KfK 5052
Mai 1992

Grafische Realzeitunterstützung für Fernhandhabungsvorgänge in komplexen Arbeitsumgebungen im Rahmen eines Systems zur Steuerung, Simulation und Off-Line-Programmierung

U. Kühnapfel
Institut für Reaktorentwicklung

Kernforschungszentrum Karlsruhe

Kernforschungszentrum Karlsruhe

Institut für Reaktorentwicklung

KfK 5052

Grafische Realzeitunterstützung für Fernhandhabungsvorgänge in komplexen Arbeitsumgebungen im Rahmen eines Systems zur Steuerung, Simulation und Off-Line-Programmierung

U. Kühnapfel

Als Dissertation genehmigt von der Fakultät für Maschinenbau der Universität Karlsruhe (TH)

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Als Manuskript gedruckt

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

Zusammenfassung

Grafische Realzeitunterstützung für Fernhandhabungsvorgänge in komplexen Arbeitsumgebungen im Rahmen eines Systems zur Steuerung, Simulation und Off-Line-Programmierung

Uwe Kühnapfel

Ein Hauptproblem in der Fernhantierungstechnik besteht darin, den Operateur mit einem leistungsfähigen Sicht- und Kontrollsystem zu unterstützen. Der vorliegenden Arbeit liegt als Basisgedanke der Einsatz von Echtzeit-Computergrafik, CAD-Methoden und Simulationstechniken zugrunde. Bei der Durchführung von fernbedienten Inspektions-, Wartungs- und Reparaturarbeiten (Teleoperation) orientiert sich der Operateur anhand von synthetisch erzeugten, animierten 3D-Darstellungen des Arbeitsgeräts (Roboter, Manipulatoren, Werkzeuge) und der Umgebung im Arbeitsbereich.

Derartige Anwendungen erfordern vom grafischen Simulationssystem hohe Geschwindigkeit bei gleichzeitiger Realitätstreue. Neben dieser Echtzeitunterstützung der Telemanipulation, auch synthetisches Sehen genannt, eignet sich der Simulator auch zur Aufgabenplanung, zum Training des Bedienpersonals und zur Automatisierung von Teilaufgaben durch Off-Line-Programmierung. Das neuentwickelte Verfahren wurde bei JET zur Bedienunterstützung für den Großmanipulator "Articulated Boom" mit dem anwendungsspezifischen Simulationswerkzeug GBSIM in die Praxis umgesetzt.

Das ebenfalls im Rahmen der Arbeit realisierte und universell einsetzbare Simulationssystem KISMET unterscheidet sich von bisherigen Ansätzen durch eine hierarchisch gegliederte Datenbasis zur Definition der Anlagengeometrie und -topologie, sowie der Gerätekinematik. Sie erlaubt die Darstellung in unterschiedlichen Detailstufen und eignet sich daher besonders zur Echtzeit-Darstellung von sehr komplexen Arbeitsszenarien. KISMET erlaubt die Modellierung der Baugruppenstruktur in einer beliebigen Anzahl von Detaillierungsstufen. Der integrierte Modellierer gestattet die interaktive Definition der Zellengeometrie, der Baugruppen-Topologie und von Mechanismen. Kinematische Strukturen werden als Starrkörpersystem betrachtet. Die Mechanismen können offene Ketten mit beliebig vielen Verzweigungen und Gelenken, ebene geschlossene Ketten, sowie Kopplungen zwischen den Gelenken enthalten.

Behandelt werden neuartige Verfahren, Datenmodelle und Algorithmen an den Beispielen der Kollisionserkennung, der Ausblendung von verdeckten Flächen und Kanten (Hidden-Surface-Problem), der verschiedenen grafischen Darstellungstechniken und der Simulation und Steuerung von Szenenkameras (Positionierung und Nachführung). Besondere Aufmerksamkeit wird dabei der Echtzeit-Problematik geschenkt. Am Beispiel dieser Systemrealisierung im UNIX-Umfeld werden geometrische und kinematische Modelliertechniken zur optimalen Ausnützung der Grafikfähigkeiten moderner Hochleistungs-Grafikstationen aufgezeigt.
Weiterhin wird die Nutzung von standardisierten Schnittstellen zum Austausch von CAD-Modellen (CAD*I, STEP) und zur Roboterprogrammierung (IRDATA) behandelt.

Die entwickelten Methoden werden an praktischen KISMET-Applikationen aus der Handhabungstechnik in Fusionsreaktoren, der Wiederaufarbeitung von Brennelementen und der sensorgeführten Robotik erläutert.

Abstract

Realtime Graphics Support for Remote Handling Operations in Complex Working Environments within the Framework of a Control, Simulation and Off-Line Programming System

Uwe Kühnapfel

One of the major problems in remote handling technology is the question of supporting the operator with a powerful vision and control system. The thesis in hand is based on the idea of using real-time computer graphics, CAD methods, and simulation techniques. During the execution of remotely controlled maintenance, inspection and repair tasks (teleoperation), the operator orients himself by means of synthetically created, animated 3D-views of the equipment (robots, manipulators, tools), and of the actual working environment.

For this type of applications the graphical scene presentation has to be as fast and realistic as possible. In addition to the real-time support of telemanipulation, also known as synthetic viewing, the simulator can be used for task planning, training operators, and for Off-Line robot programming of subtasks to achieve a high level of automation. The novel simulation technique was put into practice at JET. During control of the "Articulated Boom" transporter, the operator is supported by the application specific simulation tool GBSim.

The application independent simulation system KISMET was also developed within the framework of this thesis. This tool gives a different approach compared to previously existing robot simulators. A hierarchical data structure approach is used for the definition of workcell geometry, assembly topology and mechanism kinematics. This database structure allows for presentation of interactively selectable levels of detail and is, therefore, especially useful for real-time rigid body simulation of complex RH-scenarios. With KISMET, assembly structures can be modelled in any number of detail levels. Workcell geometry, assembly topology and mechanisms can be defined interactively by means of the integrated modeller. The mechanism simulation allows for kinematical tree structures with any number of joints, planar closed chains, and interconnections between joints.

Examples of novel simulation methods, data structures, and algorithms are presented for selected examples: the hidden surface problem, graphical presentation techniques, collision testing, and control of scene cameras (image simulation, fast positioning and tracking). Special attention is paid to the real-time problem. The way this system was realized within the UNIX world is shown as an example for geometric and kinematic modelling techniques that grant for the optimum use of the capabilities of high-performance graphics workstations. A further chapter is focussing on the use of standard interfaces for CAD model transfer (*CAD*I*, STEP) and robot programming (IRDATA).

Examples of practical KISMET applications for remote handling in fusion reactors, in a nuclear fuel element reprocessing cell and in sensor based robotics are used to present the developed methods.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Formelzeichen und Abkürzungen

Formelzeichen

$\mathbf{n}$, $\mathbf{a}_4$, $\mathbf{p}_{Base}$	*Vektoren* werden ausgedrückt durch Kleinbuchstaben in gerader Fettschrift,
$\mathbf{A}$, $\mathbf{M}_{23}$, $\mathbf{T}_{TCP}$	zur Beschreibung von *Matrizen* werden Großbuchstaben in Fettschrift benutzt,
x_1 , p_y , u	schräggestellte Normalschrift wird in Formeln und Ausdrücken für *Variable* und für *Parameter* benutzt,
$\sin(x)$, $\mathrm{atan}(n\pi)$	*Funktionen* sind gewöhnlich in Normalschrift gesetzt,
$\mathbf{n}_1 \bullet \mathbf{a}_2$	*Skalarprodukt* zweier Vektoren,
$\mathbf{a}_2 \times \mathbf{s}_1$	*Kreuzprodukt* zweier Vektoren,
$\lvert\mathbf{n}\rvert$, $\lvert(\mathbf{a}_1 \times \mathbf{b}_2)\rvert$	bezeichnet den *Betrag* eines Vektors bzw. des Resultats einer Vektoroperation,
$\mathbf{A}^t$, $\mathbf{p}^t$	bedeuten die *Transponierte* einer Matrix respektive eines Vektors,
$\mathbf{A}^{-1}$	wird verwendet für die *Inverse* einer allgemeinen Matrix,
$\mathbf{n} \parallel \mathbf{s}$, $\mathbf{a} \perp \mathbf{b}$	die Vektoren sind *parallel* bzw. stehen *senkrecht* zueinander,
$\dot{s}$, $\ddot{s}$	die *erste* bzw. die *zweite Ableitung* nach der Zeit der Funktion 's',
$\overline{\mathbf{x}_1\,\mathbf{x}_2}$	kennzeichnet die *Verbindungsstrecke* zwischen den Koordinaten $\mathbf{x}_1$ und $\mathbf{x}_2$,

Abkürzungen

B-Rep	**B**oundary **Rep**resentation
CAD	**C**omputer **A**ided **D**esign (= Rechnergestützte Konstruktion)
CAM	**C**omputer **A**ided **M**anufacturing (= Rechnergestützte Produktion)
CAT	**C**omputer **A**ided **T**eleoperation (= Rechnergestützte Teleoperation, Fernhandhabung)
CAR	**C**omputer **A**ided **R**obotics (= Rechnerunterstützter Robotereinsatz)
CIM	**C**omputer **I**ntegrated **M**anufacturing (= Computer Integrierte Fertigung)
CP	**C**ontinuous **P**ath (= Bahnsteuerungsverhalten)
CSG	**C**onstructive **S**olid **G**eometry
EMSM	**E**lektrischer **M**aster/**S**lave **M**anipulator
GBSim	**G**raphical **B**oom **Sim**ulator (Grafisches Monitoring, Simulations- und Off-Line Programmiersystem für den JET-"Articulated Boom")
HFH	**H**ochflexible **H**andhabungssysteme
IRDATA	**I**ndustrial **R**obot **Data**
IRE	**I**nstitut für **R**eaktor**e**ntwicklung
IVHU	**I**n-**V**essel **H**andling **U**nit
JET	**J**oint **E**uropean **T**orus
KI	**K**ünstliche **I**ntelligenz
KFK	**K**ernforschungszentrum **K**arlsruhe
KISMET	**K**inematic **S**imulation, **M**onitoring, and Off-Line Programming **E**nvironment for **T**elerobotics (Applikationsunabhängiges Monitor-, Simulations- und Off-Line Programmiersystem für Teleoperation)
KS	**K**oordinatensystem
NET	**N**ext **E**uropean **T**orus
PPS	**P**roduktionsplanung und -steuerung
PTP	**P**oint-**t**o-**P**oint (= Punkt-zu-Punkt Bewegungssteuerung)
TCP	**T**ool **C**enter **P**oint (= Werkzeug Bezugspunkt)
TCPF	**T**ool **C**enter **P**oint **F**rame (= Werkzeugkoordinatensystem)
WAK	**W**iederaufarbeitungsanlage **K**arlsruhe

1. Einleitung

1.1 Problemstellung

Ein Hauptproblem der **Handhabungstechnik** in unzugänglichen oder schwer einsehbaren Arbeitsumgebungen besteht darin, den Operateur mit einem **leistungsfähigen Sicht- und Kontrollsystem** auszustatten. Konventionelle Sichtkonzepte basierten in der Vergangenheit auf der direkten Einsicht in den Arbeitsbereich [1] - dies ist schon beim Entwurf der Anlage architektonisch zu berücksichtigen - oder auf dem Einsatz von Videotechnik, im letzteren Fall spricht man von **Fernhantierungstechnik**.

In Anlagen, deren physikalische und anlagentechnische Randbedingungen eine direkte, unmittelbare Einsichtnahme gestalterisch nicht erlauben oder die durch ihre kompakte Bauweise eine ausreichende Bestückung mit Videokameras nicht zulassen, werden seit etwa 1982 zunehmend Problemlösungen basierend auf **animierter Computergrafik, CAD-Methoden und Simulationstechniken** vorgeschlagen. Dadurch soll dem Operateur in jedem Arbeitsstadium die Übersicht über alle Teile des Arbeitsgeräts und der Umgebung ermöglicht werden.

Als Alternative zum Bau kostenintensiver Anlagenmodelle ("Mock-Up") schlägt *Trachsel* [3] zur Schulung von Handhabungsoperateuren für Wartungsarbeiten in Fusionsreaktoren den Einsatz grafischer Simulatoren vor, analog zu Schulung und Training von Piloten auf Flugsimulatoren. Der Aufsatz von *Titus* [4] beschreibt geometrische und dynamische Simulationsstudien für die "In-Vessel"-Handhabung des Tokamak Fusion Test Reactor (TFTR), wobei vor allem Fragen zur Erreichbarkeit von Arbeitspositionen, zur Ablaufplanung und der zu erwartenden Verfahrzeiten untersucht wurden. *Silverman* berichtet über Messungen und Testreihen zum Vergleich des direkten Sehens (direct Viewing) mit dem fernbedienten Sehen über Videokameras (remote Viewing) [2]. Er kommt zu dem Ergebnis, daß für Teleoperationsaufgaben in den meisten Fällen das Arbeiten mit Videokameras zu einer höheren Arbeitsleistung führt als das direkte Sehen. Dies setzt allerdings voraus, daß die Anlagengeometrie die Wahl der Kamera-Standorte und unbehinderte Sichtlinien gestattet, um dem Operateur für die Aufgabe günstige Sichtbedingungen zu ermöglichen. Auf die Anwendung der Computergrafik und des synthetischen Sehens als Ersatz oder Ergänzung zu Videokameras geht *Silverman* in diesem Aufsatz allerdings nicht ein.

Die Implementierung eines 2D-Vektorgrafik-Testprogramms zur Off-Line-Simulation des Austauschs von Limiterkomponenten im JET (Joint European Torus) Experimental-Fusionsreaktor mit dem Handhabungs-Vielgelenkarm "Articulated Boom" [5]-[9] wurde von *Leinemann* und *Schlechtendahl* beschrieben [10]-[12]. Dieses System wurde zur Demonstration der Möglichkeiten und des Nutzens entwickelt, die eine **grafische Mensch-Maschine Schnittstelle** für die Steuerung eines kinematisch redundanten Vielgelenkarms in einer nicht direkt einsehbaren Arbeitsumgebung, wie im Vakuumgefäß ("In-Vessel"-Bereich) einer Fusionsanlage, bieten kann.

Im fortgeschrittenen Stadium des Forschungsprogramms ist bei JET der Plasmabetrieb mit Tritium geplant. Dies führt zu einer Aktivierung der ersten Wand des Vakuumgefäßes durch schnelle Neutronen und bedingt deshalb die fernbediente Durchführung aller **vorhersehbaren Wartungsarbeiten** sowie der **unvorhersehbaren Reparaturarbeiten.**

In einschlägigen Studien werden als typische Umgebungsbedingungen Strahlungswerte von 3×10^6 rad/h γ-Strahlung, mit vernachlässigbarer α- und β-Strahlung, sowie Temperaturen von bis zu 150°C angenommen [13].

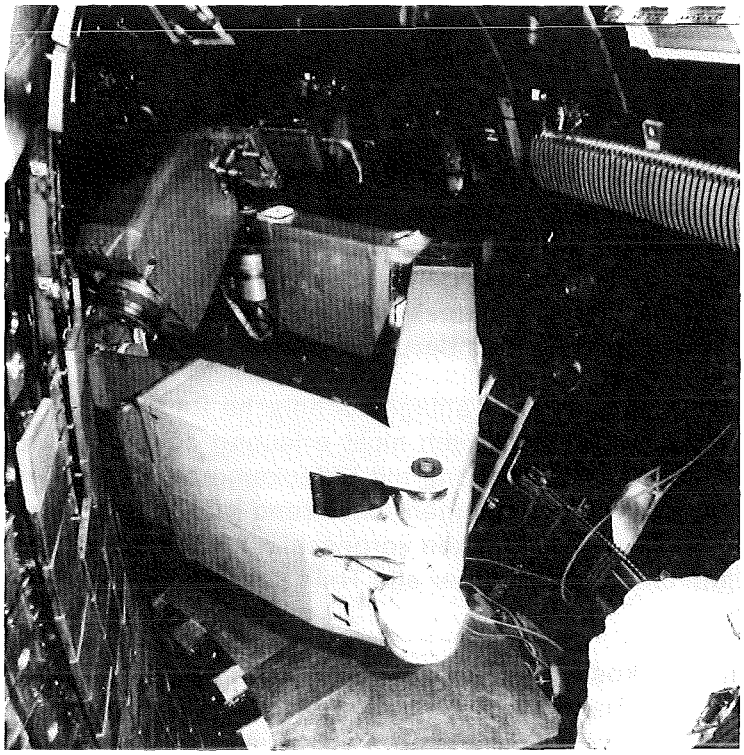


Abbildung 1. JET In-Vessel-Szenario: Foto des JET (Joint European Torus) "In-Vessel"-Handhabungsbereichs mit dem Großmanipulator und Transportersystem "Articulated Boom". (Quelle: JET)

Das bei JET ursprünglich verwendete Sichtsystem mit mehreren Videokameras ist nicht in der Lage, dem Operateur in jedem Aufgabenstadium einen ausreichenden Überblick über die augenblickliche Arbeitssituation zu verschaffen. Abbildung 1 zeigt anschaulich, daß im beengten Arbeitsraum des Vakuumgefäßes das Manövrieren des Großmanipulators ohne entsprechende Computerunterstützung zur Darstellung der Position des Arbeitsgeräts und zur **koordinierten Bewegung mehrerer Bewegungsachsen** - besonders unter dem Gesichtspunkt der **Kollisionsvermeidung** - eine Gefährdung für die Anlage und das Arbeitsgerät darstellt. Neben einer möglichen Beschädigung der Anlage können auch "sanfte" Kollisionen zu einem Materialabrieb des "Articulated Boom" an der ersten Wand führen, die dann in den nachfolgen Plasmaversuchen zu Verunreinigungen und damit zu Zusammenbrüchen des Plasmas (Disruptions), oder zumindest zu einer Beeinträchtigung der Meßreihen führen können.

Um Stillstandszeiten der Anlage gering zu halten, empfiehlt es sich außerdem, die Durchführung bestimmter Teilaufgaben zu **automatisieren**. Dazu gehört z.B. der Transport eines End-Effektors in eine vorgegebene Arbeitsposition oder der Abtransport von ausgebauten Großkomponenten.
Der Aspekt der Automatisierung erfordert deshalb **Programmierschnittstellen** für die eingesetzten Transport- und Arbeitsmanipulatoren [14].

Die Kombination von Methoden der industriellen Roboter- und Computertechnik mit der klassischen, bereits seit den 40er Jahren eingesetzten Master-Slave-Manipulatortechnik zu einem **hochflexiblen Handhabungssystem** wird auch mit dem Begriff der **Computer Aided Telemanipulation** (CAT) umschrieben. Insbesondere ist diese Arbeitsweise durch einen **häufigen Wechsel** zwischen **Automatikbetrieb** und **manueller Bedienung** gekennzeichnet [15].

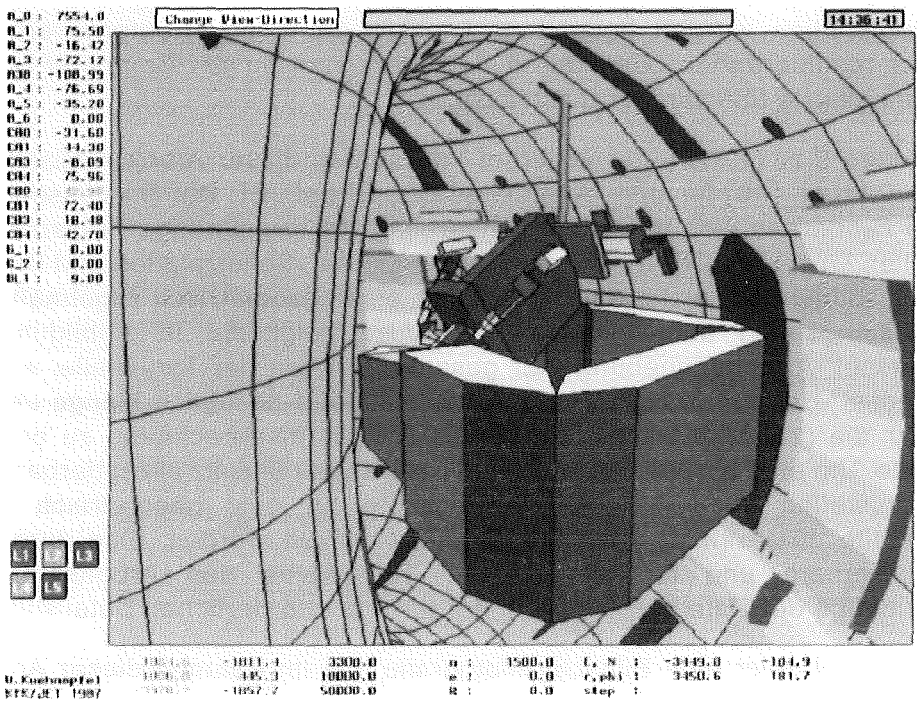


Abbildung 2. Simulationsmodell der JET In-Vessel-Arbeitsumgebung: Eine typische Darstellung des In-Vessel-Szenario wie sie dem Handhabungsoperateur durch das Simulations- und Monitor-System GBSim präsentiert wird.

Bei den vorgenannten Arbeiten handelt es sich um **Vorschläge** oder allenfalls um **Off-Line-Simulationsstudien** zum Einsatz computergrafischer Lösungen für die **Planung von Teleoperationsaufgaben**, wobei jedoch schon deutlich die zu erwartenden Vorteile und Einsatzfelder herausgearbeitet wurden.

Eine integrierte grafische Software-Umgebung eignet sich daher besonders für:

- Die **Schulung** und das **Training** des Bedienpersonals für den manuellen Betrieb,
- die **Planung** und **Off-Line-Programmierung** von Handhabungsabläufen mit den daraus resultierenden Vorteilen einer Reduzierung von Anlagen-Stillstandszeiten und der Kosten für Anlagenmodelle,
- die **grafische Simulation** von **Roboterprogrammen** durch Emulation der in der realen Steuerung ablaufenden Bewegungsalgorithmen,
- die **On-Line-Unterstützung** während der Ausführung der Teleoperationsaufgabe durch ein aufgabengerechtes, **grafisches Sichtsystem** (Monitoring) zur Ergänzung und zum teilweisen Ersatz von Videokameras zur Überwachung automatisch ablaufender Roboterprogramme und für die manuelle Steuerung der Arbeitsgeräte,
- die Unterstützung der Teleoperation durch leistungsfähige Algorithmen zur **Kollisionsvermeidung** und zur **Führung** der Manipulatoren **in Werkzeugkoordinaten** (Resolved-Motion) statt in Gelenkkoordinaten,
- die manuelle **Positionierung**, oder optional, die **automatische Nachführung** der stationären Umgebungskameras oder der kinematisch mit dem Arbeitsgerät verketteten Arbeitskameras.

Eine praktische Realisierung und Erprobung der vorgeschlagenen Konzepte wurde jedoch erstmals im Rahmen der hier dokumentierten Arbeit ausgeführt. In den angeführten Konzeptstudien wurden besonders Aspekte der Echtzeit-Datenverarbeitung, und der daraus resultierenden besonderen Anforderungen an die Computer-Hardware und die implementierte Software, vernachlässigt.

Abbildung 2 zeigt die unter anderem im Rahmen dieser Arbeit realisierte grafische Bedienerführung [16] für den "Articulated Boom" bei JET. Der "In-Vessel"- Arbeitsbereich, dabei handelt es sich um den Innenbereich des torusförmigen Plasma-Vakuumgefäßes mit allen Einbauten (HF-Antennen, Plasma-Limiter, Schutzplatten und Divertoren) und die Zugangsöffnungen, weist die Eigenschaft einer **geometrisch regelmäßig strukturierten Arbeitsumgebung** auf. Im Gegensatz dazu zeigt die in Abbildung 3 dargestellte "Ex-Vessel"-Arbeitsumgebung (der gesamte Bereich der Torushalle, ausgenommen der "In-Vessel"-Bereich) das Problem der **Komplexität** bezüglich **Menge** und **Formenvielfalt** der für die grafische Darstellung und die Hindernisberechnung zu berücksichtigenden Objekte. Um eine für den Operateur im praktischen Einsatz akzeptierbare Systemleistung bereitzustellen, mußten spezielle Konzepte für eine **geometrisch** und **topologisch** abgestufte, **hierarchische Datenstruktur** entwickelt werden. Eine durch den Operateur spezifizierbare und für die aktuelle Teilaufgabe des Arbeitsablaufs notwendige Untermenge der Datenbasis wird in der jeweils erforderlichen Detailstufe dargestellt.

Abbildung 4 zeigt eine Bildschirmfotografie des im Rahmen dieser Arbeit entwickelten grafischen Simulations-, Monitor- und Roboter-Programmiersystems **KISMET** mit der JET Ex-Vessel-Arbeitsumgebung und dem Großmanipulator "Telescopic Articulated Remote Manipulator" (TARM) [17].

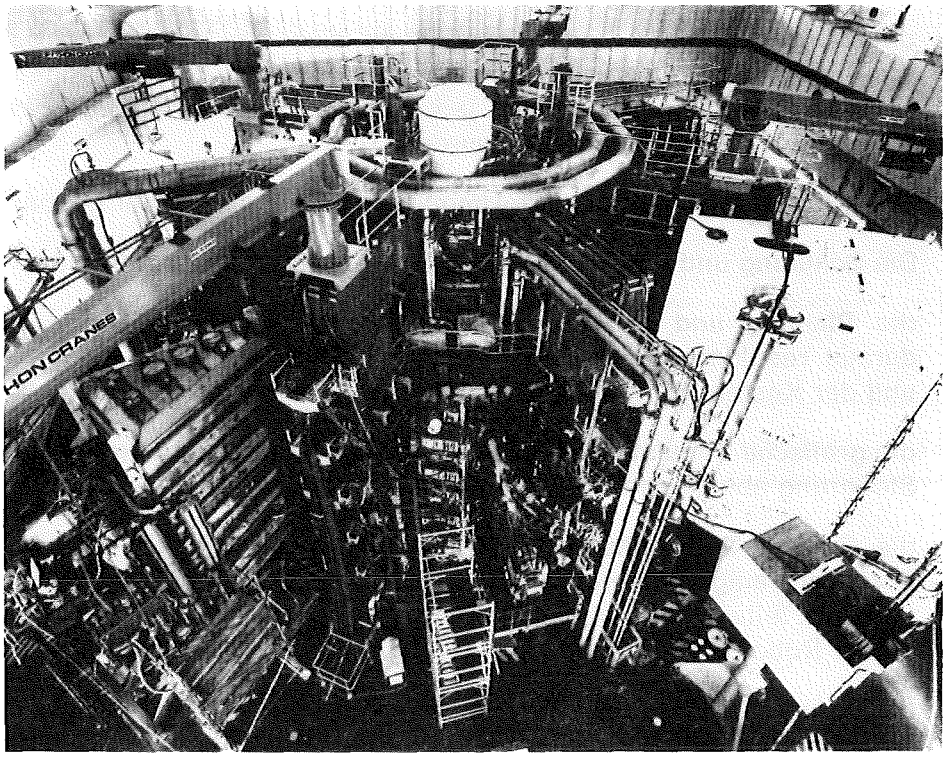


Abbildung 3. JET Ex-Vessel-Szenario: Foto der JET "Ex-Vessel"-Arbeitsumgebung mit dem Tokamak-Versuchsreaktor in der Torushalle. Die Fotografie verdeutlicht die ungeheure Komplexität der Umgebungsgeometrie und die Notwendigkeit zur Entwicklung von geeigneten Konzepten für die 3D-grafische Echtzeitsimulation derartiger Anlagen. (Quelle: JET)

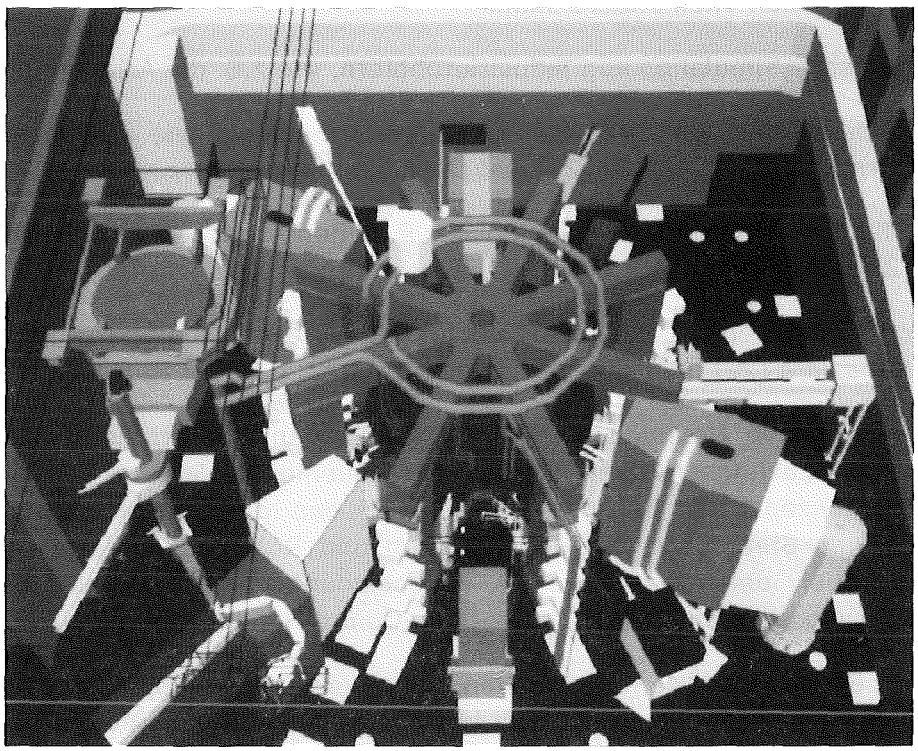


Abbildung 4. Simulationsmodell der JET Ex-Vessel-Arbeitsumgebung: Eine typische Darstellung des "Ex-Vessel"-Szenarios wie sie dem Handhabungsoperateur durch das Simulations- und Monitor-System KISMET präsentiert wird.

Die Anwendung der im Rahmen dieser Arbeit entwickelten Konzepte und Computerprogramme läßt sich auf handhabungstechnische Applikationen im Bereich der Raumfahrt, der Unter- und Überwasser-Fernhandhabung sowie der industriellen Fertigung ausdehnen.

1.2 Zielsetzung und Inhalt der Arbeit

Zielsetzung der Arbeit war die Entwicklung von Verfahren, Datenmodellen und Algorithmen zur grafischen **On- und Off-Line-Simulation von Handhabungsgeräten.** Der Arbeit liegt als Basisgedanke der Einsatz von Computergrafik, CAD-Methoden und Simulationstechniken zur On-Line Steuerung von Manipulatoren und Roboterähnlichen Handhabungsgeräten für die Teleoperation zugrunde, wie z.B. in den Aufsätzen von *Leinemann* und *Schlechtendahl* [10]-[12] vorgeschlagen wurde.

Grundsätzliche Einschränkungen bezüglich der Anzahl der gleichzeitig zu simulierenden Roboter bzw. Mechanismen, der Geometrie von Bauteilen, der kinematischen Topologie der Mechanismen sollen nicht bestehen. Anforderungen aus der Praxis erfordern die Entwicklung von Konzepten zur effizienten Strukturierung der programminternen und Definitions-Datenmodelle, um die realitätsgetreue, sensorgestützte Simulation von Teil-Arbeitsvorgängen wie z.B. Werkzeugwechsel, das Greifen und Hantieren von Objekten und die daraus resultierenden Änderungen der kinematischen Topologie zu ermöglichen.

Bisher veröffentlichte, und von der vorliegenden Arbeit thematisch berührte Arbeiten, entstammen den Themenbereichen:

- grafische Simulation von Industrierobotern, wie z.B. die Arbeit von *Wloka* [18],
- Systeme und Verfahren zur Kollisionsvermeidung, wie z.B. in den Arbeiten von *Hörmann* [19], *Schütze* [20] und *Diehl* [21] behandelt,
- Off-Line-Programmiersysteme, wie z.B. in den Arbeiten von *Zühlke* [22] und *Niehaus* [23] beschrieben,
- oder aus dem Planungsbereich für industrieller Montagevorgänge, wie z.B. von *Kandziora* [24] oder *Wanner* [25] behandelt.

Das wesentliche Unterscheidungsmerkmal zu den o.a. Arbeiten und anderen bisher bekannten Lösungsansätzen ist die konsequente Auslegung des hier vorgestellten Lösungskonzepts für den **On-Line Einsatz** des entwickelten Programmsystem zur **interaktiven Steuerung** von Manipulatoren und Robotern in der Handhabungstechnik [26], wobei der Operateur als Entscheidungsträger in die Regelschleife des Teleoperationssystems ("Man-in-the-Loop-Control") einbezogen ist [27].

Diese Betriebsart der **grafischen Überwachung** (Graphical Monitoring), auch **synthetisches Sehen** genannt, stellt besondere Anforderungen an die Leistungsfähigkeit der grafischen Software, sowie aller Algorithmen zur On-Line-Unterstützung der Handhabungsoperateure. Programmfehler im Kollisionswarnsystem, oder eine Beeinträchtigung des Realzeitverhaltens durch ineffizentes Antwortverhalten - und der daraus eventuell resultierende Verlust der Kontrolle über die zu steuernden Geräte - können zu

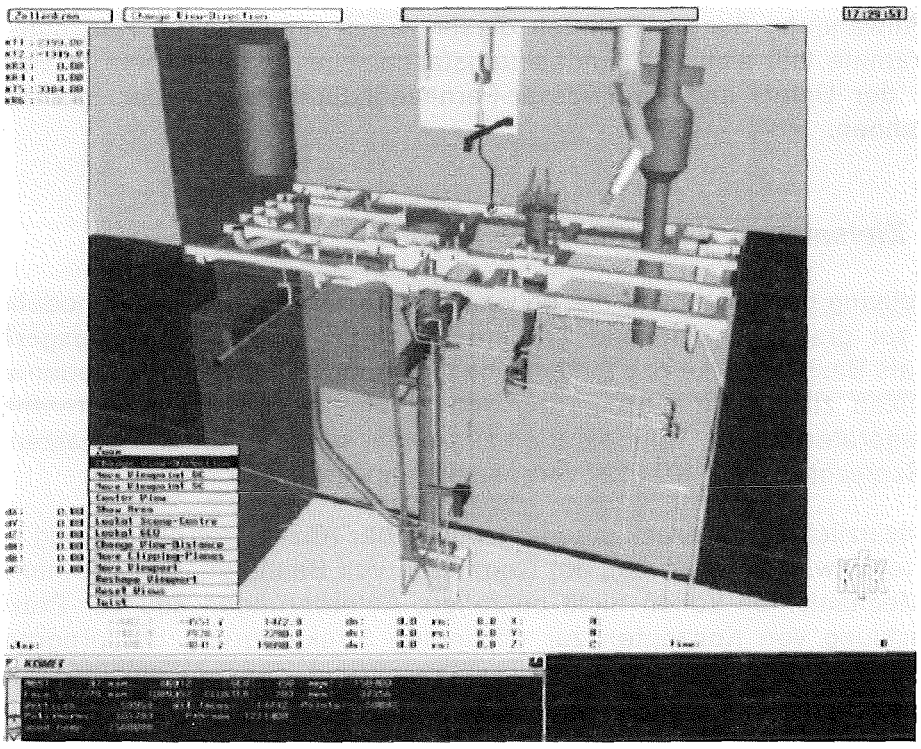


Abbildung 5. Simulation einer Heißen Zelle: Grafische Echtzeitsimulation des Ausbaus einer Rohrleitung aus einer heißen Zelle in der Wiederaufarbeitungsanlage Karlsruhe (WAK).

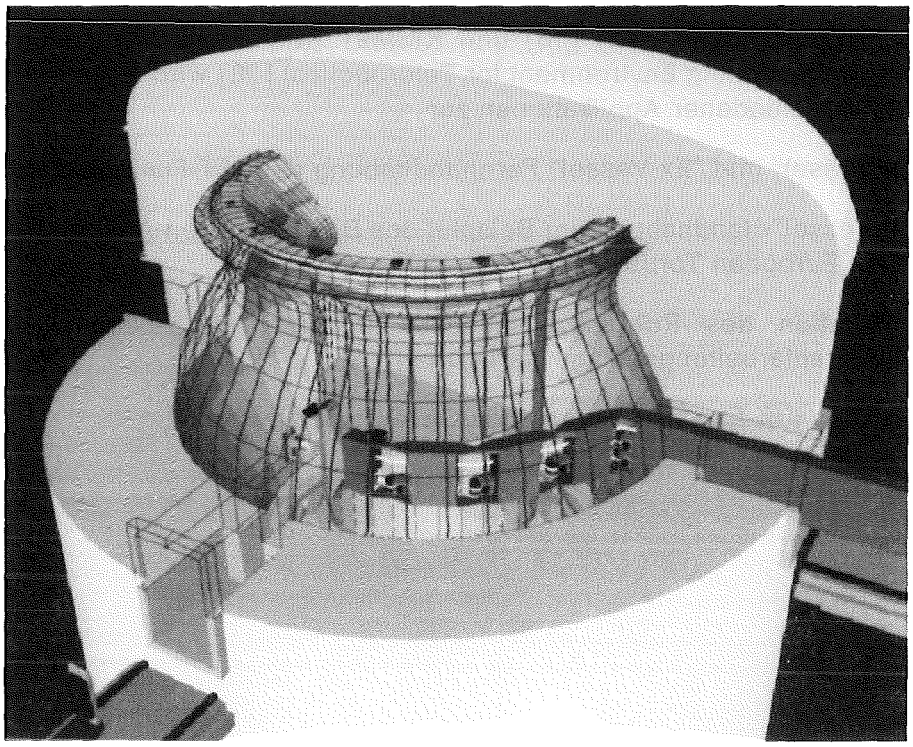


Abbildung 6. Simulationsmodell des NET In-Vessel-Szenarios: Schnitt-Darstellung des NET In-Vessel Arbeitsumgebung Simulationsmodells mit Plasmatorus, Divertorplatten, "In-Vessel Handling Unit" (IVHU), "Divertor Handling Unit" (DHU).

folgenschweren Beschädigungen des Handhabungsgeräts oder der bearbeiteten Anlagenteile führen.

Das benutzte **hierarchische Datenmodell**, sowie die adaptierten bzw. neuentwickelten Algorithmen wurden daher hauptsächlich aufgrund der praktischen Verwendbarkeit unter Echtzeitbedingungen ausgewählt.

Die Arbeit liefert daher einen Beitrag zu folgenden Teilproblemen:

- Geometrische- und kinematische Modellbildung für das neuartige synthetische, CAD-Modellgestützte 3D-Sehen im Rahmen von *"On-Line-Monitoring"*-Applikationen
- Ein neuartiger Algorithmus zur Ausblendung verdeckter Flächen und Kanten (Hidden-Surface Removal) für preisgünstige Grafiksysteme ohne Bild-Tiefenspeicher (Z-Buffer). Das beschriebene echtzeitfähige Verfahren eignet sich im Gegensatz zu anderen Methoden auch zur Anwendung auf bewegte Modelle
- Ein neuartiges, echtzeitfähiges 3D-Kollisionswarnsystem, das durch die Benutzung eines hierarchischen Datenmodells und im Gegensatz zu anderen Methoden auch für sehr komplexe Arbeitsumgebungen und zur Erfassung von Kollisionen zwischen mehreren Robotern geeignet ist und dennoch geometrisch genau arbeitet.
- Steuer- und Bewegungsmodi für kinematisch redundante Vielgelenkmanipulatoren
- Steuerung und Nachführung von Videokamera-Systemen
- Konsequente Auswahl eines geeigneten Konzepts zur Erzeugung und Simulation von Roboterprogrammen (Off-Line-Programmierung) für die Fernhantierungstechnik

Die praktische Erprobung der im Rahmen der Arbeit entwickelten Programme **GBSim** (**G**raphical **B**oom **Sim**ulator) [16] und **KISMET** (**Ki**nematic **S**imulation, **M**onitoring and Off-Line Programming **E**nvironment for **T**elerobotics) [28] wird erläutert und diskutiert am Beispiel verschiedener Applikationen zur

1. "In-Vessel" und "Ex-Vessel" Fernhandhabung des JET-Fusionsreaktors,

2. "In-Vessel"-Handhabung am Beispiel des Divertorplatten-Ausbaus im Fusionsreaktor "Next European Torus" (NET) [29], [30],

3. Simulation des Rohrleitungsausbaus in einer Prozeßzelle (Auflöserzelle) der Wiederaufarbeitungsanlage Karlsruhe (WAK) [31],

4. und des KfK-Experimentierstands CATROB [26], [32].

2. Stand der Technik grafischer Simulationssysteme

Generell bestehen bei der Realisierung einer grafischen Bedienerführung für handhabungstechnische Anwendungen die Möglichkeiten entweder ein bereits existierendes CAD- oder Off-Line Programmiersystem zu adaptieren, oder aber ein Softwaresystem von Grund auf neu zu spezifizieren und implementieren.

Um die Möglichkeit zum Einsatz eines bereits verfügbaren grafischen Simulators im Rahmen eines integrierten Systems zum **synthetischen Sehen** zur **Simulation** und zur **Off-Line-Programmierung** zu prüfen, wurde um 1986 eine Reihe von Produkten und Programmsystemen auf der Basis der im Literaturverzeichnis angeführten Veröffentlichungen, oder in Form von Produktpräsentationen, untersucht. Dabei lassen sich folgende Systemgruppen unterscheiden:

1. Modulare Erweiterungen von CAD-Programmen zur kinematischen Simulation von mechanischen Starrkörpersystemen
2. Entwicklungen aus dem Forschungsumfeld, teilweise unter Einbeziehung eines vorhandenen CAD-Kerns
3. Spezielle Off-Line-Programmiersysteme für Industrieroboter zur Anwendung in der industriellen Fertigungstechnik

An dieser Stelle sei noch der schon mehrfach verwendete Begriff des **synthetischen Sehens** näher definiert. Unter synthetischem Sehen versteht man die:

> *"Steuerung einer Hantierungsaufgabe gestützt auf eine oder gleichzeitig mehrere, computergrafisch und damit synthetisch erzeugte Darstellungen der Hantierungsszene unter Verwendung von Sensordaten aus dem Hantierungsprozeß und eines 3D-CAD-Modells der Hantierungszelle."*

2.1 *Anforderungen und Beurteilungskriterien*

Zum o.a. Zeitpunkt lagen noch keine Erfahrungen mit der praktischen Anwendung eines universell adaptierbaren, **realzeitfähigen** Systems zur 3D-grafischen Bedienerunterstützung im Handhabungsbereich in dokumentierter Form vor. Es bestand daher zu diesem Zeitpunkt keine Möglichkeit auf entsprechende Erfahrungsberichte zur Erstellung eines speziellen Anforderungsprofils für diese Klasse von Applikationen zurückzugreifen. Durch Gespräche mit den potentiellen Systemanwendern bei JET und eine Untersuchung des dortigen Handhabungsszenario, sowie von anderen, nicht auf dieses spezielle Anwendungsspektrum zugeschnittenen Simulationssystemen, ergab sich die nachfolgend aufgeführte Liste von Anforderungskriterien. Besonders sei darauf hingewiesen, daß alle genannten Punkte **gleichzeitig** erfüllt sein müssen.

Folgende Anforderungskriterien wurden bei der Untersuchung berücksichtigt:

- Um dem Operateur für das synthetische Sehen einen realistischen und aufgabengerechten visuellen Eindruck zu vermitteln ist die Möglichkeit zur **schattierten Darstellung** in **Echtzeit** notwendig. Echtzeitfähigkeit entspricht hier, je nach typischer Bewegungsgeschwindigkeit der Transporteinrichtungen, Roboter, Manipulatoren und Werkzeuge, einer Bildrate von 5-20 neuberechneten Szenendarstellungen je Sekunde.
  Diese Forderung basiert auf Untersuchungen, die für das neuromuskuläre System des Menschen eine Grenzfrequenz von etwa 5 Hz angeben [33]. Signalverzögerungen von mehr als 1/3 Sekunde können die geschlossene Regelschleife des

Mensch-Maschine-Systems destabilisieren. Die Autoren geben noch an, daß Verzögerungen von über 1 Sekunde eine direkte Teleoperation unmöglich macht [34], [35]. Obwohl die angeführten Untersuchungen mit konventionellen Sichtsystemen (Verwendung von Videokameras) durchgeführt wurden, lassen sich deren Ergebnisse analog auf das synthetische Sehen übertragen.
Einzelne Geometriekomponenten oder Baugruppen sollen zusätzlich optional als Drahtmodell oder transparent darstellbar sein, um die Orientierung und Übersicht des Operateurs zu erleichtern.

- Aus offensichtlichen Gründen sollen die Komponenten der Arbeitszelle geometrisch so **detailliert** wie möglich dargestellt werden. Dieser Punkt steht natürlich im Konflikt mit der vorherigen Forderung nach hohen Darstellungsraten.
  Um die Realzeitfähigkeit des Systems auch bei sehr komplexen Szenarien zu erhalten, sollten daher einzelne, für die Teilaufgabe nicht relevante **Modellkomponenten**, in der Darstellung **abschaltbar** bzw. auslagerbar sein und/oder in **unterschiedlichen Detaillierungsgraden** bzw. Präsentationsstufen darstellbar sein.

- Die zu untersuchenden Systeme sollen bzgl. der **kinematischen Modellierfähigkeit** die Definition aller typischerweise in der Handhabungstechnik und besonders der bei JET auftretenden Strukturen und Mechanismen gestatten. Hierzu gehören:

  - Offene und offen-verzweigte kinematische Ketten, sowie in der Praxis häufig im Bereich der Gelenkantriebe anzutreffende ebene geschlossene Ketten.
  - Über die Robotersteuerung oder mechanische Getriebe erwirkte kinematische Kopplungen zwischen einzelnen Gelenken.
  - Kaskadierte Robotersysteme bestehend z.B. aus Portalkran, Transporteinrichtung, Servomanipulator mit zwei 7-achsigen Armen, sowie beweglichen Kamera-Armen (siehe JET-TARM).
  - Kinematisch redundante Systeme.
  - Hilfseinrichtungen wie bewegliche Kranhaken und -seile (wichtig für Kollisionserkennung), Greif- und Trennwerkzeuge.
  - Die Anzahl der Gelenke/Freiheitsgrade, von möglichen kinematischen Verzweigungen, sowie von parallel arbeitenden Robotern/Mechanismen soll grundsätzlich keinen Beschränkungen unterworfen sein.

- Die Topologie der Modelldatenbasis bezüglich Kinematik und Baugruppenstruktur muß interaktiv veränderbar sein, um Vorgänge wie z.B. Werkzeugwechsel, oder das Greifen und Ablegen der hantierten Objekte darstellen zu können.

- Das System sollte in der Lage sein, den Bediener bei der **On-Line-Steuerung von Videokameras** durch geeignete Algorithmen zu unterstützen. Dazu gehören :

  - die automatische Kamera-Nachführung (Zielverfolgung),
  - das Anrichten des 3D-Zielpunkts durch Spezifikation im grafischen Anlagenmodell in mehreren Szenenansichten,
  - die automatische Berechnung der Schwenk-/Neigekopf-, Bildöffnungs- und Fokussierparameter,
  - die nachfolgende Übertragung der berechneten Parameter in die Kamerasteuerung.

  Während der Planungs- und Off-Line-Programmierphase sollte die Simulation der zu erwartenden Kamera-Ansichten (Videobild-Simulation) möglich sein. Sinnvoll ist auch das Einblenden des aktuellen Kamera-Sichtbereichs, um dem Operateur die Orientierung in der realen Szene zu erleichtern (Frage: "Wo schaue ich eigentlich gerade hin ?").

- Ein echtzeitfähiges und zuverlässiges Subsystem zur **numerischen Kollisionsvermeidung** ist zur Entlastung des Operateurs notwendig.

- **Verschiedene, beliebige Szenenansichten** in orthogonaler oder perspektivischer Projektion sollten aus Gründen der Übersichtlichkeit und zur Erleichterung der Orientierung **gleichzeitig** mittels Fenstertechnik möglich sein.

- Leistungsfähige **CAD-Schnittstellen** und **Roboter-Programmierschnittstellen** sollen verfügbar sein. Die Schnittstellen sollten genormt sein oder internationalen Standards entsprechen, um die Anpassung des Systems an neue oder geänderte Aufgabenstellungen zu vereinfachen.

- Für das **sensorbasierte synthetische Sehen** sind **Schnittstellen** von der Roboter- bzw. Zellensteuerung zum Grafiksystem notwendig.

2.2 *Kinematische Simulation als modulare Erweiterung von CAD-Systemen*

Mit der zunehmenden Verbreitung von CAD-Systemen wuchs in der Vergangenheit das Bedürfnis, deren Einsatzbereich über das Zeichnen und die Konstruktion hinaus auf kinematische Untersuchungen und die Off-Line-Programmierung von Industrierobotern - in erster Linie für die industrielle Fertigung - auszudehnen.

So wurden von verschiedenen CAD-Systemhäusern mitunter zahlreiche Erweiterungsmodule entwickelt:

- Das von *McDonnell Douglas Automation* (McAuto) entwickelte CAD-System UNIGRAPHICS wurde 1983 um das speziell für den Zellenentwurf und die Robotorsimulation entwickelte Modul PLACE erweitert. Nachfolgend wurde das Produkt noch um ein Modul für die Anpassung des Simulationsmodells an die reale Zelle (ADJUST), ein weiteres für die einfache Modellierung unter Benutzung einer umfangreichen Roboterbibliothek (BUILD) und um weitere Bausteine zur Taktzeitmessung (CTA) und gleichzeitigen Simulation mehrerer Zellenkomponenten (COMPOUND DEVICES) ergänzt [36]. Die Umsetzung der generierten Bewegungsdaten in eine spezifische Robotersprache und damit in ein durch die Robotersteuerung tatsächlich ausführbares Programm erfolgt wiederum über einen im Modul COMMAND enthaltenen Translator. Als Translator-Zielsprachen werden u.a. VAL, VAL II, ARLA, ROLF, KAREL, SRCL und BAPS unterstützt. Das Gesamtsystem wird unter der Bezeichnung ROBOTICS weltweit vertrieben.
  Die grafische Visualisierung erfolgt bei diesem System hauptsächlich als 3D-Echzeit Drahtmodell. Die verwendeten Ausgabegeräte sind für eine animierte Darstellung von schattierten Modellen nicht geeignet. Sensorschnittstellen für eine mögliche Verwendung des Systems für das grafische *Monitoring* bzw. das synthetische Sehen bei Telerobotik-Applikationen sind nicht verfügbar. Als weitere Einschränkung ist zu bewerten, daß PLACE auf Mechanismen mit maximal 6-Bewegungsachsen beschränkt ist und keine Möglichkeit für die Modellierung geschlossener kinematischer Ketten bietet.

- Für das weitverbreitete CAD-System CATIA (Dassault Systems) werden die Module KINEMATICS zur kinematischen Simulation von Mechanismen und Getrieben [37], sowie ROBOTICS zur Off-Line-Programmierung angeboten [38].

Da CATIA in der Vergangenheit ausschließlich auf Großrechnern implementiert war[1] und verteilte Arbeitsstationen zur visuellen Ausgabe benützt, ist dieses Konzept aufgrund der damit verbundenen lokalen Trennung von Applikationsrechner und Ein-/Ausgabegerät nicht geeignet für Echtzeitanwendungen. Die beiden Module sind außerdem untereinander inkompatibel bezüglich der Datenbasis.

- Von PRIME-COMPUTERVISION wird CIMSTATION als Erweiterungsmodul innerhalb der CADDS4X CAD/CAE-Produktpalette angeboten. Das System erlaubt den Entwurf und die Simulation von Fertigungszellen zur Integration von Industrierobotern und anderer Komponenten zur flexiblen Automatisierung [39].
  Die Ablaufprogrammierung erfolgt hier in der Roboter-unabhängigen, jedoch produktspezifischen Simulationssprache SIL. Die Darstellung der Simulationssequenzen erfolgt als animiertes Drahtmodell.

- Für das CAD-System BRAVO3 der Fa. Schlumberger wird das Erweiterungsmodul MECHANISMS zur Definition von Mechanismen sowie eine Schnittstelle zum Dynamik-Analysesystem ADAMS (Mechanical Dynamics Inc.) angeboten.
  Die Kombination dieser beiden Softwareprodukte erlaubt die Analyse elastomechanischer Schwingungen nahezu beliebiger Mechanismen, sowie die grafische Darstellung von Bewegungs-Datensequenzen als Drahtmodell. Die mechanische Analyse mit ADAMS wird - aufgrund des allgemeinen Ansatzes und der komplexen Natur der Algorithmen - nicht in Echtzeit abgearbeitet.

Im KfK-IRE wird BRAVO3/ADAMS eingesetzt zur Optimierung des Systems Regler-Antriebe-Mechanik für den Transport-Manipulator NET-ABT und den Teststand EDITH.
Die Anwendung der Systemkombination BRAVO3/ADAMS im Rahmen der grafischen Bedienerführung und On-Line-Überwachung (synthetisches Sehen) ist Aufgrund fehlender Schnittstellen zu den Robotersteuerungen und der um Größenordnungen zu langsamen grafischen Leistungsfähigkeit ausgeschlossen.

Eine weitere Kopplung über Pre- und Postprozessoren mit ADAMS bietet INTERGRAPH für das mechanische CAD-System I/EMS mit den Modulen I/MEM (mechanism modeling system) und I/KIN (kinematics mechanism modeling system) an. Die für eine dynamische Simulation von ADAMS benötigten allgemeinen Masseeigenschaften der Modellteile (Masse, Schwerpunkt, Trägheitsmoment) werden über einen Preprozessor unmittelbar aus dem CAD-Modell abgeleitet.

Die beiden letztgenannten Systemlösungen sind jedoch nicht direkt geeignet für die **Simulation elastomechanischer Verformungen**, dies würde die Ankopplung an ein FEM-Programm wie NASTRAN, PATRAN oder ANSYS erfordern.

Aus dem Forschungsbereich sind weiterhin verschiedene Arbeiten und Ansätze bekannt die, ebenfalls auf vorhandenen CAD-Systemen aufbauend, Module zur grafischen Visualisierung und Off-Line-Programmierung von Industrierobotern bereitstellen:

- Das am IWB der TU München für den Zellenentwurf, die Bewegungssimulation und die Roboterprogrammierung entwickelte System USIS [40] basiert auf dem CAD-System EUCLID von MATRA-Datavision und ermöglicht außerdem die Übernahme von CAD-Modellen aus GEOMOD (SDRC-IDEAS) und ROMULUS (Shape Data Ltd.) [41]. Die Darstellung der Bewegung erfolgt als Echtzeit-Drahtmodell [42].

---

1 Seit 1990 ist CATIA ebenfalls auf der IBM-Workstation RS6000 erhältlich.

- Am ISW der TH Stuttgart wurde ein Programmiersystem unter Integration des CAD-Systems PROREN2 entwickelt [43]. Die Roboter-Modellierung erfolgt hier über die Eingabesprache RDL (Robot Description Language). Zur textuellen Roboterprogramm-Erstellung wird in diesem System die Programmiersprache BAPS verwendet.

2.3 *Spezielle Roboter Off-Line-Programmiersysteme*

Die Anbindung von CAR-Systemen an CAD-Produkte, oder deren Entwicklung unter Verwendung eines bereits vorhandenen CAD-Kerns, bietet für den Entwickler vor allem den Vorteil einer drastischen Verminderung des Entwicklungsaufwands. Als Nachteile müssen bei einer derartigen Lösung in Kauf genommen werden:

- Die Datenbasis von CAD-Systemen und deren rechnerinternes Datenmodell sind vor allem auf Anwendungen des Zeichnens, des Modellierens und der Konstruktion optimiert und nicht in erster Linie bezüglich der Darstellungsgeschwindigkeit und Laufzeiteffizienz.

- Die Kommunikation zwischen dem eigentlichen Simulationssystem und dem zur Verwaltung, Transformation und Darstellung der geometrischen Modelldaten verwendeten CAD-System oder -Kern erfordert einen hohen Rechenzeitbedarf, der zu Lasten der Darstellungsgeschwindigkeit geht. Dies gilt unabhängig davon, ob Simulationssoftware und CAD-System wie bei verteilten Systemen über externe Schnittstellen, oder wie bei integrierten Lösungen über prozedurale Schnittstellen erfolgt.

- Die verwendeten geometrischen Datenmodelle sind durch das CAD-System vorgegeben. Oftmals muß zusätzlich eine weitere Kopie der Geometriedaten, oder gar eine andere Darstellungsform durch das Simulationssystem gespeichert werden, z.B. zum Zweck einer effektiven Kollisionsdetektion.

- Die Verwaltung der Modelltopologie bezüglich Kinematik, Baugruppenstrukturen und Detaillierungsstufen, sowie der roboterspezifischen technologischen Daten wie z.B. zur Definition von Achs-Verfahrbereichen und zur Beschreibung der Antriebs- und Steuerungseigenschaften, obliegt dem Simulationsmodul. Eine einheitliche und effiziente Datenverwaltung ist daher nicht möglich.

Aus den genannten Gründen ging man Anfang der 80er Jahre dazu über, Robotersimulations- und Off-Line-Programmiersysteme von Grund auf neu, und ohne Anbindung an ein bereits vorhandenes CAD-System oder Einbindung eines existierenden CAD-Kerns, zu entwickeln :

- Das ursprünglich an der Universität Nottingham entwickelte Simulationssystem GRASP (BYG Systems Ltd.) [44], [45] hat besonders aufgrund der geringen Einstiegskosten und seiner Ablauffähigkeit auf einer Anzahl von Geräten verschiedener Hersteller (APOLLO, SUN, TEKTRONIX, DEC, IBM RS-6000) im industriellen Planungsbereich, sowie in Forschung und Lehre Verbreitung gefunden [46].
  Das Produkt kann zum Entwurf von Roboterzellen und zur Visualisierung und Animation von Fertigungsabläufen benutzt werden. Facettierte Geometrie- und Kinematikmodelle werden textuell oder interaktiv am System generiert. Das Programm ist in der Lage, Roboterprogramme in der produktspezifischen Hochsprache GRDATA zu generieren. Der erzeugte Code muß über geeignete Postprozessoren in ein ablauffähiges Roboterprogramm umgesetzt werden [47].
  Flüssige Animationssequenzen lassen sich in GRASP, selbst bei Ablauf auf Grafik-Workstations neuerer Technologie, ausschließlich bei der Darstellung als Drahtmodell erzeugen.

- Im Jahre 1986 wurde, besonders erfolgreich bei Anwendern aus der Automobilindustrie, das integrierte Roboter-Programmier und -Simulationsprogramm ROBCAD von der israelischen Firma TECNOMATICS auf dem europäischen Markt eingeführt. Die Software wurde speziell für Hochleistungsgrafik-Workstations der Fa. SILICON GRAPHICS (USA) geschrieben und präsentiert sich als integriertes System zur Planung und Optimierung von Arbeitszellenmodellen, zur Off-Line-Programmierung und animierten 3D-Bewegungssimulation von Industrierobotern [48]-[50].
  Komplexe Simulationssequenzen können in der PASCAL-ähnlichen kombinierten Simulations- und roboterunabhängigen Programmiersprache TDL [51] entwickelt werden. Es existieren verschiedene Postprozessoren für die Umsetzung von TDL in die jeweils erforderliche Roboter-Zielsprache.
  Über CAD-Prozessoren (IGES, VDA-FS) und direkte Schnittstellen (GEOMOD, CATIA) ist der Import von Geometriedaten von CAD-Systemen zu ROBCAD möglich. ROBCAD besticht vor allem durch zahlreiche Funktionen zur Definition und Simulation von Roboter-Fahrbefehlen und eine umfangreiche Roboter-Bibliothek. Bewegungssequenzen werden als Drahtmodell flüssig und mit der Einführung 1988 der SILICON GRAPHICS 4D-Workstation-Serie auch mit Einschränkungen in schattierter (flat-shaded) Darstellung visualisiert.
  Schnittstellen für die Übertragung von Sensordaten von Roboter- bzw. Zellensteuerungen nach ROBCAD - und damit die Möglichkeit zu dessen Einsatz in Monitor-Systemen - existieren nicht. ROBCAD bietet vor allem keine Möglichkeit zum Modellieren von verschiedenen Detailstufen in einem hierarchischen Datenmodell. Die interne Struktur des Geometrie- und Kinematik-Datenmodells ist nicht offengelegt und bietet damit keine Möglichkeit für Erweiterungen oder Änderungen.

- Das Roboter-Simulationssystem IGRIP von Deneb Robotics Inc. [52], [53] bietet ähnliche Eigenschaften wie ROBCAD. Es wird in der BRD vorwiegend von Forschungseinrichtungen (FhG-IPK, FhG-IPA) eingesetzt und für Grafik-Workstations von SILICON GRAPHICS, HP, SUN, INTERGRAPH und TEKTRONIX angeboten. An IGRIP besonders hervorzuheben wären das variantenreiche System zur Kollisionserkennung (exakter Geometrievergleich, konfigurierbare Vergleichslisten, Ansprechen der Warnung bei Unterschreitung eines definierten Abstands oder bei direkter Berührung usw.), die effiziente grafische Darstellung auch im schattierten Modus, sowie die einfach zu bedienende und optisch ansprechende Benutzerschnittstelle (Firmenslogan: "The Best User Interface on the Market") [54].
  Eine Möglichkeit zur Modellierung von hierarchisch gegliederten, im Detaillierungsgrad interaktiv veränderlichen Anlagenmodellen ist jedoch auch in IGRIP nicht gegeben. Weiterhin bestand keine Möglichkeit zur Modellierung von kinematisch redundanten Manipulatoren und zur Einbindung eigenen Routinen zur inversen Kinematik-Transformation.

- An der Universität Saarbrücken wurde das Roboter-Simulationspaket ROBSIM entwickelt [18]. Diese Hostrechner-basierte Entwicklung benützt zur grafischen Darstellung Hochleistungs-Terminals der Fa. RASTER TECHNOLOGIES. Dieses inzwischen als Produkt vertriebene Paket ist ebenfalls in der Lage, bewegte Drahtmodelle in Echtzeit darzustellen, schattierte Visualisierung ist lediglich statisch möglich.

- Das bei SIEMENS in Erlangen entwickelte Roboter-Programmier und Simulationssystem SMS (Simulation mechanischer Systeme) erlaubt durch Integration des an der Universität Karlsruhe entwickelten Programms MESAVERDE auch die Simulation der Roboterdynamik.

Zusätzlich zu den o.g. Beispielen für Roboter-Programmier- und Simulationssysteme existieren noch Produkte wie ROBOT-SIM von CALMA [55], das am WZL der RWTH Aachen im Rahmen der Sprachentwicklung ROBEX-M [56] zur grafischen Darstellung benutzte Modul GROSIM [23] und das an der Universität Karlsruhe entwickelte und von Dillmann vorgestellte System ROSI [57], sowie umfassendere Ansätze wie z.B. das ebenfalls an der Universität Karlsruhe entwickelte Produktmodell-Konzept DICAD [58].

2.4 Zusammenfassende Bewertung der untersuchten Systeme

Die von [24] entnommene und auf den aktuellen Stand und im Hinblick auf eine mögliche Anwendung im Bereich der hochflexiblen Handhabungstechnik modifizierte Tabelle 1 auf Seite 16 zeigt eine Übersicht der verschiedenen Systeme aus dem Bereich der kinematischen und/oder dynamischen Simulation von Mechanismen, und der Off-Line Programmierung für Industrieroboter.

Auffällig ist vor allem, daß keine dieser Entwicklungen eine Sensorschnittstelle zur On-Line-Übertragung von Roboter-Gelenkwinkel oder anderer Prozeßparameter wie Werkzeugstatus, Position von hantierten Objekten usw., und damit die Möglichkeit zum grafischen Echtzeit-Überwachen oder zur automatischen Nachführung von Kameras bietet. Weiterhin erfüllt keines der Systeme die gleichzeitige Forderung nach schattierter Darstellung in Echtzeit und zum interaktiven Wechsel des Detaillierungsgrads.

Fast alle untersuchten Realzeitsysteme sind bezüglich der kinematischen Modellierfähigkeit auf Industrieroboter mit maximal 6 bewegten Gelenken beschränkt, oder bieten keine schattierte Darstellung mit ausreichenden Bildberechnungsraten, wie es für die manuelle Operation von Handhabungsgeräten notwendig ist. Dagegegen sind dort oft andere Anforderungen, wie z.B. die Portierbarkeit der Programme auf unterschiedliche Rechnertypen, im Hinblick auf den kommerziellen Erfolg von wichtigerer Bedeutung.

Letztlich ist diese Bilanz auch nicht verwunderlich, da die untersuchten Systeme sich an den Anforderungen der Konstruktion (Erweiterung von CAD-Systemen für die kinematische und/oder dynamische Simulation), oder der Fertigungsautomatisierung (Off-Line-Programmiersysteme) orientieren und nicht an den neuartigen und unkonventionellen Bedürfnissen der grafischen Echtzeitunterstützung für die hochflexible Handhabungstechnik.

Daher war trotz des zu erwartenden hohen Entwicklungsaufwands ein **neuer Ansatz** für ein integriertes System zur Simulation, Steuerung und Off-Line-Programmierung im Rahmen diese Arbeit notwendig.

	Kinematisches Modell							Kollisions-erkennung				Simulation				Zusätzliche Funktionen			
	Gelenke			Struktur															
	translatorisch	rotatorisch	Sonderformen	offen-einfach	offen-verzweigt	offen-gekoppelt	geschlossen	Visuell	Automatik	autom. Bahnkorrektur	Testumfang definierbar	mehrere Roboter	Schattierte Darst.	mehrere Ansichten	Sensorgeführt	Kamerasimulation	Detail-Hierarchie	Roboter-Dynamik	Greifen/Loslassen
BRAVO3 + ADAMS	★	★	-	★	★	-	★	★	-	-	-	▲	△	★	△	△	△	★	△
CATIA	★	★	★	★	-	-	▲	★	-	-	-	★	△	★	△	△	▲	★	★
CimStation (Prime)	★	★	-	★	-	-	△	-	★	△	▲	★	△	-	△	△	△	△	★
CIMSTATION (Silma)	★	★	-	★	-	-	-	★	-	-	-	★	▲	-	△	-	-	★	★
GRASP (BYG, GB)	★	★	▲	★	★	▲	△	-	★	△	△	★	▲	★	△	△	△	△	★
GRASP (USA)	★	★	★	★	-	-	★	-	-	-	-	-	-	-	△	△	△	-	-
IGRIP	★	★	△	★	★	★	▲	-	★	△	★	★	★	△	△	▲[3]	△	▲[1]	★
ISW Stuttgart	★	★	-	★	-	-	-	△	-	-	-	★	▲	-	△	△	△	-	-
ROBCAD	★	★	△	★	★	★	▲	-	★	△	△	★	▲	▲[1]	▲[2]	▲[1]	△	△	★
ROBOTICS (McAuto)	★	★	-	★	-	-	△	★	▲	△	△	★	△	-	△	△	-	△	★
Robot-Sim (CALMA)	★	★	-	★	-	-	★	★	-	-	-	★	-	-	△	△	-	★	★
ROBSIM (PCS)	★	★	△	★	-	-	△	★	-	-	-	-	△	-	△	△	△	△	★
ROSI (Uni Karlsruhe)	★	★	△	★	-	-	△	-	-	-	-	★	▲	△	△	△	△	△	★
SMS (Siemens)	★	★	-	★	-	-	-	-	★	-	★	★	▲	▲	△	-	△	★	△
USIS	★	★	-	★	-	-	△	★	-	-	★	-	▲	-	△	△	△	-	△
WZL-RWTH Aachen	★	★	-	★	-	-	-	-	★	-	-	-	△	-	△	△	△	△	★

★ möglich / vorhanden ▲ teilweise möglich / vorhanden △ nicht möglich / vorhanden
- aus den Unterlagen nicht ermittelbar
[1] Seit der neuesten Version (1991)
[2] Sensorkugel o.ä, jedoch keine Gelenkwinkel-Eingabe
[3] Ausgabe der Kameraparameter zur Steuerung nicht möglich

Tabelle 1. Roboter-Programmiersysteme: Fähigkeiten und Eigenschaften existierender Roboter-Programmier und -Simulationssysteme.

3. Konzeptbildung und Realisierung

Da die Untersuchung bisher existierender Simulationssysteme zeigte, daß ein neuartiger Ansatz notwendig war, führte die nachfolgende im Rahmen der vorliegenden Arbeit geleistete Entwicklungstätigkeit zur Implementierung zweier Simulationsprogramme:

- **GBSim** (**G**raphical **B**oom **Sim**ulator) ist ein Programm, das besonders zur Erprobung des Konzepts des *synthetischen Sehens* am Anwendungsbeispiel der "In-Vessel"-Handhabung bei JET für die grafische Bedienerunterstützung bei der manuellen Steuerung des Großmanipulators *Articulated Boom* (IVHU, In-Vessel Handling Unit) und dessen verschiedener End-Effektoren (u.A. ein EMS-Manipulator und Spezialeffektoren für HF-Antennen bzw. "Belt-Limiter"-Hantierung) entwickelt wurde. GBSim stellt eine **Speziallösung** für diese Applikation dar, eine Adaption auf andere Arbeitsumgebungen wäre nur unter erheblichem Entwicklungsaufwand möglich gewesen. Im Zuge dieser Entwicklungsarbeiten wurden u.A. ein neuartiger Algorithmus zur schattierten Echtzeit-Darstellung ("Hidden-Surface"-Problem) erarbeitet. GBSim erlaubt weiterhin die Off-Line-Programmierung der unterstützten IVHU, numerische Echtzeit-Kollisionserkennung und die Simulation von Kamera-Ansichten zur Vorplanung von Arbeitsabläufen. Das System befindet sich seit 1988 im Einsatz bei JET.

- Die mit GBSim gemachten positiven Erfahrungen führten rasch zum Wunsch nach einem universell einsetzbaren, d.h. nicht auf eine spezielle Anwendung spezialisierte, und verhältnismäßig leicht an andere Teleoperationsaufgaben anpassbare Systemlösung für ein grafisches Echtzeit-Simulationsprogramm. Die Berücksichtigung der bereits angeführten speziellen Anforderungen für die Hantierungstechnik[2] und die mit GBSim gemachten Erfahrungen führten daher zur Entwicklung von **KISMET** (**Ki**nematic **S**imulation, **M**onitoring and Off-Line Programming **E**nvironment for **T**elerobotics).

Die hierbei entwickelten und teilweise völlig neuartigen Konzepte werden nachfolgend erläutert am Beispiel verschiedener Detail-Realisierungen. Dazu gehören besonders die **Topologie** des **geometrischen** und **kinematischen Datenmodells**, der neuartige Ansatz zur **Kollisionserkennung** und **-vermeidung**, die gewählten **Darstellungsmethoden**, sowie das Subsystem zur **Steuerung, Nachführung** und **Simulation von Szenenkameras**. Weiterhin wird hier besonders das Problem der **effektiven geometrischen Modellierung** zur Ausnutzung der speziellen Hardwarefunktionen angesprochen.

3.1 Systemanforderungen und Entwurfskriterien

Die einander widersprechenden Anforderungen an das 3D-grafische *Monitoring* (≡ Prozeßüberwachung) von der Benutzerseite, nämlich :

1. hoher Detaillierungsgrad der modellierten Geometrie- und Baugruppenteile, und daraus resultierend eine zunehmende Erhöhung der Datenmenge für die grafische Darstellung und die Verwaltung der Modelldatenbasis, und

2. die gleichzeitige Forderung nach einer hohen Darstellungsrate (Anzahl neuberechneter Bilder je Sekunde), um die für die Überwachung von automatischen Sequenzen und das manuell bediente Verfahren notwendige Rückkopplung und damit die über den Bediener geschlossene Regelschleife überhaupt erst zu ermöglichen. Als

---

[2] siehe hierzu "Anforderungen und Beurteilungskriterien" auf Seite 9

minimale Leistungsanforderung wird für die Überwachung der z.B. bei JET eingesetzten Hantierungsgeräte eine Bildrate von 5 Darstellungen je Sekunde angesehen.

Aufgrund dieser Anforderungen wurde für KISMET eine rekursive und baumartig strukturierte, **hierarchischen Architektur** des **Geometrie-** und **Baugruppen-Datenmodells** entwickelt. Diese Datenstruktur ermöglicht insbesonders :

- Baugruppen und Geometrieteile in verschiedenen Detaillierungsstufen für die synthetische Szenenpräsentation aktivieren und darstellen zu können.
- Für die augenblickliche Arbeitssituation nicht benötigte **Modellteile** interaktiv schnell und komplett aus dem Hauptspeicher auszulagern, oder umgekehrt, für die grafische Darstellung **aktivieren** und bei Bedarf wieder **deaktivieren** zu können.
- **Algorithmen** und **Bedienfunktionen**, die den Handhabungs-Operateur bei der Auswahl der gerade aktiven Modellteile unterstützen und die Systembedienung beschleunigen.

Abbildung 7 zeigt die Darstellung einer Flanschbaugruppe bei der sensorunterstützten, automatischen und Off-Line programmierten Montage mit einem Industrieroboter in verschiedenen Detailstufen.

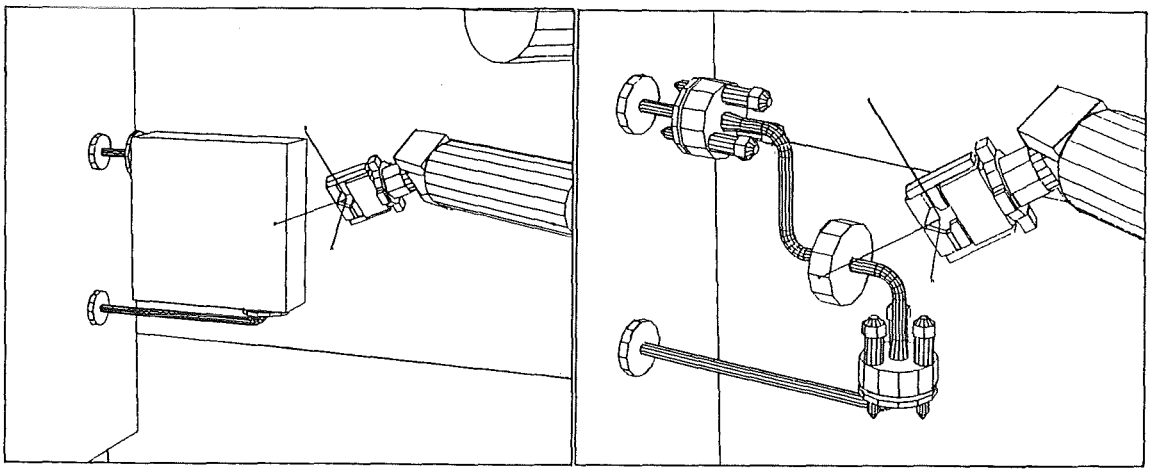


Abbildung 7. Flanschdarstellung in verschiedenen Detailstufen

Im linken Bild wird lediglich ein einhüllender Quader statt der detaillierteren Baugruppe (rechtes Bild) gezeichnet. Diese einfachere Darstellung (linkes Bild) reicht für die grobe Bewegungsplanung oder für die Darstellung der Arbeitszene außerhalb des eigentlichen Arbeitsgebiets völlig aus. Die Einteilung in verschiedene Detaillierungsebenen und Hierarchiestufen wird bei der Modellierung festgelegt.

3.2 *Hierarchisches Datenmodell*

Die Datenbasis des grafischen Monitor-Systems besteht für jede Handhabungs-Applikation aus zwei Teilen:

1. Der **Modell-Datenbasis** :
   Dies ist das auf der Festplatte gespeicherte geometrische und kinematische Modell der Arbeitsumgebung und der Manipulatoren. Es umfaßt alle Teile der Arbeitszelle in allen verfügbaren Detailstufen. Es ist so strukturiert, daß die Baugruppen-Topologie erfaßt wird.

2. Der **Darstellungs-Datenstruktur** :
   Diese Datenstruktur wird während des Simulationsablaufs im Hauptspeicher des Grafiksystems aufgebaut und während des Detailwechsels gegebenenfalls ausgetauscht. Sie stellt eine der augenblicklichen Arbeitssituation angepaßte Teilmenge der Modell-Datenbasis dar.

3.2.1 Begriffserläuterung und Datenmodell-Topologie

Ein **ABSTRACT** ist in KISMET ein Bezugsknoten innerhalb der hierarchisch gegliederten Modelldaten-Baumstruktur. Jedes ABSTRACT ist über Zeichenketten innerhalb des Modellbaums eindeutig identifizierbar. Minimal definiert ein ABSTRACT das 3D-Bezugs-KS (KS ≡ Koordinatensystem), auch **ABS-FRAME** genannt, einer Baugruppe. In der Regel (nicht zwingend) sind weitere Geometrieteile (GEO), Werkstück-Arbeitspunkte (WFRM, Workframe) oder kinematische Definitionen mit dem ABSTRACT verknüpft. Eine Liste von ABSTRACT-Datensätzen wird in einer **ABSTRACT-Datei** zusammengefaßt ('.mpc'-Datei). Innerhalb einer jeden ABSTRACT-Datenstruktur können weiterhin definiert werden :

GEO Geometrie-Elemente definieren die geometrische Gestalt aller Baugruppen des Simulationsszenarios bzw. der Arbeitszelle. Geometrie-Elemente sind über eine weitere Modellier-Transformation mit dem ABS-FRAME verknüpft. Eine Lageänderung des ABS-FRAME bewegt daher automatisch alle verknüpften Geometrieteile mit.

WFRM Werkzeug-Arbeitspunkt bzw. -Bezugs-KS (engl. Workframe) am Werkstück zur Kennzeichnung der Arbeitspositionen des Werkzeugs am bearbeiteten Objekt. Es kann auch zur Definition von Zwischenpositionen von Bewegungsabläufen verwendet werden.

TCPF Das Werkzeug-Koordinatensystem (Tool Centre Point Frame) des Roboters definiert die augenblickliche Bezugsposition und -orientierung des Werkzeugs bezüglich des aktuellen Roboter-Basis-KS. Die relative Lage des TCPF bezüglich des Werkzeug-Anschlagpunkts am Roboter ist variabel und erlaubt daher eine Transformation (Translation, Rotation) bezüglich der vom Roboter-Konstrukteur bzw. -Hersteller vordefinierten Position.

ZPF Das ROBOTER-Nullpunkt-Koordinatensystem (Zeropoint Frame) ist das aktuelle Bezugssystem für kartesische Bewegungskommandos in Roboterprogrammen. Verschiedene Robotersteuerungen erlauben das relative, programmgesteuerte Verschieben des ZPF ("Nullpunkt-Verschiebung").

CAMERA Simulierte Szenenkameras, die kinematisch stets mit Roboter-KS, bzw. Umgebungs-KS verknüpft sind. In der Regel werden simulierte CAMERAs

innerhalb der Struktur eines Mechanismus (Roboter) anstatt oder zusätzlich zu einem TCPF definiert.

USS Simulierte Ultraschall-Sensoren (Ultrasonic Sensors)

LIGHT Bewegliche, mit einem ABS-FRAME kinematisch verknüpfte Lichtquellen.

Optional kann jeder ABSTRACT-Datenknoten auf genau eine weitere ABSTRACT-Datei verweisen, wodurch die Möglichkeit gegeben ist, einen **hierarchisch gegliederte Modell-Datenbaum** der simulierten Hantierungszelle aufzubauen; vgl. dazu Abbildung 8 und Abbildung 10. In ABSTRACT-Dateien werden folgende Modell- bzw. Kinematik-Eigenschaften definiert:

- Jede Modell-Datenbasis muß genau eine Modell-Basistransformation enthalten (eine ABSTRACT-Datei des Typs BASE-FILE).
- Bewegte mechanische Strukturen wie z.B. Industrieroboter, Transportsysteme und Effektoren, Getriebe und andere Maschinenelemente, die mindestens ein bewegliches Element aufweisen. Die Datei ist dann vom Typ ROBOT-FILE.
- Werkzeuge die von den aktiven Mechanismen bewegt werden, für die jedoch selbst keine beweglichen Strukturen modelliert sind.
- Baugruppenstrukturen (Topologie) der Arbeitszelle.
- Bewegungsart, d.h. nicht aktiv bewegliche Modell-KS in Arbeitszellen, bzw. aktiv bewegliche translatorische und rotatorische Gelenke von kinematischen Strukturen. Jedes kinematische Gelenk definiert maximal einen Freiheitsgrad in einer kinematischen Struktur. Freiheitsgrade in **geschlossenen** Ketten werden in KISMET passiv bewegt, d.h. innerhalb der Kette wird ein aktiv durch den Benutzer bewegliches Führungsgelenk (der Freiheitsgrad) definiert, die restlichen Gelenke innerhalb der Schleife werden passiv nachgeführt (geometrische Zwangsführung).
- Referenzen auf Geometrie-Elemente sowie deren zugehörige Modellier-KS
- Mechanische Bewegungsgrenzen von Gelenken, maximale Gelenk-Geschwindigkeiten und -Beschleunigungen.
- Die Position in der kinematischen Kette durch Referenzierung von **kinematischen Vorgänger-** und **Nachfolgegelenken**. Nachfolgegelenke müssen nur dann definiert werden falls :
  1. ein Gelenk innerhalb einer kinematischen Schleife zwangsgeführt wird,
  2. zur Definition von Freiheitsgrad-Sonderformen. Dazu gehören in ihren Abmessungen oder in ihrer Form veränderliche Bauelemente wie z.B. Gummiseile, Schutzmanschetten, Spiralfedern usw.. Die Formänderung solcher Bauelemente, die eigentlich als Starrkörper definiert sind, kann in KISMET annähernd realistisch simuliert werden, indem die zur Modellierung verwendeten Geometrie-Elemente in einer Dimension skaliert dargestellt werden. Dabei wird der Skalierungsfaktor als Distanz zwischen dem Bezugs-FRAME der Baugruppe und dem Ursprung des definierten Nachfolgegelenk-FRAME berechnet.
- Elemente von passiv hantierten Baugruppen oder der nicht hantierbaren Arbeitsumgebung von simulierten Szenarien innerhalb einer Roboter-, Hantierungs- oder Fertigungszelle. Zur Arbeitsumgebung werden passiv hantierte Objekte und solche, die zur Kollisionsvermeidung oder zur Orientierung des Operateurs innerhalb des Simulationsmodells definiert werden, gezählt.

3.2.2 Charakterisierung der Dateitypen und der Dateibaum-Strukturen

Alle von KISMET verwendeten Benutzerdateien haben ein textuelles ASCII-Format und können daher sowohl mittels Text-Editor (auf UNIX-Systemen z.B. 'vi', 'emacs') als auch über die in KISMET vorhandenen interaktiven grafischen Editiermöglichkeiten erzeugt

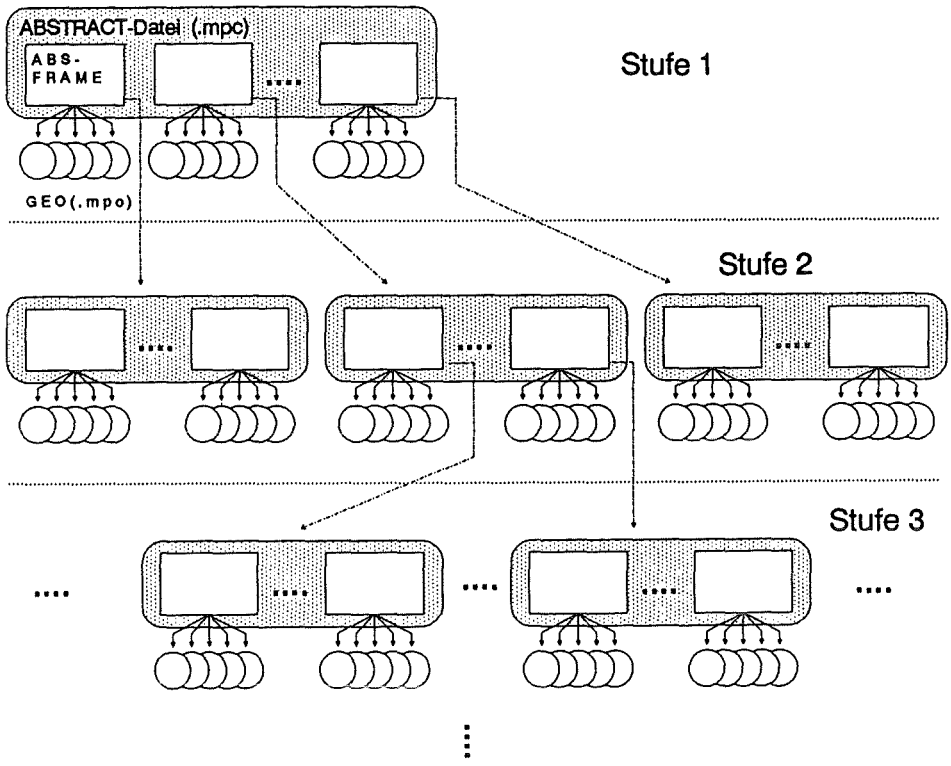


Abbildung 8. Hierarchische Struktur von ABSTRACT-Dateien

werden. Die Verwendung von textuellen Formaten erleichtert außerdem die Prozessor-Entwicklung zum Modelldaten-Transfer von kommerziellen CAD-Systemen in das KISMET-spezifische Datenformat. Dies wurde bereits unter Beweis gestellt bei der Entwicklung des für KISMET verfügbaren *CAD*I* -Postprozessors[3] **KISPOST.**

Die einer spezifischen Applikation in KISMET zugrundeliegende Dateistruktur ist in Abbildung 9 dargestellt. Verschiedene Dateitypen (z.B. '.col', '.mat') können in verschiedenen Simulationsmodellen gleichzeitig benützt werden. Während der Initialisierung von KISMET spezifiziert der Benutzer den Dateinamen der **Simulations-Steuerdatei** ('.sim'). Die angegebene Datei enthält nun Verweise in Form von Dateinamen zu den weiteren von KISMET benötigten Datensätzen.

Die in der Steuerdatei aufgelisteten Dateien definieren folgende Merkmale des Simulationsablaufs:

1. Die **VIEWING-LAYOUT**-Datei ('.lay') initialisiert das KISMET-Bildschirmlayout, d.h. Position und Größe der verschiedenen Szenen-Sichtfenster (engl. Viewports) und die Sichtparameter der verschiedenen Ansichten (ursprüngliche Betrachterposition und Blickrichtung, Dimension des dargestellten 3D-Weltausschnitts), sowie die zu verwendende Projektionsart (perspektivisch oder orthogonal).

---

3 *CAD*I* (CAD Interface) ist eine im Rahmen des ESPRIT-Projekts 322 definierte Schnittstelle zum Austausch von CAD-Geometriemodellen, siehe auch "CAD-Modelldatentransfer" auf Seite 36.

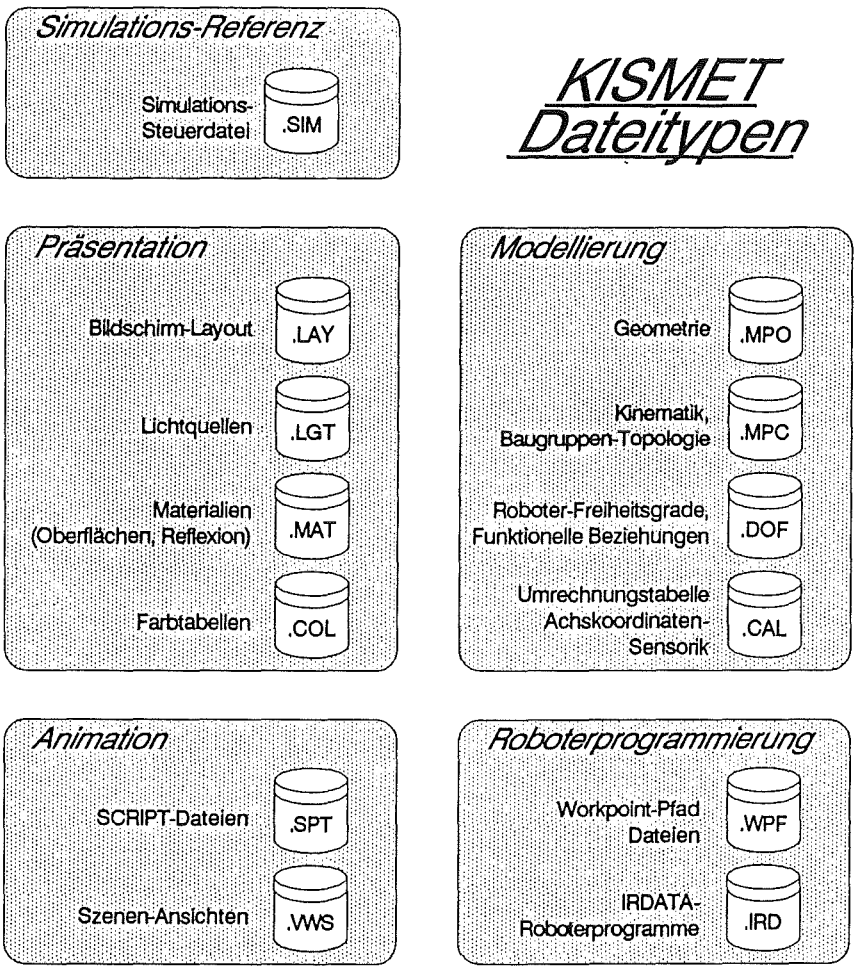


Abbildung 9. KISMET-Dateimodell

2. Die **LIGHTING**-Datei ('.lgt') gestattet die Definition von bis zu 8 Parallel- oder Punktlichtquellen mit variabler Intensität und Farbe.

3. In der **MATERIAL**-Datei ('.mat') werden die für die für eine realistische schattierte Darstellung der Reflexionseigenschaften von Geometrieoberflächen notwendigen Parameter verschiedener Materialoberflächen spezifiziert.[4]

4. Die **COLOUR**-Datei ('.col') definiert eine Farbtabelle für die Darstellung im Drahtmodell-Modus, für Beschriftungen auf Geometrieteilen, Texturen usw. .

5. Die **ABSTRACT-BASE**-Datei ('.mpc') definiert die Wurzel und oberste Hierarchiestufe des Modelldatenbaums. Auf alle weiteren Modelldatenzweige wird ausgehend von dieser Datei über deren Bezeichner verwiesen. Je nach Bedarf werden die weiteren Modelldateien während des Simulationslaufs aktiviert.

---

4 Siehe auch "KISMET Darstellungs- und Schattierungsmodelle" auf Seite 53.

3.2.3 Rechnerinterne hierarchische Darstellungs-Datenstruktur

Während der Initialisierung eines Baugruppen-Datenzweigs bzw. einer Detaillierungsebene wird die in Abbildung 10 skizzierte programminterne Laufzeit-Datenstruktur von KISMET erzeugt. Für jedes in der '.mpc'-Datei definierte topologische Element ABSTRACT wird von KISMET die Datenstruktur *MPKIN* dynamisch zugewiesen.

Es handelt sich hierbei bezüglich der implementierten Zeigerstruktur um einen **asymmetrischen Binärbaum**. Asymmetrisch deshalb, weil den zwei Knotenzeigern bezüglich der Detaillierungsoperationen eine vordefinierte Bedeutung zukommt; vgl. dazu "Operationen auf der hierarchischen Datenstruktur". Innerhalb der *MPKIN*-Struktur weist der Zeiger *p_brother* auf das nächste *MPKIN*-Element innerhalb der gleichen Detailstufe, der Zeiger *p_son* auf das erste Element der geometrisch verfeinerten, tieferliegenden Detailstufe. Der Zeiger *p_father* erlaubt ein Durchlaufen des Datenbaums in Richtung von den Blättern zur Wurzel und weist auf das Vaterelement.

Dieser Baugruppen- und Detailstufen-Zeigerstruktur ist eine zweite **kinematische Zeigerstruktur** überlagert. Der Zeiger *kin_last* weist auf das kinematische *Vorgängerelement* (in Richtung der Roboter-Basis). Der Zeiger *kin_next* wird innerhalb der *MPKIN*--Struktur zur Abarbeitung von kinematischen Schleifen benötigt.

Jede *MPKIN*-Struktur verwaltet zwei weitere Zeiger *p_geo* und *p_wfrm*. Der Zeiger *p_geo* verweist auf eine in sich einfach gekettete Liste von Geometrie-Elementen (GEO, Datenstruktur *MPGEO*; vgl. "Geometrisches Modell" auf Seite 27). Der zweite Zeiger *p_wfrm* weist auf das erste Element einer ebenfalls einfach geketteten Liste von WFRM-Datenblöcken. Beide Listen sind kinematisch über eine Transformation **T** an das ABS-FRAME angekoppelt.

Während der **Modellabarbeitung** des rechnerinternen Darstellungsmodells wird beim Durchlaufens des Datenbaums für alle Operationen (Darstellung, Neuberechnung der Kinematik, Setzen von Attributen usw.) der rekursive Algorithmus *walk_tree* verwendet. Dabei wird die Liste von *MPKIN*-Elementen innerhalb einer Hierarchiestufe (Zeiger: *'p_brother'*) linear abgearbeitet, während zum rekursiven Tieferverzweigen (Zeiger: *'p_son'*) die Verzweigungsbedingung erfüllt sein muß. Der Datenknoten muß dazu mit den Attributen *"Datenzweig geladen"* und *"Datenzweig aktiv"* versehen sein. Zur Abarbeitung des gesamten Modells (ab der Wurzel) wird die Routine mit der globalen Zeigervariablen *ABS_Root* aufgerufen, die auf das erste Element in der obersten Detailstufe (Level 1) zeigt.

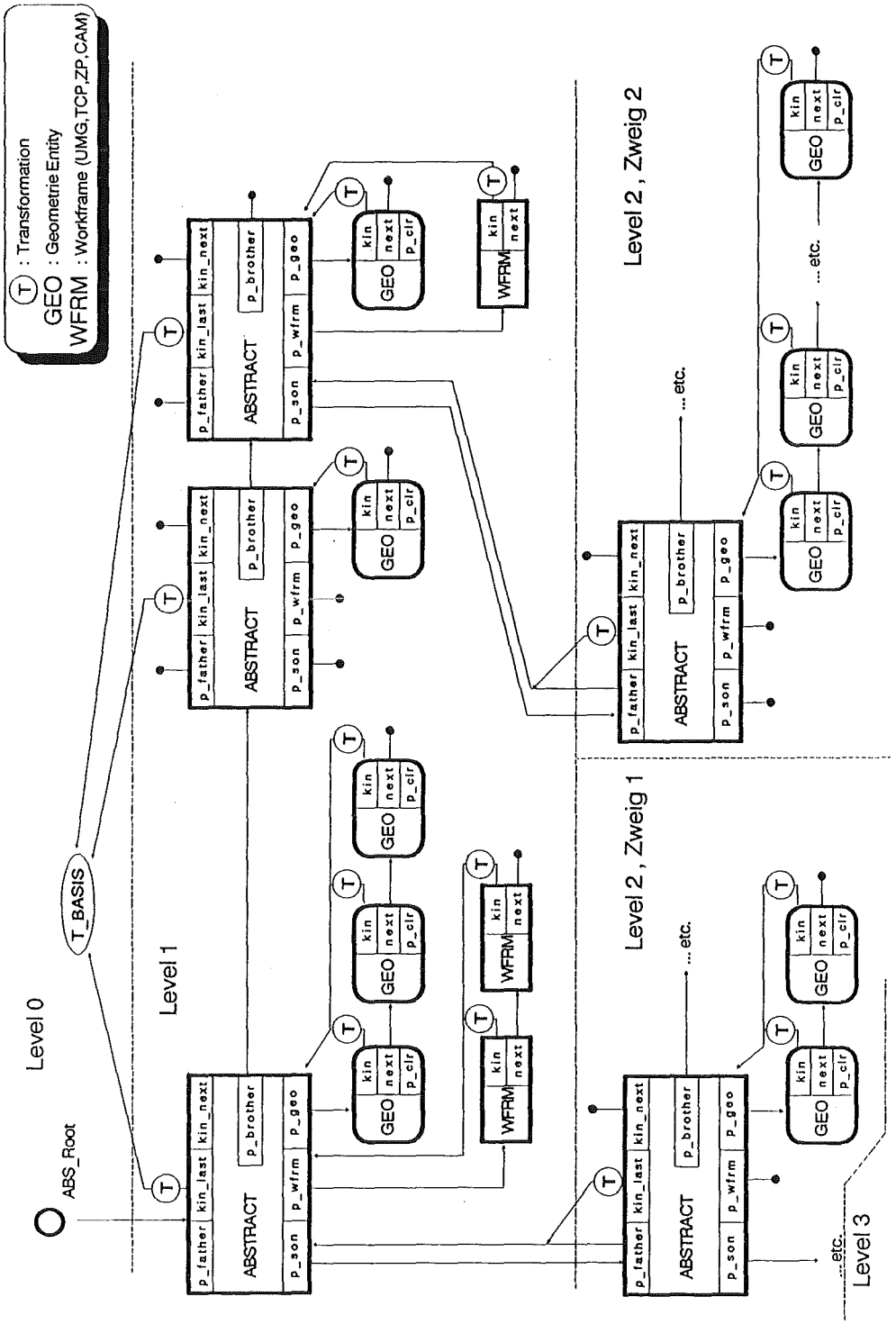


Abbildung 10. Hierarchische Datenstruktur (intern)

3.2.4 Operationen auf der hierarchischen Datenstruktur

Wie bereits in "Systemanforderungen und Entwurfskriterien" auf Seite 17 angedeutet, erlaubt die hierarchische Datenstruktur von KISMET während aller Betriebsmodi die Steuerung von Darstellungskomplexität bzw. Detaillierungstiefe und, damit eng verknüpft,

der Darstellungsgeschwindigkeit (Bildwiederholrate). Dies setzt natürlich voraus, daß die Datenbasis entsprechend modelliert ist.

Tabelle 2 gibt eine Übersicht über die implementierten Steuerungsfunktionen.[5]

LOAD	Ab der angewählten Baugruppen-Kennung (BK) wird eine ABSTRACT-Datei aus der Datenbasis geladen und automatisch für die Darstellung aktiviert. Die Auswahl erfolgt wie auch bei den nachfolgenden Operationen durch grafisches Auswählen (*Picken*) des Vater-Knotens.
DEACTIVATE	Deaktiviert alle in der Modellhierarchie unter der selektierten BK liegenden Detailebenen und Modellteile. Die Teile werden jedoch nicht aus der Laufzeit-Datenstruktur gelöscht, um das augenblickliche Reaktivieren zu ermöglichen.
ACTIVATE	Reaktiviert ein zuvor deaktiviertes Modellteil. Bei der ACTIVATE-Operation bleiben nachfolgend die mit dem Vater-Knoten verknüpften Geometrieteile ebenfalls aktiv und werden daher weiterhin dargestellt (siehe auch Operation "SWAP_NODE").
DELETE_NODE	Alle in der Modellhierarchie unter der angewählten BK liegenden Detailebenen und Modellteile werden aus der Laufzeit-Datenstruktur gelöscht. Die Operation ermöglicht später das Nachladen anderer Modellteile.
SWAP_NODE	Diese Operation ermöglicht die Darstellung eines Bauteils in verschiedenen Detailstufen. Die in der Modellhierarchie tieferliegenden Modellteile werden aktiviert, jedoch im Gegensatz zur Operation "ACTIVATE" werden die Geometrie-Elemente des Vater-Datensatzes für die nachfolgende Darstellung deaktiviert. Umgekehrt erfolgt für einen bereits feiner detaillierten Modellzweig die Rückkehr in die nächstgröbere Detailstufe. Als Erweiterung dieses Befehls wurden für die SCRIPT-Kommandoschnittstelle noch die Operationen "LEVEL_UP" und "LEVEL_DOWN" implementiert. Die Umschaltung der Modellteil/Detailstufen-Aktivierung erfolgt dort für alle ABSTRACT-Knoten innerhalb der spezifizierten ABSTRACT-Datei.

Tabelle 2. Operationen zur Steuerung der Detaildarstellung

3.3 Geometrische und kinematische Modellierung mit KISMET

Eine wesentliche Voraussetzung für das grafische Überwachen (Monitoring) komplexer Arbeitsaufgaben ist die **Modellkonsistenz** zwischen rechnerinternem Anlagenmodell und der Realität. Im Monitor-Modus wird die Konsistenz für als ROBOT definierte Modellteile über Sensorschnittstellen zu den Roboter-Steuerungen und zum übergeordneten Kontrollsystem gewahrt. Die Konsistenz der Baugruppen-Topologie sowie der Modellgeometrie ist durch administrative Maßnahmen des Operateurs bei der Modellerstellung und während des Arbeitsablaufs sicherzustellen. Als Hilfsmittel zur Feststellung von

---

5 vgl. auch "Die KISMET-SCRIPT Kommandoschnittstelle" auf Seite 43 sowie "Zusammenfassung der SCRIPT-Kommandos" auf Seite 155

qualitativen Modell-Inkonsistenzen ist die realisierte **Echtzeit-Videobildunterlagerung** anzusehen.

Der in KISMET integrierte **3D-Modelleditor** gestattet die interaktive Eingabe und Modifikation von Modell-Geometrie, -Topologie und -Kinematik. Alle Modellkomponenten lassen sich über entsprechende **Operationen** relativ zueinander plazieren (Rotieren, Verschieben), löschen, kinematisch neu verknüpfen (PICK, PLACE) und in ihren Darstellungs-Attributen verändern. Aus Sicherheitsgründen werden die genannten Operationen zunächst auf dem rechnerinternen Darstellungsmodell ausgeführt. Permanente Änderungen der Modell-Datenbasis sind über explizite Schreiboperationen der ABSTRACT-Dateien durchzuführen. Die o.g. Operationen sind durchführbar für Geometrie-Elemente (GEO), ABS-FRAMEs sowie Arbeitspunkte (WFRM).

3.3.1 Allgemeine Anforderungen an die Datenarchitektur

Aufgrund der besonderen Anforderungen des Einsatzes für die grafische, prozeßgekoppelte Überwachung ("synthetisches Sehen") im Rahmen von hochflexiblen Handhabungssystemen (HFH) ergeben sich besondere Anforderungen an die System- und Datenarchitektur:

- Programmtechnische Begrenzungen bezüglich Modellgröße, Schachtelungstiefe und Detaillierungsebenen sollen vermieden werden.
- Kinematische Strukturen sollen in beliebiger Anzahl, beliebig tief verkettet und beliebig verzweigt definierbar sein.
- Die Modellierung von ebenen, kinematisch geschlossenen Ketten soll möglich sein (nicht beschränkt auf Parallelogramme).
- Das Modellieren von neuen Kinematikstrukturen soll kein Neu-Übersetzen des Programms erfordern.

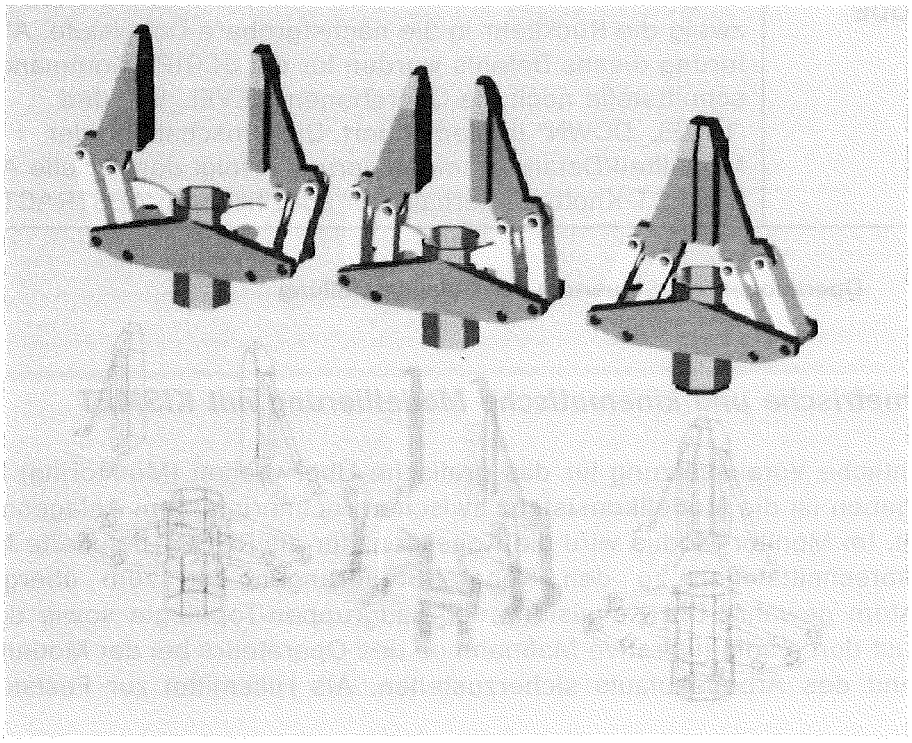


Abbildung 11. Modell einer Parallel-Greiferhand

3.3.2 Geometrisches Modell

Aus der CAD-Technik sind verschiedene Methoden der Geometrie-Modellierung und -Repräsentation bekannt geworden [62], [63]. Die am häufigsten anzutreffenden Verfahren sind die:

- **CSG-Modellierung** (Constructive Solid Geometry, [64], [65])
- **B-REP-Methode** (Boundary Representation), die sich wiederum in die Unterklassen B-REP *exakt* [66] und *facettiert* untergliedern läßt. Ebenso ist die Gruppe der *Freiformflächen-Modellierer* [67] den B-REP Systemen zuzuordnen
- **Zell-Dekomposition**. Dieser Gruppe ist die aus dem Applikationsfeld der NC-Programmierung und der Computer-Tomographie bekannte *Octree-Methode* [68]-[73] zuzurechnen.
- **Hybride-Modelliersysteme** kombinieren mehrere der o.g. Methoden. Die meisten kommerziellen CAD-Systeme lassen sich dieser Klasse zuordnen.

KISMET erlaubt die Definition komplexer Geometrien als Kombination einfacher **3D-Linien-, Flächen- und Volumen-Primitive** (Solids).

Jedes definierte Geometrie-Element (GEO) wird in der Geometrie-Bibliothek **mpolib** separat als eigene Datei gespeichert. Diese Dateien werden in der Baugruppen- und Kinematikdefinition als Bauteilreferenz benutzt.

In KISMET werden die folgenden Primitive verwendet:

```
SPHERE_FULL | CYLINDER_FULL | CONE_FULL | TRUNCATED_CONE |
BOX | ROTATIONAL_SWEEP | LINEAR_SWEEP | PIPE |
POLYHEDRON | EXTENDED_POLYHEDRON | ISOM_LINE | CURVE | SURF
```

Die Syntax der genannten Geometriedefinitionen (Dateiformat) ist detailliert beschrieben in "Anhang C. Geometrie-Datei Spezifikation".

KISMET gestattet interaktiv die Verwendung der **Booleschen CSG-Operationen** Vereinigung (Union), Differenz (Subtraction) und Schnittmenge (Intersection) für die Geometrie-Modellierung. Die Operationsergebnisse werden als Polyedermodelle abgespeichert. Komplexere GEO-Elemente, die sich nicht aus den einfachen Primitiven (Kugel, Kegel, Quader usw.) erzeugen lassen, können über die Primitive POLYHEDRON und EXTENDED_POLYHEDRON modelliert werden.

Während der Modelldaten-Initialisierung wird von KISMET aus den definierten Geometrie-Grundelementen zur Szenen-Darstellung ein facettiertes Oberflächenmodell (Polyedermodell) als internes Datenformat erzeugt. Für bestimmte Verarbeitungszwecke kann stets auf die ursprüngliche "exakte" Definition der Geometrie-Elemente zurückgegriffen werden. Die "exakte" Definition der Geometrie ist immer dann notwendig, wenn z.B. die analytische Gleichung der Oberfläche benötigt wird. Im weiteren Sinne kann KISMET daher als **B-REP-System** (Boundary Representation) klassifiziert werden.

In der internen Datenstruktur werden die GEO-Elemente durch ebene Polyeder angenähert beschrieben, wobei jedoch - im Gegensatz zu vielen anderen CAD/Grafik-Systemen - die "exakten" Oberflächen-Normalen an den zur Approximation der exakten Form gespeicherten Polygon-Eckpunkten mitverarbeitet werden; vgl. Abbildung 14. Aus dem letztgenannten Grund ist in KISMET das kontinuierliche, farbinterpolierte Schattieren von gekrümmten Geometrie-Oberflächen möglich. Diese Form der Schattierung wird in der Literatur als **GOURAUD-Shading** bezeichnet, benannt nach *H.Gouraud*, der diese Methode erstmalig vorstellte [74].

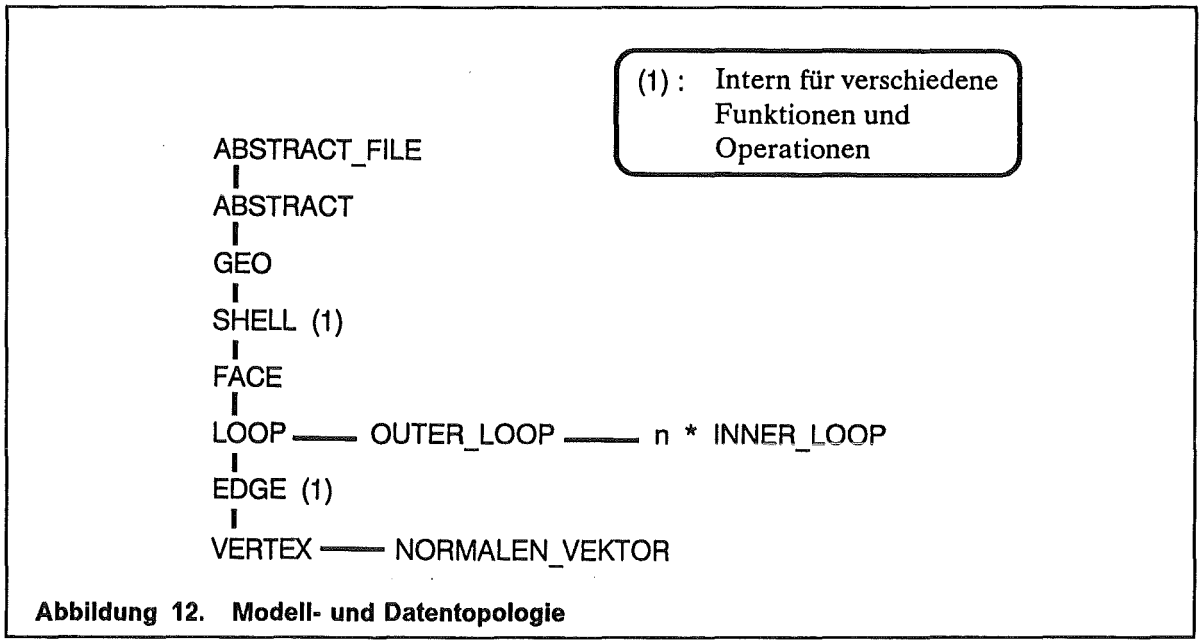


Abbildung 12. Modell- und Datentopologie

Während der **Modellaktivierung** wird die in Abbildung 13 dargestellte **Datenstruktur** im Hauptspeicher aufgebaut. Um eine logisch unbegrenzte Modellgröße zu erlauben, wird der Speicherplatz für alle Geometriedaten **dynamisch zugewiesen** und über **Zeigerstrukturen** von KISMET verwaltet.
Diese Datenstruktur ist für alle Geometrieprimitive einheitlich. Die Abbildung der verschiedenen CSG- und B-Rep-Primitive in das dargestellte **Polyeder-Datenmodell** erfolgt einmalig über entsprechende Funktionen (eine Funktion je Primitiv) während der Geometriedaten-Initialisierung, also nicht in jedem Darstellungszyklus. Jede Funktion erzeugt ein Oberflächenmodell, bestehend aus Punkt- und Normalenvektorlisten über die mathematisch einfachen, jedoch rechenintensiven Beziehungen der analytischen Geometrie. Dadurch war es möglich, die Darstellungsroutine, d.h. die eigentlichen Zeichenfunktionen, sehr kompakt zu implementieren. Die beschriebene Vorgehensweise erfordert einen erhöhten Hauptspeicherbedarf für das interne Polyedermodell, jedoch wird während der Modellabarbeitung innerhalb der Darstellungsroutinen ein erheblicher Geschwindigkeitsgewinn erzielt.

Die zur Umsetzung der Geometrieprimitive in das facettierte Geometriemodell erforderliche Mathematik zur Beschreibung der Oberflächengleichungen für die Basisprimitive (Kugel, Kegel, Zylinder, Quader usw.) ist ausführlich in der Arbeit von *Schuster* [75], und in den Standardwerken der grafischen Datenverarbeitung und des rechnergestützten Entwurfs (CAD) [59]-[62] dokumentiert.

Beschreibung der programminternen Polyeder-Datenstruktur
Für jedes in der '.mpc'-Datei der Baugruppe definierte GEO-Element wird die Datenstruktur *MPGEO* zugewiesen. Sie stellt einen **Eintrag** (*Instance*) des in einer '.mpo'-Datei

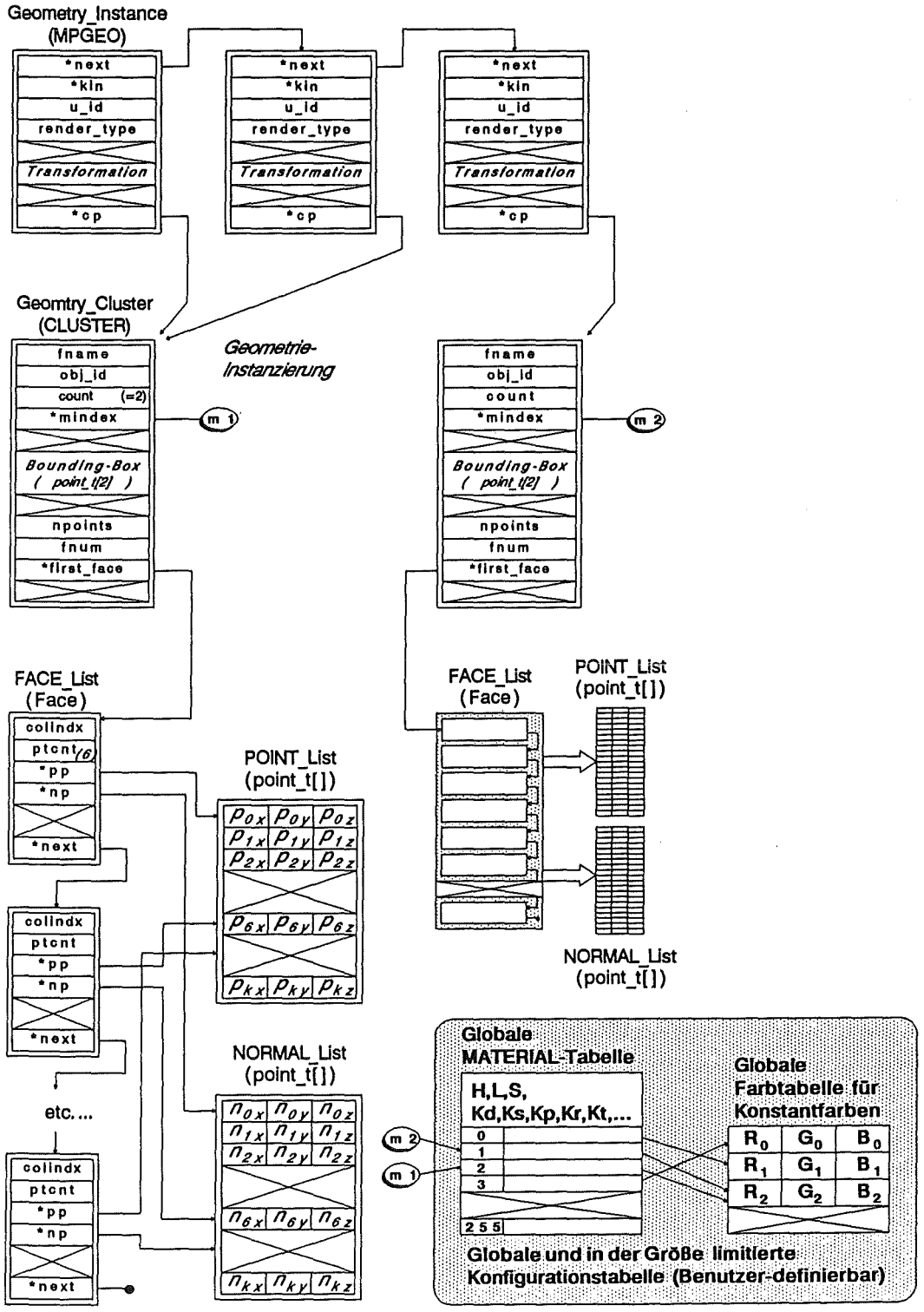
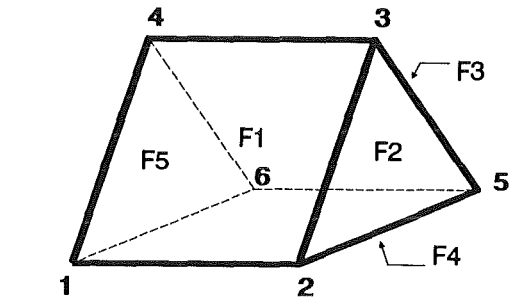


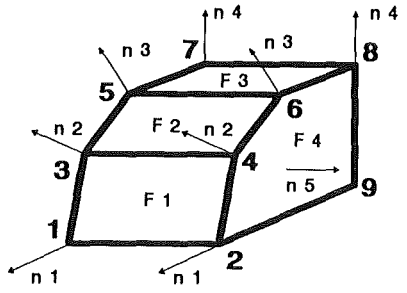
Abbildung 13. GEOMETRIE-Datenstruktur

definierten und in der Datenstruktur *CLUSTER* gespeicherten Geometriekörpers dar. Alle *MPGEO*-Strukturen sind als einfache Liste verkettet an der zugehörigen *ABSTRACT*-Struktur angehängt; vgl. auch Abbildung 10 auf Seite 24. Jede *MPGEO* erlaubt über eine Modellier-Transformation das beliebige Plazieren des GEO-Elements relativ zum *ABSTRACT-KS*.

Der Zeiger *first_face* in der *CLUSTER*-Struktur weist auf eine einfach verkettete Liste von *Face*-Elementen. Das *Face*-Element wird für jeden definierten Polygonzug, d.h. für Inner- und Outer-Loops angelegt und speichert die für die grafische Darstellung notwendigen Daten wie Eckpunktzähler (*ptcnt*), Farbindex (*colindx*), sowie die Parameter der Ebenengleichung der Facette. Der Attributspeicher *colindx* dient auch zur Markierung von "inneren Polygonschleifen" (*Inner-Loops*). Sie werden zur Definition der Randkonturlinien von Aussparungen ("Löchern") innerhalb einer ebenen Facette verwendet. Die Struktur *Face* letztendlich weist über die Zeiger *pp* und *np* auf die eigentlichen Geometriedaten, nämlich die Punktkoordinaten-Liste (PK-Liste) *POINT_List* und die Liste der Oberflächen-Normalen (ON-Liste) *NORMAL_List*.



Fi	Ni	Vertex-Liste
F1	4	1, 2, 3, 4
F2	3	2, 5, 3
F3	4	5, 6, 4, 3
F4	4	6, 5, 2, 1
F5	3	4, 6, 1



Fi	Ni	Vertex-Liste	Normalen-Liste
F1	4	1, 2, 4, 3	1, 1, 2, 2
F2	4	3, 4, 6, 5	2, 2, 3, 3
F3	4	5, 6, 8, 7	3, 3, 4, 4
F4	5	2, 4, 6, 8, 9	5, 5, 5, 5, 5
..	..	.....	.....

Abbildung 14. Interne Polyederdarstellung

Die PK- und ON-Listen werden für jedes *CLUSTER*-Element als zusammenhängender Speicherblock zugewiesen. Durch dieses Vorgehen lassen sich hohe Trefferraten im *Cache-Speicher*[6] der Rechner-Zentraleinheit (CPU) erzielen. Die PK- und ON-Listen werden relativ zum Modell-KS des GEO-Elements gespeichert, die Transformation in das Welt-KS erfolgt erst während des Zeichenvorgangs in der Matrizen-Transformationskette (Grafik-Pipeline) des Grafik-Subsystems.

---

6 Als *Cache-Speicher* wird eine Speichereinheit bezeichnet, die in einer speziellen Technologie hergestellt und eng mit der Zentraleinheit (CPU) des Rechners gekoppelt ist. Er dient als Puffer zwischen Hauptspeicher und CPU. Der *Cache-Speicher* hat im Vergleich zum eigentlichen Hauptspeicher eine geringere Größe, aber deutlich schnellere Zugriffszeiten. Die verwendete Hardware besitzt einen gesonderten *Cache* für Programmcode und Daten. Durch geeignete Strukturierung der Daten und geschickte Programmierung kann erreicht werden, daß ein Programm zu einem hohen Anteil im *Cache* abgearbeitet wird (hohe *Cache*-Trefferrate). Dadurch läßt sich eine hohe Verarbeitungsgeschwindigkeit erzielen.

Zweckmäßige Modellbildung zur optimalen Ausnutzung Grafiksystems
Die Definition von **Inner-Loops** (Innere Polygonschleifen) im CAD-Geometriemodell sorgt für Probleme, da das verwendete Grafik-Subsystem der Firma *Silicon Graphics* [76]-[77] diese in direkter Form nicht darstellen kann. Es war daher notwendig den im Aufsatz von *Chen* et al [78] skizzierten Algorithmus zur Eliminierung von *Inner-Loops* in adaptierter Form für KISMET zu implementieren.
Facetten die aus *Outer-Loop* und einer Anzahl von *Inner-Loops* definiert sind (vgl. Teilbild A in Abbildung 15) werden dabei so umgeformt, daß durch das Einfügen von Brückenkanten (engl. "Bridge Edge") ein zusammenhängender Polygonzug (*Non-self-intersecting Polygon*, NIP) entsteht; vgl. Teilbild B.

Da der in KISMET implementierte Algorithmus im Wesentlichen bereits im Aufsatz von *Chen et al* [78] dokumentiert ist und im Rahmen der vorliegenden Arbeit lediglich eine Anpassung an die Datenstrukturen von KISMET stattfand, soll hier auf dessen ausführliche Darstellung verzichtet werden.

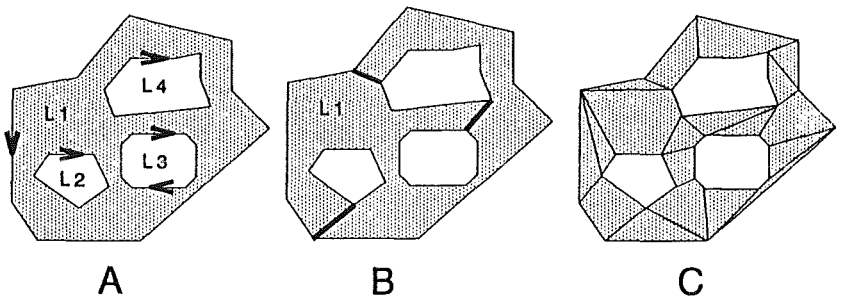


Abbildung 15. Behandlung nicht-konvexer Oberflächenfacetten

Nachdem durch das Einfügen von "Brückenkanten" auch Facetten mit *Inner-Loops* durch die Grafik-Subsysteme der *GT* und *Personal IRIS* Baureihen korrekt dargestellt wurden, ergab sich nach der Lieferung des ersten Systems der neuen, 1990 vorgestellten *VGX*-Baureihe das Problem, daß unter bestimmten Umständen Facetten mit *Inner-Loops* und eingefügten Brückenkanten nicht mehr korrekt dargestellt wurden. Ein weiterer wohlbekannter Effekt ist, daß für **nicht-konvexe POLYFACEs** das Grafik-Subsystem aller *IRIS-4D*-Workstations weit unter der von SGI angegebenen Maximalleistung arbeitet [76], [77].
Das Problem wurde deshalb für KISMET dahingehend gelöst, daß ein weiterer Algorithmus entwickelt wurde der nicht-konvexe POLYFACEs in **konvexe Teilstücke** zerlegt; vgl. Abbildung 15, Teilbild "C".

Die Implementierung des beschriebenen Algorithmus führte nun zu folgenden Resultaten:

1. Ursprünglich als *nicht-konvexe POLYFACEs* definierte Facetten werden für alle Betrachterstandpunkte korrekt dargestellt.
2. Wie aus Tabelle 3 ersichtlich ist, verbesserte sich interessanterweise durch die Zerlegung die Grafikleistung deutlich. Für Modelle mit einem hohen Anteil an nicht-konvexen Facetten wird eine Leistungssteigerung um den Faktor 2-3 erreicht.
3. Tabelle 4 zeigt, daß die interne Modellgröße und der Speicherplatzbedarf durch das Einfügen von konvexen Facetten ansteigt.

Modell	70-GT nicht unterteilt	210-VGX nicht unterteilt	210-VGX unterteilt
GREIFER	2	8-9	20-30
KUKA15	3	8-10	20-30
KUKA400	4-5	9-10	15
SUPPVEH	1-2	4-5	8-9

Tabelle 3. Leistungsdaten mit und ohne nicht-konvexe Polygone: Erzielte Bildraten mit und ohne Unterteilung von nicht-konvexen POLYFACES auf IRIS 4D 70-GT and 210-VGX Systemen.

Die in der Testreihe benutzten KISMET-Modelle GREIFER (Parallel-Backengreifer für EMSM-Systeme), KUKA15 (*Kuka*-161/15 Industrieroboter) und KUKA400 (6-Achs-Portalroboter aus dem Teststand COMETOS im HT-Labor des KFK) wurden über *CAD*I*-Prozessoren[7] aus dem Off-Line Programmiersystem *ROBCAD* in KISMET importiert (lediglich Geometrieteile). Das Modell SUPPVEH (JET Ground Support Vehicle der Fa. *Blocher*) wurde ebenfalls über *CAD*I*-Prozessoren aus dem CAD-System *CATIA* übertragen. Die Modelle GREIFER und KUKA15 weisen einen hohen Anteil an nicht-konvexen POLYFACEs und *Inner-Loops* auf.

Modell	FACETTEN		PUNKTE		SPEICHER	
	ohne	mit	ohne	mit	ohne	mit
GREIFER	581	1.356	2.824	2.824	150.212	288.584
KUKA15	728	1.305	3.536	3.536	166.586	273.970
KUKA400	1.590	1.894	7.584	8.838	333.084	403.944
SUPPVEH	4.380	5.170	15.497	21.398	543.068	687.194

Tabelle 4. Modellgrößen: Vergleich von Modellgröße und Speicherbedarf (in Bytes) mit und ohne Unterteilung von nicht-konvexen Polygonen.

Die Leistungssteigerung ist vor allem dadurch zu erklären, daß das Grafik-Subsystem für nicht-konvexe POLYFACEs die Zerlegung in konvexe Teilstücke bei jedem Bilddurchlauf durchführen mußte. Dieser Zeitbedarf wird von KISMET jetzt einmalig während der Modell-Initialisierung aufgebracht.

3.3.3 Kinematische Modellierung von Anlagenkomponenten

Ein universell einsetzbares System für die grafische Bedienerführung muß die Modellierung aller in der handhabungstechnischen Praxis auftretenden Anlagenkomponenten gestatten. Hierbei lassen sich **funktionelle Strukturen** (FS) von der Detaillierungsstruktur unterscheiden, die lediglich eine Verfeinerung der FS darstellt.

---

7 Siehe auch "CAD-Modelldatentransfer" auf Seite 36

3.3.3.1 *Modellstruktur und Baugruppen-Klassifikation*

Innerhalb der funktionellen Anlagenstruktur lassen sich folgende, für die Modellierung relevante Objektklassen unterscheiden:

Bewegliche (dynamische) Komponenten
Zu den beweglichen Modellkomponenten lassen sich diejenigen Objekte zählen, deren interne kinematische Konfiguration über definierte Bewegungsachsen dynamisch veränderbar ist. Hierzu gehören die Arbeitsgeräte (Transportsysteme wie Fahrzeuge und Krananlagen; Manipulatoren und Roboter; Kamerasysteme an beweglichen Armen oder mit Schwenk/Neigeköpfen), End-Effektoren und Werkzeuge, aber auch für den Handhabungsablauf relevante, bewegliche Teile der Arbeitsumgebung (Schiebetüren, Werkzeugwechselvorrichtungen usw.). Alle diese Komponenten werden hier mit dem abstrakten Begriff *ROBOT* umschrieben. Eine Untergruppierung dieser Objektklasse erfolgt nach dem implementieren Funktionsumfang zur **Steuerung der Modellfreiheitsgrade**. Unterschieden werden hier:

- ROBOT-Objekte die eine Führung in Werkzeugkoordinaten gestatten, d.h. die eine *Lösung des inversen kinematische Problems* erlauben, und
- ROBOT-Objekte die lediglich eine *Veränderung der Modell-Freiheitsgrade in Achskoordinaten* gestatten.

Alle ROBOT-Komponenten lassen sich in KISMET über Sensordaten (Gelenkparameter) aus dem Handhabungsprozeß dynamisch verändern. Diese Modell-Objekte können mit einem TCPF (für die Steuerung in Werkzeugkoordinaten obligatorisch) und einem ZPF (relevant für die Nullpunktverschiebung bei programmierbaren Komponenten) ausgestattet sein. Bewegliche Sichtpositionen für die Simulation von Szenenkameras werden mit diesen Komponenten über die CAMERA-Datenstruktur kinematisch verknüpft.

Statische Komponenten
Zu dieser Objektklasse gehören all diejenigen Objekte der Handhabungszelle, deren Plazierung, Konfiguration oder geometrische Gestalt durch den Handhabungsablauf nicht beeinflußt wird. Diese Komponenten sind innerhalb des Simulationsmodells jedoch relevant für den Kollisionsschutz, oder dienen zur Orientierung des Operateurs in der synthetischen Szene.

Quasistatische Komponenten
In diese Gruppe sind Komponenten einzuordnen, deren Plazierung zwar durch den Arbeitsablauf verändert wird, die aber in ihrer geometrischen Gestalt oder kinematischen Konfiguration unverändert bleiben. Hierzu können Hantierungswerkzeuge gehören und Wechselkomponenten der Anlage. Objekte dieser Klasse sind mit einem *Workframe* (WFRM) zur Kennzeichnung der Werkzeug-Koppelposition, oder - im Fall der passiv hantierten Objekte - zur Markierung der Greifposition bzw. zum Anschlagen der Last gekennzeichnet.

3.3.3.2 *Kinematische Modellier-Primitive*

Für die Modellierung der kinematischen Strukturen und der Baugruppen-Topologie der o.a. Anlagenkomponenten wird in KISMET die in Tabelle 5 aufgeführte Gruppe von ABS-FRAME-Primitiven benützt. Je nach Primitiv-Typ wird der Ort innerhalb der kinematischen Struktur über eine Referenz auf das *kinematische Vorgänger-KS* (für die Typen 0-1, und 50-51), sowie bei FRAMEs die zur Modellierung von ebenen, geschlossen kinematischen Ketten verwendet werden (Typen 2-3 und 52), zusätzlich durch eine Referenz auf das *kinematische Nachfolge-KS* festgelegt.

Bei beweglichen FRAMEs (Gelenken) ist zu unterscheiden zwischen Elementen zur Definition einfacher Modell-Freiheitsgrade (Typen 0-1) und Element-Primitiven, die zur Modellierung von Bewegungsgelenken innerhalb geschlossener, kinematischer Ketten (Typen 2-3 und 52) verwendet werden. Die letztere Gruppe von Freiheitsgraden wird durch ein weiteres FRAME, dem **kinematischen Nachfolge-KS** zwangsgeführt. Der aktuelle Parameter dieser Gelenkbausteine wird von KISMET automatisch während eines Bewegungsintervalls neu berechnet und ergibt sich aus der Position der kinematischen Vorgänger- und Nachfolge-FRAMEs.

Jedes FRAME-Element wird im programminternen Darstellungs-Modell in einem Element der *MPKIN*-Datenstruktur verwaltet. Bei beweglichen FRAMEs, sogenannten **passiven Wirkelementen** (FRAME-Typen 0-4), wird außerdem ein Element der *MPDAT*-Datenstruktur zugewiesen, die zur Speicherung der Laufzeit-Parameter des Modell-Freiheitsgrades (aktuelle Gelenkstellung, Referenzposition, sowie aktuelle und Maximal-/Minimal-Geschwindigkeiten und -Beschleunigungen) dient.
Die zur Berechnung der Vorwärts- und Rückwärts-Transformationen von kinematischen Ketten und Bäumen, sowie die zur Behandlung von geschlossenen Ketten angewandten Lösungsverfahren werden in " 4. Kinematische Strukturmodelle und Lösungsverfahren" auf Seite 65 näher behandelt.

0	Rotatorisches Gelenk innerhalb einer offenen kinematischen Kette.
1	Translatorisches Gelenk innerhalb einer offenen kinematischen Kette.
2	Rotatorisches Gelenk mit kinematischer Zwangsführung. Dieses Element wird innerhalb von geschlossenen Ketten verwendet.
3	Translatorisches Gelenk mit kinematischer Zwangsführung für die Verwendung innerhalb von geschlossenen Ketten.
50	Modellier-FRAME zur Definition von Baugruppen-Bezugssystemen oder zur Definition der Bewegungsrichtung von bewegten Achsen.
51	Modellier-FRAME. Kann zur Kennzeichnung von quasistatischen Modellkomponenten benützt werden.
52	FRAME zur Modellierung von skalierbaren Objekten in offenen Ketten (z.B. ein Kranseil). Alle damit verknüpften GEO-Elemente werden skaliert mit dem DOF-Parameter des kinematischen Vorgänger-FRAMEs, das als prismatisches Element (ID = 1) modelliert sein muß.
53	FRAME zur Modellierung von skalierbaren Objekten in geschlossenen Ketten (z.B. ein an zwei Enden eingespanntes Gummiseil). Der Skalierungsfaktor wird berechnet als Abstand zwischen FRAME-Ursprung und dem Ursprung des Nachfolge-FRAMEs. Das FRAME wird bei der Bewegung des Nachfolge-FRAMEs so gedreht, daß die Z-Achse auf dessen Ursprung weist.

Tabelle 5. ABS-FRAME Elemente zur kinematischen Modellierung

3.3.3.3 Getriebemodellierung und höherwertige Gelenke

In **DOF-Dateien** ('.dof') - DOF steht für 'degrees_of_freedom' bzw. 'Freiheitsgrade' - werden für die in KISMET definierten kinematischen Modelle Anzahl und Funktionstypen der über entsprechende Eingabekanäle angetriebenen Modell-Freiheitsgrade definiert. Als Eingabekanäle können alternativ oder gleichzeitig Maus-Valuatoren, die von einer Robotersteuerung empfangenen Positions-Sensordaten, Eingabedaten von der

"Dials-and-Buttons Box", oder der Sensorkugel (auch "Robot-Teach Ball" genannt) verwendet werden. Derartige **1-zu-n** Antriebsfunktionen können z.B. definiert werden zur Simulation von Getrieben oder anderen analytisch erfaßbaren mechanischen Strukturen.

Der implementierte DOF-Algorithmus erlaubt die Definition von Abhängigkeiten entweder in der Form

$$\Phi_i = f_d(u_j, k_0 \ldots k_n) \tag{3.1}$$

d.h. die Positionsvariable Φ_i des i-ten Gelenks wird berechnet als Funktion eines Parameters u_j und einer implizit durch die Abbildungsvorschrift $f_d(\,)$ vorgegebenen Anzahl $(n+1)$ von Konstanten $k_0, \ldots, k_n$, oder in der Form

$$\Phi_i = f_d(\Phi_F, k_0 \ldots k_n) \tag{3.2}$$

Hier wird die Gelenkvariable Φ_i als Funktion einer anderen Gelenkvariablen Φ_F berechnet. Der Modell-Freiheitsgrad mit der Variablen Φ_F soll im folgenden als das **Führungsgelenk** bezeichnet werden.

Eine Liste der in KISMET implementierten Funktionen ist in "Anhang B. Liste der DOF-Funktionstypen" auf Seite 122 aufgeführt.

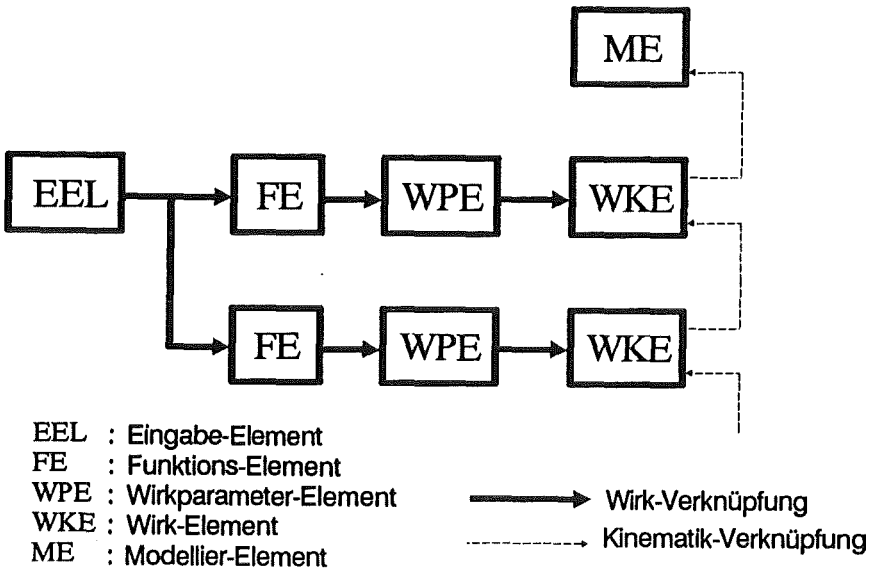


Abbildung 16. Kinematisches Wirksystem

Abbildung 16 zeigt die Struktur des in KISMET realisierten kinematischen **Wirksystems**. Über das *Eingabe-Element* (EEL) wird der kinematischen **Wirkstruktur** die Eingabevariable u_j zugeführt. Diese wird über *Funktions-Elemente* (FE) mit der zugehörigen, definierten *Wirkfunktion* $f_d(u_j, k_0 \ldots k_n)$ nach Gleichung (3.1) in die Gelenkvariable Φ_i umgesetzt, und in das *Wirkparameter-Element* (WPE) des Gelenks als aktueller Gelenkparameter eingetragen. Während der Berechnung der kinematischen Kette, wird dieser Gelenkparameter Φ_i als Freiheitsgrad zur Erzeugung der *relativen Gelenktransformation* innerhalb des *Wirk-Elements* (WKE) des kinematischen Gelenks verwendet. *Modellier-Elemente* (ME) werden zur Definition der Drehachse verwendet. Sie besitzen keinen Gelenkparameter in Form eines WPE und werden daher innerhalb der kinematischen Struktur ausschließlich passiv bewegt.

3.3.4 Modellerzeugung und neutrale CAD-Schnittstellen

Obwohl KISMET über den integrierten Geometrie-Editor die Möglichkeit zur geometrischen Modellierung bietet, empfiehlt sich die Erzeugung der Simulations-Modelldatenbasis auf einem 3D-CAD System. Als Gründe sind hierzu zu nennen:

- Die Modellkonsistenz zwischen Anlagenbau, der auf dem CAD-System erfolgt, und der Simulation sollte erhalten bleiben.
- Der in KISMET integrierte Geometrie-Modellierer bietet lediglich einen beschränkten Funktionsumfang.
- Die Echtzeit-Grafikworkstation sollte nicht durch zeitraubende Modellierarbeit blockiert werden.
- Doppelte Arbeit durch erneute Modell-Konstruktion soll vermieden werden.

3.3.4.1 CAD-Modelldatentransfer

Für den CAD-Datenaustausch sind grundsätzlich zwei unterschiedliche Systemarchitekturen

1. Datenaustausch über **Dateien** in einem neutralen, d.h. herstellerunabhängigen Dateiformat; vgl. Abbildung 17. Für jedes am Datenaustausch beteiligte System existiert ein Paar von Prozessoren (Pre- und Postprozessor). Der Preprozessor setzt das systemspezifische interne Datenmodell des Quellsystems in das neutrale Austauschformat um. Der Postprozessor wiederum übersetzt das neutrale Dateiformat in das interne Datenmodell des Zielsystems.
   Dieses Konzept eignet sich besonders für den CAD-Datentransfer zwischen Systemen über größere Entfernungen, bzw. falls Quell- und Zielsystem auf verschiedenen Rechnersystemen installiert sind. Als physikalische Übertragungsmedien Datenträger (Diskette, Magnetband, Datenkassette) oder Mittel der elektronischen Datenfernübertragung (Lokale Netzwerke, elektronische Post) genutzt. Als Nachteil wird oft die erhebliche Übersetzungszeit empfunden.

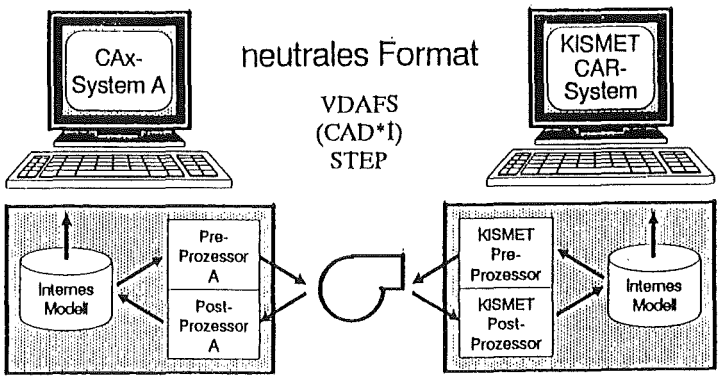


Abbildung 17. CAD-Datentransfer mittels Dateitransfer

2. Datenaustausch über eine gemeinsame **Datenbank**; vgl. auch Abbildung 18. Der Zugriff der verschiedenen am Datenaustausch beteiligten Systeme erfolgt hier über prozedurale Schnittstellen des Datenbanksystems. Das Datenbanksystem verwendet zur Definition der Strukturen das *Informationsmodell* des neutralen Formats.

   Das Konzept ist besonders geeignet für multidisziplinäre Systeme wie z.B. die Kombination von CAD-Systemen, Finite-Elemente-Methode (FEM) Programmen, CAR-Systemen (KISMET) und Systemen zur Simulation und Analyse der Dynamik

mechanischer Systeme. Charakteristisch für den Datenaustausch zwischen den Teilmodulen im multidisziplinären System ist, daß jedes Subsystem lediglich einen Teilumfang der gesamten Datenbank benötigt oder in die Datenbank einbringt. Der physikalische Datentransfer erfolgt hier - falls alle Teilsysteme auf demselben Rechnersystem installiert sind - über Verfahren der Interprozeßkommunikation, oder - falls die Teilsysteme auf verschiedenen Rechnern ablaufen - über lokale Netzwerke (LAN). Ein solches System wird z.B. im Rahmen des ESPRIT-II Projekts 5524 *"High-Performance Computing for Multidisciplinary Dynamic Simulation"* (MDS) entwickelt.

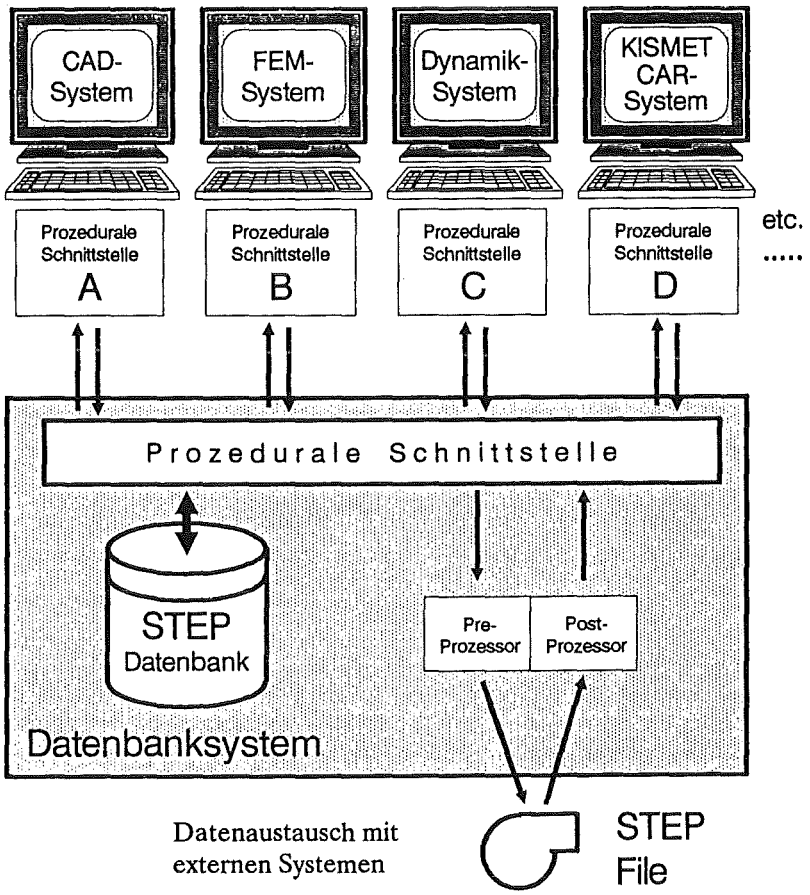


Abbildung 18. CAD-Datentransfer mittels Datenbank

Im Rahmen des ESPRIT Projekts 322 (*CAD*I*) wurde die Entwicklung der neutralen Schnittstelle *CAD*I* [79], [80] für den Transfer von CAD-Geometriemodellen zwischen CAD-Systemen durchgeführt. Die Schnittstellendefinition von *CAD*I* berücksichtigt die Modelliermethoden B-REP-approximiert (Polyedermodelle), B-REP-exakt, CSG und Freiformflächen (Bezier-Flächen). Am Institut und bei den beteiligten Projektpartner wurden bisher zahlreiche Pre- und Postprozessoren für *CAD*I* implementiert [81].

Für KISMET wurde ein *CAD*I* Postprozessor entwickelt [82], über den bisher Simulationsmodelle von CAD-Systemen wie *CATIA, EUCLID, TECHNOVISION* und vom Roboter-Programmiersystem *ROBCAD* transferiert wurden. Der Implementierungsstand des Postprozessors erlaubt den Datentransfer für Polyeder-Geometriemodelle (inklusive Baugruppenstruktur, Instanzierung, lokale Modellier-Transformationen). Die Definition

der kinematischen Struktur muß nach wie vor in KISMET erfolgen, da *CAD*I* keine Möglichkeit zur Übertragung von kinematischen Definitionen oder Mechanismen gestattet. Das in Abbildung 4 auf Seite 5 dargestellte Modell der JET Ex-Vessel Arbeitsumgebung wurde zum großen Teil in *IBM-CATIA* modelliert und über die neutrale *CAD*I* Schnittstelle in das Simulationssystem KISMET übertragen.

Da im KFK vorwiegend das CAD-System *BRAVO3* (Schlumberger) für mechanische Konstruktions- und Modellierarbeiten benutzt wird, wurde schon vor der Verfügbarkeit der Prozessoren ein unidirektionaler CAD-Modelldatentransfer realisiert. Das zu diesem Zweck entwickelte Programm *ROBOT2* [83] erweitert *BRAVO3* zur Modellierung von Mechanismen und erzeugt direkt (ohne neutrales Zwischenformat) die KISMET Eingabe-Datenstruktur (KISMET benutzt ausschließlich ASCII-Dateien für die geometrische und kinematische Modelldefinition). Die in Abbildung 5 auf Seite 6 und Abbildung 6 auf Seite 7 dargestellten Modelle wurden auf diese Weise erzeugt und in KISMET transferiert.

Um den Import von Freiformflächen und Raumkurven zu ermöglichen, wurde für KISMET im Rahmen dieser Arbeit ein Postprozessor für VDA-FS Geometrieteile (nach VDA-FS 1.0) entwickelt. Das in Abbildung 19 dargestellte Freiformflächen-Modell einer LKW-Hinterachse wurde auf dem CAD-System *PROREN 2* erzeugt und über die VDA-FS Schnittstelle in die entsprechende KISMET-Datenstruktur übertragen.

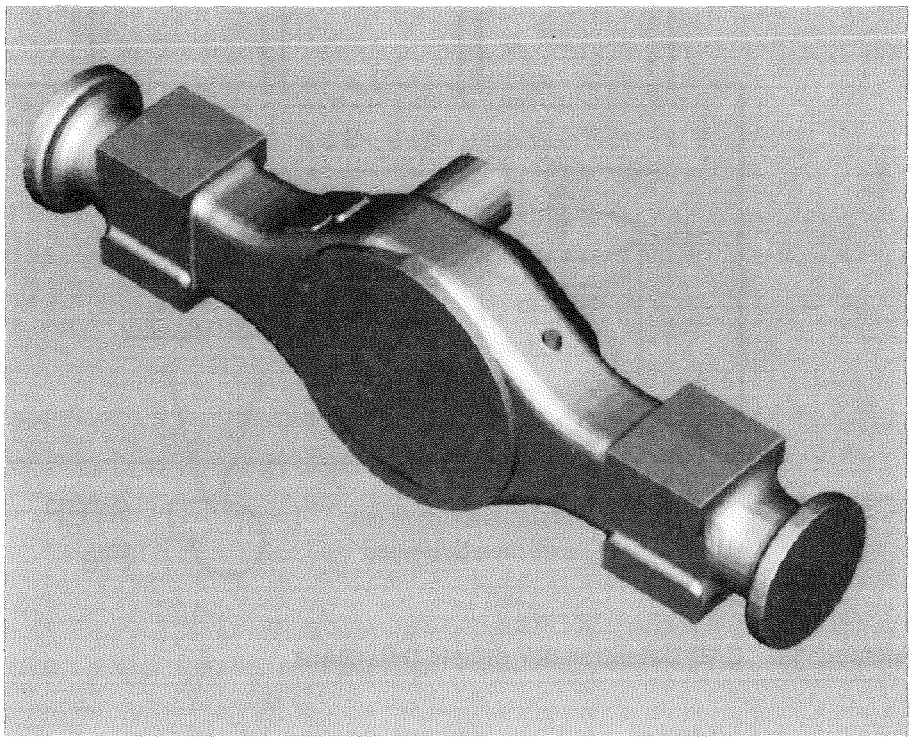


Abbildung 19. Freiformflächenmodell einer LKW-Hinterachse: Das CAD-Geometrieteil wurde über die VDA-FS Schnittstelle in das interne Datenmodell von KISMET übertragen.

Die Praxis der Benutzung von CAD-Modellen für die Realzeit-Robotersimulation ist jedoch mit Problemen behaftet:

- Aus der Konstruktion stammende CAD-Datenbestände lassen sich für Simulationsmodelle nicht direkt benutzen. Die Daten sind zu detailliert und liegen fast ausschließlich als 2D-Zeichnungen vor. Geeigneter sind hier 3D-Volumen- oder Oberflächenmodelle, die für eine mechanische Strukturanalyse bzw. für FEM-Berechnungen erzeugt wurden.

- Die für eine hohe Darstellungsgeschwindigkeit im Simulationssystem notwendige geometrische Modelloptimierung ("so wenig wie möglich, so viel wie nötig") erfordert vom Konstrukteur ein hohes Maß an Verständnis für die interne Arbeitsweise des Ziel-Simulationssystems und daher eine höhere Qualifikation, als für die Bedienung von CAD-System im Rahmen der Konstruktion üblich ist. Der Konstrukteur muß am CAD-System zielgerichtet für die grafische Realzeitsimulation Datenmodelle erzeugen.

- Im augenblicklichen Entwicklungsstand ist in KISMET noch erhebliche Nacharbeit erforderlich, um die übertragenen Modelle an die spezifischen Erfordernisse der Echtzeit-Simulation anzupassen.

Der zukünftige internationale Standard für den Transfer von Produktmodellen STEP soll gemäß der Zielsetzung auch die Übertragung von Kinematik-Daten gestatten. Er wird daher einhergehend mit der Entwicklung leistungsfähiger Pre- und Postprozessoren den Modelldatentransfer für die grafische Robotersimulation vereinfachen.

3.4 Systemarchitektur für synthetisches Sehen

3.4.1 Hardwarekonfiguration

Um das Entwicklungsziel der Echtzeitgenerierung von schattierten Szenenansichten realisieren zu können, wurde in einer frühen Phase der vorliegenden Arbeit eine Marktuntersuchung über Architekturen von verfügbaren Rastergrafiksystemen durchgeführt. Die damaligen Ergebnisse bezüglich der erforderlichen Rechenleistung und des Preis-/Leistungsverhältnisses verschiedener Systeme wurden durch die schnelle Fortentwicklung der Leistungsdaten überholt, die grundsätzlichen Überlegungen sind jedoch nach wie vor aktuell. *Akelay* diskutiert in dem Aufsatz [77] die Vor- und Nachteile der möglichen Grundkonzepte.

Zwei Grundarchitekturen kamen als Implementierungsplattform in Frage:

1. **Hostrechner-gestützte Systeme** mit einer klaren Gerätetrennung zwischen einem Universalrechner, auf dem die eigentliche Applikation abläuft, und einem Grafikterminal bzw. einer Grafikworkstation, die vom Hostrechner geladene Displaylisten verwaltet und darstellt. Als typisches Beispiel dieser Kategorie soll hier die Kombination von *DEC-VAX* Hostrechner mit dem *Evans & Sutherland PS-390* Grafiksystem angeführt werden. Diese Kombination bietet beeindruckende Leistungsdaten für die Darstellung von geometrisch und topologisch invarianten Szenen mit variablem Betrachterstandpunkt und Blickrichtung. Die in der Handhabungstechnik typischen Werkzeugwechsel (Änderung der Transformationshierarchie) oder der Geometrieteile (Schneiden von Rohren) erfordern hier jedoch eine Neugenerierung der Displaylisten auf dem Hostrechner und den Transfer von erheblichen Datenmengen in den Grafikteil. Daher wird bei dieser Architektur die Schnittstelle zwischen Host-

rechner und Grafikgerät zum leistungsbestimmenden Flaschenhals des Gesamtsystems. Als weiterer Nachteil dieser Architektur ist zu werten, daß die für eine Kollisionsrechnung erforderlichen Geometriedaten nach wie vor im Universalrechner gehalten werden müssen (neben den Displaylisten im Grafikteil) und damit zumindest doppelte Datenhaltung erfordern.

2. **Komplette Grafikworkstations**, wobei Zentraleinheit und Grafik-Subsystem eine Baueinheit bilden und über ein Hochgeschwindigkeits-Bussystem miteinander verbunden sind. Die Verwaltung von Displaylisten oder der Geometrie- und Topologie-Datenstrukturen erfolgt hier im Hauptspeicher der Zentraleinheit. Der Aufsatz von *Braun* [84] liefert eine aktuelle Übersicht über Leistungsdaten und Subarchitekturen derartiger Systeme. Der Bericht von *Levin* [85] berichtet über aktuelle Entwicklungstrends auf dem oberen Marktsegment, den *Super-Grafikworkstions*, sowie deren Einbindung in eine *Supercomputer*-Umgebung.

Die Implementierung des JET "In-Vessel"-Monitorsystems *GBSim* (vgl. Abbildung 2 auf Seite 3) erfolgte auf einer Grafikworkstation *IRIS 3020* der Firma *Silicon Graphics*; vgl. dazu [86]. Für die Fortentwicklung des grafischen Universal-Monitorsystems *KISMET* wurde die Nachfolgeserie *IRIS-4D* als Plattform gewählt. Die Implementierung beider Softwarepakete erfolgte vollständig in der Programmiersprache **"C"** unter Benutzung der **Graphics Library** (GL) von *Silicon Graphics* als Grafiktreiber.

Tabelle 6 zeigt eine Gegenüberstellung der Leistungsdaten der bisher im Rahmen der Arbeit verwendetetn Workstations. Es handelt sich dabei um Optimaldaten; je nach Datenumfang (Cache-Trefferrate), Darstellungsqualität (Zahl der Lichtquellen, benützte Schattierungsinterpolation) und Betrachterstandpunkt ergeben sich häufig weit geringere Leistungsdaten.

Work-station	CPU	Bildspeicher		Transformationen		
	MIPS MFLOPS	Auflö-sung	Bit je Pixel	3D-Vekt/s	Polyg./s	Füllrate MPixel/s
3020	1.2 0.24	1024x768	40	86300	9400	40
4D-50/GTB	7 0.7	1280x1024	80	300000	25000 100000	40
4D-70/GT	10 1.1	1280x1024	96	400000	40000 120000	40
4D-80/GTB	13 1.5	1280x1024	80	400000	55000 135000	40
4D-210/VGX	20 3.7	1280x1024	140-268	1000000	250000 1000000	80
Anmerkung: Die Leistungsdaten in MIPS basieren auf dem DHRYSTONE-Benchmark, die Floating-Point Leistungsangaben basieren auf dem LINPAK-Benchmark (Double-Precision). Die Angaben für Polygone/s bezieht sich auf 4-seitige, schattierte Polygone der Größe 10x10 Pixel.						

Tabelle 6. **Leistungsangaben der Grafik-Workstation:** Die Leistungsangaben basieren auf Herstellerangaben laut Datenblättern.

Abbildung 20 zeigt die Systemarchitektur des CAT-Systems für die In-Vessel Handhabung bei NET/ITER. Die Grafik-Workstation (GWS) ist über ein lokales Netzwerk mit den

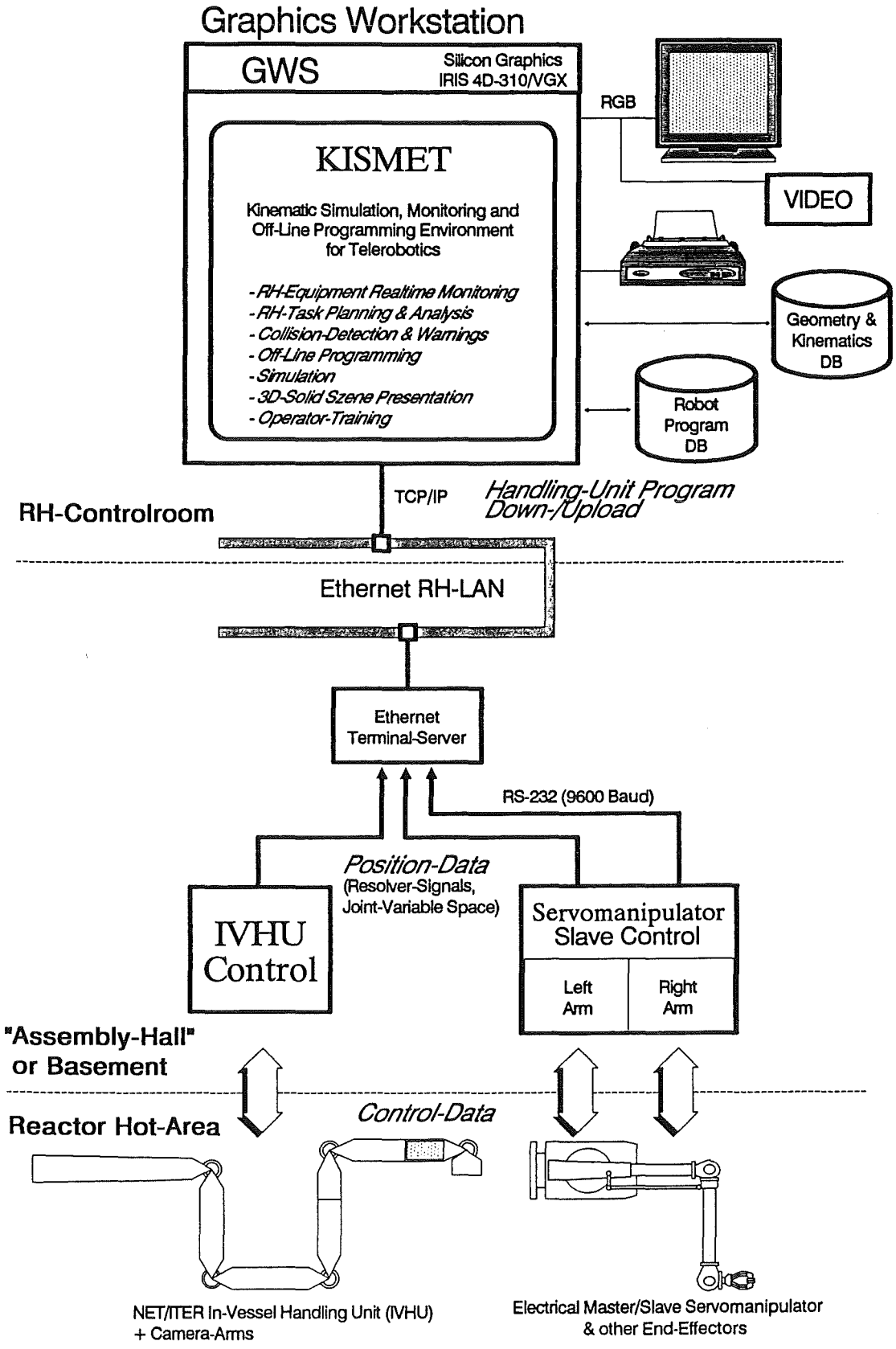


Abbildung 20. Architektur eines grafischen Monitor-Systems

Robotersteuerungen für das Transportersystem (IVHU, Boom) und der Arbeitseinheiten dargestellt. Als Arbeitseinheit ist hier der Master/Slave-Manipulator dargestellt. Aus Gründen der Betriebssicherheit besitzt jeder Manipulatorarm eine eigene Steuereinheit.

Im Monitor-Betrieb erfaßt KISMET die Gelenkwinkel-Sensordaten quasiparallel von den drei Steuerungen. Da im Fall des IVH-Systems bei JET die Ausgabe der Sensordaten über RS-232 Schnittstellen erfolgt und die Steuerungen nicht über eine direkte Kopplung zum Ethernet-LAN verfügen, wird die Umsetzung in das TCP/IP LAN-Protokoll über einen Terminal-Server realisiert. Die Bandbreite der Sensorschnittstelle wird daher weitgehend durch die serielle RS-232 Schnittstellen limitiert.

Um den Schleppfehler zwischen Master- und Slave-Arm zu eliminieren (das grafische Monitor-System soll die Position der Slave-Arme darstellen), erfolgt die Erfassung der EMSM-Sensordaten an den Steuermodulen der Slave-Arme. Zur Übung der Manipulator-Operateure kann die Kopplung auch zur Master-Steuerung erfolgen (Umstecken der Leitungen), wobei dann der Slave-Manipulator nicht benötigt wird. Diese Betriebsweise wird bei JET zeitweilig für das Operateurtraining benutzt, wobei das Mockup-Modell hier durch das synthetische Modell ersetzt ist, der Bediener aber dennoch die für den realen Betrieb gewohnten Bedienelemente (EMSM-Master) zur Verfügung hat.

3.4.2 Sensorschnittstellen und Softwarekriterien

Ein Problem für die praktische Realisierung des synthetischen Sehens ergibt sich aus der asynchronen Betriebsweise von Gelenkwinkel-Sensordatenerfassung und dem Monitor-System (GBSim, KISMET). Die Sensordatenerfassung erfolgt in der Robotersteuerung und ist daher mit deren Taktrate gekoppelt. Die mögliche Taktrate des Grafiksystems hingegen ist wesentlich abhängig von der gewählten Darstellungsqualität, dem aktivierten Modellumfang (Detailstufe und aus der Datenbasis geladene funktionale Gruppen) und der Zahl der gleichzeitig dargestellten Szenenansichten.

Um **Totzeiten** in der gesamten Übertragungsstrecke zwischen Roboter und visueller Erfassung durch den Bediener möglichst zu reduzieren, bedarf die Realisierung der Sensordaten-Schnittstelle besonderer Aufmerksamkeit. Die gesamte Totzeit T_G des Systems ergibt sich aus der Totzeit der Sensordatenerfassung T_E, der Übertragungszeit T_U und der Zeit T_B, die für die Bildberechnung und Ausgabe benötigt wird, zu:

$$T_G = T_E + T_U + T_B \tag{3.3}$$

Wird die Übertragungszeit T_U relativ groß im Verhältnis zu T_B, so hat dies negative Auswirkungen für die Bandbreite des gesamten Regelkreises zwischen Mensch und Mechanik. Dies soll am Beispiel der Implementierung von *GBSim* für den *Articulated Boom* (IVHU) bei JET näher erläutert werden.

Die Steuerung des "Articulated Boom" (IVHU)[8] bei JET war bei Aufnahme der Arbeit bereits vorhanden und deren Eigenschaften daher vorgegeben. Die Systemkopplung war so auszuführen, daß der Übergang von und zum Monitor-Modus **zu jedem Zeitpunkt** möglich ist. Die Sensordatenübertragung von der IVHU-Steuerung zum Terminalserver erfolgt über eine bitserielle RS-232 Schnittstelle. Je Datenblock werden die Gelenkwinkel der 19 IVHU-Freiheitsgrade (ein 16-Bit Wort je Gelenk) sowie ein Synchronisationswort

---

8 Die IVHU-Steuerung wurde von der Firma *Cambridge Consultants Ltd.* als Einzelauftrag für JET entwickelt. Sie basiert auf einer *General Electric* SPS-Steuerung. Der Kommunikationsteil ist in *BASIC* implementiert.

(16-Bit) übertragen. Das Synchronisationswort ermöglicht die eindeutige Zuordnung der Sensordatenwerte zu den einzelnen IVHU-Gelenken. Je Datenblock werden daher 400 Bit gesendet, wobei sich mit einer Datenrate von 4800 Baud eine Übertragungszeit von ca. 83ms ergibt. Bei der erreichten schattierten Darstellungsrate von minimal 5 Bildern/sec[9] ergibt sich T_B zu 200ms. Die durch die Sensordatenerfassung in der Robotersteuerung bedingte Verzögerungszeit T_E wird hauptsächlich durch die A/D-Wandlung der Resolverdaten (16-Bit Genauigkeit) und die weitere Verarbeitungszeit in der Steuerung verursacht. Sie beträgt bei der JET-IVHU etwa 50ms. Somit ergibt sich rechnerisch eine gesamte Signallaufzeit T_G von **333ms**. Dies entspricht der Verzögerungszeit, nach der eine mechanische Bewegung der IVHU für den Operateur grafisch visuell erkennbar wird. Für zukünftige Kontrollsystem-Implementierungen bei NET/ITER ist eine Reduktion der Übertragungs-Totzeiten ein wesentlicher Faktor zur Erhöhung des Regelkreises bestehend aus Mensch, Handhabungsgerät und grafischem Monitor-System.

3.5 Benutzer- und Kommandoschnittstellen

3.5.1 Die KISMET-SCRIPT Kommandoschnittstelle

Unter **SCRIPT**-Kommandos versteht man eine **textuelle Schnittstelle** zum KISMET-Kommandoprozessor. SCRIPT-Kommandos können auf drei verschiedene Arten erzeugt bzw. in KISMET eingegeben werden :

1. Als Kommandosequenz mittels einer sequentiellen **SCRIPT-Datei.**
   Typische Anwendungen von SCRIPT-Dateien sind beispielsweise :

   - Das automatische Abarbeiten komplexer **Animationssequenzen.**
   - Die **Rekonfiguration** des KISMET-Modelldatenbaums zur Umschaltung der hierarchischen Detaillierungsstufen. Siehe SCRIPT-Kommandos:
     ACTIVATE, DEACTIVATE, SWAP_NODE, LEVEL_UP, LEVEL_DOWN
   - Die **Definition technologischer Daten** (Grenzwerte, Parameter, Bahnverhalten) für simulierte Kinematiken (Roboter, Manipulatoren, Kräne, Werkzeuge).
   - Änderungen der **Kinematik-Topologie** bei Greifoperationen des Arbeitsgeräts oder Werkzeug- und End-Effektorwechsel (siehe SCRIPT-Kommandos:
     CONNECT, FRAME_TO_WFRM, SET_FRAME PLACEMENT

2. Als einzelne Anweisungszeile mittels einer UNIX-**message.**[10] Unter Verwendung dieser 'message'-Schnittstelle ist es möglich, ein KISMET-Kommando von einem anderen, parallel zu KISMET ablaufenden Prozeß aus anzustoßen. Eine Einsatzmöglichkeit für SCRIPT-'messages' ist die Ankopplung von KISMET an eine spezielle Steuerungsapplikation im Handhabungsbereich. Ein *applikationsspezifischer* UNIX-Prozeß bearbeitet die Kommunikation zwischen KISMET und den Steuerungsrechnern (Zellenrechner) und erledigt das für die spezielle Applikation typische Datenmanagement. KISMET gestattet auf diese Weise die **applikationsspezifische Anpassung** und **Erweiterung** des Funktionsumfangs.

---

9 Im Drahtmodell-Darstellungsmodus werden von *GBSim* für das JET IVHU-Modell 12 Bilder/sec erreicht.

10 UNIX-messages sind eine Art der Interprozeß-Kommunikation zwischen quasiparallel ablaufenden Prozessen. Die Kommunikation erfolgt hier nach dem Mailbox-Prinzip, jedoch in der Ausführung um Größenordnungen schneller da der Datentransfer im Hauptspeicher erfolgt.

Als Beispiel sei hier das automatische, sensorgeführte Nachvollziehen von Greifoperationen oder von Werkzeugwechseln im Monitor-Betrieb genannt. Normalerweise müßte der Bediener einen im realen Handhabungsablauf erfolgten Werkzeugwechsel noch einmal in KISMET nachvollziehen. Falls die Zellensteuerung über geeignete Sensorik (codierte Taststifte, optische Erkennung) einen Werkzeugwechsel erkennt, so kann der applikationsspezifische Kommunikationsprozeß den Werkzeugwechsel mit einem geeigneten SCRIPT-Kommando (z.B. *CONNECT, FRAME_TO_WFRM*) über die 'message'-Schnittstelle ebenfalls in KISMET auslösen.

3. Als einzelne Anweisungszeile eingebettet in eine **IRDATA-Kommentarzeile.** Der in KISMET integrierte IRDATA-Interpreter erkennt SCRIPT-Kommandos falls die Kommentar-Zeichenkette mit dem Sonderzeichen "%" beginnt. In diesem Fall wird der Rest des Kommentars an den SCRIPT-Kommandointerpreter gesendet. Typische Anwendungen sind hier wiederum alle mit der Ausführung des IRDATA-Roboterprogramms verbundenen Änderungen der Kinematik-Topologie.

Eine Liste der implementierten SCRIPT-Kommandos ist in "Anhang D. Liste der KISMET SCRIPT-Kommandos" dokumentiert.

3.6 *Kollisionserkennung und -vermeidung*

3.6.1 Begriffsdefinition und angewandte Verfahren

Ein Modul zur **Kollisionserkennung** erzeugt **Meldungen** oder **Warnungen** bei

- direktem Kontakt der Geometrieoberflächen, oder bei
- Unterschreitung eines entweder durch den Operateur vordefinierten Minimalabstands, oder einer - in Abhängigkeit von der aktuellen Robotergeschwindigkeit - vom Simulationssystem automatisch berechneten Distanz

zwischen dem bewegten Handhabungsgerät und Objekten der Arbeitsumgebung oder anderen Handhabungsgeräten. Die Kollisionsmeldung erfolgt bei KISMET durch optische Hervorhebung der kollidierenden Geometrieobjekte kombiniert mit einer akkustischen Warnung. Bei der Anwendung im Monitor-Betrieb kann die numerische Kollisionserkennung auch zur einfachen **Kollisionsvermeidung** benützt werden, indem z.B. der Robotersteuerung ein Kommando zum Nothalt gesendet wird.

Eine komplexere Aufgabe stellt dagegen die Kollisionsvermeidung durch **automatische Korrektur** der **Roboterbewegung** dar. Diese Techniken sind gedacht als eine Komponente innerhalb von Modulen zur **automatischen Pfadsuche** und **Bewegungsplanung** [87]-[89], eignen sich jedoch aufgrund der Komplexität der bisher eingesetzten Algorithmen nicht für den Einsatz im Monitor-Betrieb in Echtzeit. Derartige Strategien zur **automatischen Bahnplanung** unter Vermeidung von Hindernissen sind jedoch als wesentliche Komponente der **impliziten Roboterprogrammierung** sinnvoll.

Schütze stellt in seiner Arbeit [20] verschiedene Ansätze zur On-Line- und Off-Line-Kollisionsüberwachung vor und vergleicht sie untereinander. Die unterschiedlichen Methoden werden dort klassifiziert nach

1. **Verfahren der On-Line-Kollisionsüberwachung**, die während der realen Bewegungsausführung aktiviert sind. Die Überwachung erfolgt während des Betriebs vorausschauend oder parallel zur Bewegungsplanung durch die Robotersteuerung. Die Methoden basieren auf der Auswertung von Sensorinformationen und werden weiterhin eingeteilt in Verfahren der :

   - *direkten Überwachung* mittels Berührungs-, Abstands-, oder Bildanalyse-Sensorik, und der
   - *indirekten Überwachung*, wobei hier der innere Zustand des Handhabungsgeräts ausgewertet wird. Eine Kollisionsgefahr wird erkannt, indem der Zustand der inneren, für die Positionsregelung sowieso vorhandenen und damit für die Kollisionserkennung nicht spezifischen Sensorik (z.B. Achswinkelgeber, Tachogeneratoren) über geeignete Algorithmen ausgewertet wird. Die indirekten Verfahren beruhen auf der Definition von erlaubten Bewegungsbereichen der einzelnen Roboterachsen, wobei weiterhin nach der Definition von *Freiräumen* und von *Verbotsbereichen* für die Achs-Verfahrbereiche unterschieden wird.

2. **Off-Line-Kollisionserkennung bzw. -vermeidung**, die zur Überprüfung vorprogrammierter Bewegungsabläufe dienen. In der Reihenfolge zunehmender Komplexität lassen sich diese Ansätze unterscheiden nach :

   - *grafisch-visueller Kollisionsprüfung*, wobei der Operateur an einem CAD-System interaktiv anhand der grafischen Darstellung für ausgewählte Positionen mögliche Kollisionen erkennen kann

- Überprüfung mittels *CAD-Funktionen*, wie z.B. Booleschen Operationen in CSG-Modellierern
- *automatische Kollisionstests* durch spezielle *Testalgorithmen* anhand eines mathematischen Anlagenmodells
- Prüfalgorithmen die unter Anwendung von *Ausweichstrategien* zu einer *Korrektur des Verfahrwegs* führen
- Methoden die, unter Verwendung von Ausweichstrategien und zusätzlichen *Optimierungskriterien*, zu einer *Optimierung des Fahrwegs* führen sollen
- Strategien der Bahnplanung unter Anwendung der Verfahren der *künstlichen Intelligenz* und der *impliziten Programmierung*.

Die in der Literatur aufgeführten Verfahren weisen verschiedene Nachteile auf, die eine praktische Anwendung im On-Line-Betrieb für die grafischen Bedienerführung von komplexen Handhabungsaufgaben ausschließen:

- einige Verfahren arbeiten lediglich im 2D-Raum und sind vor allem für die Bewegung von Fahrzeugen und Transporteinrichtungen in ebenen Bewegungsräumen entwickelt worden,
- andere Ansätze lassen sich nicht auf beliebige kinematische Strukturen anwenden,
- verschiedene Ansätze mit hoher Genauigkeit basieren auf Booleschen Operationen mit CAD-Volumenmodellen (CSG-Modelle) und benötigen so hohe Rechenzeiten, daß deren Einsatz ausschließlich für die Off-Line-Simulation oder, wie bei verschiedenen CAD-Systemen, im Stapelbetrieb in Frage kommt,
- die *Modellerzeugung* muß bei mathematischen oder auf CAD-Modellen basierenden Verfahren mit in Betracht gezogen werden.

Ein weiteres Problem, auf das in den vorgenannten Arbeiten nicht eingegangen wird, ist der praktische Aspekt der **beabsichtigten Kollision** bei Arbeitsvorgängen wie **Greifoperationen** oder während einer **kinematischen Ankopplung** von Werkzeugen und End-Effektoren an die Roboterstruktur. Das Modul zur Kollisionsüberwachung sollte daher in der Lage sein, verschiedene Objekte der Arbeitszelle abhängig vom Modellzustand automatisch als Hindernis oder als Teil der Roboterstruktur zu erkennen.

3.6.2 Anforderungen an ein Modul zur Kollisionserkennung

Aus der Sicht einer möglichen Anwendung eines Verfahrens zur Kollisionsvermeidung als Teilmodul innerhalb eines Systems zur *Echtzeit-Überwachung*, ergeben sich die folgenden Anforderungen:

1. Der Ansatz soll gleichermaßen für den On-Line-Betrieb (Monitoring) und für die Off-Line-Simulation geeignet sein.
2. Die Kollisionsprüfung sollte weitgehend automatisch erfolgen und daher keine, oder nur minimale Benutzerinteraktion erfordern.
3. Das Verfahren soll mit ausreichender geometrischer Genauigkeit arbeiten.
4. Das für die Kollisionsprüfung benützte mathematische Modell sollte automatisch aus dem für das synthetische Sehen verwendeten CAD-Modell abgeleitet werden um zusätzliche Modellierarbeit zu vermeiden.
5. Änderungen der kinematischen Topologie (Greifoperation, Werkzeugwechsel) sollen automatisch berücksichtigt werden. Bestimmte Modellkomponenten sind daher bezüglich der bewegten Handhabungsgeräte je nach Modellzustand als Hindernis oder nicht zu betrachten und beeinflussen daher das Modul zur Kollisions-Überwachung.
6. Die bisher genannten Anforderungen sollen bei minimalem Rechenzeitbedarf erfüllt werden.

3.6.3 Strategie des realisierten Kollisionsmoduls

Die zur Implementierung von GBSim und KISMET benützte Grafik-Hardware von *Silicon Graphics* bietet die Möglichkeit, die in der "Grafik-Pipeline" und in den Geometrie-Prozessoren (Firmenbezeichnung: *Geometry-Engine* [86]) mikroprogrammierte *Clipping-Funktion* neben den Funktionen des "Picking" (Antippen eines Grafikelements mit dem Cursor zur Identifikation) und des "Clipping" (Abschneiden und Wegblenden von Grafikelementen, die über den Objekt-Darstellungsraum hinausragen) auch zur Kollisionsrechnung zu nutzen. Dieser Grafikmodus wird als "Selecting" [76] bezeichnet. Diese Methode erlaubt die Identifikation von Grafikelementen, die den in Weltkoordinaten definierten, quaderförmigen 3D-Darstellungsausschnitt (*Clipping*-Quader, 3D-*Window*) schneiden.

Die Grundidee dabei ist, die bewegten Roboterglieder durch **einhüllende Quader** zu **approximieren** und in einem hierarchischen Testverfahren die auf Kollision zu untersuchenden Modellteile im "Selecting"-Modus gegen diesen Quader zu überprüfen. Die zu untersuchenden Teile sind dabei in das Bezugs-KS des Testquaders zu transformieren bzw. die Transformationsmatrix der "Grafik-Pipeline" ist entsprechend zu setzen. Wird mit $\mathbf{T}_r$ die absolute Transformation des Roboterglieds, mit $\mathbf{K}_b$ die Transformationsmatrix der Umgebungsbaugruppe und mit $\mathbf{A}_{rb}$ die Transformation des Umgebungsbauteils im Bezugssystem des Roboterglieds bezeichnet, so erhält man im prämultiplikativen Multiplikationssystem aus

$$\mathbf{K}_b = \mathbf{A}_{rb}\,\mathbf{T}_r \tag{3.4}$$

die Transformationskette

$$\mathbf{A}_{rb} = \mathbf{K}_b\,\mathbf{T}_r^{-1} \tag{3.5}$$

Der **Kollisionsalgorithmus** arbeitet aus Effektivitätsgründen als mehrstufiger, hierarchischer Test:

1. Zunächst werden die Testquader der Umgebungsbaugruppen (der einhüllende Quader aller Einzelgeometrien einer Baugruppe) mit dem Baugruppenquader (FRAME-Quader) des zu prüfenden Roboterglieds verglichen. Falls kleinere Baugruppen zu einer Baueinheit zusammengefaßt sind, wird für diese ebenfalls wieder ein Testquader erzeugt.

2. Bei einem positiven Test der FRAME-Quader werden im 2. Schritt die Testquader aller Einzelgeometrien der Umgebungsbaugruppe mit den einhüllenden Quadern der Einzelgeometrien des Roboterglieds verglichen.

3. In einem möglichen 3. Schritt wird die Oberfläche der Umgebungs-Geometrieteile gegen die Quader des Roboterglieds geprüft. Dieser Testschritt wird nur dann ausgeführt, falls für das getestete Geometrieteil der Quadertest in Schritt 2 anspricht.

Werden kleinere Baugruppen zu einer Baueinheit zusammengefaßt, ist diese ebenfalls von einem Testquader umgeben. Dieser wird als einhüllender Quader aller Baugruppen der Baueinheit berechnet. Der Testschritt 1 erfolgt also über die gesamte ABSTRACT-Hierarchie. Die Effizienz des Algorithmus ist darin begründet, daß in Schritt 1 nur dann die Elemente einer kleineren Baugruppe zu prüfen sind, falls vorher die Testquader der größeren Baueinheiten angesprochen haben.

Durch Vergrößerung der einhüllenden Quader um den benutzerdefinierten Betrag Δe läßt sich ein Test auf Beinahe-Kollisionen (engl. *near miss detection*) ausführen.

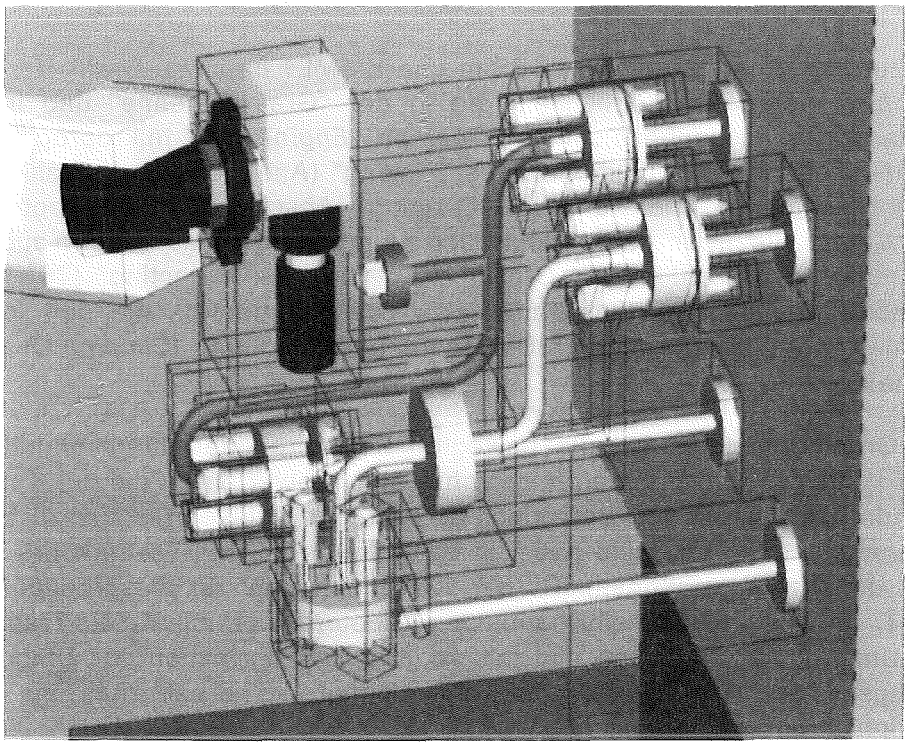


Abbildung 21. Kollisionstest über einhüllende Quader (Bounding Box): Für die ersten beiden Schritte des dreistufigen hierarchischen Tests werden von KISMET einhüllende Quader um das Baugruppen-FRAME (1.Stufe) und um die damit verknüpften Geometrie-Elemente (2.Stufe) verwendet.

Abbildung 21 zeigt die eingeblendeten Testquader für das KISMET-Modell des Roboter-Experiments CATROB [32].

Ein Problem in der Anwendung des "Selecting"-Modus für den ersten Testschritt ergibt sich dadurch, daß die Clipping-Hardware nur bei einem echten Schnitt der Testgeometrie mit dem Testquader anspricht. Liegt die Testgeometrie vollkommen **innerhalb** des Testquaders, so erfolgt kein Eintrag in die Trefferliste. Dieser Fall kann z.B. dann auftreten, wenn ein Manipulatorglied auf Kollisionen mit den umgebenden Innenwänden der Arbeitszelle zu überprüfen ist. Das Problem wird dadurch umgangen, daß ein beliebiger Geometriepunkt der Testgeometrie in das Bezugssystem des Testquaders transformiert wird. Für diesen Punkt wird geprüft, ob er innerhalb des Testquaders liegt (Minimum-/Maximum-Test)

Der einhüllende Quader $\{X(k)_{min}, X(k)_{max}, Y(k)_{min}, Y(k)_{max}, Z(k)_{min} Z(k)_{max}\}$ des geometrischen Grundkörpers k läßt sich bei der Initialisierung des internen facettierten Modells einfach berechnen.

Eine Kollision mit einer Kante E_j, gegeben durch deren Endpunkte $\{e_1(j), e_2(j)\}$, mit dem Grundkörper k ist **nicht** gegeben, wenn gilt:

$$\begin{cases} \mathrm{Max}\{e_1(x), e_2(x)\} > X(k)_{min} \\ \mathrm{Min}\{e_1(x), e_2(x)\} < X(k)_{max} \\ \mathrm{Max}\{e_1(y), e_2(y)\} > Y(k)_{min} \\ \mathrm{Min}\{e_1(y), e_2(y)\} < Y(k)_{max} \\ \mathrm{Max}\{e_1(z), e_2(z)\} > Z(k)_{min} \\ \mathrm{Min}\{e_1(z), e_2(z)\} < Z(k)_{max} \end{cases} \tag{3.6}$$

3.7 *Problem der verdeckten Kanten und Flächen*

Für die räumliche Orientierung des Operateurs im Handhabungsszenario ist die farbige und schattierte Darstellung ein wesentlicher Faktor. Hierbei tritt für die realistische Echtzeit-Darstellung das Problem der Ausblendung verdeckter Flächen und Kanten *(Hidden-Surface Removal)* auf. Dieses Problem ist seit den 60er Jahren das Thema von zahllosen wissenschaftlichen Arbeiten und Abhandlungen. Die entwickelten Algorithmen sind jedoch stark abhängig vom Datenmodell, sowie von der Art des Ausgabegeräts. Eine Klassifikation und vergleichende Gegenüberstellung der verschiedenen Methoden ist in den Werken [59], [60] und [90] gegeben.

Für die bei JET für **GBSim** benutzte Rastergrafik und auf Polyedermodelle zugeschnittene Grafik-Workstation vom Typ *SILICON GRAPHICS IRIS 3020* eignen sich verschiedene Methoden, die im wesentlichen alle nach dem Prinzip des im Englischen mit "Painters-Algorithm" bezeichneten Verfahren arbeiten:

- Geometrieteile oder Polygone werden hierbei abhängig von der Bildtiefe, d.h. der Entfernung vom Betrachterstandpunkt sortiert und von "hinten" nach "vorne" gezeichnet. Dabei werden verdeckte Teile von entfernteren Modellobjekten, bedingt durch die Reihenfolge der Darstellung, von näherliegenden Teilen überzeichnet.
- Die dem Betrachter abgewandten Oberflächenfacetten werden vor dem Zeichenvorgang identifiziert und nicht dargestellt, wobei sich bei konvexen Geometriekörpern automatisch eine korrekte Darstellung ergibt und sich die Zahl der darzustellenden Facetten wesentlich reduziert.

In modernen Grafikgeräten der oberen Preisklasse kommt ein weiteres Verfahren auf Hardwarebasis zum Einsatz, wobei ein sogenannter "Z-Buffer" bzw. "Tiefenspeicher" eingesetzt wird:

- Im Bildspeicher werden für jeden Bildpunkt neben den Farbwerten auch die Tiefenwerte, d.h. die Entfernung des gezeichneten Oberflächenpunkts vom Betrachterstandpunkt, gespeichert. Die Auflösung des Z-Speichers beträgt in der Regel 16-24 Bit pro Bildpunkt.
- Vor dem Beginn des Darstellungsvorgangs wird der Z-Speicher auf den maximalen Tiefenwert initialisiert.
- Bei der Darstellung einer Facette werden die Tiefenwerte der Polygon-Eckpunkte im Bild-KS berechnet und innerhalb des Polygons für jeden Bildpunkt interpoliert.
- Liegt der neu zu zeichnende Bildpunkt näher als der im Z-Speicher bereits gespeicherte Wert, so wird der Farbwert (Intensität) in den Bildspeicher und der neue Tiefenwert in den Z-Speicher geschrieben.
- Das Verfahren eignet sich ebenfalls für nicht-konvexe Polygone und liefert auch bei Durchdringung der Geometrieteile korrekte Ergebnisse.

Die Grafik-Workstation IRIS 3020 ist optional mit einem Z-Speicher ausrüstbar, die Zeichengeschwindigkeit wird dadurch jedoch um Größenordnungen reduziert, sodaß ein Einsatz für die Echtzeit-Simulation ausgeschlossen war.

Für **GBSim** wurde deshalb ein Software-Algorithmus entwickelt und implementiert, der den folgenden Anforderungen genügen mußte:

1. Das wesentlichste Kriterium ist die Ausführungsgeschwindigkeit, da als das wichtigste Entwicklungsziel eine hohe Darstellungsrate angesehen wurde.

2. Ein erhöhter Aufwand bei der Modellierung kann in Kauf genommen werden, da die Rate der Änderungen in der simulierten Anlage und damit im Simulationsmodell gering ist.

3. Der Algorithmus muß geeignet sein für bewegte Modelle (Roboter, Manipulatoren) und für die in der simulierten Anlage auftretenden Situationen korrekte Ergebnisse liefern.

4. Durchdringungen von Geometrieteilen können schon aus physikalischen Gründen nicht auftreten und deshalb außer Betracht bleiben.

5. Die Geometrie wird als Starrkörpersystem betrachtet.

In den Jahren ab 1988 wurden für Grafiksysteme der oberen Preisklasse ausreichend schnelle Z-Speicher entwickelt und auf dem Markt angeboten, für eine mögliche Portierung der Simulationssoftware auf Rechner der PC-Klasse ist das entwickelte und nachfolgend beschriebene Verfahren jedoch nach wie vor aktuell.

Fuchs beschreibt in [91] einen Algorithmus, wobei das Modell in einem *Binary Space Partitioning Tree* (BSP-Baum) rechnerintern dargestellt wird. Das Verfahren eignet sich für B-REP-Modellierer (Boundary Representation, Oberflächendarstellung) mit internem Polyeder-Datenmodell zur schnellen Darstellung von 3D-Szenen bei variabler Sichtposition.
Die fundamentale Grundidee der Separationsebene ist, daß bei einer 3D-Szene mit einem Betrachterstandpunkt und einer Trennebene kein Polygon, das vollständig auf der dem Betrachter zugewandten Seite der Trennebene liegt, von irgendeinem anderen Polygon auf der dem Betrachter abgewandten Seite der Trennebene verdeckt werden kann. Selbstverständlich wird diese Verdeckungsordnung vertauscht, falls der Betrachterstandpunkt auf die andere Seite der Separationsebene wandert. Der Algorithmus von *Fuchs* benutzt diesen Gedanken zur Konstruktion eines binären Datenbaums aus Polygonen und Separationsebenen.
Der Algorithmus liefert gute und ausreichend schnelle Ergebnisse bei **statischen Szenen**, da der Schritt zur Erzeugung des Datenbaums nur bei der Modellgenerierung oder Initialisierung durchgeführt werden muß. Bei der synthetischen Bilderzeugung von Handhabungsszenarien kann dieser Algorithmus jedoch keine Anwendung finden, da die Szene durch bewegte Roboterstrukturen dynamischer Natur ist. Die Berechnungszeit zur Neuerzeugung des BSP-Baums liegt für Szenen mit mehr als 1000 Polygonen im Bereich mehrerer Sekunden auf Rechnern der Leistungsklasse der IRIS 3020.

Für **GBSim** wurde der Grundgedanke der räumlichen Separation erweitert auf bewegte kinematische Modelle. Es gilt jedoch die Einschränkung, daß die Szene **Kohärenzmerkmale** aufweisen muß. Die kinematische Struktur muß mit einer Basis in der simulierten Arbeitsumgebung fixiert sein. Zur effektiven Abarbeitung wird die Szene ausschließlich mit **konvexen Geometriekörpern** modelliert. Bei Eliminierung der dem Betrachter abgewandten Oberflächenfacetten kann ein Körper ohne weitere Sortierung der einzelnen Polygone eines Geometrieteils korrekt gezeichnet werden.

Abbildung 22 auf Seite 51 zeigt eine mögliche Separation der Geometrieteile einer Umgebungsszene und deren Abbildung in einen Datenbaum, dessen Knoten durch Separationsebenen und dessen Blätter durch konvexe Geometriekörper gebildet werden. Die Separationsebene wird jeweils durch zwei Testpunkte "Back" (B) und "Front" (F) gebildet (die Ebene als der geometrische Ort aller Punkte mit gleichem Abstand zu zwei nicht zusammenfallenden Punkten). Die Vergleichsoperation, ob sich der Betrachterstandpunkt auf der einen oder der anderen Seite der Trennebene liegt, kann daher durch einfache Berechnung der Distanz zwischen Sichtposition und den Testpunkten erfolgen.

Abbildung 23 auf Seite 52 zeigt, daß die Testpunkte so positioniert werden können, daß bei bewegten Robotergelenken die Ebenen automatisch so "mitwandern", daß stets eine räumliche Trennung der Armelemente erfolgt. Die **Testpunkte** werden dazu mit dem bewegten Armelement **kinematisch verknüpft.**

Der Algorithmus wurde aufgrund der flexiblen Natur der Trennebenen und, wie sich in der weiteren Beschreibung zeigen wird, der flexiblen Sortierstrategie, als **Flexible Separation Plane Algorithmus** (FSP-Algorithmus) bezeichnet.

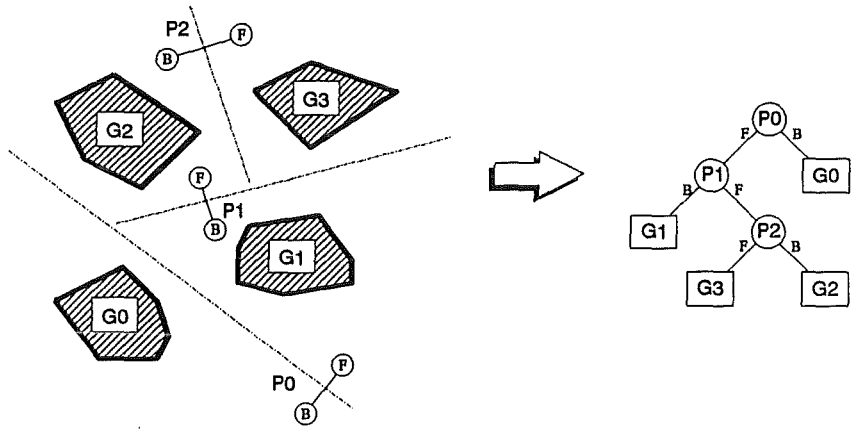


Abbildung 22. Räumlicher Separationsbaum: Erzeugung einer räumlichen Separation durch Trennebenen und Abbildung auf einen Datenbaum. Die Trennebenen sind jeweils durch 2 Testpunkte definiert.

Der implementierte Algorithmus besteht im wesentlichen aus drei Komponenten:

1. Das Lesemodul "Make_FSP_tree" wird einmalig bei der Initialisierung des Datenmodells aufgerufen und generiert aus der FSP-Definitionsdatei den FSP-Baum. Diese Datei wird bei der Szenenmodellierung einmalig definiert.

2. Das Ablaufmodul "hiddsurf" wird pro Szenenfenster bei jeder Neuberechnung einmalig aufgerufen. Das Modul transformiert zunächst die Testpunkte des gesamten FSP-Baums von relativen Objektkoordinaten in Weltkoordinaten und ruft dann das Modul "do_plane" zur Sortierung der konvexen Geometrieobjekte auf. Eine Neuerstellung der Sortierliste ist durchzuführen falls :

   - die Betrachterposition geändert wird,
   - bei Bewegung der kinematischen Modellfreiheitsgrade, wodurch sich die relative Lage der Modellteile zueinander ändern kann.

   In diesem Bearbeitungsschritt werden außerdem die dem Betrachter abgewandten Facetten identifiziert und markiert *(Backface Removal)*.

3. Das rekursive Unterprogramm "do_plane" traversiert den FSP-Baum und erzeugt eine entsprechend der Darstellungsreihenfolge von "hinten" nach "vorne" sortierte Verweisliste der Geometrieobjekte. Diese Liste wird bei der Darstellung abgearbeitet, wobei die von der aktuellen Sichtposition weiter entfernten Modellteile von nähergelegenen Objekten bei Verdeckung überzeichnet werden.

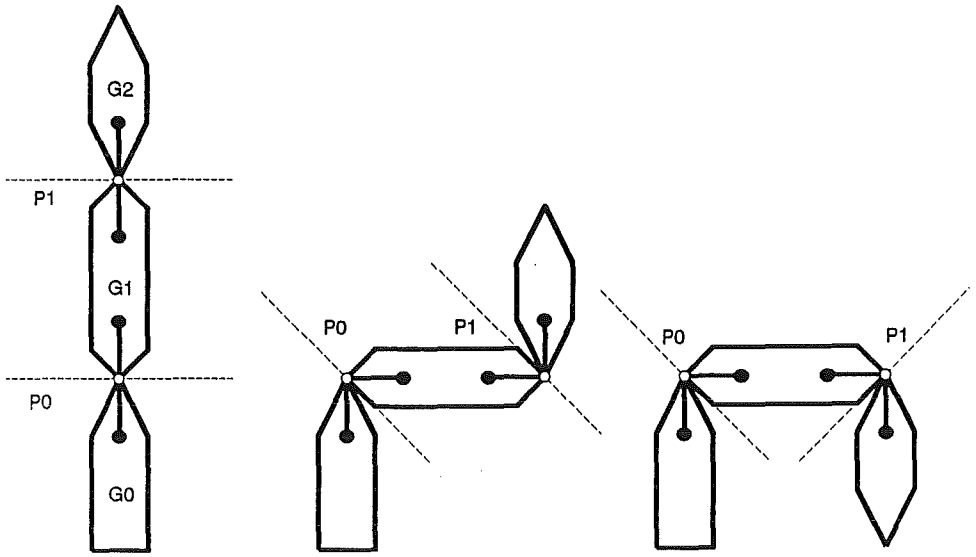


Abbildung 23. Separationsbaum für bewegte Kinematiken: Durch Mitbewegen der Testpunkte eignet sich die Methode auch für bewegte Mechanismen.

Nachfolgend ist das rekursive Unterprogramm *"do_plane"* in Pseudocode dargestellt.

```
PROC do_plane(plane)
{
  IF (plane.typ == GEOMETRIE)
    Zeichne(plane.geometrie);
  ELSE {
    DF = distanz(Auge, plane.testpunkt_Front);
    DB = distanz(Auge, plane.testpunkt_Back);
    IF ( DF < DB) {
      do_plane(plane.Zeiger_Back);
      do_plane(plane.Zeiger_Front);
    }
    ELSE {
      do_plane(plane.Zeiger_Front);
      do_plane(plane.Zeiger_Back);
    }
  }
}
```

Bewertung

Durch den für *GBSim* entwickelten FSP-Algorithmus war es möglich, die von JET gestellte Forderung nach einer schattierten Echtzeit-Darstellung des "In-Vessel"-Szenario auf dem vorhandenen Rastergrafiksystem zu erfüllen. Es unterscheidet sich vom Ansatz nach *Fuchs* dadurch, daß es auch für *bewegliche Geometriestukturen* die notwendige Leistung erbringt. Der notwendige Modellieraufwand für die Unterteilung der Geometrie in konvexe Körper und zur einmaligen Erstellung des FSP-Sortierbaums ist für Sonderentwicklungen wie *GBSim* tragbar.

3.8 *KISMET Darstellungs- und Schattierungsmodelle*

3.8.1 Beleuchtungsmodelle

Die Intensität des von irgendeinem Punkt eines Objekts reflektierten Lichts hängt bekanntermaßen ab vom Winkel zwischen der Blickrichtung und dem reflektierten Lichtvektor in diesem Punkt. Für lichtundurchlässige Objekte wurden zur quantitativen Formulierung dieser Tatsache verschiedene Methoden vorgeschlagen, als Beispiele seien hier die Arbeiten von *Gouraud* [74], *Phong* [92] und *Blinn* [93] genannt. *Whitted* [94], *Hall* und *Greenberg* [95] erweiterten dieses Lichtmodell um Komponenten zur Simulation von Spiegelungen und von transparenten Materialien. Im Beleuchtungsmodell von *Cook* und *Torrance* [96] werden in der Reflexionskomponente zusätzlich aus der *Fresnel-Gleichung* abgeleitete Farbverschiebungen berücksichtigt.

Unter der Annahme, daß die Reflexionseigenschaften eines Materials unabhängig von der Wellenlänge λ des Lichts sind, wird die Intensität $[R_p, G_p, B_p]$ des Oberflächenpunkts P im **"Ray-Tracing-Modus"** von KISMET berechnet entsprechend der Basisgleichung :

$$\begin{bmatrix} R_p \\ G_p \\ B_p \end{bmatrix} = K_a \begin{bmatrix} R_m \\ G_m \\ B_m \end{bmatrix} + K_s \begin{bmatrix} R_s \\ G_s \\ B_s \end{bmatrix} + K_r \begin{bmatrix} R_t \\ G_t \\ B_t \end{bmatrix} + \sum_{i=1}^{n} S_i L_i A(d_i) \cdot \left[\{K_d \cos \Phi\} \begin{bmatrix} R_m \\ G_m \\ B_m \end{bmatrix} + \{\omega(\Phi)(\cos \Theta)^{Kp}\} \begin{bmatrix} R_{L,i} \\ G_{L,i} \\ B_{L,i} \end{bmatrix} \right] \tag{3.7}$$

wobei gilt :

$$\cos \Phi = \mathbf{L} \bullet \mathbf{N} \tag{3.8}$$

und

$$\cos \Theta = \mathbf{V} \bullet \mathbf{R} \tag{3.9}$$

Parameter	*Erläuterung*
Φ	Winkel zwischen dem von der Lichtquelle *i* im Punkt *P* auftreffenden Lichtstrahl (in Richtung der Lichtquelle) und der Oberflächennormalen des Objekts im Punkt *P*
Θ	Winkel zwischen dem von der Lichtquelle *i* im Punkt *P* reflektierten Lichtstrahl und dem Sehvektor (Verbindungsgerade zwischen dem Punkt *P* und dem Betrachterstandpunkt) entsprechend dem Beleuchtungsmodell von *Phong*
N	Normalenvektor der Körperoberfläche am Auftreffpunkt *P* des Lichtstrahls
L	Vektor zur Lichtquelle *i*
R	An der Oberfläche reflektierter (gespiegelter) Lichtvektor
V	Vektor zum Betrachterstandpunkt

$\omega(\Phi)$ Winkelabhängige Funktion der Lichtreflexion an einer spiegelnden Oberfläche. Zur Vereinfachung wird angenommen, daß die Reflexion konstant und unabhängig vom Winkel Φ zwischen Oberfläche und Lichtvektor ist.

Es gilt daher: $\omega(\Phi) = K_s$

K_a Koeffizient für den Anteil der Umgebungshelligkeit (*Ambient Constant*) für das Material *m*.
Wertebereich: $K_a \in [0 \ldots 1]$

K_d Koeffizient für diffuse Reflexion (*Diffuse Constant*) Beleuchtung des Materials mit Index *m*.
Wertebereich: $K_d \in [0 \ldots 1]$

K_s Reflexionskoeffizient (*Specular Constant*) des Materials *m*.
Wertebereich: $K_s \in [0 \ldots 1]$

K_p Index für Glanzeffekte (*Specular Power*) des Materials *m*. Je höher dieser Koeffizient gewählt wird, desto mehr handelt es sich bei dem Material um eine ideal spiegelnde Oberfläche.
Wertebereich: $K_p \in [0 \ldots 127]$

K_r Koeffizient zur Definition der Lichtdurchlässigkeit des Materials *m* (*Refraction Coeffizient*). Für undurchlässige Materialien gilt $K_r = 0.0$, für vollkommen transparente Oberflächen (ideale Durchlässigkeit) $K_r = 1.0$

$A(d_i)$ Abhängigkeit der Lichtquellen-Intensität von der Entfernung d_i zwischen Lichtquelle *i* und Oberflächenpunkt *P* (*Attenuation Coeffizient*). Die Funktion sollte der physikalischen Abhängigkeit

$$A(d_i) \sim \frac{1}{d_i^2}$$

entsprechen, deren Nachbildung jedoch subjektiv unrealistische Darstellungsergebnisse liefert (zu schnelle Abnahme der Intensität)

Verwendet wird daher:

$$A(d_i) = \frac{d_0}{d_0 + d_i} \qquad (3.10)$$

R_p, G_p, B_p Lichtintensität (Intensitäten der Grundfarben des RGB-Farbmodells im Intervall $[0 \ldots 1]$) in der Projektionsebene des Betrachters

R_m, G_m, B_m RGB-Farbintensität des Objekts(Grundfarbe des Materials *m*)

$R_{L,i}, G_{L,i}, B_{L,i}$ RGB-Farbintensität der Lichtquelle *i*

R_s, G_s, B_s RGB-Farbintensität des im Objektpunkt *P* gespiegelten Lichtstrahls (Anteil der Totalreflexion, Licht von anderen Objekten in der Szene)

R_t, G_t, B_t RGB-Farbintensität des bei transparenten Materialien im Objektpunkt *P* aus der Oberfläche heraustretenden Lichtstrahls

3.8.2 Szenendarstellung in KISMET

Wesentliche Qualitätskriterien von 3D-Grafiksystemen sind die über die Hardware verfügbaren, und von der Software auch genutzten, **grafischen Darstellungsmöglichkeiten.** In der vorliegenden Arbeit wurde die programminterne Darstellungs-Datenstruktur für größtmögliche Geschwindigkeit bei akzeptabler Bildqualität optimiert. Innerhalb dieser Datenstruktur ist der Darstellungsmodus als **Attribut** für die gesamte Szene, einzelne Baugruppen (FRAME), oder einzelne Geometrie-Elemente (GEO) interaktiv veränderbar.

Die realisierten **Darstellungsmodi** sind:

"Drahtmodell-Darstellung" (Wireframe)
In der Drahtmodell-Darstellung werden die Geometrie-Facetten als geschlossener Polygonzug gezeichnet. Der Z-Buffer der Grafik-Hardware ist in diesem Zeichenmodus aktiviert, kann jedoch optional abgeschaltet werden, wodurch noch eine weitere Steigerung der Darstellungsgeschwindigkeit möglich ist (ca. 10%). Dies kann jedoch zu einer Störung des räumlichen Eindrucks führen. Zur Verbesserung der Darstellungsqualität führt das Aktivieren des *Antialiasing*-Modus. Hier werden die durch Überabtastung des Bildspeichers hervorgerufenen **Aliasing-Effekte** [59], die sich als "Treppenstufen" an den Polygonkanten zeigen, in der Grafikhardware durch die Berechnung von Pixel-Zwischenfarben deutlich vermindert.

"Flat-Shaded-Darstellung"
In diesem Darstellungsmodus werden die einzelnen Facetten des programminternen Polyedermodells mit einer konstanten, für jeden Polygonzug neuberechneten Farbintensität gefüllt. Als Beleuchtungsmodell wird eine einzelne, stets im Betrachterstandpunkt positionierte, Punktlichtquelle verwendet. Bei der Berechnung der Farbe eines Oberflächenpolygons werden die definierten Oberflächen-Parameter ("Diffuse-" und "Specular"-Konstanten, Grundfarbe) des Geometriekörpers und die Oberflächen-Normale der Facette berücksichtigt; siehe Gleichung (3.7). In diesem Darstellungsmodus werden hohe Zeichengeschwindigkeiten bei akzeptabler Bildqualität erzielt. Bei gekrümmten Geometrieoberflächen treten an den Facettenkanten deutliche Nichtlinearitäten in den Farbübergängen auf.

"Gouraud-Shaded-Darstellung"
Die oben erwähnten Nichtlinearitäten der Farbübergänge an den Facettenkanten werden bei der Schattierungsberechnung nach *Gouraud* [74] vermieden. Die Farbintensitäten werden an den Eckpunkten der Facette entsprechend dem gewählten Beleuchtungsmodell exakt gerechnet. Für jeden Eckpunkt der zu zeichnenden Facette ist hierzu im rechnerinternen Darstellungsmodell der Oberflächen-Normalenvektor gespeichert. Innerhalb des Polygons wird die Farbintensität durch die Hardware des Grafik-Subsystems interpoliert. Das **Beleuchtungsmodell** kann interaktiv definiert werden, wobei zwischen einer einzigen Lichtquelle im Betrachterstandpunkt, oder einem rechenintensiveren Beleuchtungsmodell mit bis zu acht, vom Benutzer definierbaren Lichtquellen umgeschaltet werden kann. Zur Definition des Beleuchtungsmodells kann zwischen Quellen mit parallel einfallendem Licht, Punktlichtquellen und gerichteten Punktlichtquellen mit von der Strahlrichtung nach außen abnehmender Lichtintensität ("Spotlight") gewählt werden. Werden mehrere Lichtquellen benutzt, so summiert die Grafik-Hardware die einzelnen Intensitätskomponenten nach Gleichung (3.7) auf.

Als weitere Darstellungsoption können einzelne Geometrie-Elemente oder Baugruppen **transparent** und schattiert (ausgefüllt) gezeichnet werden. Hierzu wird eine Methode benutzt, die in der Literatur als *Alpha-Blending* bezeichnet wird [76]. Im Transparent-

modus wird die Farbe des neu zu zeichnenden Bildpunkts als Überlagerung zwischen bereits im Bildspeicher vorhandener Bildpunktfarbe und einem additiven Anteil der Grundfarbe des transparenten Materials gezeichnet. Falls mindestens ein Geometrieelement transparent darzustellen ist, wird das gesamte Darstellungsmodell zweimal durchlaufen.

Im ersten Durchlauf werden mit aktiviertem *Z-Buffer* alle undurchsichtigen Komponenten in den Bildspeicher geschrieben, im zweiten Durchlauf alle transparenten Elemente, wobei der *Z-Buffer* zum Lesen aktiviert ist, jedoch nicht neu beschrieben wird.

Die für das transparente Zeichnen verwendete *Alpha-Blending*-Funktion für die resultierende Intensität $I_r = [R_n G_n B_n]$ eines Bildpunkts lautet:

$$I_r = I_d(1-\alpha) + \alpha I_s \tag{3.11}$$

wobei I_d die bisherige Farbintensität im Bildspeicher und I_s die Grundfarbe des transparenten Materials bezeichnen. Der Parameter α ist ein Maß für die Transparenz des Materials, wobei der Wert 0 für vollkommen transparente und der Wert 1 für vollkommen undurchsichtige Oberflächen verwendet wird. Zufriedenstellende Ergebnisse erhält man für den Wert $\alpha = 0.25$ bei "transparenten" Materialien.

Eine weitere Darstellungsoption ist das Zeichnen von **Pseudo-Schatten** (engl. "Fake-Shadows"). Darunter sei eine Echtzeit-Darstellungstechnik verstanden, die - auf der Anwendung einer **Abbildungstransformation** basierend - den Schattenwurf des Modells in **eine Projektionsebene** berechnet und darstellt. Die Technik ist daher nicht geeignet Schlagschatten, bzw. den Schattenwurf eines Körpers auf die Oberfläche eines anderen Geometriekörpers, zu berechnen. Das hierzu in der vorliegenden Arbeit benutzte Rechenverfahren basiert auf der im Aufsatz von *Blinn* erläuterten Technik [97], dort wird der "Schatten" allerdings in eine andere Projektionsebene ($z = 0$) abgebildet. Die Herleitung der für KISMET verwendeten Abbildungsmatrix sei deshalb kurz erläutert.

Ein beliebiger Geometriepunkt $\mathbf{p} = [p_x, p_y, p_z]$ soll durch die sehr weit entfernt liegende parallele Lichtquelle $\mathbf{q} = [q_x, q_y, q_z]$ in seinen Schattenpunkt $\mathbf{s} = [s_x, s_y, s_z]$ in der Ebene ($y = 0$) abgebildet werden. Der Schattenwurf beginnt am Punkt **p** und verläuft entgegengesetzt zur Einfallsrichtung **q** der Parallel-Lichtquelle, bis er im Punkt **s** auf die Ebene $y = 0$ trifft. Daher gilt:

$$\mathbf{s} = \mathbf{p} - \alpha \mathbf{q} \tag{3.12}$$

Die Lösung für α unter der Randbedingung $s_y = 0$ lautet:

$$0 = p_y - \alpha q_y \qquad \text{bzw.} \qquad \alpha = \frac{p_y}{q_y}$$

Für die *x*- und *z*-Koordinaten des Schattenpunkts **s** ergibt sich daraus:

$$s_x = p_x - \frac{p_y}{q_y} q_x \qquad \text{bzw.} \qquad s_z = p_z - \frac{p_y}{q_y} q_z$$

Für diese einfachen Beziehungen läßt sich nun eine homogene "Schattenmatrix" $\mathbf{M}_S$ finden, die eine Ausführung der o.g. Rechenoperation innerhalb der Transformationskette, d.h. in der Matrix-Pipeline, ermöglicht.

$$\mathbf{s} = \mathbf{p} \cdot \mathbf{M}_S \tag{3.13}$$

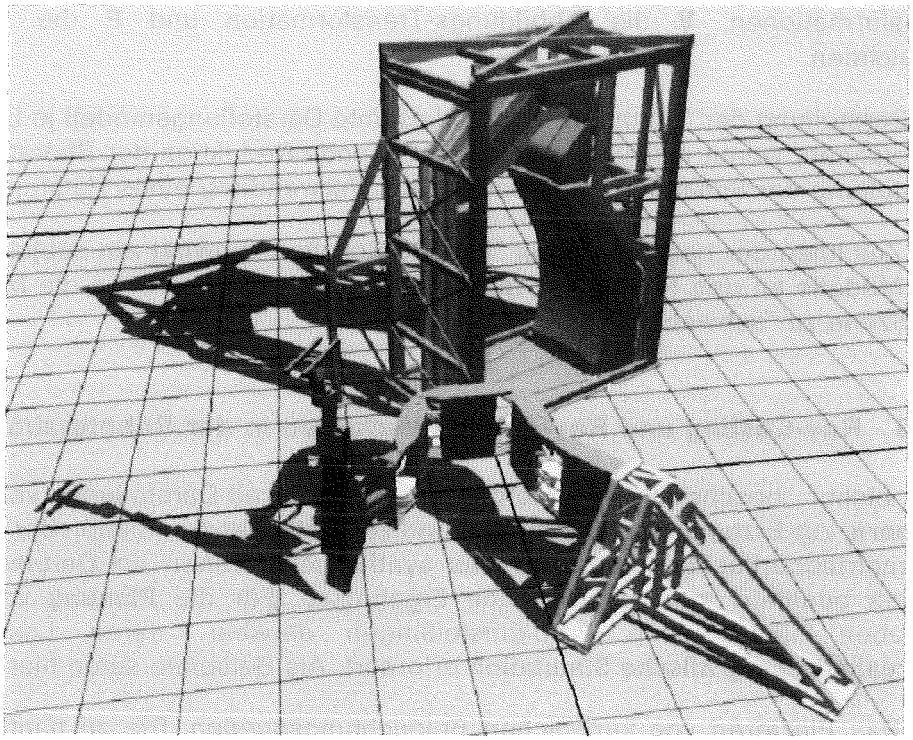


Abbildung 24. EDITH-Szene mit "Pseudo-Schatten": Das Bildschirmfoto zeigt das Simulationsmodell des KFK-Teststands EDITH mit dem eingeblendeten Schatten einer Punktlichtquelle.

oder für **parallele Lichtquellen** :

$$[s_x \;\; 0 \;\; s_z \;\; 1] \;=\; [p_x \;\; p_y \;\; p_z \;\; 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ -q_x/q_y & 0 & -q_z/q_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

Für den "Schattenwurf" einer **Punktquelle** lautet der Ansatz für den Schnittpunkt $\mathbf{s}$ mit der Projektionsebene :

$$\mathbf{s} = \mathbf{p} - \alpha(\mathbf{p} - \mathbf{q}) \tag{3.15}$$

Daraus erhält man (ohne Darstellung der Zwischenschritte) für die Schattenmatrix $\mathbf{M}_S$ der **Punktlichtquelle** in der Ebene ($y = 0$) :

$$\mathbf{M}_S \;=\; \begin{bmatrix} 1 & 0 & 0 & 0 \\ -q_x/q_y & 0 & -q_z/q_y & -1/q_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

Die **gesamte Transformationskette** für die Überführung eines Geometriepunkts $\mathbf{p}_M$ gegeben in Geometrie-Modellierkoordinaten in seinen "Schatten" $\mathbf{s}_B$ (im Betrachter-KS) lautet daher mit Berücksichtigung der Schattenmatrix $\mathbf{M}_S$:

$$\mathbf{s}_B \;=\; \mathbf{p}_M \, \mathbf{M}_G \left[\mathbf{M}_{K,(n,n-1)} \;\ldots\; \mathbf{M}_{K,21} \, \mathbf{M}_{K,10}\right] \mathbf{M}_S \; \mathbf{V} \; \mathbf{P} \tag{3.17}$$

wobei $\mathbf{M}_G$ die relative Geometrie-Modelliertransformation, $\mathbf{M}_{K,ji}$ die Kette der Kinematik-Transformationen, **V** die Abbildungs-Transformation und **P** die Projektionsmatrix bezeichnen.

Zur Darstellung der "Schatten" ist das gesamte Darstellungsmodell je Lichtquelle einmal zu zeichnen, womit natürlich eine entsprechende Reduktion der Bildrate einhergeht. Die Darstellung der Schatten erfolgt dabei unter Verwendung eines Halbton-Füllmusters.

Das Zeichnen der beschriebenen "Pseudo-Schatten" bewirkt ein subjektives "Anbinden" des Modells an den (virtuellen) Boden. Ohne Schatten scheinen die Modellelemente in der Szene zu "schweben".

3.8.3 Ray-Casting und Ray-Tracing-Verfahren in der Robotersimulation

Die bisher beschriebenen Methoden zur Echtzeit-Darstellung von **transparenten Körpern**, zur Erzeugung von **Schatten** und zur einfachen Simulation der **Lichtreflexion** auf Geometrieoberflächen liefern für das synthetische Sehen im On-Line-Teleoperationsbetrieb qualitativ zufriedenstellende Ergebnisse. Für die **Planung** von Handhabungsaufgaben sind jedoch Problemstellungen denkbar, deren Lösung eher eine **fotorealistische grafische Simulation** erfordert. Als Beispiele seien hier aufgeführt:

- Das Plazieren von künstlichen Beleuchtungsquellen. Die zu hantierenden Objekte können im Schatten von anderen Teilen der Arbeitszelle liegen und daher für die Szenenkameras schwer erkennbar sein.

- Beleuchtungsquellen können sich auf polierten Oberflächen spiegeln, was zum Überzeichnen bzw. zur "Blendung" der Videokameras führen kann.

Von allen bekannten Verfahren zur computergrafischen, synthetischen Bilderzeugung liefert die Methode des **Ray-Tracing**[11] [94], [98], [99], [100] bezüglich des erzielten **Szenenrealismus** bislang unübertroffene Ergebnisse. Das *Ray-Tracing*-Verfahren löst auf elegante Weise die computergrafischen Teilprobleme der *verdeckten Flächen* (Hidden-Surface Removal), des *Schattenwurfs*, der *Lichtreflexion* und *Spiegeleffekte*, sowie von *Lichtdurchgang* (Totalreflexion) und *Brechung* an den Oberflächen von *transparenten Materialien*. Für statische Szenen können außerdem störende *Aliasing-Effekte* durch Überabtastung und Mittelung (Tiefpaßfilterung) weitgehend reduziert werden. Der Hauptnachteil dieser Technik ist zweifellos auch noch im Zeitalter der Supercomputer, Vektor- und Parallelrechner (*Transputer*-Systeme), die zur Bildberechnung benötigte ungeheure Rechenzeit.

Eine bzgl. des Rechenaufwands vereinfachte Methode des *Ray-Tracing*, **Ray-Casting** genannt, findet vornehmlich Anwendung in der Erzeugung von schattierten Darstellungen bei CSG-Modellierern, wie z.B. im Aufsatz von *Roth* [101] beschrieben ist. Die *Ray-Casting*-Technik erlaubt zwar die Darstellung von Schatten, hingegen Spiegeleffekte zwischen den Objekten werden mit dieser Methode nicht erfaßt.

Die Arbeiten zum Themenfeld *Ray-Tracing* beschäftigen sich u.a. mit der Beschleunigung des Rechenverfahrens. *Clark* [102] verwendet dazu hierarchische Datenmodelle, *Whitted* [94] und *Weghorst et.al.* [104] benützen einfache Hüllkörper (Bounding-Volumes), *Glassner* berichtet in [103] über die Methode der Zell-Dekomposition (Octrees).

---

11 In den frühen Veröffentlichungen wurden auch einfachere, nichtrekursive Techniken, die man in späteren Arbeiten als *Ray-Casting* bezeichnet, noch *Ray-Tracing* genannt.

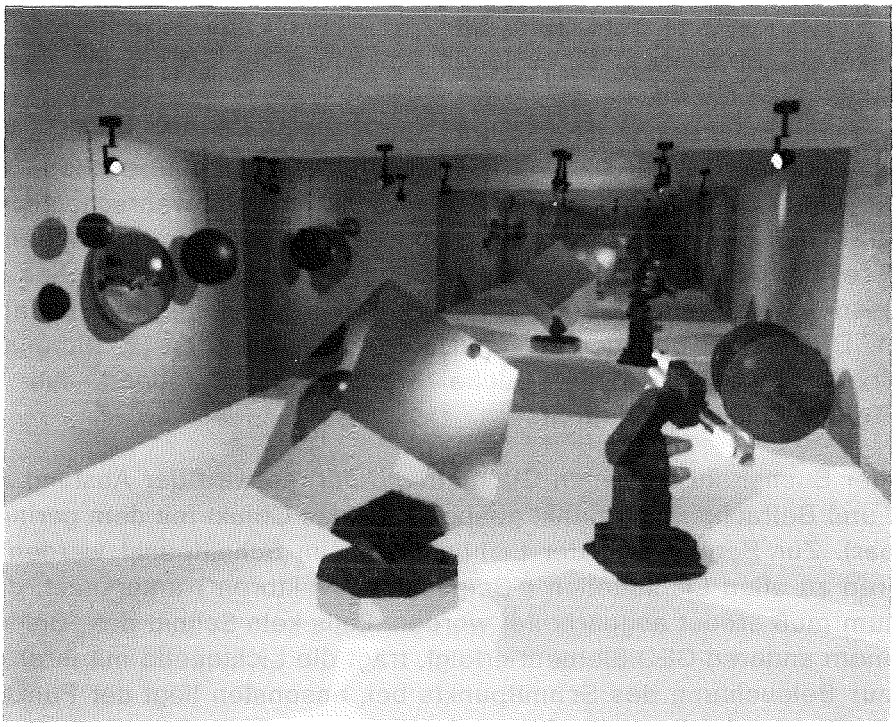


Abbildung 25. Ray-Tracing mit KISMET: Bildschirmfoto einer mit KISMET im Ray-Tracing-Verfahren erzeugten Szene.

Andere Gruppen von Autoren beschäftigen sich mit der Verbesserung der Darstellungsqualität [95], [94], [98], oder mit *Ray-Tracing* von bestimmten Objektklassen [105] (Fraktale, prozedural definierte Objekte), [106] (u.a. algebraische Flächen).

Seine Anwendung findet *Ray-Tracing* heutzutage bei kommerziellen Systemen die in den Anwendungsbereichen **Produktdesign, Werbung, Architektur**, sowie zur Erzeugung von **Animationsfilmen** eingesetzt werden.

Der in KISMET implementierte **Ray-Tracing-Algorithmus** basiert, wie auch die meisten anderen Realisierungen, auf der z.B. im Aufsatz von *Hall* und *Greenberg* [95] beschriebenen Standard-Technik. Die Szene wird ausgehend vom Betrachterstandpunkt durch eine Folge von Suchstrahlen abgetastet (Funktion ***raytrace***). Für jeden dargestellten Bildpunkt (Pixel) ist mindestens ein Suchstrahl erforderlich. Um statische *Aliasing-Effekte* zu reduzieren wird die Szene durch den Suchstrahl mit doppelter Auflösung abgetastet, d.h. jedes Bildelement wird aus dem Ergebnis von 9 Suchstrahlen gemittelt. Zur Bestimmung der Farbintensität eines Pixels (Rekursive Funktion: ***light_int***) ist es notwendig, den Suchstrahl mit allen Objekten der Szene zu schneiden. Da in KISMET ein Polyeder-Datenmodell benützt wird, reduziert sich die Suchoperation auf den Schnitt zwischen einer Geraden und konvexen Facetten (FACE).[12] Die Suche kann für den Fall des in KISMET optional ebenfalls ausführbaren *Ray-Casting* (Rekursionstiefe = 1) durch die Anwendung des *Z-Buffers* erheblich beschleunigt werden.

---

12 Für die "schnelleren" Echtzeit-Zeichenroutinen werden nicht-konvexe Facetten und solche mit *Inner Loops* (Löcher) während der Daten-Initialisierung in konvexe Teilstücke zerlegt.

In der vorliegenden Implementierung wird die Suche dadurch **optimiert**, daß :

1. zur Voruntersuchung eines jeden Polyeders der Schnitt mit dessen **einhüllender Kugel** geprüft wird.
2. gesamte Baugruppen (FRAME) mit einhüllenden Kugeln umgeben sind; daher ist nur ein geringer Teil der Geometrie-Elemente (GEO) zu untersuchen.
3. die **Szenen-Kohärenz** ausgenützt wird; es ist sehr wahrscheinlich, daß ein Suchstrahl das gleiche GEO und FACE schneidet wie der vorangegangene Suchstrahl. Die "Treffer" (GEO und FACE) werden daher für nachfolgende Suchen in einem 2D-Feld gespeichert. Die Größe des Feldes ist entsprechend der maximalen Rekursionstiefe (1. Dimension) und der Maximalzahl der Lichtquellen (2. Dimension) gewählt. Zu Beginn der jeweiligen Suche werden daher zuerst die vorhergehenden "Treffer-Objekte" abgeprüft.

Trifft der Suchstrahl auf einen Geometriekörper, so wird der Abstand zwischen Schnittpunkt und Betrachterstandpunkt gespeichert (das Objekt mit dem geringsten Abstand ist sichtbar). Zur Berechnung der Lichtintensität im Schnittpunkt werden die Verbindungvektoren zu allen Lichtquellen ("Beleuchtungsvektoren") untersucht, d.h. der Objektdatenbaum muß erneut abgearbeitet werden. Falls kein Schnitt des "Beleuchtungsvektors" mit einem anderen GEO-Element erfolgt, trägt die Lichtquelle mit ihrer definierten Intensität zur Beleuchtung des Schnittpunkts bei, ansonsten liegt der Punkt im Schatten der Lichtquelle und der Suchalgorithmus kann abbrechen.

Im Schnittpunkt des Suchstrahls mit der sichtbaren Facette muß außerdem bei gekrümmten Oberflächen (notwendig bei Kugel, Zylinder, Rotationskörper usw.) der approximierte Normalenvektor bestimmt werden. Dazu wird in der Funktion **nw_ein_vek** die Normale nach *Gouraud* [74] aus den bei der Modell-Initialisierung exakt berechneten und zwischengespeicherten Normalen der Facetten-Eckpunkte interpoliert.

Bei hochglänzenden Oberflächen trägt neben den Lichtquellen, der von anderen GEO-Elementen eingespiegelte Beleuchtungsanteil zur Farbintensität eines Schnittpunkts bei. Zur Berechnung dieses Anteils verzweigt die Funktion ***light_int*** rekursiv, d.h. ein weiterer Suchvektor wird erzeugt und verarbeitet. Transparente Oberflächen erzeugen sowohl beim Eintritt in das durchsichtige "Material" als auch beim Austritt rekursiv jeweils zwei weitere Suchstrahlen (eingespiegelter Anteil und durch die transparente Fläche eintretender Anteil), wobei zusätzlich die **optische Brechung** des Lichtstrahls berücksichtigt wird. Das rekursive Verzweigen wird beendet, falls ein Lichtstrahl die vordefinierte Mindestintensität (1 % des Maximalwerts) unterschreitet, oder bei Überschreiten der maximalen Rekursionstiefe (vordefinierter Wert: 15).

Die in Abbildung 25 dargestellte Szene wurde mit einer sichtbaren Auflösung von 1280x1024 Pixel berechnet, d.h. 2561x2049 Primär-Suchstrahlen (Antialiasing) wurden hier verarbeitet. Die Szene (*74* GEO-Elemente, *42* FRAMEs, *9366* dargestellte FACEs) enthält insgesamt 7 Lichtquellen ("Spot"-Lampen), die mit unterschiedlicher Lichtintensität und Farbe den Raum beleuchten. Bedingt durch die verspiegelten Wände und Objekte, sowie durch den gewählten Betrachterstandpunkt beträgt die durchschnittliche Rekursionstiefe *1.4473*. Die maximale Rekursionstiefe dieser Szene beträgt *15*, bedingt durch die Absorption der Spiegeloberflächen (jeweils 80% Reflexion, d.h. 20% Absorption). Die Rechenzeit betrug etwa 4331 Min. (72 Std. 11 Min.).[13]

---

[13] Diese Rechenzeit wurde auf einem Einprozessor-System *Silicon Graphics IRIS 4D/210-VGX* erreicht (CPU-Taktrate 25 MHz).

Beschleunigung der Bildberechnung für Ray-Casting
Wie bereits oben erwähnt, ist für **Ray-Casting** eine erhebliche Beschleunigung der Bildberechnung bei entsprechender Ausstattung des Bildspeichers möglich. Dabei beschränkt sich jedoch der mögliche "Bildrealismus" auf die Berechnung von *Schatten*, d.h. Spiegeleffekte und "echte" Transparenz (Lichtbrechung, Totalreflexion) können mit der nachfolgend vorgeschlagenen Technik nicht berücksichtigt werden.

Die Szene wird dabei in einem ersten Verarbeitungsschritt unter Benutzung des Tiefenspeichers in den Hintergrund-Bildspeicher geschrieben. Statt der Farbinformation wird der sequentielle Index der Facette im Bildspeicher abgelegt. In jedem Bildpunkt des Hintergrund-Speichers ist daher am Ende dieses ersten Schritts der Index der *sichtbaren* Facette gespeichert.[14] Im zweiten Verarbeitungsschritt erfolgt die eigentliche Bilderzeugung im Vordergrund-Bildspeicher. Anstatt den Suchstrahl mit der Liste aller Geometriekörper zu schneiden, erfolgt hier die Identifikation durch Auslesen des Index aus dem Hintergrund-Bildspeicher. Zur Berechnung der Lichtintensität ist mit der vorhandenen Hardware nach wie vor die geometrische Schnittoperation mit der Liste der Geometriekörper (Schatten) erforderlich. Eine Beschleunigung ist allerdings auch möglich, falls für jede definierte Lichtquelle ein weiterer Bildspeicher verfügbar wäre. In diese Bildspeicher würde die Szene, von der jeweils zugeordneten Lichtquelle aus betrachtet, geschrieben. Falls der Auftreffpunkt des Suchstrahls von der Lichtquelle aus betrachtet "sichtbar" (Vergleich der FACE-Indizes) ist, so trägt diese Quelle mit ihrer definierten Intensität zur Beleuchtung des Schnittpunkts bei, ansonsten liegt dieser Punkt im Schatten dieser Lichtquelle.

3.8.4 Kamerasimulation und -führung

Auch unter der optimistischen Annahme, daß die Leistung der Grafiksysteme in der Zukunft weiterhin so rapide zunimmt wie in den letzten Jahren, kann auf die Verwendung von "klassischen" Sichtsystemen (Röhrenkameras, CCD-Kameras) in der Fernhantierungstechnik nicht verzichtet werden. Als Gründe hierfür sind zu nennen:

- Die Konsistenz zwischen grafischem Modell und der realen Anlage muß wiederherstellbar sein (Modellierungsfehler, "Fallenlassen" von Teilen). Videokameras sind hier geeignete und - im Vergleich zu 3D-Laserscannern - relative kostengünstige Sensoren.

- Für hochgenaue bedienergeführte Operationen sowie für formverändernde Arbeiten (Schweißen, zerstörendes Trennen) fehlen im Grafiksystem noch:

  - Die notwendige Detailtreue. Diese wäre erzielbar auf Großrechnern, der Modellieraufwand steht jedoch in keinem Verhältnis zum Nutzen.
  - Geeignete Modelliermethoden, z.B. für den Materialauftrag an einer Schweißbahn.

- Kamerasysteme sind für Inspektionsarbeiten und aus zulassungstechnischen Gründen (Gesetzgebung) weiterhin notwendig.

Eine wichtige Anwendung der CAD-gestützten, grafischen Bedienerführung in der Fernhantierungstechnik ist daher die Unterstützung des Operateurs bezüglich der **Einstellung**, **Positionierung** und **Nachführung** der **Szenen-Beobachtungskameras**. Die Anwendung des synthetischen, CAD-modellbasierten 3D-Sehens und der grafischen

---

[14] Diese Technik wird auch zum "Picken" von Geometrie-Objekten in KISMET verwendet.

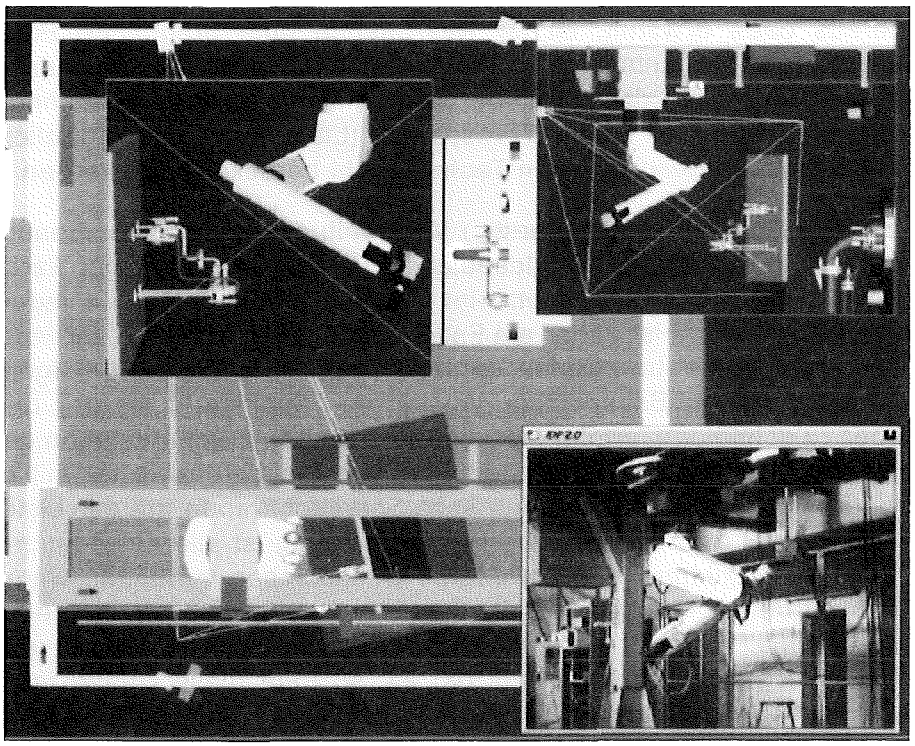


Abbildung 26. Kamerasimulation und On-Line-Positionierunterstützung: KISMET-Bildschirmfoto des CATROB-Teststandmodells. Die Sichtpyramiden der Beobachtungskameras sind in die Szene eingeblendet.

Bedienerführung ermöglicht aber eine deutliche **Reduzierung** der notwendigen **Anzahl** von "klassischen" Beobachtungskameras, besonders der Übersichtskameras.

Bezüglich des Kamera-Positionierungssystems lassen sich für HFH-Anwendungen zwei Klassen von Kamerasystemen unterscheiden:

1. **Kamerasysteme mit fester Basis.** Diese Kameras sind in der Regel an den Wänden oder an markanten Punkten der Arbeitszelle installiert. Über eine Schwenk-/Neige-Positioniereinrichtung lassen sich die Blickrichtung, über Zoom- und Fokussiermechanik der Bildausschnitt und die Bildschärfe einstellen.

2. **Bewegliche Beobachtungskameras.** Diese Kameras sind über bewegliche Arme mit dem Handhabungssystem verbunden und werden mit diesem in der Hantierungszelle verfahren. Diese Gruppe von Kamera-Positioniersystemen erlaubt zusätzlich zu den Charakteristika des vorgenannten Typs auch eine Änderung des Betrachterstandpunkts.

Die Kombination von "klassischen" Sichtsystemen (Videokameras) und der 3D-Computergrafik wird auch als **integriertes Sehen** [107] bezeichnet. Das Grafiksystem unterstützt den Bediener insbesonders durch folgende Funktionalität:

1. Schnelles Positionieren durch Definition des Zielpunkts in der CAD-Szene (CAMERA-Pointing).

2. Automatisches Nachführen (CAMERA-Tracking) der Kameras bei beweglichem Kamerastandpunkt, beweglichem Zielobjekt oder beidem.

3. Automatische Berechnung des Abstands zum Zielobjekt (Autofokus) und Nachführung des Bildöffnungswinkels (Autozoom).

4. Optionale Einblendung der Kamera-Sichtpyramide und der Schärfeebene in die 3D-Szene. Dem Operateur wird die Orientierung in der realen Szene erleichtert, besonders in komplexen und sehr diversifizierten Arbeitsumgebungen. Hierbei ist es oftmals allein aus dem Videobild schwer erkennbar, welches Bauteil oder welcher Bereich augenblicklich eingesehen wird.

5. Jede Kamera im Modell kann separat On- oder Off-Line geschaltet werden. Die o.g. Funktionalität ist in allen Betriebsarten von KISMET verfügbar, d.h. sowohl während des On-Line-Monitoring als auch im Off-Line-Simulationsbetrieb.

3.8.4.1 Modellierung von Szenenkameras in KISMET

In KISMET werden simulierte Kameraansichten (CAMERA) als ein spezieller Typus von "Workframe" (WFRM) modelliert. Die speziellen Eigenschaften einer CAMERA sind als funktionelle Attribute mit diesem Typ von WFRM assoziiert. Der Aufpunkt der Sichtpyramide und die Blickrichtung (entlang der positiven Z-Achse des WFRM) werden durch eine relative Transformation zum Bezugs-KS des WFRM definiert. Zur programminternen Datenverwaltung jeweils einer Szenenansicht wird der Datentyp *VIEWS* verwendet. Gespeichert werden unter anderem Betrachterstandpunkt, Blickpunkt, Bildöffnungswinkel, sowie die Projektions- und Darstellungs-Transformationsmatrizen.[15]

Die Darstellungsmatrix $\mathbf{V}_i$ der i-ten CAMERA wird berechnet nach der Beziehung:

$$\mathbf{V}_i = \left[\mathrm{Rot}_z(180)\, \mathrm{Rot}_x(180)\, \mathbf{M}_{w,i}\, \mathbf{M}_{k,i} \right]^{-1} \qquad (3.17)$$

wobei die Modellier-Transformation $\mathbf{M}_{w,i}$ die relative WFRM-Transformation zwischen dem Bezugs-KS und dem Sicht-KS, und $\mathbf{M}_{k,i}$ die absolute Matrix der Transformationskette zwischen Welt-KS und kinematischem Bezugs-KS bezeichnen.

Für die Kamera-Simulation wird außerdem der Datentyp *CAMERA* benutzt. Zusätzlich zu den für die Szenendarstellung notwendigen Daten von *VIEWS* werden noch Zeigerstrukturen zur Referenz auf den Datentyp *MPKIN* bereitgestellt. Ein Zeiger wird benutzt zur Definition des kinematischen Referenz-KS, ein weiterer zur Kennzeichnung des Ziel-KS bei der automatischen Zielverfolgung ("Tracking"). Die Kamera-Positioniereinrichtungen (Kamera-Arme, Schwenk-/Neigeköpfe) werden als normaler "Roboter" (Datenstruktur *ROBOT*) modelliert. Um die Anpassung an neue Applikationen zu vereinfachen, sind die Kommunikationsmodule für den Datenaustausch zwischen der Kamerasteuerung und KISMET als nebenläufiger UNIX-Prozeß implementiert. Der Datenaustausch zwischen KISMET selbst und diesen Schnittstellenmodulen ist über UNIX-Interprozeßkommunikation (IPC) realisiert.[16]

Abbildung 26 zeigt eine Darstellung des KFK-Experimentierstands CATROB [32] mit eingeblendeten Kamera-Sichtpyramiden. Im CATROB-Modell sind insgesamt vier Umgebungs-Beobachtungskameras, sowie eine vom Roboter gehaltene Meßkamera modelliert. Diese CCD-Meßkamera ist mit einem Bildverarbeitungssystem gekoppelt und wird in CATROB zur Vermessung der zu hantierenden Objekte (Flanschverbindungen) unter Anwendung eines Triangulationsverfahrens [15] benutzt. Die Simulation der Meßkame-

---

[15] Siehe auch "Mathematische Grundlagen homogener Transformationen" auf Seite 67

[16] Aus Effizienzgründen werden hierzu *UNIX-messages* verwendet.

ra-Ansicht unterstützt die Off-Line-Programmierung des Meßzyklus. Die simulierte Ansicht gestattet vor allem eine Abschätzung, ob sich für eine gewählte Meßposition die zu vermessenden Objekte innerhalb des Bildöffnungswinkels und des Schärfebereichs (60-90cm) der Meßkamera befinden. Bei der CATROB-Applikation werden zur Umsetzung der von KISMET berechneten Objektentfernung und Bildöffnungswinkel in die von der Kamerasteuerung benötigten Maschinenparameter für Zoom und Fokus Tabellen benützt. Es hat sich gezeigt, daß die verwendeten Videokameras innerhalb einer Baureihe in den Einstellparametern stark variieren, sodaß eine Umsetzung über eine analytische Beziehung nicht möglich war. Für jede Kamera wird eine Tabelle mit 64 Stützwerten benutzt, wobei zwischen den Stützwerten linear interpoliert wird.

Folgende **Operationen** werden zur Unterstützung der Kamerasteuerung durch das System bereitgestellt:

CAMERA-Pointing
Über die Maus und den Faden-Cursor wird in der Drauf- und Seitenansicht die gewünschte Kamera-Zielposition als 3D-Koordinate spezifiziert. Daraufhin werden die Gelenkvariablen der Kamera-Arme bzw. Schwenk-/Neigeköpfe, die Entfernung zum Zielpunkt und der Bildöffnungswinkel berechnet.
Die Zielentfernung kann zum Scharfstellen des Objektivs (Autofokus), der neuberechnete Bildöffnungswinkel für eine automatische Zoomfunktion verwendet werden. Der Öffnungswinkel wird dabei so berechnet, daß unabhängig von der Zielentfernung ein Raumausschnitt gleicher Größe dargestellt wird. Ein Objekt wird daher bei sich verändernder Entfernung zur Kamera in etwa gleichgroß in Bildschirmkoordinaten dargestellt, wobei sich jedoch die Perspektive ändert.

Ein weiterer wichtiger von KISMET berechneter Parameter ist bei der Verwendung von Kamera-Armen der **Bild-Rollwinkel** um die Sichtgerade. Dieser Rotationswinkel kann für eine automatische Horizontierung des Kamerabilds ausgewertet werden, um das "auf dem Kopf stehen" des Videobilds zu vermeiden. Der Kamerarm oder die Aufnahmeoptik ist daher so zu konstruieren, daß eine Rotation um die Sichtgerade möglich ist.

CAMERA-Tracking
Der Bediener wird zunächst aufgefordert ein Ziel-KS zu spezifizieren (Pick-Funktion). Das so definierte Koordinatensystem wird nun automatisch durch die Kamera verfolgt, wobei fortlaufend die in der "CAMERA-Pointing" Funktion beschriebenen Parameter berechnet werden. Die "Tracking-Funktion" kann selbstverständlich auch für mehrere Kameras gleichzeitig aktiviert werden.

Change Fovy/Dist
Diese Funktion erlaubt die Änderung des Bildöffnungswinkels (Fovy) und der Schärfeebene (Dist).

4. Kinematische Strukturmodelle und Lösungsverfahren

4.1 Grundlagen und Begriffe

Die **Kinematik** ist im Maschinenbau eingebettet in die theoretisch-mathematische Grundlage der Getriebelehre. Wesentliche Begriffe der Getriebelehre sind die kinematische Kette, der Mechanismus und das Getriebe.

Die kinematische Kette ist eine Verbindung starrer Körper über Gelenke. Der Mechanismus entsteht aus der Kette durch die Wahl eines Gestells. Werden für den Mechanismus ein oder mehrere Antriebe definiert, so entsteht das Getriebe [108]. Für die sensorgeführte Darstellung mechanischer Mehrkörpersysteme werden Bewegungen von Körpern bezogen auf ein Bezugskoordinatensystem als Funktion der Zeit betrachtet, ohne Berücksichtigung der Kräfte und Momente die diese Bewegung verursachen. Im englischen Sprachraum wird diese Problemstellung als "kinematics" bezeichnet. Eine mathematisch weitaus komplexere Betrachtungsweise, die **Kinetik** oder auch **Dynamik**, behandelt die Bewegung der Körper als Funktion der die Bewegung erzeugenden Kräfte und Momente.

4.2 Topologie kinematischer Ketten

Aus den mechanischen Grundbausteinen *Arm* (Körper) und *Gelenk* lassen sich beliebig komplexe Mechanismen konstruieren. Topologisch lassen sich derartige Strukturen als Graphenstruktur beschreiben [21], [24].

Den einfachsten Fall stellt die **offene, unverzweigte kinematische Kette** dar. Hierbei läßt sich jedem Armglied genau ein Gelenk zuordnen, wobei die Bewegungsachsen verzweigungsfrei hintereinander angeordnet sind. Je nach Anzahl und Anordnung der Bewegungsachsen unterscheidet man *redundante* und *nichtredundante* Systeme.
Nach Richtlinie VDI 2861 [109] gliedert sich die kinematische Kette einer Handhabungseinrichtung in Haupt- und Nebenachsen, sowie Werkzeuge (Greifer, Schweiß- und Trennvorrichtungen usw.). Unverzweigt offene Kinematiken sind häufig bei Industrierobotern und vielachsigen Gelenkarmsystemen (JET "Articulated Boom") anzutreffen.

Gehen von einem *Arm* mehr als zwei *Gelenke* aus, so liegt eine *kinematische Verzweigung* vor. Als Beispiel für eine **offene, verzweigte Kette** kann der *Mehrfinger-Greifer* genannt werden. Auch bei den in der Fernhandhabungstechnik oft in Transportersystemen integrierten Kamera-Armen handelt es sich um verzweigte kinematische Ketten.

Liegen in der mechanischen Wirkstruktur zyklisch geschlossene Graphen (Schleifen) vor, so werden diese als **geschlossene kinematische Ketten** bezeichnet. Bei diesen sind - im Gegensatz zu offenen Ketten - auch passive, d.h. nicht angetriebene Gelenke im Mechanismus enthalten. Beispiele sind in der Robotertechnik häufig in Form ebener Vier- oder Fünfgelenkgetriebe im Bereich der Achsantriebe zu finden. Besonders im Baumaschinenbereich (Bagger, Schaufellader usw.) werden an den hydraulischen Antrieben fast ausschließlich ebene kinematische Ketten zur Kraftübertragung benützt.

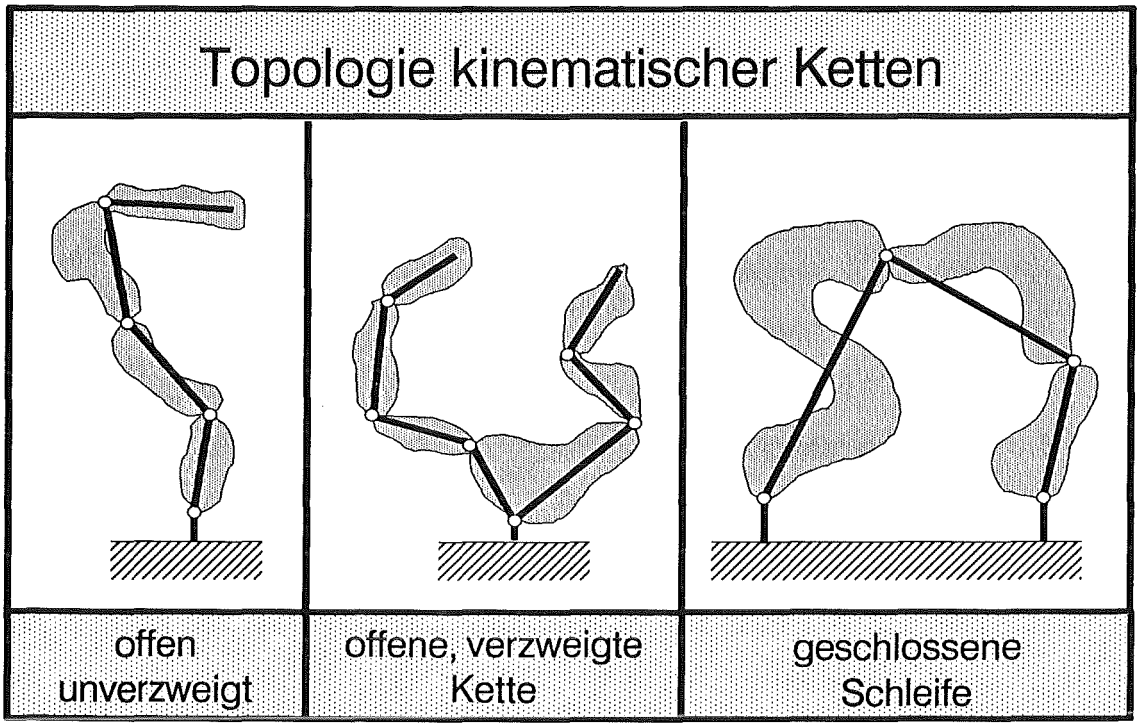


Abbildung 27. Kinematik Topologie: Eine Graphenstruktur erlaubt die Unterscheidung der Grundtypen kinematischer Strukturen.

4.2.1 Freiheitsgrad eines Mechanismus

Unter dem **Freiheitsgrad** eines Mechanismus versteht man die Anzahl der angetriebenen Gelenke, um für jedes Bindungsglied (Körper) des kinematischen Systems eine *unabhängige* Zwangsbewegung auszuführen.

Bei offenen Ketten ist die Anzahl der Freiheitsgrade des Gesamtsystems gleich der Summe der Freiheitsgrade aller Einzelgelenke. Bei geschlossenen Schleifen ist der Freiheitsgrad des Gesamtsystems aufgrund der kinematischen Zwangsführung geringer als die Summe der Einzelfreiheitsgrade.

Eine Beziehung für den Freiheitsgrad f eines Mechanismus gibt Sheth in [110] an, wobei j_k die Gelenkanzahl mit k Einzelfreiheitsgraden angibt, b ist die Anzahl der Bindungselemente (einschl. der Basis bzw. des Ständers) und f_0 ist der Freiheitsgrad des Betrachtungsraums (im 3D-Fall ist: $f_0 = 6$, in der Ebene gilt: $f_0 = 3$).

$$f = f_0 (b - 1) - \sum_{k=1}^{(f_0 - 1)} (f_0 - k) j_k \qquad (4.1)$$

In **singulären Zuständen** kann sich der Freiheitsgrad eines Mechanismus - in der Regel abnehmend - verändern. Außerdem kommt bei Montageaufgaben in der Praxis häufig vor, daß sich kinematische Strukturen ändern, z.B. bei Werkzeugwechseln oder bei Transportaufgaben von Elementen mit eigenen kinematischen Freiheitsgraden.

4.3 *Mathematische Grundlagen homogener Transformationen*

Die homogene Darstellung von Koordinaten im 3D Euklidischen Raum wird zur Entwicklung von Transformationsmatrizen benützt, die Rotation, Translation, Skalierung und Perspektive enthalten. Allgemein spricht man von homogener Koordinatendarstellung, wenn die *(n×n)*-Rotationsmatrix mit dem *(n+1)*-Positionsvektor in einer *((n+1)×(n+1))*-Transformationsmatrix dargestellt wird und damit die Multiplikation mit anschließender Addition in eine reine Multiplikation überführt werden kann [111].
Oberflächenpunkte auf geometrischen Körpern werden entsprechend dem in KISMET benützten Datenmodell beschrieben mit dem Positionsvektor $\mathbf{p}_i$ und der Flächennormalen $\mathbf{n}_i$ in homogenen Koordinaten der Form

$$\begin{aligned} \mathbf{p}_i &= (x_i, y_i, z_i, w_i) = (x_i, y_i, z_i, 1) \\ \mathbf{n}_i &= (x_i, y_i, z_i, 0) \end{aligned} \tag{4.2}$$

Geometrische Körper werden in einem körpereigenen Modellier-Koordinatensystem S_j definiert. Die Transformation in ein Bezugskoordinatensystem S_i erfolgt durch Multiplikation der im System S_j definierten Vektoren $\mathbf{p}_i$ mit der homogenen Transformationsmatrix $\mathbf{T}_{ij}$, die eine geometrische Abbildung von S_i nach S_j definiert.
Es werden - meist bedingt durch die Arbeitsweise der zur Verfügung stehenden Hardware - zwei verschiedene Systeme zur Notation der Koordinatentransformation benützt.

In **Prämultiplikativen Systemen** wird die Abbildungs-Transformation über die Operation

$$\mathbf{p}_i = \mathbf{p}_j \mathbf{A}_{ij}$$

bzw.

$$(p_{ix}, p_{iy}, p_{iz}, 1) = (p_{jx}, p_{jy}, p_{jz}, 1) \begin{bmatrix} b_{11} & b_{12} & b_{13} & 0 \\ b_{21} & b_{22} & b_{23} & 0 \\ b_{31} & b_{32} & b_{33} & 0 \\ b_{41} & b_{42} & b_{43} & 1 \end{bmatrix} \tag{4.3}$$

berechnet, wohingegen in **Postmultiplikativen Systemen** die Operation

$$\mathbf{p}_i^t = \mathbf{B}_{ij} \mathbf{p}_j^t$$

benützt wird. Die jeweiligen Abbildungsmatrizen sind einander vollständig äquivalent, und lassen sich durch Transponieren ineinander überführen, da gilt

$$\mathbf{A}_{ij} = \mathbf{B}_{ij}^t \tag{4.4}$$

Die bei der Implementierung von KISMET gewählte Notation entspricht dem **prämultiplikativen** Transformationssystem, da die gewählte Hardware-Plattform der Fa. *Silicon Graphics* entsprechend Gleichung (4.3) arbeitet. Das System basiert auf homogenen Transformationsmatrizen der Form

$$\mathbf{T}_{ij} = \begin{bmatrix} \mathbf{n} & 0 \\ \mathbf{s} & 0 \\ \mathbf{a} & 0 \\ \mathbf{p} & 1 \end{bmatrix} = \begin{bmatrix} n_x & n_y & n_z & 0 \\ s_x & s_y & s_z & 0 \\ a_x & a_y & a_z & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix} \tag{4.5}$$

Diese Form der 4×4-Transformationsmatrix beschreibt vollständig den Übergang von einem Koordinatensystem (KS) S_j zu dessen Bezugssystem S_i.
Die Vektoren $\mathbf{n}$, $\mathbf{s}$, $\mathbf{a}$ stellen die kartesischen Richtungsvektoren des Koordinatensystems S_j in den Koordinaten des Inertialsystems S_i dar. Der Vektor $\mathbf{p}$ gibt hier den

Ursprung des transformierten Systems S_j in den Koordinaten des Bezugssystems S_i an. Häufig wird - wie sich im weiteren zeigen wird - die Inverse zur Transformationsmatrix $\mathbf{T}_{ij}$ benötigt. Bei homogenen Matrizen ohne Skalierung und Projektion - d.h. die Abbildung ist rein aus Rotationen und Translation zusammengesetzt - ergibt sich die **Matrix-Inversion** zu

$$\mathbf{T}_{ij}^{-1} = \mathbf{T}_{ji} = \begin{bmatrix} n_x & s_x & a_x & 0 \\ n_y & s_y & a_y & 0 \\ n_z & s_z & a_z & 0 \\ -\mathbf{n}\,\mathbf{p}^t & -\mathbf{s}\,\mathbf{p}^t & -\mathbf{a}\,\mathbf{p}^t & 1 \end{bmatrix} \tag{4.6}$$

Für die grafische Simulation von Handhabungssystemen werden 3 Klassen von Transformationen benötigt,

- Modellier-Transformationen,
- Abbildungs-Transformationen,
- Projektions-Transformationen.

4.3.1 Modellier-Transformationen

Modellier-Transformationen werden benützt zur Bildung einer hierarchischen Geometrie- und Kinematik-Topologie [59], [60], [61] und [62]. Geometrische Primitive werden in einem für die Modellierung günstigen Koordinatensystem definiert. Diese Grundelemente werden zu Baugruppen (*Assembly*) zusammengefaßt, die wiederum eine Komponente einer komplexeren Baugruppe darstellen können. Die *räumliche Anordnung* (Position und Orientierung) der Grundkörper und dem Bezugskoordinatensystem einer Baugruppe, bzw. zwischen kinematisch gekoppelten Baugruppen wird über eine Kombination von Rotationen und Translationen hergestellt.

Für die Implementierung von KISMET werden die Grundoperationen *Translation* und *Rotation* in folgender Notation benützt [76] :

$$\mathbf{Trans}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix}$$

$$\mathbf{Rot}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Rot}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

$$\mathbf{Rot}_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Eine allgemein verwendbare Modellier-Transformation **T** mit 6 Freiheitsgraden erlaubt eine räumliche Anordnung von Objekten (Geometrie-Elemente und Baugruppen) relativ zu einem Bezugssystem, das nachfolgend als der **kinematische Vorgänger** bezeichnet wird. Für KISMET ist diese Transformation über jeweils drei Parameter für die Position $[a, b, c]$ und die Orientierung $[\alpha, \beta, \gamma]$ eindeutig definiert zu

$$\begin{aligned} \mathbf{T}(a, b, c, \alpha, \beta, \gamma) &= \mathbf{Rot}_z(\gamma) \times \mathbf{Rot}_y(\beta) \times \mathbf{Rot}_x(\alpha) \times \mathbf{Trans}(a, b, c) \\ &= \begin{bmatrix} c_\beta c_\gamma & c_\alpha s_\gamma + s_\alpha s_\beta c_\gamma & s_\alpha s_\gamma - c_\alpha s_\beta c_\gamma & 0 \\ -c_\beta s_\gamma & c_\alpha c_\gamma - s_\alpha s_\beta s_\gamma & s_\alpha c_\gamma + c_\alpha s_\beta s_\gamma & 0 \\ s_\beta & -s_\alpha c_\beta & c_\alpha c_\beta & 0 \\ a & b & c & 1 \end{bmatrix} \end{aligned} \tag{4.8}$$

wobei

$$s_\alpha \equiv \sin\alpha \;;\; s_\beta \equiv \sin\beta \;;\; s_\gamma \equiv \sin\gamma$$
$$c_\alpha \equiv \cos\alpha \;;\; c_\beta \equiv \cos\beta \;;\; c_\gamma \equiv \cos\gamma$$

Zu beachten ist, daß - entsprechend der Notation für prämultiplikative Grafiksysteme - die Transformationen in der Reihenfolge *trans*(a, b, c), gefolgt von *rot*$_x(\alpha)$, gefolgt von *rot*$_y(\beta)$, gefolgt von *rot*$_z(\gamma)$ ausgeführt werden.

Ein häufig auftretendes Problem ist die Berechnung der Abbildungsparameter $[a, b, c, \alpha, \beta, \gamma]$ aus einer gegebenen 4×4-Matrix. Dieses Problem tritt bei der Implementierung eines grafischen Simulators deshalb auf, da alle interaktiven Placierungs-Operation am effektivsten direkt über die entsprechenden Transformationen **Rot** bzw. **Trans** ausgeführt werden. Zur Rückspeicherung der interaktiv modifizierten Datenbasis werden die 6 Parameter benötigt. Für die allgemeine Abbildung in der für KISMET definierten Form liefert die analytische Lösung

$$a = t_{41} \;;\; b = t_{42} \;;\; c = t_{43} \;;$$

Fall 1: $\beta \neq \pm 90°$ (Bedingung: $t_{31} \neq \pm 1$)

$$\alpha = \arctan\left(\frac{-t_{32}}{t_{33}}\right) \;;\; \gamma = \arctan\left(\frac{-t_{21}}{t_{11}}\right) \;;\; \beta = \arctan\left(\frac{-t_{31}\sin(\alpha)}{t_{32}}\right) \tag{4.9}$$

Fall 2: $\beta = \pm 90°$ (Bedingung: $t_{31} = \pm 1$)

$$\alpha = 0 \;;\; \beta = \left\{\begin{matrix} 90° & \text{für} \;\; t_{31} = 1 \\ -90° & \text{für} \;\; t_{31} = -1 \end{matrix}\right\} \;;\; \gamma = \arctan\left(\frac{t_{12}}{t_{22}}\right) \tag{4.10}$$

Weitere Modellier-Transformationen wie *Spiegelung*, *Scherung* oder *Skalierung* [59] werden nicht benützt, da diese Operationen zu Matrizen führen können, die ungünstig für die Matrix-Inversion sind.

4.3.2 Abbildungs-Transformation

Die **Abbildungs-Transformation** placiert den Betrachter innerhalb des Modellkoordinatensystems und definiert somit die augenblickliche Szenenansicht. Die Abbildung wird stets über ein Punktepaar innerhalb des Modellsystems definiert, wobei der erste Punkt den Betrachterstandpunkt (auch virtueller Kamerastandpunkt) festlegt und über die

Verbindungsgerade zur zweiten Raumkoordinate die Blickrichtung berechnet wird. Ein weiterer Parameter *twist* rotiert das Bild um die Sichtachse.

Durch die Abbildungs-Transformation werden Geometrieobjekte vom Modellkoordinatensystem in das Betrachtersystem transformiert. Dort weist die x-Achse nach rechts, die y-Achse nach oben und die Blickrichtung zeigt entlang der negativen z-Achse.

Das genaue Verständnis der Abbildungs-Transformation wird benötigt bei Berechnungen zur Unterstützung der Videokamerasteuerung durch automatische Zielverfolgung bzw. zur Definition einer realen Szenenansicht durch Identifikation eines Zielpunkts im Simulationsszenario. Im letztgenannten Fall wird eine im simulierten Bild und somit im Betrachtersystem definierte Sichtgerade rückgerechnet auf einen Richtungsvektor im Modellkoordinatensystem, worauf im nachfolgenden Berechnungsschritt die inverse Kinematik-Transformation für die Schwenk-/Neigeköpfe der realen Kameras erfolgt.

Bei der Implementierung von KISMET wurde zur Abbildungs-Transformation ausschließlich die Operation

$$\mathbf{Lookat}(v_x, v_y, v_z, p_x, p_y, p_z, twist) = \mathbf{Trans}(-v_x, -v_y, -v_z) \times \mathbf{Rot}_y(\theta) \times \mathbf{Rot}_x(\phi) \times \mathbf{Rot}_z(-twist)$$

verwendet [76], mit

$$\begin{aligned}
\sin(\theta) &= \frac{p_x - v_x}{\sqrt{(p_x - v_x)^2 + (p_z - v_z)^2}} \\
\cos(\theta) &= \frac{v_z - p_z}{\sqrt{(p_x - v_x)^2 + (p_z - v_z)^2}} \\
\sin(\phi) &= \frac{v_y - p_y}{\sqrt{(p_x - v_x)^2 + (p_y - v_y)^2 + (p_z - v_z)^2}} \\
\cos(\phi) &= \frac{\sqrt{(p_x - v_x)^2 + (p_z - v_z)^2}}{\sqrt{(p_x - v_x)^2 + (p_y - v_y)^2 + (p_z - v_z)^2}}
\end{aligned} \tag{4.11}$$

4.3.3 Projektions-Transformation

Die **Projektions-Transformation** bildet Objekte vom Betrachtersystem entsprechend dem Prinzip der Lochkamera auf das Bildschirm-Koordinatensystem ab. Für **perspektivische Projektionen** wird das Modell auf eine 3D-Sichtpyramide, bei **orthogonalen Abbildungen** auf einen Sichtquader abgebildet. Lediglich jene Teile der Modellgeometrie innerhalb des definierten 3D-Sichtbereichs werden auf dem Bildschirm sichtbar dargestellt, alle Teile außerhalb der Darstellungsgrenzen werden abgeschnitten (*Clipping*).

Durch die Projektions-Transformation werden alle Koordinaten zusätzlich auf den Wertebereich $[\,-1, \ldots, +1\,]$ normalisiert. Details zum Aufbau der Projektionsmatrizen sind wiederum lediglich zum Verständnis der Unterstützungsfunktionen zur Kamerasteuerung wichtig. Eine detailliertere Beschreibung zur Funktionsweise des benützten Grafik-Subsystems ist in [76] zu finden.

Zur perspektivischen Abbildung werden die Transformations-Funktionen **Perspective** und **Window** benützt :

$$\textbf{Perspective}(fov, aspect, nearc, farc) = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & -1 \\ 0 & 0 & a_{43} & 0 \end{bmatrix} \tag{4.12}$$

wobei

$$a_{11} = \frac{\cot\left[\frac{fov}{2}\right]}{aspect} \quad ; \quad a_{22} = \cot\left[\frac{fov}{2}\right] \quad ;$$

$$a_{33} = \frac{farc + nearc}{farc - nearc} \quad ; \quad a_{43} = \frac{2 \times farc \times nearc}{farc - nearc}$$

oder

$$\textbf{Window}(leftc, rightc, bottomc, topc, nearc, farc) = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & -1 \\ 0 & 0 & a_{43} & 0 \end{bmatrix} \tag{4.13}$$

wobei hier gilt

$$a_{11} = \frac{2 \times nearc}{rightc - leftc} \quad ; \quad a_{22} = \frac{2 \times nearc}{topc - bottomc}$$

$$a_{31} = \frac{rightc + leftc}{rightc - leftc} \quad ; \quad a_{32} = \frac{topc + bottomc}{topc - bottomc}$$

$$a_{33} = \frac{farc + nearc}{farc - nearc} \quad ; \quad a_{43} = \frac{2 \times farc \times nearc}{farc - nearc}$$

Die Parameter sind wie folgt definiert :

fov Der seitliche Öffnungswinkel der Sichtpyramide

aspect Das Seitenverhältnis *Breite* zu *Höhe* der Projektionsfläche (der virtuellen Leinwand)

nearc, farc
Der Abstand zwischen Betrachterstandpunkt und vorderer bzw. hinterer Begrenzung des Darstellungsraums. Die Projektionsfläche befindet sich im Abstand *nearc* zum Betrachterstandpunkt.

leftc, rightc, bottomc, topc
Linker, rechter, unterer und oberer Kantenabstand der Projektionsfläche von der Sichtgeraden.

4.4 Abbildungs-Hierarchie und Matrix-Pipeline

Mit den in den vorangegangenen Abschnitten vorgestellten Transformationsmatrizen läßt sich eine Abbildungskette aufbauen, die sich in speziellen VLSI-Graphikprozessoren in Hardware technisch realisieren läßt [86]. Im Englischen wird diese Transformationskette als "Graphics-Pipeline" bezeichnet. Der augenblickliche Stand der Technik erreicht im Bereich der Grafik-Workstations der gehobenen Preisklasse (100-500 TDM) eine Leistung von ca. 500.000-1.000.000 3D-Transformationen pro Sekunde.

Die Abbildungs-Pipeline transformiert 3D-Vektoren $\mathbf{p}_m$ vom Modell-Koordinatensystem in normalisierte Bildschirmkoordinaten $\mathbf{p}_s$ über die Transformationskette

- Modellier-Transformation (engl. Modeling-Transformation) **M** ,
- Abbildungs-Transformation (engl. Viewing-Transformation) **V** und
- Projektions-Transformation (engl. Projection-Transformation) **P** .

Die von der Pipeline ausgeführte Operation lautet:

$$\mathbf{p}_s = \mathbf{p}_m (\mathbf{M V P}) \tag{4.14}$$

In der Robotersimulation eignet sich für die geometrische und kinematische Modellbildung das Verändern der Modellier-Transformation **M**, wobei sich eine Unterteilung von **M** in eine zeitkonstante **Geometrietransformation** $\mathbf{M}_G$ und in eine zeitveränderliche, von den Gelenkvariablen $\Phi_i(t)$ abhängige **Kinematik-Transformation** $\mathbf{M}_K$ anbietet. Die Abbildungskette lautet demzufolge

$$\mathbf{p}_s = \mathbf{p}_m [\mathbf{M}_G \mathbf{M}_K(\Phi(t)) \mathbf{V P}] \tag{4.15}$$

Diese Form der Modellbildung drängt den Gedanken an das biologische Analogon auf. Die Geometrie "hängt" ähnlich wie das Fleisch an einem kinematischen Skelett. Die räumliche Anordnung der Geometrie relativ zum kinematischen "Aufhänger"-System erfolgt durch die Geometrietransformation $\mathbf{M}_G$. Die Kinematik-Transformation $\mathbf{M}_{K,n}$ eines bewegten oder ortsfesten Segments läßt sich weiterhin entsprechend der kinematischen Hierarchie für das n-te Glied aufteilen :

$$\mathbf{M}_K(\Phi_1, \ldots, \Phi_n) = \mathbf{M}_{K,n}(\Phi_n(t)) \ \ldots \mathbf{M}_{K,2}(\Phi_2(t)) \, \mathbf{M}_{K,1}(\Phi_1(t)) \, \mathbf{M}_{Base} \tag{4.16}$$

Dabei gibt die Matrix $\mathbf{M}_{Base}$ den Übergang vom Ursprung des Weltkoordinatensystems zur Basis der betrachteten kinematischen Kette an.

4.5 Homogene Transformationen in kinematischen Ketten

Für die Verwendung homogener Transformationen in der Robotertechnik und für die kinematische Simulation hat sich die von Denavit und Hartenberg vorgeschlagene [112] Notation durchgesetzt, wobei deren Anwendung jedoch auf offene, unverzweigte Ketten beschränkt bleibt.
Eine allgemeinere, auch auf offene, verzweigte und geschlossene kinematische Schleifen anwendbare Notation wurde später von Sheth und Uicker [113] vorgestellt. Diese Notation gibt eine Transformationsvorschrift für Mechanismen an, die aus *niederen Elementenpaaren* [114] zusammengesetzt sind. Eine ausführliche Diskussion der verschiedenen Notationen findet man z.B. bei *Kandziora* [24].

4.5.1 DH-Notation nach Paul und deren Modifikation

Nach der Notation von *Denavit* und *Hartenberg* in der von *Paul* [111] modifizierten Form werden zur Beschreibung der relativen Achslagen der Gelenksysteme einer aus Schub- und Drehgelenken bestehenden Kinematik nur 4 Parameter je Bewegungsachse benötigt. Der Übergang vom Achssystem S_{i-1} zum nachfolgenden System S_i wird für das i-te Bewegungsgelenk durch die Größen $(a_i, \alpha_i, \theta_i, d_i)$ definiert. Die 4 Parameter werden ermittelt wie folgt; vgl. auch Abbildung 28 :

- Der Parameter a_i gibt den kürzesten Abstand von der z_{i-1}-Achse zur z_i-Achse an.
- α_i ist der Offsetwinkel von der x_{i-1} -Achse zur x_i-Achse, gedreht um die z_{i-1}-Achse.
- θ_i ist der Rotationswinkel von der z_{i-1} -Achse zur z_i-Achse, gedreht um die x_i-Achse.
- Der Parameter d_i definiert den Abstand vom Ursprung des Systems S_i zum Schnittpunkt der z_{i-1}-Achse mit der x_i -Achse.

Diese Formulierung läßt sich auch ausdrücken

$$\mathbf{T}^{i}_{i-1}(d_i, \theta_i) = \mathbf{Rot}_x(\alpha_i)\,\mathbf{Trans}(a_i, 0, 0)\,\mathbf{Trans}(0, 0, d_i)\,\mathbf{Rot}_z(\theta_i)$$

woraus die Transformationsmatrix entsteht :

$$\mathbf{T}^{i}_{i-1}(d_i, \theta_i) = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 & 0 \\ -\sin\theta_i \cos\alpha_i & \cos\theta_i \cos\alpha_i & \sin\alpha_i & 0 \\ \sin\theta_i \sin\alpha_i & -\cos\theta_i \sin\alpha_i & \cos\alpha_i & 0 \\ a_i \cos\theta_i & a_i \sin\theta_i & d_i & 1 \end{bmatrix} \tag{4.17}$$

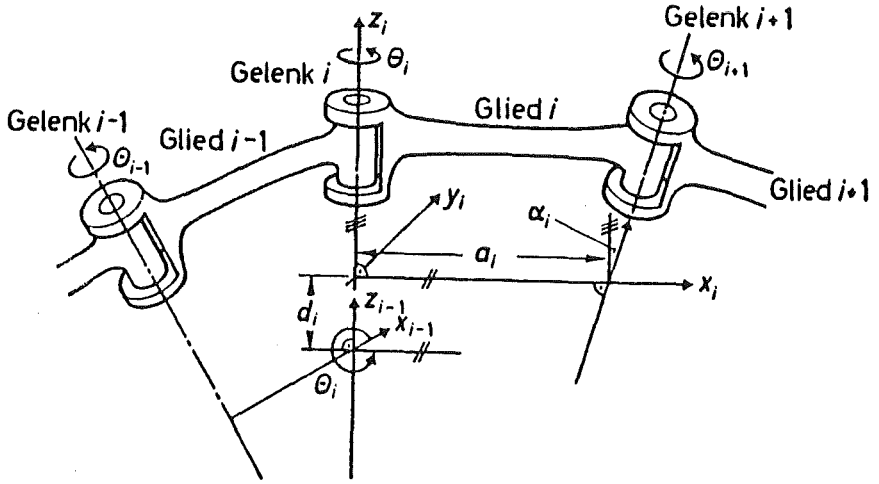


Abbildung 28. Rotatorisches Gelenk mit DH-Parametern: Darstellung nach *Paul* [111]

Werden von einer gegebenen DH-Matrix $\mathbf{T}^{i}_{i-1}$ die 4 Parameter $(a_i, \alpha_i, \theta_i, d_i)$ gesucht, so gelten für das i-te Gelenk die Beziehungen

$$\alpha_i = \arctan\left(\frac{t_{23}}{t_{33}}\right) \quad ; \quad \theta_i = \arctan\left(\frac{t_{12}}{t_{11}}\right) \tag{4.18}$$

und

$$d_i = t_{43} \quad ; \quad a_i = \begin{cases} \left(\frac{t_{11}}{t_{41}}\right) & \text{für} \quad \|t_{11}\| \geq 0.5 \\ \left(\frac{t_{12}}{t_{42}}\right) & \text{für} \quad \|t_{11}\| < 0.5 \end{cases}$$

Bei Schubgelenken stellt der Parameter d_i die Gelenkvariable dar, bei Drehgelenken ist der Parameter θ_i variabel. Zu beachten ist, daß die i -te Achse des Systems S_i um die Rotationsachse z_{i-1} das Systems S_{i-1} rotiert, bei Schubgelenken wird das System S_i parallel zur z_{i-1}-Achse verschoben. Die Bewegung des i-ten Systems wird demnach durch die relative Lage der z-Achse des (i-1)-ten Systems bestimmt.
Ein anschauliches Verfahren zur Ermittlung der DH-Parameter für vorgegebene Kinematiken ist in dem Artikel [115], sowie in [111] beschrieben.

Ein **Vorteil** der sich aus dem Einhalten der beschriebenen Regeln ergibt ist, daß die DH-Notation auf mathematisch einfach zu behandelnde Gleichungen führt, die besonders das Aufstellen von Gleichungssystemen zur expliziten Lösung des inversen Modells erleichtern.

Gravierende Nachteile der DH-Notation in der bisher dargestellten Form sind:

- Die vier DH-Parameter hängen nicht nur von der Gestalt des betrachteten Armkörpers ab, sondern auch von der Form des kinematischen Vorgängerglieds.
- Das Modellieren von kinematischen Verzweigungen in beliebiger Form ist nicht möglich.
- Simulationsmodelle mit mehreren kinematisch unabhängigen Mechanismen lassen sich nur umständlich realisieren, da die z-Achse der Roboterbasis gleichzeitig die Bewegungsrichtung des ersten Gelenks definiert.

Abhilfe schafft hier die Einführung einer allgemeinen Modellier-Transformation mit den 6 Parametern $(x, y, z, \alpha, \beta, \gamma)$

$$\mathbf{M}_i(x, y, z, \alpha, \beta, \gamma) = \mathbf{Rot}_z(\gamma) \times \mathbf{Rot}_y(\beta) \times \mathbf{Rot}_x(\alpha) \times \mathbf{Trans}(x, y, z) \qquad (4.19)$$

zur Definition der i-ten Bewegungsachse. Die z-Achse des durch diese **zeitkonstante Modellier-Transformation** gebildeten Systems definiert die Schub- oder Drehachse des kinematisch nachfolgenden beweglichen Gelenks im Simulationsmodell.

Zur **Definition eines kinematischen Skeletts** lassen sich daher folgende **Regeln** festhalten:

1. Bewegliche Armsegmente können über eine DH-Matrix $\mathbf{T}_{i-1}^{i}(d_i, \theta_i)$ definiert werden. Die Bewegungsachse ist festgelegt durch die Lage der z_{i-1} -Achse des Vorgängerelements.

2. Bewegungsachsen für nachfolgend definierte Segmente können über eine DH-Transformation $\mathbf{T}_{i-1}^{i}(d_i, \theta_i)$ oder durch eine allgemeine, zeitkonstante Modellier-Transformation $\mathbf{M}_i(x, y, z, \alpha, \beta, \gamma)$ definiert werden. Die z_i-Achse legt in beiden Fällen die Bewegungsrichtung des $(i+1)$-ten Koordinatensystems fest.

3. Die Roboterbasis und gleichzeitig die Bewegungsachse des ersten beweglichen Armglieds wird durch eine allgemeine Modellier-Transformation $\mathbf{M}_{Base}$ definiert.

4. Die Bewegungsachse einer kinematischen Verzweigung wird mittels einer Modellier-Transformation $\mathbf{M}_i$ definiert.

ROBOT-Datei	ROBOT-ID	Kinematik-ID
hitachi_1.mpc	HITACHI1	470

FRAME	Topologie			DH-/Achs-Parameter					
	Vorg.	Nachf.	Typ	α_i	β_i	γ_i | θ_i	x_i | a_i	y_i	z_i | d_i
REF	AUTO	-	50	-90	0	0	0	207	0
BAS	REF	-	50	0	0	0	0	0	750
P_1	BAS	-	0	-90	-	0	0	-	0
P_2	P_1	-	0	0	-	-90	600	-	0
P_3	P_2	-	0	0	-	90	850	-	0
P_4	P_3	-	0	90	-	90	0	-	0
P_5	P_4	-	0	0	-	0	0	-	100
P_help	P_1	-	50	0	-180	0	0	0	0
ziel	P_3	-	50	-180	0	0	-1070	0	0
P_7a	P_help	P_8a	2	0	-	90	220	-	0
P_8a	P_7a	ziel	2	0	-	0	600	-	0

FRAME	Bereich		Startparameter		
	Min_i	Max_i	$P_{start,i}$	$V_{max,i}$	$A_{max,i}$
P_1	-150.5	150.5	0	90	180
P_2	-225	45	0	90	180
P_3	-45	225	0	90	180
P_4	-85.5	105.5	45	90	180
P_5	-185.5	185.5	0	90	180
P_7a	-360	360	-90	-	-
P_8a	-360	360	-90	-	-

Tabelle 7. Kinematische Modellparameter des HITACHI PW-10

Tabelle 7 zeigt die kinematischen Modellparameter des in Abbildung 29 dargestellten 5-Achs Bahnschweißroboters *HITACHI PW-10*. Dieser Industrieroboter weist im Bereich des Schulterantriebs eine kinematische Schleife auf.

Alle Längenangaben sind in 'mm' definiert, die Winkelangaben im Gradmaß. Die Datenstruktur weist 5 aktiv bewegliche Antriebsachsen (*P_1* bis *P_5*) auf, sowie eine 4-Gelenk-Schleife. Diese ist durch die KS *P_3*, *P_help*, *ziel*, *P_7a* und *P_8a* definiert. Innerhalb der geschlossenen Kette werden die Gelenke *P_7a* und *P_8a* über die Zwangsführung[17] des Systems *ziel* passiv bewegt. Für sie ist deshalb jeweils das Nachfolge-System definiert.

---

[17] Siehe auch "Lösungsverfahren für ebene, kinematische Schleifen" auf Seite 77 und "Kinematische Modellierung von Anlagenkomponenten" auf Seite 32

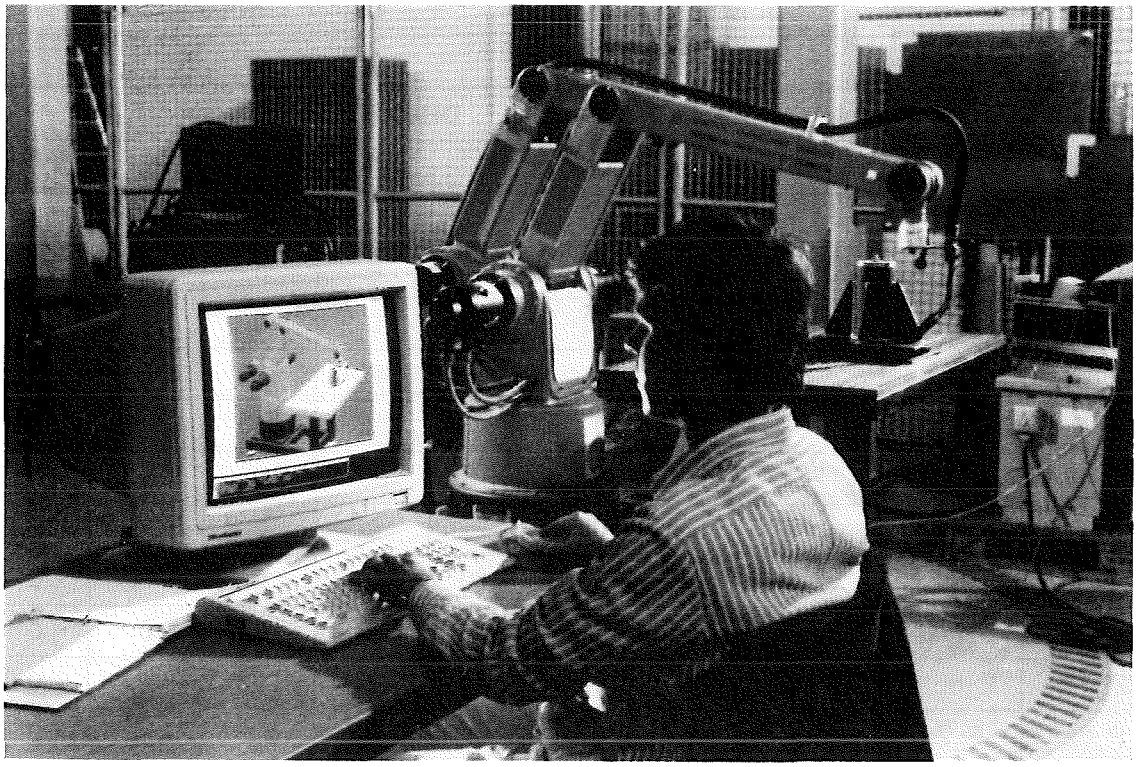


Abbildung 29. Monitorbetrieb mit KISMET: Das Foto zeigt den Autor bei der Überwachung eines ablaufenden Roboterprogramms im Monitorbetrieb mit KISMET. Das Foto wurde im Labor für Handhabungstechnik des KFK während der Vorbereitung eines Exponats für die "ESPRIT Technical Week 1988" in Brüssel/B aufgenommen. Der HITACHI PW-10 Industrieroboter führt hier im Hintergrund erkennbar den Bewegungsablauf eines Bahnschweißprogramms aus.

4.5.2 Kinematische Modellbildung

Das Problem der Koordinatentransformation bezogen auf die grafische Simulation besteht aus einer Reihe von Teilproblemen.

Im **sensorgeführten Monitorbetrieb** wird ausgehend von einem die aktuellen Achspositionen beschreibenden und von der Robotersteuerung empfangenen Zustandsvektor der Gelenkvariablen für die *N*-gelenkige kinematische Struktur *k* zu diskreten Zeitpunkten *i* ausgegangen.

$$\Phi_{ki} = (\Phi_{k1}, \Phi_{k2}, \ldots, \Phi_{kN})$$

Position und Orientierung des Werkzeugsystems (TCPF) $\mathbf{W}_{TCP,k}$ bezogen auf die Basis einer *N*-achsigen kinematischen Kette *k* läßt sich leicht in den Koordinaten eines Basissystems berechnen nach der Gleichung:

$$\mathbf{W}_{TCP,k} = \prod_{j=1}^{N} \mathbf{T}_{kj} \qquad (4.20)$$

Diese Beziehung gilt natürlich sinngemäß ebenfalls für verzweigte kinematische Ketten. Die gewählte Datenstruktur hat sicherzustellen, daß jedem Zweig der korrekte **kinematische Vorgänger** zugeordnet wird.

Betrachtet man die Werkzeugtransformation als Funktion der Zeit in Abhängigkeit von den Gelenkvariablen Φ_N so erhält man eine Beziehung der Art

$$\mathbf{W}(t) = \mathbf{W}(\Phi_1, \ldots, \Phi_N) \qquad \text{mit} \quad 1 \leq N \leq 6$$

Die Ableitung einer Komponente w_{ij} der Matrix **W** nach der Zeit erhält man über

$$\begin{aligned} \frac{dw_{ij}}{dt} &= \frac{\partial w_{ij}}{\partial \Phi_1} \frac{d\Phi_1}{dt} + \cdots + \frac{\partial w_{ij}}{\partial \Phi_N} \frac{d\Phi_N}{dt} \\ &= \sum_{k=1}^{N} \frac{\partial w_{ij}}{\partial \Phi_k} \frac{d\Phi_k}{dt} \end{aligned} \tag{4.21}$$

4.5.3 Lösungsverfahren für ebene, kinematische Schleifen

An Handhabungsgeräten findet man häufig im Bereich der Achsantriebe, als Massenausgleich bei MS-Manipulatoren oder in Greifhänden ebene, kinematisch bestimmte Gelenkvierecke. Diese bestehen aus vier rotatorischen Gelenken mit parallelen Drehachsen (Fall 1); vgl. Abbildung 11 auf Seite 26 und auch Abbildung 29 auf Seite 76. Bei Gelenken mit hydraulischen Antrieben oder in Spindelantrieben wird die Schleife aus drei Rotationsgelenken mit parallelen Drehachsen und einem Schubgelenk gebildet (Fall 2). Für den Monitorbetrieb (synthetisches Sehen) ist die Lösung dieser Fälle wichtig für die geometrische Transformation der Schleifenelemente, da ansonsten für die **Kollisionsuntersuchung** nicht die notwendige geometrische Genauigkeit gegeben ist.

Für die Behandlung dieser **ebenen, kinematisch bestimmten Schleifen** wurde in der vorliegenden Arbeit ein analytisch-geometrisches Lösungsverfahren ausgearbeitet. Dies hat gegenüber einem iterativen Interpolations- oder Approximationsverfahren den Vorteil einer konstanten, kalkulierbaren Rechenzeit und einer *geometrisch exakten Lösung*. Als Nachteil ist einzuräumen, daß die kinematische Modellierung dieser Schleifen nach einem festen Schema erfolgen muß. Auch kinematische Mehrfachschleifen können durch Kombination bzw. Kaskadierung der o.g. Basisfälle modelliert und gelöst werden. Weiterhin ermöglicht das beschriebene Verfahren das **interaktive Modellieren** und die **geometrische Analyse** von ebenen Schleifen in KISMET.

Die Lösung basiert auf dem Aufschneiden der Schleife in zwei offene Ketten nach dem **mechanischen Schnittprinzip**. Die gedankliche Auftrennung der Schleife kann prinzipiell in jeder Einzelachse durchgeführt werden, im implementierten Lösungsverfahren liegt die Trennstelle immer in einer der beiden rotatorischen Nebenachsen. Diejenige der beiden Ketten mit der sensor- oder bedienergeführten Hauptachse (Führungsachse) trägt im Modell ein zusätzliches *Führungskoordinatensystem*, dessen Z-Achse die Drehachse an der Schnittstelle der Schleife bildet und als **Zielsystem** für die **Zwangsführung** der zweiten offenen Kette fungiert.

Fall 1 (4 Drehgelenke) :
Im ersten Fall besteht die Schleife aus 4 Rotationsgelenken. Die Gelenkvariable θ_A der "Antriebsachse" *A* ist direkt gegeben durch den zugehörigen Achssensor bzw. durch die zugehörige Eingabefunktion. Die Rotationsachse des ersten Freiheitsgrades θ_1 (bewegtes Koordinatensystem: $\mathbf{T}_1$, mit Achsrichtungen und Ursprung $[\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1, \mathbf{p}_1]$) ist definiert durch die Z-Achse z_M des Modelliersystems $\mathbf{T}_M$ (DH-Konvention).

Weiterhin bekannt ist die Position des Zielsystems $\mathbf{T}_H = [\mathbf{x}_H, \mathbf{y}_H, \mathbf{z}_H, \mathbf{p}_H]$. Die Längen L_1 und L_2 der Armglieder entsprechen den DH-Parametern a_1 und a_2 und können daher - wie auch die anderen Dimensionierungsgrösen - direkt aus der Modell-Datenstruktur abgeleitet werden.

Die Gelenkvariable θ_1 ergibt sich aus :

$$\theta_1 = \beta_a + \beta_b$$

Zur Berechnung wird zunächst der Richtungsvektor **n** als die Lotrichtung auf die Gerade $[\mathbf{p}_M, \mathbf{z}_M]$ durch den Ursprung $\mathbf{p}_H$ des Zielsystems gebildet. Weiterhin wird der Abstand L_3 zwischen dem Ursprung des Zielsystems und der Drehachse berechnet. Der Winkel β_a ($\beta_a \geq 0$) wird nun über die Sätze am *schiefwinkligen Dreieck* [116] berechnet zu:

$$P = \frac{(L_1 + L_2 + L_3)}{2} \quad ; \quad R = \sqrt{\frac{(P - L_1)(P - {}_2)(P - L_3)}{P}}$$

und

$$\beta_a = 2 \bullet arctan\left(\frac{R}{P - L_2}\right)$$

Der Winkel β_b wird berechnet über :

$$c_b = \mathbf{n} \bullet \mathbf{x}_M \quad ; \quad s_b = \mathbf{n} \bullet \mathbf{y}_M \quad ; \quad \beta_b = arctan\left(\frac{s_b}{c_b}\right) \qquad (4.22)$$

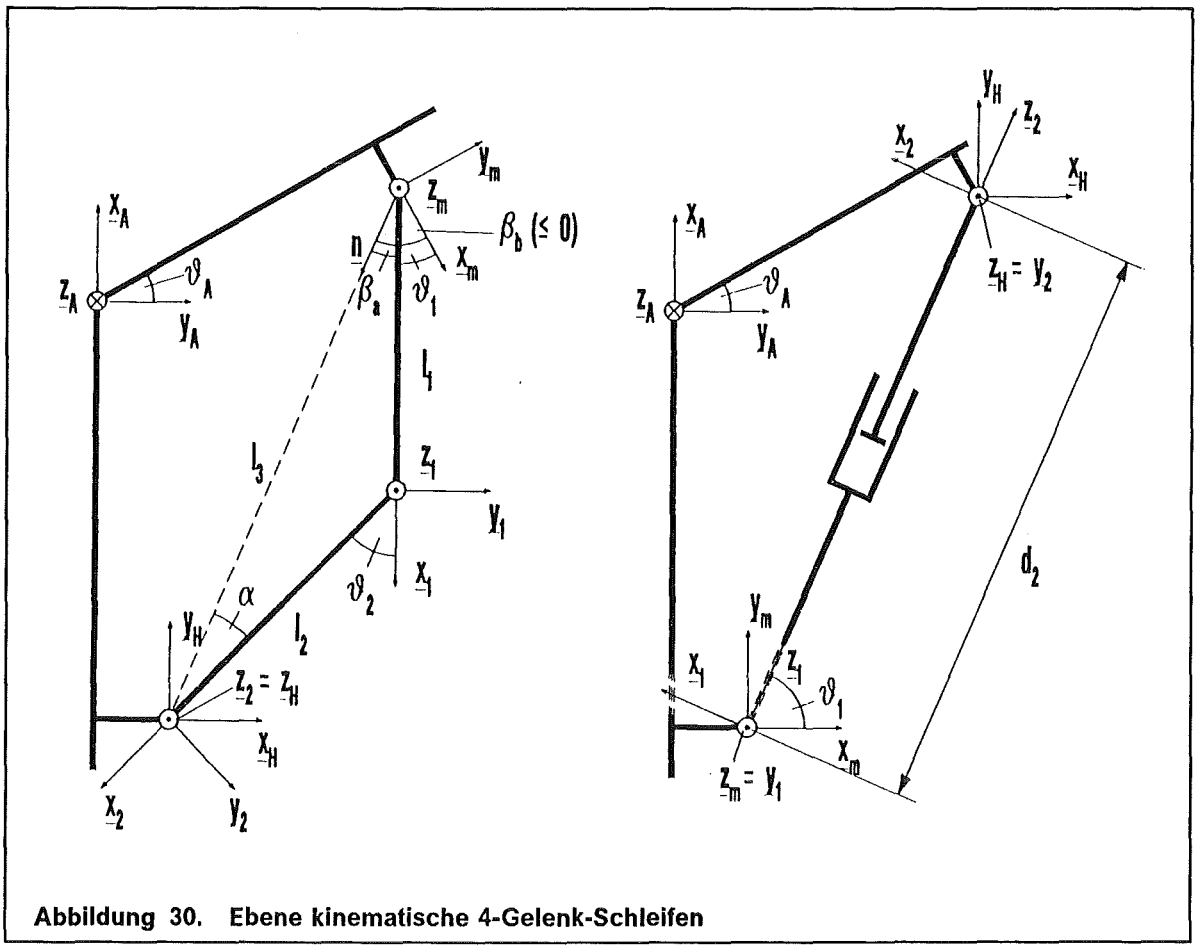


Abbildung 30. Ebene kinematische 4-Gelenk-Schleifen

Mit nunmehr bekanntem θ_1 erfolgt die absolute Transformation des Gelenksystems $\mathbf{T}_1$, sodaß im zweiten Schritt die Gelenkvariable θ_2 gelöst werden kann. Dazu wird zunächst die Richtungsgerade $\mathbf{x}_2$ gebildet als Lotrichtung vom Ursprung $\mathbf{p}_1$ des Systems $\mathbf{T}_1$ auf die Gerade $[\mathbf{p}_H, \mathbf{z}_H]$. Hierüber ergibt sich θ_2 zu :

$$c_2 = \mathbf{x}_2 \bullet \mathbf{x}_1 \quad ; \quad s_2 = \mathbf{x}_2 \bullet \mathbf{y}_1 \quad ; \quad \theta_2 = arctan\left(\frac{s_2}{c_2} \right) \tag{4.23}$$

Fall 2 (3 Drehgelenke, 1 Schubgelenk) :
Im zweiten Fall besteht die Schleife aus einem translatorischen und 3 rotatorischen Gelenken. Wieder ist die Gelenkvariable θ_A der Antriebsachse **A** gegeben (kinematisch bestimmtes System).: Gesucht werden die Parameter θ_1 des Drehgelenks und d_2 des Schubgelenks. Der Drehwinkel θ_1 läßt sich bestimmen über die Berechnung der Lotrichtung $\mathbf{z}_1$ auf die Drehachse $[\mathbf{p}_M, \mathbf{z}_M]$ durch den Ursprung $\mathbf{p}_H$ des Zielsystems $\mathbf{T}_H$. Der Richtungsvektor $\mathbf{z}_1$ bestimmt die Bewegungsrichtung des Schubgelenks (DH-Konvention).

Für θ_1 gilt :

$$c_1 = \mathbf{z}_1 \bullet \mathbf{x}_M \quad ; \quad s_1 = \mathbf{z}_1 \bullet \mathbf{y}_M \quad ; \quad \theta_1 = arctan\left(\frac{s_1}{c_1} \right) \tag{4.24}$$

Der Parameter d_2 der Schubachse wird berechnet als den Abstandsbetrag zwischen der Drehachse $[\mathbf{p}_M, \mathbf{z}_M]$ und dem Ursprung $\mathbf{p}_H$ des Zielkoordinatensystems. Für d_2 ergibt sich daher :

$$d_2 = \| (\mathbf{p}_H - \mathbf{p}_M) \times \mathbf{z}_M \| \tag{4.25}$$

5. Off-Line Programmierung und Simulation

Von ***On-Line-Programmierung*** spricht man dann, wenn die gesamte Programmierung der Bewegungsfolgen und des Programmablaufs an der Steuerung des Roboters, d.h. direkt am Einsatzort erfolgt [22], [117].

Im herkömmlichen **Teach-In-Verfahren** erfolgt die Programmerstellung mittels eines Handprogrammiergeräts (Teachbox). Bewegungskommandos werden nach Anfahren der Zielpositionen durch Eingabe des Bewegungsmodus nachträglich explizit abgespeichert. Kommandos zum Programmablauf werden an der Teachbox über spezielle Funktionstasten eingegeben. Das Teach-In-Verfahren wird hauptsächlich bei punktbezogenen Handhabungsaufgaben eingesetzt, beispielsweise bei Montageaufgaben oder dem Punktschweißen.

Eine weitere Methode der On-Line-Programmierung ist das **Play-Back Verfahren**, das in der Handhabungstechnik in heißen Zellen häufig zur Programmierung von Master/Slave-Manipulatoren (MSM) oder in der Industrierobotertechnik für die Programmierung komplexer Bewegungsfolgen mit geringen Genauigkeitsanforderungen (z.B. Lackierroboter) [20] eingesetzt wird. Bei der Play-Back-Programmierung werden im Gegensatz zum Teach-In-Verfahren nicht einzelne Punkte sondern Bewegungsbahnen eingelernt. Der Bediener führt dazu den Roboter über ein kinematisches Ersatzmodell (MS-Manipulatoren) oder mittels eines am Endeffektor angebrachten Handgriffs entlang der einzulernenden Bewegungsbahn. Der Effektor des Roboters (oder der Masterarm des MSM) wird durch den Bediener entlang der abzufahrenden Bahn geführt, wobei der Operateur durch die Aktuatoren des Manipulators unterstützt wird. Spezielle Sensoren, z.B. der DLR Kraft-Momenten-Sensor [118]-[119], wurden zur Aufnahme der Führungskräfte des Operateurs entwickelt. Die aktuellen Achskoordinaten werden durch die Gelenkwinkelsensorik in den Antriebsgelenken über die Steuerung erfaßt und in einem vordefinierten Zeit- oder Distanzraster abgespeichert. Weitere Verfahren zur Erfassung der Bewegungsbahnen sind optische Verfahren oder inertiale Meßsysteme [120]. Das Play-Back-Verfahren erlaubt in der Regel nur die Eingabe von Bewegungsbahnen und einfacher Steuerungsparameter die wie bei der Teach-In-Methode über die Funktionstasten eines Handprogrammiergeräts eingegeben werden. Kommandos zur Programmablaufsteuerung oder zur Integration von Sensorik sind in derartigen Steuerungen in der Regel nicht realisiert.

In der industriellen Praxis findet man auch den Fall, daß die Programmierung anstatt direkt in der Fertigungszelle in speziellen Programmierzellen erfolgt. Die spätere Arbeitsumgebung der Manipulatoren (Roboter) ist in diesen Handhabungszellen (Mock-Up) unter Verwendung von Originalkomponenten der Anlage bzw. der Fertigungseinrichtung, oder durch Holzmodelle nachgebildet.
Bei Anwendungen in kerntechnischen Anlagen liegt der Hauptnachteil der traditionellen On-Line-Programmierung in der Unzugänglichkeit der feindlichen Einsatzumgebung (radioaktive Strahlung) der Handhabungsgeräte.

Im Gegensatz zum On-Line-Verfahren erfolgt bei der ***Off-Line-Programmierung*** ([22], [23]) die Programmerstellung außerhalb und ohne Nutzung der Robotersteuerung in planenden Bereichen des Unternehmens, d.h. auf externen Rechnersystemen. Die Programme werden zur Abarbeitung aus Programmbibliotheken über eine Programmierschnittstelle in die Steuerung geladen.

Ansätze in Richtung Off-Line Programmierung basieren entweder auf der Entwicklung von Roboter-Programmiersprachen ([22], [121]-[122]), oder auf Verfahren und Systemen zur grafisch-interaktiven Programmerzeugung mit CAD-Systemen ([36] - [57]).

Werden Mischformen beider Verfahren angewendet, so spricht man von **hybrider Programmierung**. So können beispielsweise der Programmrumpf und die Bewegungskommandos Off-Line textuell erzeugt werden und die genauen Koordinaten der Zielpositionen durch Anfahren mit dem Roboter in einem speziellen Teach-Modus im On-Line-Verfahren nachträglich eingefügt werden.

5.1 Übersicht

Die **Off-Line Programmierung** ermöglicht die **prozeßentkoppelte Programmierung** von Handhabungsgeräten (Transporter, Manipulatoren, Sichtsysteme) und Fertigungseinrichtungen (Industrieroboter, Werkzeugmaschinen, Fördereinrichtungen) [50], [51]. Ein gemeinsames Merkmal der kommerziell verfügbaren Off-Line Programmiersysteme liegt in der Unabhängigkeit des verwendeten Rechnersystems von der Robotersteuerung eines bestimmten Anbieters (Erweiterung des potentiellen Absatzmarktes). In einem Simulationssystem läßt sich weiterhin durch simulierte Ausführung dieser Roboterprogramme der Ablauf **optimieren** hinsichtlich Bewegungsstrategie, Genauigkeit, Kollisionssicherheit und Ablaufgeschwindigkeit (Taktzeit).

Die Hauptvorteile der prozeßentkoppelten Off-Line Programmierung liegen in der:

- Herabsetzung von Gefahren für das Bedienpersonal und Maschinen durch Fehlbedienung
- Verminderung der Stillstandszeiten und dadurch höhere Auslastung der Anlagen und gesteigerte Effektivität
- Verbesserte Dokumentations- und Korrekturmöglichkeiten. Insbesonders die Anpassung umfangreicher Programme an die veränderte Geometrie der Arbeitszelle, sowie Änderungen und Erweiterungen in Folge einer Modifikation der Aufgabenstellung werden vereinfacht

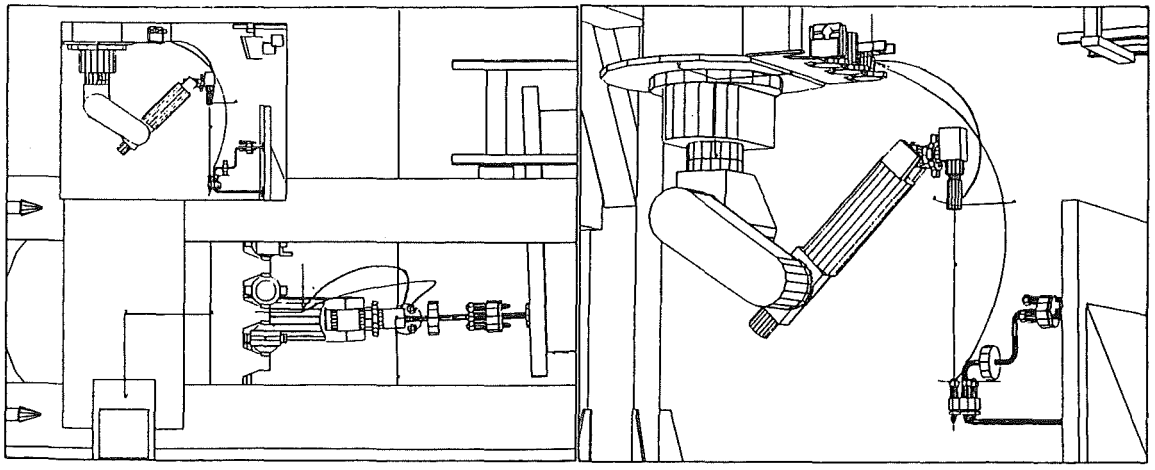


Abbildung 31. Darstellung der TCP-Bewegungsbahn

5.2 *Off-Line Programmierverfahren*

Die Einteilung der Off-Line-Programmierverfahren erfolgt aus der Sicht des Programmierers in **explizite** und in **implizite** (aufgabenorientierte) Methoden.

5.2.1 Explizite Programmierung

Bei den Methoden der expliziten Programmierung werden die Roboterbewegungen durch explizite Definition der Zielpositionen in Achs- oder Werkzeugkoordinaten (TCPF-Koordinaten), in der Regel als numerische Werte eingegeben.
Neuere Systeme gestatten auch eine symbolische Adressierung der Bewegungsziele und die Definition von **Datenlisten**, die vom eigentlichen Roboterprogramm separiert gespeichert werden. Eine Änderung der Zellgeometrie erfordert daher im Idealfall lediglich eine Anpassung der Datenlisten, jedoch keine Änderung im eigentlichen Programmcode.

Textuelle Programmierung
Unter *textueller Programmierung* versteht man die Generierung von Roboterprogrammen unter ausschließlicher Verwendung von Werkzeugen der textuellen Datenverarbeitung, in der Regel in einer höheren Roboter-Programmiersprache. Derartige Systeme lassen sich auf kostengünstigem Gerät (PC) realisieren.

Während sich die textuelle Programmierung ausgezeichnet für die Formulierung von Programmablaufstrukturen eignet, ergibt sich aus offensichtlichen Gründen das Problem der Eingabe von Zielkoordinaten der Bewegungskommandos. Bei der textuellen Programmierung ist daher auch für die Anwendung in der Praxis ein On-Line-Teach-Verfahren für die nachträgliche Definition der Zielpositionen erforderlich. Als weiterer Nachteil ist zu werten, daß dieses Verfahren Kenntnisse in einer Roboter-Programmiersprache und in einer Software-Entwicklungsumgebung erfordert.

Grafische Teach-In-Programmierung
Im Gegensatz zur textuellen Programmierung bereitet die Spezifikation von Bewegungszielen bei der *grafischen Teach-In-Programmierung* keine Schwierigkeiten. Das Programmiersystem bietet dem Bediener die grafische Darstellung einer Programmierbox. Die Programmeingabe erfolgt nach demselben Schema wie im *On-Line-Teach-In-Verfahren* . Die Eingabe von Programmbefehlen erfolgt durch Anklicken von Funktionsfeldern, die Umsetzung in die Roboter-Zielsprache wird durch das Programmiersystem vorgenommen. Die Zielkoordinaten der Bewegungskommandos werden aus dem rechnerinternen CAD-Modell abgeleitet.
Dieses Verfahren eignet sich, da keine Programmierkenntnisse erforderlich sind, auch für die Anwendung auf Werkstattebene oder für Handhabungs-Operateure. Als Nachteil ist zu werten, daß die Eingabe von komplexen Programmstrukturen nur bedingt möglich ist.

Für die praktische Anwendung in kommerziell verfügbaren Off-Line-Programmiersystemen haben sich daher **Mischformen** zwischen der textuellen Programmierung und der grafischen Teach-In-Programmierung als günstig erwiesen [50]. Die Erzeugung des Rumpfprogramms erfolgt in textueller Form, die Definition von Bewegungsfolgen als grafisches Teach-In. In einem weiteren Schritt wird aus Rumpfprogramm und Bewegungsfolgen ein ablauffähiges Roboterprogramm generiert.

5.2.2 Implizite Programmierung

Im Gegensatz zur *expliziten Methode* beschreibt die **implizite Programmierung** komplexe Roboteraufgaben durch eine Aneinanderreihung von Roboter-Teilaufgaben (Tasks) anstatt durch explizite Eingabe einzelner Bewegungskommandos [123], [124]. Deshalb wird diese Form der Roboterprogrammierung auch **aufgabenorientierte Programmierung** (*Task-Level Programming*) genannt. Roboteraktionen werden in derartigen Systemen implizit durch :

1. aufeinanderfolgende Zustände in einem Weltmodell (deklarative Form)
2. oder durch höhere Aktionsanweisungen (prozedurale Form)

beschrieben [125].

Die implizite Programmierung benötigt eine präzise, rechnerinterne Definition der Aufgabe (Task) unter Einbeziehung eines Geometrie-Modells der hantierten Objekte und deren Arbeitsumgebung, sowie der topologischen Zusammenhänge. Teilabläufe der Hantierungsaufgabe sind im Programmiersystem typischerweise als sogenannte **Elementaroperationen** vordefiniert.

Bei den bisher realisierten Ansätzen der aufgabenorientierten Programmierung [126] wird der Implementierungsaufwand von der Ebene der Anwenderprogrammierung durch das Bedienpersonal des Roboter-Endanwenders in die Ebene der Systemprogrammierung des impliziten Programmiersystems verlagert [127]. Ein Nachteil dieser Programmiersysteme ist darin zu sehen, daß die Implementierungen ausschließlich auf bestimmte Aufgabenkategorien (Leiterplattenbestückung, Baugruppenmontage), oder auf einen bestimmten Typ von Industrieroboter zugeschnitten sind. Der Anwender verliert die Flexibilität im Einsatz seiner Produktionsmittel (z.B. die Umdisposition eines Montageroboters für Palettieraufgaben) oder der Aufgabenlösung (z.B. der Einsatz eines typähnlichen Roboters eines anderen Herstellers für eine bestimmte Aufgabe).

Die implizite Programmierung wird daher in der industriellen Praxis - wenn überhaupt - durch Systemanbieter (Anlagenhäuser, Industrieausrüster) und weniger durch den Endanwender genutzt.

5.3 Programmiersprachen und -codes für Handhabungsgeräte

Die im Bereich der hochflexiblen Handhabungssysteme (HFH) eingesetzten Roboter-Programmiersprachen wurden - mit Ausnahme der Programmiercodes für die bei JET eingesetzten Systeme *Articulated Boom* [6] und *Telescopic Arm* (TARM) aus dem Einsatzfeld der Industrieroboter-Programmierung übernommen.

Die Zahl der bekanntgewordenen Entwicklungen von speziellen Programmiersprachen für Industrieroboter liegt nach Schätzungen im Bereich 100-200 [20]. *Schütze* klassifiziert die Robotersprachen in [20] entsprechend ihrer Mächtigkeit der Bewegungskommandos. Das Autorenkollektiv *Blume/Jakob* vergleicht in [121] verschiedene Robotersprachen entsprechend ihrer speziellen Charakteristika der für die Roboterprogrammierung relevanten Sprachkonstrukte. Die nachfolgende Übersicht beschränkt sich auf solche Sprachen, die in der industriellen Praxis oder in Forschungsvorhaben eine gewisse Verbreitung haben und aus diesem Grund für den Einsatz im handhabungstechnischen Bereich in Frage kommen.

Die Sprache **VAL** (Variable Assembly Language) wurde als roboterspezifische höhere Programmiersprache für die Steuerung der *PUMA*-Industrieroboter von *Unimation* ent-

wickelt [128]. *VAL* wurde später um Sprachkonstrukte für mathematische Funktionen, zur Sensordatenverarbeitung und Peripheriesteuerung, sowie um Befehle zur Steuerung des Programmablaufs ergänzt. Diese Erweiterung wurde **VAL-II** genannt [129] Die Sprachen *VAL* und *VAL-II* gehören zur Gruppe der *BASIC*-ähnlichen Sprachen. Die bei *Siemens* entwickelte Sprache **SRCL** (Siemens Robot Control Language) [130] wird vorwiegend eingesetzt in den RCM Steuerungen der Roboter-Hersteller *MANUTEC* und *KUKA*. Die zur Programmierung der Steuerung *rho2* der Firma *Bosch* eingesetzte Sprache **BAPS** [131] (Bewegungs- und Ablauf-Programmier-Sprache) zeichnet sich durch einen geringen Sprachumfang aus und erlaubt die kombinierte Programmierung von Robotersteuerungen und SPS. Weitere Beispiele für roboterspezifische industrielle Entwicklungen sind **KAREL** von *GMF*, **SIGLA** von *Olivetti* und **AML** von *IBM*.

Eine weitere umfangreiche Gruppe von Robotersprachen wurde im Umfeld von Forschungseinrichtungen entwickelt und eingesetzt.

PASRO (PAScal for RObots) wurde an der Universität Karlsruhe überwiegend aus didaktischen Gründen auf der Basis der Programmiersprache PASCAL entwickelt [122], um ein einfach erlernbares, leicht erweiterbares System für die studentische Ausbildung und Off-Line-Simulation zu schaffen. Inzwischen existiert eine weitere Version von PASRO die auf der Programmiersprache "C" basiert, von den Autoren PASRO/C genannt [121]. Im gleichen Umfeld wurde **SRL** (Structured Robot Language) [132] entwickelt. Eine weitere Programmiersprache für Roboter, **ROBEX** (ROBot EXapt), wurde an der RWTH in Aachen entwickelt [56]. Die Sprache ist strukturell und syntaktisch an die NC-Programmiersprache APT (Automatically Programmed Tools) angelehnt [133], einem de-facto Standard, und folgt semantisch dem NC-Programmiersystem EXAPT [23].

5.3.1 Kombinierte Simulations- und Programmiersprachen

Als Mischform zwischen Roboter-Programmiersprachen und Eingabesprachen für die Erzeugung von Computeranimationen und Simulationsabläufen kann man eine Klasse von höheren Programmiersprachen ansehen, die speziell für den Einsatz in grafischen Off-Line Programmiersystemen [51] entwickelt wurden. Diese Programmiersprachen sind nicht direkt ablauffähig auf kommerziellen Robotersteuerungen. Die Umsetzung in ablauffähige Roboterprogramme erfolgt nach dem Zwischenschritt der Simulation durch herstellerspezifische *Translatoren* und/oder *Compiler* [50].

Als typische Beispiele dieser Klasse von Simulationssprachen sind zu nennen :

- **TDL** (Task Description Language) der Firma TECNOMATIX wurde für den Einsatz im CAR-Produkt *ROBCAD* [48], [50]-[51] entwickelt. TDL ist ähnlich PASCAL eine strukturierte Sprache für den Entwurf modularer Programme (Firmeninformation).

- **GSL** (Graphical Simulation Language) der Firma DENEB basiert ebenfalls auf PASCAL und wird eingesetzt im CAR-System *IGRIP*. Neben Elementen zur Roboterprogrammierung enthält GSL Sprachkonstrukte zur Steuerung der Szenendarstellung ("Set Perspective", "Set User View", "Move Light", "Attach Eye Point") [52]-[54].

5.3.2 Der IRDATA-Code als Programmierschnittstelle

Der Roboter-Programmiercode **IRDATA** (Industrial Robot DATA) wurde als herstellerunabhängige *logische* Standard-Schnittstelle zwischen Programmierung und Robotersteuerung zunächst als VDI-Richtlinie 2863, später als Teil der DIN-Norm 66313 verabschiedet

[134]. Als Grundlage für IRDATA diente die NC-Programmierschnittstelle **CLDATA**, die als DIN-Norm 66215 bereits vorlag. In IRDATA wird ein Roboterprogramm als eine Folge von numerischen Code-Sätzen formuliert, die sequentiell durchnummeriert sind.

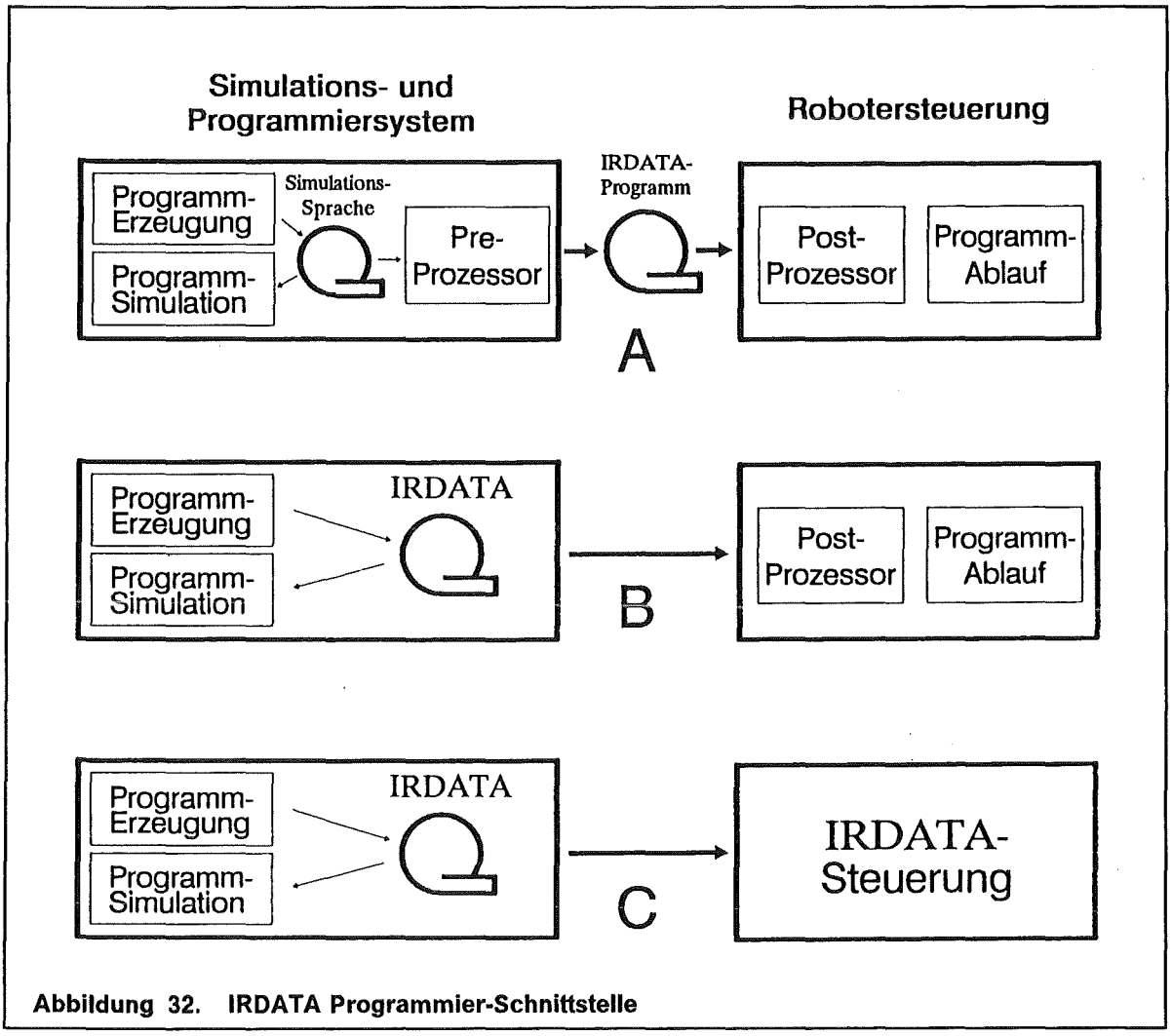


Abbildung 32. IRDATA Programmier-Schnittstelle

Die Anbindung eines Roboter-Programmiersystems an die reale Robotersteuerung kann über IRDATA nach den drei in Abbildung 32 dargestellten Alternativen erfolgen.

5.4 Off-Line Programmierung in KISMET

Für die Simulation von Roboterprogrammen wurde in KISMET ein IRDATA-Interpreter [134] integriert [135]. Außerdem steht für die textuelle Programmierung ein PASRO-Compiler [122] zur Verfügung. Der Übersetzer PASIR [135] setzt PASRO-Programme in IRDATA um, die wiederum in KISMET simuliert werden können. Durch die Implementierung von IRDATA sollte vermieden werden, daß für die verschiedenen Ziel-Robotersteuerungen eine Anzahl unterschiedlicher Programmiersprachen zu implementieren sind. Stattdessen werden die Robotersteuerungen entweder über bereits kommerziell verfügbare Softwaremodule oder durch entsprechende Eigenentwicklungen im KFK an IRDATA angepaßt.

5.4.1 Bahnplanung und Bewegungssteuerung

Bei der Untersuchung des Fahrverhaltens von Robotersteuerungen lassen sich die Bewegungsmodi [136] der marktgängigen Robotersysteme in die für die Simulation wichtigen Klassen der

- PTP-Bewegung
- CP-Bewegungen (Bahnsteuerverhalten)

einteilen.

1. **Punkt-zu-Punkt (PTP) Bewegung**
   Für das PTP-Bahnverhalten ist die Bewegung zwischen Start- und Zielpunkt die Bahnkurve undefiniert. Dabei wird zwischen **asynchronen** und **synchronen** PTP-Bewegungen unterschieden.

2. **Bahnsteuerverhalten (Linear- und Zirkularinterpolation)**
   Beim Bahnfahren (engl. CP, "Continous Path") werden die Achsen nach einem vorgegebenen funktionalen Zusammenhang bewegt, so daß ein definierbarer Punkt des Werkzeugs (TCP, Werkzeug-Bezugspunkt) nach bestimmten Geschwindigkeits-Zeit-Diagrammen die programmierte Bahn mit minimaler Abweichung abfährt. Die in der Praxis wichtigsten Bahnen sind die Gerade (Linearinterpolation) und der Kreisbogen (Zirkularinterpolation). Weitere mögliche Interpolationsarten sind :

   - Polynom-Interpolation [117], [25]
   - Interpolation über Spline-Funktionen (B-Splines, Bezier Splines) [137]

Die Bewegungsplanung laßt sich in zwei Teilaufgaben gliedern:

1. Die **geometrische** Bewegungsplanung, die aus Anfangs- und Endpunkt, sowie der durch das Roboterprogramm vorgegeben Interpolationsmethode (PTP- oder Linearverhalten) das Achs-Führungsverhalten bestimmt.

2. Die **zeitliche** Bewegungsplanung, in der mit den Parametern Beschleunigung, Geschwindigkeit und Verzögerung das Geschwindigkeits-Zeit-Diagramm berechnet wird.

5.4.1.1 Asynchrone PTP-Bewegung

Bei der asynchronen PTP-Bewegung werden die Roboterachsen unkoordiniert gesteuert. Alle Bewegungsachsen beginnen gleichzeitig mit dem Fahrzyklus, werden i.a. mit den programmierten Maximalgeschwindigkeiten und -beschleunigungen bewegt [117] und erreichen demzufolge ihre Endposition zu unterschiedlichen Zeitpunkten. Das Geschwindigkeits-Zeit-Diagramm eines Gelenks wird ausschließlich durch die Maschinenparameter für Beschleunigung, Geschwindigkeit und Verzögerung dieser einen Bewegungsachse bestimmt. Das asynchrone PTP-Verhalten hat den Nachteil, daß die Bewegungsbahn für den Roboter-Programmierer nur schwer vorhersehbar ist. Ihre Verwendung kommt nur in Frage, wenn keine Kollisionsgefahr besteht.

Für die Simulation von PTP-Bewegungen werden in KISMET symmetrische Bewegungsrampen angenommen, d.h. die Dauer des Beschleunigungsvorgangs entspricht der Dauer des Abbremsvorgangs. Da in KISMET die Simulation eines Roboter-Bewegungsvorgangs in Echtzeit erfolgen kann, wobei die Bildfolgefrequenz je nach dem vom Benutzer gewählten Darstellungsmodus und Detaillierungsgrad variieren kann, wird die gesamte Bewegung als Funktion der Zeit berechnet. Während der Berechnung des

nächsten Bewegungsschritts wird über die Systemuhr die seit Beginn des aktuellen Bewegungszyklus verstrichene Zeit ermittelt. Mit der Zeit als unabhängigem Parameter werden die Gelenkpositionen $\Phi_i(t)$ der i-ten Achse berechnet, woraus durch numerische Differentiation Geschwindigkeit und Beschleunigung gebildet werden können.

Folgende Parameter seien für die Planung einer asynchronen PTP-Bewegung durch die übergeordnete Steuerung und das bearbeitete Roboterprogramm gegeben:

$\Phi_{s,i}$	Startposition der Achse *i* in Gelenkkoordinaten
$\Phi_{e,i}$	Zielposition der Bewegung in Achskoordinaten
v_i	Geschwindigkeit des i-ten Gelenks
$v_{max,i}$	Maximale (programmierte) Achsgeschwindigkeit
$a_{max,i}$	Maximale Achsbeschleunigung, der Parameter entspricht betragsmäßig der maximalen Verzögerung

Für die Bewegungsplanung sind zu berechnen:

$t_{b,i}$	Dauer des Beschleunigungsvorgangs des i-ten Gelenks
$t_{v,i}$	Dauer des Abbremsvorgangs des i-ten Gelenks
$t_{k,i}$	Bewegungsdauer mit konstanter Geschwindigkeit
$t_{g,i}$	Bewegungs-Gesamtdauer
t_i	Gesamte Bewegungsdauer für die i-te Roboterachse
s_i	Betrag des von der i-ten Achse wärend der gesamten Bewegung überstrichenes Positionsintervall in Achskoordinaten
$s_{b,i}$	Überstrichens Positionsintervall während der Beschleunigungsphase
$s_{v,i}$	Überstrichens Positionsintervall während der Verzögerungssphase
$s_{bv,i}$	Überstrichens Positionsintervall während der Beschleunigungs- und Verzögerungssphase
$s_{k,i}$	Überstrichens Positionsintervall während der Bewegungsphase mit konstanter Geschwindigkeit

Für die i-te Achse gilt :

$$s_i = |\Phi_{e,i} - \Phi_{s,i}| \tag{5.1}$$

$$s_{bv,i} = s_{b,i} + s_{v,i} = 2 \cdot s_{b,i} = a_i \cdot t_{b,i}^2$$

Falls gilt $s_i > s_{bv,i}$, das heißt das gesamte zurückzulegende Bewegungsintervall ist größer als der während des Beschleunigungs- und Abbremsvorgangs zurückgelegte Weg, so folgt die Phase der konstanten Geschwindigkeit. Man spricht von einer *vollständigen Bewegung*. Andernfalls, d.h. es gilt $s_i \leq s_{bv,i}$, kann die vorgegebene Maximalgeschwindigkeit $v_{i,max}$ nicht erreicht werden und man spricht von der *unvollständigen Bewegung*.

Unvollständige Bewegung :

$$s_{b,i} = \frac{s_i}{2}$$

sowie

$$s_{k,i} = t_{k,i} = 0$$

und

$$v_i = \sqrt{s_i \cdot a_{i,max}}$$

Die gesamte Bewegungszeit berechnet sich zu :

$$t_{g,i} = t_{b,i} + t_{v,i} = 2 \cdot t_{b,i} = 2 \cdot \sqrt{\frac{s_i}{a_{i,max}}} \tag{5.2}$$

Vollständige Bewegung :

$$t_{k,i} = \frac{s_i}{v_{i,max}} \tag{5.3}$$

$$t_{g,i} = \frac{s_i}{v_{i,max}} + \frac{v_{i,max}}{a_{i,max}} \tag{5.4}$$

Die beiden Fälle können zur **Bahninterpolation** gleich behandelt werden, da für die unvollständige Bewegung $t_{k,i} = 0$ gilt. Die aktuelle Position $\Phi_i(t)$ des i-ten Gelenks berechnet sich während der Bahninterpolation

für $t \leq t_{b,i}$ zu :

$$\Phi_i(t) = \Phi_{s,i} + \frac{1}{2} a_{i,max} t^2 \tag{5.5}$$

für $t_{b,i} < t \leq (t_{b,i} + t_{k,i})$ zu :

$$\Phi_i(t) = \Phi_{s,i} + s_{b,i} + a_{i,max}(t - t_{b,i}) \tag{5.6}$$

für $(t_{b,i} + t_{k,i}) < t \leq t_{g,i}$ zu :

$$\Phi_i(t) = \Phi_{s,i} + s_{b,i} + s_{k,i} + v_i(t - t_{b,i} - t_{k,i}) - \frac{1}{2} a_{i,max}(t - t_{b,i} - t_{k,i})^2 \tag{5.7}$$

Im Regelfall wird die Endstellung für die verschiedenen Bewegungsachsen zu unterschiedlichen Zeitpunkten (asynchron) erreicht. Das beschriebene Verhalten führt bei der Anwendung auf mehrachsige Industrieroboter, Manipulatoren und redundante Vielgelenkarme zu markanten Knicken in der TCP-Bewegungsbahn (Unstetigkeit des Bewegungsgradienten).

5.4.1.2 *Synchrone PTP-Bewegung*

Bei der *synchronen PTP-Bewegung* werden die Roboterachsen so in ihrer Bewegung koordiniert, daß alle Gelenke *i* **gleichzeitig** die Beschleunigungsperiode $t_{b,i}$, die Bewegungsphase mit konstanter Geschwindigkeit $t_{k,i}$ (bei vollständigen Bewegungen) und die Verzögerungsphase $t_{v,i}$ beenden.

Es gilt für ein System mit *n* Achsen :

$$\begin{aligned} t_{b,0} &= t_{b,1} = \ldots = t_{b,n} \\ t_{k,0} &= t_{k,1} = \ldots = t_{k,n} \\ t_{v,0} &= t_{v,1} = \ldots = t_{v,n} \end{aligned}$$

und auch für die Gesamtbewegungszeit $t_{g,i}$:

$$t_{g,0} = t_{g,1} = \ldots = t_{g,n}$$

Die Bewegungs-Gesamtdauer richtet sich nach der langsamsten Achse. Die Bewegungsplanung für die synchrone PTP-Bewegung erfolgt im wesentlichen in drei Stufen:

1. Zunächst prüft das Planungsmodul, ob für alle Achsen das Geschwindigkeitsprofil der vollständigen oder unvollständigen Bewegung zu verwenden ist. Eine *unvollständige* Bewegung liegt dann vor, wenn für alle Achsen *i* gilt:

$$s_i \leq \frac{v_{max,i}^2}{a_{max,i}} \tag{5.8}$$

2. Im zweiten Schritt wird diejenige Bewegungsachse *m* ermittelt, die mit den vorprogrammierten Parametern für die Geschwindigkeit $v_{max,m}$ und die Beschleunigung $a_{max,m}$ die größte Bewegungs-Gesamtdauer $t_{g,max}$ benötigt. Die Berechnung erfolgt unter Berücksichtigung des in Schritt 1 ermittelten Bewegungsprofils nach Gleichung [5.2] bzw. nach Gleichung [5.4].

3. Im dritten Schritt werden für die einzelnen Bewegungsachsen aktuelle Achs-Geschwindigkeit $v_{akt,i}$ und -Beschleunigung (Verzögerung) $a_{akt,i}$ über Umformung der Gleichungen [5.2-5.4] berechnet.

Da bei PTP-Bewegungen die Bewegungsplanung auf Achsebene erfolgt, kann hier das von CP-Bewegungen bekannte Problem der *Singularitäten* nicht auftreten.

5.4.1.3 Lineares- und Zirkulares-Bahnsteuerverhalten

Bei *linearen* und *zirkularen CP-Bewegungen* erfolgt die Bahninterpolation im Werkzeug-Koordinatensystem (TCPF). Jeder Interpolationsschritt erfordert hier die Lösung des *inversen kinematischen Problems*.

Bei der *Linearbewegung* wird das TCPF von der Startposition $\mathbf{T}_S$ in die Zielposition $\mathbf{T}_Z$ entlang einer Geraden bewegt. Der Fahrzyklus läßt sich dem Prinzip nach wie bei der synchronen PTP-Bewegung berechnen und es gilt daher für das Geschwindigkeits-Zeit-Diagramm der Gleichungssatz (5.1)-(5.8), wobei jedoch hier die Interpolation über das TCP-KS erfolgt und die Variablen sinngemäß ausgetauscht werden müssen. Auf eine ausführliche Darstellung der Bewegungsgleichungen wird deshalb verzichtet. Die Bewegungsgeschwindigkeit richtet sich nach den im IRDATA-Programm prozentual definierten Maximalgeschwindigkeiten für Positions- und Orientierungsänderungen des TCPF. KISMET erlaubt die Positionsänderung mit gleichzeitiger Änderung der Orientierung, oder die getrennte Ausführung von Positions- und Orientierungsänderung. Das entsprechende Verhalten der simulierten Steuerung läßt sich für jeden Roboter im Datenmodell definieren.

Bei der *Zirkularbewegung* (Kreisbewegung) erfolgt die KISMET-interne Definition der Bewegungsbahn in vektorieller Form. Von der Kreisbahn gegeben sind Startpunkt (in der Kreisebene), Position (ebenfalls in der Kreisebene) und Richtung der Drehachse, sowie der zu überstreichende Winkel des Kreissegments. Die Bewegungsinterpolation erfolgt durch parametrische Änderung des Kreissegment-Winkels, wobei nach sinngemäßer Änderung wieder der Gleichungssatz (5.1)-(5.8) gilt. Die IRDATA-Definition der Kreisbahn-Parameter - die Bahn kann auch durch Angabe von Start- und Zielposition, sowie einem weiteren Punkt auf dem Kreisumfang definiert werden - wird während der geometrischen Bewegungsplanung in die beschriebene KISMET-interne Darstellung überführt.

Während der Simulation kann eine Reihe von potentiellen Ausnahmesituationen erkannt werden, die in der realen Steuerung zu einem Programmabbruch führen :

- Überschreiten der maximalen, oder Unterschreiten der minimalen (wichtig bei Schrittmotorantrieben) Geschwindigkeiten der einzelnen Bewegungsachsen bei vorgegebener Bahn-Sollgeschwindigkeit.
- Überschreiten des Verfahrbereichs der Einzelachsen.
- Das Erreichen von *singulären Positionen*. Diese treten bei Knickarmrobotern bei Änderungen des Winkelstatus innerhalb einer CP-Bewegung auf. Das Verhalten tritt also bei solchen Armpositionen auf, an denen die *Jakobimatrix* $\mathbf{J}(\Phi)$, bzw. deren Inverse $\mathbf{J}^{-1}$, singulär wird.
- Die Zielposition oder Teile der Bewegungsbahn liegen außerhalb des Roboter-Arbeitsraums.

5.4.2 Echtzeit Roboterprogramm-Ablaufsteuerung (Simulation)

Folgende Bedienfunktionen sind in KISMET zur Simulation von Roboterprogrammen[18] (TF) implementiert:

TF_LOAD Liest ein Roboterprogramm in die programminterne virtuelle Steuerung. Während des Lesens wird der TF auf Syntaxfehler untersucht. Im Fehlerfall wird die Programmzeile in einem separaten Textfenster markiert.

TF_RUN Startet die simulierte Ausführung eines TF. Mehrere TF können zur Simulation komplexer Hantierungsvorgäge gleichzeitig quasiparallel ablaufen (Multiprogramm-Ablaufsteuerung).

TF_DELETE Das spezifizierte Roboterprogramm wird aus der virtuellen Steuerung gelöscht und der zur Speicherung benutzte Hauptspeicher wird wieder freigegeben.

5.4.2.1 Spezielle Anforderungen in der Handhabungstechnik

Während des konzeptionellen Entwurfs und der Implementierung der Roboter-Programmierschnittstelle für KISMET wurden die nachfolgend aufgeführten Anforderungen beachtet:

1. Die parallele, konkurrierende Simulation von mehreren Robotern ist möglich. Ebenso werden in der Simulation die Synchronisation mehrerer Anlagenkomponenten, Wartezeitkommandos, Programm-Ablaufstrukturen und Programmbedingungen unterstützt.

2. Simuliert werden Punkt-zu-Punkt-Bewegungen (synchron und asynchron), lineare und zirkulare Bewegungen. Bei der Bewegungsplanung werden Geschwindigkeits- und Beschleunigungsprofile der einzelnen Achsen, Armkonfigurationen, sowie Bewegungsbeschränkungen berücksichtigt.

3. Die Simulation kann entweder in Echtzeit erfolgen, d.h. die Simulation des Roboterprogramms benötigt die gleiche Ausführungszeit wie die reale Programmabarbeitung in der Robotersteuerung, oder in einer spezifizierbaren Taktrate. Der

---

[18] In der Fernhantierungstechnik ist der englische Begriff *Teachfile* (TF) gebräuchlicher als die Bezeichnung *Roboterprogramm*.

letztgenannten Modus ermöglicht eine Messung des Roboter Arbeitstakts. Die Simulation in "Realzeit" ist wichtig zur Beurteilung der dem Operateur im Automatikbetrieb für Noteingriffe in den Programmablauf zur Verfügung stehenden Reaktionszeit (Operateur-Training).

5.4.2.2 *Virtuelle IRDATA-Steuerung*

Abbildung 33 zeigt die für KISMET implementierte Architektur einer virtuellen Multitasking-Roboterprogramm-Ablaufsteuerung. Die Roboterprogramm-Simulation basiert auf einer KISMET-internen Datenstruktur (I-Code), die von der Ablaufsteuerung während der Programmausführung interpretiert wird. Entsprechende Lesemodule bilden die Schnittstelle zwischen unterschiedlichen Robotersprachen bzw. -codes und dem I-Code.
Der I-Code entspricht dem Funktionsumfang des IRDATA-Standards mit geringfügigen Einschränkungen, aber auch funktionellen Erweiterungen. Als Beispiel für eine Erweiterung sei hier die Einbettung von KISMET-SCRIPT-Kommandos in den I-Code Kommentarsatz angeführt:

```
00011,21400,16,4,0,132,1,578.0,132,1,306.5,132,1,-10.8,
                  132,1,0.0,132,1,0.0,132,1,-180.0;
00012,5000,344,16,4,0,1,0,1,0;
00013,1000,8,'Greife Schlagschrauber';
00014,1000,9,'%CONNECT Traegersystem.WKZ_BAS.Schlagschrauber
                  TO Traegersystem.J_2.HAND;
00015,21400,16,4,0,132,1,940.0,132,1,306.5,132,1,-10.8,
                  132,1,0.0,132,1,0.0,132,1,-180.0;
00016,5000,344,16,4,0,1,0,1,0;
```

Das dargestellte IRDATA-Codesegment führt zwei Linearbewegungen zu (Sätze 11, 12) und von (Sätze 15, 16) der Werkzeugwechselposition des End-Effektors *Schlagschrauber* durch. Die Sätze 13 und 14 sind Kommentarsätze, wobei in Satz 14 eine SCRIPT-Anweisung für das Kinematik-Modul von KISMET eingebettet ist.

Die nachfolgende Beschreibung der I-Code-Ablaufsteuerung bezieht sich auf den Funktionsumfang des IRDATA-Lesemoduls.[19] Eine vollständige Tabelle aller in KISMET implementierten IRDATA-Befehlssätze kann [135] entnommen werden.

Das **Lesemodul** prüft das IRDATA-Programm auf syntaktische Fehler und wandelt die einzelnen IRDATA-Sätze in eine programminterne Datenstruktur um (I-Code). Zum Aufbau dieser Datenstruktur werden die Datentyp-Deklarationen *TC_CNTL* und *TEACHDAT* verwendet. Je TF wird ein *TC_CNTL*-Datensatz dynamisch zugewiesen und in einer doppelt gezeigerten Liste gespeichert. Diese Liste wird von der Multiprogramm-Ablaufsteuerung während eines Darstellungszyklus durchlaufen. Die *TC_CNTL*-Struktur enthält Zeiger-Referenzen auf die ROBOT-Datenstruktur und auf Symboltabellen von Datenlisten. Weiterhin werden hier Kellerstrukturen (*Stack*) für die Abarbeitung von Unterprogrammen und für *blockrelative-Adressierung* verwaltet.
Einzelne TF-Befehlssätze werden in einer doppelt gezeigerten Liste (*dual-linked list*) der *TEACHDAT*-Datenstruktur gespeichert. Hierin weist der Zeiger *next* auf den sequentiell folgenden Satz, der Zeiger *last* auf den vorhergehenden Satz und *targ* wird für Sprung-

---

19 Die weiteren implementierten Programmiercodes (z.B. der bei JET benutzte Programmiercode für den "Articulated Boom") besitzen einen weitaus geringeren Funktionsumfang als IRDATA.

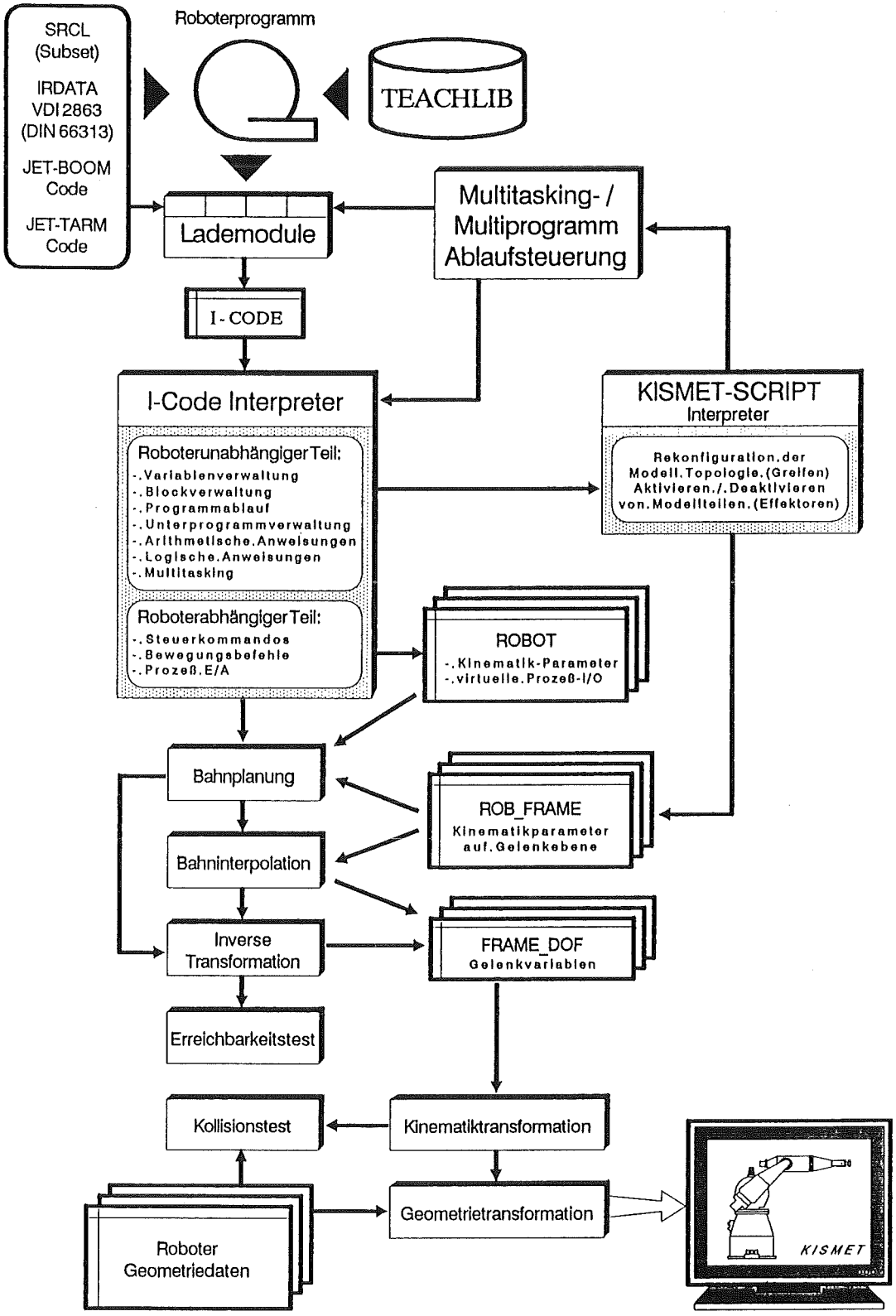


Abbildung 33. Architektur der virtuellen Roboterprogramm-Ablaufsteuerung

ziele bei Programm-Verzweigungen, -Schleifen oder in Unterprogramm-Aufrufen und -Rücksprüngen verwendet.

Bei der Abarbeitung von IRDATA-Programmen ist zu unterscheiden zwischen **roboterunabhängigen** und **roboterabhängigen Befehlssätzen**. Roboterunabhängige Kommandos werden komplett innerhalb des I-Code-Interpreters abgearbeitet. Roboterabhängige Befehlssätze (*MOVE, ACCEL, FEDRAT, USEIR, AXREL* usw.) werden durch Setzen der entsprechenden Parameter innerhalb der *ROBOT-, ROB_FRAME-* und *FRAME_DOF*-Datensätze realisiert. Bei den Bewegungskommandos erfolgt zunächst die grobe Bahnplanung. Hierzu wird bei PTP-Fahrbefehlen in TCP-Koordinaten zunächst das **inverse kinematische Problem** gelöst[20] und unter Anwendung der in "Bahnplanung und Bewegungssteuerung" hergeleiteten Gleichungen die Parameter des Bewegungsprofils berechnet. Während der **Bewegungsinterpolation** (geometrische Feinplanung) wird für jeden Darstellungszyklus (Zeitschritt) die Liste der *TC_CNTL*-Sätze durchlaufen. Innerhalb dieses Datensatz ist ein Merker gesetzt, falls sich der zugehörige Roboter in einem Bewegungszyklus befindet. Die Ausführung des auf ein Bewegungskommando folgenden I-Code-Satzes wird erst nach Ablauf des laufenden Bewegungszyklus gestartet.
Für lineare oder zirkulare CP-Bewegungen ist die Lösung des inversen Problems in jedem Interpolationsschritt notwendig, bei PTP-Bewegungen werden Gelenkkoordinaten interpoliert. Innerhalb jedes Bewegungsintervalls wird eine Überprüfung auf **Erreichbarkeit** der Zielposition (das programmierte Ziel kann außerhalb des Arbeitsbereichs liegen) und die Einhaltung der aktuell zulässigen Betriebsgrenzen (Achs-Verfahrbereich in jedem Fall, Achs-Geschwindigkeiten und -Beschleunigungen bei CP-Bewegungen) durchgeführt. Durch den Bediener kann optional außerdem die Durchführung der **Kollisionsprüfung** aktiviert werden. Bei Ansprechen einer der genannten Überprüfungen werden entsprechende Fehlermeldungen (akkustisch und textuell) an den Operateur ausgegeben.

Synchronisation parallel ablaufender Programme
Verschiedene Roboter können sowohl über ein einzelnes IRDATA-Programm, als auch über jeweils ein IRDATA-Programm je Zielroboter gesteuert werden. Im erstgenannten Fall werden die Bewegungskommandos verschiedener Roboter dadurch synchronisiert, daß die Ausführung eines nachfolgenden Kommandos erst nach komplett erfolgter Abarbeitung der vorhergehenden Bewegungssequenz erfolgen kann. Im letzteren Fall erfolgt die Synchronisation nebenläufiger Aktivitäten von mehreren Robotern durch die Task-Synchronisationssätze für Semaphore (SEMINI, WAIT, SIGNAL).

Zur Synchronisation über **Semaphore** wird in KISMET das globale Integer-Datenfeld *semaph* verwendet. Während des Roboterprogramm-Ablaufs werden die I-Code Synchronisationskommandos wie folgt ausgeführt:

SEMINI Besetzt die Semaphore mit dem Index *j* auf den Wert *n* (in der Regel wird gesetzt: $n \leq 0$).

WAIT Dekrementiert die Semaphore *j* um den Wert 1 und stoppt die Programmabarbeitung an dieser Stelle. Die Befehlszeile wird nun wiederholt durchlaufen bis die Semaphore wieder den Wert 0 annimmt. Das Rücksetzen auf den Wert

---

[20] Die Lösung des inversen Problems erfolgt analytisch über eine Bibliothek von Unterprogrammen. Der Index der Roboter-Kinematikstruktur wird KISMET innerhalb der '.mpc'-Definitionsdatei als Zahl übergeben. Diese Zahl wird zur Verzweigung auf das zur Kinematik-Familie gehörende Unterprogramm benützt.

0 erfolgt durch ein oder mehrere nebenläufig abgearbeitete Programme über die I-Code Kommandos *SEMINI* oder *SIGNAL*.

SIGNAL Inkrementiert die Semaphore *j* um den Wert 1.

Eine weitere dritte Möglichkeit zur Synchronisation von parallel ablaufenden TF ist die Verwendung der simulierten *Analog-* und/oder *Digital-Schnittstellen* über eine Einbettung der Schreib-/Lesekommandos (ANA_IN, ANA_OUT, DIG_IN, DIG_OUT) in Programmablauf-Strukturen (IF-THEN-ELSE, WHILE, GOTO usw.). Zu diesem Zweck sind für jeden simulierten Roboter in dessen ROBOT-Datenstruktur jeweils 16 Digital- und Analogschnittstellen-Register (32-Bit Worte) vorgesehen, die über die TF-Ablaufsteuerung gelesen und beschrieben werden können. Diese virtuellen Schnittstellen können außerdem für die Simulation der **sensorgeführten** Roboterprogrammierung verwendet werden.

5.4.2.3 Das WORKFRAME-Pfadlisten-Konzept in KISMET

Ein Hauptproblem der Off-Line-Programmierung ist die Verknüpfung der im CAR-System verfügbaren **Geometrie-** und **Topologiedaten** der Roboter-Arbeitszelle mit der textuellen Programmierung. Während die im CAR-System vorhandene **strukturierte Geometrieinformation**[21] der Arbeitsumgebung, wie am Beispiel von KISMET gezeigt, nach Veränderungen in der Arbeitszelle relativ schnell mittels CAD-Editierfunktionen der realen Arbeitsumgebung angepaßt werden kann, müßten im Falle der rein textuellen Programmierung alle von der Änderung betroffenen Zielkoordinaten neu eingegeben bzw. mittels Texteditor geändert werden. Bei der textuellen Programmierung ist es daher üblich, in einem weiteren Arbeitsgang die Zielpositionen der Bewegungskommandos in der realen Arbeitszelle "On-Line" einzulernen und in das textuelle Programm zu übernehmen (On-Line-Nachprogrammierung, Teachen). Bei geänderten Arbeitspositionen ist dieser Schritt zu wiederholen.

Durch das hier beschriebene Konzept einer **hybriden grafisch/textuellen Off-Line-Programmierung** soll vor allem der Arbeitsschritt einer erneuten Erzeugung des Rumpfprogramms bei einer Änderungen der Zellgeometrie vereinfacht bzw. die Übernahme der Bewegungs-Zielpositionen in das ausführbare Roboterprogramm automatisiert werden. Das Konzept sieht vor, daß Bewegungsziele in der zur Roboter-Programmierung verwendeten Hochsprache (PASRO, ROBCAD-TDL, VAL-II, IRL) symbolisch adressiert werden. In der strukturierten, geometrischen Datenbasis von KISMET werden die Arbeitspositionen (WFRM) definiert und über Referenzen mit den Objekten der Arbeitszelle unter Wahrung der Baugruppen-Topologie verknüpft. Die WFRM werden zu einem **WFRM-PFAD** zusammengefaßt. Die Reihenfolge der WFRM innerhalb eines Pfads entspricht der Bearbeitungs-Reihenfolge im Roboterprogramm. Der WFRM-PFAD ist über eine Zeichenkette eindeutig identifizierbar. Alle WFRM-PFADe einer Handhabungsaufgabe werden zu einer **WFRM-PFAD-Sequenz** zusammengefaßt und als WFRM-PFAD-Datei gespeichert. Ein auf die verwendete Programmier-Hochsprache und die Zielsprache (IRDATA, ICR) zugeschnittenes **Programmiermodul** erzeugt aus WFRM-PFAD-Dateien, Roboter-Rumpfprogrammen und einer **Elementaroperation-Bibliothek** ein ausführbares Roboterprogramm.

In einer WFRM-PFAD-Datei wird die Position jedes WFRM **redundant gespeichert**:

1. bezüglich des *Roboter-Referenzsystems* (Nullpunktsystem). Diese Information (Roboter-TCPF, Position von Zusatzachsen, Winkelstatus) wird vom Programmier-

---

[21] Eine andere mögliche Bezeichnung wäre der Begriff *"geometrische Wissensbasis"*.

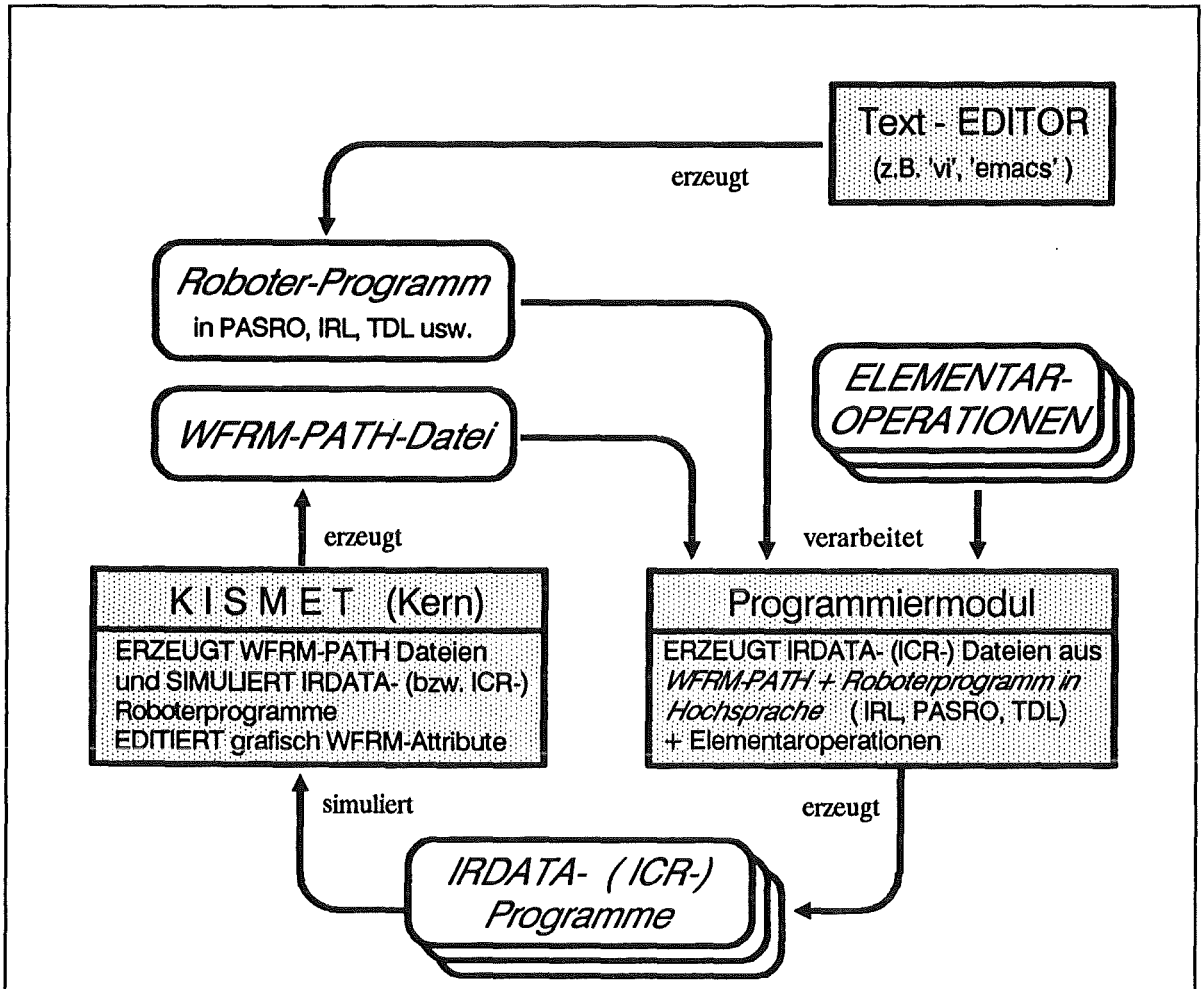


Abbildung 34. WFRM-Pfadlisten - Modul-Architektur und Datenfluß im Programmiersystem

system direkt in das IRDATA-Zielprogramm übernommen. Die Identifikation der WFRM-Daten in der WFRM-PFAD-Datei durch das Programmiermodul geschieht über die symbolische Bezeichnung des Pfads und/oder des einzelnen WFRM.

2. bezüglich des zu *bearbeitenden Zellobjekts* in der Geometrie-Datenbasis als relative Position des WFRM zum Objekt-KS. Der Objekt-Bezeichner (Objekt-ID) wird mit in der WFRM-PFAD-Datei abgelegt. Dieser wird aus den Bezeichnern der Baueinheiten entsprechend der Baugruppen-Hierarchie gebildet, z.B. :

JET_TORUS.OCTANT_5.UPPER_LIMITERS.LIMITER_2.COOLING_PIPE

Dieses Vorgehen ermöglicht insbesonders das nachträgliche Editieren von bereits existierenden Pfadlisten in KISMET nach Änderungen der realen Zellgeometrie.

Nach einer Positionsänderung einzelner WFRMs in der Geometrie-Datenbasis von KISMET werden in einem weiteren Bearbeitungsschritt alle Arbeitspositionen bezogen auf das Roboter-Referenzsystem berechnet (Kommando: **GENERATE ROBOT-Frames**).
Das Programmiermodul erzeugt mit einem bereits vorhanden, textuell erzeugten Hochsprachen-Programmrumpf und der korrigierten Pfadliste ein ausführbares IRDATA-Programm. Ein erneutes Editieren des Hochsprachen-Programms ist bei dem o.g. Verfahren nicht notwendig.

5.4.3 Testhilfen

In der Planungs- und Modellierphase ist es neben einer grafischen Ausgabe als 3D-Volumenmodell oft sinnvoll, Werkzeugbahnen, Gelenkparameter usw. als Funktion der Zeit oder des Orts in Diagrammform darzustellen.

Diese Funktionalität kann in KISMET genutzt werden, um verschiedene Parameter der Robotersteuerungen (Maximalgeschwindigkeiten, Beschleunigungen, Bahnfahrverhalten) zu ermitteln. Hierzu werden im Monitor-Modus von KISMET während der Abarbeitung von Testzyklen durch den realen Roboter die von der Robotersteuerung gesendeten Achspositions-Sensordaten "On-Line" erfaßt und gespeichert. Diese über die Sensorik erfaßten Bewegungszyklen können später als Zeitdiagramm oder als Ortskurven grafisch dargestellt werden.

Abbildung 35 zeigt einen simulierten Bewegungszyklus der aus einer PTP-Bewegung und einer Linearbewegung zusammengesetzt ist.

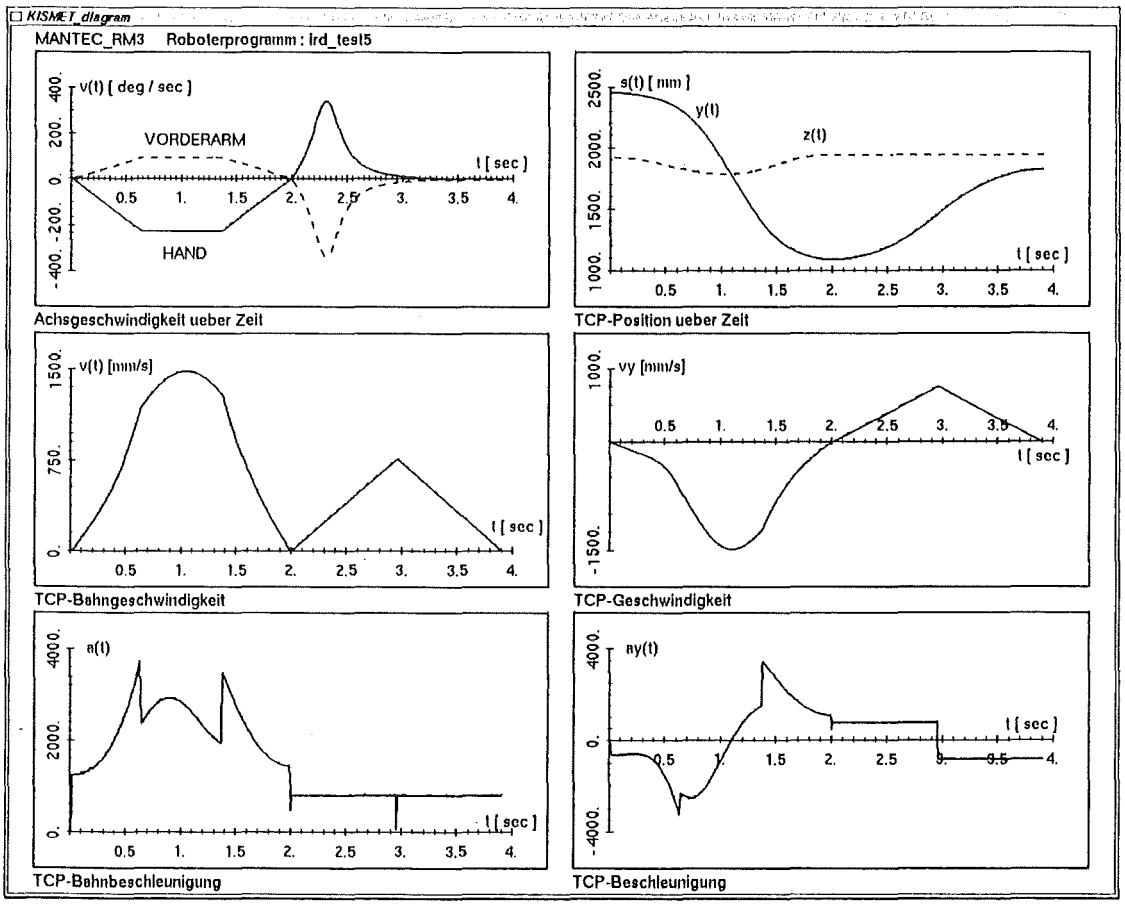


Abbildung 35. Darstellung von Bewegungszyklen in Diagrammform

6. Ausblick und Entwicklungspotential

6.1 *Allgemeine Planung und zukünftige Anwendungen*

KISMET soll im Leitstand des Großhandhabungstransporter EDITH als ein Teilsystem der "Remote Handling Workstation" [138] für Operationsversuche und zur Bedienerunterstützung eingesetzt werden. Das Experiment wird im HT-Labor des KFK für das NET/ITER-Projekt als prototypisches "In-Vessel"-Handhabungssystem aufgebaut. Die Schnittstelle zur Erfassung der Achs-Sensordaten von der Robotersteuerung ist bereits realisiert, die Inbetriebnahme des Gesamtsystems einschließlich der Mechanik und aller Antriebe ist für 1992 geplant. Weiterhin ist in diesem Rahmen die Kopplung von *KISMET* mit dem noch in der Entwicklung befindlichen wissensbasierten System zur prozeduralen Aktionsplanung und -unterstützung *PROSIM* vorgesehen [139].

In diesem Umfeld soll die Anwendung der **synthetischen Stereografik** untersucht werden. Die Stereo-Hardware im Grafiksystem wurde 1991 in Betrieb genommen und die notwendigen Softwareerweiterungen in *KISMET* sind durchgeführt. Es ist zu untersuchen wie sich die verminderte Gesamtbild-Berechnungsrate und die halbierte Bildauflösung (Hardware-bedingt) auf die Bedien- und Einsetzbarkeit des Systems auswirken.

6.2 *Erweiterung der grafischen und allgemeinen Fähigkeiten*

Erweitertes Subsystem zur numerischen Kollisionsrechnung
Bei den beschriebenen praktischen Anwendungen von *KISMET* im Bereich der Fernhantierungstechnik hat sich gezeigt, daß der bisher implementierte Algorithmus zur numerischen Kollisionserkennung trotz der hierarchisch gestaffelten Tests zu einer deutlichen Reduktion der Darstellungsrate (50-90% Rechenzeitanteil für die Kollisionstests) führen kann. Dieses Phänomen läßt sich speziell bei umfangreichen Modellen feststellen, wie z.B. der JET-Ex-Vessel-Simulation oder auch im Modell der WAK-Auflöserzelle. Der Grund für dieses Verhalten ist, daß der Kollisionstest unter Verwendung des bei der Modellinitialisierung automatisch generierten Kollisionstest-Modelldatenbaums oft auch solche Modellkomponenten mitüberprüft, für die in einer bestimmten Arbeitssituation gar keine Kollisionsgefahr besteht.
Wünschenswert wäre daher ein vom Benutzer interaktiv konfigurierbares Kollisionstest-Modell, das dem zur synthetischen Szenenpräsentation verwendeten hierarchischen Darstellungsmodell überlagert wird. Anhand von benutzerdefinierten Tabellen werden dabei in bestimmten Prozeßzuständen kollisionsgefährdete Modellkomponenten für die numerische Kollisionswarnung aktiviert, andere nicht kollisionsgefährdete Teile vom Operateur deaktiviert.

Simulation elastomechanischer und dynamischer Effekte
Alle für die "In-Vessel"-Handhabung bislang realisierten Vielgelenkarme (JET Articulated Boom, TFTR-Boom, KFK-EDITH) weisen eine signifikante vertikale Durchbiegung auf. Sie wird, trotz einer mittels Kastenbauweise konstruktiv erzielten hohen Biege- und Torsionssteifigkeit, durch die Summe der Durchbiegungen und Verdrillungen in den einzelnen Armgliedern verursacht. Dieser Effekt tritt vor allem bei extremer Reichweite in Kombination mit hohen Nutzlasten auf und kann, auf den TCP bezogen, mehrere Zentimeter betragen. Daneben treten, verursacht durch dynamische Effekte und Kopplungen zwischen den Armgliedern, Schwingungen und im Automatikbetrieb auch Abweichungen von der Sollbahn auf.

Obwohl diese Durchbiegung bei der manuellen Bedienung visuell durch den Operateur oder externe Sensorik erfaßt werden kann, und sowohl in der grafischen Darstellung als auch in der realen Mechanik über Kompensationsgelenke ausgeglichen werden kann, erschwert die Durchbiegung doch eine, über reine Transportvorgänge hinausgehende, Off-Line-Programmierung und Automatisierung von Handhabungsabläufen. Durch die Einbeziehung von Steifigkeitsmatrizen, diese können durch ein FEM-System extern berechnet und in die Datenbasis von *KISMET* übertragen werden, kann dieses Problem für die Anwendung in der Off-Line-Programmierung, Simulation und beim synthetische Sehen, im ersten Ansatz für den statischen Fall, in die Berechnung und Darstellung mit einbezogen werden. Es wurden entsprechende Arbeiten zur On-Line-Berechnung statischer Deformationen in Angriff genommen und bereits erste Erfolge erzielt.

Verfahren zur allgemeinen Lösung der Rücktransformation
In *KISMET* wird z.Zt. für jeden Kinematiktyp eine analytische Lösung des inversen Kinematikproblems benutzt. Dieses Vorgehen liefert zweifellos optimale Lösungen bezüglich der Rechenzeit und gestattet auch die Einbeziehung charakteristischer Steuerungseigenschaften (Verhalten in der Nähe von Singularitäten, "Aussteigen" der Robotersteuerung bei bestimmten Zielpositionen usw.), erfordert jedoch bei Einführung neuer Roboterkinematiken immer wieder einen hohen Implementierungsaufwand.
Die Literatur beschreibt verschiedene allgemeine Verfahren zur automatischen Rücktransformation, die bei vertretbarem Rechenzeitaufwand auch für die Anwendung in *KISMET* in Betracht kommen. Als mögliche Ansatzpunkte bieten sich die Methode des *charakteristischen Gelenkpaares* mit Verwendung von Bindungsfunktionen [140], oder das *modifizierte Newton-Raphson-Verfahren* mit *modifizierten Prädiktor-korrigierten* Algorithmen [141] an.

Methoden zur Vereinfachung der Modellerzeugung und -pflege
Ein weiteres, bisher nicht zufriedenstellend gelöstes Teilproblem ist die *Ad-hoc-Neumodellierung* oder auch Ergänzung der *geometrischen Simulations-Datenbasis*, sowie deren Pflege zur Wahrung der Konsistenz zwischen Anlagenmodell und realer Arbeitsumgebung. Inzwischen steht für die Grafikstation eine *Framegrabber*-Karte zur Verfügung, die das Digitalisieren von Videobildern (Schwarzweiß oder Farbe, 25 Bilder/s) gestattet. *KISMET* wurde um ein Modul erweitert, das die Ansteuerung der Karte, sowie die Echtzeit-Bildüberlagerung des 3D-Anlagenmodells (dargestellt als Drahtmodell; die Abbildung entspricht dem der Videokamera, wobei z.Zt. die Anpassung der Bildparameter dem Operateur obliegt) mit dem gerasterten Bitmap-Bild gestattet. Zur Vereinfachung des Modellierens sollen in *KISMET* die vorhandenen Grundfunktionen des Geometrie-Editors um weitere Operationen ergänzt werden. Ein ähnliches Modelliersystem, in dem gleichzeitig verschiedene Videoansichten benützt werden, ist in den Aufsätzen [142] und [143] beschrieben. Der Teilaspekt der Kameraparameter-Identifikation (Projektion) ist z.B. in [144] behandelt. Über eine Arbeit zur Ermittlung von Kamera-Standpunkt und -Orientierung wird in [145] berichtet, wobei allerdings 4 Punkte der Arbeitszelle *a priori* bekannt sein sollten.

Zweifellos kann mit der Methode der Bildüberlagerung keine hohe Genauigkeit erzielt werden, sie scheint allerdings zur schnellen Ad-hoc-Erfassung eines groben Umgebungsmodells geeignet. Dies kann z.B. im Rahmen von unvorhersehbaren Inspektions- und Reparaturarbeiten notwendig werden, wenn für eine genauere und detaillierte Modellierung nicht genügend Zeit zur Verfügung steht. Das grobe Modell kann dann für die Kollisionsvermeidung und zur interaktiven Planung von Grob-Bewegungen benützt werden. Für die Feinvermessung [146], [147] zur Korrektur oder Verfeinerung des groben Anlagenmodells, kann später z.B. das zum Einsatz in Fusionsanlagen und ebenfalls am IRE entwickelte Fernvermessungssystem (GMS) [148] eingesetzt werden.

7. Schlußfolgerung und Zusammenfassung

Die Integration von realitätsgetreuer, sensorgestützter Echtzeit-3D-Computergrafik in das Leitstandkonzept hochflexibler handhabungstechnischer Applikationen eröffnet neue Möglichkeiten der Beherrschbarkeit und Wartbarkeit komplexer technischer Anlagen. Das Anwendungsspektrum der neuentwickelten Methoden liegt insbesonders bei fernbedienten Inspektions-, Wartungs-, Reparatur- und Abrißarbeiten, die in den für den Menschen unzugänglichen und mit den konventionellen Methoden der Videotechnik schwer einsehbaren Arbeitsumgebungen von kerntechnischen Anlagen (Spalt- und Fusionsreaktoren, Wiederaufarbeitungsanlagen, Brennelemente-Fertigungs- und Endlagerstätten), sowie bei Arbeiten unter Wasser und im Weltraum durchgeführt werden. Die Untersuchung vorhandener CAD/CAM- und Roboter-Programmiersysteme hat gezeigt, daß diese Programme sich zur Echtzeit-Unterstützung von Fernhantierungsaufgaben und zur Überwachung des real ablaufenden Arbeitsvorgangs nicht eignen. Sie wurden in der Vergangenheit ausschließlich für Planungs- und Programmieraufgaben im Rahmen der Arbeitsvorbereitung entwickelt und erfüllen daher nur teilweise die gestellten Anforderungen hinsichtlich Zeitverhalten, Datenstrukturen, Benutzerschnittstellen und Funktionalität zur Bedienerunterstützung.

Im Rahmen der vorliegenden Arbeit wurde mit dem grafischen Simulationssystem *GBSim* erstmals das Verfahren des **CAD-modellbasierten, synthetischen 3D-Sehens** zur manuellen Steuerung, Off-Line-Programmierung, Simulation und Überwachung von Fernhantierungseinrichtungen eingesetzt. Die Pilotanwendung wurde am Beispiel von Transportabläufen des redundanten Vielgelenkarms "Articulated Boom" im unzugänglichen "In-Vessel"-Arbeitsbereich des Versuchs-Fusionsreaktors JET (Culham/UK) durchgeführt. Für die Anwendung in *GBSim* wurden neuartige Techniken und Algorithmen entwickelt :

- Für die schattierte Echtzeit-Darstellung mit Ausblendung der verdeckten Flächen und Kanten ("Hidden-Surface-Removal") unter besonderer Berücksichtigung von bewegten Kinematiken; - Die Methode eignet sich besonders zur Anwendung auf Grafiksystemen ohne Tiefenspeicher-Hardware (Z-Buffer).
- Zur numerischen On-Line-Kollisionserkennung und -vermeidung.
- Zur kinematischen Bewegungssteuerung eines redundanten Manipulators durch Reduktion der kinematischen Mehrdeutigkeit auf geometrisch eindeutig lösbare Konfigurationen; - Mehrere "Resolved-Motion"-Modi zur Steuerung in TCP-Koordinaten, sowie "Rekonfigurationsmodi" zur Repositionierung der hinteren Armglieder ohne Änderung von Position und Orientierung des End-Effektors wurden realisiert. Die hier demonstrierten Steuerungsmodi wurden inzwischen bei der Entwicklung einer Steuerung für den redundanten Vielgelenkarm EDITH berücksichtigt.

Weiterhin wurde im Rahmen dieser Arbeit, und basierend auf den mit *GBSim* gewonnenen Erfahrungen, das integrierte und universell einsetzbare, d.h. nicht auf eine spezielle Applikation beschränkte, 3D-grafische Softwarewerkzeug *KISMET* von Grund auf neu konzipiert und implementiert. *KISMET* unterstützt die rationelle Planung, Simulation, Programmierung und On-Line-Prozeßüberwachung von Arbeitsabläufen in der Fernhantierungstechnik. Als **wesentliche Neuerungen** gegenüber den bisher bekannten Systemkonzepten weist *KISMET* folgende **charakteristischen Eigenschaften** auf:

- Die Verwendung eines hierarchischen Datenmodells das den interaktiven Wechsel zwischen verschiedenen aufgabengerechten Modell-Detaillierungsstufen während eines Simulationslaufs erlaubt.

- Das realisierte Programmkonzept erlaubt den schnellen Wechsel zwischen Ad-hoc-Planung und -Programmierung, sowie die Überwachung der sofortigen Programmausführung.

- Die Möglickeit zur interaktiven Definition, Analyse und Simulation von:

  - kinematischen Ketten,
  - beliebig verzweigten kinematischen Baumstrukturen,
  - sowie dem in der Praxis häufig auftretenden Fall von ebenen, geschlossenen 4-Gelenk-Schleifen oder deren Kaskadierungen. Zur Lösung dieses Problems wurde eine besonders effiziente, "exakte" geometrische Lösung erarbeitet.
  - Getrieben und funktionellen Kopplungen zwischen den Bewegungsachsen

- Das automatische Erzeugen eines internen, hierarchischen Datenmodells zur numerischen Kollisionserkennung. Der Algorithmus basiert auf einer Modellapproximation durch einhüllende Quader, die das Ausnützen der "Clipping"-Grafikhardware erlaubt. Beim Ansprechen dieser ersten, groben Stufe wird die "exakte" Oberflächengeometrie für weitere Tests verwendet. Das abgestufte System bietet hohe Effizienz bzgl. der Rechenleistung und gleichzeitig hohe Genauigkeit.

- Aus den implementierten Geometrie-Elementen (Polyeder, CSG-Primitive, "Sweeps", 3D-Kurven und parametrisierte Flächen nach VDA-FS) wird während der Dateninitialisierung ein internes B-REP-Modell erzeugt, das durch die Einbeziehung von Oberflächen-Normalen eine realitätsnahe, GOURAUD-schattierte Echtzeit-Darstellung des Szenarios erlaubt.

- Das Darstellungs-Attribut kann interaktiv für einzelne Modellteile oder auch Baugruppen geändert werden (GOURAUD-, "Flat"-, oder transparent-schattiert, Hidden-Line und als Drahtmodell). Der Wechsel erfolgt augenblicklich, da keine "Display-Listen" aufgebaut werden müssen. Der Bediener kann sich dadurch in jeder Arbeitsphase optimale Sichtbedingungen schaffen.

- Zur Wahrung der Konsistenz zwischen realem Anlagenmodell und Darstellungsmodell sind zahlreiche Funktionen zum interaktiven Editieren der Modellgeometrie und Baugruppen-Topologie in das System integriert. Hierzu gehört u.a. das interaktive Ausführen von CSG-Operationen basierend auf dem internen Polyeder-Geometriemodell. Die Integration eines Modell-Editors erlaubt eine schnelle Anpassung und Erweiterung des Simulationsmodells in Ausnahmesituationen.

- Spezielle Dateien gestatten die Definition von applikationsspezifischen Beleuchtungsmodellen und Materialtabellen.

- Die parallele Simulation mehrerer Roboter bzw. Mechanismen, auch im gemischten On- und Off-Line-Betrieb ist möglich. Es können daher Teile der Handhabungseinrichtungen real bewegt und mit *KISMET* überwacht werden (z.B. im "Mock-Up"), während andere bewegte Komponenten der HFH-Zelle lediglich simuliert werden.

- Spezielle Funktionalität für die Fernhantierung wurde durch die Möglichkeit zur Off-Line-Simulation von Kameraansichten und zur On-Line-Unterstützung der Positionierung (Zeigefunktion für schnelle Zielfindung und automatische Zielverfolgung) von Schwenk-/Neigekopfsystemen und von mobilen Kamera-Armen in das System integriert.

- Unter Inkaufnahme eines höheren Entwicklungsaufwands wurde *KISMET* mit einer dynamischen Speicherverwaltung und -optimierung ausgestattet. Durch die dynamische Speicherzuweisung gewinnt der Anwender den Vorteil einer logisch

nicht limitierten Modellgröße, gleichzeitig wird durch die optimierte Verwaltung eine Übersegmentierung des Speichers, die bei häufigem Wechsel von Details und funktionellen Gruppen auftreten würde, vermieden.

- Durch die Integration eines Moduls zur fotorealistischen Darstellung unter Anwendung von Ray-Tracing-Techniken wird zusätzlich die Voruntersuchung der zu erwartenden Beleuchtungsverhältnisse in der Fernhantierungszelle ermöglicht.

Die **universelle Anwendbarkeit** von *KISMET* wurde bewiesen durch Applikationen als grafisches Monitor-System für die Ex-Vessel-Fernhantierung mit dem Großmanipulator TARM bei JET und des sensorgeführten Robotersystems CATROB im HT-Labor des KFK. Ein weiterer Erfolg von *KISMET* ist sein Einsatz im Rahmen der D2-Weltraummission als grafisches Kernsystem im Kontrollstand des Roboterexperiments ROTEX bei der DLR in Oberpfaffenhofen. Darüberhinaus wurde *KISMET* erfolgreich bei verschiedenen Off-Line-Planungsstudien im Vorfeld von Fernhantierungsaufgaben eingesetzt. Als typische Beispiele sind hier zu nennen:

- Die Simulation des Rohrleitungsausbaus in einer heißen Zelle (Auflöserzelle) der Wiederaufarbeitungsanlage Karlsruhe (WAK) [31]. Wichtige Ergebnis dieser Arbeit führten zur Redimensionierung der verwendeten Spezialwerkzeuge. Der außerdem durch die Betreiberfirma *DWK* der WAK in Petershagen-Lahde für reale Versuche gebaute Mock-Up konnte mit einem Teil der tatsächlichen Einbauten (in der realen Zelle) auskommen, wodurch erhebliche Kosten gespart wurden.

- Die Simulation des Ausbaus von Divertorplatten im In-Vessel-Bereich des zukünftigen NET/ITER-Fusionsreaktors.

- Eine Simulation des KFK-Experimentierstands EDITH. Der Monitor-Modus von *KISMET* erlaubt den Funktionstest der Boom-Steuerung auch schon vor der Verfügbarkeit der Mechanik.

Die durch die Implementierung der standardisierten IRDATA-Programmierschnittstelle gewählte universelle Auslegung des Systems ermöglicht dessen Einsatz auch in einem produktübergreifenden Fertigungsverbund. Das gewählte Konzept einer einheitlichen Programmierschnittstelle zwischen Off-Line-Programmierung, CAR-Simulationssystem und Robotersteuerung reduziert die oft notwendigen Anpassungskosten für den potentiellen Anwender.

Während der Pilotanwendung von *KISMET* wurde als besonderer **Problembereich** für eine breitere Anwendung des synthetischen Sehens das *Modellieren von größeren Anlagekomplexen* **identifiziert**, sowie die Verbindung mit CAD-Systemen zum Zweck der Modell-Generierung. Der Einsatz des für *KISMET* entwickelten *CAD*I*-Postprozessors mit *CATIA*-Modellen hat gezeigt, daß eine einfache Umsetzung von existierenden CAD-Modellen zu Zwecken der grafischen Echtzeitsimulation mit Schwierigkeiten behaftet ist. Besonders problematisch erscheinen mir dabei folgende Punkte:

- Ausgearbeitete Normen für neutrale CAD-Schnittstellen (*IGES, SET, VDA-FS*) enthalten z.Zt. noch nicht die für den Austausch von Roboter-Modellen und Mechanismen notwendigen kinematischen und technologischen Definitionselemente.

- Die zum Aufbau von hierarchisch gegliederten und mehrere Detaillierungsstufen umfassenden Modellen notwendigen topologischen Informationen lassen sich mit den verfügbaren neutralen Schnittstellen nicht übertragen.

- Existierende und in der Regel im Rahmen des Anlagenentwurfs generierte CAD-Modelle sind zur direkten Verwendung in Echtzeit-Simulationssystemen zu detailliert. Es sind daher Konzepte zur (automatischen) Datenreduktion zu entwickeln.

Aus den vorgenannten Gründen ist die Nutzung von kommerziellen CAD-Systemen in Kombination mit Neutralformat-Prozessoren für die Erzeugung von Simulations-Modellen noch mit erheblichem Arbeitsaufwand und Kosten verbunden. Der breite Einsatz der neuentwickelten grafischen Simulationstechniken wird dadurch erschwert. Die laufenden Standardisierungsarbeiten zum Austausch von Produktmodellen über *STEP* versprechen eine Verbesserung auch im Bereich der Modellierung für das synthetische Sehen.

Mit dem im Rahmen dieser Arbeit entwickelten Verfahren zur Manipulation in unzugänglichen Arbeitsräumen und dem realisierten Softwarewerkzeug *KISMET* konnte eine **wesentliche Basis** für die weiterführende Entwicklung und den Einsatz von CAT-Systemen, sowie von kombinierten Fernhantierungs-Arbeitsstationen geschaffen werden.

8. Literatur

[1] Köhler, W.: *"Stand der Fernbedienungstechnik."* Proc. 8th Int. Symp. on Industrial Robots, Stuttgart (1978)

[2] Silverman, E.B.: *"Robotic Technology Experiments for Nuclear Power Plant Inspection and Maintenance."* Proc. 30th Conf. on Remote Systems Technology, Washington, S.109-112 (1982)

[3] Trachsel, C.A., Rolston, D.R., Wells, M.T., Brown, L.B., Herman, H.: *"Fusion Reactor Remote Maintenance and Repair Operations Simulation using Computer Graphics."* Proc. 30th Conf. on Remote Systems Technology, Los Angeles, S.103-106 (1982)

[4] Titus, P.H.: *"Dynamic and Geometric Simulation of the TFTR In-Vessel Manipulator System."* Proc. 30th Conf. Remote Systems Technology, Los Angeles, S.81-88 (1982)

[5] Raimondi, T.: *"Remote Operations in JET : Problems and Solutions."* Proc. of 1rst European Symposium on Remote Operations on Fusion Devices, Mailand (1982)

[6] Jones, P., Maisonnier, D., Raimondi, T.: *"Design and Operation of the JET Articulated Boom."* Proc. of 11th SOFT, Austin (1985)

[7] Galbiati, L., Raimondi, T.: *"Control and Operation of JET Articulated Boom."* Proc. of 11th SOFT, Austin (1985)

[8] Raimondi, T., Galbiati, L., Jones, L.P.D.F.: *"Use of Teleoperators and Transporters in JET."* in: Robotics and Remote Maintenance Concepts for Fusion Machines, IAEA-TECDOC-495, Wien, S.115-127 (1989).

[9] Maisonnier, D.: *"Conception et modelisation de gros porteurs de systemes de teleoperation."* Dissertation Ecole Nationale Superieure de Mecanique, Nantes (1987)

[10] Leinemann, K.: *"A Graphics Based Remote Handling Control System."* KfK-Bericht 3788, Karlsruhe (1984)

[11] Leinemann, K.: *"Man-machine cooperation in remote handling for fusion plants."* Fusion Technology 1984, Proc. of 13th SOFT 1984, Varese, 24.-28.9.1984, Vol.2, S.1311-1316, Pergamon Press.

[12] Leinemann, K., Schlechtendahl, E.G.: *"Computer Graphics Support for Remote Handling Simulation and Operation."* Proc. 32nd ANS Conference on Remote Systems Technology, New Orleans, S.10-16 (1984)

[13] Suppan, A. (ed.): *"The NET Articulated Boom: Preliminary Investigations and Justification for a Full Scale Prototype."* KfK-Bericht 4809, Karlsruhe (1990)

[14] Köhler, W.: *"Typenbuch der Manipulatoren."* Verlag Karl Thiemig, München (1981)

[15] Benner, J., Blume, C.: *"Steuern leichtgemacht : Neue Anwendungsgebiete der Robotik für die flexible Handhabung."* Technische Rundschau, 42/89, S.78-83 (1989)

[16] Kühnapfel, U., Leinemann, K., Schlechtendahl, E.G.: *"Graphics Support for JET Boom Control."* Proc. International Topical Meeting on Remote Systems and Robotics in Hostile Environments. Pasco, Wa., March 29 April 4, 1987, S.28-34

[17] Jones, L.P.D.F., Galbiati, L., Gredel, M., Neddermeyer, W.: *"The Design and Construction of the TARM - A Crane-Mounted Remotely-Controlled Transporter for JET."* Proc. 16th SOFT, London, 3.-7. Sept. 1990, North-Holland, Amsterdam, Vol. 2, S.1378-1382 (1991)

[18] Wloka,D.: *"Graphische Simulation von Handhabungseinrichtungen."* Dissertation Universität Saarbrücken (1987)

[19] Hörmann, K.: *"Ein Verfahren zur Planung kollisionsfreier Bahnen für Industrieroboter."* Dissertation Universität Karlsruhe (1987)

[20] Schütze, P.: *"Kollisionskontrolle bei der Offline-Programmierung von Industrierobotern."* Dissertation RWTH Aachen (1988)

[21] Diehl, R.: *"Approximationsmodelle und ihr Einsatz bei der Kollisionsprüfung bewegter Volumenmodelle."* Dissertation Universität Karlsruhe (1989), Fortschrittsberichte VDI, Reihe10, Nr. 108, VDI-Verlag, Düsseldorf (1989)

[22] Zühlke, D.: *"Offline-Programmierung numerisch gesteuerter Industrieroboter."* Dissertation RWTH Aachen (1983)

[23] Niehaus, T.: *"Rechnergestützte Anwendungsprogrammentwicklung für Industrieroboter und flexible Automatisierungsgeräte."* Dissertation RWTH Aachen (1987)

[24] Kandziora, B.: *"CAD/CAM-System zur Planung und Simulation automatisierter Montagevorgänge."* Dissertation Universität Karlsruhe (1988), Fortschrittsberichte VDI, Reihe 20, Nr. 9, VDI-Verlag, Düsseldorf (1988)

[25] Wanner, M.C.: *"Rechnergestützte Verfahren zur Auslegung der Mechanik von Industrierobotern."* Springer, Berlin (1989)

[26] Benner, J., Leinemann, K.: *"Architecture of a telemanipulation system with combined sensory and operator control."* (Proc. RoViSeC-7, Zürich, 1988), IFS, Kempston, Bedford S.259-271 (1988).

[27] Leinemann, K., Kühnapfel, U., Ludwig, A.: *"CAD-Model Based Remote Handling Control System for NET and JET."* Proc. 15th SOFT, Utrecht, 19.-23.9.1988

[28] Kühnapfel, U.: *"KISMET - 3D-Grafik zur Planung, Programmierung und Überwachung von Telerobotics-Applikationen."* in VDI-Berichte Nr. 861.3, S.71-86, VDI-Verlag, Düsseldorf (1990)

[29] Leinemann, K., Kühnapfel, U., Schlechtendahl, E.G.: *"NET Remote Handling Control System with CAD-Support."* Proc. ANS 3rd Topical Meeting on Robotics and Remote Systems, Charleston, SC, USA, 13.-16.3.1989, S. 5.2.1-5.2.8

[30] Kühnapfel, U., and Ludwig, A.: *"Graphics and CAD Support for NET/ITER Boom Control."* Proc. 16th SOFT, London, 3.-7. Sept. 1990, North-Holland, Amsterdam, Vol. 2, S.1347-1352 (1991)

[31] Leister, P., Kühnapfel, U., Ludwig, A.: *"Computer Aided Simulation of a Remote Steam Jet Exchange in a Dissolver Cell."* Proc. ANS 4th Topical Meeting on Robotics and Remote Systems, Albuquerque, NM, USA, S.353-364 (1991)

[32] Benner, J., Fischer, C., Leinemann, K., Stratmanns, E., Till, W.: *"Towards more Automation for Remote Maintenance."* in: Robotics and Remote Maintenance Concepts for Fusion Machines, IAEA-TECDOC-495, Wien, S.249-260 (1989).

[33] Stark, L.: *"Telerobotics: Display, Control and Communication Problems."* IEEE Journal of Robotics and Automation, RA-3(1) (1987)

[34] Ferrel, W.R.: *"Delayed Force Feedback."* IEEE Trans. Human Factors in Electronics, Okt. 1966, S.449-455 (1966)

[35] Bejczy, A.K., Kim, W.S.: *"Predictive Displays and shared Compliance Control for Time-Delayed Telemanipulation."* In IEEE Int. Workshop on Intelligent Robots and Systems, Ibaraki, Japan, Juli 1990

[36] N.N.: *"Hohe Schule für Roboter."* Produktinformation Fa. MCDONNEL DOUGLAS INFORMATION SYSTEMS (1988)

[37] N.N.: *"CATIA Kinematics User Manual."* Produktinformation SH20-6633-01 der Fa. IBM (1986)

[38] N.N.: *"CATIA Robotics User Manual."* Produktinformation SH20-6634-2 der Fa. IBM (1987)

[39] N.N.: *"CIMSTATION."* Produktinformation der Fa. PRIME/COMPUTERVISION (1988)

[40] Milberg, J., Wrba, P.: *"Roboter-Einsatzplanung und Offline-Programmierung mit USIS."* ZwF 81, Nr. 9, S.484-488 (1986)

[41] Tauber, A., Schuster, G.: *"Robotersimulation - eine CIM-Komponente."* CAE Journal 4, S.30-39 (1988)

[42] Tauber, A.: *"Ein Verfahren zur allgemeinen Rücktransformation unter Berücksichtigung von Randbedingungen."* Robotersysteme 5, S.133-140 (1989)

[43] Tauber, A., Schuster, G.: *"Programmierung von roboterbestückten Produktionsanlagen."* Robotersysteme 5, S.47-56 (1989)

[44] Hegibotham, W.B., Dooner, M., Kennedy, D.N.: *"Computer Graphics Simulation of Industrial Robot Interactions."* 9th ISIR, Washington (1979)

[45] Hegibotham, W.B., Dooner, M., Case, K.: *"Robot Application Simulation."* The Industrial Robot no.2, IFS, Bedford (1979)

[46] Bonney, M.C., Dooner, M., Green, J.L.: *"Offline Programming using the GRASP Robot Simulation System."* IFIP Working Conference on Off-Line Programming of Industriel Robots, Stuttgart (1986)

[47] N.N.: *"GRASP - 3D graphical Simulation."* Firmeninformation der Fa. BYG Systems Ltd. (1989)

[48] Adler, A.: *"Robotics Workcell Design, Simulation and Off-Line Programming."* IEEE Conference on Systems, Man and Cybernetics, Atlanta (1986)

[49] Adler, A.: *"The Man-Machine Interface of a Robotics Workcell Design, Simulation and Off-Line Programming System."* Human Factors in Manufacturing, Stratford-Upon-Avon (1986)

[50] Adler, A.: *"Rechnerunterstützter Robotereinsatz."* Hüthig, Heidelberg (1988)

[51] Adler, A.: *"TDL, a Task Description Language for Programming Automated Robotic Workcells."* FMS-5, Stratford-Upon-Avon (1986)

[52] Leu, M.C., Mahajan, R.: *"Computer Graphic Simulation of Robot Kinematics and Dynamics."* Robots 8, Detroit (1984)

[53] Leu, M.C.: *"Robotics Software Systems."*
Robotics & Computer-Integrated Manufacturing, Vol. 2 No. 1, Frankfurt (1985)

[54] N.N.: *"IGRIP Offline Robot Programming and Simulation System."*
Firmeninformation der Fa. Deneb Robotics, Darmstadt (1989)

[55] N.N.: *"ROBOT-SIM : A CAD Based Workcell Design, Simulation, Analysis, Off Line Programming System."*
Firmeninformation der Fa. CALMA Deutschland, Frankfurt (1986)

[56] Weck, M., Niehaus, T., Osterwinter, M.: *"Graphisch interaktives Programmier- und Testsystem für Industrieroboter."* Robotersysteme 2, S.193-201 (1987)

[57] Dillmann, R., Huck, M.: *"A Software System for the Simulation of Robot Based Manufacturing Processes."* Robotics Vol. 2, No. 1 (1986)

[58] Grabowski, H. et.al.: *"DICAD-Produktmodell - Revision 1.0."*
Universität Karlsruhe (1987)

[59] Hearn, D., Baker, M.P.: *"Computer Graphics."* Prentice Hall International (1986)

[60] Foley, J.D., van Dam, A.: *"Fundamentals of Interactive Computer Graphics."*
Addison-Wesley, Reading, Mass. (1982)

[61] Newman, W.M., Sproull, R.F.: *"Principles of Interactive Computer Graphics."*
McGraw-Hill, New York (1979)

[62] Encarnacao, J., Schlechtendahl, E.G.: *"Computer Aided Design. Fundamentals and System."* Springer, Berlin (1983)

[63] Grätz, J.F.: *"Handbuch der 3D-CAD-Technik: Modellierung mit 3D-Volumensystemen."* Siemens AG, Berlin, München (1989)

[64] Requicha, A.A.G., Voelcker, H.B.: *"An Introduction to Geometric Modeling and its Applications in Mechanical Design and Production."* in Advances in Information Systems Science, Vol. 8, Plenum Publishing Corp., S.293-328 (1981)

[65] Requicha, A.A.G., Voelcker, H.B.: *"Solid Modelling: Current Status and Research Directions."* IEEE Computer Graphics & Applications 3, Okt. 1983, S.25-37 (1983)

[66] Miller, J.R.: *"Analysis of Quadric-Surface-Based Solid Models."*
IEEE Computer Graphics & Applications 8, Jan. 1988, S.28-42 (1988)

[67] Barsky, B.A.: *"A Description and Evaluation of Various 3-D Models."*
IEEE Computer Graphics & Applications 4, Jan. 1984, S.38-52 (1984)

[68] Samet, H., Webber R.E.: *"Hierarchical Data Structures and Algorithms for Computer Graphics."* in IEEE Computer Graphics & Applications 8, Mai 1988, S.48-68 (1988)

[69] Jackins, C.L., Tanimoto, S.L.: *"Oct-Trees and their Use in Representing Three-Dimensional Objects."* in Computer Graphics and Image Processing, 14(3), S.249-270 (1980)

[70] Meagher, D.: *"Geometric Modelling Using Octree Encoding."* in Computer Graphics and Image Processing, 19(2), S.129-147 (1982)

[71] Ayala, D., Burnet, P., Juan, R., Navazo, I.: *"Object Representation by Means of Nonminimal Division Quadtrees and Octrees."* in ACM Trans. on Graphics, 4, 1, S.41-59 (1985)

[72] Burnet, P., Ayala, D.: *"Extended Octree Representation of Free Form Surfaces."* in Computer Aided Geometric Design, 4, S.141-154 (1987)

[73] Ayala, D.: *"Boolean Operations between Solids and Surfaces by Octrees."* in Computer Aided Design, 20, 8, S.452-465 (1988)

[74] Gouraud, H.: *"Continuous Shading of Curved Surfaces."* IEEE Trans. Computers, 20(6), 623-629 (1971)

[75] Schuster, R.: *"System und Sprache zur Behandlung graphischer Information im rechnergestützten Entwurf."* Dissertation Universität Karlsruhe, KfK-Bericht 2305, Karlsruhe (1976)

[76] N.N.: *"Graphics Library Programming Guide."* Firmeninformation (Manual) der Fa. Silicon Graphics

[77] Akeley, K.: *"The Silicon Graphics 4D/240GTX Superworkstation."* IEEE Computer Graphics & Applications, 9(4), S.71-83 (1989)

[78] Chen, X., Ying, D.: *"Polygon Triangulation Algorithm as a Powerful Core Processor of PLAN-I."* Computer Graphics Forum, 8(3), S.193-198 (1989)

[79] Schlechtendahl, E.G. (ed.): *"Esprit Project 322: Specification of a CAD*I Neutral File for CAD-Geometrie. Version 3.3."* Springer, Heidelberg (1988)

[80] Schlechtendahl, E.G. (ed.): *"Esprit Project 322: CAD*I. CAD Data Transfer for Solid Models."* Springer, Heidelberg (1989)

[81] Brändli, N., Mittelstädt, M.: *"Exchange of Solid Models: Current State and Future Trends."* Computer-Aided Design, Vol.21, Nr.2, S.87-96 (1989)

[82] Mittelstädt, M.: *"The CATIA-KISMET Link at JET - Concept, Realization, and Operation."* Primärbericht (unveröffentlicht), Kernforschungszentrum Karlsruhe, Karlsruhe (1989).

[83] Leinemann, K., Kühnapfel, U., Ludwig, A.: *"Remote Handling Control with Graphical Man-Machine Interface for NET and JET."* in: Robotics and Remote Maintenance Concepts for Fusion Machines, IAEA-TECDOC-495, Wien, S.215-226 (1989).

[84] Braun, K.: *"Ein Vergleich aktueller Hochleistungs-Graphiksysteme und zukünftige Trends."* CAD/CAM 3/90, S.106-118 (1990)

[85] Levin, E.: *"The Role of Graphics Super-Workstations in a Supercomputing Environment."* in Kowalik, J.S. (Ed.), "Supercomputing", Nato ASI Series (F), Vol. 62, Springer, Berlin (1990)

[86] Clark, J.H.: *"The Geometry Engine: A VLSI Geometry System for Graphics."* Proc. SIGGRAPH '82, ACM Computer Graphics, 16(3), S.127-133 (1982)

[87] Lozano-Perez, T., Wesley, M.A.: *"An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles."*

Comm. Ass. Comput. Mach., Vol ACM 22, 560-570 (1979)

[88] Lozano-Perez, T.: *"Spatial Planning: A Configuration Space Approach."* IEEE Trans. on Computers, Vol C-32, No. 2, S.108-120 (1983)

[89] Smidt, D.: *"Automatic Path-Planning for a Multilink Articulated Boom within the Torus of a Fusion Reactor."* KfK-4120, Karlsruhe (1986)

[90] Sutherland, I.E., Sproull, R.F., Schumacker, R.A.: *"A Characterization of Ten Hidden-Surface Algorithms."* Computing Surveys, Vol. 6, No. 1 (1974)

[91] Fuchs, H., Abram, G.D., Grant, E.D.: *"Near Real-Time Shaded Display of Rigid Objects."* Proc. SIGGRAPH '83, ACM Computer Graphics, 17(3), S.65-72 (1983)

[92] Phong, B.T.: *"Illumination for Computer-Generated Images."* Comm. of the ACM, 18(6), S.311-317 (1975)

[93] Blinn, J.F.: *"Models of Light Reflection for Computer-Synthesized Pictures."* ACM Computer Graphics (Proc. SIGGRAPH '77), 11(2), S.192-198 (1977)

[94] Whitted, T.: *"An Improved Illumination Model for Shaded Display."* Comm. of the ACM, 23(6), S.343-349 (1980)

[95] Hall, R.A., Greenberg, D.P.: *"A Testbed for Realistic Image Synthesis."* IEEE Computer Graphics and Applications, 3(8), S.10-20 (1983)

[96] Cook, R.L., Torrance, K.E.: *"A Reflectance Model for Computer Graphics."* ACM Computer Graphics (Proc. SIGGRAPH '81), 15(3), S.307-316 (1981)

[97] Blinn, J.F.: *"Jim Blinn's Corner: Me and My (Fake) Shadows."* in IEEE Computer Graphics & Applications 8, Jan. 1988, S.82-86 (1988)

[98] Cook, R.L., Torrance, K.E.: *"Distributed Ray Tracing."* ACM Computer Graphics, 18(3), S.137-145 (1984)

[99] Appel, A.: *"Some Techniques for Shading Machine Renderings of Solids."* Proc. AFIPS, Vol 32, S.37-45 (1968)

[100] Goldstein, R.A., Nagel, R.: *"3-D Visual Simulation"*. in Simulation, 16(1), S.25-31 (1971)

[101] Roth, S.D.: *"Ray Casting for Modeling Solids"*. in Computer Graphics and Image Processing 18, S.109-144 (1982)

[102] Clark, J.H.: *"Hierarchical geometric models for visible surface algorithms."* Comm. of the ACM, 19(10), S.547-554 (1976)

[103] Glassner, A.S.: *"Space Subdivision for Fast Ray Tracing."* IEEE Computer Graphics & Applications 4, Okt. 1984, S.15-22 (1984)

[104] Weghurst, H., Hooper, G., Greenberg, D.P.: *"Improved Computational Methods for Ray Tracing."* ACM Trans. on Graphics, Vol. 3, 1, Jan. 1984, S.52-69 (1984)

[105] Kajiya, J.T.: *"New Techniques for Ray Tracing Procedurally Defined Objects."* ACM Trans. on Graphics, Vol. 2, 3, Juli 1983, S.161-181 (1983)

[106] Hanrahan, P.: *"Ray Tracing Algebraic Surfaces."* ACM Computer Graphics, 17(3), Juli 1983, S.83-90 (1983)

[107] Patentschrift: *"Verfahren zur Manipulation in unzugänglichen Arbeitsbereichen."* Patentschrift DE 3925275 C2, Deutsches Patentamt, Bundesdruckerei (1991)

[108] Dubbel: *"Taschenbuch für den Maschinenbau."* Springer, Berlin (1974)

[109] VDI-Richtlinie 2861, Blatt 2: *"Montage- und Handhabungstechnik: Kenngrößen für Handhabungseinrichtungen."* VDI-Verlag, Düsseldorf (1982)

[110] Sheth, P.N.: *"A Digital Computer Based Simulation Procedure for Multi-degree of Freedom Mechanical Systems with Geometric Constraints."* PhD Thesis, University of Wisconsin (1972)

[111] Paul, R.P.: *"Robot Manipulators - Mathematics, Programming and Control."* MIT-Press, Cambridge, Mass. (1981)

[112] Denavit, J.; Hartenberg, R.S.: *"A Kinematic Notation for Lower Pair Mechanisms Based on Matrices."* Journal of Applied Mechanics, vol.22, Trans.ASME Vol.77 (1955)

[113] Sheth, P.N., and Uicker, J.J.: *"A Generalized Symbolic Notation for Mechanisms."* Trans. of the ASME, Journal of Eng. for Industry (1971)

[114] Hartenberg, R.S.: *"Die Darstellung und Handhabung der niederen Elementepaare in einer auf Matrizenrechnung gegründeten Zeichensprache."* VDI-Berichte 12 (1955)

[115] Lee, C.S.G.: *"Robot Arm Kinematics, Dynamics and Control."* Computer, Dez. 1982, S.62-80 (1982)

[116] Bronstein, I.N., Semendjajew, K.A.: *"Taschenbuch der Mathematik."* Harri Deutsch, Thun, Frankfurt (1985)

[117] Desoyer, K., Kopacek, P., Troch, I.: *"Industrieroboter und Handhabungsgeräte - Aufbau, Dynamik, Steuerung, Regelung und Einsatz."* Oldenbourg, München, Wien (1985)

[118] Hirzinger, G.: *"Adaptiv sensorgeführte Roboter mit besonderer Berücksichtigung der Kraft-Momenten-Rückkopplung."* Robotersysteme 1, S.161-171 (1985)

[119] Hirzinger, G., Dietrich, J., Heindl, J.: *"Zum Stand der Robotikarbeiten der DLR."* Sonderdruck aus DLR-Nachrichten, Heft 58, November 1989

[120] Weck, M., Mehles, H., Weiß, F.: *"Prozeßnahe Roboterprogrammierung unter Einsatz eines inertialen Meßsystems."* Robotersysteme 4, Heft 4, S.245-250 (1988)

[121] Blume, C., Jakob, W.: *"Programming Languages for Industrial Robots."* Springer, Berlin (1986)

[122] Blume, C., Jakob, W., Favaro, J.: *"PASRO - Pascal and C for Robots."* Springer, Berlin (1987)

[123] Hörmann, K., Hugel, Th., Meier, W.: *"Ein Ansatz zur Realisierung intelligenter fehlertoleranter Robotersysteme."* Robotersysteme 4, Heft 4, S.223-231 (1988)

[124] Dillmann, R., Huck, M.: *"Informationsverarbeitung in der Robotik."* Springer, Berlin (1991)

[125] Hörmann, K., Werling, V.: *"Ein Verfahren zur Planung von Feinbewegungen für Montageoperationen."* Robotersysteme 5, Heft 1, S.17-28 (1989)

[126] Rembold, U., Dillmann, R.: *"Computer-Aided Design and Manufacturing - Methods and Tools."* Springer, Berlin (1986)

[127] Ravani, B. (ed.): *"CAD Based Programming for Sensory Robots."* Springer, Berlin (1989)

[128] N.N.: *"Facelifting für Realtime."* Computer&Elektronik, April 1986

[129] Shimano, B.E., Geschke, C.C., Spalding, C.H.: *"VAL-II: A Robot Programming Language and Control System."* in: Proc. First Int. Symp. on Robotics Research, MIT Press, Cambridge MA (1982)

[130] Boehnlein, E.: *"SRCL - Siemens Robot Control Language (Syntax)."* Firmenschrift der Fa. Siemens, Erlangen (1985)

[131] N.N.: *"Einfache Programmierung für vielfache Bewegung: Bosch Robotersteuerungen."* Firmenschrift der Fa. Bosch, Erbach (1987)

[132] Blume, C., Jakob, W.: *"Design of the Structured Robot Language (SRL)."* in: Proc. of Advanced Software in Robotics, Lüttich (1983), North-Holland, Amsterdam, 127-143 (1984)

[133] Weck, M., Eversheim, W., Niehaus, T., Zühlke, D., Kalde, M.: *"Requirements for Robot Off-Line Programming shown at the Example ROBEX."* in: Proc. of Advanced Software in Robotics, Lüttich (1983), North-Holland, Amsterdam, S.321-330 (1984)

[134] DIN-Norm 66313, Teil 1: *"IRDATA-Schnittstelle zwischen Programmierung und Robotersteuerung. Allgemeiner Aufbau, Satztypen und Übertragung."* Beuth Verlag, Berlin (1990)

[135] Baumann, K.: *"Werkzeuge und Experimente zur Roboterprogrammierung."* Primärbericht (unveröffentlicht), Kernforschungszentrum Karlsruhe, Karlsruhe (1990).

[136] Olomski, J.: *"Bahnplanung und Bahnführung von Industrierobotern."* Vieweg, Braunschweig, Wiesbaden (1989)

[137] Keppler, M.: *"Führungsgrößenerzeugung für numerisch bahngesteuerte Industrieroboter."* Springer, Berlin (1984)

[138] Leinemann, K.: *"NET Remote Workstation."* KfK-Bericht Nr.4785, Karlsruhe (1990)

[139] Leinemann, K.: *"Advanced Tele-Operator Support for Fusion Plant Maintenance."* Proc. '91 Int. Symp. on Advanced Robots Technology, Tokyo, Japan, 5-7 März 1991

[140] Pritschow, G., Koch, T., Bauder, M.: *"Automatische Erstellung von Rücktransformationen für Industrieroboter unter Anwendung eines optimierten iterativen Lösungsverfahrens."* in: Robotersysteme 5, S.3-8, (1989).

[141] Gupta, C.G., Kazerounian, K.: *"Improved Numerical Solutions of Inverse Kinematics of Robots."* in: Proc. IEEE Conf. on Robotics and Automation, St. Louis, S.743-748 (1985).

[142] Even, P., Marce, L.: *"3D Modelling of a Teleoperation Environment with PYRAMIDE."* Proc. ANS 3rd Topical Meeting on Robotics and Remote Systems, Charleston, SC, USA, 13.-16.3.1989, S.11.1.1-11.1.7

[143] Even, P., Marce, L., Morillon, J., Fournier, R.: *"The modelling system PYRAMIDE as an interactive help for the guidance of the inspection vehicle CENTAURE."* in: Hayward, V., Khatib, O. (Eds.): "Lecture Notes in Control and Information Sciences (139): Experimental Robotics I", Springer, Berlin, S.347-361 (1989)

[144] MacKay, S.A., Sayre, R.E., Potel M.J.: *"3D Galatea: Entry of three-dimensional moving points from multiple perspective views."* Proc. SIGGRAPH '82, ACM Computer Graphics, 16(3), S.213-222 (1982)

[145] Yuan, J.S.C.: *"A General Photogrammetric Method for Determining Object Position and Orientation."* Paper submitted to: IEEE Journal of Robotics and Automation (1988)

[146] Köhler, B.: *"GMS - A High-Precision Geometry Measurement System for Large Fusion Reactor Components."* in: Robotics and Remote Maintenance Concepts for Fusion Machines, IAEA-TECDOC-495, Wien, S.303-310 (1989).

[147] Köhler, B.: *"Surveying of Large Fusion Reactor Components."* in: Proc. 15th SOFT 19-23 Sept. 1988, Utrecht (1988)

[148] Köhler, B.: *"GMS - Ein Fernvermessungssystem zur Korrektur von CAD-Modellen."* Dissertation, Universität Karlsruhe (1991)

Anhang A. Kinematik- und Baugruppendefinitionen

A.1 Notation

Nachfolgend wird auszugsweise das physikalische Dateiformat zur Definition von KISMET-Modellen beschrieben. Zur Definition wird folgende **Notation** benützt:

< >	variable Definitionselemente (non-terminals)
::=	Substitutionsanweisung
[ ] • k	Elemente zwischen [ ] dürfen *0...k* mal wiederholt werden
|	trennt Definitionsalternativen
{ }	Elemente zwischen { } dürfen entfallen
" "	Die Zeichenkette zwischen " " ist eine Definitionskonstante (terminals).
/* */	Kommentarklammer. Der Text zwischen der Klammer dient zur Erläuterung. Der Kommentar selbst ist nicht Bestandteil der Syntaxdefinition.

Weiterhin werden als Variablentypen verwendet:

FLOAT	Real-Zahl in Fließkommaformat oder wissenschaftlicher Notation (E-Format). Die Wortbreite beträgt 32 Bit.
DOUBLE	Real-Zahl mit doppelter Genauigkeit (64-Bit).
INTEGER	Integer-Zahl. Die Wortbreite beträgt 32 Bit.
UNSIGNED	Natürliche Integer-Zahl (≥ 0).
(k1...k2)	Der zulässige Wertebereich des Elements liegt im Intervall $k \in [k1 \ldots k2]$.
STRING	ASCII-Zeichenkette. Die Zeichenkette darf keine Leer- oder sonstigen Sonderzeichen enthalten die als Lesebegrenzer erkannt werden ("C"-Code).

A.2 Syntax von '.mpc'-Dateien

```
<ABSTRACT_file> ::=
        <Version>                       /* KISMET Version ID */
        <comment>                       /* Kommentarzeile */
        <f_body>                        /* Dateirumpf */

<Version> ::= "RBv01r01"
<comment> ::= STRING
<f_body>  ::= <FRAME_file> | <ROB_file> | <BASE_file>
```

Syntax : *ABSTRACT-Datei*

```
<FRAME_file> ::=
        "0"                             /* Dateityp 'FRAME-FILE' */
        <n_abs>                         /* Anzahl ABSTRACT-Datensätze */
        [<MOD_frame>]*n_abs             /* ABSTRACT-Datensätze */

<n_abs> ::= UNSIGNED
```

Syntax : *Baugruppendatei - Ohne Basis-Transformation*

```
<ROB_file> ::=
        "1"                             /* Dateityp 'ROB_FILE' */
        <ROB_name>                      /* Roboter-Eigenname */
        <ROB_type>                      /* Kinematik-Typ */
        <n_abs>                         /* Anzahl ABSTRACT-Datensätze */
        [<ABS_frame>]*n_abs             /* ABSTRACT-Datensätze */
        <DOF_file>                      /* DOF-Datei-Referenz */
        {<CAL_file>}                    /* Optionale Kalibrier-Datei-Referenz */

<ABS_frame> ::= <MOD_frame> | <KIN_frame>

<ROB_name> ::= STRING
<ROB_type> ::= UNSIGNED
<n_abs>    ::= UNSIGNED
<DOF_file> ::= STRING".dof"
<CAL_file> ::= STRING".cal"
```

Syntax : *Spezifikation Kinematik-Datei*

```
<BASE_file> ::=
        "2"                             /* Dateityp 'BASE_FILE' */
        <BASE_par>                      /* Basis-Transformation */
        <n_abs>                         /* Anzahl der ABS-Records */
        [<MOD_frame>]*n_abs             /* Abstract-Spezifikationen */

<BASE_par> ::= <allr_ABB>
<n_abs>    ::= UNSIGNED
```

Syntax : *Baugruppendatei-Spezifikation mit Welt-Transformation*

```
<KIN_frame> ::=
        <FRAME_id>                      /* Gelenk Name */
        <kin_frame_type>                /* Gelenk-KS-Typ */
        <DH_par>                        /* DH-Transformationsparameter */
        <min_max>                       /* min. und max. Bewegungsbereich */
        <init_pos>                      /* Startposition */
        <max_vel>                       /* max. Geschwindigkeit */
        <max_acc>                       /* max. Beschleunigung */
        <Vorgänger>                     /* Vorgänger-Frame ID */
        [<Nachfolger>]                  /* Nachfolge-Frame ID */
        <n_geos>                        /* Anzahl GEO-Referenzen */
        [<GEO_spec>]•n_geos             /* Geometrie-Definitionen */
        <n_wfr>                         /* Anzahl WFRM-Referenzen */
        [<WFR_spec>]•n_wfr              /* Workframe-Definitionen */
        <ABSTR_son>                     /* ABSTARCT-Datei Referenz */

<DH_par>    ::= <wz> <dz> <dx> <wx>
<min_max>   ::= <min> <max>

<FRAME_id>,<Nachfolger>         ::= STRING
<Vorgänger>                     ::= STRING | "AUTO" | "BASE"
<kin_frame_type>                ::= "0" | "1" | "2" | "3"
<min>,<max>                     ::= FLOAT
<wz>,<wx>,<dz>,<dx>             ::= FLOAT
<init_pos>,<max_vel>,<max_acc>  ::= FLOAT
<n_geos>,<n_wfr>                ::= UNSIGNED
<ABSTR_son>                     ::= <ABSTR_spec> | "NULL"
```

Syntax : *Kinematik-FRAME Spezifikation*

```
<MOD_frame> ::=
        <FRAME_id>                      /* Frame-Name */
        <MOD_frame_type>                /* Modellier-KS-Typ */
        <MOD_trans>                     /* Transformationsparameter */
        <Vorgänger>                     /* Vorgänger-Frame ID */
        [<Nachfolger>]                  /* Nachfolge-Frame ID */
        <n_geos>                        /* Anzahl GEO-Referenzen */
        [<GEO_spec>]•n_geos             /* Geometrie-Definitionen */
        <n_wfr>                         /* Anzahl WFRM-Referenzen */
        [<WFR_spec>]•n_wfr              /* Workframe-Definitionen */
        <ABSTR_son>                     /* ABSTARCT-Datei Referenz */

<FRAME_id>,<Nachfolger> ::= STRING
<MOD_trans>             ::= <allr_ABB>
<Vorgänger>             ::= STRING | "AUTO" | "BASE"
<MOD_frame_type>        ::= "50" | "51" | "52" | "53"
<n_geos>,<n_wfr>        ::= UNSIGNED
<ABSTR_son>             ::= <ABSTR_spec> | "NULL"
```

Syntax : *Modellier-FRAME Spezifikation*

```
<GEO_spec>  ::=
        <MOD_trans>                     /* Transformationsparameter */
        <GEO_filename>                  /* Geometriedatei-Referenz */

<MOD_trans>    ::= <allr_ABB>
<GEO_filename> ::= STRING".mpo"
```

Syntax : *Geometriereferenz-Spezifikation*

```
        <allr_ABB> ::= <α> <β> <γ> <dx> <dy> <dz>

<α>,<β>,<γ>,<dx>,<dy>,<dz> ::= FLOAT
```

Syntax : *Modellier-Transformation (6 Freiheitsgrade)*

```
<WFR_spec> ::=
        <WFR_name>                      /* Workframe-Name */
        <WFR_text>                      /* beschreibender Text */
        <WFR_type>                      /* Objekttyp */
        <WFR_state>                     /* Objektzustand */
        <ref_mark>                      /* Referenzmarke */
        <visi>                          /* Sichtbarkeitsindex */
        <trans_par>                     /* Workframe-Placierung */

<WFR_name>,<WFR_text>   ::= STRING
<WFR_type>,<WFR_state>  ::= UNSIGNED
<ref_mark>              ::= UNSIGNED
<visi>                  ::= "0" | "1" | "2" | "3"
<trans_par>             ::= <allr_ABB>
```

Syntax : *Workframe-Spezifikation*

```
<ABSTR_spec> ::=
        <ABSTR_fname>                   /* Abstract-Definition */
        <action>                        /* Lesen oder Warten */

<ABSTR_fname> ::= STRING".mpc"
<action>      ::= "R" | "W"
```

Syntax : *Baugruppenreferenz-Spezifikation*

A.2.1 Erläuternde Parameter-Beschreibung

Version kennzeichnet die Simulatorversion. Dient zur Identifikation der Dateien, bzw. zur versionsabhängigen programminternen Verzweigung während des Einlesens der Daten.

comment Kommentarzeile, die bei der Modellverwaltung zur näheren Erläuterung dienen soll.

f_body Identifikation des Dateityps für den nachfolgenden Datensatz

0 ABSTRACT-File ohne Basis- bzw. Roboterdefinition
1 ROBOTER ABSTRACT-File
2 ABSTRACT-File mit Basistransformation

ROB_name Eigenname des ROBOTERs. Dient zur Identifikation, falls in einer Simulationszelle mehrere ROBOTER gleichzeitig simuliert werden.

ROB_type Kinematiktyp, dient zur Steuerung der inversen Transformation bei Roboter TCP-Bewegungen und zur Bereitstellung ROBOTER spezifischer Bewegungsmenüs. Beispielhaft seien hier genannt:

457 : MANUTEC RM3 Industrieroboter, Gelenkstruktur 'RRR.RRR'
458 : Puma 260/761/762, Gelenkstruktur 'RRR.RRR'
461 : KUKA 161/15 Industrieroboter, Struktur 'RRR.RRR'
470 : HITACHI PW-10 Schweißroboter, Struktur 'RRR.RR'
400 : KUKA 400 Portalroboter, Struktur 'PPP.RRR'
500 : KfK-CATROB-Portal, Struktur 'PP'
217 : Jet "Articulated Boom"
219 : Jet TARM
223 : Servomanipulator (Ein Arm)
287 : Autonomes Fahrzeug, Struktur 'PPR'

DOF_file Dateiname, referenziert die Definition der ROBOTER-Freiheitsgrade (degree-of-freedom).

CAL_file Dateiname. Die Datei dient zur Kalibrierung der Gelenksensor-Schnittstelle zwischen Robotersteuerung und KISMET. Die Dateireferenz ist obligatorisch für ROBOTER, die im Monitor-Betrieb dargestellt werden sollen.

BASE_par Modell-Weltkoordinaten PLACEMENT; definiert die Transformation vom Benutzer-Weltkoordinatensystem zum CAD-Weltkoordinatensystem (Bildschirmkoordinatensystem). Wird die Null-Transformation spezifiziert, so zeigt die Z-Achse des Weltkoordinatensystems aus dem Bildschirm heraus, die X-Achse nach rechts und die Y-Achse nach oben. Die Parameter sind definiert entsprechend der Transformation <allr_ABB>.

n_abs Anzahl der nachfolgend definierten ABSTRACT Datenstrukturen.

ABS_frame Bezeichnet einen ABSTRACT-Datenrecord.

FRAME_id Name des kinematischen Gelenks bzw. eines Modellier-KS.

kin_type definiert den Typus des Gelenks bzw. Umgebungsframes. Bei bewegten Freiheitsgraden (Gelenken) ist zu unterscheiden zwischen Teilen von offenen kinematischen Ketten (Bewegungsachsen) und Teilen von kinematischen Schleifen. Freiheitsgrade, die als Teilgelenk innerhalb einer geschlossenen Schleife definiert sind, werden zwangsgeführt durch ein weiteres Führungskoordinatensystem, dem kinematischen Nachfolger. Der aktuelle Parameter derartiger Gelenkbausteine wird von KISMET automatisch während einer Bewegung neu berechnet und ergibt sich aus den Positionen und Orientierungen der kinematischen Vorgänger- und Nachfolgergelenke.

- 0 : rotatorisches Gelenk (offene kinemat. Kette)
- 1 : translatorisches Gelenk (offene kinemat. Kette)
- 2 : rot. Gelenk mit Zwangsführung (in Schleifen)
- 3 : trans. Gelenk mit Zwangsführung (in Schleifen)
- 50 : Umgebungs-Koordinatensystem
- 51 : verschiebbares Umgebungs-Koordinatensystem
- 52 : alle GEOs werden in Z-Richtung skaliert
- 53 : wie 52, jedoch andere Berechnung der GEO-Skalierung

Bei Frames des Typs '52' wird die Geometrie mit der Gelenkvariablen des kinematischen Vorgängerelements skaliert. Bei Frames des Typs '53' wird der Skalierungsfaktor als Distanz zwischen dem Ursprung des Vor- und des Nachfolgegelenks berechnet.

DH_par Transformationsparameter einer Denavit-Hartenberg Matrix. Die vier Parameter haben dabei folgende Bedeutung:

- wz : Gelenkwinkel von der x_{i-1}-Achse zur x_i-Achse gedreht um die z_{i-1}-Achse (Rechte-Hand-Regel)
- dz : Entfernung vom Ursprung des (i-1)-ten Koordinatensystems zum Schnittpunkt der z_{i-1}-Achse mit der x_i-Achse.
- dx : kürzeste Entfernung zwischen der z_{i-1}-Achse und der z_i-Achse.
- wx : Offset-Winkel von der z_{i-1}-Achse zur z_i-Achse gedreht um die x_i-Achse (Rechte-Hand-Regel)

Die Rotationsparameter wz, wx müssen in Grad [-180.0,180.0] angegeben werden. Die Translationsparameter dz, dx sind in Millimeter zu definieren.

min_max Minimaler und maximaler Variablenwert des Gelenks; d.h. diese Parameter definieren den mechanischen Arbeitsbereich des Gelenks. Für rotatorische Freiheitsgrade erfolgt die Angabe in Grad, für translatorische in Millimetern.

init_pos Anfangsposition des Freiheitsgrads beim Beginn der Simulation.

max_vel Maximal zulässige Geschwindigkeit der Bewegungsachse beim Verfahren über Geschwindigkeitssteuerung. Bei rotatorischen Gelenken muß die Angabe in rad/sec, bei translatorischen in mm/sec erfolgen.

max_acc Maximal zulässige Beschleunigung Bei rotatorischen Gelenken erfolgt die Angabe in rad/sec², bei translatorischen in mm/sec².

Vorgänger identifiziert den kinematischen Vorgänger.

<name> :	Bezeichnung des kinematischen Vorgängers
AUTO :	Vorgänger ist der Vater im hierarchischen Modellbaum
BASE :	Der kinematische Vorgänger ist das definierte Basis-Koordinatensystem des Simulationsmodells.

Nachfolger identifiziert den Nachfolger bei kinematisch zwangsgeführten Bewegungsachsen (Baustein von kin. Schleifen). Dieser Parameter wird nur definiert bei den FRAME-Typen '2' und '3'.

ABSTR_son Referenz auf eine weitere Detaillierungsstufe, d.h. konkret auf eine in der Datenhierarchie tieferverzweigte ABSTRACT-Datei.

Definition von Geometrie-Objekten (GEO)

n_geos Anzahl der mit dem aktuellen FRAME (ABSTRACT-Knoten) verknüpften GEOMETRIE-Objekte.

GEO_spec Geometriespezifikation. Jede Geometriedefinition besteht aus einem Geometrie-PLACEMENT <MOD_trans> und einer Geometriedatei-Referenz <GEO_filename>.

MOD_trans Geometrie-PLACEMENT, definiert die rotatorischen und translatorischen Parameter der Koordinatentransformation vom FRAME-Referenzsystem zum neudefinierten Geometrie Objekt-Koordinatensystem.
Die Parameter α, β, γ definieren Rotationen in Grad, die Translationen dx, dy, dz sind in 'mm' anzugeben. Die aus den Angaben resultierende homogene Transformationsmatrix wird in folgender Reihenfolge berechnet:

translate(dx,dy,dz)
rotate(α,'x')
rotate(β,'y')
rotate(γ,'z')

GEO_filename Dateiname eines GEOMETRIE-Objekts.

Definition von Workframes (WFRM)

n_wfr Anzahl der mit dem aktuellen FRAME zu definierenden WORKFRAME-Objekte.

WFR_spec Workframespezifikationen. Die Workframe-Definition wird gebildet aus dem WORKFRAME-Namen <wfr_name>, aus einem ***PLACEMENT*** <trans_par>, sowie aus Angaben zur Definition von Typ und Zustand des Workframes.

WFR_name Bezeichnung des WORKFRAMES. Wird verwendet zur Identifikation des Workframes während des Programmablaufs. Der Workframe-Name ist optional während der Programmausführung darstellbar und kann dem Operateur zur Orientierung in komplexen Arbeitsumgebungen dienen.

WFR_text Beschreibender Text, Kommentar.

WFR_type Typ des zugehörigen Objektes. Umgebungs- und Werkzeug-Objekttypen sind spezifisch für eine bestimmte Simulationsumgebung und sind der Applikations-Dokumentation zu entnehmen. Es gelten folgende vordefinierte Regeln für WFR_type:

- **0..9999** definieren Umgebungs-Referenzen. KISMET verwaltet lediglich diese Information und ist z.B. in der Lage, einer Robotersteuerung das aktuelle PLACEMENT eines bearbeiteten Objekts zu liefern falls sich dessen Referenz-Koordinatensystem verändert hat und dies durch entsprechende Sensorik erkannt wurde. Unter Verwendung von Umgebungs-Referenzen können Roboterprogramme *objektorientiert* erstellt werden, statt in Weltkoordinaten.
- **10000..10999** definieren ROBOTER-Werkzeuge und entsprechende Roboter TCP-PLACEMENTS (TCPF).
- **11000** definiert ein ROBOTER-Basissystem (Nullpunkt). Das entsprechende ABS_FRAME wird automatisch zur Roboterbasis definiert. Das Workframe PLACEMENT wird umgesetzt als aktuelle Nullpunkt-Verschiebung bzw. -Rotation.
- **15000** definiert eine Kamera-Ansicht (CAMERA). Die Blickrichtung liegt entlang der Z-Achse des Workframes mit dem Betrachterstandpunkt im Ursprung des Workframes.
- **16000** definiert einen Ultraschallsesor (USS). Die Sensorachse liegt entlang der positiven Z-Achse des Workframes.
- **17000** definiert eine mit dem Kinematikelement verknüpfte, bewegliche Lichtquelle.
- **18000** definiert ein spezielles 'Fußbodenelement' (FLOOR).

WFR_state Zustand eines Objektes. Dieser Dateneintrag wird nur von KISMET verwaltet und kann auf Anforderung an die Objekt-Verwaltung der Robotersteuerung übergeben werden. Der Parameter wird z.B. verwendet um die Objektzustände ***'Schraube_festgezogen'*** oder ***'Schraube_gelöst'*** zu markieren.

ref_mark Referenzmarke. Der Dateneintrag wird nur verwaltet und kann auf Anforderung dem Objekt-Management der Robotersteuerung übertragen werden.

visi definiert, ob und in welcher Weise ein WORKFRAME in KISMET darzustellen ist. Mögliche Parameterwerte sind:

0 nicht darstellen
1 FRAME ist sichtbar
2 NAME ist sichtbar, <wfr_name> wird eingeblendet
3 FRAME + NAME sind sichtbar

trans_par Workframe-PLACEMENT, definiert die homogene Transformation zwischen WORKFRAME und ABS_FRAME. Die Transformation wird definiert und berechnet entsprechend <allr_ABB>.

A.3 Syntax von DOF-Dateien (Modell-Freiheitsgrade)

```
<DOF_file> ::=
        <Version>                       /* KISMET-Version ID */
        <comment>                       /* Kommentarzeile */
        <n_dof>                         /* Anzahl Eingabe-Freiheitsgrade */
        [<DOF_spec>]•n_dof              /* DOF-Datensätze */

<DOF_spec> ::=
        <menu_text>                     /* Text im Single-Joint-Menü */
        <outp_text>                     /* Text in Ausgabeliste */
        <n_entries>                     /* Anzahl abhängiger FRAMES */
        [<FUN_entry>]•n_entries         /* Funktions-Definition */

<FUN_entry> ::=
        <FRAME_id>                      /* ID des bewegten KS */
        <FUN_type>                      /* Typ der Umrechnungsfunktion */
        <n_const>                       /* Anzahl Konstanten der DOF-Funktion */
        [<const>]•n_const               /* Funktionskonstanten */

<Version>                       ::= "RBv01r01"
<comment>                       ::= STRING
<menu_text>,<outp_text>         ::= STRING
<FUN_type>                      ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"11"|"12"
<n_dof>,<n_entries>,<n_const>   ::= UNSIGNED
<const>                         ::= FLOAT
```

Syntax : *Freiheitsgrad-Definition(DOF-Eingabefunktionen)*

Anhang B. Liste der DOF-Funktionstypen

Funktionstyp: 0 , lineare inkrementale Funktion

Anzahl Konstanten: 1

$$\Phi_i = \Phi_i + k_0 \Delta u_j \qquad [B.1]$$

Für $\Phi_{i,min} \leq \Phi_i \leq \Phi_{i,max}$, wobei $\Phi_{i,min}$ und $\Phi_{i,max}$ die im '.mpc'-File definierten "Gelenkanschläge" des Frames bezeichnen.

Für $\Phi_i < \Phi_{i,min}$ wird automatisch $\Phi_i = \Phi_{i,min}$ gesetzt, dementsprechend wird für $\Phi_i > \Phi_{i,max}$ automatisch $\Phi_i = \Phi_{i,max}$ gesetzt. Die letztgenannten Bedingungen für "Gelenkanschläge" gelten auch für die anderen, nachfolgend aufgelisteten Funktionstypen falls nicht anderweitig dokumentiert.

Funktionstyp: 1 , periodische lineare inkrementale Funktion

Anzahl Konstanten: 3

$$\Phi_i = \Phi_i + k_0 \Delta u_j \qquad [B.2]$$

Für $k_1 < \Phi_i \leq k_2$, und

$$\Phi_i = k_2 \quad für \quad \Phi_i \leq k_1$$

und

$$\Phi_i = k_1 \quad für \quad \Phi_i > k_2$$

, wobei für die Konstanten k_1 und k_2 gelten muß: $k_1 < k_2$.

Diese Funktion kann verwendet werden für periodische Bewegungen ohne Gelenkanschläge, wie z.B. die Rotation einer Kurbelwelle.

Funktionstyp: 2

Anzahl Konstanten: 3

$$\Phi_i = k_2 \cos(k_0 \Phi_F + k_1) \qquad [B.3]$$

Diese Funktion ist nützlich für die Definition von Zahnrad/Zahnstange-Paarungen mit dem Zahnrad als "antreibendes" Bauteil. Der Vorschub Φ_i der Zahnstange wird berechnet als Funktion der Winkelstellung Φ_F des Zahnrads.

Funktionstyp: 3

Anzahl Konstanten: 3

$$\Phi_i = k_2 \sin(k_0 \Phi_F + k_1) \qquad [B.4]$$

Funktionstyp: 4 , Impulsfunktion mit Führungsachse

Anzahl Konstanten: 3

$$\Phi_i = \Phi_{i,max} \quad für \quad k_1 \leq \Phi_F \leq k_2 \qquad [B.5]$$

und

$$\Phi_i = \Phi_{i,min} \quad für \quad [\Phi_F < k_1, \Phi_F > k_2]$$

Als Anwendungsbeispiel für diesen Funktionstyp sei hier ein pneumatisch angetriebener "Greifer" angeführt. Als Gelenkpositionen sollen lediglich die Stellungen "Auf" und "Zu" möglich sein.

Funktionstyp: 5 , Trapezfunktion mit Führungsachse

Anzahl Konstanten: 5

$$\Phi_i = \Phi_{i,max} \quad für \quad k_2 \leq \Phi_F \leq k_3 \qquad [B.6]$$

und

$$\Phi_i = \Phi_{i,min} \quad für \quad [\Phi_F \leq k_1, \Phi_F \geq k_4]$$

und

$$\Phi_i = \Phi_{i,min} + (\Phi_F - k_1) \frac{\Phi_{i,max} - \Phi_{i,min}}{k_2 - k_1} \quad für \quad k_1 < \Phi_F < k_2$$

und

$$\Phi_i = \Phi_{i,max} - (\Phi_F - k_3) \frac{\Phi_{i,max} - \Phi_{i,min}}{k_4 - k_3} \quad für \quad k_3 < \Phi_F < k_4$$

Funktionstyp: 6 , lineare Funktion mit Führungsachse

Anzahl Konstanten: 1

$$\Phi_i = k_0 \Phi_F \qquad [B.7]$$

Als typischer Anwendungsfall dieser Funktion sei die Modellierung von Zahnrad-Paarungen aufgeführt. Das antreibende Zahnrad mit der Gelenkposition Φ_F und dem Radius r_1 bestimmt direkt den Drehwinkel Φ_i des angetriebenen Rads (Radius r_2). Die Konstante k_0 wird bestimmt zu:

$$k_0 = -\frac{r_1}{r_2}$$

Anmerkung: Die **Funktionstypen 7-10** sind z.Zt. nicht definiert (Reserve).

Funktionstyp: 11 , Kolbenhubfunktion für Kurbeltrieb

Anzahl Konstanten: 4

$$\Phi_i = k_3\left[1-\cos(k_0\Phi_F+k_1)+\frac{1}{k_2}\left[1-\sqrt{1-k_2^2\sin^2(k_0\Phi_F+k_1)}\right]\right] \qquad [B.8]$$

Die Konstante k_3 entspricht dem Radius der Kurbel, die Konstante k_2 ist das Verhältnis zwischen Pleuellänge und Kurbelradius. Die Konstanten k_0 und k_1 erlauben die Einbeziehung von beschleunigtem Kurbel-Drehwinkel und Phasenverschiebung. Die Konstante k_1 wird definiert in Grad.

Funktionstyp: 12 , Pleuelwinkel-Funktion für Kurbeltrieb

Anzahl Konstanten: 3

$$\Phi_i = asin[k_2\sin(k_0\Phi_F+k_1)] \qquad [B.9]$$

Die Konstanten k_0 ... k_2 sind definiert entsprechend Funktionstyp 11.

Anhang C. Geometrie-Datei Spezifikation

C.1 Generelle Syntax von '.mpo'-Dateien

Die generelle Syntax der GEO-Dateien ('.mpo') ist nachstehend dargestellt:

```
<geo_file> ::=
        <comment>
        <GEO_Body>

<GEO_body> ::= <POLYHEDRON> | <3D_SOLID_DATASET>

<3D_SOLID_DATASET> ::=
        <solid_id>
        <geo_body>
        <material_index>

<geo_body> ::= <3D_SOLID_PRIMITIVE>

<3D_SOLID_PRIMITIVE> ::= <SPHERE_FULL> | <BOX> | <CONE_FULL> |
                         <CYLINDER_FULL> | <TRUNCATED_CONE> |
                         <ROTATIONAL_SWEEP> | <LINEAR_SWEEP> |
                         <PIPE> | EXTENDED_POLYHEDRON |
                         <ISOM_LINE> | <CURVE> | <SURF>

<comment>        ::= STRING
<solid_id>       ::= "0"
<material_index> ::= UNSIGNED          /* Wertebereich  (1..255) */
```

Syntax : *Geometrie-Datei Spezifikation*

comment erlaubt das Einfügen von Kommentaren in die Geometrie-Definition. Typischerweise wird das Kommentarfeld zu einer kurzen Charakterisierung des definierten Objekts benutzt.
Der Kommentar wird als Zeichenkette gelesen und muß deshalb mindestens ein Zeichen enthalten.

solid_id Dieses Zeichen wird von KISMET zur Unterscheidung zwischen POLYHEDRON und den anderen GEO-Primitiven benötigt.

geo_body enthält die eigentliche Geometrie-Definition. Die Spezifikation der für die verschiedenen GEO-Primitive unterschiedlichen Parameterblöcke erfolgt in den nachfolgenden Textabschnitten.

material_index ist das 'Material'-Attribut des definierten GEO-Elements. Der Index bezieht sich auf die in der 'MATERIAL'-Datei definierte Tabelle ('.mat').

Beispiel einer einfachen GEO-Datei (Definition einer Kugel):

```
Dies_ist_die_Kommentarzeile__Kugel_R150__(Kugel_bsp.mpo)
0  1
150.0
3  17
```

C.2 *POLYHEDRON Primitiv*

Das POLYHEDRON Primitiv erlaubt die Definition von B-REP Oberflächen, die durch ebene Facetten angenähert beschrieben werden (Polyeder). Diese Geometriedarstellung wird - in der Regel mit unterschiedlicher Syntax - durch eine Vielzahl von CAD-Systemen verwendet.

POLYHEDRONs können durch KISMET lediglich *flat shaded* dargestellt werden. Es wird daher empfohlen das Primitiv EXTENDED_POLYHEDRON für Körper mit gekrümmten Oberflächen zu verwenden, falls weiche Schattierungsübergänge (GOURAUD-Schattierung) und realistischere Lichtberechnungen benötigt werden.

```
<POLYHEDRON> ::=
        <points_count>
        <face_count>
        <vert_list_size>
        <3D_points_list>
        <vert_per_face_list>
        <facet_col_list>
        <matarial_index>
        <vert_index_list>

<3D_points_list>     ::= [<3D_coord>]•points_count
<vert_per_face_list> ::= [<points_per_face>]•face_count
<facet_colour_list>  ::= [<facet_colour>]•face_count
<vert_index_list  >  ::= [<vertex_index>]•vert_list_size

<3D_coord>           ::= <x> <y> <z>

<points_per_face>,
<points_count>,<face_count>,
          <face_list_size>  ::= UNSIGNED
<facet_colour>              ::= "-1"|"-2"| (1..256)
<vertex_index>              ::= (1..<points_count>)
<x>,<y>,<z>                 ::= FLOAT
```

Syntax : *POLYHEDRON-Primitiv*

Erläuterung :

points_count — Anzahl der definierten 3D Oberflächenpunkte in <points_list>.

face_count — Gesamtzahl der in <vert_index_list> definierten POLYLOOPS. POLYLOOPs definieren entweder äußere oder innere Schleifen. Äußere Schleifen werden in <vert_index_list> im Gegenuhrzeigersinn definiert, innere Schleifen (Löcher in der Oberfläche) im Uhrzeigersinn (von außerhalb des gedachten "Materials" betrachtet).

vert_list_size — Ist die Summe aller <points_per_face> aller Facetten in der <vert_index_list>.

3D_points_list — Ist die Liste von kartesischen 3D Raumkoordinaten, die zur geometrischen Approximation der Oberfläche definiert werden. Die Punkte werden von 1 bis <points_count> durchnummeriert.

vert_per_face_list Diese Liste enthält die Anzahl von Eckpunkten für jede definierte POLYLOOP.

facet_col_list Diese Liste definiert den Farbindex jeder POLYLOOP für *Facetten-Schattierung*. Die Farbe dieser Facette wird also nicht durch <material_index> bestimmt.

Der Wert "-1" wählt automatische Schattierung entsprechend den durch <material_index> identifizierten Reflexionseigenschaften und dem aktuellen Beleuchtungsmodell. Der spezielle Wert "-2" kennzeichnet **innere Schleifen** (Inner Loops). Innere Schleifen werden direkt auf die korrespondierende **äußere Schleife** (Outer Loop) folgend definiert.

material_index Kennzeichnet die MATERIAL-Attribute des definierten GEO-Objekts. Dieser Index bezieht sich auf die durch den Benutzer definierbare MATERIAL-Tabelle, die in der MATERIAL-Datei spezifiziert wird ('.mat'). Der Wert '256' ist in KISMET V2.0 vordefiniert für transparente Oberflächen.

vert_index_list Jede POLYLOOP des POLYHEDRON wird durch eine Liste von Indices definiert, die in <3D_points_list> weisen. Der erste 3D Punkt in <3D_points_list> hat den Index "1". Äußere Schleifen werden, von außerhalb des Objekts betrachtet, im Gegenuhrzeigersinn definiert. Der Normalenvektor der Facette weist von innerhalb des gedachten "Materials" heraus. Äußere Schleifen werden in mathematisch positiver Richtung um den Normalenvektor definiert (Rechte Hand Regel).

C.3 *Syntax der 3D_SOLID_DATASET Geometrie Primitive*

Die nachfolgenden Abschnitte definieren die Syntax von <GEO_Body> der verschiedenen 3D_SOLID_DATASET GEO-Primitive.

C.3.1 SPHERE_FULL-Primitiv

```
<SPHERE_FULL> ::=
        "1"                             /* Primitiv Erkennung */
        <radius>                        /* Kugel-Radius */
        <type>                          /* Facettierungsgrad */

<radius> ::= FLOAT
<type>   ::= "1"|"2"|"3"
```

Syntax : *SPHERE_FULL-Primitiv*

SPHERE_FULL definiert eine Kugel durch deren Radius. Der Radius wird in 'mm' angegeben. Der Kugelmittelpunkt liegt im Ursprung des 'Modellier'-Koordinatensystems.
Die Darstellung von Kugeln erfolgt als Maschennetz von gleichseitigen Dreiecken, wobei die Anzahl der zu erzeugenden Facetten durch den Parameter <type> wählbar ist.

Es ist wählbar:

1 : Maschennetz von 32 Facetten
2 : Maschennetz von 128 Facetten
3 : Maschennetz von 512 Facetten wird erzeugt.

C.3.2 CYLINDER_FULL-Primitiv

```
<CYLINDER_FULL> ::=
        "2"                             /* Primitiv Erkennung */
        <length>                        /* Zylinder Länge */
        <radius>                        /* Zylinder Radius */
        <type>                          /* Darstellungs-Typ */
        <side_faces>                    /* Facetten Anzahl */

<length>,<radius> ::= FLOAT
<type>            ::= "0"|"1"|"2"
<side_faces>      ::= UNSIGNED          /* Anzahl ≥ 3*/
```

Syntax : *CYLINDER_FULL-Primitiv*

CYLINDER_FULL definiert eine Zylinder-Oberfläche durch Länge und Radius. Die Symmetrieachse des Zylinders liegt in der Z-Achse und die Bodenfläche in der XY-Ebene (z = 0.0) des Modellier-Koordinatensystems.
Der Parameter < side_facets > erlaubt dem Benutzer, die Zahl der zu erzeugenden Facetten zur Approximation der Zylinder-Seitenfläche in gewissen Grenzen frei zu bestimmen. Eine Erhöhung dieser Zahl ergibt eine genauere Approximation der Zylinder-Oberfläche, führt jedoch auch zu einer Leistungsminderung bezüglich der Rechenzeit.

Der Parameter < type > definiert:

0 : Nur Seitenflächen darstellen
1 : Zusätzlich die Bodenfläche (z = 0.0)
2 : Zusätzlich die Deckfläche (z = length) definieren

Auf die Darstellung der Boden- und Deckflächen kann verzichtet werden wenn diese Flächen durch andere Geometrieteile oder Baugruppen immer verdeckt werden (Geschwindigkeitsgewinn bei der Darstellung !).

C.3.3 CONE_FULL-Primitiv

```
<CONE_FULL> ::=
        "3"                             /* Primitiv Erkennung */
        <length>                        /* Kegel Länge */
        <radius>                        /* Kegel Radius */
        <type>                          /* Darstellungs-Typ */
        <side_faces>                    /* Facetten Anzahl */

<length>,<radius> ::= FLOAT
<type>            ::= "0"|"1"
<side_faces>      ::= UNSIGNED          /* Anzahl ≥ 3*/
```

Syntax : *CONE_FULL-Primitiv*

CONE_FULL definiert eine Kegel-Geometrie durch Länge und Basis-Radius. Die Symmetrieachse liegt in der Z-Achse und die Bodenfläche in der XY-Ebene des Modellier-Koordinatensystems.

Die Kegel-Seitenfläche wird zur Darstellung durch <side_faces> Facetten approximiert.

Der Parameter <type> definiert:

0 : Nur die Kegel-Seitenflächen zeichnen
1 : Zusätzlich die Bodenfläche darstellen

C.3.4 TRUNCATED_CONE-Primitiv

```
<TRUNCATED_CONE> ::=
        "4"                             /* Primitiv Erkennung */
        <length>                        /* Kegelstumpf Länge */
        <radius_1>                      /* Radius Bodenfläche */
        <radius_2>                      /* Radius Deckfläche */
        <type>                          /* Darstellungs-Typ */
        <side_faces>                    /* Facetten Anzahl */

<length>,<radius_1>,
<radius_2>          ::= FLOAT
<type>              ::= "0"|"1"|"2"
<side_faces>        ::= UNSIGNED        /* Anzahl ≥ 3*/
```

Syntax : *TRUNCATED_CONE-Primitiv*

TRUNCATED_CONE definiert die Oberfläche eines Kegelstumpfs durch Länge, oberen und unteren Radius. Die Symmetrieachse liegt in der Z-Achse und die Bodenfläche in der XY-Ebene (z=0.0) des Modellier-Koordinatensystems.

Die Parameter <side_faces> und <type> sind entsprechend dem CYLINDER_FULL Primitiv definiert.

C.3.5 BOX-Primitiv

```
<BOX> ::=
        "5"                             /* BOX-Identifikation */
        <dim_x>                         /* Größe in X-,Y-,Z-Richtung */
        <dim_y>
        <dim_z>

<dim_x>,<dim_y>,<dim_z> ::= FLOAT
```

Syntax : *BOX-Primitiv*

BOX definiert einen Quader mit den Seitenlängen <dim_x>, <dim_y> und <dim_z>. Der Eckpunkt des Quaders mit den Modell-Koordinaten (0,0,0) liegt im Ursprung des Modellier-Koordinatensystems.

C.3.6 ROTATIONAL_SWEEP-Primitiv

Für die GEO-Primitive ROTATIONAL_SWEEP und LINEAR_SWEEP wird gleichermaßen eine 2D-Randkurve **2D_BOUNDING_CURVE** definiert, die aus **Kreisbögen** und **Polygonzügen** zusammengesetzt wird.

Die Randkurve muß nicht geschlossen sein und die einzelnen Abschnitte (Kreisbögen und Polygone) müssen auch nicht zusammenhängend definiert sein. Sollen in sich geschlossene Kurvenzüge (Loops) definiert werden, so sind der erste und der letzte definierte Punkt der Randkurve identisch und doppelt definiert.
Per Definition wird als Außenseite der Randkurve (der Normalenvektor zeigt aus dem angenommenen Material heraus) der rechts der Umlaufrichtung liegende Raum betrachtet.

Nachstehend ist die Syntax der Randkurven-Definition gegeben:

```
<2D_BOUNDING_CURVE> ::=
        <ks_count>                      /* Anzahl Kurvensegmente */
        [<CURVE_ELEMENT>]*ks_count      /* Kurvensegment-Liste */

<CURVE_ELEMENT> ::= <POLY_DEF> | <ARC_DEF>

<POLY_DEF> ::=                          /* Polygon-Kurvensegment */
        "1"
        <n_pnt>                         /* Zahl der Polygon-Eckpunkte */
        [<2D_POINT>]*n_pnt              /* Koordinaten-Liste */

<ARC_DEF> ::=                           /* Kreisbogen-Segment */
        "2"
        <start>                         /* Startpunkt */
        <center>                        /* Kreismittelpunkt */
        <arc_ang>                       /* Eingeschl. Winkel */

<start>,<center>  ::= <2D_point>        /* 2D-Punktkoordinaten */
<2D_POINT>        ::= <x> <z>

<x>,<z>,<arc_ang> ::= FLOAT
<n_pnt>,ks_count> ::= UNSIGNED
```

Syntax : *2D_BOUNDING_CURVE-Primitiv*

Im folgenden Beispiel ist eine Randkurve definiert, die aus einem Polygonabschnitt und einem Kreisbogenabschnitt zusammengesetzt ist. Der Polygonzug ist definiert durch 3 Eckpunkte (2 Kanten).

```
2
1 3
300.0 250.0 300.0 50.0 500.0 50.0
2
500.0 50.0 300.0 50.0
90.0
```

```
<ROTATIONAL_SWEEP> ::=
        "32"                            /* ROTATIONAL_SWEEP-ID */
        <2D_BOUNDING_CURVE>             /* Randkurve */
        <rot_angle>                     /* Rotationswinkel */
        <type>

<rot_angle> ::= FLOAT                   /* (-360..360) */
<type>      ::= "0"|"1"|"2"
```

Syntax : *ROTATIONAL_SWEEP-Primitiv*

ROTATIONAL_SWEEP definiert die Oberfläche bzw. Teile der Oberfläche eines 3D-Rotationskörpers. Die Flächenform ist definiert durch die ebene Randkurve <2D_BOUNDING_CURVE> und den Rotationswinkel <rot_angle>. Als Definitionsebene der Randkurve wird die XZ-Ebene des Modellier-Koordinatensystems betrachtet. Die Rotation erfolgt um die Z-Achse im mathematisch positiven Sinn (Rechte-Hand-Regel).

Durch das Vorzeichen des Rotationswinkels um die Z-Achse bzw. über den Umlaufsinn der definierten Randkurve können wahlweise Innen- oder Außenflächen definiert werden.

Der Parameter <type> legt fest ob nur die Seitenflächen des Rotationskörpers oder auch die Schnittflächen (rot_winkel = 0.0 bzw. rot_winkel = <rot_angle>) zu definieren sind.

Mit <type> wird definiert:

0 : Nur Seitenflächen erzeugen
1 : Zusätzlich erste Schnittfläche (rot_winkel = 0.0)
2 : Zusätzlich die zweite Schnittfläche definieren

C.3.7 LINEAR_SWEEP-Primitiv

```
<LINEAR_SWEEP> ::=
        "31"                                /* LINEAR_SWEEP-ID */
        <2D_BOUNDING_CURVE>                 /* Randkurve */
        <length>                            /* Sweep-Länge */
        <type>

<length> ::= FLOAT
<type>   ::= "0"|"1"|"2"
```

Syntax : *LINEAR_SWEEP-Primitiv*

LINEAR_SWEEP definiert die Oberfläche bzw. Teile der Oberfläche eines durch Verschiebung einer 2D-Randkurve entstandenen 3D-Körpers. Die Flächenform ist definiert durch die ebene Randkurve <2D_BOUNDING_CURVE> und den Translationsparameter <length>. Als Definitionsebene der Randkurve wird die XZ-Ebene des Modellier-Koordinatensystems betrachtet, die Verschiebung erfolgt parallel zur Y-Achse. Die Bodenfläche (Definitionsfläche) liegt in der XZ-Ebene (y = 0.0) des lokalen Systems, die Deckfläche hat die y-Koordinaten y = <length>.
Durch das Vorzeichen von <length> bzw. über den Umlaufsinn der definierten Randkurve können wahlweise Innen- oder Außenflächen definiert werden.

Der Parameter <type> definiert:

0 : Nur Seitenflächen erzeugen
1 : Zusätzlich die Bodenfläche (y = 0.0)
2 : Zusätzlich die Deckfläche (y = <length>) definieren

C.3.8 PIPE-Primitiv

Rohre sind ein in Handhabungszellen häufig verwendetes Bauelement. KISMET verwendet daher ein GEO-Primitiv, das die einfache Definition von zusammenhängenden, gekrümmten Rohrelementen gestattet.

Rohrelemente können aus geraden (zylinderförmigen) Stücken und aus Rohrkrümmern zusammengesetzt werden. Die Symmetriekurve von Krümmern liegt immer auf einem Kreisbogen.

Gerade Rohrsegmente werden über 2 Raumkoordinaten definiert, d.h. Anfangs- und Endpunkt der Zylinder-Symmetrieachse. Rohrkrümmer benötigen zur Definition 3 Raumkoordinaten, d.h. Anfangs- und Endpunkt sowie einen weiteren Punkt auf dem Kreisbogen. Der Endpunkt eines Rohrabschnitts ist gleichzeitig der Anfangspunkt des nächsten Rohrsegments.

```
<PIPE> ::=
        "30"                            /* PIPE-ID */
        <pipe_radius>                   /* Rohr-Radius */
        <point_count>                   /* Anzahl Raumkoordinaten */
        <segment_count>                 /* Anzahl Rohrsegmente */
        <segment_list>                  /* Segmenttyp Liste */
        <3D_point_list>                 /* Koordinatenliste */

<3D_point_list> ::= [<3D_coord>]•point_count
<segment_list>  ::= ["2"|"3"]•segment_count

<3D_coord>      ::= <x>,<y>,<z>
<pipe_radius>   ::= FLOAT               /* (> 0) */
<x>,<y>,<z>     ::= FLOAT
<segment_count> ::= UNSIGNED            /* (≥ 1) */
```

Syntax : *PIPE-Primitiv*

PIPE Gerade Rohrabschnitte werden in der Reihenfolge Anfangspunkt gefolgt von Endpunkt in <3D_point_list> definiert. Rohrkrümmer sind zu definieren in der Reihenfolge Anfangspunkt, gefolgt vom Kurvenpunkt, gefolgt vom Endpunkt.
Zu beachten ist, daß der Endpunkt eines Abschnitts gleichzeitig den Anfangspunkt des nächsten Segments definiert.
In der Segmentliste <segment_list> werden gerade Rohrabschitte mit '2', Rohrkrümmer mit '3' eingetragen.

C.3.9 EXTENDED_POLYHEDRON-Primitiv

Dieses Primitiv ermöglicht die Definition von Geometrie-Elementen, die durch eine Polygonliste und eine Liste von Normalenvektoren an allen Facetten-Eckpunkten geometrisch beschrieben werden.

Optional ist die Spezifikation einer Farbindexliste für jedes Oberflächenpolygon möglich um einfache Farbtexturen der Geometrie-Oberfläche zu bewirken.

Für die Verwendung in KISMET muß die Oberfläche nicht notwendig vollkommen geschlossen definiert werden. **Inner Loops** d.h. "Löcher" werden gekennzeichnet durch eine negative Anzahl von Polygon-Eckpunkten in der Liste <vert_per_face_list>. Es gilt die Konvention, daß zuerst die "Outer Loop" und direkt darauffolgend alle "Inner Loops" eines POLYFACE definiert werden.

```
<EXTENDED_POLYHEDRON> ::=
        "16"
        <face_count>
        <points_count>
        <normal_count>
        <3D_points_list>
        <3D_normal_list>
        <vert_per_face_list>
        <vert_index_list>
        <norm_index_list>
        <texture_switch>
        {<facet_col_list>}

<3D_points_list>      ::= [<3D_coord>]•points_count
<3D_normal_list>      ::= [<3D_direct>]•normal_count
<vert_per_face_list>  ::= [<pcnt>]•face_count
<vert_index_list>     ::= [<vertex_index>]•<pcnt>•face_count
<norm_index_list>     ::= [<normal_index>]•<pcnt>•face_count
<facet_colour_list>   ::= [<facet_colour>]•face_count

<3D_coord>            ::= <x> <y> <z>
<3D_direc>            ::= <x> <y> <z>

<points_count>,<face_count>,
<normal_count>              ::= UNSIGNED
<pcnt>                      ::= INTEGER
<facet_colour>              ::= (1..256)
<vertex_index>              ::= (1..<points_count>)
<normal_index>              ::= (1..<normal_count>)
<texture_switch>            ::= "0"|"1"
<x>,<y>,<z>                 ::= FLOAT
```

Syntax : *EXTENDED_POLYHEDRON-Primitiv*

face_count gibt die Anzahl der zu erzeugenden Oberflächen-Facetten an. Werden Polygone mit mehr als 3 Eckpunkten definiert, so müssen alle Punkte in einer Ebene liegen. KISMET erlaubt die Definition von nicht-konvexen Polygonen.

points_count ist die Anzahl der definierten Raumkoordinaten.

normal_count ist die Anzahl der definierten Oberflächen-Normalenvektoren. Es wird angenommen, daß die Normale aus der Oberfläche bzw. aus dem Material herauszeigt.

3D_points_list ist eine Liste von 3D-Raumkoordinaten, die jedes Polygon geometrisch definieren.

3D_normal_list ist eine Liste von 3D-Normalenvektoren, die verwendet werden um für jede Facette die Normalenrichtung an den Eckpunkten des Polygonzugs zu definieren.

vert_per_face_list enthält die Anzahl von Eckpunkten für jedes definierte Oberflächen-Polygon. "Outer Loops" sind gekennzeichnet durch eine positive Anzahl von Eckpunkten, "Inner Loops" durch eine negative Integerzahl, wobei der Absolutbetrag die Zahl der Eckpunkte der "Inner Loop" definiert.

vert_index_list jede Facette des Polyeders ist definiert über eine Liste von Indices, wobei sich die Indexnummern auf die in <points_list> definierten Raumkoordinaten beziehen. Die Polygon-Eckpunkte werden in der Reihenfolge des Gegenuhrzeigersinns um den angenommenen Normalvektor des ebenen Polygons definiert. Die Größe dieser Liste ist die Summe aller Eckpunkte über alle Facetten.

norm_index_list jede Facette des Polyeders wird außerdem bezüglich der Oberflächen-Normalen durch eine Liste von Indices beschrieben, die sich auf die in <normal_list> definierten Normalenvektoren beziehen. Die Definition erfolgt ebenfalls im Gegenuhrzeigersinn, die Listengröße entspricht der <vert_index_list>. Soll die Facette an den Kanten in der Darstellung "eben" erscheinen, so ist für alle Eckpunkte einer Facette der gleiche Normalen-Index zu verwenden.

texture_switch dieser Parameter legt fest ob nachfolgend eine Farbindexliste definiert wird. Es gilt:

0 : keine Texturliste zu definieren
1 : in der Datei folgt eine Liste von Farb-Indices

facet_colour_list definiert für alle Facetten an den Eckpunkten des Polygonzugs den für die Darstellung zu verwendenden Farbindex. Die Indices beziehen sich auf die im 'COLOUR'-File (Extension '.col') definierte Farbtabelle für konstante Farben. Die Liste eignet sich z.B. für das Aufbringen von andersfarbigen Schriftzügen auf Geometrieteile, für die Hervorhebung von Schaltern auf Schaltschränken oder auch zur Erzeugung von marmorartigen Oberflächen-Texturen. Folgende Sonderfälle sind definiert:

-1	Der GEO-Element Material-Index ist zu verwenden
1..255	Definiert für den Eckpunkt den entsprechenden Farbindex

C.3.10 ISOM_LINE-Primitiv

Dieses Primitiv definiert einen zusammenhängenden Linienzug der aus geraden Stücken und Kreisbögen zusammengesetzt ist. Das ISOM_LINE-Primitiv kann z.B. verwendet werden um die Symmetrieachse von Rohrleitungen in einer groben Detailstufe als Kurvenzug darzustellen (d.h. die Rohrleitung ist im Prinzip doppelt zu definieren, einmal als PIPE und zum zweiten als ISOM_LINE). Der Vorteil liegt im erheblichen Zeitgewinn bei der Darstellung der einfachen Liniengeometrie im Vergleich zur Darstellung der Rohroberfläche.

Gerade Liniensegmente werden über 2 Raumkoordinaten, d.h. Anfangs- und Endpunkt definiert. Kreisbögen benötigen zur Definition 3 Raumkoordinaten, d.h. Anfangs- und Endpunkt sowie einen weiteren Punkt auf dem Kreisbogen. Der Endpunkt eines Liniensegments ist gleichzeitig der Anfangspunkt des nächsten Segments.

Der Kurvenzug wird immer mit der Strichstärke '1' dargestellt. Als aktuelle Farbe wird die für das angegebene MATERIAL definierte 'Wireframe'-Farbe gesetzt.

```
ISOM_LINE ::=
        "46"                            /* ISOM_LINE-ID */
        <point_count>                   /* Anzahl Raumkoordinaten */
        <segment_count>                 /* Anzahl Liniensegmente */
        <segment_list>                  /* Segmenttyp Liste */
        <3D_point_list>                 /* Koordinatenliste */

<3D_point_list> ::= [<3D_coord>]*point_count
<segment_list>  ::= ["2"|"3"]*segment_count

<3D_coord>      ::= <x>,<y>,<z>
<x>,<y>,<z>     ::= FLOAT
```

Syntax : *PIPE-Primitiv*

ISOM_LINE Gerade Liniensegmente werden in der Reihenfolge Anfangspunkt gefolgt von Endpunkt in <vertex_list> definiert. Kreisbogen-Segmente sind anzugeben in der Reihenfolge Anfangspunkt, gefolgt vom Kurvenpunkt, gefolgt vom Endpunkt. Zu beachten ist, daß der Endpunkt eines Abschnitts gleichzeitig den Anfangspunkt des nächsten Segments definiert. In der Segmentliste <segment_list> werden gerade Linienabschitte mit '2', Kreisbögen mit '3' eingetragen.

C.3.11 CURVE-Primitiv

Im CURVE-Primitiv werden 3D-Raumkurven als parametrische Polygonfunktionen analytisch definiert. Diese Kurven können in KISMET z.B. bei der Off-Line-Programmierung zur Definition und zur Visualisierung von Bewegungsbahnen eingesetzt werden.

Das KISMET-Definitionsformat für CURVEs orientiert sich weitgehend an dem gleichnamigen Primitiv in der VDA-Flächenschnittstelle (VDA-FS), abweichend davon können in KISMET mehrere Kurven innerhalb eines CURVE-Primitivs zusammengefaßt werden (zur Reduktion der Gesamtzahl von Modelltransformationen).

```
<CURVE> ::=
        "47"                              /* CURVE-ID */
        <n_curves>                        /* Anzahl der Kurven */
        <curve_list>

<n_curves>  ::= UNSIGNED                  /* (≥ 1) */
<curve_list>::= n_curves•[<CURVE_DEF>]

<CURVE_DEF> ::=                           /* Definiert einen stetigen Kurvenzug */
   <segnum>                               /* Zahl der Segmente des Kurvenzugs */
   (segnum+1)•[<par>]                     /* Parameter-Intervallgrenzen */
     (segnum)•[<SEGMENT_DEF>]

 <segnum>       ::= UNSIGNED              /* (≥ 1) */
 <par>          ::= FLOAT

<SEGMENT_DEF> ::=                         /* Definiert ein Kurvensegment */
   <iord>                                 /* Polynomordnung == Polynomgrad + 1 */
   (iord)•[<ax>]                          /* Polynom Koeffizientenliste */
   (iord)•[<ay>]
   (iord)•[<az>]

 <iord>         ::= UNSIGNED              /* (≥ 2) */
 <ax>,<ay>,<az> ::= FLOAT
```

Syntax : *CURVE-Primitiv*

Erläuterung:

iord ist die Polynomordnung des i-ten Segments, d.h. $iord = Polynomgrad + 1$.

par sind die globalen Parameterwerte "s" an den Anfangs- und Endpunkten der Segmente. Der Parameter-Endwert eines Kurvensegments ist gleichzeitig der Parameter-Startwert des nächsten Segments. Für die Parameter-Intervallgrenzen des i-ten Segments muß gelten:

$$(par_i < par_{i+1}) \quad \text{und} \quad (par_i > 0)$$

Zahlenbeispiel eines KISMET CURVE-Primitivs:

```
47
1
  3
  0.0 1.0 2.0 2.7
    5
      0.0000000  63.8891754 -51.4653710  77.0936432 -48.5175476
      0.0000000   6.4475098  56.9390259 -23.5717163   3.1851807
     30.0000000   0.0000000 -60.0000000  40.0000000 -10.0000000
    3
     41.0000000  -0.9153137  41.9153137
     43.0000000  31.1755678   1.8244324
      0.0000000 -20.0000000  15.0000000
    2
     82.0000000  29.0000000
     76.0000000 -21.0000000
     -5.0000000  -5.0000000
```

Die Berechnung der Kurvenpunkte erfolgt segmentweise mit dem lokalen Parameter u , der sich aus dem globalen Parameter s für das i-te Kurvensegment wie folgt ableitet:

$$u = \frac{(s - \mathrm{par}_{i-1})}{(\mathrm{par}_i - \mathrm{par}_{i-1})}$$

Der lokale Parameter u liegt demnach entlang dem Segment im Wertebereich $u \in [0 \ldots 1]$.
Die kartesischen Koordinaten der die Kurvenform approximierenden Punkte werden für das i-te Segment wie folgt berechnet:

$$x(u) = \sum_{j=0}^{iord_i - 1} \mathrm{ax}_j \bullet u^j$$

$$y(u) = \sum_{j=0}^{iord_i - 1} \mathrm{ay}_j \bullet u^j$$

$$z(u) = \sum_{j=0}^{iord_i - 1} \mathrm{az}_j \bullet u^j$$

Die Koeffizienten sind nach der Polynomordnung ansteigend sortiert aufgelistet, z.B.

$$x(u) = \mathrm{ax}_0 u^0 + \mathrm{ax}_1 u^1 + \mathrm{ax}_2 u^2$$

hat die Koeffizientenliste: ax_0, ax_1, ax_2.

Die Anzahl der in KISMET berechneten und zur Visualisierung gespeicherten Kurvenpunkte ist abhängig von der globalen, durch den Operateur beim Programmaufruf spezifizierten "Modellgenauigkeit" (Aufrufoption "-a <accuracy>").

C.3.12 SURF-Primitiv

Im SURF-Primitiv werden Freiformflächen als parametrische Polygonfunktionen analytisch definiert. Das Primitiv wurde in KISMET hauptsächlich deshalb implementiert, weil Freiformflächen häufig auf CAD-Systemen erzeugt werden, die im Karosseriebau oder bei der Entwicklung von Flugzeugzellen eingesetzt werden.

Das KISMET-Definitionsformat für SURFs orientiert sich weitgehend an dem gleichnamigen Primitiv in der VDA-Flächenschnittstelle (VDA-FS), abweichend davon können in KISMET mehrere Flächenverbände innerhalb eines SURF-Primitivs zusammengefaßt werden (zur Reduktion der Gesamtzahl von Modelltransformationen).

```
<SURF> ::=
        "48"                              /* SURF-ID */
        <n_surfs>                         /* Anzahl Flächenverbände */
        <surf_list>

<n_surfs>   ::= UNSIGNED                  /* (≥ 1) */
<surf_list> ::= n_surfs•[<SURF_DEF>]

<SURF_DEF> ::=                            /* Definiert einen Flächenverband */
   <nps>,<npt>                            /* Segmentanzahl in s- und t-Richtung */
    (nps+1)•[<pars>]                      /* Intervallgrenzen in s-Richtung */
    (npt+1)•[<part>]                      /* Intervallgrenzen in t-Richtung */
   (segnum)•[<SEGMENT_DEF>]

 <nps>,<npt>     ::= UNSIGNED             /* (≥ 1) */
 <pars>,<part>   ::= FLOAT

<SEGMENT_DEF> ::=                         /* Definiert ein Flächensegment */
   <iordu>,<iordv>                        /* Polynomordnungungen in s,t-Richtung */
   (iordu•iordv)•[<ax>]                   /* Koeffizientenliste */
   (iordu•iordv)•[<ay>]
   (iordu•iordv)•[<az>]

 <iordu>,<iordv> ::= UNSIGNED             /* (≥ 2) */
 <ax>,<ay>,<az>  ::= FLOAT
```

Syntax : *SURF-Primitiv*

Erläuterung:

nps,npt Anzahl von Flächenverband-Segmenten in s- bzw. in t-Richtung. Die Definitionsreihenfolge der Segmente entspricht dabei den Elementen einer Matrix, wobei der Parameter "s" mit dem Spaltenindex und der Parameter "t" mit dem Zeilenindex wächst.

iordu Polynomordnung des aktuellen Flächensegments in u-Richtung.

iordv Polynomordnung des aktuellen Flächensegments in v-Richtung.

pars sind die globalen Parameterwerte "s" an den Anfangs- und Endpunkten der Flächensegmente in s-Richtung. Dem globalen Parameter "s" entspricht der lokale, auf den Wertebereich [0 ... 1] normierte Parameter "u". Der Parameter-Endwert eines Flächensegments ist gleichzeitig der Parameter-Startwert des nächsten Segments. Für die Parameter-Intervallgrenzen des i-ten Segments muß gelten: $(\text{pars}_i < \text{pars}_{i+1})$ und $(\text{pars}_i > 0)$.

part sind die globalen Parameterwerte "t" an den Anfangs- und Endpunkten der Flächensegmente in t-Richtung. Dem globalen Parameter "t" entspricht der lokale, auf den Wertebereich [0 ... 1] normierte Parameter "v".

Die Berechnung der Flächenpunkte erfolgt segmentweise mit den lokalen Parametern u , v , die sich aus den globalen Parametern s , t für das Flächensegment in der i-ten Spalte und in der m-ten Zeile wie folgt ableiten:

$$u = \frac{(s - \text{pars}_{i-1})}{(\text{pars}_i - \text{pars}_{i-1})}$$

$$v = \frac{(s - \text{part}_{m-1})}{(\text{part}_m - \text{part}_{m-1})}$$

Die lokalen Parameter u liegen demnach innerhalb eines Flächenverband-Segments im Wertebereich: $u,v \in [0 \ldots 1, 0 \ldots 1]$.
Die kartesischen Koordinaten der die Kurvenform approximierenden Punkte werden für das Segment in der i-ten Spalte und der m-ten Zeile wie folgt berechnet:

$$x(u,v) = \sum_{k=0}^{iordv_m - 1} \sum_{j=0}^{iordu_i - 1} \text{ax}_{j,k} \cdot u^j \cdot v^k$$

$$y(u,v) = \sum_{k=0}^{iordv_m - 1} \sum_{j=0}^{iordu_i - 1} \text{ay}_{j,k} \cdot u^j \cdot v^k$$

$$z(u,v) = \sum_{k=0}^{iordv_m - 1} \sum_{j=0}^{iordu_i - 1} \text{az}_{j,k} \cdot u^j \cdot v^k$$

Die Koeffizienten sind nach der Polynomordnung ansteigend sortiert aufgelistet, z.B. für $iordu = 3$, $iordv = 4$

$$\begin{aligned} x(u,v) = {} & \text{ax}_0 u^0 v^0 + \text{ax}_1 u^1 v^0 + \text{ax}_2 u^2 v^0 + \\ & \text{ax}_3 u^0 v^1 + \text{ax}_4 u^1 v^1 + \text{ax}_5 u^2 v^1 + \\ & \text{ax}_6 u^0 v^2 + \text{ax}_7 u^1 v^2 + \text{ax}_8 u^2 v^2 + \\ & \text{ax}_9 u^0 v^3 + \text{ax}_{10} u^1 v^3 + \text{ax}_{11} u^2 v^3 \end{aligned}$$

hat die Koeffizientenliste: $\text{ax}_0, \text{ax}_1, \text{ax}_2, \ldots, \text{ax}_{11}$.

Anhang D. Liste der KISMET SCRIPT-Kommandos

Für SCRIPT-Files gelten die Konventionen:

- SCRIPT-Files werden von **KISMET** in dem Directory **'$kis_home/scripts'** gesucht.
- SCRIPT-Files werden in der Regel textuell erzeugt (z.B. per 'vi'-editor).
- Jedes Kommando wird im SCRIPT-File mit dem Zeichen ';' terminiert. Dies gilt nicht falls SCRIPT-Kommandos über die 'message'-Schnittstelle gesendet werden. Im letzteren Fall muß die UNIX-message genau ein SCRIPT-Kommando inklusive aller notwendigen Parameter enthalten.
- **Kommentarzeilen** enthalten in der ersten Spalte der Zeile das Sonderzeichen '#'.

D.1 SCRIPT-Sprachumfang

Die Elemente der SCRIPT-Kommandosprache lassen sich in zwei Funktionsklassen untergliedern:

1. **Allgemeine SCRIPT-Kommandos** zur Steuerung eines Simulationsablaufs bzw. einer Animations-Sequenz, und
2. SCRIPT-Kommandos zur Definition von **physikalischen Roboterparametern** bzw. zur Definition der Charakteristika einer simulierten **Robotersteuerung.**

Im Folgenden werden die einzelnen SCRIPT-Kommmandos aufgeführt.

D.1.1 Allgemeine SCRIPT-Kommandos

- **ACTIVATE <frame_node>**

  aktiviert (macht sichtbar) einen bereits von **KISMET** geladenen Teilbaum bzw. lädt Teilbaum aus der Datenbank, falls er noch nicht geladen wurde. Unter **Teilbaum** wird hier und im folgenden das Äquivalent eines '.mpc'-Files verstanden. Unter <frame_node> wird ein Textstring von **KISMET** erwartet, der den kinematischen Baumknoten eindeutig beschreibt (voll qualifiziert). Dieser Textstring besteht aus den Textstrings <FRAME_id>, wobei verschiedene Baumhierarchien durch '.' (Punkt) getrennt werden.

  Beispiel:

  ```
  ACTIVATE  Testlab_1.FL_Klein.Flansch_Typ1
  ```

- **DEACTIVATE <frame_node>**

  deaktiviert einen Hierarchie-Datenzweig. Der Teilbaum wird nicht aus der Laufzeit-Datenstruktur gelöscht, bei der Modell-Darstellung und Kollisionsberechnungen jedoch nicht berücksichtigt.

  Beispiel:

  ```
  DEACTIVATE  Testlab_1.FL_Klein.Flansch_Typ1
  ```

- **DELETE_NODE <frame_node>**

  löscht einen Teilbaum aus der Laufzeit-Datenstruktur

  Beispiel:

  ```
  DELETE_NODE  Testlab_1.FL_Klein.Flansch_Typ1
  ```

- **SWAP_NODE <frame_node>**

  Ersetzt Teilbaum durch Ersatzgeometrie bzw. je nach aktuellem Zustand auch Ersatzgeometrie durch Teilbaum.

  Beispiel:

  ```
  SWAP_NODE  Testlab_1.FL_Klein.Flansch_Typ1
  ```

- **LEVEL_DOWN <frame_node>**

  lädt und aktiviert ab dem definierten Datenknoten eine weitere Hierarchieebene von '.mpc'-Files dazu, und ersetzt gleichzeitig die Ersatzgeometrien der 'Vater-Knoten'. Der Detaillierungsgrad nimmt zu.

  Beispiel:

  ```
  LEVEL_DOWN  Testlab_1.FL_Klein
  ```

- **LEVEL_UP <frame_node>**

  deaktiviert ab dem definierten Teilbaumknoten die unterste Ebene von Teilbäumen, d.h. von geladenen '.mpc'-Files. Die Detaillierungsgrad nimmt ab.

  Beispiel:

  ```
  LEVEL_UP  Testlab_1.FL_Klein
  ```

- **ROBOT_LANGUAGE <language>**

  setzt die aktuelle "Robotersprache". Ab diesem Zeitpunkt werden zu ladende Roboterprogramme in der definierten Sprache gelesen, d.h. entsprechende Semantik- und Syntaxregeln werden angewandt. Derzeit gültige Sprachen (<language>) sind:

  irdata | srcl | jet_boom | jet_tarm

  Beispiel:

  ```
  ROBOT_LANGUAGE  irdata
  ```

- **ROBACT <Robot_id>**

  aktiviert den spezifizierten ROBOTER als das aktuelle kinematische Modell.
  Das bedeutet konkret, daß der Eingabefokus für Bewegungsbefehle, TCP-Parameterdefinitionen usw. auf den definierten ROBOTER gerichtet ist. Ebenso werden bei der Ausgabe von Gelenkwinkeln, TCP-Koordinaten, Roboter-Programmschrittnummer, Taktzeiten usw. die Parameter des betreffenden Modells verwendet.

  Beispiel:

  ```
  ROBACT  mantec_R3
  ```

- **TF_LOAD <teach_filename> <Robot_id>**

  lädt das spezifizierte Roboterprogramm und ordnet es dem definierten Roboter zu. Für das Argument <Robot_id> ist der Parameter <ROB_name> zu verwenden wie im '.mpc'-File des Roboters definiert. Zu beachten ist, daß das geladene Roboterprogramm noch nicht gestartet wird.

  Beispiel:

  ```
  TF_LOAD  r3demo  mantec_R3
  ```

- **TF_RUN <teach_filename> <Robot_id> {exec_wait}**

  startet ein zuvor mit TF_LOAD geladenes Roboterprogramm. Der optionale Parameter **exec_wait** bewirkt eine Verzögerung des Ausführungsbeginns bis ein bereits gestartetes Roboterprogramm seine Abarbeitung beendet hat.
  Das Roboterprogramm wird nebenläufig zur SCRIPT-Datei gestartet. Diese wird unmittelbar fortgesetzt, d.h. KISMET unterbricht nicht die Abarbeitung der SCRIPT-Datei bis das Roboterprogramm beendet ist.

  Beispiel 1:

  ```
  TF_RUN  r3demo  mantec_R3;
  TF_RUN  r3demo2  mantec_R3a;
  ```

  Die beiden Roboterprogramme 'r3demo' und 'r3demo2' werden gleichzeitig gestartet und parallel abgearbeitet. Eine Synchronisation der Bewegungszyklen via 'exec_wait'-Mechanismus erfolgt nicht.
  Der Implementierungsstand von IRDATA gestattet jedoch die Synchronisation von parallel simulierten Roboterprogrammen aus IRDATA-Programm heraus (Verwendung von SEMAPHOREN).

  Beispiel 2:

  ```
  TF_RUN hs1 ZELLENKRAN;
  TF_RUN hs2 ZELLENKRAN exec_wait;
  TF_RUN hs3 ZELLENKRAN exec_wait;
  ```

  Im zweiten Beispiel werden die Roboterprogramme 'hs1..hs3' gleichzeitig aktiviert und zunächst nur 'hs1' abgearbeitet. Die Ausführung von 'hs2' erfolgt erst nach Beendigung von 'hs1', 'hs3' wiederum wird erst nach Beendigung von 'hs2' gestartet usw. .

- **TF_DELETE <teach_filename> <Robot_id>**

  löscht ein zuvor mit TF_LOAD geladenes und gegebenenfalls mit TF_RUN ausgeführtes Roboterprogramm.

  Beispiel:

  ```
  TF_DELETE  ird_getschr  MANTEC_RM3
  ```

- **LOAD_VIEWS <view_filename>**

  lädt einen in **KISMET** erzeugten binären 'View-File', default extension '.vws'. View-Files werden bei der Erzeugung in die directory '$kis_home/scripts' geschrieben.

  Beispiel:

  ```
  LOAD_VIEWS  view1.vws
  ```

- **EXEC_SCRIPT <script_file>**

  Aufruf eines weiteren 'Scripts' als ein SCRIPT-Kommando, ähnlich dem Aufruf eines Unterprogramms in höheren Programmiersprachen.
  Der Parameter <script_name> darf aus offensichtlichen Gründen nicht gleich dem Filenamen des gerade ausgeführten SCRIPT-Files sein.

  Beispiel:

  ```
  EXEC_SCRIPT  trs_demo.spt
  ```

- **DRAW_SCENE**

  das Kommando bewirkt die Neuberechnung der Kinematik und eine Neuerzeugung der Szenendarstellung. Der Befehl sollte benutzt werden als Abschluß einer Sequenz von 'FRAME_TO_WFRM'-, 'LOAD_VIEWS' oder 'CONNECT'-Kommandos.

- **CONNECT <frame_2> TO <frame_1>**

  Führt eine kinematische Verkettung von <frame_1> mit <frame_2> durch, d.h. daß <frame_1> als neuer kinematischer Vorgänger von <frame_2> definiert ist. Die relative Transformationsmatrix zwischen <frame1> und <frame2> wird so berechnet, daß die augenblickliche Distanz und Orientierung zwischen den FRAMEs beibehalten wird.
  Diese Operation findet Verwendung beim Roboter-Werkzeugwechsel oder beim Greifen von Umgebungsteilen. Da die relative Lage zwischen dem 'greifenden' System und dem 'gegriffenden' Objekt beibehalten wird, eignet sich die Funktion zum Greifen von Teilen bzw. Unterbaugruppen in beliebigen Greifpositionen. Aus diesem Grund sollte sich bei einem Werkzeugwechsel die 'greifende' Kinematik im Augenblick der Operation in der exakten Koppelposition befinden.

  Beispiel:

  ```
  CONNECT  Testlab_1.FL_Klein_02.Flansch_Typ1  TO  Mantec_1.HAND
  ```

- **FRAME_TO_WFRM <frame_2> TO <frame_1> <WFRM_name1>**

  Führt eine kinematische Verkettung von <frame_1> mit <frame_2> durch, wobei hier die vordefinierte Einbauposition <WFRM_name1> zur Berechnung der relativen Transformation zwischen <frame_1> und <frame_2> benutzt wird.
  Diese Operation findet - ähnlich dem 'CONNECT'-Kommando - Verwendung bei einem Roboter-Werkzeugwechsel oder bei der Positionierung von Umgebungsteilen. Falls sich hier bei einem Werkzeugwechsel die Roboterkinematik nicht in der korrekten Werkzeug-Wechselposition befindet, so 'springt' das Werkzeug in die korrekte Koppelposition.

  Beispiel:

  ```
  FRAME_TO_WFRM Toolbox.Tool1 TO  Mantec_1.HAND Tool1_WFRM
  ```

- **POS_ROB <Robot_id> <mode> <position_data>**

  Inkrementale oder absolute Positionierung des durch <Robot_id> spezifizierten ROBOTER-Modells. Der Parameter <mode> bestimmt die Art der Positionierung. 'JOINT_ABS' selektiert absolute, 'JOINT_INC' entsprechend inkrementale Positionierung in Gelenk-Koordinaten. Die Anzahl der Gelenkvariablen muß der in KISMET für das ROBOTER-Modell definierten Anzahl von Freiheitsgraden entsprechen.

  Beispiel:

  ```
  POS_ROB MANTEC_RM3 JOINT_ABS 27.5 -17.38 -11.02 37.338 0.9 -12.271
  POS_ROB MANTEC_RM3 JOINT_INC -0.1 0 0 0.85 0 -0.3
  ```

- **SET_FRAME <frame_node> RENDER <draw_parameter>**

  setzt für den in <frame_node> spezifizierten Datenknoten (bzw. für die mit dem Datenknoten verknüpften Geometrieteile) den in <draw_parameter> spezifizierten Zeichenmodus.

  Mögliche Werte für <draw_parameter> sind:
  2 : Darstellung als Drahtmodell
  3 : "Flat-Shaded" Darstellung
  4 : "GOURAUD-Shaded" Darstellung
  20 : Transparente Darstellung

  Beispiel:

  ```
  SET_FRAME Testlab_1.FL_Klein.Flansch_Typ1 RENDER 2
  ```

  Das spezifizierte Bauteil wird als Drahtmodell dargestellt.

- **SET_ABSTRACT <frame_node> RENDER <draw_parameter>**

  setzt für den in <frame_node> spezifizierten Datenknoten **und alle darunterliegenden Detailstufen**, d.h. für alle an dem Frame angehängten ABSTRACT-Dateien des Typs '.mpc', den definierten Zeichenmodus. Die Parameter sind analog zum Kommando 'SET_FRAME' definiert.

- **SET_FRAME <frame_node> PLACEMENT <place_parameter>**

  setzt für das mit <frame_node> identifizierte Koordinatensystem (FRAME) eine neue relative Transformationsmatrix. Deren Parameter <place_parameter> werden gegeben in der Reihenfolge:

  <wx>,<wy>,<wz>,<dx>,<dy>,<dz>

  - die drei Rotationsparameter "wx,wy,wz" werden im Gradmaß gegeben, die Translationsparameter "dx,dy,dz" in 'mm'. Dieses SCRIPT-Kommando bietet somit die Möglichkeit zur Repositionierung von Komponenten im Arbeitsbereich. Die Transformationsparameter könnten z.B. "On-Line" durch ein 3D-Vermessungssystem geliefert werden. Die Kopplung zu KISMET würde über einen nebenläufigen UNIX-Prozeß und die "Message-Queue" erfolgen.

  Beispiel:

```
SET_FRAME Testlab_1.FL_Klein.Flansch_Typ1 PLACEMENT
          -88.5 16.32851 -180.0 -11920.0 2517.3 912.7
```

- **ROBOT <Robot_id> [online | offline]**

  wird für das durch den Parameter <Robot_id> definierte ROBOT Modell verwendet zum ein-/ausschalten des **Monitor-Modus**. Im Monitor-Modus wird ein Datenkanal zwischen KISMET und der Robotersteuerung zur Übertragung der Gelenkpositionen geöffnet. Die empfangenen Sensordaten werden von KISMET zur Erzeugung synthetischer Szenenansichten der Arbeitszelle entsprechend den aktuell von der Robotersteuerung übertragenen Achspositionen benutzt.

  Beispiel:

```
ROBOT MANTEC_R3 online
```

- **ROBOT <Robot_id> [send_on | send_off]**

  dieses Kommando steuert die **Ausgabe** der aktuellen Roboter-Positionsdaten über die *message* "SENSOR"-Datenschnittstelle (Sensorsimulation) durch KISMET. Die Funktion wird benutzt zum Monitor-Betrieb der KISMET-Robotersimulation durch andere UNIX Prozesse.

  Beispiel:

```
ROBOT MANTEC_R3 send_off
```

- **SET_CAMPAR <CAM_id> <view_dist> <fovy_angle>**

  setzt die Sichtparameter der simulierten CAMERA mit dem WFRM-Bezeichner <CAM_id>. Die Parameter <view_dist> und <fovy_angle) definieren die Distanz zur Fokussierebene (Schärfeebene; Maßeinheit 'Meter'), respektive den Bildöffnungswinkel der Sichtpyramide in X-Richtung (Einheit 'Grad').

  Die Funktion wird benutzt zur Simulation der "realen" Beobachtungskameras in der Arbeitszelle. Für bewegliche Szenenkameras wird deren kinematische Struktur in KISMET als ROBOT definiert.

  Beispiel:

```
SET_CAMPAR CAMERA_5 4.85 15.71
```

- **SET_USSPAR active <USS_id> [0 | 1]**

  das Kommando startet und stoppt die Simulation von Ultraschall-Sensoren (USS). Der Sensor wird identifiziert durch dessen WFRM-Bezeichner <USS_id>. Der Steuerparameter "1" schaltet den USS an, der Wert "0" entsprechend wieder aus. Im eingeschalteten Zustand berechnet KISMET in jedem Programmzyklus die Distanz zum nächsten Hindernis innerhalb der definierten Sensorreichweite. Das Ergebnis wird über den Sensordatenkanal ausgegeben (*message* Schnittstelle).

  Beispiel:

  ```
  SET_USSPAR active USS_1B 0
  ```

- **SET_USSPAR sendmode [0 | 1]**

  der Befehl konfiguriert den USS *message* Ausgabemodus. Mit dem Steuerparameter "0" wird für jeden Sensor eine *message* gesendet, mit dem Wert "1" werden in einem Datenpaket alle USS-Simulationsdaten als Datenfeld gesendet.

  Beispiel:

  ```
  SET_USSPAR sendmode 1
  ```

- **SET_USSPAR range <USS_id> <USS_range>**

  setzt für den simulierten USS mit dem WFRM-Bezeichner <USS_id> die Reichweite des Sensors. Die Reichweite <USS_range> (der positive Abstand vor dem Sensor innerhalb dessen noch Hindernisse detektierbar sind) ist in 'mm' anzugeben.

  Beispiel:

  ```
  SET_USSPAR range USS_5 3500.0
  ```

- **TAG_DRAWLEN <len_value>**

  definiert in 'mm' die Darstellungsgröße für WFRM Koordinatensysteme. Der von KISMET benutzte Initialwert von 250mm ist günstig für mittelgroße Arbeitszellen.

  Beispiel:

  ```
  TAG_DRAWLEN 50.0
  ```

D.1.2 SCRIPT-Kommandos zur Definition der Roboter-Charakteristika

Die folgenden SCRIPT-Kommandos dienen zur Spezifikation der Robotercharakteristika, bzw. zur Definition der in einer realen Robotersteuerung möglichen Bewegungsfunktionen und -parameter. Es wird empfohlem, für jeden Robotertyp eine SCRIPT-Datei anzulegen worin dessen kinematische Parameter definiert werden. Alle von KISMET zur Roboter-Bewegungssimulation verwendeten kinematischen Parameter werden auf bestimmte Werte voreingestellt bzw. initialisiert. Die SCRIPT-Datei sollte diese Werte entsprechen dem Startzustand der realen Robotersteuerung setzen um für die Off-Line-Programmierung den gleichen Ausgangszustand wie in der realen Steuerung setzen zu können.

Anmerkung: Alle folgenden Kommandos beziehen sich jeweils auf den **aktiven** Roboter. Die SCRIPT-Datei sollte daher als erstes Kommando den Befehl **ROBACT** enthalten.

- **rob_cart_lin_vel <lin_vel>**

  Definiert die maximale Bahngeschwindigkeit für kartesisches Bahnfahren (Linearbewegung) in der Maßeinheit 'mm/sec'.

  Beispiel:

  ```
  rob_cart_lin_vel 1200.0
  ```

- **rob_cart_lin_acc <lin_acc>**

  Definiert die maximale Beschleunigung für kartesisches Bahnfahren in der Maßeinheit 'mm/sec^2'. Diese Größe definiert zusammen mit <lin_vel> die Beschleunigungszeit beim linearen Bahnfahren. In dem gewählten Beispiel würde sich eine Beschleunigungszeit von 250ms ergeben.

  Beispiel:

  ```
  rob_cart_lin_acc 4800.0
  ```

- **rob_cart_rot_vel <ang_vel>**

  Definiert die maximale Rotationsgeschwindigkeit für Werkzeug-Orientierungsänderungen während des kartesischen Bahnfahrens in der Maßeinheit 'Grad/sec'.

  Beispiel:

  ```
  rob_cart_rot_vel 360.0
  ```

- **rob_cart_rot_acc <lin_acc>**

  Definiert die maximale Winkelbeschleunigung für Änderungen der Werkzeug-Orientierung während der CP-Bewegung. Die verwendete Maßeinheit ist 'Grad/sec^2'.

  Beispiel:

  ```
  rob_cart_rot_acc 720.0
  ```

- **rob_cart_vel_scal <lin_acc>**

  Definiert den Geschwindigkeits-Skalierungsfaktor für kartesische Linear-Bahnbewegungen in Prozent der Maximalgeschwindigkeit (siehe 'rob_cart_lin_vel'). Der Wertebereich beträgt [0.01..1.0] entsprechend 1% bis 100%. Im Beispiel wird die aktuelle Bahngeschwindigkeit auf 30% des Maximalwerts gesetzt. Durch die Skalierung der Bahngeschwindigkeit wird auch die Winkelgeschwindigkeit für Werkzeug-Orientierungsänderungen um denselben Betrag abgesenkt.

  Beispiel:

  ```
  rob_cart_vel_scal 0.3
  ```

- **rob_cart_acc_scal <lin_acc>**

  Definiert den Beschleunigungs-Skalierungsfaktor für kartesische Linearbewegungen in Prozent der Maximalbeschleunigung (siehe 'rob_cart_lin_acc'). Der Wertebereich beträgt [0.01..1.0] entsprechend 1% bis 100%. Im Beispiel wird die aktuelle Bahnbeschleunigung auf 65% des Maximalwerts gesetzt.

  Beispiel:

  ```
  rob_cart_acc_scal 0.65
  ```

- **rob_jnt_max_range <njoints•[max_pos]>**

  Definiert den mechanischen Maximalwert <max_pos> des Verfahrbereichs der einzelnen Roboterachsen. Die Definitions-Reihenfolge entspricht der Reihenfolge im zugehörigen '.mpc'-File. Die Maßeinheit ist 'mm' bzw. 'Grad' für prismatische, respektive rotatorische Gelenke.

  Beispiel:

  ```
  rob_jnt_max_range 162.5 110.0 145.0 187.5 115.5 270.0
  ```

- **rob_jnt_min_range <njoints•[min_pos]>**

  Definiert den mechanischen Minimalwert <min_pos> des Verfahrbereichs der einzelnen Roboterachsen. Der gesamte Achs-Verfahrbereich muß nicht symmetrisch zur Nullstellung definiert sein.

  Beispiel:

  ```
  rob_jnt_min_range -162.5 -110.0 -145.0 -187.5 -115.5 -270.0
  ```

- **rob_jnt_start_pos <njoints•[start_pos]>**

  Definiert die Startposition des Roboters zu Beginn der Simulation.

  Beispiel:

  ```
  rob_jnt_start_pos 0.0 50.0 -110.0 0.0 -150.0 -180.0
  ```

- **rob_jnt_ref_pos <njoints•[ref_pos]>**

  Definiert die Referenz-Position des Roboters. Diese Position kann z.B. die Ruheposition der Steuerung sein, oder auch die Position zum Referenzieren inkrementaler Winkelgeber.

  Beispiel:

  ```
  rob_jnt_ref_pos 17.35 0 -12.79307 1.507E-03 0.0 15.2003
  ```

- **rob_jnt_max_vel <njoints•[max_vel]>**

  Definiert den Betrag der Maximalgeschwindigkeiten für die einzelnen Roboterachsen. Die Geschwindigkeit ist hier definiert in den Maßeinheiten 'mm/sec' (prismatische Gelenke) bzw. 'rad/sec' (rotatorische Freiheitsgrade).

  Beispiel für einen sechsachsigen Industrieroboter:

```
rob_jnt_max_vel 1.919862 1.483553 5.235988
                3.14159 4.014257 3.926991
```

- **rob_jnt_min_vel <njoints•[min_vel]>**

  Manche Robotersteuerungen besitzen - typischerweise bei Schrittmotorantrieben - neben einer Maximalgeschwindigkeit ebenfalls eine Achs-Mindestgeschwindigkeit, die das lineare Bahnfahren negativ beeinflussen kann.
  Diese Minimalgeschwindigkeiten der einzelnen Roboterachsen wird über das Kommando <rob_jnt_min_vel> definiert. Es werden wieder die Maßeinheiten 'mm/sec' und 'rad/sec' benutzt.

  Beispiel für einen dreiachsigen Roboter:

```
rob_jnt_min_vel 8.5 3.14159E-04 4.014257E-04
```

- **rob_jnt_act_vel <njoints•[act_vel]>**

  Definiert den Betrag der aktuellen Achsgeschwindigkeiten nach dem Einschalten der Robotersteuerung. Dieser Wert wird bei der Simulation von Roboterprogrammen benutzt falls im Programm nicht vor dem ersten Bewegungskommando die Geschwindigkeit definiert wird.

  Beispiel für einen dreiachsigen Roboter:

```
rob_jnt_act_vel 1300.0 3.14159 1.5707963
```

- **rob_jnt_max_acc <njoints•[max_acc]>**

  Definiert den Betrag der Maximalbeschleunigungen für die einzelnen Roboterachsen. Die Beschleunigung wird definiert in den Maßeinheiten 'mm/sec^2' (prismatische Gelenke) bzw. 'rad/sec^2' (rotatorische Freiheitsgrade).

  Beispiel für einen sechsachsigen Industrieroboter:

```
rob_jnt_max_acc 1.919862 1.483553 5.235988
                3.14159 4.014257 3.926991
```

- **rob_jnt_act_acc <njoints•[act_acc]>**

  Definiert den Betrag der aktuellen Achsbeschleunigungen nach dem Einschalten der Robotersteuerung. Dieser Wert wird bei der Simulation von Roboterprogrammen benutzt, falls im Programm nicht vor dem ersten Bewegungskommando die Beschleunigung neu definiert wird. Im untenstehenden Beispiel werden die Beschleunigungswerte auf 100% des oben definierten Maximalbetrags gesetzt.

  Beispiel für einen sechsachsigen Roboter:

```
rob_jnt_act_acc 1.919862 1.483553 5.235988
                3.14159 4.014257 3.926991
```

- **rob_PTPmode [sync|async]**

  Definiert das Verhalten der Robotersteuerung bei PTP-Bahnfahren. KISMET kann synchrones (Parameter: sync) oder asynchrones (Parameter: async) PTP-Bahnverhalten simulieren.

  Beispiel:

```
rob_PTPmode sync
```

- **rob_inverse_class <class_id>**

  Definiert das Verhalten der Robotersteuerung bei der Berechnung des inversen Problems. Der Parameter <class_id> kann folgende Werte annehmen:

align_FRAME	Das TCP-KS wird bzgl. Position und Orientierung placiert
align_X	Position und X-Richtung (n)
align_Y	Position und Y-Richtung (s)
align_Z	Position und Z-Richtung (a)
align_POS	Nur die Position des TCPF wird placiert

  Beispiel:

```
rob_inverse_class align_Z
```

- **ARC [on | off]**

  bewirkt das Ein- bzw. Ausschalten des optischen "Funkeneffekts" am TCP des aktiven ROBOTERs. Das SCRIPT-Kommando wird üblicherweise aus einem IRDATA-Roboterprogramm heraus bei der Simulation von Punkt- oder Bahnschweißen aufgerufen.

D.2 Beispiel einer SCRIPT-Kommandodatei

```
RBv01r01
#
# SCRIPT_File 'trsys_demo.spt':
#
#   Laedt die hoeheren Detaillierungsstufen der Wechselflansche
#   und die IRDATA-Roboterprogramme fuer das Traegersystem und
#   den MANUTEC-R3 Industrieroboter. Die Ansicht wird so
#   umgeschaltet, dass die bearbeiteten Objekte vergroessert
#   und zentral dargestellt werden. Das IRDATA-Programm
#   fuer das Traegersystem zum Anfahren der Referenzposition
#   wird sofort gestartet.
#
SWAP_NODE HTL_UMGEBUNG.FL_Klein01;
SWAP_NODE HTL_UMGEBUNG.FL_Klein02;
ROBOT_LANGUAGE irdata;
TF_LOAD ird_trstart TRAEGERSYSTEM;
TF_LOAD ird_711 MANTEC_RM3;
LOAD_VIEWS trs_demo.vws;
TF_RUN ird_trstart TRAEGERSYSTEM;
FRAME_TO_WFRM Toolbox.Tool1 TO  Mantec_1.HAND Tool1_WFRM
```

D.2.1 Zusammenfassung der SCRIPT-Kommandos

ACTIVATE <frame_node>

DEACTIVATE <frame_node>

DELETE_NODE <frame_node>

SWAP_NODE <frame_node>

LEVEL_DOWN <frame_node>

LEVEL_UP <frame_node>

ROBOT_LANGUAGE <language>

ROBACT <Robot_id>

TF_LOAD <teach_filename> <Robot_id>

TF_RUN <teach_filename> <Robot_id> [exec_wait]

TF_DELETE <teach_filename> <Robot_id>

LOAD_VIEWS <view_filename>

EXEC_SCRIPT <script_file>

DRAW_SCENE

CONNECT <frame_2> TO <frame_1>

FRAME_TO_WFRM <frame_2> TO <frame_1> <WFRM_name1>

POS_ROB <Robot_id> <mode> <position_data>

SET_FRAME <frame_node> RENDER <draw_parameter>

SET_FRAME <frame_node> PLACEMENT <place_parameter>

SET_ABSTRACT <frame_node> RENDER <draw_parameter>

ROBOT <Robot_id> [online | offline]

ROBOT <Robot_id> [send_on | send_off]

SET_CAMPAR <CAM_id> <view_dist> <fovy_angle>

SET_USSPAR active <USS_id> [0 | 1]

SET_USSPAR sendmode [0 | 1]

SET_USSPAR range <USS_id> <USS_range>

TAG_DRAWLEN <len_value>

rob_cart_lin_vel <lin_vel>

rob_cart_lin_acc <lin_acc>

rob_cart_rot_vel <ang_vel>

rob_cart_rot_acc <lin_acc>

rob_cart_vel_scal <lin_acc>

rob_cart_acc_scal <lin_acc>

rob_jnt_max_range <njoints•[max_pos]>

rob_jnt_min_range <njoints•[min_pos]>

rob_jnt_start_pos <njoints•[start_pos]>

rob_jnt_ref_pos <njoints•[ref_pos]>

rob_jnt_max_vel <njoints•[max_vel]>

rob_jnt_min_vel <njoints•[min_vel]>

rob_jnt_act_vel <njoints•[act_vel]>

rob_jnt_max_acc <njoints•[max_acc]>

rob_jnt_act_acc <njoints•[act_acc]>

rob_PTPmode [sync|async]

rob_inverse_class <class_id>

ARC [on | off]