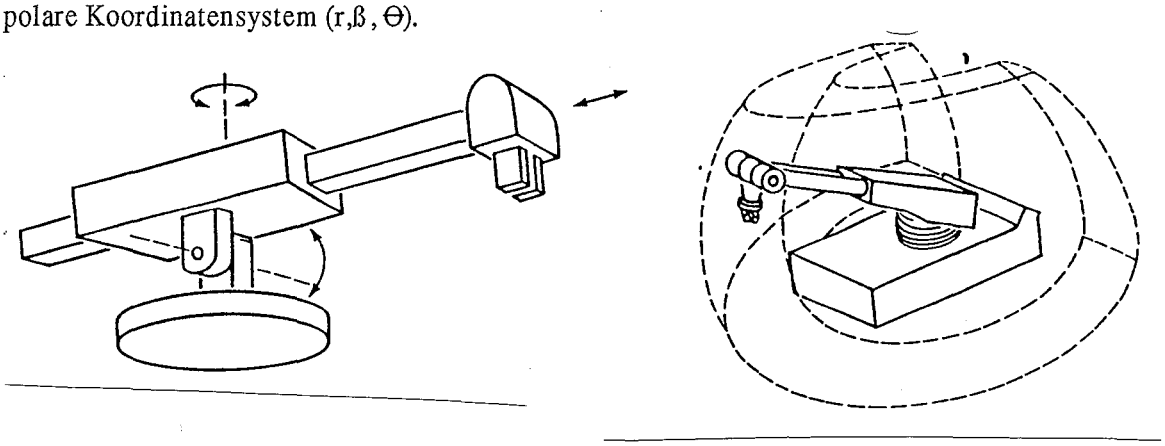


Abb. 2.1c polare Roboterkonfiguration und Arbeitsraum

Eine weitere, vielbenutzte Alternative ist es, nur noch Drehachsen bei der Roboterkonstruktion zu verwenden. Die Koordinaten sind dabei nur noch Winkel $\Theta = (\Theta_1,\Theta_2,\Theta_3)$.

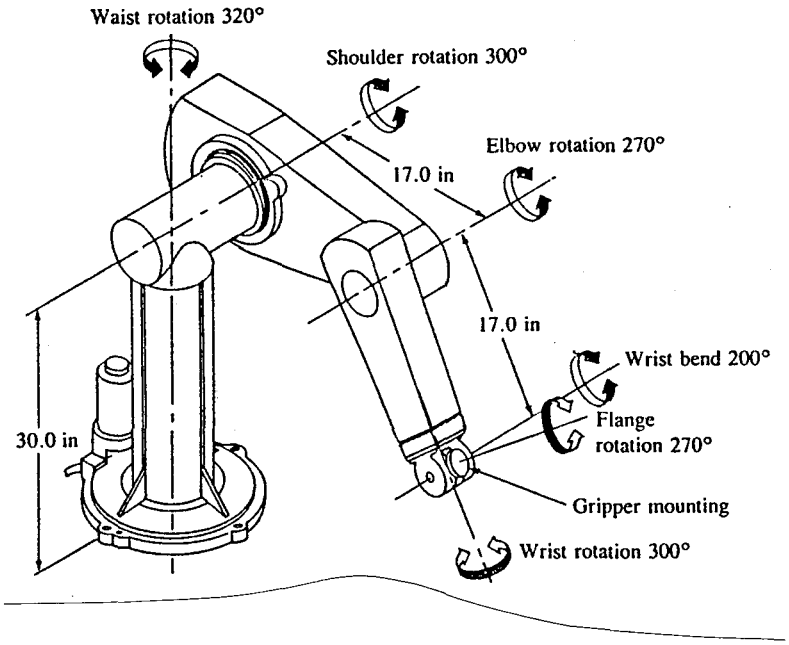


Abb. 2.1d armähnliche Roboterkonstruktion

Der Vorteil der armähnlichen Konstruktion liegt darin, daß der menschenähnliche Arm sehr einfach über Hindernisse hinweg greifen kann.
Die vier Robotertypen finden beispielsweise bei folgenden Arbeiten Verwendung:

Material-Transfer	3-5 Achsen, armähnliche Konstruktionen, pneumatischer und hydraulischer Antrieb.
Maschinen-Beladung	4-5 Achsen, polare, zylindrische und armähnliche Konfigurationen, elektrischer oder hydraulischer Antrieb, Punkt-zu-Punkt Bewegung.
Punktschweißen	5-6 Achsen, polare und armähnliche Konfigurationen, elektr. oder hydraulischer Antrieb, Punkt-zu-Punkt Bewegung.
Lichtbogenschweißen	5-6 Achsen, polare, kartesiches und armähnl. Konfigurationen elektr. und hydraulischer Antrieb, kontinuierliche Bewegung.

Farbspritzen	6 oder mehr Achsen, hydraulischer Antrieb, kontinuierliche Bewegung.
Montage	3-6 Achsen, Kartesische und armähnliche Konfigurationen, elektr. Antrieb, kontinuierliche und Punkt-zu-Punkt Bewegung, hohe Bewegungsgenauigkeit.

2.2 Greifertypen

Der ideale Greifer eines Roboterarms ist anwendungsabhängig und muß typischerweise vom Kunden oder von dem Roboterlieferanten speziell für die Einsatzaufgabe maßgeschneidert werden.
Es gibt sehr verschiedene Werkzeuge, die als "Hand" am Roboterarm befestigt werden können.

Der mechanische Greifer
Der gewöhnliche, aus zwei beweglichen "Fingern" gebildete, mechanische Greifer ist nur einer von vielen Werkzeugtypen und wird meist zum Materialtransport und Maschinenbeladung verwendet. Wird ein Werkteil in einer Maschine bearbeitet, so kann ein *Doppelgreifer* wertvollen Zeitgewinn im Produktionszyklus bringen. Der eine Greifer holt das bearbeitete Teil aus der Maschine und hält es, so daß der andere Greifer ein unbearbeitete Teil in die Bearbeitungsmaschine geben kann. Während der Arm zum Transportsystem schwenkt um das bearbeitete Teil gegen ein unbearbeitetes auszutauschen, kann parallel dazu in der gleichen Zeit das Teil bearbeitet werden.
Obwohl für manche Arbeiten wie Schweißen, Farbsprühen etc die Werkzeuge direkt am Roboterarm angebracht sind, werden gern allgemeine, mechanische Greifer verwendet, um im Produktionszyklus einem Arm zu ermöglichen, ein Werkstück mit mehreren, verschiedenen Werkzeugen zu bearbeiten. Dabei stellt sich die Frage, wie ein Greifer ein Objekt festhalten kann, ohne daß es bei einer schnellen Armbewegung aus dem Greifer rutscht. Es gibt zwei verschiedene Methoden, dies zu verhindern.

a) Die Greiferform ist an die Oberflächenform angepaßt und verhindert so ein Herausrutschen (Abb 2.2a)

b) Der Druck der Greiferfinger ist so groß, daß mit der Haftreibung u zwischen Fingeroberfläche und Objekt das Gewicht mg und die Beschleunigungskraft ma durch die Armbewegung kompensiert werden kann (Abb 2.2b)

$$F u \quad > \quad m\,(a+g)$$

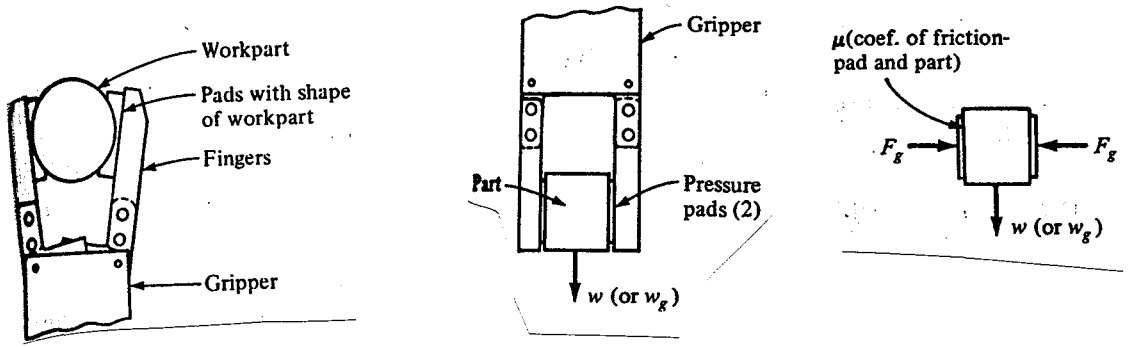


Abb.2.2a,b Greiferformen und Haftreibung

Dazu gibt es verschiedenartige, mechanische Konstruktionen des Greifers; beispielsweise läßt sich der Greifer durch eine Winkelbewegung oder aber durch eine lineare, gleichförmige Bewegung schließen. In

Abbildung 2.2c sind verschiedene Konstruktionen abgebildet.

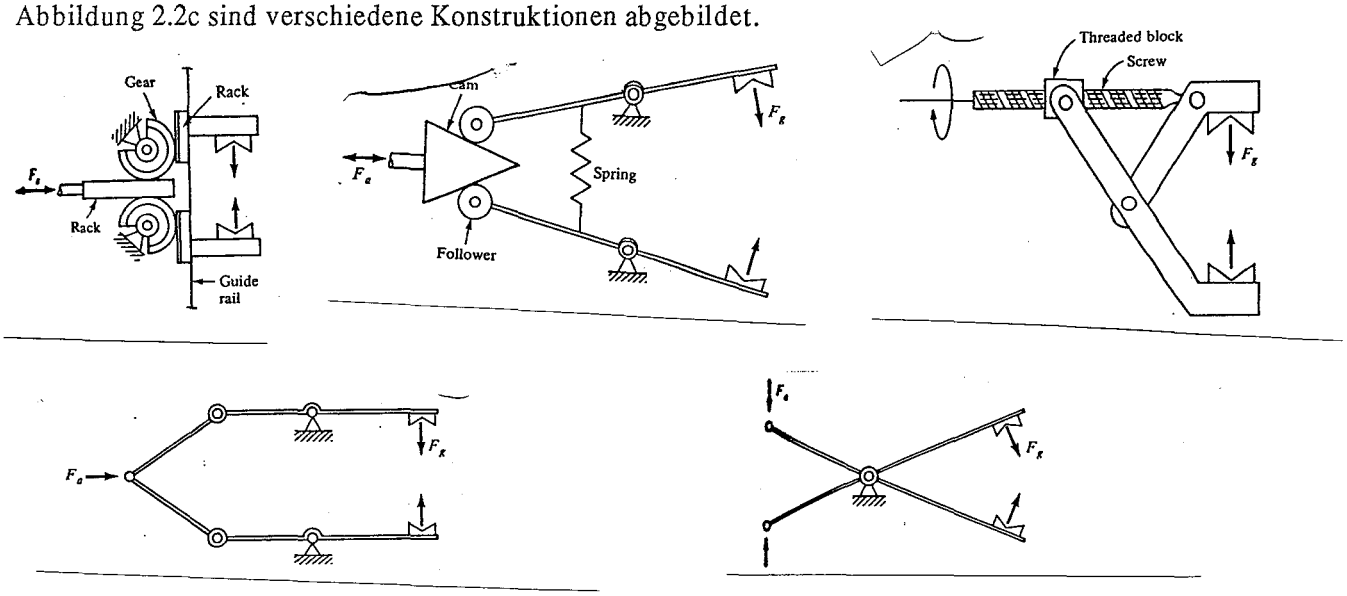


Abb.2.2c mechanische Greiferformen

Als universeller Greifer wurde eine menschenähnliche 3-Finger Hand in Stanford (JPL) konstruiert:

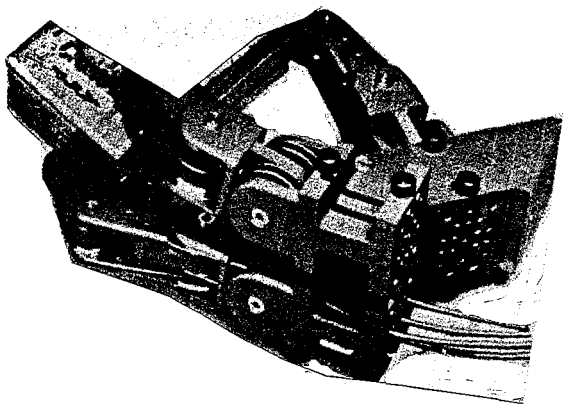


Abb.2.2d mechanischer Universalgreifer

Vakuumglocken

Muß Material mit einer glatten, ebenen Oberfläche transportiert werden (z.B. Glas- und Kunststoffplatten, dünne Stahlbleche), so lassen sich als Greifer vorteilhaft Vakuumglocken einsetzen. Da der mögliche Saugdruck mit dem Luftdruck begrenzt ist, muß man bei der Auslegung von Zahl und Fläche der Saugglocken darauf achten, bei gegebenen Saugdruck (=Saugkraft pro Flächeneinheit, max. der Luftdruck) für das Gewicht der Platten eine ausreichende Dimensionierung zu erreichen. In der folgenden Abbildung ist links das Prinzip der (preiswerten) Vakuumerzeugung durch Preßluft gezeigt und rechts eine Anwendung der Vakuumglocken bei der Handhabung von Glasplatten.

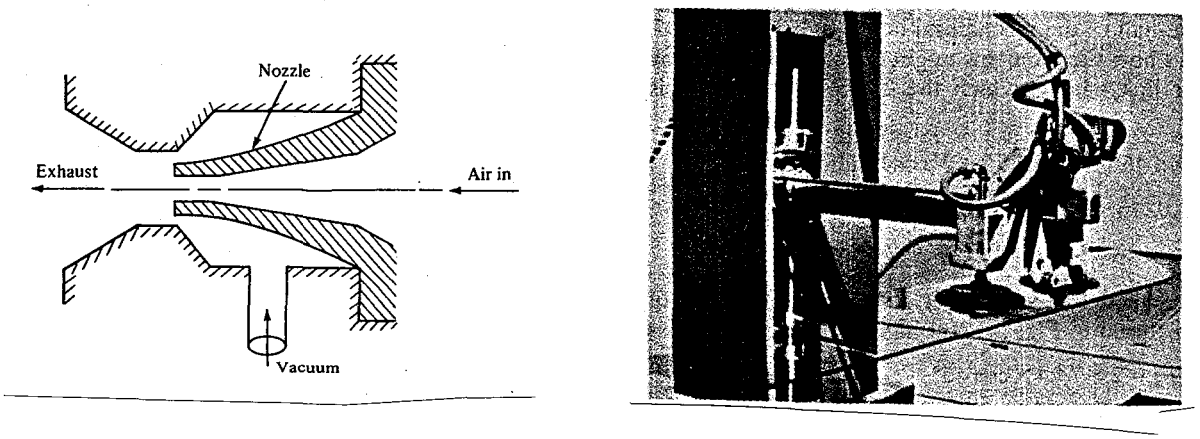


Abb.2.2e Vakuumerzeugung und Greifereinsatz

Magnetische Greifer

Hat das Material magnetische Eigenschaften, wie beispielsweise Eisenplatten, nicht aber Aluminiumteile oder rostfreier Stahl, so bieten magnetische Greifer viele Vorteile:

- schneller Zugriff, nur die Oberfläche des Werkstücks nötig
- die Größe der Materialteile kann stark variiert werden
- Material mit Löchern kann gegriffen werden (im Unterschied zu Vakuumglocken)

Nachteile sind

- verbleibender und evtl. störender Restmagnetismus im Werkstück
- Verrutschen des Werkstücks
- ungenaue Selektion (z.B. nicht nur die obersten Platte im Stapel)

Durch geeignete elektrische und konstruktive Maßnahmen lassen sich aber viele der genannten Nachteile kompensieren.

Es gibt eine Vielzahl von anwendungsspezifischen Werkzeugen, die aber hier nicht weiter aufgeführt sind. Als Beispiel soll in Abb.2.2f ein Roboter-Schraubenzieher gezeigt werden. Links wird eine Schraube angesetzt, rechts festgeschraubt.

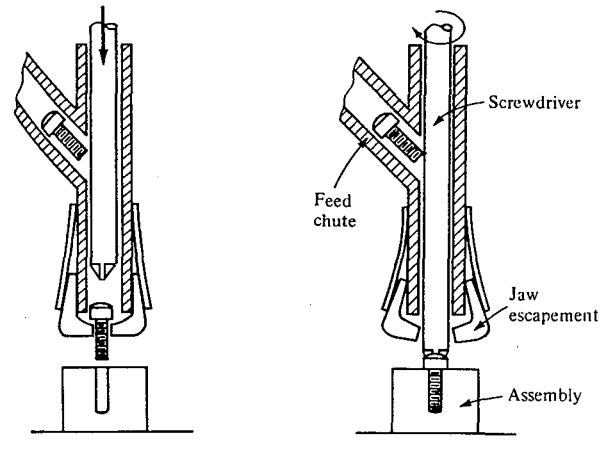


Abb.2.2f Schraubenzieher für Roboterarm mit Schraubenzuführung

Das Interface zum Roboterarm

Die elektrische und mechanische Schnittstelle zwischen Roboterarm und -hand muß, ähnlich einem Programmodul, genau spezifiziert werden.

Die physikalische Verbindung muß bieten

- ausreichende *mechanische Festigkeit*, um das Werkstück zu halten
- *elastische* oder *starre* Verbindung, je nach Anwendung
- *Überlastungsschutz* für Zwischenfälle, wahlweise durch
    - Sollbruchstellen . Vorteil: billig. Nachteil: muß von Menschen gewechselt werden
    - einen durch Federspannung kontrollierten Ausrastmechanismus
    - Sensorkontrolle. Nachteil: relativ teuer. Vorteil: Bei Fehlern können komplexe Meldungen und Behandlung programmiert werden (Ausweichen, Fließband abschalten etc)

Die Energie für die Greiferfunktionen richtet sich nach der im Roboterarm benutzten Energie. Im einfachen Fall existiert pneumatische Energie. Dies ist zwar billig und unproblematisch in rauher und exposionsgefährdeter Umgebung (Lackieren), ermöglicht aber meist nur EIN-AUS Bewegungen, da die Kontrolle der Preßluft nicht sehr genau ist. Für anspruchsvollere Aufgaben eignet sich besser elektrische Energie, die feine, sensor-kontrollierte Bewegungen durch Elektromotoren ermöglicht. In Anwendungen mit hohem Kraftaufwand oder einer Feinsteuerung trotz Explosionsgefahr wird hydraulische Energie verwendet.

Die Kontrollsignale für den Greifer müssen natürlich in ihren elektrischen Charakteristika zum Greifer passen, ebenso wie die**Rückkopplungssignale** von den Greifersensoren.

Greiferdesign

Bei der Auswahl des Greifers sollten, abgesehen von der Wahl des Greifer-Interfaces und der anwendungsspezifischen Sensoren- einige kritische Punkte nicht aus den Augen gelassen werden:

- das zu kontrollierende Werkstück muß nicht nur auf eine Art gehalten werden können, sondern auch genauso greifbar sein (nicht verdeckt von Halterungen etc.)

- Beim Be- und Entladen von Bearbeitungsmaschinen sowie beim Halten von Werkstücken bei der Bearbeitung (Drehbank etc) muß beachtet werden, daß Form und Größe des Wwerkstücks sich ändern können. Beispiel: Be- und Entladen von Formpressen.

- Bei zerbrechlichen oder leicht verbiegbaren Teilen sowie Teilen mit empfindlicher Oberfläche muß der Greifer entsprechend konstruiert werden.

- Kann ein Teil an einer schmalen oder breiten Stelle gefaßt werden, so sollte aus Stabilitätsgründen die breite Stelle vorgezogen werden.

- Die Greiferfinger sollten auswechselbar und selbst-anpassend sein, so daß bei einer Vielzahl von Anwendungen eine möglichst große Kontaktfläche entsteht.

2.3 Zuführungsautomaten

Werden Roboter in der Montage von Teilen eingesetzt, so ist eine wichtige Vorbedingung der Robotertätigkeit dadurch gegeben, daß die nötigen Teile am genau festgelegten Orten (Positionen) in genau bekannter Orientierung zur Verfügung gestellt werden müssen. Einer der bekanntesten Zuführungsautomaten, der diese Bedingungen erfüllt, ist der Vibrationstopf. Dabei handelt es sich um ein topfförmiges Gefäß, das durch einen Motor im Sockel in Vibrationen versetzt wird. Der Drehpunkt der Vibrationsschwingung ist in der Mitte des Sockels, so daß alle Kleinteile im Topf eine mittlere Kraft nach außen und oben erfahren. Versieht man den Topf innen mit einer spiralförmig aufsteigenden Kante (s. Abb. 2.3a), so bewegen sich die Kleinteile darauf empor.

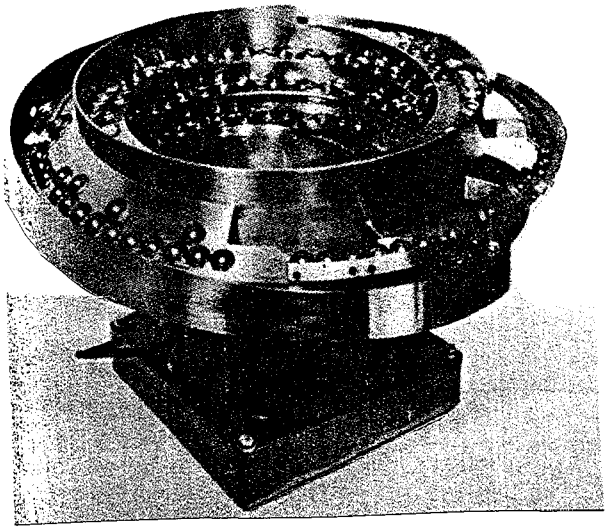


Abb.2.3a Vibrationstopf

Durch trickreich angeordnete Fallen (Nuten etc) werden alle Teile, die falsche Lage oder Orientierung haben, wieder in den Topf zurückgelenkt. Am Ende der der Zuführung sorgt dann ein Mechanismus für eine getrennte Entnahme jedes Kleinteils.

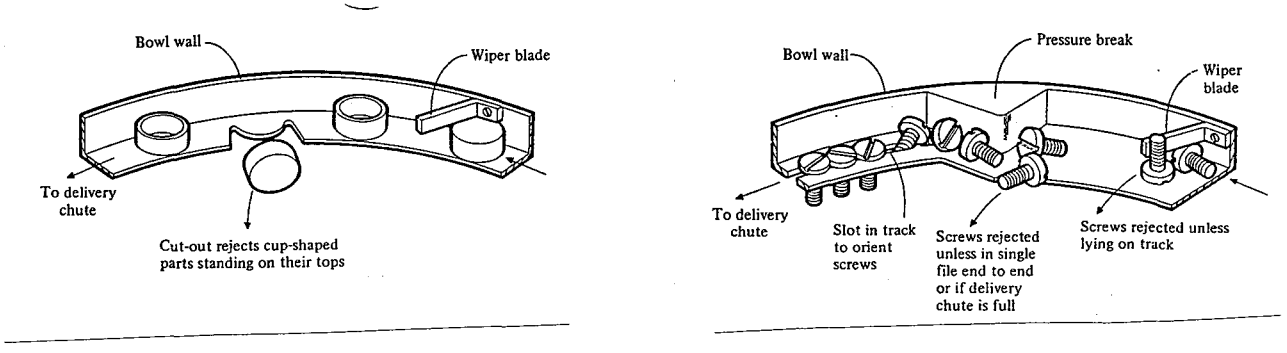


Abb. 2.3b Sortierung von Kleinteilen mit bestimmter Orientierung

2.4 Fertigungszellen

Der Einsatz von Robotern in der Produktion ist meist nicht isoliert (*Inselbetrieb*), sondern innerhalb einer Produktionsumgebung. Die Gesamtheit von Zuführungsautomaten, Förderbändern, Bearbeitungsmaschinen und der Koordinationselektronik für einen Bearbeitungsvorgang wird als **Arbeits-** oder **Fertigungszelle** (*workcell*) bezeichnet. Gibt es Teilaufgaben (Qualitätsprüfung etc), die bei dem Bearbeitungsvorgang nötig sind und nur von Menschen übernommen werden können, so wird vom Planer der Fertigungszelle auch der Mensch als Teil der Fertigungszelle betrachtet.

Typen von Fertigungszellen

Im allgemeinen werden drei verschiedene Arten von Fertigungszellen unterschieden: die *Roboterzentrierte Zelle*, die *in-line Roboterzelle* und die *mobilen Roboterzellen*.
Bei den **roboterzentrierten Zellen** sind die Zuführungs- und Bearbeitungsautomaten um den Roboter als zentralen Punkt angeordnet.

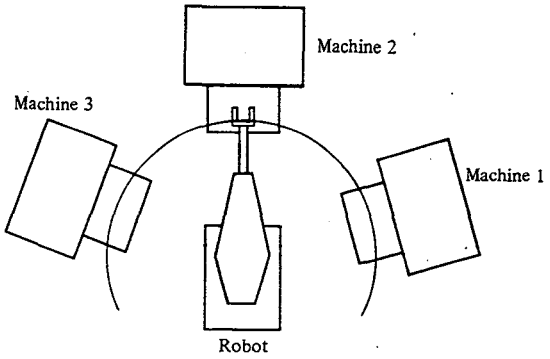


Abb. 2.4a Roboterzentrierte Fertigungszelle

Diese Anordnung wird meist für Metallguß, Plastik-Spritzguß und ähnliche Aufgaben verwendet, bei denen der Roboter die einzelnen Teile direkt aus der Maschine holt und so Position und Orientierung der Teile genau festgelegt sind.
Die **in-line Roboterzellen** werden in einer Reihe am Transportsystem (Fließband) angeordnet und dienen zur sequentiellen Bearbeitung von Werkstücken.

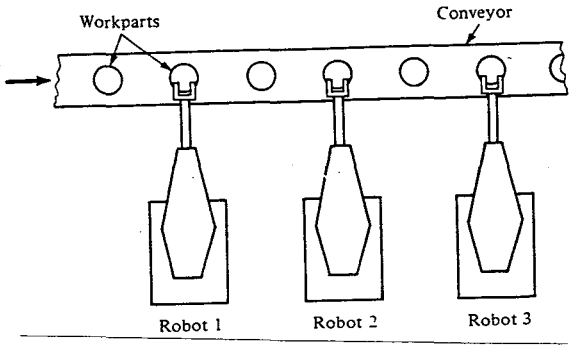


Abb. 2.4b In-line Roboterzellenanordnung

Bewegt sich das Werkstück im Takt ("ruckartig") vorwärts, so läßt sich die Programmierung der Roboter relativ unproblematisch durchführen. Selbst asynchrone Bearbeitung kann man über Sensoren (Lichtschranken etc), die die Ankunft des Werkstücks melden, gut programmieren.
Wesentlich mehr Schwierigkeiten machen Förderbänder, die sich mit konstanter Geschwindigkeit bewegen.

Der Roboter muß das sich bewegende Teil orten, erfassen und innerhalb der Zeit bearbeiten, in der der leere Platz auf dem Fließband durch den für den Roboter erreichbaren Ausschnitt des Fließbands *(tracking window)* sich bewegt. In der Abb. 2.4c ist dies als Schnittmenge zwischen Arbeitsraum des Roboters und Bewegungsraum des Fließbands gezeigt.

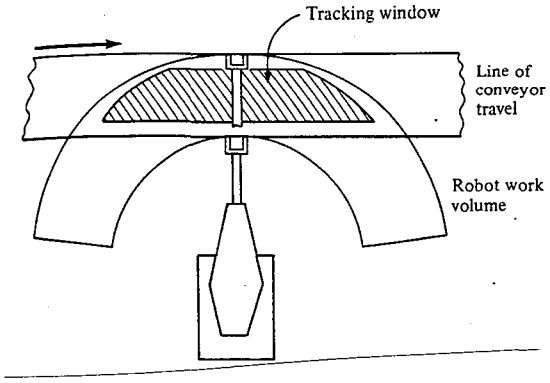


Abb. 2.4c tracking window

Sind die Zwischenräume zwischen den Bearbeitungsmaschinen für einen ortsfesten Roboter zu groß, so kann man ein Leitsystem für die Roboter zwischen den Maschinen installieren. Diese **mobilen Roboterzellen** haben Schienensysteme auf dem Boden (Abb. 2.4d links) oder an der Decke (Abb.2.4d rechts). Bei größeren Abständen, bei spielsweise bei Transportrobotern in Gebäuden, bieten sich eher kontaktlose, im Boden verlegte, elektronischen Leitschienen an.

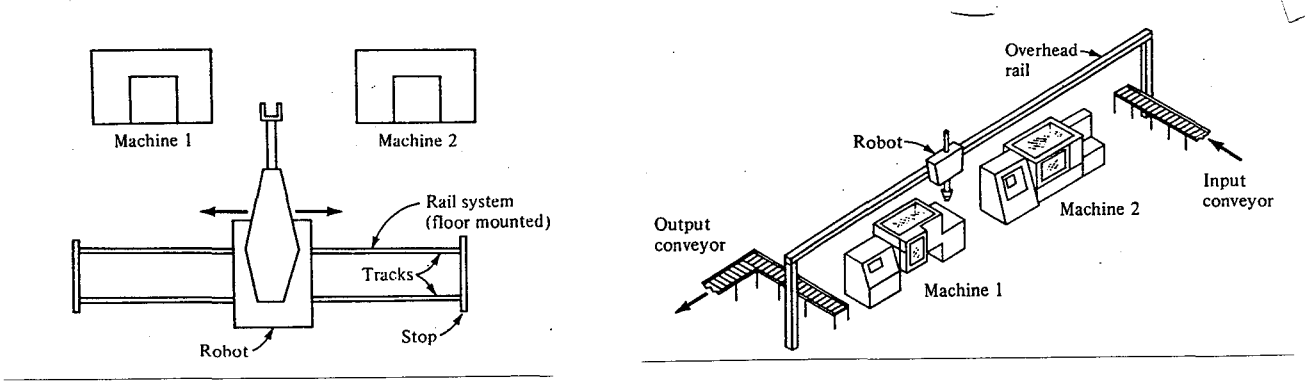


Abb. 2.4d mobile Roboterzellen

Planung der Fertigungszelle und Roboterkontrolle

Bei der Planung der Fertigungszelle versucht man, Zahl und Einsatz der Roboter dem vorgegebenen Bearbeitungsvorgang und den Bearbeitungsmaschinen so anzupassen, daß die Bearbeitungsmaschinen möglichst voll ausgelastet sind und keine Leerlaufzeiten aufweisen. Den Grad dieser *Maschineninterferenz* M_i definiert man zu

$$M_i = \frac{\text{Summe aller Leerlaufzeiten der Bearbeitungsmaschinen}}{\text{Zeit für einen Roboterzyklus}} \text{ [Prozent]}$$

Beispiel: Ein Roboter bedient 3 Maschinen und benötigt zum Be.- und Entladen jeder Maschine 20

Sekunden, so daß er einen Zyklus von 60 sec hat. In diesen 60 sec hat jede Maschine eine Leerphase von 10 sec, so daß $M_i = 30/60 = 50\%$ ist (s. Abb. 2.4e).

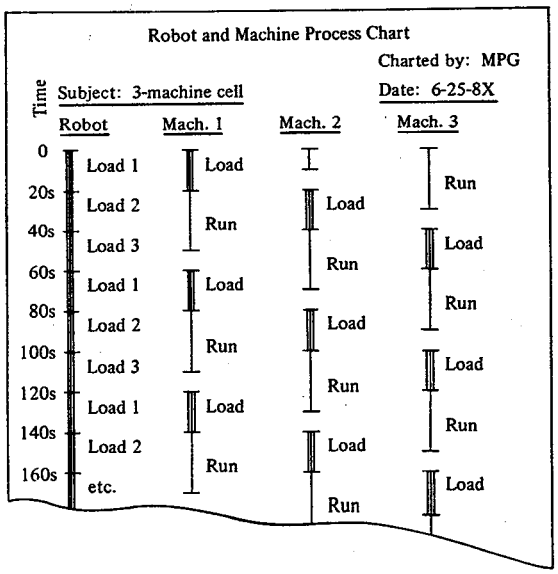


Abb. 2.4e Beispiel der Maschineninterferenz

Die Roboteraktivität wird mit den anderen Aktivitäten (Bearbeitungsmaschinen, Roboter etc) in der Fertigungszelle über ein Interface koordiniert, das typischerweise 10-20 binäre, logische Signale der Robotersteuerung zur Verfügung stellt und ebenfalls 10-20 binäre Ausgänge besitzt. Diese logischen Signale (**Interlocks**) stellen Ereignisse dar, auf deren Eintreten gewartet werden muß oder Fehlermeldungen, die über Interrupt-Service Routinen Reaktionen (Fehlerbehandlung) auslösen.
Realisierte man den ereignisgesteuerten Ablauf früher noch mit Relais und speicherprogrammierten Steuerungen (SPS), so erlaubt die heutige Mikroprozessortechnik komplexere Ereignisse (Botschaften), die vom Zentralrechner der Gesamtproduktion oder von Einheiten der Fertigungszelle seriell über 2-Draht-Leitungen an den Roboter geschickt werden. Dies beinhaltet nicht nur normale Synchronisationsmeldungen, sondern auch Statusänderungen ("jetzt Karosserietyp A153"), Sicherheitsmeldungen ("Absperrzaun wurde geöffnet") und Fehlermeldungen ("Werkstück in der Presse verklemmt"). Programmtechnisch wird die Koordination über WAIT und SEND-Konstrukte gesteuert (s. Abschnitt 6.3). Einen besonders wichtigen Platz bei der Robotersteuerung nimmt die Ausnahmebehandlung (*Exception-handling*) ein. In der folgenden Abbildung sind einige Fehlermöglichkeiten aufgeführt.

Error source category	Particular malfunction or error
1. Tooling	Tool wear-out Tool breakage Vibration (chatter) Tool not present Wrong tool loaded
2. Workpart	Workpart not present Wrong workpart Defective workpart Oversized or undersized part

3. Process	Wrong part program Wrong part Chip fouling No coolant when there should be Vibration (chatter) Excessive force Cutting temperature too high
4. Fixture	Part in fixture (yes or no) Part located properly (yes or no) Clamps actuated Part dislodged during processing Part deflection during processing Part breakage Chips causing location errors Hydraulic or pneumatic failure
5. Machine tool	Vibration Loss of power Power overload Thermal deflection Mechanical failure Hydraulic or electrical failure
6. Robot/end effector	Improper grasping of workpart No part present at pickup Hydraulic or electrical failure Loss of positioning accuracy Robot drops part during handling

Die Reaktionen auf die Fehlerzustände lassen sich in vier Kategorien einteilen:

1) **Behandlung am Zyklusende**
Nur für unkritische Fehler, wie Justierungen etc.
Beispiel: Ein Werkstück fällt auf den Boden

2) **Behandlung im Zyklus**
Der Fehler kann schnell während der normalen Operation beseitigt werden.
Beispiel: Werstück zu groß; muß vor der Bearbeitung geschnitten werden.

3) **Bearbeitungsprozeß anhalten** und Korrekturprozedur einleiten.
Der Fehler ist ernst, kann aber vom Roboter selbst korrigiert werden.
Beispiel: Werkzeugbruch

4) **Bearbeitungsprozeß anhalten** und um menschliche Hilfe rufen
Dies ist dann nötig, wenn Eigenhilfe unmöglich ist oder ein unbekannter Fehler vorliegt.
Beispiel: Hydraulischer Fehler.

Die korrekte Reaktion auf einen Fehler muß vom Programmierer explizit festgelegt werden, was bei moderneren Systemen (ca. 7000 vorgeseheneFehler) einer besonderen Aufmerksamkeit und Anstrengung bedarf. Ist menschliche Hilfeleistung nötig, so kann dies sowohl im Eingreifen bei den Teilen als auch in der Bedienung der Steuer-Terminals der Fertigungszelle geschehen. Andere Aspekte, die bei dem **Design** der Fertigungszelle wichtig sind, sind

- *Anpassung* der in der Fertigungszelle verwendeten Maschinen an die Roboterbedienung
- exakte *Positionierung* und *Orientierung* der vom Roboter verwendeten Teile
- *Identifizierungsmechanismus* für die Werkstücke bei Auslegung auf verschiedene Typen
- *Schutz des Roboters* vor Schmutz, Staub, Korrosionsdämpfen, Lackpartikel etc
- Integration der *notwendigen Installationen* für elektrische Versorgung, Preßluft, Schutzgas, etc.
- *Sicherheitsabsperrung* der Fertigungszelle

3.0 Sensoren und maschinelles Sehen

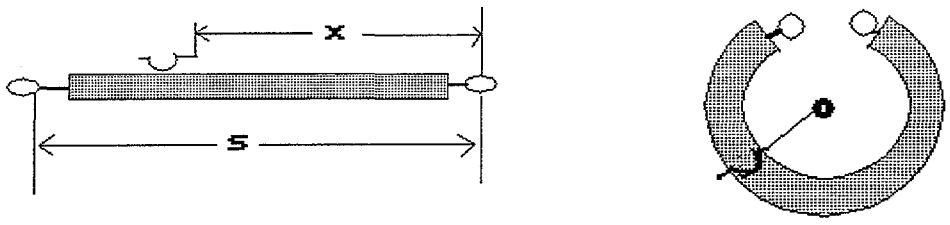
Obwohl die Masse der industriell eingesetzten Roboter heutzutage ohne Sensoren auskommt, zeigt doch die Entwicklung der Roboter, daß für "höher qualifizierte Tätigkeiten", die präzise und mit hoher Wiederholgenauigkeit ablaufen sollen, für die Armführung *interne* und *externe* Sensoren nötig sind. Besteht außerdem die Möglichkeit, berührungslos die Konfiguration der Umgebung zu erfassen, so kann man den Roboter so programmieren, daß er äußeren Hindernissen ausweichen kann und Menschen in seinem Arbeitsbereich dabei nicht gefährlich wird.

3.1 Interne Sensoren

Die Bewegung der einzelnen Segmente des Roboterarms läßt sich nur dann präzise kontrollieren, wenn über Position und relative Bewegungsgeschwindigkeit Sensordaten zum Controller (Mikroprozessor) zurückgekoppelt werden. Je nach aufgenommener Last sind natürlich die dynamischen Reaktionen des Arms unterschiedlich, so daß die "internen " Sensoren indirekt externe Objekte miterfassen. Deshalb kann man die Wirkungsgebiete der internen Sensoren und der externen, zur Steuerung des Roboters innerhalb der Fertigungszelle benötigten Sensoren nicht scharf voneinander abgrenzen.

Positionssensoren

Zur Feststellung der Strecke, die sich ein lineares Gelenk bewegt oder des Winkels, um den sich ein Gelenk gedreht hat, lassen sich klassischerweise **Widerstandsbahnen** einsetzen. Dies sind im linearen Fall blanke Drähte (Konstantan-Draht) mit bekanntem elektrischen Widerstandswert, die parallel zur Bewegungsrichtung angebracht sind und von einem Schleifer, der mit dem bewegten Teil verbunden ist, abgegriffen wird. Wird an den Draht eine Spannung angelegt, so ist die abgegriffene Spannung U_x am Schleifer proportional zur gesuchten Strecke x, siehe Abb. 3.1a.



$x = s\ U_x/U_s$ $\qquad$ $ß = 360^0\ U_x/U_s$

Abb. 3.1a elektrische Messung der Position

Will man die Winkelposition bestimmen, so läßt sich die Widerstandsbahn auch in einem Kreis anordnen (**Potentiometer**).
Diese Art der Positionsmessung hat allerdings zwei Nachteile. Zum einen ist der *mechanische Kontakt sehr unzuverlässig*; mechanische Abnutzung, Oxydation der Widerstandoberfläche, etc stören leicht die Messung. Zum anderen muß die *analoge Form der Messung* für die heutigen, digitalen, mikroprozessorgesteuerten Controller erst durch einen Analog/Digital- Umsetzer (A/D-Konverter) in numerische Werte übertragen werden, was einen zusätzlichen Aufwand und zusätzliche Fehlermöglichkeiten bedeutet.

Unter den anderen, kontaktlosen Messmöglichkeiten hat der **Winkelresolver** einen festen Platz. Er ist ähnlich wie ein Elektromotor aufgebaut und besteht aus einer fest angeordneten Wicklung (*Stator*) und einer darin drehbar angebrachten, zweiten Spule (*Rotor*). Beide Spulen wirken zusammen wie ein lose gekoppelter Transformator, dessen Ausgangsspannung proportional zum Koppelungsgrad (dem Sinus des verdrehten

Winkels) ist. Dieser Meßsensor besitzt zwar im Prinzip keine mechanischen, unzuverlässigen Kontakte, aber das Signal muß wieder A/D transformiert und linearisiert werden.

Zur Positionsmessung werden deshalb heutzutage gerne Sensoren aus der Gruppe der **optischen Enkoder** verwendet. Hierbei wird die Position kontaktlos über Lichtstrahlen direkt digital gemessen.
Ein optischer Enkoder für die absolute Positionsbestimmung besteht aus mehreren, nebeneinander angeordneten Lichtschranken (z.B. LED/ Fotodioden Kombination), durch die ein Streifen Plastik oder Glas geführt wird, siehe Abb. 3.1b.

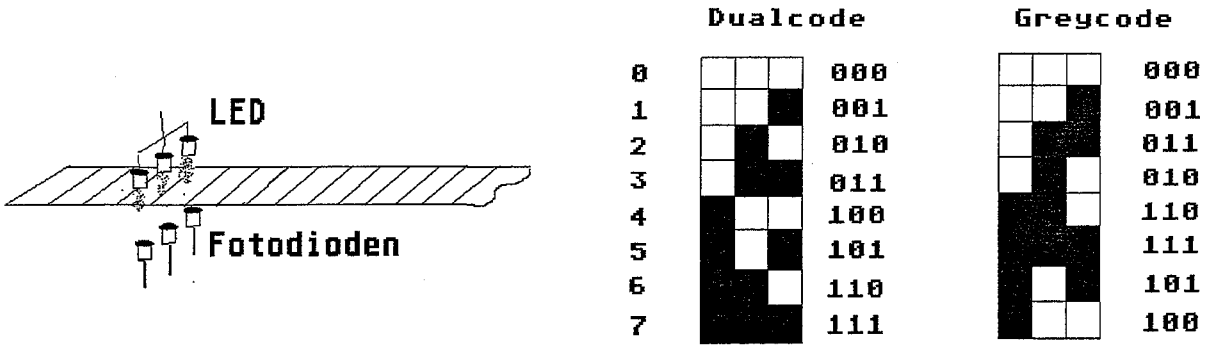


Abb. 3.1b optischer Encoder zur Positionsbestimmung und zwei Positions- Kodes

Die Positionsmarkierungen auf dem Streifen sind in binärer Form durch Schwärzung des transparenten Materials aufgebracht. Üblicherweise wird dafür eine geätzte Glasplatte oder ein auf die Platte geklebter, belichteter Film benutzt. In der obigen Abbildung 3.1b ist rechts das Schwärzungsmuster für acht aufeinanderfolgende Positionsmarkierungen im Dualcode gezeigt. Leider aber ergeben sich bei der Abfrage der Signale der Fotodioden genau dann Schwierigkeiten, wenn eine Position an der Grenze zwischen zwei Zahlen erreicht ist. Geschieht die Positionsabfrage genau in diesem Moment, so können beispielsweise beim Übergang **011** zu **100** Bit 2 noch nicht eins, Bit 1 und Bit 0 dagegen schon null sein. Damit resultiert aber die Fehlmessung **000**. Dieses Problem kann man verhindern, indem man nicht den Dualcode verwendet, sondern den Greycode, bei dem sich immer nur ein Bit in den Kodierungen zwischen zwei benachbarten Positionen ändert. Der mögliche Fehler beschränkt sich damit gerade auf ein Bit.
Biegt man die geraden Streifen wieder zu einem Kreis, so resultieren die **Winkelenkoder** in Abbildung 3.1c

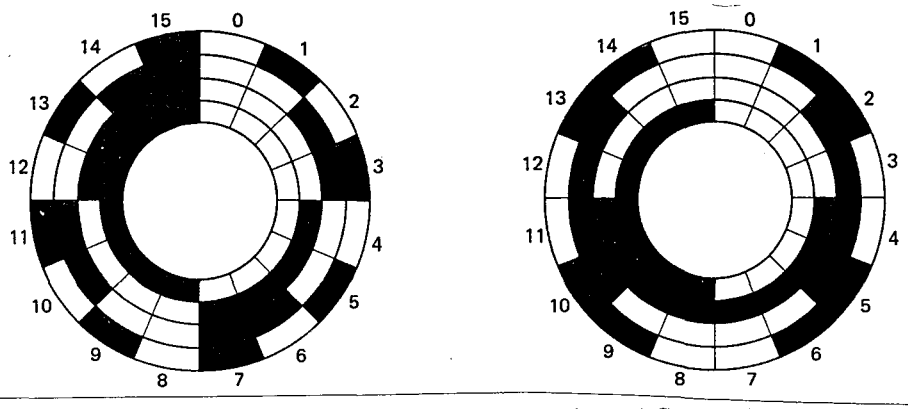


Abb. 3.1c Winkelencoder im Dual- und Greycode

Die absoluten optischen Encoder haben allerdings einen Nachteil: Will man eine Auflösung der Positionsbestimmung von n Bits, so muß man eine Zeile von n Lichtsensoren aufbauen. Bei größerer Zahl von n führt dies zu unerwünschten Leitungs- und Zuverlässigkeitsproblemen. Einen Ausweg daraus bietet die relative oder *inkrementelle Positionsbestimmung.* Bei der optischen Anordnung wird hierbei im Prinzip nur eine einzige Lichtschranke verwendet, die ein regelmäßiges Schwarz-Weiß Muster abtastet. Ein elektronischer Binärzähler registriert jeweils die Helligkeitsänderung, so daß jeder Zahl von Impulsen

eindeutig eine Zahl von Weginkrementen und damit eine bestimmte Wegposition entspricht. Ergänzt man die Anordnung durch eine zweite, leicht verschobene Lichtschranke (s. Abb. 3.1c), so kann man auch feststellen, ob sich das Muster rückwärts bewegt und in diesem Fall den Binärzähler rückwärts zählen lassen.

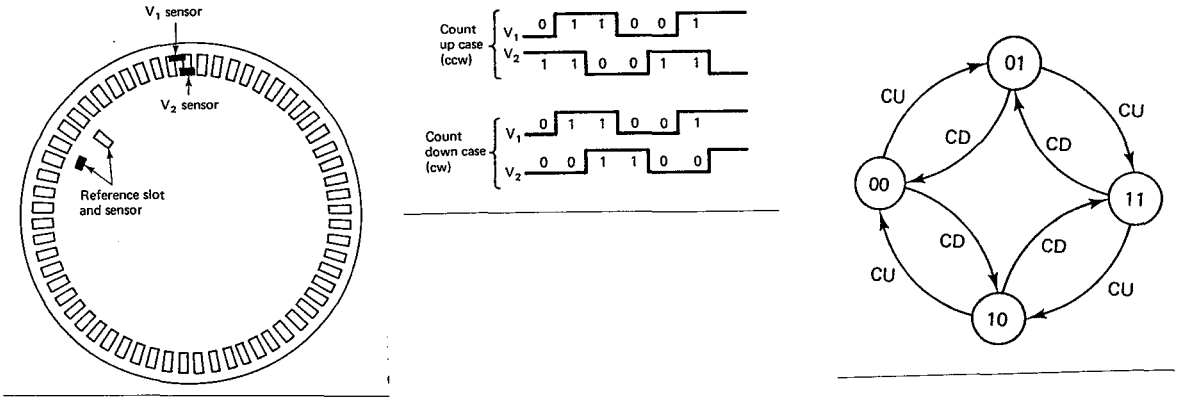


Abb. 3.1c inkrementeller Positionssensor und der Zustandsgraph der Sensorsignale

In der obigen Abbildung 3.1c sind die Phasenbeziehungen der Signale der beiden Fotodioden V_1 und V_2 bei Vorwärts- und der Rückwärtsbewegung sowie der resultierende Zustands-Übergangsgraph gezeigt. Die Zustandbezeichnung ist dabei identisch mit dem Input. Aus diesem Graphen läßt sich leicht die Wahrheitstafel und daraus das Gatternetzwerk ableiten, das zwischen Sensoren und Binärzähler geschaltet werden muß. Ein zusätzlicher Sensor bewirkt ein Referenzsignal, mit dem Zählfehler ausgeglichen werden können.
Der Vorteil der inkrementellen Positionsbestimmung gegenüber der absoluten Positionsbestimmung liegt nicht nur in dem geringeren optomechanischen Aufwand, sondern auch in den geringen Änderungen, die für eine höhere Auflösung nötig sind. Im Prinzip muß man für eine höhere Bitzahl (abgesehen von einem feiner auflösenden Schwarz-Weiß Muster) nicht die Optomechanik mit zusätzlichen Fotodioden verbreitern, sondern nur in der Elektronik den Bitzähler erweitern, was meist weniger Probleme aufwirft.
Der Nachteil der inkrementellen Positionsbestimmung liegt darin, daß am Anfang der Messung die absolute Position definiert werden muß, beispielsweise durch Zurückfahren in eine durch Mikroschalter festgestellte, initiale Ausgangsposition. Fällt während der Arbeit der Strom aus, so müßte ein Roboter mit inkrementellen Positionssensoren sich erst initialisieren, was ziemlich zeitaufwendig und deshalb nicht gern gesehen ist. In einem solchen Fall bewähren sich EEPROM- Speicher oder Batterie-gepuffertes RAM, in das bei Stromausfall neben dem Programmstatus auch die Zählerstände notgespeichert werden können.

Aufgabe: Wie sieht eine Winkelencoder-Scheibe für einen 5-Bit Gray-Code aus?
Wie sieht das Gatter- Netzwerk des inkrementellen Encoders aus?
Wie werden Zählfehler ausgeglichen ?

Motor-sensing

Eine sehr elegante Methode zur Positionsbestimmung liegt darin, die Ansteuerdaten der Motoren auch zur Positionsmessung zu verwenden. Beispielsweise ließe sich der Positionszähler direkt mit der Steuerung des Schrittmotors des Gelenks koppeln. Diese Methode führt allerdings bei den heutigen Robotern zu ungenauen Ergebnissen, da die Ansteuermotoren meist im Sockel des Roboters untergebracht sind und die mechanischen Variabilitäten der Kraftübertragung wie Schlupf, Dehnung, Spiel etc den Zusammenhang zwischen Motorzustand und Armposition sehr kompliziert machen. Ein Ausweg dazu stellt der *direkte Motorantrieb* dar, bei dem der Antriebsmotor direkt im Gelenk sitzt und die Motorachse die Gelenkachse darstellt. Obwohl diese Antriebsform bei Plattenspielern gern benutzt wird, hat sie sich in der Robotik praktisch noch nicht durchgesetzt. Den Haupthindernisgrund stellt die Tatsache dar, daß kräftige Motoren

zu groß und zu schwer sind, um im Arm untergebracht werden zu können, und die kleinen, leichten Motoren für die meisten Anwendungen zu schwach sind. Abhilfe werden in der Zukunft Motoren mit supraleitenden Spulen schaffen.
Ein anderer Ausweg ist bei bestimmten Anwendungen möglich, bei denen man alle Drehachsen des Arms vertikal orientieren kann, so daß die kleinen Motoren nur die Reibung überwinden und nicht das Gewicht der Armsegmente halten müssen.

Geschwindigkeits-Sensoren

In die Kontrolle der Bewegung des Roboterarms beim Beschleunigen und Abbremsen geht oft auch seine Bewegungsgeschwindigkeit ein. Um die reale Geschwindigkeit zu messen kann man entweder die Änderungsgeschwindigkeit der Signale der Positionssensoren benutzen (beispielsweise die Zahl der Inkremente pro sec bei dem inkrementellen Encoder) oder aber direkt Geschwindigkeitssensoren einbauen.
Beispielsweise ist die Spannung, die ein Gleichstromgenerator liefert, proportional zur Drehgeschwindigkeit. Auch hier ist es denkbar, anstelle eines eigenen Sensors gleich die induzierte Gegenspannung zu nutzen, die in den Gleichstrommotoren durch die Bewegung auftritt. Allerdings trifft man auch hier wieder auf das oben erwähnte Problem des motor-sensing: die mechanische Kraftübertragung vom Motor zur Armspitze hat zu viele Nebeneffekte; die Messung wird zu stark verfälscht. Die direkte Messung der Geschwindigkeit ist deshalb ebenfalls erst bei den direkt angetriebenen Robotern der Zukunft möglich.

Kraftsensoren

Im Roboterarm eingebaute Sensoren zur Messung der auftretenden Kräfte gestatten nicht nur, bei zu großen Lasten und Spannungen Ausweich- und Abschaltreaktionen zu programmieren ("Greifer festgeklemmt"), sondern auch die bei der Armbewegung beteiligten Massen vorher zu erfassen und die Bewegungskräfte daran anzupassen.
Es ist nicht sinnvoll, zur Kraftmessung die Energieanforderungen (z.B. den Strom) der Antriebsmotoren zu verwenden, da bei kleinen Lasten die Reibungsenergie durch Kraftübertragung und Bewegungsenergie des Gesamtarms den Energieverbrauch durch die Bewegung der Last weit übersteigen und damit verdecken. Stattdessen ist es sinnvoll, am und im Arm extra Kraftsensoren zu installieren. Da die am Greifer angebrachten taktilen Sensoren im Abschnitt *externe Sensoren* behandelt werden, soll hier nur stellvertretend für viele Systeme der internen Kraftmessung ein universeller Kraftsensor vorgestellt werden, der zwischen Greifer und Arm geschraubt wird und es gestattet, Stauchungen und Verdrehungen des Greifers zu messen.
Der Sensor besteht aus einem soliden Metallring von ca 10 cm Durchmesser, in dem vier viereckige Metallstempel eingelassen sind (Abb. 3.1d)

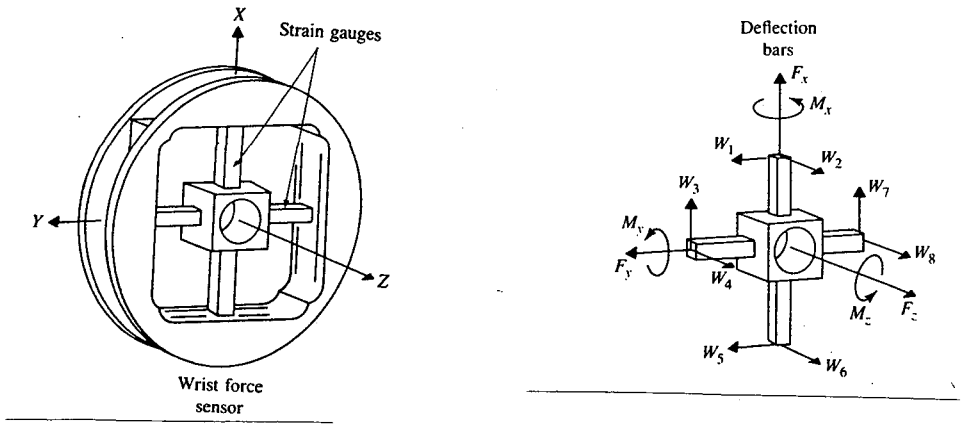


Abb. 3.1c Kraftsensor im "Handgelenk" des Roboterarms

An jedem Metallstempel sind zwei Halbleiter-Dehnungsmeßstreifen angebracht, die bei einer Dehnung eine

dazu proportionale Spannung abgeben. Die insgesamt 8 Spannungen werden digitalisiert und ergeben als Tupel einen 8-dimensionalen Vektor **w**. Alle drei räumlichen Komponenten der auf den Sensor wirkenden Kraft F und des Drehmoments M lassen sich als Linearkombination aus den 8 gemessenen Werten von **w** erschließen. Faßt man F und M zum Vektor **F** aus 6 Komponenten zusammen, so gilt die Gleichung

$$\mathbf{F} = \mathbf{R}\,\mathbf{w}$$

mit der 6x8 Sensor-Kalibrierungsmatrix **R**. Zwar sollten durch die rechtwinklige Anordnung der Meßstreifen bestimmte Koeffizienten der Matrix **R** null sein, aber die endliche mechanische Präzision ergibt in der Praxis immer eine Matrix mit 48 von Null verschiedene Komponenten. Belastet man das Sensorkreuz zur Eichung mit definierten Gewichten in den entsprechenden Richtungen, so erhält man die Komponenten der zu **R** gehörenden, pseudoinversen Matrix **R***

$$\mathbf{w} = \mathbf{R}^*\,\mathbf{F} \qquad \text{mit } \mathbf{R}^*\mathbf{R} = \mathbf{I} \text{ (8x8 Einheitsmatrix)}$$

Die zu **R*** gehörende Matrix **R** kann nun über Iteration mit dem kleinsten quadratischen Fehler bestimmt werden, worauf aber hier nicht näher eingegangen werden soll.
Eine besondere Form der Iteration mit dem kleinsten quadratischen Fehler stellt die stochastische Iteration dar. Verwendet man die topologie-erhaltende Abbildung von Kohonen (s. Kapitel 8.2), so läßt sich die Matrix auch von Neuronennetzen direkt iterativ bei der Benutzung des Armes erlernen.

3.2 Externe Sensoren

Die zu einem Robotersystem gehörenden, in der Fertigungszelle vorhandenen Sensoren werden für folgende Zwecke benötigt

1) Sicherheitsüberwachung der Fertigungszelle
2) Koordination der Zellenaktivitäten (*Interlocks*)
3) Überprüfung von Werkteilen zur Qualitätskontrolle
4) Bestimmung der Position,Orientierung usw von Werkteilen und anderen Objekten (Zulieferautomaten, Ausrüstung, Werkteilen, Menschen etc.) im Roboterarbeitsbereich für
   a) *Identifikation* von Werkteilen
   b) zufällige *Orientierung und Position* von Werkteilen
   c) *genaue Positionierung* des Roboterarms durch äußere, zusätzliche Sensoren der Fertigungszelle

Während die Funktionen 1) und 2) meist nur sehr einfache Sensoren (Schalter, Endkontakte..) benötigt und auch Punkt 3) mit sehr einfachen Meßapparaten durchgeführt werden kann, ist bei der Realisierung von Punkt 4b) meist ein hochwertiges Bilderkennungssystem nötig.
Auch bei der Identifikation von Werkteilen in 4a) kann man meist (aber nicht immer!) auf ein teures und störanfälliges Bildverarbeitungssystem verzichten und einfachere Sensorsysteme (z.B. angeheftete Typenmerkmale: Funkboxen für die Kodierung von Farbe und Schweißprogramm in der Karosseriefertigung, auffällige Farbmarken an Werkteilen, etc) einsetzen.
Im Folgenden sollen nun einige gebräuchliche, externe Sensortypen charakterisiert werden.

Taktile Sensoren

Die taktilen, mechanischen Sensoren sind meist als AN/AUS- Schalter realisiert und werden in verschiedenen Bereichen eingesetzt. Am Greifer montiert dienen sie meist dazu, das Vorhandensein von Objekten zur Steuerung der Greifbewegung zu signalisieren und werden auch zur Fehlerermittlung ("Objekt heruntergefallen?" benutzt. Konstruiert man winzige Matrizen von Kontakten, die auf den beiden Greiferfingern angebracht werden, so läßt sich aus den Signalmustern der beiden Sensormatrizen auf die Oberfläche des gefaßten Gegenstands schließen. In Abb.3.2a wird dies dazu benutzt, die Verkantung eines Werkstücks beim Einsetzen in ein Loch zu erfühlen.

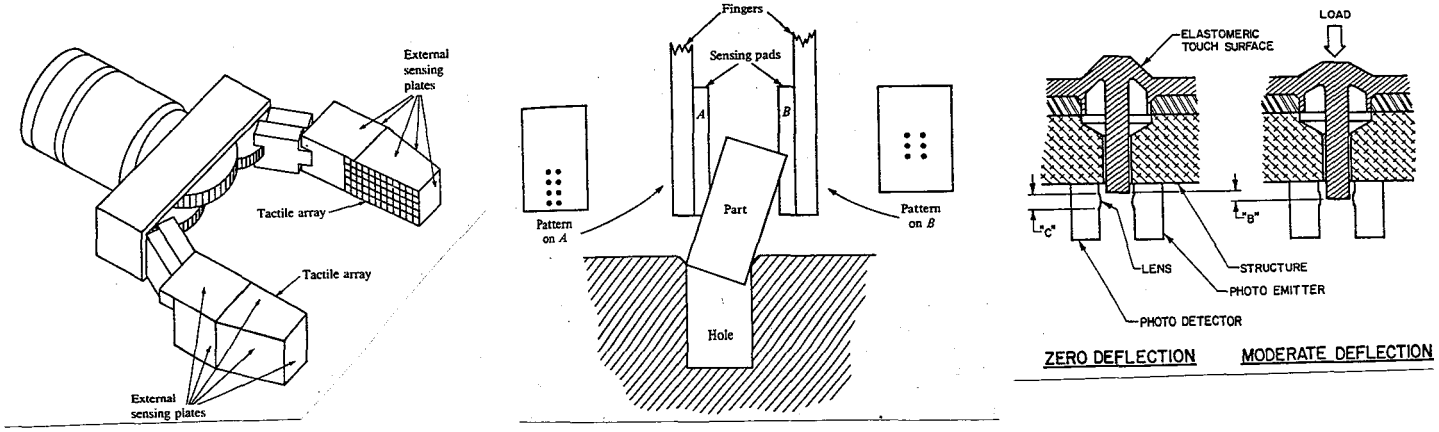


Abb. 3.2a Sensordaten eines verkanteten Werkstücks und taktile Sensormatrix

Rechts daneben ist das Funktionsprinzip eines taktilen Sensors gezeigt, der auf in eine Plastik"haut" eingegossenen, winzigen Lichtschranken beruht. Dabei kann man entweder das Signal des Fototransistors dazu benutzen, mittels einer Schwelle (Schmitt-Trigger) ein binäres AN-AUS Signal zu erzeugen, oder das nicht-lineare Signal als analoges Maß für die einwirkende Kraft benutzen.
Die taktile Matrix läßt sich weiterentwickeln zu einer *künstlichen Haut*, von der in Abb. 3.2b verschiedene Typen dargestellt sind.

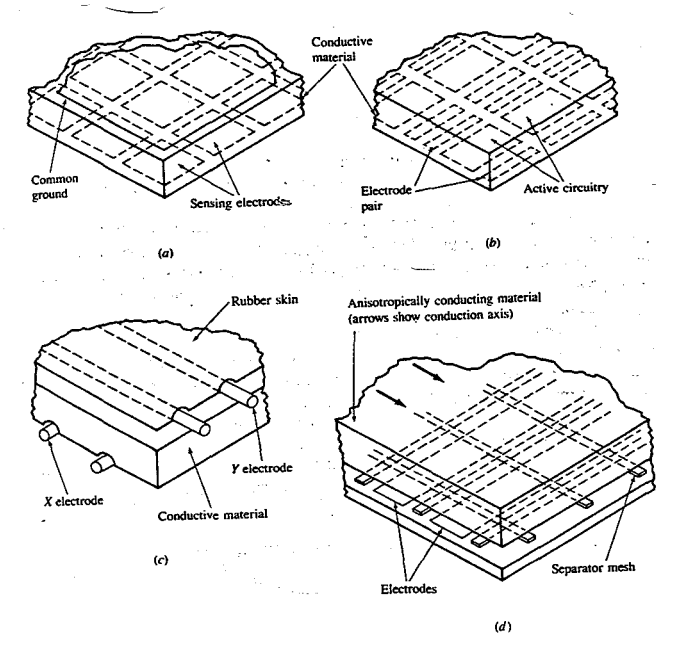


Abb. 3.2b Typen künstlicher Haut

Bei den Typen a) und b) wird für jede Sensorzelle passiv (a) und aktiv (b) der Übergangswiderstand der leitfähigen, drucksensiblen "Haut" gemessen und für jede Sensorzelle einzeln abgeführt.
Demgegenüber wird bei den Typen (c) und (d) die gesamte Matrix von außen abgerastert, indem zeilenweise Spannung eingespeist wird [über Drähte c) oder die Leitungsrichtung eines nur in einer Richtung leitenden, anisotropen Materials d)] und an den Spalten der resultierende Strom bestimmt wird. Allerdings müssen dabei die anderen Zeilen geerdet werden, um unerwünschte Einflüsse von Umwegen zu vermeiden.

Näherungssensoren und Entfernungsmessung

Nähert sich der Greifer einem Werkteil oder einer Maschine, so ist es sehr nützlich zu wissen, wieviel der Greifer vom Ziel noch entfernt ist. Benutzt man **kapazitive Näherungssensoren** (Funktionsprinzip:Verstimmung eines als Sensorplatte herausgeführten Kondensators eines Schwingkreises), so ist die Kapazitätsänderung (bzw. die hervorgerufene Frequenzänderung des Schwingkreises) stark nicht-linear von der Entfernung und vom Material abhängig, s. Abb. 3.2c.

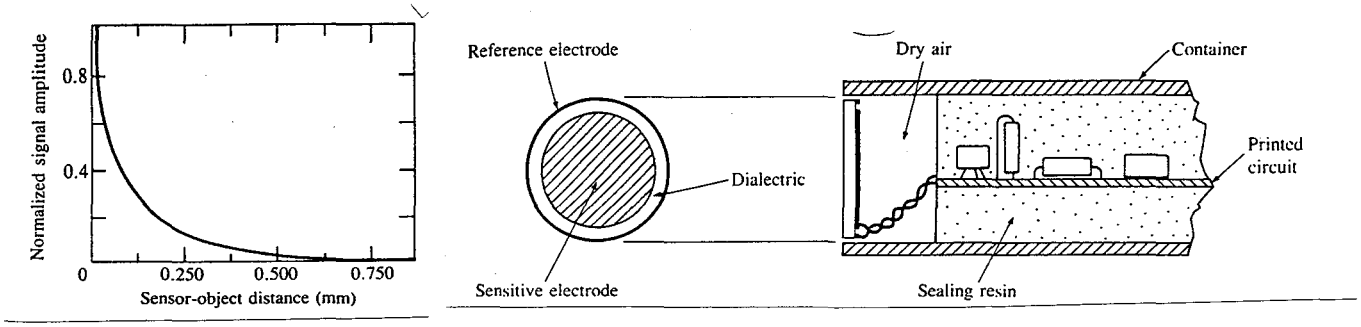


Abb. 3.2c Meßcharakteristik und Aufbau des kapazitiven Sensors

Dies ist auch analog dazu bei dem **induktiven Näherungssensor** in Abb. 3.2d der Fall.

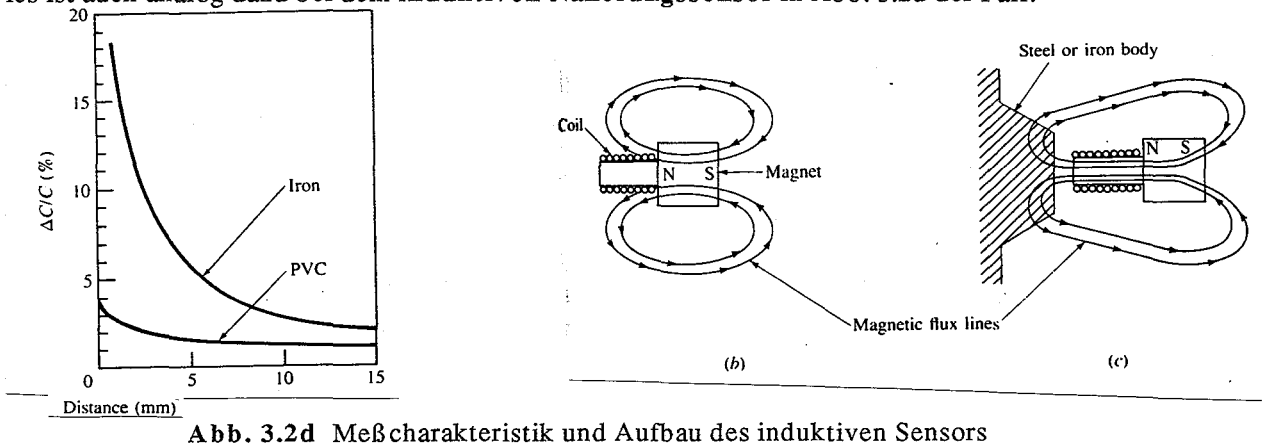
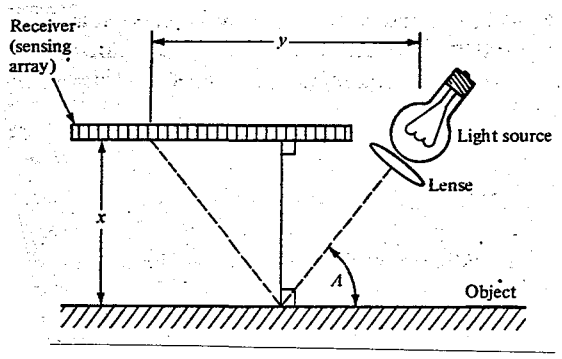


Abb. 3.2d Meßcharakteristik und Aufbau des induktiven Sensors

Deshalb werden kapazitive und induktive Näherungssensoren nur für Schalterfunktionen (Objekt vorhanden -Objekt nicht vorhanden) eingesetzt. Eine optische Version (**optischer Näherunssensor**) benutzt ein Paar Lichtquelle/Lichtsensor, um durch diffuse Rückstreuung vom Objekt die Gegenwart des Objekts festzustellen.
Hat man reflektierende Gegenstände, so kann man durch eine Erweiterung der optischen Anordnung mit einer **Triangulationsmessung** ziemlich genau die Entfernung zum Objekt feststellen. In Abb. 3.2e ist das Funktionsprinzip dargestellt.



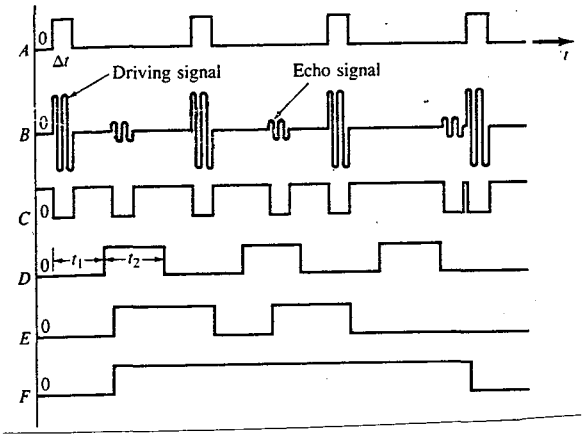
$x = 1/2\ y \tan(A)$

Abb. 3.2e Prinzip der Triangulationsmessung

Ein fokussierter Lichtstrahl (z.B. ein Laserstrahl) fällt auf das Objekt (z.B. ein Spiegel an einem Werkstück) und wird reflektiert auf eine Sensorzeile (z.B. ein CCD-Halbleitersensor). Die gesuchte Entfernung ergibt sich dann nach elementaren, geometrischen Proportionen.
Eine andere, gern benutzte Methode ist die Laufzeitmessung von ausgesandten, hochfrequenten Schall- und Lichtimpulsen. Bei Verwendung von Ultraschall ist die Reflektionsamplitude stark abhängig vom Reflektionskoeffizienten a des Objektmaterials und von der Entfernung d

$$p \sim d^{-1} e^{-ad}$$

Um die frequenzabhängigen Absorptionsmaxima des Materials zu kompensieren, wird meist (z.B. bei der Polaroid-Kamera) ein Frequenzgemisch ("Tschirp") gesendet (Polaroid: 60KHz, 57KHz, 53KHz, 50KHz). Da die Amplitude des Empfangssignals mit steigender Entfernung abfällt (1m:10m = 10^6), müssen Teile der Verstärkungselektronik mit dem Empfangssensor integriert sein. In Abb.3.2f ist die Impulslogik für die Auswertung des empfangenen Signals dargestellt.



Sendeimpuls

Empfangs- und Sendesignal

Binärhülle von B

Erlaubter Bereich (Fenster) für Empfangsimpuls

Laufzeit des Impulses

"E ist auswertbar"

Abb. 3.2f Signallogik der Laufzeitmessung

Verwendet man Laserlichtimpulse, um auch Entfernungen von mehr als 10 m zu messen oder um eine bessere Auflösung (Richtungsselektivität) zu erreichen, so kann man zur Laufzeitanalyse die Interferenz von Sende- und Empfangsstrahl verwenden. In Abb. 3.2g ist die Meßanordnung dargestellt.

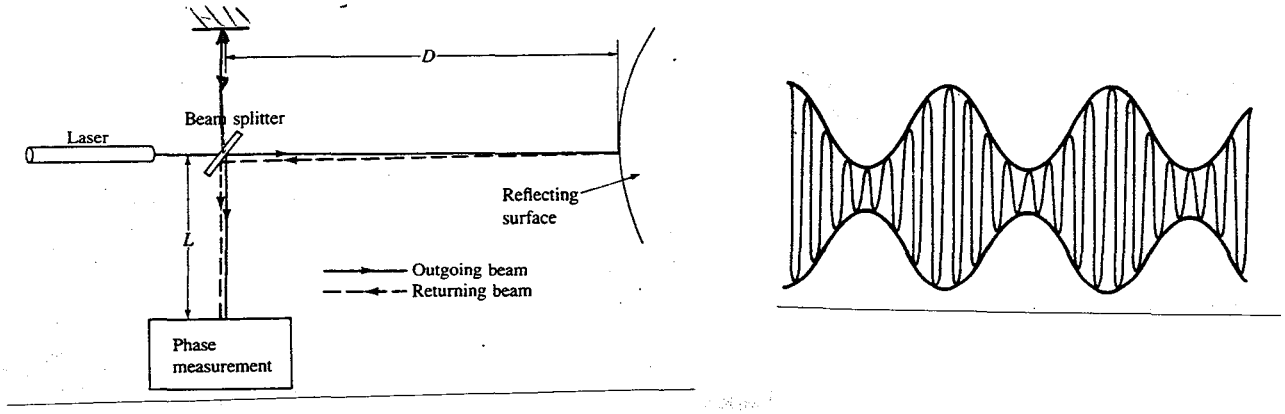


Abb. 3.2g Entfernungsmessung mit amplitudenmoduliertem Laserlicht

Ein Referenzstrahl wird dabei an einem halbdurchlässigen Spiegel vom Meßstrahl ausgekoppelt und dem reflektierten Signal überlagert. Die Intensität der resultierenden Interferenz wird an einem Lichtsensor (Fototransistor) gemessen. Da der Wegunterschied gerade proportional der Phasenverschiebung der beiden Strahlen ist, kann man über die Intensitätsmessung die Phasenlage und damit die fragliche Wegstrecke bestimmen. Bei einem Laserstrahl im sichtbaren Licht (Wellenlänge von ca 0,6 Mikrometer) ist allerdings die Entfernungsmessung auf Multiple der halben Wellenlänge, also 0,3 Mikrometer, beschränkt. Um mit diesen Lasern eine akzeptable Entfernung zu messen, greift man zu einem Trick und moduliert die Lichtamplitude des Lichtstrahls zusätzlich mit einer Schwingung mit großer Wellenlänge (z.B. 30 m bei 10Mhz).
Schaltet man dem Laserstrahl einen rotierenden Spiegel vor, der eine periodische Strahlablenkung bewirkt, so läßt sich damit ein Entfernungsbild gewinnen. In Abb. 3.2h ist links ein solches Bild dargestellt; nähere Punkte sind heller gezeichnet.

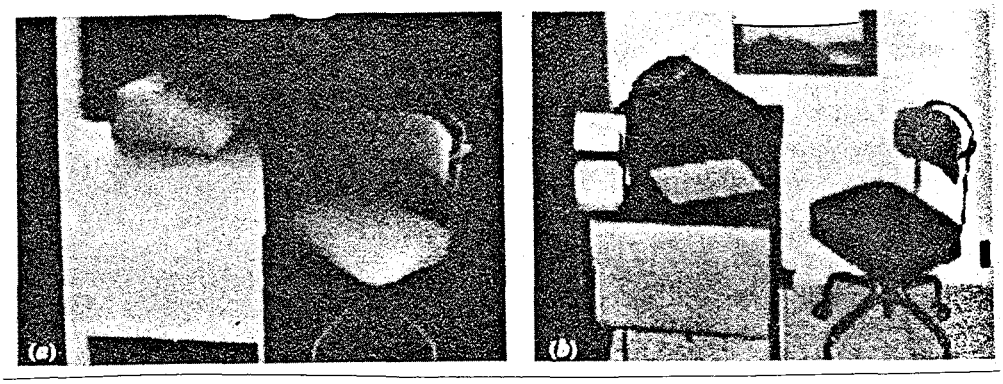


Abb. 3.2h a) Entfernungsbild b) Helligkeitsbild der selben Szene

Rechts daneben ist die gleiche Szene als normales Helligkeitsbild einer Fernsehkamera gezeigt. Interessanterweise enthält das Entfernungsbild Informationen, die dem Helligkeitsbild nicht zu entnehmen sind (*welche?*) und vice versa.

3.3 maschinelles Sehen

Im Unterschied zu der allgemeinen Bilderkennung, die Ansätze und Algorithmen zur Erkennung von allgemeinen, komplexen Objekten (Texturen!) behandelt, sollen hier nur die einfachen, robusten Ansätze vorgestellt werden, um die relativ einfachen Aufgaben der Robotik zu lösen.
In der Robotik haben nur wenige, einfache Verfahren Bedeutung, da die Aufgabenstellung nur wenige Typen bestimmter Werkstücke oder bestimmte, vorher festgelegte Bildkriterien (Qualitätskontrolle) vorsieht.
Maschinelles Sehen mit Fernsehkamera und analysierendem Computer wird immer dann interessant, wenn viele Variationen eines Werkstücks bearbeitet werden sollen, deren Lage (Position und Orientierung) nur schwer durch den vorherigen Bearbeitungsprozeß festgelegt werden können. In Abb. 3.3a ist die Kostensituation für solche Systeme mit und ohne Bilderkennung (*vision systems*) zu sehen.

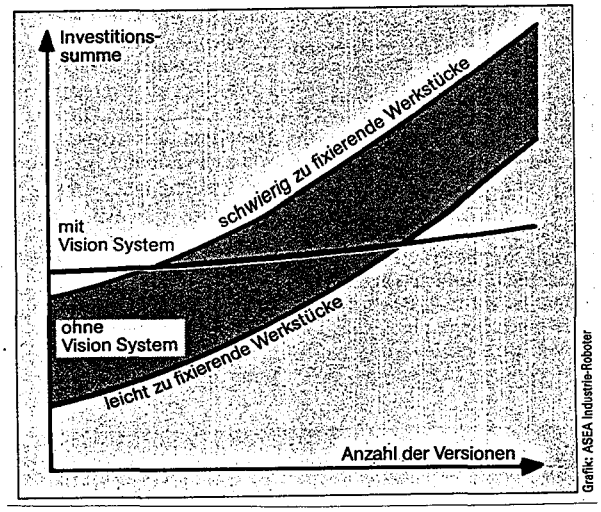


Abb. 3.3a wirtschaftlicher Einsatz von Bilderkennungssystemen

Ein Bilderkennungssystem besteht grundsätzlich aus folgenden Funktionsblöcken:

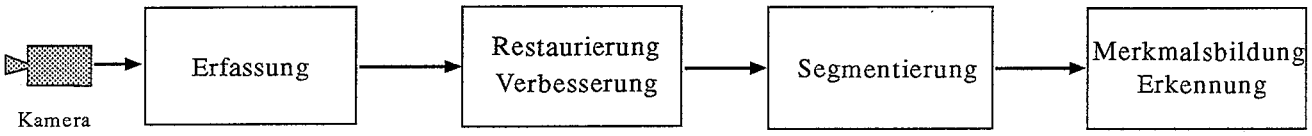


Abb. 3.3b Bilderkennungssystem

Von einer Videokamera wird ein Bild **erfaßt**, die Helligkeitswerte digitalisiert und in einem Speicher abgelegt. Aufgabe der **Bildverbesserung** ist es dann, Digitalisierungsfehler und andere Fehler der Bildgewinnung (Rauschen, Kontrastminderung etc) auszugleichen (*Digitale Filterung*). Dies läßt sich beispielsweise mit *globalen*, auf allen Bildpunkten (*Pixeln*) wirkenden Operatoren wie Fouriertransformation und Grauwerttransformation oder *lokalen*, nur auf wenigen Pixeln einer Nachbarschaft definierten Operatoren (Mittelwertoperator, Medianfilter etc) bewerkstelligen.
Obwohl die Fouriertransformation durch schnelle Algorithmen und Prozessoren als Filteroperation relative populär ist, beinhaltet sie doch als globale Operation einige Probleme. Unterdrückt man beispielsweise hohe Frequenzen, d.h. alle feinen Helligkeitsstrukturen im Bild, so unterdrückt man mit dem Bildrauschen auch erwünschte Objektkanten im Bild. Verwendet man einfache, lokale Operationen wie die Mittelwertbildung (jeder Pixelwert wird als Mittelwert seiner Umgebung ermittelt) so verwischt dies ebenfalls die Bildkonturen. Ein Ausweg für dieses Problem bietet die Medianfilterung, bei der jeder Pixelwert durch den

Median seiner Umgebung ersetzt wird. Der Median wird aus der nach Werten geordneten Pixelmenge von n Pixeln als der Wert des n/2-ten Pixels bestimmt. Dies entspricht einer Betrachtung, die den wahrscheinlichsten Wert für ein Pixel aus seiner Umgebung heraussucht: Störungen, die sehr große und sehr kleine Pixelwerte erzeugen, werden ignoriert. In Abbildung 3.3c ist in a) ein Orignalbild, in b) die verrauschte Version, in c) die mit dem Mittelwertoperator und in d) mit dem Medianoperator gefilterte Version gezeigt.

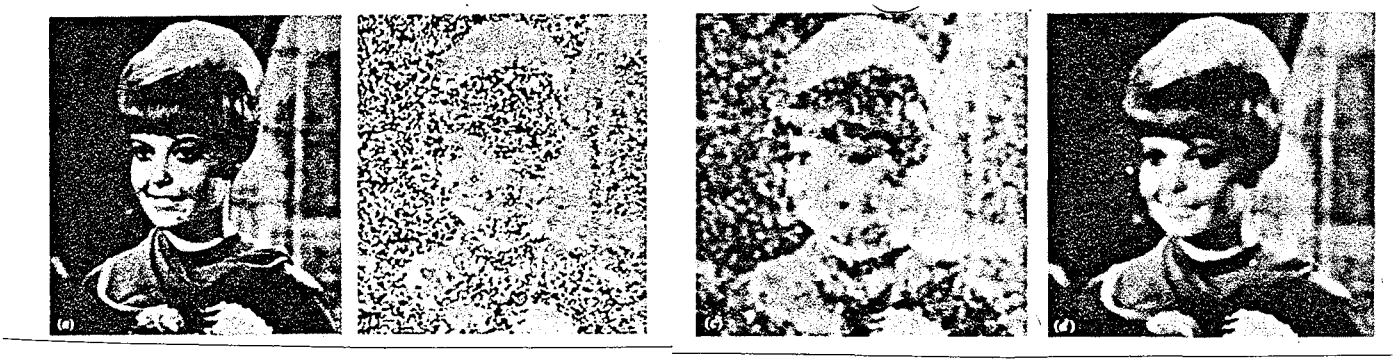


Abb. 3.3c Orignalbild, verrauschte, mittelwertgefilterte und mediangefilterte Version

Bei der **Segmentierung** gibt es verschiedene Möglichkeiten der Kantenreduktion. Lokale Operatoren wie die *Gradientenoperatoren* (eine starke Helligkeitsänderung entspricht einer Kante) z.B. Roberts- und Sobeloperator) haben den Nachteil, daß unterbrochene und unvollständige Kanten nicht als solche erkannt werden. Berücksichtigt man bei der Entscheidung "Kante" für ein Pixel auch die Kantenentscheidungen der Nachbarpixel mit, so erhält man *Relaxationsverfahren*, die zwar bessere Ergebnisse bringen, aber auch aufwendiger sind.
Ein anderer Ansatz besteht darin, anstelle der Kanten die angrenzenden Flächen (*Regionen*) zu bestimmen. Hat man ein Kriterium für die Einheitlichkeit *(Homogenität)* einer Region definiert, so lassen sich bei gegebener Regionenaufteilung eines Bildes einige Regionen bei Inhomogenität iterativ in Unterregionen zerteilen oder mit gleichen, homogenen Nachbarregionen verschmelzen (*Top down* bzw. *Bottom up*). Die Kombination beider Methoden wird als *Split-and-merge* bezeichnet.
Leider kann die Segmentierung bei kontrastreicher Oberflächenstruktur (*Textur*) schnell versagen. In Abb. 3.3d ist als Beispiel ein mit Textur gezeichneter Würfel gezeigt, dessen Flächenkanten mit den genannten Verfahren nicht direkt bestimmt werden können (*Warum nicht?*).

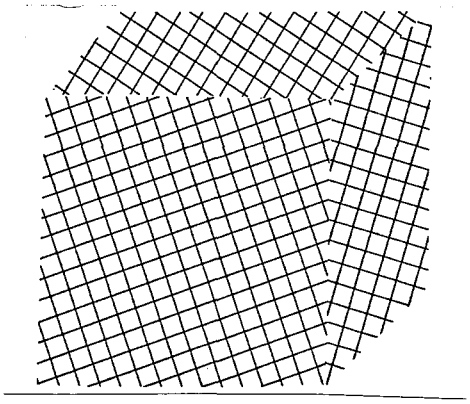


Abb. 3.3d Würfel aus Textur

In diesem Fall muß man zuerst Verarbeitungsstufen für das Erkennen und Egalisieren der Textur vor die Segmentierung setzen.

Ist dasd Bild ausreichend segmentiert, so müssen durch **Merkmalsbildung** typische Elemente (*features*) wie Kanten, Ecken etc bestimmt und in der **Objekterkennung**sphase als Elemente von bereits gespeicherten Objekten wiedererkannt werden. In Abb. 3.3e ist eine solche Verarbeitungsfolge (Bilderfassung, Segmentierung, Merkmalsbildung und Objekterkennung) zu sehen.

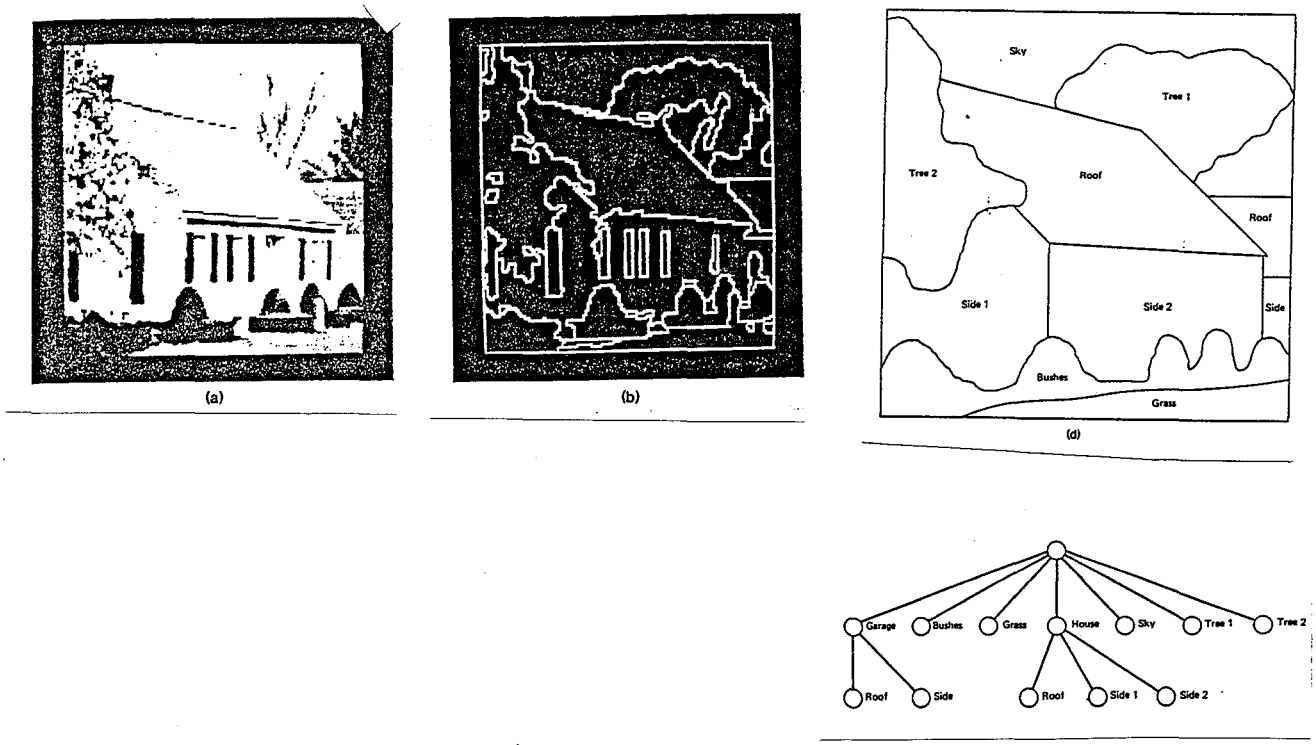


Abb. 3.3e Erkennung eines Hauses

Im Unterschied zu den allgemeinen Merkmalen in Abb. 3.3e wird bei der industriellen Robotersteuerung nur die Erkennung einer kleinen, begrenzten Anzahl von Werkstücken benötigt. Da die Werkstücke bei manchen Anwendungen immer in der gleichen Entfernung präsentiert werden, lassen sich (je nach Anwendungsfall) folgende Maße als features verwenden:

- **Grauwert** (Maximum, Minimum, Mittelwert)
- **Fläche**
- **Umfang**
- **min. objektumfassendes Rechteck**
- **Schwerpunkt** = 1/n (Summe aller x-Werte, Summe aller y-Werte) der Pixel (x,y) des Objekts
- **Exzentrizität** = $\frac{\text{max. Länge einer Sehne A}}{\text{max. Länge der Sehne B}}$ wobei A $\perp$ B
- **Bildverhältnis** **Länge : Weite** eines objektumfassenden Rechtecks
- **Schmalheit** z.B. $\frac{(\text{Umfang})^2}{\text{Fläche}}$ oder $\frac{\text{max.Durchmesser}}{\text{Fläche}}$
- **Momente** r s-ter Ordnung Summe $x^r y^s$ über alle Pixel des Objekts (Rotations-, Translations-, Skalierungs- invariant !)
- **Zahl der Löcher**

Kodierung der Form

Abgesehen von der Beschreibung durch den Flächeninhalt ist auch eine Bechreibung über die Form möglich. Eine sehr einfach zu erstellende Möglichkeit bietet die **Kettenkodierung** (*chain code)* des Objekts. Dazu legt man über die Objektform ein regelmäßiges Gitterraster (s. Abb. 3.3f) und beschreibt für jede Gitterzelle, in welcher Richtung darin die Linie des Objektumfangs verläuft. Die möglichen 4 oder 8 Richtungen sind dabei durch Zahlen kodiert.

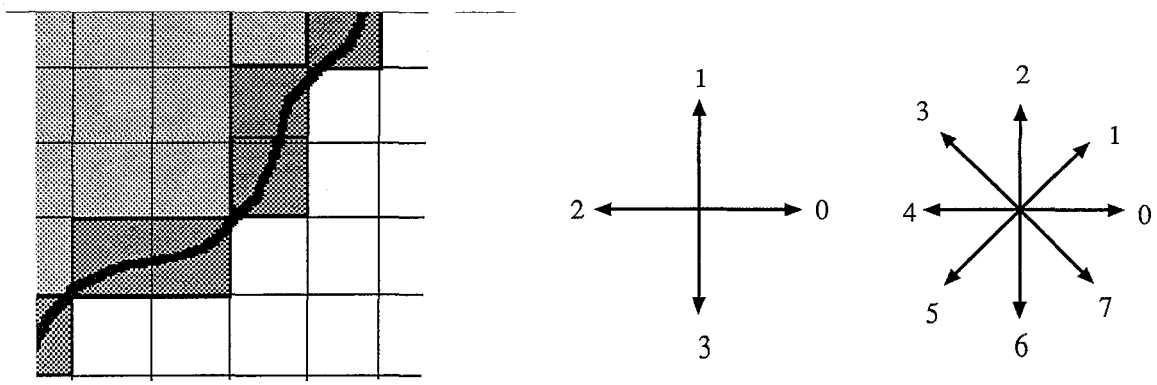


Abb. 3.3d Kettenkodierung einer Objektform

Wählt man sich einen Startpunkt, so erhält man als Beschreibung der Objektform eine Kette von Zahlen. Die Länge der Kette ist dabei durch die Zahl der durchlaufenen Zellen gegeben.
Möchte man die Form unabhängig von der Größe kodieren, so empfielt es sich, jede Form durch ein Polygon mit gleicher Anzahl von Stützpunkten und gleichlangen Polygonzügen zu approximieren. Jedem Stützpunkt ist dann eine Zelle zugeordnet.
Betrachtet man anstelle der absoluten Richtungen nur die Richtungsänderungen (Zahl der Stufen gegen den Uhrzeigersinn), so läßt sich durch diese Umkodierung der Kettenkode invariant gegenüber Rotationen des Objekts machen. Vermerkt man als zusätzliches Kettenglied die Richtungsänderung zwischen Start- und Endpunkt, so resultiert eine in sich geschlossener Kettekode (zykl. Kode), der unabhängig vom Startpunkt ist. Rotiert man die Kette, bis sie als numerische Zahl den kleinsten möglichen Wert hat, so erhält man eine Zahl, die bei gegebener Kettenlänge eineindeutig einer Objektform entspricht. Diese Zahl wird als *Formzahl* (*shape number*) bezeichnet.

4.0 Räumliche Beschreibungssysteme und Transformationen

Mit welchen Methoden und Algorithmen läßt sich die Bewegung der Roboterarme beschreiben und programmieren?
Betrachten wir dazu die Situation eines Manipulators (Greifarms) in einer Fertigungszelle. Um die relative Lage aller Maschinen und Roboter zueinander einheitlich zu beschreiben (globaler Adressraum), sei als Welt- oder Basiskoordinatensystem der Fertigungszelle das Kartesische Koordinatensystem zugrunde gelegt. Der Greifer am Ende des Roboterarms hat zweckmäßigerweise sein eigenes Koordinatensystem, das aus dem Annäherungsvektor **a**, dem Vektor der Greiferorientierung **o** und dem zu beiden orthogonalen Normalvektor **n** besteht.

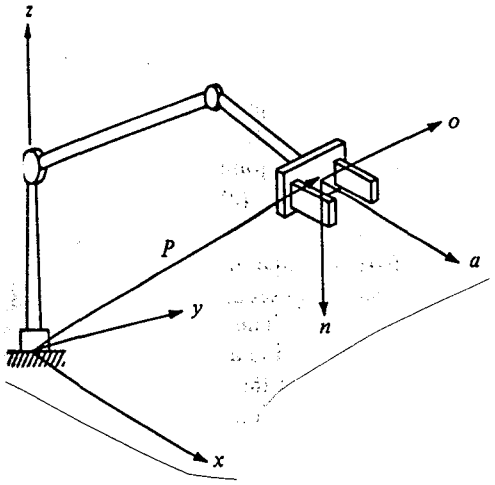


Abb. 4.0a Basis- und Greiferkoordinatensystem

Wie läßt sich nun das Greifersystem im Basiskoordinatensystem beschreiben? Zweifelsohne ist die Lage des Greifers durch die Lage der Segmente des Roboterarms festgelegt. Je nach Robotertyp läßt sich die Lage der Segmente nach Gelenkwinkeln, Translationslängen etc beschreiben.

4.1 Roboterarme, Gelenk- und Bewegungstypen

Für eine einheitliche Lagebeschreibung der Segmente ist es nötig, die Art der Gelenke und Segmente näher zu betrachten.

Gelenktypen
Man unterteilt die Gelenke bzw Segmente in 4 verschiedene Typen, die mit einem der Buchstaben L, T, R oder V kodiert werden.

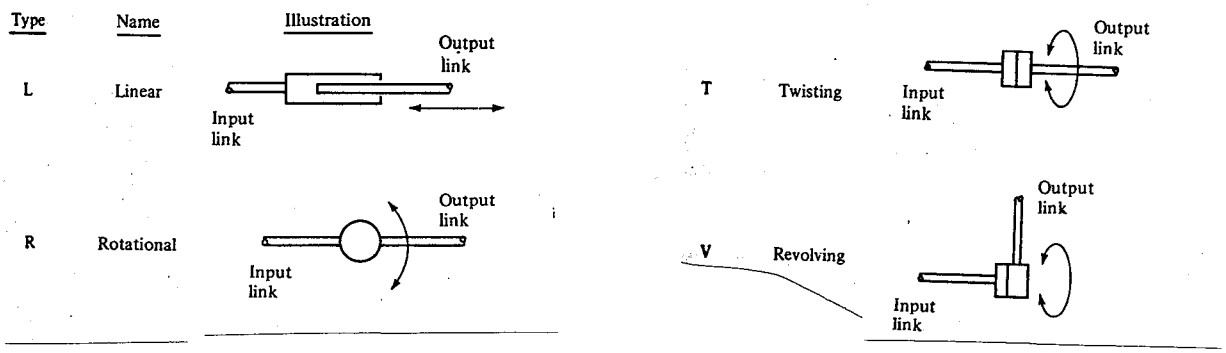


Abb. 4.1a Gelenktypen

Jedes Gelenk addiert einen Freiheitsgrad zu dem Roboterarm, was aber nicht unbedingt auch eine neue Bewegungsdimension bedeutet. Mit den obigen Bezeichnungen lassen sich die Robotertypen in Abschnitt 2 auch anders beschreiben:

	Typ
Kartesischer Roboter :	LLL
Zylindrischer Roboter:	LVL
Polarer Roboter:	TRL
armähnlicher Roboter:	TRR

Im Vergleich dazu hat der menschliche Arm ebenfalls 3 Freiheitsgrade: 2 durch eine R-V Kombination im Schultergelenk und einen im Ellbogengelenk vom Typ R.
Die Greifer besitzen normalerweise 3 Freiheitsgrade, um Werkstücke in einer beliebigen Greiferorientierung zu erfassen.

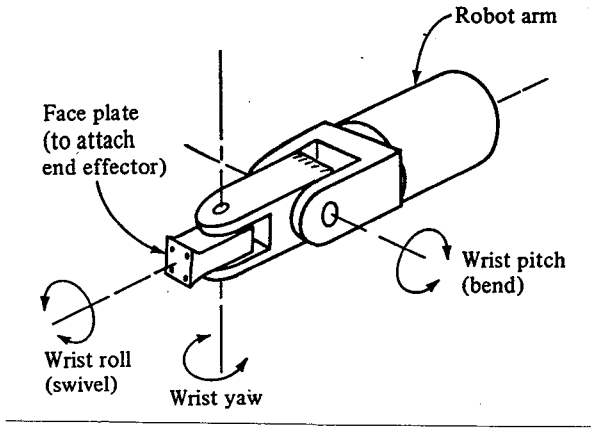


Abb. 4.1b Freiheitsgrade des Greifers

Beim Menschen entspricht dies den 3 Freiheitsgraden der Drehbewegung der Hand (*roll*), dem Abkippen (z.B. beim Winken) (*pitch*) und dem seitlichen Abwinkeln (*yaw*) der Hand. Wie man bei der Hand leicht merkt, sind nicht alle Winkeleinstellungen möglich; die Bewegung ist begrenzt. Dies gilt auch für Roboterarme. Es gibt also sowohl beim menschlichen als auch beim Roboterarm im Arbeitsraum Kombinationen von Koordinaten und Orientierung der Hand, die nicht erreicht werden können.

Bewegungstypen

Die aus der Folge von verschiedenen Gelenken und Segmenten bestehenden Roboterarme sind - je nach Kontrolle- zu verschiedenen Bewegungsformen in der Lage. Die wichtigsten drei sind

1) Roboter mit begrenzten Sequenzen *(limited sequence robots)*

Diese einfachste Form von Roboterarmen kommt ohne Rückkopplung (Sensoren und Servomotoren) und fast ohne Computerkontrolle aus. Die Bewegung der Armsegmente wird in einer festgelegten Sequenz durchgeführt; jede Bewegung dauert so lange, bis ein mechanischer Haltepunkt oder ein elektrischer Endabschalter erreicht wird. Die Programmierung erfolgt also sowohl durch die Festlegung der Bewegungssequenz als auch durch die mechanische Einstellung der Bewegungsendpunkte. Da bei dieser Steuerung nur wenige Bewegungssequenzen und kein dynamisches Feedback existiert, eignet sich dieser Robotertyp vor allen Dingen für einfache, pneumatisch betriebene Verlade- und Transportaufgaben.

2) Roboter mit Bewegungswiederholung *(playback robots)*

Bei diesem Robotertyp erfolgt die Programmierung durch "Vormachen" der gewünschten Bewegung. Beispielsweise kann man mit einem (genügend) leichten Roboterarm die gewünschte Bewegung durchfahren und die Sequenz der Positionen intern aufzeichnen lassen. Der Roboter wiederholt dann mit einer gewünschten Geschwindigkeit die Bewegung, wobei für eine zufriedenstellende Reproduktionsgenauigkeit Positionssensoren und eine aufwendigere, meist digitale Steuerung notwendig ist.
Die Bewegung selbst kann (je nach Anwendung) entweder eine grobe Punkt - zu - Punkt Bewegung sein (Maschinen be- und entladen, Transportaufgaben, Punktschweißen etc), wobei jeder Punkt als Koordinate mit den dort auszuführenden Arbeitssequenzen gespeichert ist, oder aber eine gleichförmige Bewegung für kontinuierliche Arbeiten wie Flächenlackierung, Schweißnähte ziehen, etc. Der Programmierer gibt nur die Anfangs- und Endpunkte der Bewegung ein; die Punkte dazwischen müssen vom Controller errechnet und in die notwendige Maschinensteuerung umgesetzt werden. Dabei muß man beachten, daß eine geradlinige Bewegung für einen Roboter mit Drehgelenken etwas absolut "unnatürliches" darstellt und deshalb ziemlich viel Rechenaufwand erfordert. Eine zusätzliche Real-Time Bedingung an den Controller wird durch die Forderung gestellt, daß bei der ruckhaften Bewegung von einem errechneten Zwischenpunkt zum nächsten keine Schwingungen auftreten dürfen. Nehmen wir eine typische Resonanzfrequenz von 5 Hz an, so muß jeder Zwischenpunkt ca 10 mal schneller ausgerechnet werden, also mit 50Hz oder alle 20 msec.

3) Intelligente Roboter

Fügt man zu den Fähigkeiten der Roboter der vorigen 2. Stufe die Möglichkeit hinzu, interaktiv fehlende Daten selbst zu besorgen oder aus Sensordaten logische Schlüsse zu ziehen, so erhält man die zukunftsträchtige Gruppe der "intelligenten" Roboter. Die Programmierung dieser Roboter geschieht mittels Hochsprachen, englisch-ähnlichen oder symbolischen Sprachen. Die zur Erreichung des Ziels notwendige Bewegung setzt sich aus Bewegungselementen zusammen, deren Prozeduren und Daten schon früher eingegeben wurden und bekannt sein müssen.
Typische Anwendungen dieser Robotergruppe beinhalten Schweiß- und Montagearbeiten.

4.2 Koordinatentransformationen

Die Bewegung der linearen Gelenke und der Drehgelenke lassen sich durch Translationen und Rotationen beschreiben. Mit diesen elementaren Operationen lassen sich somit die Transformation der Segmentkoordinaten durchführen.

Translation
Der Punkt (x,y,z) wird auf den neuen Punkt (x', y', z') durch eine einfache Addition erreicht:

$$(x', y', z') = (x,y,z) + (d_x, d_y, d_z)$$

$$\mathbf{x'} = \mathbf{x} + \mathbf{d}$$

Rotation
Betrachten wir die Rotation eines Vektors $\mathbf{r} = (x, y, 0)$ um die z-Achse.

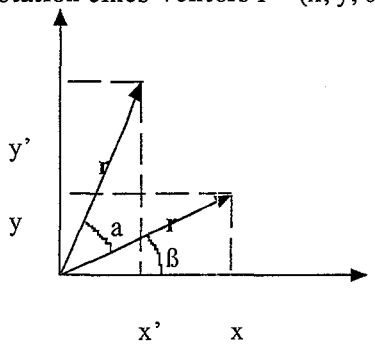


Abb. 4.2a Rotation

Da die Länge des Vektors gleich bleibt, gilt bei der Rotation in positiver Richtung (gegen den Uhrzeigersinn)

$$x' = r \cos(a+ß) \qquad y' = r \sin(a+ß) \qquad (4.2a)$$

$$x = r \cos(ß) \qquad y = r \sin(ß) \qquad (4.2b)$$

Mit der Beziehung

$$\cos(a+ß) = \cos(a)\cos(ß) - \sin(a)\sin(ß)$$
$$\sin(a+ß) = \sin(a)\cos(ß) + \cos(a)\sin(b)$$

und Einsetzen von (4.2b) wird (4.2a) zu

$$x' = x \cos(a) - y \sin(a) \qquad (4.2c)$$
$$y' = x \sin(a) + y \cos(a)$$

Dies läßt sich als Matrizenmultiplikation mit der Matrix **R** formulieren:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad (4.2d)$$

$$\mathbf{x'} = \mathbf{R}(z,a)\,\mathbf{x}$$

Entsprechende Matrizen ergeben sich für die Rotation um die x- und y-Achse:

$$\mathbf{R}(x,a) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) \\ 0 & \sin(a) & \cos(a) \end{pmatrix}$$

$$\mathbf{R}(y,a) = \begin{pmatrix} \cos(a) & 0 & \sin(a) \\ 0 & 1 & 0 \\ -\sin(a) & 0 & \cos(a) \end{pmatrix}$$

Eine räumliche Rotation um einen Winkel ß mit den Koordinaten $ß_x, ß_y, ß_z$ läßt sich durch Hintereinanderausführen von Rotationen in x, y und z- Richtung erreichen, wobei allerdings die Koordinatenachsen nicht mitgedreht werden dürfen.

$$\mathbf{R}(ß)\,x = \mathbf{R}_x(ß_x)\,\mathbf{R}_y(ß_y)\,\mathbf{R}_z(ß_z)\,x$$

Durch Ausmultiplizieren der drei Matrizen der rechten Seite (Assoziativgesetz) erhalten wir die allgemeine Rotationsmatrix **R**(ß). Mit der Abkürzung *c* für cos und *s* für sin lautet sie

$$\mathbf{R}(ß) = \begin{pmatrix} cß_z cß_y & cß_z sß_y sß_x - sß_z cß_x & cß_z sß_y cß_x + sß_z sß_x \\ sß_z cß_y & sß_z sß_y sß_x + cß_z cß_x & sß_z sß_y cß_x - cß_z sß_x \\ -sß_y & cß_y sß_x & cß_y cß_x \end{pmatrix} \qquad (4.2e)$$

Drehen wir dagegen die Koordinatenachsen mit, so läßt sich eine räumliche Drehung auch durch mehrmalige Rotation um die gleiche Achse erreichen. Die spezielle Rotation $\mathbf{R}(ß) := \mathbf{R}_z(ß_x)\,\mathbf{R}_x(ß_y)\,\mathbf{R}_z(ß_z)$ wird als *Euler-Rotation* bezeichnet.

Homogene Transformation

Jede Rotation eines Gelenks läßt sich durch eine Matrizenmultiplikation ausdrücken. Damit ist es möglich, die Wirkung von mechanischem Hintereinanderschalten einiger Rotationsgelenke der Typen T, R und V durch das Ausmultiplizieren der zu den Gelenken gehörenden Matrizen auszurechnen. Allerdings stört in diesem harmonischen Bild die Translation. Wenn es möglich wäre, auch die Vektoraddition durch eine Multiplikation auszudrücken, so wäre die Form der Koordinatentransformation für alle Gelenkarten gleichförmig, also **homogen.**
Dies läßt sich durch einen Trick erreichen. Wir erweitern die 3-dimensionalen Punktkoordinaten um eine vierte Komponente, die wir zunächst eins setzen.

$$(x, y, z)^T \longrightarrow (x, y, z, 1)^T$$

Damit läßt sich die Translation schreiben als

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad (4.2f)$$

$$\mathbf{x'} = \mathbf{T}(\mathbf{d})\,\mathbf{x}$$

Die Matrix der Rotation **R** (ß) aus (4.2a) wird ebenfalls um eine vierte Zeile und Spalte erweitert, die allerdings bis auf R_{44} Null bleibt.

$$\mathbf{R}(ß) \longrightarrow \begin{pmatrix} & & & 0 \\ & \mathbf{R}(ß) & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4.2g)$$

Multipliziert man noch **T(d)** mit **R**(ß), so erhält man die homogene Gesamttransformation. Gilt in einem Koordinatensystem eine andere Skalierung (z.B. bei einer besseren Positionsauflösung am Greifer), so drückt sich dies in der vierten Komponente aus, die größer oder kleiner eins wird.
Die Umrechnung der Punkte von Kartesischen Koordinaten in das Greifer-Koordinatensystem (s. Abb. 4.0a) läßt sich durch die Transformation **T** vornehmen. Die Matrix **T** muß dabei eine Translation um den Vektor **p** (s. Abb.4.0a) enthalten sowie eine Rotation von x, y und z. Betrachten wir die Darstellung der neuen Einheitsvektoren $\mathbf{e}_x'$ und $\mathbf{e}_y'$ im alten Koordinatensystem nach einer Rotation um den Winkel ß.

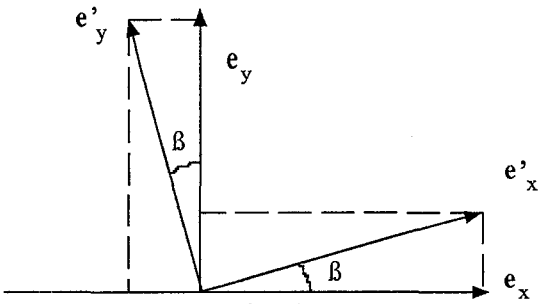


Abb. 4.2b Rotation der Basisvektoren

Die x und y- Komponenten lassen sich leicht aus Abb. 4.2b ablesen, so daß

$$\mathbf{e}_x' = \begin{pmatrix} \cos ß \\ \sin ß \\ 0 \end{pmatrix} \mathbf{e}_x \qquad \mathbf{e}_y' = \begin{pmatrix} -\sin ß \\ \cos ß \\ 0 \end{pmatrix} \mathbf{e}_y$$

Vergleichen wir dies mit der Rotationsmatrix aus (4.2d), so bemerken wir, daß die Spalten der Rotationsmatrix gerade die Beschreibung der neuen Basisvektoren im alten System darstellen. Die Transformationsmatrix **T** zwischen Weltkoordinaten (x, y, z) und Greiferkoordinaten (n, o, a) ist damit

$$\mathbf{T} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4.2h)$$

Angenommen, die Darstellung in Greiferkoordinaten ist gegeben. Wie erreichen wir die Kartesischen Weltkoordinaten?
Wir suchen x' mit

$$\mathbf{x} = \mathbf{T}\,\mathbf{x}'$$

oder

$$\mathbf{x}' = \mathbf{T}^{-1}\mathbf{x}$$

Um x' zu errechnen benötigen wir also die inverse Transformationsmatrix $\mathbf{T}^{-1}$. Wie sieht die aus?

Betrachten wir wieder **R**(z,a) aus (4.2d). Die Rotation wird zweifelsohne durch **R**(z,-a) rückgängig gemacht. Da cos (-a) = cos (a) und sin (-a) = -sin (a) ist **R**(z,a) = **R**(z,-a)^T, die transponierte Matrix. Da dies auch für **R**(x,a) und **R**(y,a) gilt, ist es auch für **R**(ß) zutreffend. Also ist der Rotationsteil von $\mathbf{T}^{-1}$ der transponierte Rotationsteil von **T**. Da außerdem $\mathbf{T}^{-1}\,\mathbf{T} = \mathbf{I}$ (Einheitsmatrix) gelten muß, ergibt sich der Translationsteil automatisch und $\mathbf{T}^{-1}$ lautet

$$\mathbf{T}^{-1} = \begin{pmatrix} n_x & n_y & n_z & -\mathbf{pn} \\ o_x & o_y & o_z & -\mathbf{po} \\ a_x & a_y & a_z & -\mathbf{pa} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4.2i)$$

mit den Skalarprodukten **pn**, **po** und **pa**.
Die Formulierung von $\mathbf{T}^{-1}$ ist in (4.2i) sehr einfach; allerdings ist dies nicht für beliebige 4x4 Matrizen so, sondern nur für homogene Transformationen.

Die inverse Problemstellung, bei gegebenen Greiferkoordinaten die Gelenkkoordinaten zu finden, ist sehr schwierig zu lösen. Leider gibt es keine vergleichbare elegante, einheitliche Methode der Rückrechnung (*inverse Kinematik*). Ein problembewußter Roboterkonstrukteur verwendet deshalb ausschließlich Gelenke mit parallelen oder orthogonalen Drehachsen, so daß sich die inversen Gleichungen auf quadratische Form reduzieren lassen, was aber hier nicht extra gezeigt werden soll.

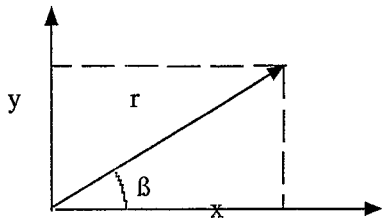
4.4 Trajektorien- und Pfadapproximation

Für die große Gruppe der *play-back robots* (s. Abschnitt 4.1) besteht das Kernproblem sowohl der Punkt-zu - Punkt Bewegung als auch der kontinuierlichen Bewegung darin, die Bewegung zwischen zwei Koordinatenpunkten zu errechnen. Die einfachste Art, dies zu realisieren, ist sicher, eine gewünschte kartesische Koordinate direkt durch die inverse homogene Transformation in Gelenkkoordinaten umzusetzen und über die Servomotoren die Gelenke entsprechend anzusteuern, bis die Positionssensoren den Vollzug der Bewegung melden (**Kartesische Bewegung**).
Da bei der Koordinatenumrechnung viele Gleitkomma-Operationen nötig sind, dauert dies in real-time Systemen meist zu lange und ist nur für langsame, genaue Bewegungen direkt am Objekt brauchbar. Außerdem ergibt sich aber noch ein anderes Problem. Betrachten wir dazu einen Roboterarm, der von Punkt A nach Punkt B mit der gleichmäßigen Geschwindigkeit von 100 cm/sec fahren soll, beispielsweise um eine Farbschicht aufzusprühen. Nehmen wir an, daß alle 5 ms eine neue Koordinate errechnet wird, so bewegt sich der Arm mit $1\,m/sec = (5\ 10^{-3}m)/(5\ 10^{-3}\ sec)$, also 5 mm in einem Abtastintervall. Stand der Arm vorher still, so bewegt er sich zwar auf der Gesamtstrecke mit einer konstanten Geschwindigkeit, aber er wird im ersten Wegintervall von 5 mm auf diese Geschwindigkeit beschleunigt. Für unser Beispiel bedeutet dies eine Beschleunigung von $a = v/t = 1\ m / (5\ 10^{-3})\ s^2 = 200\ m/s^2$, was den 20-fachem der Erdbeschleunigung entspricht! Von einem kräftigen, hydraulischem Roboterarm wird berichtet, daß er bei einer solchen Fehlprogrammierung so kräftig sich bewegte, daß er abbrach und als Geschoß durch die Luft flog.
Wir wir sehen, müssen wir bei der Bewegung des Roboterarms die Beschleunigungsphase am Anfang und am Ende der Bewegung mit in die Kontrolle einbeziehen. Allgemein muß die Bewegung eines Roboterarms als Folge von nicht-sensorgeführten, ballistischen oder nicht-kartesischen Einzelbewegungen vorausgeplant werden, um eine "fließende" Bewegungsfolge zu ermöglichen. Durch vorheriges Ausrechnen der Folge von anzusteuernden Koordinatenpunkten im OFF-line mode (Simulation der Roboterbewegung, Kap.6.7) läßt sich auch das konstante Zeitintervall zwischen den Punkten sehr klein wählen.
Bei der Bewegungs- oder Trajektorienplanung rechnet man also mit bestimmten, weiter unten erläuterten Methoden eine Liste von Koordinatenpunkten aus, die zur Bewegungssteuerung mit einer uhrgesteuerten Interrupt-Routine zur Laufzeit (ON-line) jeweils nach Ablauf des Zeitintervalls ausgelesen wird. Die

Koordinatenwerte der Liste werden dann der Steuerung der Gelenk-Servomotoren übergeben (*dog-race* Technik).
Welche Art von Koordinaten sollte man verwenden: *Kartesische oder Gelenkkoordinaten?*
Ist auf dem Weg ein Hindernis, so läßt es sich am Besten in Kartesischen Koordinaten angeben.
Wird dagegen die Liste der Bewegungskoordinaten zur Laufzeit ausgelesen, so besteht keine Zeit mehr für langwierige Transformationen, so daß die Liste in Gelenkkoordinaten vorliegen sollte.
Zweckmäßigerweise verwendet man ein Mischsystem: Start- und Endpunkt sowie die Kollisionsvermeidung wird in Kartesischen Koordinaten angegeben; die eigentliche Trajektorienplanung erfolgt in Gelenkkoordinaten.

Singularitäten

Dabei müssen wir allerdings ein Phänomen als Nebenbedingung beachten: Die Existenz von Singularitäten. Betrachten wir dazu einen polaren Roboterarm mit zwei Koordinaten (r,ß). Die Umrechnung der Position aus Kartesischen Koordinaten ist



$x = r \cos(ß)$

$y = r \sin(ß)$

Abb. 4.4a

Mit

$$\frac{y}{x} = \frac{\cos(ß)}{\sin(ß)} = \tan(ß) \quad \text{und} \quad x^2 + y^2 = r^2$$

folgt

$$ß = \text{atan}(y/x) \quad \text{und} \quad r = (x^2 + y^2)^{1/2}$$

Spezifizieren wir eine lineare Folge von Kartesischen Punkten (x,y), so ist die korrespondierende Folge von Gelenkkoordinaten (r,ß) nicht linear.

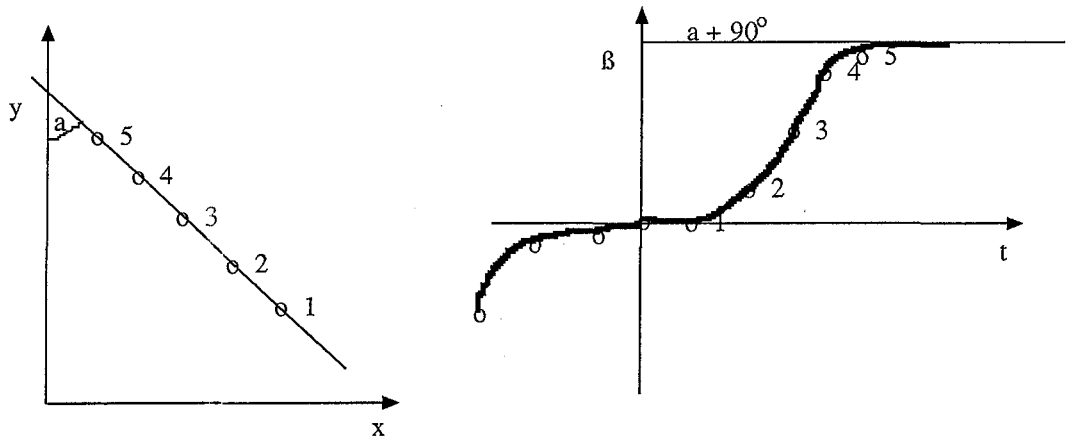


Abb. 4.4b lineare kartesiche Bewegung und die Gelenkwinkel

Verwendet man den atan2 (y,x), der mit dem atan () der positiven Argumente je nach Vorzeichen von x und y den richtigen Quadranten wählt undWinkel zwischen null und 360 Grad ausgibt, so sehen wir, daß immer dann, wenn die Trajektorie durch den Koordinatenursprung geht (r=0) die Polstellen (Singularitäten) des atan() auftreten.

Im Allgemeinen hat jeder Roboterarm mit Rotationsgelenken Singularitäten, die aber durch geeignete Konstruktion meist außerhalb des Arbeitsbereiches gelegt werden können.

Aufgabe: Welche Funktion f(r,ß) ist im vorigen Bild 4.4b rechts gezeigt? Wann hat sie Singularitäten?

Polynome zur Wegkontrolle

Aus dem Beispiel des gleichmäßig bewegten Roboterarms ist ersichtlich, daß für die Festlegung einer Bewegung außer der Positionsangabe von Anfangs- und Endkoordinaten noch weitere Randbedingungen gegeben sein müssen: Geschwindigkeit und Beschleunigung am Anfang und am Ende der Bewegung. Betrachten wir dazu eine Bewegung zwischen zwei Punkten A und B.

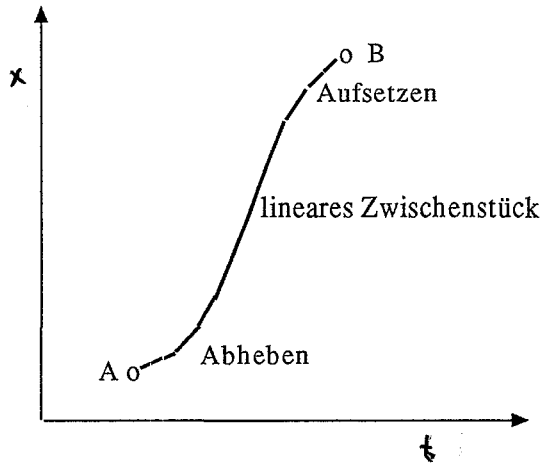


Abb. 4.4c Phasen einer Bewegung

Die Vorgaben bei Anfangs- und Endpunkt sind verschieden von den dazwischen liegenden Trajektorienteilen:

Anfangs- und Endpunkt: Position, Geschwindigkeit und Beschleunigung sind als Null gegeben.

Zwischenpunkte: Die Abhebeposition ist gegeben; Geschwindigkeiten und Beschleunigungen sollen kontinuierlich an das vorige Zeitsegment anknüpfen.
Die Aufsetzposition ist wieder gegeben; Geschwindigkeit und Beschleunigung sollen kontinuierlich in das folgenden Segment übergehen.

Um diesen Randbedingungen gerecht zu werden, approximiert man die Trajektoriensegmente meist mit Polynomen höherer Ordnung. Die Konstanten des Polynoms n-ter Ordnung werden dabei durch n Nebenbedingungen ermittelt. Betrachten wir dies an einem Beispiel.
Sei eine eindimensionale Bewegung festgelegt durch 6 Nebenbedingungen; die zeitlichen Ableitungen sind jeweils durch einen Punkt gekennzeichnet.

t = 0	t = T
$x(0) = x_0$	$x(T) = x_1$
$\dot{x}(0) = v_0$	$\dot{x}(T) = v_1$
$\ddot{x}(0) = a_0$	$\ddot{x}(T) = a_1$

Das approximierende Polynom ist damit

$$x(t) = c_0 + c_1t^1 + c_2t^2 + c_3t^3 + c_4t^4 + c_5t^5$$

und somit die Ableitungen

$$\dot{x}(t) = c_1 + 2\,c_2t^1 + 3c_3t^2 + 4c_4t^3 + 5\,c_5t^4$$

und

$$\ddot{x}(t) = 2c_2 + 6c_3t^1 + 12c_4t^2 + 20c_5t^3$$

Setzen wir in diese drei Gleichungen je t=0 und t=T ein und identifizieren mit unseren Nebenbedingungen, so erhalten wir 6 Gleichungen für 6 Unbekannte. Addieren und Subtrahieren wir die Gleichungen voneinander, so erhalten wir schließlich die Lösung

$$c_0 = x_0 \qquad c_1 = v_0 \qquad c_2 = a_0/2$$

$$c_3 = \frac{1}{2T^3}\left(20\,(x_1 - x_0) - (8v_1 + 12v_0)\,T - (3a_0 - a_1)\,T^2\right)$$

$$c_4 = \frac{1}{2T^4}\left(-30\,(x_1 - x_0) + (14v_1 + 16v_0)\,T + (3a_0 - 2\,a_1)\,T^2\right)$$

$$c_5 = \frac{1}{2T^5}\left(12\,(x_1 - x_0) - (6v_1 + 6v_0)\,T - (a_0 - a_1)\,T^2\right)$$

Die Approximation ist sowohl auf Kartesische als auch auf Winkelkoordinaten anwendbar.
Zweifelsohne gibt es noch weitere Taktiken, die Trajektorien zu approximieren. Eine andere Methode beispielsweise sieht vor, eine konstante Beschleunigung solange anzuwenden, bis die gewünschte Geschwindigkeit erreicht ist. Auch hier müssen die nicht-linearen Stücke so eingepaßt werden (Wahl der Beschleunigungwerte und -dauer), daß eine gleichförmige Bewegung resultiert.
In Abbildung 4.4d sind die zeitlichen Verläufe von Winkel, Winkelgeschwindigkeit und -beschleunigung bei Approximation durch ein Polynom 3. Grades und alternativ durch die Methode mit konstanter Beschleunigung gezeigt.

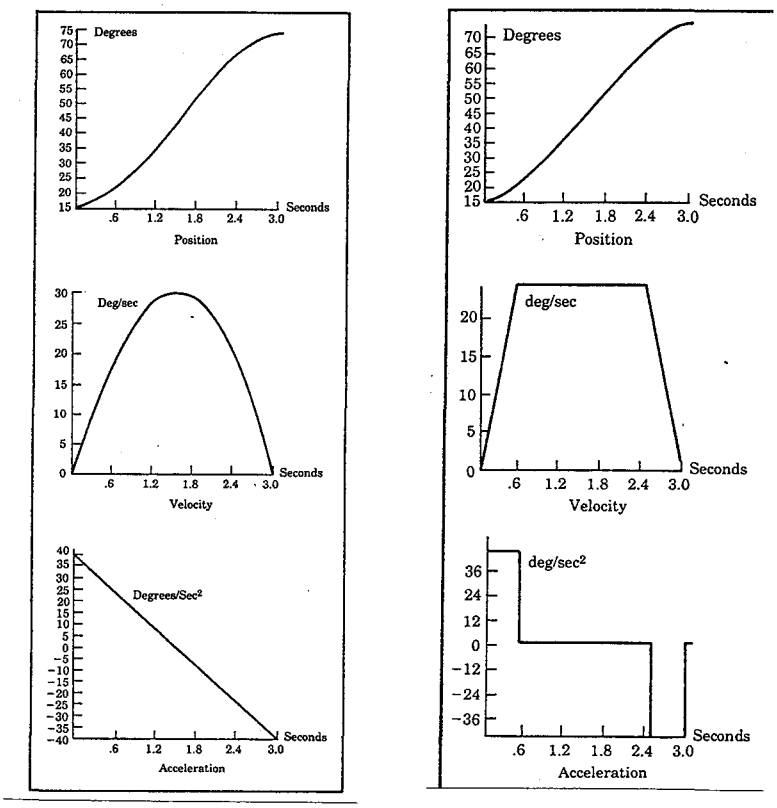


Abb. 4.4d Trajektorienapproximationen

5.0 Kinetik: Dynamik der Bewegung

Im vorigen Kapitel betrachteten wir den Roboterarm nur als abstraktes, geometrisches Gebilde und erhielten geometrische Transformationsregeln zwischen den Koordinatensystemen. Diese Modellierung ist aber zu ungenau; tatsächlich ist der Roboterarm ein physikalisches Gebilde mit Massen, Reibung, Kräften etc. Wir wollen also im Folgenden die grundlegenden dynamischen Eigenschaften der Roboterarme untersuchen, wobei wir uns auf das Verständnis der prinzipiellen Fragestellungen und Ansätze beschränken wollen. Benötigt ein Leser das Wissen, um tatsächlich einen Kontrollalgorithmus für einen konkreten Roboter zu programmieren, sei er auf detailliertere Fachliteratur (z.B. [Fu 87], Kap 3) verwiesen.
Auch unser Modell sei nur um die wesentlichsten dynamischen Elemente wie Kräfte, Massen und Gravitation erweitert; anderer Einflüsse wie Reibung und Durchbiegung (Resonanzen!) seien vernachlässigt. Das Modell ist so ohnehin schon komplex genug, wie wir sehen werden.

5.1 Die Lagrange-Euler Beschreibung

Betrachten wir als Beispiel den einfachen, polaren Manipulator mit den zwei Freiheitsgraden Radius r und Winkel ß

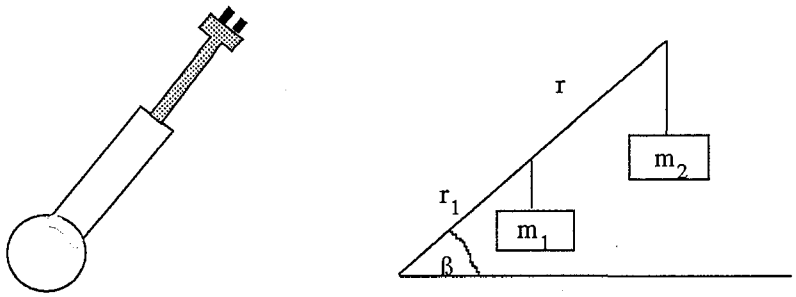


Abb. 5.1a polarer Roboterarm und sein Modell

Für unsere Betrachtung reicht es aus, sich die Massen m_1 des Dreharms und m_2 des Teleskoparms jeweils in einem Punkt im Abstand r_1 und r vorzustellen (s. obige Abbildung 5.1a rechts).
Bewegen sich Dreharm und Teleskoparm, so treten Kräfte auf. Wie groß sind diese Kräfte und woher kommen sie?
Aus der Physik wissen wir, daß auf eine mit a beschleunigte Masse m die Kraft F

$$F = m\,a \qquad \text{(Newton)}$$

angewendet werden muß. Ist die Kraft während einer Zeit t konstant, so resultiert der Impuls

$$I = m\,a\,t = m\,v$$

so daß

$$F = @\ I/@t\ =: \dot{I} \qquad \text{(Euler)}$$

resultiert.
Das Integral des Impulses wiederum ist die kinetische Energie der Masse m

$$E_k = 1/2\ m\ v^2$$

Die potentielle Energie der Masse m auf der Höhe h im Schwerkraftfeld ist

$$E_p = F\,h = m\,g\,h \qquad \text{mit der Erdbeschleunigung } g = 9{,}8\ m/s^2$$

Es gibt nun in der Physik eine Formulierung der Mechanik von Lagrange, die alle auftretenden, von uns gesuchten Kräfte der bewegten Massen angibt, wenn die potentiellen und kinetischen Energien bekannt sind. Mit der Lagrange-Funktion

$$L := E_k - E_p$$

gilt

$$F_{qi} = \frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} \qquad (5.1a)$$

wobei q_i ein allgemeiner Parameter ist. Betrachten wir $q_i = \beta$ als Drehwinkel, so ist F_β das Drehmoment; ist $q_i = x$ eine Kartesische Koordinate, so ist F_x die Kraft in x-Richtung.

Wie groß ist nun das Drehmoment in unserem Beispiel? Um L zu bilden und nach obiger Formel (5.1a) abzuleiten, müssen wir die potentielle und die kinetische Energie des Manipulators finden.

Die potentielle Energie ist (s. Abb. 5.1a)

$$E_p = m_1 g h_1 + m_2 g h_2 = m_1 g r_1 \sin(\beta) + m_2 g r \sin(\beta) = (m_1 r_1 + m_2 r) g \sin(\beta)$$

Für die kinetische Energie benötigen wir die Geschwindigkeit $v = \dot{x}$.

Die Position ist $\mathbf{x} = r_1 \begin{pmatrix} \cos(\beta) \\ \sin(\beta) \end{pmatrix}$ und die Geschwindigkeit $\dot{\mathbf{x}} = r_1 \dot{\beta} \begin{pmatrix} -\sin(\beta) \\ \cos(\beta) \end{pmatrix}$

so daß die kinetische Energie der Masse m_1 folgt

$$E_{k1} = 1/2\, m_1 \dot{\mathbf{x}}\, \dot{\mathbf{x}} = 1/2\, m_1 r_1^2 \dot{\beta}^2$$

Für m_2 ist r eine Funktion r(t) der Zeit, so daß

$$\dot{\mathbf{x}} = \dot{r} \begin{pmatrix} \cos(\beta) \\ \sin(\beta) \end{pmatrix} + r \dot{\beta} \begin{pmatrix} -\sin(\beta) \\ \cos(\beta) \end{pmatrix}$$

und so

$$E_{k2} = 1/2\, m_2 \dot{\mathbf{x}}\, \dot{\mathbf{x}} = 1/2\, m_2 (\dot{r}^2 + r^2 \dot{\beta}^2)$$

Die Lagrangefunktion lautet somit

$$L = E_{k1}(\dot{\beta}) + E_{k2}(r, \dot{r}, \dot{\beta}) - E_p(r, \beta)$$

Die Ableitungen für Gleichung (5.1a) sind

$$\frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{\beta}} = \frac{\partial}{\partial t}\left(m_1 r_1^2 \dot{\beta} + m_2 r^2 \dot{\beta} \right) = (m_1 r^2 + m_2 r^2) \ddot{\beta} + 2 m_2 r \dot{r} \dot{\beta}$$

$$-\frac{\partial L}{\partial \beta} = (m_1 r_1 + m_2 r_2) g \cos(\beta)$$

so daß sich das Drehmoment des Beispiels sich ergibt als

$$F_\beta = (m_1 r^2 + m_2 r^2) \ddot{\beta} + 2 m_2 r \dot{r} \dot{\beta} + (m_1 r_1 + m_2 r_2) g \cos(\beta)$$
$$=: D \ddot{\beta} + h(\dot{\beta}) + c(\beta) \qquad (5.1b)$$

Aufgabe: Rechnen Sie die Kraft F_r in radialer Richtung aus.

Die obige Form (5.1b) heißt **geschlossenen Form** und gibt an, wie sich die resultierende Kraft aus verschiedenen Komponenten zusammensetzt.
Der Beitrag des Koeffizienten D rührt, erkennbar an der zweifachen Ableitung von ß, von den Trägheitsmomenten der Massen her.
Der Kraftanteil h(ß) ergibt sich aus der scheinbaren Ablenkung von der radialen Richtung während der Drehbewegung bei der Bewegung nach außen (Coriolis-Kraft); bei F_r taucht hier noch zusätzlich die Zentripetalkraft auf.
Der Term c(ß) spiegelt statische Kräfte wider, wie sie beispielsweise die Gravitation darstellt.
Aus Aufteilung der geschlossenen Form lassen sich nun Vorteile für das Design der Controller der einzelnen Gelenke ableiten. Ist ein Gelenk so ausgelegt, daß bestimmte Kraftanteile zu vernachlässigen sind, (Beispiel: Graviataion bei einem Rotationsgelenk mit Achse in z-Richtung), so kann der Kontrollalgorithmus auf die Berechnung dieser Anteile verzichten.

Allgemeine Lagrange-Formulierung

Die allgemeine Formulierung der Lagrange-Gleichungen soll hier nur kurz gezeigt werden, um einen Eindruck von der Komplexität der Lösung zu geben.
Bezeichnen wir die homogene Transformation vom Koordinatensystem des i-1ten Gelenks zum i^{ten} Gelenk mit ${}^{i-1}A_i$ und die Ableitung

$$\frac{\partial}{\partial q_i} {}^{i-1}A_i = Q_i \, {}^{i-1}A_i$$

mit

$$Q_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Rotationsgelenk

$$Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Translationsgelenk

Sei weiterhin die Ableitung $U_{ij} := \frac{\partial}{\partial q_j} {}^{0}A_i$ definiert, so ist

$$U_{ij} = {}^{0}A_{j-1} \, Q_j \, {}^{j-1}A_j \qquad \text{für } j \le i, \text{ sonst null.}$$

Sei außerdem die Masse in ihrer Verteilung bekannt, so ist die träge Masse J_i des i-ten Armsegments durch den Tensor im Basiskoordinatensystem gegeben

$$J_i = \int {}^{i}\mathbf{r}_i \, {}^{i}\mathbf{r}_i^T \, dm = \begin{bmatrix} \int x_i^2 \, dm & \int x_i y_i \, dm & \int x_i z_i \, dm & \int x_i \, dm \\ \int x_i y_i \, dm & \int y_i^2 \, dm & \int y_i z_i \, dm & \int y_i \, dm \\ \int x_i z_i \, dm & \int y_i z_i \, dm & \int z_i^2 \, dm & \int z_i \, dm \\ \int x_i \, dm & \int y_i \, dm & \int z_i \, dm & \int dm \end{bmatrix}$$

Dann lautet die Lagrange- Funktion

$$L = \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{i} [\mathrm{Tr}\,(U_{ij} J_i U_{ik}^T) \dot{q}_j \dot{q}_k] + \sum_{i=1}^{n} m_i \mathbf{g} ({}^{0}A_i \, {}^{i}\bar{\mathbf{r}}_i)$$

wobei Tr(A) die *Spur* (Summe der Hauptdiagonalelemente) der Matrix A ist.
Die Lagrange-Gleichung (5.1a) läßt sich ausrechnen und umformen, wobei Summen teilweise durch Matrixoperationen ersetzt werden können. Die allgemeine geschlossene Form (vgl. Formel 5.1b) lautet dann

$$\mathbf{F}(t) = \mathbf{D}(\mathbf{q}(t))\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}(t),\dot{\mathbf{q}}(t)) + \mathbf{c}(\mathbf{q}(t))$$

mit den Koeffizienten

$$i, k, m = 1, 2, \ldots, n$$

$$D_{ik} = \sum_{j=\max(i,k)}^{n} \mathrm{Tr}\,(\mathbf{U}_{jk}\mathbf{J}_j\mathbf{U}_{ji}^T)$$

$$h_i = \sum_{k=1}^{n}\sum_{m=1}^{n} h_{ikm}\dot{q}_k\dot{q}_m \qquad h_{ikm} = \sum_{j=\max(i,k,m)}^{n} \mathrm{Tr}\,(\mathbf{U}_{jkm}\mathbf{J}_j\mathbf{U}_{ji}^T)$$

$$c_i = \sum_{j=i}^{n} (-m_j\mathbf{g}\mathbf{U}_{ji}{}^j\bar{\mathbf{r}}_j)$$

Für einen Roboter mit 6 Freiheitsgraden θ haben schon die D-koeffizienten folgende Form:

$$\mathbf{D}(\theta) = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} \\ D_{12} & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} \\ D_{13} & D_{23} & D_{33} & D_{34} & D_{35} & D_{36} \\ D_{14} & D_{24} & D_{34} & D_{44} & D_{45} & D_{46} \\ D_{15} & D_{25} & D_{35} & D_{45} & D_{55} & D_{56} \\ D_{16} & D_{26} & D_{36} & D_{46} & D_{56} & D_{66} \end{bmatrix}$$

where

$$D_{11} = \mathrm{Tr}\,(\mathbf{U}_{11}\mathbf{J}_1\mathbf{U}_{11}^T) + \mathrm{Tr}\,(\mathbf{U}_{21}\mathbf{J}_2\mathbf{U}_{21}^T) + \mathrm{Tr}\,(\mathbf{U}_{31}\mathbf{J}_3\mathbf{U}_{31}^T) + \mathrm{Tr}\,(\mathbf{U}_{41}\mathbf{J}_4\mathbf{U}_{41}^T) + \mathrm{Tr}\,(\mathbf{U}_{51}\mathbf{J}_5\mathbf{U}_{51}^T) + \mathrm{Tr}\,(\mathbf{U}_{61}\mathbf{J}_6\mathbf{U}_{61}^T)$$

$$D_{12} = D_{21} = \mathrm{Tr}\,(\mathbf{U}_{22}\mathbf{J}_2\mathbf{U}_{21}^T) + \mathrm{Tr}\,(\mathbf{U}_{32}\mathbf{J}_3\mathbf{U}_{31}^T) + \mathrm{Tr}\,(\mathbf{U}_{42}\mathbf{J}_4\mathbf{U}_{41}^T) + \mathrm{Tr}\,(\mathbf{U}_{52}\mathbf{J}_5\mathbf{U}_{51}^T) + \mathrm{Tr}\,(\mathbf{U}_{62}\mathbf{J}_6\mathbf{U}_{61}^T)$$

$$D_{13} = D_{31} = \mathrm{Tr}\,(\mathbf{U}_{33}\mathbf{J}_3\mathbf{U}_{31}^T) + \mathrm{Tr}\,(\mathbf{U}_{43}\mathbf{J}_4\mathbf{U}_{41}^T) + \mathrm{Tr}\,(\mathbf{U}_{53}\mathbf{J}_5\mathbf{U}_{51}^T) + \mathrm{Tr}\,(\mathbf{U}_{63}\mathbf{J}_6\mathbf{U}_{61}^T)$$

$$D_{14} = D_{41} = \mathrm{Tr}\,(\mathbf{U}_{44}\mathbf{J}_4\mathbf{U}_{41}^T) + \mathrm{Tr}\,(\mathbf{U}_{54}\mathbf{J}_5\mathbf{U}_{51}^T) + \mathrm{Tr}\,(\mathbf{U}_{64}\mathbf{J}_6\mathbf{U}_{61}^T)$$

$$D_{15} = D_{51} = \mathrm{Tr}\,(\mathbf{U}_{55}\mathbf{J}_5\mathbf{U}_{51}^T) + \mathrm{Tr}\,(\mathbf{U}_{65}\mathbf{J}_6\mathbf{U}_{61}^T)$$

$$D_{16} = D_{61} = \mathrm{Tr}\,(\mathbf{U}_{66}\mathbf{J}_6\mathbf{U}_{61}^T)$$

$$D_{22} = \mathrm{Tr}\,(\mathbf{U}_{22}\mathbf{J}_2\mathbf{U}_{22}^T) + \mathrm{Tr}\,(\mathbf{U}_{32}\mathbf{J}_3\mathbf{U}_{32}^T) + \mathrm{Tr}\,(\mathbf{U}_{42}\mathbf{J}_4\mathbf{U}_{42}^T) + \mathrm{Tr}\,(\mathbf{U}_{52}\mathbf{J}_5\mathbf{U}_{52}^T) + \mathrm{Tr}\,(\mathbf{U}_{62}\mathbf{J}_6\mathbf{U}_{62}^T)$$

$$D_{23} = D_{32} = \mathrm{Tr}\,(\mathbf{U}_{33}\mathbf{J}_3\mathbf{U}_{32}^T) + \mathrm{Tr}\,(\mathbf{U}_{43}\mathbf{J}_4\mathbf{U}_{42}^T) + \mathrm{Tr}\,(\mathbf{U}_{53}\mathbf{J}_5\mathbf{U}_{52}^T) + \mathrm{Tr}\,(\mathbf{U}_{63}\mathbf{J}_6\mathbf{U}_{62}^T)$$

$$D_{24} = D_{42} = \mathrm{Tr}\,(\mathbf{U}_{44}\mathbf{J}_4\mathbf{U}_{42}^T) + \mathrm{Tr}\,(\mathbf{U}_{54}\mathbf{J}_5\mathbf{U}_{52}^T) + \mathrm{Tr}\,(\mathbf{U}_{64}\mathbf{J}_6\mathbf{U}_{62}^T)$$

$$D_{25} = D_{52} = \mathrm{Tr}\,(\mathbf{U}_{55}\mathbf{J}_5\mathbf{U}_{52}^T) + \mathrm{Tr}\,(\mathbf{U}_{65}\mathbf{J}_6\mathbf{U}_{62}^T)$$

$$D_{26} = D_{62} = \mathrm{Tr}\,(\mathbf{U}_{66}\mathbf{J}_6\mathbf{U}_{62}^T)$$

$$D_{33} = \mathrm{Tr}\,(\mathbf{U}_{33}\mathbf{J}_3\mathbf{U}_{33}^T) + \mathrm{Tr}\,(\mathbf{U}_{43}\mathbf{J}_4\mathbf{U}_{43}^T) + \mathrm{Tr}\,(\mathbf{U}_{53}\mathbf{J}_5\mathbf{U}_{53}^T) + \mathrm{Tr}\,(\mathbf{U}_{63}\mathbf{J}_6\mathbf{U}_{63}^T)$$

$$D_{34} = D_{43} = \mathrm{Tr}\,(\mathbf{U}_{44}\mathbf{J}_4\mathbf{U}_{43}^T) + \mathrm{Tr}\,(\mathbf{U}_{54}\mathbf{J}_5\mathbf{U}_{53}^T) + \mathrm{Tr}\,(\mathbf{U}_{64}\mathbf{J}_6\mathbf{U}_{63}^T)$$

$$D_{35} = D_{53} = \mathrm{Tr}\,(\mathbf{U}_{55}\mathbf{J}_5\mathbf{U}_{53}^T) + \mathrm{Tr}\,(\mathbf{U}_{65}\mathbf{J}_6\mathbf{U}_{63}^T)$$

$$D_{36} = D_{63} = \mathrm{Tr}\,(U_{66}J_6U_{63}^T)$$

$$D_{44} = \mathrm{Tr}\,(U_{44}J_4U_{44}^T) + \mathrm{Tr}\,(U_{54}J_5U_{54}^T) + \mathrm{Tr}\,(U_{64}J_6U_{64}^T)$$

$$D_{45} = D_{54} = \mathrm{Tr}\,(U_{55}J_5U_{54}^T) + \mathrm{Tr}\,(U_{65}J_6U_{64}^T)$$

$$D_{46} = D_{64} = \mathrm{Tr}\,(U_{66}J_6U_{64}^T)$$

$$D_{55} = \mathrm{Tr}\,(U_{55}J_5U_{55}^T) + \mathrm{Tr}\,(U_{65}J_6U_{65}^T)$$

$$D_{56} = D_{65} = \mathrm{Tr}\,(U_{66}J_6U_{65}^T)$$

$$D_{66} = \mathrm{Tr}\,(U_{66}J_6U_{66}^T)$$

Wie man sieht, erfordert die allgemeine Berechnung der Terme viele Multiplikationen und Additionen. Die Berechnungskomplexität bei der Lagrang'schen Methode ist bei n Segmenten ziemlich hoch. Sie liegt in der Ordnung $O(n^4)$, was die Verwendung in real-time Systemen ausschließt. Die Berechnung der obigen Gleichungen wird deshalb hauptsächlich für Simulationen zur Konstruktion von Robotern und Fertigungszellen verwendet. Dabei durchfährt man den gewünschten Weg simulativ mit Höchstgeschwindigkeit und prüft die auftretenden Belastungen.
Eine andere Anwendung besteht darin, bereits vorher bei genau festliegenden Wegen mit den errechneten, auftretenden Kräften die Ansteuerung der Elektronik für die entsprechenden Trajektorien festzulegen.

5.2 Der Newton-Eulersche Ansatz

Im Unterschied zu dem Lagrange-Ansatz, der zum ersten Mal 1960 in 4-dimensionalen, homogenen Koordinaten für Roboter veröffentlicht wurde, läßt sich alternativ dazu eine rekursive Beschreibung formulieren, die roboterunabhängig für jedes einzelne Segment gilt. Sie beruht auf den Grundgleichungen von Newton und Euler, die ja auch dem Lagrange-Formalismus zugrunde liegen. Dabei zeigte sich jedoch, daß die Berechnungskomplexität der Kräfte, Beschleunigungen und Geschwindigkeiten von der Ordnung O(n) ist, was für die Computerauswertung einen großen Fortschritt bedeutet. Damit wird es möglich, auch ON-Line die auftretenden, dynamischen Kräfte zu berechnen, um Abweichungen bereits bei der Trajektorienplanung zu berücksichtigen.
Die Entkopplung der Berechnungen für die einzelnen Segmente können gut von modernen, dezentralisierten Steuerungen ausgenutzt werden. Legen wir eine Computerarchitektur wie in Abbildungen 6.0a und b zugrunde, bei der jedem Segment bzw. Gelenk ein eigener, mit den anderen gekoppelten Controller zugeordnet wird, so wird die Berechnung separat von jedem Controller nur für das Segment bzw. das Gelenk gemacht, für das er zuständig ist. Die errechneten Werte reicht er weiter an den Controller des nächsten Segments, der damit seine eigenen Werte errechnen kann und so fort (s. Abb.6.0a).
Als Beispiel berechnen wir die Geschwindigkeit eines gedrehten Koordinatenvektors s(t)

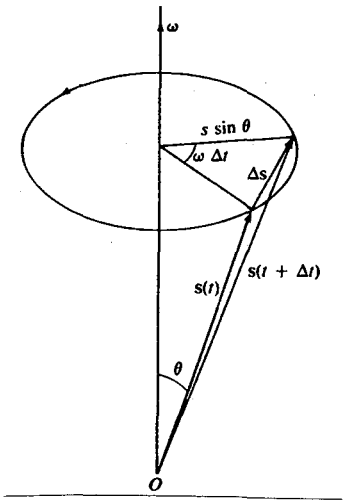


Abb. 5.2a Rotation eines Koordinatenvektors

Aus obiger Abbildung ist ersichtlich, daß der Abstand des Vektors s(t) zur Rotationsachse mit dem Sinus gewichtet ist

$$r = s \sin(\theta)$$

Betrachten wir nun die Differenz $s := \Delta s$ für kleinere Zeiten $t := \Delta t$, so ist mit der Winkelgeschwindigkeit w

$$s = r \sin(wt) \quad \text{und mit } t \to 0 \text{ zu } \quad r\ wt$$

da bei kleinen Winkeln der sinus(x) proportional zu x wird.

$$s/t = r\ w$$

und im Grenzwert bei $t \to 0$

$$ds / dt = r\ w = s\ w \sin(\theta)$$

Bezeichnen wir das *äußere Produkt* zweier Vektoren w und s mit wxs, so ist

$$|ds / dt| = |s\ w \sin(\theta)| = |wxs|$$

und die Richtung der Winkelgeschwindigkeit ist parallel zu der Rotationsachse

$$\dot{\mathbf{s}} = \mathbf{wxs} \qquad (5.2a)$$

Die Richtung von wxs ist zweifelsohne nach Abb. 5.2a tangential zum Rotationskreis der Spitze von s und rechtwinklig zu s und w.

Die Beziehung (5.2a) benutzen wir nun, um die Vektorgeschwindigkeit $\dot{\mathbf{r}}$ zu transformieren.
Sei r in dem rotierenden Koordinatensystem mit den Einheitsvektoren i*,j*, k* dargestellt als $\mathbf{r} = (x^*, y^*, z^*)^T$, so ist

$$\dot{\mathbf{r}} = \dot{x}^* \mathbf{i}^* + \dot{y}^* \mathbf{j}^* + \dot{z}^* \mathbf{k}^* + x^* \, di^*/dt + y^* \, dj^*/dt + z^* \, dk^*/dt$$

und mit (5.2a)

$$\dot{\mathbf{r}} = \dot{\mathbf{r}}^* + x^* (\mathbf{wxi}^*) + y^* (\mathbf{wxj}^*) + z^* (\mathbf{wxk}^*) = \dot{\mathbf{r}}^* + \mathbf{wxr} \qquad (5.2b)$$

Mit dieser Grundbeziehung lassen sich nun rekursive Gleichungen für Geschwindigkeiten und Kräfte herleiten. Dabei werden zuerst Rotationsgeschwindigkeit, -beschleunigung, Translationsgeschwindigkeit und -beschleunigung für das Gelenk i mit der Rotationsachse in z_i-Richtung nacheinander von i=1 bis n errechnet. Mit den letzten Werten vom Greifer (i=n) wird dann rekursiv von i=n bis 1 auf die jeweils anliegenden Kräfte zurückgeschlossen. In Abb. 52b sind die Gleichungen gezeigt.

Forward equations: $i = 1, 2, \ldots, n$

$$\omega_i = \begin{cases} \omega_{i-1} + z_{i-1}\dot{q}_i & \text{if link } i \text{ is rotational} \\ \omega_{i-1} & \text{if link } i \text{ is translational} \end{cases}$$

$$\dot{\omega}_i = \begin{cases} \dot{\omega}_{i-1} + z_{i-1}\ddot{q}_i + \omega_{i-1} \times (z_{i-1}\dot{q}_i) & \text{if link } i \text{ is rotational} \\ \dot{\omega}_{i-1} & \text{if link } i \text{ is translational} \end{cases}$$

$$\dot{v}_i = \begin{cases} \dot{\omega}_i \times p_i^* + \omega_i \times (\omega_i \times p_i^*) + \dot{v}_{i-1} & \text{if link } i \text{ is rotational} \\ z_{i-1}\ddot{q}_i + \dot{\omega}_i \times p_i^* + 2\omega_i \times (z_{i-1}\dot{q}_i) & \\ \quad + \omega_i \times (\omega_i \times p_i^*) + \dot{v}_{i-1} & \text{if link } i \text{ is translational} \end{cases}$$

$$\bar{a}_i = \dot{\omega}_i \times \bar{s}_i + \omega_i \times (\omega_i \times \bar{s}_i) + \dot{v}_i$$

Backward equations: $i = n, n-1, \ldots, 1$

$$\mathbf{F}_i = m_i \bar{\mathbf{a}}_i$$

$$\mathbf{N}_i = \mathbf{I}_i \dot{\omega}_i + \omega_i \times (\mathbf{I}_i \omega_i)$$

$$\mathbf{f}_i = \mathbf{F}_i + \mathbf{f}_{i+1}$$

$$\mathbf{n}_i = \mathbf{n}_{i+1} + \mathbf{p}_i^* \times \mathbf{f}_{i+1} + (\mathbf{p}_i^* + \bar{\mathbf{s}}_i) \times \mathbf{F}_i + \mathbf{N}_i$$

$$\tau_i = \begin{cases} \mathbf{n}_i^T \mathbf{z}_{i-1} + b_i \dot{q}_i & \text{if link } i \text{ is rotational} \\ \mathbf{f}_i^T \mathbf{z}_{i-1} + b_i \dot{q}_i & \text{if link } i \text{ is translational} \end{cases}$$

where b_i is the viscous damping coefficient for joint i.

The "usual" initial conditions are $\omega_0 = \dot{\omega}_0 = \mathbf{v}_0 = \mathbf{0}$ and $\dot{\mathbf{v}}_0 = (g_x, g_y, g_z)^T$ (to include gravity), where $|\mathbf{g}| = 9.8062$ m/s^2.

Abb. 5.2b Rekursive Newton-Eulersche Segmentgleichungen

Die gequerten Terme stellen die Größen bezüglich des Massenschwerpunkts $\bar{s}_i$ des einzelnen Segments dar. Das Trägheitsmoment N_i des Segments und die Beschleunigungskräfte F_i tragen zum Gesamtträgheitsmoment n_i bei, das summarisch auf das Gelenk einwirkt und sich auf das vorige Segment überträgt. Auch die Kräfte f_{i+1} vom vorigen Segment wirken sich aus, ebenso wie der Abstand $\mathbf{p}_i^*$ des Koordinatennullpunkts vom Basiskoordinatensystem.
Diese Bewegungsgleichungen haben allerdings ein Problem: Sowohl die Massetensoren $\mathbf{I}_i$ als auch $\mathbf{p}_i^*$ sind bezüglich des Basiskoordinatensystems dargestellt und ändern sich während der Bewegung. Dies läßt sich aber durch Umtransformation der Größen in relative Gelenk-Koordinatensystem reformulieren.
Durch die konfigurationsneutrale, rekursive Formulierung der Bewegungsgleichungen ließ sich eine lineare Berechnungskomplexität erreichen, die sogar real-time Anwendungen möglich macht und den Newton-Euler Ansatz gegenüber dem Lagrange-Formalismus hervorhebt. Allerdings gibt es einige neuere Arbeiten, die die Äquivalenz der beiden Formulierungen zeigt und eine rekursive Formulierung der Lagrange-Gleichungen vorschlägt, die fast ebenso effektiv ist wie die Newton-Euler Formeln , s. [Snyder], p205.

5.3 Rückkopplungskontrolle

Vielfach ist eine Ausrechnung der vollen, dynamischen Gleichungen für die Ansteuerung der Gelenkmotoren innerhalb weniger Millisekunden nicht möglich. Deshalb sind die heutigen Roboter meist nur mit einer einfachen Rückkopplung von den Positionssensoren ausgerüstet, wobei versucht wird, die Abweichung des Pfades von der gewünschten Trajektorie so klein wie möglich zu machen. Da eine der wichtigsten, äußeren Einwirkungen in der Schwerkraft besteht, ist es ein guter Ansatz, bei der Ansteuerung bereits die von der Schwerkraft bewirkte, voraussichtliche Abweichung einzubeziehen. Allgemein lassen sich die Variablen $\mathbf{u}(t)$ der Servo-Motorkontrolle als Funktion der Abweichung steuern:

$$\mathbf{u}(t) = \mathbf{f}\,(\mathbf{x}_{\text{geplant}}(t) - \mathbf{x}_{\text{real}}(t)) \qquad (5.3)$$

Die Funktion $\mathbf{f}$ läßt sich eventuell aus der Umformulierung der allgemeinen Kontrollgleichung

$$\dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{x},t) + \mathbf{B}(\mathbf{x},t)\,\mathbf{u}(t)$$

gewinnen, deren Matrizen $\mathbf{A}$ und $\mathbf{B}$ prinzipiell aus den nicht-linearen, dynamischen Bewegungsgleichungen gewonnen werden können. Um die Funktion $\mathbf{u}(t)$ eindeutig zu charakterisieren, werden dabei noch Nebenbedingungen (z.B. minimale Energie: Integral über u^2 ist minimal) berücksichtigt.

5.4 Kräftekontrolle

Bei vielen Aufgaben können sich der Schlupf in der Kraftübertragung, die Fehler der Positionssensoren oder eine kleine Variation in der Positionierung des bearbeitenden Teils verhängnisvoll auswirken. Ein Beispiel ist das Waschen einer Glasscheibe oder das Abkratzen von Lack: Schon eine geringe Abweichung zerstört die Scheibe oder bewirkt miserable Arbeitsergebnisse. In einem solchen Fall ist es sinnvoll, durch Kräftesensoren im Verbund mit den Positionssensoren für einen gleichmäßigen Andruck des Werkzeugs während der Bewegung zu sorgen. Abbildung 5.4a illustriert das Grundproblem, einen Manipulator mit gleichmäßigem Andruck über eine Oberfläche fahren zu lassen.

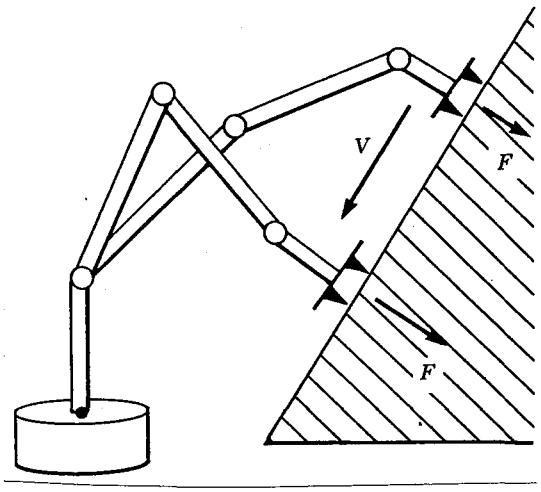


Abb. 5.4a Bewegung mit kontrolliertem Andruck

Bei der Betrachtung unserer Newton-Eulerschen Gleichungen aus Abb.5.2b sehen wir, daß dabei die Kräfte $\mathbf{f}_{i+1}$ eingesetzt werden. Nehmen wir die resultierende Kraft von n+1 als "fest gegeben" an, so resultieren daraus Kräfte bzw. Beschleunigung und Geschwindigkeit des vorigen Arm-Segments. Nehmen wir der Einfachheit halber an, daß jedes Gelenk entweder nur einer Positionskontrolle oder aber einer Kraftkontrolle unterworfen wird, so läßt sich folgendes einfache Schema einer zusammengefaßten (hybriden) Positions- und Kraftrückkopplung für einen Kartesischen Roboterarm aufstellen:

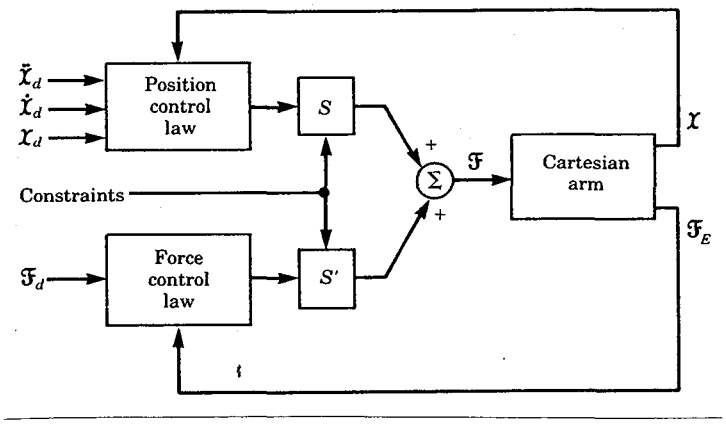


Abb. 5.4b hybride Positions- und Kraftkontrolle

Die sich einander ergänzenden Konstanten Matrizen S und S' enthalten nur null und eins und regeln damit, welches Gelenk der Positions- und welches der Kraftkontrolle unterliegt.

6.0 Die Programmierung der Roboter

Die Kontrolle der Servomotoren der einzelnen Gelenke geschieht bei modernen Robotern meist über eigene Mikroprozessoren, deren Aktivität koordiniert wird. In Abbildung 6.0a sind zwei mögliche Hardwarekonfigurationen gezeigt. In beiden Architekturen wird jede Verstärkerendstufe eines Gelenkmotors durch einen eigenen Mikroprozessor gesteuert, der mit den Daten des Positionssensors am Gelenk rückgekoppelt wird.
Die zeitkritischen Assemblerprogramme sind direkt (in ROM) bei den Mikroprozessoren vorhanden; bei der Architektur eines PUMA-Roboters links in der Abbildung errechnet ein zentraler Prozessor (PDP11/03) die Kenndaten der Bewegungskinematik und transferiert sie anschließend an die einzelnen Gelenk-Controller (Prozessoren).

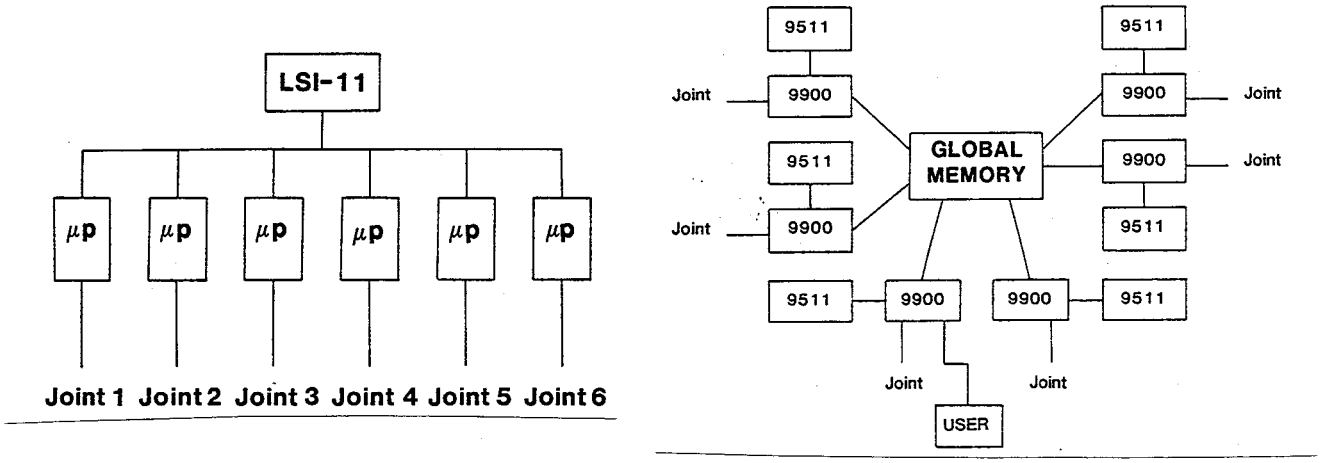


Abb. 6.0a Zwei Hardwarearchitekturen der Roboterkontrolle

In der rechten Architektur ist das Gleiche in einer dezentralen Koordination mit Prozessoren (Texas Instruments 9900 mit Gleitkomma-Coprozessor) verwirklicht, wobei die Synchronisation durch Systemdaten im globalen, zur Kommunikation benutzten Speicher gesteuert wird. Die Berechnung der kinematischen Daten wird von jedem Gelenk-Controller separat für das von ihm kontrollierte Gelenk durchgeführt (*funktionale Verteilung*), wobei die errechneten Daten der anderen Gelenke beachtet werden müssen (Newton-Eulersche Gleichungen, Abb. 5.2d).

Wie wird nun die Bewegung eines Roboters auf den Computern programmiert?

Die Programmierung eines Roboters läßt sich in einzelne Schichten zerlegen, die einerseits Aktivität (Befehle) initiieren und andererseits Sensorinformation erhalten und sie zur Rückkopplung (Korrektur der Aktivität) verwenden.

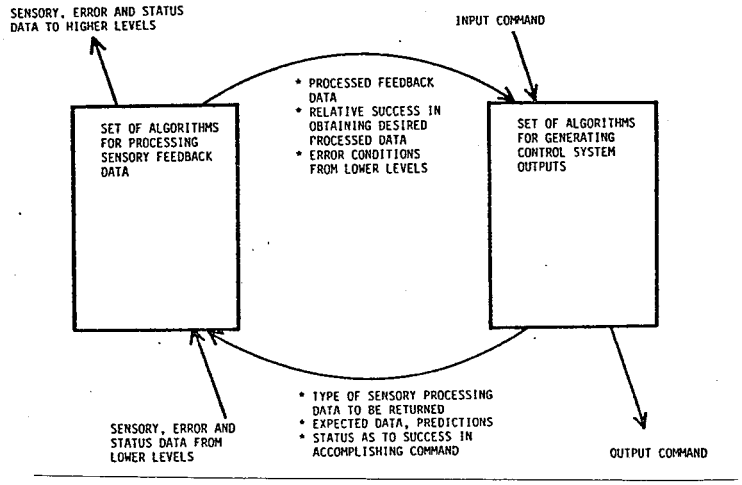


Abb. 6.0b Kontrolle einer Schicht

Die einzelnen Schichten oder Stufen sind hierarchisch angeordnet; jeder Schicht entspricht einer Abstraktionsebene der Befehlsgebung und der Sensorinformation: von der allgemeinen Zielvorgabe hinunter bis zur Ansteuerung des Gelenkmotors; von dem digital codierten Positionswert hinauf bis zur Meldung "Greifer ist im kritischen Bereich vor dem Objekt".

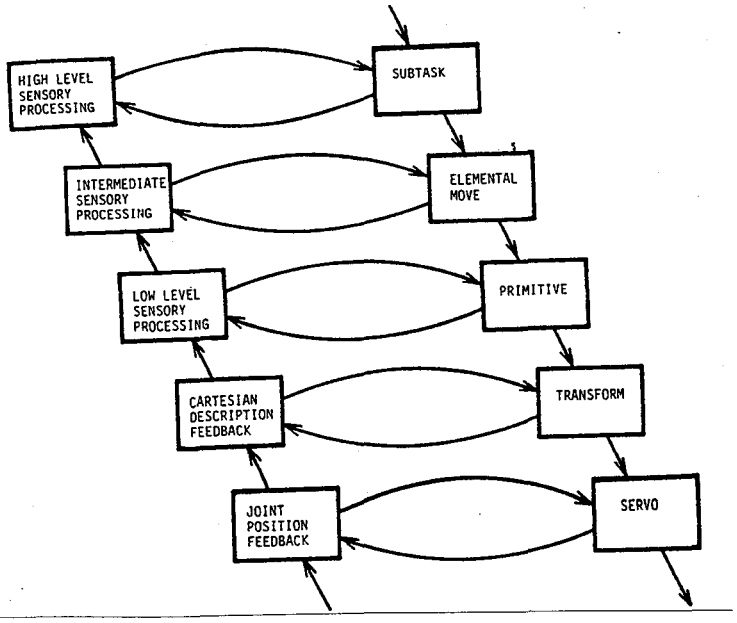


Abb. 6.0c hierarchisches Schichtenmodell der Roboterkontrolle

Historisch entwickelte sich die Roboterprogrammierung aus dem teach-in Verfahren der playback robots (s. Abschnitt 4.1). für die immer abstraktere und mächtigere Formen der Bewegungsplanung entwickelt wurden. Die Schichten aus Abb. 6.0c entsprechen damit ungefähr den historischen Schichten der Entwicklung der Robotersprachen. Dabei besteht eine enge Verwandschaft zur Entwicklung der allgemeinen Programmiersprachen.

Roboterprogrammierung	allg.Programmierung
menschl. Intelligenz	**menschl. Intelligenz**
. . .	. . .
Aufgabenorientierte Schicht	**Aufgabenorientierte Schicht**
Autopass	*Expertensysteme, 4.Generation, Prolog, Lisp*
Strukturierte Programmierung	**Strukturierte Programmierung**
AL, MCL, MAPLE, PAL, HELP, AML, VAL II	*Algol, Pascal, Modula2, Ada*
Bewegungs-Ebene	**GOTO-Sprachen**
VAL, EMELY, RCL, SIGLA, RPL, ANORAD	*Basic, Cobol, Fortran*
Punkt-zu-Punkt Bewegung	**Maschineninstruktion**
Funky, T3	*Assembler*
Mikroprozessor-Ebene	**Mikroprogrammierung**
Assembler, C	*Mikroassembler*

Im Folgenden seien die Programmiersprachen etwas näher erläutert.

6.1 Mikroprozessor-Ebene

Auf der Ebene des Mikroprozessors, der für die Kontrolle eines einzelnen Gelenks zuständig ist, sind die Programme der Realzeitkontrolle in Assembler oder C geschrieben und meist in einem ROM abgelegt. Die Wahl der Sprache ist dabei hauptsächlich durch die Schnelligkeit des erzeugten Codes gegeben; dies waren bisher der normale Assembler oder der (höhere) Assembler C. In neuerer Zeit gibt es allerdings auch Compiler für die höhere, strukturierte Programmiersprache Modula-2, deren erzeugter Code noch besser optimiert und damit schneller ist als die Standardimplementationen des C-Compilers dies fertigbringen. In diesem Fall sollte man natürlich Assembler nur noch für die zeitkritschen Prozeduren verwenden.
Der Befehlsinput der Mikroprozessorebene sind die gewünschten Gelenkkoordinaten, der Sensoroutput an die nächsthöhere Ebene die rohen, digitalen Positions- und Sensordaten.

6.2 Die Punkt-zu-Punkt Bewegung

Die weitaus meisten Roboter heutzutage in der Produktion funktionieren nach dem Modell der playback robots; die Programmierung ist ähnlich dem Ändern eines Tonbands. Die Gesamtaktion ist als Folge von Einzelbewegungen gespeichert; jede Einzelbewegung kann gelöscht oder neu eingefügt werden. Die Gesamtsequenz kann im Schnellauf vor- und zurückgespielt werden.
Die Einzelschritte werden mit Hilfe einer Knopfdruck-Fernbedienung (*teach pendant*) eingegeben (teach-in), oder, für die kontinuierliche Bewegung, direkt per Hand durch Führen des Roboterarms oder eines kleinen Modells. Versteht man ein Bewegungsprimitiv (MOVE, OPERATE) als elemntares Kommando für die Maschine "Roboterarm", so entspricht dieses Niveau der Programmierung der Sequenz von herkömmlichen Maschineninstruktionen (Assembler-Mnemonics).
Charakteristisch für diese Art der ON-line Programmierung ist die Tatsache, daß die Sprache direkt interpretiert wird. Die Prozeduren haben, falls sie existieren, keine Parameter. In der folgenden Abbildung ist eine Gesamtkonfiguration gezeigt.

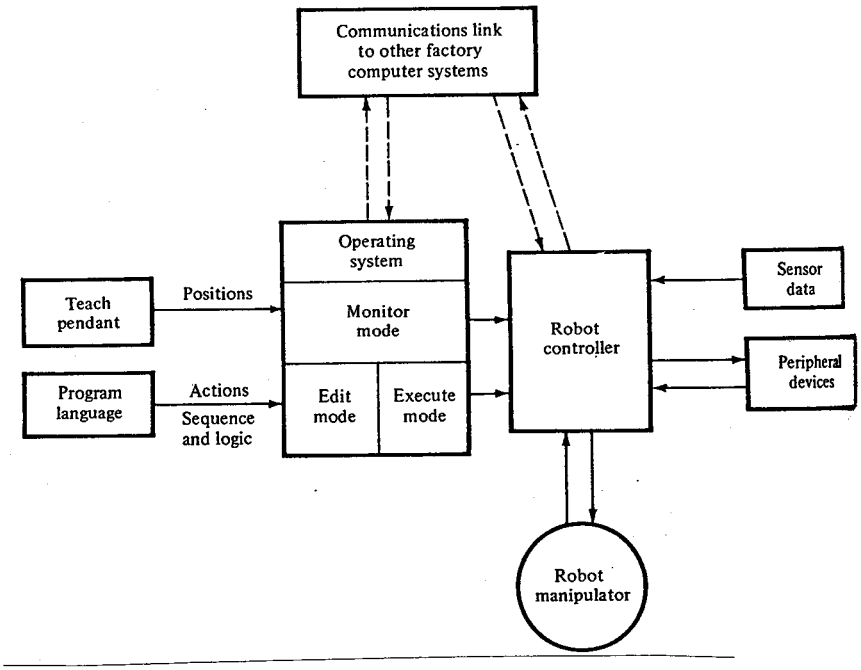


Abb. 6.2a Konfiguration zur ON-line Programmierung

6.3 Die Bewegungsebene

Die erste Generation der explizit als "Programmiersprachen" gekennzeichneten Hilfsmittel zur Robotersteuerung erlauben komplexere Operationen als die teach-in Sprachen. Zwar sind auch die elementaren Operationen (z.B. MOVE) übernommen worden, aber es existieren zusätzliche Sprachelemente wie GOTO, BRANCH, WAIT und SIGNAL für binäre Sensoren und Kontrollsignale, mit denen die Arbeit des Roboters mit den Aktivitäten der Fertigungszelle synchronisiert werden kann.
Die Prozeduren erlauben Parameter und es gibt Operatoren, mit denen *frames* (Koordinatensysteme) verschoben und gedreht werden können.
Die Sprachen werden meist interpretiert und ermöglichen manchmal, Prozeduren auf files abzuspeichern, sie bei Bedarf einzulesen und abzuarbeiten.

6.4 Strukturierte Programmiersprachen

In der zweiten Generation der Roboter-Programmiersprachen wurde versucht, verschiedene Probleme der ersten Generation zu beseitigen. Sobald man nämlich von der ON-line Programmierung direkt am Roboter sich entfernte und für die roboterunspezifischen Probleme (z.B. der Kommunikation mit der Fertigungszelle) die Programmierung teilweise auf allgemeine Computer (OFF-line) verlagerte, zeigten sich die alten Mängel der nicht-strukturierten Programmiersprachen. Durch das Fehlen von strukturierter Syntax (Blockstruktur), fehlenden Möglichkeiten zur Definition von komplexen Datentypen und unkontrollierten Schnittstellen von Prozeduren und Bibliotheksmodulen entstanden schwer verständliche, komplexe Programmstrukturen ("Spaghetti-Code"). Da sie schwer zu verstehen sind, ist auch das Verbessern und Warten sehr schwierig und zeitaufwendig.
Die 2. Generation der Roboterprogrammiersprachen trug diesen Problemen Rechnung und übernahm die wesentlichen Syntaxstrukturen (IF..THEN..ELSE, WHILE..DO.., etc) und die anderen Möglichkeiten der allgemeinen, strukturierten Computersprachen, wobei allerdings die Bewegungsprozeduren von der 1. Generation beibehalten wurden. Die ON-line Fernsteuerung dient nur noch zur bequemen Eingabe von Positionsdaten des Greifers.
Zusätzlich wurden in einigen Sprachen robotertypische Sprachelemente aufgenommen:

- einfache **Manipulation von Frames**
Beispiel: Verschieben des Frames KL mit der homogenen Matrix A durch den Befehl

MOV KL + A *(in PAL)*
MOVE KL : A *(in VALII)*

Außerdem ist die Fixierung von Frames relativ zueinander möglich, so daß bei dem Verschieben eines frames auch alle anderen umgerechnet werden.
Anwendung: Beispielsweise können die Frames von Werkteilen mit dem des Transportwagens gekoppelt werden, auf dem sie befördert werden. So werden die Werkteile automatisch richtig gegriffen.

- einfache **Referenz von komplexen Sensordaten**, beispielsweise der reellen Sensorwerte von Bilderfassungssystemen oder Kraftsensoren

- **Sprachkonstrukte für Parallelarbeit**

implizit durch die Spezifikation		explizit durch Prozeduren
IN_PARALLEL	*(in Maple)*	SendMessage (), ReceiveMessage ()
COBEGIN / COEND	*(in AL)*	

- **Sprachkonstrukte für asynchrone Ereignisbehandlung** wie Fehler, Ausnahmen oder allgemeine Nachrichten durch Interrupt- Service Routinen Beispiel: REACT

Zweifelsohne könnte man die robotertypische Sprachsyntax auch als Prozeduren formulieren. Es ist ein Trend sichtbar, bei einem neuen Roboter anstelle einer eigenen Programmiersprache eher eine hohe, allgemeine Programmiersprache wie Pascal zu verwenden und dafür eine spezielle Roboter-Laufzeitbibliothek zu schreiben. Der Vorteil dieser Vorgehensweise liegt nicht nur darin, eine weitere der zahllosen Roboterprogrammiersprachen vermieden zu haben, sondern auch in der

- **erhöhten Portabilität**
Die Standardsprache ist meist schon auf den verwendeten Prozessor portiert worden.

- **kürzeren Fertigstellung** der roboterspezifischen Sprachkonstrukte
Der Compiler muß nicht neu geschrieben oder erweitert werden.

- **modularen Erweiterbarkeit** der Sprache
Für neue Konstrukte kann auf Primitive der vorherigen Erweiterungen zurückgegriffen werden.

- **Standard-Librarys** zur Kommunikation und Synchronisation sind oft in der Zielsprache schon verfügbar.

6.5 Die aufgabenorientierte Schicht

Die strukturierten Programmiersprachen erlauben zwar eine vorprogrammierte Fehlerbehandlung, aber sie übernehmen nicht die Vermeidung von Kollisionen, falls andere Gegenstände sich in der Bahn des Greifers befinden.Um diese und andere Probleme zu lösen, muß das Programmiersystem die dreidimensionale Welt modellieren und ein ständig aktualisiertes Wissen darüber besitzen. Dieses Wissen über Typ, Ort, usw. des Manipulators, der einsetzbaren Werkzeuge, Zuführungs- und Bearbeitungsautomaten und Fließbänder ermöglicht dem System, die für eine Aufgabe nötigen Bewegungen und ihre Reihenfolge selbstständig auszurechnen. Die Befehle in einer solchen Sprache sind also nicht mehr bewegungsorientiert, sondern aufgaben- bzw. objektorientiert.

Beispiel: Der Befehl

PLACE obj1 ON obj2

verlangt, die Objekte 1 und 2 zu suchen, die Bewegung zum Aufeinanderstellen zu berechnen (wobei zur Kollisionsvermeidung auch die Positionen aller anderen Objekte beachtet werden müssen) und die Bewegung bei der Durchführung zu überwachen. Der veränderte Zustand der Welt muß sodann abgespeicher werden.

Um diese Aufgabe erfüllen zu können, benötigt das System komplexe Sensoren, meist ein Bildauswertungssystem. Wegen der komplexen Aufgabestellung gibt es noch sehr wenige derartige Roboterprogrammiersprachen. Eine davon ist die von IBM spezifizierte, bisher noch nicht (!) implementierte Sprache AUTOPASS, die mit vier Befehlskategorien arbeitet:

- Zustandänderungen (z.B. PLACE)
- Werkzeugbetrieb (z.B. OPERATE)
- Befestigungsoperationen (z.B. RIVET *("niete")*)
- Verschiedenes (z.B. VERIFY)

Da keine höheren, semantischen Kontexte beachtet werden, ergeben sich bei vagen Formulierungen zwangsläufig Mehrdeutigkeiten, so daß meist eine Debugging-Phase im Dialog mit dem Benutzer nötig ist. Darauf soll später im Kapitel 7 näher eingegangen werden.

6.6 Probleme der Roboterprogrammierung

Die Programmierung von Robotern unterscheidet sich in bestimmten Punkten wesentlich von der allgemeinen Programmierung von Computern. Ursache aller Probleme ist meistens die Tatsache, daß im Unterschied zur normalen CPU der Zustand der Maschine "Roboter" beim Ausführen eines Befehls nicht immer gleich ist. Der Zustandsraum des physikalischen Gebildes "Roboterarm und Werkzelle" ist zu groß. Die Wirkung eines Befehls ist deshalb stark kontextabhängig:

- Die **selben Prozedur** bewirkt trotz gleicher Parameter bei verschiedenen Aufrufen anderes, da die Wirkung von der Lage, Geschwindigkeit, etc des Arms abhängig ist. Auch die **Präzision** der Bewegung (Gelenkwinkel!) variiert über das Arbeitsfeld. Die für das gleiche Ziel erforderliche Trajektorie ist also verschieden, je nach Zustand des Roboterarms. Dies gilt natürlich auch für die Tatsache, ob ein Weg langsam oder schnell durchfahren wird: die optimalen Trajektorien sind unterschiedlich.
  Damit wird hier die sonst erfolgreiche Technik, aus ausgetesteten, kleinen, einfachen Einzelprozeduren hierarchisch eine komplexe Prozedur zusammenzubauen (*Bottom - up*), sehr problematisch.

- Beim **Beseitigen der Programmierungsfehler** (*Debugging*) muß zum erneuten Abarbeiten einer korrigierten Prozedur das gesamte, physikalische System auf einen reproduzierbaren, früheren Zustand zurückgesetzt werden, was nicht immer einfach ist. Darin sind auch das gerade bearbeitende Werzeug und das bearbeitete Werkstück eingeschlossen, wobei das letztere bei irreversibler Bearbeitung (z.B. Fräsen) ersetzt werden muß.

- Die **Ausnahme- und Fehlerbehandlung** muß extensiv erweitert werden, da praktisch die Ausführung aller Befehle fehlschlagen kann. Man beschränkt sich deshalb bei der Fehlerabfrage auf die wichtigsten, fehleranfälligsten Befehle.
  Voraussetzung für die nötigen Überprüfungen sind Sensoren, die zusätzliche Informationen (Servo-Information, Bildauswertung, taktile Sensoren, Entfernungsmeßgeräte, Kraftsensoren) zum Abprüfen auf plausible Werte ermöglichen.

- Bei der **Planung der Parallelarbeit** muß bereits vorher bedacht werden, welche Fehler bei den anderen Maschinen auftreten können, die für den Roboter wichtig sind. Die Ausnahmebehandlung umfaßt somit nicht nur die eigenen Fehler, sondern auch die Fehler, die in der Fertigungszelle geschehen können.

6.7 Simulation der Fertigungszelle und Roboterkontrolle

Wie in Abschnitt 6.3 angedeutet wurde, nimmt mit zunehmender Integration der Roboter in eine programmierte Fertigungszelle die Zahl der Fehler zu, die in konventionellen Programmstücken für die Steuerung der Interaktion mit der Umgebung (externe Sensordaten, Synchronisations- und Fehlermeldungen) auftreten können. Diese Programmstücke lassen sich mit den konventionellen Programmentwicklungshilfsmitteln (Debugger, Tracer, etc) OFF-line korrigieren. Simulieren wir zusätzlich mit einem geeigneten Modell das Verhalten des Roboterarms und andere beteiligte Automaten, so läßt sich mit einem solchen Simulationsprogramm (besser: "Programmentwicklungsumgebung") die Roboterprogrammierung vollkommen OFF-line ohne Roboter vornehmen.
Ein weiterer wichtiger Punkt bei der Programmierung ist die Parallelität der zu programmierenden Vorgänge. Die Planung des Zusammenspiels von mehreren, parallel arbeitenden Maschinen und Robotern ist ein

Problem der parallelen Programmierung. Um bei der Komplexität des Problems die Übersicht zu behalten und schnell verschiedene Alternativlösungen untersuchen zu können, wurden verschiedene Programmpakete entwickelt, die eine Simulation der Roboterfunktionen gestatten. Durch eine interaktive Steuerung kann der Programmierer leicht die optimalen Bewegungssequenzen der Roboterarme festlegen.
Der Roboterzyklus läßt sich dabei überschlägig durch eine Gliederung der für die Bearbeitung notwendigen Vorgänge in EInzelschritte (unabhängig vom konkret verwendeten Roboter) vorausplanen. Ein bekanntes System RTM (von Nof und Lechtman, Purdue-University) benutzt dazu folgende Funktionsprimitive (Roboterprogrammiersprache):

	Element	Symbol	Definition of element	Element parameters
Bewegungsprimitive	1	Rn	*n*-segment reach: Move unloaded manipulator along a path comprised of *n* segments	Displacement and velocity (or path geometry and velocity)
	2	Mn	*n*-segment move: Move object along path comprised of *n* segments	Displacement and velocity (or path geometry and velocity)
	3	ORn	*n*-segment orientation: Move manipulator mainly to reorient	Displacement and velocity (or path geometry and velocity)
	4	SEi	Stop on position error	Error bound
	4.1	SE1	Bring the manipulator to rest immediately without waiting to null out joint errors	
	4.2	SE2	Bring the manipulator to rest within a specified position error tolerance	
Sensorprimitive	5	SFi	Stop on force or moment	Force, torque, and touch
	5.1	SF1	Stop the manipulator when the force conditions are met	
	5.2	SF2	Stop the manipulator when the torque conditions are met	
	5.3	SF3	Stop the manipulator when either the force or torque conditions are met	
	5.4	SF4	Stop the manipulator when the touch conditions are met	
	6	VI	Vision operation	Time function
Greiferprimitive	7	GRi	Grasp an object	Distance to open/close
	7.1	GR1	Simple grasp of object by closing fingers	
	7.2	GR2	Grasp object while centering hand over it	
	7.3	GR3	Grasp object by closing one finger at a time	
Verzögerungsprimitive	8	RE	Release object by opening fingers	
	9	T	Process time delay when robot is part of the process	Time function
	10	D	Time delay when robot is waiting for a process completion	Time function

Für die verwendten Konstanten kann man beispielsweise annehmen, daß bei jeder Bewegung ein "Einschwingzeit" von 0,4 sec dazukommt. Werden Lasten von 0,5 bis 2,5 Kg benutzt, so erhöht sich dies auf 0,6 sec und bei 2,5 bis 7,5 Kg auf 0,9 sec.

Eine Beispielrechnung sei für die einfache Aufgabe angeführt, ein Werkstück von einem Fließband zum Nächsten zu befördern.

Sequence	RTM symbol	Distance, ft	Velocity, ft/sec	Delay time, s	Element time, s	Description
1	D	—	—	15.0	15.0	Await delivery
2	R1	0.1	0.1	—	1.4	Approach part
3	GR1				0.1	Grasp part
4	M1	0.1	0.1	—	1.6	Lift part
5	M1	1.25	0.5	—	3.1	Move part
6	M1	0.1	0.1	—	1.6	Approach release point
7	RE	—	—	—	0.1	Release part
8	R1	0.1	0.1	—	1.4	Depart release point
9	R1	1.25	0.5	—	2.9	Reposition
Total					12.2	

Abb.6.7a Zeittabelle einer Gesamtbewegung

In der folgenden Abbildung ist der Bildschirm eines solchen Simulationsprogrammes gezeigt; aus Effektivitätsgründen (Berechnungszeit) ist der Roboterarm nur im Drahtmodell gezeichnet.

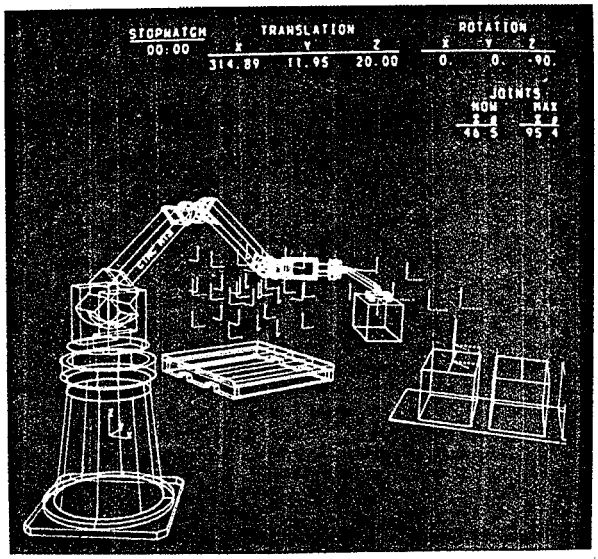


Abb. 6.7b Bildschirminhalt eines Simulationsprogrammes

Funktioniert in der Simulation alles, so ist im Prinzip das Steuerprogramm des Roboters ausreichend spezifiziert.
Die hauptsächlichsten Vorteile einer Simulation lassen sich folgendermaßen zusammenfassen:

- Es ist möglich, für sehr unterschiedliche Robotertypen eine gemeinsame Programmiersprache zu verwenden, falls es im Simulationssystem Übersetzer zur roboterspezifischen Sprache gibt. Der Code kann dann direkt als Programm in den Roboter geladen werden (*Down-load*).

- Liegen in vollintegrierten Fabriken (CIM) die Daten über das Werkstück aus CAD-Programmen vor, so können sie prinzipiell von Simulationssystemen dazu benutzt werden, die Bearbeitung und damit die Roboterbewegungssequenz zu erstellen.

- In der Simulation lassen sich leicht die Auswirkungen von Änderungen in der Synchronisation der Parallelarbeit feststellen undso die Parallelarbeit optimieren. Die notwendigen SEND/WAIT- Aufrufe werden dann automatisch generiert.

- Die teuren Roboter werden besser ausgenutzt, da sie nicht mehr zum Programmieren aus der Produktion genommen werden müssen. Oder aber: die Programmierer müssen keinen Nacht- und Wochenenddienst tun, um die Roboter ohne Produktionsverlust nach Feierabend zu programmieren.

- Die dynamischen Kräfte in schnellen Bewegungen können besser bei der Trajektorienberechnung berücksichtigt werden .

- Es besteht die Möglichkeit, automatisch auf Kollisionen in der Fertigungszelle zu prüfen.

Viele Simulationssysteme haben auch einen ON-line Modus, in dem man mit einem direkten Roboteranschluß ausschließlich diejenigen Bewegungssequenzen debuggen kann, die in der Realität nicht einwandfrei funktioniert haben.

7.0 Intelligenz und Handlungsplanung

Spezifiziert man anstelle einer Bewegung nur die Aufgabe, so muß sich der Roboter, wie in Abschnitt 6.5 beschrieben, aus dem gewünschten Handlungsziel die Handlung , d.h. die Bewegungsfolge, selbst generieren. Für eine allgemeine Aufgabenstellung ist sicher "Intelligenz" nötig, um die für die Bewegungsfolge nötige Planung der Handlung durchzuführen. Wie läßt sich die Planung nun formalisieren, um zur Lösung formale Methoden einsetzen zu können?

Betrachten wir dazu als **Beispiel** die "**Klötzchenwelt**" (*blocks world*).

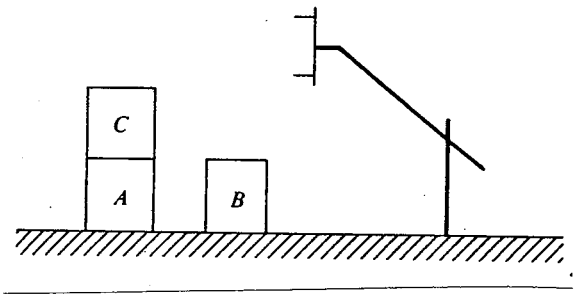


Abb. 7.0a Konfiguration von Klötzchen und Roboterarm

Ein Greifer hat die Aufgabe, mit Hilfe einer bekannten Bewegung MOVE(x,y,z), die einen Würfel x vom Würfel y nimmt und auf den Würfel z setzt, aus der obigen Konfiguration einen Turm A, B, C von Blöcken zu bauen mit Block A auf der Spitze.
Wie findet man die richtige Bewegungssequenz?
Mit dem Symbol *T* für "Tisch" sind folgende Bewegungssequenzen möglich

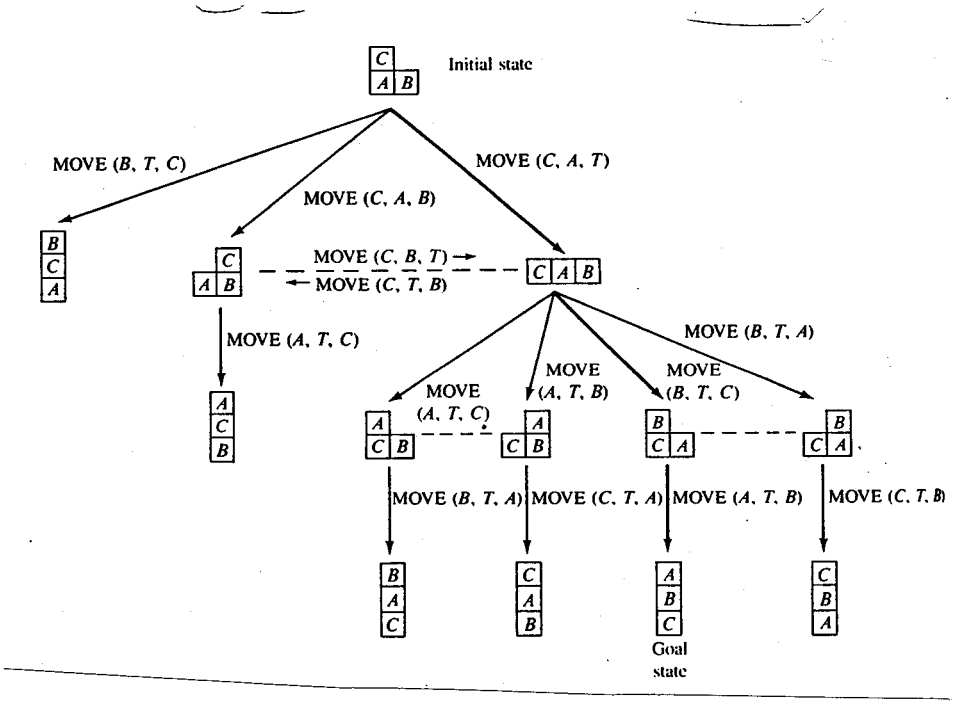


Abb. 7.0b Mögliche Bewegungssequenzen des Roboterarms

Die richtige Sequenz kann man ablesen: MOVE (C,A,T), MOVE (B,C,T), MOVE (A,T,B).
Finden kann man sie beispielsweise durch systematisches Durchsuchen des gesamten, aufspannenden Baumes *(uniforme* oder *blinde Suche)*; man spricht von **Breitensuche.**

Versucht man, mögliche Lösungen zuerst bis an ein Ende (*Terminierungsbedingung!*) zu verfolgen, so heißt dies **Tiefensuche**.

Betrachten wir ein anderes Beispiel.

Ein Roboter hat eine lange Stange, die er durch ein Labyrinth (z.B. eine Menschenmenge) transportieren soll. In Abbildung 7.0c ist die Startsituation gezeigt.

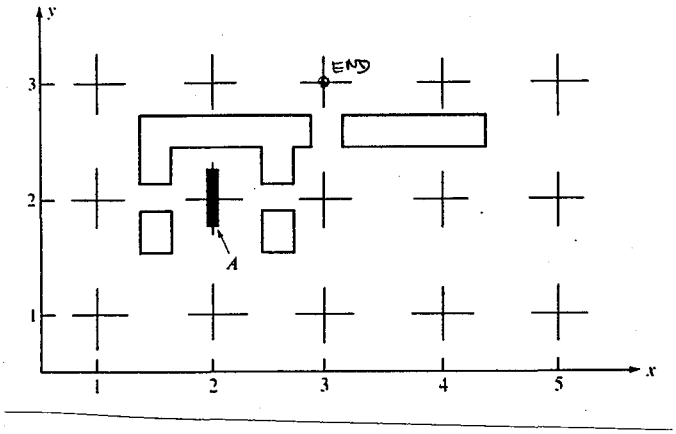


Abb. 7.0c Ortsraum des Roboters aus dem 2.Beispiel

Der Roboter kann sich in seiner Bewegungsebene (x,y) jeweils um eine Einheit in x bzw. y Richtung verschieben und den Stab um den Winkel a = 90^0 drehen.

Was muß er tun, um zur Koordinate (3,3) zu gelangen?

Sein jeweiliger Zustand wird durch die drei Variablen x,y,a charakterisiert, so daß die Menge aller möglichen Zustände in einer dreidimensionalen Grafik dargestellt werden kann:

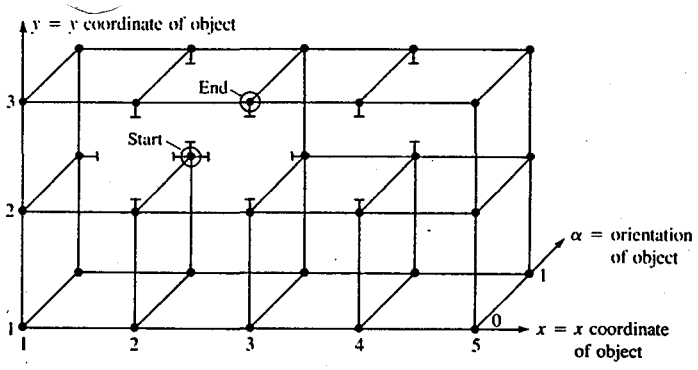


Abb. 7.0d Zustandsraum des Roboters

Die kürzeste Strecke vom Start zum Ziel ist in diesem Graphen nicht eindeutig:

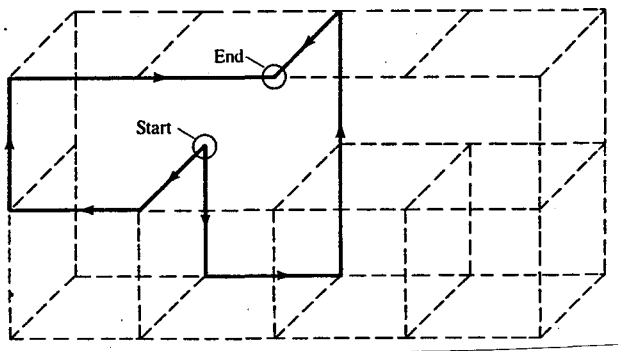


Abb. 7.0e Lösungsmenge der Aufgabe

Wie man sieht, läßt sich das Problem der Handlungsplanung vielfach als Problem beschreiben, im Zustandraum des Roboters den günstigsten Weg vom gegebenen Anfangszustand zu einem gewünschten Endzustand zu suchen. Dazwischen gibt es eine Vielzahl von verzweigenden Lösungen, die man normalerweise für die beste Lösung nicht alle absuchen kann. Vielmehr muß entweder mit einer quantitativen Bewertung der jeweiligen Situation eine Entscheidungshilfe zum Durchlaufen des Baumes geschaffen werden (**heuristische Suche**, z.B. A*-Algorithmus) oder die Zahl der möglichen Lösungen muß durch Nebenbedingungen eingeschränkt werden (z.B. PROLOG-basierte Systeme).
Allerdings stoßen bei realen Aufgaben beide Ansätze bald an ihre Komplexitätsgrenzen. Beispielsweise wurde deshalb das auf der Prädikatenlogik 1. Stufe (PROLOG!) basierende System STRIPS (*Stanford Research Institute Problem Solver*) bald so abgeändert, daß der Problemraum in hierarchische Ebenen aufgeteilt wurde. In dem neuen System ABSTRIPS (*Abstraction Based STRIPS*) müssen sich die höheren Planungsebenen nicht mehr mit den Details der Planung herumschlagen, sondern überlassen dies den unteren Ebenen.
Diese Überlegung läßt sich auch auf das in Abb. 6.0c gezeigte, hierarchische Modell der Robotersteuerung anwenden. Allerdings muß für die Handlungsplanung das Modell der Zwischenschicht verfeinert werden.

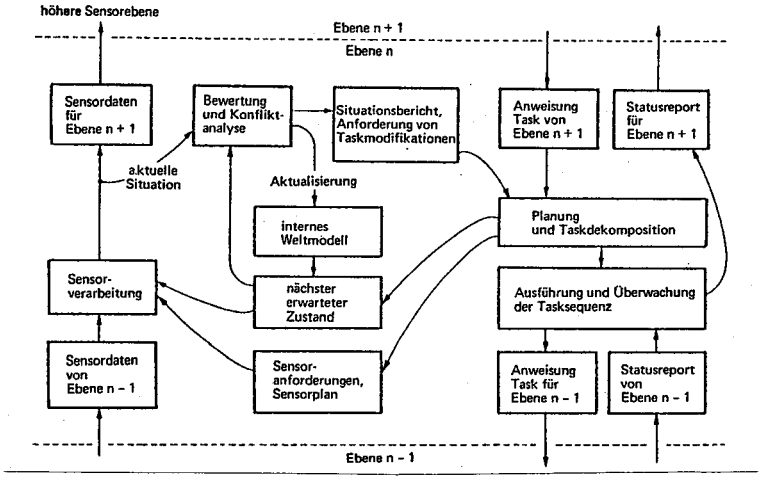


Abb. 7.0f erweiterte Schicht einer Steuerungsebene

Dabei werden Aktionen, Sensorsignale und Objekte nicht flach als einfache, prädikatenlogische Klauseln und Fakten dargestellt, sondern in strukturierter Form.
Ein Beispiel ist die Beschreibung durch *Skripte*.

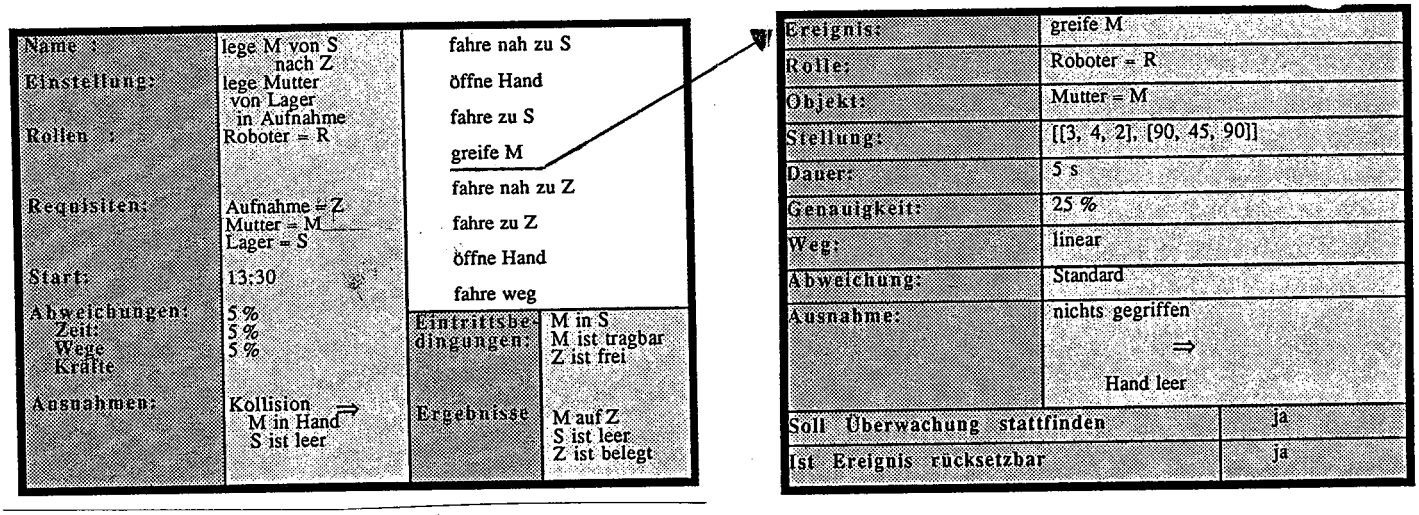


Abb.7.0g Datenstrukturen von Skripten

Das Planungssystem sieht dabei folgendermaßen aus:

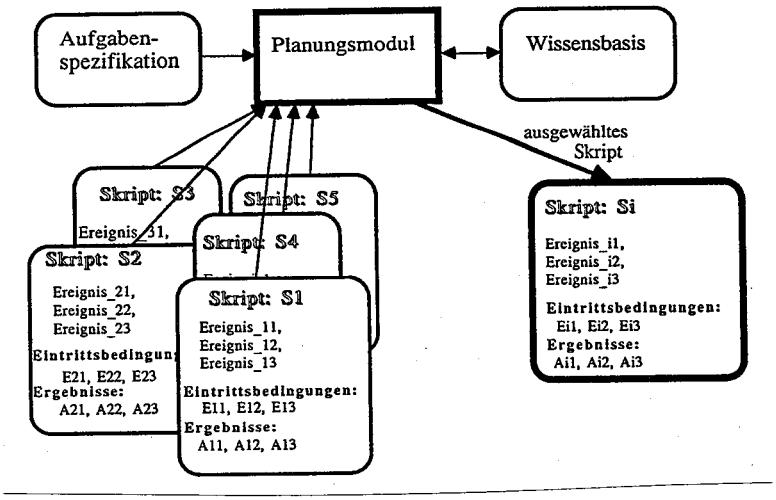


Abb. 7.0h Planungssystem mit Skripten

7.1 Autonome Roboter

Besonders hohe Ansprüche werden an das Planungssystem von selbstständigen, fahrbaren Robotern (*Autonome, mobile Roboter*) gestellt.
Diese müssen

a) zur **Kommunikation** mit der Umwelt fähig sein
b) die Umwelt durch den **Gebrauch von Modellen** verstehen können
c) eigenständig **Pläne formulieren** können
d) diese Pläne **selbst ausführen** können
e) und dazu in der Lage sein, eigene Operationen zu **überwachen**

Die Anforderungen an das Planungssystem dieser zukünftigen Generation von Robotern sind groß:

- Vorgabe von **mehrfachen Zielen** mit unterschiedlichen Dringlichkeiten
- Vorhandensein von **Randbedingungen**, die sich gegenseitig behindern
- Darstellung **komplexer Aktionen**
- Einbeziehung einer **dynamischen Umwelt** (unsicherheiten, Zeit)
- **Flexible** Aufgabendurchführung
- **Adaption** an neue, ähnliche Aufgaben
- **Synchronisierte Planung** auf mehreren Ebenen.

Um dies durchführen zu können, muß ein Roboter auch Abstrahieren und Lernen können, was aber bisher (trotz einiger interessanter Ansätze) noch immer ungelöste Probleme sind.

8.0 Neuronale Modelle motorischer Leistungen

Betrachtet man im Unterschied zu den vorigen Kapiteln das, was ein Roboter heutzutage *nicht* leisten kann, so findet man im Vergleich zum Menschen viele Mängel: die Koordination von zwei Armen macht Probleme, die ballistische Steuerung von zwei Beinbewegungen, und insgesamt, die Integration aller Funktionen bei der Planung. Dabei stößt man immer wieder auf das Problem, in real-time Bewegungsvariationen errechnen zu müssen und Rückkopplungen auf verschiedenen Ebenen wirken zu lassen, ohne das dies explizit programmiert werden kann.
Im scharfen Gegensatz dazu funktionieren wir Menschen durch einfaches Training ganz gut, ohne daß uns jemand bis in die elementarsten Muskelanspannungen erklären muß, was wir genau tun sollen um ein Ziel zu erreichen.
Wir betrachten deshalb im folgenden Abschnitt 8.1 analytisch die Funktionen der menschlichen Muskelkontrolle und wollen dann in Abschnitt 8.2 synthetisch Modelle zur Bewegung untersuchen, die mathematisch gut formuliert werden können.

8.1 Die menschliche Muskelkontrolle

Die Kontrolle der Muskelerregung verteilt sich über eine Hierarchie von phylogenetisch (entwicklungsgeschichtlich) unterschiedlich alten Hirnteilen. Die verschiedenen Schichten dieser Hierarchie lassen sich ungefähr analog zu dem uns bekannten Schema (s. Abschnitt 7.0) von Bewegungsplanung, Bewegungsprogrammierung und Bewegungsausführung (*Dispatching*) zuordnen:

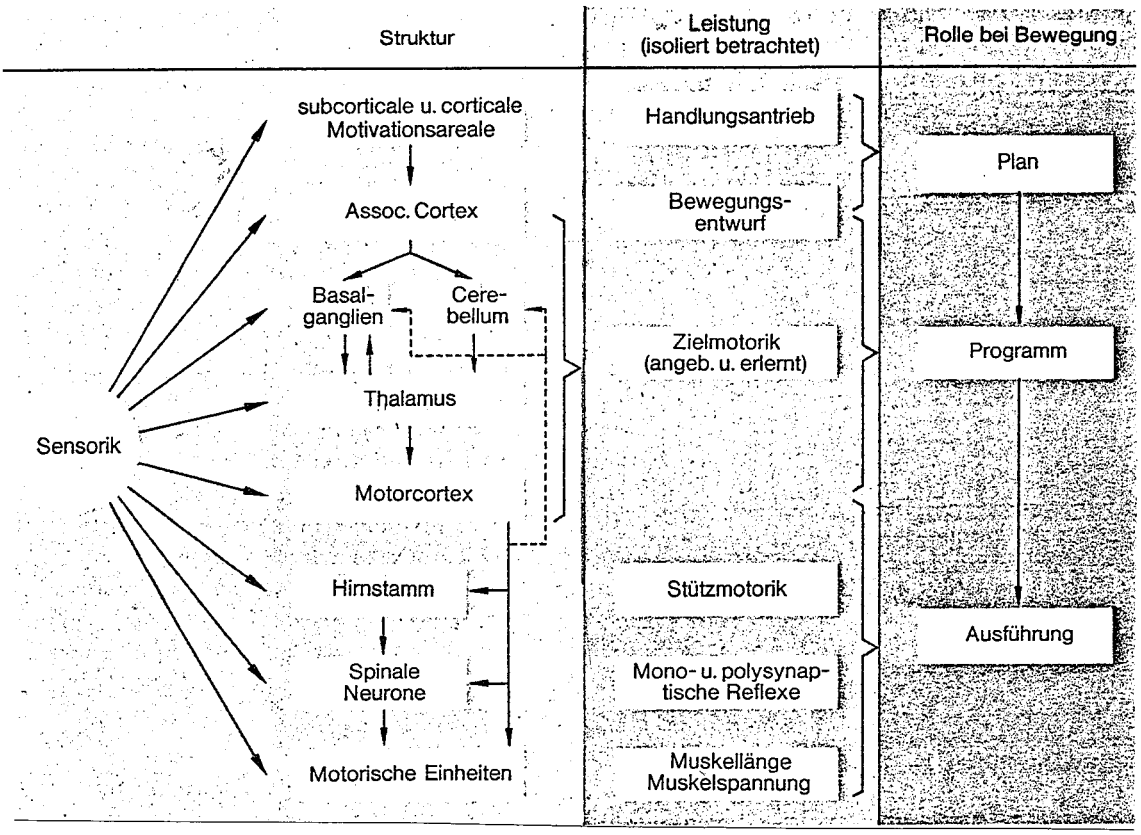


Abb. 8.1a Übersicht über den Aufbau des motorischen Systems

Beginnen wir mit der Betrachtung der am besten erforschten, untersten Schichten.

Die Muskeln

Vom Motorkortex (s. Abb.8.1a und 8.1f) führen neuronale Leitungen (*Axone*, hier *alpha-Fasern*) zu den Muskeln und enden dort auf den *Endplatten*. Gelangt eine Erregung von dem Motoneuron durch das Axon auf die Endplatten, so wird ein Strom von Kalziumionen ausgelöst, der eine chemische Reaktion bei einer Substanz *ATP* bewirkt. Um die Wirkung dieser Reaktion besser zu verstehen, betrachten wir den Querschnitt einer Muskelfaser etwas näher.

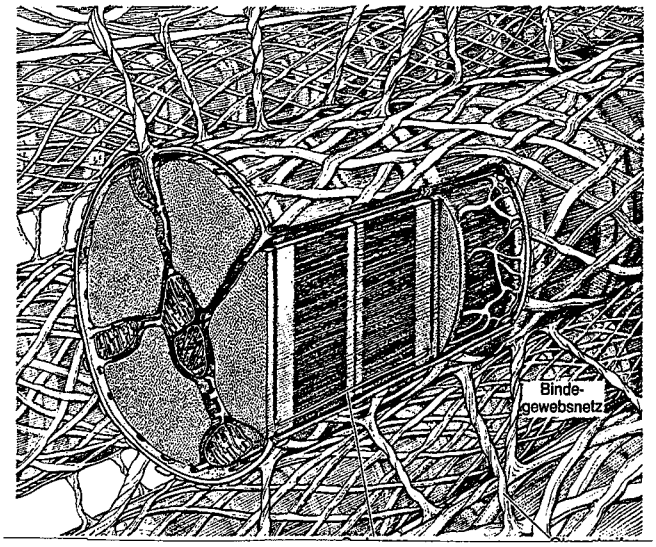


Abb. 8.1b Muskelfaser im Schnitt

Jede Muskelfaser der Skelett- und Herzmuskulatur ist in einzelne, kleine Abschnitte (*Sarkomer*) aufgeteilt und erscheint deshalb von außen wie *quergestreift*.
Jedes Sarkomer besteht aus periodisch verzahnten *Myosin-* und *Actin*filamenten. Das Myosin-Molekül "berührt" dabei mit dem Molekül-"Kopf" das Actin (Abb. 8.1c links). Bei der Aktivierung bewirken die einströmenden Kalziumionen eine Spaltung des ebenfalls anwesenden ATP, was wiederum über Anlagerungen usw. dazu führt, daß sich der "Kopf" schräg krümmt und kurzzeitig eine zusammenziehende Kraft bewirkt (Abb. 8.1c rechts).

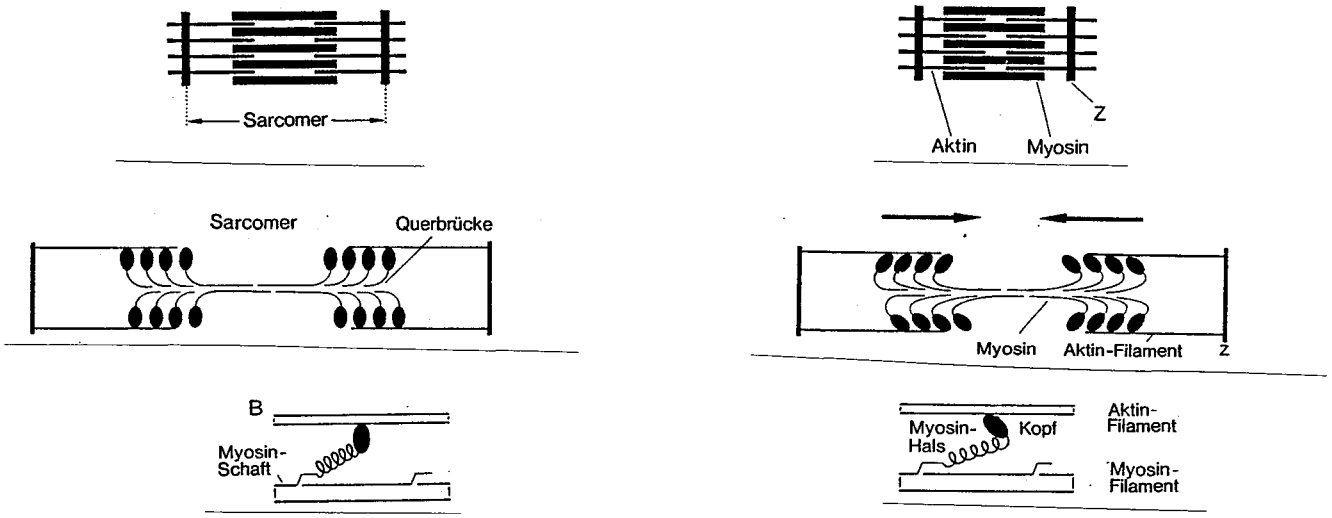


Abb. 8.1c chemische Erzeugung der Muskelkraft

Dies geschieht periodisch und asynchron bei allen Molekül-"Köpfen", so daß im Mittel eine kontinuierliche Kraft resultiert und das Sarkomer sich verkürzt. Verkürzt es sich sehr stark, so gleiten die dünnen Aktin-Filamente übereinander und der Muskel "schwillt an".
Ein Axon erregt in der Regel mehrere Muskelfasern, die zusammen als *motorische Einheit* (ME) bezeichnet werden. Da jede motorische Einheit bei einem Erregungsimpuls (*Spike*) voll kontrahiert, ist die resultierende Kraft nicht nur von der Erregungsfrequenz (Spikefrequenz) abhängig, sondern auch von der Zahl der Fasern pro motorischer Einheit. Je feiner die Kontraktion dosiert werden muß, um so weniger Fasern werden von jeder ME kontrolliert und um so mehr Neuronen müsen aktiviert werden, um eine bestimmte Kraft zu generieren.

Beispiel: Augenmuskel 1740 ME mit 13 Fasern pro ME und 0,1 g pro ME
Bizeps 774 ME mit 750 Fasern pro ME und 50g pro ME

Die bei der Verkürzung des Muskels auftretende Kraft ist stark von dem Zustand der Filamente abhängig und erreicht ein Maximum, wenn möglichst viele Myosin-Köpfe die Gegenseite berühren:

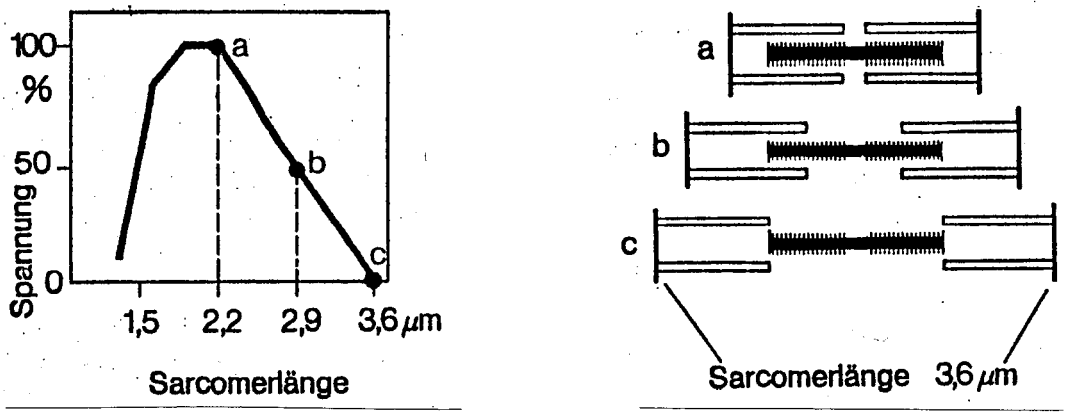


Abb. 8.1d Kontraktionskraft und Sarkomerlänge

Die durch die maximal eingespeiste chemische Leistung erzielte Muskelleistung (Arbeit pro Zeiteinheit) ist natürlich konstant, so daß eine große Verkürzungsgeschwindigkeit nur bei geringen Kräften möglich ist.

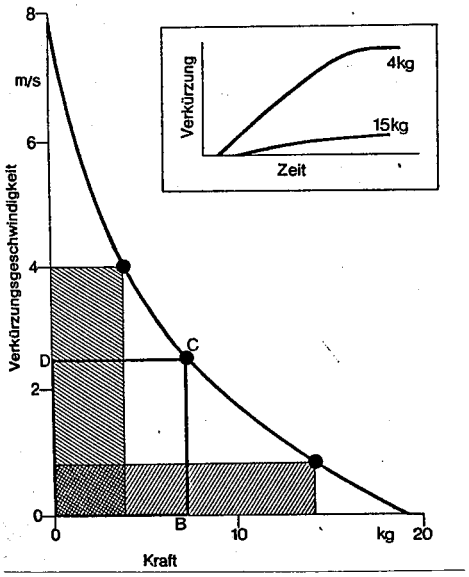


Abb. 8.1e Muskelleistung und Verkürzungsgeschwindigkeit

Die erzielte Kraft ist von der Vordehnung und der Kontraktionsgeschwindigkeit sowie dem Ermüdungszustand des Muskels (ATP-Versorgung!) abhängig und ist deshalb ohne Rückkopplung nicht exakt von den Neuronen steuerbar. Für die exakte Kontrolle für Feinbewegungen verfügen die Muskeln noch über verschiedene Sensoren, die die Wirkung der Muskelkontraktion messen.

Muskelsensoren

Die Muskelsensoren sind im Prinzip Dehnungssensoren und existieren in zwei Typen:

1) **Muskelspindeln**
In den Muskeln eingelagert existieren in schmalen, abgekapselten Stäben sog. Muskelspindeln, die über dünne Axone (*gamma-Fasern*) extra erregt werden können. Um diese Spindelfasern gewickelt sind die Sensorfasern, die die Spindeln verlassen.
Es gibt zwei Arten von Spindeln: große Spindeln, deren Sensoren auf die Dehnungsgeschwindigkeit x reagieren (*dynamische Ia-Fasern*) und kleinere Spindeln, deren Sensoren auf die Dehnung x selbst reagieren (*statische II-Fasern*).

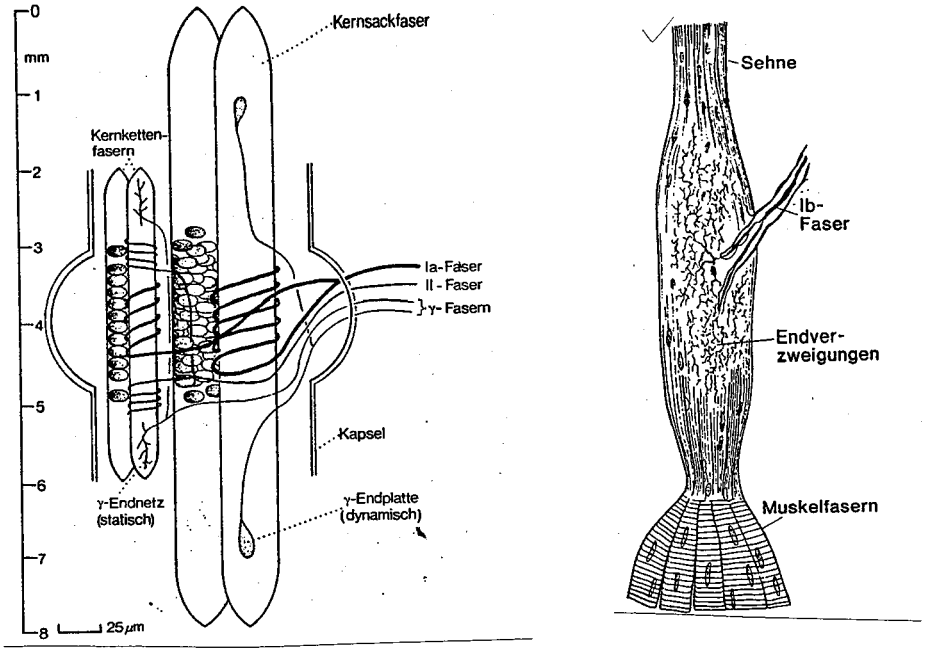


Abb. 8.1f Muskelspindeln und Golgi-Sehnenorgane

2) **Sehnenorgane**
Am Ende der Muskelfasern auf der Sehne sitzen noch spezielle Zellen, die von Ib-Fasern abgeleitet werden (s. Abb. 8.1f rechts). Diese senden ein Erregungssignal, was weitgehend von der Kraft auf den Muskel bestimmt ist.

Bei den menschlichen Sensoren werden also prinzipiell die drei Variable *Kraft (Drehmoment), Position* und *Winkelgeschwindigkeit* erfasst.

Reflexe

Steht ein Tier still auf allen vier Beinen, so wird die statische Schwerkraftkompensation nicht voll von den höheren Instanzen gelenkt, sondern (wie sich durch Amputation zeigen läßt) von einer Rückkopplung Sensor-Rückenmark-Muskel geregelt (*Haltetonus*). Stören wir diesen Regelkreis durch eine Muskeldehnung

(Klopfen auf den Muskel), so wird sofort durch die Dehnung der Spindelmuskelfasern eine Aktivierung der alpha-Motoneurone bewirkt (*Reflexbogen*). Eine zweite Möglichkeit, die Hauptmuskeln zu beeinflussen, ist über die Aktivierung der gamma-Motoneuronen und somit der Spindelmuskelfasern, die über die Spindelsensoren den Reflexbogen aktivieren.
Die Reflexe sorgen also hauptsächlich für die Ausführung von vorprogrammierten, einfachen Servomechanismen (*Stützmotorik*).

Das Kleinhirn

Zur Bewegung ist das Kleinhirn nicht unbedigt nötig; bei Patienten ohne Kleinhirn (z.B. Krebsoperation) ist immer noch eine langsame, etwas wackelige Bewegung möglich.
Das Kleinhirn scheint die Koordination der Stützmotorik mit genauen, zielgerichteten Bewegungen oder schnellen Bewegungen ohne Rückkopplung (z.B. schnelle, gelernte, ballistische Bewegung im Sport) zu übernehmen (*Bewegungskontrolle*).

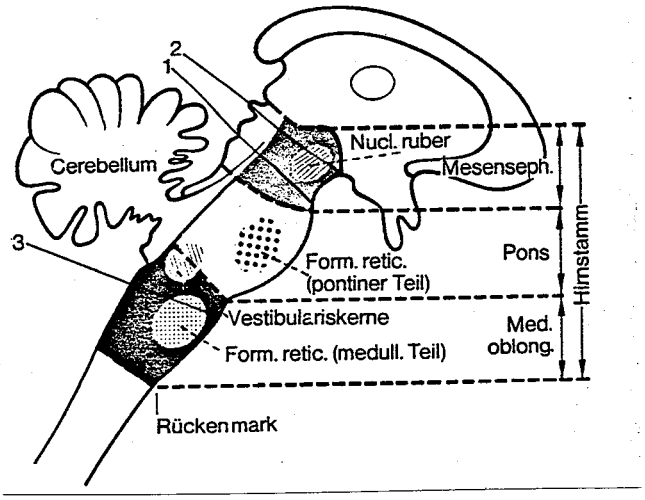


Abb. 8.1e Lokalisierung des Kleinhirns (Cerebellum)

In der folgenden Abbildung sind Modelle der zielgerichtetn Bewegung und der schnellen Bewegung gezeigt; die vom Kleinhirn beteiligten Teile sind mit einer Textur gekennzeichnet.

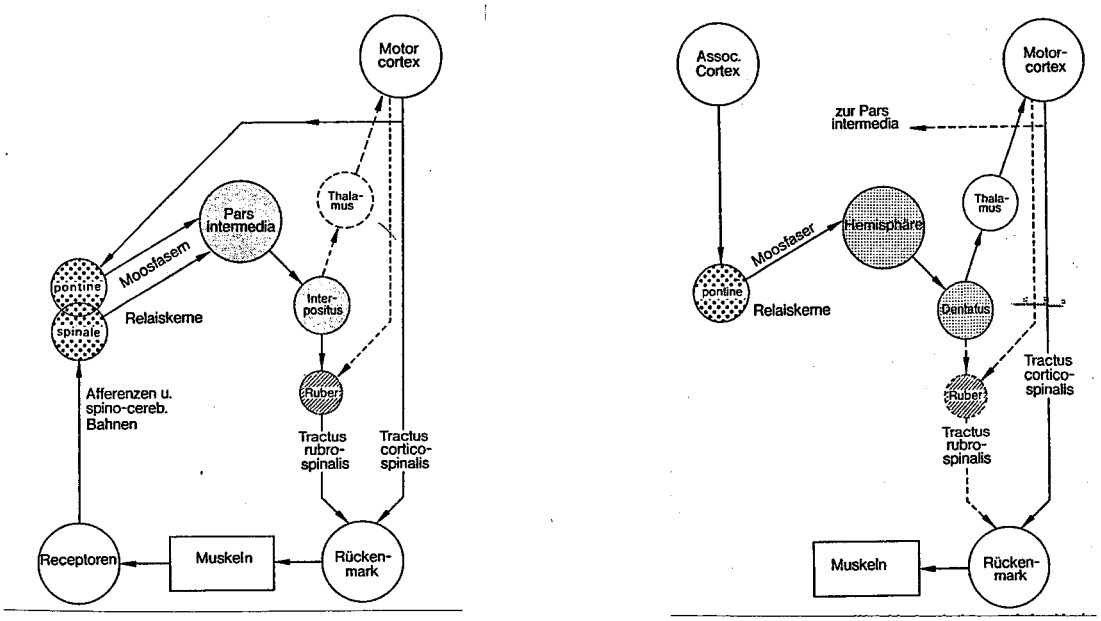


Abb. 8.1f Modelle der Kleinhirnfunktion

Allerdings sind diese Modelle nicht gesichert; die Funktion des Kleinhirns ist noch weitgehend unerforscht.

Der Motorkortex

Die Motoaxone (alpha-Neuronen), die direkt die Muskeln innervieren, kommen von sog. *Pyramidenzellen*, die im Großhirn in einem langen, schmalen Areal (Motorkortex) lokalisiert sind. Durch Reizversuche zeigte sich, daß nachbarschaftliche Motoneuronen auch benachbarte Muskeln ansprechen (*Somatopie*). Eine Karte der den Gehirnarealen entsprechenden Muskelregionen ist in Abb.8.1f rechts zu sehen.

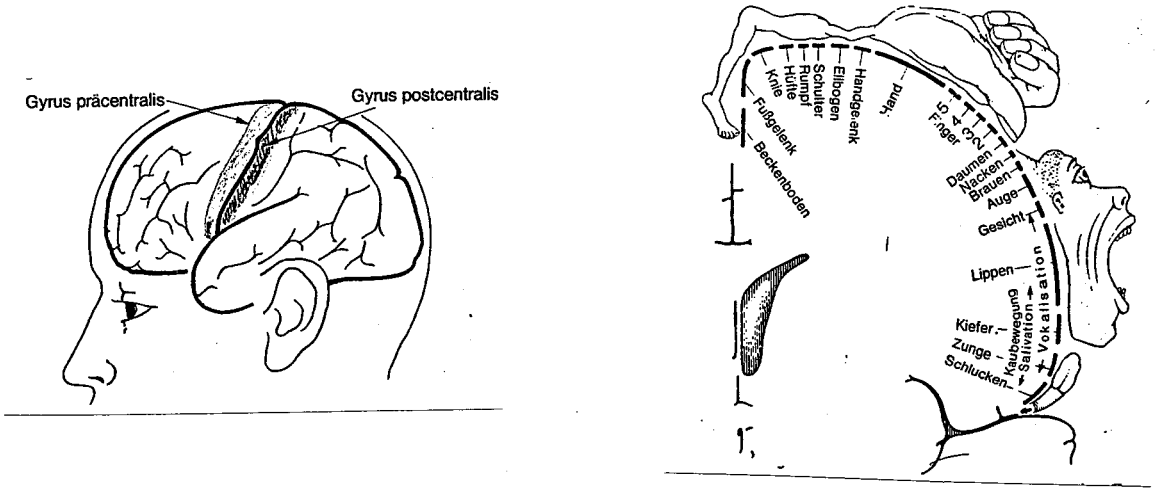


Abb. 8.1f Lage des Motokortex (Gyrus präcentralis) und die Somatopie

Das dem Motokortex angrenzende Gebiet (Gyrus postcentralis) ist interessanterweise als Empfangsgebiet des sensorischen Nervensystems bekannt; es existiert ebenfalls eine entsprechende somatopische Karte. Durch die enge Nachbarschaft spricht man auch von einem senso-motorischen Cortex.
Der Motorkortex ist histologisch (gewebsmäßig) in ca. 80 Mikrometer dicke, säulenartige Teile untergliedert. Innerhalb jeder histologischen Säule sind die Pyramidenzellen weitgehend übereinander angeordnet. Aus Ableitungen mit Mikroelektroden weiß man, daß mehrere histologische Säulen zusammen eine gemeinsame Erregung oder Hemmung bewirken. Sie sind damit funktionelle Einheiten, funktionelle Säulen von ca. 1 mm Dicke. Die Erregungszustände der Pyramidenzellen innerhalb jeder Säule können sehr unterschiedlich sein (erregend oder hemmend); gemeinsames Merkmal ist die Wirkung auf die Muskeln (Beuger und Strecker) *eines* Gelenks. Aus diesem Grund glaubt man, daß nicht direkt die Muskeln im Motorkortex repräsentiert sind, sondern Bewegungen. Die genauen Funktionen sind allerdings unbekannt. Es wird vermutet, daß im Motorkortex lediglich die zu einer Bewegung notwendigen Muskeln ausgewählt werden (*Bewegungsprogramm*).
Aus Gehirnpotentialableitungen (EEG) weiß man, daß jeder Bewegung ein sehr unscharf im Gehirn lokalisierbares Potential (*Bereitschaftspotential*) ungefähr 800 ms vorangeht. Die Planung der Bewegung scheint deshalb große Teile des Gehirns zu beteiligen und ziemlich komplex zu sein.

8.2 Roboterkontrolle mit neuronalen Netzen

Zu dem Funktionsmodell des komplexen, sequentiellen von-Neumann Computers gibt es verschiedene, alternative, parallele Modelle. Die interessantesten und derzeit sehr aktuellen Alternativen sind die **konnektionistischen Modelle.** Bei diesem Ansatz werden zur Leistungssteigerung und parallelen Informationsverarbeitung viele Einheiten miteinander verbunden. Im Unterschied zu Multi-Prozessor- und Multi-Computernetzwerken handelt es sich nicht um wenige, komplexe Einheiten, sondern um viele Einheiten geringer Komplexität. Durch historische und formale Ähnlichkeiten mit formalen Neuronen werden solche Netze als **Neuronale Netze** bezeichnet und bilden den Forschungsgegenstand einer zur Zeit stark expandierenden Gruppe von Psychologen (Modelle der Sensorverarbeitung), Physikern (Spin-Glas Modelle), Neurobiologen (Theorie des Nervensystems) und, last not least, den Informatikern (Parallele, fehlertolerante Computerarchitekturen).
Aus den vielen existierenden Modellen wollen wir für unser Problem, einen Roboter zu steuern, das von dem finnischen Physiker T. Kohonen entwickelte Modell der *topologie-erhaltenden Abbildung* herausgreifen.

Topologie-erhaltende Abbildungen

Sei ein Ereignis, z.B. ein Punkt im Kartesischen Koordinatensystem, durch einen Vektor **x** beschrieben. Der Vektor **y** sei linear von **x** bestimmt:

$$\mathbf{y} = \mathbf{M}\,\mathbf{x}$$

Die Matrix **M** enthält die Gewichte M_{ij}, mit der die Komponenten x_j von **x** in die einzelnen Komponenten y_i eingehen. Ordnen wir im Hardware-Modell jeder Komponente x_j eine Input-Leitung (Dendriten) zu und jedem y_i eine Ausgabeleitung (Axon), so modellieren die Gewichte als Stärke der Verbindungsknoten die Synapsen der neuronalen Wirklichkeit.

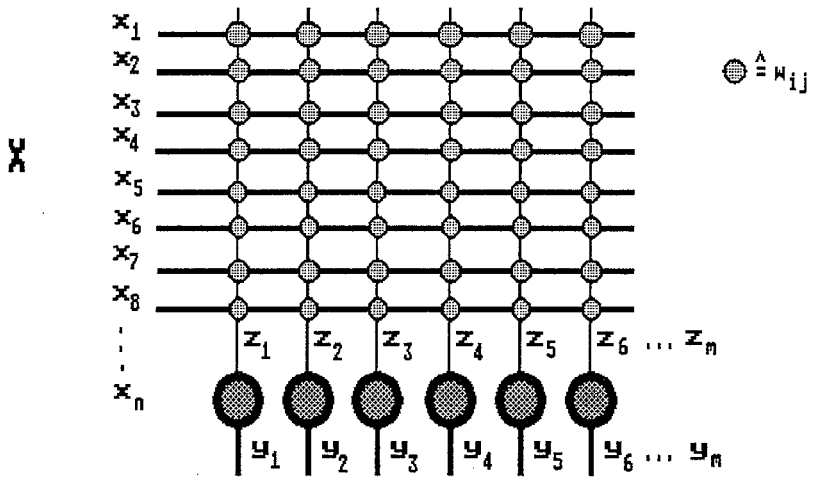


Abb. 8.2a Hardwaremodell eines Neuronalen Netzes

Die Matrix **M** definiert damit eine lineare Abbildung der **x** aus einer Menge **X** auf die **y** aus **Y**. Kohonen gab nun an, wie man die Gewichte iterativ zu verändern hat, damit benachbarte **x** am Ende der Iteration auch auf benachbarte Komponenten y_i abgebildet werden. Ist beispielsweise $\mathbf{x} = (x_1, x_2)$ ein Punkt auseiner Fläche **X** und sind die Ausgabeinheiten y_i in Form einer Flächenmatrix angeordnet, so läßt sich jedem y_i der Matrix ein viereckiges Gebiet aus **X** zuordnen. Dabei ist sogar unerheblich, ob **x** ein-, zwei- oder n-dimensional ist.

Wichtig ist allein die Tatsache, daß bei der angestrebten, topologie-erhalteneden Abbildung benachbarte Punkte aus **X** auch auf benachbarte Punkte aus **Y** abgebildet werden. Es ist dabei von geringer Bedeutung, wie die Nachbarschaft aussieht. In Abb.8.2b sind zwei Beispiele gezeigt.

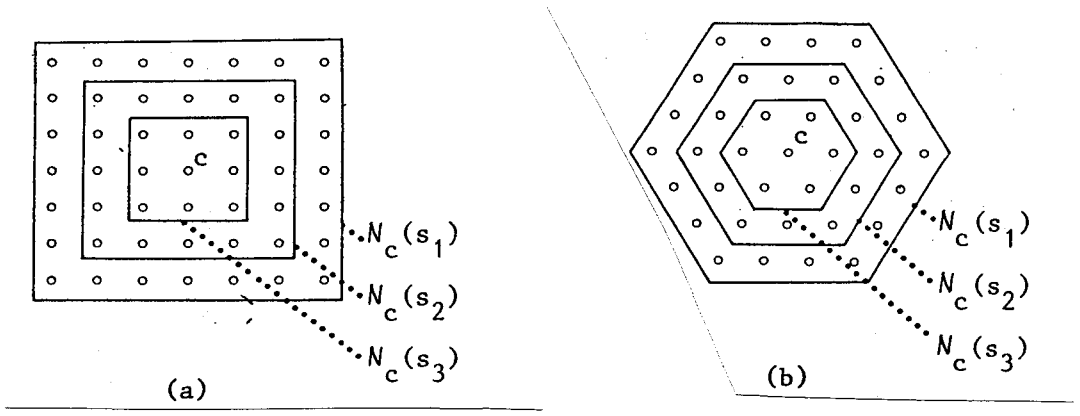


Abb. 8.2c Nachbarschaften N_c um das Zentrum y_c

Die Antwort bei y_i wird zweifelsohne dann am stärksten sein, wenn mit

$$y_i = \sum_j M_{ij} x_j =: \mathbf{m}_i \mathbf{x}$$

bei Eingabe von einem $\mathbf{x}^i$ die Spalte $\mathbf{m}_i$ der zu y_i gehörenden Gewichte am stärksten korrelliert ist. Liegen die $\mathbf{x}^i$ nur als Erwartungswerte einer Wahrscheinlichkeitsverteilung vor, so lassen sich die $\mathbf{m}_i$ mit Hilfe einer auf die Nachbarschaft begrenzten , stochastischen Iteration in den Zeitschritten t_k lernen:

1) Suche dasjenige $\mathbf{m}_c$, das von $\mathbf{x}$ den **kleinsten Abstand** hat

$$\| \mathbf{x}(t_k) - \mathbf{m}_c(t_k) \| = \min_j \| \mathbf{x}(t_k) - \mathbf{m}_j(t_k) \| \qquad (8.2a)$$

2) **stochastische Iteration:**

Nur für alle y_l aus der Nachbarschaft von y_c verändere die Gewichte

$$\mathbf{m}_l(t_{k+1}) := \mathbf{m}_l(t_k) + f(t_k) [\, \mathbf{x}(t_k) - \mathbf{m}_c(t_k)] \qquad (8.2b)$$

mit den üblichen Bedingungen der stochastischen Iteration für f(t)

$$\lim_{k \to \infty} f(t_k) \to 0 \qquad \sum_{k=0}^{\infty} f(t_k) > \infty \qquad \sum_{k=0}^{\infty} f(t_k)^2 < \infty$$

Beispiel : $f(t_k) = 1/k$

Es läßt sich analytisch zeigen, daß diese Iteration tatsächlich das gewümschte Ergebnis bringt. Dabei ist wichtig, jeweils die Nachbarschaft um y_c herum mitzuverändern und nicht nur y_c. Dadurch werden stagnierende $\mathbf{m}_i$ von gut konvergierenden $\mathbf{m}_i$ "mitgezogen" und es erfolgt eine gute Gesamtkonvergenz.
Als Beispiel sei in Abb. 8.2c die Entwicklung der $\mathbf{m}_i$ demonstriert. Dabei wurden die $\mathbf{x} = (x_1, x_2)$ zufällig gleichverteilt aus einer Fläche ausgewählt, von der die Umrandung (Grenze) in den Bildern der Iterationsfolge eingezeichnet ist. Die Gewichte $\mathbf{m}$ wurden im Koordinatensystem der $\mathbf{x}$ ebenfalls eingezeichnet. Zusätzlich wurden direkt benachbarte $\mathbf{m}_i$ miteinander verbunden, so daß die Menge von gleichmäßig verteilten $\mathbf{m}$ das Bild eines Koordinatennetzes ergibt. Die Iteration der $\mathbf{m}$ zeigt damit auch die Abbildung der Nachbarschaftsmetrik der Eingabewerte $\mathbf{x}$ auf die Metrik der Ausgabewerte $\mathbf{y}$.

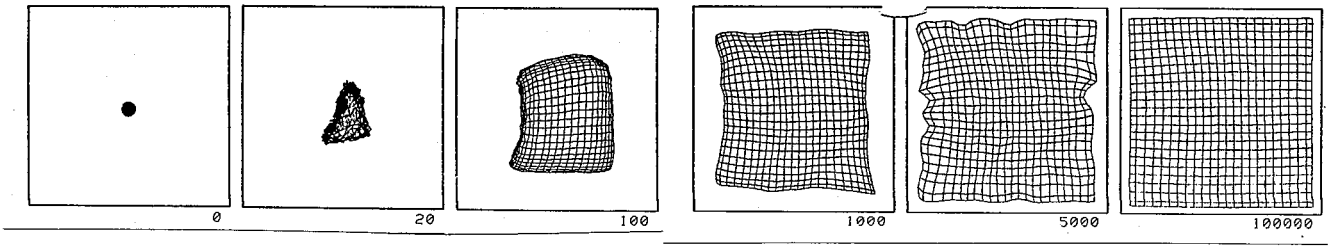


Abb. 8.2c Konvergenz der $\mathbf{m}_i$

In Abb.8.2d sind die Iterationszustände gezeigt, bei denen die Wertemenge der **x** kein Quadrat, sondern ein Dreieck ist. Dabei sind die Nachbarschaften von **y** eindimensional definiert worden, so daß sich eine Dimensionsreduzierung bei dieser Abbildung ergibt.

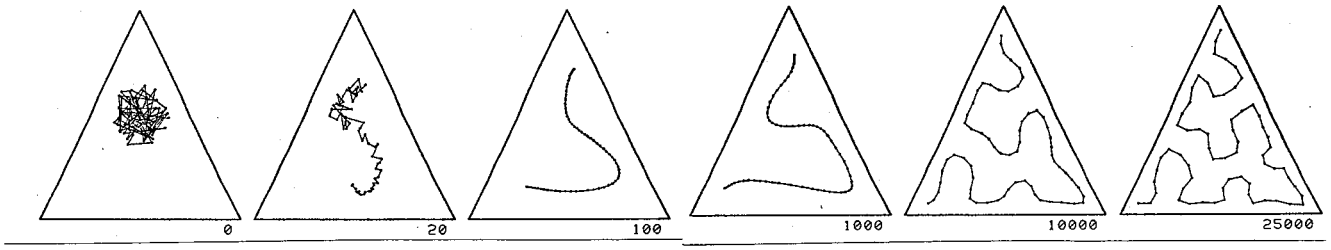


Abb.8.2d Dimensionsreduzierung bei der topologie-erhaltenden Abbildung

Roboterpositionierung

Wie läßt sich mit einer solchen topologie-erhaltenden Abbildung ein Roboter steuern?
Angenommen, wir erhalten als Koordinaten nur die Gelenkkoordinaten. Um die Kartesischen Koordinaten zu erhalten, müßten wir erst mit etlichen aufwendigen Multiplikationen mit Sinus und Cosinus eine homogene Transformation durchführen. Benutzen wir die Gelenkkoordinaten des Arms direkt als Eingabe $\mathbf{x} = (\theta_1,\theta_2)$ für unser Neuronennetz, so kann der Arm durch verschiedene, sequentiell durchgeführte Positionierungen iterativ eine Abbildung auf das Kartesiche Koordinatensystem der y_i bewirken. In Abb.8.2e ist das Ergebnis eines simulierten Trainings aus zwei Armen gezeigt, die ein 4-dim. $\mathbf{x} = (\theta_1,\theta_2,\theta_3,\theta_4)$ erzeugen.

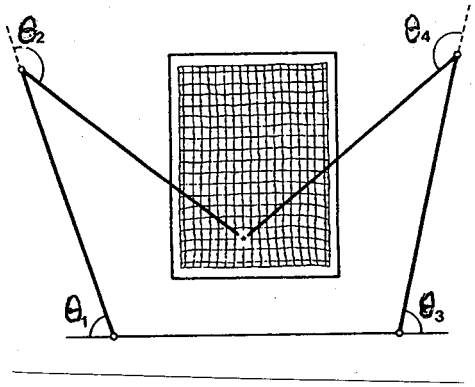


Abb. 8.2e Abbildung der Gelenkkoordinaten auf Kartesische Koordinaten

Wie wir in der Abbildung sehen, bewirkt die gleichmäßige Wahrscheinlichkeitsverteilung der Punkte im Kartesischen Koordinatensystem trotz Umrechnung in Gelenkkoordinaten eine kartesiche Metrik der y_i.

Roboterbewegung

Die theoretischen Physiker Ritter und Schulten (TU München 1987) erweiterten diesen Ansatz auf ballistische Bewegungen. Sie gingen davon aus, daß nach einer Positionsfeststellung (Abbildung der Gelenkkoordinaten auf Kartesische Koordinaten) diejenige Kraft gesucht ist, die den Roboterarm auf eine gewünschte Geschwindigkeit **u** bringt. Bei einer Drehbewegung (keine Schwerkraft) besteht das nötige Drehmoment **d**(t) hauptsächlich aus Trägheitskraft und Corioliskraft

$$\mathbf{d}(t) = \mathbf{A}\,\ddot{\mathbf{x}} + \mathbf{B}(\mathbf{x})\dot{\mathbf{x}}\dot{\mathbf{x}}$$

mit der Matrix **A** und dem 3-dim Tensor **B**.
Nehmen wir an, daß in jeder Kartesischen Koordinate ein kurzer Kraftstoß ausgelöst wird (ballistische Armbewegung) mit $\mathbf{d}(t) = \mathbf{F}\,\mathrm{delta}(t)$, so resultiert als Instantgeschwindigkeit $\mathbf{u} = \dot{\mathbf{x}}$ mit

$$\int_{-\infty}^{+\infty} \mathbf{F}(t)\,\mathrm{delta}(t-t')\,dt' = \qquad \mathbf{F}(t) = \mathbf{A}(\mathbf{x})\,\dot{\mathbf{x}}$$

oder

$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x})\,\mathbf{F}(t)$$

Bei einem sehr kurzen Kraftimpuls wirkt sich im Grenzwert die Corioliskraft nicht weiter aus und wir erhalten eine lineare Beziehung zwischen aufgewendeter Kraft und resultierender Geschwindigkeit.
Da die Matrix **A**(**x**) von der Kartesischen Position abhängt, muß auch sie iterativ bestimmt werden. Die Iterationsvorschrift ist analog zu Gleichung (8.2b). Nach der Bestimmung der Position **x** bzw. dem Neuron y_c erfolgt Schritt 3 :

3) Ermittle mit $\mathbf{F} := \mathbf{A}(y_i,t_k)\,\mathbf{u}$ die nach den bisherigen Iterationen nötige Kraft, um die Geschwindigkeit **u** zu erhalten.

4) Schätze mittels der gemessenen, tatsächlichen Geschwindigkeit **v** die Koeffizienten neu:
$$\mathbf{A}(y_c,t_{k+1}) := \mathbf{A}(y_c,t_k) + f(t_k)\,[\mathbf{F} - \mathbf{A}(y_c,t_k)\mathbf{v}]\mathbf{v}^T \qquad (8.2c)$$

5) Aktualisiere alle Werte der Nachbarschaft mit
$$\mathbf{A}(y_i,t_{k+1}) := \mathbf{A}(y_i,t_k) + h(y_c,y_i,t_k)\,[\mathbf{A}(y_c,t_{k+1}) - \mathbf{A}(y_i,t_k)]$$

wobei $h(y_c,y_i,t_k)$ eine Gaußfunktion um y_c ist mit der die Bereiche außerhalb der Nachbarschaft "weich ausgeblendet" werden.

Zur graphischen Veranschaulichung der (streng bewiesenen) Konvergenz des Verfahrens wird parallel zur Entwicklung der Metrik von **m** die Entwicklung von **A**(**x**) an jedem der 100 Punkte der 10x10 Matrix gezeigt. Dazu wird die tatsächliche Geschwindigkeitsrichtung des Greifers eingezeichnet, wenn an dieser Stelle eine Bewegung in x_1 und eine in x_2 Richtung verlangt wäre.

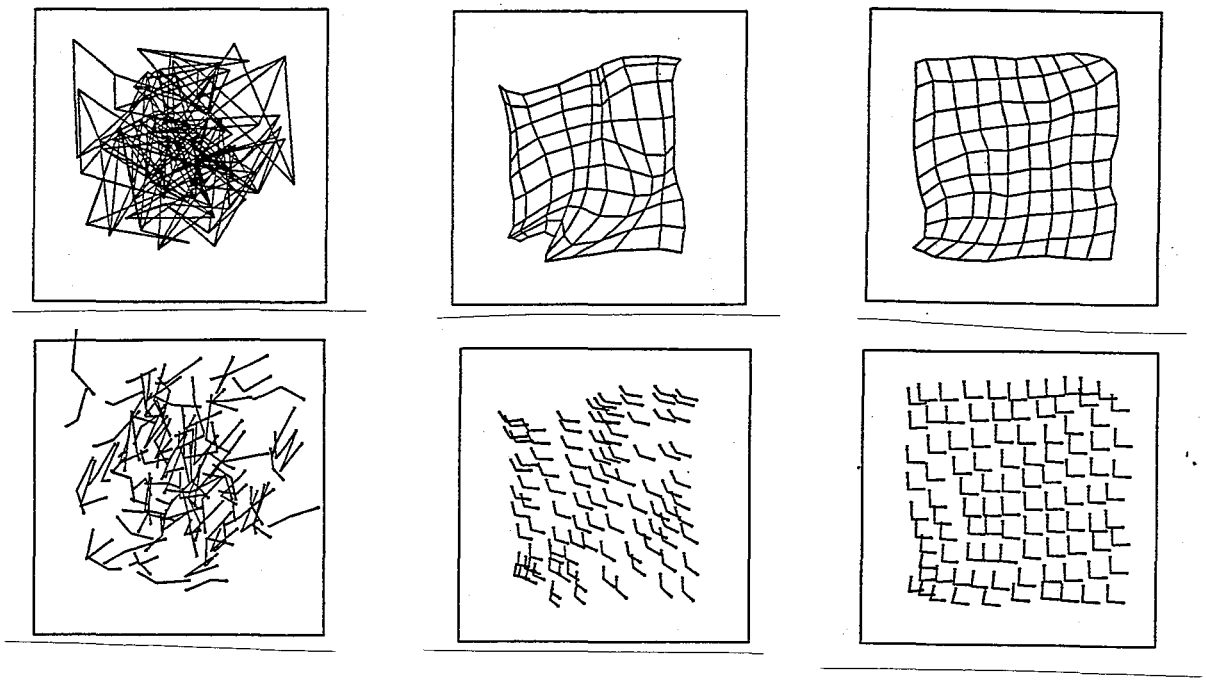


Abb. 8.2f Konvergenz der Positionsabbildung und Bewegungsmatrix

Wie man sieht, erfolgt eine gute Konvergenz.
Würde man nicht nur die Abbildung der Gelenkkoordinaten mit **M**, sondern auch die Abbildung der Geschwindigkeit mit **A** mit Hilfe eines neuronalen Netzes modellieren, so könnte der gesamte real-time Controller des Armes aus einem großen, Neuronalen Netz bestehen und ohne die explizite Kalkulation von Winkelfunktionen durchgeführt werden.
Selbst der allmähliche Verschleiß oder der Austausch eines Armes ist durch ständiges Nachlernen der Verbindungsgewichte des Neuronalen Netzes leicht auszugleichen, ohne daß man durch ständiges Nachkorrigieren (Änderungen der Konstanten!) oder Neuschreiben der Programme (andere Armform!) in die Roboterkontrolle eingreifen müßte.