Hubert Reinisch

Planungs- und Steuerungswerkzeuge zur impliziten Geräteprogrammierung in Roboterzellen

Hubert Reinisch

Planungs- und Steuerungswerkzeuge zur impliziten Geräteprogrammierung in Roboterzellen

Herausgegeben von
Professor Dr.-Ing. Klaus Feldmann,
Lehrstuhl für
Fertigungsautomatisierung und Produktionssystematik
FAPS

Carl Hanser Verlag München Wien

Als Dissertation genehmigt von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der Einreichung:	25. Mai 1992
Tag der Promotion:	29. Juli 1992
Dekan:	Prof. Dr. rer. nat. H. Mughrabi
Berichterstatter:	Prof. Dr.-Ing. K. Feldmann
	Prof. Dr.-Ing. F. Pfeiffer

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Reinisch, Hubert:
Planungs- und Steuerungswerkzeuge zur impliziten Geräteprogrammierung in Roboterzellen / Hubert Reinisch. - München; Wien: Hanser, 1992
(Fertigungstechnik - Erlangen; 31)
Zugl.: Erlangen, Nürnberg, Univ., Diss., 1992
ISBN 3-446-17380-3
NE: GT

Herstellung: Copy Center 2000, Erlangen-Eltersdorf
Printed in Germany

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Friedrich-Alexander-Universität Erlangen-Nürnberg.

Herrn Prof. Dr.-Ing. K. Feldmann, dem Leiter des Lehrstuhls für Fertigungsautomatisierung am Institut für Fertigungstechnik, danke ich herzlich für den Freiraum zur Durchführung dieser Arbeit, sowie für die großzügige und vertrauensvolle Förderung.

Herrn Prof. Dr.-Ing. F. Pfeiffer, dem Leiter des Lehrstuhls B für Mechanik an der Technischen Universität München, danke ich ebenso herzlich für sein Interesse an meiner Arbeit und sein Engagement bei der Übernahme des Korreferates.

Herrn Prof. Dr.-Ing. D. Seitzer, dem Leiter des Lehrstuhls für Technische Elektronik, sowie Herrn Prof. Dr.-Ing. H. Meerkamm, dem Leiter der Abteilung für Maschinenelemente und fertigungsgerechtes Konstruieren, danke ich für die unkomplizierte Unterstützung.

Ohne die Mithilfe meiner Kollegen, Studenten und wissenschaftlichen Hilfskräfte wäre die Arbeit in der bestehenden Form nicht möglich gewesen. Hierfür möchte ich an dieser Stelle Dank sagen. Stellvertretend gilt dies für Herrn Jürgen Hirath, Herrn Dipl.-Inf. Jörg Gewalt, Herrn Dipl.-Inf. Jens Rohde, Herrn Jürgen Schabhüser, Herrn Dipl.-Inf. Harald Nagy, Herrn Jürgen Püschner, Herrn Volker Landgraf, Herrn Wilfried Thoben, sowie Herrn Uwe Goldhammer. Herrn Joachim Leiblein möchte ich für seinen Einsatz und seine kontinuierliche Mitarbeit bei den Versuchen in der Werkstatt danken. Meinem Kollegen Herrn Dr.-Ing. Georg Geyer gilt mein Dank für manche Anregung und die oft unkonventionellen Hintergrundgespräche, die sich nicht nur auf komplexe Fragen der Informatik und Rechnertechnik beschränkten.

Einen Dank möchte ich auch an meine Familie richten. Die entgegengebrachte Geduld, und der oft daraus entstandene Verzicht, haben mir geholfen und sorgten für das nötige Umfeld zum Gelingen der Arbeit.

Hubert Reinisch

Planungs- und Steuerungswerkzeuge zur impliziten Geräteprogrammierung in Roboterzellen

Inhaltsverzeichnis

1. Einführung

In allen industriellen Bereichen ist eine fortschrittliche Automatisierung der Fertigungsprozesse unverzichtbar geworden. Die Dynamik der Rechnertechnologie und die Entwicklung hochintegrierter elektronischer Schaltungen, erschließen zunehmend Felder /34/, die für eine Automatisierung bislang als nicht realisierbar galten.

Selbständig agierende Roboterarbeitszellen sind ein derartiges Anwendungsfeld (s. /83/). Eine hochflexible Fertigungszelle soll zukünftig in der Lage sein, direkt produktnahe Fertigungsbefehle entgegenzunehmen. Die **implizite** Generierung von NC-Code ersetzt die **explizite** Programmierung von Automatisierungskomponenten wie Robotersteuerung (**RC**), **SPS** oder **Sensorik**.

Hierzu müssen Verfahren entwickelt werden, um den Anlagenzustand rechnerintern und dynamisch mitzuführen und daraus die erforderlichen, unterlagerten Operationen (Aktionen) abzuleiten. Als Voraussetzung hiervon bedarf es der Modellierung und Speicherung von fertigungstechnologischem Wissen. Es gilt Methoden zu konzipieren, um automatisiert Steuercode für die beteiligten, freiprogrammierbaren Betriebsmittel zu erzeugen und diesen dann selbständig zur Ausführung zu bringen.

Es ist Ziel der vorliegenden Arbeit, über diese Ansätze zur Aktionsplanung und -steuerung hinaus, einen Beitrag zum Erreichen eines Fernzieles der Automatisierungstechnik zu leisten. Dieses Fernziel ist die Behandlung von Lernprozessen in automatisierten Anlagen. Es müssen Lösungen gesucht werden, die gegenüber konventionellen Verfahren einen anderen, nichtalgorithmischen Weg beschreiten.

Für das maschinelle Lernen, als die denkbar mächtigste Entwicklungsstufe impliziter Geräteprogrammierung, müssen zunächst Lernmethoden bereitgestellt werden.

Künstliche neuronale Netzwerke werden hierzu aufgrund ihrer Lernfähigkeit gewählt. Für die Realisierung von Lernprozessen müssen Konzepte erarbeitet werden die praktikabel sind für die industrielle Fertigung. Schwierigkeiten bereiten hier vor allem problemnahes Training und Verschalten neuronaler Netzwerke, sowie die Umsetzung ihrer, für die Fertigungsautomatisierung wichtigen Robustheit gegenüber Fehlern.

Für die praxisnahe Nutzung der entwickelten Verfahren müssen in einem weiteren Schritt CAE/CAM-Werkzeuge konzipiert werden. In dieser Arbeit wird hierzu ein Ansatz vorgestellt, der auf verteilten Werkzeugen in einer soft- und hardwaremäßigen CAD/CAM-Kette basiert.

Es gilt die bislang noch unzureichende, industrielle Umsetzung der potentiell hohen Flexibilität und Leistungsfähigkeit von Betriebsmitteln wie z.B. 6-Achs-Industrieroboter oder Bildverarbeitungssensor zu beschleunigen. Lösungsansätze für derartig leistungsfähige Werkzeuge besitzen stark interdisziplinären Charakter. Zukünftige Fertigungssysteme, wie meist auch die damit gefertigten Produkte, erfordern deshalb die Integration von Mechanik, Elektronik und Informationstechnik zu mechatronischen Systemen (Abb. 1.1).

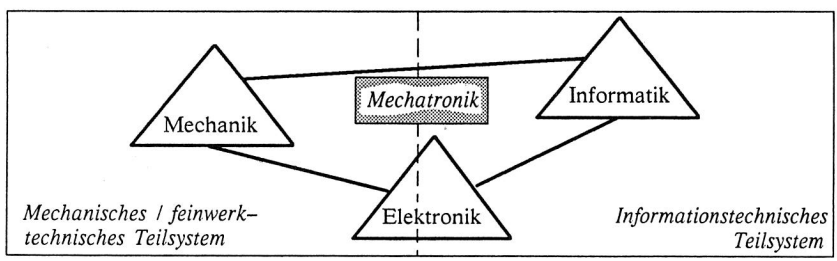


Abb. 1.1: Interdisziplinäre Mechatronik

Für das werkstattseitige Ende der CAD/CAM-Verfahrenskette muß ein prozeßnahes Werkzeug entwickelt werden, das Offline-NC-Geräteaktionsplanung und Online-Sensorik kombiniert. Es übernimmt Routineaufgaben wie z.B. geometrische Berechnungen und Logik. In einem Teilschritt gilt es die Architektur eines werkstattorientierten Systems im Hinblick auf moderne Multiprozessor-Gerätesteuerungen zu erarbeiten. Es soll der Fertigungsanlage eigene, lokale Handlungsfähigkeit verleihen, wie auch arbeitsplanbasierte Fertigungsaufträge verarbeiten. Für kleinere Betriebseinheiten sollte es optional auch autark, d.h. ohne Anschluß an die Planungsebene, dann jedoch mit eingeschränkter Funktionalität, in der Werkstatt eingesetzt werden können. Basis

ist das rechnerinterne Anlagenmodell - das Online-"Gedächtnis' der Gerätesteuerungen. Es dient nicht nur dazu, implizite Aktionen des Systems zu verfolgen und Gerätesteuerungen, Sensoren oder Aktorenkinematik zu simulieren /125/. Ebenso wichtig ist es, die Zellensituation kontinuierlich während des Betriebes der Fertigungsanlage mitzuführen und somit die Grundlage für selbständige Reaktionen zu schaffen. Die Auswirkungen impliziter Fertigungsinstruktionen müssen sich beschränken lassen auf das Simulationsmodell und/oder online die reale Fertigungszelle steuern.

Bislang scheitern offline erstellte NC-Programme häufig aufgrund stets vorhandener Abweichungen zwischen rechnerinternem Simulationsmodell und realer Zelle. Im Rahmen der vorliegenden Arbeit sind hierfür Lösungen zu entwickeln, da implizite Verfahren eine hinreichend präzise Übereinstimmung voraussetzen. Hierzu werden Ansätze zur impliziten Sensorintegration erarbeitet, mit deren Hilfe es möglich wird, das Anlagenmodell mit der Realität automatisiert abzugleichen. Permanent aktive Sensorik wird so reduziert oder vermieden.

Besondere Bedeutung besitzt der Aufbau einer möglichst scharf von den eigentlichen Werkzeugen abzugrenzenden, zentralen Datenbasis für Automatisierungskomponenten, AV-Arbeitspläne und Fertigungstechnologie. Hierfür muß ein konzeptionelles Datenbankschema, sowie Werkzeuge zur datenbank- und CAD-gestützten Betriebsmittelkonstruktion, wie auch zur Füllung mit anwendungsneutraler und langfristig gültiger Fertigungstechnologie entworfen und realisiert werden. Als Grundlage wird eine relationale Datenbank gewählt, auf der alle Werkzeuge durch verteilten Zugriff arbeiten.

Für das fertigungstechnologische Wissen entsteht daraus zunächst die Notwendigkeit der inhaltlichen, rechnergeeigneten Definition, beginnend bei einfachen Aktionsprimitiven ('Sensor_auswerten', 'schnelle Grobbewegung', 'Abrücken', ..) bis hin zu hochwertigen Montageanweisungen. Diese variieren stark und sind deshalb schwer zu automatisieren; andererseits ist die Behandlung besonders lohnend, da sie in fast jedem Fertigungsprozess (z.B. Lichtbogenschweißen oder Fertigen elektronischer Komponenten) auftreten.

Ergänzend wird ein Versuchsstand als Teil einer CIM-Modellfabrik aufgebaut; einschließlich prozeßnaher (DNC) und prozeßferner (LAN) Kommunikation. Die entwickelten Werkzeuge werden hieran getestet und deren Leistungsfähigkeit verifiziert.

2. Architektur impliziter Aktionssteuerung und -planung für Fertigungszellen

Die in der vorliegenden Arbeit entwickelten, impliziten Verfahren wurden für Fertigungszellen konzipiert, die aus freiprogrammierbaren Geräten (z.B. Bildverarbeitungssensor, SPS), CNC-Werkzeugmaschinen oder nicht freiprogrammierbaren Betriebsmitteln (z.B. einfache, schaltende Sensoren in Vorrichtungen) bestehen.

Aufgrund der besonders hohen Flexibilität, räumlichen Beweglichkeit und komplexen Steuerungstechnik des Industrieroboters (RC), ist er als zentraler Bestandteil einer flexibel automatisierten Roboterarbeitszelle ein besonders anspruchsvoller Prüfstein für die erarbeiteten Konzepte. Hinzu kommt, daß es bislang noch nicht gelang, dieses hohe Potential an flexiblen und leistungsfähigen Betriebsmitteln, neben den Großserien, im großen Maßstab in der Werkstatt einzusetzen; und dies obwohl in Industrie und Forschungsinstituten die Bedeutung der Robotik schon seit den 70-er Jahren erkannt wurde und entsprechende Entwicklungen hierzu noch andauern.

2.1 Definition impliziter CNC-Geräteprogrammierung

Das Ziel einer flexiblen Fertigungszelle besteht darin, Anpassungen an neue Aufgaben anstatt mit aufwendiger Umrüstung durch Umprogrammierung zu lösen. Damit müssen aber auch alle freiprogrammierbaren Einzelgeräte (der IR selbst, SPS, NC-WZM, Sensorik etc.) in der Fertigungszelle, wie auch deren Koordination durch Signal- und Datenleitungen programmiert werden. Sie ermöglichen die Automatisierung

komplexer und wechselnder Fertigungsprozesse – sie bedingen jedoch auch die Erstellung und den Test komplexer Programme. Die Steuerungstechnik obiger Automatisierungskomponenten besitzt eine Reihe Gemeinsamkeiten mit der von Industrierobotern, so daß die Entwicklungsstufen für implizite Methoden im weiteren teilweise parallel skizziert werden können.
Folgende Entwicklungsstufen wurden im Rahmen der vorliegenden Arbeit, in Erweiterung von /110/, definiert (s. Abb. 2.1):

◆ **Entwicklungsstufe 1: NC-orientiert**
Am Beispiel von Bewegungssätzen für Industrieroboter mußten zunächst die Achswinkelstellungen aller beteiligten CNC-Achsen vorgegeben werden. Diese Programmierart kennt keine Trennung nach geometrischen, technologischen bzw. ausführenden Sprachelementen. Auf Seiten der CNC-Werkzeugmaschinen entspricht dies den Achswerten eines NC-Programmes nach DIN 66025, das ein NC-Postprozessor nach Interpolation und Rücktransformation in das Maschinenkoordinatensystem erzeugt (steuerungs- und maschinenspezifisch). Im nächsten Schritt folgte die an NC-Werkzeugmaschinenprogrammen der spanenden Fertigung orientierte Programmierung. Entwicklungen dieser Hierarchiestufe haben meist Ähnlichkeit mit der NC-Sprache APT. Sogenannte "APT-ähnliche" (auch **ALL** = **A**PT-**L**ike **l**anguages) Sprachen sind RAPT /88/ oder EXAPT bei NC-Werkzeugmaschinen. Sie lassen teilweise getrennt Technologie- (z.B. Verfahrgeschwindigkeiten) und Geometriedefinitionen (2D/3D-Geometrieelemente wie Punkte oder Werkstückkonturlinien) zu, auf die im Ausführungsteil des NC-Programmes Bezug genommen wird.

◆ **Entwicklungsstufe 2: Explizit Assembler-ähnlich**
Im folgenden wurden Steuerungssprachen eingesetzt, die es direkt erlauben, den TCP des Roboters mit einem Zielframe, unter Anwendung unterschiedlicher Interpolationsarten für die Bewegungsbahn, zur Deckung zu bringen. Bewegungsanweisungen definieren die Position des TCP und die jeweils zugehörige Orientierung des TCP-frames unabhängig von den tatsächlichen Achswerten des Roboters. Im Bereich der CNC-Werkzeugmaschinen ist dies vergleichbar mit dem CLDATA-Format (DIN 66215) zum Beispiel beim 5-Achs-Fräsen. Der NC-Code spezifiziert Position und Orientierung von TCP (IR) bzw. von Fräserspitze und Fräserdrehachse (5-Achs-Fräsen) ohne direkten Zusammenhang zur Werkstückgeometrie.

Exakte, absolute Positionen und Orientierungen lassen sich in einem zusätzlichen Arbeitsgang (Teach-In) im Online-Betrieb nachträglich ergänzen (industrieller Standard). Der Industrieroboter wird manuell verfahren und die ausgezeichneten Posi-

tions- und Orientierungswerte in die Steuerung bzw. in das aktive Programm übernommen.

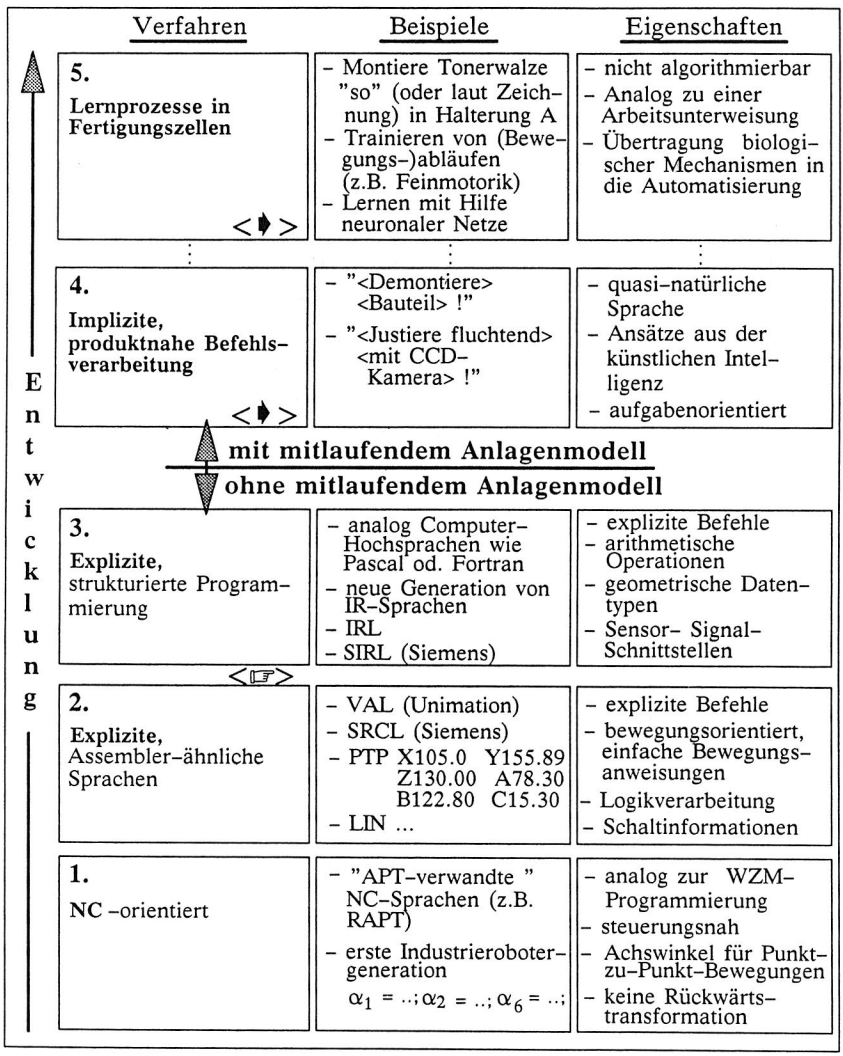


Abb. 2.1: Entwicklungsstufen der RC(NC)-Programmierung (⧫: Ziele vorliegender Arbeit)

Ab diesem Entwicklungsschritt wurde damit begonnen, an Offline-Simulatoren (z.B. /6/ oder CARo nach /60/) zu arbeiten. Graphisch und ohne Einsatz der realen Betriebsmittel, also ohne Anlagenstillstand und Unfallgefahr, lassen sich explizite CNC-Programme mit der Funktionalität von Standard-Texteditoren manuell editieren. Für komplexe Anwendungen mit Logik- bzw. Technologiebausteinen, kann mit Hilfe der textuellen Offline-Programmierung das Programm symbolisch formuliert werden. Das so erstellte IR-Programm wird durch einen Steuerungsspracheninterpreter interpretiert und am Bildschirm kontrolliert. Der Benutzer arbeitet mit dem Simulationssystem analog wie an der realen Steuerung, einschließlich des "simulierten Teach-In" aus dem rechnerinternen Geometriemodell.

Die Sprache, in der das IR-Programm am Simulator erstellt wird, kann eine eigens geschaffene Sprache sein, die dann mit Hilfe von Postprozessoren in die Zielsprache konvertiert wird. Ansätze zur Verwendung des genormten Robotersprachen-Formates IRDATA /56/ konnten sich bislang nicht ausreichend durchsetzen.

◆ **Entwicklungsstufe 3: Explizit strukturiert**

Neue Generationen von höheren NC-Sprachen (z.B. IRL /57/) für Roboterarbeitszellen erlauben strukturierte Programmierung analog Computer-Hochsprachen. Deklarationen, Geometrische Datentypen (z.B. Frames) mit zugehörigen Operationen, Konzepte für zeitsynchrone und asynchrone Ausführung von Befehlen, sowie allgemeine Signal-/Sensorschnittstellen stehen zur Verfügung (s. /14/). Auch hier ist jedoch kein rechnerinternes Modell zur Beschreibung von Anlage, Fertigungstechnologie, Produkt und Aufgabe vorhanden. Es ist demzufolge nicht möglich, mit "höherwertigen", an die primäre Fertigungsaufgabe angenäherten Anweisungen zu arbeiten, die sich auf eine dynamisch verändernde Umwelt beziehen.

Explizite NC-Programmierung von Roboterarbeitszellen heißt detaillierte Vorgabe von Steuersignalen, Sensoranweisungen und Programmlogik der frei programmierbaren Geräte der Roboterzelle. Sind Bewegungsbefehle beteiligt, so müssen Position und Orientierung des Werkzeugbezugssystems, sowie sämtliche Freiheitsgrade des Roboters ebenfalls detailliert angegeben werden, abhängig von Abmessung, Auskragung und den Raumlagen des Werkzeugs (z.B. Greifer) und der Werkstücke, sowie von der Greifsituation. Dies gilt sowohl für ganze Bewegungsbahnen (z.B. Spritzlackieren), wie auch für ausgezeichnete Positionen (z.B. in der Montage). Hierzu bedarf es im räumlichen Fall umfangreicher Positionsarithmetik, die sich auch auf physikalisch nicht vorhandene Symmetrieachsen, Verlängerungen von Körperkanten etc. beziehen kann (auch durch Teach-In nicht erreichbar; z.B. stark 3-dimensional

ausgerichtete Montage). Mit der Zunahme der räumlichen Beweglichkeit von Handhabungsgeräten steigt auch die Anzahl der festzulegenden Freiheitsgrade. Im Beispiel eines 6-Achs-Gelenkroboters sind dies bezogen auf den TCP:

- 3 translatorische Freiheitsgrade
- 3 rotatorische Freiheitsgrade (z.B. Eulerwinkel /81/)
- mehrere Winkelstati für eindeutige Roboterstellung (z.B. Grund- /"Über-Kopf")

Hinzu kommen, je nach Fertigungszelle, zusätzlich zu koordinierende NC-Achsen, wie beispielsweise in kraft- und weggeregelten Parallelgreifern oder in Zusatzkinematiken. Darüberhinaus besteht ein wesentlicher Teil der Steuerung einer Fertigungszelle aus Logik, einschl. Ablaufsteuerung, SPS- und Sensorprogrammierung. Fragen der Koordinierung von mehreren Bewegungssystemen, z.B. bei Parallel-Betrieb von Robotern, verstärken diese Problematik zusätzlich. Weitere Anforderungen, wie Optimierung hinsichtlich Zeitbedarf, Energie oder Abweichungen zwischen Soll- und Istbahnen, bzw. -position, können nicht, oder nur unter großem Aufwand eingebracht werden.

Explizite Methoden sind fehleranfällig und zeitaufwendig und überfordern wegen der Komplexität räumlicher und steuerungstechnischer Parameter das räumliche Vorstellungsvermögen von Programmierer bzw. Werkstattpersonal vor Ort. In der Praxis kommt eine erhebliche Unfallgefahr für Mensch und Fertigungsanlage sowie Zeitdruck hinzu. Aufwendige und steuerungsnahe Programme auf dem Niveau einer Assembler-ähnlichen Sprache sind zudem unzureichend wart- und lesbar.

Die explizite Programmierung der Anlagen führt dadurch zur Abkehr von der eigentlichen Fertigungsaufgabe. Die Tätigkeiten verschieben sich in Richtung Rechner-/Steuerungstechnik und entfernen sich von dem primären Ziel, der Fertigung des Werkstückes.

In dieser Problemstellung liegt einer der Hauptgründe für die derzeit noch unzureichende Umsetzung von Ergebnissen der Robotik in die industrielle Praxis. Die potentiell hohe Flexibilität einer Roboterzelle mit den darin enthaltenen, frei programmierbaren Betriebsmitteln wird nur unzureichend in der Werkstatt umgesetzt.

◆ **Entwicklungsstufe 4: Implizite Geräteprogrammierung** (s. Abb. 2.1)
Implizite NC-Programmierung überläßt die Umsetzung problemnaher Anweisungen in, für die beteiligten Einzelsteuerungen ausführbaren NC-Code, einem Aktionsplanungswerkzeug (s. /31/, /35/, /44/, /45/, /39/). Die Fertigungsaufgabe wird der Fertigungsanlage auf hohem Abstraktionsniveau zur Bearbeitung übergeben. Mit Hilfe

einer Aktionsplanung, einem permanent mitgeführten, rechnerinternen Umweltmodell, sowie einer Wissensbasis kann die FFZ bedingt autonom agieren. Der Bediener benötigt keine detaillierte Kenntnis über Steuerungstechnik oder Geometriedaten der Anlage. Die Programmiertätigkeit wird durch die automatisierte Auflösung und Ausführung impliziter Befehle ersetzt. Das NC-Programmiersystem wird zum Aktionsplanungs- und Steuerungssystem.

Derartige, implizite Fertigungsbefehle lauten beispielsweise im Kontext einer konkreten Fertigungsanlage:

1. – **Greife** Steckerleiste
2. – **Bündig ausrichten** zu Führungsschiene
3. – Ausführen einer **schnellen Grobbewegung** zu Ventil A
4. – **Füge** Steckerleiste **in** Spannvorrichtung (Wiederhole m mal)
5. – **Justiere** Relais 5 in Prüfvorrichtung
6. – **Demontiere** Bauteil
7. – **Zurücknehmen** (undo) der letzten n Anweisungen (nur in der Simulation)
8. – **Montiere** Baugruppe XY (z.B. Schalter) komplett **gemäß** Arbeitsvorbereitung und CAD-Implosionsmodell
9. – Blechteil 1 **gegen Anschlag** Z **führen**
   – ...

◆ **Entwicklungsstufe 5: Maschinelle Lernprozesse**

Stufe 5 nach Abb. 2.1 umfaßt Lernprozesse in Fertigungsanlagen. Vorbild für diese höchste Entwicklungsstufe ist die Lernfähigkeit biologischer Systeme (z.B. Unterweisen einer menschlichen Arbeitskraft durch Vormachen, Nachahmen, Lernen von feinmotorischen Bewegungen). Das Training ersetzt das Umprogrammieren von Steuerungsalgorithmen. Zum Beispiel kann beim Einsatz neuronaler Netze die Anpassung an eine geänderte Problemstellung durch neue Festlegung der Netzwerktopologie und Durchführung eines erneuten Trainingslaufes erfolgen.

Der Stand der Technik ist durch explizite Programmerstellung (Ende Stufe 2), sowohl im Bereich der Online- wie auch der Offline-Verfahren, gekennzeichnet (☞ in Abb. 2.1).

Das Ziel der vorliegenden Arbeit ist das Erreichen der Entwicklungsstufen 4 bzw. 5 nach Abb. 2.1. Hierzu werden Ansatzmöglichkeiten aufgezeigt und Werkzeuge entwickelt, die zusammen mit der Fertigungszelle Befehle bearbeiten können, die

nahe an der eigentlichen Fertigungsaufgabe liegen. Dies schließt automatisch ablaufende, geometrische und logische Berechnungen sowie Symbolverarbeitung unter der Nutzung von Vorwissen mit ein.

2.2 Verteilte Werkzeuge für die implizite NC-Programmierung [1] von Roboterarbeitszellen

Implizite NC-Geräteprogrammierung erfordert die Entwicklung prozeßnaher und prozeßferner, verteilter Werkzeuge, die in die für FFZ etablierte Steuerungsstruktur, bzw. CIM-Hierarchie (Konstruktionsebene - LAN - Zellenrechner (ZR) - Zelleninterner Bus (Feldbus) - Gerätesteuerungen) integriert werden müssen.

Neben einem prozeßnahen Werkzeug zur Online-Aktionssteuerung und -planung müssen CAE-Werkzeuge für Lernprozesse in Fertigungszellen, sowie zur datenbank- und CAD-basierten Wissensakquisition entwickelt werden. Auf Basis dieser Werkzeuge gilt es eine CAD/CAM-Verfahrenskette für die implizite NC-Geräteprogrammierung zu realisieren.

2.2.1 Entwicklung der CAD/CAM-Verfahrenskette

Derzeit eingesetzte Verfahren zur Roboterzellenprogrammierung nutzen verfügbares Wissen über Roboterzelle, Betriebsmittel, zu fertigendes Werkstück und beispielsweise zur Fügetechnik nicht aus. Sie ermöglichen daher nicht die präzise Ausführung der gestellten Fertigungsaufgabe durch automatische, der aktuellen Roboterzelle angepaßten Generierung von Steuercode. Am Beispiel der Montagetechnik werden folgende Tätigkeiten meist noch getrennt behandelt:

A) Konstruktion: Die Auswahl und, falls nötig, die Neukonstruktion einer dem Montageproblem entsprechenden Hardware/Anlage, die die nötigen Bewegungs-, Positionier- und Meßvorgänge ausführen kann (incl. Steuerungstechnik). Die Einzelkomponenten (Betriebsmittel wie Sensorik, Handhabungsgeräte, Steuerungen, ..) sind in einer kaum mehr überschaubaren Vielfalt verfügbar. Bei der Konfiguration zwingt dies zum Einsatz von Datenbanken, da hier zudem bereits das Basiswissen zur impliziten Geräteprogrammierung entsteht.

1) Im Rahmen dieser Arbeit: Überbegriff von zu generierendem Steuercode (RC, SPS, Sensorik) für die freiprogrammierbaren Betriebsmittel einer IR-Zelle

B) Programmierung: Die Programmierung der Rechnersysteme zur Ansteuerung der Hardware (Mechanik und Steuerungen). Zukünftig soll dies implizit nach einem vorgegebenen Zielzustand in der Fertigungszelle erfolgen, abhängig von der mechanischen und logischen "Verschaltung" der Betriebsmittel einer Zelle aus A).

Körpergeometrie oder Hilfskoordinatensysteme sind besonders wichtig für die Programmierung von Bewegungsabläufen, wogegen die Verschaltung der Anlage wesentlichen Einfluß auf die Programmierung logischer Funktionen (z.B. durch eine SPS realisiert) besitzt. Der Wissenstransfer erfolgte bislang über den Engpass der Konstruktionszeichnung oder des CAD-Modells (meist ohne Verschaltungen). Die für Konstruktion/AV und NC-Programmierung erforderlichen Datenmengen besitzen nur einen geringen Überlappungsgrad (Abb. 2.2, oben).

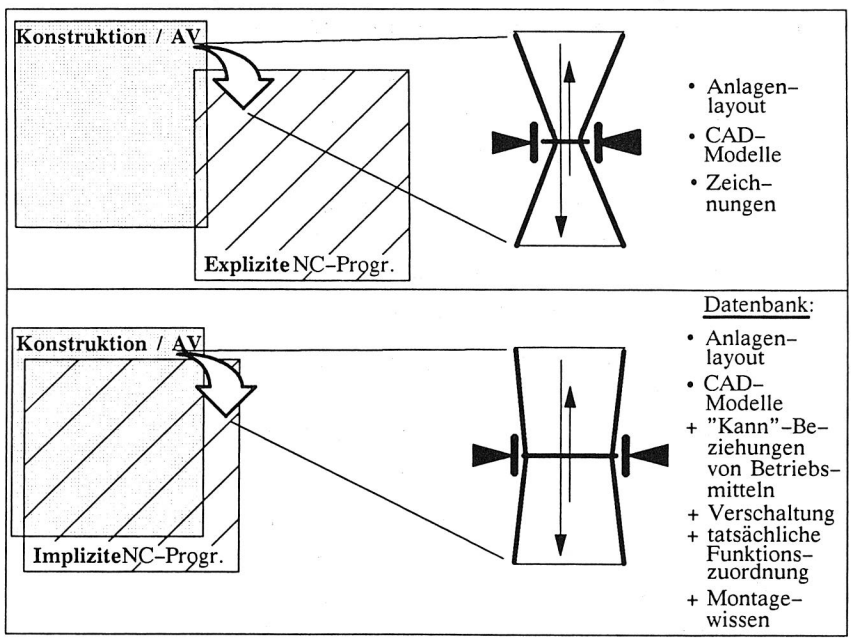


Abb. 2.2: Engpaß bei der Programmierung von Fertigungszellen

Die Engstelle wird aufgeweitet, indem die für die implizite Aktionsplanung wichtigen Basisdaten, parallel zu Geräteauswahl und Zellenkonstruktion, mitprotokolliert und in der Datenbank gespeichert werden (Abb. 2.2, unten). Implizite NC-Programmierung beginnt damit bei Betriebsmittelauswahl und -konstruktion. Durch Zugriff auf die zentrale Datenbank kann das Aktionsplanungswerkzeug die in der Konstruktion getroffenen Zuordnungen zwischen auszuführenden Funktionen und Betriebsmittel innerhalb der Roboterzelle direkt auswerten ("Gedanke der CIM-Fähigkeit von Daten"). Diese Zuordnungen fallen bei einer funktionsorientierten, datenbankgestützten Betriebsmittelauswahl und Wiederholteilsuche neben dem CAD-Modell ohnehin an (s. Kap. 5.3.1).

Bislang wurde diese Zuordnung nur konventionell, in den Köpfen der Konstrukteure getroffen und stand für die Roboterzellenprogrammierung nicht mehr zur Verfügung. Der Programmierer hatte die Aufgabe das in der Konstruktion ja prinzipiell vorhandene Wissen wieder zu rekonstruieren, mit dem eigenen Wissen zur Fertigungstechnologie zu kombinieren und in CNC-Programme umzusetzen. Für eine eigenständig agierende Fertigungszelle muß diese Wissensbasis jedoch rechnerintern vorliegen.

Relationale Datenbanken eignen sich dazu, die gesamten Basisdaten, einschließlich der CAD-Modelle, aufzunehmen (Abb. 2.3). Hierauf arbeiten die folgenden, im Rahmen dieser Arbeit entwickelten Werkzeuge (WZ):

1. **-WZ zur datenbankgestützten Betriebsmittelkonstruktion**
2. **-Wissensbasiertes Entwicklungs-WZ für Neuronale Netz-Applikationen**
3. **-WZ zur Montageplan-Erstellung**
4. **-Prozeßnahes Aktionsplanungs und -steuerungs-WZ in der Werkstatt**

Ein weiteres, zentrales Element ist der CAD-Volumenmodellier-Kernel (s. Kap. 3.4), vor allem für die Werkzeuge 1 bis 3 auf CAD-Ebene ('CAD' in Abb. 2.3). Das implizite Aktionsplanungswerkzeug in der Werkstatt ('W') muß ebenfalls einen Datenbankanschluß besitzen. Es erzeugt NC-Daten für die reale Fertigungszelle.

Durch implizite Sensorintegration kann mit Hilfe des Aktionsplanungswerkzeuges das rechnerinterne Anlagenmodell den realen Geometrieverhältnissen angepaßt werden. Darüberhinaus ist für die langfristige Weiterentwicklung dieser Werkzeuge von besonderer Bedeutung, daß der Aktionsplaner Verhaltensweisen lernen kann. Beispielsweise um das Bewegungsverhalten des Roboters zu ändern und direkt dessen Steuerung zu übernehmen bzw. entsprechende Programmbausteine in die Steuerung zu laden.

Verteilte Datenbankzugriffe, wie durch Zugriff des impliziten Programmiersystems in der Werkstatt auf die auf CAD-Ebene liegende Datenbank, und insbesondere moderne CNC-Steuerungstechnik mit umfangreichem Funktionsumfang der DNC-Schnittstellen, erzeugt besondere Anforderungen an die Kommunikation.

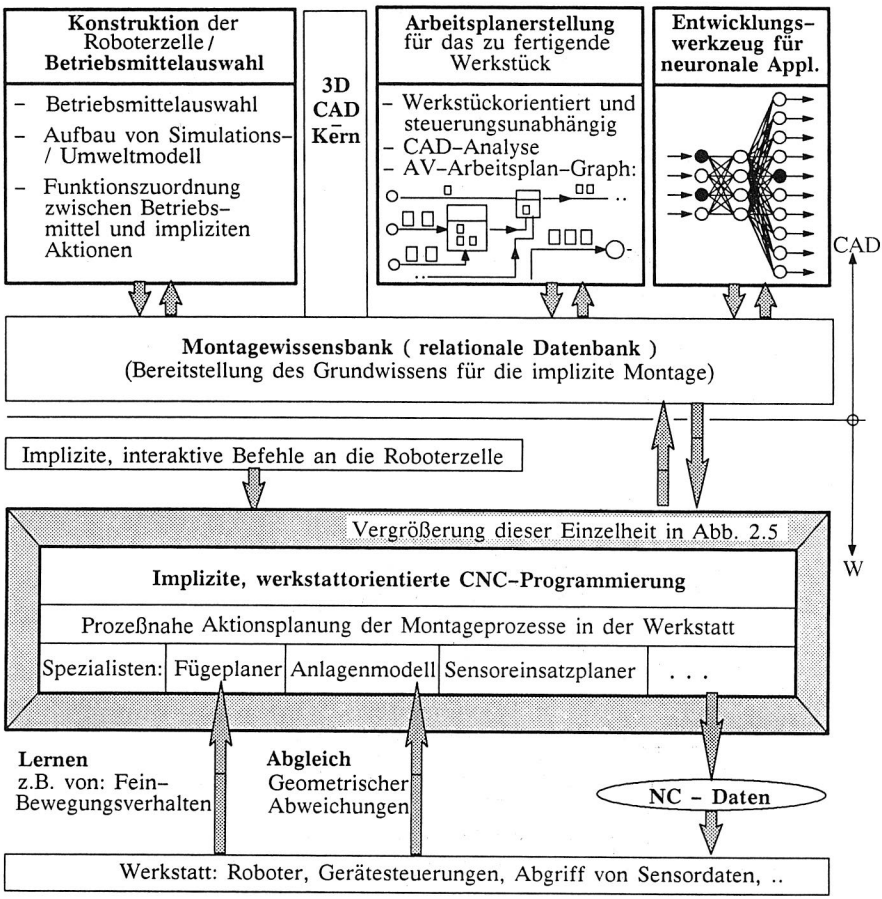


Abb. 2.3: CAE/CAM-Gesamtkette

Die Offline-Programmierung in der Arbeitsvorbereitung verhindert Ausfallzeiten für die Fertigung. Fertigungsprozeß und NC-Programmierung sind entkoppelt. Probleme

entstehen jedoch durch Abweichungen zwischen dem CAD-Anlagen- und Produktmodell (ideale Geometrie) und der realer Zelle. Weiterhin fehlt die Kopplung zum Prozess wie z.B. zur Robotermechanik oder zu Sensorsignalen (Abb. 2.2).

Durch die ausschließliche Programmierung von Roboterzellen in der Arbeitsvorbereitung lassen sich zwar in der Werkstatt noch Optimierungen vornehmen, jedoch keine weitergehenden Änderungen mehr durchführen. Die erstellten Roboter- und Steuerungsanweisungen sind ohne Kontext vom Techniker an der Anlage nicht nachvollziehbar. Im Weiteren darf nicht vorausgesetzt werden, daß der Facharbeiter in der Werkstatt Informatik- bzw. Steuerungskenntnisse besitzt oder über Detailkenntnisse der Fertigungsanlage (Verschaltung, Mechanik, NC-Sprachen) für Programmanpassungen verfügt. Ebenso wäre dies durch zu hohe Stillstandszeiten der Roboterzelle ausgeschlossen. In der Arbeitsvorbereitung läßt sich der aktuelle Betriebszustand nicht direkt in die Anlagensteuerung einarbeiten. Interaktives Arbeiten mit der Fertigungszelle ist nicht möglich.

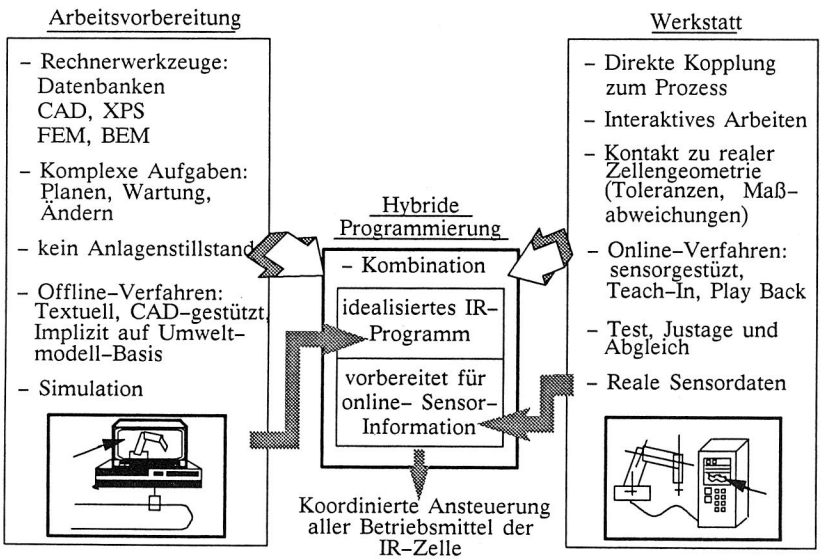


Abb. 2.4: Hybride Programmierung

Die bei einem produktnahen Befehl implizit beteiligten Sensoren wie Lichttaster und -schranken, Näherungssensoren oder CCD-Sensor müssen koordiniert angesteuert und Sensordaten aufgenommen werden. Dies gilt für die Sensorsimulation, wie für die

reale Anlage. Das prozeßnahe Aktionsplanungswerkzeug muß deshalb auf Werkstattebene sitzen und wahlweise online an die Zielsteuerung gekoppelt werden können.

Reine Online-Verfahren schließen implizite NC-Programmierung (nach Kap. 2.1) oder gar autonome Handlungsfähigkeit der Roboterarbeitszelle aus. Die Steuerungen auf Werkstattebene besitzt keine Abbildung der sich fortlaufend verändernden Umwelt. Komplexe Programme oder Optimierungen können aufgrund der dazu erforderlichen Stillstandszeiten der Anlagen nicht getestet werden. Zudem können die im Bereich der Konstruktion/AV vorhandenen, leistungsfähigen Rechnerwerkzeuge (CAD, DBMS, Simulatoren, Expertensystemtools, ..) nicht genutzt werden.

NC-Code für z.B. Draht-/Senkerodieren, Drehen, 5-Achs-Fräsen oder Laserstrahlschneiden läßt aufgrund der Relativbewegung Werkstück/Werkzeug auf die erzeugte Geometrie rückschließen. In der Montagetechnik dagegen sind in der Regel nur Endzustände definiert (Implosionsmodelle); Fügeablauf, Feinmotorik etc. sind aus dem Zusammenstellungsmodell nicht direkt zu entnehmen. Sie variieren stark je nach Anwendungsfall. Sie sind meist nur durch die handwerkliche Erfahrung des Montagearbeiters präsent und derzeit -trotz ihrer vermeintlichen Einfachheit- auch noch nicht systematisch für den Rechner aufbereitet (vgl. /75/).

Aufgrund dieser und anderer Mehrdeutigkeiten sind Techniken zur Generierung von Post-Prozessoren, in Anlehnung an CAD/CAM-Verfahrensketten der spanenden CNC-WZM, nicht übertragbar. Montage beinhaltet ein inhomogenes Spektrum unterschiedlicher Technologien und massiven Sensoreinsatz, so daß Anlagen bislang meist nur starr automatisiert werden konnten.

Ein weiterer wesentlicher Unterschied liegt in der großen Bandbreite unterschiedlichster Betriebsmittel (Bildverarbeitung, binäre Sensorik, Roboter, Zuführeinrichtungen,..) die als Komponenten in einer flexiblen Roboterzelle enthalten sind. Unterschiedliche Steuerungen müssen angesprochen werden. Hinzu kommen größere Toleranzen durch offene kinematische Ketten der Handhabungssysteme.

Für die Entwicklung des prozeßnahen Werkzeuges ist eine wesentliche Voraussetzung die Loslösung von Ansätzen, die anlagen- oder steuerungsspezifisch sind oder nur auf spezielle Problemstellungen (z.B. Palettieren von Werkstückträgern, Leiterplattenbestückung /71/,/90/) abzielen. Vorgefertigte und parametrisierte Makroprogrammierung (vgl. CAD-Makros) scheitert an der kombinatorischen Vielzahl möglicher Situationen in selbständig agierenden, automatisierten Fertigungszellen. Die aufgabenorientierte Programmierung der hochflexiblen Roboterzelle muß deshalb aus

den aktuellen Zuständen von Roboter, Umwelt, Werkstück und Sensorik heraus erfolgen. Steuercode kann erst dynamisch zur Laufzeit erzeugt werden.

Der Grundgedanke besteht darin, zwischen diesen beiden Extremen ein werkstattorientiertes, implizites NC-Programmiersystem (**WOP**) zu konzipieren, das in der Lage ist, die Vorteile beider Verfahren in sich zu vereinigen und beide Datenbereiche zu nutzen (Hybride Programmierung, Abb. 2.4). Mit der Platzierung des Aktionsplanungswerkzeuges auf Werkstattebene soll zudem die autarke, eingeschränkte Nutzung auch für kleinere Betriebseinheiten möglich werden, die nicht über eine AV oder CAD/CAM-Kette verfügen. Ist dies nicht der Fall kann eine weitergehende Funktionalität durch Anbindung an die weiteren, verteilten Werkzeuge erzielt werden.

Zusammenfassend läßt sich die Aktionsplanung in der vorliegenden Arbeit wie folgt einordnen (**X**):

	AV	Werkstatt, Prozeßnähe (WOP)
explizit	☐	☐
implizit	☐	☒

Die hybride Programmierung ermöglicht beispielsweise vorab geplante (Sensor-)Aktionen zur Laufzeit mit realen Sensordaten zu ergänzen und bedingt weitere Operationen auszulösen.

2.2.2 Multiprozessing für Systemtasks prozeßnaher Roboteraktionsplanung

Steuerungen von NC-Werkzeugmaschinen und insbesondere NC-Robotersteuerungen müssen Bewegungen in Echtzeit steuern. Im Rahmen dieser Arbeit zur Aktionsplanung impliziter Befehle wird die Anlagensimulation und die Aktionsplanung auf einem an die Gerätesteuerungen der Roboterzelle angekoppelten Rechner (PC-Standardhardware unter UNIX) durchgeführt. Die hier erzeugten Daten können direkt in die Zielsteuerung übertragen werden. Echtzeitaufgaben, wie die Lageregelung der Achswinkel, verbleiben in den Gerätesteuerungen.

Neue Steuerungsgenerationen unterstützen diesen Ansatz, da sie aufgrund steigender Leistungsfähigkeit zunehmend auf komfortablen Steuerrechnern (PC) basieren und in der Lage sind komplette Zellenrechnerfunktionen (z.B. parallele Steuerung zweier kooperierender Roboter, vgl. /113/) zu übernehmen.

Für das Ziel einer autonom handlungsfähigen Arbeitszelle muß auch das Verhalten der NC-Steuerung innerhalb der Aktionsplanung simuliert werden. Wirklich-

keitsgetreue Reaktionen der Fertigungszelle, wie auch ein stets aktuelles Umweltmodell sollen dadurch erreicht werden.

Die Steuerungsalgorithmen moderner (Industrieroboter)Steuerungen werden aus Gründen der Wartung, Änderung und Verifikation zunehmend in Hochsprache (meist C) programmiert (Größenordnung derzeit ca. 95%). Hohe Rechengeschwindigkeiten der Hardware erlauben dies, ohne auf maschinennahe und auf Laufzeit optimierte Assemblerprogrammierung zurückzugreifen.

Damit ergibt sich ein wesentlicher Aspekt für die Simulation des Steuerungsverhaltens: Ein Großteil der, bei der Steuerungsentwicklung entstandenen und z.B. in C verfügbaren Steuerungsalgorithmen lassen sich aus der Echtzeitumgebung herauslösen und direkt 1:1 in die Simulationsumgebung portieren. Damit kann eine größtmögliche Identität zwischen Simulation und NC-Steuerung, z.B. bezüglich der Bahntreue der Bewegungsführung, erzielt werden.

Die Hardware moderner Industrierobotersteuerungen wird zukünftig häufig auf dem Konzept einer modularen Multiprozessorarchitektur basieren (s. /113/). Die Simulation der NC-Steuerung sollte demzufolge ebenfalls in Mehrprozeßarchitektur realisiert werden. Dem steuerungsinternen Bus der Robotersteuerung entspricht die Interprozesskommunikation des UNIX-Systems, über die, je nach Aufgabe, getrennte Prozesse miteinander verbunden sind. Abb. 2.5 zeigt die erforderlichen Tasks im Vergleich zwischen realer Industrierobotersteuerung und der Steuerungssimulation am Beispiel der Bewegungsführung.

Reale NC-Steuerung:
Die NC-Steuerung decodiert das explizite NC-Programm und steuert die Achsantriebe mit Sollwerten an, bzw. verarbeitet Schaltinformationen. Die einzelnen Tasks der Bewegungssteuerung werden über einen steuerungsinternen, leistungsfähigen Bus, z.B. VME-Bus, verbunden /113/. Über DUAL-PORT-RAM können einzelne Tasks Daten austauschen.

NC-Steuerung in der Aktionsplanung:
Dem steuerungsinternen Bus der Robotersteuerung entspricht die Interprozesskommunikation des UNIX-Systems, über die, je nach Aufgabe, getrennte Prozesse miteinander verbunden sind.

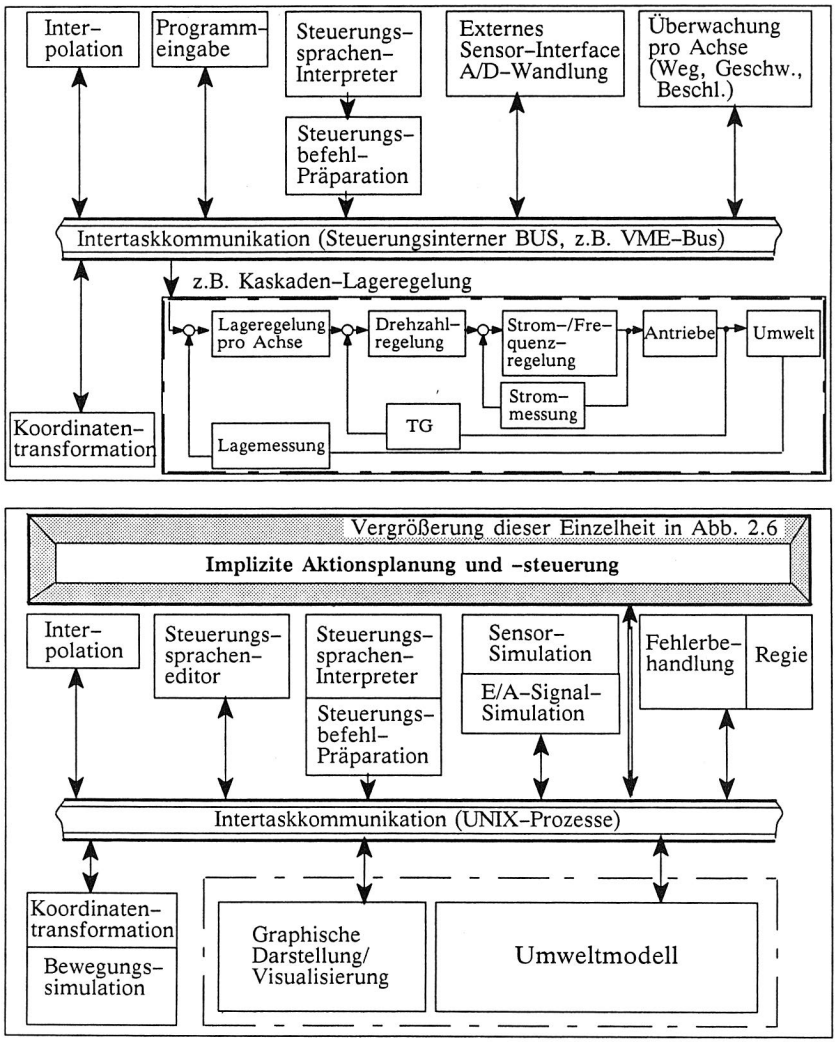


Abb. 2.5: Analoge Taskstruktur der realen IR-Steuerung (oben) und Prozeßbereiche des impliziten Aktionsplanungs-Werkzeuges (unten); vgl. /77/ u. Simulator GPS (Siemens AG)

Im Falle des Einsatzes objektorientierter Systeme sollte die Interprozesskommunikation auf den Message-Austausch der Objekte reduziert werden können (vgl. Kap. 2.3).

Das DUAL-PORT-RAM der Robotersteuerung kann innerhalb der Aktionsplanung durch "shared-memory" (UNIX System V) simuliert werden. Den detaillierten Aufbau des implizites Aktionsplanungswerkzeug mit den erforderlichen Komponenten zeigt Abb. 2.5,unten (=Ausschnittsvergrößerung aus Abb. 2.3).

Die **Bewegungssimulation** simuliert vorhandene Kinematiken. Die **Koordinatentransformation** berechnet die Rückwärts-/ Vorwärts-Kinematik einschließlich zu koordinierender NC-Zusatzachsen. Der Prozeßbereich **Umweltmodell** umfaßt das Anlagen- und Werkstückmodell (Geometrie, Kinematik, Schalt- und Signalzustände, Sensordaten, etc.; vgl. Kap. 3.1). Der **Interpolator** unterteilt einen Bewegungssatz in eine Folge von Frames entsprechend der gewählten Interpolationsart (PTP, Linear, Zirkular, Spline). Die **Sensorsimulation** beschreibt das Verhalten von Sensorik in der Roboterzelle, so daß Roboterprogramme, die Sensorfunktionen beinhalten (z.B. eine Anweisung an ein Bildverarbeitungssystem zur Lageerkennung eines Bauteils) abhängig von Sensordaten simuliert werden können (Verzweigungen). Der Prozess zur **graphischen Darstellung** ermöglicht die Visualisierung von Operationen oder Systemreaktionen auf Benutzereingaben, sowie Ansichtenwechsel etc. Der Prozess **Steuerungsspracheninterpreter** interpretiert explizite Steuerungscodes und übergibt diese an die Bewegungssimulation, sofern Bewegungssätze im Roboterprogramm enthalten sind. Der Prozess des **Steuerungssprachen**editors erlaubt das textuelle Erstellen und Verändern von expliziten Industrieroboterprogrammen. Der Prozess **Fehlerbehandlung** meldet bzw. fängt System- bzw. Bedienerfehler ab. Der Prozess **Regie** erledigt Verwaltungsaufgaben wie Starten und Terminieren von Prozessen, Instanziieren, An- und Abmelden der Spezialisten aus Abb. 2.6. Er führt Regie über das gesamte System.

Die **Implizite Aktionsplanung** führt die eigentliche aufgabenorientierte Planung und Steuerung von Roboterzellenaktionen durch. Deren Architektur sollte wiederum in mehreren UNIX-Prozessbereichen (=□ in Abb. 2.6) angelegt werden.

Die grundlegende Idee besteht nun darin, die Fertigungsaufgaben, die vom Benutzer an das System gestellt werden, durchgängig im Rechner zu halten und mit Hilfe von Spezialisten zu bearbeiten und zunehmend feiner aufzulösen ("Roter Faden"). Dieser "Rote Faden" entspricht einem, zur Laufzeit entstehenden **Arbeits-** oder **Aktionsplan** der Roboterzelle. Er wird von einem **Arbeits-** oder **Aktionsplaner** angesprochen und verändert. Der Aktionsplaner koordiniert und stößt die einzelnen Aktivitäten der **Spezialisten** an, nimmt deren (Zwischen)Ergebnisse entgegen und erzeugt letztlich den Aktionsplan. Er ist 'Quellsprache' des **Aktionsplan-Compilers**, der den Ak-

tionsplan in ausführbaren Zielsprachen-Steuerungscode der frei programmierbaren Betriebsmittel (RC, NC, BV, ..) übersetzt.

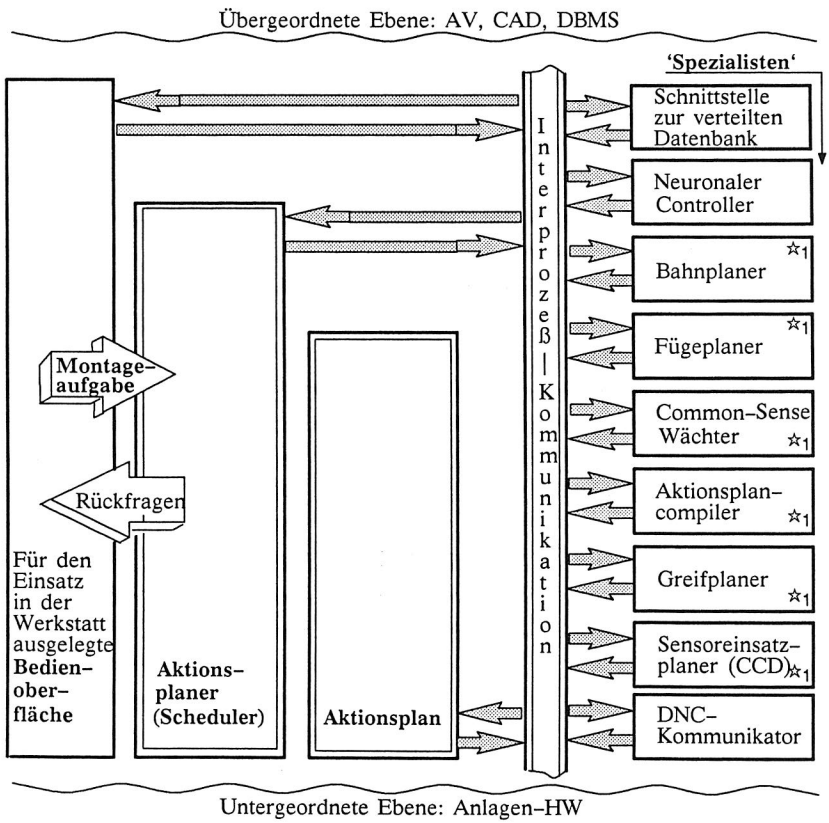


Abb. 2.6: Intertaskkommunikation für fertigungsnahe Roboteraktionsplanung

Dieser Aufbau entspricht einer Blackboardarchitektur mit einer Reihe unabhängiger Spezialisten. Der Online-Aktionsplan ist das zentrale Blackboard des Aktionsplaners.

Besondere Anstrengungen müssen während der Entwurfsphase der Systemarchitektur unternommen werden, eine Vermischung von Aktionsplan, rechnerintern modelliertem fertigungstechnologischem Wissen und Anlagenmodell zu verhindern. Zusammen

☆1 Eine detaillierte Behandlung dieser Spezialisten erfolgt in Kap. 3, 4 und 7

mit dem dynamischen Aufbau des Aktionsplanes zur Laufzeit ist die konzentrierte Repräsentation von statischem Montagewissen Grundlage für Ergänzung und Wartung der im Laufe des Systemeinsatzes wachsenden, vom System beherrschbaren Fertigungsbefehle. Diese Struktur muß auch bereits für sehr einfache Anweisungen eingehalten werden.

Aktionen die der Arbeitsplan beschreibt, müssen (!), sofern sie für das Umweltmodell relevant sind, unmittelbar eine Aktualisierung des Anlagenmodells (z.B. Geometriedaten oder Signalzustände) auslösen. Dazu reicht es im Regelfall jedoch nicht aus diese Aktualisierungen (z.B. von Geometriedaten) gesammelt, am 'Ende' eines impliziten Befehls, auszuführen (z.B. über den expliziten Programm-Interpreter). Die veränderten Anlagenzustände könnten bereits vor diesem Zeitpunkt zur weiteren Abarbeitung implizierter Sub-Aktionen benötigt werden.

Ebenso wie der Arbeitsplan, auf dem der Arbeitsplaner primär arbeitet, arbeitet der Aktionsplaner mit einer Reihe Spezialisten (=Klasse von Werkzeug-Objekten) zusammen. Der **Common-Sense-Wächter** überprüft implizierte Aktionen auf physikalisch nicht mögliche Zustände oder berechnet und setzt beispielsweise topologische Beziehungen zwischen Körpern. Bei den nachfolgenden Aktionen, z.B. automatisierte Berechnung von Bewegungssequenzen, fließen diese Erkenntnisse des Common-Sense-Wächters ein. Die **Schnittstelle zur Wissensbasis** bündelt die Datenbankaktivitäten auf Werkstattebene. Über sie wird beispielsweise das fertigungstechnologische Wissen aus der AV geladen, bzw. gelesen. Der **DNC-Kommunikator** ist für die Abwicklung der DNC-Kommunikation zu den angekoppelten Betriebsmitteln der Roboterzelle zuständig. Der **Fügeplaner** berechnet Fein- und Fügebewegungen. Der **Bahnplaner** segmentiert die Roboterzelle in eine Gitterstruktur, berechnet den Suchraum und hierin kollisionsfreie Grobbewegungen. Der **Greifplaner** soll körperfeste Greifkoordinatensysteme anhand des CAD-Modells berechnen. Der **Sensoreinsatzplaner** wählt beispielsweise Bildverarbeitungsroutinen aus, setzt sie zu einem Sensor-Skelettprogramm zusammen oder gibt hierfür Parametereinstellungen vor (z.B. für den Abgleich zwischen geometrischem Anlagenmodell und realer Zelle).

Bislang existieren nur im Bereich der spanenden Werkzeugmaschinen Systeme zur werkstattorientierten, prozeßnahen NC-Programmierung. Sie stellen spezielle Anforderungen an Hard- und Software (WOP-Softwareanforderungen s. DIN 66234). Aufgrund der Entwicklung der Rechnerhardware wird die Bedeutung der WOP noch weiter ansteigen; vor allem in Verbindung mit den sehr benutzerfreundlichen impliziten Verfahren, sowie für alle Anwendungen ohne Arbeitsvorbereitung. Abb. 2.7

zeigt die Kennzeichen des im Rahmen der vorliegenden Arbeit entwickelten, prozeßnahen Werkzeuges im Überblick.

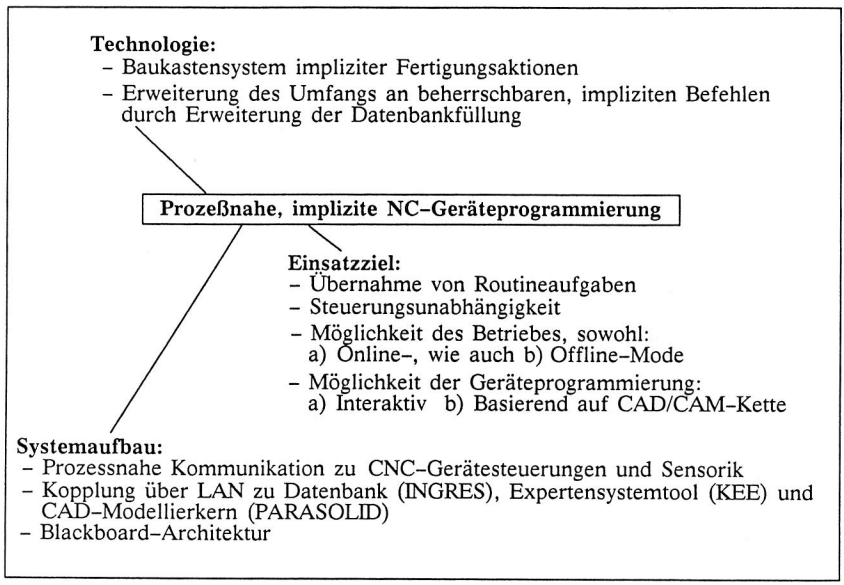


Abb. 2.7: Kennzeichen des prozeßnahen Werkzeuges in der Werkstatt

Große Bedeutung in der impliziten Roboteraktionsplanung besitzt die graphische Simulation. Hier unterscheidet sich die Fügetechnik wesentlich von z.B. der Drehbearbeitung, die meist auf ein zweidimensionales Problem reduziert werden kann, und deshalb hier auch die ersten Schritte in Richtung WOP unternommen wurden. Diese Reduzierung ist in der Bewegungs- und Positioniertechnik nicht zulässig. Vergleichbar sind nur Spezialfälle, wie die SMD-Leiterplattenbestückung. Hier konnten, aufgrund der planaren Problemstellung, höhere Automatisierungsgrade erreicht werden.

Aufgrund geringerer Fertigungstiefen, Produktkonstruktionen mit steigender Modularität und gemischt mechanischen und elektronischen Komponenten, sowie Demontage (Rohstoffrecycling) steigt die Bedeutung der automatisierten Montage und damit auch die Behandlung von fügetechnischen Aktionsprimitiven.

2.3 Objektorientiertheit und Verträglichkeit mit Prozeßnähe

Entscheidend für die hard- und softwaretechnische Realisierung der Werkzeuge sind Programmiersprache und Entwicklungsumgebung, da dadurch bereits Rückwirkungen auf die Konzeptionsphase entstehen. Der in den letzten Jahren entwickelte objektorientierte Ansatz (/91/, /116/, /115/) unterstützt hierbei durch folgende Charakteristika:

◆ Unterteilung umfangreicher Software in eindeutig adressierbare, gegeneinander abgekoppelte Kapseln, bestehend aus lokalen Instanzvariablen (Slots) und lokalem Funktionsumfang (Methoden, Operationen)
◆ Einmalig anfallende Bildung von Klassen (Objekte gleicher Art z.B. NC-Achse, vgl. abstrakte Datentypen in z.B. Modula) und davon konkrete Abzüge (=Instanziierung) von Instanzen (= lauffähige Objekte)
◆ (Multiple) Vererbung von Methoden und Daten in Klassenhierarchien (Klassen besitzen Ober- und Unterklassen, die selbst wieder Ober- bzw. Unterklassen besitzen usw.)
◆ Bildung isolierter Kapseln derart, daß zwischen deren Funktion und Semantik in der Realität und im Rechner möglichst eine Abbildung im Maßstab 1:1 gilt (Vorteil: Selbsterklärung für Wartung, Änderung, Erweiterung)
◆ Abgrenzung und Entkopplung der Objekte für effiziente Programmentwicklung und Wiederverwendbarkeit
◆ Aktivierung der Objekte (Agenten) untereinander durch Verschicken von Botschaften bzw. Nachrichten (=Message) und Ausführung der korrespondierenden Methode. Klassen, und damit all deren Objekte können Standardmethoden besitzen die defaultmäßig abgearbeitet werden (z.B. bei Initialisierung oder Terminierung). Im Fehlerfall können Nachrichten zurückgewiesen werden (vgl. Abb. 2.9).
◆ Kontextabhängige, unterschiedliche Reaktionen von Objekten auf ein und dieselbe Botschaft (Polymorphismus).

Vererbung ist beispielsweise wichtig für den hierarchischen Baukasten impliziter Befehle mit teilweise deckungsgleichen Eigenschaften. Aus diesen Gründen ist es günstig die Werkzeuge zur impliziten Aktionsplanung und -steuerung objekt-orientiert aufzubauen (vor allem für Werkzeuge im CAD/AV-Bereich), mit Einschränkungen bei prozeßnahen Teilen, die ein definiertes Zeitverhalten erfordern (diese z.B. in C). Probleme bereiten dynamisches Binden (Ändern der Objektmethoden zur Laufzeit) und Beseitigung überflüssiger, dynamisch zur Laufzeit angelegter Objekte, bzw. die

auf sie gerichteten Zeiger (automatische Speicherverwaltung, 'Garbagge collection'). Sie verringern die Geschwindigkeit und erhöhen den Speicherbedarf.

Objektorientierte Entwicklungstools, z.B. KEE /59/, gestatten schnelles Austesten neuer Ansätze ("Rapid Prototyping") und schnellen Durchlauf von Änderungszyklen. Dies gilt insbesondere, falls Regeln formuliert werden und die aus der Expertensystemtechnik bekannten Regelinterpreter-Mechanismen greifen. Die Ineffizienz des Durchsuchens eines Regelwerkes nach anwendbaren Regeln kann durch Formulieren von Metaregeln verbessert werden. Die Suchverfahren lassen sich so auf Regelpakete beschränken. Aus Gründen der Rechengeschwindigkeit kann anschließend eventuell eine Portierung in z.B. C++ folgen.

Die regelbasierte, wie auch die objektbasierte Wissensrepräsentation sind zwei mögliche Formen der rechnerinternen Repräsentation desselben semantischen Netzes, das das aufzunehmende Wissen modelliert. Die regelbasierte Wissensrepräsentation konzentriert sich auf die Kanten, die objektbasierte Wissensrepräsentation auf die Knoten dieses Netzwerkes /91/. Für prozeßnahe Teilsysteme (Laufzeitanforderungen) ist es günstig regelbasierte Repräsentationen möglichst durch objektorientiere Wissensrepräsentation und -verarbeitung zu ersetzen. Andere Formen sind z.B. Regeln, Constraints oder Prädikatenlogik erster Stufe (nach /91/).

'Wissen' (wissensbasierte Systeme) unterscheidet sich von 'Daten' (konventionelle Programme) dadurch, daß Wissen neben Fakten, auch vom "Rechner interpretierbare Anweisungen" enthält. Kommt darüberhinaus noch die Trennung zwischen Problemlösungsstrategie und heuristischem Wissen, sowie eine Erklärungsfähigkeit und eine Wissenserwerbskomponente hinzu, so wird ein wissensbasiertes System zum Expertensystem (vgl. /91/). Das in Kap. 2.2.2 konzipierte Werkzeug ist danach ein wissensbasiertes, prozeßnahes System. Die Übergänge sind jedoch fließend. In der Aktionsplanung eignen sich Expertensysteme zwar für die Aufbereitung klassifizierter Standardsituationen (/91/, /129/), sie scheitern jedoch an Problemen der Geometrieverarbeitung zur Laufzeit.

Zeitkritische Teile (z.B. direkte Prozeßkopplung) sollten konventionell prozedural programmiert werden. Sog. hybride Entwicklungsumgebungen (z.B C++), die prozedurale Sprachen (z.B. C) um objektorientierte Konzepte erweitern, lösen dieses Problem durch wahlweise Kombination prozeduraler und objekt-orientierter Module. Teile mit definierten Zeitanforderungen werden prozedural, Teile mit "offline-

Charakter" (z.B. Aktionsplaner) werden objekt-orientiert realisiert. Tabelle 2.8 faßt die in der vorliegenden Arbeit eingesetzten Umgebungen und Sprachen zusammen.

Einsatzbereiche, Beispiele	**In der vorliegenden Arbeit verwendet**
Teile der impliziten Aktionssteuerung und -planung mit überwiegendem offline-Charakter, z.B. Aktionsplaner	Hybride, objekt-orientierte Erweiterung von C (vgl. C++)
Datenbank-Manipulationen interaktiv, direkt In prozedurale Umgebung eingebunden	SQL embedded-SQL in C und Pascal /123/
Anlagen- Umweltmodellkonstruktion Trigger, Datenbank- CAD- Applikation	OSQ/OSL, 4.-GL /20/
Prozeßnahe Teile mit online-Charakter z.B. Steuerungsteil des neuronalen Controllers	C
Anlagensteuerung Industrieroboter, SPS, Sensorik	IRL, SRCL (RC), STEP5 (SPS), Assemblerähnliche Steuerungssprachen
Wissensbasiertes Konstruktionstool für neuronale Automatisierungslösungen	Common-Lisp, C, KEE (Lisp) /59/

Abb. 2.8: Eingesetzte Programmiersprachen und Entwicklungsumgebungen

Den Aufbau von Objektinstanzen am Beispiel des werkstattorientierten Aktionsplanungs- und Steuerungswerkzeuges zeigt Abb. 2.9. Die Gesamtmenge an Objekten kann grob in zwei Bereiche unterteilt werden. Die erste Gruppe umfaßt Objekte des Anlagenmodells, während die zweite Gruppe aktive Objekte der Aktionsplanung ('Werkzeug-Objekte') enthält. Objekte können in der für die vorliegende Arbeit eingesetzten objektorientierten Erweiterung von C über Unix-Prozeßgrenzen hinweg aktiviert werden. Eine Nachricht besteht aus der Angabe des Senders, des Empfängers und der Nutzdaten.

Liegt das Objekt, dessen Methode aktiviert werden soll, im selben Prozeß, dann entspricht die Methodenaktivierung einem sequentiellen Sub- bzw. Koroutinen-Konzept (vgl. KEE). Die Methodenaktivierung ist dann synchron, d.h. die aufrufende Objektinstanz wartet vor dem Weiterarbeiten auf eine Quittung von dem aufgerufenen Objekt und ist selbst zwischenzeitlich blockiert. Klassen sind selbst auch Objekte mit spezifischen Methoden wie z.B. Erzeugung einer Instanz dieser Klasse.

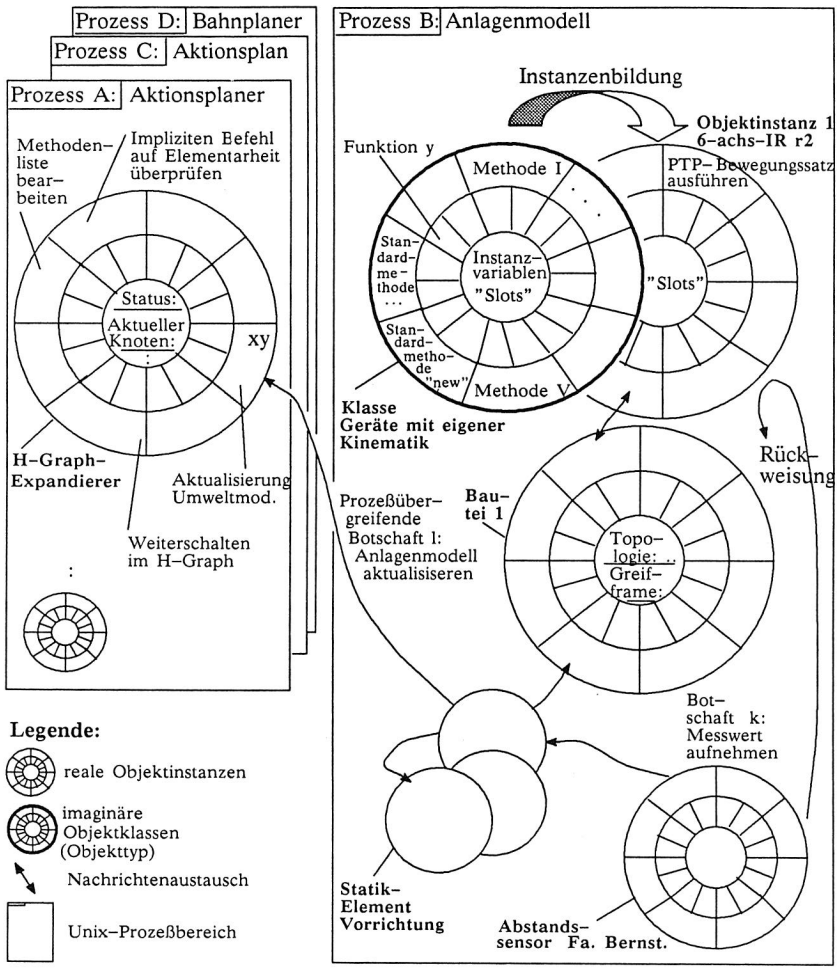


Abb. 2.9: Schalenaufbau von Objektinstanzen mit prozeßübergreifender Kommunikation (beispielhafter Auszug)

Liegt das aufgerufene Objekt in einem getrennten Prozeßbereich, kann der Methodenaufruf asynchron sein, d.h. es wird keine Antwort zurückgesandt, und die aufrufende Objektinstanz arbeitet ohne Unterbrechung weiter (paralleles Prozeß-Konzept). Sie können z.B. erst durch den Methodenaufruf aktiviert werden, d.h. sie

überprüfen nicht selbst ob eine Botschaft ansteht (kein "busy wait"). Erreicht in diesem Fall eine weitere Methode eine Objektinstanz, während diese arbeitet, wird die neu hinzukommende Message in eine Warteschlange eingereiht.

2.4 Robotik und künstliche Intelligenz

Künstliche Intelligenz (KI) konnte bislang nur unscharf definiert werden /42/. Für die praxisnahe Fertigungsautomatisierung ist es deshalb besser, anstatt formalen Versuchen, die KI anhand ihrer Applikationen und Teilbereiche zu definieren. Hier zeigt sich eine große Übereinstimmung mit den Einzeldisziplinen der Robotik (Bildverarbeitung, wissensbasierte Planerstellung, Logische Programmierung (Inferenzbildungen), etc.).

Entwicklungs-schwerpunkt in der Robotik	Mechanischer Teilbereich	Elektrotechnischer Teilbereich	Informatik-Teilbereich
Modellierung mechanischer Systeme	Hybride Mehrkörpersysteme (MKS), Bewegungsgleichung Eigenform-/ Eigenfrequenzermittlung	Parameteridentifikation der elektr. Antriebe	Numerische und symbolische Berechnungsmethodik
Implizite NC-Geräteprog.	CAD von Roboterzellenkomponenten, Kinematische Simulation	Simulation der Gerätesteuerungen Neuronale Steuerungstechnik und Regler	KI,"intelligentes Verhalten", Lernvorgänge, Datentechnische Integration,CAM, DB
Entwicklung von Roboterzellenkomponenten	Leichtbau Stellglieder für die Beeinflussung mechanischer Systeme	Schnelle Signalverarbeitung Sensorik (Bildverarbeitung, taktile Sensorik)	Genormte Kommunikation Mobile Systeme
Steuerungs-Optimierung ⋮	Trajektorienplanung (Energie-/Zeitoptimierungen)	Echtzeitbetriebssysteme für Gerätesteuerungen	Mustererkennungsalgorithmen

Abb. 2.10: Beispiele wichtiger Schwerpunkte in der Robotik

Die Robotik wiederum, ist Entwicklungsgrundlage zukünftiger Roboterarbeitszellen. Sie stellt ein stark interdisziplinäres Forschungsgebiet dar, mit wachsendem Einfluß

der Informationstechnik, z.B. gegenüber klassischem Maschinenbau. Abb. 2.10 zeigt die Schwerpunkte der vorliegenden Arbeit im Bereich der Robotik.

Für den Sondermaschinen- bzw. Betriebsmittelbau von flexibel automatisierten Anlagen entstehen hieraus völlig neue Anforderungen. Es gilt die Flexibilität der heutigen IR-Generation, die auf einer kinematischen Kette aus z.B. sechs Achsen basieren und analog dazu weitere, freiprogrammierbare Betriebsmittel zum Aufbau einer Roboterarbeitszelle, in der Praxis umzusetzen.

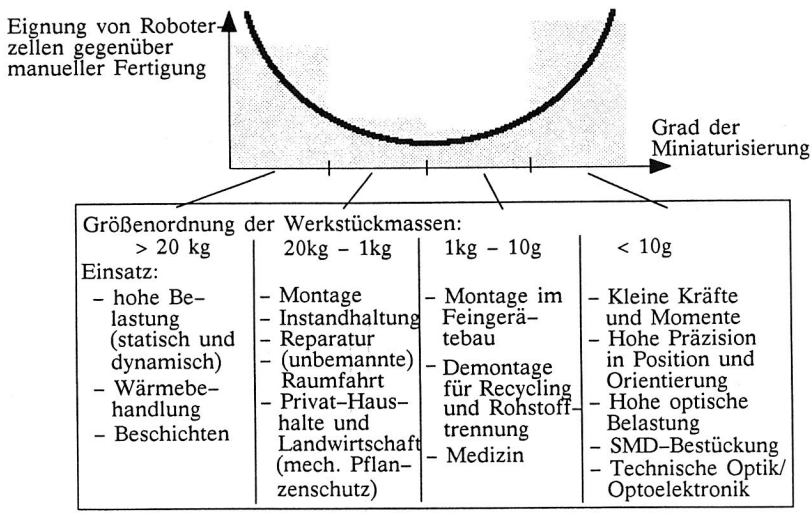


Abb. 2.11: Einsatz von Industrierobotern (einschl. Zellenverbund)

Selbständig handelnde Fertigungszellen, und damit verbunden die Robotertechnik, gewinnen nicht nur aufgrund des Zwangs zu hohen Automatisierungsgraden in der industriellen Fertigungstechnik an Bedeutung, z.B. als Folge steigender Lohn- und Nebenkosten. Sie wird neue Anwendungsfelder in der Raumfahrt, Medizintechnik und Haushalt /Dienstleistung (Landwirtschaft und Umweltschutz) erschließen /83/. Besondere Stärken gegenüber der menschlichen Arbeitskraft (Abb. 2.11) liegen bei sehr feinen (hohe sensorische Anforderungen), sowie bei physisch stark belastenden Tätigkeiten (z.B. statische Kraftaufbringung über längeren Zeitraum). Zukünftige Entwicklungen sind dadurch vorgezeichnet.

Manuelle Montage stößt z.B. durch die zunehmende Miniaturisierung der zu montierenden Bauteile an Grenzen. Qualitätsprobleme führen hier zu einem Rationalisierungsdruck im Bereich kleiner Stückzahlen.

Die Informationsverarbeitung in CAD/CAM-Werkzeugen zur impliziten Programmierung einer Fertigungszelle aus der Fertigungsaufgabe heraus, beinhaltet Teilprobleme, die sich einteilen lassen in die drei Bereiche "Analytik", "Symbolik" und "Lernen" (Abb. 2.12). Die Abgrenzungen sind starken Veränderungen unterworfen. Die Unterteilung besitzt fließende Grenzen; exakte Definitionen und Abgrenzungen sind noch nicht vorhanden. Dies gilt vor allem für Lernprozesse deren biologische Grundlagen noch nicht ausreichend verstanden sind.

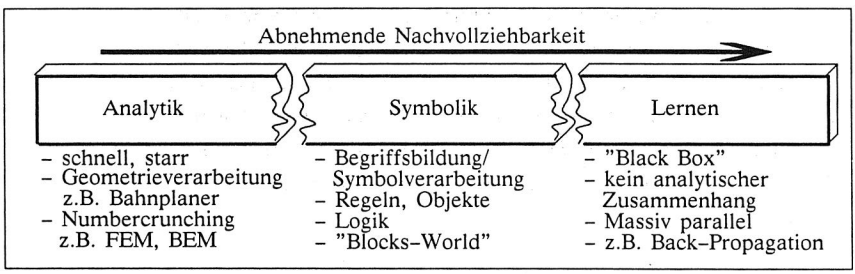


Abb. 2.12: Einteilung mit fließenden Grenzen

Von besonderer Bedeutung für die praktische Umsetzung der in diesem Kap. 2 entworfenen Konzepte ist die Integration von symbolverarbeitenden Methoden und schneller, numerischer Geometrieverarbeitung (Transformationen, Vorwärts-Rückwärts-Kinematik). Bei der symbolischen Behandlung von impliziten Befehlen finden sich Ansätze in der sogenannten "Blocks-World" /74/.

Im Beispiel neuronaler Netzwerke ist die Wissensrepräsentation über alle Verarbeitungseinheiten verteilt und damit für den Betrachter nicht mehr lesbar. Praktische Einsatzgrenzen zeigen sich im Bereich des Lernens auf Rechnern mit Von-Neumann-Architekturen im Vergleich zu massiv parallelen Systemen auf der Basis neuronaler Netze.

3. Autonome Abarbeitung impliziter Montagebefehle

Die Fähigkeit eines impliziten Aktionsplanungswerkzeuges, komplexe Fertigungsanweisungen für Roboterzellen entgegenzunehmen und die Detailplanungen der erforderlichen Aktionen autonom durchzuführen, basiert auf einem laufend aktualisierten Umwelt- bzw. Anlagenmodell. Es ist Grundlage online ablaufender, geometrischer Algorithmen und Logik der in Kapitel 2 vorgestellten Spezialisten. Die Aktualisierung des Anlagenmodells ist Voraussetzung für selbständige Reaktionen des Systems und Basis für alle nachfolgend implizierten Befehle, sowohl im interaktiven Betrieb in der Werkstatt, wie auch bei der Abarbeitung von Arbeitsplänen, die in der AV erstellt wurden.

3.1 Simulation einer Roboterarbeitszelle

Mit dem rechnerinternen Anlagenmodell wird nicht primär das Ziel verfolgt, die Aktionen des Systems zu visualisieren, sondern wesentlich wichtiger ist, die Zellensituation zu jedem Zeitpunkt dynamisch mitzuführen. Die Ansteuerung der Simulation zur graphischen Darstellung des Systemzustandes stellt nur einen Seiteneffekt dar.

Die permanent mitlaufende Aktualisierung des begleitenden Prozeßmodells (vgl. /29/) umfaßt:

- Robotermechanik und -kinematik, sonstige Kinematiken
- Sensorik, z.B. Sensorsignale (im 'Rohzustand' und vorverarbeitet)
- Steuerungsinternes Abbild: Schalt- und Signalzustände, z.B. von

analogen oder digitalen I/O-Schnittstellen, Merkern, Bitspeichern, Variablenspeicher

- ◆ Geometrie raumfester und ortsveränderlicher Bauteile, einschließlich der zu montierenden und zu fertigenden Werkstücke
- ◆ Weitere Betriebsmittelzustände

Das Umweltmodell stellt ein dynamisches System dar; es ist während des Betriebes des Aktionsplanungssystems stets Änderungen unterworfen. Während der Auflösung eines impliziten Fertigungsbefehls wird das Anlagenmodell in der Regel selbst mehrmals verändert, bzw. die gerade aktuellen Zustände als Ausgangsbasis für weitere, unterlagerte Aktionen benötigt.

Im Falle der direkten Aktionssteuerung ist es das "Gedächtnis" der Roboterzellensteuerung und muß gegenüber der realen Roboterzelle zu jedem Zeitpunkt einen konsistenten Stand besitzen. In diesem Abschnitt wird eine Auswahl der wesentlichsten Modelle beschrieben, die für eine implizite Aktionsplanung erforderlich sind.

3.1.1 Dynamisches, rechnerinternes Geometriemodell

Roboter- und Roboterzellensimulation, sowie automatisierte Offline-Programmgenerierung beruht auf der rechnerinternen Geometrierepräsentation der realen Zellenkomponenten. Systeme, die diese Grundlage nicht besitzen, sind reine Texteditoren und werden hier nicht weiter betrachtet. Für die Realisierung des Geometriemodells bestehen zwei grundsätzliche Möglichkeiten:

1.) Verwendung eigener Datenstrukturen
2.) Direktes Arbeiten auf dem CAD-Volumenmodell

Die erste Alternative der eigenen Datenstrukturen kann bei effizienter Wahl die Laufzeiten der Simulationsalgorithmen wesentlich verkürzen. Kanten- (Drahtmodelle (wire-frames)), Flächen- oder Volumenmodelle sind prinzipiell möglich. Die Datenstrukturen können für die Belange des Simulationssystems optimal angelegt werden und auch technologische Werte aufnehmen. Im Falle der Verwendung eigener Datenstrukturen existieren wieder zwei Möglichkeiten: Die Geometrie kann aus CAD-Volumenmodellen (z.B. CSG (Constructive Solid Geometry) oder BRep (Boundary Representations)) abgezogen werden (Beispiel: System CARo mit Schnittstelle zu ROMOLUS, Shape Data Ltd., /60/). Die Konvertierung erfolgt z.B. über CAD-Schnit-

tstellen-Formate wie STEP, IGES, VDA-FS, SET, VDA-PS /62/. Eigenständige Editoren mit vereinfachter Konstruktionsfunktionalität ('Erzeugen von Stützen, transl. od rot. NC-Achse etc.') haben den Vorteil der Unabhängigkeit von verfügbaren CAD-Systemen und sind für die werkstattorientierte Programmierung von Bedeutung. Günstig ist eine wahlweise Kombination beider Möglichkeiten.

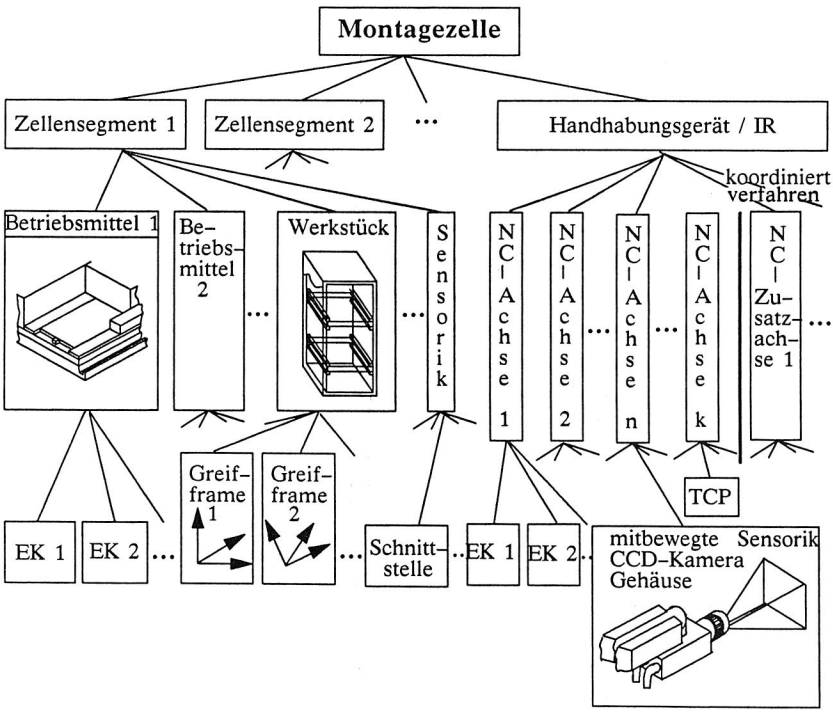


Abb. 3.1: Anlagenmodell in der Aktionsplanung (EK = Elementarkörper)

Die zweite Alternative, direkt 1:1 auf dem CAD-Modell zu arbeiten führt zu Robotersimulationssystemen, die häufig als Zusatzmodul von CAD-Systemen ausgelegt werden. Freiformflächen können durch Splines bzw. B-Splines /37/ modelliert werden. NURBS (non-uniform B-spline surface) /84/ werden zukünftig durch neue Grafikstandards wie PHIGS /122/ unterstützt. Der große Vorteil dieser Lösung besteht in der

problemlosen Integration in die Verfahrenskette der Produktion, AV und Konstruktion. Die heute verfügbaren CAD-Kinematik-Zusatzmodule verwenden dennoch rechnerintern wieder eigene Datenstrukturen. Die Funktionalität die sich dadurch ergeben würde, direkt auf einem Standard-CAD-Modellierer (z.B. PARASOLID) zu arbeiten, geht dadurch verloren.

Grundsätzlich sollte der Detailliertheitsgrad des Geometriemodells nur so groß sein, wie sie für die Aktionsplanung (vor allem die Fügeplanung) erforderlich ist. Dies wirkt sich günstig auf die Rechenzeiten und die Hardwareanforderungen aus (Echtzeitbetrieb). Parallel dazu können schnelle graphische Simulationen in Realzeit zur Darstellung der Auswirkungen impliziter Aktionen genutzt werden. Sie unterstützen den Benutzer und visualisieren den Zellenzustand zu jedem Zeitpunkt. Mitzeichnen der Verfahrbahnen, Blickrichtungen, Ansichten und Manipulationen können gewählt werden, die in der Realität nicht möglich sind (Ein-/ Ausblendung verdeckter Kanten, Zeitlupe, Sichtbarkeit/Unsichtbarkeit von Körpern, Ausschnittsvergrößerungen etc.).

Für ein prozeßnahes Aktionsplanungswerkzeug, das auf Werkstattebene eingesetzt werden soll, muß das Anlagenmodell so ausgelegt werden, daß die erforderlichen Berechnungsalgorithmen mit einem begrenzten Rechenaufwand auskommen. Die Datenstrukturen können dazu ähnlich einem B*-Baum aufgebaut und die eigentlichen Datensätze in den Blattknoten plaziert werden. Einfache dreidimensionale Zellenmodelle von Roboter und Anlage basieren beispielsweise auf 3D-Drahtmodellen geometrischer Grundkomponenten wie Kegelstumpf, Quader, Pyramide, etc. (nach /77/). Das im Rahmen dieser Arbeit auf Werkstattebene eingesetzte, prozeßnahe Geometriemodell basiert auf einem derartigen, vereinfachten Geometriemodell mit Klassenhierarchien (vgl. /39/). Knoten der hierarchisch höheren Niveaus dienen sowohl zur Navigation im Baum als auch zur Strukturierung des Umweltmodells. Durch horizontale Verknüpfungen in einem B*-Baum lassen sich Selektions-, Durchlauf und Löschvorgänge beschleunigen, da Auf- und Absteigen im Baum entfällt (gutes Laufzeitverhalten durch effiziente Zugriffe auf mehrfach verkettete Strukturen).

Die gesamte Roboterarbeitszelle ist unterteilt in mehrere Zellensegmente und Kinematiken (Abb. 3.1). Jedes Zellensegment setzt sich aus Starrkörpern (Assemblies) zusammen, die sich relativ zueinander im Raum bewegen lassen (z.B. Betriebsmittel-, Werkstückteile, raumfeste Sensoren). Jede dieser Baugruppen besteht aus starr miteinander verbundenen Elementarkörpern und besitzt ein körperfestes

Bezugssystem. Jedem Objekt lassen sich körperfeste Greifkoordinatensysteme konstruktiv zuordnen.

Handhabungsgeräte werden durch eine kinematische Kette von NC-Achsen aufgebaut. Mitbewegte Sensorik, wie z.B. das CCD-Gehäuse eines Bildverarbeitungssensors werden mit der entsprechenden NC-Achse gekoppelt. Der TCP wird meist der letzten Achse, z.B. der 6. Grundachse des Industrieroboters, zugeordnet. Hierauf beziehen sich die Vorwärts- und Rückwärts-Transformationen (vgl. /81/). Die NC-Achsen-Körper werden wieder analog aus starr verbundenen Elementarkörpern aufgebaut. Synchronisierte NC-Zusatzachsen werden angebunden, sofern die Robotersteuerung die Steuerung der Zusatzachsen für eine koordinierte Bewegung von TCP und Hilfskinematik, z.B. Zwei-Achs-Dreh/Kipptisch für Schweißaufgaben, übernehmen kann (Abb. 3.1).

3.1.2 Mechanisches Modell

Das Anlagenmodell muß nicht nur die Geometrie, sondern auch neben der Kinematik weitere Technologiedaten (Geschwindigkeits- und Beschleunigungsrampen, Achsbegrenzungen etc.) beinhalten. Für die geometrische Modellierung der Zelle, wie auch der Werkstücke sind heutige CAD-Systeme ausreichend. Dies gilt nicht für technologische Daten; sie müssen meist extern bereitgestellt werden.

Wesentlicher Bestandteil ist die Simulation der CNC-Steuerung (z.B. RC). Hierbei ist die Simulation der Kinematik von Bewegungssystemen, wie z.B. von Industrieroboter, von besonderer Bedeutung. In der komplexen Kinematik, z.B. eines 6-Achs Roboters, besteht ein wesentlicher Unterschied zur Simulation spanender Verfahren (z.B. NC-Drehmaschine).

Grundlage der Simulation von Handhabungssystemen ist deren vollständige kinematische Beschreibung1). Kinematische Mechanismen können sowohl aus geschlossenen als auch offenen kinematischen Ketten bestehen. Beispiele von Handhabungssystemen sind Roboter oder auch bereits die Kombination zweier linearer NC-Achsen in Modulbauweise, bis hin zu einfachen Pneumatikzylindern. Mit Hilfe von, mit den einzelnen Achskörpern des aktiven Bewegungsmechanismuses körperfest verbundenen Koordinatensystemen, entsprechend der Anordnung der Schub- und Drehachsen, lassen sich Transformationsmatrizen bilden, die die Bezugssysteme der

1) Die kinematischen Grundlagen wurden bereits umfassend in der Literatur behandelt. Siehe z.B. /83/, /47/, /60/.

Vorwärts- bzw. Rückwärts-Transformationen definieren (Transformationen zwischen Achswinkelsystem und Arbeitskoordinatensystem und umgekehrt). Rechenvorschriften, z.B. nach Denavit-Hartenberg /22/, vereinheitlichen dieses Vorgehen und ermöglichen eine Automatisierung der kinematischen Modellierung (s. Abb. 5.6). Die vier D-H-Parameter können in einer relationalen Datenbank (Kap. 5.2.1 Anlagenwissen: WZ zum Aufbau kinematischer Ketten aus einzelnen NC-Achsen) gehalten und für die Bewegungssimulation gelesen werden. Analog lassen sich gerätetypspezifisch Gelenkbegrenzungen, Beschleunigungskurven etc. zu dem jeweiligen Gelenk zugehörig ablegen. Die Definition des TCP kann im Euler-Koordinatensystem, mit den Eulerwinkeln A, B, und C für die Orientierung gemäß /81/ erfolgen:

$$\textbf{Euler}\ (\phi, \theta, \psi) = \text{Rot}\ (z,\phi)\ \text{Rot}\ (y,\theta)\ \text{Rot}\ (z,\psi) \qquad (3.1)$$

Für die rechnerinterne Darstellung eignet sich die Verwendung homogener Koordinaten /47/, /37/; die auch in der vorliegenden Arbeit eingesetzt wurden.

Über die kinematische Modellierung hinaus, wird bei höheren Genauigkeits- oder Geschwindigkeitsanforderungen ein dynamisches Modell der Industrieroboter erforderlich. Effekte der Roboterdynamik sind vor allem dann nicht mehr vernachlässigbar, wenn Bewegungssysteme Segmente enthalten, die nicht mehr als Starrkörper modelliert werden können (vgl. /61/, /112/). Dies gilt besonders für zukünftige Entwicklungen hochmobiler Handhabungssysteme. Reibungseinflüsse, Massenträgheiten, Spiele (z.B. in event. vorhandenen Getrieben) oder Antriebsdynamiken beeinflussen das Arbeitsergebnis. Für den Reglerentwurf (z.B. Zustandsregler) und die Bestimmung der Bewegungsgleichungen hat sich die Modellierung zu hybriden Mehrkörpersystemen (**MKS**, gemischt starre und elastische Segmente) bewährt /83/, /61/.

Aufwendig sind stark redundante Kinematiken wie z.B. antromorphoische Roboterarme (bionische Systeme) mit 7 Freiheitsgraden die die hohe Beweglichkeit des menschl. Armes nachbilden (z.B. Hintergreifen eines Medizin-Roboters zur Unterstützung bei chirurgischen Eingriffen).

3.1.3 CNC-Robotersteuerung

Abweichungen zwischen gerätetechnischer Realisierung (Regelkreis, A/D-Wandlung etc.) und z.B. auf dem Rechner nachgebildeten RC-Steueralgorithmen, führen zu Unterschieden zwischen realem Verhalten und Simulation, bzw. Aktionsplanung. Ziel ist, die Algorithmen aus der realen Steuerung herauszulösen und möglichst 1:1 in die

Bewegungssimulation zu übertragen. Im Falle neuer Steuerungsgenerationen (s. Kap. 2.2.2), sind die Steueralgorithmen meist z.B. in der Hochsprache C implementiert, anstatt überwiegend maschinennaher Programmierung (Assembler).

Das Verhalten der Robotersteuerung kann dadurch detailliert emuliert werden. Dies gilt für alle Interpolationsarten wie unsynchronisiertes Verfahren der Achsen (PTP), Linear-, Circular-, oder Splineinterpolation. Allgemeingültige Algorithmen auf Seiten des Simulationssystems (z.B. numerische Rückwärtskinematik für unterschiedliche Gerätesteuerungen) sind zwar prinzipiell erstrebenswert, liefern jedoch Abweichungen zwischen realer Anlage und Simulation. Robotersteuerungen zeigen vor allem spezielle Eigenheiten in folgenden Punkten:

- Vorwärts- und Rückwärtstransformation mit Fehlerbehandlung (z.B. Anschläge)
- Die Behandlung von Mehrdeutigkeiten (singuläre Stellen speziell in der Rückwärtstransformation)
- Genauigkeiten (Abweichung Soll-/Istpositionen und Einflüsse der Achsregelungen, Schleppabstände etc.), Bahntreue bei unterschiedlichen Einstellungen des Geschwindigkeitsoverrides (besonders wichtig für den Spezialist 'Bahnplaner') etc.

Das konkrete Verhalten des Roboters bezüglich dieser Eigenheiten ist abhängig von der aktiven Robotersteuerung, d.h. von der echtzeitfähigen Implementierung der zugehörigen Algorithmen.

Für logische Berechnungen in der Aktionsplanung ist die RC-Simulation (einschl. Editieren, Anzeigen) der Maschinendaten (z.B. Verzögerungswerte), der an Ein- und Ausgangsbaugruppen anliegenden Signale, sowie der steuerungsinternen Merker, Register etc. wichtig. Eine weitere Funktionalität sind nachgebildete (Anwendungs-) Regler-Mechanismen, z.B. von in Roboterprogrammen integrierten Sensorfunktionen. Sie werden im Vergleich zur Bahninterpolation mit höherer Taktfrequenz, zeitgleich zum IR-Hauptprogramm, ausgeführt (z.B. SDA in SRCL). Die Sensorsignale (analog oder digital je nach Schnittstelle) liefern z.B. online einen 'Abdräng'-Vektor, der der ideal programmierten Bahn überlagert wird. Voraussetzung ist die Modellierung der Sensorik.

3.1.4 Gerätemodelle und Sensorik

Geräte ohne aktive Kinematik, wie schaltende Sensoren oder Bildverarbeitungssysteme müssen in ihrer Funktion im Simulationssystem abgebildet werden (Sensor-

geometrie nach Kap. 3.1.1). Hierzu werden zwei in dieser Arbeit verwendete Sensoren eingehender behandelt:

Physikalisches Prinzip	Anwendung in der vorliegenden Arbeit
1. Eindimensionale Abstandssensorik und Laserscanner	Neuronaler Bewegungscontroller (Kap. 7.2)
2. CCD-Bildverarbeitungssensor	Impliziter CAD-Abgleich (Kap. 4.2)

Abb. 3.2: Beispiele zur Sensorsimulation

Problematisch ist hier die Genauigkeit der Modellierung der physikalischen Effekte (vgl. /92/). Am Beispiel der Bildverarbeitung soll dies verdeutlicht werden. Für viele Anwendungsfälle zeigt sich, daß z.B. die Berechnung geometrischer Umrisse (Projektionen in die CCD-Bildebene) von Körpern, die sich zum Zeitpunkt der Aktivierung der CCD-Kamera im Sichtkegel befinden, ausreicht. Zufällige Abweichungen in Position und Orientierung der so erfaßten Teile fehlen, da die Berechnung alleine aus dem rechnerinternen Geometriemodell heraus erfolgt. Durch Vorschalten eines Zufallsgenerators für Verschiebung und Rotation der aufzunehmenden Teile (z.B. vorgegebene Größenordnung < 3mm, bzw. < 2°) zum Zeitpunkt der Bildaufnahme können reale Toleranzen simuliert werden. Die Reaktionen einer Roboterzelle auf derartige Fehler (Ausweichstrategien, Unterprogrammverzweigungen etc.) lassen sich so austesten.

Das in dieser Arbeit verwendete, idealisierte Kameramodell für die Sensoreinsatzplanung (nach Kap. 4.2) berücksichtigt keine physikalischen Effekte wie z.B. Reflexion. Abb. 3.3 zeigt ein Beispiel des binärisierten Graubildes eines Wälzlagerkäfigs mit metallisch hochreflektierender Oberfläche (bei konstanten Parametereinstellungen des Bildverarbeitungssystems und der Beleuchtung). Die Seitenlage der Käfige läßt sich nur bei Modellierung dieser physikalischen Effekte bestimmen. Beispielsweise lassen sich mit der Ruler-Technik (vgl. Adept-Vision) strahlenförmig (z.B. mit Zentrum Flächenschwerpunkt) Lineale werfen und die Flankensprünge der Grauwertverläufe entlang dieser Lineale auswerten. Durch die DNC-Kopplung des werkstattorientierten Aktionsplanungswerkzeug zum realen Bildverarbeitungsrechner (reale Sensorwerte) kann die vereinfachte Simulation jedoch teilweise kompensiert werden (s. Kap. 4.2.2). Diese Prozeßkopplung ist der

wesentliche Unterschied zu Werkzeugen, die ausschließlich im Bereich AV, CAD sitzen.

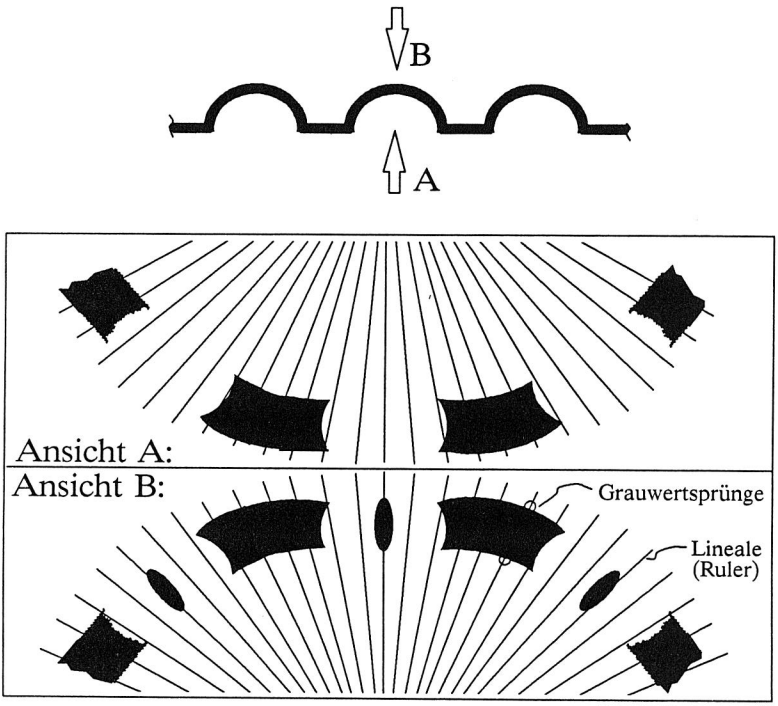


Abb. 3.3: Binärisiertes Grautonbild einer Wälzlagerkäfig-Hälfte aus Tiefziehblech

Eindimensionale Laserabstandssensoren arbeiten z.B. nach dem Prinzip des Michelson-Interferometers (Zählung von Intensitätsmaxima von sich überlagernden Meß- und Referenzstrahlen) /54/, /120/. Sie sind mittlerweilen stark miniaturisiert verfügbar und werden deshalb für die Sensorsimulation nach Kap. 7.2 (Neuronaler Bewegungscontroller) zugrundegelegt.

Hierfür weniger geeignet ist der Triangulationssensor, dessen Meßprinzip auf der diffusen Reflexion eines Laserstrahls an der Zieloberfläche basiert. Die nicht-fluchtende Anordnung zwischen Lichtquelle und Empfängeroptik führt zu Abschattungs- und Integrationsproblemen.

Weitergehende Kontur- und Strukturinformationen liefern Laserscanner (Halbleiter-CCD-Liniendetektor), die z.B. zur Online-Nahtverfolgung beim Schutzgasschweißen verwendet werden. In Kap. 7.2 werden sie zur Abstands- und Strukturbestimmung als Input für neuronale Lernprozesse von Feinmotorik benutzt. Folgende Eigenschaften realer Laserscan-Sensoren (z.B. nach /101/) müssen hierzu simuliert werden (Abb. 3.4, Teil 1):

- Anzahl scans/s: 2.5, 5, 10
- Scan-Winkel α: 10°, 14°, 20°, 28°, 40°
- Meßpunkte/scan: 200 bei 10 scans/s, 400 bei 5 scans/s, 800 bei 2.5 scans/s

Die rechnerinterne Simulation eindimensionaler Abstandssensorik im Umweltmodell des Aktionsplanungswerkzeuges beruht auf der gebündelten Aussendung von Licht und dem Empfang von reflektierten Strahlen. Die Simulation des Laserscanners kann vereinfacht werden, indem für jeden Winkel (z.B. 1,2,3) innerhalb des Meßraumes die eindimensionale Abstandssensorik aufgerufen und daraus die räumliche Information abgeleitet wird .

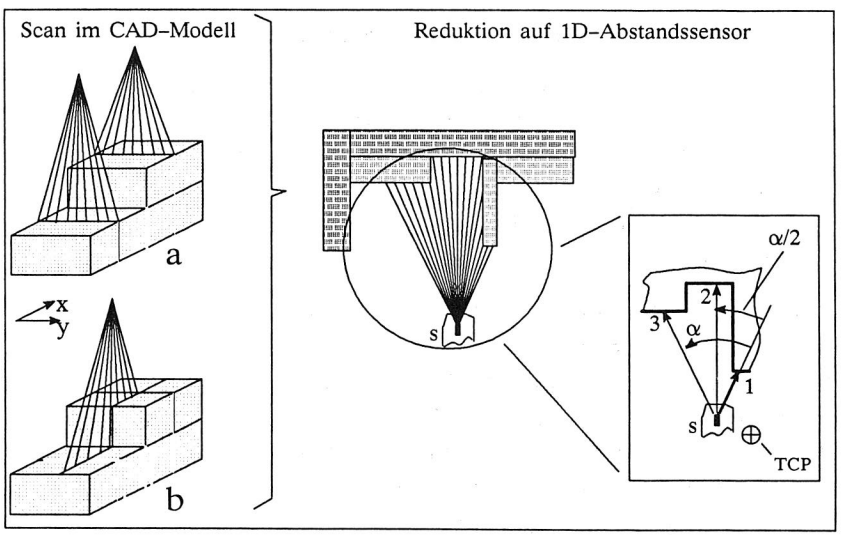


Abb. 3.4; Teil 1: Simulation eines Laserscan-Sensors im Zellenmodell

Der Algorithmus berechnet die Länge des Meßstrahles zwischen der Austrittsstelle am Sensorkörper und dem ersten Auftreffpunkt auf einer Fläche des Anlagenmodells.

Aufgrund des häufigen, internen Zugriffs auf das Umweltmodell wurde die Berechnung dieser Sensorrohdaten nicht im Prozeß Sensorsimulation durchgeführt, sondern in den Prozeß Umweltmodell ausgelagert.

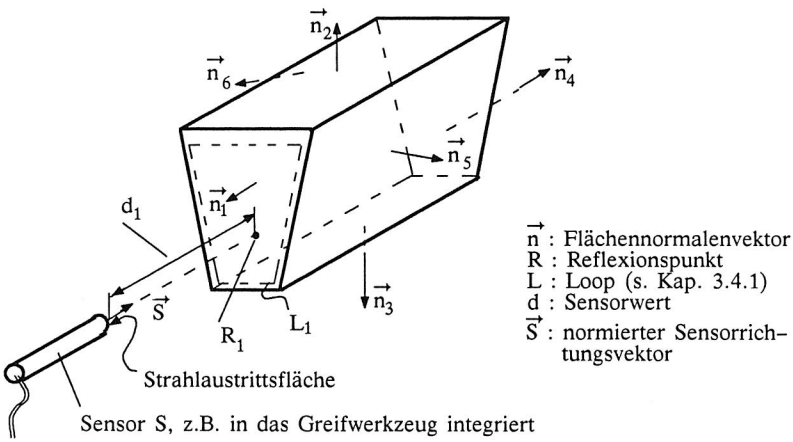


Abb. 3.4, Teil 2: Simulation eines eindimensionalen, optischen Abstandssensors S

Für die Berechnungen im Aktionsplanungssystem muß der Algorithmus auf minimale Rechenzeit optimiert werden, da im allgemeinen Fall die Abprüfung aller Flächen aller Körper sehr rechenaufwendig wird. Für Spezialfälle von Körpern mit ebenen Begrenzungsflächen kann ein vereinfachter und schneller Algorithmus zur Sensorwertberechnung benutzt werden. Reflexionspunkt R_1 (Abb. 3.4, Teil 2) muß folgende Bedingungen erfüllen:

- d_1 liegt im Meßbereich des Sensors (Abstand und Einfallswinkel)
- R_1 liegt auf einer Fläche des Körpers des Umweltmodells
- R_1 liegt innerhalb des zugehörigen Loops L_1 der die Fläche L begrenzenden Umfangskurve (Loop) L (vgl. Kap. 3.4.1)
- $d_1 < d_n$; n = alle weiteren Auftreffpunkte auf Flächen des gleichen Körpers oder Flächen aller weiteren Körper, d.h. anschaulich: keine Abschattung

Für mitbewegte Sensoren, die z.B. körperfest mit der letzten NC-Achse (Greif-WZ) des Industrieroboters verbunden sind, müssen $\vec{S}$ sowie obige Werte entsprechend den Transformationsmatrizen der kinematischen Kette während der IR-Bewegung stets neu berechnet werden.

3.2 Roboteraktionsplanung im Online-Betrieb

Die Online-Aktionsplanung setzt implizite Montagebefehle in, für die einzelnen Aktoren und Sensoren der Roboterzelle ausführbare Aktionen um, indem sie den einzelnen Spezialisten, wie z.B.:

- Fügeplaner: Zuständig für Fein- und Fügebewegungen (Kap. 7)
- Bahnplaner: Gittersegmentierung und Suchraumberechnung (Kap. 3.2.3)
- Greifplaner: Geometriealgorithmen für körperfeste Greifframes (Kap. 3.2.4)
- Sensoreinsatzplaner: z.B. Parametrisieren von Bildverarbeitungsroutinen (Kap. 4.2)

Aufgaben bei Bedarf zuteilt und deren Lösungen entgegennimmt (Aufgabenverteilung). Der Aktionsplaner aktualisiert das Umweltmodell, einschließlich Bildschirmgraphik, und veranlaßt die Compilierung des Aktionsplanes zum Steuercode, bzw. übergibt diesen dem DNC-Kommunikator. Im interaktiven Mode fordert er eventuell fehlende Daten über die Benutzeroberfläche an.

3.2.1 Prozeßnaher Dialog

Die WOP-Eignung und die stark 3-dimensionale Montagetechnik bestimmen die Gestaltung der Dialogschnittstelle zum prozeßnahen Aktionsplanungswerkzeug. Für die Benutzung auf Werkstattebene muß die Aufgabenformulierung einfach sein, d.h. mit einer möglichst geringen Anzahl an Parametern auskommen.

Für die implizite NC-Geräteprogrammierung (sowohl interaktiv wie arbeitsplanbasiert) eignet sich folgendes Grundformat einer Anweisung:

<AKTOR/SENSOR>	**<OPERATOR>**	**<Objekt(e)/Einzelheit>**
Wer	*Was*	*Operand, Zielobjekt(e)*

Die Bestandteile eines impliziten Befehls sind demnach:

- **Aktor:** Primär aktives Betriebsmittel oder Betriebsmittelgruppe (z.B. Roboter, Sensor, bis hin zur gesamten Zelle),
- **Impliziter Befehl:** z.B. "Abrücken", "Justieren", etc.,
- **Objekt:** Bauteil, Assembly oder Einzelheit (Geometriefeature), auf den der implizite Befehl angewandt werden soll.

Im interaktiven Betrieb in der Werkstatt werden diese Befehlskomponenten durch "Mouseklick" zu einem impliziten Befehl zusammengesetzt und dem System zur

Bearbeitung übergeben (Abb. 3.5 zeigt Beispiele impliziter Montagebefehle). Bei der Abarbeitung von AV-Arbeitsplänen sind sie Bestandteile des problemnahen Arbeitsplanes (s. Kap. 3.4).

Experimente in /9/ liefern vergleichbare Ergebnisse. Sie analysierten die Kommunikation zwischen einem Experten (Instruktor) und einer Person, die ohne visuellen Kontakt zum Instruktor und ohne produkt- bzw. montagespezifische Vorkenntnisse ein feinwerktechnischen Gerät (Drucker) montiert.

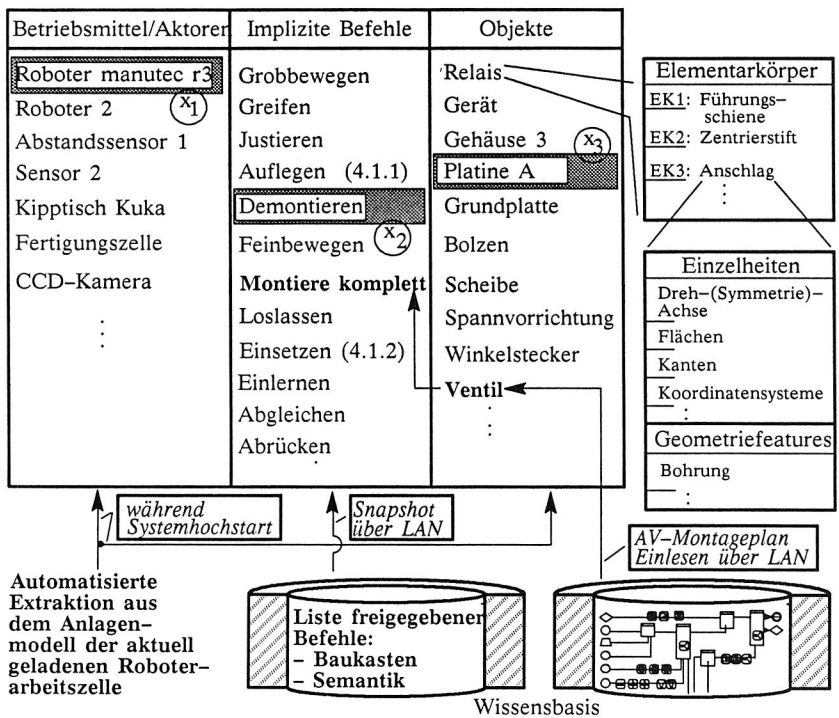


Abb. 3.5: Grundaufbau der Schnittstelle zur Bedienperson, Mensch-Maschine-Interaktion

Diese Konzeption der Dialogschnittstelle, mit einem begrenzten Umfang an Schlüsselwörtern (Aktor/Befehl/ Objekt), erleichtert es, optional eine Spracherkennung vorzuschalten. Das Fernziel ist eine Fertigungszelle (repräsentiert durch das Aktionsplanungswerkzeug) die, angenähert an die einfachste Kommunikation mit einer menschlichen Arbeitskraft, bedient werden kann.

Nach Abb. 2.3 müssen zwei Betriebsmodi unterstützt werden:

(I) Interaktives Arbeiten mit dem Aktionsplanungswerkzeug in der Werkstatt im Stil einer Arbeitsunterweisung:
- Es ist kein in der AV erstellter Montageplan vorhanden (sehr geringe oder sehr hohe Komplexität der Montageaufgabe oder keine AV vorhanden).
- Aktor, Befehl und Objekt werden interaktiv festgelegt (falls kein Aktor spezifiziert wird, wird wie unter (II) verfahren).
- Anwender ohne CAD/CAM-Verfahrenskette.

(II) Arbeitsplanbasiertes Abarbeiten eines vorab gespeicherten Arbeitsplanes (z.B. Montagegraph nach Kap. 3.4.3) einschl. Geometrieanalyse von CAD-Modellen:
- Nur gültig für ein spezielles Werkstück oder Baugruppe (vgl. Zeichnungslesen).
- Mittlere Komplexität der Montageprozesse.
- Lesen eines Arbeitsplanes in Graphenform aus der Datenbank.
- Analyse der CAD-Volumenmodelle über CAD-Kernel.
- Auswertung der während der Betriebsmittelauswahl mitprotokollierten Zuordnung zwischen Funktion und Aktor (DB).

Für (I) ist die graphische Kontrolle der Reaktionen der Roboterzelle auf die abgesetzten Fertigungsanweisungen wichtig. Facharbeiter und Techniker besitzen im allgemeinen ein besseres graphisches Vorstellungsvermögen gegenüber abstrakten, textuellen Steueranweisungen.

Der Aufbau der Grundoberfläche in Abb. 3.5 muß dynamisch erfolgen. Das Anlagenmodell der aktuell geladenen Roboterarbeitszelle muß dazu durchsucht und die Betriebsmittel- (1. Spalte von links) und Objektliste (3. Spalte) entsprechend aufgefüllt werden. Der aktuelle Vorrat an verfügbaren, impliziten Einzelbefehlen (2. Spalte) darf nur Befehle enthalten, deren zugehöriges Fügewissen in der Methodenbank vollständig vorhanden und freigegeben ist. Befehle ähnlicher Fertigungsprozesse (z.B. relevante Befehle für Klebungen) können gruppiert und in Graphenform angeboten werden. Diese Befehlsliste muß deshalb beim Systemhochstart, abhängig vom Füllstand der Wissensbasis, eingelesen werden. Sie wird durch SQL-Abfragen (ver-

teilter Datenbankzugriff, vgl. Kapitel 5.1) an die Datenbank gefüllt (Snapshot über LAN). Dadurch wird die leichte Erweiterbarkeit der Befehlsliste mit wachsendem Fügewissen sichergestellt bzw. die Anpassung an spezielle Fügetechnologien erleichtert. Gleiches gilt für die Liste der Werkstücke bzw. Baugruppen, deren Montagepläne bereits vorab (Voraussetzung!) in der AV bearbeitet wurden (Betriebsmodus (II); z.B. Komplettmontage von BG Ventil in Abb. 3.5). Der implizite Befehl 'Montiere komplett' nimmt dadurch eine Sonderstellung ein (s. Kap. 3.4).

Das Fehlen von Daten für die Zerlegung eines interaktiv eingegebenen, impliziten Befehls in Subaktionen hängt von den, zur Laufzeit aktuellen Zuständen des Anlagenmodells ab und ist a priori durch den Aktionsplaner nicht erkennbar. Fehlende Daten für die Bearbeitung eines impliziten Befehls sollten vom Benutzer durch Rückfragen (z.B. Rückfrage-Window) angefordert werden.

Befehle	**Selektiertes Objekt 1** (Körper X->Feature Y)		**Selektiertes Objekt 2** (Körper W->Feature Z)
Parallele Bewegung zur	Schiene S->Kante K1		- -
Senkrechtes Ausrichten von	Bauteil BT->Kante K1	zu	Schiene S->Kante K1
Paralleles Ausrichten von	Bauteil BT->Fläche F1	zu	Schiene S->Nut N1
Fluchtendes Ausrichten von	Bauteil BT->Achse D1	zu	Fassung F->Symm.achseA1
Bündiges Ausrichten von	Bauteil BT->Fläche F1	zu	Grundplatte G->Fläche F1
⋮			

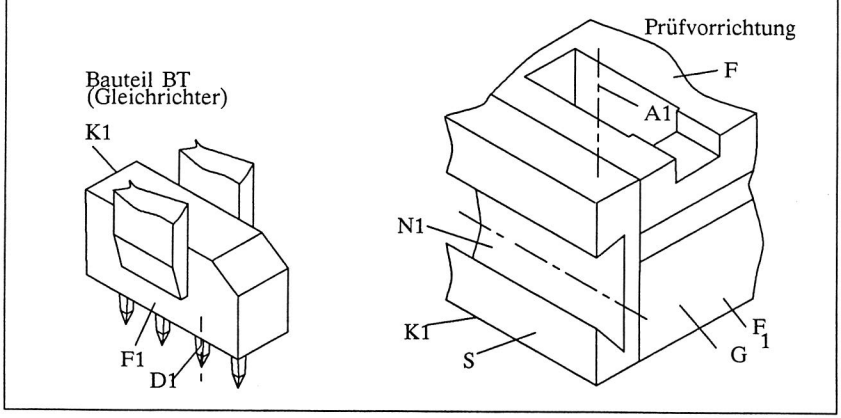


Abb. 3.6: Möglichkeiten geometrischer, bewegungsbezogener Befehle

Zur Präzisierung der produktnahen Befehlsspezifikation und zur Beantwortung von Rückfragen sollten Geometrieelemente (Flächen, Kanten, Achsen, Geometriefeatures) im Anlagenmodell namentlich wie auch direkt in der graphischen Darstellung selektierbar sein. Die Objektliste sollte dazu neben den Bauteilen und Einzelheiten von Körpern auch Baugruppen enthalten, die nicht einzeln bewegt werden können. Eine Darstellung der verdeckten Kanten ist hierzu durchaus hilfreich, da sich Kanten und Punkte als Bezugsgrößen und Orientierungshilfen für Positionierbefehle benutzen lassen. Wichtig für Roboterzellen ist die Definition von häufig in der Fertigungstechnik verwendeten Bewegungsaktionen, die sich auf Geometrieelemente beziehen (Abb. 3.6).

Ein Beispiel ist bündiges Ausrichten z.B. zweier ebener Flächen (Def.: a) beide Flächen sind in der gleichen Ebene und b) ihre Normalenvektoren sind kollinear). Schwierig ist es dabei entstehende Mehrdeutigkeiten (Freiheitsgrade) festzulegen, bzw. Überbestimmungen aufzulösen (vgl. /11/, auf der Basis von CSG-Modellen). Wichtig sind logische Verknüpfungen von Aktionen unter der Formulierung von Nebenbedingungen, z.B. Ausführen einer Aktion unter Aufrechterhaltung ("while") von Bedingungen (z.B. Kontaktkraft) oder bis Erreichen ("until") eines Sollzustandes (z.B. Zähler).

Rein textuelle Befehlseingabe in einer höheren Roboterzellensprache mit der ihr eigenen, dann natürlich komfortableren Syntax, stellt nur eine Problemverlagerung dar. Optimal ist eine quasi-natürlichsprachliche Oberfläche im Werkstattbereich (Ansatzweise in Abb. 3.5).

3.2.2 Aktionsplaner

Der Aktionsplaner (auch Arbeitsplaner oder Scheduler) übernimmt die Ableitung von Fertigungszellenaktionen und die zentrale Koordinierung der Ablauflogik in sog. Task-level-programmingsystems (vgl. /125/). Meist können zwei grundsätzliche Ansätze zur Aufgabenbeschreibung und Taskzerlegung unterschieden werden:

1. Spezifizierung der Aufgabenstellung mit Hilfe einer Folge von (geometrischen, physikalischen, kinematischen) Zielzuständen des Umweltmodells z.B. räumliche Relationen zwischen Objekten der Umwelt.

2. Spezifizierung der Aufgabenstellung durch eine Sequenz von Instruktionen.

Es existieren eine Reihe modellhafter, theoretischer Ansätze wie RAPT (**R**obot **A**utomatically **P**rogrammed **T**ools, /67/, /88/), AUTOPASS (**A**utomated **P**arts **A**ssembly

System, /65/), ATLAS (Automatic Task Level Assembly Syntheziser, /66/) und ASP-KAMRO /40/. Sie sind meist aus Gründen von Laufzeit und Effizienz (vor allem für WOP) nicht praxistauglich umsetzbar; wie z.B. die Einführung eines Konfliktlösers und Planprüfers /64/, der generierte Sensor- und Manipulationspläne auf ihre Durchführbarkeit hin überprüft und durch Modifikation sukzessive verbessert.

Für die aufgabenorientierte Aktionsplanung in Roboterarbeitszellen wurde deshalb im Rahmen dieser Arbeit ein kompakter, auf Werkstattebene anwendbarer Ansatz entwickelt.

Die zentrale Instanz, der Aktionsplaner (=Online-Arbeitsplaner), empfängt Montageaufgaben von der Benutzerschnittstelle. Er expandiert die höherwertigen Fertigungsbefehle mit Hilfe einer Wissensbasis (Methodenbank für Fügeprozesse) in eine Struktur (=Aktionsplan) von Subaktionen, bzw. überprüft deren Elementarheit. Dies wird so lange unter der Kontrolle des Aktionsplaners fortgeführt, bis nur noch Elementaraktionen (=Aktionsprimitive: AP) vorliegen, die nicht mehr feiner aufgelöst werden können. Aktionsprimitive sind direkt in Steuercode übersetzbare Aktionen (z.B. 'Bewegung ausführen'; Gegenbeispiel: 'Greifen').

Kernpunkt ist die Zerlegung eines impliziten Befehls in eine oder mehrere Subtasks, die von den Spezialisten (Sensoreinsatzplaner, etc.) bearbeitet und anschließend eventuell weiter aufgespalten werden. Jeder Spezialist kann eine Liste von Aufgaben bearbeiten (Methoden, vgl. Kap. 2.3), auf deren Lösung er alleine spezialisiert ist. Der Aktionsplaner synchronisiert, löst jedoch selbst keine Subtask (deshalb Metaplaner).

Wie ein Fertigungsprozess in einzelne Arbeitsschritte aufgespalten werden muß, ist in der Wissensbasis gespeichert. Die anlagenunabhängige Modellierung dieser Fertigungstechnologie sollte in der AV auf Workstationbasis erfolgen (im Rahmen dieser Arbeit beginnend mit einfachen Prozessen). Zur Aufspaltung jeder einzelnen Subtask muß das Wissen aus der Planungsebene auf Werkstattebene gelesen werden (aus Laufzeitgründen z.B. mit verteiltem Datenbank-Snapshot über LAN). Zusammen mit dem aktuellen Zustand des Anlagenmodells und event. Daten aus dem Echtzeitbereich der Gerätesteuerungen wird so eine Subtask bearbeitet.

Bereits das einfache, in nahezu allen Abläufen auftretende Greifen eines Bauteiles (weitere Beispiele: Auflegen auf Tisch, Führen gegen 2D-Anschlag, etc.) impliziert mehrere elementare Subtasks. Es sind:

0. Abschluß der Vorgängeraktionen

:

1. Einlesen der gesamten logischen Befehlsstruktur (=Aktionsgraph (s. Kap. 3.3), z.B. die hier angegebene Reihenfolge 2., 3., 4., ..) (aus Wissensbasis).
2. Überprüfung ob Greifwerkzeug bereits belegt. Falls ja: Ablegen dieses Bauteiles. (=Durchlauf eines alternativen Pfades durch den Aktionsgraph, der beispielsweise die Aktion 'Greifen' wieder rekursiv enthält).
3. Ermittlung der aktuellen Lage des zu greifenden Bauteils (aus akt. Anlagenmodell) und der Roboter–Ausgangsstellung.
4. Berechnung und Ausführung (Simulationsmodell !) kollisionsfreier Bewegungsbahnen zur schnellen Grobbewegung in die Nähe des Zielkörpers.
5. Reduzierung des Geschwindigkeitsoverrides.
6. Berechnung und Ausführung der Feinbewegung zur Greifposition.
7. Betätigung des Greifwerkzeuges (digitales Ausgangssignal setzen –>Pneumatikventil).
8. Berechnung und Bahnführung des Bauteils zur Trennung (='Abrücken') von benachbarten Körpern (Feinbewegung, jetzt mit gegriffenem Bauteil)
9. Geschwindigkeitsoverride wieder erhöhen.

:

10. Nachfolgeaktionen (Compilierung in Steuercode, DNC–Kommunikation, ...)

Dieses Wissen ist ein Beispiel für einfachstes, fertigungstechnisches 'Know–How'. Es definiert möglichst situations– und anlagenunabhängig den impliziten Befehl **'Greifen'** und muß formalisiert in der Wissensbasis (s. Kap. 5.2.5) abgelegt werden. Für die Instandhaltung dieses Basiswissens müssen die eigentlichen Werkzeuge und die Wissensbasis scharf getrennt werden (soweit möglich).

Die implizite Synthetisierung von Steuerprogrammen kann in drei Stufen eingeteilt werden (diese Einteilung ist sehr schwer zu treffen und kann aufgrund der hohen Komplexität und Flexibilität der Aktionsplanung nur als Orientierungshilfe dienen):
I. Das Basiswissen ist weitestgehend **anlagenunabhängig und Gerätetyp–unabhängig** (z.B. Fügewissen in der Methodenbank), d.h. z.B. unabhängig von der eingesetzten Roboterkinematik.
II. Konnte ein Aktionsplan (H–Graph, Kap. 3.3) erfolgreich (vollständig) expandiert werden, so ist er **anlagenabhängig, jedoch noch Betriebsmitteltyp–unabhängig.** Ein Beispiel: Der Test der elektrischen Werte eines Bauteils erfordert die fliegende Messung an einem raumfest installierten Sensor (S1) oder kann direkt durch Abgriff der

Werte an den Greiferbacken, online und parallel zur Bewegung (durch S2) erfolgen. Dies bedeutet, daß sich die Struktur des Arbeitsplanes nach dem konkret in der Zelle vorhandenen Sensor (S1 oder S2) richtet. Die Expandierung des Arbeitsplanes muß deshalb mit Hilfe der in der Konstruktion gefüllten Faktenbank über Geräte, Betriebsmittel und Montagezelle erfolgen (z.B. Greifer-Signalzuordnung).

III. Die Compilierung des Aktionsplanes ist **massiv anlagen- und gerätetypabhängig** (z.B. Grammatik der RC-Zielsprache SIRL oder SRCL, Ansteuerung der Bildverarbeitung, s. Kap. 4.1 und Kap. 5.2.1).

Für eine universelle Roboter-Aktionsplanung lassen sich folgende Definitionen aus der Analyse von Montageprozessen treffen (Abb. 3.7):

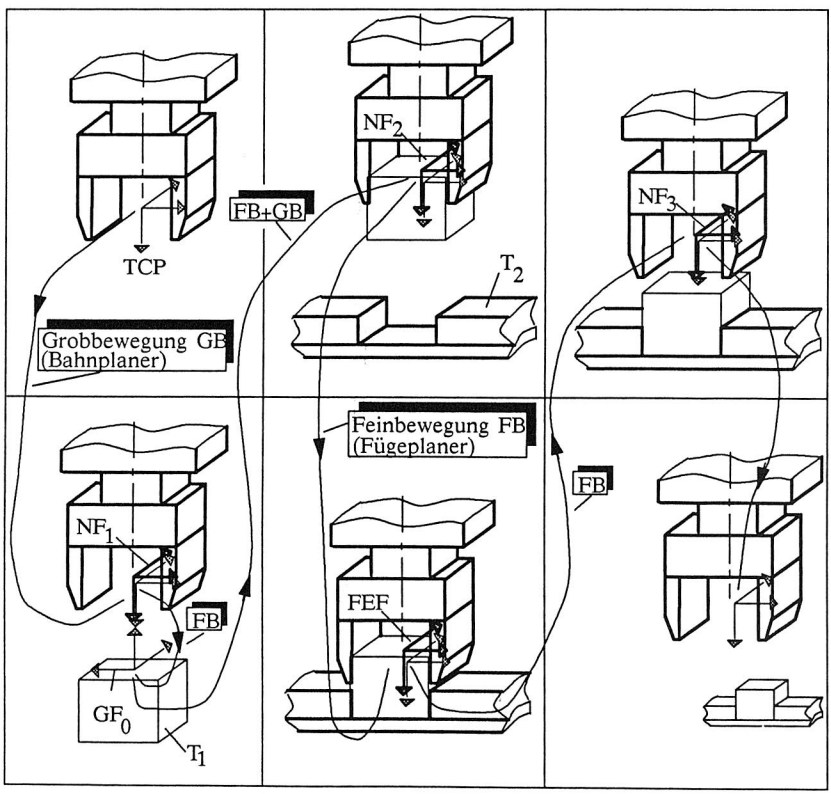


Abb. 3.7: Definitionen zur Roboteraktionsplanung (Beispiel ohne Orientierungsänderungen)

Mit: NF_1: Nahpunkt-  des aktiven Fügepartners T_1
NF_2: Nahpunkt-  des passiven Fügepartners T_2 (abh. v. T_1 + T_2-Geometrie)
FEF: Fügeend-
NF_3: Abrück-  von teilmontierter Baugruppe
GF : Greif-

Ein **Raumpunkt** (oder kurz Punkt (x,y,z,a,b,c)T bzw. frame ) repräsentiert eine Transformationsmatrix, mit der Ursprung (x,y,z) und Orientierung (z.B. um die Behandlung singulärer Stellen erweiterte Euler-Winkel a,b,c (s. Kap. 3.1.2)) eines Koordinatensystems relativ zu einem Bezugskoordinatensystem eindeutig festgelegt sind.

Ein **Greifframe (GF) (bzw. Greifpunkt)** ist ein körperfest mit dem zu montierenden Bauteil verbundenes Koordinatensystem (aus dem CAD-Modell berechnet oder konstruktiv angeheftet).

Das **Nahpunktframe (NF)** ist das mit dem passiven Fügepartner bzw. mit dem zu greifenden Bauteil körperfest verbundene (Hilfs-) Koordinatensystem, an dem der Übergang ("Umschaltpunkt") zwischen Roboter-Grob- und Feinbewegung stattfindet.

Nahframe = Trennstelle zwischen Grob- und Feinbewegung.

Dieser Annäherungspunkt ist nur für einen konkreten, aktiven Fügepartner und Griffsituation gültig und muß online aus dem aktuellen Zustand des CAD-Anlagenmodells heraus errechnet werden (abh. von Auskragungen des gegriffenen Teils etc.). Beim Greifen ist für die Berechnung des Nahframes das Greifframe ein wichtiger Input; im Falle einer Fügebewegung das Fügeendframe.

Das **Fügeendframe (FEF)** ist ein mit dem passiven Fügepartner körperfest verbundenes Koordinatensystem zur Beschreibung der Zielposition und -orientierung eines mit dem aktiven Fügepartner verbundenen Fügebezugssystems. Das Fügebezugssystem kann beliebig gewählt werden, nach der Definition muß es jedoch beibehalten werden.

Der **Tool-Center-Point (TCP)** ist ein mit dem Roboter/Greifer körperfest verbundenes Bezugssystem auf das sich die Vorwärts- und Rückwärtstransformationen beziehen. Anfahren eines NFs, GFs oder FEFs bedeutet, daß TCP und NF, bzw. GF und FEF zur Deckung gebracht werden.

Im weiteren wird zwischen **aktivem** und **passivem Fügepartner** unterschieden. Der aktive Fügepartner führt die Hauptfügebewegungen aus, während der passive Fügepartner seine absolute Position und Orientierung im Raum beibehält (derzeit einarmige IR für die meisten Montageoperationen), oder z.B. bei kooperierenden In-

dustrierobotern (vgl. /113/) passiv bewegt wird. Im ersten Spezialfall werden die Transformationen entsprechend einfacher, da die mit dem passiven Fügepartner verbundenen Frames zugleich raumfest werden.
In der impliziten Aktionsplanung kann der Greifvorgang ebenfalls als Fügevorgang betrachtet werden (Unterschied: Der aktive Fügepartner des Greifvorganges ist der Greifer selbst).

Die gesteuerte Bewegung eines Industrieroboters kann in die beiden grundsätzlichen Bewegungsanteile Grob- und Feinbewegung unterteilt werden (Abb. 3.8).

Feinbewegung (Feinmotorik)	**Grobbewegung**
- gewünschte Kollision (Kontakt bzw. "Beinahe-Kontakt")	- keine Kollision
- geringe Bewegungsgeschwindigkeiten	- hohe Bewegungsgeschwindigkeiten
- geringe Beschleunigung	- hohe Beschleunigungswerte
- hohe Genauigkeitsanforderungen (z.B. LIN)	- geringe Genauigkeitsanforderungen (z.B. PTP)
- eventuell Sensoreinsatz (taktile Sensoren)	- im allg. keine Sensorik

Abb. 3.8: Kennzeichen von Grob- und Feinbewegung

Die Grobbewegung wird demnach als Bewegung zwischen Nahframes, bzw. Stützframes aus der Bahnplanung, definiert. Die Feinbewegung ist die Bahnführung zwischen Nahframe und Greifframe, bzw. Fügeendframe. Eine spezielle Art der Feinbewegung ist "Abrücken". Diese implizite Befehl bedeutet eine "vorsichtige" Trennung eines aktiven Fügeteils, bzw. des leeren Greifers, von seinen umgebenden Körpern und ist Grundlage der impliziten Demontage.

Der aktuelle Zustand des Umweltmodells muß für jeden impliziten Befehl zur Laufzeit der Aktionsplanung berücksichtigt werden. Zwei unterschiedliche Ausgangskonfigurationen lösen deshalb bei gleicher Anweisung unterschiedliche Bewegungen bzw., Reaktionen aus. (Abb. 3.9 zeigt das adaptive Verhalten an einem Beispiel).

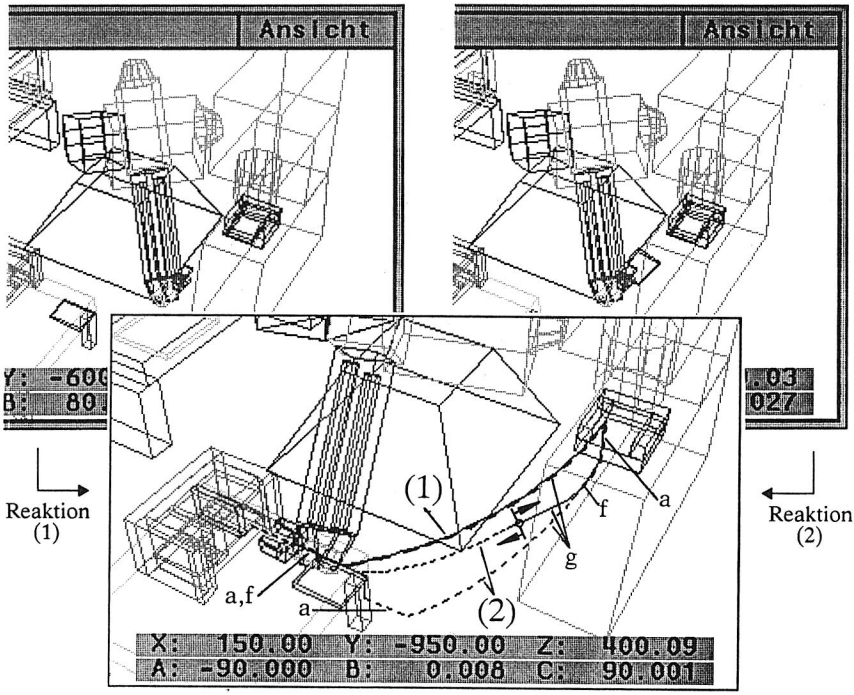


Abb. 3.9: Reaktionen auf den impliziten Befehl "Einsetzen Platine B in Baugruppenträger" (Bildschirmabzug, TCP-Verfahrbahn mitgezeichnet, Online berechnete Trajektorien: f: Feinbewegung (berechnet von Fügeplaner), g: Grobbewegung (Bahnplaner), a: Abrückbewegung (Fügeplaner))

Abb. 3.9 zeigt die unterschiedlichen Reaktionen auf eine vollständig identisch an das System übergebene, implizite Instruktion; je nach Ausgangszustand des Umweltmodells zum Zeitpunkt der Expandierung des impliziten Befehls (Reaktion (1): IR-Greifwerkzeug ist geöffnet und frei, (2): Platine B befindet sich durch vorausgegangene Aktionen im Greifwerkzeug).

Alle Aktionen müssen unter der permanenten Kontrolle eines Common-Sense-Wächters ausgeführt bzw. synthetisiert werden. Abb. 3.10 zeigt den Ablauf am

Beispiel einer Aktion, die die Elementaraktionen "Grobbewegung" und "Feinbewegung" enthält. Im realisierten Prototypen gibt der Aktionsplaner den gerade aktiven Spezialisten (+ dessen Methoden) nach außen dem Bediener in einer Statuszeile an. Nach Bearbeitung eines impliziten Befehls meldet sich das System *"bereit für weitere Aktionen !"*.

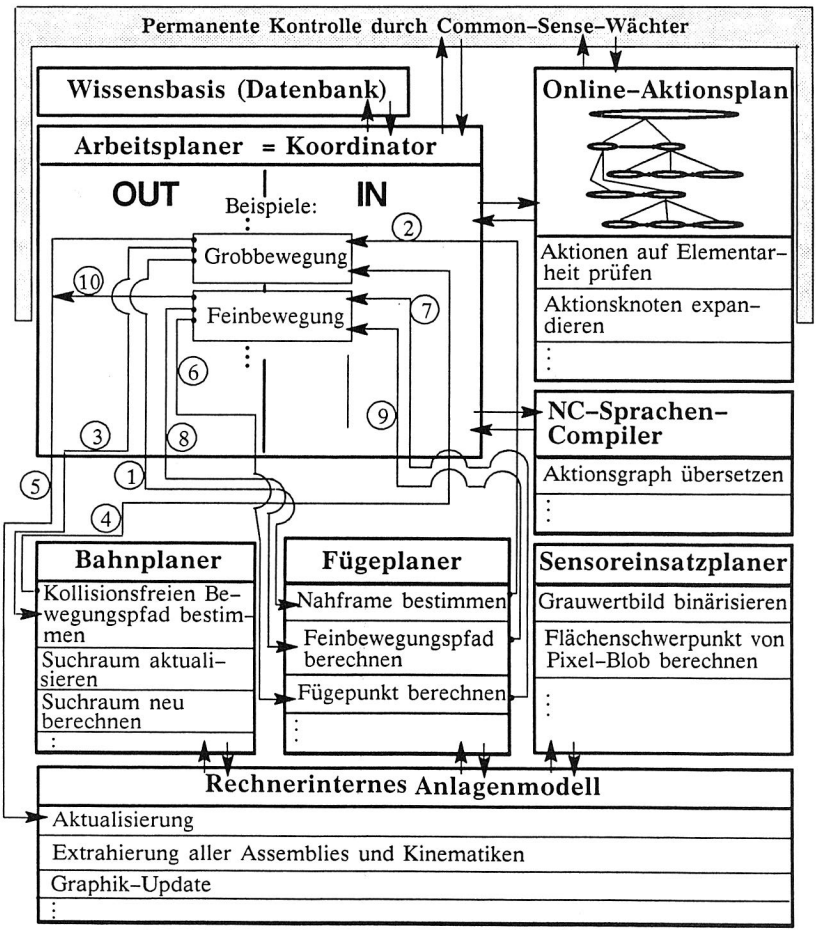


Abb. 3.10: Nachrichtenaustausch zwischen Aktionsplaner und Spezialisten

Die übertragenen Nachrichten sind für dieses Beispiel nachfolgend aufgelistet (die Numerierung bezieht sich auf die Aktivierungsreihenfolge):

① Name des zu greifenden Bauteils = z.B. Platine A
② Nahframe von Platine A
③ (Istposition des IR) + (Zielframe = Nahframe Platine A)
④ Liste von Frames des kollisionsfreien Pfades: (x1, y1, z1, a1, b1, c1)T,
(x2, y2, z2, a2, b2, c2)T, ... ,
⑤ Aktualisierung des Anlagenmodells (+ einschl. Bildschirmgraphik)
⑥ Name des passiven Fügepartners
⑦ Fügeendframe
⑧ Anstoß Berechnung Feinbewegungspfad
⑨ Frameliste der Feinbewegung
⑩ Aktualisierung des Anlagenmodells
usw. :
ⓝ Übersetzen des Aktionsplanes in Zielcode für die Gerätesteuerungen

Dieses Wissen (welcher Spezialist, welche Methode mit zugehörigem Input und Output (Zwischenergebnissen), ist in welcher Sequenz anzusprechen) ist unter anderem Inhalt der Methodenbank (Kap. 5.2.5). Bei Komplettmontagebefehlen muß der Aktionsplaner zunächst AV-Arbeitsplan (DB) und CAD-Daten über LAN anfordern und die Montagepläne anschließend weiterverarbeiten (Betriebsart (II) S. 43).

3.2.3 Bahnplaner

Die Bahnplanung stellt ein algorithmisches Problem dar (sog. Path-Find-/Mover's Problem). Der Bahnplaner muß die Bahnberechnung über weite Strecken unter Beachtung der Kollisionsproblematik übernehmen, mit:

Input:
- Startzustand des Anlagenmodells (z.B. aktuelle IR-Position)
- Zielpunkt : z.B. berechneter Nahframe (Kap. 3.2.2)
- Optimierungskriterien (z.B. Zeit, Energie), Nebenbedingungen (z.B. $\perp$ - Lage des TCPs für die Bewegung eines mit Flüssigkeit gefüllten Behälters)

Output:
- Kollisionsfreie Bahn zwischen Start- und Zielpunkt unter Berücksichtigung gegriffener Werkstücke.

Das Hauptproblem der Grobbewegungsplanung ist die Verhinderung einer Kollision zwischen Teilen der Fertigungszelle, Effektoren, Greifwerkzeugen, Werkstücken und NC-Achskörper (sog. Linked Bodies Mover's Problem).

Im Gegensatz zur Feinbewegung (vgl. Kap. 7.1) ist die Bahnplanung relativ unempfindlich gegenüber geringfügigen geometrischen Abweichungen (Position und Orientierung). Die Grobbewegungs-Trajektorie muß bereits auf Grund von dynamischen Effekten des mechanischen Systems (z.B. Trägheitseinflüsse, Corioliseinflüsse, Überschwingen) bzw. den Steuerungsalgorithmen (Überschleifen, Bahntreue bei unterschiedlichem Geschwindigkeitsoverride) einen Sicherheitsabstand zu Körpern einhalten.

Zu dem Bereich der Bahnplanung existieren umfangreiche theoretische Vorarbeiten (s.u.). Die Ansätze müssen aufgrund von Laufzeitproblemen bei der Realisierung eines prozeßnahen Aktionsplanungswerkzeuges häufig vereinfacht werden.

Die Grobbewegungen sollten zur Taktzeitminimierung möglichst im PTP-Modus der Steuerung auszuführen sein; anstatt z.B. mit 'langsamer' Linear-Interpolation. Dies widerspricht jedoch teilweise der Intention der Bahnplanung, da PTP-Bewegungen abhängig vom Arbeitspunkt, und somit nicht allgemein vorausbestimmbar sind. Bahnplanungsalgorithmen setzen ein definiertes Abfahren der berechneten, kollisionsfreien Trajektorien voraus, bzw. müssen mit Hilfe der RC-Simulation abprüfen, daß sich die berechneten Bahnen auch tatsächlich innerhalb eines zulässigen 'Toleranzschlauches' befinden. Durch Steuerungsalgorithmen neuer RC-Generationen und deren 1:1 Portierung in die Simulation (s. Kap. 3.1.3) läßt sich diese Voraussetzung erfüllen. Die errechneten Trajektorien-Stützpunkte und vor allem die PTP-Bahnen zwischen diesen Stützpunkten müssen, mit hoher Bahntreue, d.h. unabhängig von der programmierten Bahngeschwindigkeit (Override), eingehalten werden.

Exakte Nachrechnung geplanter Bahnen, z.B. für Bahnabschnitte bei denen nur ein geringer Sicherheitsabstand eingehalten werden kann, oder bei kraftschlüssigen Bahnen, muß die Simulation das Mehrkörpersystem einschl. Regelung umfassen (vgl. /82/, /112/).

Für ein Aktionsplanungswerkzeug, und hier besonders auf Werkstattebene, ist die benötigte Rechenzeit der Bahnplanungsalgorithmen von besonderer Bedeutung. Bahnplanung bedingt aufwendigen Aufbau und Nachziehen von Suchräumen, komplexe Suchverfahren, große Datenbereiche (z.B. ca. 1 MB für Suchraum einer stark vereinfacht modellierten FFZ, s.u.). Die meisten dieser Teilschritte müssen zur

Laufzeit gerechnet werden (Abb. 3.11). Für die Konzeption und Realisierung eines Bahnplaners, muß deshalb besonderer Wert auf Laufzeitoptimierungen gelegt werden (z.B. C-Zeigerstrukturen für dynamische Geflechte mit mehrfach verketteten Listen).

Zur Vereinfachung der eigentlichen Bahnplanungsverfahren wird häufig der Arbeitsraum in einen n-dimensionalen Konfigurationsraum (**K-Raum** oder Synonym: Configuration **Space**, **C-Space**) transformiert, in dem z.B. IR und Effektor auf einen Punkt reduziert werden können /66/, /51/, /52/. Die Körper der FZ werden entsprechend zu n-dimensionalen Konfigurationsraumhindernissen (Configuration Space Obstacle, **CSO**) expandiert. Die eigentliche Bahnplanung findet dann in diesem K-Raum statt; das Ergebnis wird am Ende wieder zurücktransformiert. Das komplexe Problem der Bahnplanung wird damit teilweise in die Berechnung der K-Raum-Hindernisse verlagert (in /51/ beispielsweise konvexe Hüllpolyeder).

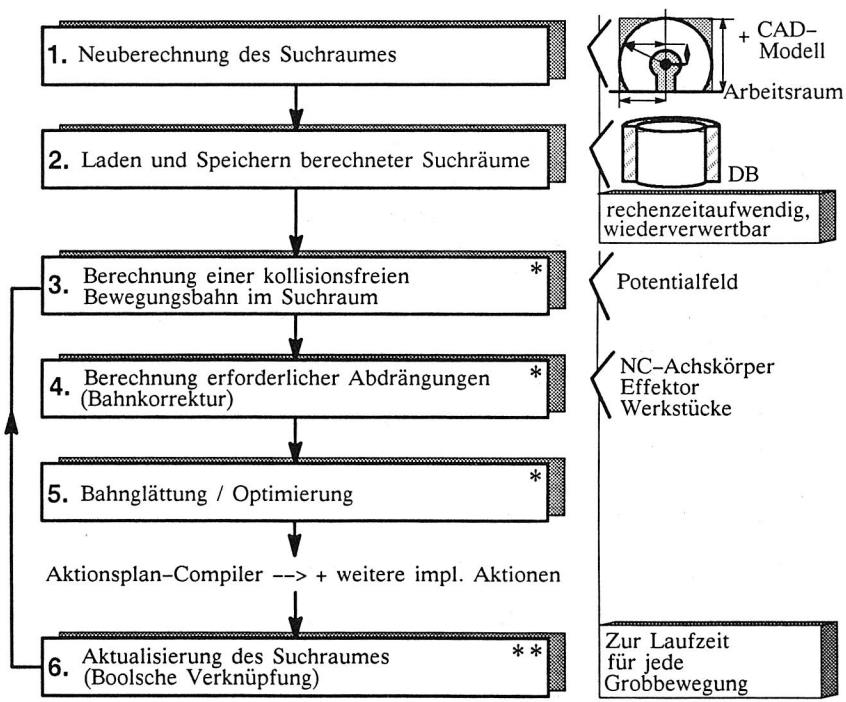


Abb. 3.11: Schritte der Bahnplanung (: ausschaltbar, **: nicht ausschaltbar)*

Die Dimension des K-Raumes kann sehr hoch werden, da die kinematischen Mehrdeutigkeiten von HHS (z.B. Überkopfstellung) zusätzliche Dimensionalität (weitere Koordinatenachsen) des K-Raumes erfordern (neben den z.B. 6 IR-Freiheitsgraden).

In /24/ wird der Suchraum analog zur Konvertierung in ein CSG-CAD-Modell aufbereitet ('Hierarchisches Approximationsmodell'). Das Ergebnis ist eine Baumstruktur aus Volumenprimitiven (z.B. Octrees). In /52/ wird die Wegsuche in diesem Suchraum mit einem hierarchisch-rekursiven Zerteilungsalgorithmus (A*-Klasse) auf ein Graphendurchlaufproblem reduziert.

Stand der Technik bei kommerziellen Systemen (vgl. UNIGRAPHICS II Simulation, Mc Donnel Douglas Corp., MDC) ist die Offline-Kollisions**erkennung** (Ja/Nein-Aussage). In /110/ werden hierfür, anhand konkreter Bahnvorgaben Kollisionstests durchgeführt. Die Berechnung erfolgt mit ansteigenden Genauigkeitsstufen, beginnend mit einfachen Hüllkörpern. Die Bahnplanung kann nach Abb. 3.11 in folgenden Schritten erfolgen:

In Schritt 1 wird zunächst der leere Suchraum angelegt. Die Auflösung der über die FFZ gelegten Gitterstruktur (vgl. Netzgeneratoren zur FEM-Diskretisierung), d.h. die Elementarzellenabmessungen, kann automatisch an die FFZ, bzw. an deren CAD-Modell, angepaßt werden (z.B. feine Rasterung für die Montage elektronischer Bauteile, grob für große Blechteile). Anschließend wird das CAD-Modell der FFZ in den K-Raum transformiert (alle statischen, als auch beweglichen Körper). Input ist das CAD-Modell. Eine Möglichkeit einer laufzeitoptimierten Transformation wird in /2/ vorgestellt (OCMEM-Verfahren, Obstacle-Coding-Memory). Der K-Raum wird dann über die Gitterstruktur aus Elementarzellen weiter in den Suchraum transformiert, wobei die Suchraumgrenzen zu den Arbeitsraumgrenzen des IR äquivalent sind. Die kinematischen Restriktionen des Industrieroboters (Achs-Endanschläge) schränken den Suchraum weiter ein.

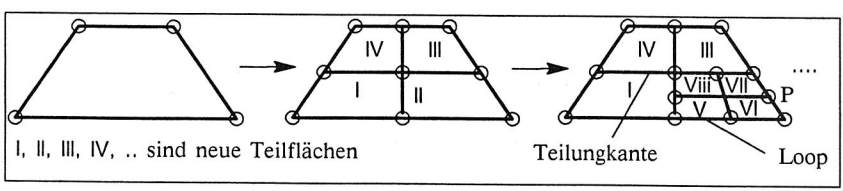


Abb. 3.12: Adaptive Rasterung von Begrenzungsflächen zur K-Raum-Transformation

Nach /2/ eignet sich für IR mit rotationssymmetrischem Arbeitsraum (z.B. manutec r2 mit annähernder Symmetrie um NC-Achse 1) die Einführung von Zylinderkoordinaten für den Suchraum. Eine Suchraumzelle wird dann mit (r,ϕ,h) adressiert.

Eine Elementarzelle des Suchraumes kann nur leer oder gesperrt sein. Sie kann jedoch mehrfach belegt sein, wie z.B. an den Kontaktflächen von Körpern. Sie gilt bereits als gesperrt, wenn auch nur ein Teilvolumen der Elementarzelle ein expandiertes Konfigurationsraumhindernis schneidet. Gesperrte Elementarzellen können vom IR nicht kollisionsfrei erreicht werden, wobei gesperrte Elementarzellen mit '**virtueller CSO-Materie**' gefüllt sind (kurz: 'mit Materie gefüllt', entspricht jedoch **nicht** realen 3D-Körper-Volumina !).

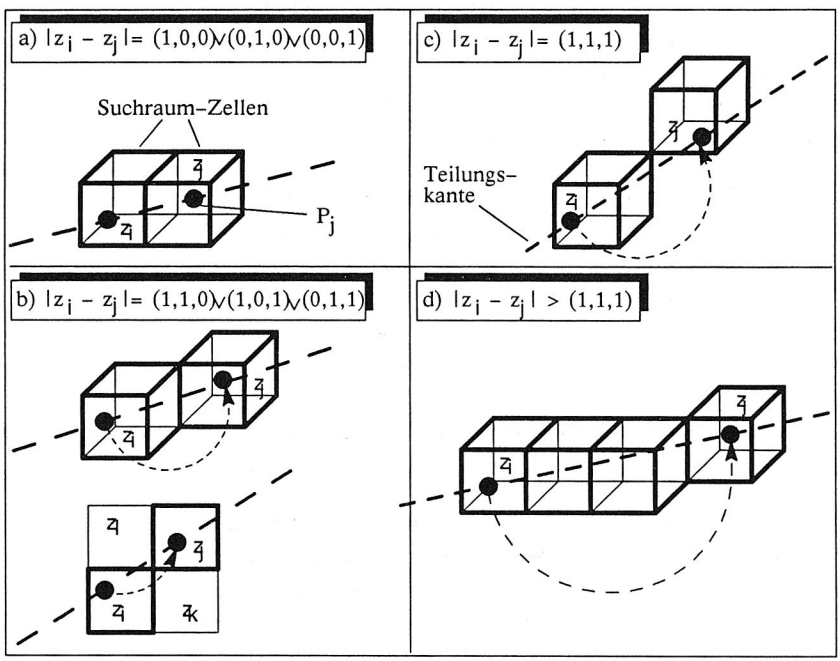


Abb. 3.13: Topologische 'Benachbart-Beziehung' zur Markierung von Elementarzellen: a) Direkt benachbart, b) Diagonal-benachbart (zur Sicherheit werden alle 4 Zellen i, .., l markiert), c) Über Eck benachbart, d) Nicht benachbart, d.h. feinere Rasterung notwendig

Für das gezielte Löschen und Neuanlegen von positionsveränderlichen Körpern im Suchraum ist eine eindeutige Zuordnung zwischen den durch diesen Körper belegten Elementarzellen und dem CAD-Modell dieses Körpers erforderlich (in C realisierbar über mehrfach verkettete Listenstrukturen).

Im Rahmen dieser Arbeit wurde das einfache CAD-Drahtmodell mit Hilfe einer rekursiven, adaptiven Rasterung der Körper-Begrenzungsflächen in den K-Raum transformiert. Die Rasterung, und damit der Rechenaufwand ist an die Abmaße des Körpers angepaßt. Das Verfahren approximiert dabei Körper durch die sie vollständig umhüllenden Elementarzellen. Anschaulich: Die CSO-Körper sind "innen hohl".

Eine ebene Begrenzungsfläche von CAD-Körpern wird sukzessive feiner gerastert (Abb. 3.12). Die Rasterung und die Markierung der betroffenen Elementarzellen im Suchraum erfolgt abhängig von den ermittelten topologischen Beziehungen (Abb. 3.13) zwischen den Elementarzellen. Sind die markierten Elementarzellen benachbart (normierter Abstand $(\delta r, \delta\phi, \delta h) \subseteq (1,1,1)$), bringt eine feinere Rasterung dieser Fläche keinen weitere Vorteil. In der Praxis ist dies schnell erreicht (z.B. sehr schnell bei der Oberflächenmontage elektronischer Bauelemente mit kleinen Abmessungen).

Punkt 1 fällt prinzipiell für jede Aktualisierung des Umweltmodells erneut an. Da die **Axiome der Addition** bezüglich des Suchraumes gelten, können raumfeste Zellenkomponenten vorab gerechnet und anschließend, nach Abschluß einer impliziten Subaktion, nur noch die bewegten Teile aufaddiert (neue Position) bzw. subtrahiert (alte Position) werden. Es werden nur die Komponenten neu berechnet, die auch von der Änderung betroffen sind. Die zeitintensive Neuberechnung des Suchraumes kann zum Zeitpunkt des Hochstartes des gesamten Systems ausgeführt werden (Vorverarbeitungsschritte in der Initialisierungsphase des Bahnplaners). Der Großteil der FFZ-Komponenten ist statisch und "belastet" damit den Bahnplanungsalgorithmus nicht.

Derart vorverarbeitete Suchräume können in der Datenbank abgelegt werden und zusammen mit dem Umweltmodell beim Systemstart geladen werden (Schritt 2 in Abb. 3.11), so daß nur in dem Fall, daß kein vorab berechneter Suchraum gefunden wird, Schritt 1 erforderlich wird.

Für nachfolgenden Schritt 3 der Online-Bahngenerierung zeigten, aufgrund ihres Laufzeitverhaltens und ausreichender Genauigkeit, die **Entfernungs-Potentialfeldmethoden** /2/,/63/ gute Ergebnisse. Auf diesen Methoden basiert die Realisierung des Prototypen im Rahmen der vorliegenden Arbeit.

Hierbei wird jeder Elementarzelle (sofern nicht bereits als Hindernis oder Arbeitsraumgrenze markiert) ein **skalarer Potentialwert** angeheftet, der die minimale Entfernung zum Zielframe (Senke !) angibt (Abb. 3.14). Die Berechnung des Potentialfeldes startet am Zielframe. Die Zelle, in der sich die Senke befindet, hat den Potentialwert "0". Der Ausgangspunkt der Bewegung repräsentiert die Quelle.

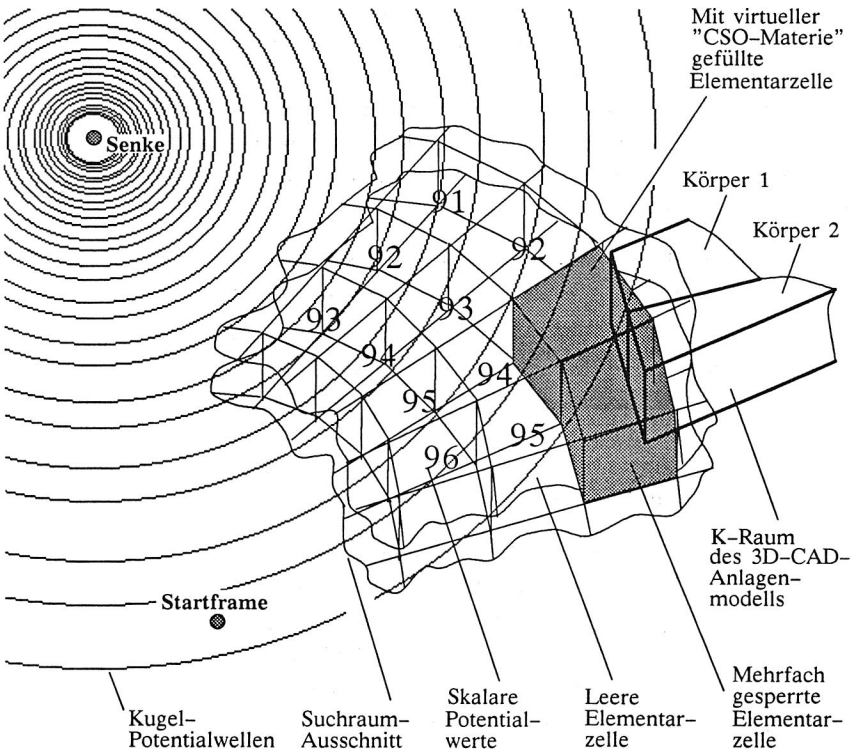


Abb. 3.14: Segmentierung des Suchraumes in ein Potentialfeld

Zur Erfassung des gesamten Suchraumes lassen sich vom Zielpunkt ausgehend Potentialwellen auslösen, die sich im Suchraum ausbreiten und alle Hindernisse umfließen

(z.B. NORMPOT-Algorithmus nach /2/). Die Wegsuche erfolgt anschließend mit Gradientenverfahren im dreidimensionalen Potentialfeld.

Der Nachteil der hohen Rechenkosten dieses Verfahrens kann durch Kombination mit schnellen, vereinfachten Algorithmen gemindert werden. Sie arbeiten mit einem lokal begrenzten Potentialfeldaufbau und Durchstoßversuchen (z.B. zuerst in ϕ-Richtung) in bestimmten Vorzugsrichtungen (vgl. MINPOT-Algorithmus nach /2/).

Die so gefundene Bewegungsbahn (z.B. der Potentialwert-Pfad in Abb. 3.14: ..-91-92-93-94-95-96-..) dient als Vorgabe für die Berechnung der erforderlichen Abdrängungen (Schritt 4 in Abb. 3.11) der endgültigen Bahn, die aus der realen Geometrie von z.B. Effektor und Last (Rückkehr zum Umweltmodell !) entstehen. Korrekturen werden aus der ideal geplanten Bahn und z.B. aus den Flächennormalenvektoren (größter Trennungseffekt) der Flächen der nicht bewegten Körper, für die event. Kollisionsgefahr besteht, berechnet.

Experimente mit Bahnplanungsalgorithmen zeigten, daß reine algorithmische Lösungen alleine nicht befriedigend arbeiten, bzw. der Rechenaufwand sehr hoch wird. Durch Kombination mit wissensbasierten Methoden (vgl. Common-Sense-Wächter Kap. 3.2.5) können beispielsweise symbolische Topologiedaten für das Umfahren von Hindernissen, vor allem für Schritt 3 und 4, eingebunden werden (z.B. Standardverhalten bei Konfigurationen mit topologischen Attributen 'konkave' bzw. 'konvexe Kante'). Nur wenn diese Standard-Strategien nicht zum Ziel führen, können algorithmische Verfahren nachgeschaltet werden.

Zuletzt können die vom Bahnplaner berechneten, event. stark unstetigen (treppenförmig, vgl. /52/) Bewegungsbahnen geglättet werden. Glättungsfilter können redundante Stützpunkte löschen oder Stützframes geringfügig ausrichten (Bedingung: keine neuen Kollisionen). Zusammen mit der RC-Überschleiffunktion des Aktionsplan-Compilers lassen sich so 'ruckfreie' Bewegungsbahnen und kompaktere RC-Programme erzeugen (Schritt 5 in Abb. 3.11).

Bei der exemplarischen Realisierung des Bahnplaners wurden folgende Voraussetzungen getroffen:

◆ Das 3D-CAD-Anlagenmodell sei vollständig und korrekt (vor allem Übereinstimmung mit der realen Anlage !, Maßnahmen s. Kap. 4.2.2). Einfaches Drahtmodell.
◆ Alle Objekte der FFZ, mit Ausnahme der gerade aktiven Kinematik, sind statisch in Position, Orientierung und Gestalt.

◆ Zeitgleich bewegte Körper (z.B. auf Förderband) oder kooperierende IR werden nicht betrachtet.

◆ Es wurden nicht alle NC-Achskörper der kinematischen Kette des IR in die Bahnplanung mit einbezogen. Für Experimente in der Testzelle (Kap. 4.3.2) wurden nur die 5. und 6. Achse des IR (manutec r2), sowie Greifwerkzeug und gegriffene Werkstücke betrachtet.

◆ Auflösung der Elementarzellengitter z.B. zu ca. (dr,dϕ,dh) = (50 mm, 2 Grad, 70 mm) gewählt; resultierender Suchraum mit ca. 140 000 Elementarzellen.

◆ Vereinfachte Transformation der CAD-Modelle in den Suchraum (s.o.).

◆ Versuchsaufbau, Fertigungszelle und Hardware nach Kap. 4.3.2.

◆ Realisierung in C mit dynamischer Speicherplatzverwaltung . In der Praxis sind bei FFZ meist nur max. 40% des Suchraumes mit Hindernissen belegt.

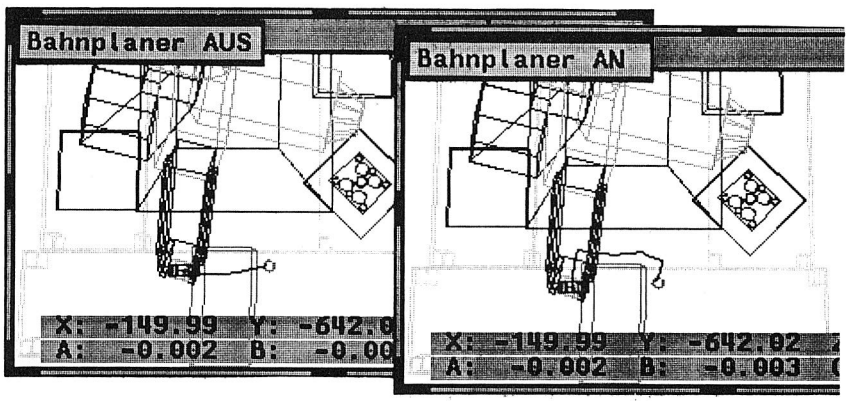


Abb. 3.15: Aktivierter und deaktivierter Bahnplaner (Draufsicht, Bildschirmkopie)

Ermittelte Versuchsergebnisse für Beispiel in Abb. 3.15 (Hardware nach Kap. 4.3.2):

Neuberechnung des Suchraumes der einfach modellierten FFZ (Schritte 1 und 2 nach Abb. 3.11):	CPU-Zeit: 38 s
Bahnplanung für die Grobbewegung (Schritte 3 bis 6):	5 s
Durchlauf des Aktionsplan-Compilers:	4 s

Die Experimente zeigten, daß einfache Geometriemodelle (3D-Drahtmodelle, geometrische Grundkörper) der Fertigungszelle für die Bahnplanung nicht nur ausreichend, sondern darüberhinaus sehr gut geeignet sind (ohnehin erforderlicher Sicherheitsabstand + Laufzeit).

Der Bahnplaner kann vom Benutzer jederzeit ein- und ausgeschaltet werden (Abb. 3.15). Ist er aktiviert, werden für alle, durch Aktionen event. implizierte Grobbewegungen automatisch Bahnen generiert. Ausschalten heißt jedoch nur, daß die Schritte 3 bis 4 in Abb. 3.11 deaktiviert werden können. Da der Bahnplaner möglicherweise wieder zugeschaltet wird, muß der Suchraum zum Umweltmodell stets einen konsistenten Zustand einnehmen. D.h. die Aktualisierung des Suchraumes (Schritt 6, Abb. 3.11) muß auch bei ausgeschalteter Bahnplanung mitlaufen.

Zusätzlich zur Online-Berechnung sollten an jedem einzelnen Objekt feste, relative (bezogen auf lokales Koordinatensystem) Grobpfade konstruktiv angebunden werden können. Der Bahnplaner stößt die Online-Berechnung nur an, falls in der Datenbasis kein Pfad gefunden wird.

Der Spezialist 'Bahnplaner' wurde als eigenständiger Unix-Prozeß realisiert. Er muß in der Lage sein, nur durch Angabe eines TCP-Zielframes selbständig eine kollisionsfreie Bewegungsbahn für den Roboter in der Fertigungszelle zu berechnen. Alle weiteren Daten werden intern dem Geometriemodell entnommen. Der Bahnplaner kommuniziert ansonsten ausschließlich mit dem Common-Sense-Wächter (Kap. 3.2.5), der über alle Änderungen des Umweltmodells wacht, und mit dem Arbeitsplaner, der von ihm kollisionsfreie Bahnen anfordert.

3.2.4 Greifplaner

Ein Greifplaner bestimmt online die Menge möglicher, körperfester Greifframes an einem zu greifenden Bauteil anhand dessen CAD-Volumenmodell. Folgende Punkte sind hierbei zu beachten:

- Die Greif- und Einbausituation.
- Alle nachfolgenden (Füge)-Situationen, solange die Greifsituation beibehalten wird ("Vorausschauende Planung").
- Geometrie des aktuell am Roboterarm angeflanschten Greifers (Abmessungen, Greifprinzip (z.B. Zweibackenparallelgreifer)).
- Robustheit des Griffes (Optimierung der Kontaktkräfte- u. momente /15/).

Neben diesen Online-Berechnungen muß der Greifplaner aus einer, einem Objekt zugeordneten Menge möglicher Greifframes, ein optimales Frame auswählen bzw. event. modifizieren (z.B. den Greifüberdeckungsgrad, s.u.). Aufgrund der Kombinatorik muß dies jedoch wieder zur Laufzeit erfolgen. Der aktivierte Greifplaner muß im Zuge der Auflösung von Fertigungsbefehlen mit implizierten Greifoperationen das Anlagenmodell durchsuchen. Die Framemenge kann z.B. über den Namen des zu greifenden Bauteils identifiziert werden.

In der Konstruktion können ein oder mehrere Greifframes (Greifframe-Schar) an die Bauteile körperfest angeheftet werden (einschl. Fangradien). Zusätzlich können im Umweltmodell dem Greifframe Werte über Greifkraft, Greifbackenöffnung und Anfahrrichtung mitgegeben werden. Frei programmierbare Greifkraft und Greiferbackenöffnung setzten jedoch eine weitere NC-Achse im Greifer voraus.

Beispielsweise werden in /43/ speziell für Sauggreifsysteme mit Hilfe wissensbasierter Methoden die Aufsetzpunkte von Saugköpfen aus dem CAD-Modell (B-Rep) ermittelt. Damit ist eine diskrete Anzahl an Greifsituationen festgelegt. Zu den Grundlagen der Greiftechnik vgl. z.B. /19/ und Übersicht in /43/.

Für das Zusammenspiel von Greifplaner $\Leftrightarrow$ Fügeplaner werden zunächst folgende Größen definiert (s. Abb. 3.16):

$T_{N,G}$: Transformationsmatrix von Nahpunktframe N zu Greifframe G
$T_{Z,G}$: Transformationsmatrix von Zielframe Z zu Greifframe G
$T_{N,TCP,I}$: Transformationsmatrix von TCP im Nahpunkt bezüglich Inertialsystem I
$T_{Z,TCP,I}$: Transformationsmatrix von TCP im Zielpunkt bezüglich Inertialsystem I
$T_{G,I}$: Transformationsmatrix von Greifframe bezüglich Inertialsystem I

GÜ : Greifüberdeckung (definiert: weicher Griff/harter Griff/ Stabilität)
FA : Durch Feinbewegungspfad zu überwindender Abstand
GA : Greiferauskragung
SA : Sicherheitsabstand im Nahframe

Online, während der Ausführung der Greifaktion, müssen neben dem Greifframe selbst, $T_{N,G}$ und $T_{Z,G}$ online berechnet werden. Zu der Abhängigkeit von dem CAD-Modell des zu greifenden Assemblies sind folgende Abhängigkeiten besonders wichtig:

- $T_{N,G}$ = f(SA) mit SA = f(Zugänglichkeit, Abschattungsverhältnisse)
- $T_{Z,G}$ = f(GÜ, Sicherheitsabstände D1, D2)

GÜ, FA, GA und SA sind in dem hier behandelten Beispiel aus Gründen der Übersichtlichkeit als Skalare dargestellt. Im allgemeinen räumlichen Fall der Greifplanung der impliziten Aktionssteuerung sind dies teilweise wieder 4x4-Transformationsmatrizen. Für die Ansteuerung des Industrieroboters muß dann entsprechend

- $T_{N,TCP,I} = T_{G,I} \times T_{N,G}$
- $T_{Z,TCP,I} = T_{G,I} \times T_{Z,G}$

auf absolute Koordinaten umgerechnet werden.

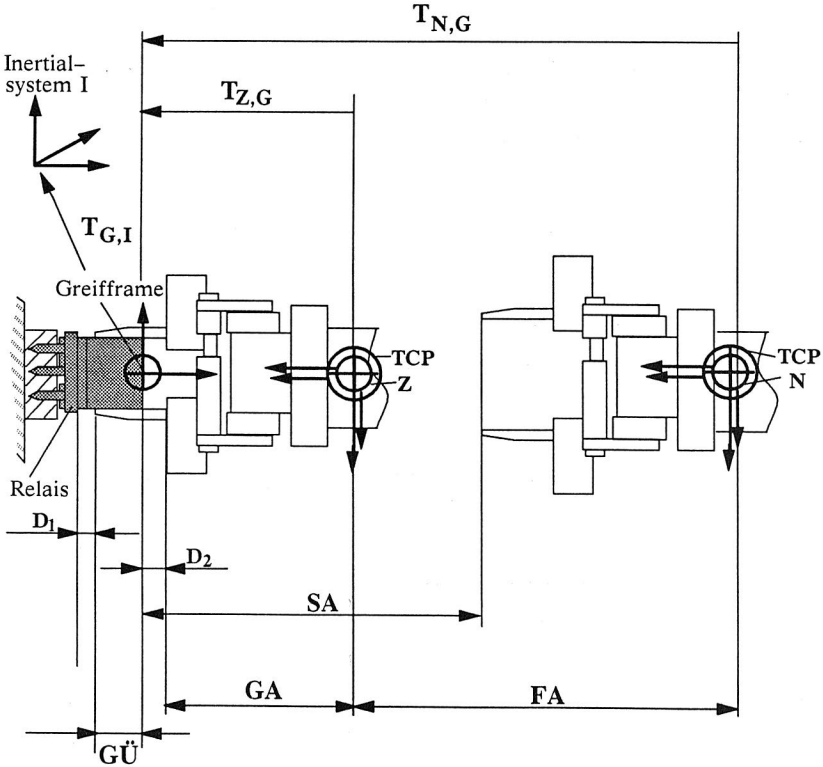


Abb. 3.16: Online-Berechnungen für eine beispielhafte Greifsituation

Die Auswahl geeigneter Greifframes kann z.B. anhand von Greiffeatures (=erweiterte Geometriefeatures, s. Kap. 3.2.1) erfolgen. In Zusammenarbeit mit dem Fügeplaner müssen GÜ bzw. D1, D2, noch vor dem Schließen des Greifers, vorausschauend so berechnet und eingestellt werden, daß die Zielposition und alle Positionen der Feinbewegung mit dem Greifer und dem gewählten Griff erreicht werden können, d.h. keine Kollision zwischen Greifer und einem passiven Bauteil o.ä. stattfindet. Ist diese Bedingung nicht erfüllt oder z.B. aufgrund der komplexen Montageoperationen nicht erfüllbar, muß ein Umgreifen stattfinden (z.B. indem die Greifframeliste eines Bauteils unter den geänderten Randbedingungen erneut durchsucht wird). Umgreifen muß jedoch möglichst vermieden werden.

Die Greifüberdeckung bestimmt die Sicherheit bzw. Stabilität des Griffs, bzw. über die programmierbare Greifkraft und den Reibwert die Härte oder Weichheit des Griffes. Damit wird festgelegt inwieweit noch Zentriervorgänge während des Fügens möglich sind. Im Falle begrenzter Spannkraft, z.B. wegen empfindlicher Bauteile, kann durch sehr geringe Werte von D1 oder D2 statt Kraft- ein Formschluß hergestellt werden und damit trotzdem hohe Fügekräfte erreicht werden.

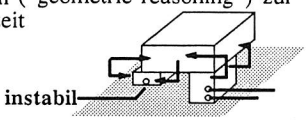
Die Greiferauskragung GA, wie auch weitere wichtige Abmessungen von Greifer und Werkstück, müssen aus dem aktuell geladenen Robotermodell extrahiert werden. Sicherheitsabstand SA wird mit der Berechnung des Nahpunktes festgelegt und ist damit Aufgabe des Fügeplaners.

3.2.5 Praxisnaher Common-Sense-Wächter

Von besonderer Bedeutung für die implizite Aktionsplanung ist die permanente "Einarbeitung" eines allgemeinen Wissensschatzes, dem Common-Sense. Common-Sense ist derjenige Teil eines Wissensvorrates, der sich auf allgemein gültige und zumindest für den menschlichen Montagearbeiter bekannte Zusammenhänge bezieht. So kann z.B. ein Werkstück ohne statisch ausreichend bestimmte Lagerung keine stabile Position im Raum einnehmen. Abb. 3.17 zeigt Beispiele aus dem Spektrum des Common-Sense in der Montagetechnik.

Trotz der Bedeutung des Common-Sense für die aufgabenorientierte Programmierung sind derzeit noch keine Ansätze für eine industrielle Umsetzung bekannt. Die Überprüfung auf Common-Sense gewinnt jedoch mit der impliziten Aktionsplanung an Bedeutung. Im Falle der expliziten Anlagenprogrammierung wird diese Überprüfung durch den Programmierer übernommen. Im Falle der impliziten Ak-

tionsplanung muß dies durch das Aktionsplanungswerkzeug selbst geschehen. Die permanente Hintergrundaktivität des Common-Sense-Wächters bestimmt das Laufzeitverhalten des Gesamtsystems wesentlich.

Common-Sense-Bereiche	**Auswirkungen in der Aktionsplanung**	**Berücksichtigung in der Aktionsplanung**
Sinnvolle Position eines Bauteils (Konsistenz-Tests am Ende von Fügeprozessen)	- instabile, indifferente Lagen - freier Fall	- Freiheitsgradbestimmung von Starrkörpern relativ zueinander: $F = 6(n-1-g) + \sum_{i=1}^{g} b_i$ Grübler - Gleichung - Abprüfen auf Mindest-Körperkontakt - Teilweise durch Modellierung physikalischer Gesetze (Reibung, ..)
Temporäre, topologische Beziehungen zwischen Körpern	- Seiteneffekte auf Bewegungsplanung (Fügeplaner, Nahpunktbestimmung, Bahnplaner) - Mitbewegen von Körpern	- Affixmentkonzept (vgl. V.Aken /121/) - Extraktion aus dem CAD-Anlagenmodell ("geometric reasoning") zur Laufzeit 
Sinnhaftigkeit einer Aktion in Abhängigkeit von Objekt und Operator (Zulässigkeit)	- Greifbarkeit von Objekten (z.B eine Führungssäule als Teil eines Masch.-gestells sollte nicht gegriffen werden)	- Techniken objektorientierter CAD-Systeme (OOCAD)

Legende: n = Anzahl der Körper (Glieder)
g = Anzahl der Gelenke (= verallg. alle Kontaktstellen zwischen Körpern)
F = Anzahl der Freiheitsgrade (F=0: statisch bestimmt, F>0: statisch unbestimmt, F<0: statisch überbestimmt)
b_i = Anzahl der jeweiligen Gelenkbeweglichkeiten pro Gelenk i
z.B.: $b_{\text{Fläche-Fläche}} = 3$ $b_{\text{Kante-Kante}} = 5$
$b_{\text{Fläche-Kante}} = 4$ $b_{\text{Kante-Punkt}}$ } nicht relevant
$b_{\text{Punkt-Fläche}} = 5$ $b_{\text{Punkt-Punkt}}$ }

Abb. 3.17: Beispiele zu Common-Sense

Implizite Aktionsplanung muß die Berücksichtigung temporärer, topologischer Beziehungen zwischen Werkstücken beinhalten. Für die meisten der möglichen Aktionen in einer flexiblen Roboterzelle muß diese Verkettung dynamisch zur Laufzeit aufgebaut und ausgewertet werden. Das skalare Ergebnis der Grübler-Gleichung in

Abb. 3.17 eignet sich dabei für automatisiertes Überprüfen; weniger jedoch für die anschließende Weiterverarbeitung.

Klassifikationen von Verbindungen (z.B. Ansätze nach /102/), die in Form von Schlußmatrizen (einschließlich darauf anwendbaren Operationen) Freiheitsgrade von Körpern ausdrücken (je 6 Halbachsen für Rotation und Translation) besitzen den Vorteil der einfachen Auswertung in der Symbolverarbeitung. Probleme bereitet die algorithmisch schwer faßbare Klassifikation anhand des Umweltmodells und Beweglichkeiten nicht in Halbachsenrichtung.

Ein einfaches Beispiel ist das Mitbewegen eines auf einer Grundplatte aufgelegten Bauteils. Wird das Bauteil z.B. senkrecht nach oben entfernt, hat dies keinen Einfluß auf die darunterliegende Grundplatte. Wird umgekehrt die Grundplatte im Laufe nachfolgender Aktionen vom Roboter gegriffen und bewegt, so muß aus dem Umweltmodell berechenbar sein, ob, und wenn ja, welche weiteren Werkstücke (zusätzlich zur Grundplatte) betroffen und z.B. mitzubewegen sind (in Abhängigkeit von Bewegungsrichtung, -art, etc.).

Für diese primären Auswirkungen muß der Common-Sense-Wächter nach jeder impliziten Aktion erneut topologische Beziehungen aus dem aktualisierten Umweltmodell extrahieren ("liegt-unter" im obigen Beispiel) und eventuell nicht mehr gültige verwerfen. Diese selbstermittelten, berechneten Fakten (interne Fakten) haben dann Einfluß auf nachfolgende Aktionen in der Fertigungszelle. Für die "liegt-unter"-Beziehung genügt eine Analyse der Menge der Kontaktstellen der Objekte mit den Winkeln zwischen Senkrechter und den Kontaktflächennormalenvektoren.

Attribute wie "Stabile Lage", Kontaktzustände und z.B. "liegt-unter"-Beziehungen pflanzen sich mit unterschiedlichen Vererbungsmechanismen über mehrere Körper hinweg fort. Durch die Art der topologischen Beziehungen/Attribute ist festgelegt, welche Art von topologischen Beziehungen/Attributen in welcher Richtung (⟷ zweiseitig, ⟶ einseitig, vgl. Abb. 3.18) vererbt werden (Phase der Synthese der Common-Sense-Daten). Als Ergebnis entsteht bereits bei einfachen Anlagenmodellen (z.B. Abb. 3.20) eine komplexe Verschachtelung der Beziehungen.

Beispiele für topologische Beziehungen **und** Attribute sind z.B.:

- stabile Lage
- liegt unter, liegt an (bzw. weitere topologische Kontaktzustände)
- ist verbunden mit (lösbar, unlösbar (verklebt, verschweißt, ..))

- Bodenkontakt (= Berührstelle mit Abstand Null zum Boden)
- unbeweglich (führt z.B. zur Zurückweisung eines Greifbefehls)

Sie müssen vom Common-Sense-Wächter gesetzt, verworfen bzw. dynamisch mitgeführt werden. Während des Betriebes müssen sie und ihre Zuordnung zu Körpern ständig neu generiert und nicht mehr zutreffende gelöscht werden. Verarbeitbare topologische Beziehungen müssen leicht erweiterbar und veränderbar sein.

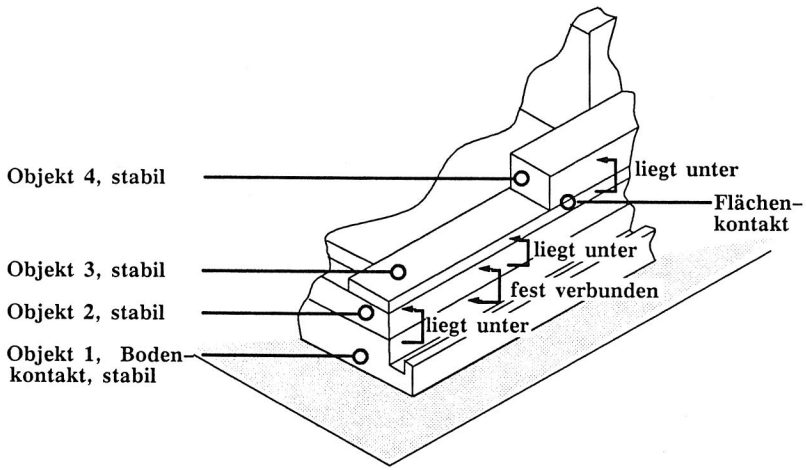


Abb. 3.18: Vom Common-Sense-Wächter ermittelte, topologische Beziehungen

Bei jeder Aktion muß der Aktionsplaner die Konsequenzen (z.B. das Mitbewegen von Körpern) der aktuellen Beziehungsmenge beachten (z.B. Einhängen in die kinematische Kette des Industrieroboters).

Grundsätzlich besteht eine Möglichkeit darin, temporäre Beziehungen direkt aus **erfolgreich** abgeschlossenen, impliziten Fertigungsbefehlen abzuleiten. (Bsp.: WST1 hat eine liegt-unter-Beziehung mit WST2 nach Abschluß des impliziten Befehls "Auflegen von WST2 auf WST1"). Nachteilig ist, daß nicht alle Beziehungen aus adäquaten impliziten Befehlen ableitbar sind, sowie daß beim Laden eines Umweltmodells und Start des Systems zwangsläufig noch keine impliziten Befehle ausgeführt wurden und somit auch noch keine temporären Verknüpfungen existieren. In der vorliegenden Arbeit wurden aufgrund der Einfachheit und Schnelligkeit, tem-

poräre Verknüpfungen aus impliziten Befehlen gewonnen, wo dies möglich ist (externe Fakten, Abb. 3.19).

Damit können zwei Klassen topologischer Beziehungen definiert werden:

- **Interne Fakten**
- **Externe Fakten**

Die internen Beziehungen lassen sich aus dem Umweltmodell berechnen (Geometrie-Reasoning). Die externen Fakten können z.B. von "außen" durch implizite Befehle zugeführt werden. Beide Klassen besitzen Interferenzen (Abb. 3.19).

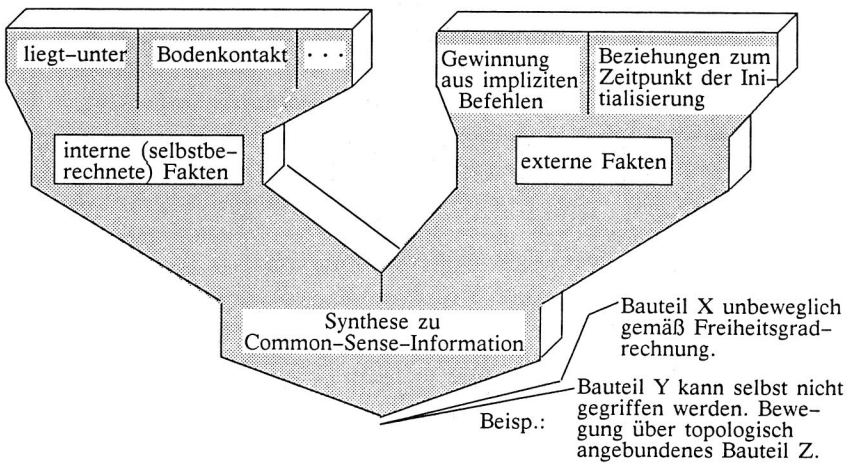


Abb. 3.19: Einteilung der durch den Common-Sense-Wächter ermittelten Fakten

Die topologischen Beziehungen müssen gespeichert und für die spätere Auswertung aufbewahrt werden. Langfristige, externe Fakten müssen zusammen mit dem Umweltmodell abgelegt (DB) und gleichzeitig mit dem Laden des Umweltmodells initialisiert werden. Anschließend werden die berechenbaren, internen Fakten bestimmt (+Vererbung) und Tests (z.B. Stabilität der Werkstücke) durchgeführt. Erst im Anschluß daran ist das System für implizite Aktionen bereit.

Die Teilaufgaben des Common-Sense-Wächters zur Berechnung der internen Fakten lassen sich mit, dem Fügeplaner ähnlichen Algorithmen (Kontaktart, Kontaktstellen- und Sperrvektorrechnung) lösen (s. Kap. 7.2.1).

Mit Hilfe von Monitoren können die Ergebnisse des Common-Sense-Wächters online mitverfolgt und protokolliert werden. Auf jedes im Umweltmodell vorhandene Objekt kann im realisierten Prototypen ein Monitor (=Mithörfenster) gelegt werden, der die Entwicklung (Entstehung und Verwerfung) der aktuell im System gültigen Fakten (topolog. Bez., Attribute) dynamisch anzeigt (Abb. 3.20).

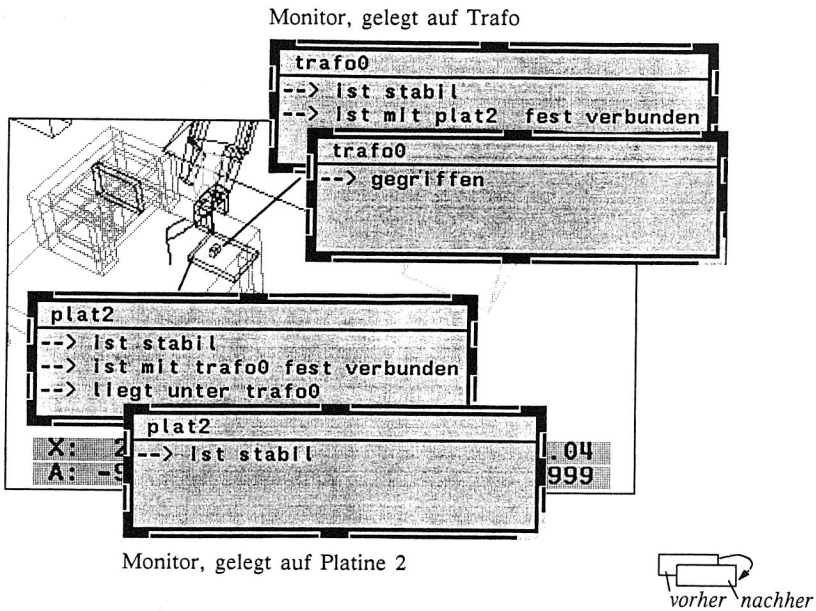


Abb. 3.20: Monitoring zur Verfolgung der intern berechneten Common-Sense-Fakten (Auswirkungen am Beispiel des impliziten Befehls "Demontiere Trafo !"); (Bildschirmkopie)

3.3 Online-Aktionsplan

Die zentrale Komponente des impliziten Aktionsplanungswerkzeuges ist der sog. Aktionsplan (oder Online-Arbeitsplan). Der Aktionsplan ist eine zur Laufzeit aufgebaute Struktur, die alle Teilergebnisse, die die Spezialisten im Laufe der Bearbeitung einer Fertigungsanweisung zuliefern, aufnimmt.

Erzeugung und Verfeinerung der Aktionsplanstruktur muß aufgrund der Kombinatorik möglicher Zellensituationen automatisch und dynamisch zur Laufzeit erfolgen, in

Abhängigkeit von der Befehlsstruktur, des aus dem Befehlsbaukasten (Kap. 5.2.5) aktivierten, impliziten Befehls.

Der Aktionsplan wurde im Rahmen dieser Arbeit in Anlehnung an eine Blackboard-Architektur (/46/, vgl. auch Überblick in /43/) als zentrales Blackboard des gesamten Kontroll- und Steuerungsmechanismuses konzipiert. Im folgenden wird der Aktionsplan sowohl hinsichtlich seiner Bedeutung als zentrale Struktur in der impliziten Aktionssteuerung und -planung behandelt, als auch im Hinblick seiner Eigenschaft als Quelle für den in Kap. 4.1 beschriebenen Aktionsplan-Übersetzer.

3.3.1 Darstellung von Aktionsplänen

Für die optimale Auslegung des Aktionsplanes müssen die an ihn gestellten Anforderungen untersucht werden. Diese Anforderungen teilen sich auf in, für die implizite Anlagenprogrammierung spezifischen Ziele, und Grundanforderungen, wie sie beim Entwurf effizienter Datenstrukturen beachtet werden müssen. Diese Zweiteilung entsteht aus der Zwischenstellung des Aktionsplans als 'Output' des Aktionsplaners und 'Input' des Übersetzers. Zunächst wird auf die Anforderungen an die Konstruktion des Aktionsplanes eingegangen:

Fähigkeit zur schrittweisen Verfeinerung:
Während der schrittweisen Verfeinerung einer impliziten Anweisung werden ständig Aktionen in mehrere Unteraktionen mit ihren korrespondierenden Daten, Reihenfolge-Beziehungen und Kontrollflüssen aufgespalten. Der Aktionsplan muß diese dynamischen Strukturen aufnehmen.

Flexibilität:
Flexibilität ist von besonderer Bedeutung für den Arbeitsplan. Sie wird von mehreren Seiten gefordert.
Zur Laufzeit müssen alle sukzessive gewonnenen Daten und Zwischenergebnisse wie Nahpunktframes, Greifframes, kollisionsfreie Pfade, Compileranweisungen, Schaltzustände, Signalwerte etc. aufgenommen werden. Sowohl Umfang, als auch Art dieser Daten sind variabel und erst zur Laufzeit bekannt.
Der Aktionsplan muß ein dynamische Gebilde sein, da eine implizite Fertigungsaufgabe durch schrittweises Verfeinern in immer *konkretere* Aktionen zerlegt wird. Die Aktionen müssen zur Laufzeit in Unteraktionen zerlegt und in den aktuellen Arbeitsplan integriert werden.

Erweiterbarkeit des Befehls-Baukastens, bzw. Wachsen des fertigungstechnologischen Wissens während des Einsatzes des Werkzeuges:
Die Arbeitsplan-Repräsentation soll insofern erweiterbar sein, daß eine Anreicherung des impliziten Befehlsumfangs um neue (immer komplexere) Fertigungsanweisungen möglich ist. Diese Forderung setzt eine abstrakte, allgemeine Darstellung voraus. Nicht geeignet ist ein Aktionsplan, der für spezielle Fertigungsaufgaben -wenn vielleicht auch mächtige- Datenstrukturen bereitstellt.

(NC)-Zielsprachen-Unabhängigkeit:
Die Repräsentation des Aktionsplanes muß unabhängig sein von konkreten, zu erzeugenden NC-Programmiersprachen bzw. Zielsprachen für RC, SPS, Bildverarbeitung etc. Das Spektrum reicht vom Assembler-nahen bis zu Hochsprachen-ähnlichen Code.

Anforderungen	textuelle Darstellung	Ablaufdiagramme/ Struktogramm	Petri-Netze	H-Graphen
schrittweise Verfeinerung	◒	◒	◒	●
Flexibilität	○	○	◒	◒
Erweiterung des Befehls-baukastens	●	◔	◔	●
(NC)-Ziel-sprachen-unabhängigkeit	●	●	●	●
Übersetzbarkeit	●	◔	◔	◒
Steuerungs-technische Eignung	◔	◒	●	●

○ nicht erfüllt ◒ gut erfüllt
◔ teilweise erfüllt ● vollständig erfüllt

Abb. 3.21: Bewertung rechnerinterner Repräsentationen von Aktionsplänen

Die folgenden Anforderungen entstehen aus der Verwendung des Aktionsplanes als "Quellsprache" des Aktionsplan-Übersetzers. Anforderungen, die darüberhinaus durch den Compilerbau entstehen, finden sich z.B. in /119/.

Übersetzbarkeit:
Der Aktionsplan muß auf der Basis einer Grammatik in die (NC)-Zielsprache übersetzbar sein, da keine Transformation in eine Zwischendarstellung mehr durchgeführt wird und die Übersetzung regelbasiert erfolgen soll.

Steuerungstechnische Eignung:
Mit Hilfe des Aktionsplanes müssen zeitparallele Tasks (Bewegungs-, Sensor-, Schaltaktionen etc.) beschreibbar sein. Dies ist besonders von Bedeutung vor dem Hintergrund der Hardwarebasis von IR-Steuerungen, die zunehmend auf dem Konzept modularer Mehrprozessorsysteme basieren (vgl. Kapitel 3.1.3) und (quasi)parallele Prozesse (Tasks) zulassen (move .. while .. für Bahnschweißen, Lageregelung/Interpolation gleichzeitig mit Sensorsignalauswertung etc.). Der Aktionsplan muß so mächtig sein, diese Nebenläufigkeiten sowie die in der Robotik und Steuerungstechnik wichtigen Kontrollstrukturen (Alternativen, Schleifen) aufzunehmen.

Zur Erfüllung der genannten Anforderungen zeigt Abb. 3.21 Alternativen.

Für textuelle Darstellungen, auf der Basis eindimensionaler, algorithmischer Sprachen, kann zwar die Grammatik (Regelwerk) vergleichsweise einfach entworfen werden, sie besitzen jedoch unzureichende steuerungstechnische Eignung (s.o.).

Die Übersetzung schrittweise verfeinerter, impliziter Instruktionen, bzw. fehlende Mechanismen für Echtzeitaktionen (Zeitparallelitäten, Ausnahmebehandlungen) sind Nachteile von Ablaufdiagrammen und Struktogrammen.

Das Konzept der Petri-Netze (s. /58/) eignet sich besonders zur Steuerung von nichtsequentiellen Prozessen. Ein für die Automatisierungstechnik geeignetes Petri-Netz besteht aus Transitionen (Aktionen), Stellen (Beschreibung von Zuständen), verbindenden Kanten und Markierungen. Eine Transition hat in der Regel mehrere Ein- und Ausgangsstellen. Feuern einer Transition heißt Ausführen der an die Transition angekoppelten Aktionen. Dies setzt voraus, daß alle Eingangsstellen gesetzt sind und führt zur Markierung aller Ausgangsstellen bzw. zum Löschen aller Eingangsmarkierungen (Abb. 3.22). In /51/ sind Erweiterungen bezüglich zeitbehafteter Transitionen, Prolog-Interpreter-Mechanismen etc. beschrieben (vgl. /94/). Schwierig-

keiten entstehen bei der vollständigen Übersetzung in Programmiersprachen (Verträglichkeit mit Perti-Netz eigenem Interpreter-Mechanismus /51/).

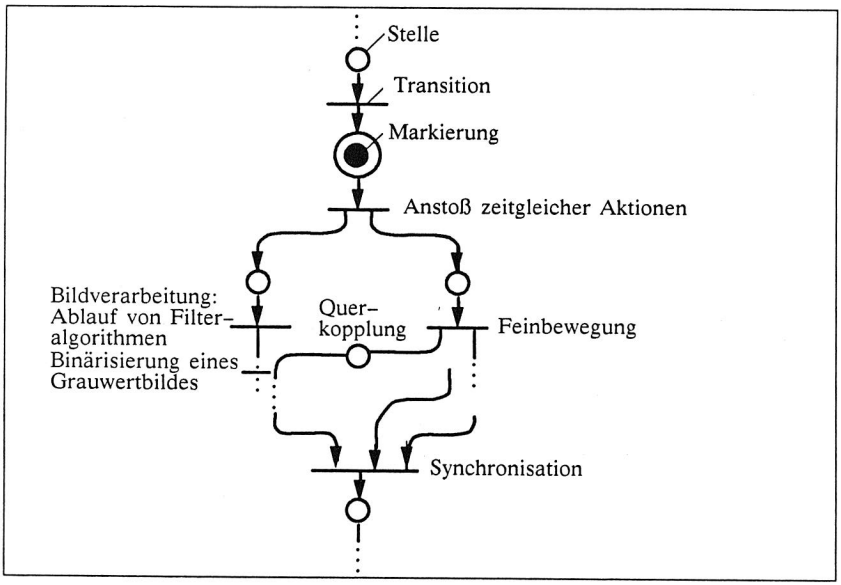


Abb. 3.22: Beispiel für nicht-sequentielle Aktionen

Im folgenden wird eine gut geeignete, und im Rahmen der Prototypenentwicklung dieser Arbeit eingesetzte Aktionsplan-Repräsentation vorgestellt - der H-Graph.

3.3.2 H-Graph

Der **H**-Graph (**H**ierarchischer Graph) ist eine Erweiterung eines **b**ewerteten, **g**erichteten, **i**nitialisierten Graphen (**bgi**-Graphen) z.B. zur Beschreibung kontextfreier Sprachen 1). Hierarchien lassen sich aufbauen, da Knoten des H-Graphen selbst wieder bgi-Graphen (Ablaufstrukturen, Rekursionen) enthalten können usw. Der H-Graph eignet sich deshalb sehr gut zur schrittweisen Verfeinerung impliziter Instruktionen. Für den praktischen Einsatz in der impliziten Aktionssteuerung und -planung in Fertigungsanlagen muß die H-Graph-Theorie 1) wie folgt übertragen werden:

Definition: Ein Aktions-H-Graph ist ein Paar

1) Theoretische Grundlagen und formale Definitionen s. /89/

AP = (A,c) mit: $A \subset A^{\sim}$

$A^{\sim}$: potentiell unendliche Menge impliziter, fertigungstechnischer Aktionen

$c: A^{\sim} \mapsto Q \cup (Q \times bgi\ (A^{\sim}, K, a_B, E))$

Inhaltsfunktion einer fertigungstechnischen Aktion + Erweiterungen s.u.

Q : Menge der elementaren Aktionen (nicht weiter zerlegbare Aktionsprimitive, z.B. 'Grobbewegung')

$bgi\ (A^{\sim}, K, a_B, E)$: bgi-Graph mit:

K : nichtleere Selektorenmenge (Kanten)

$a_B \in A^{\sim}$ Startknoten des aktuellen bgi-Graphen

$E: A^{\sim} \times K \mapsto A^{\sim}$ partielle Abbildung (Kantenbeschriftung K, z.B. Alternative, Verzweigung je nach Sensorwert einer Sensoraktion, etc.)

Abbildung E ist partiell, da es Aktionen gibt, aus denen keine Kanten herausführen. Ein, eine fertigungstechnische Aktion repräsentierender Knoten des H-Graphen kann als Inhalt einen bgi-Graphen mit erneut impl. Fertigungsaktionen oder Aktionsprimitiven enthalten. Die Hierarchiestufe HS einer Aktion R, z.B. Sensoraktion, ist definiert zu:

$$HS\ (R) := \min \{ i \mid R \in A^{(i)} \}$$

wobei:

$$A^{(1)} := \{ R \in A \mid c(A) \in Q \}$$

$$A^{(i+1)} := A^{(i)} \cup \{ R \in A \mid c(A) \in bgi\ (A^{(i)}, K, a_B, E) \}, \quad i \geq 1$$

Die Hierarchiestufe einer Aktion richtet sich damit nach deren erstmaligen Auftreten. Es sind auch rekursive Aktionen möglich.

Wichtige Mechanismen sind darüberhinaus **Abwicklung** und **Streckung**. Unter Abwicklung ist der passive Durchlauf entlang eines bestimmten Pfades (bestimmter Kontrollfluß) innerhalb des Aktionsplanes zu verstehen. Streckung dagegen wird als Ersetzung einer Aktion durch deren nichtelementaren Inhalt definiert. Streckung 1)

1)Synonym: Expandierung

bewirkt, im Gegensatz zur Abwicklung, eine strukturelle Veränderung des aktuellen Aktionsplanes.

Streckung eines impliziten Befehls bedeutet schrittweises Verfeinern des H-Graphen. Abb. 3.23 zeigt die Expandierung einer impliziten Anweisung.

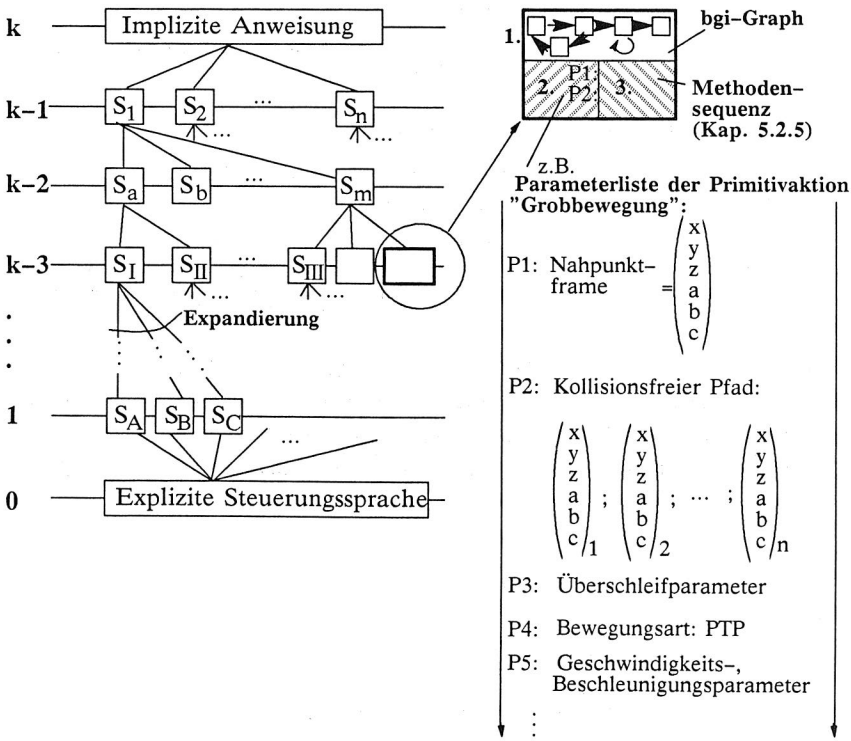


Abb. 3.23: Expandierung zum Aktionsplan (H-Graph)

Für die auszuführende, implizite Anweisung (Aktion) wird ein H-Graph-Knoten (Hauptknoten) angelegt, dessen Inhalt expandiert wird. Es entstehen Subtasks (Subaktionen) S, die wiederum verfeinert werden, bis der Hauptknoten vollständig abwickelbar ist. Bei jeder Zerlegung wird eine neue (niedrigere!) Hierarchiestufe angelegt und betreten. Jede bei der impliziten Aktionsplanung auftretende Fertigungszellenaktion entspricht einem Knoten des H-Graphen. Falls die Aktion elementar (nicht weiter zerlegbar) ist, ist der Inhalt ihres H-Graphen-Knoten ebenfalls elementar, anderen-

falls ist der Inhalt ein bgi-Graph, der weitere Subaktionen definiert. Es entstehen k Ebenen, wobei die Anzahl k von dem konkret angestoßenen Befehl abhängt. Aufgrund der, in der Praxis stark unterschiedlichen, impliziten Fertigungsbefehle ist diese flexible Anzahl von Zwischenebenen erforderlich (vgl. Flexibilitätsforderung). Eine fest vorgegebene Anzahl von Zwischenstufen für die Auflösung impliziter Befehle ist zu starr; auch im Hinblick darauf, daß während des Einsatzes der Werkzeuge Umfang und Mächtigkeit der implizit beherrschbaren Fertigungsprozesse ständig wächst.

Jeder Knoten A des H-Graphen besitzt mit der angegebenen Inhaltsfunktion c drei wesentliche Inhalte: A] bgi-Graph (abhängig von seiner Elementarheit, d.h. nur bei nichtelementaren Aktionen) und Bezeichnung B
B] Parameterliste P
C] Operationsliste (=Methodensequenz der Spezialisten) O

so daß gilt: $A = B \times P \times O$

Der bgi-Graph ist dabei die Grundlage für die fortgesetzte Streckung (s.o.). Die Operationsliste beschreibt, welche Methoden in welcher Reihenfolge bei der Expandierung des aktuellen Knotens ausgeführt werden müssen, z.B. Befragung von Spezialisten oder Aktualisierung des Umweltmodells. Die Liste von Parametern (Instanzenvariablen: vgl. Kap 2.3) ist für die vollständige Beschreibung dieses Knotens erforderlich und wird sukzessive von den Spezialisten gefüllt ("lokales Blackboard").

Jede Aktion ist durch eine ihr eigene Parameterliste gekennzeichnet. Welche Parameter zur vollständigen Beschreibung einer Aktion erforderlich sind, ist Bestandteil des langfristig gültigen Fertigungsprozesswissens (Kap. 5.2).

Experimente zeigen, daß mit steigendem Füllungsgrad der Wissensbasis (Stand der Wissensakquisition) und dem Fortschritt der Fertigungstechnologie bzw. Gerätetechnik häufig Änderungen und Erweiterungen der drei Inhalte eines Aktionsknotens erforderlich werden. Dieses Grundwissen sollte deshalb mit einer Datenbank zentral bereitgestellt und dem Entwicklungsstand angepaßt werden. Der Aktionsplan muß mit diesen Daten versorgt werden.

Bei der Initialisierung extrahiert der Aktionsplaner die aktuell gültigen Listen und allokiert sie anschließend. Die Parameterfüllung kann erst zur Laufzeit (abhängig vom aktuellen Anlagenmodell) erfolgen.

Operationssequenzen – und bgi-Graphen (Subaktionszerlegung) – sind ebenso langfristig und problemunabhängig gültig und müssen leicht erweiter- und veränder-

bar sein (Wartung entsprechend dem Stand der Technik). Sie sind zu jedem Aktionsknoten zugehörig in der Wissensbasis (s. Kap. 5.2) abgelegt.

Auf Ebene 1 dürfen nur Knoten stehen, denen genau eine Produktion (Regel) der linken Seite der Paargrammatik des NC-Sprachencompilers entspricht (s. Kap. 4.1). Bei einem vollständig aufgebauten Arbeitsplan sitzt ausschließlich der aufzulösende, implizite Befehl (Hauptknoten) auf der höchsten auftretenden Hierarchiestufe. Wird für eine Aktion auf einer Ebene < k die Elementarheit erkannt, dann endet dort die Expandierung für diesen Zweig (z.B. S_b in Abb. 3.23). Spätestens hier, oder je nach impliziter Anweisung bereits auf höheren Ebenen, muß der ausführende Aktor (Betriebsmittel) festgelegt sein. Höherwertige Befehle können sich auf einen Verbund von Betriebsmitteln beziehen, da z.B. mehrere Betriebsmittel an der Ausführung eines impliziten Befehles beteiligt sind. Die Funktionsaufteilung auf ein konkretes Betriebsmittel kann erst beim Auflösen in feinere Subtasks getroffen werden (z.B. Greifvorgang: Sensor zur Feststellung der 2D-Lage des Bauteils und IR zur Ausführung), da sie abhängt von der aktuellen Fertigungsanlage (d.h. geladenes Anlagenmodell) in Verbindung mit dem Teil der Wissensbasis (s. Kap. 5.2.1) der die Fertigungszelle beschreibt.

Der Aktionsplaner arbeitet auf dem Arbeitsplan; der Nachrichtenaustausch zwischen Aktionsplaner und Aktionsplan ist während der Expandierung eines impliziten Befehls sehr intensiv. Hierzu muß der Aktionsplan dem Aktionsplaner folgende Operationen bereitstellen:

1. Initialisierung des H-Graphen bzw. Knoten
2. "Aktion expandieren" bzw. "H-Knoten strecken"
3. Löschen von Teilen bzw. des gesamten Aktionsplanes, Terminieren von Aktionsplan-Instanzen
4. Füllung der Parameterliste eines H-Graph-Knoten
5. Lesen von Knoten-Inhalten (auf H-Graph und bgi-Ebene)
6. Überprüfen eines H-Graphen-Knoten auf Elementarheit
7. Weiterschalten (Durchlaufen) im H-Graph (Hierarchiestufen)
8. Markieren von H-Graph-Knoten (z.B. für spätere Rückkehr)

Unter Initialisierung (1.) fällt die Instanziierung von mehreren Aktionsplänen (zeitgleiche Aktionen). Die Punkte 1, 2 und 3 sind generische Operationen, die konstruktive wie destruktive Modifikationen beinhalten. Zu jeder generischen Operation müssen Plausibilitäts- und Konsistenzkontrollen durchgeführt werden. Der Ar-

beitsplan stellt nur die oben beschriebenen Mechanismen passiv bereit. Deren Ansteuerung muß alleine der Aktionsplaner leisten. Abb. 3.24 zeigt das Prinzip an einem Beispiel.

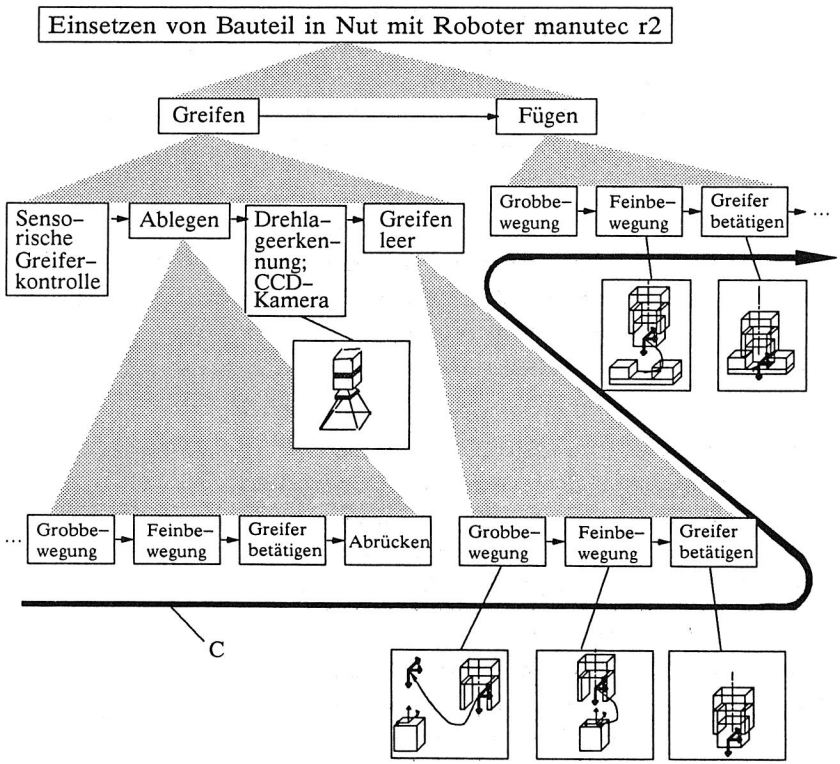


Abb. 3.24: Expandierung eines impliziten Befehls in Aktionsprimitive

Durchlauf C in Abb. 3.24 kennzeichnet die Sequenz der NC-Code-Erzeugung durch den abschließenden Compiler-Pass (Details s. Kap. 4.1). Die Steuerung des Aktionplan-Compilers erfolgt mittels den, zu den Subaktionen gehörigen Parameterlisten.

Es existieren zwei Arten von Sensoraktionen, die nicht nur physikalische Größen erfassen, sondern von diesen abhängig logische Alternativen durchführen:

Typ 1: Sensoraktionen, die in logische Alternativen verzweigen, welche direkt aus dem Umweltmodell heraus entschieden werden können. Sie müssen nicht in NC-Codes übersetzt bzw. auf der realen Fertigungsanlage durchgeführt werden. Häufig tritt dieser Typ bei interaktivem Arbeiten mit dem impliziten System auf. Es erscheint nur der zutreffende Zweig der Alternative im Aktionsplan.
Typ 2: Sensoraktionen und die mit ihnen verbundenen Alternativen können nur während des realen Fertigungsprozesses entschieden werden. Die Sensoraktionen und mögliche Alternativen müssen in NC-Code (z.B. in Form von Unterprogrammen) umgesetzt werden. Der Aktionsplan enthält alle Sensoraktionen und die weiteren Alternativ-Teilbäume.

Für die Simulation dieser Sensoraktionen und logischer Verzweigungen können am impliziten System durch Aufschalten von Zufallsgeneratoren (z.B. für Position und Orientierung eines Bauteils, das durch Bildverarbeitung erkannt werden soll) die realen Störgrössen simuliert werden. Anschließend ist im impliziten System eine eindeutige Entscheidung möglich (vgl. Kapitel 3.1.4).

Probleme bereiten logische Verzweigungen, die im Arbeitsplan nur in eine Richtung konkret weiterverfolgt werden können. Das Anlagenmodell kann nur gemäß einer (1!) Aktion verändert werden. Alle nachfolgenden Aktionen setzen dann auf dieses Anlagenmodell auf bzw. verändern es ihrerseits. D.h. die Aktionsknoten der nicht weiterverfolgten Alternativen können nicht mit konkreten Werten gefüllt werden, da die Daten nicht vorliegen (vgl. Opportunistische und zurückstellende Planung, /64/).

Eine Möglichkeit der Abhilfe besteht darin, die Verzweigung zu speichern, und erst im nachhinein diese Alternativen zu bearbeiten. Dazu muß das Umweltmodell auch entsprechend dem Zustand dieser Knoten (Historie) zurückgesetzt werden. Das Umweltmodell müßte hierzu definiert verändert werden, so daß die Bedingungen für eine Verzweigung auch zutreffen (z.B. Drehlage für Sensor). Für umfangreiche Arbeitspläne und große Anlagenmodelle wird diese Vorgehensweise sehr rechenaufwendig.

3.4 CAD/CAM-Verfahrenskette der Komplettmontage

Der denkbar komplexeste Befehl, den der Bediener an eine autonom handelnde Roboterzelle zur Ausführung übergeben kann, lautet beispielsweise:

"Montiere Netzgerät"

Derartige Befehle werden im weiteren zu **Komplettmontage-Befehlen** definiert. Sie entsprechen Betriebsmodus (II) in Kap. 3.2.1 und werden in diesem Kapitel behandelt. Sie besitzen gegenüber Betriebsmodus (I) einen wesentlich höheren Automatisierungsgrad, da das indirekt durch die Interaktionen des Benutzers eingebrachte Wissen durch Wissen aus der Produktkonstruktion und AV ersetzt werden muß. Da prozeßnahe Aktionsplanung auf Werkstattebene und die CAD/AV-Basisdaten verteilt sind, muß das Aktionsplanungswerkzeug diese Daten über den Fabrikbus (s. Kommunikation Kap. 4.3.1) beschaffen (Abb. 3.25).

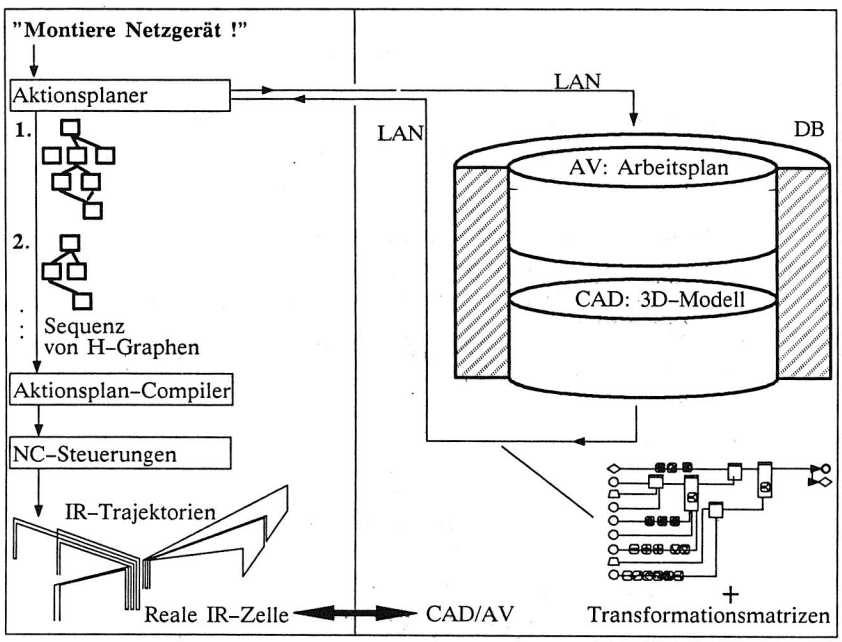


Abb. 3.25: Einlesen und Abarbeiten eines AV-Arbeitsplanes in der Werkstatt

Die Struktur der dadurch implizierten Subbefehle ist produktspezifisch. Sie müssen von den Komponenten der Montagezelle sequentiell oder parallel abgearbeitet werden. Das implizite System auf Werkstattebene liest dazu den in der AV erstellten Arbeitsplan (Montageplan) aus der DB über LAN ein. Der Arbeitsplan enthält als elementarste Einheit oben behandelte implizite Befehle. Die Graphstruktur des Arbeitsplanes bestimmt die Reihenfolgebeziehungen.

Der AV-Arbeitsplan kann manuell, unter Verwendung der impliziten Befehle, mit Hilfe eines datenbankgestützten Werkzeuges (s. Kap. 3.4.3) erstellt werden. Das Ziel sind Arbeitspläne auf dem Niveau der eigentlichen Fertigungsaufgabe, die von dem Werkzeug auf Werkstattebene 'verstanden' werden können.

Eine zusätzliche Voraussetzung ist die Analyse der 3D-CAD-Produktmodelle (z.B. Netzgerät s.o.) im "implodierten" Zustand. Hierzu wird ein CAD-Implosionsmodell im weiteren zu demjenigen Volumenmodell definiert, das alle Baugruppen, Einzelteile etc. des automatisiert zu montierenden Gerätes, bzw. der Baugruppe im zusammengebauten Zustand enthält.

Das Zusammenstellungsmodell, einschließlich aller enthaltenen, vorgefertigten Baugruppen, muß für die Abarbeitung der Komplettmontage-Befehle gespeichert sein, so daß die **zueinander relativen** Transformationsmatrizen der Bauteile berechnet werden können. Die Geometrieanalyse kann hierzu mit Hilfe der Datenbank (Kap. 5) auf die CAD-Modelle von Einzelteil und Komplettprodukt zugreifen.

Ansätze, die die Montagereihenfolge durch die Analyse der CAD-Modelle berechnen (vgl. /41/) sind nur für einfache Werkstücke und feste Fügerichtung erfolgversprechend. Montageaufgaben, wie sie in der Gerätetechnik vorherrschen, sind aufgrund der komplexen und unterschiedlichen Fügetechnik (beliebige, wechselnde Fügerichtungen/-bahnen etc.) jedoch schwer faßbar.

3.4.1 CAD-Modelliererkern

In dieser Arbeit werden für Fertigungsanlage und Produkt ausschließlich 3D-CAD-Modelle verwendet, die in sog. **B**oundary-**Rep**resentation (**B-Rep**) rechnerintern angelegt sind (vgl. Kap. 3.1.1). Sie repräsentieren Geometrieelemente (z.B. Flächen) durch sie begrenzende Elemente (z.B. Linienzug). Weitere CAD-Modell-Strukturen wie CSG (**C**onstruktive **S**olid **G**eometry) besitzen den Vorteil, die Konstruktionshistorie direkt abzulegen (Boolsche Operatoren auf Volumenprimitiven, vgl. /7/). Sie sind jedoch nachteilig für Algorithmen der Feinbewegungsplanung (z.B. Berührstellenanalyse, s. u.). Sie werden nicht weiter behandelt. CAD-Systeme (z.B. Sigraph) besitzen einen Modellier-Kern - den Geometriemodellierer, auf dem Graphik und Benutzeroberfläche des CAD-Systems aufsetzen. Der Modellierer erzeugt und arbeitet auf CAD-Modellen.

Ein derartiger Standardmodeller, wie ihn z.B. PARASOLID darstellt, kann Freiformflächen verarbeiten und besteht aus einer Bibliothek von ca. 250 C-Routinen

/79/. Diese Routinen des CAD-Kerns können in C-Programme eingebunden (z. B. alternativ statische oder dynamische Bindung) werden und lassen eine direkte Extrahierung wie auch Manipulation der CAD-Modelle zu (= **K**ernel-**I**nterface, **KI**). Sie beinhalten Identifikation und Manipulation einzelner geometrischer (z.B. Punkt, Strecke) und topologischer Elemente sowie Klassen geometrischer Elemente (z.B. Klasse der Hilfspunkte) über Schlüsselattribute (sog. "Tags" od. "Tokens"). Abb. 3.26 zeigt das semantische Schema der Geometrie /126/.

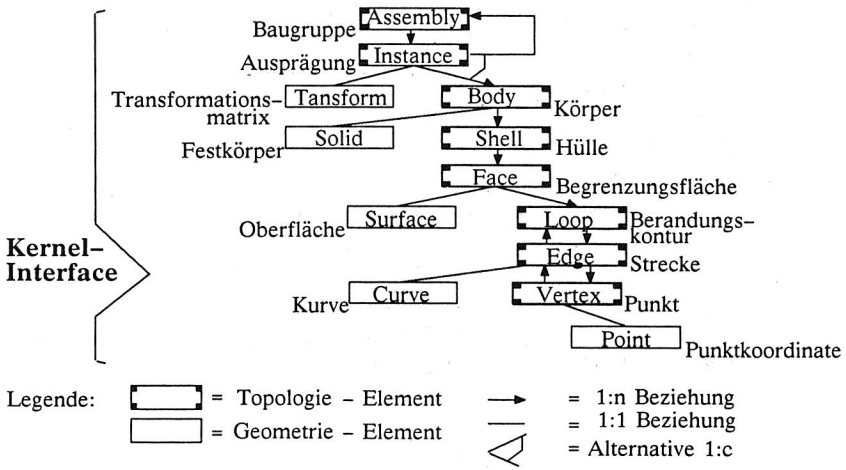


Abb. 3.26: KI-(Interface) zu B-Rep nach /126/

Geometrie und Topologie sind getrennt. Instanzen sind virtuelle Kopien von nur einmalig modellierten Körpern (z.B. mehrfach verwendete Normteile wie Schrauben). Die zugehörigen Transformationen sind in Matrixform abgespeichert. Aus Laufzeitgründen ist es günstig, analog zum Connect und Disconnect an die Datenbank, beim Start eines Montagekomplexbefehls den Modeller zu starten, und erst nachdem alle Zugriffe des impliziten Systems abgeschlossen wurden, den Modeller wieder zu stoppen.

3.4.2 CAD-Geometrieanalyse

Generell gilt, daß sich die laufzeitoptimierten, geometrischen Berechnungen auf Werkstattebene (auf Basis einfacher CAD-Modelle) durch bedarfsgesteuerten Zugriff (z.B. über LAN, TCP/IP) auf den Modellier-Kernel mit den erweiterten Berechnungsmöglichkeiten des Kernels (CAD-Ebene) sinnvoll erweitern lassen. Die

geometrische Analyse des CAD-Modells liefert Endposition und Orientierung der Körper für die Aktionsplanung (homogene Koordinaten bezogen auf körperfeste Koordinatensysteme) sowie berechnete Feinbewegungstrajektorien für die End- und Baugruppenmontage. Die Aktionsplanung benötigt diese relativen Transformationen bei der Auflösung impliziter Befehle. Die relativen Transformationsmatrizen müssen zur Ausführungszeit unter Berücksichtigung der absoluten Lagen und Orientierungen (aus dem Anlagenmodell) in absolute Werte umgerechnet werden (s. Abb. 3.27).

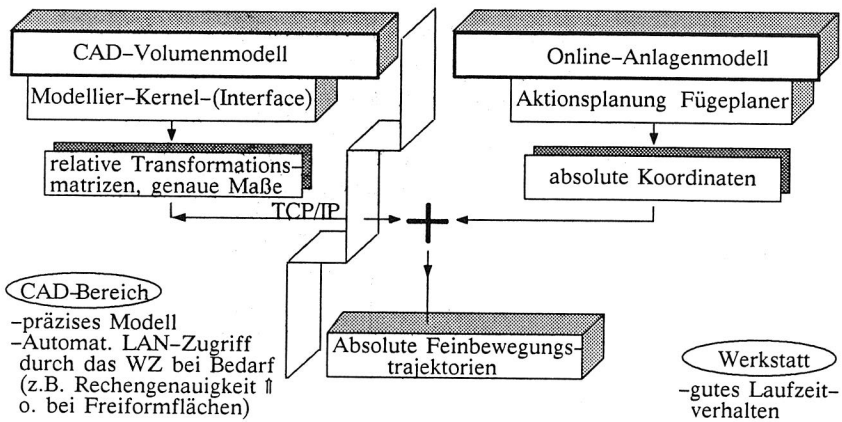


Abb. 3.27: Gewinnung absoluter Koordinaten auf Werkstattebene

Aus dem CAD-Implosionsmodell lassen sich die

◆ Zwangsbedingungen (Sperrvektoren-Rechnung s. Kap. 7.1.2), ◆ Verbleibende Restfreiheitsgrade und ◆ die relativen Feinbewegungstrajektorien berechnen.

Die Berechnung der Feinbewegungstrajektorien (im einfachsten Fall linear unter Beibehaltung der Orientierung) kann analog zu den online-Verfahren in Kap. 7.1.2 erfolgen. Der Feinbewegungsplaner berechnet eine Feinbewegungstrajektorie zwischen zwei gegebenen bzw. vorab berechneten Stellungen (Montageend- und Nahpunktframe) des AF innerhalb des abgelegten CAD-Umweltmodells. Voraussetzung: Der kollisionsfreie Nahpunkt wurde bereits bestimmt. Feinbewegungstrajektorien lassen sich dabei vereinfacht als eine Liste homogener 4x4-Matrizen ablegen.

Zur Erleichterung der Online-Fügeplanung sollten optional mit einem "**halbautomatischen Trajektorienedtior**" relative Fügetrajektorien im CAD-System definiert werden können. Er liegt zwischen den beiden Extremen "manueller Trajektorienedtior" und automatische Berechnung und kann die Funktionalität des CAD-

Modellier-Kerns nutzen. Der Bediener schränkt durch die Vorgabe von Hilfsgeometrie die Suchräume für die Algorithmen des Fügeplaners ein (in Form von Hüllkörpern, z.B. innerhalb eine Folge von Zylindern). Hohe Rechengenauigkeit der Geometriealgorithmen kann so mit praktischer Erfahrungen des Bedieners in der Montagetechnik kombiniert werden.

3.4.3 DB-basierte Arbeitsplanerstellung

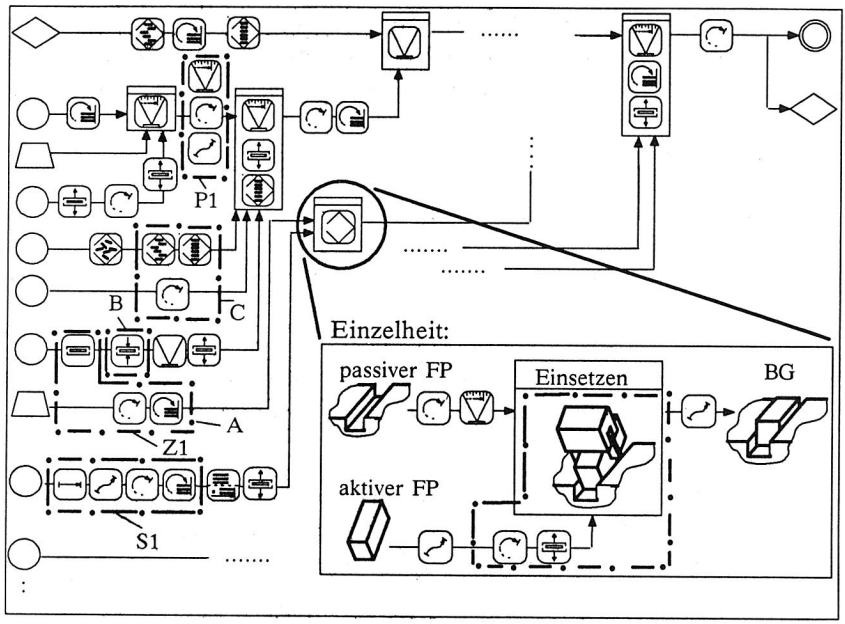
Das Aktionsplanungs- und -steuerungswerkzeug kann einen Befehl wie "**Montiere Netzgerät !**" nur dann ausführen, wenn für das zu montierende Produkt (Netzgerät in obigem Beispiel), ein Arbeitsplan im Rechner vorliegt, auf den der Aktionsplaner online zugreifen kann. Dieser in der AV auf Workstationbasis erstellte Montageplan darf nicht (!) mit dem zur Laufzeit vom System erzeugten Aktionsplan verwechselt werden.

Die Fügetechnik bestimmt den Montagegraphen des zu montierenden Produktes. Der Montagegraph wird in der Datenbank gehalten und ist entsprechend Abb. 3.28 aufgebaut. Er wird mit einem datenbankgestützten Werkzeug erstellt, das auf einer Erweiterung (implizite Geräteprogrammierung) des Montagegraph-Editors nach /109/ basiert.

Input dieses datenbankgestützten Werkzeuges zur Arbeitsplanerstellung ist der über Kontaktflächenmatrix und gerichtetem Fügeflächengraph ermittelte Teilevorranggraph. Dieser Teilevorranggraph beschreibt die technologisch sinnvollen Reihenfolgebeziehungen der zu montierenden Bauteile und Baugruppen. Er ist ebenso wie die Produkteinzelteile in der Datenbasis gespeichert und wird durch das Arbeitsplanerstellungswerkzeug zu Beginn der Sitzung geladen.

Output des Arbeitsplanerstellungswerkzeuges ist ein erweiterter Montagegraph, der ebenfalls vollständig in der DB abgelegt wird (einschl. eventueller Zwischenversionen) und die Montageaufgabe beschreibt. Dieser Montagegraph wird im Falle von Montagekomplettbefehlen vom prozeßnahen Aktionsplanungswerkzeug über LAN aus der Datenbank eingelesen. Ein Beispiel hierfür zeigt Abb. 3.28.

Die in /109/ vorgestellte, strukturelle und operationale Beschreibung dient als Grundlage und wurde um die Belange der impliziten Programmierung erweitert. Die Knoten und Kanten des Graphen können je nach Fertigungsaufgaben mit Sequenzen von erforderlichen impliziten Aktionen (z.B. Montage-, Füge, Meßoperationen) belegt werden.



Legende:
= Einzelteil
= Produkt
= Baugruppe
= Hilfsteile (z.B. Vorrichtungen)
= Fertigungsprozeßblock, kennzeichnet den eigentlichen Fertigungsprozeß (z.B. Messen, Einstecken)
Nummer/Bezeichnung des Fertigungsverfahrens bzw. der Handhabungs- und Kontrolloperation (nach DIN 8593, VDI 2860 oder eigendefinierte implizite Fertigungsaktionen)
= Montagepfade, kennzeichnen Übergänge von einem Fertigungsprozeß zum nächsten (Reihenfolgebeziehungen), bzw. das akt. Werkstück, auf das der implizite Befehl angewandt wird.
= Explizit vorgegebene Zwangsbedingungen (z.B. Gerätetechnische Zusammenfassung, Zeitparallelität, ...)
= Implizite Befehle in Piktogramm-Form (z.B. Aktionsprimitive). Sie können auf Pfaden oder innerhalb von Fertigungsprozeßblöcken plaziert werden.

Abb. 3.28: Fügetechnik und Arbeitsplan in Graphenform (Montagegraph)

Der Aktionsplaner liest diesen erweiterten Montagegraphen, sofern in der DB gespeichert und freigegeben, ein, und arbeitet ihn im wesentlichen von links nach rechts ab. In den Graphen werden vorher die relativen Fügepfade als Ergebnis der CAD-Analyse (über Kernel-Interface, s.o.) eingetragen. Mit Hilfe der absoluten Koordinaten des Anlagenmodells müssen die relativen Transformationsmatrizen des Montagegraphen in absolute Koordinatenwerte umgerechnet werden. Die konstruktive Zuordnung zwischen Betriebsmittel (Gerät) und impl. Aktion wird für die implizite Aktorenspezifizierung ausgewertet.

Wichtig ist ein datenbankgestütztes Werkzeug zum Editieren, Laden, Speichern etc. von produktbezogenen Montagegraphen. Der Inhalt von Abb. 3.28 muß hierzu z.B. in der Datenbank vorhanden sein. Mit diesem datenbankgestützten Werkzeug zur interaktiven Erstellung eines derartigen Montagegraphen müssen sich Sequenzen von Aktionen (S1 in Abb. 3.28), Parallelisierung von Aktionen (P1) wie auch eine gerätetechnische Zusammenfassung (Z1, A, B, C) von Aktionen durch einen Aktor der Fertigungszelle erzwingen lassen. Expertensysteme zur Erstellung dieses Arbeitsplanes können die manuelle Generierung nur unterstützen, da schlecht formulierbares Wissen, fehlende logische Schlußfolgerungen und unzureichende Bewertungskriterien vorliegen.

Der Montageprozeß kann aufgespalten werden in eigentliche Fügevorgänge (=Fertigungsblock) und vorab oder nachher nötige "Zusatzaktionen" (z.B. Handhabungs- oder Meßvorgänge zur Vorbereitung bzw. Nachbearbeitung). Die Eigenschaften der beiden Komponenten im zusammengebauten Zustand kann wesentlich von den Eigenschaften der Einzelkomponenten abweichen.
Beisp.: Passiver Fügepartner: Reagenzglas mit Flüssigkeit gefüllt; Aktiver FP: Verschluß. Die zwingende Nebenbedingung "Winkel zwischen Drehachse und Horizontaler ca. 90°", die für die Bewegungssequenzen des passiven Teils noch galt, entfällt für weitere Positionier- und Bewegungsvorgänge.

Die auszuführenden Aktionen müssen sich aus den freigegebenen, impliziten Befehlen des Befehlsbaukastens (s. Kap. 5.2.5) auswählen und durch einfache Maus-Selektion den Knoten und Kanten zuordnen lassen. Es dürfen nur solche Aktionen im Montageplan verwendet werden, deren Expandierung auf Aktionsprimitive und Umsetzung in NC-Code auch vollständig durchgeführt werden kann, d.h. deren zugehöriges Wissen in der Datenbank vorhanden ist. Im Arbeitsplanerstellungswerkzeug dürfen dazu nur implizite Befehle anwählbar sein, die diese Bedingung erfüllen.

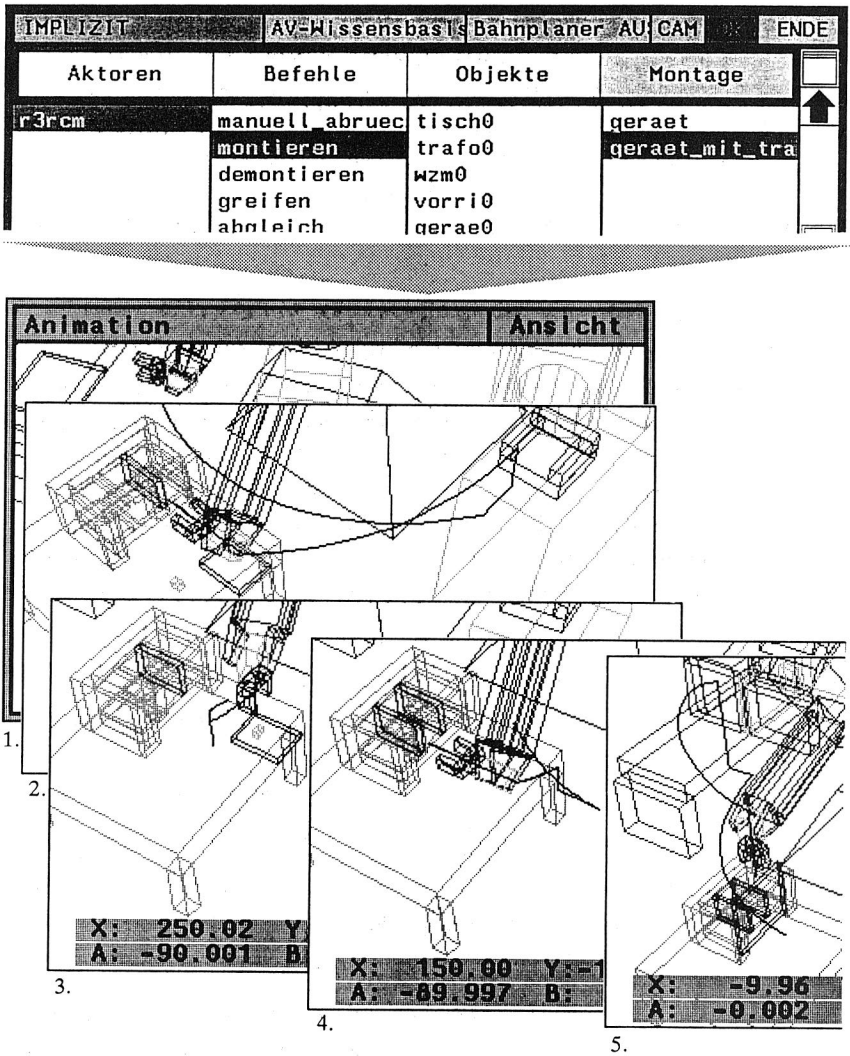


Abb. 3.29: Ausführungsphasen eines exemplarischen AV–Montagegraphen in der Werkstatt (Bildschirmkopie; Selektion der Baugruppe bzw. Gerät (Montagegraph) 4. Spalte v. links)

Die Menge der angebotenen Befehle ist im wesentlichen identisch mit dem Befehlsmenü auf Werkstattebene (Abb. 3.5, Kap. 3.2.1). Gegenüber dem interaktiven

Arbeiten sind weitere Informationen wie z.B. die "Kann-Zuordnung", bzw. "Konstruktions-Zuordnung" zwischen Betriebsmittel und Fertigungsaktion erforderlich (s. konzeptionelles DB-Schema).

Ein Ausschnitt des Montageplanes wird wie folgt abgearbeitet (Einzelheit in Abb. 3.28):
Abarbeitung eines Fertigungsprozeßblocks erst nachdem alle seine linken Vorgänger in obigem Graphen abgearbeitet sind. Zuerst müssen die von unten (aktiver Fügepartner) und die linksseitig (passiver FP) in den Fertigungsblock einmündenden, impliziten Aktionen ausgeführt werden. Anschließend wird der eigentliche Inhalt des Fertigungsprozeßblocks ausgeführt. Während der sukzessiven Abarbeitung dieser Arbeitsplan-Struktur arbeiten die Komponenten (z. B. Mitteilung von Topologieänderungen an den Common-Sense-Wächter) des impliziten Systems wie beschrieben.

Abb. 3.29 zeigt das Ergebnis der CAD/CAM-Verfahrenskette anhand der Abarbeitung eines einfachen Montagegraphen zur Gerätemontage (Komponenten: 19"-Einschubgehäuse, Frontabdeckung, zu bestückender Transformator und zwei auf den Rückwandbus zu steckende Platinen).

Die Fertigungszelle in der Werkstatt ist dadurch in der Lage einen in der AV erstellten Arbeitsplan auszuführen, der implizite Fertigungsanweisungen enthält. Die Arbeitsplanerstellung wird damit wesentlich erleichtert und liegt näher an der Fertigungsaufgabe.

4. Ankopplung zum Fertigungsprozeß

4.1 Generalisierte Umsetzung von Aktionsplänen in ablauffähigen Steuercode

Der Zielsprachencompiler bewältigt den letzten der erforderlichen Schritte, um eine implizite Fertigungsanweisung in Steuercode der Gerätesteuerungen zu transformieren. Er analysiert den vollständig aufgebauten Aktionsplan (aus Kap. 3.3) und synthetisiert daraus den korrespondierenden Zielcode (z.B. SRCL, IRL).

Für die Entwicklung eines Aktionsplan-Compilers muß eine Vielzahl von generell im Compilerbau wichtigen Punkten beachtet werden. Diese grundlegenden Compiler-Techniken werden an dieser Stelle nicht behandelt; sie finden sich in /119/, /95/ und /3/.

Im Gegensatz zur Compilertechnik für Computerprogrammiersprachen muß der Aktionsplan-Compiler zusätzlich Daten des Online-Anlagenmodells einarbeiten (Abb. 4.1). Dies sind beispielsweise Verschaltungen der binären/analogen RC-Ausgänge mit einem F/M-Sensor an der IR-Handwurzel oder dem Greifwerkzeug. Bereits projektierte Technologie-Funktionsbausteine müssen ebenfalls eingearbeitet werden. Die anzusteuernde Geräte- und NC-Steuerung (=Betriebsmittel im Anlagenmodell) legt die Schalterstellung S fest. Der Compiler extrahiert die zur Zielsprache zugehörige Paar-Grammatik aus dem anlagenspezifischen Teil der Datenbasis (Kap. 5). Schalterstellung S kann innerhalb der Übersetzung des Aktionsplanes wechseln.

Mehrere übersetzte H-Graphen müssen event. durch einen nachgeschalteten Linker gebündelt werden.

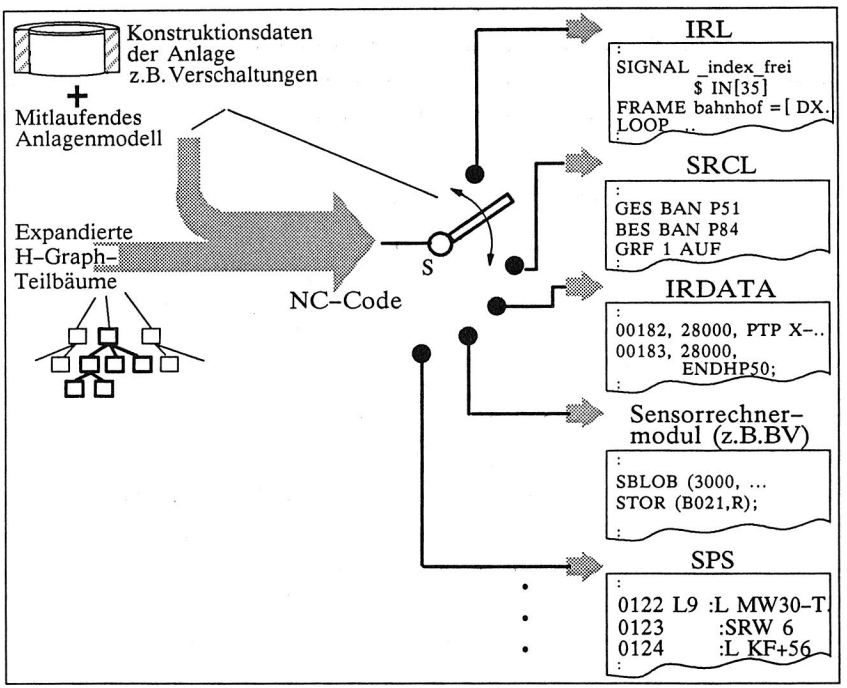


Abb. 4.1: Erzeugung von Zielcode

Wie in Kap. 3.3 dargelegt, ist der maschinenunabhängige Aktionsbaum Grundlage für die Generierung von NC-Code. Die Menge aller möglichen Aktionspläne ist die "Quellsprache" des Aktionsplan-Compilers. Der Compilerpass zur Erzeugung des Steuercodes wird nach vollständiger Expansion des H-Graphen, einschließlich enthaltener zeitparallelen Aktionen, angestoßen. Der Compilerteil, der den (Quell-)Aktionsplan vollständig durchläuft wird als Paß bezeichnet. Bei Mehrpaß-Compilern werden, im Gegensatz zu Einpaß-Compilern, Zwischenergebnisse erzeugt, die vom nachfolgenden Paß wieder gelesen werden.

Im Gegensatz zu Standardcompilern ist die Quelle des Aktions-Plan-Compilers ein Graph, d.h. eine nichtlineare, mehrdimensionale (genau: 3-dimensionale) Struktur,

die nicht sequentiell in eine Richtung abgearbeitet werden kann. Für die Umsetzung des Aktionsplanes sind bekannte Verfahren teilweise unbrauchbar. Es müssen neue Methoden entwickelt werden.

Für die implizite NC-Programmierung eignet sich besonders das Konzept der "Paar-Grammatik" /89/, das für den Aktionsplan-Compiler angepaßt werden muß. Die Paar-Grammatik R (Quintupel) wird wie folgt definiert:

$$R = (N_Q \cup N_G , T_Q \cup T_G, M, \{S_Q , S_G\}, P);$$

mit: $Q = (N_Q , T_Q , M, S_Q , P_Q)$ = Grammatik des Aktionsplanes (Graph-Grammatik),

$G = (N_G, T_G, S_G, P_G)$ = Grammatik der NC-Zielsprache, (Zeichenkette);

mit: T = Terminalalphabet
N = Nichtterminalalphabet
M = Markierungsalphabet
S = Startsymbol mit $S \in N$
P_Q, P_G = Produktionsmenge und
$(l, r, h) \in P$: Produktionen der Paar-Grammatik,

mit: l = linke Regel von P (Arbeitsplan)
r = rechte Regel von P (Zielcode)
h = Nichtterminal-Paarung zwischen Arbeitsplan-Regel l und Zielcoderegel r

Beispiel für IRL:

N = { IMPLIZIT, Signal-Vereinbarung, Signal-Name, Feinbewegung, ...}

T = { SIGNAL, DEF, END, PTP, LIN, &, ... }

Die Übersetzung erfordert eine große Anzahl von Schritten, falls die NC-Sprachen-Grammatik feiner ist (d.h. mehr Regeln besitzt) als die linke H-Graphen-Grammatik. Die Analyse des Arbeitsplanes und die Synthese des Zielprogrammes steuert und synchronisiert sich gegenseitig über gepaarte Ableitungs-Regeln (sog. Produktionen) der linken und rechten Grammatik (syntaxgesteuerte Übersetzung). Produktionen definieren in Form von Regeln Ersetzungsmöglichkeiten, wie die Quell-Grammatik ("linke Seite" der Produktion) auf die Zielgrammatik ("rechte Seite" der Produktion) abgeleitet werden kann.

Der Aktionsplan mündet in einen sog. Quell-Ableitungsbaum, dessen Knoten entweder Terminalsymbole T, d.h. sie sind nicht weiter zerlegbar (Blätter des Baumes), oder Nichtterminalsymbole N, sind, die sukzessive weiter zerlegt werden, bis auch sie in eine Abfolge von Terminalsymbolen transformiert werden können (einfach bei kontextfreier Grammatik). Der Ableitungsbaum wird durchlaufen und an Stellen, die mit Nichtterminalen der NC-Sprachen-Grammatik gepaart sind, wird entweder nur ein (1!) Nichtterminal (genauer: mit einseitiger "Null"-Produktionsregel) oder ein korrespondierendes Paar von Nichtterminalsymbolen (parallel auf linker wie auf rechter Seite) entsprechend einer Produktion aus P der Paar-Grammatik ersetzt, bzw. die Zielcode-Generierung angestoßen. So wird sukzessive Zielcode erzeugt bzw. die Übersetzung vorangetrieben. Die Linke Seite (Menge von H-Graphen) und die Rechte Seite (Menge von Zeichenketten des Steuercodes) werden parallel, beginnend bei den jeweiligen Startsymbolen, durchlaufen, bzw. generiert.

Während der NC-Code-Erzeugung müssen ständig Objektmethoden (z.B. des Aktionsplanes nach Kap. 3.3.2, Gerätedaten-Anfragen) aktiviert werden. Dies geschieht mit Hilfe von, in den Zielsprachen-Ableitungsbaum eingestreuten, zielsprachenabhängigen Semantik-Routinen.

Der Aktionsplan-Compiler soll in der Lage sein, ein weites Spektrum von Zielcode zu generieren. Das reicht von Assembler-ähnlichen Sprachen wie z.B. SRCL, die aufgrund ihrer einfachen Struktur leicht zu erzeugen sind, bis zu strukturierten, höheren Robotersprachen wie z.B. IRL. Deren Erzeugung ist aufgrund ihrer mächtigen Sprachkonstrukte (Kontrollstrukturen, Unterprogrammaufrufe) wesentlich aufwendiger. Besonders wichtig sind Deklarationen bzw. Fernzusammenhänge (Behandlung von Vorwärts-/Rückwärtsreferenzen, s.u.).

Die Konzeption des Übersetzers sollte unabhängig von dem, auf seiten der Zielsprache konkret zu generierenden (NC-)Code sein, da z.B. (noch) keine de-facto RC-Sprache existiert (deshalb 'generalisiert'). Der Compiler selbst und die rechnerinterne Darstellung der zu erzeugenden Sprachen muß getrennt werden. Dies soll unter anderem mit dem Aufbau eines Prototypen anhand von zwei, in ihrer Mächtigkeit sehr unterschiedlichen RC-Sprachen (Beisp.: assemblerähnliche SRCL und Hochsprachen-ähnliche IRL), getestet werden.

Im Falle der Verwendung von H-Graphen als Aktionsplan kann der Quell-Ableitungsbaum direkt aus dem Aktionsplan konvertiert werden. Dies wirkt sich sehr günstig auf das Laufzeitverhalten aus.

Bestehen Alternativen bei der Zerlegung, können hierfür zwar Backtracking-Verfahren eingesetzt werden, ihr schlechtes Laufzeitverhalten macht sie jedoch für ein prozeßnahes Werkzeug ungeeignet. Für die Aktionsplanung auf Werkstattebene sind deterministische Quell-Grammatiken bzw. Verfahren /95/ wesentlich besser geeignet.

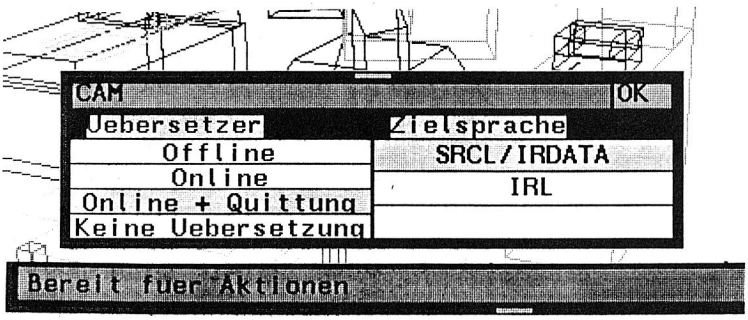


Abb. 4.2: Exemplarische Einstellung des Compilers (Bildschirmabzug)

Die Einstellung des Compilers des entwickelten Prototypen geschieht in der abgebildeten Benutzeroberfläche in Abb. 4.2 (CAM-Pull-Down-Fenster). Die linke Spalte ("Übersetzter") läßt nachfolgende Auswahl zu; der praktische Einsatz eines derartigen Compilers läßt sich daran verdeutlichen:

1. Offline: Der Übersetzer ist aktiviert. Der aus den H-Graphen generierte NC-Code wird im Filesystem des Rechners abgelegt und nicht direkt an die Zielsteuerungen weitergegeben. Eventuell erfolgt zunächst Simulation und Test mit den Interpretern für expliziten CNC-Code ('Vorlauf'-Betrieb) und anschließender DNC-Übertragung im Paket.

2. Online + Quittung: Der Übersetzer ist aktiviert. Der erzeugte Code steuert, nach einer Sicherheitsrückfrage, die reale Anlage (Interaktives Arbeiten, 'Spiegel'-Betrieb).

3. Online: Wie 2., jedoch wird der Code direkt an die Steuerung hinuntertransformiert. Sofortige Ausführung impliziter Befehle.

4. Keine Übersetzung: Der Übersetzer ist deaktiviert. Die generierten Aktionspläne werden nicht übersetzt.

In der rechten Spalte wird die zu erzeugende Sprache (sofern nicht durch den Aktionsplaner anhand des Umweltmodells festgelegt) ausgewählt. Es sind natürlich nur Zielsprachen möglich, deren Grammatik in der Datenbank auch vorhanden ist. Die gewünschte NC-Zielsprache (sofern in DB) kann dann zur Laufzeit des Aktionsplanungswerkzeuges gewählt werden (Abb. 4.2). Punkt 3 ist z.B. für implizite Aktionen der CCD-Kamera (keine Bewegung des Gehäuses. s. Kap. 4.2.1) sinnvoll, da hier z.B. keine Kollisionsgefahr besteht und ohnenin große Sicherheitsabstände vorliegen.

Das Übersetzungsverfahren muß auf Seiten der Quellsprache auf leichte Erweiterbarkeit der Quell-Grammatik ausgerichtet sein (fortschreitende Einbringung von Fertigungstechnologie und Erweiterung des impliziten Befehlsumfanges), ohne daß konzeptionelle Änderungen am Übersetzer notwendig werden. Die Quellsprachen-Grammatik (H-Graph, entspricht dem Befehlsbaukasten impliziter Befehle nach Kap. 5.2.5) wird hierzu in der DB gespeichert.

Neben den technischen Daten der Betriebsmittel werden auch die, für die Code-Generierung impliziter Aktionen erforderlichen Sprachbeschreibungen der zu erzeugenden NC-Sprache in der Datenbank bereitgestellt. Die Produktionen, ebenso wie die Zielsprachen-Grammatiken (Festlegung, welche Sätze der Zielsprache zulässig sind) und Semantik-Routinen sind Bestandteil der Geräte-Datenbank (Kap. 5.2) und müssen über LAN aus ihr entnommen werden. Sie können zusammen mit dem wachsenden fertigungstechnologischen Wissen in maschinenlesbarer Form (vgl. Kap. 5.2.5 u. 5.3.3) mitgeliefert (nachgerüstet) werden ("Wissens-Instandhaltung" während des Einsatzes des Programmierwerkzeug).

Damit können Umfang und Inhalt der vom System beherrschbaren Fertigungsaktionen durch Erweiterung der Datenbasis erweitert bzw. gepflegt werden. Das Tool nach Kapitel 5.2.5 muß bei Erweiterung des Befehlsbaukastens auch die Zielsprachen-Grammatik ergänzen (=Seiteneffekt; meist nur auf Linke Seite).

Die Syntaxbeschreibung der NC-Sprache in Form von sog. **G-Code** (**G**rammatik-**Code**) /95/ ist gut für einen generalisierten Compiler und die Speicherung in einer Datenbank geeignet. Sie wurde für den Aktionsplan-Compiler der vorliegenden Arbeit verwandt. Sie erleichtert die automatisierte Umsetzung einer Zielsprachen-Gram-

matik (z.B. in BNF) in G-Code. Abb. 4.3 faßt die Arbeitsweise des Aktionplancompilers zusammen (vgl. /95/).

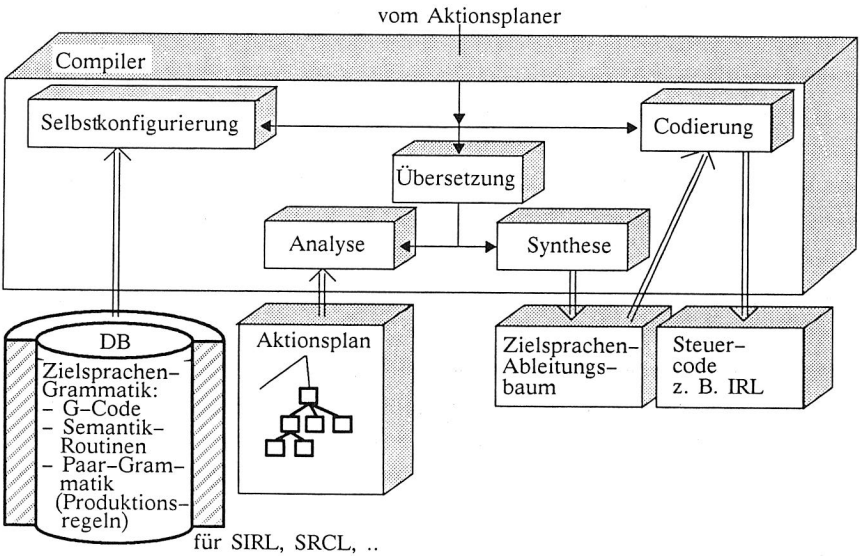


Abb. 4.3: Aufbau des Aktionsplan-Compilers für NC-Zielcode

Bei der Entwicklung eines generalisierten Aktionsplan-Compilers ist die Behandlung von Fernzusammenhängen besonders aufwendig. In Zielsprachen für Gerätesteuerungen lassen sich zwei Hauptarten unterscheiden, die der Aktionsplan-Compiler verarbeiten muß:

1.) Vorwärtsreferenz (Bezugnahme eines Terminalsymbols der Sprache auf ein anderes, im Programm **weit vor** ihm stehendes Terminal)
2.) Rückwärtsrefernz (Bezugnahme eines Terminalsymbols der Sprache auf ein anderes, im Programm **weit nach** ihm stehendes Terminal)

Beispiele sind Unterprogrammkonstrukte (z.B. Sensorfunktionen) und Funktions- bzw. Variablendeklarationen (s. Abb. 4.4). In der impliziten NC-Sprachen-Generierung (z.B. SRCL) kann eine Funktion oder Unterprogramm (UP) oft erst nach der aufrufenden Programmeinheit (PE) (=Baustein, HP, UP oder Funktion) definiert werden. Bei der Syntax-Synthese der Steuerungssprache wird dazu ein Aufruf er-

zeugt, und erst nach der Generierung der restlichen Programmeinheit darf die aufgerufene Funktion bzw. das Unterprogramm definiert werden.

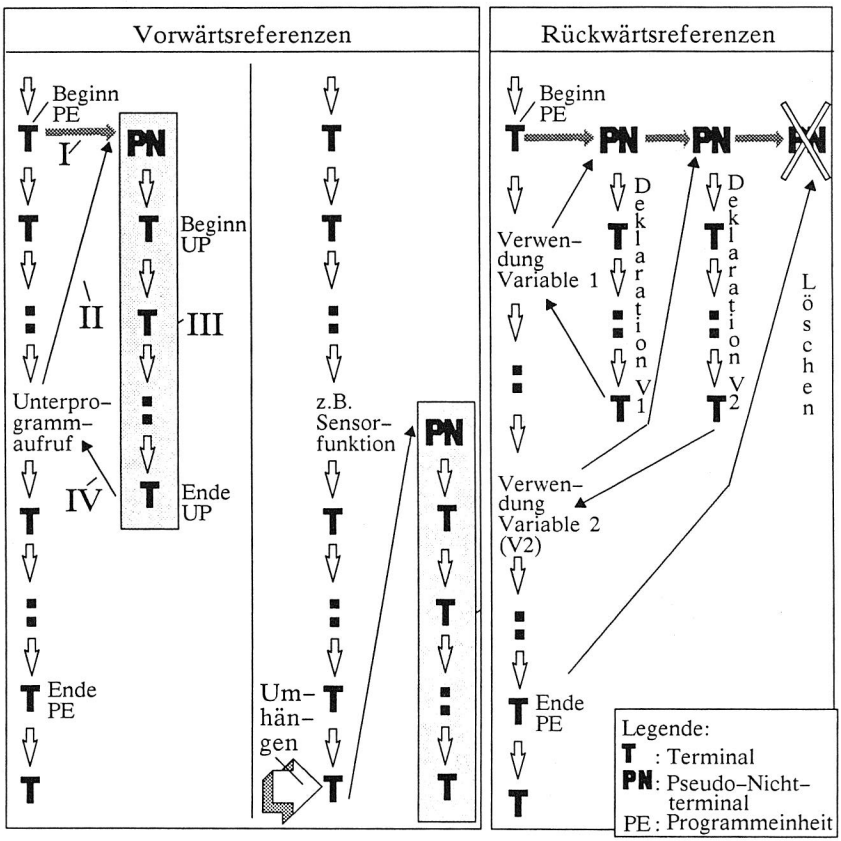


Abb. 4.4: Vorwärts- und Rückwärtsreferenzen in der Zielsprache

Zur Lösung dieses Problems sind in der Aktionsplanung Compiler-Techniken wie der Durchlauf mehrerer Compiler-Pässe ungeeignet. Bei der Syntax-Synthese, als Bestandteil der Übersetzung des Aktionsplanes, wird deshalb bei Beginn einer neuen Programmeinheit ein sog. Pseudo-Nichtterminal (PN) in den Zielsprachen-Ableitungsbaum eingehängt (I) und anschließend bei dessen Vater mit Terminal-Eintragungen fortgesetzt (Abb. 4.4, links). Tritt innerhalb der aktuellen Programmeinheit ein UP-Aufruf auf, so kehrt der Compiler zum Pseudo-Nichtterminal zurück (II)

und hängt darunter die Terminale dieser Unterprogramm-Definition (III). Anschließend kehrt er an die Stelle nach dem Aufruf zurück (IV). Am Ende der Programmeinheit wird der gesamte Unterbaum unter dem Pseudo-Nichtterminal im Zielsprachen-Ableitungsbaum an die aktuelle Position umgehängt und eine Programmeinheit gebildet (analog für mehrere Unterprogramm-Aufrufe in der Programmiereinheit und Unterprogramm-Aufrufe in Unterprogrammen, etc.).

Rückwärtsreferenzen sind weniger problematisch, da das Umhängen von Unterbäumen entfällt.

Neue Generationen von RC-Sprachen sind deklarativ (z.B. IRL). Für die Deklaration von Variablen (z.B. Datentyp: "Kollisionsfreier Pfad" = Liste von berechneten Soll-TCP-Koordinatensystemen) wird wieder ein Pseudo-Nichtterminal am Beginn einer Programmeinheit in den Zielsprachen-Ableitungsbaum eingehängt, unter den dann die Variablen-Teilbäume gehängt werden (Abb. 4.4, rechts).

4.2 Aktionsintegrierte Sensorsignalverarbeitung

Implizite Anlagenprogrammierung muß optional impliziten Sensoreinsatz beinhalten. Der Spezialist Sensoreinsatzplaner koordiniert diese Aufgaben. Ein wichtiges Einsatzbeispiel für die implizite Sensorprogrammierung stellt ein Bildverarbeitungssystem mit ladungsgekoppeltem Sensor (CCD) dar, mit dem das rechnerinterne Geometriemodell mit der realen Fertigungszelle abgeglichen werden kann (=Beseitigung systematischer Fehleranteile, vgl. Kap. 7.). Das Kamera-Gehäuse kann hierzu vom Industrieroboter gegriffen und geeignet positioniert werden (s. Abb. 4.5).

Geometrische Abweichungen führen häufig zum Scheitern der mit Offline-Systemen erstellten NC-Programme. Ziel ist der Abgleich des CAD-Modells mit der realen Fertigungszelle bezüglich absoluter und relativer Lage und Orientierung der Starrkörper; nicht die Vermessung von Starrkörpern selbst. Die Starrkörper sind meist hinreichend genau modelliert. Ebenso wird hier eine kinematische Kalibrierung, z.B. nach /28/, nicht behandelt.

4.2.1 Impliziter Toleranzabgleich durch mitbewegte CCD-Kamera

Der CCD-Sensor wandelt die Projektion einer dreidimensionalen Szene in ein Rasterbild mit verschiedenen Graustufen. Die anschließende Reduktion des Grautonbildes auf Merkmale hängt von der konkreten Aufgabe ab und muß deshalb je nach aktueller Situation geplant bzw. programmiert werden.

Anhand des Anlagenmodells kann der Sensoreinsatzplaner Parameter für das Bildverarbeitungssystem vorgeben und z.B. vorhandene Bilddatenverarbeitungsroutinen (z.B. Filter) damit starten. Er definiert weiterhin, welche Ausschnitte des Werkstückes oder des Betriebsmittels aufgenommen oder vermessen werden sollen.

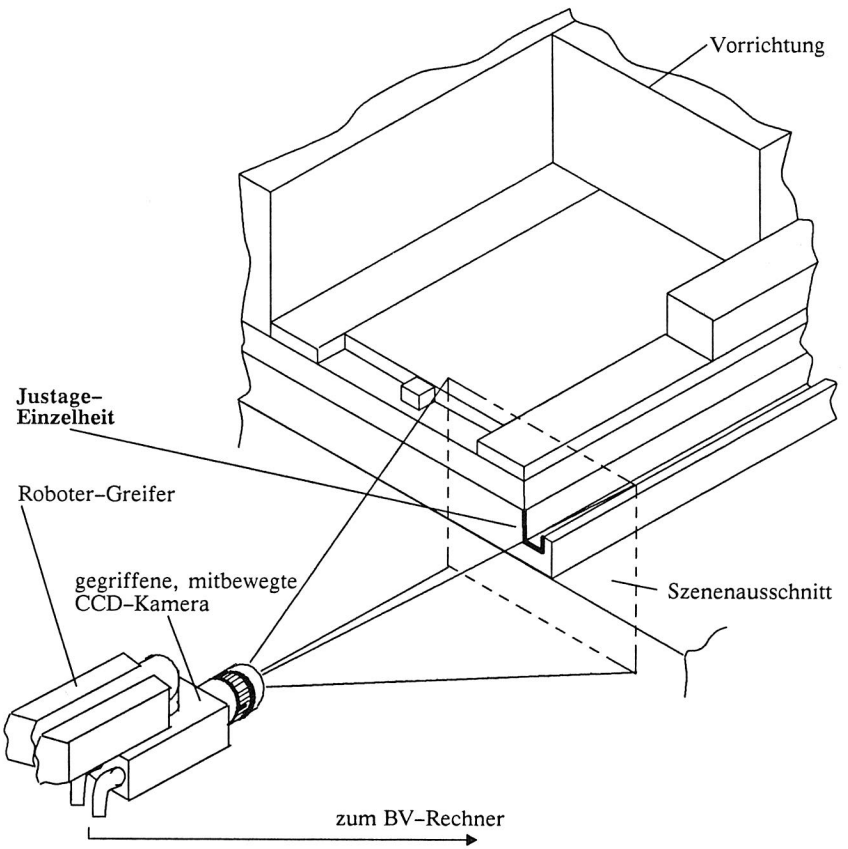


Abb. 4.5: Gegriffene CCD-Kamera zum einmaligen Abgleich mit Hilfe von Justage-Einzelheiten

Der Output des Sensoreinsatzplaners muß hierzu beinhalten:

1. Soll-Position und Orientierungen der mitbewegten CCD-Kamera, bezogen auf die abzugleichenden Körper.
2. Vorgaben für die Bildvor- und Bildnachverarbeitung der aufgenommenen Szenen.

Beide Vorgaben 1 und 2 sind voneinander abhängig. Sie müssen vom Sensoreinsatzplaner koordiniert bestimmt werden.

In der Praxis treten Verkippungen und Verschiebungen von Vorrichtungen, Zuführeinrichtungen, Werkstücken, etc. gegenüber dem idealen CAD-Modell der Aktionsplanung in der Größenordnung von ≤ 30 mm bzw. $\leq 3°$ auf. Unter diesen Annahmen berechnet der Sensoreinsatzplaner Sollposition und -orientierung der CCD-Kamera **zunächst** anhand des idealen, noch nicht korrigierten Anlagenmodells. Werden diese Sollwerte in der Zelle real angefahren, müssen sich die Erkennungs- bzw Justagefeatures (s. Abb. 4.5) im Bildausschnitt der CCD-Kamera befinden. Der Bildausschnitt und damit der Fangbereich kann erweitert werden durch Abrücken des Industrieroboters (+ gegriffenes Kamera-Gehäuse) vom idealen Aufnahmepunkt oder durch Einsatz eines motorischen ZOOM-Objektives. Da Aufnahmeformat und Szenenauflösung konstant bleiben, nimmt die Meßgenauigkeit ab. Umgekehrt läßt sich bei gerade noch zulässiger Verkleinerung des Bildausschnittes die Meßgenauigkeit erhöhen.

Zur Vermeidung von Verzerrungs- und Parallaxefehlern müssen Orientierung und Position der CCD-Kamera so berechnet und durch den Industrieroboter eingestellt werden, daß sich:

a) die optische Achse der CCD-Kamera senkrecht zur Schnittebene befindet, in der die 2D-Projektionen der Justagefeatures liegen, und
b) der Flächenschwerpunkt der Justagefeatures in etwa mit dem Durchstoßpunkt zwischen optischer Achse der Kamera und Schnittebene zusammenfällt (Abb. 4.5).

Justagefeatures sind markante geometrische Grundelemente wie z.B. Stifte, Bohrungen, Kanten, Linienendpunkte und Flächengrenzen. Deren Auswahl und Kombination trifft der Sensoreinsatzplaner anhand des Umweltmodells und folgender Kriterien:

- Eignung der Justagefeatures, d.h. beispielsweise Kontrast und Blob-Größe, für die digitale Bildverarbeitung. Ein Blob ist eine isolierte, zusammenhängende Gruppierung von Bildpunkten mit ähnlichen Grauwerten (mathematisch zusammenhängende Region).
- Extraktion "aussagekräftiger" Justagefeatures aus dem Umweltmodell (Abb.4.6).

Ein rotationssymmetrisches Muster (z.B. einer Bohrung) kann alleine nur die zweidimensionale x-y-Lage festlegen (Abb. 4.6,a). In Kombination mit z.B. rotationsvarianten Linien und Kanten kann die vollständige Lage und Orientierung in der Ebene bestimmt werden (Abb. 4.6,b).

Die Genauigkeit des Rotationsabgleiches läßt sich durch Auswahl von maximal voneinander entfernten Justagefeatures über die Positioniergenauigkeit des Industrieroboters (z.B. Größenordnung $\leq 0{,}1$ mm für den zu Experimenten eingesetzten R2) hinaus steigern (Negativbeispiel: Abb. 4.6, c).

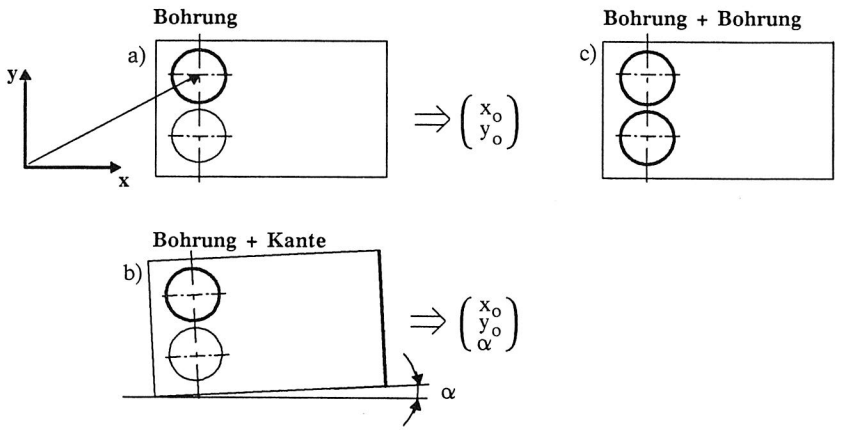


Abb. 4.6: Beispiele von Justagefeatures

Durch den Zugriff des Sensoreinsatzplaners auf das rechnerinterne Anlagenmodell lassen sich Szenenaufnahmen von, z.B. senkrecht aufeinander stehenden Ansichten des zu justierenden Objektes verarbeiten. Ausgehend von mehreren Bildaufnahmen zweidimensionaler geometrischer Justagefeatures eines Körpers können diese zu räumlichen Korrekturwerten zusammengesetzt werden.

Meist genügt es jedoch Position und Orientierung in einer Schnittebene zu bestimmen, da die Werte für angrenzende Objekte (z.B. Flächenkontakt der Vorrichtung mit liegt-auf-Beziehung zu Tisch) bekannt sind oder bereits bestimmt wurden (Vererbungsmechanismen des Common-Sense-Wächters, Kap. 3.2.5).

Sollposition und -orientierung des CCD-Kamera-Gehäuses müssen, durch die, während des Kalibrierungsvorganges ermittelten Transformationen in Gl. 4.4 (s.u.), in Sollwerte des Industrieroboters-TCPs umgerechnet werden.

4.2.2 Bilddatenverarbeitung

Bei der Auflösung eines impliziten Befehls

"Abgleich Objekt X"

muß die CCD-Kamera in der Fertigungszelle gegriffen, verfahren und nach Vorgabe des Sensoreinsatzplaners deckungsgleich zu einem berechneten Aufnahmeframe positioniert werden. Kollisionsfreie Pfade, Nahpunktframe, Aufnahmeframe, etc. , der dabei implizit beteiligten Subaktionen Greifen, Grobbewegung, etc. werden analog zu Kap. 3 berechnet.

Der Sensoreinsatzplaner und der Aktionsplan-Compiler (Kap. 4.1) gibt Vision-Parameter für bereits im Bildrechner vorhandene Operatoren vor (Auswahl von BV-Routinen), bzw. veranlaßt die Compilierung der im Aktionsplan enthaltenen Sensoraktionen ("Aufnahme auslösen", "Grautonbild binärisieren", ..) zu BV-Steuercode (Abb. 4.7). Beispiele für BV-Operatoren der Bildvor- und Bildnachverarbeitung sind Gradientenoperatoren, Grauwert-Histogramm-Schwellen, Gauß- und Laplace-Operatoren (Glättungsverfahren), Hough-Transformationen, Werfen und Auswerten von Rulern (vgl. Kap. 3.1.4), Medianfilter /36/, sowie weitere Filter und Verfahren zur Bildsegmentierung und Datenreduktion des Grauwertbildes.

Im Rahmen der vorliegenden Arbeit wurde der Sensoreinsatzplaner exemplarisch realisiert und für den Abgleich des rechnerinternen Geometriemodells (einfaches 3D-Drahtmodell) mit der realen Zellengeometrie eingesetzt. Während des Justagevorganges wird das abzugleichende Objekt am Bildschirm durch Aufblinken markiert.

Der Abgleich des CAD-Modells erfolgt nach Gl. (4.4) mit automatischer Größenanpassung je nach Abstand zwischen Kamera-Objektiv und Körper zum Zeitpunkt der Aufnahme. Nach erfolgter Aktualisierung des Umweltmodells kann die Kamera wieder abgelegt werden (impliziter Befehl "Ablegen"). Die CCD-Kamera muß nicht permanent in das IR-Werkzeug integriert oder am IR befestigt werden. Dies ist wichtig für die Montage miniaturisierter, feinwerktechnischer Produkte. Platz- bzw. Kollisionsprobleme werden vermieden. Das kostenintensive Bildverarbeitungssystem braucht nicht in der Fertigungszelle zu verbleiben - es kann nach und zwischen den

Justagevorgängen für weitere Fertigungszellen bzw. -aufgaben genutzt werden. Die aufwendige Online-Sensorik (Kosten!) kann auf ein Minimum beschränkt werden.

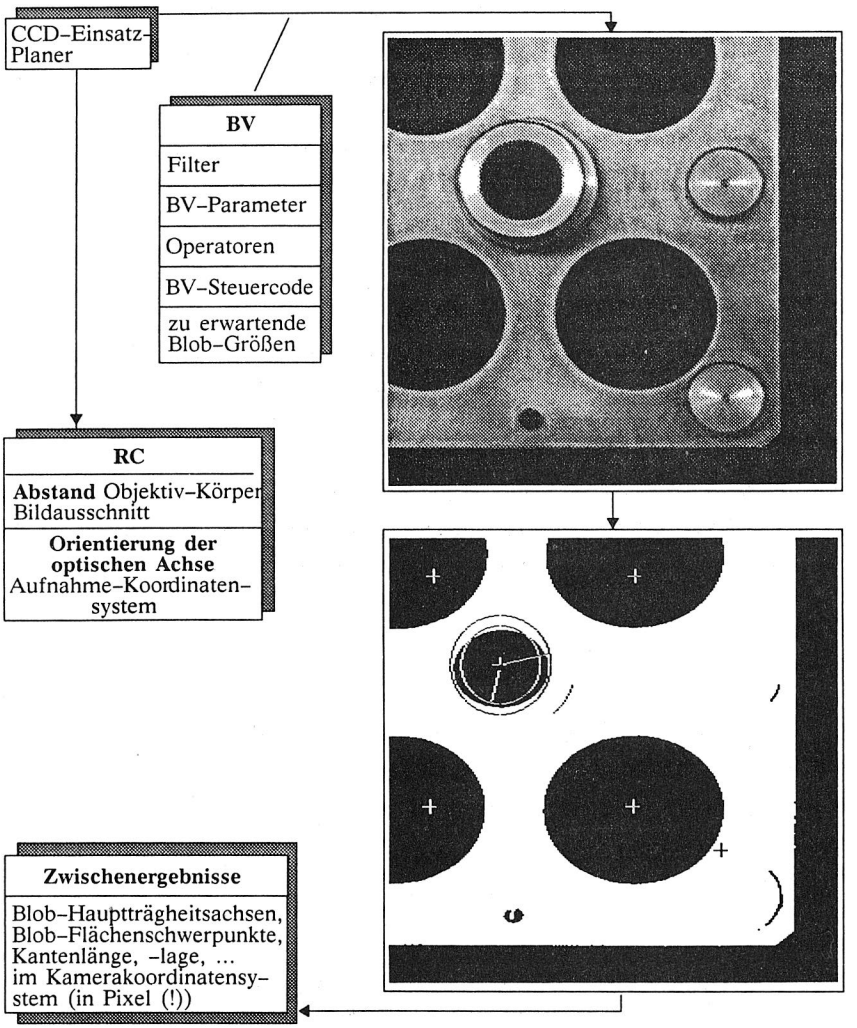


Abb. 4.7: Graubild-Binärisierung und Ermittlung von Justagefeature-Schwerpunkten (BV-Ausdrucke)

In /36/ wird ein Verfahren zur photogrammetrischen Rekonstruktion von räumlichen Daten aus CCD-Aufnahmen mit raumfesten Kameras vorgestellt. Durch die implizite Justage, basierend auf ein mitlaufendes Anlagenmodell und einer mitbewegten, vom IR greifbaren Kamera, kann der Aufwand wesentlich vermindert werden.

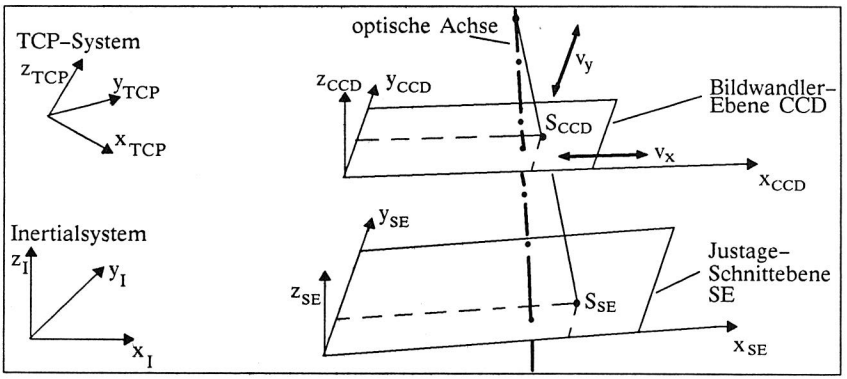
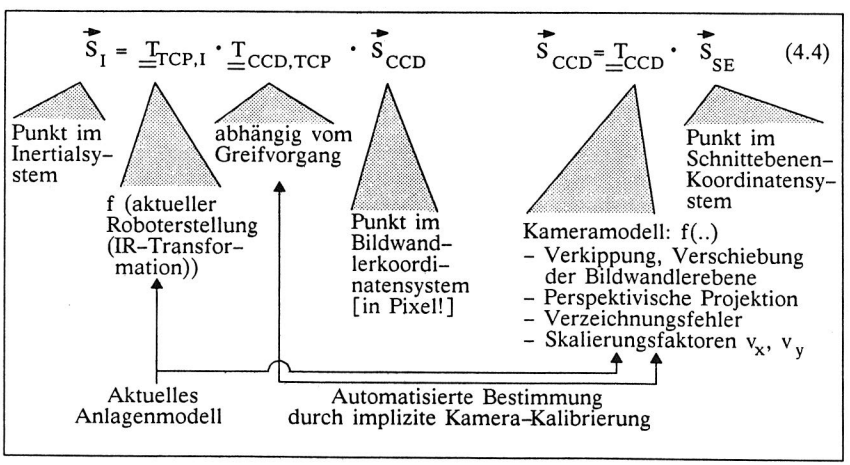


Abb. 4.8: Vereinfachte geometrische Optik einer CCD-Kamera (vgl. /36/)

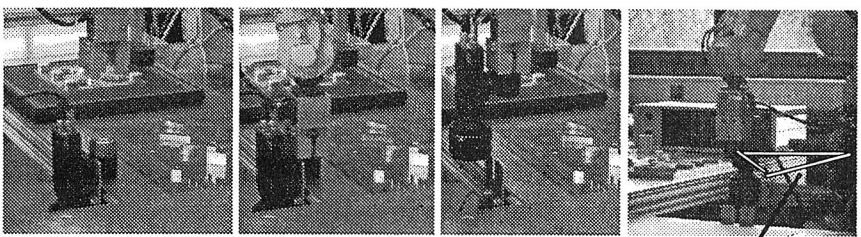
Für die implizite Aktionsplanung wurde im ersten Ansatz ein an /36/ angelehntes, jedoch stark vereinfachtes Kameramodell verwendet (Abb. 4.8). Die gesamte Umrechnung zwischen Bildwandler- und Inertialsystem erfolgt nach GL. (4.4).



Durch unterschiedliche Auflösung in x- und y-Richtung der Bildwandlerchips müssen die in erster Näherung konstanten Verzerrungsfaktoren v_x und v_y (zur Umrechnung

von Pixel in mm) (≙ Skalierungsfaktoren in homogenen Koordinaten, vgl. Kap. 3.1.2) automatisch bestimmt werden. Ebenso die Transformation $\underline{T}_{CCD,TCP}$ vom Bildwandlerkoordinatensystem zum TCP-System, die durch Ungenauigkeiten beim Greifvorgang des Kameragehäuses bestimmt wird.

Mit dem mitlaufenden Anlagenmodell lassen sich Mehrdeutigkeiten auflösen, die dadurch entstehen, daß mehrere Punkte S_{SE} mit unterschiedlichen Abständen zwischen den Ebenen CCD und SE den gleichen Bildpunkt erzeugen (vgl. /36/).



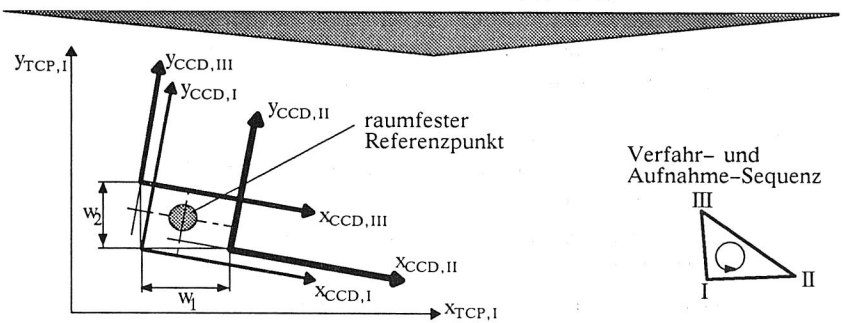


Abb. 4.9: Implizites Greifen (1.–3.) und Kalibrieren (4.) der CCD-Kamera

Versuche zeigten, daß z.B. aufgrund von Ungenauigkeiten während des Greifvorgangs des CCD-Gehäuses (Vibrationen, geringfügige Verschiebungen und Verdrehungen in der Halterung des CCD-Gehäuses) zunächst ein Kalibrierungsverfahren durchgeführt

werden muß. Es kann mit Hilfe der impliziten Verfahren automatisiert werden. Es umfaßt die Berechnung der Transformationen des Kamera-Koordinatensystems (in Pixel) relativ zum TCP-Koordinatensystem (Berücksichtigung von Auskragung, etc.) sowie die Ermittlung von Kamera-Parametern. Eingeleitet wird der Vorgang durch den impliziten Befehl "Justiere Kamera". Er repräsentiert einen H-Graphen aus mehreren Bewegungs- und Sensoraktionen.

Die Kalibrierungsprozedur umfaßt mehrere automatische Bildaufnahmen (I, II, III in Abb. 4.9). Zwischen den Bildaufnahmen wird das Kamera-Gehäuse senkrecht über einen Referenzpunkt durch den Industrieroboter verfahren. Die anzufahrenden Aufnahmeframes werden implizit bestimmt; beteiligte implizite Subaktionen (z.B. Grobbewegung) werden nach Kap. 3 geplant und ausgeführt. Aus der Kenntnis der Verfahrwege des Industrieroboters w_1, w_2 nach Abb. 4.9, und der Auswertung der BV-Szenen lassen sich $\underline{T}_{CCD,TCP}$ (Translation, Rotation) und $\underline{T}_{CCD}$ (Verzerrungsfaktoren) mit ausreichender Genauigkeit bestimmen (Abb. 4.9). Die zuletzt ermittelten Transformationen werden als Startwerte für die nachfolgenden Kalibrierungsvorgänge gespeichert.

4.3 Roboterzellen-Anbindung

4.3.1 DNC-Kommunikation

Die prototypische Realisierung des impliziten Aktionsplanungs- und -steuerungswerkzeuges erfordert vier Arten der Kommunikation (vgl. Abb. 4.10):

1. Konstruktion/AV-Ebenen-interne Kommunikation (Kapitel 5.3)
2. Kommunikation zwischen den Werkzeugen der Konstruktion/AV-Ebene und dem impliziten Aktionsplanungswerkzeug auf Werkstattebene (verteilte DB-Zugriffe)
3. Kommunikation zwischen dem impliziten Programmsystem und den Gerätesteuerungen (RC, BV-Rechner, ..)
4. Kommunikation auf Feldebene (Feldbusse, Realzeitbusse)

Als gemeinsame Basis der in der vorliegenden Arbeit erforderlichen Kommunikationsumgebung kann die ISO/OSI (Open Systems Interconnection) bzw. ISO/DIN-Referenzdefinition mit sieben funktionell abgeschlossenen Protokollschichten dienen.

Kommunikationssoftware mußte teilweise eigenentwickelt werden, da für die implizite Geräteprogrammierung und implizite Sensorintegration Eingriffe in die Kommunika-

tion benötigt werden, die abgeschlossene, kommerzielle Kommunikationspakete nicht zulassen. Sie sollten, soweit möglich, nach ISO/OSI strukturiert werden.

Ein Beispiel ist die RC-DNC-Schnittstelle zur Robotersteuerung RCM3. Für sie mußten die Telegrammdienste- (Transport-) und LSV2- (Sicherungs-) Schicht (nach DIN 66019, DIN 66267), sowie der Anschluß an die RS232-Schnittstelle (9600 Baud) (Bitübertragungsschicht) realisiert werden. Dienste der Vermittlungsschicht können aufgrund der Punkt-zu-Punkt-Verbindung entfallen.

Die DNC-Schnittstelle wurde in der Programmiersprache C mit Hilfe von Telegramm-Ringpuffern und Interruptsteuerung realisiert. Es wurde nur eine Teilmenge der DNC-Telegramme implementiert. Mit dem Telegramm-Protokoll werden Einzeltelegramme (z.B. Geschwindigkeitsoverrides) oder logische Telegramm-Sequenzen von verketteten Telegrammen übertragen, sowie die Rahmen-Einbettung der Nutzdaten festgelegt. Telegramme von der Robotersteuerung zum impliziten System werden entweder der Anwenderschicht (Aktionsplaner, bzw. Spezialist DNC-Kommunikator; s. Kap. 2.2.2) sofort mitgeteilt (z.B. IR-Betriebszustand, Alarm-Telegramme) oder in einen Ringpuffer eingegliedert und in der Regel von der Anwenderschicht bereits erwartet. Analoges gilt für Sensortelegramme zur Kommunikation mit dem BV-Rechner. Dieses Kommunikationskonzept erlaubt den Anschluß mehrerer, mit dem Zellen-/Werkstattrechner verbundener Geräte der Fertigungszelle.

In der Erprobungs- und Inbetriebnahmephase des impliziten Systems wurden Aufbau und Test durch die Hilfe moderner Meßtechnik (Protokollanalyzer) erleichtert. Zusätzlich wurden Testprogramme entwickelt, die die reale Gerätesteuerung aus der Sicht der oberen ISO/OSI-Schichten (Anwender-, Telegramm- und Sicherungsschicht) imitieren.

Zukünftig werden genormte Kommunikationsschnittstellen, wie z.B. das standardisierte Protokoll MMS (Manufacturing Message Specification; ISO 9506,1,2) gerätespezifische Protokolle, wie sie in dieser Arbeit noch eingesetzt wurden, ersetzen. Durch 'MMS-fähige' Geräte (und Zusatzbaugruppen) wird die Kommunikation des Aktionsplanungs- und Steuerungswerkzeuges mit den unterschiedlichen Ziel-Betriebsmitteln wesentlich erleichtert (Alarmbehandlung, Sensormeßwerte, ..). Es ist deshalb zu erwarten, daß die implizite Geräteprogrammierung, aufgrund der vereinheitlichten Protokolle, in der Praxis noch effektiver eingesetzt werden kann.

4.3.2 Versuchsaufbau und Versuchsdurchführung

Für Entwicklung, Experimente und Tests der, in dieser Arbeit entstandenen Werkzeuge, wurde ein Versuchsstand nach Abb. 4.10 aufgebaut:

Abb. 4.10: Implizite Geräteprogrammierung einer Roboterarbeitszelle zur Montage feinwerktechnischer Komponenten (links: aktionsintegrierter CCD-Sensor zum impliziten Abgleich des rechnerinternen Anlagenmodells mit der realen Zellengeometrie)

Die gesamte CAE/CAM-Verfahrenskette besitzt folgende Komponenten (Abb. 4.11):

Montagezelle:

- 6-achs-Gelenkroboter manutec r2
- Robotersteuerung RCM 3 (mit analog/digital Sensor-BG und DNC-Baugruppe)
- Speicherprogrammierbare Steuerung:
  S5 115 U mit Kommunikationsprozessor CP 525 und mehreren I/O-Baugruppen
- Grauwert Bildverarbeitungssystem Sirotec VPS:
  - 512 x 512 Pixel, 64 Graustufen
  - Mehrprozessorsystem mit Bildvorverarbeitungsfuntionen in Hardware und Bildrechner mit Signalprozessor (TMS 320), Bilddatenreduktion (z.B. Lauflängenkodierung von Grauwertbildern, Grauwerthistogramme) mit 8088/8087 und Steuerprozessor 80186

- CCD-Kamera mit 756 x 581 Pixel Auflösung (Objektiv 1:1,4/25). Eignung für Greifen und Mitbewegen durch Industrieroboter r2 (da: geringes Gewicht (ca. 200g), kleine Abmessungen (ca. 100 x 30 x 45), ausreichende Vibrationsfestigkeit); =Vorteile moderner Halbleiter-Bildwandlerchips).

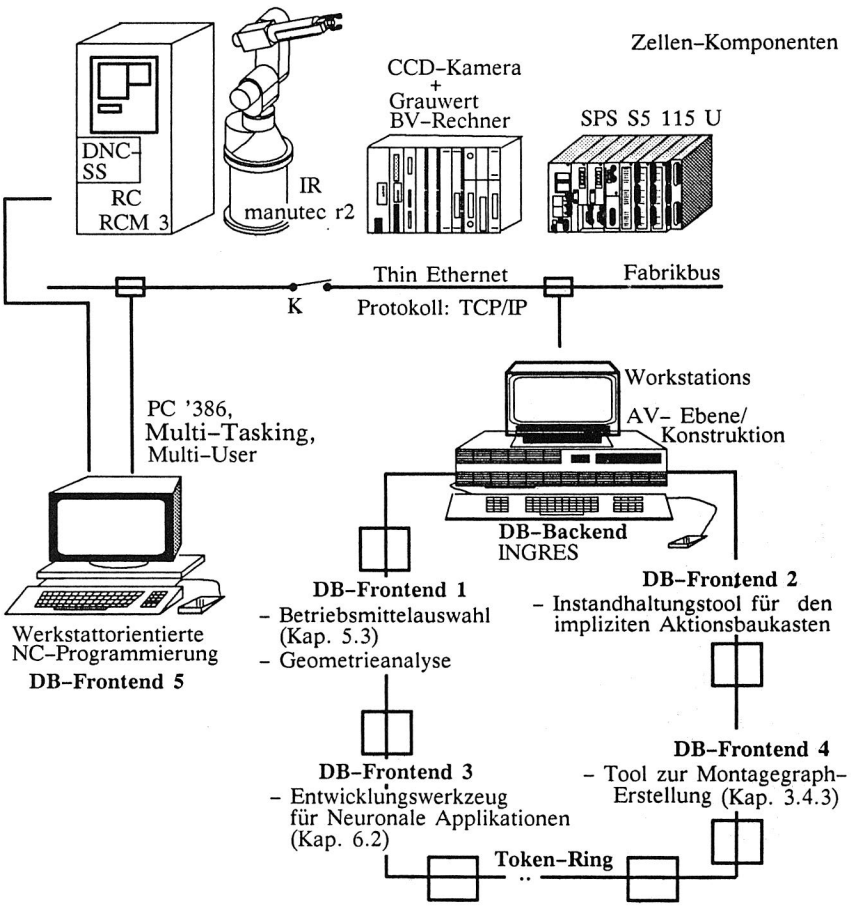


Abb. 4.11: Verteilte Werkzeuge zur impliziten Aktionsplanung und Geräteprogrammierung

Hardware-Umgebung auf Werkstattebene: Implizites Programmiersystem

- Standard Hardware unter XENIX System V-386 (Industrie-PC)

- 25-MHz-INTEL-80386-Prozessor mit 8-MHz-80387 Arithmetik-Coprozessor
- 8 MB RAM (vgl. "Rule Based NN") und Festplatte 260 MB; Ethernet mit TCP/IP

Hardware-Umgebung auf Ebene Konstruktion / AV:

- Graphik-Workstations mit lokalen Prozessoren in Token-Ring-Verbund (HP/Apollo unter AEGIS, UNIX Sys V und Unix Bsd4.2 im Wechsel oder gleichzeitig), z.B.
- 50-MHz-Motorola-68030-Prozessor und 68882-Gleitpunktprozessor
- 32 MB RAM und Festplatte 860 MB; Netzanschluß; Token-Ring und Thin-Ethernet

Dieser Gesamtaufbau ist Teil einer CIM-Modellfabrik und Basis aller Versuche im Rahmen dieser Arbeit.

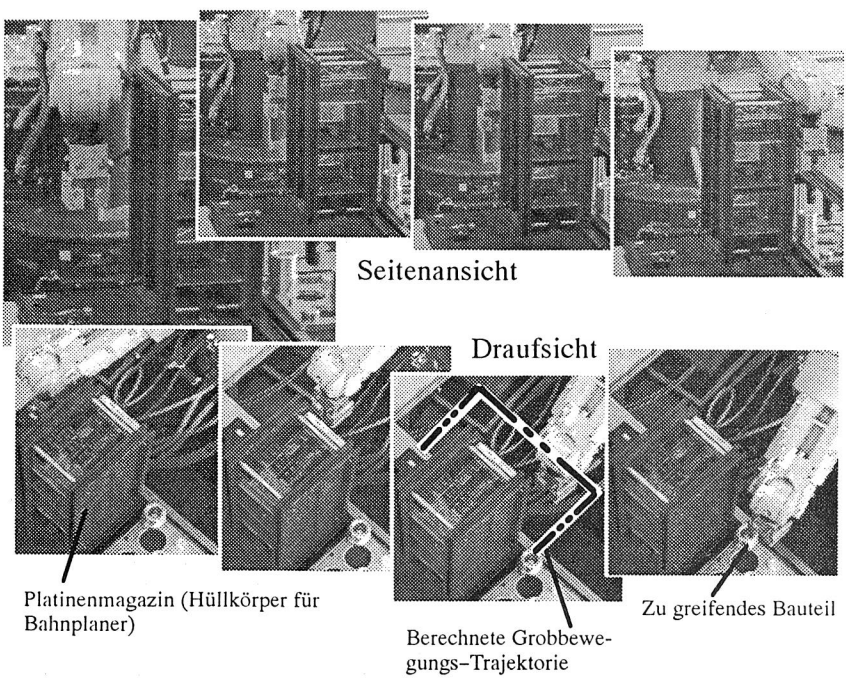


Abb. 4.12: Online-Ausführung einer impliziten Montageaufgabe mit aktiviertem Bahnplaner

Entgegen der logischen Darstellung in Abb. 4.11 wurde dem impliziten Aktionsplanungssystem ein Kommunikationsrechner nachgeschaltet, der eigenständig die Kommunikation zu den Gerätesteuerungen (RC, BV-Rechner) abwickelt (Protokoll, Programmkontrolle, Überwachung), und so das implizite System entlastet. Damit konnte die Reaktionszeit wesentlich reduziert werden, da zeitparallel zur Aktionsplanung die DNC-Kommunikation abgewickelt werden kann. Das Aktionsplanungs- und Steuerungssystem kann auch autark in der Werkstatt betrieben werden, indem es auf der letzten Kopie (Snapshot) der DB-Daten arbeitet (Verbindung K in Abb. 4.11 offen).

Die auf Werkstattebene verwendete Unix-Version XENIX System V 386 (SCO Inc.) kann nur bedingt für Echtzeitaufgaben verwendet werden (siehe Untersuchungen im Vergleich zu 'Standard-Unix' und Betriebssystemen mit 'harten' Echtzeitanforderungen in /127/). Für die prozeßnahe, implizite Aktionsplanung- und Steuerung wären jedoch weitergehende Echtzeitmechanismen wünschenswert.

Das Beispiel in Abb. 4.12 zeigt die Online-Ausführung eines impliziten Befehls. Die Genauigkeiten, der vom System im Sekundenbereich berechneten Positionen und Orientierungen sind an der realen Zelle nicht erreichbar. Die implizite Aktionsplanung umfaßt die Fügeplanung, Greifplanung, Compilierung in RC-Zielcode und die Aktualisierung des Umweltmodells. Die zugehörige Bahnplanung für die implizierte Grobbewegung dieses Beipiels ist in Abb. 3.15, Kap.3.2.3 in der Simulation dargestellt.

5. Technische Datenbank der CAE/CAM–Verfahrenskette

Damit implizite Fertigungsbefehle ausgeführt werden können, bedarf es eines umfangreichen Wissensvorrates. Im Gegensatz zu Daten, die aus der aktuellen Zellensituationen zur Laufzeit berechnet werden (s. Kap. 3.1), ist ein großer Teil des erforderlichen Basiswissens von langfristiger Natur. Dieser stabilere Teil sollte über eine technische **Datenbank** (**DB**) direkt von den verteilten Werkzeugen, einschließlich des prozeßnahen Aktionsplanungs– und Steuerungswerkzeuges gelesen werden. Das **Datenbankmanagementsystem** (**DBMS**) wird dadurch zum zentralen Bezugspunkt der CAD/CAM–Verfahrenskette.

Für das fertigungstechnologische Wissen bedeutet das zunächst die inhaltliche Definition einfacher Aktionsprimitive (z.B. "Schnelle Grobbewegung", "Abrücken"...) bis hin zu hochwertigen Montageanweisungen. Dieses Basiswissen muß möglichst anwendungsneutral modelliert und in einer Methodenbank gespeichert werden.

5.1 Datenbanktechnologie

Für die Verwaltung des gesamten Montagewissens, einschließlich der CAD–Modelle, ist der Einsatz von Datenbanktechnologie erforderlich. Bereits in der Betriebsmittelkonstruktion müssen die, beim Aufbau des Anlagenmodells anfallenden Daten, der Datenbank übergeben werden. Beispielsweise hat die Verschaltung innerhalb der Anlage wesentlichen Einfluß auf die Programmierung der Logikteile (z.B. durch SPS realisiert).

Im Gegensatz zu lokalen, anwendungsspezifischen Dateisystemen ist die logische Struktur der Daten in einem DB-System zentralisiert. Daten und Anwendungen, die auf sie zugreifen, sind getrennt (=Datenunabhängigkeit).

Die Datenunabhängigkeit wird durch Strukturierung (keine Teilung!) der Datenobjekte in externe Schemata (Programm-/ Benutzersicht), konzeptionelle Schemata (logische Gesamtsicht) und interne Schemata (physische Speichermedien, Speicherstrukturen und Zugriffspfade) erzielt (3-Schema-Konzept nach ANSI-SPARK /126/). Zentrales und wichtigstes Schema ist das konzeptionelle Schema, das den relevanten Ausschnitt der realen Welt beschreibt und meist mit dem näher an der Anwendung liegenden, semantischen Schema gleichgesetzt wird (z.B. **ER**-Modell (Entity-**R**elationship)).

Die Assoziationen zwischen den Objekten werden durch unterschiedlich komplexe Datentypen definiert (von 1:1, 1:n bis m:n; vgl. /131/). Jeder Objekt- und Beziehungstyp besitzt Attribute und identifizierbare Attribute (Primär-, Fremdschlüssel oder Attributkombinationen).

Alle semantischen Schemata dieser Arbeit werden mit Hilfe eines modifizierten Relationenmodells ("nach Normalisierung", /131/) dargestellt. Alle n:m-Beziehungen (+ nc:mc, ...) sind bereits durch Einführung von Beziehungstypen (=Hilfsentitätsmenge) auf 1:x Assoziationen reduziert (Voraussetzung für konventionelle, relationale DB).

Nach dem Entwurf des semantischen Schemas erfolgt die Implementierung (genauer: Datendefinition mit einer **D**ata **D**efinition **L**anguage (DDL)). Die anschließende Manipulation erfolgt über eine DML (= **D**ata **M**anipulation **L**anguage), die in eine höhere (Wirts-)Programmiersprache (z.B. Pascal) eingebettet (**embedded**) sein kann und z.B. mit einem Precompiler übersetzt wird (**Cursor-Konzept** /20/). Dieses Vorgehen ist relativ unflexibel. Der eigentliche Betrieb der Datenbank, kann frei (d.h. über einen DML-Interpreter) oder 'embedded' erfolgen.

Die Datenbank benötigt mächtige Mechanismen zur Einhaltung der Integrität. Im Beispiel der Datensicherung sind dies Anlegen von Checkpoints und Transaktionsprotokollen sowie Recovery-Verfahren nach Fehlerfällen. Eine Transaktion ist eine (auch vom Anwender zusammenfaßbar) atomare Operation auf der Datenbank. Sie kann nur die Zustände "vollständig ausgeführt" oder "nicht ausgeführt" (0/1) annehmen. Durch Transaktionen wird die Datenbasis von einem konsistenten Zustand A in einen konsistenten Zustand B überführt. Konsistenzerhaltung bedeutet, sicherzustellen, daß die Datenbankfüllung mit dem semantischen Schema zu jedem Zeitpunkt

verträglich ist (Synchronisation sich zeitlich überlappender Operationen, Widerspruchsfreiheit, Ausschluß von "unmöglichen", sinnlosen oder verbotenen Zuständen).

Die physische Speicherung (physisches Schema) ist besonders für die prozeßnahe, implizite Aktionsplanung und -steuerung zu beachten, da die Werkzeuge auf Teile der Datenbasis online zugreifen müssen (Laufzeit!).

Zur Laufzeitoptimierung sollte sich für Gruppen von Datenobjekten die Speicherart (vgl. /20/), z.B. aus

- Heap (sequentieller Zugriff)
- Isam (indexsequentieller Zugriff)
- Hash (Schlüsseltransformation)
- Btree (balancierter Mehrwegbaum)

erzwingen lassen.

Heap (ungeordnetes Ablegen auf einem Stapel) wird zur schnellsten Speicherart für das Anfügen großer Datenmengen (wichtig für das Nachliefern von Daten in maschinenlesbarer Form, Kap. 5.3.3). Suchzugriffe werden jedoch sehr langsam. Hash berechnet und verschlüsselt mit einem "Hash-Algorithmus" die Adressen der Datensätze und wird so zu einer schnellen Zugriffsmethode für Abfragen über exakte Schlüsselwerte. Ungünstig ist jedoch die Suche über Nicht-Schlüssel-Attribute, sprechende Schlüssel und Bereichsabfragen (häufig nach Kap. 5.3.2). Die "Isam"-Speicherart verwendet einen statischen Index und ist damit geeignet für Bereiche der Datenbasis, die sehr stabil sind (Fügetechnologie des impliziten Befehlsbaukastens, Kap. 5.2.5). "Btree" ist eine sehr dynamische Speicherstruktur und eignet sich deshalb für dynamische Datenobjekte (vgl. /97/).

5.1.1 Verteilte, relationale Datenbanken

Der Datenbankeinsatz im Rahmen dieser Arbeit basiert auf relationalen Datenbanken (RDBMS). Datenbanken, die dem Relationenkonzept nach Codd /17/ nicht entsprechen (Netzwerk-DB, hierarchische DB), sowie die im weiteren als bekannt vorausgesetzte Theorie des Relationenkonzeptes finden sich in /21/, /131/, /33/, /78/, /80/, /106/. Die in dieser Arbeit eingesetzten Entwurfsverfahren für konzeptionelle DB-Schemata sind in /97/ beschrieben.

Für die implizite Anlagenprogrammierung besitzen Pflege, Wartung und Wissensakquisition und -instandhaltung (Anpassung an den Stand der Technik) der fer-

tigungstechnischen Datenbasis besondere Bedeutung. Sie sollten in den Bereichen Konstruktion, Betriebsmittelkonstruktion und AV erfolgen, abgekoppelt z.B. von dem prozeßnahen Aktionsplanungswerkzeug, das bei der Abarbeitung eines impliziten Befehls auf Werkstattebene auf den aktuellsten, freigegebenen Stand dieses Basiswissens zugreift. Arbeitsplanerstellung und Anlagenkonstruktion muß parallel zum Betrieb des Programmiersystems möglich sein.

Die Gesamtheit der genannten Anforderungen erfordert mindestens verteilte Datenbankzugriffe über LAN (Fabrikbus). In der nächsten Stufe können nicht nur die Zugriffe verteilt werden, sondern die DB-Relationen selbst (=Verteilte Datenbanken). Letzteres ist wichtig für die autarke, implizite Geräteprogrammierung in der Werkstatt.

5.1.2 Multimedia-Datenbanken

Eine Wissensbasis für hochautomatisierte Fertigungsanlagen, Fertigungstechnologie und komplexe Gerätetechnik ist gekennzeichnet durch (vgl. /69/):

- lange Transaktionszeiten,
- komplexe konzeptionelle Schemata,
- komplexe referentielle Integritätsbedingungen,
- große Datenmengen,

sowie durch die starke Inhomogenität der zu verarbeitenden Datenmengen /97/:

- **Alphanumerische Daten:** Meßwerte, Zeiten, Namen, Primär-, Fremdschlüssel, ...
- **Texte:** Erfahrungsberichte, Inbetriebnahmeanleitungen, Hilfen bei Geräteausfällen, ...
- **Grafische Daten:** Abbildungen, Meßkurven, Zeichnungen, 2D-Ansichten von 3D-CAD-Modellen
- **Struktur-, Verschaltungsdaten** (nicht als Grafik!): Vorranggraphen, Schaltpläne, ...
- **Geometrische Daten:** CAD-Modelle (Volumenmodelle)
- **Raster-Szenen:** CCD-Aufnahmen (Grauwert- und binärisierte Bilder) Farbbilder, Bitmaps, ...

Diese Inhomogenität ist das Kennzeichen von Multimedia-Datenbankmanagementsystemen (MMDBMS). Sie verwalten erweiterte Datentypen (z.B. Image, Audio-Signal)

mit den zugehörigen Vergleichsoperationen. Durch Übertragung des Hypertext-Prinzips (tief referenzierte Objekte) lassen sich MMDBMS zu Hypermedia-Datenbanken erweitern (vgl. /70/).

Von Multimedia-DBMS existieren derzeit nur erste Ansätze (z.B. ORION, OMEGA; konzentrierte Darstellung in /70/). Sie sind eng an das Konzept der Objekt-orientierten Datenbanken (OODB) (vgl. Kap. 2.3) gebunden. Die Kennzeichen und Definitionen sind jedoch bislang noch nicht einheitlich formuliert worden /27/. Als Behelfslösung der vorliegenden Arbeit kann die Umsetzung der nach außen hin erscheinenden Datenobjekte auf interne Relationen (+ Trigger und referentielle Integritätsbedingungen) mit Hilfe der Meta-Datenbank (Kap. 5.2.4) erfolgen.

Zwischen relationaler Datenbanktechnik und objektorientierten Systemen (DB-z.B. KEE) bestehen starke Korrespondenzen. Dies sind (vgl. /43/): Trigger(referenzielle Integritäten)<->Methode, Relation,Tabelle,Entitätsmenge<->Unit, Attribut<->Slot,Instanzvariable, Attributwert<->Slotvalue,Wert der Instanzvariable, Tupel,Entity<->Member-Unit,Instanz, durch Joins verbundene Relation<->Struktur/Objekttyp,Klasse.

Besondere Anforderungen entstehen bei dem Entwurf des konzeptionellen Datenbankschemas (s. Kap. 5.2); vor allem für technische Gerätedaten. In der Praxis ist der Entwurf schwierig, da Funktions- und Leistungsumfang vergleichbarer Betriebsmittel häufig durch stark unterschiedliche, oft nur geringfügig überlappende Datenmengen charakterisiert werden (vgl. Abb. 5.1).

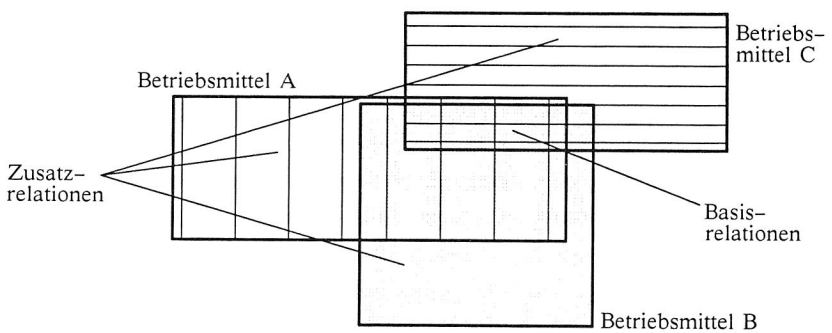


Abb. 5.1: Technische Daten von drei, in etwa leistungs- und funktionsgleichen Betriebsmitteln A, B, C (herstellerspezifisch und unterschiedliche, technologische Detaillösungen)

Spezifische Attributmengen resultieren daraus, daß Details eines Betriebsmittels in einer bestimmten Technologie gelöst wurden. Sie sind dann auch nur in diesem Kontext sinnvoll.

Beispielsweise wird für rotatorische NC-Achsen die Positioniergenauigkeit mit Nebenbedingungen angegeben (Vorspannkräfte, Schwerpunktlage und Trägheitsmomente der bewegten Massen, spezielle Meßvorschriften, ..). Für eine weitere Dreheinheit mit anderer Lagerungsart ist eine abweichende Attributmenge (gleiches Konstruktionselement, jedoch andere Tragfähigkeitsberechnung) von Bedeutung. Dieser Teil der technischen Spezifikationen erlaubt vorweg keine Bildung von Relationen mit allgemein gültiger Attributmenge. In der Praxis macht dies Kompromisse gegenüber klassischen Datenbankentwurfsverfahren erforderlich. Die Datenbasis muß flexibel entworfen werden (Schema-Änderung und -Erweiterung). Die nötige Flexibilität kann durch folgende Unterteilung des konzeptionellen Schemas erreicht werden (/96/):

Basisobjekte (-relationen):
Sie stellen den statischen Teil des Schemas mit einer gemeinsamen Attributmenge dar (analog komplexen Sachmerkmals-Leisten). Änderungen an dieser Stelle sind sehr aufwendig und von zentraler Bedeutung. Sie treten nur selten auf, z.B. wenn ein völlig neuer Gerätetyp eingeführt wird. Beispielsweise sei hier die Detaillierung der Kinematik von Bewegungsgeräten (Industrieroboter, modulare Handhabungsgeräte) genannt (vgl. Kap. 5.2.1).

Zusatzobjekte (-relationen):
Die Struktur dieser Datenobjekte muß hinsichtlich einfacher Erweiterbarkeit ausgelegt werden. Aus Gründen der Datenkonsistenz sollte dies unter der Kontrolle und Überwachung eines wissensbasierten Datenbank-Werkzeuges erfolgen. Dieses Werkzeug arbeitet auf der Metadatenbank (Kap. 5.2.3) und manipuliert bzw. legt die Datenobjekte an. Sie bestehen aus sich nicht überlappenden Attributmengen, die sich zum Zeitpunkt des Entwurfsprozesses der logischen Datenstruktur angeben lassen. Darüberhinaus müssen die, zum Zeitpunkt des DB-Entwurfsprozesses unbekannten Attributmengen (Beisp.: Diagnosedaten, Meßreihen, Fehlerfälle, etc.) gespeichert werden.

Erst im Anschluß an diese Definition wird die Datenbasis (soweit zu diesem Zeitpunkt möglich) normalisiert. Zur Beseitigung von Mutationsanomalien mit Normalisierungsverfahren (meist bis zur dritten Normalform (**3. NF**)) sei auf /131/,/106/ verwiesen.

5.2 Datenbankeinsatz für automatisierte Montageprozesse

Das für die implizite Geräteprogrammierung erforderliche Basiswissen läßt sich in drei Bereiche einteilen (Abb. 5.2).

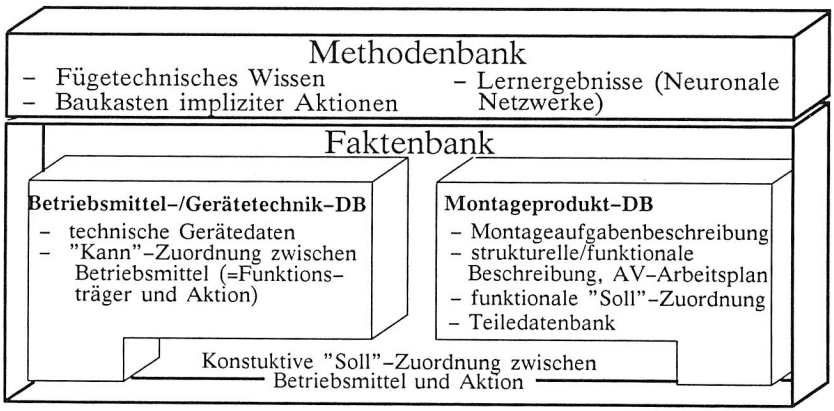


Abb. 5.2: Gruppierung der Montagewissensbasis für die implizite Geräteprogrammierung

Die Betriebsmitteldatenbank muß einen schnellen und umfassenden Überblick über die verfügbaren Baukastenelemente der Fertigungsanlage (Automatisierungskomponenten, Eigenkonstruktionen und bestehende Anlagen, bzw. deren Teilsysteme) mit den dazugehörigen technischen Daten ermöglichen (Anlagenkonstruktion und Aufbau des Anlagenmodells). In der Montageproduktdatenbank sind die Einzelteile der zu montierenden Erzeugnisse erfaßt. Geräte- und Teiledatenbank bilden zusammen die Faktenbank.

Die Methodenbank enthält fertigungstechnologisches Wissen (Fügetechnologie) für die implizite Aktionsplanung und -steuerung bzw. für die automatisierte Arbeitsplanung. Diese drei Bereiche der Datenbasis sind über eine zentrale Relation **fertigungszellen_aktion** verbunden (Abb. 5.3).

5.2.1 Fertigungsanlagen-DB

Dieser Teil der Datenbasis speichert:

A) Geräte/Betriebsmittel (Einzelkomponenten) mit:

- Zuordnung zwischen Gerät und Funktion bzw. ausführ**barer** Aktion ("Was kann ein Betriebsmittel")
- Simulationsmodelle (z.B. IR-Kinematik) und CAD-Modelle (B-Rep)
- Technische Daten
- Bitmaps und 2D-Projektionen

B) Fertigungsanlagen, die unter Verwendung von Elementen aus A) konstruiert wurden:

- Stückliste der Betriebsmittel in der Fertigungsanlage
- CAD-Zusammenstellungsmodelle
- Konstruktive Zuordnung zwischen Gerät und Funktion ("Für welche Funktion ist das Gerät vorgesehen?" (auszuführende Aktionen)).
- Szenenaufnahmen der Bildverarbeitung
- Verschaltungen

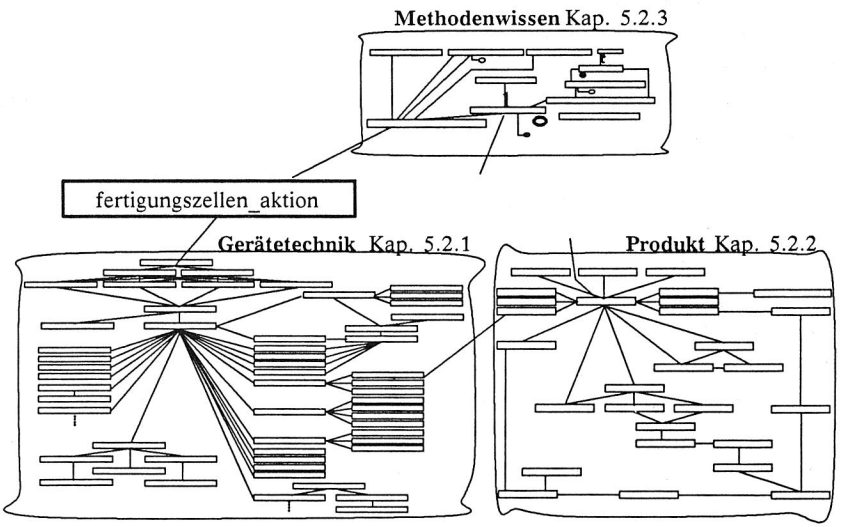


Abb. 5.3: Teilbereiche des semantischen DB-Gesamtschema mit 'fertigungszellen_aktion' im Zentrum 1)

Im Mittelpunkt des, in dieser Arbeit entwickelten Anlagen-Teilschemas (Abb. 5.4), steht das verallgemeinerte **Gerät** (=Betriebsmittel). Verallgemeinerung deshalb, da die

1) Meta-DB (und Beziehungen mit ihr) nicht dargestellt (s. Kap. 5.2.3)

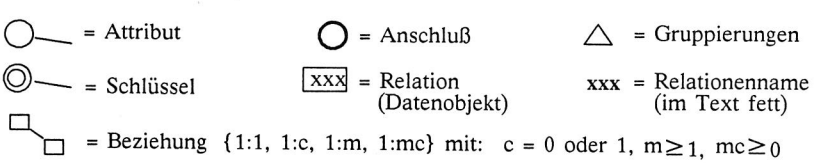
Komponente 'Gerät' von **Fördergerät, Segment** (s.u.) und **Sensorik** bis zu konstruierten (Teil)-Anlagen (=ausgeführte Teilsysteme) reicht. Zusammen mit der Relation **Gerät_info** dient sie als Verteiler und enthält selbst nur wenige technische Daten (Flexibilität nach Kap. 5.1.2). Geräte können von dem Datenbankwerkzeug nach Kap. 5.3.2 in frei definierbare **Geräteklassen** mit zulässigen und ausführbaren impliziten Aktionen (**Fertigungszellen_Aktion**) eingeteilt werden.

Die **Sensorik** besitzt im allgemeinen Fall mehrere **Aufnehmer** (**Sensortopologie**), die nach **physikalischen Prinzipien Meßgrößen** aufnehmen, diese verarbeiten und Nutzsignale über Schnittstellen (s.u.) bereitstellen. Organisatorische Daten von Werkstücken und Geräten werden in Relationen wie **entwickeln, anwenden, vertrieb, geschaefts_partner, adressen** etc. verwaltet.

Bereich △2 und vor allem Verbindungen △1 können während der Anlagenkonstruktion (Betriebsmittelauswahl) vom datenbankgestützten Konstruktionswerkzeug nach Kap. 5.3 in die Datenbank eingetragen werden (Zuordnung: Betriebsmittel<->Aktion). Die Beziehungen △1 , △2 und △3 müssen zur Laufzeit vom impliziten Aktionsplanungs- und Steuerungswerkzeug beim Auflösen eines impliziten Befehls gelesen werden.

Frei programmierbare Automatisierungsgeräte werden in einer Steuerungssprache programmiert. Der generalisierte Aktionsplan-Compiler (Kap. 4.1) benötigt die Zuordnung Grammatik<->Steuerungssprache für die Code-Generierung und für die Ansteuerung der einzelnen Zielsteuerungen. Für jede Steuerungssprache benötigt er weiterhin die zugehörige **NC-Sprachen-Grammatik** (z.B. Erzeugung eines Programmes zum Laden in den Bildverarbeitungsrechner). Für mächtige Steuerungssprachen (z.B. IRL) muß hier eine große Anzahl von Tupeln in den Relationen **G-Code, Terminale, Nicht-Terminale, Semantik-Routinen, Vorgriff** (nach Kap. 4.1) gespeichert werden (Bereich △7 in Abb. 5.4).

Für das gerätetechnische Teilschema in Abb. 5.4 (+ folgende) gilt die Legende:



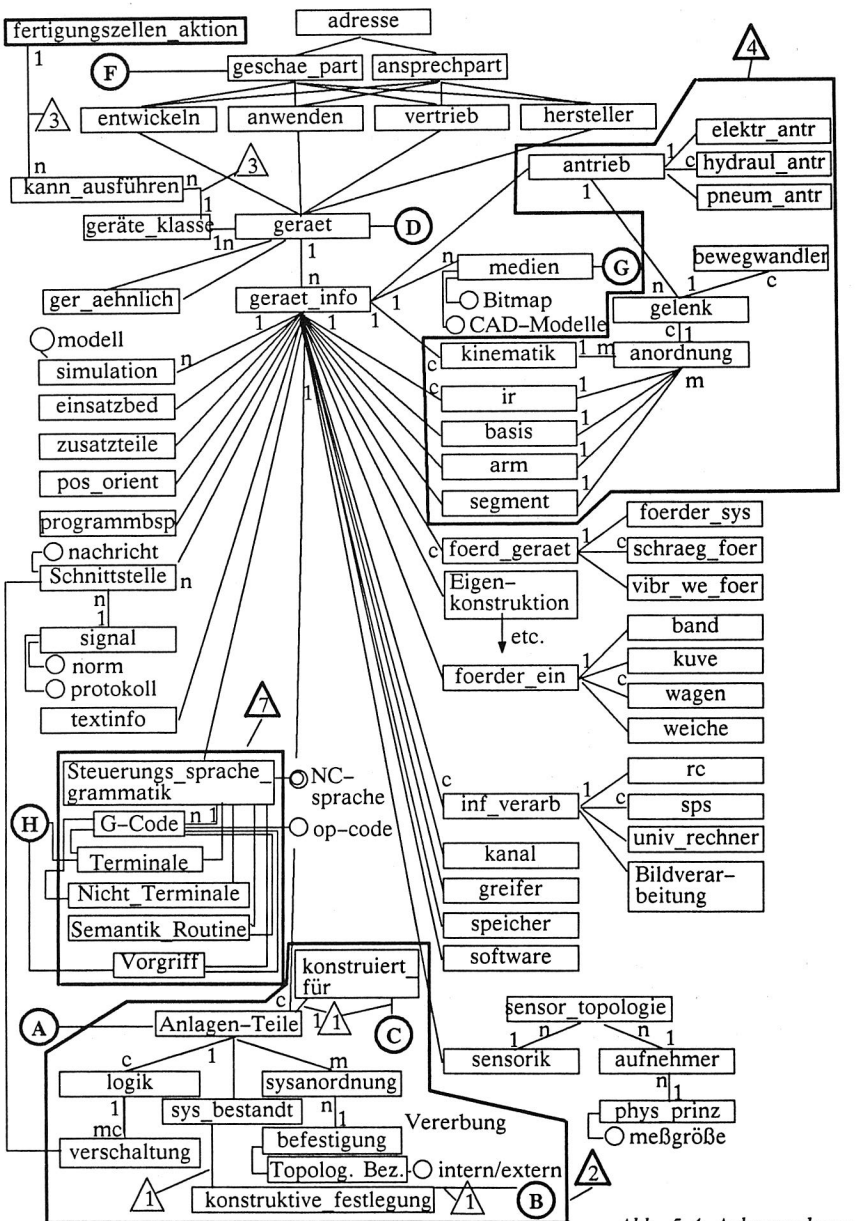


Abb. 5.4: Anlagenschema

Aus Gründen der Übersichtlichkeit sind in Abb. 5.4 Beziehungen und Schemata nur auszugsweise dargestellt.

Dynamische und kinematische Kenngrößen (z.B. von IR oder modulare Handhabungsgeräte) werden in den mit △4 gekennzeichneten Relationen verwaltet. Dazu werden **Segmente** (Starrkörper) gespeichert, die durch **Gelenke** (rotatorisch oder translatorisch) und **Antriebe** gekoppelt sind bzw. mit einem raumfesten Element (**Basis**), **Armen** (kinematische Ketten) oder Robotern (**IR**) kombiniert werden. Diese Detaillierung der Kinematik ist wahlfrei, je nach vefügbaren Daten.

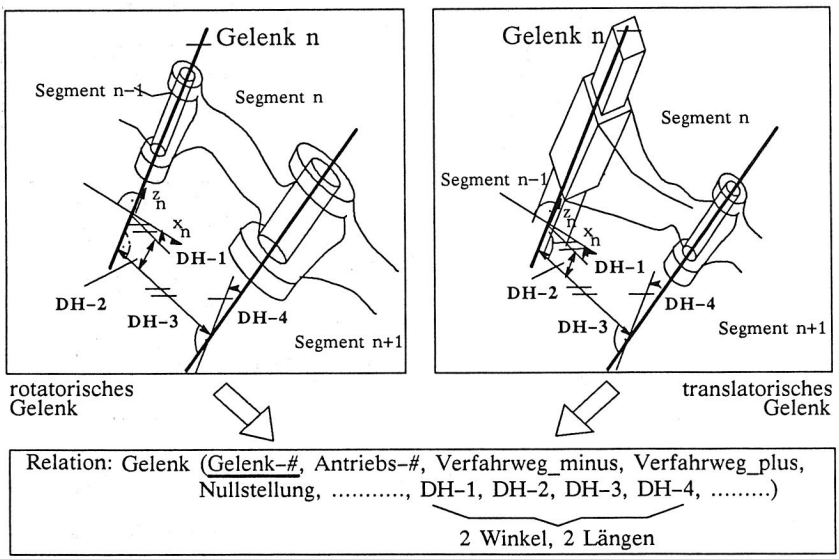


Abb. 5.5: Ablegen von Gelenkkinematik mit dem Tupel (DH-1, DH-2, DH-3, DH-4) 1)

Durch die Speicherung der Topologie (**Anordnung**) obiger Elemente lassen sich neben offenen kinematischen Ketten auch parallele Strukturen abspeichern. Die Gelenk- oder NC-Achsen-**Kinematik** ist durch Angabe von vier Parametern nach Denavit-Hartenberg /22/ beschrieben und kann direkt zur Bewegungssimulation eingelesen werden (Abb. 5.5). Meßwerte, wie z.B. Fehlerverhalten durch Lastverformung als

1) Primärschlüssel von Relation R sind durch Unterstrich gekennzeichnet
R($\underline{S_1, ..., S_n}$, $A_1, ..., A_m$) mit: $S_1, ..., S_n$ = Schlüsselattribute
$A_1, ..., A_m$ = weitere Attribute

Funktion des Arbeitspunktes im Arbeitsbereich des Industrieroboters lassen sich in Zusatzrelationen (s. o.) aufnehmen.

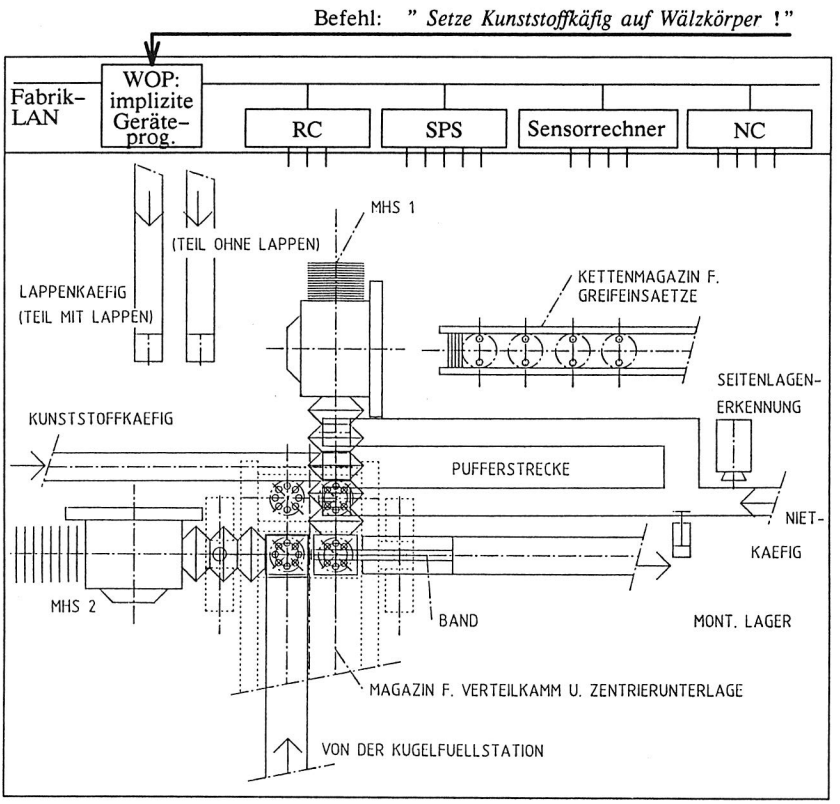
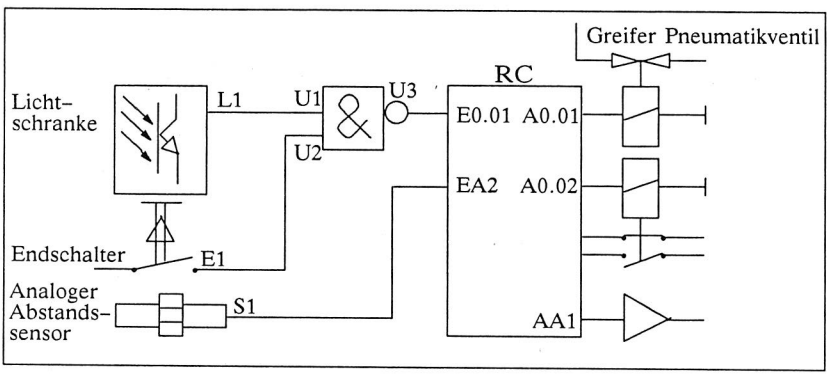


Abb. 5.6: Beispiel einer modular aufgebauten Zelle zur Wälzlagermontage (Draufsicht)

Hochautomatisierte Roboterzellen (wie allg. FFZ) sind meist stark modularisiert (vgl. Abb. 5.6). (Teil-)Anlagen (**Anlagen_Teile**) müssen dazu aus den oben beschriebenen Komponenten aufgebaut werden. Anlagenteilsysteme sind z.B. gekennzeichnet durch Konstruktionsdaten (**konstruiert_für**, **konstruktive_Festlegung**), **Befestigung**, **System_Bestandteile**, wobei jede Komponente eine Positionsnummer erhält, um gleiche Anlagenteile zu unterscheiden (Stückliste). Sie sind durch **Simulations**modelle und einer Aufgabenzuweisung (**Kann_ausführen**) aus der Konstruktion beschrieben.

Neben der geometrischen Konstruktion ist für automatisierte Fertigungsanlagen, bzw. Teilsysteme davon, der logische Entwurf (steuerungs-/ regelungstechnisch) und die **Verschaltung** (hydraulisch, pneumatisch, elektrisch) ihrer Komponenten von besonderer Bedeutung. Analog dem Entwurf elektronischer Schaltungen sollte die Datenbasis die Verschaltungen mit den zugehörigen **Signalen** verwalten und bereitstellen. Dies ist wichtig, da hierüber Datenbankabfragen während der Konstruktion gestartet bzw. Filter (z.B. Verträglichkeit der Signalpegel von **Schnittstellen**) gesetzt werden können (Kap. 5.3.2).



Verschaltung	Anschluß	Von Pin	Zu Pin	Schnittstelle	...
	A1	S1	EA2	analog	...
	A2	E1	U2	0-20 mA	
	⋮	⋮	⋮	⋮	

Abb. 5.7: Wissen über die Verschaltung der Geräte

Die Relationen **SIGNAL, SCHNITTSTELLE, VERSCHALTUNG, LOGIK** etc. beinhalten regelungs- und steuerungstechnische Daten wie vorhandene Anschlüsse, Zuordnung Signal/Ein-, Ausgang, Signalpegel etc. (vgl. Abb. 5.7). Anschlüsse können genormt/nicht genormt, analog, mit Bussystemen (z.B. IEC-Bus) oder seriell mit unterschiedlichen Protokollen arbeiten (z.B. LSV2 für RC-DNC).

Nach der Konstruktionsphase kann die Verschaltung der Geräte in der Datenbank gespeichert werden. So muß, um z.B. den am Roboterarm angeflanschten Greifer implizit anzusteuern, extrahierbar sein, welcher Signal-Ausgang (z.B. binärer Ausgang) der RC mit dem Pneumatikventil des Greifers verbunden ist. Analoges gilt z.B.

für Endschalter oder die Verbindung zwischen analogem Eingang und einem Sensor mit analoger Ausgangsspannung.

Die implizite Aktionsplanung (z.B der Aktionsplan-Compiler in Kap. 4.1) benötigt obiges Wissen über die konkrete Zelle, da z.B. die Generierung des Steuerungscodes abhängig von der konkreten Verschaltung innerhalb der Montagezelle ist. Robotersteuerungen neuer Generationen können nicht nur den Roboter selbst steuern. Die Rechner sind bereits so leistungsfähig, daß der Synchron-Betrieb mehrerer Roboter, sowie Schalt- oder Sensorfunktionen, die früher durch eine externe **SPS** oder **Sensor-Rechner** bearbeitet wurden, durch die **RC** selbst ausgeführt werden können. Begünstigt wird dies durch die Multiprozessorarchitektur künftiger Robotersteuerungen. Ob und wie diese Optionen für eine konkrete Fertigungszelle genutzt werden, hängt von der Ausbaustufe der Robotersteuerung und den Echtzeitanforderungen ab. Dadurch entstehen Mehrdeutigkeiten für die implizite Aktionsplanung, da z.B. Sensorfunktionen durch die Robotersteuerung selbst, als auch durch einen externen Sensorrechner realisiert werden können.

5.2.2 Werkstück/Produkt-DB

Dieser Teil der Datenbasis muß den interaktiv (alternativer Ansatz: Algorithmus nach /41/ auf Basis einer Fügeflächenmatrix) und mit dem AV-Werkzeug nach Kap. 3.4.3 erstellten Montageplan in Graphenform aufnehmen. In diesem Graphen kann der technologische und strukturelle Ablauf einer Fertigungsaufgabe in allen Detaillierungsstufen gespeichert werden. Darüberhinaus müssen die in der Produktkonstruktion entstandenen Werkstückdaten aufgenommen werden:

- CAD-Modelle, 2D-Ansichten, Stücklisten
- Einzelelemente, Baugruppen, Normteile, Wiederholteile (s. /30/), Fügehilfsteile

Bereich ⚠5⚠ in Abb. 5.8 speichert den gesamten Montagegraphen nach Abb. 3.28 einschließlich der Zuordnung zwischen Bauteilen und auszuführenden Aktionen (Werkstück-Aktion-Zuordnung). Bei der Ausführung eines produktabhängigen Montagebefehls "Montiere Gerät XY!" (im Gegensatz zu dem Befehlsbaukasten im folgenden Kap. 5.2.3) wird dieser Arbeitsplan in Graphenform ausgewertet. Der Montagegraph, selbst mit einem datenbankgestützten Werkzeug (Kap. 3.4.3) erstellt, kann Strukturen von impliziten Aktions-Bausteinen enthalten.

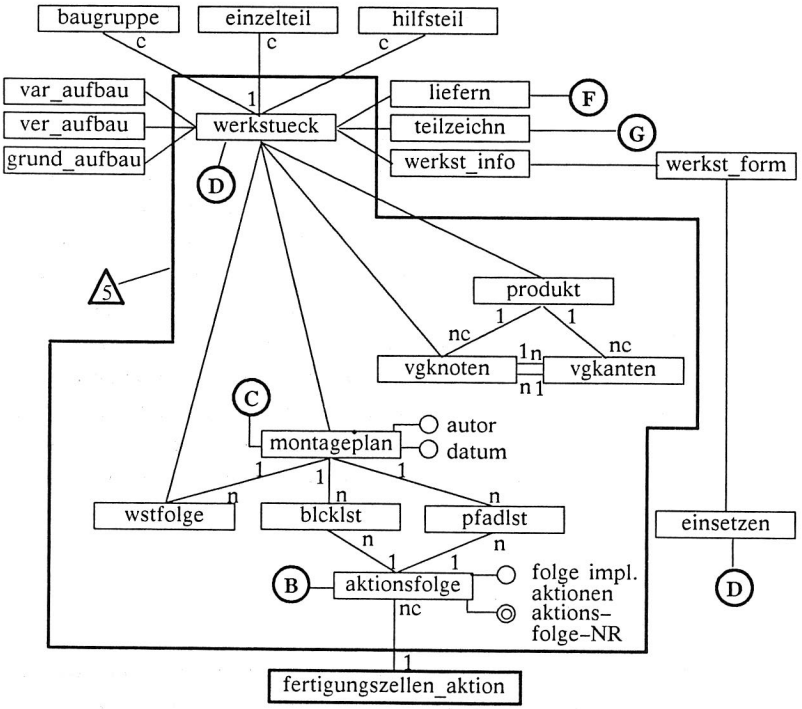


Abb. 5.8: Konzeptionelles Schema der zu montierenden Werkstücke und AV-Arbeitsplan

Werkstücke sind unterteilt in **Einzelteile**, **Baugruppen** und **Hilfsteile** mit **Varianten-** und **Versionenstücklisten** (s. /109/). **Produkte** werden durch einen Vorranggraphen mit **Knoten** und **Kanten** beschrieben. Der erweiterte **Montagegraph** wird durch eine Struktur aus **Werkstückfolgen**, Montage**blocklisten**, **Pfadlisten**, sowie auszuführende **Aktionsfolgen** von impliziten **Fertigungszellen-Aktionen** beschrieben (Kap. 3.4.3). Die Montageaufgabe wird dadurch in rechnergeeigneter Form definiert und in der Datenbank abgelegt.

5.2.3 Meta-Datenbank

Neben den Benutzerrelationen verwalten Datenbanksysteme einen Satz von Systemrelationen. Die Datenbank verwaltet sich selbst mit einer Meta-Datenbank und ohnehin vorhandenen DB-Mechanismen (Datenbank der Datenbank-Schemata). Um zu dokumentieren, welche Information an welcher Stelle in der Datenbank verwaltet

wird, können die DB-Systemrelationen zu einem sog. Data-Dictionary erweitert werden. Mit Hilfe dieses Data Dictionaries (Abb. 5.9) lassen sich neue Datenobjekte überwacht anlegen und Daten (**Attribute**, **Meta_Attribute**, **Wertebereiche**, ...) konsistent und redundanzarm in die Datenbasis eintragen.

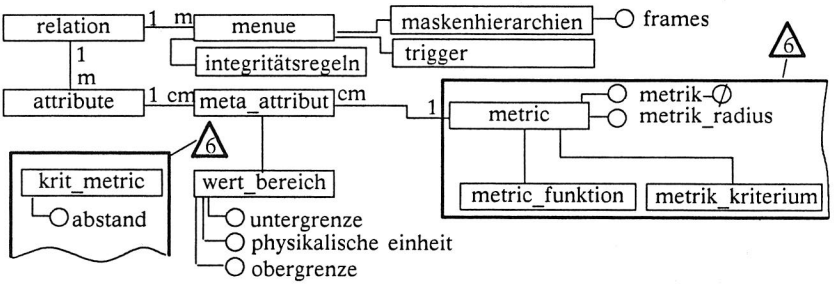


Abb. 5.9: Ausschnitt aus dem konzeptionellen Schema der Metadatenbank

5.2.4 Metrisierte Datenbank

Mit Hilfe der Meta-DB kann die Ähnlichkeit, z.B. von Betriebsmitteln, durch Ermittlung bzw. Speicherung eines Ähnlichkeitsmaßes /73/ bestimmt werden (wichtig für Such- und Konstruktionsprozesse und die Aktionsplanung). Als Ähnlichkeitsmaß dient ein "abstrakter Abstand". Im Gegensatz zur Speicherung des Ähnlichkeitsmaßes in Tabellen (nur für kleine, statische Wertebereiche), ist die Berechnung der Ähnlichkeit von Attributwerten zweier unterschiedlicher Betriebsmittel (') und (") durch einen Algorithmus wesentlich universeller.

Beispiel:

$t_1 = | A'_1 - A''_1 |$
$t_2 = | PU(A'_2) - PU(A''_2) |$

mit A_1= numerisches Attribut
mit PU= z.B. phonetische Umwandlung von Strings A'_2 und A''_2
und A_2= nicht-numerisches Attribut
und A'_1, A''_1, A'_2, A''_2 = Attributwerte

Im einfachsten Fall kann dieser Algorithmus den "Ähnlichkeitsabstand" zweier numerischer Attribute (auch bei nicht-numerischen Attributen, z.B. durch phonetische Umwandlung von Strings (ähnliche Namen und Bezeichnungen)) als den Betrag ihrer Differenz berechnen. Die Erweiterung hiervon zu einem normalisierten Abstand t zweier Betriebsmittel lautet (sog. referentielle Metrik; vgl. /73/):

mit:
$R(S_I, S_{II}, ..., A_1, A_2, A_3, ...)$ und: l = Normierungsfaktor = $f(d_n)$

Primär-Schlüssel-kombinationen (Annahme: nicht-sprechend)

n = n-tes Attribut der DB-Relation R
d_n= Durchmesser= max. Abstand, der zwischen zwei Werten des Wertebereiches möglich ist
r_n= Radius = zulässige Abweichung
t_n= Ähnlichkeitsabstand für Attribut n (s.o.)
u_n= Prioritäten

$$t = l \cdot \sqrt{\left(\frac{u_1 \cdot t_1}{r_1}\right)^2 + \left(\frac{u_2 \cdot t_2}{r_2}\right)^2 + \left(\frac{u_3 \cdot t_3}{r_3}\right)^2 + \cdots}$$

Diese Abstandsbestimmung führt auf eine sog. **metrisierte** Datenbank (/73/), die auch die Suche mit unscharf formulierten Abfragebedingungen zuläßt (vgl. Fuzzy-Sets). Es lassen sich damit bezüglich einiger Attributwerte Kompromisse eingehen (s. Kap. 5.3.2). Mit Metrikunterstützung kann auf exakte Übereinstimmung verzichtet werden. Ist für einen Wertebereich keine geeignete Metrik in der Meta-DB zu finden, wird mit einer Standardmetrik (r=0; entspricht dem Zustand nicht metrisierter DBen) gearbeitet: Jeder Wert ist nur sich selbst ähnlich! Bereich ⚠6 in Abb. 5.9 beschreibt die Metriken.

5.2.5 Modellierung des Aktions-Baukastens

Zur impliziten Generierung von Fertigungszellenaktionen bedarf es einer Methodenbank. Sie beinhaltet das fertigungstechnische Know-How und legt die Semantik impliziter Befehle fest. Die Methodenbank ist der statischste Teil der gesamten Datenbasis. In der Praxis wird dieser Teil meist nur erweitert (Mächtigkeit als auch Umfang).

Für die Begriffsbildung kann man sich teilweise an DIN- bzw. VDI-Funktionen anlehnen (Beispiele in Abb. 5.10/oben). Die in Abb. 5.10 enthaltenen Zahlenangaben (relative Häufigkeit von Fügeoperationen) beziehen sich auf ein Beispiel zur Montage eines feinwerktechnischen Gerätes /114/. Für das Aktionsplanungswerkzeug ist dies nicht ausreichend, und selbst die verfügbaren, begrifflichen Definitionen sind so nicht ohne weiteres brauchbar.

Inhaltliche Definitionen von Fertigungsprimitiven, die für die implizite Geräteprogrammierung ausreichend sind, sind derzeit nicht bekannt (Abb. 5.10/unten). Dieser DB-Teil muß deshalb zunächst, beginnend mit einfachster Fer-

tigungstechnologie am Beispiel von Montage-Aktionen modelliert und gefüllt werden. Hilfreich sind Analogiebetrachtungen zur Natur, z.B. für Befehl "Greife Bauteil XY". Der Mensch impliziert hier schnelle Grobbewegung in die Nähe von XY, feinfühlige Bewegung zur Erreichen der Endposition der Hand, Schließen der Hand, feinfühliges Trennen von den umgebenden Bauteilen, etc. . Zunächst müssen alle, an ein implizites System absetzbaren Fertigungsbefehle (z.B. BV-Aktionen), unabhängig von ihrer Komplexität, bereitgestellt werden.

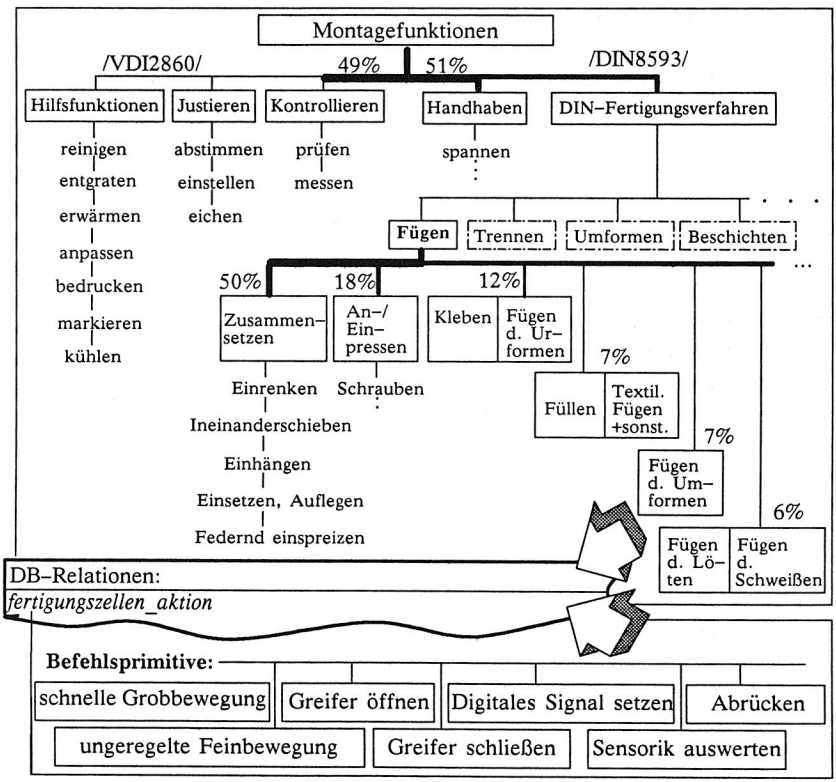


Abb. 5.10: Fertigungstechnologie in der Methodenbank

Die Funktionen müssen feiner in Elementarfunktionen (H-Graph aus Aktionsprimitiven nach Kap. 3.3) aufgelöst und diese Graph-Strukturen, neben den übertragbaren Bewegungs-, Positionier-, und Meßaktionen nach DIN/VDI, in der DB gespeichert werden (Abb. 5.10). Das gilt analog für Peripherie-Aktionen

(Förderbänder, Schleusen, Anwesenheitssensoren, Wendeeinheiten), die in der Praxis fast ausschließlich mit "Boolschen Funktionen" gesteuert werden /76/.

Die Relation **Fertigungszellen_aktion** enthält alle fertigungstechnischen Operationen, die das System kennt (Abb. 5.11). Fertigungstechnische Operationen, deren Semantik so weit in der Datenbank vorhanden ist, daß sie zur impliziten Aktionsplanung- und Steuerung genutzt werden können, müssen durch ein Freigabezeichen gekennzeichnet sein. Dieser Selektor definiert alle beim aktuellen Füllungsstand der Methodenbank beherrschbaren Geräteaktionen.

Aktions-nr	bezeichnung	norm	freigegeben	status	...
12	demontieren	Eigendefinition	1.11.91	online	...
81	einsetzen	DIN 4.1.2	gesperrt	online	
7	Justage	Eigendefinition	9.12.91		
8	Graubild_ binärisieren	Eigendefinition	in der Model- lierungsphase	intern	
4	abrücken	Eigendefinition			
:	:	:	:	:	

Aktions-nr	Methoden-nr	spezialist	methode	..
2	1	FÜGEPLANER	nahpunkt_bestimmen	
2	2	BAHNPLANER	kollisionsfreien_pfad_bestimmen	
2	3	ARBEITSPLANER	umweltmodell_aktualisieren	
4	1	FÜGEPLANER	zielpunkt_bestimmen	
4	2	FÜGEPLANER	zielpfad_bestimmen	
4	3	ARBEITSPLANER	umweltmodell_aktualisieren	
7	1	GREIFPLANER	greifer_name_bestimmen	
7	2	GREIFPLANER	greifer_status_bestimmen	
8	1	FÜGEPLANER	nahpunkt_bestimmen	
8	2	BAHNPLANER	kollisionsfreien_pfad_bestimmen	
8	3	ARBEITSPLANER	umweltmodell_aktualisieren	
1047	1	COMMONSENSE-WÄCHTER	cs_fest_verbinden	
9	2	SENSOREINSATZ-PLANER	...	
:	:	:	:	

Abb. 5.11: Relationen Fertigungszellen_aktion (oben) und Aktions_Methoden (unten)

Die Befehlssemantik kann in **Aktions_parameter, Aktions_Methoden** und **Aktions_Bestandteile** abgelegt werden (s. Abb. 5.12). Aktions_parameter enthält die ungefüllte Parameterliste nach Abb. 3.23 in Kap. 3.3.2. Aktions_Bestandteile enthält die Bestandteile (+Struktur), in die sich eine Aktion zerlegen läßt (genau: =Ad-

jazenzmatrix des bgi-Graphen, zu dem ein H-Graph-Knoten expandiert wird). Jede Subaufgabe muß entweder elementar sein oder wieder aus Subaufgaben bestehen. Durch diese Aufspaltung in unterlagerte Operationen können Aktionsstrukturen (z.B. Sequenzen) relativ allgemeingültig abgelegt werden (Strategienbank, z.B. für Sensoreinsatz, Fügestrategien, etc.). Strategien können so überarbeitet ('editiert') und verbessert werden.

Nach Kap. 3.2.2 müssen während der Expandierung eines impliziten Befehles verschiedene Spezialisten Subaufgaben erfüllen. Aktions_methoden beschreibt die zu den H-Graph-Knoten zugehörigen Listen von Methodenaufrufen der Spezialisten.

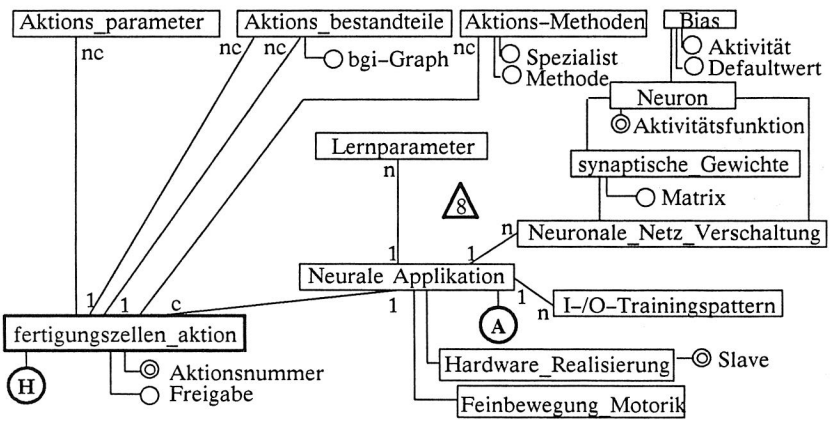


Abb. 5.12: Konzeptionelles Schema der Methodenbank

Die Mächtigkeit des impliziten NC-Programmiersystems an beherrschbaren Fertigungsaktionen kann durch Nachladen dieser Relationen mit maschinenlesbarer Füllung (s. Kap. 5.3.3) erweitert, bzw. aktualisiert werden. Diese Relationen sind mit Fertigungsaktionen unterschiedlichster Komplexität gefüllt. Die Füllung wird je nach Umfang des Fügewissens sehr umfangreich (hohe Tupelanzahl) und erfordert nicht zuletzt deshalb den DB-Einsatz.

Die derart gespeicherten Aktionen lassen sich zu neuen **Bausteinen** kombinieren. Sie bilden den wesentlichen Bestandteil einer Wissensbasis zur Fügetechnik. Jeder neue, vollständig dokumentierte Baustein steht ab seiner Freigabe für die online-Aktionssteuerung und/oder -planung, sowie für die Konstruktion neuer Fertigungsak-

tionen zur Verfügung (Abb. 5.13). Die Aktionen sind produktunabhängig und definieren eine Operations-Grundmenge.

Bereich ⚠8⚠ in Abb. 5.12 speichert Lernergebnisse, synaptische Wichtungen und Topologie der trainierten Neuronalen Netze nach Kap. 6.

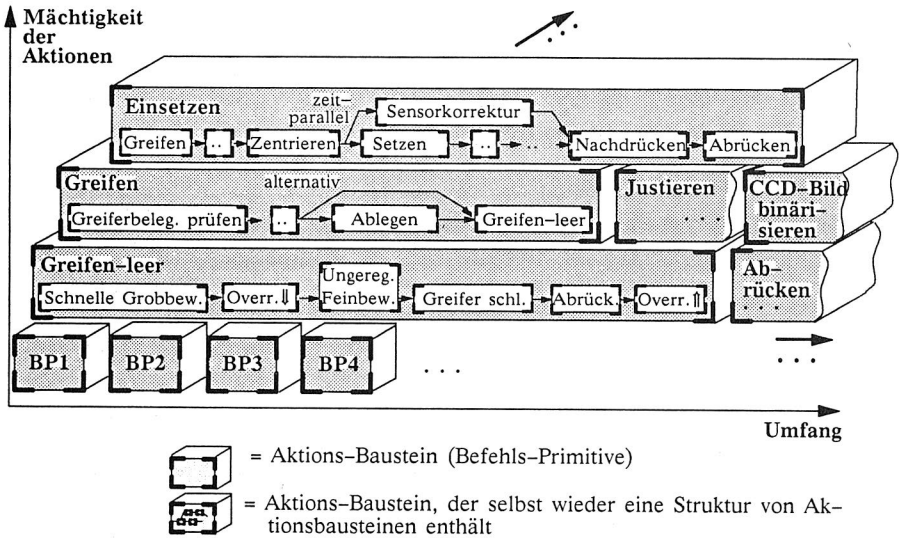


Abb. 5.13: Baukasten impliziter Aktionen

Diese einfachen (Roboter-)Aktionen sind die Grundbausteine weitaus komplexerer (Füge-) Anweisungen, die durch Kombination der Grundbausteine entstehen. Sie sind auch in technologiespezifischen Befehlsmengen (z.B. für Laserstrahlschneiden, Spritzlackieren, Koordinatenmessung) enthalten. Projektierung, Konstruktion und Einbringung neuer/abgewandelter, implizit integrierbarer Bausteine sollten zentral durch ein wissensbasiertes und datenbankgestütztes Werkzeug geschehen (Abb. 5.14). Dieses Werkzeug muß die Selektion (Welche Aktionen sind bereits verfügbar?), das Modifizieren (Parameterliste, Methodenliste), Trennen und Zusammenfügen (Abprüfen von Verträglichkeiten, etc.) der Aktionsbausteine ermöglichen.

Es muß die Datenbank nachtragen, sichert alle referentiellen Integritäten (Compiler-Grammatik, konsistente Integration in den bestehenden Befehlsbaukasten, etc.) und gibt den Befehl der Aktionsplanung frei. Befehlsbausteine dürfen nur Methoden auslösen, über die die Spezialisten auch verfügen. Sollte ein Spezialist neue Methoden

benötigen (z.B. neue Bildvorverarbeitung des CCD-Einsatz-Planers), so müssen sie (nachdem sie programmiert wurden) in der Relation nach Abb. 5.11 ergänzt werden.

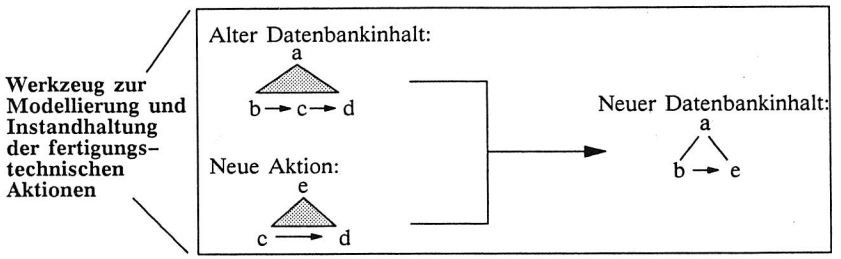


Abb. 5.14: Beispiel für die Einführung eines neuen, häufig benötigten Befehls e

Dieses Tool zur Konstruktion von fertigungstechnischen Aktionen muß das fertigungstechnologische Wissen optional maschinenlesbar bereitstellen, so daß auch autark arbeitende Werkstattsysteme damit aktualisiert werden können (z.B. anwenderspezifisches Technologie-Wissen kleiner Firmen).

5.3 Datenbankgestützte Anlagenkonstruktion auf CAD-Basis

Wie bereits erwähnt entsteht während der Konstruktion der Fertigungsanlage, und hier vor allem bei der Betriebsmittelauswahl, ein wesentlicher Teil der für die implizite Aktionsplanung- und steuerung erforderlichen Basisdaten (vgl. /128/). Deshalb müssen bereits hier (CIM !) mit Hilfe eines datenbankgestützten Informationssystems die Basisdaten für die spätere, implizite NC-Programmierung angelegt werden. Das datenbank- und CAD-gestützte Werkzeug muß hierfür zwei Ergebnisse liefern:

A) Konstruktion der Montageanlage (z.B. CAD-Modelle) und damit gleichzeitig
B) Aufbau und interaktive Erstellung des für die implizite Aktionsplanung und -steuerung erforderlichen Basiswissens (Anlagenmodelleditor, s.o.)

Im Gegensatz zu den direkten Konstruktionstätigkeiten, wie die reine Geometriemodellierung, existieren für die indirekten Tätigkeitsbereiche (Suchvorgänge, Beschaffung, und Vergleich technischer Gerätedaten), mit einem Anteil von ca. 50% der gesamten Konstruktionstätigkeit, kaum geeignete Werkzeuge. Konventionelle Informationsquellen wie Konstruktionsunterlagen (bereits erstellte CAD-Modelle oder Zeichnungen), Gerätekataloge, Handbücher, Datenblätter sowie per-

sonenabhängiges Wissen über das Verhalten von Betriebsmitteln erhöhen die Datenredundanz. Nachträglich ergänzte Informationen (Genauigkeiten, Meßergebnisse, etc.) und die parallele Verwaltung von CAD-Modellen verhindern die technisch optimale Lösungsfindung. Der Umfang technischer Daten, deren Verträglichkeiten und Seiteneffekte sind ohne Datenbankeinsatz nicht mehr beherrschbar.

5.3.1 Funktionszuweisung zu Betriebsmitteln

Derzeitige CAD-Systeme erzeugen, für sich alleine betrachtet, keine CIM-fähigen Daten. Sie verwalten nur Geometrieelemente - es fehlen "hochwertige Datenstrukturen" wie sie für die implizite Geräteprogrammierung benötigt werden. Durch Kombination von Datenbanktechnologie mit wissensbasierten Methoden (z.B. zur Konfiguration von Datenbankmanipulationen) können die Geometriedaten mit darüberhinausgehenden Informationen angereichert werden.

Die primäre Aufgabe technischer Geräte ist die Einhaltung von Funktionalität. Um für eine vorgegebene Füge- bzw. Bewegungsoperation die gerätetechnische Realisierung möglichst komfortabel zu ermitteln und das Konstruktionswissen in der impliziten Geräteprogrammierung auswerten zu können, muß die Anbindung an die Montageaufgabe automatisiert werden.

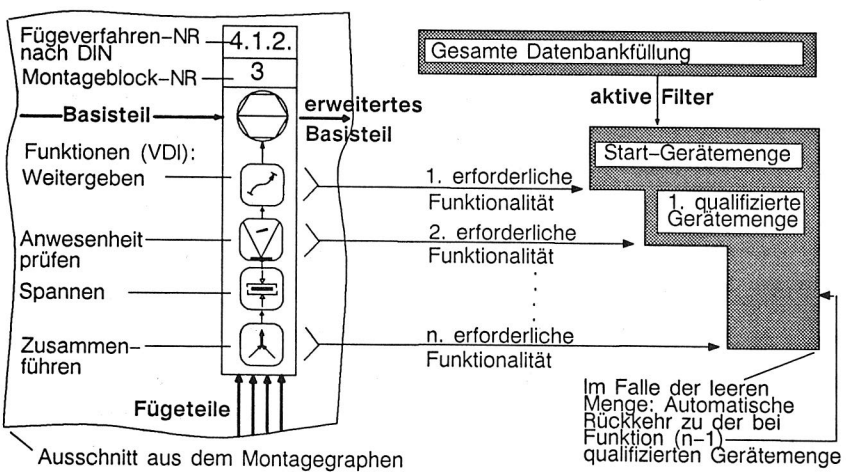


Abb. 5.15: Beschreibung der Montageaufgabe (links Graph-Ausschnitt aus Abb. 3.28, vgl. /109/) und Strategie zur automatisierten Betriebsmittelauswahl

Hierzu wird der in der Datenbasis gespeicherte Montagegraph (AV-Arbeitsplan nach Abb. 3.28 und Kap. 5.2.2) des zu montierenden Produktes ausgewertet und vom Rechner Gerätevorschläge für die Betriebsmittelauswahl unterbreitet.

Ein Ausschnitt dieses Graphen (hier Knoten) ist in Abb. 5.15 dargestellt. Die derart formulierte Bewegungs- und Fügeaufgabe dient als Grundlage für eine automatisierte Betriebsmittelauswahl. Dazu kann die geforderte Funktionalität mit dem Funktionsumfang der in der Datenbank verfügbaren Betriebsmittel abgeglichen werden. Eine Auswahlstrategie hierfür zeigt Abb. 5.15,rechts. Wurden die Betriebsmittel derart ausgewählt und die FFZ konstruiert, so kann mit der Zuordnung zwischen Betriebsmittel und zu realisierender Fertigungsaktion gleichzeitig das Fundament für die implizite Aktionsplanung geschaffen werden. Hierauf kann der Aktionsplaner zugreifen.

5.3.2 Prototyp eines Planungswerkzeuges

Die Hardwarebasis des prototypisch realisierten Werkzeuges zur Betriebsmittelauswahl und Aufbau des Anlagenmodells bilden 32-bit Graphik-Workstations (Apollo/HP mit UNIX/AEGIS-Betriebssystem) und lokaler Rechenleistung. Sie sind über ein Netzwerk in Ring-Topologie verbunden (s. Abb. 4.11 in Kap. 4.3.2). Das entworfene konzeptionelle Gesamtschema nach Kap. 5.2 wurde mit dem relationalen Datenbankmanagementsystem INGRES unter dem Betriebssystem UNIX implementiert. Es umfaßt auch das vom impliziten Programmiersystem benötigte fertigungstechnologische Wissen. Die Netzwerkkommunikation für den verteilten Datenbankzugriff basiert auf TCP/IP-(Protokoll). Der Bediendialog wurde mit dem Softwaretool "DOMAIN-DIALOGUE" erstellt. Daten, Transaktionsprotokolle und Checkpoints (zusammengefaßt in Areas und Locations /20/) sind auf unterschiedliche Magnetplatten verteilt (Medienfehler !). Das Werkzeug muß komplexe Datensuche, Konsi-stenzprüfungen, sowie Korrektur und Erweiterung der aktuellen Datenfüllung (nach Kap. 5.2) rechnergestützt durchführen.

Als Sprache zur Datendefinition bzw. -manipulation wurde **SQL** (**S**tructured **Q**uery **L**anguage), sowohl in interaktiver als auch in eingebetteter Form verwendet. DB-Objekte (in INGRES sog. **Frames**) mit zugehörigen Masken und Triggern wurden mit **OSL/OSQ** (**O**peration **S**pecification **L**anguage) und **ABF** (**A**pplication **b**y **F**orms) realisiert (Frame-Hierarchie dargestellt in Abb. 5.21). Diese systemeigene, oft als 4Gl (4th generation language) bezeichnete Datenbanksprache, ist Bestandteil einer Software-Entwicklungsumgebung. In ihr werden Prinzipien der nichtprozeduralen

Programmierung verwirklicht. Die eigene Sprache **QUEL** (**Q**uery **L**anguage) ist kein Standard, jedoch teilweise mächtiger (für Problemfälle). Das Laufzeitverhalten von embedded SQL-Manipulationen kann dadurch verbessert werden, daß die Datenbank während der Dauer der Applikation an diese angebunden (geöffnet) bleibt.

Die Ankopplung der CAD-Daten wurde mit dem **K**ernel-**I**nterface (**KI**, s. Kap. 3.4.1) der CAD-Volumenmodellierer ROMULUS und PARASOLID (z.B. in Sigraph, Fa. Siemens) realisiert. Ein Haupthindernis zur Realisierung des Prototypen stellten, neben zu geringer Funktionalität der verfügbaren Rechnerwerkzeuge (CAD,DB), deren unzureichende Kopplungsmöglichkeiten dar.

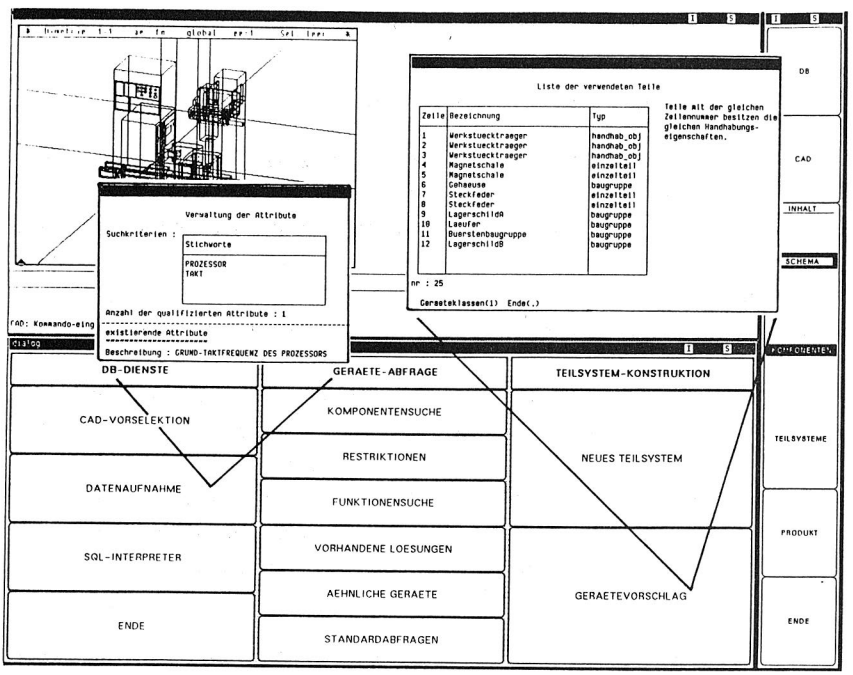


Abb. 5.16: 3-Teilung der Bildschirmoberfläche: Oben: CAD, Unten: Datenbank-Zugang, Rechts: Steuerung (Bildschirmkopie)

CAD-Volumenmodelle sind deshalb nach wie vor in dem, auf schnellen Zugriff optimierten CAD-Fileverwaltungssystem abgelegt. Durch diese Aufteilung soll die

Mächtigkeit einer relationalen Datenbank mit schneller und effizienter Fileverwaltung des CAD-Systems kombiniert werden. Analoges gilt für Bitmaps (schnelles, elektronisches Blättern).

Um effizient mit dem Werkzeug arbeiten zu können, muß ein zeitparalleler Zugang zur Gerätedatenbank, wie auch zum CAD-System, an einem einzigen Rechner-Arbeitsplatz möglich sein (Betrieb in den beiden Modi "DB" oder "CAD"; Abb. 5.16). Im CAD-Mode wird, bis auf die Steuerung, die gesamte Bildschirmoberfläche für das CAD-Programm genutzt. Im Modus DB wird die CAD-Arbeitsfläche verringert (einschließlich Anpassung geladener Modelle) und die Datenbankdienste werden eingerichtet. Auslösen von Datenbank-Funktionen, die das CAD-Modell betreffen, sind damit nach wie vor ausführbar (s. Abb. 5.16).

Obige Forderung gilt nicht für die Arbeitsplanerstellung nach Kap. 3.4.3 (event. personell getrennte Erstellung), d.h. die datenbankbasierte Erstellung des AV-Montagegraphen und die Betriebsmittelauswahl sollten parallel auf zwei getrennten Rechner-Arbeitsplätzen möglich sein. Hinzu kommen Zugriffe der verteilten Werkzeuge nach Abb. 4.11 (Frontend 1 bis 4) sowie Zugriffe des prozeßnahen Werkzeuges (Frontend 5) über LAN (Extrahieren des fertigungstechnologischen Wissens des impliziten Befehlsbaukastens beim Hochstart des Systems in der Werkstatt und Komplettmontage nach Kap. 3.4).

Da diese Prozesse alle auf die gleiche Datenbasis zugreifen, ist mindestens ein verteilter Datenbankzugriff z.B. nach dem Client-Server-Konzept /80/ mit mehreren Clients (Frontend-Prozesse mit Anwenderprogrammen) und einem Server (Datenbank-Backend-Prozeß mit DB-Manager und Query-Optimizer) erforderlich.

Für die Suche nach Betriebsmitteln werden die Geräte klassifiziert. Grobklassen sind implizit durch die Zugehörigkeit zu dem System von Basis-Relationen nach Abb. 5.4 gegeben. Grobklassen sind z.B.:

- SPS
- IR (Komplettgerät)
- Segment
- Kanal
- Sensor
- RC

usw.

Abb. 5.17 zeigt am Beispiel der Grobklasse SPS den möglichen Umfang an Attributen (Zeichenketten in den eingerahmten Feldern sind mousesensitiv und tiefer referenziert).

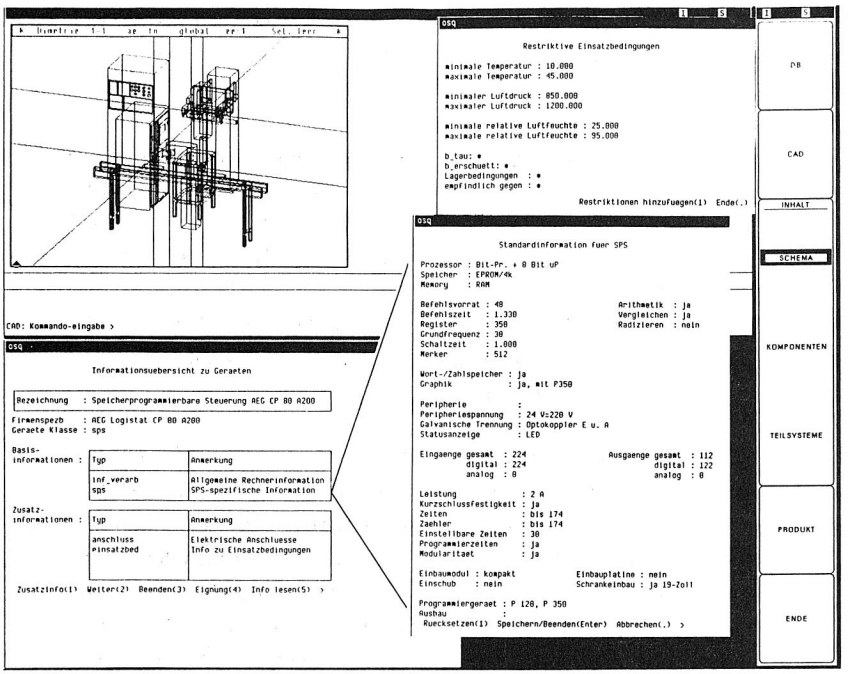


Abb. 5.17: Basis–Attributmenge der Automatisierungskomponente SPS (Bildschirmkopie)

Diese vordefinierten Grobklassen alleine sind für den praktischen Einsatz zu starr. Ideal ist eine flexible Klassifizierung der Betriebsmittel, die der Benutzer zur Laufzeit und abhängig von speziellen Bedingungen festlegt. Im Prototyp lassen sich während des Betriebs häufig wiederkehrende feine Unterklassen eröffnen (z.B. induktive Wegaufnehmer, die aufgrund vorhandener Erfahrungswerte für den bündigen Einbau geeignet sind). Außerdem sollte einer Geräte–Subklasse eine Menge ausführbarer Aktionsprimitive oder höherer Aktionsbausteine (Kap. 5.2.5) zugeordnet werden können. Diese Aktionszuordnung kann für eine "treffsichere" Geräteauswahl verwendet werden.

Individuelle Zusammenstellungen von Baukastenelementen werden als ausgeführte Teilsysteme betrachtet (Montagezelle in Abb. 5.17). Das Werkzeug muß nach Beendigung der Konstruktion das CAD–Modell eines (Teil)–Systems auf seine Assembly–Bestandteile hin untersuchen (1–stufige Auflösung) und anschließend mit dem Daten-

bankinhalt vergleichen. Dies geschieht durch den direkten Zugriff auf den Modellier-Kernel des CAD-Standardmodellierers (KI-Routinen, s.o.).

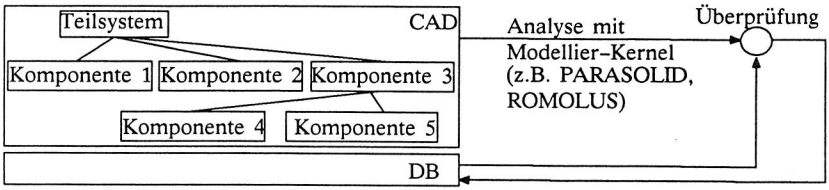


Abb. 5.18: Analyse der CAD-Modelle mit Hilfe des Modellier-Kernels

Mit Hilfe des Modeller-Kernels kann, von der Datenbank aus, das CAD-Modell direkt manipuliert bzw. automatisiert analysiert werden (Abb. 5.18). Vor dem Eintrag eines Anlagen-Teilsystems in die DB muß geprüft werden, ob alle darin enthaltenen Komponenten bereits in der Datenbank erfaßt wurden und die Speicherung entsprechend dem Geräteteilschema in Abb. 5.4 konsistent möglich ist. Für den Fall, daß Bestandteile gefunden werden, die nicht in der Datenbank gespeichert sind (z. B. durch externes Zuladen oder Eigenkonstruktion), wird dies erkannt. Es muß ein entsprechender Nachtrag veranlaßt werden.

Strategien zur Suche nach Geräten bilden den Kernpunkt der Betriebsmittel-Datenbank. Die aus dem Entwurf integrierter Schaltungen bekannten Logikoperatoren (OR, NOR, AND, NAND, etc.) können zusammen mit Wildcards über die Attribute gelegt werden. Interaktive Abfragen lassen sich so iterativ (z.B. SQL-Monitor-Programme) staffeln. Eine dynamisch erweiterbare Liste vordefinierter Standardabfragen (Kennzeichen: 0 bis 2 Parametern, einfache "Stationstasten"), sowie Maskenprogrammierung ermöglichen das Arbeiten mit der Datenbank auch ohne Kenntnis von Abfragesprache und Datenbankschema. Häufig benötigte Standardabfragen und Filter werden mit einem DB-Tool rechnergestützt erstellt und in der Meta-DB abgelegt.

Ein Suchprozeß, der die Auswahl eines bestimmten Betriebsmittels zum Ziel hat, muß dazu sowohl auf die Basis- wie auch auf die Zusatzinformationen zugreifen. Dabei ergeben sich zwei häufig wiederkehrende Fragestellungen: a) Finden einer Komponente, die zu einem gegebenen Gerät "ähnlich" ist. b) Suche von Baukastenkomponenten derart, daß jedes Einzelelement bestimmte Nebenbedingungen erfüllt (und damit auch das aus diesen Elementen aufgebaute Teilsystem der Fertigungsanlage).

Problem a) tritt auf, wenn man bei der Suche nach Betriebsmitteln, die bestimmte Anforderungen erfüllen, auf Geräte stößt, die dem Gesuchten zwar schon sehr nahe kommen, aber aus unterschiedlichen Gründen nicht gewählt werden können. Mit Hilfe der Metriken (nach Kap. 5.2.4) kann die Suche auf solche Komponenten beschränkt werden, die sich dem zunächst gefundenen Betriebsmittel in den qualifizierten Attributen ähneln, sich aber in den Attributen unterscheiden, die die Auswahl verhindert haben.

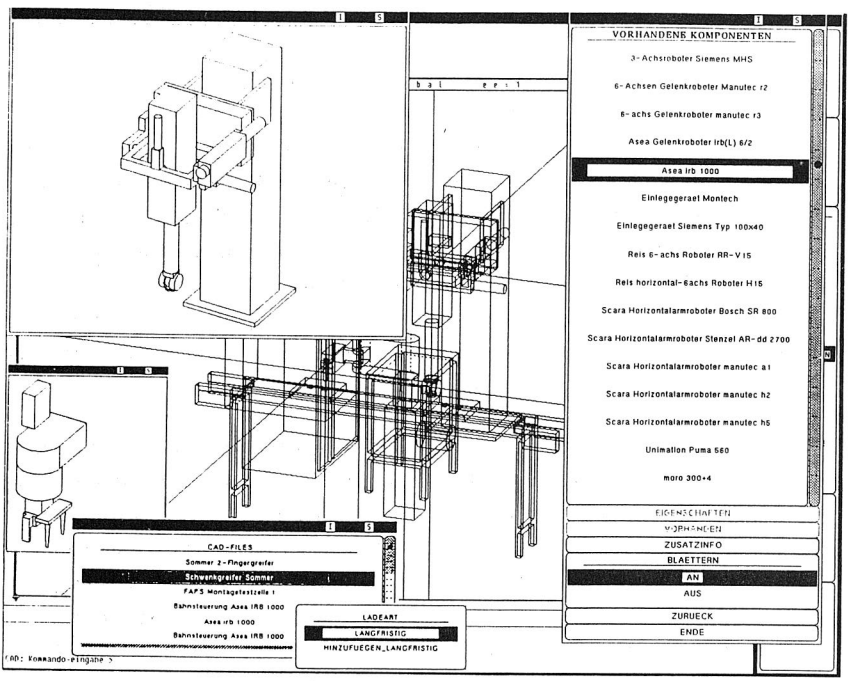


Abb. 5.19: Dynamische Liste der aktuell verfügbaren Komponenten vom Typ 'IR' (CAD-Modell-Analyse und Bitmap-Verwaltung ("Blättern an"), Bildschirmkopie)

Problem b) wird durch Zwischenschalten von Filtern gelöst. Der Benutzer kann ein-/ausschaltbare bzw. konfigurierbare und langfristig wirkende Filter setzen. Die Informationsmenge des nachfolgenden, "normalen" DB-Betriebes wird durch die jeweils aktiven Filter permanent gesiebt. Auf diese Weise können auch Bedingungen definiert werden, die für mehrere Anfragen und über längere Zeit gelten sollen. Für

zeitgleichen Zugriff muß für jeden zugreifenden Prozeß eine individuelle Relation H existieren, damit sich die Filter nicht gegenseitig beeinflussen.

Embedded DB-Zugriffe (z.B. Abb. 5.15) lassen sich häufig mit dynamisch konfigurierbaren Filtern lösen. In SQL kann mit einer Hilfsrelation H (nach H werden zunächst alle Schlüssel geschrieben) und einer Nutzrelation R formuliert werden:

```
create H
insert into H select Schlüssel from R
delete from H where Schlüssel not in (select Schlüssel from R where Filter 1
                                                                and Filter 2
                                                                and ...
                                                                and Filter n)

select * from R where qualification 1 and ... qual m and Schlüssel in
                                                (select Schlüssel from H)
```

Jeder Benutzer kann damit seine individuellen Restriktionen setzen und konfigurieren. Vorgebbare Filter sind z.B. Vorhandensein des CAD-3D-Volumenmodells eines Betriebsmittels oder einer einheitlichen Rechnerschnittstelle (z.B. V.24), Einsatzbedingungen (Abb. 5.17, oben rechts) oder Verträglichkeit von Signalpegeln. Die ausgewählten Betriebsmittel können in einem Kommissionierungspuffer gesammelt und, sofern die zugehörigen CAD-Modelle vorhanden sind, in das CAD-System geladen werden (Abb. 5.19).

Bei der Implementierung mußten einige Kompromisse eingegangen werden, die zukünftig durch folgende Weiterentwicklungen der Datenbanktechnologie überflüssig werden:

a) Erweiterung der verfügbaren Datentypen durch objektorientierte, multimediafähige Datenbanken. Für technische Objekte sind die derzeitigen Datenbanksysteme noch zu starr (gilt besonders für die große Bandbreite technischer Gerätedaten).

b) Bildung einer homogenen Einheit aus CAD-System und Datenbank.

c) Schaffung mächtiger Mechanismen um komplexe Integritätsbedingungen komfortabler als bisher zu implementieren.

d) Lieferung der Katalogdaten in maschinenlesbarer Form.

5.3.3 Maschinenlesbare Datenbankfüllung

Punkt d) ist entscheidend für die schnelle Füllung und Aktualisierung der Datenbasis. Die Hersteller von Geräten müssen langfristig dazu übergehen, das relationale Datenbankschema einschließlich Datenfüllung der technischen Gerätedaten auf

Datenträgern, (z.B. Band oder öffentliche Netze (Leitungsvermittlung oder festgeschaltete Verbindungen), vgl. /97/), in maschinenlesbarer Form vorzulegen (Abb. 5.20).

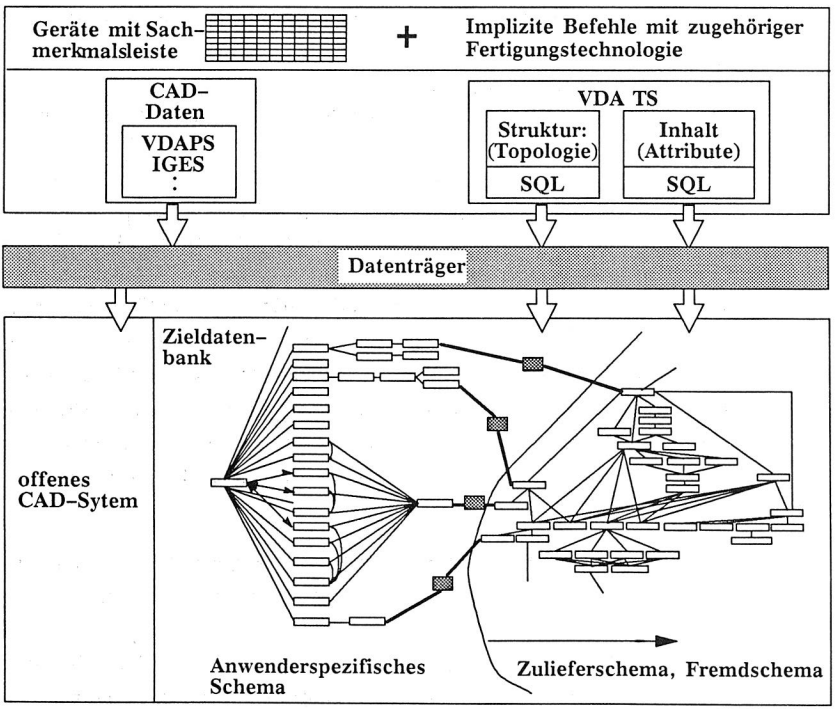


Abb. 5.20: Automatisierte Füllung der Datenbasis

Wichtig ist die Wartung, Pflege und automatisierte Füllung des fertigungstechnologischen Basiswissen des impliziten Aktions- und Befehlsbaukastens. Eine leistungsfähige "Wissens-Instandhaltung" (vgl. Wissenserwerbskomponenten von Expertensystemen, z.B. in /43/) ist von besonderer Bedeutung, da während dem Betrieb der CAD/CAM-Verfahrenskette die Mächtigkeit und der Umfang der beherrschbaren Fertigungsbefehle ständig wächst. Die Fertigungstechnologie impliziter Aktionen (nach

Kap. 5.2.5) sollte deshalb, analog zu den Gerätedaten, automatisiert in der Datenbank nachgetragen werden können (z.B. anwenderspezifische Technologien).

Der Normungsprozess für die DIN 4001 (Vornorm im Herbst 1987) gliedert die Beschreibung von Norm- und Zulieferteilen in Daten der Sachmerkmalsleiste und in Geometriedaten (einfache Norm- und Zulieferteile). Auf dieser Grundlage wird eine Transaktionsschnittstelle (TS) entwickelt /124/ mit der konzeptionelle Datenbankschemata, einschließlich Füllung, übertragen werden können. Die Geometrieübertragung erfolgt parallel z.B. über VDA PS, DIN V 66304.

Die Transaktionsschnittstelle ist zweigeteilt. Die Schematopologie wird mit Hilfe eines Strukturedítors, die Füllung mit einem Relationeneditor beschrieben. Mit Hilfe eines VDA-TS-Parsers /124/ wird über das DIN V 4001 TS-Format die Topologie und die Füllung der Tabellen einer relationalen Datenbank übertragen. Die Rekonstruktion der Ziel-DB erfolgt mittels der Sprache SQL (Abb. 5.20). Um das Fremdschema benutzten zu können, kann in die Zieldatenbank die Meta-Datenbank des Fremdschemas mit übertragen werden und (mit Hilfe von Andock-Relationen) die Struktur des Fremdschemas bekannt gemacht werden.

6. Lernen mit Neuronalen Netzwerken in automatisierten Fertigungsanlagen

Implizite Aktionssteuerung führt zwangsläufig zur Behandlung von Lernprozessen, da **Lernen** die denkbar mächtigste Entwicklungsstufe impliziten Programmierens von Fertigungsanlagen darstellt.

Neuronale Netze eignen sich zur Realisierung von Steuerungsfunktionen, die entweder nicht analytisch beschreibbar sind oder deren Beschreibung nicht wirtschaftlich möglich ist. Sie sind in der Lage aus unpräzisen und verrauschten Eingangsdaten (z. B. in der BV) fehlersichere Ergebnisse zu produzieren. Sie entstammen der Neurophysiologie /108/, /38/ und werden durch Analogiebetrachtungen in den Bereich der Technik übertragen. Das menschliche Gehirn arbeitet hochgradig parallel und erreicht so höchste Leistungsfähigkeit, obwohl eine isoliert betrachtete Nervenzelle, im Vergleich zu technischen Systemen, nur relativ langsam arbeitet. Die im Ergebnis trotzdem brauchbaren Verarbeitungsgeschwindigkeiten sind eine Folge der geringen Berechnungstiefe (im Durchschnitt ca. 5 Neuronen) und der massiven Parallelität.

In der Rechnertechnik nutzen parallele Rechnerstrukturen die nicht-sequentielle Problembearbeitung mit dem Ziel hoher Berechnungsdurchsätze. Ein Ansatz, biologische Systeme der Natur zu kopieren, führt auf neuronale Netzwerke, die das Funktionsprinzip einer Ansammlung von Nervenzellen mathematisch nachbilden und für die sich parallele Berechnungsvorschriften angeben lassen (/103/, /104/, /99/, /5/, /18/) . Das hieraus entstehende Gebiet der Neuroinformatik läßt vor allem neue Ver-

fahren zur Lösung komplizierter Aufgaben in der Signalverarbeitung, Mustererkennung oder Robotik erkennen.

Herkömmlich wird eine exakt definierbare Sequenz von Berechnungsschritten ausgeführt; sie liefern hohe Genauigkeiten. Die Verarbeitungsgeschwindigkeit einer Einzelanweisung ist hoch. Der grundsätzliche Unterschied gegenüber konventioneller Informationsverarbeitung besteht darin, daß das gewünschte Verhalten nicht durch explizites Programmieren, sondern durch Training erzielt wird. Training ersetzt das Umprogrammieren von Steuerungsalgorithmen, d.h. bei einer Änderung der Problemstellung muß anstatt einer Neuimplementierung die Netzwerkstruktur erneut festgelegt und Trainingsläufe durchgeführt werden. Gerade diese Eigenschaft macht sie besonders interessant für die implizite Aktionsplanung von Fertigungsaufgaben, wie beispielsweise das Lernen von feinmotorischen Montagebewegungen eines Industrieroboters oder die implizite Sensorintegration.

Die Robustheit gegenüber unvollständigen Eingaben, sowie die Fähigkeit aus unscharfen Eingaben hinreichend korrekte Ausgaben zu erzeugen, kann dazu genutzt werden, die Fehleranfälligkeit im praktischen Umfeld der Produktion zu verringern.

Die Grundidee besteht darin, komplexe Bewegungsabläufe wie sie z.B. in der Montagetechnik auftreten und z.B. manuell gelöst werden, ähnlich dem "Vorbild Natur" zu trainieren. Die Geschicklichkeit eines Menschen, koordinierte Bewegungsvorgänge oder Haltefunktionen (Montagevorgänge) auszuführen, beruht auf jahrelangem Training seit der Kindheit. Gegenüber den konventionellen, algorithmischen Lösungen (z.B. Fügeplanung nach Kap. 7.1) benötigen z.B. neuronale Regelkreise des zentralen Nervensystems /38/ keinen mathematisch faßbaren Zusammenhang mehr, bzw. falls dieser besteht ist er nicht mehr aus dem trainierten Netz extrahierbar.

Zukünftig sollen Fertigungszellen über die Fähigkeit, selbständig Agieren und Planen zu können hinaus, sich der unerreichten Flexibilität des Menschen annähern. Das Ziel sind hochflexible Fertigungszellen, die die Fähigkeit besitzen, schrittweise Verhaltensweisen zu verbessern.

6.1 Analogie neuronaler Netze in Biologie und Steuerungstechnik

Im folgenden wird kurz auf Referenzmodelle für konnektivistische Architekturen - die neuralen Netzwerke - eingegangen. Die Nachbildung der Funktionsweise des Gehirns bzw. menschlicher Denkvorgänge wird am Beispiel einer **FFBP**-Netzwerkarchitektur (Feed-Forward-Back-Propagation, s.u.) erläutert.

6.1.1 Einzelneuron und Verschaltung

Ein neuronales Netz ist gekennzeichnet durch eine Verschaltung gleichartiger Elementarzellen mit jeweils mehreren Eingängen und einem Ausgang, ähnlich der Verschaltung von einfachen TTL-Gattern (Transistorlogik, jedoch FAN-IN und FAN-OUT beliebig). Diese Elementarzellen werden sowohl in der Physiologie, als auch bei der Modellierung des neuronalen Netzes am Rechner als Neuronen bezeichnet. Neuronen sind am Rechner als isolierte Recheneinheiten realisiert, in der Biologie, z.B. im menschlichen Gehirn, entsprechen sie den Nervenzellen einschließlich Zellkern, verästelter Verbindungen und Fortsätzen.

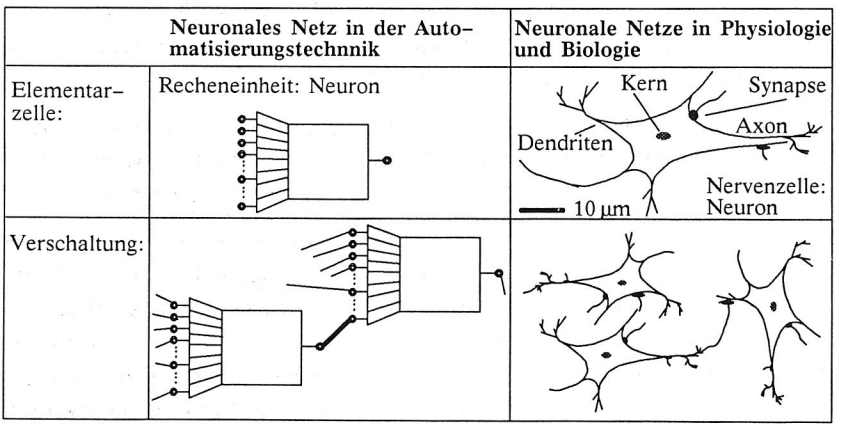


Abb. 6.1: Analogie in Automatisierungstechnik und Neurophysiologie (Teil 1); Physiologie (rechte Tabellenspalte) nach Schmidt /108/

Neuronen sind morphologisch und funktionell eine eigenständige Grundeinheit des Nervensystems. Die "Verschaltung" der Nervenzellen z.B. im menschlichen Gehirn erfolgt über sogenannte Dendriten (Eingänge). Synapsen sind die Kontaktstellen und können entweder auf Dendriten oder direkt auf dem Zellkörper sitzen. Axone fungieren als astartig verzweigte Ausgänge (Abb. 6.1).

Neuronen können mit sensorischen (Eingabe-) oder motorischen (Ausgabe-) Zellen verbunden werden (Randelement). Sie können aber auch vollständig im Netzinneren eingebunden sein und nur zur Informationsverarbeitung dienen (keine sensorischen/ motorischen Ein-/Ausgänge, vgl. Abb. 6.5).

Die sehr große Anzahl von Verbindungen (Größenordnung 10^{14} im menschlichen Gehirn, ein Neuron ist durchschnittlich mit 10 000 weiteren Neuronen verbunden) muß im Bereich der Steuerungstechnik und der Abbildung auf dem Rechner natürlich eingeschränkt werden. Simulation des neuronalen Netzes auf konventionellen Rechnern (von Neumann-Architektur) zwingt zu einer Reduzierung auf eine entsprechend geringe Anzahl von Neuronen und eine Untergliederung in ein- oder mehrlagige Schichten, die vollständig oder teilweise vernetzt sind. Die Beschränkung der Netzkomplexität aufgrund fehlender Rechenleistung bzw. noch nicht verfügbarer Hardware für Parallelverarbeitung (NN-Chip) stellt derzeit die Hauptschwierigkeit dar. Dies zeigt sich vor allem bei der Erzeugung von Bewegungen mit sechs räumlichen Freiheitsgraden (vgl. Kap. 7.2). Abb. 6.2 zeigt die Topologie eines neuronalen Netzes an einem Beispiel.

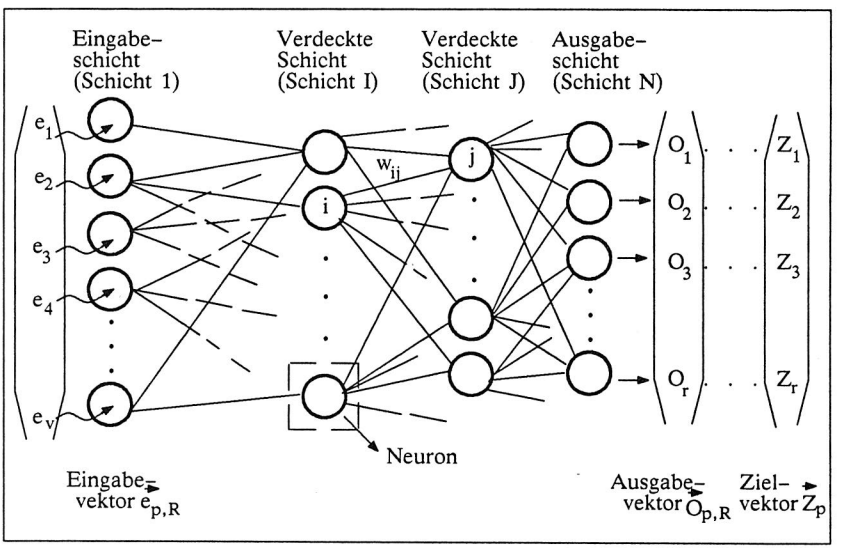


Abb. 6.2: Topologie eines n-schichtigen neuronalen Netzes

Das gesamte Netz besteht aus einer Eingangsschicht, einer Ausgangsschicht und einer oder mehreren verdeckten Schichten. Die Verarbeitungseinheiten (Neuronen) der verdeckten Schichten werden auch "hidden units" genannt und besitzen keinen direkten Anschluß nach außen. In ihnen und ihren Verbindungen ist das gesamte, in der Lernphase aufgenommene Wissen gespeichert. Neuron j ist ein Neuron in der Schicht

J, Neuron i ein Neuron in der Schicht I. Schicht J folgt auf Schicht I. Der Netzeingangsvektor $\vec{e}_{P,R}$ wird von außen eingespeist. Abb. 6.3 zeigt ein derartiges Neuron im Detail.

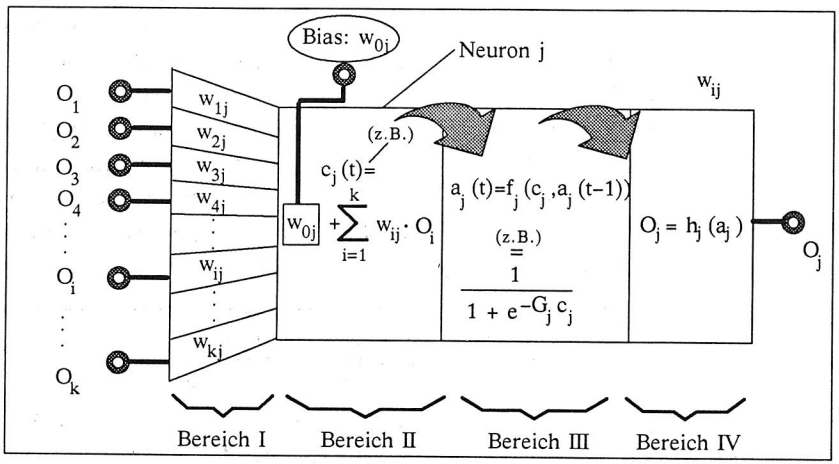


Abb. 6.3: Mathematische Nachbildung der Eigenschaften eines j-ten Neurons in Schicht J

Die Unterteilung jedes Neurons in die Bereiche Gewichtungsfaktoren (Bereich I), Eingangsfunktion (Bereich II), Aktivitätsfunktion (Bereich III) und Ausgangsfunktion (Bereich IV) entspricht dem biologischen Vorbild (vgl. Abb. 6.4).

An den k Eingängen des Neurons liegen k Signale an, entsprechend den über die Dendriten zum Zellkern einlaufenden aktivierenden und hemmenden Erregungen. Die Anzahl der Eingänge hängt dabei von der Verschaltungsart der Neuronen ab und ist damit für eine gegebene Netztopologie festgelegt. Die Signale o_j (j = 1, ..., k) bilden den k-dimensionalen Eingangsvektor $\vec{e}_p$ für ein bestimmtes Muster p. Eine Schicht j ist vollständig verschaltet mit Schicht (j-1), falls gilt: k = Anzahl der Neuronen aus Schicht (j-1).

Ebenso bilden die pro Eingang vorgegebenen Gewichtsfaktoren w_{ij} den k-dimensionalen Gewichtungsvektor $\vec{w}$ (= Maß für die Empfindlichkeit). w_{ij} entspricht einem gewichteten Anschluß (Synapse) bzw. ist unter der Annahme verästelungsfreier Verknüpfungen (entgegen der Darstellung in Abb. 6.1) identisch mit einer gewichteten Verknüpfung der Neuronen i und j.

Es gilt : $w_{ij} < 0$: hemmender (inhibitorischer) Einfluß
$w_{ij} > 0$: erregender (exzitatorischer) Einfluß
$|w_{ij}|$: Bedeutung der Verbindung (je größer $|w_{ij}|$ desto wichtiger)
$w_{ij} = 0$: irrelevante Verbindung

$\vec{w}$ stellt einen Spezialfall einer Verknüpfungsmatrix $\underline{W}$ dar, mit der sich mehrere, unterschiedliche Verknüpfungsarten in einem Netzwerk beschreiben lassen.

	Neuronales Netz in der Automatisierungstechnnik	**Neuronale Netze in Physiologie und Biologie**
Neuronenzahl:	Größenordnung <200 [1)]	$\approx 2.5 \times 10^{10}$
Technologie:	Halbleiter (Transistorlogik)	chemisch, biologisch
Realisierung:	Von-Neumann-Architektur digital, sequentiell parallelisierbar (Multiprozessorsystem)	analog, massiv parallel Reaktionszeit eines Neurons relativ langsam
Verbindungsanzahl:	z.B. 3-schichtiges Netz mit insgesamt k+l+m Neuronen k x l + l x m + k + l +m	$\approx 5 \times 10^{13}$ (Zahlenwert nach /108/)

Abb. 6.4: Analogien neuronaler Netze (Teil 2)

Die Eingangsfunktion (z.B. Summation der positiven wie negativen Eingangssignale) stellt die Propagierungsregel dar und dient zur Vermittlung von Aktivierungsmustern durch das Netzwerk. Das Skalarprodukt dieser beiden Vektoren ergibt den Inputimpuls c_{pi} (t) des Neurons (Erregung) auf das gegebene Muster p zum Zeitpunkt t.

$$c_{pj} = \sum_{i=1}^{k} w_{ij} \cdot O_{pi} + w_{oj} = \vec{w}_j^T \cdot \vec{O}_{pI} + w_{oj} \qquad (6.1)$$

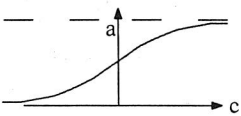
Aus dem resultierenden Inputimpuls c_{pi} sämtlicher Inputs der Einheit i und dem alten Aktivitätsniveau $a_{pi}(t-1)$ wird der aktuelle Aktivitätszustand bestimmt. a_{pi} zeigt den internen Zustand des Neurons an, d.h. ob es aktiv ist oder sich eher im Ruhezustand befindet. Dem Aktivitätszustand können binäre, diskrete und stetige Wertebereiche (z.B. Untermengen von $\mathbb{R}$) zugeordnet werden. Der additive Offset w_{oj} kann inter-

1) Stark abhängig von Rechnerarchitektur und Rechenleistung (ca. Obergrenze in der vorliegenden Arbeit)

pretiert werden als der synaptische Wichtungskoeffizient zu einem imaginären "bias-Neuron", dessen Aktivitätszustand immer zu 1 gesetzt wird. Der Koeffizient w_{0j} kann analog zu den Gewichten w_{ij} gelernt werden und repräsentiert eine Grundaktivität, die auch dann vorhanden ist, wenn keine Signale von anderen Neuronen eintreffen.

Neurophysiologisch beschreibt die Intensität die Anzahl der Reizimpulse (sogenannte Aktionspotentiale, s. /108/) pro Zeiteinheit, die in einem Nerv auftritt. Diese Impulsfrequenz ist eine Funktion der Reizstärke und besitzt einen s-förmigen Verlauf. In der Simulation wird dies meist durch einen Exponentialansatz (Sigmoidalfunktion) modelliert.

$$a_j(t) = a_j = f_j(c_j, a_j(t-1)) = \underset{\text{Beispiel}}{\Uparrow} \frac{1}{1 + e^{-G_j c_{pj}}} \qquad (6.2)$$



Gleichung (6.2) beschreibt einen Spezialfall mit nur einem Verknüpfungstyp. Diese nichtlineare Funktion muß stetig und differenzierbar sein. G_j stellt einen Verstärkungsfaktor dar (meist 1) und gibt die Steilheit der Aktivierungsfunktion an.

Durch das asymptotische Verhalten von f wird die Aktivität eines Neurons begrenzt und Sättigungserscheinungen bei zu großen Eingangswerten vermieden ($a \approx 0$: inaktives Neuron, $a \approx 1$: aktives Neuron).

Die Ausgangsfunktion h_j ergibt den Output $o_{pj}(t)$ zum Zeitpunkt t. h_j ist meist eine Schwellwertfunktion mit dem Schwellwert s. Dieser kann festgelegt oder durch das Training modifiziert werden.

Der Output eines Neurons j lautet im Beispiel der Einheitsfunktion:

$$o_{pj} = \left[1 + \exp\left(-G_j\left(\sum_{i=1}^{k} w_{ij} \cdot O_{pj}\right) + w_{oj}\right)\right]^{-1} \qquad (6.3)$$

Er wird als Ausgangswert an alle nachfolgend angeschlossenen Zellen geleitet. Der berechnete Netzausgangsvektor $\vec{o}_{P,R}$ kann an der Ausgabeschicht abgegriffen werden.

6.1.2 Backpropagation 1): Delta-Rule für Multilayer-Netzwerke

In biologischen Systemen ist die Vernetzung der Neuronen nicht starr. Verbindungen und Gewichtungen ändern sich ständig, indem wichtige Verbindungen wachsen ("Hinzulernen") und nicht benötigte Verbindungen verkümmern ("Vergessen").

1) Verallgemeinerte 'Fortschaltung'

Ziel ist es, mit Hilfe einer Lernvorschrift die Adaption des Netzverhaltens an die Problemstellung zu erreichen (Abb. 6.5). Für Trainingsprozesse eines neuronalen Netzes sind mehrere Lernregeln bekannt. Meist wird das sogenannte Backpropagation-verfahren nach Rumelhart /103/, /104/ verwendet. Im Ausgangszustand werden alle Gewichtfaktoren w_{ij} aller Neuronen zufällig voreingestellt (randomisiert). Diese Initialisierung ist für den erfolgreichen Anlauf des Lernvorganges erforderlich.

Anschließend wird ein Eingangsmuster $\vec{e}_p$ an das neuronale Netz angelegt. Entsprechend der internen Koeffizienten und der Aktivitätsfunktion stellt sich an der Ausgabeschicht N ein zugehöriges Ausgangsmuster $\vec{o}_p$ ein. Dieses Ausgangsmuster wird mit dem gewünschten Zielmuster $\vec{z}_p$ verglichen und der Fehler

$$F_p = \frac{\sum_{j=1}^{r} (Z_{pj} - O_{pj})^2}{2} \gtrless S \qquad (6.4)$$

berechnet. Die Änderung der synaptischen Gewichte nach diesem Kriterium bedeutet "überwachtes Lernen" (Ein- **und** Ausgang bekannt).

Es sei J = N. $F_{j,p}$ ist der Fehler des j-ten Neurons in der Ausgangsschicht, zugehörig zu den beiden Mustern $\vec{e}_p$ und $\vec{z}_p$ (Methode der kleinsten Quadrate). Dabei gilt der Output des j-ten Neurons nur dann als fehlerhaft (d.h. ein Lernvorgang wird erforderlich), falls $F_{j,p}$ eine Fehlerschranke S überschreitet.

Der Gesamtfehler (**TSS** = **T**otal **S**um of **S**quares) über alle Lernvektoren ergibt sich zu:

$$F = \sum_{p=1}^{g} F_p \qquad , g = \text{Anzahl der verfügbaren Lernpatternpaare} \qquad (6.5)$$

Rumelhart zeigt, daß bei Änderung der Gewichtskoeffizienten w_{ij} gemäß:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta p\, w_{ij} \qquad (6.6)$$

der quadratische Fehler F minimiert wird. Die Gewichtsänderung $\Delta p\, w_{ij}$ wird jeweils lokal für die Verknüpfung der Neuronen i <-> j berechnet.

Die Suche nach synaptischen Gewichten w_{ij}, so daß die quadratische Fehlersumme über alle Ausgabeneuronen und alle Muster minimiert wird, stellt für praxisrelevante Netzwerke ein komplexes Problem dar. Zur Lösung dieses Problemes lassen sich die, aus der nichtlinearen Optimierung bekannten, numerischen Gradientenverfahren anwenden. Die Übertragung dieser Gradientenverfahren auf die Hyperfläche

$F = F(w_{ij})$ (Fehlergebirge) lautet wie folgt:

$$\Delta_p w_{ij} \sim -\frac{\partial F_p}{\partial w_{ij}} \tag{6.7}$$

d.h. Abstieg in Richtung lokaler Minima der Hyperfläche

Anschaulich: Der Fehler F_p soll bezüglich jeder Verknüpfung minimal werden.

Die partielle Ableitung ergibt mit Gl. (6.4) für das j-te Neuron der Ausgangsschicht:

$$\frac{\partial F_p}{\partial w_{ij}} = \frac{\partial F_p}{\partial c_{pj}} \cdot \frac{\partial c_{pj}}{\partial w_{ij}} \tag{6.8}$$

Faktor 1 Faktor 2

Faktor 1 kann weiter aufgelöst werden zu:

$$\frac{\partial F_p}{\partial c_{pj}} = \frac{\partial F_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial c_{pj}} = (O_{pj} - Z_{pj}) \cdot f_j'(c_{pj}) = \delta p_j \quad \text{mit } f_j'(c_{pj}) = \frac{\partial f_j}{\partial c_{pj}} \tag{6.9}$$

δp_j gibt die Empfindlichkeit des Fehlers F_p bezüglich Änderungen des Eingangsimpulses des j-ten Neurons für Muster p wieder. Faktor 2 in Gl. (6.8) berechnet sich mit Gl. (6.1) zu:

$$\frac{\partial c_{pj}}{\partial w_{ij}} = o_{pi} \tag{6.10}$$

Die Anpassung des Gewichtskoeffizienten w_{ij} lautet nach Gleichung (6.7) wie folgt:

$$\Delta p, w_{ij} = \eta \cdot \delta p_j \cdot o_{pi} \tag{6.11}$$

Anschaulich bedeutet dies, daß die Änderung der Koeffizienten w_{ij} des Neurons j proportional zum Einflußgrad erfolgt, den dieses Neuron auf den Ergebnisvektor besitzt, multipliziert mit einem Maß für die tatsächliche Beanspruchung der zu w_{ij} gehörigen Signalleitung zwischen Neuron i und j der vorgelagerten Schicht bezüglich Muster p.

Proportionalitätsfaktor η bestimmt die Feinheit und damit die Geschwindigkeiten mit der w_{ij} geändert werden. Die Lernrate η wird extern eingestellt, wobei der zum Abschluß des Trainingslaufes eingestellte Wert η^* die erzielte Genauigkeit des neuronalen Netzes festlegt.

Lernrate η ist dann optimal, wenn sie maximal ist, jedoch der Fehler noch keine Schwingungen zeigt.

Für die praktische Anwendung eines Neuronalen Netzes reicht Gleichung (6.11) jedoch nicht aus, da sie nur für die Anpassung der Wichtungskoeffizienten der Ausgangsschicht gilt. Reale Neuronale Netze, z.B. für die Steuerung der Feinmotorik eines Industrieroboters, erfordern mehrere hidden units in mehreren hidden layers, für die, im Gegensatz zur Ausgangsschicht, kein Zielvektor bekannt ist. Gleichung (6.11) muß deshalb so erweitert werden, daß sich auch für ein beliebiges Neuron in einer beliebigen Schicht die Wichtungskoeffizientenänderungen berechnen lassen. Dazu werden die Empfindlichkeiten δ_{pi} rückwärts durch das Netz geschleust ("Backward Propagation").

Gleichung (6.8) erneut angesetzt für Neuron y in Schicht Y mit Y = (N-1). Faktor 2 aus Gleichung (6.8) berechnet sich analog Gleichung (6.10). Für Faktor 1, d.h. für die Empfindlichkeit eines beliebigen Neurons y in der der Ausgangsschicht vorangegangenen Schicht gilt jetzt mit Gleichung (6.4):

$$\delta_{py} = \frac{\partial F_p}{\partial c_{py}} = \left[\sum_{n=1}^{r}\left(\frac{\partial F_p}{\partial o_{py}}\right)_{"n"}\right] \cdot \underbrace{\left(\frac{\partial o_{p,y}}{\partial c_{p,y}}\right)}_{= f'_y(c_{p,y})} \tag{6.12}$$

$$\left[\sum_{n=1}^{r}\left(\frac{\partial F_p}{\partial o_{py}}\right)_{"n"}\right] = \sum_{n=1}^{r}\left(\frac{\partial F_p}{\partial c_{p,n}} \cdot \frac{\partial c_{p,n}}{\partial o_{p,y}}\right) \tag{6.13}$$

Zu beachten ist, daß y jetzt kein Output-Unit mehr ist, und $c_{p,n}$ sich auf das Y nachfolgende Layer N bezieht !
Mit Gleichung (6.1) gilt weiter:

$$\sum_{n=1}^{r}\left(\frac{\partial F_p}{\partial c_{p,n}} \cdot \frac{\partial c_{p,n}}{\partial o_{p,y}}\right) = \sum_{n}\left(\frac{\partial F_p}{\partial c_{p,n}} \cdot \frac{\partial}{\partial o_{p,y}} \sum_{n}\left(o_{py} \cdot w_{yn}\right)\right) =$$

$$= \sum_{n} \frac{\partial F_p}{\partial c_{p,n}} \cdot W_{yn} = \sum_{n} \delta_{pn} \cdot W_{yn} \tag{6.14}$$

Damit folgt:

$$\Delta p, w_{xy} = \eta \cdot o_{p,x} \cdot \underbrace{f'_y(c_{p,y}) \cdot \sum_{n}\left(\delta_{pn} \cdot W_{yn}\right)}_{\delta_{py}} \tag{6.15}$$

Gleichung (6.15) läßt zusammen mit Gleichung (6.9) die Definition eines rekursiven Algorithmus zu und damit einer Rechenvorschrift zur Bestimmung aller Gewichtskoeffizientenänderungen der Netzneuronen aller Schichten (einschl. hidden units) für ein Trainingsmuster p. Nacheinander werden alle verfügbaren Lernvektoren (Input- und

Zielvektoren) entsprechend dem oben beschriebenen Verfahren an das Neuronale Netz angelegt. Die gesamten c Lernmuster werden paketweise in sogenannten Epochen trainiert, beginnend mit randomisierten Gewichtsfaktoren. Innerhalb jeder Epoche wird der aktuelle einzulernende Satz von Lernvektoren zufällig angewählt, damit "Einbrennvorgänge" durch feste Reihenfolgen verhindert werden.

	Neuronales Netz in der Automatisierungstechnnik	**Neuronale Netze in Physiologie und Biologie**
Informationsverarbeitung:	Nichtalgorithmischer Ansatz, Abbildung auf Algorithmus Modelle mit Selbstorganisation	Nichtalgorithmisch, Training von Verhaltensweisen, Permanentes Lernen
Kontaktierung:	Gewichtskoeffizienten w_{ij}	Synapsen: 'chemische' Kontaktstellen 
Training:	z.B. Backpropagation Δw_{ij} (Keine Grundfunktionen)	Training z.B. Grund-Feinmotorik
Motorik:	z.B. Bewegungssteuerung IR, 6 - 8 NC-Stellglieder	Motorische Ausgabeneuronen, 256 Muskeln
Sensorik:	z.B. Winkelgeber, optische Abstandssensoren, etc.	Sensorische Eingabeneuronen. z.B.Tastsinn

Abb. 6.5: Analogien in Automatisierungstechnik und Neurophysiologie (Teil 3)

Die Trainingsphase wird abgebrochen, falls für alle c Lernmuster die Fehlerschranke S (Gleichung (6.4)) nicht mehr überschritten wird (Konvergenz). Die Fehlerschranke kann einzeln oder für ein Lernmusterpaket definiert werden. Experimentelle Untersuchungen von Rumelhart/McClelland haben gezeigt, daß durch einen zusätzlichen Term:

$$\Delta_p w_{ij}(t+1) = \eta \cdot \delta_{p_j} \cdot o_{pi} + \alpha \cdot \underbrace{[\Delta_p w_{ij}(t)]}_{\text{Einfluß der letzten Änderung}} \qquad (6.16)$$

bei hoher Lernrate die Schwingungen des Gesamtfehlers durch einen hohen Wert von α ($\approx 0{,}9$) gedämpft werden können. Das System lernt schneller (s. Kap. 6.2).

Nachdem der Trainingslauf abgeschlossen wurde, steht das Neuronale Netz für den "Betriebsmodus" bereit. Ein "fertig trainiertes" Neuronales Netz benötigt zwischen Anlegen des Eingangsvektors und Berechnung des Ausgangsvektors, je nach Hard-

ware, Rechenzeit in der Größenordnung von weniger als 1ms. Der zeitaufwendige Teil liegt also in der Trainingsphase, die jedoch als Hintergrundprozeß über Nacht gestartet werden kann und nur einmalig für eine Problemklasse anfällt.

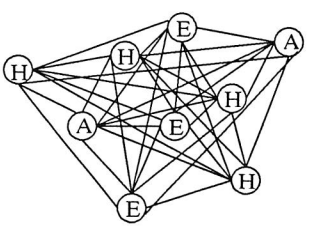
6.1.3 Maschinelle Lernverfahren

Neuronale Netzwerke werden z. B. in /18/ zur zerstörungsfreien Prüfung von Druckbehältern durch akustische Emissionen, sowie zur Inspektion von Lötstellen elektronischer Bauteile eingesetzt. Fast alle eingesetzten, künstlichen neuronale Netze (KNN) besitzen dabei folgende Gemeinsamkeiten (vgl. die in Kap. 6.1.1 und 6.1.2 behandelten Mechanismen):

- Neuronen mit Aktivitätszuständen und Netztopologie/Verschaltung
- Regel zur Fortschaltung von Aktivitätsmustern von Neuron zu Neuron
- Regel zur Aktivierung eines Neurons aus anliegenden Signalen und altem Aktivitätszustand
- Lernregel zur Modifikation der Verbindungen
- Anschluß an Peripherie (Hard-/Software)

Über die monolithischen Netze hinaus existieren hybride Konzepte, bei denen jede Verarbeitungseinheit Gesamtwissen repräsentiert und eigenständig arbeitet (one-unit-one-concept). Der Vorteil liegt in der Option der hohen Parallelität der Verarbeitung .

Neben dem in dieser Arbeit eingesetzten FFBP-Netzwerk gehört das Hopfield-Netz (eine Schicht mit Rückkopplung), das Perceptron (eine Schicht) mit dessen Modifikation CMAC (Cerebellar Model Articulation Controller), sowie das Multilayer-Perceptron (vorwärts-verbundene Schichten) in die Klasse der (Mehr-)Schichtenarchitekturen. Ein Vertreter voll-vernetzbarer, schichtenloser Netzstrukturen ist die Bolzmann-Maschine. Sie besitzt festgelegte Ein- und Ausgabeneuronen und unstrukturierte Hidden-Units (Abb. 6.6).



Legende:

E: Eingabeneuronen
A: Ausgabeneuronen
H: versteckte Neuronen

Abb. 6.6: Bolzmann-Maschine

Ein wichtiges Netzwerkmodell ist das Modell von Kohonen (Topographische Merkmale durch **Selbstorganisation**). In /132/ wird beispielsweise ein Kohonen-Netz für die Plazierung elektronischer Bauelemente im Leiterplattenentwurf eingesetzt. In /99/, /100/ bilden die 2D-Daten zweier CCD-Kameras die Eingangsgröße eines mehrdimensionalen Kohonen-Netzes mit lokal linearisierten Verhalten in einem Neuron ('lokale Jakobimatrix') zur simulierten Positionierung (ohne Orientierung) eines IR-TCPs. Weitere Modelle bzw. theoretische Grundlagen finden sich in /49/, /103/, /99/, /48/.

6.2 Wissensbasiertes CAE-Entwicklungswerkzeug für den Einsatz neuronaler Netzwerke

Generell gilt, daß z.B. FFBP, wie auch das unüberwacht lernende Kohonen-Modell, noch viel Erfahrung und Intuition des Anwenders benötigt (beim aktuellen Stand der Technik). Die wesentlichen Grundregeln und Zusammenhänge werden in diesem Kap. 6.2 beschrieben. Für den praktischen Einsatz in der Fertigungsautomatisierung müssen deshalb Konzepte entwickelt werden, zum Aufbau einer problemnahen CAE-Verfahrenskette für neuronale Automatisierungslösungen.

Die Entwicklung einer neuronalen Lösung für eine spezielle Automatisierungsaufgabe (z.B. in der Steuerungstechnik) erfordert eine große Anzahl zu definierender Parameter (einschließlich 'Try- and Error-Testläufe'). Kenntnisse und Erfahrungen des Anwenders müssen hierzu in den in Abb. 6.7 gezeigten Teilschritten einfließen. Ein Werkzeug für Entwurf und Adaption eines Neuronalen Netzes an eine konkrete Aufgabenstellung sollte diese Punkte unterstützen.

Die einzelnen Schritte sind meist nicht unabhängig voneinander behandelbar, sondern in einen Entwicklungszyklus eingebunden. Beispielsweise kann eine spezielle Hardware eine spezielle Aktivierungsfunktion bedingen. In diesem Teil der Arbeit soll ein Werkzeug entwickelt werden, das die Entwicklungskette von der Adaption an das Problem über Training bis zur Hardwarerealisierung unterstützt.

Für die Realisierung von Lernprozessen in der Fertigung muß der Entwurf neuronaler Applikationen durch das Werkzeug bereits zum Zeitpunkt der Problemdefinition unterstützt werden. Training kann interaktiv oder systemgesteuert als Hintergrundprozeß ablaufen. Netzstruktur, I-/O-Pattern und Initialisierungen sind Eingangsdaten. Lern-Protokolle und die im Training erzeugten Gewichte können gespeichert , bzw. für den

Betrieb des Netzes wieder geladen werden. Speichern und schnelles Wiederfinden von umfangreichem, neuronalem Wissen (Trainings- und Zwischenergebnisse nach Abb. 6.7) sollte in Datenbanken erfolgen (s. Kap. 5.2.5).

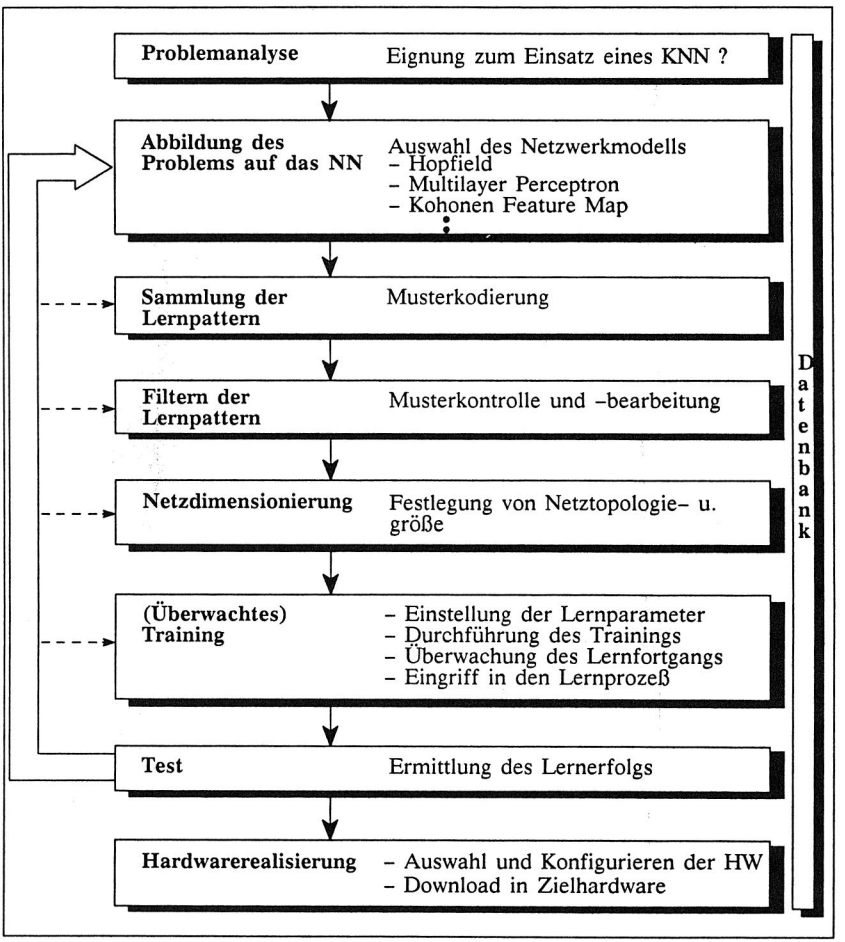


Abb. 6.7: Entwicklungsschritte für den Einsatz neuronaler Netzwerke

Da die Gewichte vor jedem Trainingsablauf zufällig gesetzt werden, ist die Konvergenz des totalen Fehlers F nicht gesichert. Auch bei einer hohen Anzahl von hidden

units kann F bei ungünstiger Initialisierung der Gewichte in einem lokalen Minimum festlaufen, so daß Trainingsläufe event. mehrmals wiederholt werden müssen. Je größer η eingestellt wird, desto größer ist die Gefahr, daß Minima nicht gefunden werden, und eventuell keine Konvergenz stattfindet. Konvergiert der Trainingslauf, so verkürzt sich in diesem Fall die Zeit für das Training.

Je nach Gewichtsvorbelegung verändert sich selbst bei feststehender Neuronenzahl die Dauer der Trainingsläufe. Bei einer geringen Anzahl hidden units läßt sich bei ausbleibender Konvergenz (trotz wiederholter Trainingsläufe) häufig nicht eindeutig bestimmen, ob der Grund hierfür in einer ungünstigen Gewichtsinitialisierung oder in einer zu geringen Neuronenzahl liegt. Die Geschwindigkeit der Konvergenz (!) ist grundsätzlich stark von der Gewichtsinitialisierung abhängig.

Im Falle von Neuronen, die nur diskrete Zustände, z.B. "0" oder "1", annehmen können, kann die Trainingszeit dadurch verkürzt werden, daß anstatt auf die Extremwerte auf Werte wie "0,15" (für "0") und "0,85" (für "1") trainiert wird /103/. Dies ist jedoch nur möglich, falls die Eingabe nicht verrauscht und die Signifikanz der einzelnen Eingabebits hoch ist. In der Betriebsphase kann für die einzelnen Ausgabeneuronen die Entscheidungsschranke auf beispielsweise "0,5" gesetzt werden (vgl. Fuzzy-Sets). Eine weitere Möglichkeit der Reduzierung neuronaler Netze besteht darin, falls die Applikation dies zuläßt, statt Absolutwerten nur Tendenzen zu verarbeiten (vgl. Kap. 7.2.2).

6.2.1 Problemadaption und Kontrolle

Konventionelle Berechnung und Neuronale Netze sollten nicht konkurrieren, da beim derzeitigen Stand der Technik Neuronale Netze nicht für Aufgaben eingesetzt werden sollten, für die konventionelle Verfahren prädestiniert sind (z.B. Numerik). Der erste Schritt ist deshalb die Überprüfung des Problems (Abb. 6.7) hinsichtlich Eignung zum Einsatz eines neuronalen Netzes (Faustregel: Dort einsetzen, wo auch der Mensch Stärken zeigen würde - z.B. Bewegungskoordination).

Während der gesamten Entwicklungskette (Abb. 6.7) muß der Benutzer in der Lage sein, aktuelle Trainingszustände zu beobachten, sowie über Kontroll- und Eingreifinstrumente bezüglich des Lernprozesses verfügen.

Die Auswahl des Netzwerkmodells kann nach den Anhaltspunkten erfolgen: Mustererkonstruktion mit Hopfield-Netz, Mustererkennung mit FFBP und Kohonen-Feature-

Maps für das Finden von Gemeinsamkeiten in Eingangsdaten. Meist erleichtert eine einfache Datenvor- und/oder Nachverarbeitung (z.B. Umkodierungen) oder geeignete Datenreduktion (z.B. in /87/ oder Kap. 7.2) das Lernen.

Ein Expertensystem kann den Anwender bei folgenden Entscheidungen unterstützt:

- ◆ Neuronales Netz: Ja/Nein
- ◆ Geeignetes Netzwerkmodell (z.B. datenbankgestützte Verwaltung charakteristischer Eigenschaften realisierter Anwendungen : Netzwerktyp, -topologie, etc. s. Kap. 5)
- ◆ Festlegung der Ausgangs- und Eingangsschicht durch Anschluß des NN an die Umgebung (Sensorik, Aktorik)

Ein derartiges Entwicklungswerkzeug kann in eine Master-Slave-Konfiguration strukturiert werden (Abb. 6.8). Die zentrale Kontrollinstanz, der Master, überwacht und steuert die Abfolge der einzelnen Entwicklungsschritte (s.o.). Die untergeordneten Slaves bewältigen die jeweiligen Entwicklungsschritte von Mustergewinnung bis Hardwaredownload. Sie arbeiten unabhängig voneinander und kommunizieren ausschl. mit dem Master. Der Master versorgt die Slaves mit Daten, kontrolliert deren Zwischenergebnisse und steuert deren Weiterverarbeitung. Abb. 6.14 zeigt die Instrumente zur Beobachtung laufender Lernprozesse und charakt. Kenngrößen.

Der Master regelt die Kommunikation mit dem Benutzer. Über eine graphische Bedienoberfläche besteht Zugriff auf die einzelnen Slaves. Der Master kann auf Expertenwissen zugreifen, um in die Abarbeitung obiger Entwicklungsschritte einzugreifen, indem er etwa gescheiterte Lernversuche analysiert und Abhilfen aufzeigt.

Vorteilhaft für die Realisierung des oben beschriebenen Werkzeuges ist der Einsatz von Expertensystem-Tools, wie z.B. KEE (/59/, /130/). In der vorliegenden Arbeit wurde die Softwareentwicklungsumgebung **KEE** (**K**nowlegde **E**ngineering **E**nvironment) eingesetzt, auf Apollo bzw. HP-Workstation (im Token-Ring-Verbund) unter AEGIS/UNIX. KEE besitzt starke Objektorientierung und baut auf der Programmiersprache **LISP** (**List-P**rocessing) auf. KEE verfügt über leistungsfähige 2D-Graphik, Regeln als Wissensrepräsentationsformen, logische Programmierung und das sog. Foreign-Language-Interface /59/ mit dem sich Slaves, z.B. der aus Laufzeitgründen in C implementierte Slave 'Lernen', einbinden lassen. KEE zeigte sich bzgl. der Anforderungen eines NN-Entwicklungstools auf Workstationebene als geeignetes, wissensbasiertes Werkzeug. Details und einordnende Vergleiche (z.B. mit Knowledge Craft, ART und Wissenserwerbswerkzeug, z.B. CONSTRUCT) siehe /43/, /64/.

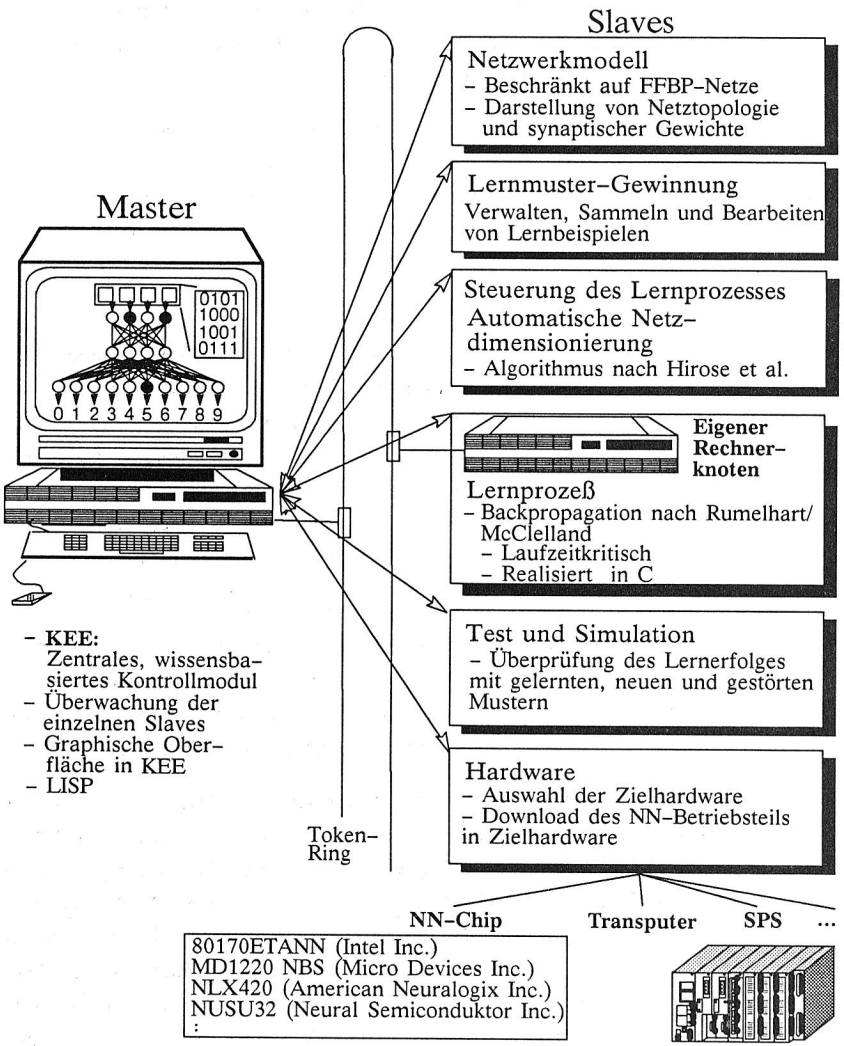


Abb. 6.8: Verteiltes Master-Slave-Konzept

Für den Entwurf eines neuronalen Netzwerkes ist die Kombination mit wissensbasierten Methoden aufgrund der großen Anzahl festzulegender Parameter sehr gut geeignet. Das gilt vor allem für das Kontrollmodul Master im Master-Slave-Konzept, das

vollständig in LISP realisiert wurde. Abb. 6.8 zeigt das grundlegende Konzept für ein Werkzeug zur Entwicklung neuronaler Systeme bis hin zur Hardwarerealisierung von Neuronalen Netzen.

In der vorliegenden Arbeit wurden nur wesentliche Slaves exemplarisch realisiert. Durch KEE wird die erforderliche Flexibilität bereitgestellt Slavegrenzen zu verschieben oder neue Slaves zu bilden (z.B. eigener Slave Parametrisierung).

Unterschiedliche Slaves besitzen unterschiedliche Anforderungen an die HW. So sollten Slaves auf dafür geeigneten Rechnern ablaufen. In dem hier entwickelten Tool wurde der zeitintensive Slave 'Lernprozeß' auf den schnellsten (Numerik!), im Token-Ring verfügbaren Rechner als Hintergrundprozeß verteilt. Master und die restlichen Slaves laufen ebenfalls auf einem eigenständigen Rechnerknoten. Die Kommunikation des Masters mit dem Slaves 'Lernprozeß' erfolgt über Interprozeßkommunikation (vgl. Abb. 6.8).

6.2.2 Dimensionierung, Parametrierung und Training

Von besonderer Bedeutung ist die Beschaffung der Lernmuster. Am Beispiel eines BV-Binärbildes steigt der Aufwand für Training und Verwaltung mit höherer Auflösung. Geringere Auflösung führt zu geringerer Unterscheidbarkeit und eventuell zu Konvergenzproblemen beim Lernen. Die Fähigkeit verrauschte Eingangsdaten noch ausreichend zu klassifizieren nimmt bei reduzierten Bildern ab.

Bei der Aufnahme der Trainingsstichprobe darf die Anzahl der Lernpattern nicht zu groß (Trainingsaufwand) und nicht zu klein (geringe Erkennrate) gewählt werden. Es muß festgelegt werden, ob nur "Gutmuster" oder zusätzlich Stördaten, bzw. Rauschen in den Lernprozeß eingebracht werden sollen.

Neben der Musterverwaltung und Mustermanipulation (Einfügen neuer Muster, Löschen von Mustern, Verändern bestehender Muster), sowie der Handhabung von Probeläufen, sollte auch die Aufnahme der benötigten Lernmuster, z.B. direkt aus dem realen, laufenden Prozeß, und die Einleitung in das neuronale Netz unterstützt werden.

Im Falle von FFBP-Netzen (im Gegensatz zu Kohonen-Netzen) müssen deshalb Eingangs- als auch die zugehörigen Ausgangsmuster aufgezeichnet werden. Am Beispiel des Lernens der IR-Feinmotorik (Kap. 7.2) wird eine Folge von definierten Ein- und Ausgangsmustern dadurch erzeugt (und vom neuronalen Netz nachgelernt), daß Sen-

sorwerte gleichzeitig mit der ausgeführten Feinbewegung eines manuell programmierten Industrieroboters während der Offline-Simulation aufgezeichnet werden.

Die richtige Netzdimensionierung, d.h. die optimale Anzahl von Neuronen und deren Verbindungen für eine gegebene Aufgabe der Fertigungsautomatisierung ist vorher nicht absehbar. Häufig ist für die Wahl der Netzgröße eine Vielzahl von Testläufen und Erfahrung erforderlich. Minimale Konfigurationen sind z.B. optimal bezüglich des Hardwareaufwandes bei der nachfolgenden Realisierung. Zu klein dimensionierte Netze führen dagegen zu Konvergenzverlust. Das im Rahmen dieser Arbeit vorgestellte Entwicklungswerkzeug bietet besonders hier Unterstützung durch automatische Ermittlung der minimalen Neuronenzahl. Hierzu wurde ein Verfahren von Hirose, Yamashita et. al. /50/ verwendet.

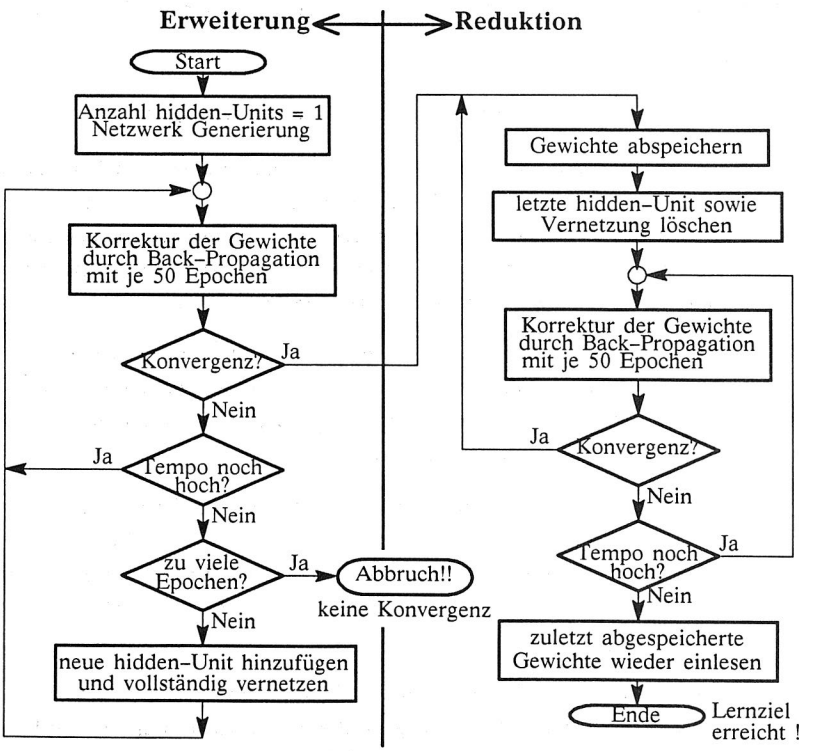


Abb. 6.9: Algorithmus nach Hirose, Yamashita und Hijiya /50/

Dieser Algorithmus ändert selbständig die Netzkonfiguration, um das Lernziel zu erreichen. Er setzt auf der Back-Propagation-Regel für Multilayer-Perceptoren auf (Flußdiagramm in Abb. 6.9). Der Anwender benötigt keine Kenntnisse über den Netzwerkaufbau.

Zunächst wird die Netz-Startkonfiguration festgelegt. Der Anschluß der Ein- und Ausgangsneuronen nach außen (Sensoren, Aktoren) wird durch eine Input- und eine Output-Retina repräsentiert (= rechteckige Sensor- und Aktorfelder, s.u.). Sie werden direkt von dort übernommen. Der Algorithmus startet mit einem verborgenen Neuron.

Das Ausgangsnetz wird mit einem festen Paket von 50 Epochen trainiert. Der Fortschritt des Trainings wird ausgenutzt, um Anzeichen zu finden, daß der Lernprozeß in einem lokalen Minimum festläuft (z.B. TSS, nach Kap. 6.1.2, bleibt über mehrere Lernepochen konstant). Die Veränderung ΔTSS wird rückblickend aus den letzten 50 Epochen berechnet. Für $\Delta TSS \leq 1\%$ wird ein lokales Minimum erkannt und versucht, das Lernziel durch Einfügen eines weiteren hidden units, das mit allen Ein- und Ausgangsneuronen vernetzt ist, zu erreichen. Die Anzahl der hidden units wird so sukzessive erhöht. Für $\Delta TSS \geq 1\%$ wird das neuronale Netz, in unveränderter Topologie, weitere 50 Epochen trainiert.

Wurde das Lernziel (TSS < Abbruchschwelle) erreicht, wird auf die anschließende Reduktionsphase umgeschaltet. Durch die "Befreiung" aus einem lokalen Minimum produziert der Algorithmus häufig zu viele hidden units (Überschwingen!). Diese überdimensionierten Netze werden deshalb anschließend reduziert (Abb. 6.9, rechts), indem Neuronen solange entfernt werden, bis gerade noch Konvergenz erreicht wird.

Die so ermittelten Minimalnetze können nur Ausgangspunkt für die Verarbeitung mit zusätzlichen Anforderungen sein, wie z.B. verrauschte Eingangsdaten oder Robustheit gegenüber partiellem Ausfall von Netzteilen, für die die minimalen Netzgrößen nicht ausreichen.

Bei fast jedem Lernverfahren müssen zahlreiche Parametereinstellungen vorgenommen werden, die über die Konvergenz des Lernens entscheiden. Für eine minimale Entwicklungszeit muß zwischen zu schnellem (event. Konvergenzverlust) und zu langsamen (Zeitbedarf) Lernfortgang optimiert werden. Der Einsatz des Entwicklungswerkzeuges in der Trainingsphase besitzt einen hohen Rationalisierungseffekt, da Parametrisierung, Training und dessen Überwachung einen Großteil der Entwicklungszeit festlegen.

Vorhandene Faustregeln zum Einstellen der Lernparameter lassen sich in ein wissensbasierte System einbringen (z.B. **wenn** Konvergenz fehlgeschlagen, **dann** Lernrate η verringern oder Strategien zur Überwindung lokaler Minima aktivieren z. B. durch bewußte, massive Störung).

Das Entwicklungswerkzeug sollte darüberhinaus Standardparametersätze bereithalten, die für die meisten Anwendungen einen zwar nicht optimalen, aber akzeptablen Lernprozeß sicherstellen und zu einer lauffähigen Applikation führen. Folgende Voreinstellungen der Lernparameter (Kap. 6.1) durch das wissensbasierte NN-Entwicklungstools sind sinnvoll:

- Lernrate η : $\eta \sim \frac{1}{\text{Anzahl der hidden units}}$; d.h. dynamische Anpassung an die Netzgröße (nach Ansatz von /85/, vgl. /117/)
- Momentum-Faktor α : $\alpha \approx 0{,}2$ (Zusammen mit /50/ ergaben sich bei großem α Probleme mit Oszillation und Überschwingen der Neuronenzahl).
- Verstärkungs-Faktor G ("Slope"): $G \approx 1{,}0$
- $(\Delta TSS)_{50,\ Schwelle} \approx 0{,}01$; (= "Tempo-Schwelle")des "Lerntempos" ΔTSS mit:
  $\Delta TSS = 1 - \frac{TSS}{TSS_{alt}}$ nach 50 Epochen .
  Fällt ΔTSS unter diesen Wert, so wird ein lokales Minima erkannt.
- TSS_{max} : (**Ecrit**, **E**rror **Crit**erium) maximal zulässiger Geamtfehler = Abbruchsgrenze (z.B. 0,25 (Binärmuster) ... 0,1) .
- $T_{max} \approx$ 0,05 ... 0,0 ;
  Abweichungen von den Idealwerten bei binären Muster für zu trainierende Zielwerte $0{,}0 + T_{max}$ und $1 - T_{max}$ (s.o.) (sigmoide Aktivierungsfunktion).
- Weitere Schwellwerte, wie die maximal zulässige Anzahl der Lernepochen ($\approx$10000), innerhalb deren ein Lernvorgang beendet sein muß.
- Maximale Anzahl der hidden units, die der Algorithmus nach Abb. 6.9 anlegen darf.
- Intervall [-g, +g] , aus dem die synaptischen Gewichte mit Zufallszahlen vorbesetzt werden.

Die durch Experimente bewährten Einstellungen und Regeln lassen sich in einem wissensbasierten Tool (z.B. KEE) repräsentieren und zum direkten Eingriff in den Lernprozess bzw. zur Unterbreitung von Vorschlägen auswerten.

Finden Lernprozeß und Betriebsphase auf unterschiedlicher Hardware statt, so kann auch für den Lernprozeß die geeignetste verfügbare Hardwarekonfiguration aus-

gewählt werden. Klar zu unterscheiden sind deshalb Zielhardware und Lernhardware. Eine gegebene Netztopologie kann auf einer Zielhardware (s. u.) betrieben werden, die für den Lernprozeß völlig ungeeignet wäre.

6.2.3 Test und Experiment

Mit dem prototypisch realisierten Werkzeug können während und nach dem Lernprozeß, durch Experimente an gelernten oder ungelernten Mustern oder Modifikationen davon, die Reaktionen des neuronalen Netzes getestet werden. Beim Test gelernter Muster kann direkt zwischen Soll- und Istreaktion verglichen werden (durch Einblendung beider Muster in die Output-Retina, Abb. 6.14, rechts unten).

6.2.4 Zielhardware und Download

Für den Einsatz eines erfolgreich trainierten Netzes ist es (in der Fertigungsautomatisierung) besonders wichtig, die Komponente/Gerät der automatisierten Anlage festzulegen, auf der das Netz betrieben werden soll (oder eventuell zu erstellende Hardware festzulegen). Die Zielhardware kann Netzwerk-, Neuronen- oder Synapsen-orientiert /86/ sein.

An einfach überprüfbaren, für neuronale Netze jedoch nicht triviale bzw. ungeeignete Benchmarks, wie z.B. das Exklusiv-Oder (XOR, linear trennbare Funktion), oder Realisierung eines Multiplexers können komplette Entwicklungszyklen mit dem Tool durchlaufen werden.

Wie bereits erwähnt sind neuronale Netze aufgrund der einfachen Kommunikation zwischen Neuronen und der parallelen Struktur für die Abbildung auf parallele Rechnerarchitekturen geeignet. Spezielle Hardware ("Neuro-Chips") ist jedoch derzeit noch unzureichend verfügbar. Sie ist häufig für spezielle Netzwerkmodelle ausgelegt, so daß die Hardwarerealisierung hier Einfluß auf die in Abb. 6.7 dargestellten Schritte des Entwicklungszykluses besitzt. Beispielsweise sind FFBP-Netze und Kohonen-Modell unterschiedlich gut für Parallelisierung mit z.B. Transputernetzwerken geeignet.

Das Entwicklungswerkzeug sollte deshalb folgende Hardwarekriterien beachten:

- Eignung bezüglich des Netzwerkmodells
- Ausführgeschwindigkeit (einmaliges oder permanentes Lernen?)
- Verfügbarkeit/Standardhardware und Hardwarekosten

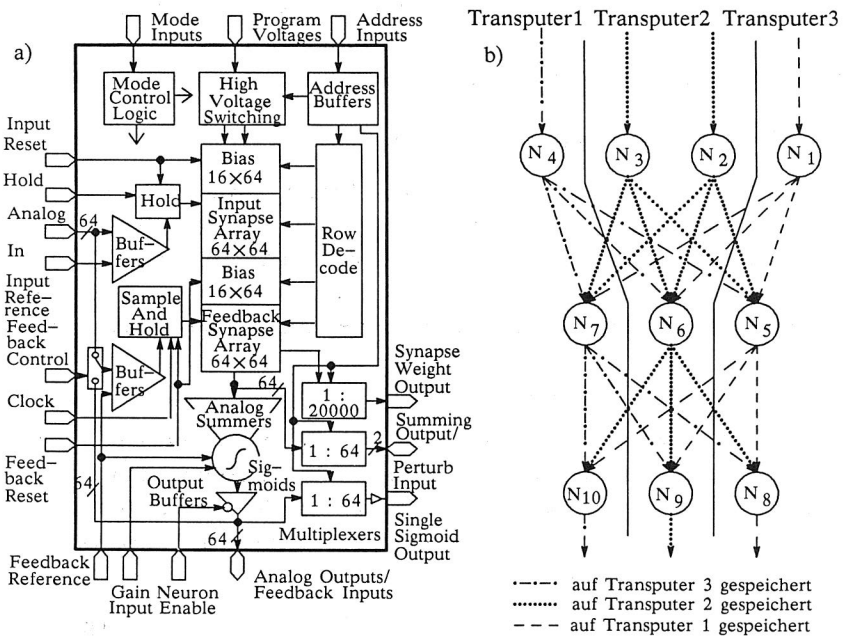


Abb. 6.10: Mögliche Zielhardware: a) Blockschaltbild des analogen 80170 NW ETAAN (EEPROM), b) Aufteilung auf Transputer; (nach /55/, /86/; vgl. /10/)

Im Rahmen des entwickelten Prototypen wird ausschließlich nicht permanentes Lernen behandelt. Lernvorgänge während der Betriebsphase werden nicht betrachtet. Durch Downloading wird das trainierte und betriebsfertige Netz in die Zielhardware überspielt. Standard-Bausteine wie Transputer oder sonstige digitale Signalprozessoren können dabei flexibler eingesetzt werden als z.B. Spezialhardware wie der kaskadierbare 80170 NW **ETANN** (electrical trainable, analog neuronal network-IC; jeweils 64 Neuronen, 10240 Synapsen, /55/ und /53/). Vergleiche Abb. 6.10.

6.3 Implizite SPS-Programmierung zur Musterverarbeitung

Einen wesentlichen Anteil an der (impliziten) Fertigungszellenprogrammierung hat neben der Bewegungsführung die Logik. Beispielsweise die konventionelle Programmierung von SPS durch assemblerartige Anweisungsliste (AWL). Das prototypische Werkzeug nach Kap. 6.2 wird im folgenden zur impliziten SPS-Programmierung ein-

gesetzt. Hierzu wird der in Abb. 6.11 dargestellte Versuchsaufbau für Lernprozesse an Fallbeispielen der Mustererkennung verwendet.

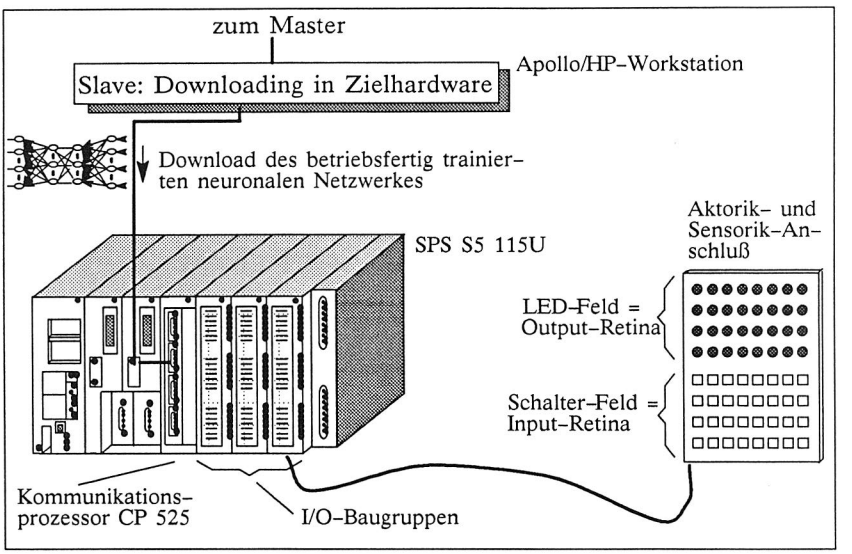


Abb. 6.11: Versuchsaufbau mit Kopplung zur SPS

Der Slave "Automatische Netzdimensionierung" wurde auf die Behandlung von FFBP mit drei Neuronen-Schichten beschränkt. Der Algorithmus nach Abb. 6.9 zeigte gute Ergebnisse bezüglich Konvergenzrate, Lerngeschwindigkeit und Robustheit gegenüber Parameterverstellungen, die teilweise deutlich über denen des herkömmlichen BP-Verfahrens lagen.

Als Anwendungsbeispiele wurde eine Klasse von häufig in der Automatisierungstechnik auftretenden Mustererkennungsaufgaben gewählt. Das neuronale Netz soll Teile identifizieren und ihre Lage erkennen (Verkippung, Verdrehung, Seitenlage). Ein Beispiel aus der Montage von Wälzlagerkäfigen zeigt Abb. 6.12.

Hierzu wurden für Ein- und Ausgang rechteckige Sensorfelder (= verallgemeinerte Input-Retina) und Aktorfelder (= verallgemeinerte Output-Retina) angelegt. Sie repräsentieren den Anschluß an die Umgebung (z.B. Lichttaster, Lichtschranken, Bildverarbeitungssystem). Input- und Outputmuster sind jeweils getrennt in einer 2D-

Retina angeordnet, wobei jedes Teilfeld der Retinas mit einem Input- oder Output-Neuron verbunden ist.

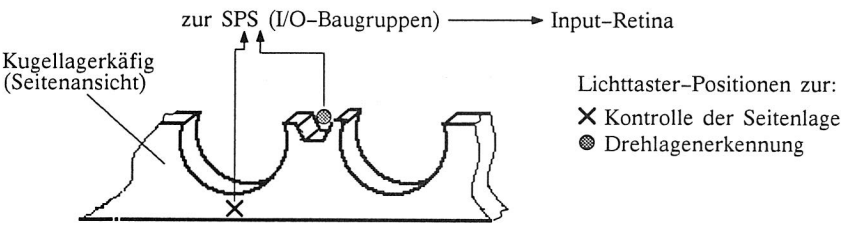


Abb. 6.12: Beispiel aus der Montage von Kunststoff-Kugellager-Käfigen

Die Abmessungen der rechteckigen Sensor- und Aktorfelder und deren Unterteilung in Teilfelder können vom Benutzer beliebig vorgenommen werden, bzw. sind durch die Anwendung festgelegt (z.B. direkt 1:1 durch den Anschluß einer mechanischen Taststift-Matrix). Durch Definition der Zeilen- und Spaltenzahl dieser Retina ist gleichzeitig die Ausgangs- und Eingangsschicht des neuronalen Netzes definiert (Abb. 6.13). Input- und Outputfelder können mit beliebigem Text gekennzeichnet werden, um z.B. die Bedeutung von speziellen Verbindungen zu Eingangssensoren auszudrücken. Durch die Erweiterung um einen Slave "Vor-/Nach-Verarbeitung" läßt sich eine noch stärkere Problemanpassung erzielen.

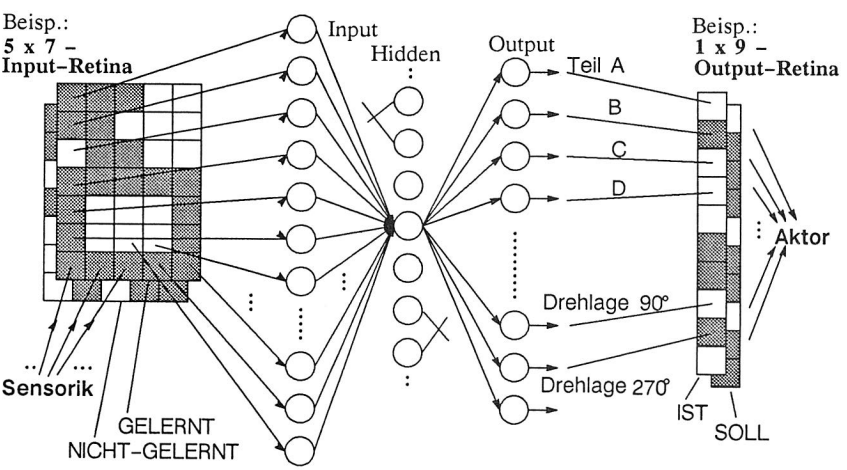


Abb. 6.13: Anschluß von Sensoren und Aktoren mittels verallgemeinerten Retinas

Die Strukturierung der Werkzeugoberfläche (Abb. 6.14 und 6.16) entspricht dem Master-Slave-Aufbau nach Kap. 6.2. Im linken Teil befinden sich die Kontrollen der einzelnen Slaves, im rechten Teil die Input-Muster (oben) und Output-Muster (unten).

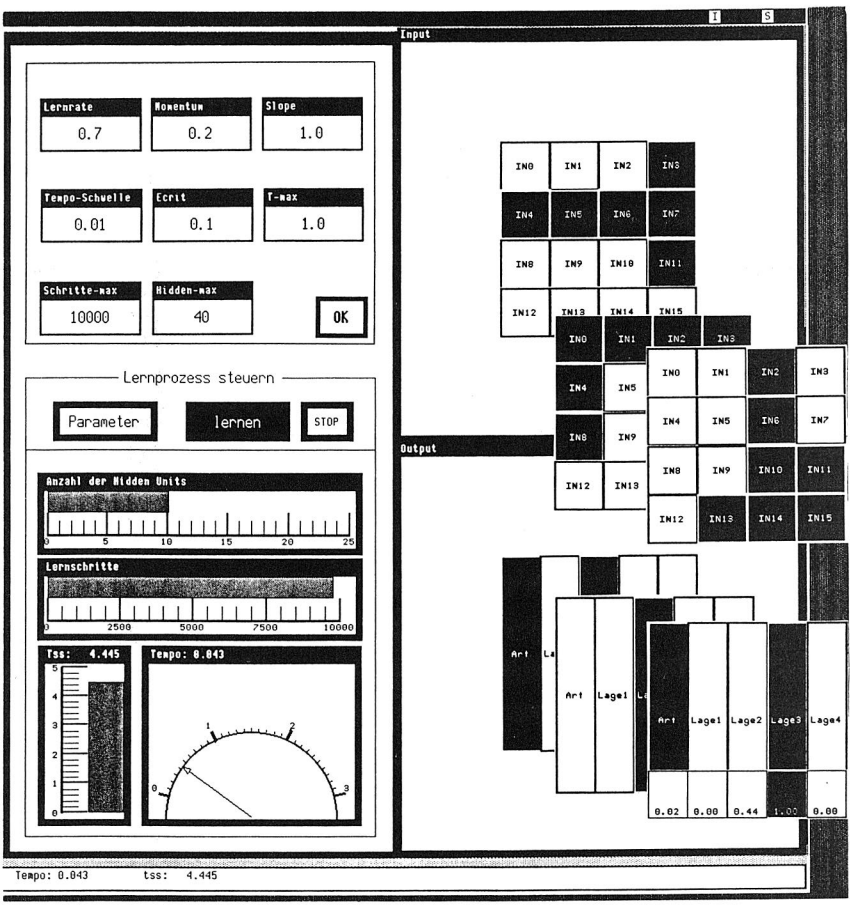


Abb. 6.14: Beispiel zur Erkennung von Art und Lage T-C-förmiger Teile, unabhängig von Translation, sowie möglicher Verdrehung der Muster um 0°, 90°, 180°, 270°

Geladene Muster können durch Mouseclick editiert werden. In den Beispielaufgaben in Abb. 6.14 und Abb. 6.16 wurden binäre Muster (schwarz -> "1" -> aktives Neu-

ron, weiß -> "0" -> inaktives Neuron) betrachtet (-> verringerte Trainingszeiten). Alle weiteren Slaves können jedoch auch kontinuierliche Werte (Grauwerte) aus dem Intervall [0,1] verarbeiten.

Bild 6.14 zeigt die Bedienung des Masters am Beispiel der Lernprozeß-Steuerung. Trainingsläufe können unterbrochen und z.B. mit geänderter Fehlerschwelle erneut aufgenommen werden. Der Tachometer zeigt die "Lerngeschwindigkeit" als Maß des Gradientenabstieges. Anstatt der schwer interpretierbaren, numerischen Auflistung der synaptischen Gewichte sollten Netztopologie und Gewichte graphisch angezeigt werden (vgl. /4/, /5/, /13/). Mit Hilfe der Netzgraphik läßt sich der Einfluß einzelner Neuronen auf das Gesamtnetzwerk untersuchen (Abb. 6.15).

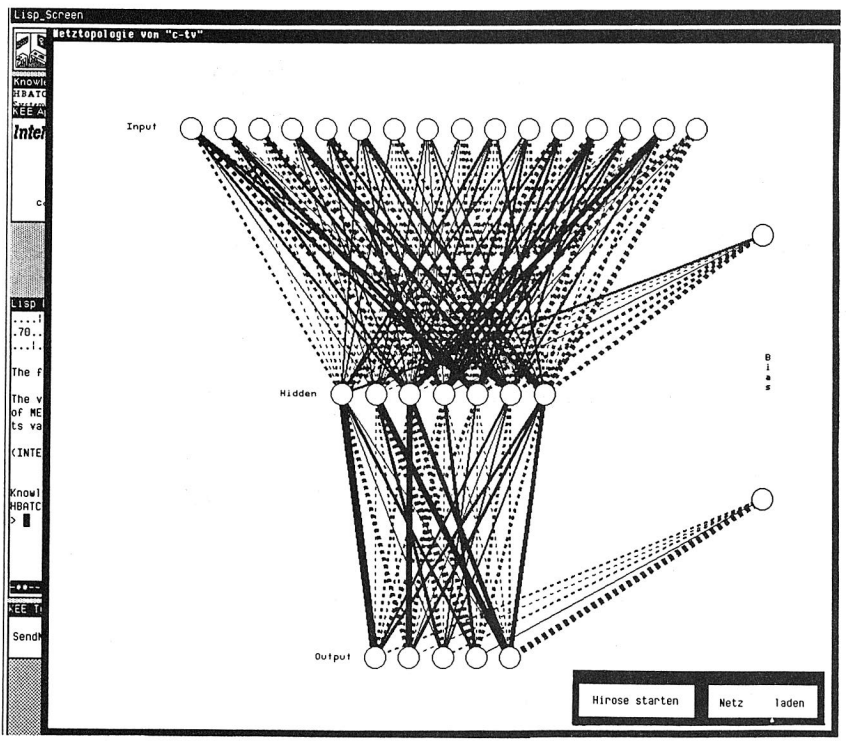


Abb. 6.15: Anzeige von Netzwerktopologie und der synaptischen Gewichte, mit:
---: $w_{ij} < 0$, hemmend; ——: $w_{ij} > 0$, erregend; Liniendicke :~$|w_{ij}|$, Verbindungsstärke

Das Downloading in die Zielhardware zeigt Abb. 6.16 am Beispiel der Zeichenerkennung.

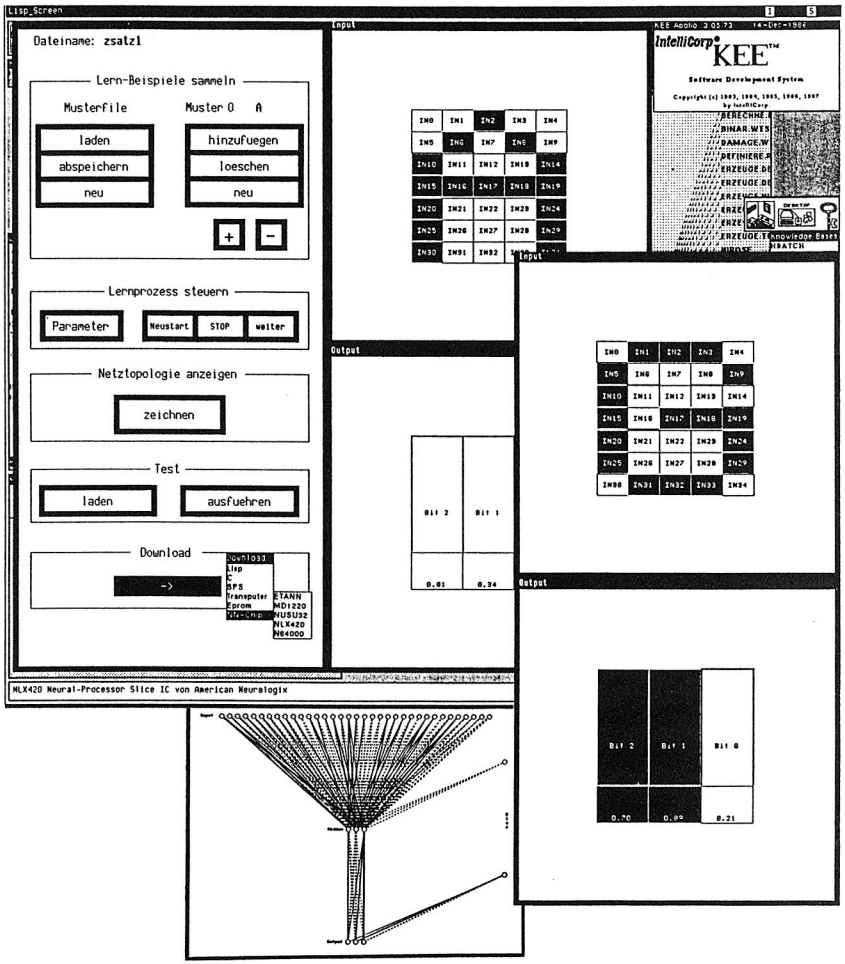


Abb. 6.16 : Hardwareauswahl und Download

Der Slave 'Hardware-Download' nach Kap. 6.2 generiert implizit für ein betriebsfertig trainiertes Netzwerk SPS-**Datenbausteine**. Diese werden dann in eine **SPS S5 115U**

mit Kommunikationsprozessor **CP 525** (regelt die Datenübertragung von Workstation zu SPS mit RK 512 Protokoll) geladen und dort aktiviert. Das Übertragungsprotokoll wurde auf seiten der Workstation implementiert (vgl. Kap. 4.3.1). Die SPS-seitigen Kommunikationsprozeduren wurden in **STEP 5** realisiert. Es werden Netzwerkaufbau und die synaptischen Gewichte in die SPS übertragen. Das Konzept "Trainieren anstatt Programmieren" wurde damit an einer in der Praxis häufig eingesetzten Zielhardware (Standardhardware) realisiert.

Da das Entwicklungswerkzeug intern selbst Module enthält, die ein neuronales Netz in C oder LISP implementieren, können die Netze damit auch auf Hardware übertragen werden, die mit C oder LISP arbeitet.

Zum Test der Prozeßanbindung wurden die Ein-/Ausgabebaugruppen der SPS an ein 8 x 8 Rasterfeld (= verallgemeinerte Retinas, s.o.) aus Schaltern und LED-Anzeigen angeschlossen (Abb. 6.11). Der Durchlauf des oben beschriebenen Entwicklungszykluses (einschließlich Training und Downloading) benötigte am Beispiel der Teile in Abb 6.14 und mit Hilfe des hier entwickelten Werkzeuges weniger als 5 Minuten.

7. Steuerung von Montagefeinbewegungen

Für die Entwicklung einer Verfahrenskette zur impliziten Geräteprogrammierung bedarf es der Konzentration auf die Bereitstellung elementarster Basisaktionen. Sie sind Teilfunktionen von übergeordneten Fertigungsprozessen (deshalb Eignung zur Aktionsplanung mit "immer feinerer Expandierung"). Im Beispiel von Montagezellen sind dies z.B. Greifen, Grobbewegen und Feinbewegen. Komplexaktionen (z. B. Auflegen, Aufsetzen, Einlegen, Ineinanderschieben, ..) werden durch Kombination von Elementaraktionen realisiert (s. Kap. 3). Für die implizite Geräteprogrammierung geeignete, inhaltliche. Normungen sind derzeit nicht bekannt (vgl. /128/ und Kap. 5.2.5).

Letztendlich sind die meisten Operationen der Handhabungs- und Montagetechnik (und allg. in Fertigungszellen) aus wenigen Basisoperationen aufgebaut. Die große, explosionsartige Vielfalt an Operationen entsteht durch deren Kombinatorik. Die Feinbewegung nimmt hierbei eine Sonderstellung ein, da sie in der Praxis generell die größte Fehlerquelle in automatisierten Anlagen darstellt.

7.1 Feinbewegungs- und Fügeplaner

Wie bereits behandelt, ist in der Roboteraktionsplanung eine Differenzierung zwischen Grobbewegung (einwirkende Kontaktkräfte/-momente auf den aktiven Fügepartner in der Regel ≈ 0) und Feinbewegung (Bewegungsanteile mit Kräfte/Momente $\neq 0$) sinnvoll. Ein wesentliches Hindernis, z.B. implizit generierte Roboterprogramme in der

realen Zelle auszuführen, sind unvermeidliche Abweichungen zwischen Realität und rechnerinternem Geometriemodell, die sich vor allem bei Fein- und Fügebewegungen auswirken. Dieses Problem wurde bislang, vor allem in Verbindung mit Offline-Systemen in CAD/CAM-Verfahrensketten, noch nicht praxisgerecht gelöst. Hierin liegt ein Hauptgrund für den noch unzureichenden Einsatz von Industrierobotern.

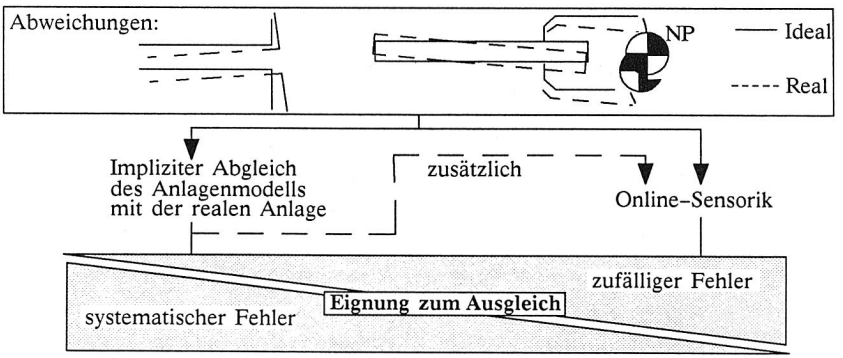


Abb. 7.1: Erhöhung der Präzision bei Feinbewegungen (Abweichungen überhöht gezeichnet)

Der Gesamtfehler besteht aus einem zufälligen und einem systematischen Anteil. Zum Ausgleich bestehen zwei grundsätzliche Möglichkeiten (vgl. Abb. 7.1):

A) Online-Sensorik (z.B. Kap. 7.2)
B) Implizite Mechanismen zur Adaption des Anlagenmodells (behandelt in Kap. 4.2)

Der implizite Abgleich des Anlagenmodells, mittels eines implizit erzeugten Einrichtprogramms, eignet sich vor allem für die Beseitigung systematischer Fehleranteile; die Online-Sensorik zum Ausgleich zufälliger Fehleranteile (Toleranzen). Werden anschließend, auf der Basis des adaptierten Umweltmodells, implizite Aktionen expandiert, so entsteht dabei angepaßter Steuercode, da der tatsächliche Anlagenzustand mit dem rechnerinternen Anlagenmodell wesentlich besser übereinstimmt. Verbleibende, zufällige Fehleranteile können online durch implizite Sensoraktionen abgefangen werden. Durch den impliziten Abgleich des Umweltmodells läßt sich der online-Sensoreinsatz wesentlich reduzieren oder event. vermeiden. Feinjustierungen werden durch direkte Prozeßkopplung (Vorteil von WOP) möglich (sog. "aktive" Prozesse, vgl. /93/).

7.1.1 Anforderungen an die Fügeplanung

Obige Verfahren zum Fehlerausgleich haben wesentlichen Einfluß auf die Konzeption des Fügeplaners. Er sollte robust sein gegenüber Geometrieabweichungen. Im weiteren sind nur geringfügige Fehler in Lage und Orientierung zugelassen (nach Kap. 4.2.1). Grobe Abweichungen, z.B. um 180° gedrehte Lagen, würden einen erneuten Erkennungsvorgang erforderlich machen und sind nicht Gegenstand eines Fügeplaners.

In /25/ werden z.B. die Positionswerte eines manuell erstellten Quellroboterprogrammes, das selbst keine Sensorfunktionen enthält, modifiziert. Dies erfolgt über ein ebenfalls manuell erstelltes, sensorgestütztes Hilfsprogramm und Mitprotokollieren der Kräfte und Momente während des Fügevorgangs. Es ist kein Umweltmodell vorhanden.

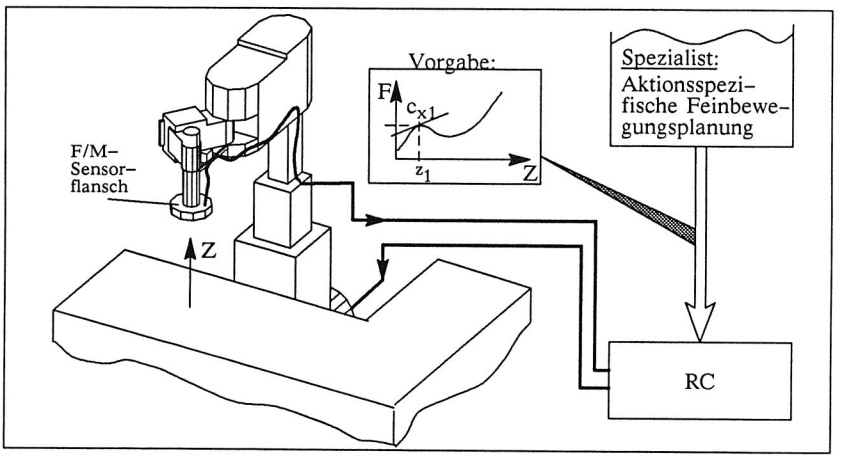


Abb. 7.2: Fügeaufgaben-spezifische Nachgiebigkeitsvorgaben durch den Fügeplaner

Ziel des Fügeplaners hier ist die Generierung prozeßflexibler, universell einsetzbarer Fügebewegungen. Bei vorhandener Sensorik, müssen Montagebefehle mit Feinbewegungsanteilen mit Hilfe von technologisch orientiertem Erfahrungswissen (Fügeweg, Greifkraft, Fügekraft, Fügemoment, Toleranzen,...) auf Roboterebene aufgelöst werden. Mit einem taktilen Kraft-Momenten-Sensor am Roboterflansch lassen sich z.B. Reaktionskräfte u. -Momente während der Kontaktphase verarbeiten (/12/, /72/, /107/).

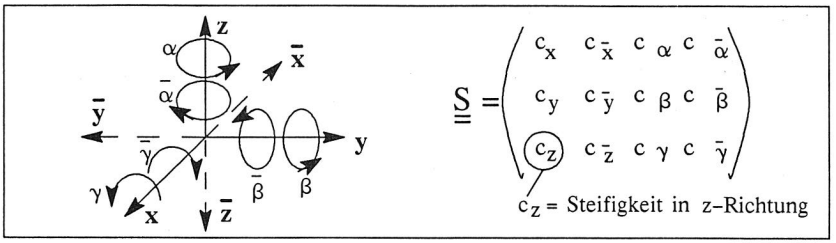
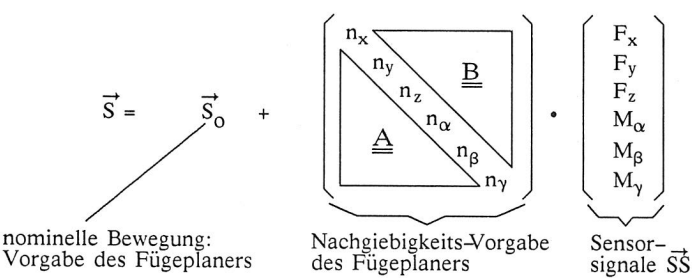


Abb. 7.3: Steifigkeitsmatrix für Fügevorgänge im lokalen Arbeitskoordinatensystem

Füge- und Demontageplaner besitzen die gleiche Problemstellung. Sie können zu einem Spezialisten 'Fügeplaner' zusammengefaßt werden und bestimmen primär Anfahr- und Abrückbewegungen. Mit ihnen lassen sich über die Fügepfadgenerierung hinaus, abhängig von der Fügesituation, passende Fügesteifigkeiten (3 rotatorische, 3 translatorische) für jeden Feinbewegungsabschnitt berechnen (/16/, /68/). Im allgemeinen Fall ergibt sich ein Verlauf nach Abb. 7.2, der aus der Analyse der Fügesituation heraus (CAD-Modell, Werkstoff der Fügepartner) für ein z.B. mit F/M-Sensorik ausgestattetes Bewegungsgerät vorgegeben werden kann (im allg. Steifigkeitsmatrix S nach Abb. 7.3). Industrielle Applikationen nutzen meist indirekt die mechanischen Steifigkeiten des Greiferführungsgetriebes (vgl. /32/) oder komplienter Werkzeugaufhängungen (s. Systematik zur Auslegung in /111/).

Die aus dem Anlagenmodell heraus bestimmten zeit- und wegabhängigen Steifigkeitsvorgaben (abhängig von Greifframe, Auskragung des Greifers und aktiven Fügepartners etc.) können entsprechend x=nF und n=1/c in die Fügepfadvorgabe (Interpolation) eingearbeitet werden; z.B. in der Form:



Vektor SS muß in Echtzeit (z.B. mit SDA in SRCL, vgl. Kap. 3.1.3) eingelesen und verrechnet werden. Ansätze, wie z.B. in /105/, lösen SS in Ströme der

Antriebsmotoren auf. Die Teilmatrizen A und B beschreiben Kopplungseffekte (Deviationen). Durch Bestimmung von Elementen aus A und B zu $\neq 0$ lassen sich sehr komplexe Fügeaufgaben lösen. Betrachtet man Fügevorgänge, die die menschliche Hand ausführt, so wird die große Bedeutung dieser Deviationsglieder ersichtlich. Im Falle n -> 0 bestimmt die "eingebaute" (natürliche) Steifigkeit der Kette IR-Mechanik, -greifer, Greifer-Griff, aktive und passive Fügepartner den Prozeß.

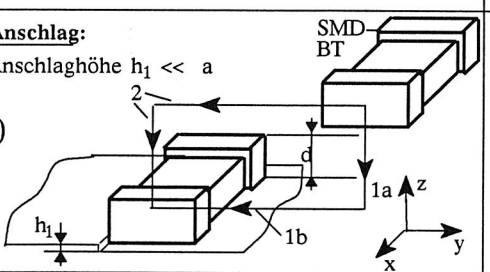
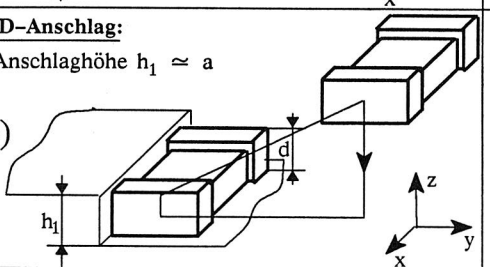
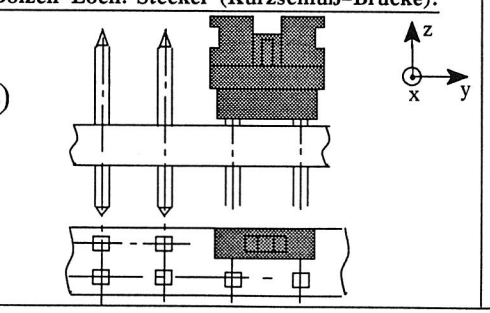
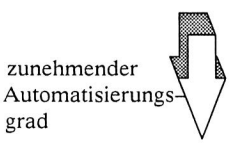
Fügesituations-Beispiele	Erwünschtes Verhalten nahe Fügeendposition
Anschlag: Anschlaghöhe $h_1 << a$ a)  	$\|c_x\|$ $\|c_{\bar{x}}\|$ groß (steif) $\|c_y\|$ klein (weich) $\|c_z\|$ klein (weich) $\|c_\alpha\|$ $\|c_{\bar{\alpha}}\|$ klein (weich) $\|c_\beta\|$ $\|c_{\bar{\beta}}\|$ klein (weich) $\|c_\gamma\|$ $\|c_{\bar{\gamma}}\|$ groß (steif)
2D-Anschlag: Anschlaghöhe $h_1 \simeq a$ b)  	wie a), jedoch mit: $\|c_\gamma\|$ $\|c_{\bar{\gamma}}\|$ klein (weich)
Bolzen-Loch: Stecker (Kurzschluß-Brücke): c)  	$\|c_x\|$ $\|c_{\bar{x}}\|$ klein (weich) $\|c_y\|$ klein (weich) $\|c_z\|$ groß (steif) $\|c_\alpha\|$ $\|c_{\bar{\alpha}}\|$ klein (weich) $\|c_\beta\|$ $\|c_{\bar{\beta}}\|$ klein (weich) $\|c_\gamma\|$ $\|c_{\bar{\gamma}}\|$ klein (weich)

Abb. 7.4: Einfluß der Fügesituation auf das vorzugebende Steifigkeitsverhalten

Abb. 7.4 zeigt qualitative Nachgiebigkeitsvorgaben in Abhängigkeit von Fügesituationen und Formelement–Paarung. In Beispiel a) muß die α–Richtung weich eingestellt werden, um ein 'Selbstausrichten' des SMD–Bauteils zuzulassen; die γ–Richtung dagegen steif, da die Absatzhöhe h_1 für einen 'Ausrichteffekt' zu gering im Verhältnis zu Höhe d ist (s. Gegenbeisp. b)). Die Steifigkeitswerte kennzeichnen jedoch nur die Werte in unmittelbarer Nähe des Fügeendframes. Zwischendurch sind, je nach Kontaktlage, andere c–Werte vorzusehen. Die Steifigkeitswerte gelten unabhängig von der Art ihrer Aufbringung, z.B. durch aktive Regelkreise oder die Art des Griffes (weicher oder harter Griff). Voraussetzung ist, wie generell in dieser Arbeit, eine quasistatische Betrachtungsweise. Über die Starrkörpermodellierung hinaus werden z.B. in /82/ Bewegungsgleichungen kraftschlüssiger Fügebewegungen ermittelt, einschließlich dynamischer Wechselwirkungen zum HHS, sowie Reibung, Bauteilelastizitäten und Stoßeffekten.

Abhängig von der Kontaktsituation (s. Kap. 7.1.2) müssen die Fügetrajektorien bestimmt werden. Z.B. ist Fügepfad 2 in Abb. 7.4a aufgrund von Toleranzen und Ungenauigkeiten bei der geringen Anschlaghöhe h_1 nicht sinnvoll. Der Fügeplaner muß deshalb Heuristik und Geometriealgorithmen kombinieren. Die Strategien zur Aufspaltung in Subtasks müssen in der Methodenbank für Fügewissen nach Kap. 5.2 abgelegt werden. Ebenso die Behandlung von Ausnahmesituationen wie die Überschreitung (max. Kräfte und Momente, max. Abdrängung von der Sollposition) oder Unterschreitung von Grenzwerten (z.B. Kontaktverlust, Anpreßkraft).

Der Feinbewegungsplaner kann auf unterschiedlich komplexen Verfahren basieren:



- Editieren der Feinbewegung direkt durch den Benutzer (Trajektorieneditor)
- Berechnung aus dem CAD–Modell des passiven, aktiven Fügepartners und umgebenden Körpern (s. Kap. 7.1.2)
- Lernen von Feinmotorik (s. Kap. 7.2)

7.1.2 Algorithmierte Fügeplanung durch virtuelle Demontage

In diesem Kapitel wird der in Abb. 2.6 eingeführte Spezialist Fügeplaner, basierend auf (Geometrie–)Algorithmen, behandelt. Ansätze in der Literatur formulieren z.B. für jede Kontaktstelle der Fügepartner eine Kontaktgleichung. Das Lösen dieser Gleichungssysteme unter geometrischen Zwangsbedingungen für praxisrelevante Bauteile ist jedoch nicht praktikabel. Darüberhinaus werden Feinbewegungen mittels Festlegung eines Weges durch einen Kontaktgraphen (/52/) definiert. Die Knoten

dieses Graphen kennzeichnen Kontaktzustände (z.B. Fläche/Fläche, Kante/Fläche, Punkt/Fläche) und zugehörige Freiheitsgrade. Die Kanten können Bedingungen, die während Kontaktübergängen (Wechsel des Kontaktzustandes) einzuhalten sind bzw. überprüft werden müssen, repräsentieren. In /23/ wird anhand von Sensordaten (F/M) zunächst eine statische Kontaktanalyse durchgeführt. Mißlingt die Kontaktart-Bestimmung anhand von Gleichgewichtsbedingungen, folgt in einem zweiten Schritt eine aktive Kontaktanalyse (d.h. mit aktiver IR-Bewegung). Sie ist Voraussetzung für einen sog. Replaner, der Fehler durch Zurückführen in eine definierte Berührkonfiguration korrigieren soll. Aufgrund der Zeitanforderungen wurden obige Verfahren (z.B. Navigation durch komplexe Kontaktgraphen realer Bauteilgeometrien) nicht weiter betrachtet.

Im folgenden wird ein in dieser Arbeit entwickeltes, vereinfachtes Verfahren beschrieben, das sich für Online-Berechnungen (WOP) eignet. Es basiert auf der Einführung von **Sperrvektoren** und sog. **Trennpyramiden**. Hiermit kann der Feinbewegungspfad und die Anfahrrichtungen bestimmt werden. Ausdehnung von Greifer, gegriffenem Teil und Sicherheitsabstände müssen eingerechnet werden.

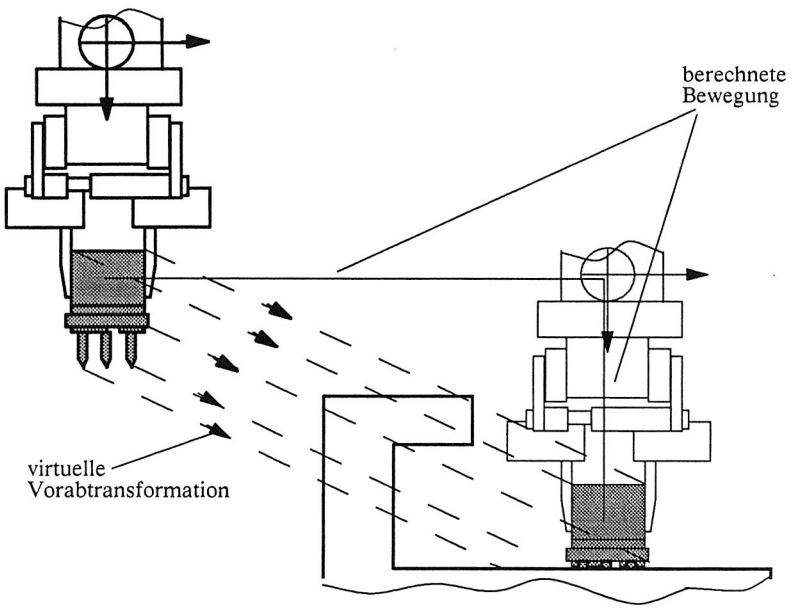


Abb. 7.5: Virtuelle, temporäre Vorabtransformation

Virtuelle Projektion des aktiven Fügepartners:
Sperrvektoren werden durch Analyse der Kontaktsituation gewonnen. Für die zu berechnende Feinbewegung des aktiven Fügeteils in unmittelbarer Umgebung der Fügeendposition sind die Kontaktzustände von Interesse, wie sie sich ergeben, wenn sich das Objekt in Endposition befindet. Im Ausgangszustand des Bewegungsablaufes ist dies jedoch nicht der Fall. IR und aktiver Fügepartner befinden sich noch im Nahpunktframe - an der Grenze zum Feinbewegungsbereich.

Der aktive Fügepartner wird deshalb zunächst, in einem vorgelagerten Schritt, virtuell in seine Zielposition überführt. Diese temporäre Projektion wird im internen Anlagenmodell real ausgeführt. Bei diesen Transformationen werden keine Kollisionen beachtet (Abb. 7.5). In der so erreichten, virtuellen Fügeendsituation und der sich dabei ergebenden Kontaktsituation kann nun die eigentliche Sperrvektorenrechnung (s.u.) durchgeführt werden. Nachdem der aktuelle Abschnitt der Feinbewegungstrajektorien berechnet wurde (+ event. Sicherheitstests), wird die Vorwegnahme der Zielsituation im Anlagenmodell zurückgenommen. Das Umweltmodell wird wieder in den Ausgangszustand versetzt. Diese virtuellen Transformationen werden nicht visualisiert (keine Grafik-Aktivierung) und sind somit für den Bediener nicht sichtbar. Erst die gesichert berechneten Feinbewegungstrajektorien werden dann endgültig im Anlagenmodell (+Bildschirm) und event. in der realen Anlage ausgeführt. Das gesamte Verfahren entspricht einer **virtuellen Demontage** (Gemeinsamkeiten mit impliziten Befehl "Demontiere!"und "Abrücken!", s. Kap. 3.2).

Dieses Verfahren kann durch rekursive Anwendung Stützframes der Feinbewegungstrajektorie bis zum Nahframe bestimmen (Rückwärtsrechnung). Der zuletzt berechnete Stützframe wird dazu zum Zielframe für die darauffolgende, temporäre Projektion (zyklenfrei!). Die Nahframeberechnung selbst gehört zum Aufgabenbereich des Fügeplaners, da damit der Startframe der Feinbewegung festgelegt wird. Der Nahpunktframe ist identisch mit dem ersten Stützpunkt der Feinbewegungstrajektorie. Hier wird der große Vorteil einer, auf einem ständig aktualisierten, mitlaufenden Anlagenmodell basierenden Aktionsplanung deutlich. Virtuelle Transformationen sind nur dadurch möglich. Der Fügeplaner benötigt intensiven Zugriff auf das Umweltmodell (Geometrie, Greifsituation des aktuell gegriffenen Bauteils, etc.).

Der Zielzustand wird im Falle der Komplettmontage (Kap. 3.4) aus der CAD-Analyse des 3D-Implosionsmodells bestimmt. Im interaktiven Betrieb in der Werkstatt ist er durch implizite Montagebefehle vorgegeben, bzw. wird durch konstruktiv und körperfest angeheftete Fügeendframes oder Frames-Scharen (z.B. Tischplatte) definiert.

Sind relativ an die Fügepartner angeheftete, manuell editierte Fügepfade verfügbar, so besitzen diese gegenüber der automatischen Berechnung höhere Priorität.

Berechnung der Sperrungen:
Die Schlußfolgerung von Kontaktarten aus dem Umweltmodell heraus ist im allgemeinen Fall aufwendig. Einfacher ist z.B. die Detektion von Flächenkontakt (über Normalformen) oder Spezialfälle geometrischer Grundkörper (s. Kap. 3.1.1). In der Praxis müssen noch Toleranzen für Translation und Rotation z.B. der Kontaktstellennormalenvektoren (Hauptebenen der Berührstellen) vorgegeben werden, für die eine Berührung noch als definiert gilt.

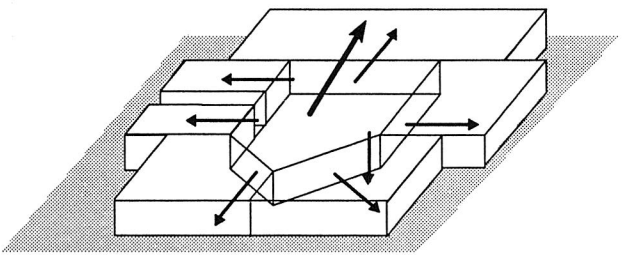


Abb. 7.6: Aufstellen von Sperrvektoren

Das Ergebnis der Kontaktanalyse (im Nahbereich des aktiven Fügepartners) ist eine Menge von Berührstellen. Jede Berührstelle ist durch eine Sperrung von Bewegungs-Freiheitsgraden gekennzeichnet, die sich durch eine Schar von Sperrvektoren repräsentieren läßt. Im Spezialfall des Flächenkontaktes ist dies der Flächennormalenvektor (s. Abb. 7.6). Eine einfache Vektoraddition und Komplementbildung von Sperrvektoren (mit Ausfiltern paralleler Vektoren, etc.) ergibt bereits eine grobe Bewegungsmöglichkeit in der Umgebung der virtuellen Zielposition (=Arbeitspunkt).

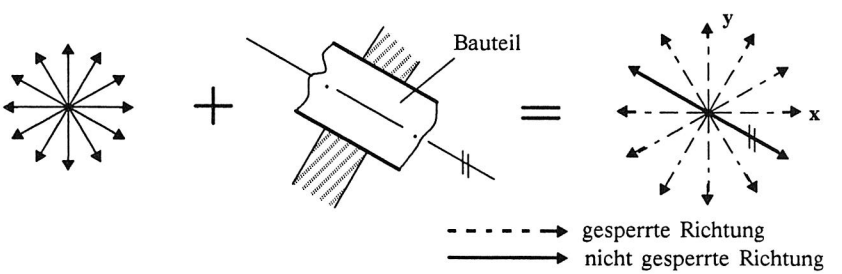


Abb. 7.7: Rasterung in 2D (3D analog)

Zu jedem aktiven Fügepartner wird systemintern ein **Feinbewegungsraum** mitgeführt, der stets den aktuellen Zustand von freien und gesperrten Richtungen beschreibt. Dieser Feinbewegungsraum wird durch sukzessive Abarbeitung der gefundenen Berührstellen so modifiziert, daß die Freiheitsgrade auch sukzessive eingeschränkt werden. Für den Feinbewegungsraum sind mehrere Repräsentationen möglich. Eine dreidimensionale Rasterung, wie z.B. nach Abb. 7.7, ist nicht geeignet, da auch bei sehr feiner Rasterung potentielle Freiheitsgrade event. nicht erkannt werden.

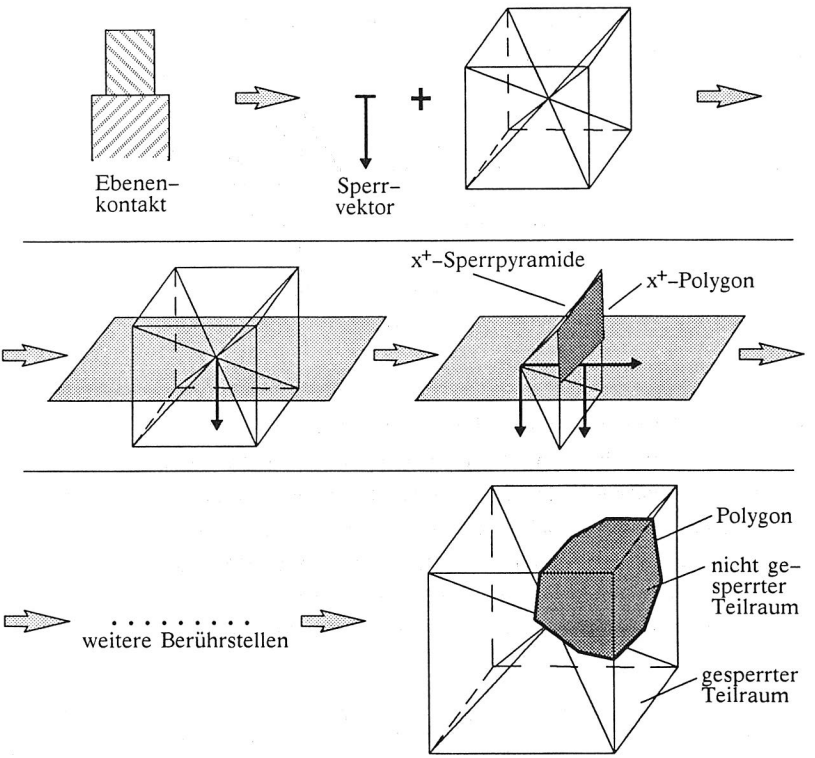


Abb. 7.8: Feinbewegungsraum

Für die rechnerinterne Fortschreibung des Feinbewegungsraumes sind z.B. sechs zu einem Feinbewegungsraum zusammengesetzte Sperr- bzw. Trennpyramiden geeignet. Für jede dieser Pyramiden wird dynamisch während der Aktionsplanung berechnet und gespeichert, welche Volumensegmente daraus (noch) nicht gesperrt sind. Um den

Istzustand einer Pyramide festzuhalten reicht es in einem ersten Ansatz aus, die durch Sperrvektoren entstandenen Polygone in den Basisflächen der Pyramiden zu speichern (Abb. 7.8). Die Hinzunahme neuer Sperrungen darf alte Restriktionen, die durch vorher verarbeitete Sperrungen aufgestellt wurden, nicht aufheben (Fügen = schrittweises Verringern von Freiheitsgraden des aktiven Fügepartners).

Zum Startzeitpunkt sind die Polygone Quadrate. Es besteht keine Einschränkung der Beweglichkeit (analog für Rotation). Im allgemeinen Fall entstehen für jedes aktive Fügeteil konvexe, mathematisch zusammenhängende Flächensegmente (vgl. Blop der BV). Bei Mehrdeutigkeiten muß z.B. nach Bewegungsnebenbedingungen (Kontaktbedingungen) oder, aus der Kinematik des Handhabungsgerätes resultierende Vorzugsrichtungen (z.B. radial zur IR-Hochachse), entschieden werden.

7.2 Einsatz eines Neuronalen Netzes für das Erlernen von Feinmotorik eines 6-Achs-Industrieroboters

Eine Erweiterung der Fügeplanung nach Kap. 7.1 stellt ein 'umschaltender' Fügeplaner dar. Der Aktionsplaner kann hierzu abprüfen, ob ein, für die aktuelle Formelement-Paarung betriebsfertig trainiertes, neuronales Netzwerk vorliegt. Ist dies erfüllt, kann alternativ auf neuronale Bewegungssteuerung umgeschaltet werden (=Kombination analytischer mit neuronaler Methoden; Abb. 7.9).

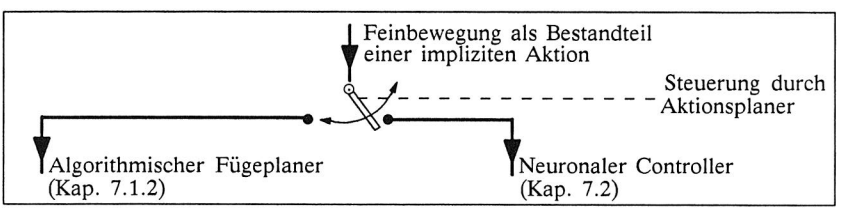


Abb. 7.9: Aktionsplaner-gesteuerte Umschaltung

Die Analyse von Feinbewegungen zeigt, daß eine Reihe von Standard-Fügesituationen gehäuft auftreten. Ziel ist das Lernen dieser Standard-Fügesituationen (meist identisch mit Klassen von Formelementpaarungen), wie z.B.:

- Führen gegen 2D/3D-Anschlag (Ein-, Auflegen: Tisch, Vorrichtung, Raumecke)
- Bolzen-Loch-Problem (und Abwandlungen)
- Feder-/Nut-Eintauchen
- Leichtgängigkeits-Prüfungen, Justagevorgänge, Eindrücken gegen Druckpunkt

Jede dieser wiederkehrenden Fügesituationen ist durch eine ihr eigene Feinmotorik bzw. Reaktionen auf Sensorsignale, d.h. sensor-motorisches-Verhalten, gekennzeichnet. Sie unterscheiden sich in den ihnen eigenen Charakteristika wie Bewegungsfolge, Nachgiebigkeiten zu bestimmten Abschnitten der Feinbewegungstrajektorie etc. Das gilt sowohl für den Menschen, wie auch z.B. für Montageroboter (vgl. Abb. 7.4).

Derzeit besitzen Industrieroboter keine Lernfähigkeit. Roboterprogramme, die z.B. Sensorkorrekturdaten in die Bewegungsbahn miteinrechnen, werden konventionell programmiert (z.B. 6-achsige Kraft-Momenten-Sensorik, mit Echtzeitberechnung der 3 Kräfte + 3 Momente zwischen Roboter und Greifwerkzeug). Die mathematisch nachvollziehbare Modellbildung ist dabei Voraussetzung für den Aufbau komplexer Regelkreise. Weiterhin sollten z.B. Greif- und Kraftangriffspunkte mit den Annahmen der Modellbildung (z.B. Aufstellen von Bewegungsgleichungen) übereinstimmen. Derzeitige Sensorfunktionen basieren auf der Abarbeitung von Programmen, die einen analytischen Zusammenhang z.B. zwischen Positionen, Geschwindigkeiten und Sensorwert beschreiben. Bei Änderung der Problemstellung bedingt dies eine Neuprogrammierung und damit Programmierkenntnisse des Benutzers. Konventionelle, analytische Methoden der Signalverarbeitung basieren in der Regel auf folgende Einzelschritte und Reihenfolge:

1. Analyse und Verstehen der Steuerungs- und Regelungsaufgabe
2. Modellierung des dynamischen Systems als Voraussetzung für den Entwurf der Steuerung, bzw. Regelung (z.B. Aufstellen der Bewegungsgleichungen für hybride Mehrkörpersysteme, wie z. B. Industrieroboter)
3. Reglerentwurf (z.B. Riccati-Zustandsregler), Entwicklung der Algorithmen
4. Implementierung in Steuerungscode

Die Stärke, von aus Beispielen lernenden Neuronalen Netzen, liegt in ihrer Fähigkeit Problemstellungen zu lösen, für die es schwierig ist obige Schritte zu durchlaufen, z.B. mathematische Modelle oder Gesetzmäßigkeiten zu erstellen (vgl. Kap. 6.1). Konventionelle Systeme sind sehr störanfällig gegenüber Toleranzen (zu fertigendes Produkt und Betriebsmittel). Darüberhinaus muß z.B. für automatisierte Montageprozesse häufig heuristisches Wissen in die Steuerung eingearbeitet werden. Ansätze der künstlichen Intelligenz, wie z.B. derzeitige Expertensysteme besitzen ihre Stärke in der Symbolverarbeitung und sind nicht in der Lage die Steuerungsaufgaben in Echtzeit zu lösen. Häufig führen Änderungen zu Redesign der Steuerungs- und Regelungssoftware.

Gerade hier zeigt sich die große Überlegenheit des Menschen gegenüber der automatisierten Steuerung stark nichtlinearer, dynamischer Systeme, wie es z.B. ein 6-Achs-Handhabungssystem für Montageaufgaben darstellt. Das feinmotorische Verhalten des Menschen bleibt nicht konstant und wird durch Übung permanent verbessert und trainiert. Der Mensch macht besonders im Kleinkindalter starke Lernfortschritte bezüglich der Bewegungssteuerung und bildet dadurch Grundfertigkeiten aus; ohne Kenntnis von Bewegungsgleichungen, Physik oder Regelungsstrukturen. Das Vorbild 'Natur' ist damit nicht durch ein analytisches, sondern durch ein lernendes Vorgehen gekennzeichnet.

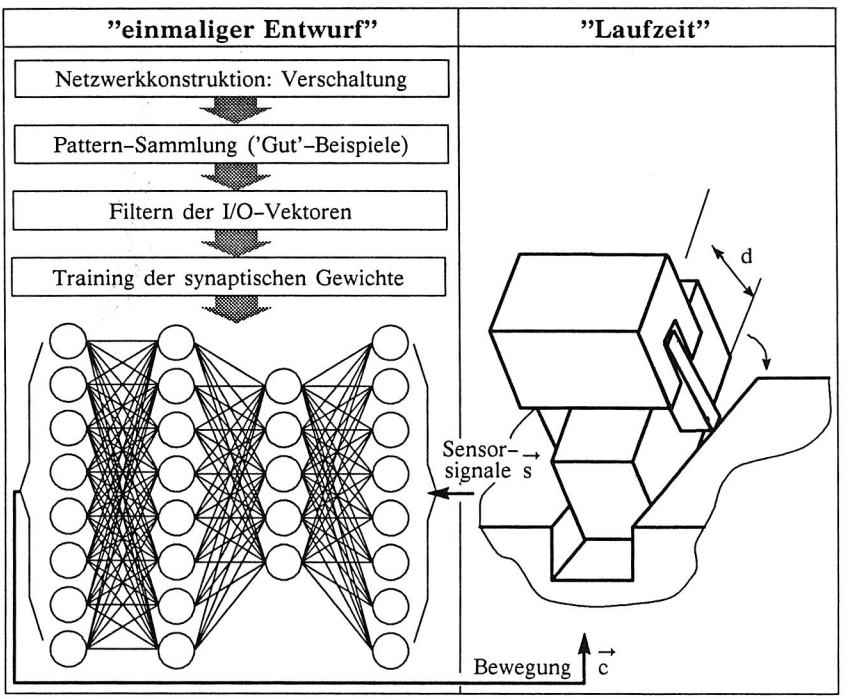


Abb. 7.10: Entwicklung eines neuronalen IR-Controllers für eine Standardfügesituation

Analog hierzu versucht der vorliegende Ansatz die IR-Feinmotorik nicht analytisch zu beschreiben, sondern durch Lernprozesse mittels einer "Black box" zu erfassen (s. /8/, /100/). Das Innere der "Black box" soll aus einem neuronalen Netzwerk bestehen.

Feinmotorik soll ohne Kenntnis physikalischer Systemmodelle (Robotermechanik, Antriebselemente, Sensorik, ..) maschinell eingelernt werden. Auf der Basis eines Feed-Forward, rekursiven neuronalen Netzwerkes (kurz: NN) sollen obige Standard-Feinbewegungsaktionen nach dem Back-propagation-Verfahren (s. Kap. 6.1) problemangepaßt trainiert werden. Permanentes Lernen während des Betriebes (s. /26/) wird in der vorliegenden Arbeit nicht behandelt. Abb. 7.10 zeigt die Entwicklungsschritte bis zur Übernahme der Robotersteuerung durch den neuronalen Controller.

Die sensorgeführte Bewegung wird nicht durch Zerlegung in Programmteilschritte und Abarbeitung dieser Anweisungssequenz realisiert, sondern das Neuronale Netz lernt sich "als Ganzes" auf einen Eingangsvektor zu verhalten (vergleichbar mit einem Reflexverhalten). Abbildung T wird nicht programmiert, sondern trainiert.

$$\vec{s} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \end{pmatrix} \quad \xrightarrow{\;T\;} \quad \vec{c}_I = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta A \\ \Delta B \\ \Delta C \end{pmatrix} \quad \text{bzw.} \quad \vec{c}_{II} = \begin{pmatrix} \Delta \alpha_1 \\ \Delta \alpha_2 \\ \Delta \alpha_3 \\ \Delta \alpha_4 \\ \Delta \alpha_5 \\ \Delta \alpha_6 \end{pmatrix}$$

Getestet wird das Funktionsprinzips des neuronalen Controllers mit Hilfe des Roboterzellensimulators des Aktionsplanungswerkzeuges (s. Abb. 2.5).

7.2.1 Lernprozeß mit optischen Abstandssensoren

BV-Sensoren sind für die Feinbewegung nur bedingt geeignet, da der Fügeort meist durch Greifer, passiven und aktiven Fügepartner, Vorrichtungen und Peripherie abgeschattet wird (vgl. Kap. 4.2 u. 3.1.4). Zudem werden für die Fügeplanung räumliche Geometriedaten benötigt, die nur unter großem Aufwand aufzunehmen sind (vgl. sog. Sensorfusion /1/, /98/). Die Domäne der Bildverarbeitung bleibt deshalb im Regelfall (Ausnahme: planare z.B. SMD-Bestückung) auf Szenenaufnahmen vor dem eigentlichen Fügevorgang beschränkt (z.B. impliziter CAD-Abgleich Kap. 4.2.2).

Taktile Sensoren, wie z.B. eine F/M-Dose an der Handwurzel, besitzen diesen Nachteil nicht, jedoch fallen nur bei direktem Körperkontakt Daten an. Probleme ergeben sich hier durch Oberflächenbeschädigungen infolge der Gleitvorgänge, etc. Besonders nachteilig sind Stick-Slip-Effekte, die gemessene Sensorwerte verrauschen. Optimale Feinmotorik sollte jedoch, analog zu manuellen, feinfühligen Bewegungen den

Körperkontakt auf das erforderliche Maß reduzieren. Das führt zum Einsatz nichttaktiler Sensorik.

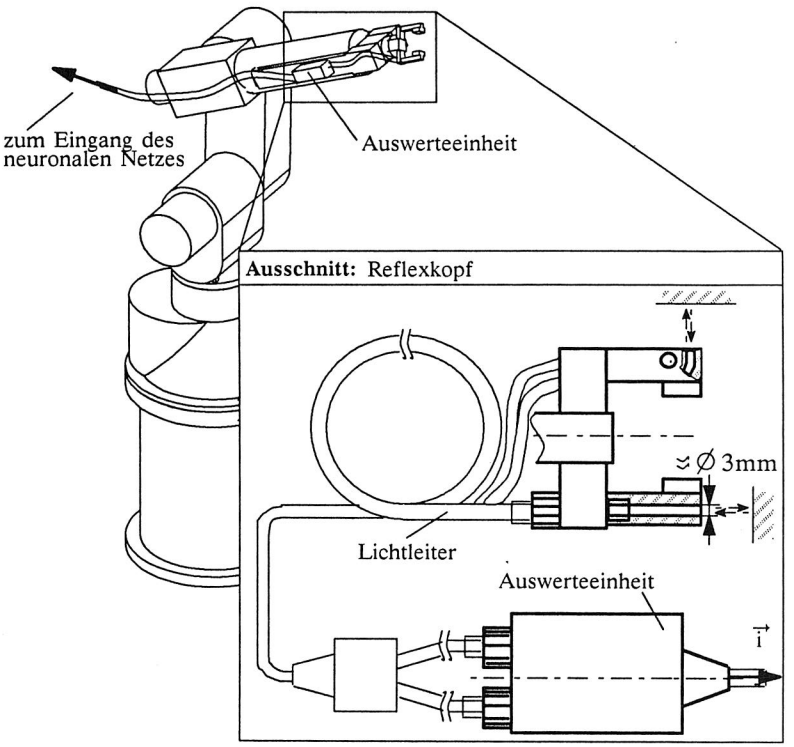


Abb. 7.11: Abstandssensor-Signale zur Gewinnung des Input-Vektors

Für die neuronale Feinbewegungssteuerung wurden zwei Arten nichttaktiler, mitbewegter Sensorik eingesetzt: Eindimensionaler Laser-Abstandssensor und Laserscanner. Die optischen Sensoren müssen im Simulator nachgebildet werden. Die Sensorsimulation erfolgt nach Kapitel 3.1.4 (Sensorsignal-Berechnung, Scanverfahren und Strahlreflexion). Abb. 7.11 zeigt die im Umweltmodell des prozeßnahen Werkzeuges simulierte Sensorik. Durch Beschränkung auf relevante Körper bzw. Flächen in der Umgebung (Markierung im CAD-Modell) lassen sich die Berechnungszeiten der Sensorsimulation wesentlich verkürzen.

Neue Möglichkeiten im Bereich der Feinbewegung versprechen Entwicklungen auf dem Gebiet der Präzisionsmeßtechnik; speziell neue Entwicklungen im Bereich der mittlerweilen stark miniaturisierten Glasfaseroptiksensoren bzw. Sensoren mit integrierter Optik. Glasfasersensoren benutzen das Glasfaserkabel direkt als Meßfühler und/oder zur Übertragung des Lichtes an die Auswerteeinheit /54/. Analoge Abstandssensoren auf Lichtleiterbasis sind mittlerweilen sehr kostengünstig verfügbar.

Sensoren mit integrierter Optik benutzen optische Komponenten, die direkt auf einem Halbleiterchip aufgebracht sind /120/. Im Gegensatz zu induktiven oder kapazitiven Näherungssensoren wird sehr genau und fokussiert gemessen. Die entscheidenden Vorteile liegen in den optischen Übertragungsmöglichkeiten. Zwischen Meßstelle und Auswerteeinheit sind große Entfernungen möglich. Die Auswerteeinheit kann z.B. am Roboterarm angebracht werden. Die Abmessungen am Wirkort sind gering und ermöglichen daher die Integration der Meßaufnehmer (z.B. Fa. Bernstein) in das IR-Werkzeug. Derartige Sensoren liefern auch ohne Körperkontakt Werte. Während der Grobbewegung liefern sie keine, oder nicht sinnvolle Signale (aufgrund des mitlaufenden Umweltmodells auch nicht erforderlich).

7.2.2 Neuronaler Bewegungs-Controller

Wie bereits behandelt sollte der neuronale Controller robust sein gegenüber Ungenauigkeiten der Toleranzkette (geometrische u. kinematische Fehler, Fehler der Sensoranbringung, Rechenungenauigkeiten). Es entsteht eine Füge-Ausgangssituation im Nahframe nach Abb. 7.1. In der Praxis besitzen die Abweichungen der passiven Fügepartner meist den größten Anteil an dem relativen Gesamtfehler.

Am Beispiel des Führens eines Bauteils gegen einen 3D-Anschlag, bzw. Einlegen in eine Raumecke (z.B. bündig) soll die Feinbewegung gelernt und die Steuerungseigenschaften eines NN-Controllers für Roboterfeinmotorik untersucht werden. Darüberhinaus zeigten Versuche, daß dies zu Training und Steuerung (Betrieb des Controllers) beliebiger Feinbewegungen verallgemeinert werden kann. Die 'Raumecke' kann dazu nur noch als Referenz-Umgebung (Referenzflächen für die Abstandssensorik) dienen. Beispiele möglicher Referenzflächen sind Bezugsflächen von Teilen des zu fertigenden Produktes selbst (z.B. Gehäuseteile) oder umgebende Anlagen- bzw. Vorrichtungsteile. Es wäre auch denkbar, sofern vom Produkt her möglich, eine eigene Referenz-Vorrichtung einzusetzen, die das Ziel hat geeignete Referenzflächen für die Strahlreflexion der Abstandssensorik bereitzustellen

und innerhalb der eine Feinbewegung trainiert und ausgeführt wird. Als Voraussetzung für die Referenzflächen gilt, daß sie statisch sind, d.h. daß ihre **relative** Lage und Position zueinander während und zwischen Training und Betrieb des NN-Controllers nicht verändert wird.

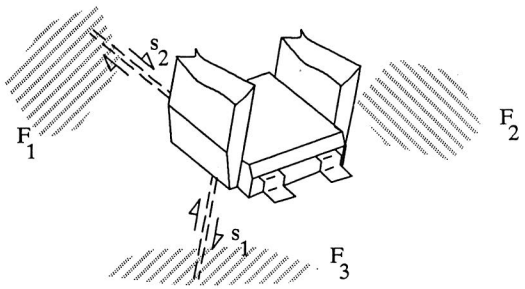


Abb. 7.12: Referenzflächen für den neuronalen Bewegungs-Controller

Position und Orientierung der Referenzflächen, Ort und Winkel der Strahlaustritte der Sensoren sind prinzipiell beliebig. Voraussetzung ist, daß die aus den (konventionellen) Verfahren zur Reglersynthese bekannten Beobachtbarkeits- und Steuerbarkeitsbedingungen erfüllt sind. Für die Anzahl und Anbringung der mitbewegten Sensoren bestehen Restriktionen derart, daß die Meßgrößen grundsätzlich genügend Information zur Rekonstruktion der relativen Position und Orientierung der beiden Fügepartner zueinander liefern.

Die im IR-Greifwerkzeug integrierten Abstandssensoren (eindimensional, analog, nicht-taktil; Abb. 7.11) messen die Abstände zu den umgebenden Flächen. Die gelieferten Sensorwerte sind ein Maß für die relativen Abstände zwischen IR-Werkzeug (bzw. indirekt auch des aktiven Fügepartners) zu den Referenzflächen während den Feintrajektorien (Abb. 7.12). Die Fügeendposition soll ebenso wie die Trajektorien durch Beispiele eingelernt werden. Der Stelleingriff des NN-Controllers kann prinzipiell an zwei Punkten erfolgen:

(I) Das Neuronale Netz erzeugt einen Vektor im Euler-Koordinatensystem
(II) Das Neuronale Netz erzeugt einen Vektor im IR-Gelenkkoordinatensystem

Stelleingriff II ist prinzipiell einfacher, da die Zwischenschaltung von Interpolation und Rückwärtstransformation entfällt (vgl. /100/). Da das Neuronale Netz ohnehin eine nichtanalytische Abbildung realisiert, und Training erforderlich wird, kann die Rückwärtstransformation ebenfalls mit in diese Abbildung integriert werden. Ex-

perimente zeigten jedoch keine befriedigenden Ergebnisse. Die im Rahmen dieser Arbeit noch behandelbare Neuronenzahl des Controllers ist zu gering für die Abbildung der starken Nichtlinearität in Abhängigkeit vom Arbeitspunkt im IR-Arbeitsraum (z.B. nahe singulärer Stellen). Die gelernte Feinmotorik ist stark arbeitspunktabhängig und nur innerhalb eingeschränkter Kontrollvolumen im IR-Arbeitsbereich gültig. Versuche zeigten, daß Stelleingriff I, im Vergleich zu II, mit einer geringeren Anzahl Neuronen auskommt. Er wurde aufgrund der derzeit noch sehr stark begrenzten, behandelbaren Neuronenzahl für die Ansätze in dieser Arbeit verwendet.

Für den neuronalen Bewegungs-Controller wurden eine Reihe unterschiedlicher Netze trainiert und untersucht. Variiert wurden:

- Der Kopplungsgrad, durch Trennung in Translations- und Rotations-Subnetze
- Sensoranzahl und -anordnung
- Sensortyp (eindimensionaler Abstandssensor und Laserscanner)
- Sensordaten-Vorverarbeitung (Fehlerfunktion)
- Zerklüftete Referenzumgebung und Referenzumgebung mit ebenen Flächen

In der Praxis ist das Beispiel der ebenen Referenzflächen meist unrealistisch (z.B. Montage von Bauteilen in ein feinwerktechnisches Gerät). Es erbringt aber Lösungswege und Konzepte, die sich auch auf komplexere Fügeprobleme übertragen lassen. Stark zerklüftete Umgebungen benötigen mehr Informationen, so daß Sensoren mit größerem Informationsgehalt der Sensorwerte günstiger sind (z.B. Laser-Scanner). Hierzu wurden Laser-Scanner nach Kapitel 3.1.4 simuliert. Die Ergebnisse konnten dadurch deutlich verbessert werden. Nachteilig für den praktischen Einsatz sind die derzeit noch höheren Kosten gegenüber den kostengünstigen 1D-Abstandssensoren, sowie der größere Bauraum.

Stellvertretend zeigt Abb. 7.13 ein neuronales FFBP-Netzwerk mit insgesamt 30 Neuronen, das zur Verarbeitung von Sensorsignalen (Eingang) und Berechnung von IR-Feinbewegungen (Ausgang) eingesetzt wurde. Es handelt sich um ein (17 - 16 - 8)-Schichten-feedforward perceptron mit den Rückkopplungen R1, R2 und R3. Die Rückkopplungen verbinden Teile der Outputneuronen mit den Inputneuronen zur Erfassung des letzten Bewegungszustandes und zur Verarbeitung der Bewegungshistorie (k-1-te, k-te, k+1-te Bewegung).

Die Output-Neuronen lassen sich grob in Neuronen für Rotation (Δa, Δb, Δc, oberer Teil in Abb. 7.13) und in Neuronen für Translationen (Δx, Δy, Δz, unterer

Teil) unterteilen. Rotation verändert die Orientierung des TCP unter Beibehaltung seiner Position. Translation verändert die Position des TCP unter Beibehaltung seiner Orientierung. Diese Aufspaltung ist günstig, da, falls sich das aktive Fügeteil in der Nähe der Fügeendposition oder grundsätzlich in der Nähe von umgebenden Wandungen befindet, Rotationen häufig zu Kollisionen führen. Für diese Fälle ist es deshalb günstiger Ausrichtbewegungen bei ausreichendem Abstand von umgebenden Körpern auszuführen bzw. definiert nur Rotationen oder nur Translationen zuzulassen (z.B. Bewegung parallel zu ebenen Flächen).

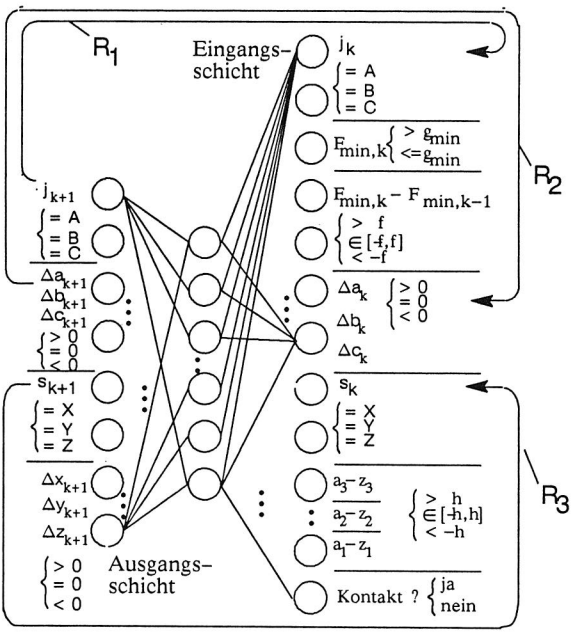


Abb. 7.13: Rückgekoppelte Netzstruktur für die Feinbewegungssteuerung

Die sechs Input-Neuronen der Translationen nehmen die Ergebnisse aus dem Vergleich zwischen den gemessenen Sensorwerten und den Sollwerten im Zielpunkt für Sensor i auf ($a_i - z_i$). Die Selektoren j bzw. s kennzeichnen die, bei der aktuellen Bewegung beteiligten Euler-Drehwinkel bzw. kartesische Koordinaten. Zwei Input-Neuronen beschreiben den Zustand der letzten Bewegung zum Zeitpunkt t-1. Ein weiteres Input-Neuron zeigt an, ob die Zielposition erreicht wurde. Zwei Output-Neuronen legen die im nächsten Schritt anstehende Bewegungsrichtung fest. Zwei

Neuronen definieren das Inkrement der Feinbewegung (>0, =0, <0). Die Input-Neuronen der Rotation arbeiten analog zur Translation. Der Vergleich zwischen gemessenen und zu erreichenden Sensorwerten kann durch eine konventionelle, algorithmische Vorverarbeitung (Fehlerfunktion) ersetzt werden; z. B. nach Gl. (7.1):

$$F_{min} = \frac{1}{r}\sum_{i=1}^{r}\left(\frac{1}{m}\sum_{j=1}^{m}\left(a_{ij} - z_{ij}\right)\right)^2 \qquad (7.1)$$

mit: i = aktueller Sensor (1 - 6), r = Anzahl der Sensoren im Greifer, z = Zielsensorwert, a = Gemessener Wert, m = Anzahl der Meßpunkte (nur für scannende Sensoren; m=1 für Abstandssensoren). Durch diese Sensorsignal-Vorverarbeitung kann die Neuronenzahl des NN-Controllers, zumindest zu Testzwecken, verringert werden. Für Laserscanner kann der Scanwinkel in Sektionen unterteilt werden, da sich in ungünstigen Fällen Meßwerte kompensieren können (vgl. Abb. 3.4). Der Funktionswert F(t) kann mit F (t-1) verglichen werden und als Kennzeichen des 'Erfolges' der letzten Bewegung als Input eingegeben werden. Schwellwerte f und h sind vorgegebene Genauigkeitsschranken, die die Dreh- bzw. translatorische Bewegung beenden. In Erweiterung können für die binär codierten Eingangsvektoren Grenzbereiche definiert werden (Fuzzy-Eingang; Die Neuronen werden mit dem Grad der Zugehörigkeit zu vorgeg. Fuzzy-Subsets belegt).

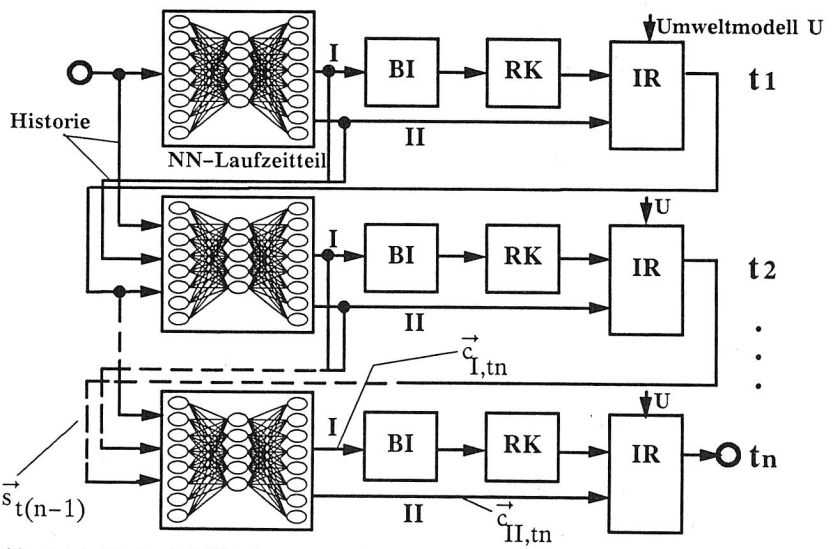


Abb. 7.14: Blockschaltbild des neuronalen Controllers

Der entwickelte Controller (NN) wurde als eigenständiger Prozeß in die Multi-Prozessing-Architektur des Aktionsplanungs- und Steuerungssystems integriert (vgl. Abb. 2.6). Abb. 7.14 zeigt das Blockschaltbild des Sensor-Roboter-Regelkreises. Der Controller wurde aus Laufzeitgründen in C implementiert. Ist die neuronale Steuerung aktiv, so sind der konventionelle RC-Sprachen-Interpreter und die Befehl-Präparation nach Kapitel 2.2.2 grundsätzlich deaktiviert. Die Aktivierung von Interpolation und Transformation richtet sich nach der Wahl des Stelleingriffs (I) oder (II). Im Falle des Stelleingriffs I wird ein Output-Vektor im 6-dim. Euler-Koordinatensystem erzeugt. Bahninterpolator (BI) und Rückwärtskinematik (RK) müssen deshalb nachgeschaltet werden. Für Stelleingriff II wird Interpolation und Kinematikmodul überbrückt. Das Neuronale Netz erzeugt direkt Gelenkwinkel. Die Kommunikation zwischen Objekten (in unterschiedlichen Prozeßbereichen BI, RK, IR, U, NN) erfolgt teilweise mit synchronem und teilweise asynchronem Nachrichtenaustausch (Abb. 7.15). Die Datenbasis stellt die erforderlichen Grunddaten bereit.

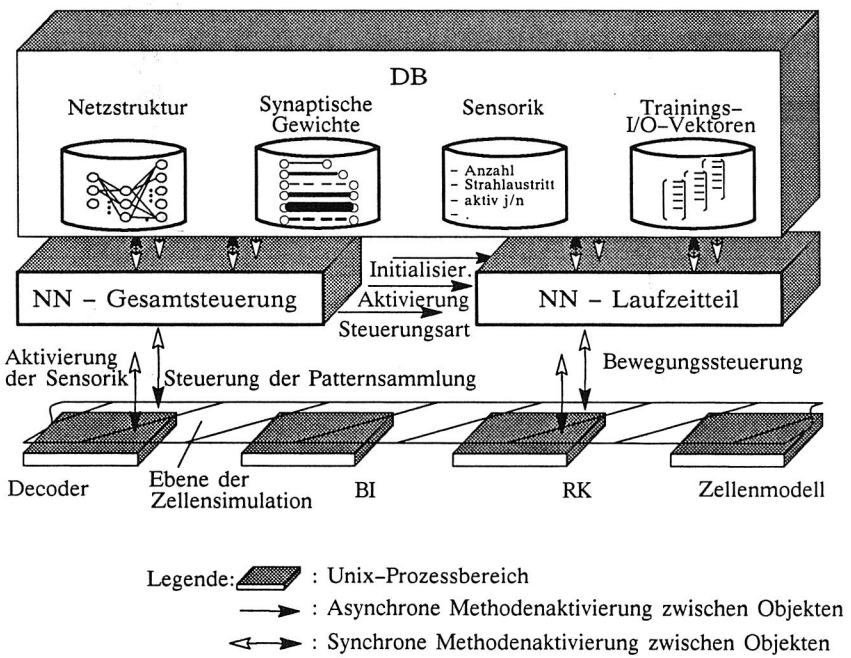


Abb. 7.15: Nachrichtenaustausch zur neuronalen Bewegungssteuerung

7.2.3 Aufnahme und Filtern von I/O-Pattern

Für den Trainingsprozess ist eine ausreichende Anzahl charakteristischer Pattern erforderlich. Besitzen die Input- und Output-Vektoren, wie z.B. bei der Feinbewegung, noch große Dimensionen, so ist das manuelle Erstellen der Input- und Output-Muster nicht mehr sinnvoll. Im folgenden wird deshalb ein Verfahren zur automatisierten Patternaufnahme vorgestellt (Abb. 7.16).

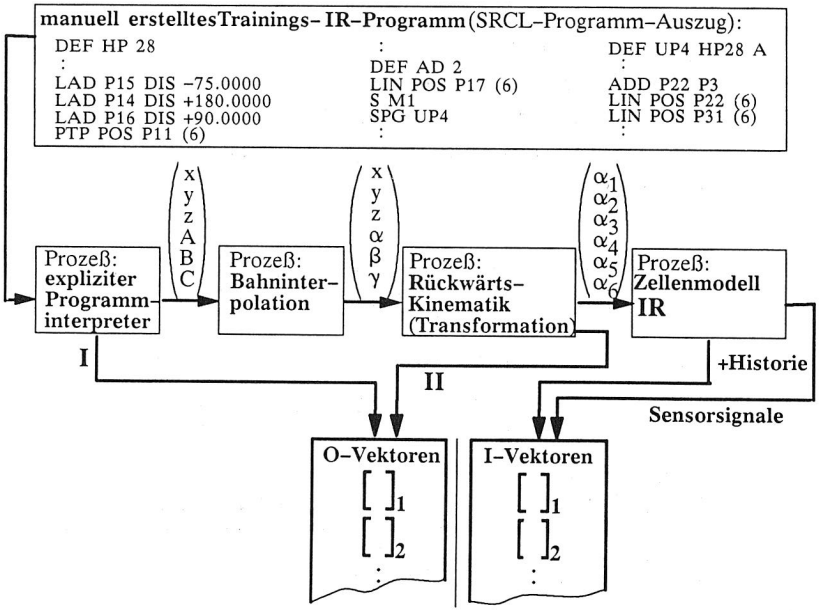


Abb. 7.16: Blockschaltbild für die Aufnahme der I/O-Muster

Während der Ausführung manuell erstellter IR-Programme (SRCL) kann der Abgriff der Input- und Output-Vektoren (pro Bewegungsinkrement) aus dem Anlagenmodell zu- oder abgeschaltet werden. Die konventionellen IR-Programme werden im Simulator interpretiert (RC-Interpreter). Parallel dazu werden die Pattern durch paarweises Speichern der Sensorvektoren in Korrelation zu den Bewegungsvektoren für jeden Zeitschritt aufgezeichnet. Diese Programme müssen Feinbewegungen beinhalten, die die Bewegungsaufgabe lösen ('Gut-Beispiele'). Die so gesammelten I/O-Vektor-Paare sind die Beispiel-Muster für das anschließende Training des neuronalen Netzwerkes.

Abb. 7.17 zeigt mögliche Bewegungsabläufe zum Training von Linearbewegungen (1. und 2.) und Rotationsbewegungen (3.: von a nach b: Verdrehen, von b nach c: Ausrichten). Das betriebsfertig trainierte Netzwerk soll dadurch unempfindlich gegen geringfügige Verkippungen und Verschiebungen der Fügepartner relativ zueinander werden.

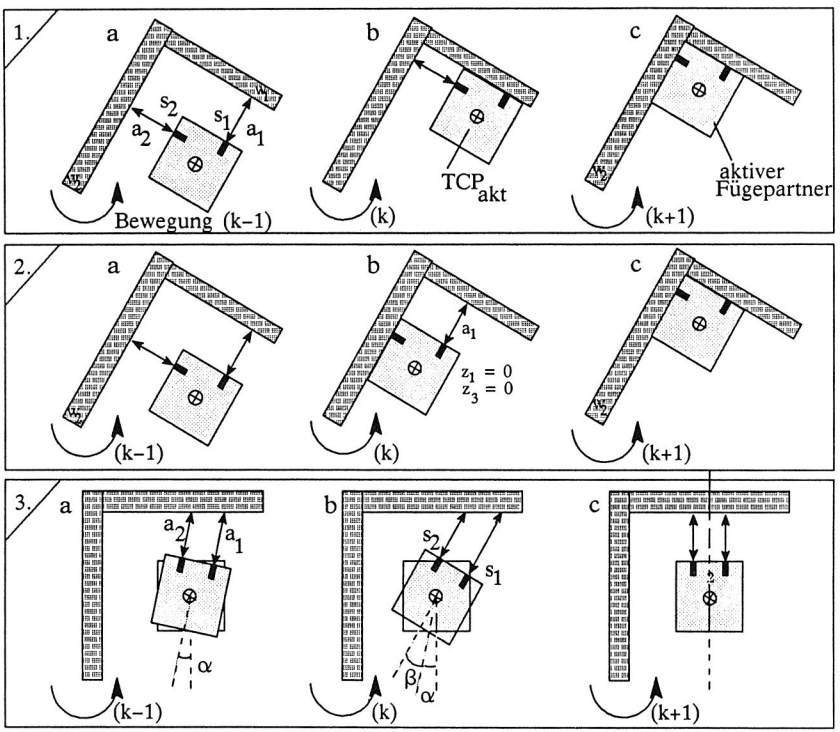


Abb. 7.17: Exemplarische 'Gut'-Bewegungsabläufe (zur Veranschaulichung nur zweidimensional dargestellt am Beispiel 'in Raumecke legen')

Der Vorteil der Lernvektoren-Aufzeichnung im Roboterzellensimulator besteht darin, daß die zeitaufwendige Lernphase und Patternsammlung nicht an der realen Anlage erfolgen muß; d.h. kein Ausfall der Anlage für die Produktion. Die Phase der Vektoraufzeichnung ist aufwendig (umfangreiche SRCL-Programme) und würde das reale HHS (Mechanik, Getriebe, Antriebe) stark beanspruchen (Verschleiß, Einlaufgefahr durch Bewegungen in einem sehr kleinen Arbeitsbereich). Die Trainingsphase fällt

jedoch für eine Fügesituation nur einmalig an. Die Aufzeichnung der Vektoren kann am Simulator "über Nacht" durchgeführt werden. Anschließend müssen die so gewonnenen I/O-Vektoren noch gefiltert werden. Filtern soll folgende Punkte umfassen:

- Redundante Vektor-Paare entfernen
- Fehlerhafte und unvollständige I/O-Vektor-Paare entfernen
- Widersprüchliche I/O-Vektor-Paare entfernen (z.B. I/O-Vektor-Paare, die zu einem identischen Input-Vektor unterschiedliche Output-Vektoren besitzen).

In der Praxis hat sich gezeigt, daß eine gründliche Filterung der I/O-Vektoren besonders bei maschineller Patternsammlung wichtig ist, da z.B. unentdeckte Widersprüche zu zunächst unerklärbarem Konvergenzverlust der Lernverfahren führen.

7.2.4 Training des neuronalen Netzes

Während des Trainingsprozesses werden die Beispiel-Muster gelernt (Abb. 7.18). Der Lernfehler wird, anstatt erst nach einer Trainingsepoche, nach jedem Einzelmuster durch das Netz propagiert.

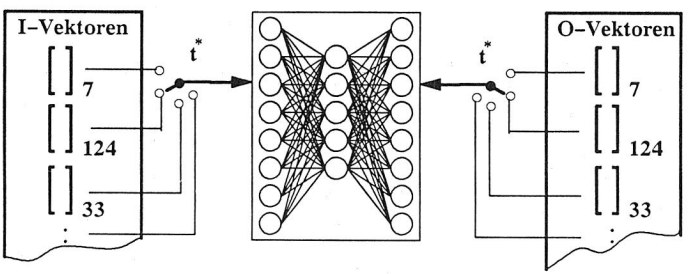


*Abb. 7.18: Training durch getaktetes Anlegen der I/O-Vektor-Paare im Takt t**

Abb. 7.19,oben zeigt den Trainingsverlauf eines einfachen, in ein Rotations- und Translations-Subnetz unterteilten Netzwerkes. Über der Anzahl der Trainings-Epochen ist der Gesamtfehler E logarithmisch aufgetragen (TSS; Kap. 6.1) als Funktion der Anzahl von hidden units (Kurvenschar). Die Abhängigkeit ist sehr stark, so daß z. B. durch Erhöhung von 5 auf 6 der Lernprozess konvergiert, während bei 4 Hidden-Units Schwingungserscheinungen auftreten. Abb. 7.19,unten zeigt den Trainingsverlauf für das Netzwerk nach Abb. 7.13. Trainiert wurden ca. 183 Vektor-Paare. Er konvergiert ab 6 hidden units. Hohe Rechenzeiten entstehen durch Festlaufen in den lokalen Minima bei F = 1 und F = 2.

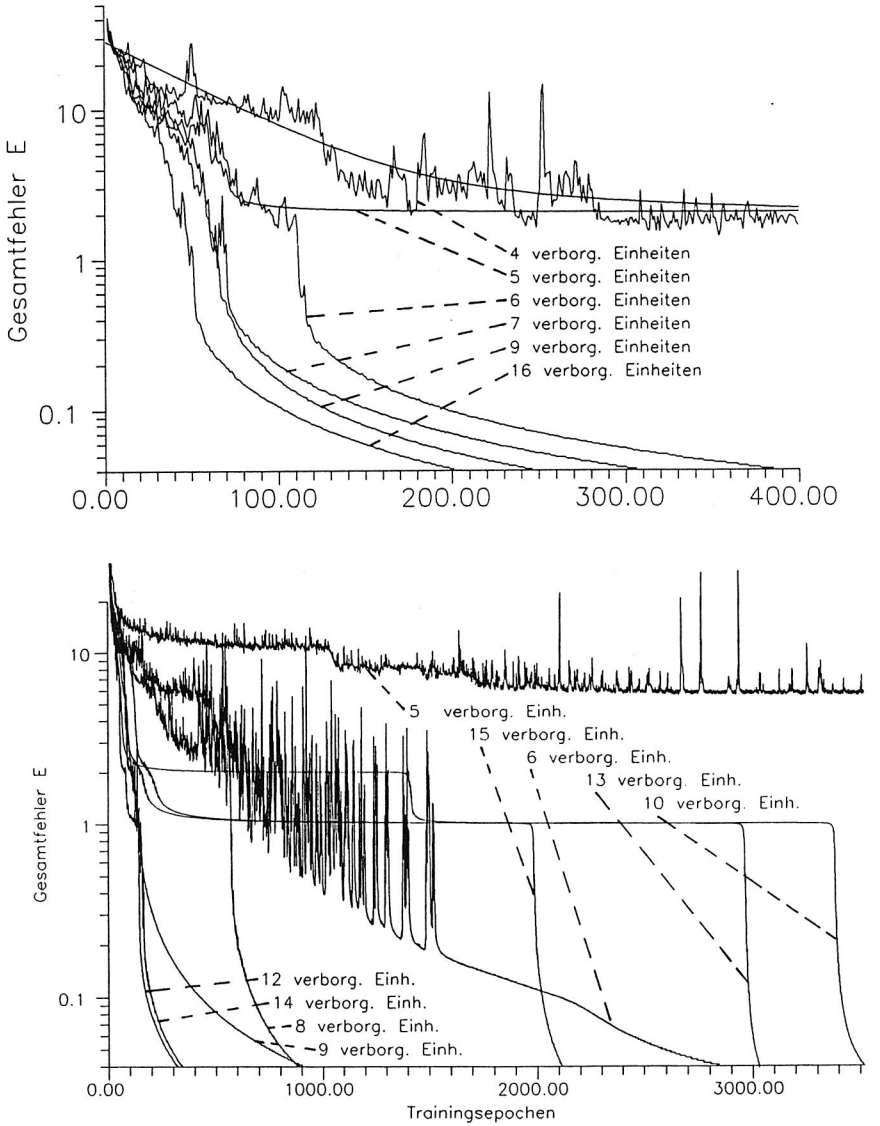


Abb. 7.19: Entwicklung des Gesamtfehlers F (einfachlogarithmisch aufgetragen)

7.2.5 Experimente und Beurteilung

Dieser Abschnitt behandelt die Übernahme der Steuerung durch den Controller. Der Trainingsprozeß der synaptischen Gewichte ist beendet. Aus den an einem Neuron eintreffenden Signale werden die berechneten Ausgangssignale über unidirektionale Verschaltungen zu allen angeschlossenen Neuronen geleitet. Abb. 7.20 zeigt die Integration in das prozeßnahe Werkzeug.

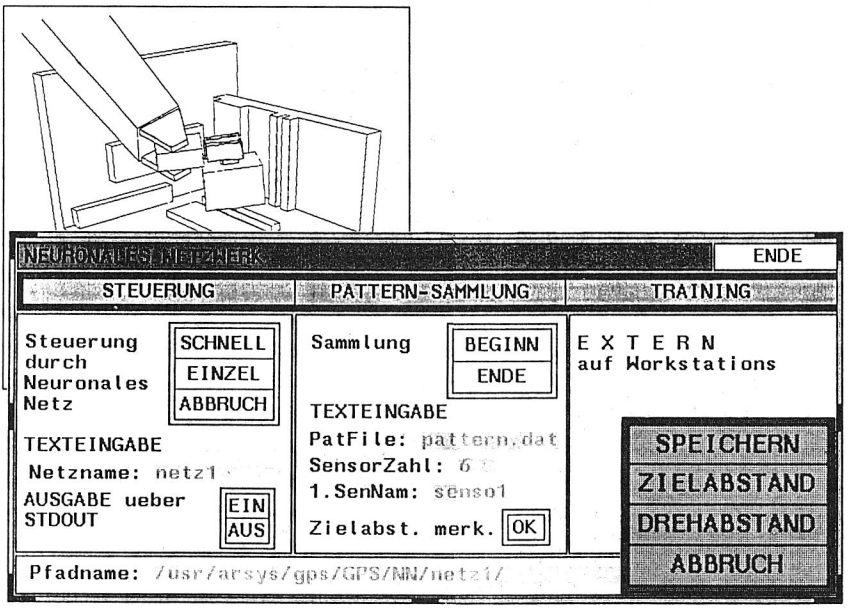


Abb. 7.20: Musteraufzeichnung in der Zellensimulation (Bildschirmkopie)

Die PATTERN-SAMMLUNG wird vom Benutzer (Oberfläche, Abb. 7.20) angestoßen bzw. abgeschaltet, sowie die Sensorsimulation aktiviert. Die rechenintensive Trainingsphase wurde nicht auf PC durchgeführt, sondern auf leistungsfähigere Workstations ausgelagert (s. Kap. 6.2, Abb. 7.21).

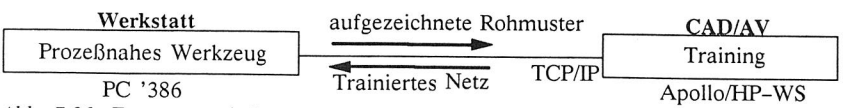


Abb. 7.21: Trennung zwischen Muster-Aufzeichnung und Training

Abb. 7.22 stellt typische, während der Feinbewegung aufgenommene Sensorwertverläufe über den Netzwerkaufrufen dar. Die Fügeaufgabe bestand im Einlegen eines Quaders in eine Raumecke mit zerklüfteten Referenzflächen (s. Abb. 7.20). Aktives und passives Fügeteil sind gegenüber der Trainingssituation relativ zueinander verkippt und verschoben. Roboter und Anlage werden simuliert. Die Sensoren erreichen nach Abschluß der Fügebewegung stabile Zielwerte.

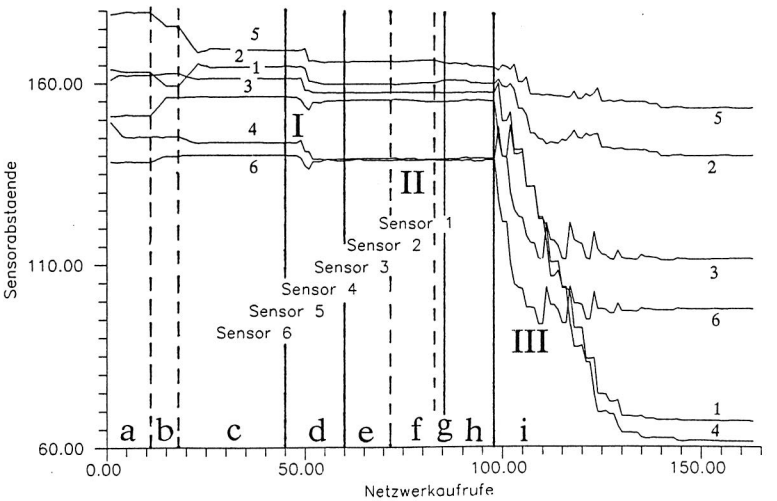


Abb. 7.22: Sensorwerte in Abhängigkeit der Netzwerkaufrufe

Wurden zwei Sensoren so angebracht, daß ihre Meßstrahlen parallel zueinander verlaufen und beide in der Fügeendposition senkrecht auf eine Wandung zeigen, so müssen im Falle der nicht ebenen Referenzflächen die beiden Sensorwerte stabile, aber um den Höhenunterschied verschobene Signale liefern (Abb. 7.22, Abschnitt III). Die Kurven haben gemeinsam, daß sie in Sektionen a, b, c, etc. unterteilt sind. Innerhalb einer Sektion findet die überwiegende Bewegung in Richtung einer generalisierten Koordinate aus x,y,z,A,B,C statt (von a bis h: kombinierte Bewegungen, ab Bereich i: nur noch Translation).

Um die Zielposition mit höherer Genauigkeit zu erreichen, kann der Controller so trainiert werden, daß der Betrag der Bewegungsinkremente um so geringer wird, je näher die Fügeendposition erreicht ist (dynamisch angepaßt).

Obige Experimente zeigten deutliche Schwächen bei komplexeren Feinbewegungen. Hierzu wäre eine Erhöhung der Neuronenzahl erforderlich, die jedoch an den hohen Trainingszeiten scheiterte. Die Trainingszeiten (CPU-Zeit auf WS) betrugen für ein Netz mit geringem Vernetzungsgrad der Rotations- und Translationssubnetze unter einer Stunde. Die Trainingszeiten für das Netzwerk nach Abb. 7.13 lagen im Bereich von fünf Stunden. Komplexe Fügebewegungen (> 3000 Muster-Paare) führten auf Rechenzeiten von mehreren Tagen.

Das größte Hindernis für den Einsatz von neuronalen Controllern bildet derzeit die fehlende Rechenleistung, die sich vor allem beim Training sehr auswirkt. Spezielle Hardware z.B. Mehrprozessorsysteme (Neurocomputer nach /118/) versprechen hier Abhilfe. Bildverarbeitungssysteme sind für die Feinmotorik nicht sinnvoll einsetzbar. Weitere Fortschritte in der Miniaturisierung der nicht-bildverarbeitenden Sensorik sind Voraussetzung für den zukünftigen praktischen Einsatz.

8. Zusammenfassung

Für das Ziel der impliziten Sensor- und Geräteprogrammierung wurden verteilte Werkzeuge konzipiert. Sie kommunizieren in einer CAE/CAM-Verfahrenskette. Hierzu wurden im Rahmen der vorliegenden Arbeit Werkzeuge im CAD-Bereich, sowie ein prozeßnahes Werkzeug auf Werkstattebene entwickelt. Zunächst wurde deren Architektur erarbeitet. Dies erfolgte auf der Basis objektorientierter Ansätze und Standards (Unix, C), sowie in Hinblick auf zukünftige Multiprozessor-Gerätesteuerungen in Roboterzellen. Methoden der Künstlichen Intelligenz wurden für die implizite Geräteprogrammierung genutzt. Fertigungszellen werden dadurch intelligenter. Leistungsfähige, komplexe Betriebsmittel werden einfacher bedienbar.

Im weiteren wurden wesentliche Komponenten der impliziten Aktionsplanung und -steuerung konzipiert und prototypisch realisiert. Hierzu mußten Verfahren entwickelt werden, zur selbständigen Zerlegung von impliziten, produktnahen Fertigungsanweisungen (Zielvorgaben) in Aktionsprimitive. Dies erfordert massiven Rechnereinsatz. Hierauf, und auf der Geometrieanalyse von CAD-Volumenmodellen, basiert auch die Abarbeitung von vorab erstellten Montageplänen, die implizite Fertigungsaktionen beinhalten. Die Auflösung der Fertigungsbefehle mündet in die automatisierte Erstellung von Steuerungscode für die beteiligten Gerätesteuerungen.

In der Praxis entstehen bei reinen Offline-Systemen häufig Schwierigkeiten aufgrund der Entfernung vom eigentlichen Prozeß. Es wurden deshalb Lösungen entwickelt, die die Vorteile der Online-Verfahren mit denen der Offline-Verfahren kombinieren. Hierzu wurden Methoden zur impliziten Sensorintegration (z.B. vom Industrieroboter

greifbarer, mitbewegter CCD-Sensor) entwickelt, mit deren Hilfe das stets mitlaufende Anlagenmodell mit der realen Anlagengeometrie abgeglichen werden kann. Gegenüber CAD- oder AV-Werkzeugen mußten für das werkstattorientierte System verschärfte Anforderungen, wie beispielsweise bezüglich Laufzeiteigenschaften und Einfachheit der Bedienung, berücksichtigt werden.

Zentrales Element im Hintergrund ist eine Daten- und Methodenbank, auf die die Werkzeuge verteilt zugreifen. Hierfür wurde ein konzeptionelles Datenbankschema entworfen und in einer relationalen Datenbank implementiert. Dies umfaßt die Entwicklung eines Werkzeuges, das die Phase der Betriebsmittelauswahl bzw. Konstruktion unterstützt, und parallel dazu die Wissensbasis füllt. Implizite Fertigungsoperationen mußten zunächst inhaltlich (Fertigungstechnologie) und rechnergeeignet definiert und modelliert werden.

Als konsequente Weiterentwicklung konventioneller, analytischer Verfahren der Betriebsmittelprogrammierung wurde ein Ansatz entwickelt, der Lernen mit künstlichen, neuronalen Netzwerken in Produktionsanlagen umfaßt. Auf der Basis des Expertensystem-Tools KEE wurde hierzu ein CAE-Werkzeug konzipiert und exemplarisch realisiert, das wissensbasierte Verfahren und neuronale Ansätze verbindet. Es automatisiert die Lernmuster-Aufzeichnung und den Trainingsprozeß bis hin zur automatischen Netzwerkverschaltung und Downloading in die Zielhardware. Zum Nachweis der praktischen Eignung wurden betriebsfertig trainierte, neuronale Netzwerke, vom Entwicklungswerkzeug aus, in die Zielhardware 'SPS' geladen und dort am Beispiel von Mustererkennungsaufgaben erprobt. Alle Lernprozesse wurden ausschließlich auf Grundlage eines erweiterten Back-Propagation-Verfahrens ausgeführt.

Besondere Bedeutung in automatisierten Montagezellen besitzt eine Feinbewegungssteuerung, die gegenüber Ungenauigkeiten tolerant ist. Auf der Grundlage eines rekursiven Feed-Forward-Netzwerkes wurden hierzu Möglichkeiten zum Lernen der Feinmotorik (sensor-motorisches Verhalten) eines 6-Achs-Industrieroboters während Montagefeinbewegungen untersucht. Dieser Ansatz wurde mit Hilfe der Sensorsimulation und anhand einfacher Bewegungen (Einlegen in Raumecke) in der Zellensimulation realisiert. Hierzu wurde ein neuronaler Bewegungs-Controller implementiert, dem Abläufe eintrainiert, anstatt programmiert werden.

Zur Erprobung der entwickelten Verfahren und Werkzeuge wurde als Versuchsstand eine Roboterzelle mit Bildverarbeitungssystem und Rechnerkommunikation aufgebaut.

9. Literaturverzeichnis

/1/	Abidi M. A.; Gonzalez R.C.:	The Use of Multisensor Data for Robotic Applications. IEEE Transactions on Robotic and Automation, Vol. 6, No. 2., (1990).
/2/	Adolphs, P.; Horsch, Th.; Schmid, D.(Hrsg.):	Bewegungssteuerung zukünftiger Robotersysteme. in : Fortschrittliche Robotersteuerungstechnik. Fachberichte Messen, Steuern, Regeln, Springer-Verlag, Berlin; Heidelberg (1991), S. 137-165.
/3/	Aho, A. V.; Sethi, R.; Ullman, J. D.:	Compilers - Principles, Techniques, and Tools. Addison-Wesley Publishing Company, New York (1986)
/4/	Aiken, S.W.; Koch, M.W.; Roberts, M.W.:	A Parallel Neural Network Simulator. IJCNN International Joint Conference On Neural Networks, San Diego, California, June 17-21, 1990, p. II-611 - 616.
/5/	Anderson, J.A.; Wisniewski, E.J.; Viscuso, S.R.:	Software for Neural Networks. Computer Architecture News, Vol. 16, No.1, März 1988, ACM Press, p. 26-36.
/6/	Angermüller, G.; Niedermayr, E.; Roth, N.:	Off - line programming and simulation of flexible assembly. Assembly Automation 9(2), IFS Publications (1989), p.97-102.
/7/	Arbab, F.:	Set Models and Boolean Operations for Solids and Assemblies. IEEE Computer Graphics & Appl. 11/90 (1990), p.76-86.
/8/	Asada, H.:	Teaching and learning of compliance using neural nets: representation and generation of nonlinear compliance.

IEEE Int. Conf. on Robotics and Automation, Cincinnati, May 13-18, 1990, p. 1237-1243.

/9/ Barber, K.S.; Agin, G.J.: Analysis of Human Communication During Assembly Tasks. Technical Report No. CMU-RI-TR-86-13, Carnegie-Mellon University, Pittsburgh (1986).

/10/ Barr, T.: Netz im Aufwind. c't, Heft 4/1991, (1991); mit Beilage: Unveröffentlichte Diplomarbeit von Barr, T.

/11/ Bartholomew, O.N.; Jau-Yen, Chu; Akrep, M.: A Schema for CAD-based Robot Assembly Task Planning for CSG-modeled Objects, Journal of Manufacturing Systems, Vol. 7/No. 2, (1988) p. 131-145.

/12/ Bathke, H.: Kräftig in allen Richtungen - Kraft- und Momentensensoren in der Robotertechnik. QZ 34 (1989) Heft 9, Hanser Verlag, München, S.473-476.

/13/ Bavan, A.S.: NPS: A Neural Network Programming System. IJCNN International Joint Conference On Neural Networks, San Diego, California, June 17-21, 1990, p. I-143 - 148.

/14/ Blume, C.; Jakob, W.: Programmiersprachen für Industrieroboter, Vogel Verlag, Würzburg (1983).

/15/ Bologni, L.: Robotic grasping: How to determine contact positions. IFAC Robot control, Karlsruhe (1988), p. 395-399.

/16/ Buckley, S. J.: Planning Compliant Motion Strategies, The International Journal of Robotics Research, Vol. 8, No. 5, October MIT Press 1989, p. 28-44.

/17/ Codd, E.F. ; Rustig, R. (Hrsg): Further Normalization of the Data Base Relational Model. in: Courant Computer Science Symposium "Data Base Systems", Prentice Hall, New York (1971), p. 33-64.

/18/ Croall, I.; Mason, J.P. (eds.): Applications of Neural Networks for Industry in Europe. Project Handbook, ANNIE (ESPRIT Project 2092), Oxon (1991).

/19/ Cutkowsky, M.R.: Robotic Grasping and Fine Manipulation. Kluwer Academic Publishers, Standford University, Boston (1985).

/20/ Date, C.J.: A Guide to INGRES. Addison-Wesley, Reading; Menlo Park (1987).

/21/ Date, C. J. : An Introduction to Database Systems Volume II, Addison-Wesley, Reading, Mass. (1985).

/22/ Denavit, J. ; Hartenberg R. S.: A kinematic notation for lower-pair mechanisms based on matrices. ASME J. APPL. MECH. 22, p. 215-221, (1955).

/23/ Desai, R.; Volz, A.: Contact Formations and Design Constraints: A New Basis for the Automatic Generation of Robot Programms. in CAD-Based Programming for Sensory Robots. NATO ASI Series F, Vol. 50, Springer-Verlag, New York p. 361-395, (1988).

/24/ Diehl, R.; Grabowski, H.; Huber, R.: Kollisionsüberprüfung mit hierarchischen Approximationsmodellen. VDI-Z 132, Nr. 1(1990), S.32-39.

/25/ Diep, T.-V.: Spur G. (Hrsg.): Ein Beitrag zur Kompensation von geometrischen Fehlern in flexibel automatisierten Fertigungssystemen. Reihe Produktionstechnik Berlin, Hanser Verl., München (1989).

/26/ Dillmann, R.: Lernende Roboter. Fachberichte Messen, Steuern, Regeln, Springer-Verlag, Berlin (1988).

/27/ Dittrich, K.R.; Dayal, U.; Buchmann A.P.: On Object-Oriented Database Systems. Springer-Verlag, New York (1991).

/28/ Duelen, G.; Schröer, K.: Praktische Resultate der Roboter-Kalibration. ZwF85(1990)2, Hanser-Verlag, München, S.113-116.

/29/ v. Dungern, O.; Schmidt, G.: Vorbereitende und begleitende Ablaufplanung für flexible Montagezellen in industrieller Umgebung. Robotersysteme 6, 225-235(1990), Springer-Verlag, Berlin; Heidelberg.

/30/ Ehrlenspiel, K.: CAD-Teileverwaltung mit relationaler Datenbank, CAD/CAM 5/89, (1989).

/31/ Eisele, R.: Konzeption und Wirtschaftlichkeit rechnerintegrierter Planungssysteme. Hanser Verlag, München; Wien (1990).

/32/ ElMaraghy, H. A.; Johns, B.: An Investigation Into the Compliance of SCARA Robots Part 1: Analytical Model, Part 2: Experimental and Numerical Validation. J. of Dynamic Sys., Measurement, and Control, Transactions ASME, (1988), Vol. 110, p.18-30.

/33/ Encarnacado, J; Krause, F.L.: File structures and databases for CAD. North Holand (1982).

/34/ Feldmann, K.: Fortgeschrittene Produktionstechnik - Stand der Entwicklungsarbeiten in der Bundesrepublik Deutschland. Internationales Seminar "Excellence in Manufacturing", Mailand, Italien, 17.05.1989

/35/ Feldmann, K.; Reinisch, H.: Implizite NC-Geräte-Programmierung. CIM-Management 2/92, Oldenbourg Verlag München (1992).

/36/ Föhr, R.: Photogrammetrische Erfassung räumlicher Informationen aus Videobildern. Fortschritte in der Robotik, Vieweg Verlag, Braunschweig (1990).

/37/ Foley, J. D.; Van Dam, A.: Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading Mass. (1983).

/38/ Forssmann, W.G.; Heym, C.: Neuroanatomie. Basistext Medizin, Springer-Verlag, Heidelberg (1985).

/39/ Freund, E.; et al.: Osiris-ein objektorientiertes System zur impliziten Roboterprogrammierung und Simulation. Robotersysteme 6, S. 185-192, Springer (1990).

/40/ Frommherz, B.; Hörmann, K.: A Concept for a Robot Action Planning System, NATO ASI Series, Vol. 29, Springer-Verlag, New York (1987).

/41/ Gairola, A.: Design Analysis for Automatic Assembly. Proceedings of the 8th International Conference of the Frauenhoferinstitute for Industrial Engineering at the University of Stuttgart, 8/85, Springer-Verlag 1985, S.441-447.

/42/ Gevarter, W.B.: Intelligente Maschinen. VCH, Weinheim; New York (1987).

/43/ Geyer, G.: Entwicklung problemspezifischer Verfahrensketten in der Montage. Hanser Verlag, München; Wien (1991).

/44/ Göhner, M.; Schmid, K.-H.: Einheitliche Programmierung v. Industrierobotersystemen mit einem Strukturedítor. Robotersysteme 5, 141-148 (1989), Springer-Verlag, Berlin, Heidelberg.

/45/ Gruhler, G.; Wieland, E.: Produktorientierte Programmierung von Montagerobotern. Feinwerktechnik & Meßtechnik 97(1989) 4, Hanser Verlag, München, S.167-170.

/46/ Hayes-Roth, B.: Blackboard Architecture for Control. Journal of AI, 26 (1985), p. 251-321.

/47/ Heiß, H.: Theorie und Anwendung der Koordinatentransformation bei Roboterkinematiken. Informatik Forschung und Entwicklung (1987)2:19-33, Springer-Verlag, Berlin, Heidelberg.

/48/ Hemani, A.; Postula, A.: Cell Placement By Self-Organisation. Neural Networks, Vol. 3, 1990, p. 377-383.

/49/ Hertz, J.; Krogh, A.; Palmer, R.G.: Introduction of the theory of neural computation. Santa Fe Insitute, Addison-Wesley, Redwood-City (1991).

/50/ Hirose, Y.; Yamashita, K.; Hijiya, S.: Back-Propagation Algorithm Which Varies the Number of Hidden Units. Neural Networks, Vol. 4, p. 61-66, (1991).

/51/ Hörmann, A.; Rembold, U. (Hrsg.): Petrinetze zur Darstellung von Aktionsplänen für Multirobotersysteme. 4. Fachgespräch, Autonome Mobile Systeme, Univ. Karlsruhe (1988), S. 57-71.

/52/ Hörmann, K.; Werling, V.: Ein Verfahren zur Planung von Feinbewegungen für Montageoperationen. Robotersysteme 5, 17-28(1989), Springer-Verlag, Berlin; Heidelberg.

/53/ Holler, M.; et al.: An Electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses. ICNN, Int. J. Conf. on Neural Netw.,(1989).

/54/ Honkomp, E.; Lehner, H.: Distance Measurement by light shift of Laserdiode light. Applied Optics, 25/2976, (1986).

/55/ N.N.: Intel data sheet on 80170NW. Intel corporation, (1990).

/56/ N. N.: IRDATA, Allgemeiner Aufbau, Satztypen und Übertragung, VDI 2863/1, (1987)

/57/ N.N.: IRL Sprachentwurf, Normenausschuß Maschinenbau im DIN, FB-IA, UA 96.2.2, Frankfurt (1989).

/58/ Jeng, M.D.; DiCesare, F.: A Review of Synthesis Techniques for Petri Nets. IEEE Proc., Computer Integrated Manufacturing, New York (1990).

/59/ N.N.: KEE-Software Development System, Technical Manuals, KEE-3.05. Intellicorp., Menlo Park (1987).

/60/ Kleineidam, G.: CAD/CAP: Rechnergestützte Montagefeinplanung. Hanser Verlag, München; Wien (1990).

/61/ Kleemann, U.: Regelung elastischer Roboter. Fortschrittsbericht VDI, Reihe 8, Nr. 191, VDI-Verlag, Düsseldorf (1989).

/62/ Kroszynski, U.I. et al.: Geometric Data Transfer Between CAD Systems: Solid Models. IEEE Computer Graphics & Appl. 9/89 (1989), p.57-71.

/63/ Latombe, J.-C.: Robot Motion Planning. Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston (1991).

/64/ Levi, P.: Planen für autonome Montageroboter. Informatik-Fachbericht 191, Springer-Verlag, Berlin (1988).

/65/ Liebermann, L.; Wesley, M.: Autopass: An Automatic Programming System for Computer Controlled Mechanical Assembly. in IBM, J.Res. Dev., (1977).

/66/ Lozano-Perez, T.; Brooks, R. A.: An Approach to Automatic Robot Programming. Massachusetts Institute of Technology, MIT Press (1985) and in: Boyse, J. et al. (eds.), Solid Modeling by Applications, Plenum Press, New York (1984).

/67/ Malcolm, C. A.; Fathergill, A. P.: Some Architectual Implications of the Use of Sensors. NATO ASI Series, Vol. 29, Springer-Verlag, New York (1987).

/68/ Mason, M. T.: Compliance and Force Control for Computer Controlled Manipulators. IEEE Transaction on systems, man, and cybernetics, Vol. SMC-11, No 6, June 1981, p. 418-432.

/69/ Meier, A.: Erweiterung relationaler Datenbanken für technische Anwendungen. Springer-Verlag, Berlin Heidelberg (1987).

/70/ Meyer-Wegener,K.: Multimedia-Datenbanken. B.G.Teubner, Stuttgart (1991).

/71/ Mertens, P.; Deisenroth, M. P.: Programming industrial robots for flexible palletizing. Robotersysteme 4, 9-16 (1988), Springer-Verlag, Berlin.

/72/ Mollath, G.; Nickolay, B.: Systeme für mehrdimensionale Kraft- und Momentenmessung, ZwF82 (1987)6, Hanser-Verlag, München, S. 352-357.

/73/ Motro, A. : VAGUE: Interface to Relational Databases. ACM Transactions Information Systems, Vol. 6, No. 3, p.187-214, (1988).

/74/ Nilsson, H.J.: Prinziples of Artifical Intelligence. Springer-Verlag, Berlin, Heidelberg (1982).

/75/ N.N. : Die Montage im flexiblen Produktionsbetrieb. Universität Stuttgart, SFB 158, SFB-Ergebnisbericht (1989).

/76/ Olschewski, U.: Roboter-Montagesysteme. Verlag TÜV Rheinland, Dortmund (1990).

/77/ Osterwinter M.: Steuerungsorientierte Robotersimulation. Vieweg Verlag, Braunschweig (1992).

/78/ Pätzold, B.; Raflick, M.: CAD*I Database, An Approach to an Engineering Database. Springer-Verlag (1990).

/79/ N.N.: Parasolid, Version 3.0, Release Notes, Apollo Aegis SR9 Implementation. Shape Data Ltd., Cambridge (1990).

/80/ Parthier, U.: Ingres: Datenbankmanagement der Zukunft. Chip special, Vogel-Verlag, Würzburg (1989).

/81/ Paul, R.P.: Robot Manipulators: Mathematics, Programming, and Control. MIT-Press, Cambridge (1983).

/82/ Pfeiffer, F.; et al. : Simulation und graphische Darstellung von Robotern bei kraftschlüssigen Bahnen. Arbeits- und Ergebnisbericht SFB 336, München (1991).

/83/ Pfeiffer, F.; Reithmeier, E.: Roboterdynamik. Teubner Verlag, Stuttgart (1987).

/84/ Piegl, L.: On NURBS: A Survey. IEEE Computer Graphics & Applications, 1/91 (1991).

/85/ Plaut, D.C.; Nowlan, S.J.; Hinton, G.E.: Experiments on learning Back-Propagation. CMU-CS-86-126, Carnegie-Mellon-University, Pittsburg, Pa. (1986).

/86/ Poechmueller, W.; Glesner, M.: Evaluation of state-of-the-art neural network customized hardware. Neurocomputing, Vol. 2, 90/91, p. 209-231, (1991).

/87/ Pomerleau, D.A.: Efficient Training of Artificial Neural Networks for Autonomous Navigation. Neural Computation, 3/1991, p. 88-97.

/88/ Poppelstone, R.J.; Ambler, A.P.; Bellos, I: RAPT: A Language for Describing Assemblies, Department of Artifical Intelligence. University of Edinbourgh, Edinbourgh (1978).

/89/ Pratt, T.W.: Definition of Programming Language Semantics using Grammars for Hierarchical Graphs. LNCS 73, Springer-Verlag, Berlin (1979).

/90/ Pritschow, G.; Frager, O.; Schumacher, H.; Wieland, E.: Programmierung von roboterbestückten Produktionsanlagen. Robotersysteme 5, 47-56(1989), Springer-Ver lag, Berlin; Heidelberg.

/91/ Puppe, F.: Einführung in Expertensysteme. Springer-Verlag, Berlin; Heidelberg (1988).

/92/ Raczkowsky, J.; Mittenbuehler K.H.: Simulation of Cameras in Robot Applications, IEEE Comp. Graphics a. Applications, (1989)

/93/ Rademacher, L.: Lösungsmöglichkeiten für das automatisierte Präzisionsfügen. ZwF85(1990)1, Hanser-Verlag, München,S.39-43.

/94/ Ramirez-Trevino, A.: Assembly Task in Robot Cells using Coloured Petri Nets. IEEE Proc., Computer Integrated Manufacturing, New York (1990).

/95/ Rechenberg, P; Mössenböck, H.: Ein Compiler-Generator für Mikrocomputer. Carl Hanser Verlag, München (1988).

/96/ Reinisch, H.: Datenbank für Betriebsmittel der Montageautomatisierung, ZwF, 85-7, Heft 7/90, Carl Hanser Verlag, München (1990).

/97/ Reinisch, H.: Entwicklung und Aufbau einer technischen Datenbank für die Lösungsunterstützung von Handhabungs- und Montageaufgaben. GID-Bericht Nr. K0902/T 6804, Erlangen (1990).

/98/ Ren, C. L.; Min-Hsiung, L.; Scherp, R. S.: Dynamic Multi-Sensor Data Fusion Systems for Intelligent Robots. IEEE Journal of Robotics and Automation Vol. 4, No. 4, (1988), S.386-396.

/99/ Ritter, H.; Martinetz, Th.; Schulten, K.: Neuronale Netze, Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke. Addison-Wesley, Bonn; München (1990).

/100/ Ritter, H.; Cruse, H.: Neural Network Approaches for Sensory-Motor-Coordination. it 6/91, 310-316, (1991).

/101/ Rogos, J. (Hrsg.): Intelligente Sensorsysteme in der Fertigungstechnik. Springer-Verlag, Berlin; Heidelberg (1989).

/102/ Roth, K.: Konstruieren mit Konstruktionskatalogen. Springer-Verlag, Berlin; Heidelberg (1982).

/103/ Rumelhart, D.E.; McClelland, J.L.: Parallel Distributed Processing. Vol. 1, MIT Press, Cambridge (1986).

/104/ Rumelhart, D.E.; McClelland, J.L.: Explorations in Parallel Distributed Processing. MIT Press, Cambridge (1988).

/105/ Sailsbury, J. K.: Active Stiffness Control of a Manipulator in Cartesian Coordinates. IEEE 01912216 95$00.75, 1980, S. 95-100.

/106/ Schlageter, G.; Stucky, W.: Datenbanksysteme: Konzepte und Modelle. B.G. Teubner, Stuttgart (1983).

/107/ Schlaich, G.; Gweon, D. G.; Cho, H.-S.: Industrieroboter mit taktilen Sensoren zur Kontaktteilmontage. wt Werkstattstechnik 77(1987) Nr. 12, Springer-Verlag, Berlin; Heidelberg, S.671-674.

/108/ Schmidt, R.F.: Grundriß der Neurophysiologie. Basistext Medizin, Springer-Verlag, Berlin (1987).

/109/ Scholz, W.: Modell zur datenbankgestützten Planung automatisierter Montageanlagen. Hanser Verlag, München; Wien (1989).

/110/ Schütze, P.: Kollisionskontrolle bei der Offline-Programmierung von Industrierobotern. Diss., RWTH Aachen, Aachen (1988).

/111/ Schugmann, R.: Nachgiebige Werkzeugaufhängungen für die automatische Montage. Forschungsbericht 24, Springer-Verlag, Berlin (1990).

/112/ Seyfferth, W.; Pfeiffer, F.: Dynamics of Rigid and Flexible Part Mating with a Manipulator. Proc. of IMACS-IFAC Symposium, Vol. 1, Lille (1991), p. 419-424.

/113/ Sommer, E.: Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen. Hanser Verlag, München; Wien (1992).

/114/ Steinmüller, P.; Mittler, H.: Montageautomation auf dem Prüfstand, RKW, Teil I,II. Eschborn (1985).

/115/ Stoyan, H.: Programmiermethoden der künstlichen Intelligenz, Band1, Springer-Verlag (1988).

/116/ Stoyan, H.; Görz, G.: LISP - Eine Einführung in die Programmierung. Studienreihe Informatik, Springer-Verlag, Berlin (1984).

/117/ Tollenaere, T.: SuperSAB: Fast Adaptive Back Propagation with Good Scaling Properties. Neural Networks, Vol. 3, 1990, p. 561-573.

/118/ Treleaven, P.C.: Neurocomputers. International Journal of Neurocomputing, ECN Neurocomputing GmbH, Ismaning b. München, Vol. 1/1989, p. 4-31.

/119/ Tremblay, J.-P.; Sorenson, P.G.: The Theory and Practice of Compiler Writing. McGraw-Hill Book Company, New York (1987).

/120/ Ulbers, G.: A sensor for dimensional metrology with an interferometer using integrated optics technology. Measurement,Vol.9, No.1, Jan.-Mar.1991, p. 13-16.

/121/ Van Aken, L.; et al. : Simplification of A Robot Task Specification by Incorporating a Structural Geometric Database. in: CAD-Based Programming for Sensory Robots. NATO ASI Series F, Vol. 50, Springer-Verlag, New York (1988).

/122/ Van Dam, A.: PHIGS+ Functional Description Revision 3.0, ACM Computer Graphics, 22(3), p. 125-218, (1988).

/123/ Van D. Lans, F.; Das SQL Lehrbuch. Addison-Wesley, (1988).

/124/ N.N.: Datentechnische Beschreibung der Merkmaldaten im DIN V 4001 TS Format. Arbeitspapier, VDA, 20. November (1989).

/125/ Weck, M.; Weeks, J.K.: Aufgabenorientierte Roboterprogrammierung für die flexibel automatisierte Montage Industrieanzeiger 13(1989).

/126/ Wedekind, H.: Datenbanksysteme I (II). BI Wissenschaftsverlag, Reihe Informatik, Mannheim (1981).

/127/ Wells, G.C.: An evaluation of XENIX System V as a real-time operating system. Microprozessing and Microprogramming, 33 (1991), North-Holland, p. 57-66.

/128/ Weule, H.; et al.: Integration von Aktionsplanung und Konfigurierung, Forschungsbericht Universität Karlsruhe, Karlsruhe (1990).

/129/ Winston, P.H.: Artificial Intelligence. Addison-Wesley, Reading; Menlo Park (1984).

/130/ Wolf, S.; Setzer, R.: Wissensverarbeitung mit KEE. Oldenbourg Verlag, München (1991).

/131/ Zehnder, C.A.: Informationssysteme und Datenbanken. B.G. Teubner, Stuttgart (1987).

/132/ Zhang, C.-X.: Vogt, A., Mlynski, D.A.: Neuronale Plazierungsalgorithmen. Elektronik, 15/91, S. 68-72, (1991).

Anhang

Abkürzungen und Begriffe

APT	=	Automatically Programmed Tool
ASCII	=	American Standard Code of Information Interchange
AV	=	Arbeitsvorbereitung
bgi-Graph	=	bewerteter, gerichteter, initialisierter Graph. Interne Struktur aus Subaktionen, in die ein Fertigungsbefehl expandiert wird.
B-Rep	=	Boundary-Represantation; Geometrierepräsentation durch begrenzende Geometrieelemente
BV	=	Bildverarbeitung
CIM	=	Computer Integrated Manufacturing
CSG	=	Constructive Solid Geometry
CSO	=	Configuration Space Obstacle. Suchraum-Hindernis
DNC	=	Direkt Numerical Control
ESQL	=	Embedded Structured Query Language
EXAPT	=	Extended Subset of APT
FFBP	=	Neuronales Feed-Forward-Back-Propagation-Netzwerk
FFZ	=	Flexibel automatisierte Fertigungszelle
FMZ	=	Flexibel automatisierte Montagezelle
H-Graph	=	Hierarchischer Graph zur Repräsentation des rechner-internen Aktionsplanes. Besteht aus bgi-Graphen und wird dynamisch zur Laufzeit angelegt.
HHS	=	Handhabungssystem
IGES	=	Initial Graphics Exchange Specification
Implosions-modell	=	CAD-Volumenmodell des zu fertigenden Produktes

		oder der zu fertigenden Baugruppe im zusammengebauten Zustand der Baugruppen und Einzelteile
INGRES™	=	Relationales Datenbankmanagementsystem, Ingres Inc.
IRDATA	=	Industrial Robot Data, VDI-Norm 2863, Norm für Industrieroboterspache
IRL	=	Industrial Robot Language, höhere, strukturierte Programmiersprache für Roboter(-zellen)
ISO	=	International Organisation of Standardization
KEE™	=	Knowledge Engineering Environment
KI	=	Kernel Interface, Routinen-Paket des CAD-Modellierer-Kernels
KI	=	Künstliche Intelligenz
K-Raum	=	Suchraum des Bahnplaners, Synonym: C-Space
MAP	=	Manufacturing Automation Protocol
MMS	=	Manufacturing Message Specification
NN	=	Neuronales Netzwerk
OSI	=	Open System Interconnection
SIRL™	=	Firmenspez. Robotersprache, basierend auf IRL; Siemens AG
SPS	=	Speicherprogrammierbare Steuerung
SRCL	=	Siemens Robot Control Language, Assemblerartige Roboterprogrammiersprache
STEP	=	Standard for the Exchange of Product Model Data
STEP5	=	Programmiersprache für SPS (Fa. Siemens)
TCP	=	Transmission Control Protocol
TCP	=	Tool-Center-Point des HHS
TSS	=	Total Sum of Squares
UNIX™	=	Betriebssystem, geschütztes Warenzeichen von AT&T
VDA-FS	=	VDA-Flächenschnittstelle
w_{ij}	=	Synaptisches Gewicht w_{ij} von einem Neuron j zu einem Neuron i
WOP	=	Werkstattorientierte Programmierung, Möglichkeit der Geräteprogrammierung, im Gegensatz dazu: Programmerstellung innerhalb der AV
XPS	=	Expertensystem
XPS-Shell	=	noch nicht einsatzfähiges XPS, ohne "Wissens -/ Datenfüllung"
XPS-Tool	=	Werkzeug zur Erstellung von XPS

Lebenslauf

Hubert Reinisch
geb. am 30. September 1959 in Erlangen

1965 – 1971	Grundschule in Erlangen
1971 – 1975	techn.–math. Realschule in Erlangen
1975 – 1978	Lehrzeit als Werkzeugmacher bei Fa. Siemens AG, Unternehmensbereich Medizinische Technik, 28.7.1978 Facharbeiterbrief zum Werkzeugmacher
1978 – 1980	Staatl. Berufsoberschule in Nürnberg, Richtung: Technik und Gewerbe, Abschluß 4.7.1980,
1980 – 1987	Studium Maschinenwesen an der Technischen Universität München, Fachrichtung allgemeiner Maschinenbau, Abschluß 13.3.1987 : Dipl.–Ing. (Univ.)
1987 – 1992	wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik (Prof. Dr.–Ing. K. Feldmann) der Friedrich–Alexander–Universität Erlangen–Nürnberg
ab 1991	Oberingenieur der Gruppe "CAD" am gleichen Lehrstuhl

Reihe Fertigungstechnik Erlangen

Band 1
Andreas Hemberger
Innovationspotentiale in der rechnerintegrierten Produktion durch wissensbasierte Systeme
208 Seiten, 107 Bilder. 1988. Kartoniert.

Band 2
Detlef Classe
Beitrag zur Steigerung der Flexibilität automatisierter Montagesysteme durch Sensorintegration und erweiterte Steuerungskonzepte
194 Seiten, 70 Bilder. 1988. Kartoniert.

Band 3
Friedrich-Wilhelm Nolting
Projektierung von Montagesystemen
201 Seiten, 107 Bilder, 1 Tabelle. 1989.
Kartoniert.

Band 4
Karsten Schlüter
Nutzungsgradsteigerung von Montagesystemen durch den Einsatz der Simulationstechnik
177 Seiten, 97 Bilder. 1989. Kartoniert.

Band 5
Shir-Kuan Lin
Aufbau von Modellen zur Lageregelung von Industrierobotern
168 Seiten, 46 Bilder. 1989. Kartoniert.

Band 6
Rudolf Nuss
Untersuchungen zur Bearbeitungsqualität im Fertigungssystem Laserstrahlschneiden
206 Seiten, 115 Bilder, 6 Tabellen. 1989. Kartoniert.

Band 7
Wolfgang Scholz
Modell zur datenbankgestützten Planung automatisierter Montageanlagen
194 Seiten, 89 Bilder. 1989. Kartoniert.

Band 8
Hans-Jürgen Wißmeier
Beitrag zur Beurteilung des Bruchverhaltens von Hartmetall-Fließpreßmatrizen
179 Seiten, 99 Bilder, 9 Tabellen. 1989. Kartoniert.

Band 9
Rainer Eisele
Konzeption und Wirtschaftlichkeit von Planungssystemen in der Produktion
183 Seiten, 86 Bilder. 1990. Kartoniert.

Band 10
Rolf Pfeiffer
Technologisch orientierte Montageplanung am Beispiel der Schraubtechnik
216 Seiten, 102 Bilder, 16 Tabellen. 1990. Kartoniert.

Band 11
Herbert Fischer
Verteilte Planungssysteme zur Flexibilitätssteigerung der rechnerintegrierten Teilefertigung
201 Seiten, 82 Bilder. 1990. Kartoniert.

Band 12
Gerhard Kleineidam
CAD/CAP : Rechnergestützte Montagefeinplanung
203 Seiten, 107 Bilder. 1990. Kartoniert.

Band 13
Frank Vollertsen
Pulvermetallurgische Verarbeitung eines übereutektoiden verschleißfesten Stahls
XIII + 217 Seiten, 67 Bilder, 34 Tabellen. 1990. Kartoniert.

Band 14
Stephan Biermann
Untersuchungen zur Anlagen- und Prozeßdiagnostik für das Schneiden mit CO_2 - Hochleistungslasern
VIII + 170 Seiten, 93 Bilder, 4 Tabellen. 1991. Kartoniert.

Band 15
Uwe Geißler
Material- und Datenfluß in einer flexiblen Blechbearbeitungszelle
124 Seiten, 41 Bilder, 7 Tabellen. 1991. Kartoniert.

Band 16
Frank Oswald Hake
Entwicklung eines rechnergestützten Diagnosesystems für automatisierte Montagezellen
XIV + 166 Seiten, 77 Bilder. 1991. Kartoniert.

Band 17
Herbert Reichel
Optimierung der Werkzeugbereitstellung durch rechnergestützte Arbeitsfolgenbestimmung
198 Seiten, 73 Bilder, 2 Tabellen. 1991. Kartoniert.

Band 18
Josef Scheller
Modellierung und Einsatz von Softwaresystemen für rechnergeführte Montagezellen
198 Seiten, 65 Bilder. 1991. Kartoniert.

Band 19
Arnold vom Ende
Untersuchungen zum Biegeumformen mit elastischer Matrize
166 Seiten, 55 Bilder, 13 Tabellen. 1991. Kartoniert.

Band 20
Joachim Schmid
Beitrag zum automatisierten Bearbeiten von Keramikguß mit Industrierobotern
XIV + 176 Seiten, 111 Bilder, 6 Tabellen. 1991. Kartoniert.

Band 21
Egon Sommer
Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen
188 Seiten, 102 Bilder. 1991. Kartoniert.

Band 22
Georg Geyer
Entwicklung problemspezifischer Verfahrensketten in der Montage
192 Seiten, 112 Bilder. 1991. Kartoniert.

Band 23
Rainer Flohr
Beitrag zur optimalen Verbindungstechnik in der Oberflächenmontage (SMT)
186 Seiten, 79 Bilder. 1991. Kartoniert.

Band 24
Alfons Rief
Untersuchungen zur Verfahrensfolge Laserstrahlschneiden und -schweißen in der Rohkarosseriefertigung
VI + 145 Seiten, 58 Bilder, 5 Tabellen. 1991. Kartoniert.

Band 25
Christoph Thim
Rechnerunterstützte Optimierung von Materialflußstrukturen in der Elektronikmontage durch Simulation
188 Seiten, 74 Bilder. 1992. Kartoniert.

Band 26
Roland Müller
CO_2-Laserstrahlschneiden von kurzglasverstärkten Verbundwerkstoffen
141 Seiten, 107 Bilder, 4 Tabellen. 1992. Kartoniert.

Band 27
Günther Schäfer
Integrierte Informationsverarbeitung bei der Montageplanung
195 Seiten, 76 Bilder. 1992. Kartoniert.

Band 28
Martin Hoffmann
Entwicklung einer CAD/CAM-Prozeßkette für die Herstellung von Blechbiegeteilen
149 Seiten, 89 Bilder. 1992. Kartoniert.

Band 29
Peter Hoffmann
Verfahrensfolge Laserstrahlschneiden und -schweißen : Prozeßführung und Systemtechnik in der 3D-Laserstrahlbearbeitung von Blechformteilen
186 Seiten, 92 Bilder, 10 Tabellen. 1992. Kartoniert.

Band 30
Olaf Schrödel
Flexible Werkstattsteuerung mit objektorientierten Softwarestrukturen
180 Seiten, 84 Bilder. 1992. Kartoniert.

Band 31
Hubert Reinisch
Planungs- und Steuerungswerkzeuge zur impliziten Geräteprogrammierung in Roboterzellen
XI + 212 Seiten, 112 Bilder. 1992. Kartoniert.