Konzeption und Realisierung eines integrierten Moduls zur Simulation und Steuerung von Kinematiksystemen

von der
Fakultät für Elektrotechnik und Informationstechnik
der
Universität Dortmund
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften

von
Dipl.-Ing. Andreas Keibel
Dortmund
2002

Tag der mündlichen Prüfung:	23. April 2003
Hauptreferent:	Prof. Dr.-Ing. E. Freund
Korreferent:	Prof. Dr.-Ing. R. Eckmiller

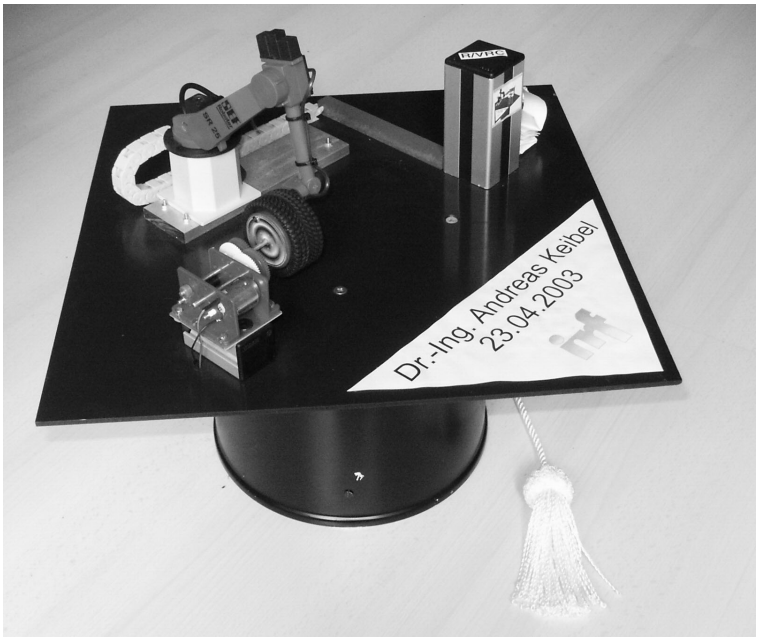
Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Roboterforschung der Universität Dortmund. Dem Institutsleiter, Herrn Prof. Dr.-Ing. E. Freund, gilt mein besonderer Dank für die stetige Förderung meiner Arbeit, die Schaffung hervorragender Arbeitsbedingungen und die Möglichkeit, meine wissenschaftlichen Ergebnisse in bedeutenden Forschungsprojekten praktisch umsetzen zu dürfen. Ferner Danke ich Herrn Prof. Dr. Eckmiller sehr herzlich für das der Arbeit entgegen gebrachte Interesse und für die freundliche Übernahme der Aufgabe des Zweitgutachters. Auf die gleiche Weise möchte ich mich hier auch bei Professor Kulig bedanken, der die Rolle des Drittgutachters auf sich genommen hat.

Weiterhin danke ich allen meinen nunmehr ehemaligen Kollegen am Institut für Roboterforschung, insbesondere den Abteilungsmitgliedern, für die vielen anregenden und kritischen Diskussionen, die zum Gelingen dieser Arbeit beigetragen haben. Bei der reaktionellen und inhaltlichen Durchsicht dieser Arbeit haben Herr Dr.-Ing. Jürgen Roßmann, Herr Dr.-Ing. Dierk Lothar Hahn, Herr Dipl.-Ing. Kai-Olaf-Rustige, Herr Dipl.-Inform. Bernd Lüdemann Ravit, Herr Dipl.-Ing Axel Meister, Dipl.-Ing. Marcel Brasch, mein Vater Dipl.-Ing. Joachim Keibel und in besonderem Maße Herr Dipl.-Ing. Alfred Hypki und Herr Dr.-Ing. Roger Bauer mitgewirkt, denen allen ich hiermit meinen ergebensten Dank aussprechen möchte.

Augsburg, im Mai 2003

Andreas Keibel, a.keibel@systragon.de



Allen ehemaligen Kollegen, die bei diesem ”Hut” mitgewirkt haben, möchte ich hiermit noch einmal meine besondere Freude und meinen besonderen Dank über dieses ausgezeichnet durchdachte, sehr liebevoll aufgebaute und funktionale Werk ausdrücken!

Inhaltsverzeichnis

Kapitel 1

Einleitung

Einer der ersten Roboterarme, der programmiert werden und einfache, spezielle Aufgaben verrichten konnte, wurde 1954 von George Devol entwickelt. Zusammen mit Joseph F. Engelberger gründete er später die Firma Unimation, die den ersten industriellen Roboterarm fertigte, der 1961 erstmals in der automobilen Fertigung bei General Motors in New Jersey eingesetzt wurde. Bis Ende der 70er-Jahre blieb die Robotik beinahe ausschließlich das Betätigungsfeld von Hochschulen und Universitäten, denn die Roboter der ersten Generation waren wegen ihres hohen Preises noch nicht rentabel genug für einen breiten industriellen Einsatz. Standardwerke der Robotik stammen aus dieser Zeit: Asada [AS86], Paul [Pau84]. Erst zu Beginn der 80er Jahre, nachdem die Systempreise deutlich gesenkt werden konnten, rentierte sich der Robotereinsatz in der automobilen Fertigung und Roboter wurden in Serie gefertigt. Einen guten Überblick über diese Epoche der Robotik gibt Scott [Sco84].

Simulation und Steuerung sind Begriffe der Robotik, die oft gemeinsam genannt werden, jedoch bisher nur selten gleichzeitig auf einer Plattform kombiniert wurden. Die 3D-Simulation einer Arbeitszelle läuft abgesetzt von der realen Robotersteuerung, die im Allgemeinen in einem autonomen Steuerschrank untergebracht ist. Zusätzlich basieren die relevanten Steuerungsteile der Robotersimulation und der realen Robotersteuerung nur selten auf einer gemeinsamen Quellcodebasis, so dass sich unerwünschte Differenzen zwischen Simulations- und realem Ablauf ergeben können. Roboter- und Arbeitszellensimulationen, die von Roboterherstellern angeboten werden, sind bisher nur zur Simulation der eigenen Systeme geeignet. Bereits seit 1991 versuchen roboterherstellerunabhängige und von der Automobilindustrie finanzierte Gremien [RRS94] Vereinheitlichungen bei der Steuerungsschnittstelle zu erzielen, mit der die Qualität der Simulation durch den Einsatz herstellerspezifischer VRC[1]-Module gesteigert werden soll. Dieser Vorgang gestaltet sich sehr aufwändig, da z.B. sämtliche interessierte und gleichzeitig konkurrierende Roboterhersteller zusammenarbeiten und sich auf gemeinsame Lösungen einigen müssen. Hat das Projekt Erfolg, so lässt sich die Robotersteuerungssoftware verschiedener Hersteller an ein Simulationssystem ankoppeln und letztendlich sogar mittels einer einheitlichen Datenbasis pro-

[1] VRC=**V**irtual **R**obot **C**ontroller sind Komponenten von Simulationssystemen, die innerhalb der Arbeitszellensimulation die Robotersteuerungssimulation übernehmen.

grammieren[2]. Allerdings dient diese Vereinheitlichung ausschließlich einer verbesserten Simulation. Die Lösung, bei der die Arbeitszellen-Simulationssoftware gleichzeitig auch als Steuerung eingesetzt wird, und sowohl mit als auch ohne angeschlossenen Roboter einsatzfähig ist, steht in diesem Umfeld z.Z. nicht zur Verfügung.

In dieser Arbeit wird der Weg der Entwicklung und Realisierung eines solchen kombinierten Steuerungs- und Simulationskerns beschrieben, mit dem sowohl Roboter, als auch andere kinematische Geräte simuliert und wahlweise gleichzeitig angesteuert werden können. Beim offline Einsatz dient die Software ausschließlich der Simulation, während beim Einsatz in der Anlage sowohl der 3D-Simulations- als auch der Steuerungsbetrieb zur Verfügung stehen. Die Umschaltung zwischen diesen Betriebsarten ist zu jeder Zeit und binnen Sekunden möglich. Durch die Möglichkeiten dieses einfachen Wechsels verringern sich sowohl die Programmentwicklungs- als auch die Einrichtungszeiten. Die Programmpflege wird erheblich vereinfacht. Ebenfalls wird die Systemsicherheit erhöht, da der Vorgang vor jedem Programmablauf mit dem realen Roboter dreidimensional dargestellt und überprüft werden kann, ohne den angeschlossenen Roboter mitzubewegen. Beim Steuerungsbetrieb stellt das System die Arbeitszelle online in der 3D-Zellenansicht dar. Da der Simulations- und Steuerungsbetrieb von derselben Steuerung ausgeführt wird, existiert auch keine Inkompatibilität zwischen Simulations- und Steuerungsmodus.

Im folgenden Kapitel wird ausführlich der Schwerpunkt dieser Arbeit, die Entwicklung eines universellen Systems zur Simulation und Steuerung von Kinematiken, dargestellt.

[2] Bisher existiert jedoch immer noch keine VRC mit einer derartigen Steuerungsschnittstelle.

Kapitel 2

Ziel der Arbeit

In dieser Arbeit wird das Ziel der Konzeption und Realisierung eines universellen, skalierbaren und flexiblen Robotersteuerungsmoduls verfolgt, das sowohl dazu geeignet ist, Roboter innerhalb eines Simulationssystems zu simulieren, als auch gleichzeitig reale Roboter anzusteuern. Dieses kombinierte System aus realer bzw. virtueller Robotersteuerung (R/VRC[1]), soll dazu in der Lage sein, verschiedenste Robotertypen und kinematische Geräte umfassend zu unterstützen, sowohl als Robotersteuerung für reale, als auch für rein virtuelle Roboter innerhalb eines Simulationssystems. Die Ankopplung an die umgebenden Systeme wie Simulationssystem und ggf. reale Kinematikhardware erfolgt dabei durch definierte Steuerungsschnittstellen. Implementierungstechnisch soll das Steuerungsmodul auf beliebigen Hardware-Plattformen lauffähig und einfach portierbar sein.

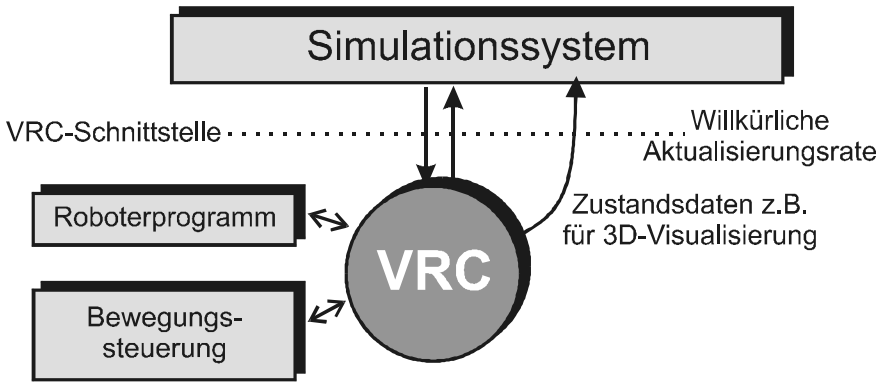


Abbildung 2.1: VRC und Simulationssystem

Bei der Konzeption der R/VRC liegt ein Schwerpunkt auf der akkuraten und umfassenden Robotersimulation real existierender Systeme unterschiedlicher Roboterhersteller, weshalb eine weitreichende Parametrierbarkeit und eine hohe Flexibilität erforderlich sind. Diese Skalierbarkeit erstreckt sich über sämtliche Komponenten des Systems, zu denen Programminterpreter, Bewegungssteuerung und Kinematikmodule gehören. Diese Komponenten sind Bestandteil des Kerns

[1] R/VRC=**R**eal/**V**irtual **R**obot **C**ontroller

der R/VRC und kommen automatisch sowohl dem reinen Simulations-, als auch dem Steuerungsbetrieb mit realem Roboter zugute.

Abbildung 2.1 stellt die Kombination aus Simulationssystem und einer VRC dar, die ausschließlich der nicht echtzeitfähigen Simulation von Robotersteuerungen dient. Die Achsstellungen werden von der Bewegungssteuerung der VRC erzeugt und an die Simulationsoberfläche übertragen, wobei der Datenaustausch mit der Oberfläche, in der auch die 3D-Ansicht der Arbeitszelle berechnet und gezeichnet wird, nicht in einem festen Zeittakt, sondern asynchron erfolgt.

Abbildung 2.2 stellt die Erweiterung der VRC um eine Robotersteuerung für reale Kinematiken dar, was eine besondere Herausforderung darstellt, da hierbei Randbedingungen eingehalten werden müssen. Diese Bedingungen sind in dieser Komposition nur schwer zu erreichen, denn die von der Bewegungssteuerung der R/VRC berechneten Achsstellungen werden nicht nur an das überlagerte Simulationssystem übertragen, sondern müssen auch *pünktlich* an die reale Roboterhardware gemeldet werden.

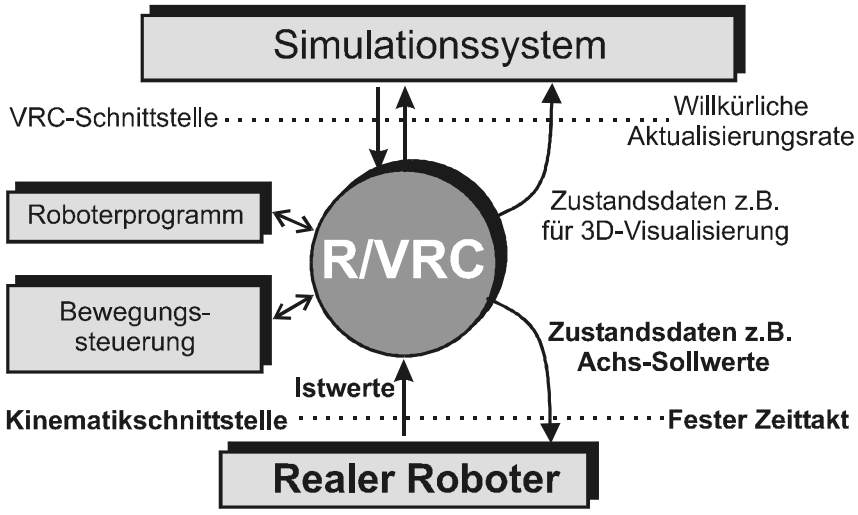


Abbildung 2.2: Verhältnisse mit angekoppelter Kinematik

Die Erzeugung dieses festen Zeittaktes hängt entscheidend von der Wahl der Komponenten des Zielsystems wie Rechnerhardware, Betriebssystem und der Programmiersprache ab. Gleichzeitig soll der aktuelle Zustand der Arbeitszelle in beliebigen 3D-Ansichten auf dem Bildschirm angezeigt werden. Parallel dazu verarbeitet der Roboterprogramminterpreter das Steuerungsprogramm und die Bewegungssteuerung interpoliert im Steuerungstakt die Bewegungen der angeschlossenen Hardware. Zu dem definierten Ziel dieser Arbeit gehört es, alle genannten Aufgaben auf einem einzigen Steuerungs-/Simulations-PC zu integrieren.

Die Basis dieser Arbeit stellt das vorhandene Simulationssystem COSIMIR[2] [Uth97] dar, das über eine umfassende Steuerungsschnittstelle [Hyp94] und ein bestehendes Robotersimulationsmodul zur Simulation einiger statisch definierter Robotertypen [FUHdV93], [dV95] verfügt.

Abbildung 2.3 stellt den Zusammenhang schematisch dar. Innerhalb der Simulation findet der Programmentwurf statt und bei Fertigstellung kann der reale, angeschlossene Roboter aktiviert

[2]COSIMIR: Cell Oriented SImulation of Industrial Robots

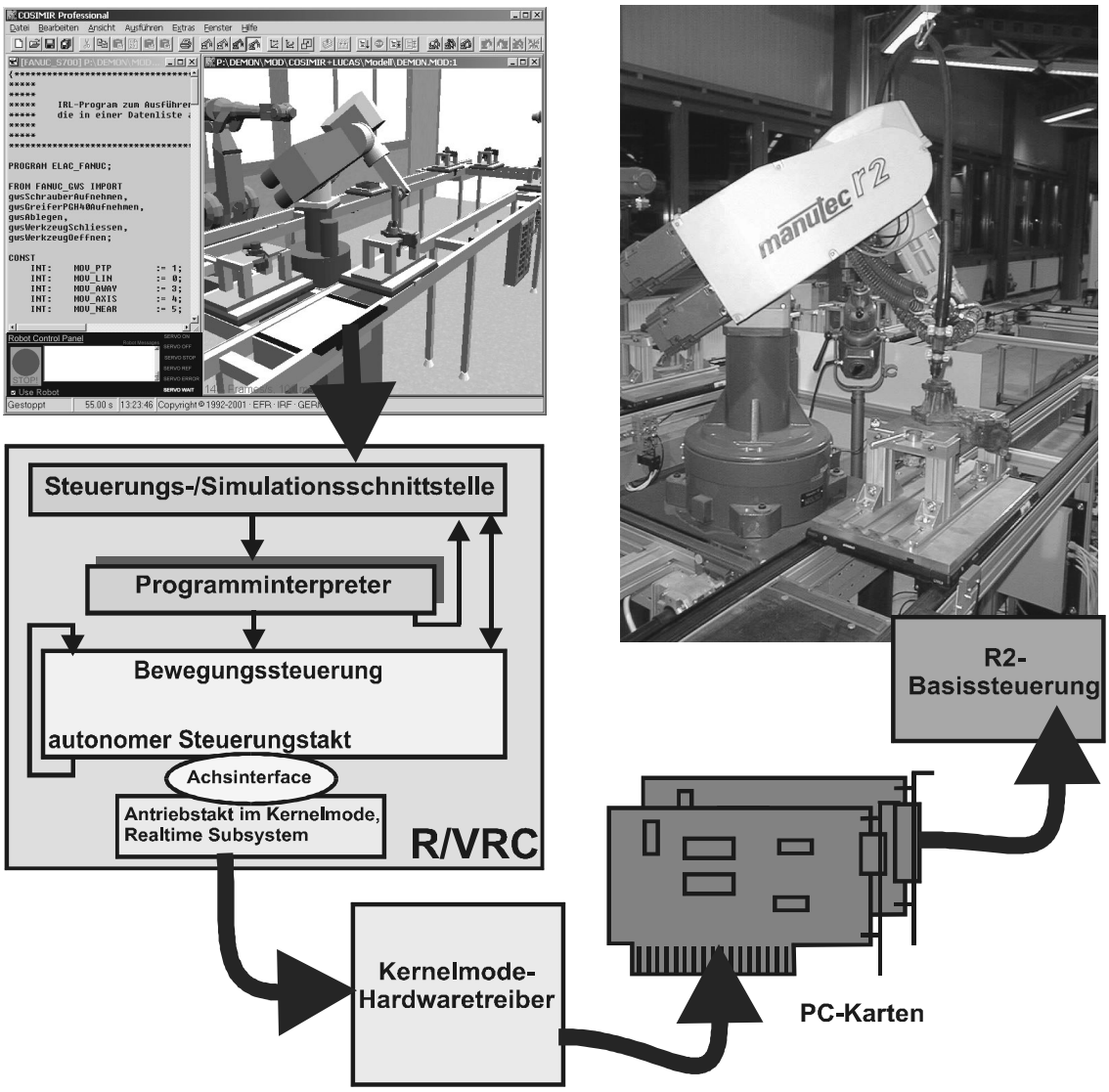


Abbildung 2.3: Systemhierarchie

werden. Bei dem Ausführen von Bewegungsanweisungen bewegt sich der simulierte Roboterarm auf dem Bildschirm synchron zur realen Bewegung. Diese Kombination aus Steuerung und Simulation bietet viele Vorteile, auf die in Kapitel 2.3 genauer eingegangen wird.

2.1 Universelle Robotersteuerungssimulation

Um das im vorangegangenen Kapitel definierte Ziel zu erreichen, wird zunächst eine umfassende Robotersimulation als VRC realisiert.

Mit der wachsenden Zahl an Robotern und Handhabungskinematiken, die in Forschung und Industrie eingesetzt werden, hat sich auch die Typenvielfalt vergrößert. Für diese neue Vielfalt an Kinematik- und Steuerungstypen wurden jedoch bisher keine einheitlichen Mensch-Maschine-

Schnittstellen entwickelt. Nahezu jeder Hersteller verwendet seine eigene Roboterprogrammiersprache und seine eigenen Verfahren bei der Bewegungssteuerung.

Das Ziel von Robotersimulationssystemen ist es, die Zellen-Planung, die Programmierung und die Fehlersuche beim Robotereinsatz erheblich zu vereinfachen. Aus diesen Anforderungen ergeben sich einige in dieser Arbeit relevante Qualitätskriterien für die Robotersimulation und damit für die vorliegende VRC:

- Anzahl unterstützter Roboterkinematiken:
  Je mehr Robotertypen innerhalb der Robotersimulation zur Verfügung stehen, desto mehr ist das System zur objektiven Layout-Planung geeignet, da herstellerunabhängig die am besten geeignete Kinematik ausgewählt werden kann.
- Anzahl unterstützter Roboterprogrammiersprachen:
  Besteht die Möglichkeit, die unterstützten Kinematiken in ihrer nativen Roboterprogrammiersprache zu programmieren, so steht ein umfassendes System zur Verfügung, mit dem Anlagen objektiv geplant und gestaltet werden können.
- Anpassbarkeit des Modells zur Minimierung von Differenzen zwischen Realität und Simulation:
  - Räumliche Differenzen beruhen auf ungenauen geometrischen und dynamischen Modellen der realen Umgebung.
  - Elektronische Abweichungen basieren auf fehlerhaft bzw. nicht exakt modellierten Regelungsverfahren.
  - Algorithmische Differenzen ergeben sich aus der Unterschiedlichkeit der Programmabarbeitung zwischen realer und simulierter Robotersteuerung.

Im Idealfall verhält sich der simulierte Roboter in der simulierten Umgebung exakt wie das reale Pendant. Dieser Fall ist jedoch in der Praxis nicht zu erreichen. Lediglich eine mit den zur Verfügung stehenden Mitteln bestmögliche Annäherung an die Realität kann das Ziel sein. Selbst bei Verwendung der originalen Robotersteuerung des Herstellers bleibt die Ungenauigkeit des mathematischen Modells des Roboters als großes Problem bestehen. Die mathematische Modellierung des Verhaltens kinematischer Ketten unter dem Einfluss von Störgrößen ist Bestandteil zahlreicher Forschungen, deren Ergebnisse sich zwar stetig verbessern und vielfach zu ausreichend genauen Resultaten führten, dabei jedoch extreme Mengen an Parametern erfordern, die nur selten oder mit sehr großem Aufwand ermittelt werden können und stets speziell ausfallen [Ber95]. Erschwerend kommt hinzu, dass sich entscheidende Parameter des mathematischen Modells während des Betriebes und mit dem Alter des Systems verändern (Öl-Viskositäten, Getriebeverschleiss, Änderungen nach Servicearbeiten...). Eine allgemeine Simulation zur Abbildung des Verhaltens beliebiger kinematischer Ketten unter Berücksichtigung von Reibungen, Verbiegungen, Reglerqualitäten, statischen und dynamischen Kräften und Abhängigkeiten von gegriffenen Lasten existiert bis heute nicht. Sie kann auch nicht Bestandteil dieser Arbeit sein [PG94].

Der Ansatz dieser Arbeit sieht den Einsatz einer universellen VRC innerhalb eines Simulationssystems vor, mit der ein breites Spektrum von Kinematiksteuerungen simuliert werden kann.

Kommen mehrere Kinematiken zum Einsatz, werden weitere Instanzen[3] dieser Steuerungssimulation zur Laufzeit geladen, passend initialisiert und parametriert. Eine solche Steuerung muss so universell und parametrier- und skalierbar sein, dass eine Simulation verschiedener Roboter und Steuerungstypen mit nur dieser einen Steuerung möglich wird.

Dieser Weg ist gegenüber dem Einsatz von originalem, statischem Steuerungscode vorteilhafter, denn er gestattet zusätzlich auch den wichtigen Einsatz des Simulationssystems als Anlagenplanungs- und Layoutsystem, bei dem möglichst alle überhaupt in Frage kommenden Roboter auch in der Simulation funktional zur Verfügung stehen sollen, obwohl kein Steuerungscode der Hersteller zur Verfügung steht. Ein weiterer, nicht zu unterschätzender Vorteil beruht auf der ständigen Verfügbarkeit des "Know-Hows"[4], das bei der Entwicklung von Robotersimulations- und Steuerungssystemen und deren stetig steigender Flexibilität mitwächst, bei einem gekauften Steuerungsmodul jedoch verschlossen bleibt. Erweiterungen des einen universellen Steuerungsmoduls kommen automatisch sämtlichen unterstützten Robotersystemen zugute.

Sind die Schnittstellen zwischen Simulationsoberfläche und Steuerungen genügend flexibel ausgelegt, so ist ohne weiteres auch eine Koexistenz zwischen herstellerspezifischen Modulen (mit dem originalen Steuerungscode) und der universellen Steuerungssimulation möglich. Dabei werden Roboter, für die kein derartiges Modul zur Verfügung steht, mit dieser Steuerung simuliert.

Auf die Schwierigkeiten und Problematiken sowie die Realisierung dieser allgemeinen Robotersteuerungssimulation wird in dieser Arbeit ausführlich eingegangen.

2.2 Steuerungsunabhängige Simulation und Programmierung

Robotersimulationssysteme werden heute als Offline-Programmier- und Anlagen-Layoutsysteme eingesetzt, mit denen -abgesetzt von Fertigungshallen- Anlagenlayouts und Steuerungsprogramme offline erzeugt und getestet werden können. Hierbei treten bei Einsatz gängiger Programmier- und Simulationswerkzeuge immer wieder die gleichen Probleme und Fakten hervor:

1. Die Differenzen zwischen offline und online erzeugten Daten erfordern im Allgemeinen Korrekturen am Roboterprogramm und an den Positionen.

2. Die Offline-Programmierung ist beliebter, als der aufwändige Umgang mit den Handbediengeräten der Steuerungen, da mit Bildschirm und Maus in vertrauter, komfortabler Umgebung gearbeitet werden kann und verschiedene Robotersysteme einheitlich bedient werden können.

3. Die offline erzeugten Programme müssen in kompatible Steuerungsprogramme übersetzt werden, falls die Roboterprogrammiersprache des Simulationssystems eine andere als die der Roboter in der Anlage ist. Selbst das Format, in dem Positionen abgelegt werden, unterscheidet sich teilweise so stark, dass eine Umrechnung nur sehr aufwändig zu realisieren ist.

[3] Als weitere Instanz wird hier die Wiederverwendung des Programmcodes eines Moduls bezeichnet. Nur die Daten eines Modules müssen pro Instanz neu angefordert und initialisiert werden. Der Programmcode ist stets derselbe und liegt nur einmal im Arbeitsspeicher des Rechners.

[4] Als "Know-How" wird hier insbesondere das im Quellcode implementierte und dokumentierte Wissen, das dadurch auch Entwicklern anderer Anwendungen zur Verfügung steht, bezeichnet.

4. Die datentechnische, netzwerktechnische und physikalische Übertragung in die reale Robotersteuerung gestaltet sich u.U. aufwändig.

5. Nach erfolgtem Programmdownload sind Positionskorrekturen und Kalibiermaßnahmen aufgrund mangelnder Übereinstimmung zwischen Modell und der Realität erforderlich.

6. Der Umgang mit den realen Robotersteuerungen ist oft unkomfortabel und nur mit den zugehörigen PHGs[5] möglich.

Diese Punkte legen nahe, als Steuerung das gleiche System einzusetzen, wie innerhalb der Simulation, so dass sich ein integriertes Steuerungs- und Simulationssystem ergibt. Dabei ist die Wahl des Rechner- und Betriebssystems ein wesentlicher Konzeptionspunkt. Windows 9x und Windows NT sind nicht dafür entwickelt worden, Anlagen zu steuern und verfügen nicht über die erforderliche Echtzeitfähigkeit. In dieser Arbeit wird ein Verfahren vorgestellt, mit dem dieses Problem für die Robotersteuerung gelöst wird.

2.3 Vorteile einer integrierten Simulation und Steuerung

Mit der Integration von Simulation und Steuerung steht zum einen eine Robotersteuerung zur Verfügung, die den Komfort aller Funktionen und Werkzeuge eines Simulationssystems bietet, und dem Anwender gleichzeitig die Möglichkeit bietet, am Bildschirm zwischen Simulations- oder Steuerungsbetrieb zu wählen. Die Merkmale, angelehnt an die obigen Punkte 1.-6. der ”Probleme und Fakten” in Kapitel 2.2 sehen bei einem integrierten System deutlich vorteilhafter aus:

1. Differenzen zwischen on- und offline erzeugten Daten werden eher erkannt, weil ein und dasselbe System sowohl zur Steuerung als auch zur Simulation verwendet wird, und vor Ort der reale mit dem online dargestellten 3D-Simulationslauf verglichen werden kann. Unterschiede treten dadurch schnell zu Tage und können durch Anpassung des Modells minimiert werden.

2. Die Online-Programmierung ist genauso komfortabel wie die Offline-Programmierung und findet in gewohnter Windows-Umgebung oder einem einheitlichen, an den PC angeschlossenen PHG statt.

3. Die offline erzeugten Programme müssen nicht in kompatible Steuerungsprogramme übersetzt werden, da die VRC verschiedene Roboterprogrammiersprachen direkt unterstützt. Die Programme sind zu den realen Robotersteuerungen kompatibel. Dadurch, das das Simulationssystem verschiedene Sprachen unterstützt, hat der Anwender auch beim kombinierten System die Möglichkeit der Wahl verschiedener Sprachen.

4. Der Programmdownload in die Robotersteuerungen ist nicht erforderlich, da der Anwender bereits mit der Steuerung arbeitet. Findet die Programmentwicklung offline statt, so sind beim Transfer der Daten keinerlei Konvertierungen erforderlich, da sich die Programminterpreter der beiden Steuerungsvarianten identisch verhalten. Der Datenaustausch kann bequem über das LAN[6] abgewickelt werden.

[5] PHG: Programmier-Hand-Gerät
[6] LAN: Local Area Network.

5. Kalibriermaßnahmen gestalten sich mit der integrierten Version erheblich einfacher, da der reale Roboter (in Verbindung mit passender Sensorik) jederzeit als Messmaschine zur Verfügung steht und Positionen, wie etwa Greifpunkte, einfach in das Modell übernommen werden können. Auf diese Weise lassen sich Objekte innerhalb des Arbeitszellenmodells in ihrer Lage an die reale Umgebung anpassen. Roboter sind aufgrund ihrer Ungenauigkeit ([Spu96], [Dyn99]) nicht im gesamten Arbeitsraum zur Vermessung geeignet, in vielen Fällen kann jedoch die Lage der realen Arbeitspunkte mit denen innerhalb des geometrischen Modells dadurch in Übereinstimmung gebracht werden, dass die Objekte innerhalb lokaler Zonen etwas verrückt werden[7]. Dieses Prinzip kommt einer Linearisierung der Ungenauigkeiten um den betreffenden Arbeitspunkt nahe und hat sich als sehr wirkungsvoll herausgestellt.

6. Der Umgang mit der Robotersteuerung und dem Simulationssystem ist identisch.

Komponenten, die bisher nur innerhalb der Simulation zur Verfügung standen, sind auch in dem Betriebsmodus als reale Robotersteuerung verfügbar, und stellen damit sofort ein umfangreiches Funktionsangebot zur Verfügung:

1. Sämtliche Roboterkinematiken, die im Simulationsmodus funktional zur Verfügung stehen, werden softwaretechnisch automatisch auch im Steuerungsbetrieb unterstützt. Roboterspezifisch ist daher lediglich die hardwareseitige Ankopplung der Antriebssysteme, die in Kapitel 7.3 behandelt wird.

2. Durch die vorhandene Netzwerkintegration des Simulationssystems profitiert automatisch auch die Robotersteuerung, so dass diese z.B. mit Zellensteuerungen kommunizieren kann.

3. Das Gesamtsystem aus Robotersteuerung und Simulationsoberfläche hat Zugriff auf die Daten des Arbeitszellenmodells und ”kennt” daher seine Umgebung. Hiermit lassen sich intelligente Verfahren zur Programmerstellung oder Kollisionsvermeidungsverfahren, wie in [Ros93] beschrieben, realisieren. Durch die Präsenz von erwarteten Größen innerhalb der Simulation, die durch das geometrische Modell bestimmbar sind (Abstände, Kamerabilder, Farben, Abmessungen...), in Verbindung mit intelligenten Sensoren der Realität, die ihre Umgebung erfassen können, steht mit dem integrierten System eine Plattform zur Verfügung, die viele Möglichkeiten für weitere Forschungs- und Entwicklungsaufgaben bietet. So ist z.B. eine automatische Kalibrierung denkbar, die das geometrische Modell der Umgebung und des Roboters mit der Realität abgleicht, indem reale mit simulierten Messwerten verglichen werden. Für eine Qualitätssicherung können erwartete Bilder aus der VR erzeugt und mit den realen korreliert werden. Greifpunkte könnten online ermittelt werden, und müssten nicht mehr statisch im Roboterprogramm enthalten sein, sondern können innerhalb des Roboterprogramms symbolisch, etwa mit ”Greifpunkt.Deckel”, losgelöst von statischen Positionsangaben, angesprochen werden.

4. Stand der Technik

[7] Bei diesem Verfahren werden auch lokale Ungenauigkeiten des Roboters in dem geometrischen Modell berücksichtigt.

Kombinierte Systeme aus Steuerung und Simulation, sind in der Automatisierungstechnik bisher selten und vor allem nicht bei Robotersteuerungen anzutreffen. Die meisten Simulationen, ganz gleich, ob es sich um SPS-, Roboter-, Pneumatik- oder andere Simulationen handelt, erfolgen offline und nicht auf dem Steuerungsrechner in Echtzeit. In der Robotik, insbesondere der Telerobotik, hat sich bisher die parallele Visualisierung durch mitlaufende Simulationssysteme etabliert. Auch das Simulationssystem COSIMIR ist durch dessen zahlreiche und flexible Schnittstellen als Online-Visualisierungssystem einsetzbar [Hah99], [FR99], [FKR]. Flexible Robotersteuerungen als integraler Bestandteil von 3D-Simulationssystemen sind in der hier vorgestellten Form z.Z. nicht verfügbar, so dass in diesem Kapitel lediglich auf den Stand der Technik der Einzelkomponenten, der Simulation und der Steuerung, eingegangen wird.

2.4 Stand der Technik bei Robotersteuerungssystemen

Seit den 80er Jahren hat sich die Robotik permanent und mit hoher Geschwindigkeit weiterentwickelt. Industrielle Robotersysteme verfügen heute über die Möglichkeiten sensorgeführter Bewegungen, leistungsfähige, modulare und prozedurale Hochsprachen zur Programmierung, eine Vielzahl an Standard-Schnittstellen und unterstützen die Einbindung in bestehende Netze. Auch prädiktive Regelungsverfahren, die dynamische Größen wie Gewichts-, Zentrifugal- und Corioliskräfte berücksichtigen [Fre82], finden bereits Verwendung bei einigen industriellen Robotersystemen, wie z.B. Systemen von ABB Flexible Automation [ABB98]. Verschiedene Systeme berücksichtigen sogar die Lastmasse am Roboterflansch für eine bessere Prädiktion von Antriebskräften und -Drehmomenten. Auch findet sich mittlerweile bei vielen Robotern die Unterstützung von Sperrräumen, mit denen Zonen definiert werden können, in die der Roboterarm nicht hineinfahren darf. Hiermit können auf einfache Art und Weise Kollisionen mit der statischen Umgebung des Roboters ausgeschlossen werden. Bei Fanuc-Robotern beispielsweise kann zusätzlich die Größe einer Last definiert werden, die dadurch ebenfalls nicht in diese Sperrräume fährt.

Diese Arbeit knüpft an die Dissertation von Uwe van der Valk [dV95] an, der ein Robotersteuerungssimulationsmodul für das Simulationssystem COSIMIR entwickelt hat, mit dem sieben verschiedene Robotertypen unterstützt wurden[8]. Die Flexibilität der Kinematikmodule beschränkte sich auf die Variation der Gelenklängen. Konfigurationsräume, Achs-Rotationszähler (Turnangaben) und verschiedene Handtypen wurden nicht unterstützt, wodurch nur ein deutlich eingeschränkter Teil der industriell relevanten Systeme abgedeckt werden konnte. Der IRDATA-Interpreter umfasste knapp den Umfang der Norm [fNe94b] (siehe Kapitel 5.3). Das Modul war ausschließlich für den Simulationsbetrieb ausgelegt. Im Rahmen dieser Arbeit werden hier Konzepte erarbeitet, dieses Modul mit einer Flexibilität auszustatten, so dass auch moderne Systeme industriekonform mit ihrer großen Variantenvielfalt an Kinematiken und Programmiersprachen unterstützt werden.

Steuerungen bestimmter Hersteller sind bereits PC-basiert und verfügen über Windows-basierte Oberflächen, z.B. die Steuerung KRC 1 der Fa. Kuka [KUK00] und die Steuerung rho4.1 der

[8] Unterstützt wurden im einzelnen: Kinematiken des Typs Kuka 364, Niko N900, Niko N601, VW G8, Manutec, Adept 500, Focke und ähnliche.

Fa. Bosch. Eine integrierte Simulation und 3D-Darstellung der Arbeitszelle ist jedoch nicht innerhalb der Steuerung vorhanden. Als Roboterprogrammiersprache hat sich bisher kein einheitlicher Standard durchgesetzt[9]. Jeder Roboterhersteller verfolgt hier nach wie vor eigene Konzepte. Funktional lässt sich eine Basis-Schnittmenge bilden, die mittlerweile von den meisten Herstellern unterstützt wird. Aufbauend auf diesen Basisfunktionalitäten werden spezielle Lösungen heute nicht nur von Universitäten und Fachhochschulen, sondern auch von Roboter-Systemhäusern erarbeitet, die je nach Anwendungsfall auch komplexe Forschungsaufgaben übernehmen. Oft werden diese Teilbereiche an sehr spezielle Aufgaben gekoppelt, so dass sich thematisch kein allgemeiner Schluss für die Robotik ziehen lässt. Leider ist es sogar so, dass Forschungsergebnisse häufig sehr lange brauchen, bis diese in industriellen Anwendungen eingesetzt werden, oder gar durch allgemeine Verfügbarkeit zum Stand der industriellen Technik werden. An dieser Stelle kann keine Auflistung einzelner Forschungsergebnisse erfolgen - allein die Liste der interessanten Veröffentlichungen würde den Rahmen dieser Arbeit sprengen. Einen guten Überblick über die Leistungsfähigkeit moderner Robotersteuerungen geben z.B. die Handbücher zu den Robotersystemen von Kuka, Fanuc, Adept und ABB, sowie die Schulungsunterlagen der Ausbildungszentren der Hersteller und Systemhäuser. Eine gute Übersicht über die weltweiten universitären und industriellen Forschungen geben die Forschungsbände zu Themen der Robotik [VDI00]. Einen weiteren Überblick geben die Anforderungen der Automobilindustrie an die Roboterhersteller. Diese Anforderungen spiegeln sich z.B. in [Cor00] wieder, in dem General Motors die ”technischen Spezifikationen für Produktionsroboter” beschreibt.

Zusätzlich zu der technologischen Leistungsfähigkeit einzelner Steuerungen spielt jedoch auch deren Bedienungskomfort, Programmierbarkeit, Einbindbarkeit in bestehende Strukturen, Flexibilität und Verfügbarkeit in Simulationssystemen eine große Rolle. So ist es heute immer noch der Fall, dass die Steuerungen untereinander nicht kompatibel sind. Das bedeutet: Roboterprogramme eines Herstellers laufen nicht auf Steuerungen eines Zweiten. Selbst das Datenformat zur Positionsdarstellung kann nicht herstellerübergreifend ausgetauscht werden. Die Möglichkeit, die Steuerung eines Herstellers an den Roboterarm eines anderen Herstellers anzukoppeln, oder gar Eigenentwicklungen an die Steuerung anzuschließen, ist nur bei einzelnen industriellen Systemen vorgesehen.

2.5 Stand der Technik bei Robotersimulationssystemen

Zusätzlich zu den Programmiermöglichkeiten vor Ort, die meist mittels PHG zu erfolgen haben, werden von Roboteranbietern optional Simulationssysteme angeboten, mit denen die eigenen Roboter und ggf. auch andere Automatisierungskomponenten simuliert und offline programmiert werden können. Der Vorteil dieser Systeme liegt darin, dass sie als Robotersteuerung eine Kopie der realen Robotersteuerung als virtuellen Steuerungskern enthalten. Diese virtuellen Robotersteuerungen (VRC) verhalten sich erwartungsgemäß innerhalb der Simulation sehr ähnlich zum realen Pendant. Selbst Fehler der realen Steuerung treten wegen des identischen Codes auch in der Simulation auf.

Einige industrielle Simulationssysteme unterstützen bereits VRCs. Beispiele hierfür sind ”ABB:

[9] Die genormte Roboterprogrammiersprache IRL [fNe94a] kommt bei keinem namhaften Roboterhersteller zum Einsatz.

RobotStudio-S4 Lite" in Verbindung mit ABBs "Virtual Controller Technologie", Kuka, mit der Technologie von Anysim, das in die Produktpalette von Tecnomatix aufgenommen wurde [Tec95] und Adept mit deren Simulationssystem "Digital Workcell". Hierbei fällt auf, dass diese Systeme ausschließlich von den Roboterherstellern selbst stammen und aus marktwirtschaftlichen Gründen nur die herstellereigenen Produkte unterstützen. Das Einbringen von Produkten unterschiedlicher Hersteller wird nicht unterstützt. Auch der Datenaustausch zwischen verschiedenen Simulationssystemen wird nicht vereinfacht. So ist ein Fanuc-Programm nicht auf einem ABB-Roboter lauffähig, und selbst das Umwandeln und Übertragen konsistenter Daten wie etwa Positionslisten[10] wird in diesen Systemen nicht unterstützt. Diese Inkompatibilitäten behindern den gemischten Einsatz verschiedener Systeme.

Gegenüber den Entwicklungen der Roboterhersteller existieren auch herstellerübergreifende Simulationssysteme, die sich in der Automobilbranche bereits seit Jahren etabliert haben. Diese sind z.B. "eM-Workplace" [Tec95], das sich mittlerweile in die eM-Power Produktfamilie der Fa. Tecnomatix eingliedert und Igrip [Rob92b] der Fa. Delmia, ehemals Deneb. Aus den anfänglichen Systemen zur einfachen Robotersimulation sind umfangreiche Pakete zur vollständigen Offline-Planung, -Organisation und -Programmierung geworden. Module zur interaktiven Kostenkalkulation, zur Organisation des Handels, Verteilung der Fertigungsschritte auf verschiedene Standorte und Ergonomie-Studien mit Mensch-Modellen sind heute Bestandteile der umfassenden Planungswerkzeuge geworden, so dass die Robotersimulation innerhalb dieser Produktgruppen nur einen kleinen Teil ausmacht.

Die Komponenten zur Robotersimulation innerhalb dieser Systeme unterstützen verschiedenste Robotertypen und eignen sich daher im Gegensatz zu den Systemen der Roboterhersteller auch für eine roboterunabhängige (objektive) Layout-Planung. Die Programmierung der Roboter erfolgt bei diesen Systemen in simulationssystemeigenen Programmiersprachen TDL und GSL (siehe hierzu [Rob92a], [DH91]). Diese Programmiersprachen dienen der vollständigen Programmierung des Systems. Mit ihr können z.B. Kinematiken definiert, Betrachterpunkte beeinflusst und Roboterbewegungen vorgegeben werden. Zur Generierung von Roboterprogrammen werden die roboter-relevanten Teile extrahiert und mittels spezieller Compiler in die Zielsprache übersetzt. Die Sprachelemente, die der Programmierung von Robotern dienen, sind dabei allgemein gehalten und dienen der Programmierung sämtlicher Roboter, so dass Anwender des Systems daher zusätzlich zur Programmiersprache des Roboters in der Fertigung auch die Sprache des Simulationssystems erlernen müssen, um effizient arbeiten zu können. Ferner kann innerhalb der Simulationssprache keine Rücksicht auf besondere Eigenschaften bestimmter Robotersteuerungen genommen werden, da die Funktionalität häufig nur aus der Schnittmenge der Funktionalitäten der unterstützen Robotersysteme besteht, nicht aus der Vereinigungsmenge. Der gravierendste Nachteil dieser allgemeinen, für sämtliche Systeme einheitlichen Sprache, besteht jedoch darin, dass keine Roboterprogramme von realen Robotersteuerungen in die Simulationsumgebung importiert und dort verwendet werden können, da eine Übersetzung der herstellerspezifischen Sprachen in die Simulationssprache nur bedingt möglich ist und nur selten automatisch erfolgen kann.

[10] Positions- bzw. Datenlisten werden zusätzlich zu den Roboterprogrammen im nicht flüchtigen Speicher der Robotersteuerungen abgelegt. Roboterprogramme können zur Laufzeit darauf zugreifen. Häufig werden in diesen Listen die räumlichen Koordinaten von Bewegungszielen abgelegt.

2.6 Verwendung des Steuerungscodes des Roboterherstellers

Der Weg, den einige Roboterhersteller z.Z. beschreiten, indem sie den Steuerungscode ihrer realen Robotersteuerungen in ihren Simulationssystemen einsetzen, führt zu einer guten Übereinstimmung zwischen Roboter und Robotersimulation.

In den Jahren 1992 und 1993 entwickelten branchenführende Automobil-, Roboter- und Simulatorhersteller gemeinsam die RRS-Schnittstelle (Realistische Roboter Simulation) [RRS94]. Die Schnittstelle ermöglicht es, die originale Bewegungssoftware von Robotersteuerungen in Simulationssysteme zu integrieren. Nach Verabschiedung dieses Standards wurden weitere Ziele in Angriff genommen, die in dem Projekt RRS-II [BSW01] bearbeitet werden.

Im Rahmen von RRS-II wird eine Schnittstelle definiert, die es ermöglicht, Robotersteuerungen vollständig in Simulatoren zu integrieren. Steuerungsmodelle, die anhand der neuen Virtual Robot Controller (VRC)-Interface-Spezifikation erstellt wurden, sind in der Lage, Anwenderprogramme aus der Fertigung ohne Modifikationen auszuführen. Umgekehrt können Anwenderprogramme in Virtual Robot Controllers weitgehend ausgetestet werden, bevor sie an die Produktion übergeben werden. Das Projekt wurde im April 1998 unter Beteiligung von 35 Firmen und mit dem Fraunhofer IPK als Projektmanager begonnen. In der Anfangsphase wurde eine Vorversion der VRC-Interface-Spezifikation erarbeitet. In der zweiten Phase wird diese Spezifikation durch prototypische Implementierungen verifiziert und anhand der gewonnenen Erfahrung optimiert. Um eine nahtlose Integration in Simulationssysteme zu gewährleisten, wurde das Projekt um folgende Aspekte erweitert:

- automatische Programmgenerierung,
- automatisierte Datenkonsistenz und
- Anwenderprogrammeditoren.

Die Hauptmotivation von RRS-II besteht für die Roboterhersteller darin, dass die Automobilindustrie als Hauptabnehmer von Robotern auf einheitliche Lösungen drängt, mit denen eine vollständige Anlagen-/Layoutplanung bis hin zur Roboterprogrammierung mittels Simulation möglich sein soll, so dass komplette Fertigungsanlagen und Fabriken in der virtuellen Realität funktional zur Verfügung stehen. Dabei treten wirtschaftliche Analysen in den Vordergrund [SMG98]. Die Robotersimulation soll dabei so realitätsnah wie möglich sein, wobei sich die Verwendung des Steuerungscodes der realen Steuerung als sehr aussichtsreich erwiesen hat.

Die Roboterhersteller, vor allem jene, die große Stückzahlen an die Automobilindustrie absetzen können, sind daher bereit, in die Entwicklung von RRS-II zu investieren. Grundsätzlich bedeutet Entwicklung eines RRS-II-Moduls jedoch einen erheblichen Aufwand, den sich nicht jeder Roboterhersteller leisten kann. Demnach werden mit dieser Methode auch in Zukunft nicht sämtliche am Markt verfügbaren Robotersysteme als RRS-II-Modul zur Verfügung stehen. Durch diese eingeschränkte Anzahl an RRS-II-Modulen zur Robotersteuerungssimulation, geht der Anspruch einer objektiven Layoutplanung verloren, weil Roboter, für die keine RRS-II-Module zur Verfügung stehen, nicht unterstützt werden, vielleicht aber besser geeignet wären als jene, die dem Planer am Simulationssystem bereitstehen. Sämtliche Systeme, unabhängig davon, ob für sie

RRS-II-Module verfügbar sind oder nicht, können von einer universellen Robotersteuerungssimulation, wie sie auch in dieser Arbeit behandelt wird, simuliert werden.

Ferner ist auch selbst bei Einsatz von RRS-II noch nicht sicher gewährleistet, dass sich die Simulation *exakt* wie das reale Pendant verhält [Ber95], [PG94], da innerhalb der Simulation auf dynamische und parasitäre Effekte wie Elastizitäten, Fertigungstoleranzen, Lasten und andere Positionierungs-Ungenauigkeiten, wenn, dann nur eingeschränkt Rücksicht genommen werden kann. Lediglich der Programmablauf wird dem der realen Steuerung sehr gut entsprechen können.

Simulationssysteme, mit denen herstellerunabhängige Zellen-Layout-Studien durchgeführt werden sollen, profitieren von der Anzahl der unterstützten Robotersysteme, da bei Planungsbeginn u.U. noch gar nicht feststeht, welche Systeme zum Einsatz kommen sollen. Das Simulationssystem soll den Entscheidungsprozess der Roboterauswahl *objektiv* begünstigen. Diese Objektivität ist mit einer generischen Robotersteuerung, wie sie in dieser Arbeit beschrieben wird, einfacher zu erreichen, indem pauschal eine große Anzahl an Robotern unterstützt wird und ggf. schnell an neue Kinematiken angepasst werden kann. Eine einheitliche Bedienoberfläche ermöglicht zusätzlich den schnellen Einstieg und führt daher oft deutlich schneller zum Erfolg, als etwa das Einarbeiten in die jeweilige Programmierumgebung des Roboterherstellers, der in seinem RRS-II-Modul die eigene Oberfläche[11] exakt nachbildet.

[11] Hierbei wird sogar so weit gegangen, dass selbst PHGs mit umständlicher Bedienung (mit Mehrfach-Tastenbelegungen) am Bildschirm originalgetreu auf dem Bildschirm simuliert werden.

Kapitel 3

Anforderungen an eine R/VRC unter einem grafischen Simulationssystem

Eine Robotersteuerungssimulation (VRC) stellt innerhalb eines Simulationssystems eine virtuelle Robotersteuerung zur Verfügung, mit der das Verhalten einer realen Steuerung möglichst exakt nachvollzogen werden soll. In diesem Kapitel werden die relevanten problembehafteten Schwerpunkte herausgegriffen, denen bei der Entwicklung einer flexiblen VRC besondere Aufmerksamkeit gewidmet werden muss. Beim Betrieb als **R**/VRC sollen sämtliche Eigenschaften und Funktionalitäten der VRC zur Verfügung stehen, ergänzt um die Eigenschaften einer Steuerung für reale Roboter.

3.1 Lösungen durch Simulationen

Bevor auf die Problematiken und Anforderungen von VRCs eingegangen wird, soll zunächst der Sinn und Zweck von Simulationen näher erläutert werden.

Prinzipielles Ziel des Einsatzes von Simulationssystemen ist es, Erkenntnisse über reale Vorgänge zu gewinnen und Methoden zu ihrer Beeinflussung zu finden, ohne dass diese Vorgänge wirklich real stattfinden müssen. Der Verzicht auf die Durchführungen des relevanten Vorganges in der Realität kann verschiedene Gründe haben:

1. Die Durchführung in der Realität verursacht Kosten, die mit Simulationsvorgängen minimiert werden können. Z.B. Online-Roboterprogrammierung, für deren praktische Durchführung eine Produktionslinie angehalten werden muss.
2. Die Durchführung in der Realität ist gefährlich. Beispiele sind biologische, physikalische, chemische oder atomare Experimente.
3. Die Durchführung in der Realität ist nicht machbar. Z.B. Probeaufbauten industrieller (Groß-)Anlagen, geologische Forschungen.
4. Die Durchführung in der Realität liefert keine qualitativen oder quantitativen Ergebnisse, etwa weil die interessierenden Größen gar nicht, nur ungenau oder schwierig mess- und beobachtbar sind.

5. Die Durchführung in der Simulation führt schneller zu Ergebnissen als der reale Ablauf.

Alle Simulationen setzen ein Modell der Umgebung voraus, innerhalb dessen die Vorgänge simuliert werden. Je genauer dieses Modell mit der Realität übereinstimmt, desto genauer fällt auch das Simulationsergebnis aus. Die Genauigkeit ist daher direkt korreliert mit der Exaktheit und dem Umfang des Modells.

Übertragen auf Robotersimulation kann die Simulation von Bewegungsabläufen einzelner Kinematiken und deren Einflussnahme auf ihre Umgebung aus allen oben angeführten Gründen sinnvoll sein: (1.) Enthält direkt den Anwendungsfall der Kostenreduktion. (2.) Der Aufbau einer Roboterarbeitszelle kann, je nach Arbeitsumfeld, gefährlich sein, etwa in Wiederaufarbeitungsanlagen, so dass der Versuchsaufbau in der Simulation stattfindet. (3.) Probeaufbauten werden in der Realität nur selten durchgeführt, da dies mit weiteren Kosten verbunden ist. Ohne simulationsgestützte Zellenplanung kommt es regelmäßig vor, dass Roboter in ihrer Arbeitszelle verschoben werden müssen, da erforderliche Stellungen nicht angefahren werden können. Diese Aktionen können durch vorangestellte Simulationsläufe vermieden werden. (4.) Dynamische Größen, wie Kräfte, Beschleunigungen oder Verschleiß am Roboter, sind in der Realität schwer messbare Größen, die mit Simulationssystemen einfacher zur Verfügung gestellt werden können. (5.) Roboterprogramme können mit modernen Simulationssystemen schneller als in Echtzeit bearbeitet werden, so dass auch schneller Ergebnisse etwa über Kollisionsfreiheit des Roboterprogramms vorliegen.

3.2 Eingliederung der Robotersteuerungssimulation in ein Simulationssystem

Prinzipiell werden die Zustandsdaten der VRC, wie in Abbildung 2.1 dargestellt wurde, nicht an eine Hardware zur Ansteuerung der Kinematik weitergegeben, sondern an das Simulationssystem, innerhalb dessen die Zustandsdaten, wie z.B. Achswinkel und Geschwindigkeiten aufbereitet und dem Anwender auf Wunsch z.B. in Form einer 3D-Arbeitszellenansicht zur Verfügung gestellt werden.

Das Abbildung 3.1 zeigt einen ”Screenshot” der Benutzerschnittstelle des Simulationssystems. Hierbei handelt es sich um das am IRF entwickelte COSIMIR, siehe [Uth97] oder [FRUvdV]. COSIMIR dient als übergeordnetes Simulationssystem der in dieser Arbeit beschriebenen R/VRC.

Das Modell der Arbeitszelle innerhalb eines Simulationssystems enthält verschiedene aktive und passive Objekte, die sich in den unterstützen Objekttypen zuordnen lassen. In dieser Arbeit ist der Objekttyp “**Kinematik-Steuerung**” relevant, der benötigt wird, um eine kinematische Kette zu bewegen bzw. zu programmieren und zu steuern. Einige weitere verallgemeinerte Typen, die jedoch nicht in jedem Simulationssystem zur Verfügung stehen müssen, seien hier nur der Vollständigkeit halber aufgelistet: **Inaktive Objekte** sind passive geometrisch modellierte Körper; **Fließbänder** sind Flächen, auf denen Objekte haften bleiben und mit eingestellter Geschwindigkeit in definierte Richtung bis zur Begrenzung der Fläche befördert werden; **Einleger** sind Primitivkinematiken mit digitaler Positionierung und Festanschlägen. Mit **SPS-Modulen** können innerhalb der Simulation SPS-Programme simuliert werden; **formbare Objekte** mit zugehöriger Steuerung z.B. Objekte, auf deren Gestalt mit Hilfe anderer Objekte Einfluss genommen

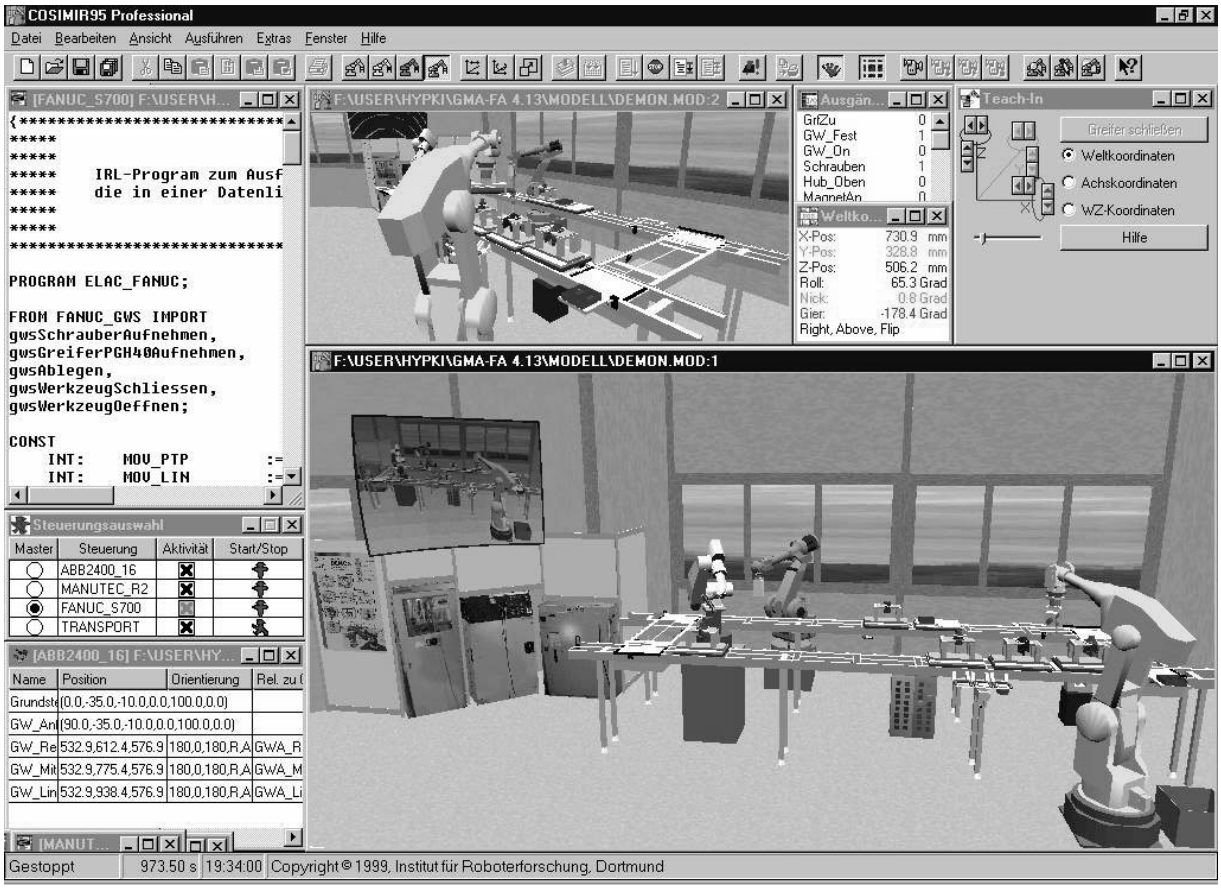


Abbildung 3.1: Bildschirm des COSIMIR Simulationssystems

werden kann, z.B. zu bearbeitende Werkstücke oder etwa zersägbare Bäume bei einer Waldarbeitsmaschinensimulation (siehe [FKR]); **beeinflussende Objekte** z.B. Laserstrahl, Sandstrahl, Polier- oder Schleifscheibe, Messer, Säge, Fräser; **Sensor-Objekte** ermitteln aus anderen simulierten Objekten in ihrem Erfassungsbereich Ausgangsdaten, die z.B. in einer Robotersteuerung oder SPS ausgewertet werden können, z.B. Abstands- und induktive Sensoren, (Reflex-)Lichtschranken, Fächerscanner oder Farb-Sensoren.

Vorraussetzung für die Unterstützung eines Objekttyps ist eine passende Schnittstelle des Simulationssystems, verbunden mit einer internen Objektverwaltung, die den entsprechenden Objekttypen konsistent in das Gesamtmodell aller Objekte eingliedert. Diese Mechanismen sind nicht Bestandteil dieser Arbeit. Einzelheiten hierzu können z.B. unter [Uth97] genauer nachgelesen werden.

3.3 Simulationslaufzeiten

Innerhalb des Simulationssystems müssen während eines Simulationslaufes mehrere Aufgaben gleichzeitig bearbeitet werden:

1. Bearbeitung geladener Programme.
2. Informationsaustausch zwischen mehreren Steuerungen und Objekten.
3. Ausführung von Überwachungsfunktionen, wie z.B. einer Kollisionserkennung.
4. Darstellung von Informationen auf dem Bildschirm, z.B. die 3D-Arbeitszellenansicht.

Bei jeder relevanten Änderung des Systemzustandes, etwa wenn die Robotersteuerung die Einnahme einer neuen Position meldet, das Ende eines Interpolationstaktes oder die Änderung eines Ausganges, muss dieser Zustand auf Konsequenzen innerhalb der kompletten Simulationsumgebung überprüft werden (z.B. Kollision, 3D-Ansicht neu zeichnen, verknüpften Eingang einer SPS-Simulation setzen, Sensorwerte bestimmen und weiterleiten usw.). Je aufwändiger diese Folgen einer Änderung des Systemzustandes sind, desto mehr Zeit nimmt der Simulationslauf in Anspruch. Die Dauer des Simulationsvorganges kann dadurch mehr Zeit in Anspruch nehmen als derselbe Ablauf in der realen Zelle. Daher muss eine ”simulierte Zeit” eingeführt werden, die bei jedem Vorgang um einstellbare elementare Zeitschritte voranschreitet. Sie kann z.B. während der *Auswertung* des Zustandes stehen bleiben, falls dieser Vorgang in der Realität keine relevante Zeit in Anspruch nimmt. Wenn die Steuerungen innerhalb ihres festen Interpolationstaktes[1] neue Werte ermittelt haben, läuft die ”simulierte Zeit” ein entsprechendes Zeitquantum weiter. Diese ”simulierte Zeit” gibt am Ende die ermittelte Laufzeit des Vorganges in der Realität an, unabhängig davon, wie lange der Simulationsvorgang gedauert hat.

Um dem Anwender bei stark unterschiedlichen Simulationslaufzeiten und ”simulierten Zeiten” dennoch ein realistisches Zeitverhalten auch während des Simulationsvorganges bieten zu können, stellen manche Simulationssysteme Mechanismen zur Verfügung, die Simulationslaufzeiten mit den realen Zeiten zu synchronisieren, indem die Ausführung von Operationen, die für das Simulationsergebnis irrelevant sind, eingeschränkt oder gesteigert wird. Zu diesen Operationen gehört z.B. die grafische 3D-Darstellung der Arbeitszelle. So kann, etwa bei einer langsamer als in der Realität ablaufenden Simulation, die Aktualisierung der 3D-Ansicht seltener vorgenommen werden, um genügend Rechenzeit für die Steuerungssimulation und Zustandsauswertung zur Verfügung zu stellen[2]. Bei schneller als in der Realität ablaufender Simulation wird bei dieser Methode die Darstellungsfrequenz erhöht oder es werden entsprechende Warteintervalle eingefügt. Mit diesen Methoden ist aber keineswegs eine Echtzeit auf Steuerungsebene zu erreichen, da die Achswinkel nicht in einem festen Zeittakt, basierend auf der realen Zeit, ermittelt werden.

Durch passende Parametrierung der Zeitquantisierungsschritte einzelner Vorgänge innerhalb des Simulationssystems und innerhalb der VRC kommt die ”simulierte Zeit” der Laufzeit in der Realität sehr nahe.

Neben den schlecht messbaren realen Verzögerungen, die durch Signalübertragung und Auswertung entstehen, trägt die Bewegungssteuerung der VRCs bzw. die Bewegungssteuerung der realen Robotersysteme entscheidend zur Laufzeit von Roboterprogrammen bei. Die Signalauswertungszeiten können bei Handhabungsaufgaben gegenüber den Bewegungsdauern vernachlässigt werden. Die Exaktheit der Bewegungssteuerung der VRC ist der entscheidende Faktor der

[1] Auch modernste Robotersteuerungen verfügen über einen festen Steuerungstakt, der i.A. im Bereich mehrerer Millisekunden liegt. Asynchrone Steuerungen sind bisher nicht auf dem Markt.

[2] Die 3D-Darstellung benötigt bei dem verwendeten Simulationssytem deutlich mehr Rechenzeit als ein Steuerungstakt einer VRC.

Simulation, denn Taktzeitanalysen von Automatisierungsprozessen liefern relevante Kenndaten, die bei Systemplanungen, Kostenanalysen und der Roboterauswahl einfließen.

3.4 Unterstützung der Robotervielfalt durch eine VRC

Zu dem Ziel dieser Arbeit gehört es, eine möglichst umfassende Robotersteuerungssimulation durch ein VRC-Modul zu erreichen, mit der verschiedenste Robotersteuerungen simuliert werden sollen. Die Probleme, die hierbei zu lösen sind, sind bedingt durch herstellerspezifische Eigenschaften der zu simulierenden Robotersteuerungen und die Vielfalt der unterschiedlichen Systeme:

- Die Mengen und Arten verfügbarer kinematischer Ketten.
- Die Roboterprogrammiersprachen und deren Eigenheiten.
- Die Datentypen und Formate, mit denen die Steuerungen arbeiten.
- Die Daten, auf die der Anwender Zugriff und Einfluss haben kann.
- Die Eigenschaften der Bewegungssteuerung und der Interpolation.

In diesen Eigenschaften unterscheiden sich Robotersteuerungen teilweise eklatant und um eine Abbildbarkeit auf eine einzige VRC realisieren zu können, muss ihr Funktionsumfang die Vereinigungsmenge aller Eigenschaften enthalten. Daher muss eine derart skalierbare Steuerungssimulation eine funktionale Obermenge aller unterstützten Robotersteuerungen darstellen.

3.5 Kinematikvielfalt

Eine Robotersteuerung dient der kontrollierten Ansteuerung eines Kinematiksystems, das in der Regel aus mehr als einer Achse besteht und damit eine kinematische Kette bildet. Eine besondere Eigenschaft gegenüber einer reinen Einzelachssteuerung ist, dass diese Achsen koordiniert angesteuert werden können und sich das Ende der kinematischen Kette auf definierten, kartesischen Bahnen bewegen kann. Um diese Funktionalität zu erreichen, ist die Lösung des inversen kinematischen Problems (IKP) für die angeschlossene Kinematik erforderlich, die zu einer gegebenen kartesischen Position im Raum eine korrespondierende Gelenkstellung ermittelt und eine Bewegungssteuerung, mit der die interpolierten Stellungen berechnet werden. Zur Verbesserung der Güte eines ggf. unterlagerten Regelkreises (bei Verwendung als R/VRC mit angeschlossenem realen Roboter) werden zusätzlich zu den Achsstellungen auch die Achsgeschwindigkeiten berechnet.

In Abbildung 3.2 sind einige gängige Industrieroboterkinematiken dargestellt, die teilweise genau bestimmt (sechs Freiheitsgrade) und teilweise unterbestimmt sind (weniger als sechs Freiheitsgrade).

Sollen mittels einer gemeinsamen VRC mehrere Roboter gleichzeitig angesteuert werden, so ist diese VRC in entsprechend vielen Instanzen zu laden, wobei jede Instanz die Steuerung oder Simulation einer vollständigen Robotersteuerung übernimmt.

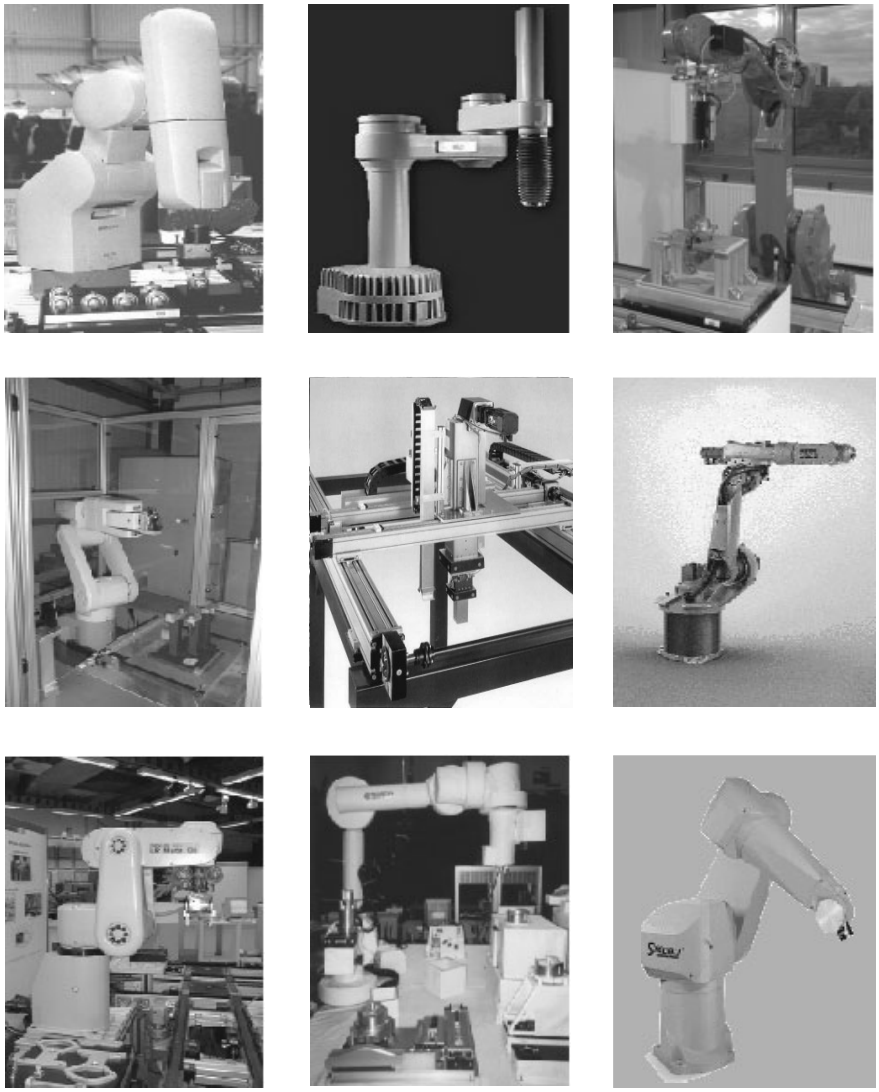


Abbildung 3.2: Auswahl zu unterstützender Kinematiken

3.6 Programmiersprachenvielfalt

Wie bereits in dem einleitenden Kapitel 2.1 erwähnt, konnten sich bisher keine systemübergreifenden Standards für Roboterprogrammiersprachen etablieren, und beinahe jeder Hersteller hat eigene Konzepte realisiert. Die Programmiersprachen haben sich gegenüber den Anfängen der Robotik zu leistungsfähigen und umfangreichen Hochsprachen mit prozeduralen und funktionalen Strukturen weiterentwickelt. Modulare Konzepte sind bei fast jeder Roboterprogrammiersprache zu finden. Auch Multitasking wird von vielen Systemen unterstützt.

Eine VRC, mit der verschiedene Industrieroboter simuliert werden sollen, muss idealer Weise auch diese Vielfalt an leistungsfähigen Programmiersprachen unterstützen, indem sie über ein umfangreiches und parametrierbares Interpretersystem verfügt, auf das sämtliche dieser verschiedenen sprachlichen Merkmale abgebildet werden können.

3.7 Flexible Bewegungssteuerung

Ein weiterer wesentlicher Bestandteil einer Robotersteuerung ist die Bewegungssteuerung. Das Verhalten der Bewegungssteuerungen verschiedener Robotersteuerungen kann sich, wie die Programmiersprachen und die anzusteuernden Kinematiktypen, eklatant unterscheiden.

Um auf die Verschiedenenartigkeit genauer eingehen zu können, muss zunächst der Funktionsumfang klassifiziert werden. Zu den Standard-Bewegungstypen einer Bewegungssteuerung gehören:

- PTP-Interpolation,
- lineare Interpolation und
- zirkulare Interpolation.

Darüber hinausgehend bieten nur wenige Robotersteuerungen besondere Bahninterpolationsformen an, wie:

- Bezier-,
- Spline- und
- funktionsbasierte Interpolation.

Jede einzelne Interpolationsart kann durch spezielle Parametersätze charakterisiert werden.

Bei der PTP-Bewegung z.B. durch:

- Start- und Zielvorgaben in verschiedenen Datentypen, z.B. absolut, relativ zu anderen Positionen, in kartesischen oder in Achskoordinaten,
- Beschleunigungsverläufe der Start- und Stopprampen jeder einzelnen Achse,
- synchrone oder asynchrone Koordination der Achsen untereinander,
- maximale Beschleunigungen und Geschwindigkeiten der Achsen,
- Art eines Verschleifkriteriums,
- Methoden mit denen Bewegungen verschliffen werden sollen und
- Verhalten im Fehlerfall.

Bei einer Linearbahn lassen sich folgende Charakteristika aufführen:

- Start und Zielvorgaben in verschiedenen Datentypen, z.B. absolut, relativ zur aktuellen oder zu anderen Positionen oder Geräten, in kartesischen oder in Achskoordinaten,

- Beschleunigungsverläufe der Start- und Stopprampen,
- Art der Orientierungsinterpolation,
- Verfahren bei kurzer Bahn und größerer Umorientierung,
- Methoden, mit denen Bahnen verschliffen werden sollen,
- Art des Verschleifkriteriums,
- Art eines ungeplanten Bewegungsstopps (bahntreu oder PTP),
- maximale Beschleunigungen und Geschwindigkeiten der Achsen und
- Verhalten im Fehlerfall.

Bei der Zirkularinterpolation kommen noch weitere Spezifika hinzu:

- Die Orientierungsführung kann im Welt- oder im mitdrehenden Koordinatensystem interpoliert werden.
- Ein Kreisbogen kann auf viele unterschiedliche Arten definiert werden: z.B. durch drei Punkte auf der Bahn, durch zwei Punkte auf der Bahn und dem Mittelpunkt oder durch den Mittelpunkt, einen orientierten Radiusvektor und dem Drehwinkel, um nur einige Methoden zu nennen. Zusätzlich kann stets durch einen Winkel der Drehsinn und der Drehwinkel vorgegeben werden.

Zusätzlich zu den Standard-Interpolationsarten verfügen Robotersteuerungen in der Regel über Verfahren zum Verschleifen von Bewegungen. Dabei wird ohne Halt von einer Bewegung in die anschließende Bewegung übergegangen, und es muss während oder vor Beginn einer Bewegung bereits die anschließende Bewegung bekannt sein. Der Programminterpreter muss also mit einem einstellbaren Vorlauf[3], asynchron zur Bewegungssteuerung, nach dem nächsten Bewegungssatz suchen können.

Es lassen sich zwei Arten zum Verschleifen von Bewegungen anführen:

- Verschleifen im kartesischen Raum (Bahn-Verschleifen).
- Verschleifen auf Achsebene (PTP-Verschleifen), wobei die kartesische Bahn des TCP sekundär und nicht von Interesse ist.

Die Bewegungsabschnitte während des Verschleifens lassen sich durch Antworten auf folgende Fragen charakterisieren:

[3] Der Vorlauf bestimmt über eine einstellbare Einheit (Zeit, Programmzeilen, Quellcode-Programmzeilen), wie weit der Programminterpreter innerhalb des Programmes gegenüber der Bewegungssteuerung in die “Zukunft” schauen darf, damit Bewegungen verschliffen werden können.

- Wann wird während einer Bewegung mit dem Verschleifen begonnen und an welcher Stelle wird das Verschleifen auf der anschließenden Bahn beendet?
- Auf welche Weise werden Achsen während des PTP-Verschleifens interpoliert?
- Wie sieht die Raumkurve während des kartesischen Verschleifens aus?
- Wann wird PTP und wann kartesisch verschliffen?
- Wie groß, und mit welchem Maß soll der Vorlauf des Interpreters definiert sein?

Parallel zur Roboterkinematik unterstützen moderne Robotersteuerungen auch Zusatzachsen, die auf unterschiedlichste Weise koordiniert und mit dem Roboterprogramm synchronisiert werden können. Die Konzepte und die Leistungsfähigkeiten der realen Robotersysteme unterscheiden sich hierbei so stark voneinander, dass erneut eine Obermenge aller Funktionalitäten zur Verfügung gestellt werden muss. Eine Vereinfachung und Zurückführung auf eine Minimal-Unterstützung ist hier nicht gewollt.

- Zusatzachsen können zu Gruppen zusammengefasst werden, deren Achsstellungen untereinander synchron interpoliert werden.
- Zusatzachsgruppen oder einzelne Achsen müssen asynchron und synchron zu Roboterbewegungen und anderen Achsgruppen positionierbar sein.
- Es müssen Synchronisationsmechanismen zur Verfügung gestellt werden, damit innerhalb von Programmen auf Bewegungsereignisse gewartet oder reagiert werden kann.
- Zusatzachsen können gruppiert und zu einer kinematischen Kette kombiniert werden, zu der mindestens das DKP (Direktes Kinematisches Problem, Vorwärtstransformation) bestimmt werden kann.
- Zu einer kinematischen Kette kombinierte Zusatzachsgruppen definieren ein kinematisches Gerät[4], dessen Gerätekoordinaten[5] auch als kartesische Basis der Bahninterpolation des Roboters dienen kann, so dass in diesen Gerätekoordinaten operiert werden kann.
- Es sollten prinzipiell beliebig viele Achsgruppen unabhängig voneinander synchron oder asynchron steuerbar sein.
- Gruppendefinitionen und Gruppenaufhebungen sollen innerhalb des Roboterprogramms dynamisch möglich sein.

Stellt man diese Funktionalitäten zur Verfügung, so werden alle wichtigen in der Industrie vorkommenden Einzelfälle auf eine Steuerung abbildbar. Es ist damit möglich, aufwändige Roboteranlagen mit Zuführern und Positionierern mit nur einer Steuerung zu betreiben.

[4]Gerät: In dieser Arbeit soll damit ein aus mindestens einer Achse bestehender kinematischer Aufbau bezeichnet werden, dessen Achsstellung von der Robotersteuerung berechnet wird. Auch der Roboter selbst ist ein kinematisches Gerät.

[5]Gerätekoordinaten: Position (inkl. Orientierung) des Endes des Gerätes, z.B. die Mitte eines Dreh-Schwenktisches oder die Werkzeugspitze eines Roboters.

3.8 Parametrierbarkeit der Robotersteuerungssimulation

In den vorangegangenen Kapiteln wurde beschrieben, in welchen Bereichen Unterschiede zwischen verschiedenen Robotersteuerungen bestehen und welche Anforderungen an die Parametrierbarkeit einer einzigen Robotersteuerungssimulation gestellt werden müssen, damit diese ein dem realen Roboter möglichst ähnliches Verhalten zeigt.

Eine Zusammenstellung aller Parameter, mit denen eine Robotersteuerung *vollständig* und *exakt* charakterisiert ist, stellt eine unlösbare Aufgabe dar [Bal96], [Rie97]. In einem kombinierten System aus Robotersteuerungssimulation und grafischer Simulation wird der Weg beschritten, *relevante* Robotersteuerungsparameter zu klassifizieren und schrittweise zur Parametrierung einzusetzen.

Zunächst kann eine Menge an Eigenschaften extrahiert werden, die bei jeder Robotersteuerung vorhanden ist. Diese als global bezeichneten Parameter werden innerhalb des Simulationssystems unter den Modelldaten der Arbeitszelle abgelegt. Mit diesen Daten wird die VRC beim Initialisieren der Arbeitszelle vorparametriert. Zur weiteren genauen Einstellung des Systemverhaltens auf die sprach- und herstellerspezifischen Eigenschaften sind besondere Anweisungen im ”Start-Up-Code”[6] des Roboterprogramms vorhanden.

An dieser Stelle wird auf Parameterdaten eingegangen, die in Datentyp und Umfang auf jede Robotersteuerung anwendbar sind. Die spezielle Parametrierung mittels IRDATA-Anweisungen wird in Kapitel 5.3.5 erläutert.

Folgende Daten klassifizieren ein kinematisches Gerät:

- Definition der Kinematiken mittels Denavit Hartenberg-Parametern (siehe [DH55] oder [AS86]) und der folgenden Kinematik-Klassifikation zur Auswahl des zuständigen Kinematikmoduls.
- Beschreibung der Verfahrbereiche der Achsen.
- Definition der dynamischen Grenzwerte auf Achsebene, wie max. Geschwindigkeiten und Beschleunigungen.
- Definition der dynamischen Grenzwerte im kartesischen Raum (falls für diese Kinematik die Lösung des IKP zur Verfügung steht).
- Definition der vorhandenen Ein-/ und Ausgangskanäle (digitale und analoge I/O).
- Definition immanenter Achsverkopplungen, z.B. zur Definition von Parallelogramm- bzw. Schleifenkinematiken, also Kinematiken mit Achswinkeln, die sich gegenseitig beeinflussen.
- Position des Aufbaus im Raum.
- Definition von Greifer- bzw. Tooltransformationen.

[6] Start-Up-Code: Code der bei Programmstart als erstes bearbeitet wird, bevor die Bearbeitung des Anwenderprogramms beginnt.

- Definition von zusätzlichen kinematischen Geräten und ggf. deren Eingliederung in andere kinematische Ketten (Geräte, die mit dem Roboter zu einer kinematischen Kette verkoppelt sind, etwa Schienen oder zusätzliche Handachsen).

3.9 Anforderungen an die Erweiterung zur universellen Steuerung realer Kinematiksysteme

Die in den vorangegangenen Kapiteln beschriebenen Anforderungen gelten uneingeschränkt auch für den Betrieb als R/VRC, mit dem der wahlweise Betrieb als Simulations- oder Steuerungssystem möglich sein soll. Mittels zu definierender offener Schnittstellen zu den Roboterantrieben steht damit eine universelle Robotersteuerung zur Verfügung, mit der sämtliche in der Simulation unterstützte Systeme auch real ansteuerbar werden.

Folgende Punkte bilden die Basis für die zusätzlichen Anforderungen an die R/VRC:

1. Sämtliche für die Ansteuerung von realen Kinematiken erforderlichen Größen werden über definierte Schnittstellen rechtzeitig, d.h. in Echtzeit, zur Verfügung gestellt.
2. Der Anwender kann jederzeit zwischen reinem Simulations- und Steuerungsbetrieb umschalten.
3. Die Sicherheitskriterien einer Robotersteuerung werden erfüllt.
4. Mittels Handbediengerät kann der Roboter exakt positioniert werden.
5. Die Zeitbasis im Steuerungsbetrieb ist die ”echte” Zeit. Eine ”simulierte Zeit”, wie sie in Kapitel 3.3 erklärt wurde, steht nicht zur Verfügung.
6. Die Schnittstelle zum Simulationssystem bleibt unverändert, so dass mehrere R/VRC gleichzeitig aktiv sein können.
7. Es soll ein simultaner Mischbetrieb von R/VRC und VRC möglich sein.
8. Die Programmierung der Roboter soll mittels verschiedener Roboterhochsprachen erfolgen können, die in einen einheitlichen Zwischencode, hier IRDATA, übersetzt werden.

Kapitel 4

Konzeption der Integration von Simulation und Steuerung

Bei der Konzeption des Gesamtsystems, das in der Lage sein soll, die 3D-Ansicht einer Arbeitszelle darzustellen und die enthaltenen Roboter auch anzusteuern, sind einige Nebenbedingungen zu beachten. Vorgehensweisen und Verfahren zur nicht echtzeitfähigen Anwendungsentwicklung [Röß97] sind nicht ausreichend, da auch das Betriebssystem und seine Eigenschaften mit in die Entscheidungen einbezogen werden müssen. Entscheidende Merkmale sind hierbei z.B. Betriebssicherheit, bisherige Marktdurchdringung und -etablierung sowie Echtzeitfähigkeit.

Zur Integration von Steuerungssimulation und Steuerung zu einer R/VRC müssen im Vorfeld die genauen Anforderungen eines solchen Systems an das Betriebssystem und die zu Grunde liegende Hardware definiert werden. Im Anschluss daran sind Untersuchungen bestehender Betriebssysteme oder Erweiterungen erforderlich.

4.1 Echtzeit-Forderung

Eine Robotersteuerung verfügt über eine Bewegungssteuerung, die in einem Steuerungstakt Achswinkel/ -strecken und Geschwindigkeiten errechnet und an eine unterlagerte Regelung übergibt. Kann dieser Takt nicht eingehalten werden, so bleiben die alten Sollwerte i.A. beim Regler anliegen und der Regler versucht diese Position beizubehalten. Wenn die Achse jedoch in Bewegung war, resultiert hieraus eine Stellgröße, die in der Stellgrößenüberwachung zu einem Fehler führen kann, in Folge dessen der Roboter augenblicklich angehalten wird. Das liegt daran, dass ein Roboterarm trägheitsbehaftet ist und bei nennenswerten Geschwindigkeiten nicht innerhalb eines Steuerungstaktes auf Stillstand geregelt werden kann. Manche Reglersysteme überwachen den Aktualisierungszeitpunkt der vorgegebenen Sollwerte und wechseln in den ähnlich schwerwiegenden Notaus-Zustand.

Dieser Sachverhalt verdeutlicht die Notwendigkeit der *pünktlichen* Bereitstellung der Sollwerte. Die Frequenz des Steuerungstaktes hat naturgemäß einen Einfluss auf die Genauigkeit einer Roboterbewegung. Je höher der Steuerungs- und Reglertakt, desto bahntreuer werden die Sollwerte erzeugt und desto mehr gewinnt letztlich die Reglergüte an Einfluss. Diese Anforderun-

gen, die hier an die Bewegungssteuerung gestellt werden, verdienen daher die Bezeichnung harte Echtzeit-Anforderungen.

Ferner muss untersucht werden, welche Betriebssysteme diesen harten Echtzeitanforderungen gerecht werden und mit welchen Mitteln diese ggf. erreichbar sind.

Nach DIN 44300 gilt:

”Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig derart betriebsbereit sind, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind”. Über die Größe dieser Zeit sagt die Norm nichts aus. Das ist auch nicht erforderlich, da die Latenzzeiten benennbar, also als Eigenschaften des Systems bekannt sind. Diese Eigenschaften muss der Entwickler beim Systemdesign berücksichtigen.

4.2 Zielplattform der R/VRC

Die Konzeption und Realisierung der hier beschriebenen R/VRC soll möglichst systemunabhängig erfolgen, so dass eine Portierung der Steuerung auf unterschiedliche Rechnersysteme leicht möglich ist. Zu diesem Zweck werden sämtliche systemspezifischen Funktionen mittels separater Interface-Funktionen gekapselt. Benutzerein- und -ausgaben erfolgen ausschließlich über das Simulationssystem. Auf diese Weise ist der Kern der R/VRC auf verschiedenen Betriebssystemen und Rechnerplattformen lauffähig.

Da sich in den letzten Jahren der PC zu einer universellen, kostengünstigen und leistungsfähigen Rechnerplattform entwickelt hat, wird er als Entwicklungsplattform für die R/VRC bevorzugt. Vorbehalte, die noch vor kurzer Zeit gegen den Einsatz von PCs in der Industrie sprachen, sind immer mehr entkräftet worden, so dass ihre Verbreitung mit großen Schritten auch in der Automatisierungstechnik voran schreitet**.**

Für den Einsatz von PCs sprechen folgende Punkte:

1. kostengünstig,
2. umfassendes Angebot an Hard- und Software,
3. ausgezeichnet dokumentierte Hard- und Software,
4. hervorragende Entwicklungsumgebungen,
5. standardisierte Schnittstellen,
6. große Auswahl an Modellen und Typen,
7. Erweiterbarkeit,
8. Steckkarten für fast jeden Zweck erhältlich,
9. verschiedene Betriebssysteme verfügbar und

10. Mehrfachverwendbarkeit: Der PC kann neben der geforderten Steuerungsaufgabe auch andere Applikationen verarbeiten. Je nach Wahl des Betriebssystems sogar parallel zu den eigentlichen Steuerungsaufgaben.

Diese Gründe sprechen dafür, die Implementierung und den Betrieb der universellen R/VRC auf dem PC durchzuführen.

4.3 Geeignete Systemstrukturen

Soll die Simulation und Steuerung von Roboterarbeitszellen in einem einzigen System zusammengefasst werden, so stellt das Gesamtsystem ein komplexes Gebilde dar, das in mehrere Schichten gegliedert werden kann.

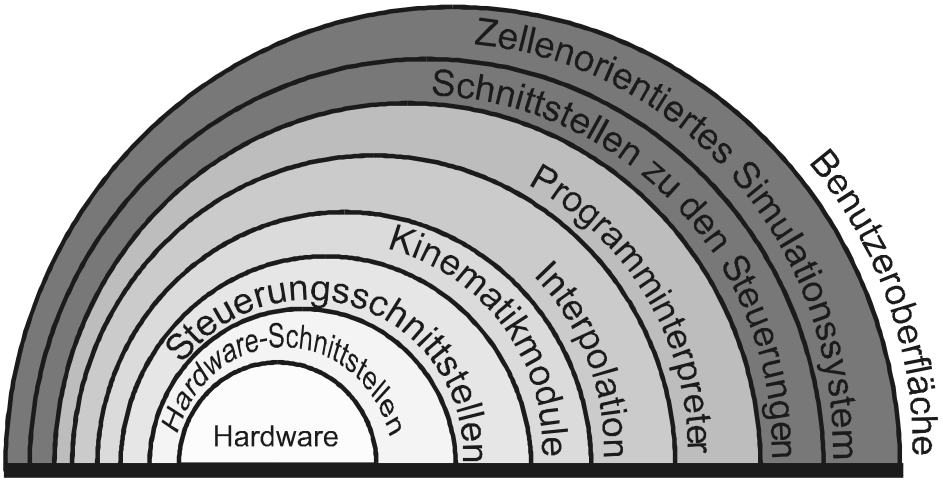


Abbildung 4.1: Schichtenmodell mit den Ebenen der Steuerung

Abbildung 4.1 stellt ein solches Schichtenmodell der Steuerung dar, in dem neun Ebenen von der Anwendungsoberfläche des Simulationssystems bis hinunter zu den Schnittstellen zum Roboter angelegt sind. Dieses Modell verdeutlicht die Integrations- und implementierungsbreite eines derartigen Systems, indem sich die R/VRC über sämtliche Ebenen des Betriebssystems, angefangen an der Benutzeroberfläche, bis hinunter zur Hardware erstreckt. Neben den in der Grafik verständlichen Bezeichnungen der Ebenen wird hier aus Platzgründen auf eine genaue Erläuterung der Ebenen verzichtet.

4.3.1 Lösung mit verteiltem System

Bei der Realisierung einer R/VRC auf PCs unter Windows NT ergeben sich wegen der erforderlichen Echtzeitfähigkeit betriebssystembedingte Schwierigkeiten, denn Windows NT ist kein Echtzeit-Betriebssystem. Für eine Robotersteuerung ist es jedoch zwingend erforderlich, die

Achsstellungen pünktlich bereitzustellen. Für diese Aufgabe sind in der R/VRC die Ebenen innerhalb der Interpolation (Abbildung 4.1) verantwortlich. Die Benutzeroberfläche und der Kern des Simulationssystems müssen nicht echtzeitfähig sein. Daher liegt der konzeptionelle Gedanke nahe, Komponenten unterhalb des Kerns des Simulationssystems[1] auf einen separaten Rechner auszugliedern, der mit einem Echtzeitsystem arbeitet, wodurch ein auf mehrere Rechner verteiltes System entsteht, bei dem die Anwenderoberfläche von den Steuerungskomponenten getrennt ist. Dazu ist eine breitbandige Verbindung zwischen den beiden Systemen erforderlich, da der Anwender permanent über den Systemzustand informiert werden möchte und auch in Gegenrichtung die Steuerung beeinflussen können muss.

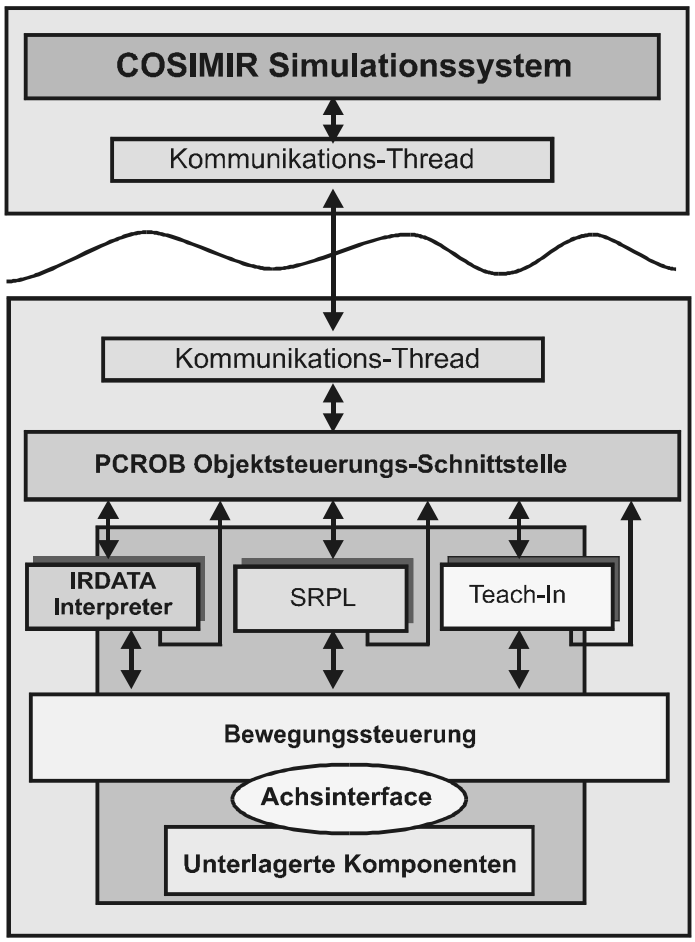


Abbildung 4.2: Verteilte Robotersteuerungen

Die Trennungslinie zwischen Steuerung und Zentralrechner kann dabei auf zwei verschiedenen Ebenen gezogen werden. Die beiden Abbildungen stellen die unterschiedlichen Konzepte dar:

1. Typ 1, dargestellt in Abbildung 4.2: Abkopplung der gesamten R/VRC vom zentralen Rechner, so dass die Daten der Schnittstelle zwischen Robotersteuerung und Simulations-

[1] Unterhalb des Kerns des Simulationssystems bedeutet unterhalb der Schnittstellen zu den Steuerungen.

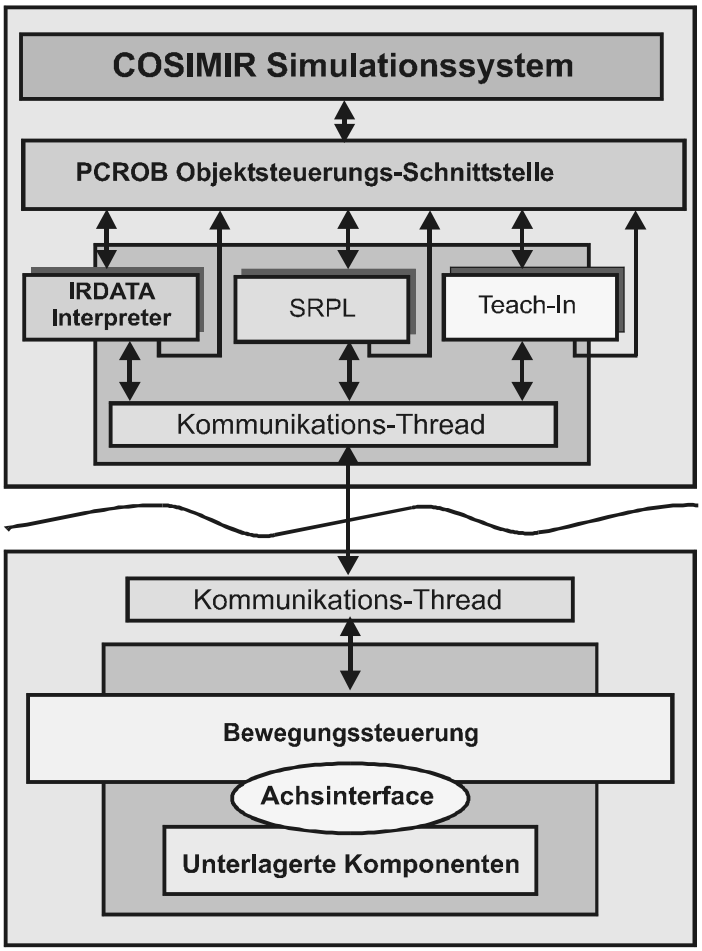


Abbildung 4.3: Verteilte Bewegungssteuerungen

system über das Netzwerk übertragen werden müssen[2].

2. Typ 2, dargestellt in Abbildung 4.3: Abkopplung der Bewegungssteuerung (Interpolation) der R/VRC vom Anwendungs-PC, so dass die Daten der Schnittstelle zwischen Bewegungssteuerung und den übergeordneten Steuerungskomponenten per Netzwerk übermittelt werden müssen. Hierbei befindet sich der Programminterpreter innerhalb des zentralen PCs und liefert Bewegungsanforderungen über die Verbindungen zur Bewegungssteuerung.

4.3.2 Realisierung eines verteilten Test-Systems

Ein verteiltes System nach Typ 1 (Abbildung 4.2) wurde zu Testzwecken mittels zweier PCs aufgebaut. Dabei wurde die Robotersteuerung vollständig auf einen QNX-basierten[3] PC portiert, und die gesamte Kommunikation zu den oberen Schichten des Simulationssystems mittels

[2]Nicht alle in der Grafik enthaltenen Komponenten sind für diesen Abschnitt relevant. Auf sie wird in einigen folgenden Kapiteln genauer eingegangen.

[3]QNX ist ein Echtzeit PC-Betriebssystem der Firma QNX Software Systems.

TCP/IP realisiert. Dabei stellte sich heraus, dass die beschränkte Bandbreite der Netzwerkverbindung während des Konfigurationsvorganges der Steuerung zu erheblichen Zeitverzögerungen führte und auch während des Betriebes nicht annehmbare Antwortzeiten auftraten. Diese "Kommunikations-Flutwellen"[4] treten immer dann auf, wenn sich große Datenblöcke innerhalb des Zustandsraumes der Steuerung oder des Simulationssystems ändern und diese Änderungen augenblicklich zu weiteren Datenübertragungen führen, z.B. in folgenden Fällen:

1. Die Steuerung wird (re-) konfiguriert, z.B. wenn ein anderer Robotertyp parametriert wird.
2. Die Steuerung lädt ein neues Roboterprogramm über die Netzwerkverbindung.
3. Während des Programmablaufes werden programmgesteuert viele Ausgangsports der Steuerung gesetzt oder gelöscht.
4. Während des Programmablaufes lädt oder schreibt die Steuerung persistente Datenlisten, etwa mit Positions- und Parameterdaten.
5. Weil dem Anwender während eines Programmlaufes eine Quellcode-Ablauf-Verfolgung[5] zur Verfügung stehen soll, ist die Netzwerkverbindung durch den Austausch von aktuellen Programm-Informationen stark belastet.

Die Datenmengen, die zur korrekten Darstellung des Roboters in der 3D-Ansicht erforderlich sind, stellen kein Problem dar.

Vorteile eines verteilten Systems:

1. Es liegt eine klare Trennung zwischen der grafischen Oberfläche und der R/VRC mit den zeitkritischen Komponenten vor.
2. Der Steuerungsteil läuft auf einem autonomen Echtzeit-Multitasking-Betriebssystem, auf dem keine weitere Software parallel zur Steuerungssoftware Rechenzeit verbraucht.
3. Der Steuerungsrechner kann beim Start des Simulationssystems "in-Time" via Netzwerk mit der aktuellsten Steuerungssoftware gebootet werden und verlangt keine weiteren Aktionen vom Anwender.
4. Je nach verwendetem Betriebssystem für den Steuerungsrechner kann die Software-Entwicklung auf dem Host-System unter Windows erfolgen.

[4] In der Kraftwerks-Leittechnik als "Trips" bezeichnet.

[5] Während das Programm läuft, kann der Anwender in einem Fenster den Fluss seines Roboterprogrammes verfolgen, indem die aktuell von der Steuerung bearbeitete Zeile hervorgehoben dargestellt wird. Dazu muss die Steuerung der Oberfläche ständig mitteilen, wo der Programmzeiger gerade steht. Da die Steuerung einen schnellen IRDATA-Interpreter enthält (mit gegenwärtig bis 100.000 Zeilen pro Sekunde), kann das zu einer erheblichen Netzbelastung führen.

Nachteile eines verteilten Systems:

1. Es sind zwei Rechner erforderlich.
2. Es sind zwei (nicht unbedingt verschiedene) Betriebssysteme (Lizenzen) notwendig.
3. Der Steuerungsrechner steht nicht für weitere Verwendungszwecke zur Verfügung.
4. Es müssen ggf. verschiedene Entwicklungs-Tools verwendet werden, wobei u.U. weitere Einarbeitung erforderlich ist..
5. Der Einarbeitungsaufwand in ein neues Betriebssystem ist u.U. mit erheblichem Zeit- und Kostenaufwand verbunden.
6. Es muss für eine schnelle, echtzeitfähige Verbindung zwischen den Systemkomponenten gesorgt und diese angesteuert werden.
7. Der Entwicklungs- und Wartungsaufwand erhöht sich durch den Einsatz zweier PCs.

Insgesamt verursacht ein verteiltes System einen erheblichen Mehrkosten- und erhöhten Zeitaufwand, der nicht nur durch einen erhöhten Entwicklungsaufwand entsteht. Auch die Hardware, die ein solches System bildet, ist deutlich teurer als ein Einzelrechner-System. Daher soll im nächsten Abschnitt die Variante mit einem einzigen PC beschrieben werden.

4.3.3 Lösung mit Einzelrechner-System

Eine Alternative zum verteilten System ist ein Einzelrechner-System. Hierbei laufen sämtliche Module und Threads auf demselben PC. Bei diesem PC darf es sich zweifellos auch um ein Mehrprozessor-System handeln. Ein auf einem Rechner integriertes Steuerungs- und Simulationssystem erfordert folgende Eigenschaften:

- Das Betriebssystem stellt in idealer Weise alle für den Betrieb eines Simulationssystems notwendigen Ressourcen und Komponenten zur Verfügung, wie etwa
  - Multitasking,
  - grafische Benutzeroberfläche,
  - Netzwerk-Technologien,
  - Standard-Schnittstellen und
  - File-System-Treiber [Cus93].
- Die Anwendungsoberfläche läuft in einem anderen Prozessorthread mit deutlich geringerer Priorität als die echtzeit-relevanten Teile, da die Visualisierung sehr rechenzeitintensiv ist, aber dennoch die pünktliche Erzeugung von Gelenkwinkeln nicht behindern darf.
- Die Trennung zwischen Echtzeit- und den restlichen Systemkomponenten kann, wie beim verteilen System, auf unterschiedlichen Ebenen erfolgen.

Vorteile des Einzel-PCs:

- Kostenersparnis, da nur ein Rechnersystem angeschafft werden muss.
- Es ist keine schnelle Verbindung zwischen zwei Rechnersystemen erforderlich, denn der Datenaustausch kann über gemeinsame Speicherbereiche abgewickelt werden (Shared Memory).
- Der gesamte Hardware-Aufwand verkleinert sich.
- Ein konzentriertes System vereinfacht den Systeminstallations- und -wartungsaufwand.

Nachteile der Einzel-PC-Lösung:

- Je nach Wahl des Betriebssystems ist die Integration des unbedingt echtzeitfähigen Steuerungsmoduls eventuell sehr aufwändig.
- Es ist die Beschaffung von Betriebssystemerweiterungen zur Erlangung der Echtzeitfähigkeit erforderlich.
- Die Akzeptanz von PC-basierten Lösungen in der Automatisierungstechnik nimmt zwar stark zu, hat aber noch nicht das Maß der ”Embedded Systeme” erreicht.

4.4 Anforderungen an das Betriebssystem

Wie bereits im Kapitel 4.1 angeführt, ist es erforderlich, im harten Echtzeittakt Achssollwerte für die Regelung zur Verfügung zu stellen. Diese Anforderungen werden nur von speziellen Echtzeit-Multitasking- Betriebssystemen oder von speziell erweiterten Standardbetriebssystemen wie Linux oder Windows NT 4.0 erfüllt. Soll die Integration von Simulation und Steuerung auf einem Rechner laufen, also kein verteiltes System sein, dann kommt zusätzlich die Forderung nach einer 3D-Grafik-geeigneten Oberfläche hinzu, um die Arbeitszelle in einer akzeptablen Bildaufbaurate darstellen zu können. Obwohl bereits Windows NT als Wunschbetriebssystem deklariert wurde, müssen auch die Alternativen und Erweiterungen zu Windows NT untersucht werden.

Die folgende kommentierte Liste enthält in Betracht gezogene Betriebssysteme und Erweiterungen für PCs aus einer Liste von 102 zur Verfügung stehenden Echtzeit-Betriebssystemen und -Erweiterungen[6].

Erweiterungen zu Windows NT:

1. Hyperkernel von Nematron Corporation:
   Hyperkernel erlaubt Software-Entwicklern die Integration hochdeterministischer Echtzeit-Anwendungen in Windows NT.
   Diese Erweiterung ist für das Vorhaben geeignet, jedoch fallen Lizenzgebühren für jedes System an.

[6] Unter diesen 102 Systemen befinden sich auch Betriebssyteme für Embedded Designs.

2. INtime (real-time Windows NT), iRMX von Radisys Corp.:
   Erweitert Windows NT um Echtzeitfähigkeiten.
   Diese Erweiterung ist für das Vorhaben geeignet, jedoch fallen Lizenzgebühren für jedes System an.

3. LP-RTWin Toolkit von LP Elektronik GmbH:
   LP-RTWin verwendet LP-Elektroniks harte Echtzeit-Technologie, um Wind-Rivers VxWorks parallel zu Windows NT auf dem gleichen Prozessor zu betreiben.
   Diese Erweiterung ist die am häufigsten verwendete Echtzeit-Erweiterung für Windows NT und wird von Industrieroboter-Herstellen bereits verwendet. Die Kosten für dieses System liegen über denen sämtlicher Mitbewerber.

4. LP-VxWin von LP Elektronik GmbH [Mun97]:
   Werkzeug, um echtzeitfähige Interrupt-Serviceroutinen[7] unter Windows NT zu entwickeln und installieren.
   Die Echtzeitfähigkeit im Interrupt und innerhalb des Kernels von Windows NT ist auch ohne diese Software bereits ausreichend.

5. Tornado/VxWorks von Wind River Systems Inc.:
   Tornado repräsentiert nach eigenen Angaben die letzte Generation von Entwicklungs- und Ausführungsumgebungen für Embedded und Echtzeitanwendungen, u.a. für Windows NT.
   Diese Erweiterung ist für das Vorhaben geeignet.

6. Real-time Extension (RTX) für Windows NT von VenturCom, Inc.:
   Stellt eine allgemeine Echtzeiterweiterung für Windows NT dar.
   Diese Erweiterung stellt nur rudimentäre Funktionen zur Verfügung und ist nur bedingt geeignet.

Andere Betriebssysteme:

1. ARTOS von Locamation:
   Erweiterter Echtzeitkernel mit transparenter Netzwerk-Fähigkeit, geeignet für Embedded Controller, Multiprozessor-Systeme und verteilte Steuerungen, basierend auf Standard-PC Hardware.
   Dieses System scheidet als Betriebssystem aus, da keine OpenGL-Unterstützung für verfügbare 3D-Grafikbeschleuniger vorliegt und damit keine schnelle, standardisierte 3D-Arbeitszellenansicht dargestellt werden kann.

2. OS-9 von Microware Systems Corp.:
   Vollständiges Echtzeit-Multitasking-Betriebssystem, modular und skalierbar, u.a. für Intel-Plattformen.
   Auch hier fehlt bisher eine OpenGL-Unterstützung für moderne 3D-Grafikkarten.

3. QNX von QNX SOFTWARE SYSTEMS EUROPA:
   Skalierbares POSIX kompatibles Echtzeit-Betriebssystem, sowohl für Highend Netzwerklösungen und Steuerungsaufgaben als auch für Embedded Systeme geeignet.

[7] Zur Vermeidung von Missverständnissen soll hier stets der englische Begriff verwendet werden.

Durch die Offenlegung der Quellcodes dieses Unix-Betriebssystem und die Ähnlichkeit mit Linux werden OpenGL-Treiber entwickelt. Das Simulationssystem und die R/VRC können grundsätzlich auf dieses System portiert werden.

4. Hard Hat Linux von MontaVista Software Inc.:
   Linux mit allen seinen Funktionalitäten, erweitert um Echtzeit-Eigenschaften.
   Durch die Offenlegung der Quellcodes dieses Unix-Betriebssystems und der Kompatibilität zu Linux stehen OpenGL-Treiber zur Verfügung. Das Simulationssystem und die R/VRC können grundsätzlich auf dieses System portiert werden.

5. RT-Linux von New Mexiko Tech.:
   Linux Echtzeiterweiterung.
   Durch die Offenlegung der Quellcodes dieses Unix-Betriebssystem und der Kompatibilität zu Linux stehen OpenGL-Treiber zur Verfügung. Das Simulationssystem und die R/VRC können grundsätzlich auf dieses System portiert werden.

Nicht alle Systeme konnten aus Kosten- und Zeitgründen vollständig untersucht werden, sondern die Studie beschränkte sich teilweise auf Erfahrungsberichte und Vergleiche, die von verschiedenen Magazinen und Lehrstühlen im Rahmen von Marktanalysen und Produktreports durchgeführt und veröffentlicht wurden [MS98], [TK98], Internet[8]. Informationen über aktuelle Echtzeit-Betriebssysteme und Erweiterungen sind zu finden unter: [Kop97], [Kle93], [HP88], [WM87].

Sämtliche der hier aufgeführten Systeme eigenen sich als Zielplattform für die Robotersteuerung. Als Zielplattform für die 3D-Simulationsumgebung kommen jedoch von vornherein nur Systeme in Frage, die OpenGL-Unterstützung für moderne Grafik-Hardware bieten. Da auf eine Portierung des Simulationssystems aus Zeitgründen zunächst verzichtet wird, bleibt Windows NT vorerst als primäre Implementierungsplattform für das Zielsystem bestehen. Die unter ”Andere Betriebssysteme” aufgeführten Pakete eignen sich nur für den Aufbau eines verteilten Steuerungssystems. Die Erweiterungen für Windows NT sind besonders interessant, weil sich mit ihnen ein kombiniertes System wunschgemäß auf einem einzigen PC unter Windows realisieren lässt.

Als Alternative zu den zur Auswahl stehenden kommerziellen NT-Echtzeiterweiterungen, wird auch eine Eigenentwicklung in Betracht gezogen, mit der Windows NT um die geforderte Echtzeitfähigkeit erweitert wird.

In der folgenden Tabelle werden die Vor- und Nachteile zwischen kommerziellen und eigenen Entwicklungen gegenübergestellt:

[8] Eine gute Ausgangsadresse hierfür ist http://www.realtime-info.be/

	Kommerziell	**Eigenentwicklung**
Vorteile:	Sofort verfügbar Gewährleistungsfrist Service und Support	Quellcode ist verfügbar Kontrollierter, angepasster Eingriff in das Betriebssystem ”Know-How” vor Ort Keine Lizenzgebühren Einfache Anpassungen
Nachteile:	Ggf. hoher Anschaffungspreis Meist teurer Support Kostenpflichtige Updates Aufwändige Einarbeitung Abhängigkeit vom Hersteller Laufzeit-Lizenzen	Aufwändige Einarbeitung Hoher Programmieraufwand

4.5 Untersuchung der Zielplattform Windows NT

Windows NT ist von Hause aus, wie bereits mehrfach betont, explizit *nicht* echtzeitfähig! Der Anwender hat mit den Standard-Entwicklungsumgebungen von Microsoft, Borland, Novell, Watcom, Sybase, Lotus, Symantec, Zortec, usw. keine Möglichkeit, Anwendungen zu schreiben, die als echtzeitfähig bezeichnet werden können, ohne dabei die Latenzzeiten in den Sekundenbereich legen zu müssen. Das liegt daran, dass diese Anwendungen im ”Anwender-Mode” = ”User Mode” des Betriebssystems laufen und vom Kern des Systems überwacht werden. Dessen ”Scheduler” kann die Prozessorzeit nicht echtzeitfähig auf die Anwendungen verteilen.

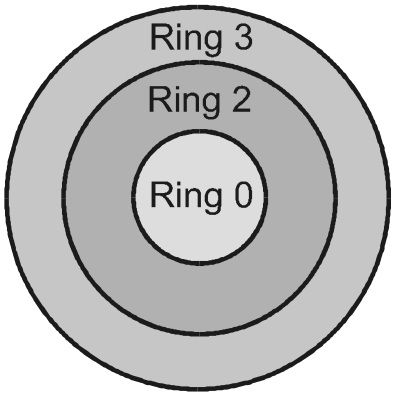


Abbildung 4.4: Prioritätenebenen

Abbildung 4.4 veranschaulicht die Prioritätenebenen eines Betriebssystems, eingeteilt in Ringe.

Ring 0 stellt die unterste Ebene des Betriebssystems dar. Dieses Niveau hat die höchste Priorität und befindet sich unterhalb der Sicherheitsmechanismen des Systems. Hier sind Zugriffe auf sämtliche Ressourcen möglich, so dass z.B. auf sämtliche Speicher und Port-Adressen zugegriffen werden kann. Jedoch sind die Ressourcen dieser Ebene auf Zugriffe von höheren Ebenen geschützt.

Ring 1 wird unter Windows NT nicht verwendet.

Ring 2 findet teilweise im 16-Bit Subsystem Anwendung, hat allerdings nur geringe Bedeutung.

Ring 3 mit der kleinsten Priorität wird als User Mode bezeichnet.

Die Grafik 4.5 aus [Mic01] zeigt Komponenten von Windows NT und die Teilung von User und Kernel Mode auf.

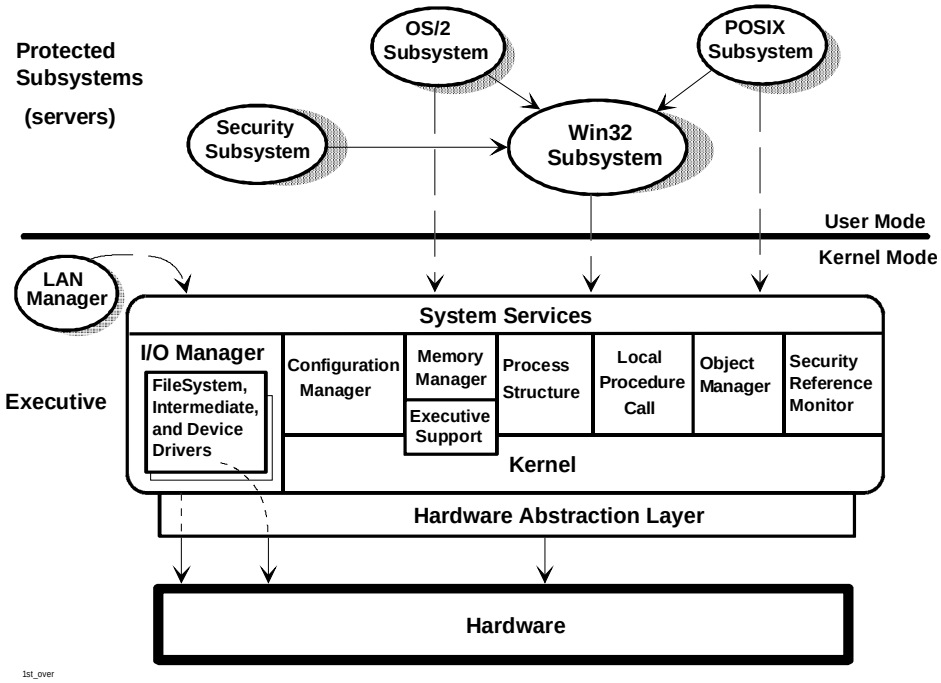


Abbildung 4.5: Windows-NT Systemkomponenten (nach [Mic01])

Um dennoch die notwendigen kleinen Latenzzeiten unter Windows erreichen zu können, müssen zusätzliche Software-Komponenten für den Ring 0 bzw. Kernel des Betriebssystems entwickelt werden. Software, die in dieser Ebene läuft, hat praktisch Zugriff auf alle Ressourcen des Rechners. Zu diesen Ressourcen gehören:

- beliebige Speicherzugriffe,
- sämtliche Systemroutinen,
- beliebige Port-Zugriffe,
- Registry-Zugriffe,
- sämtliche Synchronisationsobjekte und -Mechanismen,
- Interrupt-Verarbeitung,
- Adapterspeicher integrierter Karten,
- DMA-Transfers.

Die Kommunikation mit der im Rechner integrierten Hardware wie Festplatten, Schnittstellen und Netzwerkkarten wird ausschließlich von den im Kernel Mode laufenden Gerätetreibern abgewickelt. Diese Gerätetreiber besitzen eine einheitliche Schnittstelle in den User Mode. Anwendungen, die auf Geräte zugreifen wollen, müssen sich ausnahmslos über diese Schnittstelle der Gerätetreiber mit den Geräten austauschen [Bak97].

4.5.1 Entwicklung eines Diagnose - Gerätetreibers

Um die Latenzzeiten innerhalb eines Gerätetreibers von Windows NT messen zu können, ist die Entwicklung eines eigenen Treibers erforderlich, der auf externe Ereignisse reagiert und dessen Antwortzeiten messbar sein müssen. Für die Treiberentwicklung bietet Microsoft das DDK (Device Driver Development Kit) an.

Im Rahmen dieser Arbeit wurde ein Gerätetreiber entwickelt, der solche Messungen ermöglicht.

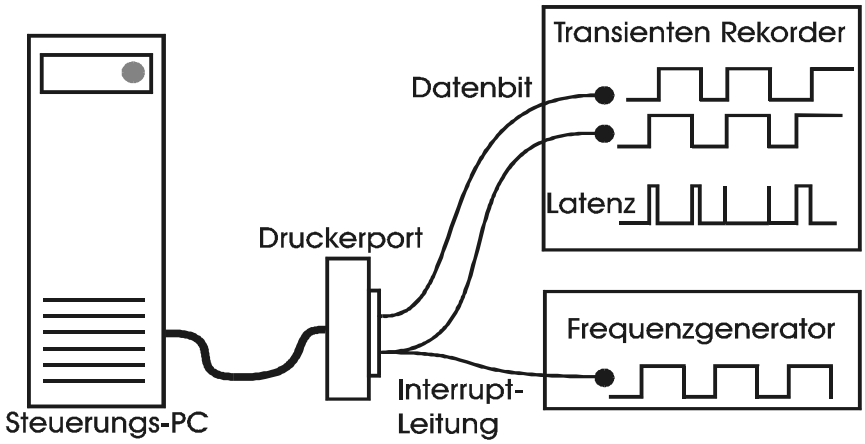


Abbildung 4.6: Messen der Interrupt-Latenzzeiten

In diesem Treiber wurde als Ereignisquelle die Signalleitung `STROBE` des Druckerports verwendet, so dass mittels einer externen Signalquelle auf dieser Leitung Ereignisse generiert werden konnten. `STROBE` hat die Eigenschaft, Interrupts auslösen zu können. Innerhalb der installierten Interrupt-Routine kann dann beliebig auf dieses Signal reagiert werden. In diesem Diagnose-Treiber wurde ein Ausgangsbit des Druckerports als Antwort auf die Interrupt-Anforderung invertiert. Auf diese Weise ist es möglich, mit externen Geräten wie Frequenzgenerator und Zeitmesseinrichtung die Latenzzeiten des Betriebssystems zu messen. Abbildung 4.6 stellt den Versuchsaufbau zum Messen der PC-Interrupt-Latenzzeiten schematisch dar.

4.5.2 System-Zeitverhalten bei Reaktionen auf Hardware-Ereignisse

Die Versuche fanden auf einem Pentium 166 mit 64 MB RAM statt und führten zu folgenden Ergebnissen:

- Die Interrupt-Latenzzeiten lagen bei maximal 50 μs, im Mittel bei 40 μs und liegen damit deutlich unter der kritischen Grenze bei etwa 1 ms.
- Wurde dem Interrupt ein sog. ”Deferred-Procedure-Call”[9] (DPC) zugeordnet, lagen die Latenzzeiten der ”Deferred-Procedure” bei maximal 50 ms, was für den hiesigen Verwendungszweck eindeutig ungeeignet langsam ist.

Bei den Messungen wurde der PC auf unterschiedlichste Weise durch andere Aktionen belastet. Zu diesen Belastungen gehörte das Starten und Ausführen von Anwendungen, intensiver Datentransfer via Netzwerk, Darstellung von aufwändigen 3D-Grafiken und Ausführung von ”Benchmark”-Programmen.

Die Messergebnisse bestätigen die Tatsache, dass die Ausführungszeiten anderer Interrupts des gleichen Prozessors den entscheidenden Einfluss auf diese Messergebnisse haben, jedoch nicht Anwendungen im ”User Mode”. Da eine Richtlinie bei der Entwicklung von Windows NT-Treibern vorgibt, möglichst wenig Rechenzeit innerhalb eines ”Interrupt-Handlers” zu verbringen und anstehende Datentransfers innerhalb von DPC-Routinen auszuführen, existieren innerhalb eines wohlkonfigurierten Systems keine Gerätetreiber, deren ”Interrupt-Handler” das erforderliche Zeitverhalten stören können. Dennoch existieren am Markt Geräte mit derartigen Treibern, da sie ihre Aufgabe nur innerhalb des Interrupts zufriedenstellend ausführen können. Ein populäreres, aber extrem schlechtes Beispiel ist der Parallelport Zip-Treiber[10], der den gesamten Datentransfer mit dem am Druckerport angeschlossenen Massenspeicher innerhalb des ”Interrupt-Handlers” abwickelt, das System bis zu hunderten von Millisekunden unterbricht und nur Bruchteile der zur Verfügung stehenden Rechenzeit dem ”User Mode” zur Verfügung stellt. Daher kann ein solches Gerät nicht in Verbindung mit einer Robotersteuerung auf demselben System betrieben werden.

Auf die grafische Darstellung der Messdaten der Untersuchungen kann hier verzichtet werden, da der einzige ausschlaggebende Wert die maximal aufgetretene Latenzzeit (50 μs) ist. Wie stark die Latenzzeit um einen Mittelwert rauscht, ist für diese Aufgabe nicht von Interesse.

Entscheidend ist die Erkenntnis, dass Windows NT mittels des ”Interrupt-Handlers” durchaus dazu in der Lage ist, echtzeitfähig auf Ereignisse zu reagieren. Genauere Einzelheiten zur Realisierung von Low-Level-Komponenten im Kernel Mode können dem Kapitel 7.4 entnommen werden.

[9] Als ”Deferred Procedure” bezeichnet Microsoft eine Kernelroutine, deren Ausführung aus einem Interrupt angestoßen wird. Die tatsächliche Zuteilung von CPU-Zeit erfolgt jedoch erst, wenn keine weiteren, höher priorisierten Routinen, wie etwa andere Interrupts oder andere ”deferred procedures”, mehr laufen.

[10] ”ZIP-Laufwerke” sind von der Fa. IOMEGA entwickelte, am Druckerport anschließbare Massenspeicher für Wechselmedien.

Kapitel 5

Gemeinsame Komponenten von VRC und R/VRC

Aus den im Kapitel 3 beschriebenen Zielen ergibt sich ein Gesamtkonzept, bei dem VRC und R/VRC weitgehend übereinstimmen und sich lediglich in den Erfordernissen zur Ansteuerung realer Hardware unterscheiden. Der erhebliche Aufwand einer vollständigen Neuentwicklung der R/VRC braucht nicht betrieben zu werden, da mit dem VRC-Modul bereits ein großer Teil an Basisfunktionalität zur Verfügung steht. Ausgehend von diesem Simulationsmodul bietet es sich an, eine bedingte Erweiterung dieses Moduls vorzunehmen, die dieses um die Erfordernisse der R/VRC ergänzt. Entwicklungstechnisch kann dieses umfangreiche Software-Projekt auf gemeinsamen Quellcode-Dateien beruhen und je nach Projekteinstellungen in der Entwicklungsumgebung eine Echtzeitsteuerung (R/VRC) oder eine Steuerungssimulation (VRC) erzeugt werden.

Eine endgültige Verschmelzung von VRC und R/VRC zu einer einzigen Software-Komponente ist nicht realisierbar, da die R/VRC stets auf der ”echten Zeit” beruhen muss, innerhalb des Simulationssystems zur korrekten Laufzeitsimulation allerdings eine ”simulierte Zeit” erforderlich ist, wie bereits in Kapitel 3.3 dargelegt wurde. Konzeptionell widersprechen sich diese Implementierungen, denn bei der reinen Simulation ist das Simulationssystem dafür verantwortlich, keine Zustandsänderung zu ”verpassen” und diese korrekt an sämtliche betroffenen, ebenfalls simulierten Komponenten weiterzuleiten. Unter Umständen kann für diesen Vorgang die ”echte Zeit” nicht ausreichen und die Simulation des Vorganges benötigt mehr Zeit, als der reale Vorgang gedauert hätte. Weil daher bei einem reinen Simulationssystem die Kontrolle der Steuerungsobjekte im Kern des Simulationssystems liegt und dieser die ”simulierten Zeitquanten” an die Steuerungsobjekte verteilen können muss, um stets korrekte Simulationsergebnisse liefern zu können, widerspricht dies dem Einsatz von Steuerungskomponenten mit eigenem Zeittakt, der auf der ”echten Zeit” basiert.

Dennoch können R/VRCs unter grafischen Simulationssystemen eingesetzt werden, wenn die Simulationen der Modelle weniger Zeit verbrauchen als die gleichen Vorgänge in der Realität. In diesen Fällen erhält das Simulationssystem genügend Zeit, alle relevanten Zustandsänderungen beim Betrieb der R/VRC zu verarbeiten. Schwierigkeiten treten erst dann auf, wenn z.B. mehrere Steuerungen[1] parallel betrieben werden und die (Weiter-)Verarbeitung derer Zustände das

[1] Hier sind nicht unbedingt Robotersteuerungen gemeint, sondern zum Beispiel auch SPSsen, Transportsysteme,

Simulationssystem überlasten.

In diesem Fall können wichtige Zustände ”verpasst” werden, etwa ein Impuls auf einem digitalen Ausgang, der eigentlich einen Vorgang in einer anderen Steuerungskomponente anstoßen sollte. Diese Szenarien müssen in der Simulation unbedingt vermieden werden, indem die Simulation in diesem Fall langsamer abläuft, als der gleiche Vorgang in der Realität.

Die Steuerung, die in dieser Arbeit beschrieben wird, kann deshalb sowohl als reine VRC als auch R/VRC aus den gemeinsamen Quellcode-Dateien erzeugt werden. Diese Methode hat den Vorteil, dass bei Änderungen in gemeinsamen Komponenten automatisch beide Systeme modifiziert werden und nicht zwei Software-Projekte getrennt voneinander gepflegt werden müssen. Zudem kann die grundsätzliche Struktur der Anbindung an das Simulationssystem beibehalten werden, wobei das Simulationssystem die Steuerung regelmäßig aufruft und dabei sowohl beim Eintritt in das Modul als auch beim Verlassen Zustandsdaten austauscht.

Beim Betrieb als VRC (Abbildung 5.1) wird bei jedem Aufruf ein Steuerungstakt ausgeführt.

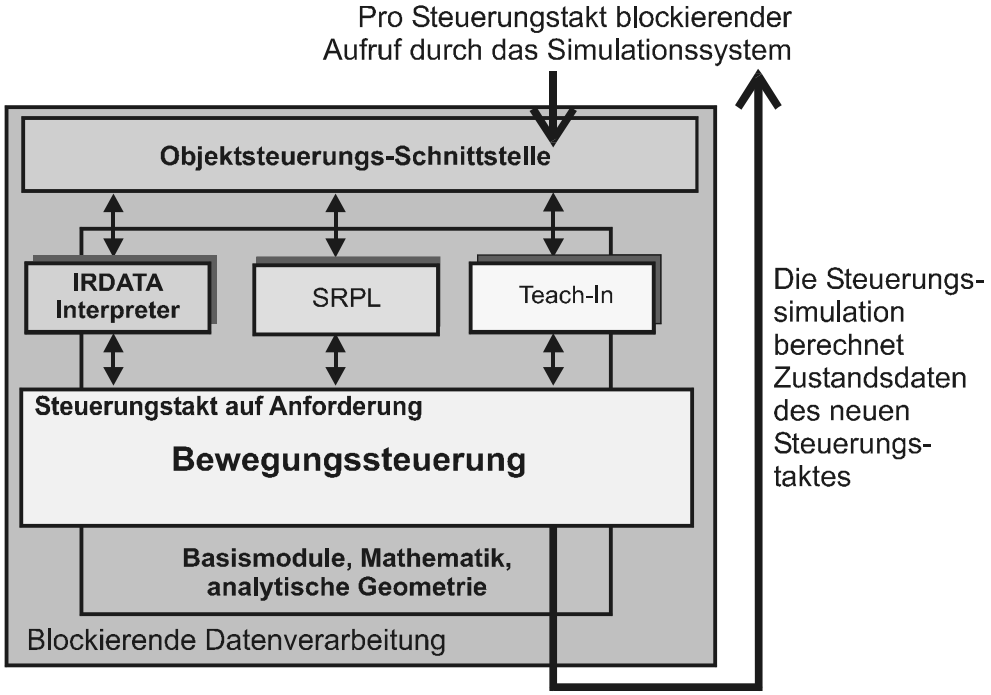


Abbildung 5.1: Betrieb der VRC

Findet kein Aufruf des Moduls statt, so werden auch keinerlei Daten innerhalb des Simulationsmoduls verarbeitet, und es kann keine Zustandsänderung innerhalb der VRC auftreten. Zu den Aktionen, die während eines Aufrufes stattfinden gehören:

- Interpretation von IRDATA-Zeilen,

Zellensteuerungen.

- Ausführung eines SRPL[2]-Programmschrittes,
- Bearbeitung einer Teach-In[3] Anforderung,
- Ausführung eines Interpolationstaktes.

Dabei stehen IRDATA, SRPL und Teach-In in Konkurrenz, d.h. nur eine dieser Komponenten kann z.Z. aktiv sein, jedoch kann/muss parallel zu diesen Komponenten ein (Bahn-) Interpolationstakt ausgerechnet werden. Mit diesem Systemkonzept der VRC kann das Simulationssystem die Ausführung von Bearbeitungsschritten in jeder geladenen Instanz der VRC genau kontrollieren, da nur beim Aufruf eine Datenverarbeitung innerhalb der VRC stattfindet. Nur so ist gewährleistet, dass jede Zustandsänderung verarbeitet wird.

Als reale Steuerung bzw. als R/VRC (Abbildung 5.2) ist die Bewegungssteuerung von dem Prinzip des "Aufgerufenwerdens" des Simulationssystems getrennt und läuft asynchron zum Simulationssystem weiter, so dass auch bei ausbleibenden Aufrufen Achswinkel generiert werden oder andere Zustandsänderungen eintreten können, die für das Simulationssystem und andere angeschlossene Hardware-Komponenten wichtig sind.

Zustandsänderungen werden bis zum nächsten Aufruf für das Simulationssystem zwischengespeichert und danach konzentriert zurückgemeldet.

Ausgehend von diesen beiden Systemstrukturen gewinnt der Titel dieses Kapitels an Bedeutung, denn beim Vergleich der beiden Abbildungen 5.1 und 5.2 fallen die wichtigen gemeinsamen Komponenten sofort auf:

1. PCROB-Objektsteuerungsschnittstelle,
2. IRDATA-Interpreter,
3. Bewegungssteuerung (incl. Kinematikmodulen),
4. SRPL-Interpreter,
5. Teach-In-Umgebung,
6. Basismodule mit den erforderlichen mathematischen Funktionen.

Auf die Punkte 1. bis 3. wird in den nächsten Kapiteln genauer eingegangen, während die Punkte 4. bis 6. für diese Arbeit nur eine untergeordnete Rolle spielen.

[2]SRPL = Simple Robot Programming Language, eine sehr einfache Roboterprogrammiersprache mit einfachen Bewegungsanweisungen und der Möglichkeit zum Setzen, Abfragen und Warten auf digitale und analoge Ein- bzw. Ausgabekanäle, wurde definiert in [dV95].

[3]Beim Teach-In bedient der Anwender Verfahrtasten am Handbediengerät (PHG) oder auf dem Bildschirm, um die Kinematik zu bewegen.

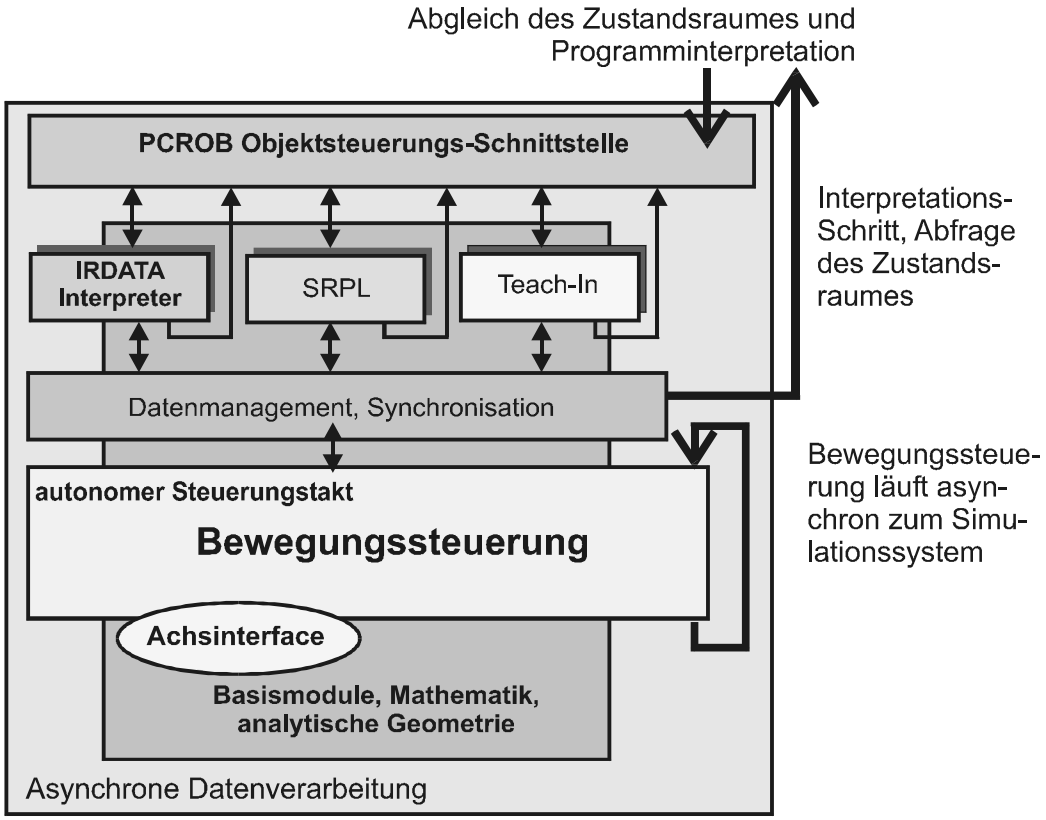


Abbildung 5.2: Systemstruktur als Steuerung

5.1 Integration in das Simulationssystem COSIMIR

Da bei der Realisierung des Steuerungsmoduls auf bestehende Komponenten der Steuerungssimulation zurückgegriffen werden kann, sollte auch die Schnittstelle zwischen dem Simulationssystem COSIMIR und der Steuerung nur so gering wie möglich von Änderungen betroffen sein. Dadurch soll der problemlose Austausch einer Steuerungssimulation gegen eine Echtzeit-Steuerung gewährleistet sein, ohne dass am Simulationssystem Änderungen vorgenommen werden müssen.

5.2 Steuerungsschnittstelle

Die Schnittstelle zwischen Simulationssystem und R/VRC wird in [Hyp94] ausführlich erklärt. Hier werden nur die wichtigsten Eigenschaften erläutert.

Die Steuerungsschnittstelle wurde entwickelt, um innerhalb des Simulationssystems verschiedene Steuerungen ansprechen zu können. Bei der Auslegung des Funktionsumfanges wurde großer Wert auf eine hohe Flexibilität gelegt, so dass auch andere Steuerungen angebunden werden können. Eine solche Steuerung wird bei Bedarf als dynamisch ladbare Bibliothek in das Simulationssystem eingebunden. Grundsätzlich verfügen diese Steuerungen über drei Funktionen, die vom Simulationssystem aufgerufen werden können.

5.2.1 Initialisierung der Steuerung

Zur Initialisierung einer Instanz der Steuerung wird eine Funktion aufgerufen, mit der interne Datenstrukturen angelegt und initialisiert werden. Ggf. werden weitere Bibliotheken geladen und notwendige Hardware für den Betrieb vorbereitet.

In der VRC bzw. R/VRC werden hier der IRDATA-Interpreter und die unterlagerte Bewegungssteuerung geladen und gestartet. In der Echtzeitsteuerung wird bei einer existenten Treiberstufe zur Ansteuerung einer Kinematik diese ebenfalls initialisiert, ohne sie jedoch an dieser Stelle bereits zu parametrieren.

5.2.2 Deinitialisierung der Steuerung

Wird eine initialisierte Steuerung nicht mehr für Simulationsvorgänge benötigt, so wird sie zur Wahrung der Systemkonsistenz durch den Aufruf einer Deinitialisierungsfunktion entladen. Hierbei wird allozierter Speicher freigegeben, verwendete Netzwerkverbindungen, Dateien, Verbindungen zu Hardware usw. werden getrennt bzw. geschlossen.

Bestand eine Verbindung zu einer Kinematik-Hardware, wird diese in den ausgeschalteten Zustand versetzt und in einem definierten Zustand von der Steuerung getrennt.

5.2.3 Aufruf des Steuerungsmoduls

Die tragende Kommunikationsfunktion einer in das Simulationssystem eingebundenen Steuerung stellt eine Prozedur dar, die Kommandos des Simulationssystems entgegennimmt, auswertet und an untergeordnete Strukturen weiterleitet. Am Ende eines Steuerungstaktes, wenn die Kontrolle wieder dem Simulationssystem übergeben wird, sendet die Steuerung sowohl angeforderte als auch unangeforderte Zustandsdaten zurück an das Simulationssystem. Diese von nun an als Steuerungsaufruf bezeichnete Prozedur ist der einzige Eintrittspunkt in die Steuerung. Sie wird während des gesamten Programmlaufes verwendet. Sämtliche Daten, die hier bidirektional ausgetauscht werden, unterliegen einer exakt beschriebenen Syntax und Semantik, die in [Hyp94] beschrieben sind. Die Kommandos des Simulationssystems lassen sich in folgende Klassen gliedern und umfassen insgesamt etwa 75 Funktionen:

1. **Programmfunktionen**
   Mit diesen Kommandos wird die Steuerung dazu veranlasst, Programme und Daten vom Datenträger oder vom Netz zu laden.

2. **SRPL-Funktionen**
   Diese Anweisungen betreffen den ”Simple Robot Programming Language”-Interpreter der Steuerung, mit dem sich auf sehr einfache Weise Bewegungsabläufe zusammenstellen lassen. Mit diesen Funktionen lassen sich die einzelnen Programmschritte bearbeiten.

3. **Ablauffunktionen**
   Sie dienen dem Start, Stopp, Fortfahren, Abbrechen und Einzelschrittbearbeiten von geladenen Roboterprogrammen.

4. **Teach-Funktionen**
   Mittels dieser Funktionen kann der Roboter interaktiv in verschiedenen Koordinatensystemen bewegt werden.

5. **Ein-/Ausgabefunktionen**
   Setzen und Abfragen von I/O-Konfiguration, Ein-/Ausgängen, Benutzerein-/ausgaben.

6. **Zustandsfunktionen**
   Austausch von Position, Geschwindigkeit und Zeit.

7. **Kinematikfunktionen**
   Definition von Kinematiken wie Robotern und Zusatzgeräten.

8. **Unaufgeforderte Meldungen der Steuerung**
   Diese Meldungen der Robotersteuerung beinhalten Informationen, die dem Simulationssystem zur Verfügung gestellt werden müssen, ohne dass diese ständig nachgefragt werden, z.B. wenn ein aktuell bearbeitetes Roboterprogramm einen digitalen Ausgang setzt.

Mit den Funktionen der Klasse Kinematikfunktionen wird die Kinematik der Steuerung direkt im Anschluss an ihre Initialisierung ausgewählt und parametriert. Erst danach kann die Steuerung vom Anwender als Robotersteuerung verwendet werden. Im Kapitel 3.5 wurde bereits auf die Notwendigkeit einer solchen Parametrierung hingewiesen. Mit diesen Funktionen wird nun die gewünschte Kinematikklasse ausgewählt und so parametriert, dass sich die Steuerung im Rahmen der erreichbaren Genauigkeiten so wie ihr reales Pendant verhält.

5.3 Erweiterter IRDATA-Interpreter

Eine umfassende Robotersteuerungssimulation, mit der verschiedene moderne Robotersysteme simuliert werden sollen, benötigt nicht nur eine parametrierbare Kinematikbibliothek und eine universelle Bewegungssteuerung, sondern auch mindestens einen leistungsfähigen Programminterpreter, auf dessen Sprachelemente alle in den jeweiligen Hochsprachen implementierten Methoden funktional abgebildet werden können. Um den Anforderungen aus Kapitel 3.6 gerecht zu werden, bieten sich konzeptionell die in den nächsten Kapiteln erläuterten Realisierungsformen an.

5.3.1 Einheitliche Simulationssprache

Eine bei verschiedenen kommerziellen Simulationssystemen angewandte Möglichkeit, um verschiedene Robotersysteme zu unterstützen, besteht darin, innerhalb der Simulation *eine einheitliche* Programmiersprache zur Verfügung zu stellen. Mit dieser einheitlichen Sprache werden sämtliche vom System unterstützte Roboter programmiert. Nach erfolgreicher Simulation wird diese Sprache mittels systemspezifischer Postprozessoren in die Zielsprache übersetzt. Abbildung 5.3 stellt diese Struktur dar.

Ein entscheidender Nachteil dieses Aufbaus besteht darin, dass der Informationsfluss nur vom Simulationssystem in die Robotersteuerung erfolgen kann, nicht aber von der Robotersteuerung

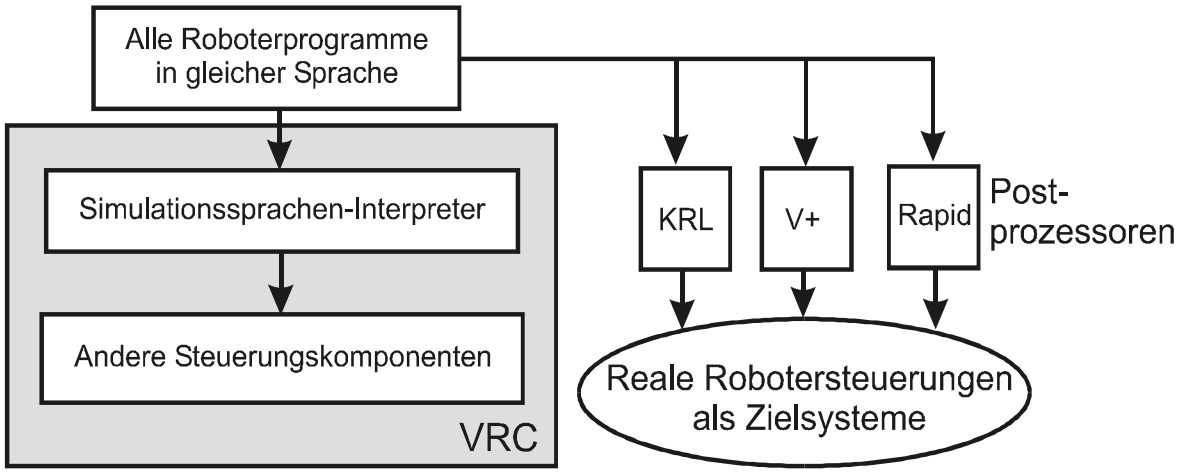


Abbildung 5.3: Einheitliche Simulationssprache wird in Zielsprachen übersetzt

in das Simulationssystem. Das bedeutet, dass auf einem Zielsystem vorhandene Roboterprogramme nicht ohne weitere Umsetzung in das Simulationssystem übertragen werden können, um dort etwa optimiert oder weiter entwickelt zu werden. Der Anwender dieser Simulationssysteme muss in diesem Fall bestehende Roboterprogramme zunächst manuell in die Simulationssprache übersetzen. Ferner kann eine Programmpflege ebenfalls nur in der Simulationssprache erfolgen, da bei Änderung des automatisch generierten Programmtextes kein Abgleich mit dem Roboterprogramm in der Simulationsumgebung stattfinden kann. Häufig sind jedoch gerade auf Ebene des ”Shop-Floor” in der Fertigung noch Korrekturen am Roboterprogramm erforderlich. Genau dort ist aber im Allgemeinen weder das Simulationssystem noch das ”Know-How” für dessen Bedienung verfügbar, um das Roboterprogramm in der Simulationssprache zu korrigieren und neu in die Zielsprache übersetzen zu können. Daher spaltet sich die Entwicklung des Roboterprogramms nach der Übertragung in die reale Robotersteuerung oft irreversibel vom Simulationssystem ab.

Dieses Konzept hat seine Vorteile deshalb bei der Anlagen- und Layoutplanung. Ein nützliches Werkzeug für die Roboterprogrammentwicklung ist es daher definitiv nicht. Diesen Umstand könnten Cross-Recompiler-Programme aufwiegen, mit denen die Roboterhochsprachen der Zielsysteme in die Simulationssprache übersetzt werden können. Jedoch beschränkt sich der funktionale Umfang der Simulationssprachen bisher auf eine Untermenge der Funktionalitäten von Roboterhochsprachen, so dass dieser Vorgang nicht zuverlässig funktionieren kann. Daher scheidet dieses Konzept für die in dieser Arbeit beschriebene R/VRC aus. Siehe hierzu auch [Rob92a] und [DH91].

5.3.2 Unterstützung durch sprachspezifische Interpreter

Sollen innerhalb der R/VRC *verschiedene* Roboterhochsprachen unterstützt werden, so kann der Weg verfolgt werden, für jede Sprache einen eigenen Interpreter zur Verfügung zu stellen, der genau diese Roboterhochsprache laden und verarbeiten kann. Jeder dieser Interpreter verfügt über einheitliche Schnittstellen zur Bedienoberfläche und zur Bewegungssteuerung, so dass ein modularer Aufbau möglich ist. Abbildung 5.4 stellt diese Struktur im linken Zweig dar.

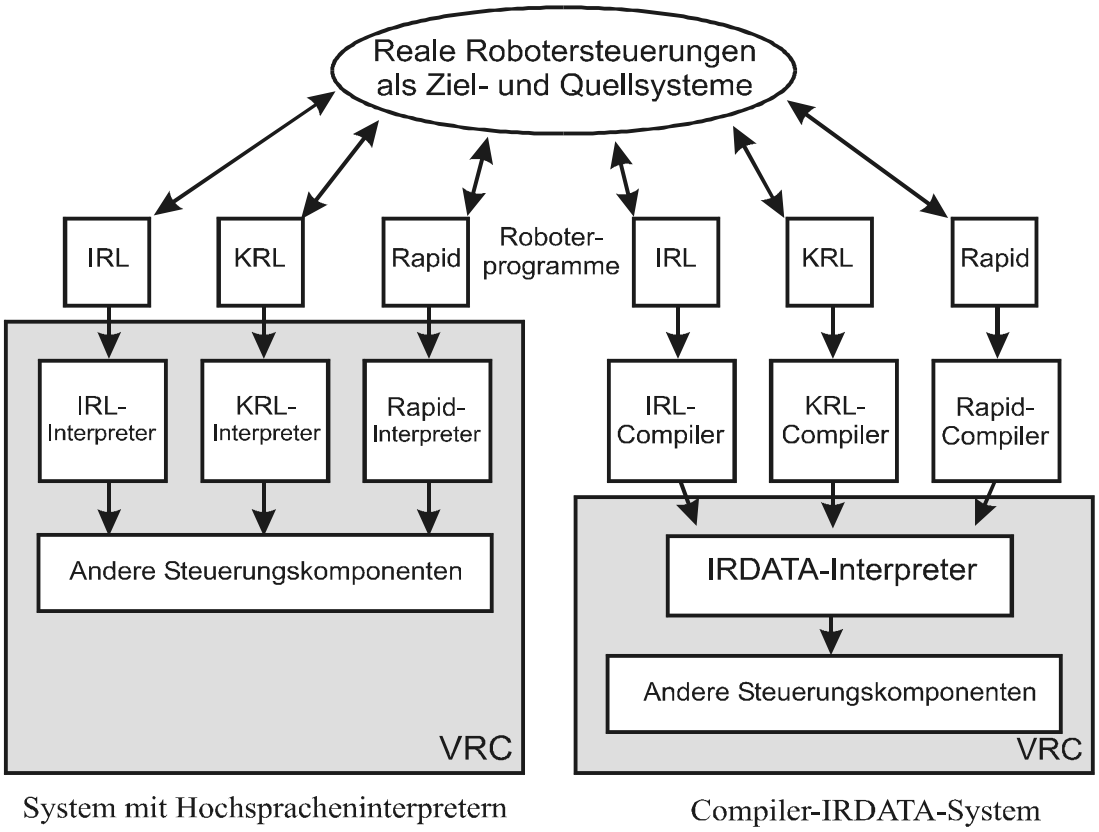


Abbildung 5.4: Unterstützung verschiedener Hochsprachen

Ein solches System weist folgende Vorteile auf:

- Die Systemstruktur kann modular aufgebaut werden, so dass eine Bibliothek von Interpretern ein eigenständiges, erweiterbares Subsystem darstellt.
- Innerhalb eines Interpreters kann individuell auf spezielle Merkmale der Sprache Rücksicht genommen werden.
- Es ist die Verwendung von Interpretermodulen des Roboterherstellers denkbar, so dass der Programmfluss und das Zeitverhalten exakt mit der realen Robotersteuerung übereinstimmt.

Nachteile des Systems:

- Der Aufwand zur Entwicklung eines neuen Interpreters ist hoch.
- Abläufe, die den Funktionsumfang des Systems demonstrieren sollen, erfordern den gesamten Funktionsumfang der Implementierung des Interpreters.
- Bei Änderung der Schnittstellen zur Oberfläche oder zur Bewegungssteuerung müssen je nach Systemdesign alle vorhandenen Interpreter angepasst werden.

- Die Schnittstelle zwischen den Interpreter-Modulen und dem Gesamtsystem ist sehr aufwändig und muss in jedem Interpretermodul implementiert werden, so dass sich insgesamt ein großer ”Overhead” ergibt.

5.3.3 Unterstützung durch Compiler – Interpreter System

Ein weiteres Konzept zur Unterstützung *verschiedener* Roboterhochsprachen beruht auf einem kombinierten System aus Compilern und einem einzigen, leistungsfähigen Interpretersystem. Abbildung 5.4 stellt diese Struktur im rechten Zweig dar. Dabei wird die zu unterstützende Roboterhochsprache mittels eines korrespondierenden Compilers in einen Zwischencode übersetzt, den die Steuerung verarbeiten kann. Bei diesem Zwischencode wird auf den vom Normungsausschuss Maschinenbau verabschiedeten Standard IRDATA zurückgegriffen [fNe94b] und dieser entsprechend erweitert. Diese Art der Unterstützung verschiedenster Roboterhochsprachen weist folgende Vorteile auf:

- Compiler sind eigenständige Systeme, die als Zugabe zum Gesamtsystem und auch autonom funktionieren. Sie müssen nicht derart komplex wie Hochspracheninterpreter mit dem Gesamtsystem verzahnt werden, sondern werden bei Bedarf vom Simulationssystem gestartet und übersetzen dann die Eingabedateien in den IRDATA-Code. Dadurch entfallen aufwändige systeminterne Schnittstellen.

- Einmal erfolgreich übersetzte Programme sind auch ohne Compiler auf der R/VRC lauffähig. Erforderlich ist dazu lediglich der ”Maschinencode”, das IRDATA-Programm.

- Da die Zielsprache stets IRDATA ist, können durch Einsatz von Zielsprachenbibliotheken und effizienten Compiler-Werkzeugen wesentliche Komponenten der einzelnen Compiler gemeinsam genutzt werden, etwa IRDATA-Bibliotheken und Linker (siehe [FLRSK01]).

- Bei Erweiterungen des IRDATA-Interpreters bleiben alle bis zu diesem Zeitpunkt generierten IRDATA-Programme lauffähig.

- Bei Erweiterungen des IRDATA-Interpreters bleiben alle bis zu diesem Zeitpunkt entwickelten Compiler uneingeschränkt verwendbar[4].

- Die Programmentwicklung findet in der Sprache des Zielsystems statt. Dadurch können Programme auf einfache Weise sowohl von den realen Robotersteuerungen importiert als auch dorthin exportiert werden.

Die Entwicklung der Compiler ist nicht Bestandteil dieser Arbeit. Im Rahmen der Entwicklung neuer Compiler finden permanent Erweiterungen des IRDATA-Interpreter-Systems der Steuerung statt. Die Erweiterungen des IRDATA-Interpreters sind Bestandteil dieser Arbeit.

Mit den Erweiterungen lässt sich beinahe jede relevante Roboterhochsprache funktional auf die Sprache des Interpreters abbilden. Auf die Basisfunktionalitäten eines Interpreters, zu denen ein

[4] Diese Abwärtskompatibilität ist für Aktualisierungen ausgelieferter Systeme erforderlich.

Speichermodell, Stacks, arithmetische, trigonometrische, logische und Zeichenkettenoperationen gehören, wird hier nicht eingegangen, sondern auf die Anforderungen, die aus den verschiedenen Quell-Sprachen und deren speziellen Elementen resultieren. Spezielle Sprachelemente sind z.B. Anweisungen zur

- Spezifikation von Bewegungen,
- Steuerung von Multitasking,
- Synchronisation von Programm, Tasks und Roboter,
- Kommunikation mit der Umgebung,
- Verarbeitung von roboterspezifischen Datentypen,
- Definition von Ereignissen,
- Definition von Reaktionen auf Ereignisse,
- Ansteuerung von Schnittstellen und Ports,
- Abfrage und Beeinflussung von systeminternen Daten.

Der Interpreter muss für diese speziellen Elemente verallgemeinerte Methoden zur Verfügung stellen, so dass etwa die Definition eines ”Triggers” in der Hochsprache KRL [KUK00] genau wie die Definition eines Interrupts in MELFA Basic [Ele00] auf ein allgemeines Ereignis-Konzept abgebildet werden kann. Siehe hierzu auch Kapitel 5.4.

5.3.4 Struktur des IRDATA-Interpreters

Abbildung 5.5 stellt die Schnittstellen und Komponenten innerhalb und außerhalb des IRDATA-Interpreters dar. Kernelement ist ein Modul, das als Ausführungseinheit bezeichnet wird. Dieses Modul kann in mehreren Instanzen betrieben werden, es ist für die Interpretation der Programmzeilen verantwortlich und hat über die definierten Schnittstellen Zugriff auf die umgebenden Komponenten. Innerhalb einer Ausführungseinheit übernimmt der IRDATA-Kernel das Scheduling zwischen den gestarteten Ausführungseinheiten und den IRDATA-Tasks innerhalb der Ausführungseinheiten. Mit Hilfe des IRDATA-Kernels können mehrere Anweisungsinterpreter innerhalb einer Ausführungseinheit gleichzeitig an verschiedenen Stellen eines IRDATA-Programms unabhängig voneinander Code bearbeiten. Die Anweisungsinterpreter sind in der Grafik als Threads bezeichnet. Diese Struktur ist erforderlich, um die verschiedenen Multitasking-Konzepte der Roboterhochsprachen auf IRDATA abbilden zu können.

Vor dem Programmstart wird das Roboterprogramm in den Programmspeicher geladen, wobei gleichzeitig verschiedene Vorverarbeitungen stattfinden, wie z.B. das ”Hashen” von Sprungmarken (Labels), damit bei Ablauf des Programms die Sprungzeiten minimiert werden. Der Programmspeicher ist von jeder Ausführungseinheit, und damit auch von jedem Anweisungsinterpreter, lesbar. Wird Multitasking verwendet, so arbeitet jede aktive Ausführungseinheit i.A. an einer anderen Stelle des IRDATA-Codes.

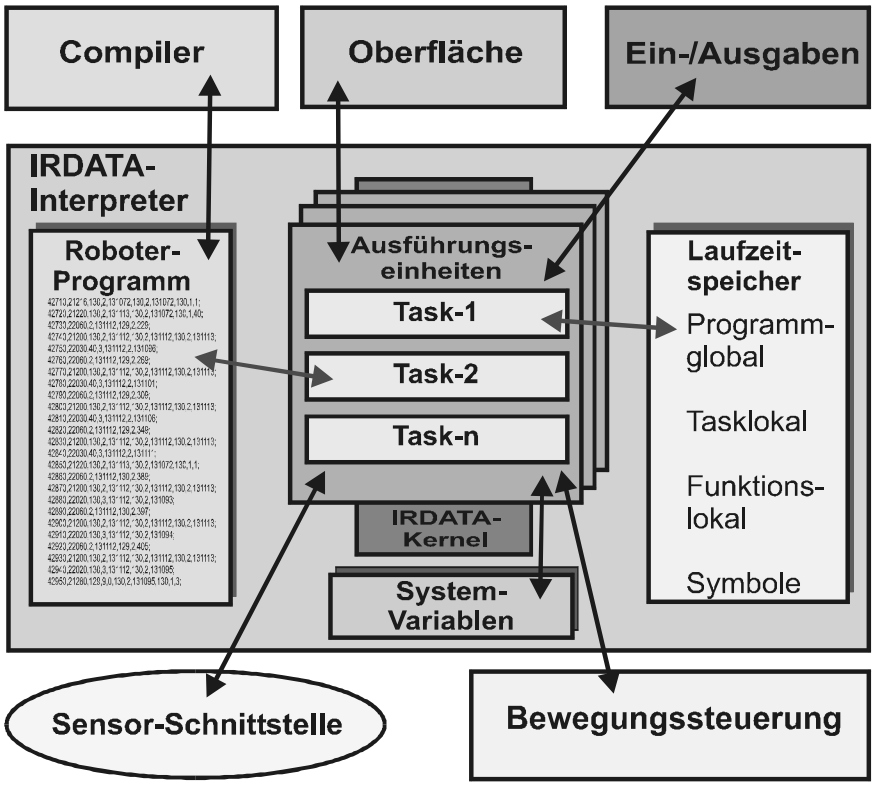


Abbildung 5.5: Einbettung des IRDATA-Interpreters in die Steuerung

Als Laufzeitspeicher wird der Arbeitsdatenspeicher der IRDATA-Programme bezeichnet. Dieser wird bei Programmstart initialisiert und enthält noch keine Daten. Vorgänge, die den Laufzeitspeicher betreffen, geschehen ausschließlich durch IRDATA-Speicherzugriffe mittels spezieller Zugriffsfunktionen. Der Laufzeitspeicher gliedert sich in folgende Komponenten:

- globale Daten, die von der IRDATA-Haupttask[5] angelegt werden,
- tasklokale Daten, die innerhalb von weiteren Tasks angelegt werden,
- funktionslokale Bereiche innerhalb der tasklokalen Daten,
- symbolische Variablen, die vom IRDATA-Programm mittels Zeichenketten im Klartext adressiert werden können,
- Stapelspeicher (Stacks) innerhalb der tasklokalen Daten für Rücksprungadressen und Daten.

Abbildung 5.6 stellt diese Struktur dar.

[5] Mit der Haupttask beginnt der Programmablauf.

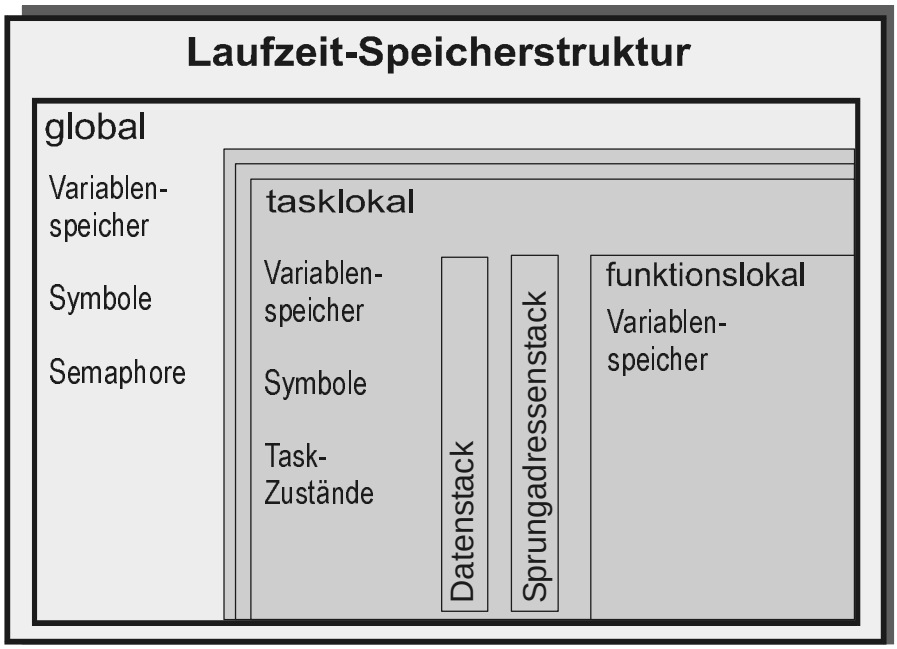


Abbildung 5.6: IRDATA-Laufzeitspeicher

5.3.5 Parametrierung der Steuerung mittels IRDATA-Code

Neben den globalen Parameterdaten, mit denen die Steuerung während des Ladens einer Arbeitszelle parametriert wird, und die im Wesentlichen die Kinematik und die Bewegungssteuerung spezifizieren, existieren auch roboterhochsprachenspezifische Parameterdaten, die auf unterschiedliche Konventionen in den Definitionen der Roboterhochsprachen basieren. Zu diesen Daten gehören u.a.:

- **Einheiten**, die in den Sprachen verwendet werden (z.B. mm oder m, Bogenmaß oder Grad) Das IRDATA-Programm kann dem Interpreter die bevorzugte Einheit mitteilen, so dass kein IRDATA-Code zur Konvertierung erzeugt werden muss.

- **Das Format der Achsdarstellung**.
  Einige Roboterhersteller stellen die Achswinkel im Anwenderprogramm anders dar, als es in der Bewegungssteuerung erforderlich ist, um im Format kompatibel zu anderen Robotern der Serie zu sein. So kann es vorkommen, dass bei Fünf-Achsen-Knickarmrobotern, stets sechs Achswinkel angegeben werden, dann aber der vierte Wert immer Null ist. (z.B. Mitsubishi RV-2AJ)[6]. Diese Darstellung muss für die Bewegungssteuerung konvertiert werden. Zur Umrechnung der Darstellung können mittels IRDATA-Programm lineare Transformationen definiert werden, auf die in Kapitel 5.5.5 genauer eingegangen wird.

- **Bremsfaktoren** (Overrides) für Geschwindigkeiten und Beschleunigungen bei Programmstart.
  Bei verschiedenen Robotersprachen ist die Voreinstellung der Overrides nicht 100%.

[6]Diese Darstellung liegt sogar nahe, da die Kinematik des fünfachsigen Roboters dem eines sechsachsigen mit blockierter vierte Achse gleicht.

- **Verschleifradien** bei Programmstart.
  Je nach Roboterhochsprache werden verschiedene Verschleifkriterien verwendet.

- **Verfahren zur Steuerung von Zusatzachsen**.
  Je nach Roboterhochsprache werden zusätzliche Achsen standardmäßig synchron oder asynchron zu Roboterbewegungen, einzeln oder in Gruppen interpoliert. Die Voreinstellung der Robotersteuerung kann mittels IRDATA-Anweisungen überschrieben werden.

- **Format der Beschleunigungsangabe**.
  Bei einigen Roboterhochsprachen wird die Beschleunigung nicht als $\frac{Weg}{Zeit^2}$ spezifiziert, sondern als Beschleunigungszeit, also wie lange es dauern soll, bis die Plateaugeschwindigkeit erreicht sein soll. Auch Voreinstellungen zum Verschleifen von Bewegungen sind abhängig von der Roboterprogrammiersprache.

- **Auflösung von Achsstellungsmehrdeutigkeiten**.
  Achsstellungsmehrdeutigkeiten werden durch Daten aufgelöst, die zusätzlich zur kartesischen Position angegeben werden und je nach Roboterhochsprache verschieden vorinitialisiert sind. Zu diesen Daten gehören die Spezifikation der kinematischen Konfiguration einer kartesisch gegebenen Zielposition und die Auflösung von Redundanzen, die durch mehrfach rotationsfähige (Multiturn-)Achsen entstehen. Die Roboterhersteller verwenden teilweise sehr ungewöhnliche Darstellungsformen für diese zusätzlichen Strukturdaten. So ändert sich der Turn-Zähler bei Rapid-Programmen (ABB) alle 90°, wobei mit diesen Turn-Angaben auch die strukturellen Stellungsmehrdeutigkeiten aufgelöst werden. Sind diese Daten bei einer kartesischen Bewegung angegeben, darf sich diese Achse dann nicht um mehr als 90° drehen. Bei MELFA Basic (Mitsubishi) erhöht sich der Turn-Zähler hingegen ab 180°, und dann alle 360° wieder. (-541° liefert als Turn-Angabe dann -2). Bei KRL (Kuka) existiert lediglich ein Turn-Flag, das anzeigt, ob die betreffende Achse einen negativen Winkel in der Stellung aufweist. Mittels adaptiven Konvertierungsmechanismen der Robotersteuerung können diese Daten auf eine einheitliche Darstellungsform zurückgeführt und verarbeitet werden.

- **Definition des Vorlaufs**.
  Wenn Bewegungen verschliffen werden sollen, dann muss spätestens zu dem Zeitpunkt, an dem die Bahn in Richtung der anschließenden Bahn verlassen werden soll, die neue Bewegung bekannt sein. Daher ist es erforderlich, dass der Interpreter auch während oder vor der Bewegungsinterpolation nach der anschließenden Bewegungsanweisung sucht. Dieses als Vorlauf bezeichnete parallel zur Bewegung ausgeführte Bearbeiten des Programms soll aber nicht unter allen Umständen erfolgen, da es abhängig von der Quellsprache ist, welche Anweisungen nicht vorlauffähig sind. So sind Anweisungen zum Setzen von digitalen Ausgängen nur in einigen Sprachen während Bewegungen möglich, bei den anderen wird die Programminterpretation so lange angehalten, bis der Roboter die Zielposition erreicht hat; in diesem Falle wird dann natürlich auch nicht verschliffen. Mit speziellen IRDATA-Anweisungen kann jede einzelne IRDATA-Funktion als vorlauffähig oder nicht spezifiziert werden. Relevante Anweisungen sind hier z.B. digitale und analoge I/O, Datei I/O, Benutzer I/O, Verzögerungen (Delays), Verstellen von Werkzeug- oder Basis-Versatz o.Ä., Definition von Interrupts, Starten von Tasks und Spezifikation von Sensorfunktionen.

Mit den neuen IRDATA-Anweisungen zur Spezifikation diverser Voreinstellungen vereinfacht sich ferner der Compilerbau erheblich, so dass die Entwicklungszeiten deutlich verringert werden können. Ferner wird mit der Möglichkeit, derartige Parameter überhaupt aus der Sprache heraus einstellen zu können, eine höhere Flexibilität bei der Unterstützung verschiedener Hochsprachen erreicht. Auf diese Weise können der IRDATA-Interpreter und die Bewegungssteuerung den Bedürfnissen der Sprache und des Zielsystems sprachabhängig angepasst werden. So verfügt der Mitsubishi RV-2AJ Roboter, programmiert mit IRL, über ”echte” fünf Achsen, ohne ”Dummy”-Achse. Die Aufnahme derartiger Spezifikationen und Parameter in das allgemeine Modell zur Beschreibung des Roboters würde daher die Möglichkeiten der sprachunabhängigen Verwendung von Robotern stark einschränken.

5.3.6 Unterstützung von kooperativem Multitasking

Wie bereits erwähnt, stellen viele moderne Roboterhochsprachen Multitasking-Konzepte zur Verfügung, mittels derer der Anwender parallel zum Hauptprogramm nebenläufige Aktivitäten implementieren kann. Solche Mechanismen eignen sich sehr gut dazu, neben dem Bewegungsprogramm für den Roboter auch Zellensteuerungsaufgaben in die Robotersteuerung zu verlagern. Mittels digitaler oder analoger I/O und anderer Schnittstellen lassen sich z.B. weitere in der Arbeitszelle vorhandene Geräte ansteuern und synchronisieren. Teilweise kann mit dieser Methode sogar der Einsatz von SPSen entfallen. Zur Unterstützung solcher Mechanismen wurde ein kooperatives Multitasking-Konzept in die Robotersteuerungssimulation integriert [Kei94].

5.3.7 Synchronisierungsmechanismen

Bei der Verwendung von Multitasking innerhalb eines Roboterprogramms sind häufig Mechanismen gefragt, mit denen die Ausführung parallel laufender Interpreter untereinander oder mit Bewegungen synchronisiert werden kann. Für diese Mechanismen stellt die Robotersteuerung definierbare und System-Semaphore bzw. Ereignisse zur Verfügung, auf die innerhalb des Programms gewartet werden kann, oder deren Stati innerhalb von Programmen beeinflusst werden können. Auf die Verfahren und Funktionalitäten dieser grundlegenden Signalisierungsmechanismen wird hier nicht näher eingegangen, siehe [Kop97]. Ferner werden hier Steuerungs-Semaphore erläutert, die auf IRDATA-Ebene zur Verfügung gestellt werden:

- **Roboter-Bahnende**
  Wird gesetzt, wenn eine einzelne Bewegungsinterpolation beendet wurde, bzw. die Zielposition erreicht wurde, oder beim Verschleifen der Punkt der größten Annäherung an den Verschleifpunkt erreicht wird.

- **Roboter-Bahnbeginn**
  Wird gesetzt, wenn mit einer einzelne Roboter-Bewegungsinterpolation begonnen wird.

- **Roboter-Bewegungsende**
  Wird gesetzt, wenn ein ganzer Satz an verschliffenen Roboterbewegungen beendet wurde.

- **Roboter-Verschleifbeginn**
  Wird gesetzt, wenn eine Roboterbahn verlassen und mit einer Verschleifbahn begonnen wird.

- **Roboter-Verschleifende**
  Wird gesetzt, wenn eine Verschleifinterpolation beendet wird und auf die anschließende Bahn aufgesetzt wird.

- **Roboter-Verschleif-Mittelpunkt**
  Wird gesetzt, wenn der Punkt der größten Annäherung an den zu verschleifenden Punkt erreicht ist.

- **Achsgruppen-Bewegungsende**
  Wird gesetzt, wenn eine Gruppe asynchroner Achsgruppen die Bewegung beendet hat.

Mittels dieser Ereignisse kann der IRDATA-Programmfluss mit der Bewegungssteuerung synchronisiert werden. So kann etwa in dem Hauptprogramm auf das Bewegungsende eines Werkstückpositionierers gewartet werden, um den Roboter nicht schon mit der Bearbeitung des Werkstücks beginnen zu lassen, bevor es sich in der korrekten Lage befindet.

5.3.8 Ein- und Ausgabefunktionen

Eine wichtige Gruppe an IRDATA-Funktionen stellt die Klasse der Ein- und Ausgabefunktionen dar, mit der das IRDATA-Programm über die speziellen Kanäle in Kontakt mit anderen Komponenten der Anlage oder dessen Betreiber treten kann.

Benutzerein- und ausgaben

Innerhalb von Roboterprogrammen ist es möglich, textuelle Eingaben vom Benutzer zu erwarten oder textuelle Ausgaben an den Anwender auszugeben. Wird eine solche Interaktion ausgeführt, erscheint auf der Oberfläche eine Dialogbox, über die mit der Steuerung interagiert werden kann. Die Datentypen, die hierbei verarbeitet werden können, umfassen alle in IRDATA unterstützten Formate.

Digitale und analoge Ein- und Ausgaben

Zu der Gruppe der Hardware-Schnittstellen gehören die Signalkanäle, mit denen die Steuerung digitale und analoge Signale setzen und abfragen kann. Im Simulationsbetrieb werden sämtliche dieser Aktionen an das Simulationssystem weitergeleitet, da hier interne Signalverknüpfungen mit anderen Steuerungskomponenten konfiguriert sein können. In der Steuerungsversion zur Ansteuerung realer Hardware kann der Anwender mittels spezieller Dialoge konfigurieren, welcher Ein- und Ausgangskanal an die reale Hardware weitergeleitet werden soll und welche nur innerhalb des Simulationssystems verarbeitet werden sollen. Abbildung 5.7 veranschaulicht den Zusammenhang für die Eingänge der R/VRC.

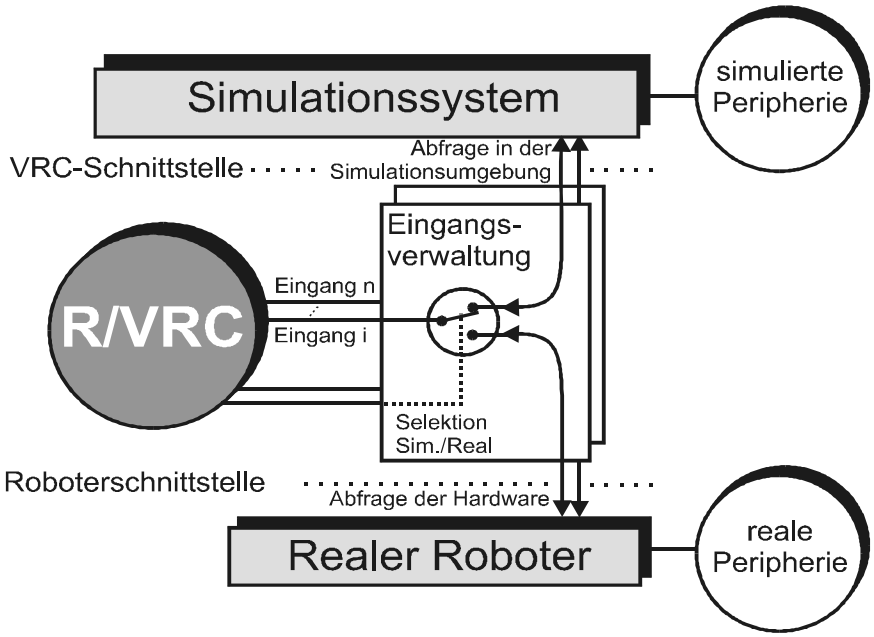


Abbildung 5.7: Konfiguration der Signalquelle für einen Eingang

Auf diese Weise ist ein Mischbetrieb aus simulierten und realen Komponenten möglich, was während der Realisierung einer Anlage sehr vorteilhaft sein kann, in der noch nicht alle Komponenten real in Betrieb genommen wurden. Dabei werden die noch nicht bereitstehenden Komponenten simuliert und der Anwender kann sich das Ergebnis seiner Arbeiten in der Simulationsumgebung anschauen, während die bereits betriebsbereiten Kommunikationspartner der Steuerung schon direkt über Hardware-Kanäle angesteuert werden können. Dieser Mischbetrieb gestattet Systemtests ohne vollständigen, realen Systemaufbau.

Dateiein- und -ausgaben

Auch wenn nur wenige Robotersteuerungen über ein dem PC-ähnliches Dateisystem verfügen, so ist es sehr vorteilhaft, wenn die VRC Dateiein- und -ausgaben tätigen kann. Die Implementierung dieser Schnittstelle ist so ausgelegt, dass alle IRDATA-Datentypen aus Dateien gelesen und geschrieben werden können. Ferner kann bei Verwendung von Gerätenamen als Dateinamen mit dieser Schnittstelle auch auf im System installierte Geräte zugegriffen werden [Cus93], [Bak97]. Zu diesen Geräten gehören standardmäßig serielle Schnittstelle, USB und Druckerport. Damit erschließt sich ein weiter Verwendungsbereich für Dateiein- und -ausgaben.

Schnittstellen Ein- und Ausgaben

Bei systemkonformer Anwendung der Dateiein- und ausgabe kann unter Windows NT durch Wahl des Dateinamens auf alle im System installierten ”File System Drivers” zugegriffen werden. Standardmäßig gehören zu diesen Treibern auch die parallelen und seriellen Schnittstellen,

so dass mit der hier vorgenommenen systemtransparenten Abbildung auf IRDATA auch alle installierten Schnittstellen des Host-PCs innerhalb des IRDATA-Interpreters zur Verfügung stehen. Werden Feldbussysteme wie Interbus oder Profibus unter NT auch als ”File System Drivers” in das System eingebunden, so steht deren uneingeschränkter Verwendung unter IRDATA nichts im Wege. Standardmäßig können auf diese Weise ”LPTx”, ”COMx”, Netzwerkressourcen wie Drucker, ”named Pipes” usw. angesprochen werden.

Persistente Datenlisten

Zur Trennung von Programmen und Daten wird in der Robotersteuerung ein Konzept zur Unterstützung persistenter Datenlisten verwendet. Das IRDATA-Programm kann bei Bedarf bestehende Datenlisten öffnen, und aus diesen Listen die dort enthaltenen Daten in interne Variablen laden. Als Datentypen werden alle in IRDATA vorhandenen Datentypen unterstützt. Programmgesteuert können diese Daten mit den Inhalten der Variablen abgeglichen werden, so dass beim nächsten Programmlauf die neuen Werte zur Verfügung stehen, und nicht in ”Vergessenheit” geraten sind. Persistente Datenlisten bieten folgende Verwendungsmöglichkeiten:

- **Positionslisten**. Die am häufigsten verwendeten Datenlisten werden als reine Positionslisten verwendet. Dabei werden Positionen des Roboters, die etwa durch Teach-In angefahren wurden, als Koordinaten in die Datenliste aufgenommen. Diese Daten werden bei Programmstart in Positionsvariablen übernommen und können ohne Weiterverarbeitung in Bewegungsanweisungen verwendet werden. Automatische Roboter-Programmgeneratoren z.B. zur Oberflächenbearbeitung verwenden Positionslisten, um die Bahnstützpunkte dort als Positionen abzulegen [Rok99].
- **Definition einer Aufgabenbeschreibung**. Eine Datenliste kann durch eine überlagerte Zellensteuerung vor dem Start des Roboterprogramms angelegt werden und darin eine Aufgabenbeschreibung mit den erforderlichen Parametern abgelegt sein, mit der spezifiziert wird, welcher vorimplementierte Teil des Roboterprogramms ausgeführt werden soll. Dieses Verfahren wurde im Projekt DEMON eingesetzt [TH99].
- **Spezifikation von Parametern der Robotersteuerung**. In Datenlisten können Systemvariablen abgelegt werden, die innerhalb des Roboterprogramms zur Verfügung stehen sollen.
- **Sicherung von Systemzuständen** der betriebenen Anlage, etwa an welchem Ort sich welche Teile in der Zelle befinden, welches Werkzeug am Roboterflansch befestigt ist oder Betriebsdaten, wie Laufzeiten der Steuerung und Statistiken zu Störungsfällen.

Multimedia-Erweiterungen

Um audiovisuelle Meldungen an den Operator der Steuerung ausgeben zu können, wurden spezielle IRDATA-Anweisungen implementiert, mit denen spezifizierte Audio- und Video-Dateien abgespielt werden können. Solche Ausgaben sind als Hinweise oder ausgefallene Signalisierungsmethoden sinnvoll, etwa wenn ungewöhnliche Betriebszustände in der Anlage aufgetreten sind. Z.B.:

- Verschiedene akustische Warnungen im Fehlerfalle, etwa wenn keine Teile mehr in einer Zuführeinheit vorhanden sind.
- Nachempfinden von Warnsignalen der simulierten Robotersteuerungssysteme, z.B. Signalton bei Mitsubishi-Controllern.
- Abspielen audiovisueller Anweisungen für den Werker vor Ort, etwa mit Bedienungshinweisen zur Anlage, z.B. ein Video mit dem Abtransport einer vollen und Anlieferung einer neuen, leeren Palette.

5.4 Ereignisinterpreter

Moderne Roboterhochsprachen bieten die Möglichkeit, auf unterschiedliche Ereignisse innerhalb des Roboterprogramms zu reagieren. Welche Ereignisse das sein können, ist sehr sprachspezifisch. In vielen Robotersteuerungen werden diese Ereignisse verschieden klassifiziert: als Interrupts, Timer, ausdruckgekoppelte Bedingungen oder Bahnschalt-Funktionen.

Beispiele für Interrupts:

- Wenn digitaler Eingang 4 auf den Zustand 1 wechselt, dann springe in Unterprogramm X.
- Wenn analoger Eingang 2 kleiner als 4.8 V wird, halte den Roboter und das Programm an und gebe eine Warnung aus.

Beispiele für Timer-Ereignisse:

- Springe nach 4.3 Sekunden in Unterprogramm Y.
- Starte alle 250 Millisekunden die Task Z.
- Beende in 20 Sekunden die Bewegung.

Beispiel für ausdruckgekoppelte Bedingungen:

- Wenn der Ausdruck $\sqrt[2]{(dx^2 + dy^2 + dz^2)} \langle Distance$, wahr wird, dann führe eine Fehlerbehandlung aus.
- Wenn der Wert eines Produktzählers größer als 100 und die Programmlaufzeit größer 2 Stunden ist, lege eine Pause von 30 Minuten ein.

Beispiele für Bahnschalt-Funktionen:

- 50 mm nach Bahnstart setze digitalen Ausgang 4 auf 1 und 50 mm vor Bewegungsende schalte ihn wieder aus.
- 0.54 Sekunden nach Bewegungsbeginn starte Task B.

- 134 mm nach Bahnstart warte noch 1.2 Sekunden, setze Ausgang 4 auf 0 und stoppe dann die Bewegung.

Werden gleichzeitig mehrere Ereignisse bearbeitet und tritt während der Ausführung einer Ereignisbehandlung ein weiteres Ereignis ein, muss zusätzlich eine Priorisierung Klarheit darüber schaffen, welche Behandlung nun Vorrang hat.

Diese vielfältigen Möglichkeiten der Ereignisse legen die Konzeption einer umfassenden Ereignisverwaltung nahe, mit der sämtliche auftretenden Arten und möglichst auch zukünftig auftretende neue Klassen von Ereignissen realisiert werden können.

Das hier realisierte Verfahren gliedert sich in die zwei Komponenten: ein speziell implementierter allgemeiner Ereignisinterpreter und die Einheit zur Programmflusskontrolle der Ausführungseinheiten, auf die in den nächsten Kapiteln genauer eingegangen wird.

5.4.1 Berechnung von Ausdrücken parallel zum Roboterprogramm und zur Roboterbewegung

Eine entscheidende Kenngröße, die bei der Konzeption des Systems zur Auswertung von Ausdrücken berücksichtigt werden muss, ist die Zeitdauer, die eine solche Auswertung in Anspruch nimmt. Reaktionen auf Bahnschaltereignisse z.B. betreffen häufig die Bahn selbst, etwa wenn während einer Bewegung Klebstoff aufgetragen wird, die Klebstoffzufuhr aber an einer Stelle kurz unterbrochen werden muss, weil sich hier eine Lücke in dem Werkstück befindet, in die kein Klebstoff gelangen darf. Bei solchen Anwendungsfällen muss der Ausdruck, der bestimmt, wann das Ereignis[7] eintritt, schnell berechnet werden. Auch die Reaktionszeit auf das Ereignis[8] sollte so kurz wie möglich sein. Solche Anforderungen verbieten die Auswertung der Ereignis-Ausdrücke innerhalb *gewöhnlicher* IRDATA-Tasks, die sich die gesamte Rechenzeit teilen und dem Scheduling des Interpreters unterliegen. Aufgrund des kooperativen IRDATA-Multitasking-Konzepts wäre nicht sichergestellt, dass der gesamte Ausdruck schnell genug ausgewertet würde, weil die Auswertung jederzeit von anderen Tasks unterbrochen werden könnte. Es ist vielmehr erforderlich, sämtliche definierten Ereignisse in jedem Steuerungstakt und nach jeder Ausführung einer IRDATA-Zeile zu überwachen. Der IRDATA-Interpreter, der den regulären Programmablauf bearbeitet ist hierfür ungeeignet, denn er meldet permanent seinen Status an das übergeordnete Simulationssystem, damit der Anwender Rückkopplung vom Programmablauf erhält. Die Ausführungseinheit dieses Interpreters ist eine komplexe Einheit, die mit anderen Komponenten des gesamten Systems protokollbasiert in Verbindung steht, wodurch keine parallele Bearbeitung möglich ist.

Es bieten sich an dieser Stelle zwei Lösungswege an:

1. Definition einer speziellen optimierten Ereignis-Definitions-Sprache und die Implementierung des korrespondierenden Ausdrucksinterpreters, der diese Art Makro-Sprache bearbeitet, in der die Bedingungen definiert werden. Dieser Ausdrucksinterpreter könnte parallel zum IRDATA-Interpreter mit dem Steuerungstakt gekoppelt werden. Die spezielle Sprache

[7] Ein- /Ausschaltpunkt der Klebstoffdüse erreicht.
[8] Düse wirklich ein- oder ausschalten.

könnte optimal an die Anforderungen angepasst werden, so dass die Auswertungszeiten minimal gehalten werden können. Ferner kann diese Sprache erweiterbar gestaltet werden, so dass auch später auftretende Anforderungen in den Sprachumfang aufgenommen werden können.

2. Implementierung eines speziellen IRDATA-Interpreters, dem der erforderliche Befehlsumfang zur Verfügung steht und der in jedem Takt alle definierten Bedingungen ohne Unterbrechung auswertet.

Für Punkt 2 spricht, dass die Ausdrücke, mit denen Ereignisse innerhalb der Hochsprache definiert werden, sprachspezifisch sind und in der Syntax der jeweiligen Roboterhochsprache definiert werden. Daher bietet es sich an, diese mittels der bestehenden Compiler nach IRDATA-Code zu übersetzen. Die Code-Generierungseinheit der Compiler könnte mit diesem Konzept ohne Änderungen weiter verwendet werden und die vollständige Neuentwicklung eines Makrointerpreters kann unterbleiben. Es soll daher auf die Einführung einer neuen Makro-Sprache zur Definition von Bedingungen verzichtet werden. Im Folgenden soll auf das implementierte Verfahren eingegangen werden.

Zunächst zeigt das Abbildung 5.8 den gewünschten Ablauf innerhalb des IRDATA-Programms, das eine Ereignis-Behandlung initialisiert.

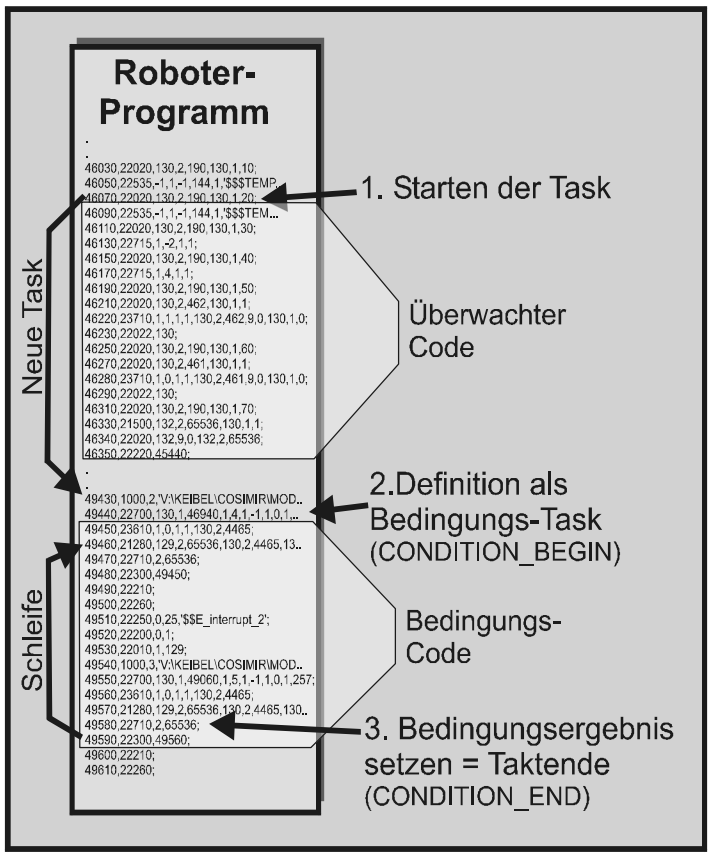


Abbildung 5.8: Initialisierung einer Überwachung

Dabei werden die Codeabschnitte, mit denen die Bedingung für das Ereignis berechnet werden sollen, zunächst als gewöhnliche Tasks gestartet[9]. Dieser Vorgang ist exakt identisch mit dem Start einer gewöhnlichen IRDATA-Task.

Innerhalb der Task trifft die Ausführungseinheit auf die Anweisung ”CONDITION_BEGIN”, mit der die Task als Bedingung definiert wird und die weitere Ausführung des Codes vollständig an den neuen Bedingungs- bzw. Ereignisinterpreter übergeben wird. Mit der Anweisung werden auch die erforderlichen Parameter der Bedingung übergeben, wie:

- Priorität des Ereignisses.
- Soll die Behandlung flanken- oder pegelgetriggert ausgelöst werden?
- Welche Art der Verzweigung soll erfolgen: Starten einer Task (Run Task), Aufruf eines Unterprogramms (Call Function), endgültige Unterbrechung des Programmflusses an der aktuellen Stelle und Fortsetzung an anderer Stelle (Goto Label)?
- Soll die Bedingung sofort aktiviert werden oder erst später durch eine spezielle Anweisung?

Steht das Ergebnis der Bedingung fest, wird dieses mittels der Anweisung ”CONDITION_END” an den Bedingungsinterpreter und damit in die Ereignis-Verwaltung übergeben. Die erneute Berechnung des Ergebnisses wird bis zum nächsten Takt angehalten und es wird mit dem normalen IRDATA-Interpreter fortgefahren. Dieser bearbeitet dann genau eine IRDATA-Anweisung und veranlasst beispielsweise einen Bewegungs-Interpolationstakt.

Sind mehrere Bedingungen gleichzeitig aktiv, dann werden sie in der Reihenfolge ihrer Prioritäten bearbeitet. Wird bei der Interpretation der Bedingungen ein gewisses Zeitlimit (R/VRC) oder eine gewisse Anzahl an IRDATA-Zeilen (VRC) überschritten, dann wird der Ereignisinterpreter an dieser Stelle angehalten und mit dem ”normalen” Interpreter fortgefahren, so dass es nicht möglich ist, den Programmfluss des Hauptprogrammes durch unzählige Ereignisse zum Erliegen zu bringen.

Nachdem die Ergebnisse der Bedingungen bestimmt wurden, werden diese innerhalb der Ergebnisverwaltung ausgewertet und eingetretene Ereignisse, abhängig von den Prioritäten, in die Liste zu bearbeitender Ereignisse einsortiert, dabei verweist der Kopf der Liste auf die dringendste Bedingung. Besteht die Behandlung des Ereignisses aus einem Unterprogrammaufruf (ähnlich einem Interrupt auf einem PC), wird die Bearbeitung des Bedingungscodes solange ausgesetzt, bis das Unterprogramm beendet wurde.

5.4.2 Ereignisorientierte Beeinflussung des Programmablaufes

Abbildung 5.9 zeigt die internen Zusammenhänge der Ereignisverarbeitung. Nachdem ein Ereignis eingetreten ist, befindet sich die Bedingung in der Liste der wartenden Behandlungen und wartet dort auf Bearbeitung. Dazu stellt die Programmflusskontrolle eine Unterbrechungsanforderung an die betroffene Ausführungseinheit, den normalen Programminterpreter. Dieser erteilt,

[9] Das erleichtert den Compilerbau, weil dieser Vorgang bereits von den Linkern unterstützt wird.

sobald möglich, die Genehmigung, und die Programmflusskontrolle verzweigt den Programmfluss an die vorgegebene Stelle. In der Abbildung ist als Beispiel die Methode des Aufrufes eines Unterprogramms dargestellt. Hierbei wird vor der Verzweigung der aktuelle Status des Interpreters, wie aktuelle Programmzeile und aktuelle Stacks, gesichert und die Verzweigung wird durchgeführt. Die Bearbeitung des Unterprogramms kann ihrerseits von dringenderen Bedingungsbehandlungen unterbrochen werden. Sollten in der Liste Ereignisse mit geringerer Priorität warten, so werden diese erst nach Rückkehr in das Hauptprogramm ausgeführt.

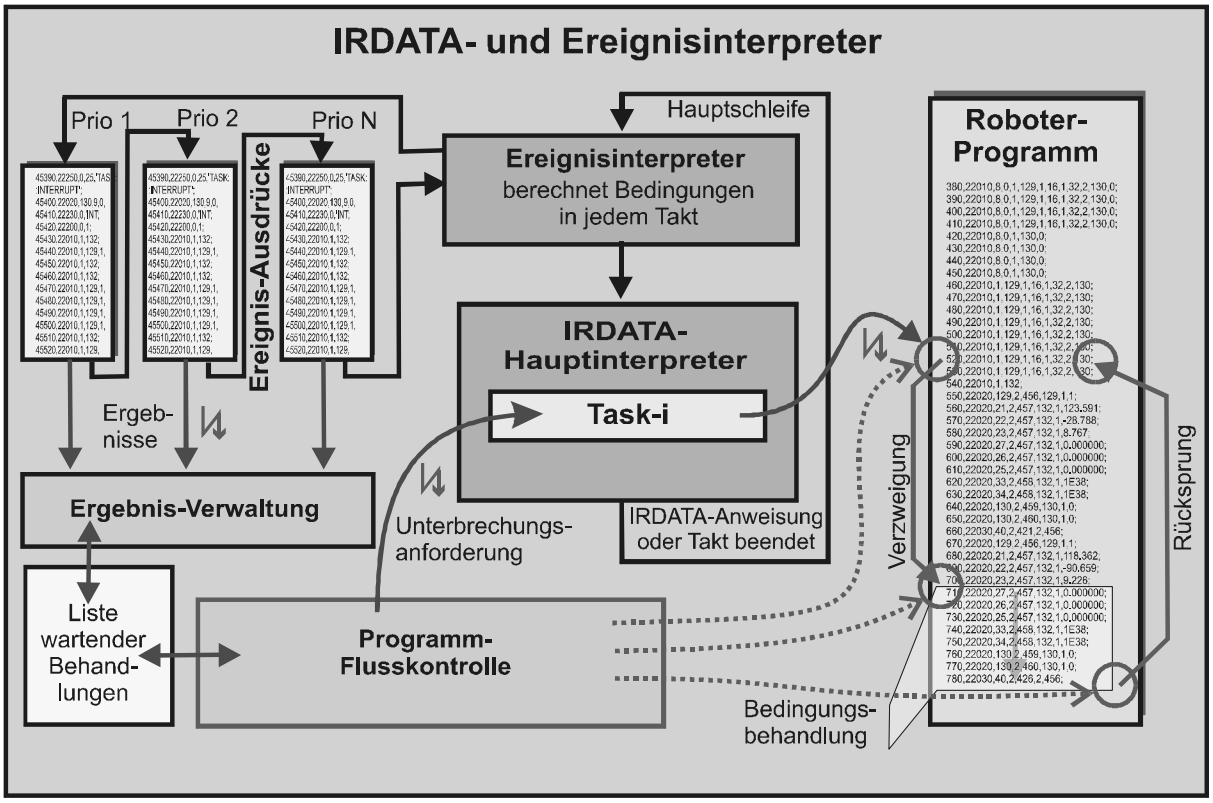


Abbildung 5.9: Funktionsschema der Ereignisverarbeitung

Bei der Realisierung der Ereignisverarbeitung wird für jedes Ereignis ein Zustandsautomat innerhalb der Programmflusskontrolle realisiert, der folgende Zustände annehmen kann:

1. **Undefiniert:**
   Die Datenstruktur dieses Ereignisses ist nicht initialisiert und darf nicht verwendet werden.

2. **Aktiv:**
   Der Code zur Ermittlung des Bedingungsergebnisses wird vom Ereignisinterpreter in jedem Takt bearbeitet, lieferte bisher jedoch noch kein positives Ergebnis.

3. **Wartend auf Ausführung:**
   Der Bedingungscode lieferte ”wahr” und das Ereignis wurde in die Liste der wartenden Behandlungen aufgenommen. Der Code zur Ermittlung des Bedingungsergebnisses wird von der Bearbeitung ausgenommen.

4. **Ausführend:**
   Die Programmflusskontrolle konnte die Bedingungsbehandlung im Hauptprogramm beginnen und der Hauptinterpreter ist mit der Bearbeitung des Unterprogramms beschäftigt. Besteht die Bedingungsbehandlung in der Aufgabe, eine Task zu starten oder den Programmablauf lediglich zu verzweigen (ähnlich einer ”GOTO”-Anweisung), wird dieser Zustand nicht angenommen, sondern sofort wieder auf ”Aktiv” gesetzt, da hierbei keine weiteren Handlungen wie ein Rücksprung anstehen. Die Bedingungsbehandlung wird in diesem Fall parallel zur Bedingungsüberprüfung bearbeitet.

5. **Ausführung beendet:**
   Die Behandlung des Unterprogramms wurde beendet, und der Hauptinterpreter wird dazu veranlasst, seine Programmausführung in der nach der Unterbrechung folgenden Programmzeile fortzusetzen. Der Code zur Ermittlung des Bedingungsergebnisses wird wieder bearbeitet. Danach ist der Zustand wieder ”Aktiv”.

Mittels dieser umfassenden und flexiblen Möglichkeit lassen sich alle Arten von bekannten Ereignissen, Interrupts, Triggern und Bahnschaltfunktionen komfortabel mittels der vorhandenen Compilerstrukturen auf IRDATA abbilden. Durch die gewählte Struktur ist auch eine hohe Sicherheit gegeben, bei zukünftigen Roboterhochsprachen mit neuen Arten von Ereignissen keine Änderungen an der Steuerung vornehmen zu müssen, so dass der Implementierungsaufwand lediglich bei der Herstellung der Compiler liegt. Aber auch hier ist der erforderliche Aufwand recht gering, da die Bedingungen selbst in IRDATA kodiert werden können und die Parametrierung der Ereignisse mittels weniger Befehle möglich ist. Dadurch ist kein Eingriff in das Codegenerierungs- und Linker-Modul erforderlich.

Laufzeituntersuchungen ergeben für den Ereignisinterpreter eine Ausführungsgeschwindigkeit von bis zu 300 000 Anweisungen pro Sekunde, gemessen auf einem Pentium 300 unter Windows NT. Mit dieser Leistungsfähigkeit lassen sich auch aufwändige Ausdrücke ohne spürbaren Zeitverlust innerhalb des Ereignisinterpreters auswerten. Der Vorgang der Verzweigung des Hauptprogramms bei eingetretenem Ereignis nimmt in etwa doppelt so viel Zeit in Anspruch wie die Ausführung einer einzigen IRDATA-Anweisung. Diese erstaunliche Performance hat verschiedene Gründe:

1. Im Gegensatz zum Hauptinterpreter müssen keine Zeileninformationen an das Simulationssystem geliefert werden, die gewöhnlich dafür sorgen, dass der Anwender den Ablauf seines Programms durch eine hervorgehobene Darstellung der betreffenden Quellcode-Zeile verfolgen kann.

2. Es vergeht innerhalb des Ereignisinterpreters keine simulierte Zeit, die das Simulationssystem dazu veranlasst die Ansicht des Arbeitszellenfensters im Simulations-Zeittakt neu zu berechnen und zu zeichnen.

3. Es werden bestimmte Anweisungen innerhalb der Auswertung von Ausdrücken verboten bzw. ”überlesen”. Zu diesen Anweisungen gehören u.a.:
   - Alle Arten von Benutzer Ein-/ und Ausgaben,
   - Datei Ein- und Ausgaben,
   - Anweisungen zur Beeinflussung des Roboters (Bewegungsanweisungen und Parameter),

- Synchronisations-Anweisungen (Semaphore und Events),
- Verzögerungs-Anweisungen,
- Task-Starts.

4. Es findet innerhalb des Ereignisinterpreters keine Vorlaufkontrolle und synchrone Bewegungsinterpolation statt.

5. Bei der Konzeption des Ereignisinterpreters wurde keine Zustandsmaschine verwendet, da sich der Interpreter lediglich in dem Zustand ”laufend” oder ”angehalten” befinden kann. Stati wie ”warte auf Bewegungsende”, ”warte auf Semaphor”, ”Warte auf Kommunikation mit dem Simulationssystem”, ”hole nächste IRDATA-Anweisung”, ”lese Eingang x”, usw. sind hier nicht erforderlich, da innerhalb einer Schleife alle Anweisungen der Bedingung sequenziell bearbeitet werden, bis entweder das Ergebnis gesetzt wird oder die maximale Anzahl an Anweisungen erreicht ist bzw. die diesbezügliche maximale Zeitdauer überschritten ist.

Die Implementierung stellt daher die schnellstmögliche Bearbeitung von IRDATA-Anweisungen dar, die durch keinerlei unnötige oder zusätzliche Aktionen und Kommunikationen gebremst wird.

5.5 Kinematikmodule

Die Kinematikmodule der Steuerung stellen die Funktionalität der Vor- und Rücktransformation der parametrierten Kinematik zur Verfügung. Da die VRC zur Robotersteuerungssimulation verschiedenster Robotersysteme eingesetzt wird, sollte sie ein breites Spektrum der in der Industrie relevanten Kinematiken unterstützen.

5.5.1 Klassifizierung von Kinematiktypen zur geschlossenen Lösung inverser kinematischer Probleme

Bei der Implementierung der kinematikspezifischen Funktionalitäten wird in dieser Arbeit der analytische Lösungsweg bevorzugt. Dabei werden die zu unterstützenden Roboterkinematiken klassifiziert und für jede Klasse eine eigene, optimierte analytische Lösungsmethode angewendet. Alle Lösungsmethoden verfügen dabei über eine einheitliche Schnittstelle zur Bewegungssteuerung, so dass mit dieser Modularität eine dynamische Umschaltung zwischen verschiedenen Kinematiken möglich ist.

Dieses Verfahren weist folgende Vorteile auf:

- Die Lösungen sind bevorzugt analytisch, was zu einer hohen Geschwindigkeit führt[10].
- Die Algorithmen arbeiten mit der Genauigkeit der Fließpunkt-Einheit des bearbeitenden Prozessors, wobei die resultierende Exaktheit nicht abhängig von einstellbaren $\varepsilon-$Umgebungen ist, wie i.A. bei iterativen Lösungen.

[10] Eine analytische Lösung bedarf keiner Iteration und der kompakte Code wird nur einmal durchlaufen.

- Berechnungen des IKP können sofort und unabhängig von aktuellen Achsstellungen erfolgen, so dass schnell die Erreichbarkeiten von Zielpositionen oder ganzen Positionslisten überprüft werden können.
- Stellungsmehrdeutigkeiten werden eindeutig aufgelöst bzw. ausgewertet.
- Das IKP ermittelt bei erreichbaren Positionen stets sinnvolle Achsstellungen, sogar in singulären Stellungen.
- Es werden kinematische Schleifen und Parallelogramme unterstützt.

Eine Einschränkung dieses Verfahrens besteht jedoch: Wenn sich eine zu unterstützende Kinematik nicht auf implementierte Klassen abbilden lässt, so handelt es sich bei dieser Kinematik um einen neuen Kinematiktyp, für den eine neue Klasse implementiert werden muss. Im Allgemeinen wird mit der Aufnahme einer neuen Kinematikklasse ein ganz neuer Bereich mit einer Vielzahl an Untertypen abgedeckt, die damit ebenfalls verfügbar werden, so dass dieser Aufwand gerechtfertigt ist.

5.5.2 Kinematik ohne analytische inverse Transformation

Kinematiken ohne Lösung des IKP werden in der Steuerung als Zusatzachsengeräte definiert, deren Achsanzahl lediglich durch den Speicherplatz des Steuerungsrechners beschränkt ist. Die mathematische Definition der kinematischen Kette eines derartigen Gerätes erfolgt mittels Denavit Hartenberg-Parametern (DH-Parametern) während der Parametrierung der Steuerung durch das Simulationssystem. Eine allgemeine Funktion zur Berechnung des DKPs beliebig langer, offener kinematischer Ketten übernimmt die Vorwärtstransformation für Zusatzachsengeräte. Mittels der aus den DH-Parametern bestimmten homogenen Transformation wird die gesamte kinematische Kette in der folgenden Form berechnet:

$$DKP = \sum_{i=Startachse}^{Endachse} T_{dh}^{i} \tag{5.1}$$

mit

$$T_{dh}^{i} = Rot_z(\theta_i) * Trans_z(d_i) * Trans_x(a_i) * Rot_x(\alpha_i) \tag{5.2}$$

als homogene Transformation eines einzelnen Denavit Hartenberg-Parametersatzes $(\theta_i, d_i, a_i, \alpha_i)$ der Achse i, die sich zusammensetzt aus der Rotation um die Z-Achse (Rot_z) des Ausgangskoordinatensystems, anschließender Translation in Richtung der neuen Z-Achse ($Trans_z$), Translation in Richtung der neuen X-Achse ($Trans_x$) und abschließender Rotation um die neue X-Achse (Rot_x). Je nach Gelenkart, ist hierbei einer der Parameter $(\theta_i, d_i, a_i, \alpha_i)$ als Achsstellungswert variabel.

Innerhalb des Programmablaufes kann auch die gegenwärtige kartesische Geschwindigkeit der Werkzeugspitze abgefragt werden, die dann als kartesischer Geschwindigkeits- und Winkelgeschwindigkeitsvektor mit Betrag an das Programm übergeben wird. Durch Bildung des Differenzenquotienten ist mittels dieser Transformation auch die Ermittlung von kartesischer Winkelgeschwindigkeit und Translationsgeschwindigkeit allgemein und einfach möglich. Dabei wird wie folgt verfahren:

1. Berechnung des DKP zur gewünschten Achsstellung liefert eine konstante homogene Transformation T_0.

2. Berechnung des DKP zur Achsstellung am Zeitpunkt $t + \Delta t$, wobei Δt einen Bruchteil der Interpolationszeit hinter t liegt. Diese Achsstellung wird mittels der übergebenen Achsgeschwindigkeit zum Zeitpunkt $t + \Delta t$ unter Annahme konstanter Geschwindigkeit berechnet. Das Ergebnis ist T_1.

3. Berechnung der differenziellen homogenen Transformation:

$$dT = T_0 * T_1^{-1}. \tag{5.3}$$

4. Aufteilung der differenziellen homogen Transformation dT in Positionsdifferenz dP und Orientierungsdifferenzmatrix dM.

5. Konvertierung von dM in Drehvektor-Drehwinkel-Darstellung.

6. Multiplikation von dP und Drehwinkel mit $\frac{1}{\Delta t}$, um auf Geschwindigkeitswerte bzw. SI-Größen[11] zu kommen. Ergebnis: die gesuchte kartesische Geschwindigkeit im Raum $\vec{v}$ in $\frac{mm}{s}$ und die Winkelgeschwindigkeit $\vec{\omega}$ in $\frac{rad}{s}$ um einen Drehvektor.

Diese Werte können z.B. innerhalb des Ereignisinterpreters (siehe Kapitel 5.4) oder einer IRDATA-Task überwacht werden.

Eine analytische Berechung der kartesischen Geschwindigkeitswerte aus den DH-Parametern beliebiger kinematischer Ketten gestaltet sich deutlich aufwändiger. Der Zusammenhang zwischen Achsgeschwindigkeiten und der kartesischen Geschwindigkeit kann durch das Produkt der Jacobimatrix J mit den Achsgeschwindigkeiten zu jeder Achsstellung angegeben werden:

$$\frac{dPos}{dt} = J(Q)\frac{dQ}{dt} \tag{5.4}$$

[mit Pos = Lagevektor im Raum, J = Jacobimatrix, Q = Gelenkstellungsvektor]

[11] SI = **S**ystème **I**nternationale d'unités

Daher setzt sich die Jacobimatrix aus den partiellen Ableitungen sämtlicher kartesischer Freiheitsgrade nach allen Achsen zusammen:

$$J(Q) = \begin{bmatrix} \frac{\partial P_1}{\partial Q_1} & \cdot & \cdot & \frac{\partial P_1}{\partial Q_M} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \frac{\partial P_N}{\partial Q_1} & \cdot & \cdot & \frac{\partial P_N}{\partial Q_M} \end{bmatrix} \tag{5.5}$$

[Mit P_i = der i-ten Komponente der N Freiheitsgrade der Position Pos im kartesischen Raum und Q_k der k-ten Achse der Kinematik von insgesamt M Achsen]

Diese Komponenten der Jacobimatrix lassen sich selbstverständlich analytisch aus dem Produkt sämtlicher DH-Transformationen berechnen. Eine Vereinfachung kann hier durch aufwändige Fallunterscheidungen nach den veränderlichen Größen jeder Achse erfolgen, was letztlich aber deutlich mehr Rechenzeit und Implementierungsaufwand erfordert, als den Differenzenquotienten aus zwei Vorwärtstransformationen zu berechnen, von denen eine ohnehin in jedem Takt berechnet werden muss. Zu beachten ist, dass die Jacobimatrix achsstellungsabhängig ist und daher in jedem Takt berechnet werden muss. Auch andere Ansätze, wie z.B. der von Zomaya [Alb93], der auf der Formel 5.4 aufbaut, oder Methoden basierend auf Lie Groups mit Lie Algebra [MLS94], benötigen mehr Rechenzeit, als die Methode mit dem Differenzenquotienten, der daher an dieser Stelle eindeutig der Vorzug geben werden muss.

5.5.3 Allgemeiner Ansatz zur Lösung des inversen kinematischen Problems

Zur Lösung des IKP beliebiger Kinematiken existieren verschiedene allgemeine Ansätze, die vorwiegend auf der iterativen Näherungsmethode mit der inversen Jacobimatrix beruhen. Dabei wird durch Lösung des direkten kinematischen Problems, die kartesische Position im Raum zu einer gegebenen Gelenkstellung berechnet. Soll diese Position verändert werden, so wird mit der Jacobimatrix der Gelenkraum um diese Stellung linearisiert und mit dieser Linearisierung kann die gewünschte Gelenkstellung in wenigen Iterationsschritten erreicht werden. Formel 5.4 gibt diesen Zusammenhang wieder. Eine genaue Beschreibung, wie mit diesem Verfahren eine Gelenkwinkelermittlung durchgeführt werden kann, ist in [Fre00] beschrieben.

Dieses Verfahren hat folgende Vorteile:

- Das Verfahren funktioniert unverändert für jede Kinematik mit genau sechs Freiheitsgraden, deren DKP bekannt und bei der mindestens der Differenzenquotient des DKP berechenbar ist.
- Bei Verwendung einer Pseudoinversen der Jacobimatrix und Zusatzbedingungen können auch Achsstellungen für redundante oder unterbestimmte kinematische Ketten bestimmt werden [Hah99], [AS86], [Bau92].
- Das Verfahren ist bei besonderen kinematischen Konstruktionen, bei denen eine analytische Lösung nicht bestimmbar ist, eine elegante Methode zur Bestimmung der gesuchten

Achswinkel/-Strecken[12].

Neben diesen Vorteilen werden aber auch Nachteile des Verfahrens mit der Jacobimatrix genannt:

- In der Nähe von Singularitäten treten Konvergenzprobleme auf, so dass die Iteration keine Lösung findet.
- Mit der Anzahl der Freiheitsgrade des Systems steigt der Rechenaufwand überproportional, so dass die Invertierung der Jacobimatrix, die ja in jedem Iterationsschritt erforderlich ist, bei vielachsigen Systemen einen erheblichen Teil der Rechenzeit der gesamten Transformation ausmachen kann. Während Bahnbewegungen ist dieser Fall i.A. nicht kritisch, da jeweils die Achsstellung des vorangegangenen Taktes als Startvektor verwendet werden kann. Zur Überprüfung der Erreichbarkeit des Zielpunktes ist dieser Aspekt jedoch sehr wohl relevant.
- Zu einer gegebenen kartesischen Lage muss immer eine Gelenkstellung in hinreichend kleiner Umgebung, also ein Startvektor vorhanden sein. Dieser Startvektor darf nicht durch singuläre Stellungen von der Zielstellung getrennt sein, da das Verfahren sonst nicht zu einer Lösung führt.
- Das Verhalten wird mit Annäherung an eine singuläre Stellung schnell ungenau und instabil. Genau in einer singulären Stellung ist keine Lösung ermittelbar, obwohl mathematisch unendlich viele existieren.

Heute verfügen die Steuerungssysteme über ausreichend Rechenleistung, um bei Einhaltung bestimmter Rahmenbedingungen und bei Kinematiken mit wenigen Achsen, wie sie bei Industrierobotern vorkommen, die Lösung selbst auf PCs (z.B. Pentium 1 GHz) mit hohen Frequenzen größer 1 kHz zu berechnen. Der vorletzte Punkt der hier angeführten Nachteile stellt allerdings eine Einschränkung dar, die in modernen Robotersteuerungssystemen nur bedingt hinnehmbar ist. Denn sie bedeutet, dass es nicht explizit möglich ist, zu einer beliebigen Position im Raum eine korrespondierende Gelenkstellung zu ermitteln, etwa um deren Erreichbarkeit zu überprüfen. Steht nur diese Lösungsweise des IKP mittels Jacobimatrix zur Verfügung, so muss, ausgehend von einer aktuellen Gelenkstellung in Richtung der Zielposition interpoliert werden, bis diese erreicht ist oder ein Fehler auftritt (z.B. Überschreiten der Achsgrenzen). Dieser Aufwand kann nicht innerhalb des festen Zeittaktes der laufenden Bewegungssteuerung erfolgen, da die Laufzeit der Iteration nicht genau vorhersehbar ist. Die in dieser Arbeit beschriebene Steuerung muss ständig Erreichbarkeiten von Positionen überprüfen, die nicht in der Nähe der aktuellen Stellung liegen müssen, etwa für die Zielposition der nächsten Bewegung, oder durch Anfragen anderer Komponenten des Simulationssystems. Da diese Anfragen entweder aus dem Roboterprogramm (also dem IRDATA-Interpreter) oder über die Steuerungsschnittstelle angestoßen werden, die beide asynchron zur Bewegungssteuerung laufen, steht der Verwendung dieses allgemeinen Ansatzes als generische Lösung für die R/VRC nichts entgegen, jedoch sollte dem analytischen Verfahren mit deterministischer Laufzeit stets Vorrang vor iterativen Verfahren gegeben werden.

[12] In Kapitel 5.5.4 wird zur Bestimmung der Achsen einer besonderen Handkonstruktion ein schnell konvergierendes, iteratives Verfahren beschrieben, in dem die o.a. Beziehung ausgenutzt wird.

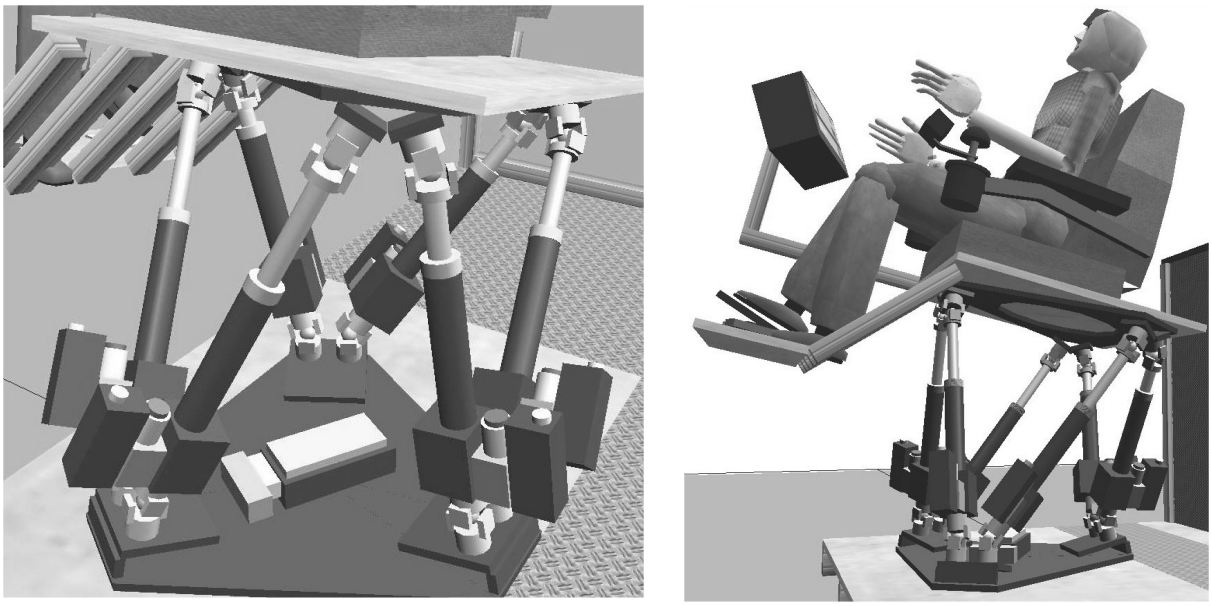


Abbildung 5.10: Darstellungen eines simulierten Hexapods

Innerhalb der Steuerung findet dieses Verfahren bei der kartesischen Positionierung von Zusatzachsengeräten Verwendung. Diese Zusatzachsengeräte können vom Anwender selbst frei modelliert werden und passen daher nicht unbedingt in eine Kinematik-Klasse, für die in der Steuerung eine Lösung für das IKP vorhanden ist.

Abbildung 5.10 zeigt die Simulationsansicht eines Hexapods zur Ausrichtung des Arbeitsplatzes einer simulierten Forstmaschine. Diese Plattform bewegt in der Realität den Stuhl des Bedieners, damit dieser auch rein physikalisch eine Rückkopplung über die Lage seines simulierten Fahrzeuges erhält. Innerhalb des Simulationssystems wird die Steuerung der Kinematik dieser Plattform von der Robotersteuerung übernommen, wobei sich die sechs einzelnen Gelenkkörper mit je sechs Achsen keiner bisher implementierten Kinematik-Klasse zuordnen lassen und daher als Zusatzachsen definiert wurden.

Mittels des vorgestellten iterativen Verfahrens kann dennoch eine Rücktransformation für jeden einzelnen Körper bestimmt werden, so dass bei kartesischer Bewegung der Plattform sämtliche Gelenkkörper korrekt berechnet und angezeigt werden.

Im nächsten Kapitel wird beispielhaft genauer auf eine Kinematik-Klasse eingegangen, mit der das inverse kinematische Problem verschiedener sechsachsiger industrieller Knickarmroboter analytisch gelöst wird.

5.5.4 Universal Sechs-Achsen-Knickarm-Kinematik

Diese Kinematikklasse deckt die am häufigsten eingesetzten Industrieroboter mit sechs Freiheitsgraden ab. Sie besteht ausschließlich aus rotatorischen Achsen. Die erste Achse steht senkrecht auf dem Boden, die zweite und dritte liegen kollinear und parallel zum Boden und bilden die Grundachsen. An die Grundachsen schließen sich drei Handachsen an. In der implementierten

Version dieses Kinematikmoduls werden drei Hand-Typen unterstützt, die verschieden aufwändig zu berechnen sind. Insbesondere unter Berücksichtigung der kinematischen Redundanzen kommt es hier u.U. zu komplizierten Fallunterscheidungen.

Zentralhand

Die Zentralhand ist die am weitesten verbreitete Variante, weil diese mathematisch am einfachsten zu handhaben ist und daher mit der zu Beginn der Robotik verfügbaren Rechenleistung am besten vereinbar war. Lösungsverfahren für die Rücktransformation von Robotern dieser Art sind in der Literatur an vielen Stellen zu finden. Hier sei auf [Meh85] verwiesen.

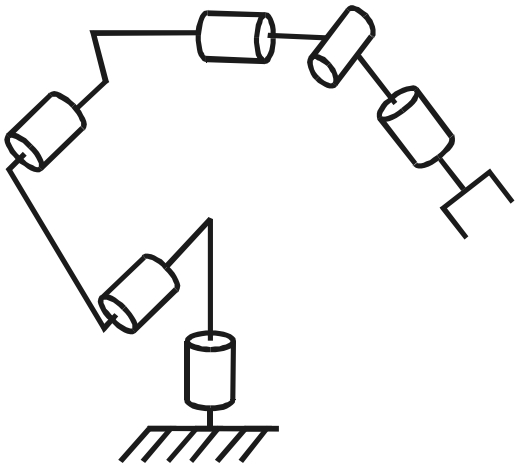


Abbildung 5.11: Sechs-Achsen-Knickarm-Kinematik

Abbildung 5.11[13] zeigt symbolisch die Struktur einer solchen Kinematik, bei deren Konstruktion sich alle Handachsen in einem Punkt schneiden - daher der Name der Hand. Bei der Lösung des IKP wird wie folgt vorgegangen:

1. Berechnung der Position des Handachsenschnittpunktes im Raum aus der gegebenen Soll-Orientierung und der Soll-Position.
2. Berechnung der Stellung der Grundachsen aus der Position des Handachsenschnittpunktes im Raum.
3. Vorwärtstransformation der Grundachsen, um die Orientierung am Ende der Grundachsen zu bestimmen.
4. Berechnung der Differenzorientierung zwischen Orientierung am Ende der Grundachsen und der Sollorientierung.
5. Berechnung der Handachsen aus der Differenzorientierung.

[13] Der Übersichtlichkeit wegen werden Strichzeichnungen abgebildet. Gelenke werden dabei als Zylinder dargestellt.

Die Lösung des IKP profitiert von dem Umstand, dass sich die Handachsen in einem Punkt schneiden und damit von der kartesischen Sollstellung einfach auf die Stellung der Grundachsen geschlossen werden kann.

Winkelhand

Zur Unterstützung des japanischen Weltraumroboters ETS-VII wurde die Kinematikklasse um die Winkelhand erweitert. Die Winkelhand ähnelt einer Kurbel mit einer zusätzlichen Achse auf dem Verbindungssteg der beiden Kurbelachsen. Der im GETEX-Projekt verwendete Weltraum-Roboter ETS-VII verfügt über diese Art von Handachsen [FHR00]. Abbildung 5.12 zeigt eine Simulationsansicht des Roboters ERA, montiert auf dem Satelliten, der im Rahmen des GETEX-Projektes die Erde seit dem 28. November 1998 in 550 km Höhe umrundet.

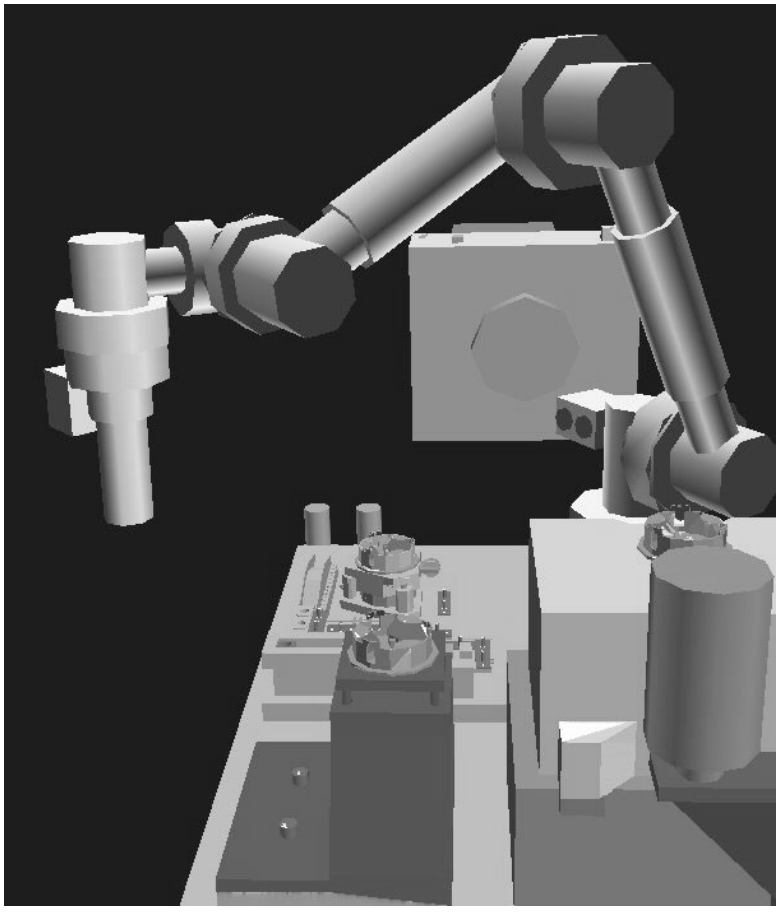


Abbildung 5.12: ETS-VII auf der Satellitenplattform (Simulationsansicht)

Während der Laufzeit des Projektes wurde das IRF an Experimenten mit dem frei schwebenden Roboter beteiligt. Zur Bahnsteuerung des Roboters wurde dabei der folgende analytische Lösungsweg zur Lösung des IKP erfolgreich eingesetzt. Abbildung 5.13 stellt den kinematischen Aufbau des Roboters dar.

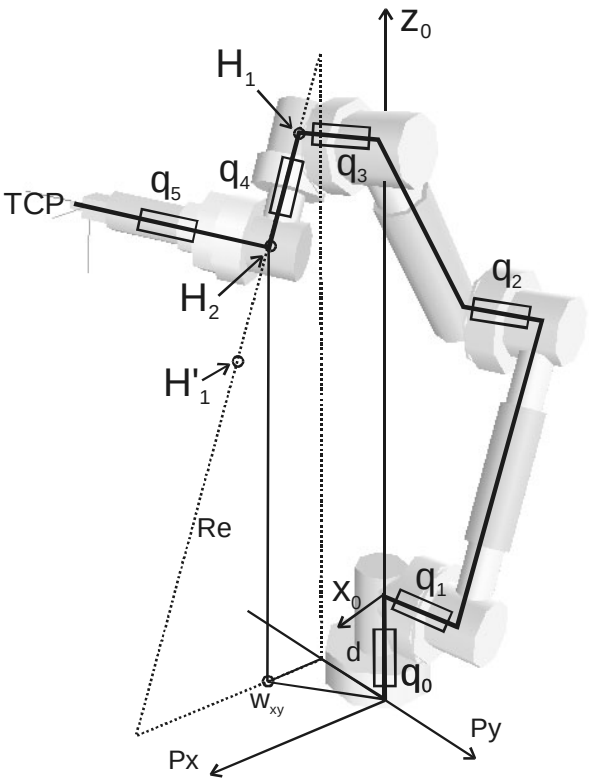


Abbildung 5.13: Kinematik des ETS-VII Roboters

Da sich die Handachsen bei dieser Kinematik nicht in einem gemeinsamen Punkt schneiden, fällt die Lösung des IKP etwas schwerer als bei der Zentralhand, denn die Grundachsen enden in der vierten Achse (hier q_3), deren Lage nicht direkt aus der Position und Orientierung des TCP berechnet werden kann. Hier ist ein entscheidender Zwischenschritt erforderlich, der zudem Mehrdeutigkeiten des Armes berücksichtigen muss. In Abbildung 5.13 und im anschließenden Text werden die Abkürzungen aus Tabelle 5.1 verwendet.

$q_0 - q_5$	Achsen des Roboters
H1	Handachsenschnittpunkt1
H2	Handachsenschnittpunkt2
TCP	Tool-Center-Point
Px, Py	X- und Y-Achse des Weltkoordinatensystems
x_i, y_i, z_i	Koordinatenachsen des Koordinatensystems der i-ten Achse
Re	Roboter-Ebene, in der H1 und H2 liegen und die parallel zu z_0 verläuft
w_{xy}	Auf die Bodenfläche projizierte Position des H1
$\overline{q_i q_{i+1}}$	Gelenklängen als Abstände der Achsen i und $i+1$
d	Abstand der Roboterebene von der z_0-Achse in Richtung der y_0-Achse

Tabelle 5.1: Transformationsvariablen

Die Mehrdeutigkeiten sind in den folgenden Abbildungen dargestellt, bei denen eine identische kartesische Position auf acht verschiedene Art und Weisen eingenommen werden kann.

Zur Spezifikation, welche Stellung gewünscht wird, werden drei Strukturbits verwendet, die als ”left/right”, ”above/below” und ”flip/noflip” bezeichnet werden. Diese Strukturbits müssen der Rücktransformation übergeben werden.

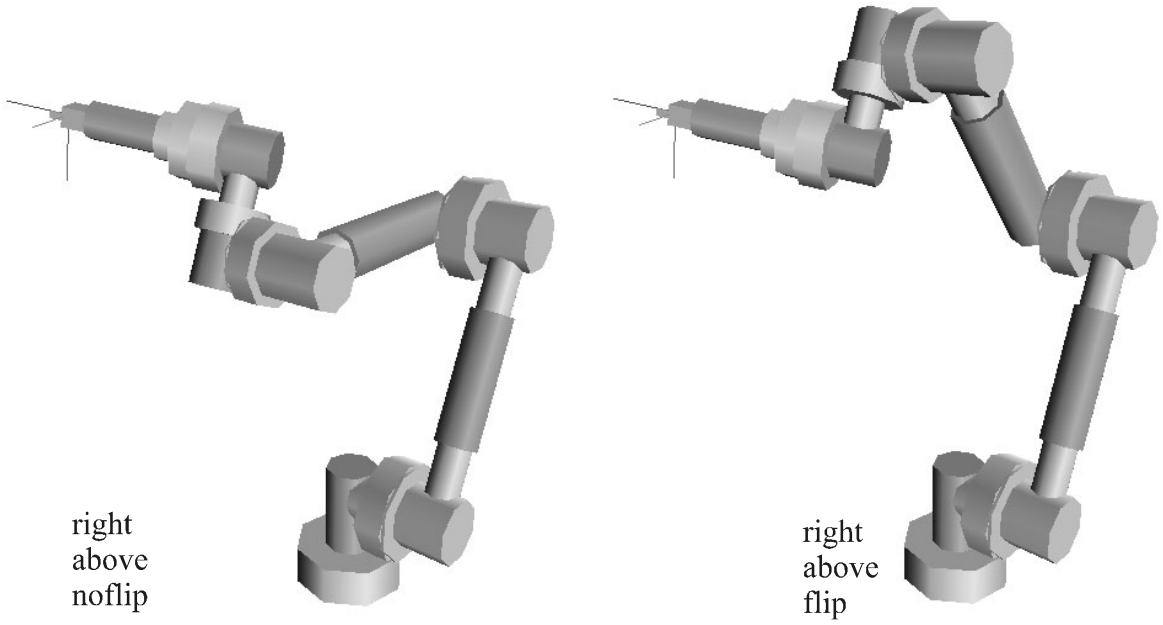


Abbildung 5.14: Zwei ”above” Konfigurationen

Das Bit ”flip/noflip” spezifiziert im Allgemeinen, auf welcher Seite der Handsingularität sich der Roboter nach der Rücktransformation befinden soll. Die Abbildungen 5.14 und 5.15 stellen einige Konfigurationen des Roboters dar, bei denen die exakt gleiche kartesische Position eingenommen wird. Abbildung 5.14 verdeutlicht dabei, auf welche Weise die Stellung dieses Roboterarmes durch das Bit ”flip/noflip” beeinflusst wird: Für die Position des Handachsenschnittpunktes H1 existieren zwei mögliche Positionen, und mittels dieses Bits wird festgelegt, welche Variante gewünscht wird. Die Handsingularität liegt vor, wenn Achse q_3 und q_5 parallel zueinander sind, und für die Lage von H1 unendlich viele Positionen in Frage kommen. In diesem Falle kommen nämlich alle Punkte auf dem Vollkreis mit $r = \overline{q_3 q_5}$ um H2 in der Roboterebene Re in Betracht.

Das Bit ”above/below” spezifiziert, auf welcher Seite die Gelenkkörper der Grundachsen bezüglich der Ebene liegen, in der q_1 und q_3 liegen.

Bit ”left/right” bestimmt, ob die X-Koordinate der Position von H2 in Koordinaten der ersten Achse positiv oder negativ sein soll, was in Abbildung 5.15 deutlich wird. Anschaulich wird diese Definition etwas klarer: in der Stellung ”right” arbeitet der Roboter vor sich, während er in ”left” nach hinten übergeklappt, hinter sich positioniert ist.

Bei der Lösung des inversen kinematischen Problems wird wie folgt verfahren:

Berechnung der konstanten, inversen Transformation von H2 bis zum TCP $[H2TCP]^{-1}$.

Zur Bestimmung von H2 wird an das Werkzeug-Frame[14] $[TCP]$ diese Inverse Transformation

[14]Mit ”Frame” werden in dieser Arbeit homogene Transformationen bzw. Koordinatensysteme bezeichnet, die

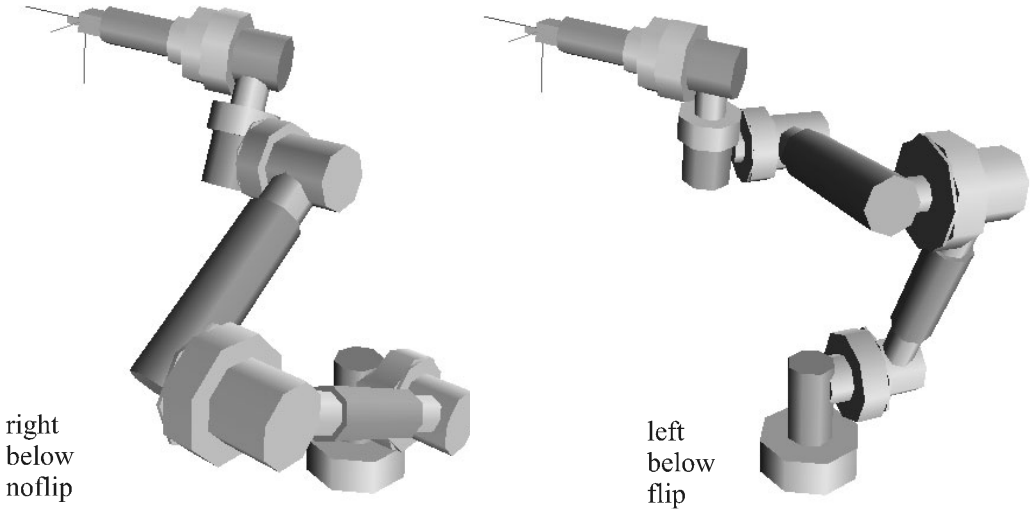


Abbildung 5.15: Zwei ”below” Konfigurationen

multipliziert:

$$H2 = TCP * [H1TCP]^{-1} \tag{5.6}$$

Aus der Position von H2 können mittels dessen Projektion auf die Bodenebene des Roboters die beiden in Frage kommenden Achswinkel für q_0 berechnet werden. In Abbildung 5.16 sind diese beiden Möglichkeiten in der Aufsicht auf den Roboter schematisch dargestellt.

Zur Berechnung wird das komplex-geometrische Lösungsverfahren mit den hier eingeführten komplexen Hilfsvariablen $c, c1, c2 \in \mathbb{C}$ verwendet, mit denen rechtwinklige Dreiecke beschrieben werden können.

Gesucht sind zunächst die beiden Lösungen des Winkels q_0, die hier als q_{0_1} und q_{0_2} bezeichnet werden.

Wegen

$$wr^2 = w_x^2 + w_y^2 \tag{5.7}$$

und

$$wr^2 = d^2 + h^2 \tag{5.8}$$

sich aus der Orientierungsmatrix und dem Positionsvektor zusammensetzen.

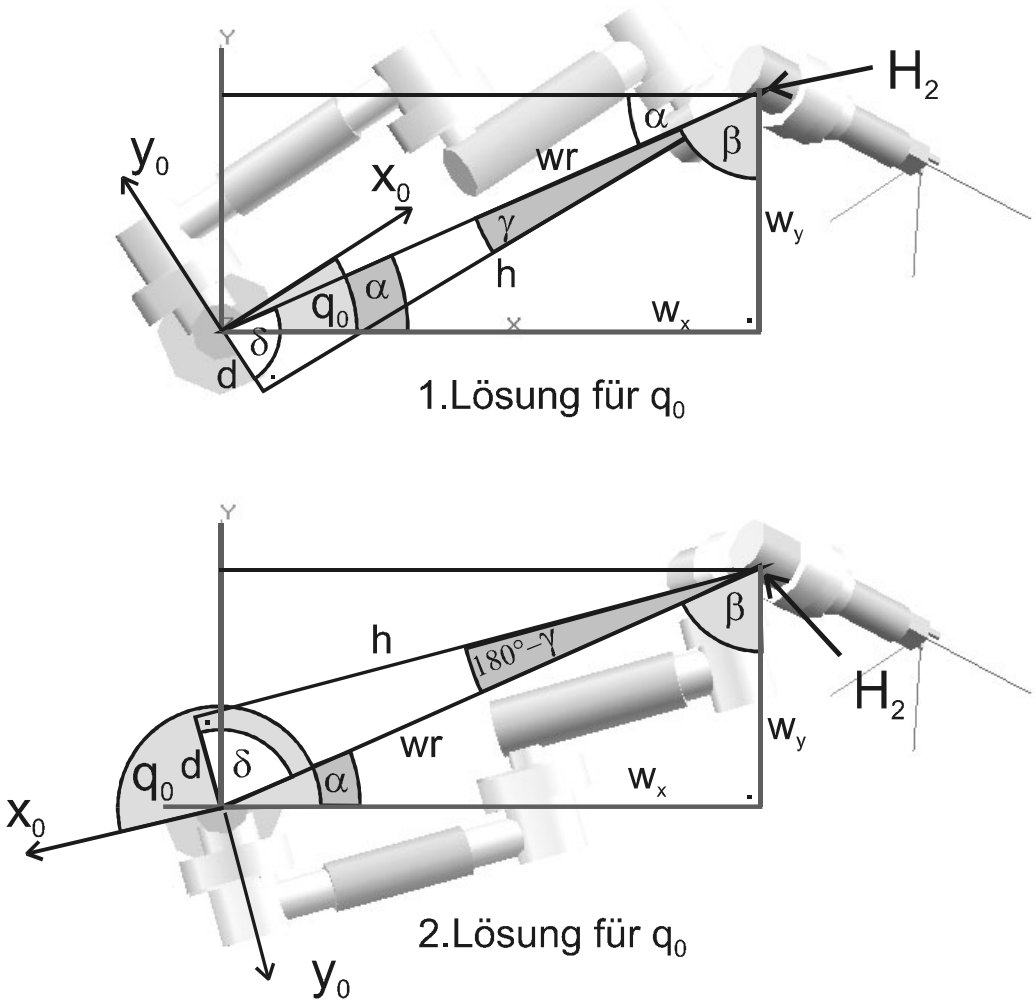


Abbildung 5.16: Redundanz der ersten Achse

folgt

$$h^2 = w_x^2 + w_y^2 - d^2. \tag{5.9}$$

Setzt man

$$c1 = w_y + j w_x, \tag{5.10}$$

dann beschreibt c1 damit das Dreieck, das durch die Koordinatenachsen X und Y des Weltkoordinatensystems und H1 in der Bodenebene geben ist. Das Argument von $c1$ entspricht dem Winkel β und ist bei beiden möglichen Lösungen gleich. Zu beachten ist jedoch, dass d per Definition aus Tabelle 5.1 der Abstand der Roboterebene von der z_0-Achse in Richtung der y_0-Achse ist, und daher bei diesem Roboter negativ ist.

$$c2 = h + jd, \tag{5.11}$$

c2 beschreibt das Dreieck, das durch die Verbindungsstrecken zwischen der z_0-Achse und H1 bzw. zwischen der z_0-Achse und der Roboterebene gegeben ist. Das Argument entspricht dem Winkel γ. In der "right"-Konfiguration haben die Koordinaten von H1 im System der ersten Achse positive x-Werte, in der "left"-Konfiguration negative. Die x_0-Koordinatenachse verläuft parallel zur Roboterebene Re, in der h liegt, d.h. mit dem Winkel, den h mit der X-Achse des Basiskoordinatensystems einschließt, ist auch die Lage der Roboterebene, und damit q_0, bestimmt. Für diesen Winkel existieren zwei Lösungen.

Die erste Lösung für q_0 ergibt sich geometrisch durch den Term $\frac{\Pi}{2} - (\beta + \gamma)$.

Das Argument des Produktes zweier komplexer Zahlen ergibt die Winkelsumme der Faktoren,

$$c = c1 * c2 \tag{5.12}$$

Zu beachten ist hierbei, dass wegen $d < 0$ auch $\gamma < 0$ wird!

Die erste Lösung für den Winkel q_0 (hier als q_{0_1} bezeichnet) ergibt sich zu:

$$q_{0\_1} = \frac{\Pi}{2} - \arg(c). \tag{5.13}$$

Die zweite Lösung für q_0 (hier als q_{0_2} bezeichnet) erhält man durch Invertierung des Einflusses von h, also des Realteiles von c2:

$$\begin{aligned} c2 &= -h + jd, && (5.14)\\ c &= c1 * c2, && (5.15)\\ q_{0\_2} &= \frac{\Pi}{2} - \arg(c) && (5.16) \end{aligned}$$

Damit sind die beiden möglichen Lösungen für Achse 1 berechnet. Welche der beiden Stellungen nun "right" oder "left" entspricht, hängt davon ab, ob die X-Koordinate des H2 in Koordinaten der 1. Achse positiv (right) oder negativ (left) ist. Da beide Möglichkeiten bekannt sind, kann die gewünschte Stellung hier aufgelöst werden, indem H2 um $-q_{0\_1}$ bzw. um $-q_{0\_2}$ gedreht wird und die resultierende X-Koordinate in Basiskoordinaten betrachtet wird.

Ziel der folgenden Operationen ist es, die beiden möglichen Lagen des ersten Handachsenschnittpunktes H1 zu bestimmen, weil aus dessen Position die beiden restlichen Grundachsen bestimmt werden können.

Mit der Stellung der ersten Achse ist auch die Lage der Roboterebene Re im Raum bekannt. Auf dieser Ebene liegen stets beide Handachsenschnittpunkte. Die Darstellung dieser Ebene ist

mittels Hessescher Normalenform einfach mit den bereits berechneten Größen zu beschreiben, denn der Normalenvektor ist y_0 und der Abstand vom Ursprung ist d.

Mit

$$\overrightarrow{y0} = \begin{pmatrix} \sin(q0\_r) \\ -\cos(q0\_r) \\ 0 \end{pmatrix} \tag{5.17}$$

lautet die Ebenengleichung

$$\overrightarrow{y_0}\,\overrightarrow{x} + d = 0 \tag{5.18}$$

Zusätzlich liegen beide Handachsenschnittpunkte in der Ebene, die durch x_5, y_5 und H2 gegeben ist. Diese Ebene steht damit senkrecht auf der letzten Achse q_5. Ferner haben beide Handachsenschittpunkte den konstanten Abstand l_5 des Gelenkkörpers voneinander. Mit diesen Zusammenhängen lässt sich zunächst die Schnittgerade der beiden Ebenen bestimmen, auf der H1 liegen muss[15]. Ausgehend von H2 können nun in beide Richtungen auf der Geraden im Abstand l_5 H1 und H1' gefunden werden. Welcher der beiden mittels der angegebenen Handkonfiguration ("flip/noflip") gemeint ist, muss nun bestimmt werden. Wie bereits zuvor erläutert, werden diese beiden Konfigurationen durch die Handsingularität voneinander getrennt, die vorliegt, wenn Achse q_3 und q_5 parallel sind (und die beiden Ebenen sich nicht in einer Geraden schneiden, sondern vielmehr exakt aufeinander liegen). In dieser singulären Stellung soll q_4 gleich Null sein.

Zunächst wird der Vektor von H2 nach H1 berechnet:

$$\overrightarrow{lh2} = \overrightarrow{H2} - \overrightarrow{H1} \tag{5.19}$$

und ein temporärer Vektor $\overrightarrow{vt}$ aus dem Kreuzprodukt der z-Achse des letzten Koordinatensystems[16] und $\overrightarrow{lh2}$.

$$\overrightarrow{vt} = \overrightarrow{z_5} \times \overrightarrow{lh2} \tag{5.20}$$

Zeigt dieser Vektor in Richtung von $\overrightarrow{y_0}$, so ist Achse 5 negativ und die Konfiguration ist "flip". Ermittelt werden kann dieses über das Punktprodukt:

$$pp = \overrightarrow{y_0} \cdot \overrightarrow{vt} \tag{5.21}$$

[15] Die Berechnung der Schnittgeraden zweier Ebenen soll hier aus Platzgründen nicht ausgeführt werden.

[16] Die z-Achse des letzten Koordinatensystems zeigt aus dem Flansch heraus.

Ist also $pp > 0$, so befindet sich der Roboterarm in der "flip"-Stellung. Entspricht dies nicht der gewünschten Konfiguration, dann muss $\overrightarrow{lh2}$ mit dem anderen möglichen Handachsenschnittpunkt H1' berechnet werden.

Entspricht "flip" hingegen der Wunschkonfiguration, so kann q_{4_f} (=q_4 flip) nun aus dem Schnittwinkel der beiden Ebenen berechnet werden:

$$q_{4\_f} = -Schnittwinkel\_der\_Ebenen - \frac{\Pi}{2} \tag{5.22}$$

Wenn $pp < 0$ ist, so ist die Stellung "noflip". Entspricht dies nicht der gewünschten Konfiguration, dann muss $\overrightarrow{lh2}$ mit dem anderen möglichen Handachsenschnittpunkt H1' berechnet werden.

Entspricht "noflip" der Wunschkonfiguration, so kann q_{4_n} (=q_4 noflip) nun aus dem Schnittwinkel der beiden Ebenen berechnet werden:

$$q_{4\_n} = Schnittwinkel\_der\_Ebenen - \frac{\Pi}{2} \tag{5.23}$$

Damit sind nun gleichzeitig q_4 und der korrekte H1 berechnet worden und zusammen mit q_0 lässt sich nun die Stellung der fehlenden Grundachsen bestimmen. Auch hierbei wird komplexgeometrisch verfahren. Die Ausführung erfolgt ähnlich wie in [Meh85] beschrieben und wird daher an dieser Stelle nicht erneut abgedruckt. Aus der Stellung der Grundachsen $q_0 - q_2$ kann nun eine Vorwärtstransformation bis kurz vor Achse q_3 erfolgen. q_3 ergibt sich aus dem Winkel zwischen dem Gelenkkörper, der die Achsen q_2 und q_3 verbindet und der Richtung des Vektors $\overrightarrow{lh2}$. q_4 wurde bereits bestimmt, und q_5 ergibt sich aus dem Winkel zwischen y_5 und dem Kreuzprodukt von $\overrightarrow{-lh2}$ und z_5.

Flexible Hand

Die flexible Hand, auch "Flexwrist" oder "Flexiwrist" genannt, nimmt in der Robotik eine Sonderstellung ein, da sie nur bei sehr wenigen Robotern verwendet wird und einige besondere Eigenschaften aufweist. Abbildung 5.17 zeigt ein Foto der technischen Realisierung durch die Fa. ABB dar, und Abbildung 5.18 stellt einen Roboter mit dieser Hand-Kinematik schematisch dar. Sie setzt sich aus drei Gelenkkörpern zusammen, die mittels dreier Kugelgelenke[17] untereinander verbunden sind. Jedes dieser Kugelgelenke hat zwei Freiheitsgrade. Am Ende der Gelenkkörper befindet sich eine Rotationsachse, die aus dem Gelenkkörper herauszeigt, und durch das letzte Kugelgelenk verläuft, so dass die Konstruktion der Hand alleine, ohne weitere Einschränkungen, schon sieben Freiheitsgrade aufweist.

Die beiden Freiheitsgrade der einzelnen Kugelgelenke sind jedoch mit denen der anderen mechanisch, elektronisch oder mathematisch verkoppelt:

Seien a_i und $b_i \in R$ mit $i \in [0..2]$ die Winkel der Kugelgelenke, die sich um die X- und Y-Achse des Vorgängerkoordinatenssystems drehen, dann sind

[17] Diese Kugelgelenke sind in der schematischen Darstellung als Kugeln und deren Lagerpfanne skizziert.

Abbildung 5.17: Foto der ”Flexiwrist” eines IRB 540 Roboters der Firma ABB

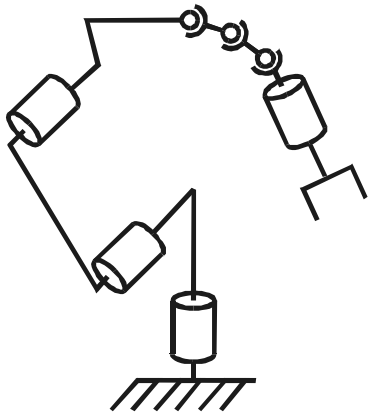


Abbildung 5.18: Kinematik mit flexibler Hand

$$a_1 = a_0 k_1, \tag{5.24}$$

$$b_1 = b_0 k_1, \tag{5.25}$$

$$a_2 = a_0 k_2, \tag{5.26}$$

$$b_2 = b_0 k_2 \tag{5.27}$$

von ersten beiden Handachsen und den Konstanten $k_i \in R$ abhängig, so dass insgesamt keine Redundanzen auftreten. Im Folgenden werden die Handachsen lediglich als a, b, c bezeichnet.

Die Vorwärtstransformation der Hand lässt sich nun wie folgt beschreiben:

1. Rotation um a_0 um die x-Achse,
2. Rotation um b_0 um die y-Achse,
3. Translation entlang der neuen z-Achse um Gelenkkörperlänge l_0,
4. Rotation um $a_0 k_1$ um die neue x-Achse,
5. Rotation um $b_0 k_1$ um die neue y-Achse,
6. Translation entlang der neuen z-Achse um Gelenkkörperlänge l_1,
7. Rotation um $a_0 k_2$ um die neue x-Achse,
8. Rotation um $b_0 k_2$ um die neue y-Achse,
9. Translation entlang der neuen z-Achse um Gelenkkörperlänge l_2,
10. Rotation um die neue z-Achse um c.

Mit dieser Struktur existiert keine Handsingularität im konstruktionsbedingt erreichbaren Bereich[18], weswegen Roboter mit diesem aufwändigeren Hand-Typus vorwiegend bei Lackieraufgaben Verwendung finden (z.B. beim ABB IRB 540). Leider wird diese Flexibilität mit einem erhöhten Rechenaufwand in den Kinematikmodulen erkauft, denn im Gegensatz zur Zentral- oder Winkelhand kann nicht direkt auf einen Handwurzelpunkt geschlossen werden, aus dessen Lage sich zunächst die Grundachsen bestimmen lassen. Daher wird hier ein Weg beschritten, bei dem die in Gleichung 5.4 beschriebenen Zusammenhänge ausgenutzt werden.

Zunächst wird bei der Lösung des IKP die Länge der gestreckten Hand ($a_0 = 0$, $b_0 = 0$) der Länge des letzten Grundachsen-Gelenkkörpers aufgeschlagen, und das IKP der Grundachsen wie bei den anderen Hand-Typen so gelöst, als wenn diese so verlängerte Handwurzel den TCP erreichen müsste. Die Orientierung wird dabei außer Acht gelassen. Ist diese Position nicht erreichbar, dann wird die Länge der Hand auf Null gesetzt und der Vorgang wiederholt. Für mindestens einen dieser beiden Fälle lässt sich eine Stellung der Grundachsen ermitteln, sofern die Position überhaupt erreichbar ist.

Mit der so errechneten Grundachsenstellung wird eine Vorwärtstransformation bis zur Handwurzel berechnet, woraus sich eine von der Hand zu überbrückende Orientierungs- und Positionsdifferenz ergibt. Aus der Orientierungsdifferenz wird das IKP der flexiblen Hand mittels des unten beschriebenen Verfahrens berechnet. Mit den nun vollständig berechneten Achsen werden das gesamte DKP und die resultierende Positionsabweichung zwischen Sollposition des TCP und der gerade ermittelten Lösung des DKP bestimmt. Der Differenzvektor $dTCP = TCPist - TCPsoll$ wird nun von der aktuellen Position der Handwurzel subtrahiert und der Vorgang wird wiederholt, wobei im ersten Schritt nun nicht mehr von einer gestreckten Hand ausgegangen wird, sondern mit den Winkeln des vorangegangenen Schrittes:

[18] Bei rein mathematischer Betrachtung ließen sich zu einer vorgegebenen Orientierung unendlich viele Lösungen und damit Konfigurationsräume finden, die durch singuläre Stellungen voneinander getrennt wären. Allerdings müssten die Kugelgelenke der Achsen dazu mehrfach rotationsfähig sein und die Konstruktion würde sich dabei selbst mehrfach durchdringen. Daher wird hier o.B.d.A. auf die mathematische Darstellung dieser Mehrdeutigkeiten verzichtet.

1. Berechnung der Grundachsen aus der Position der Handwurzel oder bei gestreckter Hand.
2. Vorwärtstransformation der Grundachsen zur Ermittlung der Handwurzelorientierung.
3. Berechnung der Orientierungsdifferenz zwischen Handwurzel und Handflansch.
4. Rücktransformation der Flexwrist aus dieser Orientierungsdifferenz.
5. Vorwärtstransformation des gesamten Roboters mit den Flexwrist-Achsen.
6. Berechnung des Positionsfehlers.
7. Wenn dieser gering genug ist, ist die Berechnung abgeschlossen.
8. Kompensation der Position der Grundachsen (Handwurzel) mit diesem Positionsfehler, damit der bestehende Fehler sich primär als Orientierungsfehler auswirkt.
9. zurück zu Punkt 1. mit der korrigierten Position der Handwurzel.

Dieses Verfahren konvergiert schnell und gelangt mit wenigen Iterationsschritten in den gewünschten Genauigkeitsbereich von wenigen Nanometern, so dass auch eine differenzielle Geschwindigkeitsberechnung ohne großen Aufwand möglich ist.

Nun muss auf die Lösung des IKP der Hand in Schritt 4 mittels Jacobi-Matrix eingegangen werden. Durch die Konstruktion der Hand führt die Bewegung einer Achse auch stets zu einer Positions- *und* Orientierungsänderung des Endeffektors. Durch die Verkopplung der Achsen hat die Hand dennoch nur drei Freiheitsgrade. Daher kann die Auswahl der Eingangswerte für die Rücktransformation auf drei nicht abhängige Werte beschränkt werden.

Hier soll von der Orientierungsdifferenz auf die Handachsenstellung geschlossen und die Position vernachlässigt werden. Das Ziel besteht darin, eine Achsstellung der Hand zu finden, bei der die Orientierungsmatrix der Vorwärtstransformation der Hand gleich der Sollorientierung ist.

Um die Berechnungen zu vereinfachen, sollte die Anzahl der unbekannten Ein- und Ausgangsvariablen möglichst minimiert werden, weshalb als Ausgangsgrößen lediglich die beiden ersten Handachs-Winkel a und b gewählt werden, denn der Winkel c lässt sich anschließend einfach aus der Verdrehung der Soll und Ist-Orientierung um die resultierende z-Achse ermitteln. Schwieriger ist die Wahl einer möglichst kompakten Orientierungsdarstellung ohne Definitionsgrenzen und Singularitäten im Arbeitsbereich. Darstellungsmöglichkeiten wie Roll-Pitch-Yaw, Euler oder Kugelkoordinaten u.a. sind hier ungeeignet, da deren Winkel bei 360° bzw. bei $\pm 180°$ umschlagen (”wrappen”), und daher nicht stetig sind. Ihre Verwendung in den partiellen Ableitungen der Jacobi-Matrix ist daher ausgeschlossen. Die hier bevorzugte Lösung verwendet den kompletten Einheitsvektor $W = \begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix}$ der Orientierungsmatrix $O = \begin{bmatrix} U & V & W \end{bmatrix}$ des DKP der Hand, da sich dieser stetig mit der Änderung der Achswinkel verhält. Gleichung 5.4 wird damit zu

$$\begin{bmatrix} \frac{\partial a}{\partial t} \\ \frac{\partial b}{\partial t} \end{bmatrix} = J \frac{\partial W}{\partial t} \tag{5.28}$$

und J hat somit die Form

$$J = \begin{bmatrix} \frac{\partial W}{\partial a} & \frac{\partial W}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{\partial W_x(a,b)}{\partial a} & \frac{\partial W_x(a,b)}{\partial b} \\ \frac{\partial W_y(a,b)}{\partial a} & \frac{\partial W_y(a,b)}{\partial b} \\ \frac{\partial W_z(a,b)}{\partial a} & \frac{\partial W_z(a,b)}{\partial a} \end{bmatrix} \tag{5.29}$$

Die Ist-Orientierungsmatrix ergibt sich aus der Berechnung des DKP ohne Berücksichtigung der Gelenkkörperlängen. Die Transformation eines Kugelgelenkes kann damit auf die folgende Form gebracht werden:

$$T_0(a,b) = \begin{bmatrix} \cos b & 0 & \sin b \\ \sin a \sin b & \cos a & -1.0 \sin a \cos b \\ -1.0 \cos a \sin b & \sin a & \cos a \cos b \end{bmatrix} \tag{5.30}$$

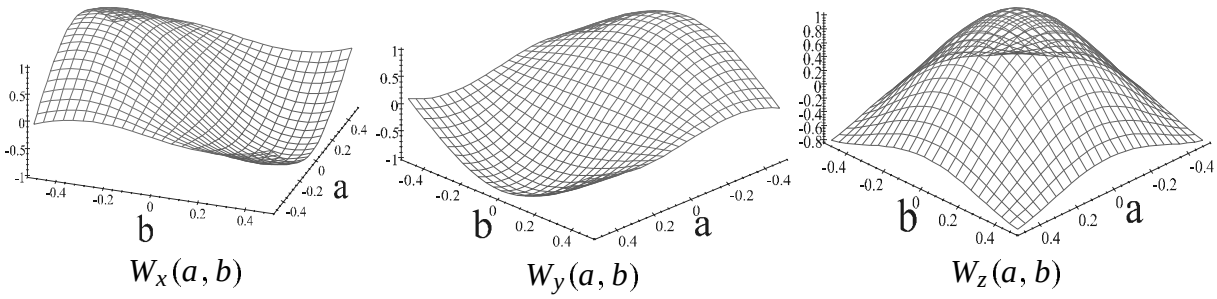
Die Gesamt-Orientierungs-Vorwärtstransformation der Hand kann damit als Verkettung dieser Transformation gebildet werden:

$$T_g(a,b) = T_0(a,b)T_0(k_1a,k_1b)T_0(k_2a,k_2b) = \begin{bmatrix} U & V & W \end{bmatrix} \tag{5.31}$$

mit U, V, W als Einheitsvektoren e_x, e_y, e_z der Orientierungsmatrix. Die Ausmultiplikation soll an dieser Stelle aus Platzgründen unterbleiben.

$$W_x(a,b) = T_g(a,b)_{1,3} \tag{5.32}$$
$$W_y(a,b) = T_g(a,b)_{2,3} \tag{5.33}$$
$$W_z(a,b) = T_g(a,b)_{3,3} \tag{5.34}$$

Die Grafiken in Abbildung 5.19 stellen den Verlauf der Komponenten des Einheitsvektors W abhängig von den Eingangswinkeln a und b (in rad) des Hand-DKPs dar. Es lässt sich erkennen, dass $W_y(a,b) = W_x(b,a)$ ist, und $W_z(|a|,|b|) = W_z(b,a)$.



$W_x(a,b)$ $W_y(a,b)$ $W_z(a,b)$

Abbildung 5.19: Verlauf des W-Vektors

Mit der Iteration der Universaltransformation [Bau92] ergibt sich bei genügend gutem Startvektor[19] für a und b bereits nach dem ersten Iterationsschritt eine akkurate Lösung, die häufig schon im Hundertstel-Grad-Bereich liegt:

1. Bestimmen der Startwerte für $\begin{bmatrix} a \\ b \end{bmatrix}$ aus einer vorher angelegten Tabelle oder, falls vorhanden, den Achswerten eines vorangegangenen Interpolationstaktes.
2. $W_{ist}(a,b)$ bestimmen und die Differenzorientierung als Differenzvektor der beiden Einheitsvektoren $W_{diff} = W_{soll} - W_{ist}$ ermitteln.
3. Ist W_{diff} genügend klein, so hat man das Ziel erreicht und es kann mit der Berechnung der letzten Achse fortgefahren werden, hier Punkt 9.
4. Berechnung von J, also $\frac{\partial W}{\partial a}$ und $\frac{\partial W}{\partial b}$mittels

$$\frac{\partial W}{\partial a} = W(a + da, b) - W(a, b) \tag{5.35}$$

und

$$\frac{\partial W}{\partial b} = W(a, b + db) - W(a, b), \tag{5.36}$$

mit da und db als klein gewählte Konstanten.

5. Die Matrix J ist nicht quadratisch. Daher muss zur Berechnung der erforderlichen Korrekturwerte für a und b die Pseudoinverse für unterbestimmte Systeme ermittelt werden:

$$J^+ = \left(J^T * J\right)^{-1} * J^T \tag{5.37}$$

6. Die Korrekturwerte für a und b ergeben sich nun zu:

$$\begin{bmatrix} a_k \\ b_k \end{bmatrix} = J^+ W_{diff} \tag{5.38}$$

7. Damit folgt für die korrigierten Achswinkel:

$$\begin{bmatrix} a \\ b \end{bmatrix} \leftarrow \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} a_k \\ b_k \end{bmatrix} \tag{5.39}$$

[19] Während einer Interpolation wird als Startwert die Achsstellung des letzen Interpolationstaktes verwendet, und dieser liegt naturgemäß immer dicht am gesuchten Punkt.

8. Fortsetzung bei Punkt 2.

9. Die Stellung der letzten Achse ergibt sich aus dem Winkel, den die X-Einheitsvektoren der Soll- und Ist-Orientierungsmatrizen einschließen:

$$c = arg(U_x soll + iU_x ist). \tag{5.40}$$

Damit sind alle Achsen eines Roboters mit der speziellen flexiblen Hand bestimmt. Da bei der hier vorgestellten Lösung J numerisch aus der Vorwärtstransformation der Hand bestimmt wird, keine partiellen, kinematikspezifischen Ableitungen in analytischer Form in den Quellcode eingeflossen sind und lediglich das DKP bekannt zu sein braucht, eignet sich dieses Vorgehen daher auch zur Berechnung der IKPs von Robotern mit anderen Handtypen. Testweise wurde es gegen die analytischen Verfahren der Zentral- und Winkelhand ausgetauscht und konnte die Funktionsweise belegen.

Wird eine Rücktransformation außerhalb einer Bahninterpolation berechnet, bei der nicht auf Achsstellungen eines vorangegangenen Interpolationstaktes zurückgegriffen werden kann, geschieht die Gewinnung des Startvektors aus Zeitgründen durch eine Tabelle, die bei der Initialisierung der Datentypen für die Hand angelegt wird. Der erforderliche Speicherplatz hält sich hierbei in Grenzen, da die Eingangsgrößen zwei Soll-Winkel in Polarkoordinaten und die Ausgangsgrößen die beiden Achsen a und b sind. Mit etwa 40 Eingangsvektoren pro Quadrant für W erreicht der beschriebene Algorithmus nach einer einstelligen Anzahl an Durchläufen die gewünschte Genauigkeit.

Da die Universal-Rücktransformation innerhalb dieses Verfahrens auf nur zwei Freiheitsgrade reduziert werden konnte, hält sich erstens die Rechenzeit der Iteration sehr in Grenzen und zweitens lässt sich so ein kleiner, ausreichender Raum von Startvektoren anlegen, der keine Belastung des Rechners darstellt. Daraus ergibt sich der entscheidende Vorteil des Verfahrens gegenüber der Standard-Universaltransformation des gesamten Roboterarmes, indem das IKP an beliebigen Stellen, z.B. zur Erreichbarkeitsüberprüfung, berechnet werden kann und die Möglichkeit der Vorgabe der kinematischen Struktur (right/left, above/below) voll erhalten bleibt.

5.5.5 Lineare Achstransformationen

In den vorangegangenen Kapiteln wurden die wichtigsten Klassen von Kinematiktypen, die von der Steuerung unterstützt werden, beschrieben. Soweit erforderlich wurde auch detaillierter auf die mathematische Lösung des IKP eingegangen. Dabei wurde bisher nicht erläutert, auf welche Weise die real existierende Robotervielfalt einer Klasse auf das betreffende Kinematikmodul abgebildet werden kann. Zunächst wird auf die zulässigen Unterschiede der Roboter eingegangen, die auf eine Klasse abgebildet werden können:

- Die Nullstellungen der Roboterachsen sind roboterspezifisch.
- Die Achsanschläge sind roboterspezifisch.
- Die Verfahr-/ Drehrichtungen der Achsen sind roboterspezifisch.

- Einige Roboter enthalten Parallelogramme.
- Die Achsen mancher Roboter sind softwaretechnisch, elektronisch oder mechanisch verkoppelt.
- Die Darstellung der Achswinkel enthalten teilweise konstante Werte[20].
- Die Darstellung der Achswinkel lässt sich durch Vertauschung einzelner Achswinkel auf die Darstellung in den Kinematikmodulen abbilden.

Diese Differenzen lassen sich grundsätzlich durch roboterspezifische Implementierungen von IKP und DKP in speziellen Kinematikmodulen berücksichtigen. Das Ziel dieser Arbeit ist es aber, ein Höchstmaß an Flexibilität und an Parametrierbarkeit zu erreichen, wobei ein Kinematikmodul für möglichst viele Roboter verwendet werden soll. Deshalb wurde ein Verfahren entwickelt, mit dem sich sämtliche der angeführten Differenzen vereinheitlichen lassen. Dabei werden verschiedene Schnittstellen innerhalb der Steuerung, bei denen die Achsstellungen ausgetauscht werden, um Konvertierungsfunktionen bzw. deren Inverse erweitert. Diese Umrechnungsfunktionen sollen von nun an als "lineare Achstransformationen" bezeichnet werden, da sich dieser Name auf Grund ihrer Form anbietet:

$$q^{out} = T_{ax}(q^{in}) \tag{5.41}$$

$$T_{ax}(q) = M^{trans} q + L \tag{5.42}$$

mit

$T_{ax}(q)$: lineare Achstransformationsfunktion,

q: Achswinkel-Vektor,

q_i^{out}: Ausgangs-Achswinkel der Achse i,

q_i^{in} : Eingangs-Achswinkel der Achse i,

L_i : Differenzwinkel/-Strecken zwischen der Achsnullstellung der Ein- und Ausgangsachswinkel,

M^{trans}: Achstransformationsmatrix.

Bei der Umrechnung von Geschwindigkeiten und Beschleunigungen entfällt das konstante Glied L innerhalb von T_{ax}.

Zunächst wird auf die Schnittstelle zu den Kinematikmodulen eingegangen, da der Einsatz der linearen Achstransformationen an dieser Stelle die Flexibilität der Steuerung und die Skalierbarkeit der Kinematikmodule erheblich steigert. Hierbei rechnet die lineare Achstransformation

[20] Z.B. verwendet Mitsubishi bei den fünfachsigen Knickarmrobotern sechs Achswinkel, bei denen der Wert für Achse vier stets Null ist.

die Achsdarstellungen innerhalb der Kinematikmodule in die der Bewegungssteuerung um. q^{in} entspricht also der Achsdarstellung im Kinematik-Bereich, q^{out} den Stellungen, die bei PTP-Bewegungen von der Bewegungssteuerung berechnet werden.

Beispiel:

Abbildung 5.20 stellt eine Kinematik mit zwei Parallelogrammen dar. Das erste befindet sich zwischen der 2. und 3. Achse, das zweite zwischen der 3. und 4. Achse. Parallelogramme führen dazu, dass die Winkel aus den Kinematikmodulen nicht direkt verwendet werden können. Die Abbildung stellt einen Ablauf dar, der den Missstand ohne Achsumrechnung verdeutlicht. Die Winkel q^T sind dabei die Winkel der Kinematikmodule und q^R die Winkel der Bewegungssteuerung. Soll nur die 2. Achse des Roboters (q_1^T) in negative Richtung bewegt werden, so würde sich der Roboterarm ohne Parallelogramm wie in Abb. 5.20 (2) des Bildes nach vorn neigen, da das Kinematikmodul von einem konstanten Winkel der dritten Achse ausgeht. In Wahrheit bleibt die Orientierung allerdings konstant, wie in Abb. 5.20 (3) dargestellt, da die Achsen durch das Parallelogramm verkoppelt sind. Solche und andere lineare Verkopplungen beliebiger Achsen untereinander lassen sich mit linearen Achstransformationen auf die nicht-verkoppelten Achsen der Kinematikmodule abbilden. Bei der Aufstellung der linearen Achstransformation für die-

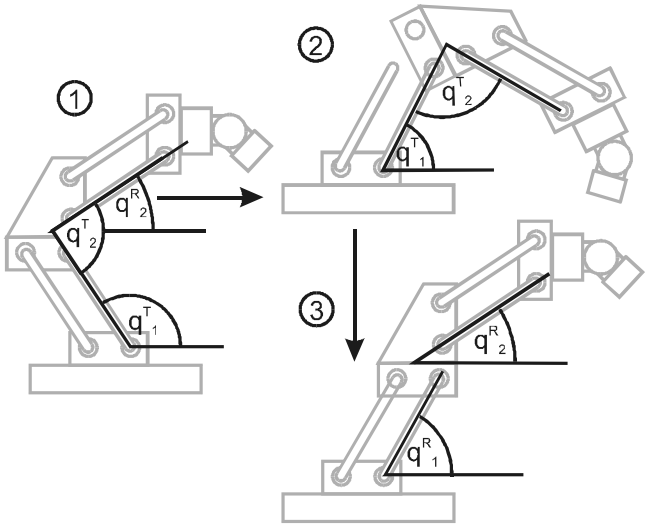


Abbildung 5.20: Roboter mit Parallelogrammen

ses Beispiel wird der Übersichtlichkeit halber lediglich das erste Parallelogramm berücksichtigt. Innerhalb der Robotersteuerung ist die Verallgemeinerung auf beliebig viele Achsen implementiert. Bei der Umrechnung von Achswinkeln, die aus der Rücktransformation stammen, müssen die Winkel der Bewegungssteuerung q_i^R mit den Winkeln der Transformation q_i^T ausgedrückt werden:

$$q_1^R = q_1^T,$$

$$q_2^R = q_2^T + q_1^T - 180^\circ$$

Die korrespondierende lineare Achstransformation hat damit für die beiden Achsen, die in das erste Parallelogramm involviert sind, folgende Form:

$$\begin{bmatrix} q_1^R \\ q_2^R \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} q_1^T \\ q_2^T \end{bmatrix} + \begin{bmatrix} 0 \\ -180^\circ \end{bmatrix} \tag{5.43}$$

Bei der Lösung des DKP müssen die Achswinkel, die dem Kinematikmodul übergeben werden, aus den Winkeln der Bewegungssteuerung durch die inverse lineare Achstransformation berechnet werden:

$$\begin{bmatrix} q_1^T \\ q_2^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} q_1^R \\ q_2^R \end{bmatrix} + \begin{bmatrix} 0 \\ 180^\circ \end{bmatrix} \tag{5.44}$$

Zusammenfassung

Die meisten der oben aufgelisteten erlaubten Unterschiedlichkeiten vom Anfang dieses Kapitels, lassen sich mit der linearen Achstransformation an dieser Stelle bereits lösen:

- Die Differenzen der Nullstellungen der Roboterachsen zwischen den implementierten Kinematikmodulen und der Bewegungssteuerung werden durch das Glied L der linearen Achstransformation bestimmt.
- Die Achsanschläge werden während der Initialisierung durch das Simulationssystem mittels linearer Achstransformation auf die VRC-interne Darstellung abgebildet.
- Die Verfahrrichtungen der Achsen lassen sich durch die Vorzeichen auf der Diagonalen der Achstransformationsmatrix parametrieren.
- Manche Roboter enthalten Parallelogramme. Dieses Beispiel wurde exemplarisch erläutert.
- Sämtliche lineare Verkopplungen können mit dem vorgestellten Verfahren ohne Änderungen am Quellcode abgebildet werden.
- Stets konstante Werte innerhalb der Achsdarstellung werden durch Werte in dem Vektor L und Nullen auf den entsprechenden Positionen der Diagonalen der Matrix M^{trans} realisiert.
- Achsvertauschungen lassen sich durch den Aufbau der Achstransformationsmatrix abbilden. Würden in dem oben genannten Beispiel die Achsen der Bewegungssteuerung gegenüber denen des Kinematikmoduls vertauscht, wäre die Matrix zu transponieren.

- Mit den linearen Achstransformationen lassen sich Getriebe innerhalb des Simulationssystems realisieren, bei denen der Winkel des Abtriebs proportional zum Eingangswinkel ist. Besteht das Getriebe aus mehreren Teilen, so lassen sich auch die Stellungen sämtlicher interner Getriebeachsen bestimmen.

Mit den linearen Achstransformationen zwischen den Kinematikmodulen und der Bewegungssteuerung wurden die Flexibilität und Skalierbarkeit und damit die Wiederverwendbarkeit bzw. die Universalität der implementierten kinematischen Funktionalitäten erheblich gesteigert.

Lineare Achstransformationen zwischen anderen Modulen des Systems

Zur weiteren Steigerung der Flexibilität des Steuerungs- und Simulationssystems werden lineare Achstransformationen auch bei Schnittstellen zu anderen Steuerungsmodulen eingeführt und verwendet.

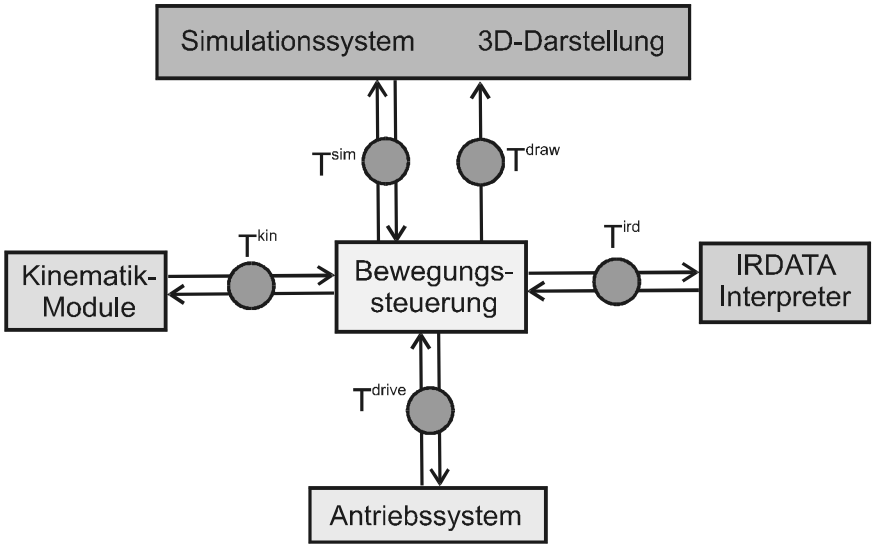


Abbildung 5.21: Systemschnittstellen mit linearen Achstransformationen

Abbildung 5.21 zeigt die Systemmodule, zwischen denen Achsdaten ausgetauscht werden und an welchen Stellen diese linearen Achstransformationen angewendet werden. Dabei wurden folgende modulspezifischen Transformationen eingeführt.

$\mathbf{T}^{kin}$ wurde im vorangegangenen Kapitel ausführlich erläutert und dient der Anpassung der implementierten Kinematikklassen an die zu unterstützenden Roboter.

$\mathbf{T}^{ird}$ dient der Umrechnung von programmspezifischen Achsdarstellungen. Mit Hilfe der an dieser Stelle verwendeten Achsumrechnung können bereits bestimmte sprachspezifische Eigenheiten abgefangen und korrekt weiterverarbeitet werden, wie z.B. Achsstellungen mit irrelevanten Komponenten, wie bei einem fünfachsigen Mitsubishi Roboter, dessen Achsdarstellung auch die Stellung der nicht vorhandenen vierten Achse enthält.

$\mathbf{T}^{sim}$ dient der Umrechnung von Achsstellungen zwischen Simulationssystem und Bewegungssteuerung. Die Winkel, die an dieser Stelle mit dem Simulationssystem ausgetauscht werden,

dienen der Ein- und Ausgabe. Sie werden vom Anwender vorgegeben und dienen der Anzeige der aktuellen Achsstellung und können z.B. in Positionslisten übernommen werden. Diese Winkel werden nicht zum Zeichnen verwendet.

$\mathbf{T}^{draw}$ dient der Vereinfachung der Darstellung des Roboters innerhalb des Simulationssystems. Beim Zeichnen komplexerer kinematischer Ketten, etwa mit Verzweigungen und Parallelogrammen bzw. kinematischen Schleifen oder Robotern mit flexibler Hand (siehe Kapitel 5.5.4), müssen zusätzliche Gelenkkörper gezeichnet werden, deren Stellung sich mittels linearer Achstransformationen aus den Achswinkeln ermitteln lassen, die innerhalb der Steuerung bereits berechnet wurden. Diese Schnittstelle wurde konzipiert, um den Rechenaufwand zu minimieren, indem die Steuerung dem Simulationssystem die bereits berechneten Winkel über diesen Weg mitteilt. Da die 3D–Darstellungsfunktion innerhalb des Simulationssystems die kinematische Kette sukzessive aufbaut und dabei stets relativ zum letzten Gelenkkörper arbeitet, müssen die Achsstellungen an dieser Stelle wieder von Verkopplungen bereinigt sein, ähnlich den Kinematikmodulen. Da keine Winkel aus dem Darstellungsmodul in die Steuerung zurückfließen, ist dieser Kanal als ”Einbahnstraße” ausgelegt.

$\mathbf{T}^{drive}$ dient der Umrechnung der berechneten Achsstellung in die innerhalb des Antriebssystems verwendete Darstellung. Mit dieser Umrechnung kann innerhalb des Antriebssystems eine Verkopplung vorgenommen, aufgehoben oder simuliert werden. Hiermit ist es möglich, das Verhalten des realen Roboterarmes ohne mechanisches Parallelogramm so einzustellen, als enthielte er ein oder mehrere Parallelogramme.

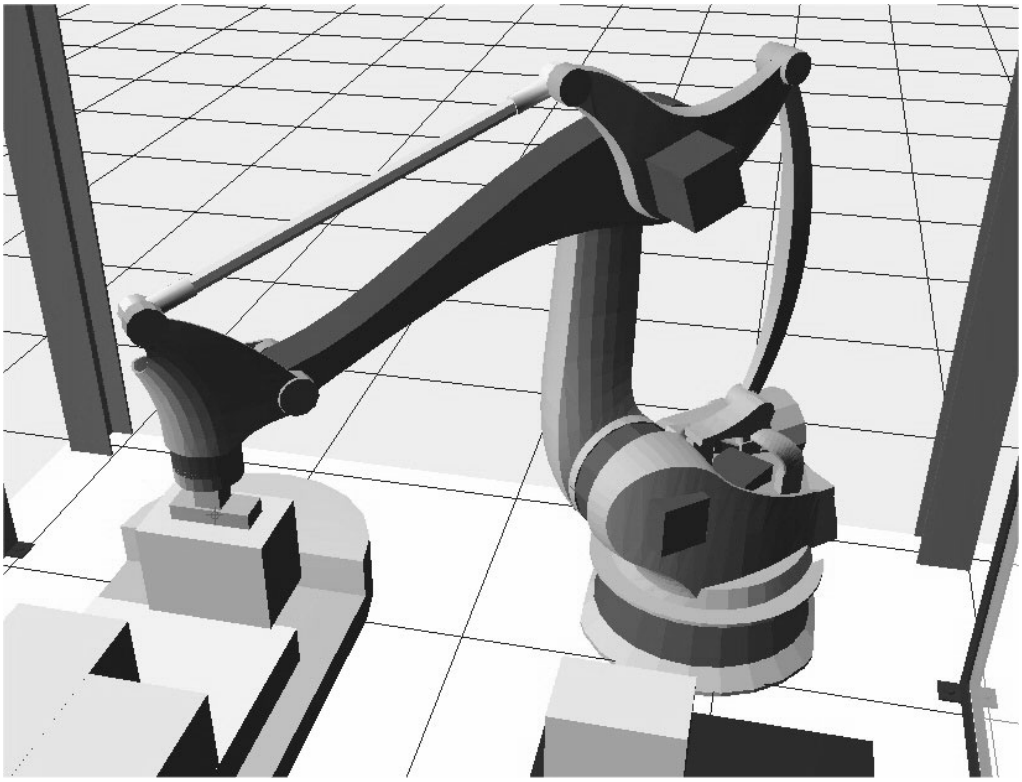


Abbildung 5.22: KUKA KR 180 PA Roboter in einer Arbeitszelle

Abbildung 5.22 zeigt eine simulierte Arbeitszelle mit einem KR 180 PA Roboter der Firma Kuka. Dieser Roboter kann mittels linearer Achstransformationen auf die Kinematik des ETS-VII Roboters aus Kapitel 5.5.4 abgebildet werden, obwohl er statt sechs nur über vier angetriebene Achsen verfügt. Die Stellung der zwangsgeführten Gestänge, die sich mit drei weiteren Winkeln beschreiben lassen, sind mittels linearer Achstransformationen aus den Winkeln der Rücktransformation bestimmbar und werden in $\mathbf{T}^{draw}$ definiert. Damit kann das Simulationssystem die Gelenkkörper in ihrer korrekten Lage zeichnen.

Die Lineare Achstransformation $\mathbf{T}^{kin}$ zwischen Kinematikmodul und der Bewegungsteuerung lautet damit

$$\mathbf{T}^{kin} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} q + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

und die Transformation zur Berechnung der Winkel für die Anzeige inklusive der zwangsgeführten Gelenkkörper ergibt sich bei diesem System zu:

$$\mathbf{T}^{draw} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} q + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Dabei ist q der Vektor der Achsstellungen der Bewegungssteuerung. Die Offsets L sind hierbei Null, da die Konfiguration und Modellierung des Aufbaus der kinematischen Kette mittels DH-Parametern bereits passend erfolgte und dadurch keine Differenzen zwischen der Nulllage der Achsen innerhalb der Bewegungssteuerung und dem Grafik-Modul zum Darstellen der kinematischen Kette bestehen.

Parametrierung der linearen Achstransformationen

Ohne explizite Parametrierung sind alle im System verwendeten linearen Achstransformationen zunächst Einheits-Abbildungen[21]. Bei Bedarf können während der Initialisierung der Kinematik

[21] Einheitsabbildungen bilden den Eingangsvektor ohne Veränderung auf den Ausgangsvektor ab, indem die Matrix M^{trans} die Einheitsmatrix ist, und der Versatz L Null.

oder im Start-Up-Code eines IRDATA-Programms die einzelnen Transformationen modifiziert werden. Dabei ist jeweils nur die direkte Transformation zu definieren, die inverse Transformation wird -sofern möglich- automatisch ermittelt. Zur Bildung der Inversen Transformationsmatrix $\left(M^{trans}\right)^{-1}$ werden sämtliche Null-Zeilen und Null-Spalten eliminiert. Ist M^{trans} danach quadratisch, wird die Inverse gebildet, und diese wieder um die Null-Zeilen und -Spalten erweitert- allerdings in transponierter Nummerierung. Ist $\left(M^{trans}\right)$ nicht quadratisch, wird die Pseudoinverse $\left(M^{trans}\right)^{+}$ verwendet. Diese ergibt sich zu:

$\left(M^{trans}\right)^{+} = \left(\left(M^{trans}\right)^{T}\left(M^{trans}\right)\right)^{-1}\left(M^{trans}\right)^{T}$ für Anzahl der Eingangsvariablen $<$ Anzahl der Ausgangsvariablen und

$\left(M^{trans}\right)^{+} = \left(M^{trans}\right)^{T}\left(\left(M^{trans}\right)\left(M^{trans}\right)^{T}\right)^{-1}$ für Anzahl der Ausgangsvariablen $<$ Anzahl der Eingangsvariablen.

Diese Vorschrift ermöglicht die Abbildung aller Robotertypen der kinematischen Grundklassen auf die implementierten Methoden und Verfahren und stellt damit eine sehr effiziente Neuerung im Bereich der Robotersimulation dar.

Kapitel 6

Die Bewegungssteuerung

Die Bewegungssteuerung ist ein wichtiger Bestandteil einer Robotersteuerung bzw. einer Robotersteuerungssimulation. Ihre Aufgabe ist es, zeit-diskret Achsstellungen und Geschwindigkeiten zu ermitteln, die dazu führen, dass der simulierte oder reale Roboterarm, sowie ggf. periphere Geräte, die gewünschten Bewegungen ausführen. Basierend auf einem Interpolationstakt wird die vorgegebene Bahn abgetastet. Bahnarten, die sich seit Beginn der Robotertechnik etabliert haben, wurden bereits in Kapitel 3.7 erwähnt und werden in der Literatur ausführlicher beschrieben: [dV95], [Meh85], [Hei93].

Innerhalb des Simulationssystems besteht die Aufgabe einer VRC darin, sämtliche Bewegungs-Interpolationsverfahren der unterstützten Roboter zur Verfügung zu stellen. Die Qualität der Simulation zeigt sich in den möglichst kleinen Differenzen zum realen System. Eine stets exakt identische Simulation setzt die genaue Kenntnis der Algorithmen und Verfahren der realen Steuerungs-Pendants voraus oder macht den Einsatz des originalen Steuerungscodes erforderlich, wie es in RRS-I vorgesehen ist [RRS94]. Da die Verfahren, mit denen die realen Steuerungen arbeiten, meist nicht zur Verfügung stehen, wird die Bewegungsteuerung dieser Arbeit umfangreich parametrierbar ausgelegt, um damit eine hohe Abbildungsgenauigkeit realisieren zu können. Ziel dabei ist es, sämtliche relevanten Systeme ohne weiteren Programmieraufwand genau simulieren zu können, in dem die VRC korrekt parametriert wird.

Beim Einsatz als Robotersteuerungssystem zur Steuerung eines realen Roboters (R/VRC) entfällt i.A. der Anspruch einer hohen Abbildungsgenauigkeit, da nicht das Verhalten eines anderen Industrieroboters bestmöglich nachempfunden werden muss, sondern die Steuerungsprogramme mit dem System selbst erzeugt und ausgeführt werden. Hier profitiert der Anwender von der besonders hohen Flexibilität, weil er die Steuerung in zahlreichen Parametern feinjustieren und seiner Aufgabe bestmöglich anpassen kann.

6.1 Systemstruktur der Bewegungssteuerung

Die in dieser Arbeit beschriebene R/VRC, die möglichst alle verfügbaren Roboter originalgetreu simulieren soll, muss über ein Systemkonzept verfügen, auf das sich die Unterschiedlichkeit der zu simulierenden Systeme abbilden lässt. Da mit dieser Steuerung nicht nur simuliert, sondern auch reale Roboter angesteuert werden sollen, erwachsen daraus weitere Anforderungen an das

Design der Bewegungssteuerung, die asynchron von anderen Steuerungskomponenten autonom laufen muss, indem die Bewegungssteuerung vollständig in einen eigenen ”Thread” ausgelagert wird. Die Struktur der reinen VRC synchronisiert sich durch den Code selbst, d.h. die zeitlichen Zugriffe auf auszutauschende Daten können sich, wie bei Singlethread-Systemen, naturgemäß nicht überlappen, so dass hier keinerlei Synchronisationsmechanismen wie ”Semaphore” und ”Events” erforderlich waren. Dabei wurde der Robotersteuerung nur dann Rechenzeit zugeteilt, wenn sie durch das Simulationssystem aufgerufen wurde. Dieser Vorgang wurde in bereits Abbildung 5.1 durch die breiten Pfeile verdeutlicht. Da das Simulationssystem teilweise sehr rechenzeitintensive Operationen auszuführen hat, kann zwischen den Aufrufen eine relativ lange Zeit vergehen, etwa beim Aktualisieren der Arbeitszelle auf dem Bildschirm. Je nach verwendetem System und Komplexität des geometrischen Modells kann dies sogar zu einer ruckartigen Darstellung von nur einzelnen Bildern pro Sekunde führen. Diese Struktur lässt keinerlei Berechnungen zwischen den Aufrufen durch das Simulationssystem zu. Daher ist diese Struktur für die pünktliche Erzeugung von Achswerten nicht geeignet. Die Verhältnisse im Betrieb als reale Robotersteuerung stellt Abbildung 5.2 dar, wobei die Bewegungssteuerung autonom in einem eigenen ”Thread” mit Echtzeit-Priorität läuft. Auf die Bewegungssteuerung dieser Struktur wird im Folgenden genauer eingegangen[1].

Die Systemstruktur der Bewegungssteuerung ist in Abbildung 6.1 dargestellt, organisiert in funktionalen Blöcken. Der Programm- und Datenfluss ist mit Pfeilen gekennzeichnet. Ein aus dem Programminterpreter eintreffender, gültiger Bewegungssatz wird in den Mechanismus einer Warteschlange eingetragen, die noch nicht ausführbare Bewegungen enthält[2] (1)[3]. Auf diesen Vorgang wird in Kapitel 6.1.2 genauer eingegangen. Sobald die Bewegungsinterpolation bereit für einen Bewegungsbeginn ist, wird die korrespondierende Bewegung aus der Warteschlange entnommen und einheitlich auf die Interpolationsdatentypen abgebildet (2). Hierbei werden automatisch die Bewegungs-Datentypen und das Zielkoordinatensystem auf ein einheitliches Format gebracht und dazu gegebenenfalls transformiert. Bei diesen Berechnungen wird implizit überprüft, ob die Position grundsätzlich erreichbar ist und prinzipiell von der Ausgangsstellung erreicht werden kann, denn Erreichbarkeit heisst bei Bahnbewegung nicht unbedingt, dass die Zielposition auch von der gegenwärtigen Position aus angefahren werden kann. Das scheitert etwa dann, wenn die Zielposition in einer anderen Konfiguration oder mit anderen Achs-Turnangaben vorgegeben ist. Danach werden die Daten der Profil-Planung (3) übergeben, in der u.a. das Geschwindigkeitsprofil berechnet wird.

Die Profil-Planung ist hierbei ein Modul, das in der Lage ist, sowohl PTP-, als auch kartesische Bahnen zu planen. Die resultierenden Polynome für Beschleunigungs- und Bremsrampen werden an die Interpolationsdatenstruktur übergeben. Im Allgemeinen wird eine Profil-Planung nur einmal vor dem ersten Interpolationstakt berechnet, danach bleibt das Geschwindigkeitsprofil konstant. Bei speziellen Bewegungsarten jedoch kann sich das Bewegungsziel während der Bewegung in jedem Takt ändern und die Profil-Planung muss in jedem Interpolationstakt aufgeru-

[1] Die Komplexität der Struktur im Simulationsbetrieb stellt funktional eine Untermenge des Betriebs zur Ansteuerung realer Roboter dar, sie soll daher hier nicht weiter erläutert werden. Einzelheiten über die Struktur des Simulationsbetriebes finden sich bereits auch in [dV95].

[2] Diese Bewegungssätze können dann noch nicht ausgeführt werden, wenn der Roboter noch keine Zeit dafür hat, z.B. weil der korrekte Servozustand noch nicht eingenommen wurde oder die vorhergehende Bewegung noch andauert.

[3] Die in Klammern angegebenen Zahlen sind eine Referenz auf die eingekreisten Ziffern im Bild 6.1.

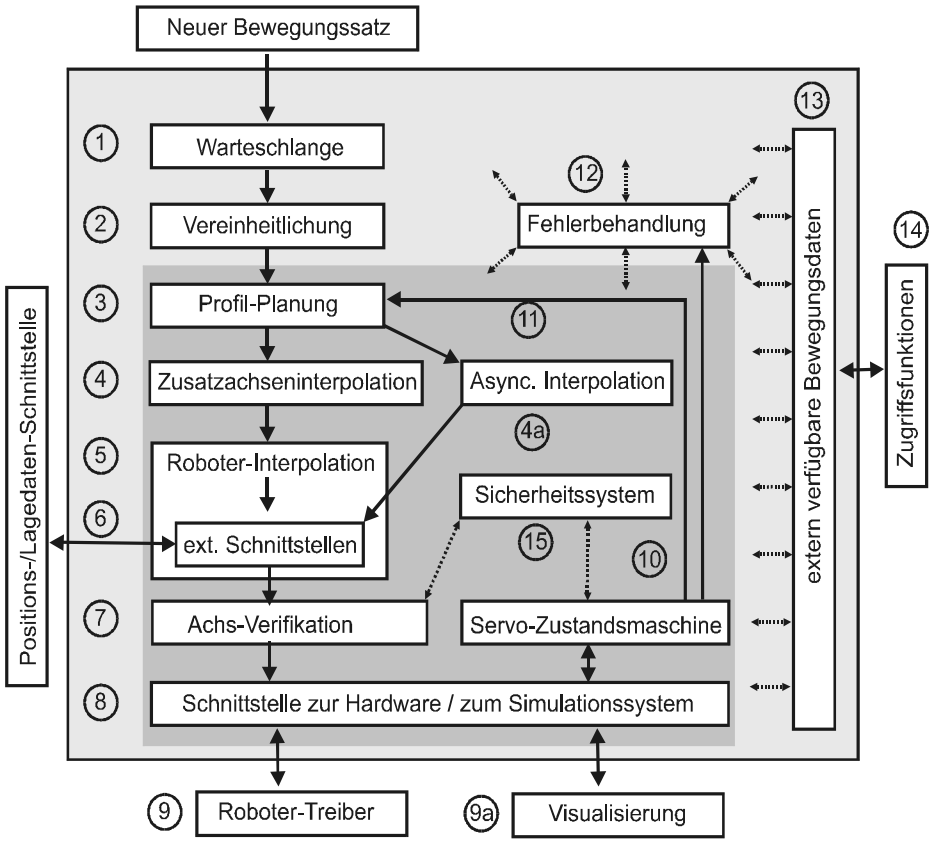


Abbildung 6.1: Struktur der Bewegungssteuerung

fen werden. Beispielsweise dann, wenn eine externe Sensorsteuerung das Bewegungsziel setzt, oder wenn sich die Soll-Bahngeschwindigkeit ändert. Daher befindet sich dieser Block innerhalb der in der Grafik dunkel-grau hinterlegen Interpolationsschleife von Punkt (3) bis (11). Die Profil-Planung stellt alle zur Interpolation erforderlichen Daten zur Verfügung, so dass unmittelbar danach die Achsstellungen des Taktes berechnet werden können. Dabei werden grundsätzlich zunächst die Zusatzachsen berechnet (4), da aus diesen Stellungen gegebenenfalls Positionen von internen oder externen Geräten resultieren, die das Roboterkoordinatensystem betreffen. Das ist etwa der Fall, wenn der Roboter auf einer Schiene steht oder der Roboter in den Koordinaten eines Dreh-/Schwenktisches verfahren soll. Auf die genauen mathematischen Verfahren wird an dieser Stelle nicht eingegangen. Grundlagen hierzu findet man z.B. in [dV95] und [Dzi92].

Parallel hierzu werden gegebenenfalls als asynchron definierte Achsen zeitgleich zur kinematischen Kette interpoliert (4a). Auf diese besondere Interpolationsart wird in Kapitel 6.3 genauer eingegangen. Nach der Interpolation der Zusatzachsen bzw. asynchroner Achsen wird die Interpolation des Roboters aufgerufen (5), die in Abbildung 6.2 detaillierter dargestellt ist. Hierbei wird je nach Interpolationsart in die Funktionen zur Berechnung der Takt-Daten gesprungen. Bei kartesischen Bewegungen kann die berechnete Position über externe Schnittstellen (6) modifiziert werden, auf die in Kapitel 7.6 eingegangen wird. Danach erfolgt bei Bahnbewegungen die Lösung des IKP mit Hilfe der Kinematikmodule. Jetzt stehen, unabhängig von der Bewegungsart, die Achsstellungen zur Verfügung. Auch diese können über die externen Schnittstellen

modifiziert werden.

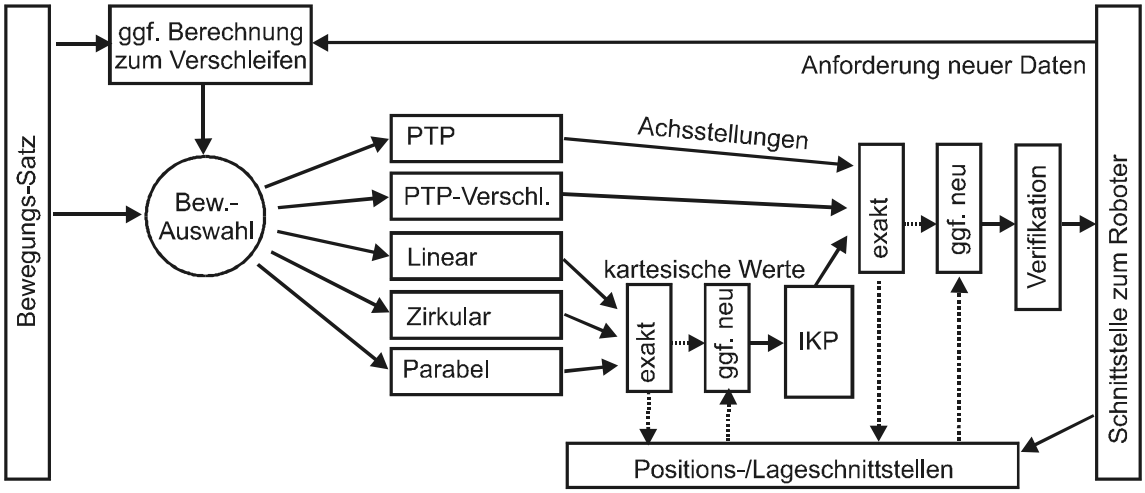


Abbildung 6.2: Struktur der Roboter-Bewegungsinterpolation

Die Verifikation der Achsstellungen (7) wird unabhängig von der Bewegungsart ausgeführt. Hierbei werden die Achsstellungen, Geschwindigkeiten und Beschleunigungen auf Gültigkeit, Erreichbarkeit und Grenzwerte überprüft. Tritt hier während der Interpolation ein Fehler auf, so wird je nach Schwere des Fehlers bahntreu angehalten oder mit maximaler Achsbeschleunigung abgebremst. Sind die Achswerte verifiziert, so können die Werte dem Schnittstellenmodul (8) übergeben werden, das mit dem Robotertreiber kommuniziert. Bei einem angeschlossenen Roboter werden die Achswerte der Hardware bzw. dem Roboter übertragen (9). Ist kein Robotertreiber vorhanden, so wird ein generischer, idealisierter Roboter nur zur Simulation angenommen. Das Schnittstellenmodul stellt auch die Achs-Istwerte und den aktuellen Servozustand zur Verfügung und fordert die Berechnung neuer Achswerte von der Bewegungssteuerung an. Bei einem angeschlossenen Roboter ist zur Sicherstellung korrekter Übergänge zwischen Servozuständen eine Zustandsmaschine erforderlich (10), auf die in Kapitel 7.3 genauer eingegangen wird.

Nachdem die Schnittstelle zur Hardware neue Sollwerte angefordert hat und der Zustand des Systems evaluiert wurde, wird der nächste Interpolationstakt berechnet (11). Die Fehlerbehandlung (12) ist eine globale Einrichtung innerhalb der Steuerung. Auf die Darstellung aller ihrer Kommunikationskanäle wurde der Übersichtlichkeit halber verzichtet. Daten, die auch außerhalb der Bewegungssteuerung zur Verfügung stehen sollen (13), werden mittels Zugriffsfunktionen innerhalb des gesamten Systems zur Verfügung gestellt, beispielsweise um innerhalb des Ereignisinterpreters die aktuelle Bahnlänge abfragen zu können, wenn ein Bahnschalt-Ereignis definiert wurde.

6.1.1 Bewegungsanforderungen durch Übergabe von Bewegungssätzen

Innerhalb des Roboterprogramms wird eine Bewegung definiert, indem zunächst mittels Parametrierungsfunktionen die Einstellungen der folgenden Bewegung vorgegeben werden und danach mit der Bewegungsanweisung das Bewegungsziel und die Bewegungsart bestimmt werden. Die

Gesamtheit der Daten, die eine Bewegung beschreibt, wird als ”Bewegungssatz” bezeichnet. Abbildung 6.3 stellt die Definition eines solchen Bewegungssatzes dar.

```
typedef struct one_movement{
    /* Datenstruktur zur Beschreibung eines Bewegungssatzes.                    */
    BOOL          valid;    /* Wird auf TRUE gesetzt, wenn Satz komplett definiert  */
    BOOL          startflag;/* Wird auf TRUE gesetzt, wenn die Bewegung starten soll*/
    MOV_TARGET    target;   /* Zielposition                                       */
    MOV_TYPE      type;     /* Art der Bewegung                                   */
    MOV_PARAMETERS    param;/* Parameter der Bewegung                             */
    MOV_VOLATILE_DATA mvdat;/* online Daten                                       */
    int  queuelen;          /* Warteschlangenlänge der Achssollwerte im Kernelmode  */
    int  HSTI_Mode;         /* high speed torque interface - mode                 */
    int  Number;            /* Nummer des Bewegungssatzes                         */
}ONE_MOVEMENT;
```

Abbildung 6.3: Datenstruktur eines Bewegungssatzes

Dieser Bewegungssatz enthält im Wesentlichen die Unterdatentypen zur Definition des Bewegungsziels, der Bewegungsart und der Bewegungsparameter, die in Abbildung 6.4 bis 6.6 abgedruckt sind. Das Bewegungsziel kann innerhalb des Roboterprogramms mittels verschiedener Datentypen vorgegeben werden, wobei diese Datentypen jeweils auch noch auf verschiedene Art und Weise interpretiert werden können: Die kartesische Positionsangabe vom Typ FRAME in der Datenstruktur setzt sich aus einem dreidimensionalen Vektor V und einer Orientierungsmatrix O zusammen[4]. Bei jedem kartesisch vorgegebenen Ziel können jeweils V, O oder beide gültig sein. Fehlende Komponenten werden durch Werte des vorangegangenen Bewegungsziels ergänzt. Zusätzlich können Bewegungen relativ zu anderen Objekten und Positionen des Systems angegeben werden, so auch zur aktuellen Position. Dabei wird die kartesische Position des Zielpunktes in Weltkoordinaten durch eine homogene Transformation aus der aktuellen Position und der angegebenen ermittelt. Auch bei Bewegungszielvorgaben in Achskoordinaten sind achsrelative Bewegungen möglich. Folgende Ziel-Typen sind daher definiert:

- ZIEL_TYP_FRAME: Die Zielangabe ist ein vollständiges Frame,
- ZIEL_TYP_MAT: Bei der Zielangabe ist lediglich die Orientierungsangabe O des Datentyps FRAME gültig,
- ZIEL_TYP_VECTOR: Bei der Zielangabe ist lediglich der Vektor V des Datentyps FRAME gültig,
- ZIEL_TYP_JOINT: Das Ziel ist in Achskoordinaten angegeben,
- ZIEL_TYP_ADAX: Es ist nur ein Bewegungsziel der Zusatzachsen angegeben,
- ZIEL_TYP_REAL: Wird bei Zirkularbewegungen als Drehwinkel um ein Drehzentrum verwendet.

Zusätzlich zur Zielvorgabe für den Roboter können auch Zusatzachsen angegeben werden, die dann unabhängig (siehe Kap. 6.3) oder synchron zum Roboter interpoliert werden.

[4]Sie entspricht daher einer homogenen Transformation.

```
typedef struct movtarget{
    int     endtyp;     /* ZIEL_TYP_FRAME,ZIEL_TYP_MAT,ZIEL_TYP_VECTOR
                           ZIEL_TYP_JOINT,ZIEL_TYP_ADAX,ZIEL_TYP_REAL    */
    JOINT   jt;         /* Ziel-Achsstellung                             */
    FRAME   end;        /* Roboter Zielposition in Weltkoord.            */
    ROBOT_KIN_CONFIG kinconf;  /* Roboter-Konfiguration                  */
    /* ggf. Ziel der Zusatzachsen */
    BOOL    with_adaxes;/* TRUE, wenn mit Zusatzachsenbewegung           */
    AXIS    adx;        /* Zielachsstellung der Zusatzachsen             */

    /*zusätzl. Kreisbahn-Spezifikationen */
    int     intertyp;   /* Typ der Zwischenposition                      */
    FRAME   inter;      /* Roboter Zwischenposition bei Zirkularbahnen   */
    JOINT   inter_jt;   /* Achskoordinaten des Zwischenpunktes           */
    ROBOT_KIN_CONFIG circ_inter_kinconf;  /* Roboter-Konfiguration       */
    int     CircleType; /* Interpolationsart bei der Zirkularbahn        */
    double  CircAngle;  /* Winkel der Zirkularbahn, wenn gegeben         */
}MOV_TARGET;
```

Abbildung 6.4: Definition des Bewegungsziels

Zur Definition von Kreisbewegungen sind weitere Größen erforderlich, da sich ein Kreis nicht durch zwei Punkte im Raum definieren lässt. Auf welche Art die zusätzlichen Werte zur Bestimmung der zu fahrenden Kreisbahn herangezogen werden sollen, wird durch die Variable `CircleType` vorgegeben, wobei folgende Möglichkeiten implementiert werden:

1. Der Kreis wird durch die Ausgangsstellung, die angegebene Zwischenstellung (`inter` bzw. `inter_jt`) und die Zielstellung gelegt.

2. Der Kreis wird durch den Mittelpunkt und den Startpunkt definiert, wobei die Ebene, in welcher der Kreis liegt, durch die Einheitsvektoren e_x und e_y der Orientierungsmatrix des Mittelpunktes definiert wird.

3. Der Kreis wird durch den Mittelpunkt, den Startpunkt und den Zielpunkt definiert.

Hiermit ist die Lage des Kreises im Raum definiert, jedoch noch nicht unbedingt die Bogenlänge und der Umlaufsinn. Auch bei der Art der Orientierungsinterpolation haben sich verschiedene Varianten durchgesetzt:

- Die Orientierungsinterpolation findet wie bei der Linearinterpolation in den Koordinaten des Bezugssystems statt (z.B. Weltkoordinaten).

- Die Orientierungsführung findet in einem sich auf dem Radius des Kreisbogens mitdrehenden Koordinatensystem statt, womit das Programmieren von Bahnen auf runden Körpern erheblich vereinfacht werden kann.

Wie bei Linear- und PTP-Bewegungen kann die Position `inter` auch nur teilweise gültig sein, so dass bei der Spezifikation eines Kreisbogens durch Permutationen insgesamt über einhundert

verschiedene Möglichkeiten entstehen. Diese Spezifikationsvielfalt wird zwar nie von einer einzigen zu simulierenden Robotersteuerung ausgeschöpft, die Gesamtheit aller in der Simulation zu unterstützenden Roboter macht allerdings diesen Aufwand erforderlich. Auf die Algorithmen der notwendigen Berechnungen und der Kreisinterpolation wird in dieser Arbeit aus Platzgründen nicht näher eingegangen.

```
// Bewegungsart-Spezifikation
typedef struct mov_type{
    int     movtype;      /* Bewegungstyp: MOV_LINEAR,MOV_PTP,MOV_CIRCULAR*/
    BOOL    timeforced;   /* TRUE, wenn Bewegungszeit vorgegeben          */
    REAL    total_movtime;/* Gesamtdauer der Bewegung                     */
    BOOL    relative;     /* TRUE, wenn relativ zur letzten Position      */
    int     device_mode;  /* NO_DEVICE_MODE, TCP_KEEP, TCP_KEEP_REL,
                             TCP_DEVICE_REL, TCP_EXTERN                   */
    int     device_no;    /* Nummer des betroffenen Devices               */
}MOV_TYPE;
```

Abbildung 6.5: Definition der Bewegungsart

In der Datenstruktur zur Vorgabe des Bewegungsziels werden die Datentypen und ihre Bedeutung definiert. In der Struktur zur Angabe der Bewegungsart wird definiert, auf welche Weise und in welchen Bezugskoordinaten interpoliert werden soll (Abbildung 6.5). Hierbei wird mittels der Integer-Variablen `movtype` und vordefinierter Zahlenwerte vorgegeben, auf welche Weise sich der Roboter bewegen soll (Linear / Zirkular / PTP / Parabel). Die folgenden Variablen dieser Datenstruktur erklären sich durch die Kommentare von selbst oder sind in diesem Kontext von minderer Bedeutung.

Abbildung 6.6 stellt die Datenstruktur der Bewegungsparameter dar.

Mit ihnen kann das Geschwindigkeitsprofil der Bewegung beeinflusst werden. Variablen mit ”`path`” und ”`Ori`” betreffen ausschließlich Bahnbewegungen. Variablennamen mit ”`Joint`” ausschließlich PTP-Achsbewegungen. `AAdax` und `VAdax` betreffen die PTP-Bewegungen von ggf. vorhandenen Zusatzachsen. `veloverride` und `accoverride` sind Werte zur Herabsetzung der o.a. Grenzwerte. Mittels `Dragtype` kann das Roboterprogramm vorgeben, welches Kriterium erfüllt sein muss, um mit einem Verschleif-Vorgang zu beginnen.

6.1.2 Warteschlange gültiger Bewegungssätze

Leistungsfähige Bewegungssteuerungen verfügen über die Möglichkeit zwischen einzelnen Bewegungen zu verschleifen. Zur Berechnung des Übergangs sind die Kenntnis der Bahn, die verlassen werden soll, und die Kenntnis der Bahn, auf die verschliffen werden soll, erforderlich. Bei Wechsel des Bezugskoordinatensystems muss die Zielposition der letzten Bewegung in das neue Bezugskoordinatensystem transformiert werden, damit innerhalb der Interpolation die Startposition nicht inkonsistenter Weise auf das falsche Koordinatensystem bezogen wird. Hieraus kann man ableiten, dass die Bewegungssteuerung mindestens Zugriff auf den letzten, den aktuellen und den ggf. schon bereitstehenden folgenden Bewegungssatz haben muss. Eine hierfür sehr passende Struktur stellt eine Warteschlange aus *bereiten* Bewegungssätzen, der *aktuell* ausgeführten und der *letzten* Bewegung dar.

```
typedef struct mov_parameters{
    // programmierte Grenzwerte:
    REAL    pathvel;    /* Roboter-Bahngeschwindigkeit          */
    REAL    pathacc;    /* Roboter-Bahnbeschleunigung           */
    REAL    veloverride;/* Override ]0.0 .. 1.0]                */
    REAL    accoverride;/* Override ]0.0 .. 1.0]                */
    REAL    OriSpeed;   /* Winkelgeschwindigkeit beim Bahnfahren */
    REAL    OriAcc;     /* Winkelbeschleunigung  beim Bahnfahren */
    JOINT   AJoint,     /* Maximale Achsbeschleunigung          */
            VJoint;     /* Maximale Achsgeschwindigkeit         */
    AXIS    AAdax,      /* Maximale Zusatzachsbeschleunigung    */
            VAdax;      /* Maximale Zusatzachsgeschwindigkeit   */
    JOINT   AstopJoint; /* Maximale Achsverzögerung             */
    AXIS    AstopAdax;  /* Maximale Zusatzachsverzögerungung    */

    /* Verschleifkriterien: */
    enum    DRAGTYP{ NO_DRAGTYP,    /* Kein Verschleifen erwünscht                          */
                     DIST_DRAG,     /* ab einer bestimmten Nähe in mm zum Zielpunkt         */
                     ORI_DRAG,      /* ab einer bestimmten Winkelnähe zum Zielpunkt         */
                     VEL_DRAG,      /* ab Unterschreitung einer bestimmten Geschwindigkeit */
                     PERCENT_DRAG,  /* ab einem Prozentsatzes des bisherigen Weges          */
                     CONDITION_DRAG,/* wenn Condition TRUE wird                             */
                     START_TIME_DRAG,/* ab Bewegungszeit                                    */
                     END_TIME_DRAG, /* ab Zeit vor Zielpunkt                                */
                     INVALID_DRAGTYP /* Ab hier nicht definiert                             */
                    }Dragtype;
    int     ConditionNo;            /* Nummer der Bedingung bei Typ CONDITION_DRAG          */
    REAL    DragStartValue;         /* Wert je nach Typ, ab dem verschliffen werden soll    */
}MOV_PARAMETERS;
```

Abbildung 6.6: Definition der Bewegungsparameter

Ein Bewegungssatz wird von den übergeordneten Systemkomponenten ausgefüllt, als gültig markiert und in eine Pipeline von gültigen Sätzen eingereiht. Wird zu diesem Zeitpunkt keine Interpolation ausgeführt, wird die Bewegung bereits im nächsten Takt auf die Position ”*wird ausgeführt*” verschoben und bei erfolgreicher Validierung gestartet. Bewegt sich der Roboter hingegen zum Zeitpunkt des ”Gültig-Werdens” bzw. befinden sich noch weitere gültige Bewegungssätze vorher in der Warteschlange, dann bleibt sie in ihrem Zustand, bis alle vorherigen Bewegungsanweisungen beendet wurden. Bei Ausführung eines Roboterprogramms während einer Roboterbewegung kann der Programminterpreter im Programmvorlauf einige Bewegungsanweisungen ”einsammeln” und so die Warteschlange an gültigen Bewegungssätzen auffüllen. Abbildung 6.7 stellt diesen Vorgang schematisch dar. Die Bewegungssteuerung hat aus o.a. Gründen Zugriff auf alle validierten Bewegungssätze der Warteschlange.

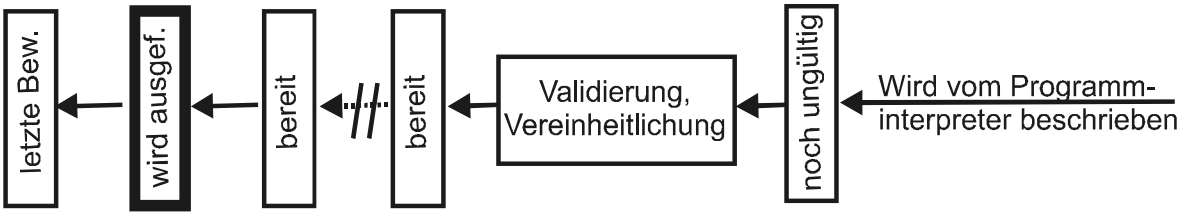


Abbildung 6.7: Pipeline der Bewegungssätze

Dieses Verfahren bietet mehrere Vorteile:

1. Alle Bewegungsobjekte besitzen dieselbe Struktur.

2. Als Datentyp der Elemente der Pipeline kann uneingeschränkt die im vorangegangenen Kapitel beschriebene Struktur verwendet werden, da sie alle erforderlichen Parameter enthält.

3. Der kompakte Datentyp der Bewegungssätze kann auch via Netzwerk übertragen werden, so dass eine einfache räumliche Trennung zwischen Bewegungssteuerung und Programminterpreter möglich ist.

4. Die erforderliche Trennung bzw. ”Asynchronisierung” von Programminterpreter und Interpolationstakt der Bewegungssteuerung wird dadurch erreicht, dass der Interpreter den Bewegungssatz sukzessive mit gültigen Werten füllt und abschließend gültig setzt. Der Bewegungssatz muss nicht mehr innerhalb einer einzigen IRDATA-Anweisung vollständig definiert werden und der Interpreter kann parallel zur Bewegungssteuerung weiter interpretieren.

5. Durch eine Vorgabe des maximal zulässigen und der Überwachung des aktuellen Füllungsgrades der Warteschlange kann der Programmvorlauf des Programminterpreters gesteuert werden.

6.2 Kartesisches Verschleifen linearer Bewegungen mittels Parabelbahn

Eine wichtige Eigenschaft moderner Robotersteuerungen ist die Möglichkeit zwischen einzelnen Bewegungen zu verschleifen, ohne dass der Roboter bei den Bewegungszielen anhalten muss. Besondere Bedeutung kommt dieser Eigenschaft zu, wenn kartesische, kontinuierliche Bewegungen ausgeführt werden müssen, die nicht durch einzelne gerade Strecken im Raum beschrieben werden können. Solche Bewegungen sind erforderlich, wenn z.B. Freiformflächen bearbeitet werden sollen. In [Rok99] werden Verfahren zur automatischen Trajektoriengenerierung beschrieben, die das Vorhandensein kartesischer Verschleiftechniken voraussetzen. Hierbei werden Bewegungssequenzen aus geometrischen Modelldaten zur Oberflächenbearbeitung generiert.

6.2.1 Achsbasierte Interpolation während des Übergangs

Eine universelle Möglichkeit der Interpolation während des Verschleif-Vorganges stellt eine achsbasierte Interpolation dar, bei der die Achsstellung und Geschwindigkeit jeder einzelnen Achse vom Verschleifstartpunkt zum Verschleifendpunkt überführt wird. Die Start- und Endachsstellungen des Überganganges werden dabei durch Lösung des IKP an den Übergangsstellen auf den kartesischen Bahnen berechnet. Verfahren, die für diese Art des Übergangs in Frage kommen, wurden bereits in [Kei92] als Möglichkeit zum Durchfahren von Singularitäten entwickelt und erläutert. Sie werden wird in dieser Steuerung eingesetzt, um zwischen PTP und

kartesischen Bewegungen zu verschleifen. Wie bei allen PTP Bewegungen, besteht der Nachteil dieser Art von Interpolation darin, dass die resultierende Bahn des TCP nicht kartesisch im Raum vorgegeben ist, sondern auf Achsebene. Da der Roboteranwender nicht intuitiv mit dem DKP des Roboters vertraut ist, kann diese Bewegungsart zu unerwarteten Abweichungen von der gewünschten Bahn führen. Genau diese Abweichungen sind es auch, die den Einsatz für die oben angeführte Aufgabe verhindern, da etwa bei einer Oberflächenbearbeitung die Lage des Werkzeugs stets eingeprägt sein muss und nicht von der vorgegebenen Soll-Bahn abweichen darf.

6.2.2 Kartesisches Verschleifen

Beim kartesischen Verschleifen sollte der TCP des Roboters auf einer für den Anwender leicht zu erkennenden und nachvollziehbaren (i.d.R. kartesisch beschriebenen Kurve) fahren. Beim Übergang zwischen zwei linearen Bahnen bietet sich hier eine Parabel als Verfahrkurve an, die zwischen die beiden Bahnen gelegt wird.

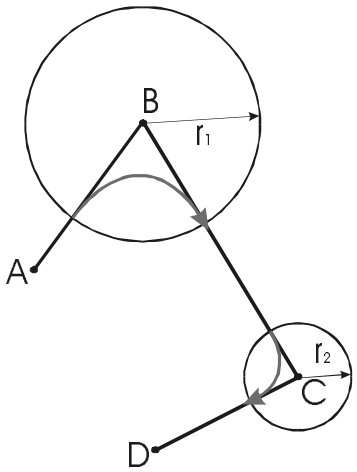


Abbildung 6.8: Verschleifen mit Parabeln

Abbildung 6.8 stellt einen Bewegungsablauf dar, bei dem drei lineare Bewegungen mittels Parabelsegmenten miteinander verschliffen werden. Ist das Verschleifkriterium erfüllt, das angibt, wann mit dem Verschleifen begonnen werden soll, dann muss die Parabelbahn geplant werden. Dazu muss die anschließende lineare Bahn bereits bekannt sein. Ist das noch nicht der Fall, etwa weil der Interpreter noch nicht auf den entsprechenden Bewegungssatz gestoßen ist, dann kann noch nicht mit dem Verschleifen begonnen werden und die alte Bahn wird ggf. bis zu ihrem Zielpunkt unverändert durchfahren. Ist die anschließende Bahn bekannt, aber zu kurz, um ab dem Punkt, in dem das Verschleifkriterium erfüllt ist, zu verschleifen, dann kann ebenfalls noch nicht mit der Parabelbahn begonnen werden, und es muss die alte lineare Bahn solange fortgesetzt werden, bis der Aufsetzpunkt[5] auf der anschließenden Bahn liegt. Ein weiterer Sonderfall liegt vor, wenn die anschließende Bahn zu kurz ist und ihrerseits verschliffen werden soll. Abbildung 6.9 stellt einen solchen Fall dar. Die gepunktete Parabel stellt dabei die nicht ausführbare

[5] Der Aufsetzpunkt ist der Punkt, in dem die Parabel die lineare Bahn berührt, auch ”Tangentenpunkt”.

Parabel dar, denn die Strecke BC ist kürzer als der Radius r_1, bei dessen Unterschreitung das Verschleifkriterium erfüllt war.

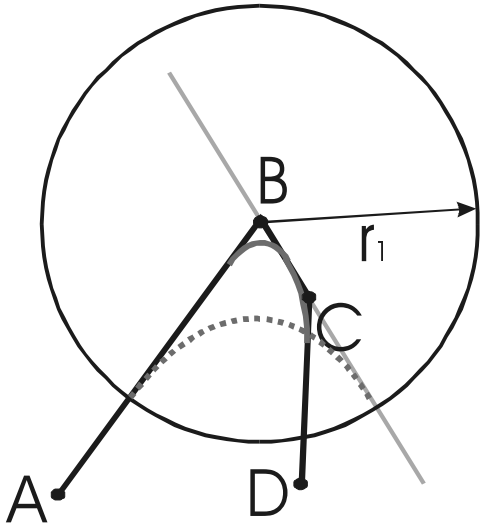


Abbildung 6.9: Verschleifen mit kurzer Anschlussbahn

Im Fall zu kurzer Anschlussbahnen, kann auf verschiedene Weisen verfahren werden:

1. Der Verschleifstart der anschließenden Bahn wird erst begonnen, wenn die erste Parabel vollständig beendet wurde.
   Dies kann jedoch dazu führen, dass die Parabel erst genau im Zielpunkt endet und daher von hier aus nicht mehr verschliffen werden kann.

2. Es wird gar nicht mehr in Richtung bzw. tangential zur anschließenden linearen Bahn gefahren, sondern die erste Parabel wird vorzeitig verlassen und die zweite Parabel setzt am Ende der ersten Parabel auf.
   Hierbei hat die resultierende Bahn nur noch wenig Ähnlichkeit mit der nicht verschliffenen Bahn. Bei den Verhältnissen, die in der Abbildung 6.9 dargestellt sind, würde kein Punkt der resultierenden Roboterbahn auf der Strecke BC liegen.

3. Die erste Parabel wird erst begonnen, wenn auf der anschließenden linearen Bahn noch genügend Platz für eine Parabel verbleibt. Dieser Fall ist in der Abbildung 6.9 eingezeichnet und stellt die verständlichste Form des Verschleifens unter diesen Umständen dar. Es wird in jedem Falle verschliffen, so dass der Roboter nicht anzuhalten braucht, und die resultierende Bahn liegt mindestens für kurze Zeit auf der linearen Bahn. Daher genügt dieses Verfahren auch den oben angeführten Anforderungen zur automatischen Bahngenerierung aus Körper-Geometriedaten.

Auf die Steuerungsmechanismen für das favorisierte Verfahren, bestehend aus der Steuerung des Interpreter-Vorlaufs, der Steuerung der Bahninterpolation und der Umschaltung zwischen den Interpolationsarten und Bahnabschnitten, wird hier nicht näher eingegangen, vielmehr wird genauer erläutert, welche Vorgänge erforderlich sind, um die Parabel abzufahren und stoß- bzw. ruckfreie Übergänge zu gewährleisten.

6.2.3 Berechnung der Parabelbahn

Die Berechnungen der Parabelbahn werden ”just in time” durchgeführt, d.h. genau in dem Interpolationstakt, in dem die Parabel begonnen werden soll. Diese Berechnungen können nicht eher durchgeführt werden, da das Verschleifkriterium auch volatile Daten, wie etwa Sensordaten, berücksichtigen kann. Dabei müssen sämtliche Bahndaten der aktuellen und der anschließenden Bahn bekannt sein[6]. Da die Bewegungssteuerung zeit-diskret abläuft, wird auch die Parabelbahn naturgemäß zu Interpolationstakten diskret abgetastet. Abbildung 6.10 verdeutlicht durch die Punkte auf den Bahnen diese Diskretisierung. Wird das Verschleifkriterium wahr, dann soll

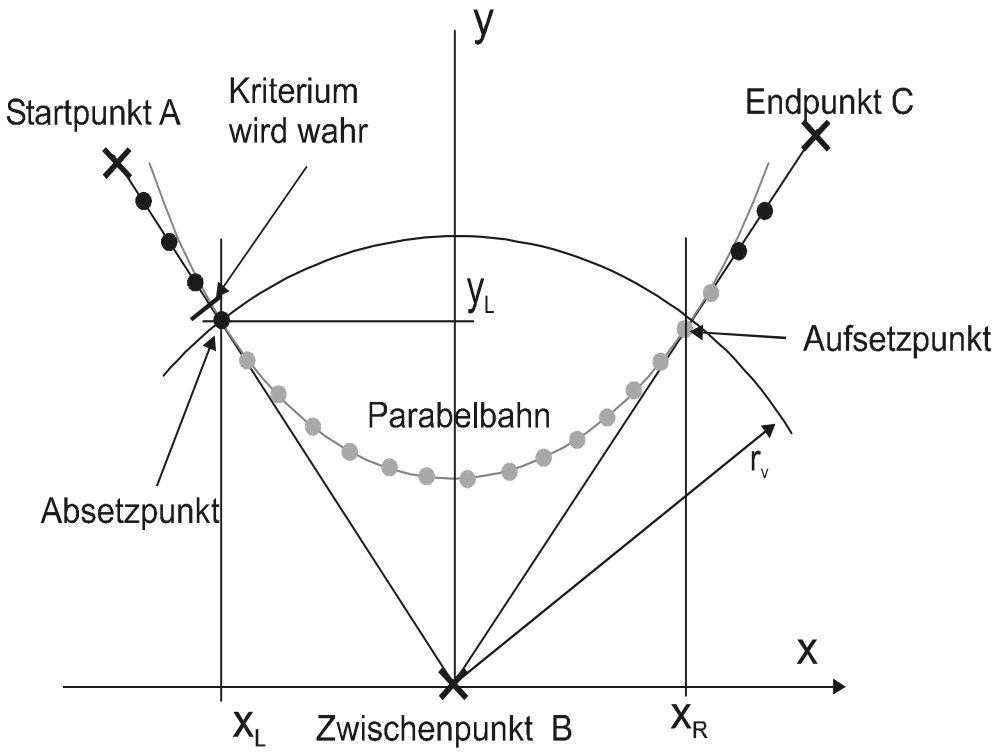


Abbildung 6.10: Diskretisierte Bahn

der anschließende Punkt der Bahn bereits auf der Parabel liegen. Daher haben im anschließenden Interpolationstakt alle im Folgenden aufgeführten Planungsaufgaben zu erfolgen:

Berechnen des Parabel-Frames, der homogenen Transformation vom Bezugskoordinatensystem zu einem Koordinatensystem, in dessen X-Y-Ebene die Parabel und in dessen Ursprung der Zwischenpunkt liegt. Damit ergibt sich die Parabelgleichung zu:

$$y(x) = cx^2 + y_{offset} \tag{6.1}$$

mit c = Parabelstreckfaktor und y_{offset} der Ordinate, bei der die Parabel die y-Achse schneidet.

[6] Ist das nicht der Fall, wird solange auf der alten Bahn weitergefahren, bis diese zu Ende ist, oder alles über die anschließende Bahn bekannt ist.

Gegeben sind hierbei die Koordinaten des Punktes, in dem die Linearbahn verlassen werden soll (x_L, y_L), also des Tangentenpunktes zwischen Parabel und alter Bahn. Ebenfalls bekannt ist die Steigung der Parabel im Tangentenpunkt. Nach einfachen Umformungen ergibt sich:

$$c = \frac{y_L}{2x_L^2} \tag{6.2}$$

$$y_{offset} = \frac{y_L}{2} \tag{6.3}$$

Durch die Symmetrie der Parabel ist automatisch auch der Berührungspunkt auf der anschließenden Linearbahn gegeben, weil sie die Tangente an die Parabel darstellt.

Es folgt die Berechnung des Aufsetzpunktes auf die anschließende Bahn. Da das Geschwindigkeitsprofil der anschließenden Bahn durch ihre Bahnparameter vorgegeben ist, stehen auch die Diskretisierungspunkte bzw. die Abtastpunkte zu jedem einzelnen Interpolationstakt der Anschlussbahn fest. Diese Interpolationspunkte liegen im Allgemeinen nicht auf dem Tangentenpunkt, so dass er selbst nicht als Interpolationspunkt in Frage kommt. Da auch das Geschwindigkeitsprofil auf den linearen Abschnitten durch das Verschleifen nicht verändert werden soll, muss die Position zum letzten Interpolationstakt auf der Parabel bestimmt werden, so dass ein ruck- und stoßfreier Übergang gewährleistet ist. Bei konstanter Bahngeschwindigkeit müssen alle Interpolationspunkte den gleichen räumlichen Abstand voneinander haben. Variiert dieser Abstand innerhalb des Übergangstaktes zu stark, so springt auch die Geschwindigkeit im Raum entsprechend, was zu parasitären Beschleunigungssprüngen auf Gelenkebene führt. Um das zu vermeiden, wird zunächst die Position nach dem Tangentenpunkt berechnet, was nicht immer analytisch erfolgen kann, denn das Geschwindigkeitsprofil besteht während der Start- und der Stoppphase aus Polynomen vierten Grades. Liegt der Aufsetzpunkt in einer dieser Rampen, so muss mittels einer iterativen Näherung der Zeitpunkt innerhalb des Geschwindigkeitsprofils berechnet werden, bei dem die Parabel die Strecke berührt. Die Eingangsgrößen sind hierbei die Parameter der Polynome und die Strecke seit dem Zwischenpunkt B.

Als Algorithmus wird hier das Newton-Verfahren [BS99] zur schnellen Bestimmung von Nullstellen bei Polynomen verwendet. Ist der genaue Zeitpunkt des Tangentenpunktes bekannt, dann wird dieser auf exakte Interpolationstakte auf- oder abgerundet.

Abbildung 6.11 stellt die beiden in Frage kommenden ersten Interpolationspunkte L_t und L_{t+1}der neuen Bahn dar. In der Grafik wurden die geometrischen Verhältnisse und die Abstände der Interpolationspunkte bewusst übertrieben gezeichnet, um die Problematik deutlicher herauszustellen. Welcher der beiden Punkte die bessere Lösung darstellt, wird anhand ihrer Abstände von der Parabelbahn bewertet. Die korrespondierenden Punkte der Parabelbahn werden gefunden, indem die Senkrechten auf die Anschlussbahn durch die Punkte L_i gelegt werden und ihre Schnittpunkte mit der Parabel bestimmt werden. Daraus ergeben sich leicht die Abstände d_i. Ist der Abstand $\|P_t L_t\|$ geringer, dann wird L_t als erster Interpolationspunkt auf der Anschlussbahn ausgewählt, sonst L_{t+1}. Der letzte Punkt auf der Parabelbahn ist damit ebenfalls gegeben. Bei der in Abbildung dargestellten Situation fällt die Entscheidung auf L_t. Der Wert X_R stellt das Ende des für diesen Übergang relevanten Intervalls $[X_L, X_R]$ dar. Zu beachten ist hierbei, dass $|X_L| \neq |X_R|$ ist, die beiden Schenkel der Parabel also unterschiedlich lang sind.

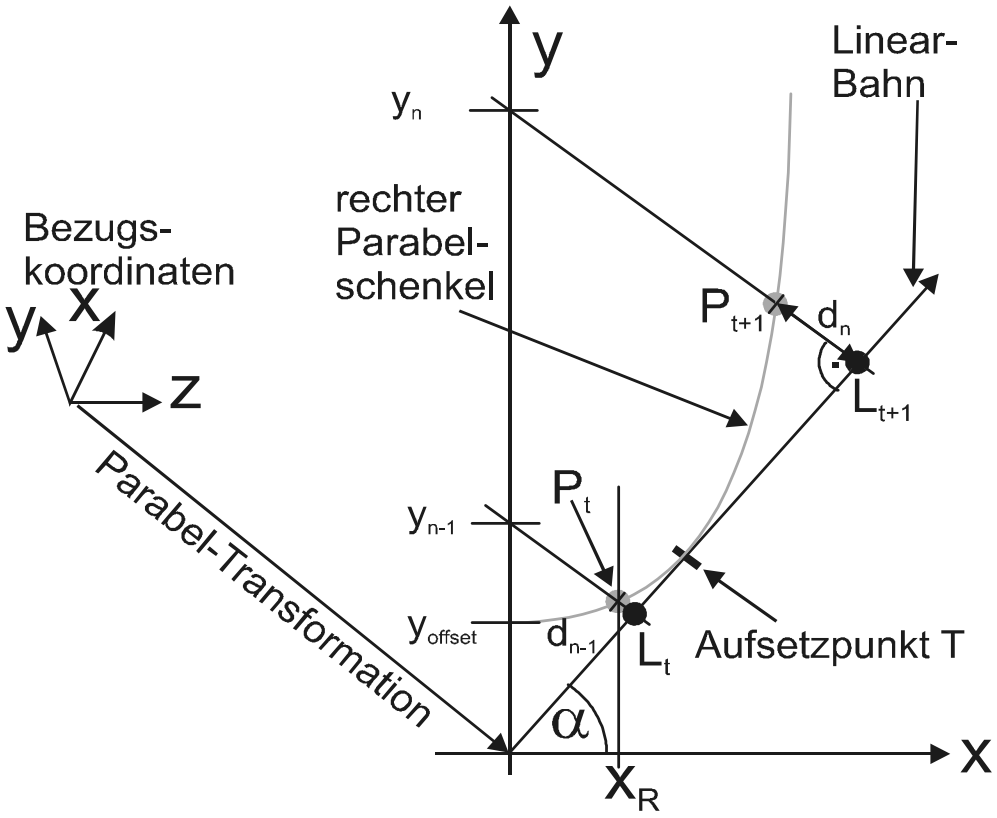


Abbildung 6.11: Aufsetzen auf die Anschlussbahn

Zur Vereinfachung der Schreibweisen bezeichnet im Folgenden der Index ”L” oder ”left” Variablen, die sich auf die erste Bahn bzw. den linken Schenkel der Parabel beziehen und entsprechend ”R” oder ”right” auf die rechte Seite.

Als nächstes kann das Geschwindigkeitsprofil über die Zeit der Parabel berechnet werden, mit dem die Geschwindigkeit bei Verlassen der alten Bahn an die im Aufsetzpunkt angeglichen wird, wobei die Geschwindigkeit im Scheitel der Parabel möglichst den Mittelwert v_m der Start- und Endgeschwindigkeit betragen sollte. Als Parameter der Geschwindigkeitsfunktion wird, wie bei der Linearinterpolation, die Strecke bzw. Bogenlänge verwendet. Die Gesamtbogenlänge der Strecke auf der Parabel ergibt sich aus der Summe der Strecken vom Scheitel zu X_L und zu X_R. Zur Berechnung der Bogenlänge B_p lässt sich eine analytische Lösung finden, die sich aus einer Formel in [BS99] herleiten lässt.

Mit den Hilfsvariablen

$$h(c,x) = 2cx$$

und

$$k(c,x) = \sqrt{1 + h(c,x)^2}$$

ergibt sich

$$B_p(c,x) = \frac{1}{4c} * (h(c,x)k(c,x) + log(k(c,x) + h(c,x))) \tag{6.4}$$

Bei der Konzeption eines geeigneten Geschwindigkeitsprofiles sollte die kartesische Beschleunigung im Raum berücksichtigt werden, die vom Krümmungsradius und der Tangentialgeschwindigkeit abhängt und daher im Scheitel ihren höchsten Wert erreicht. Da eine maximale Raumbeschleunigung $a_{\max}$ zu den Bahnparametern gehört (zur Definition des Geschwindigkeitsprofiles), kann die Geschwindigkeit im Scheitel so gewählt werden, dass die Querbeschleunigung diesen maximal zulässigen Wert nicht überschreitet. Die Querbeschleunigung einer Parabel ergibt sich im Scheitel zu:

$$a_s = 2v_s c \tag{6.5}$$

mit a_s = Querbeschleunigung im Scheitel, v_s = Tangentialgeschwindigkeit, c =Parabel-Streckfaktor. Maximal sollte die Scheitelgeschwindigkeit daher den Wert

$$v_{s\,\max} = \frac{a_{\max}}{2c} \tag{6.6}$$

nicht überschreiten (mit $a_{\max}$ der maximal zulässigen Bahnbeschleunigung). Ist der Mittelwert v_m also größer als $v_{s\,\max}$, so ist $v_{s\,\max}$ als Scheitelgeschwindigkeit zu wählen.

Zur Realisierung des Gesamt-Geschwindigkeitsprofiles bietet sich eine zusammengesetzte Funktion an, die aus zwei Polynomen besteht, jeweils für den linken und rechten Schenkel der Parabel.

Zur Auswahl des Grades der Polynome muss die Anzahl der gegebenen Größen bekannt sein:

1. Startgeschwindigkeit v_0,
2. Endgeschwindigkeit v_1,
3. Start-Bogenlänge = Null,
4. End-Bogenlänge B,
5. Tangential- bzw. Bahnbeschleunigung im Scheitel soll Null sein, so dass lediglich die Querbeschleunigung auf die Antriebe wirkt.

Mittels dieser Vorgaben lässt sich ein Polynom dritten Grades bestimmen:

Die Koeffizienten eines Weg-Polynoms

$$S(t) = p_o t^3 + p_1 t^2 + p_2 t + p_3 \tag{6.7}$$

ergeben sich aus den o.a. Randbedingungen 1.-5. wie folgt:

$$p_0 = \frac{v_0 + v_1}{t_{end}^2} - \frac{2B}{t_{end}^3} \tag{6.8}$$

$$p_1 = -\frac{2v_0 + v_1}{t_{end}} + \frac{3B}{t_{end}^2} \tag{6.9}$$

$$p_2 = v_0 \tag{6.10}$$

$$p_3 = 0 \tag{6.11}$$

Aus der Vorgabe, dass der Wendepunkt des Polynoms im Scheitel liegen soll, ergibt sich für die Fahrzeit auf dem linken Schenkel:

$$a(t_{end}) = \frac{dS(t)}{dt^2} = 0 = 6p_0 t_{end} + 2p_1 \Longrightarrow \tag{6.12}$$

$$t_{end} = -p_1/3p_0 \tag{6.13}$$

Einsetzen von 6.8 und 6.9 in 6.12 ergibt nach Auflösung:

$$t_{endleft} = \frac{3B_{left}}{v_{start} + 2v_{scheitel}} \tag{6.14}$$

Mit diesem Wert können nun die Koeffizienten des Weg-Polynoms $S_{left}(t)$ berechnet werden[7].

[7] Bei der Implementierung werden die Zeiten auf volle Interpolationstakte aufgerundet, so dass der Scheitelpunkt exakt auf einem Interpolationstakt liegt. Damit können dann an dieser Stelle z.B. Bahnschaltfunktionen ausgeführt werden.

Bei dem Polynom des rechten Schenkels soll die Beschleunigung zu Beginn der Fahrt (im Scheitel) Null sein:

$$a(0) = \frac{dS(t)}{dt^2} = 0 = 6p_0(t = 0) + 2p_1 \Longrightarrow \tag{6.15}$$

$$2p_1 = 0$$

Für $p_1 = 0$ und anschließender Auflösung von 6.9 nach t_{end} ergibt die Verfahrzeit des rechten Schenkels:

$$t_{endright} = \frac{6B_{right}}{(4v_{scheitel} + 2v_{end})}. \tag{6.16}$$

Hieraus können nun die Koeffizienten des Weg-Polynoms $S_{right}(t)$ berechnet werden.

Mittels der soeben ermittelten Polynome kann nun die Soll-Bogenlänge zu einem beliebigen Zeitpunkt während der Interpolation auf der Parabel ermittelt werden. Aus dieser Bogenlänge lässt sich dann die Position auf der Parabel und damit letztlich die Position im Raum errechnen. Die Interpolation auf der Parabelbahn wird in Kapitel 6.2.5 ausführlicher erläutert.

6.2.4 Orientierungsführung während des Verschleifens

Bisher wurde lediglich beschrieben, mit welchem Verfahren eine Position auf der Parabelbahn gefunden werden kann, die zu einem bestimmten Interpolationstakt bzw. zu einer bestimmten Zeit[8] eingenommen werden soll. Die Orientierung des TCPs wurde bisher nicht berücksichtigt und soll in diesem Kapitel berechnet werden. Als Eingangsgrößen liegen die beiden linearen Bahnen mit ihren eigenen Geschwindigkeitsprofilen vor, mit denen an jeder Stelle der Bahn die entsprechende Position und Orientierung und deren Ableitungen berechnet werden können. Gleiches soll auch dieses Verfahren für die Parabelbahn liefern, so dass sowohl die Orientierungsmatrix als auch die Winkelgeschwindigkeit an jeder Stelle der Parabel berechnet werden kann.

Die Orientierungsdifferenz einer *linearen* Bahninterpolation lässt sich durch einen Drehvektor und einen Drehwinkel darstellen. Während der Interpolation wird die Startorientierung in die Zielorientierung überführt, indem der TCP um den Drehvektor um den Differenzdrehwinkel rotiert wird. Bei diesem implementierten Verfahren wird zur Berechnung des Drehvektors und des Drehwinkels zunächst die Differenztransformation zwischen Start- und Zielpunkt der Bahn berechnet: $T_D = T_S^{-1} T_E$ mit T_D als homogene Differenztransformation, mit der sich der Startpunkt

[8] Sämtliche implementierten Interpolationsverfahren der Steuerung sind so ausgelegt, dass stets die Zeit der Parameter eines Interpolationsvorganges ist. D.h. es kann nach Berechnung der Bewegungsparameter zu jedem beliebigen Zeitpunkt die Position, Orientierung und Geschwindigkeit bestimmt werden. Das ist u.a. deshalb erforderlich, weil die Interpolationsfrequenz einstellbar ist.

T_S in den Zielpunkt T_E überführen lässt. Während der linearen Interpolation wird das Start-Frame mit jedem Interpolationstakt in Richtung des End-Frames hinzu interpoliert, wobei das Bezugskoordinatensystem der Rotation das Zielkoordinatensystem ist. Daher wird die zu überbrückende Orientierungsänderung relativ zum Zielkoordinatensystem ausgedrückt, indem zu der Inversen von T_D die korrespondierende Drehvektor-Drehwinkeldarstellung (R_{vec} und R_{ang}) in Weltkoordinaten berechnet wird und danach der Drehvektor in Koordinaten des Ziel-Frames ausgedrückt wird:

$$R_{relvec} = T_E^{-1} R_{vec}.$$

Für diesen Drehwinkelverlauf muss, wie für die Bahngeschwindigkeit, ein stetiges Geschwindigkeitsprofil verwendet werden, da eine maximal zulässige Orientierungsbeschleunigung nicht überschritten werden darf. Für den Orientierungsverlauf auf der Parabel bietet sich auf den ersten Blick ebenfalls dieses Verfahren an, bei dem um einen Drehvektor von der Start- in die Endorientierung im Aufsetzpunkt gedreht wird und mit stetigem Geschwindigkeitsprofil sanft verschliffen wird. Doch wird hierbei nicht berücksichtigt, dass die Drehvektoren der Linearbahnen i.A. verschieden sind. An den Übergängen würde das zu starken Stößen (Geschwindigkeitssprüngen) führen, die innerhalb des Sicherheitskreises eine Fehlerbehandlung auslösen würden.

Im Folgenden werden weitere Konzepte erläutert:

1. Während der Fahrt auf der Parabel könnten die Handachsen im PTP-Betrieb von ihren Stellungen und Geschwindigkeiten im Absetzpunkt zu denen im Aufsetzpunkt interpoliert werden, und die Grundachsen werden so gestellt, dass Position und Geschwindigkeit im Raum exakt eingenommen werden.
   - Hierbei kann es passieren, dass wie bei allen PTP-Bewegungen die Handachsen nicht im vom Anwender erwarteten Bereich bleiben. Ferner erfordert diese Art der Interpolation die Erweiterung sämtlicher Kinematikmodule, da hierbei nicht das IKP des gesamten Roboters gelöst werden muss, sondern nur das der Grundachsen. Es erfordert gewissermaßen eine neue Art von Rücktransformation. Daher wird diese Art zunächst außer Betracht gelassen.

2. Der Orientierungsverlauf der ersten Bahn kann bis zum Scheitel der Parabel vollständig zu Ende geführt werden, so dass die Orientierungsgeschwindigkeit im Scheitel der Parabel Null wird. Anschließend wird die Orientierung der anschließenden Bahn so ausgeführt, als würde gar nicht verschliffen.
   - Bei diesem Verfahren wird die Orientierungsführung nicht verschliffen, im Scheitel wird die Orientierung eingenommen, die auch im Zwischenpunkt eingenommen würde, wenn nicht verschliffen würde. Das entspricht nicht dem Prinzip des Verschleifens, bei dem die Position und Orientierung des Zwischenpunktes nicht erreicht werden soll. Daher wird auch dieses Verfahren, das mit Abstand am einfachsten umzusetzen ist, nicht realisiert.

3. Der Orientierungsverlauf wird auf der Parabel geometrisch durch Überblendung der Orientierungsführungen der beiden Linearbahnen stetig von der Start-Orientierung und -Geschwindigkeit kartesisch zu den Werten im Aufsetzpunkt interpoliert.
   - Die Schwierigkeit dieses idealen Verfahrens besteht darin, einen kartesischen Übergang zu finden, mit dem diese ”Überblendung” realisierbar ist.

Der Grundgedanke des implementierten Verfahrens sieht die Überblendung zwischen den beiden Orientierungsführungen der Linearbahnen vor, bei welcher der Einfluss der Orientierungsführung der ersten Bahn im Verlauf der Parabel sanft aus- und der Einfluss der Orientierungsführung der anschließenden Bahn sanft eingeblendet wird, so dass im Aufsetzpunkt der Einfluss der ersten Bahn exakt Null ist und der Einfluss der zweiten Bahn Eins und damit an dieser Stelle genau die Werte im Aufsetzpunkt erreicht werden. Abbildung 6.12 stellt die gesuchten Verläufe der Einflüsse der einzelnen Orientierungen während der Parabelbahn dar.

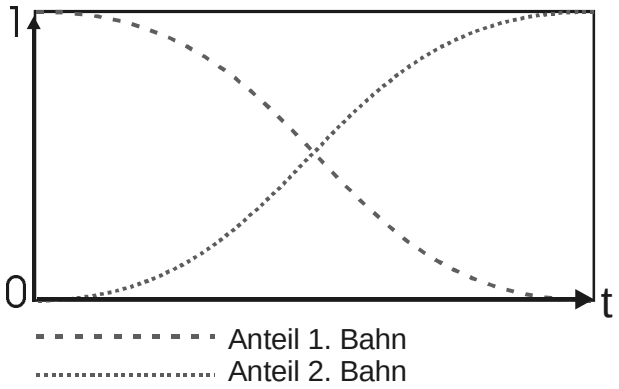


Abbildung 6.12: Einfluss der einzelnen Orientierungen auf die Gesamtbahn

Bei der Realisierung dieses Überblendens reicht allerdings eine normierte Funktion -wie in der Grafik- nicht aus, vielmehr müssen die realen Verhältnisse auf den Bahnen berücksichtigt werden, mit denen die Anfangs- und Endwerte der Polynome gegeben sind:

Bahnabschnitt	erste Bahn	Anschlussbahn
Startwinkel	φ_{0L}	$\varphi_{0R} = 0!$
Startwinkelgeschw.	ϖ_{0L}	$\varpi_{0R} = 0!$
Endwinkel	$\varphi_{1L} = 0!$	φ_{1R}
Endwinkelgeschw.	$\varpi_{1L} = 0!$	ϖ_{1R}

(6.17)

φ_{0L} wird während der Parabel von einem Startwert auf Null heruntergefahren, so dass im Aufsetzpunkt die Drehung der ersten Bahn vollständig beendet ist. Der Startwert ist genau der verbleibende Drehwinkel, um den der TCP auf der linken Bahn noch gedreht werden muss. φ_{0R} soll bei Null beginnen und bis zum Aufsetzpunkt genau den Wert annehmen, welcher der Orientierung im Aufsetzpunkt entspricht, wenn nicht verschliffen würde. Bei der Interpolation werden diese beiden Orientierungen kombiniert und ergeben die resultierende Orientierung des Werkzeugs.

Zur vollständigen Bestimmung der Polynome fehlt noch eine Größe: Die zur Verfügung stehende Verfahrzeit ist die Summe der Fahrzeiten auf den beiden Schenkeln der Parabeln aus den Formeln 6.14 und 6.16:

$$t_{ori} = t_{endleft} + t_{endright.} \tag{6.18}$$

Zu den Polynomen der Winkelinterpolation

$$\Phi_{left/right}(t) = p_o t^3 + p_1 t^2 + p_2 t + p_3 \tag{6.19}$$

ergeben sich allgemein die Koeffizienten aus den Randbedingungen:

$$p_0 = \frac{\varpi_1 + \varpi_0}{t_{ori}^2} + 2\frac{\varphi_0 - \varphi_1}{t_{ori}^3}, \tag{6.20}$$

$$p_1 = 3\frac{\varphi_1 - \varphi_0}{t_{ori}^2} - \frac{2\varpi_0 + \varpi_1}{t_{ori}}, \tag{6.21}$$

$$p_2 = \varpi_0, \tag{6.22}$$

$$p_3 = \varphi_0. \tag{6.23}$$

Damit sind sämtliche Funktionen zur Parabelinterpolation parametrisch definiert, und es kann der erste Punkt auf der Parabel interpoliert werden.

6.2.5 Parabelinterpolation

Im vorangegangenen Kapitel wurden die zur Interpolation erforderlichen Parameter bestimmt. Mittels dieser kann nun die Parabelbahn abgefahren werden. Der Parameter sämtlicher Berechnungen ist hierbei die Zeit t, mit der ein beliebiger Interpolationspunkt seit dem Absetzpunkt auf der Parabelbahn berechnet werden soll.

Dabei wird wie folgt verfahren:

1. Zur Berechnung des **Positionsvektors** werden, ausgehend von dem zu verwendenden Geschwindigkeitsprofil (6.7) des aktuellen Schenkels, die Bogenlänge und die Geschwindigkeit an dieser Stelle berechnet.

2. Aus der gegebenen Bogenlänge ist der korrespondierende Punkt auf der Parabelbahn zu finden. Hierzu muss der Parameter x der Parabelgleichung (6.1) aus der Bogenlänge berechnet werden.
   Allgemein ergibt sich die Bogenlänge einer Raum-Kurve aus dem Integral infinitesimal

kleiner Polygonabschnitte auf der Kurve

$$S = \int \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2} \tag{6.24}$$

Da die Parabel in der Ebene liegt, entfällt die Komponente d_z des Integrals und es kann für die Parabelgleichung (6.1) analytisch gelöst werden. Es gilt nach Umformung von (6.4):

$$S_{parabel}(x) = \frac{c}{2}(2cx\sqrt{1+(2cx)^2} + log(\sqrt{1+(2cx)^2} + 2cx), \tag{6.25}$$

mit dem Parabelstreckfaktor c und x, der Abszisse der Parabel.
Zur Berechnung des Punktes auf der Parabelbahn ist die Umkehrfunktion von (6.25) erforderlich. Diese wird für jeden Interpolationspunkt iterativ bestimmt. Als Verfahren wird hier wieder das Newton-Verfahren verwendet, das sehr schnell konvergiert, da die Funktion streng monoton steigend ist und keine Extremwerte aufweist. Durch die Verwendung des x-Wertes aus dem vorangegangenen Interpolationstakt, der im Allgemeinen schon sehr nahe an dem gesuchten Wert liegt, ist der gesuchte Parabelparameter x meist schon nach dem ersten Iterationsschritt in hoher Genauigkeit gefunden. Hierzu wird in letzter Ausführungsinstanz der Startwert solange korrigiert, bis die Bogenlänge an der Stelle x auf 1 nm übereinstimmt.
Mit

$$f(x_i) = S_{parabel}(x_i) - S(t) \tag{6.26}$$

wird solange mit

$$x_{i+1} := x_i + \frac{f(x)}{\sqrt{1+(2cx)^2}} \tag{6.27}$$

iteriert, bis $f(x_i)$ kleiner 10^{-9} ist.

3. Aus dem gefundenen Parabel-Parameter x lässt sich die Ordinate y im Parabel-Koordinatensystem durch einfaches Einsetzen in Formel (6.1) bestimmen:

$$y = cx^2 + y_{offs}.$$

4. Die Position in den Bezugskoordinaten ergibt sich aus der homogenen Transformation des Parabel-Koordinatensystems mit den Koordinaten des Punktes im Parabel-Frame:

$$P(x) = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} \tag{6.28}$$

$$TCPpos(x) = PT * P(x) \tag{6.29}$$

mit PT der Parabeltransformation aus Abbildung 6.11 und $TCPpos$ der Position im Bezugskoordinatensystem.

5. Die Geschwindigkeit in Welt- bzw. in Bezugskoordinaten ist das Produkt des normierten Tangentenvektors der Parabel in dem Punkt P(x) und der Ableitung von S(t). Der Geschwindigkeitsvektor im Parabel-Koordinatensystem ergibt sich damit zu

$$V(x) = \left\| \begin{pmatrix} 1 \\ 2cx \\ 0 \end{pmatrix} \right\| * \frac{dS(t)}{dt}. \tag{6.30}$$

Der resultierende Geschwindigkeitsvektor des TCP lautet dann

$$TCPvel(x) = PT * V(x). \tag{6.31}$$

6. Zur Berechnung der **Orientierung** werden Drehwinkel $\Phi_{left}(t)$ und $\Phi_{right}(t)$ aus ihren Polynomen 6.19 sowie deren Ableitungen als Winkelgeschwindigkeiten berechnet.

7. Danach werden die Rotationen sukzessive hintereinander ausgeführt, wobei zunächst die Rotationsmatrizen der Einzel-Drehungen erzeugt werden. Die erste Matrix Rot_L wird gebildet, indem zunächst eine Einheitsmatrix um die Rotationsachse der ersten Linearbewegung um den Winkel $\Phi_{left}(t)$ gedreht wird. Das Bezugskoordinatensystem der Drehung ist hierbei unverändert gegenüber der regulären, nicht verschliffenen Bahn, nämlich der Zwischenpunkt B. Hieraus resultiert die Orientierungsmatrix M_L, die lediglich den Orientierungsanteil der linken Bahn enthält:

$$M_L = B * Rot_L \tag{6.32}$$

Für die Bestimmung der Achsgeschwindigkeiten an dieser Stelle ist auch die Berechnung der Winkelgeschwindigkeit erforderlich. Der resultierende Winkelgeschwindigkeitsvektor $\Omega_L(t)$, der kollinear zum Drehvektor liegt und die Länge der Ableitung von $\Phi_{left}(t)$ aufweist, muss dazu allerdings in das Basiskoordinatensystem des Roboters transformiert werden, indem das Koordinatensystem des Zwischenpunktes B mit $\Omega_L(t)$ multipliziert wird. $\Omega_{Lbasis}(t) = B * \Omega_L(t)$ stellt damit den Einfluss der Winkelgeschwindigkeit der linken Linearbahn an der Gesamtorientierungsgeschwindigkeit auf der Parabelbahn in Ba-

siskoordinaten dar.

8. Die zweite Drehung, die den Einfluss der rechten Linearbahn darstellt, muss der Konsistenz der Bezugskoordinaten wegen ebenfalls um Zwischenpunkt B ausgeführt werden. (Regulär bezieht sich die Orientierungsführung auf das Zielkoordinatensystem, wie bereits am Anfang von 6.2.4 beschrieben). Dazu wird der Rotationsvektor $R_{relvecR}$ zunächst ins Basiskoordinatensystem transformiert: $R_{vecR} = C * R_{relvecR}$, mit C, dem Zielpunkt der zweiten Linearbahn. Damit die Kombination der einzelnen Rotationen sinnvolle Werte ergibt, muss nun der resultierende Rotationsvektor in die Koordinaten von M_L transformiert werden, denn verkettete homogene Transformationen beziehen sich stets auf das Vorgängerkoordinatensystem:

$$R_{relLvecR} = M_L^{-1} * R_{vecR} \tag{6.33}$$

9. Nun kann die 2. Rotation durch Umwandlung der Drehvektor-Drehwinkeldarstellung - ($R_{relLvecR}$, R_{angR}) in die Orientierungsmatrix M_R erfolgen. Durch Multiplikation erhält man die Gesamtorientierung an dieser Stelle der Parabel:

$$TCP_{ori} = M_L * M_R \tag{6.34}$$

10. Der Anteil der Winkelgeschwindigkeit der rechten Drehung ist

$$\Omega_R(t) = \|R_{vecR}\| \frac{d\Phi_{right}(t)}{dt}. \tag{6.35}$$

also korrespondierend zu $\Omega_L(t)$ aus Punkt 7, die Norm des Rotationsvektors in Basiskoordinaten, multipliziert mit der Ableitung des aktuellen Drehwinkels.

11. Die Gesamtorientierungsgeschwindigkeit ergibt sich aus der Summe der einzelnen Winkelgeschwindigkeiten:

$$\Omega_{gesamt}(t) = \Omega_L(t) + \Omega_R(t) \tag{6.36}$$

Damit sind sämtliche kartesischen Werte zu einem Interpolationspunkt auf der Parabel bestimmt:

Position des TCPs: $TCPpos$, Formel 6.29,

Geschwindigkeitsvektor des TCPs: $TCPvel$, Formel 6.31,

Orientierungsmatrix des TCP: TCP_{ori}, Formel 6.34 und

Winkelgeschwindigkeit des TCP: Ω_{gesamt}, Formel 6.36.

Mit diesen Werten können die Funktionen zur Lösung des IKP und des erweiterten IKP zur Bestimmung der Achsgeschwindigkeiten aufgerufen werden.

6.3 Dynamische Definition und Ansteuerung synchroner und asynchroner Achsgruppen

In Roboterarbeitszellen werden häufig zusätzlich zum Roboter weitere zu regelnde, kinematische Geräte benötigt, wie z.B. Werkstückpositionierer, Fließbänder, Linearachsen, Zuführeinheiten, Stanzen und Dreh- oder X-Y-Z-Tische.

Um die Automatisierung solcher Zellen zu vereinfachen, stellen verschiedene industrielle Robotersteuerungen sehr unterschiedliche Konzepte zur Ansteuerung von zusätzlichen Achsen zur Verfügung:

- Zusatzachsen als einzelne oder mehrere Achsen, die synchron zum Roboter bewegt werden können, indem ihre Interpolation von der Roboter-Bahninterpolation mit übernommen wird. Dabei beginnen und enden die Bewegungen des Roboters und die der anderen Achsen gleichzeitig. Van der Valk beschreibt hier Verfahren, mit denen der Roboter damit auch relativ zu Geräten fahren kann, die ihrerseits bewegt werden [dV95].
- Zusatzachsen als einzelne Achsen, die jeweils für sich, mit eigenen Geschwindigkeitsprofilen bewegt werden und steuerungstechnisch nicht mit anderen Komponenten interagieren[9].
- Zusatzachsen als Achsgruppen, die ein konkretes Gerät, etwa einen Lagerarm bewegen. Diese Achsen bewegen sich asynchron zum Roboter, aber synchron untereinander.

Die Unterschiedlichkeit der Konzepte besteht also darin, dass die Zusatzachsen zum Einen synchronisiert mit den Roboter angesteuert werden, so dass die Bewegungen des Roboters und die anderer Achsen stets gleichzeitig beginnen und enden (synchron), oder, zum Anderen völlig losgelöst und unabhängig von den Roboterbewegungen interpoliert werden (asynchron) [ABB98]. Bei manchen Herstellern werden die Achsgruppen vom Aufsteller der Anlage statisch, für den Anwender nicht änderbar vorgegeben werden [GF95]. Bei wieder anderen Systemen können sind Zusatzachsen stets Einzelachssteuerungen [Ele97].

Innerhalb eines leistungsfähigen Simulationssystems sollten idealer Weise alle diese Konzepte verfügbar sein. Eine Lösung stellt hierbei die getrennte Unterstützung der verschiedenen Systeme dar, wobei für jede Struktur eigene Module implementiert werden und diese bei Bedarf aktiviert werden. Das Ideal stellt jedoch auch hier eine Verallgemeinerung aller bisher relevanten Konzepte dar, mit der die verschiedenen Varianten simuliert werden können.

6.3.1 Definition asynchroner Achsgruppen

Zu diesem Zweck wurde ein umfassendes und neuartiges Konzept entwickelt, mit dem sämtliche Achsen des Systems jeder Zeit mit anderen Achsen zu Gruppen kombiniert und absolut unabhängig angesteuert und verfahren werden können. Zudem ist die Anzahl der möglichen asynchronen Achsgruppen nicht durch die Anzahl der in der Zelle vorkommenden Achsen begrenzt, da Achsen auch in mehreren Achsgruppen gleichzeitig vertreten sein dürfen. Gruppen mit gemeinsamen Achsen können dann allerdings nicht gleichzeitig bewegt werden.

[9] Dieser Fall tritt dann ein, wenn die Robotersteuerung um Einzelachssteuerungen erweitert und deren ”on board-Interpolation” verwendet wird.

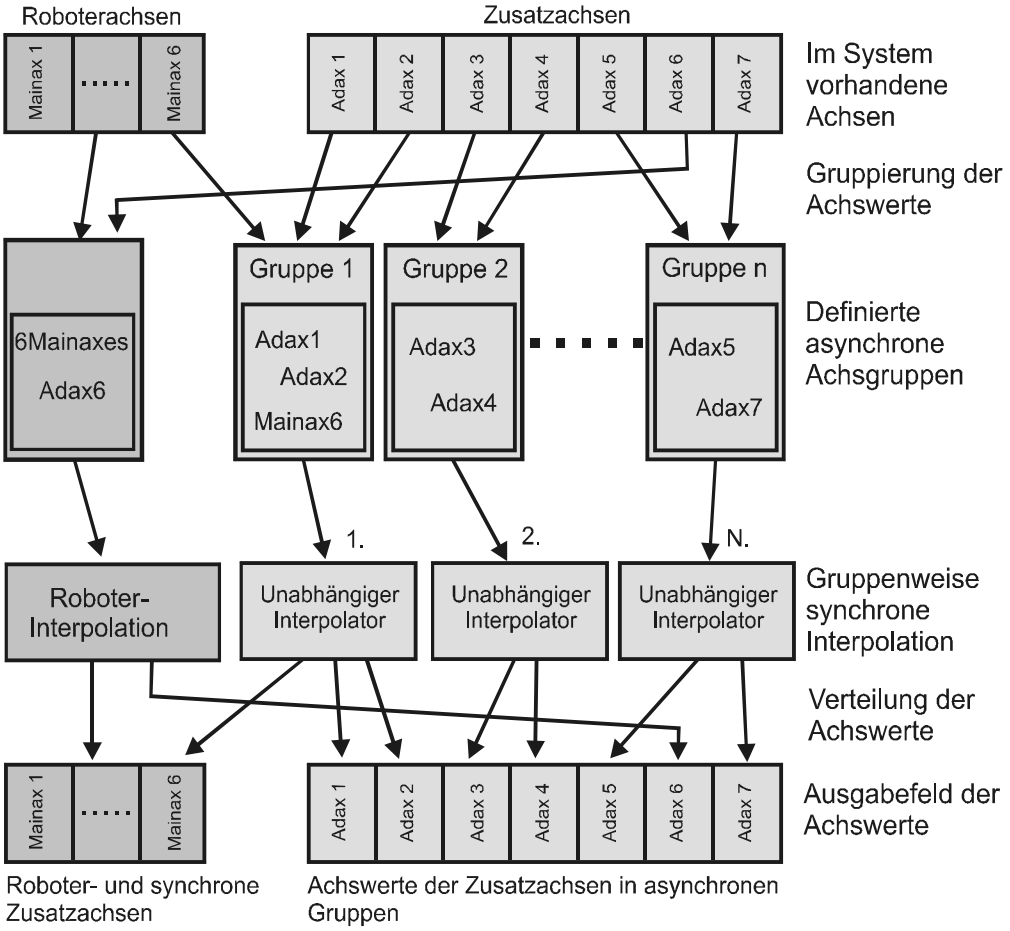


Abbildung 6.13: Beispielhafte Organisation bei drei asynchronen Achsgruppen und einem Roboter

Abbildung 6.13 zeigt beispielhaft das Prinzip der Organisation der Achsgruppen und die Integration in das Steuerungs- bzw. Simulationssystem. Dabei werden Achsen, die untereinander im Synchro-PTP-Verfahren[10], jedoch asynchron zu anderen Bewegungsinterpolationen bewegt werden sollen, zu Gruppen kombiniert (Gruppe 1 bis n). In der Grafik werden Zusatzachsen mit ”Adax” und Hauptachsen, die den Roboter bilden, als ”Mainax” bezeichnet.

Liegt für eine Gruppe ein Bewegungsbefehl vor, dann werden ihre Achsen von dem unabhängigen Interpolator parallel zur Interpolation anderer Gruppen und zum Roboter interpoliert. Achsen, die nicht explizit als zu einer Gruppe gehörend definiert wurden, gehören automatisch der Robotergruppe an und werden bei Bedarf synchron zu Roboterbewegungen interpoliert, d.h. die Bewegungen dieser Achsen beginnen und enden mit ggf. gleichzeitig ausgeführten Roboterbewegungen.

Zu beachten ist die Möglichkeit, dass auch Roboterachsen in Gruppen unabhängiger Achsen aufgenommen werden können. Im Allgemeinen ist das nur für die letzte Achse des Roboters sinnvoll, die (wenn überhaupt) meistens als einzige Achse ohne Endanschlag konstruiert ist. Hiermit

[10] Auf die Verfahren der PTP-Interpolation soll an dieser Stelle jedoch nicht weiter eingegangen werden, da diese bereits an anderen Stellen ausführlich besprochen wurden, z.B. in [dV95].

lässt sich dann z.B. ein gegriffenes, rotationssymmetrisches Teil um seine Rotationsachse drehen und dabei etwa an einer Poliermaschine entlang führen. Der Wert, den die Lösung des IKP für diese Achse des Roboters liefert, hat in diesem Falle eine mindere Priorität gegenüber dem des unabhängigen Interpolators und wird überschrieben.

Finden gleichzeitige Bewegungen mehrerer Gruppen statt, so werden zunächst sämtliche Achswerte berechnet und die Ergebnis-Werte im Anschluss in das Ausgabefeld der Achsen verteilt. Dort stehen sie zur weiteren Verarbeitung zur Verfügung (siehe Abbildung 6.1). In Abbildung 6.13 sind drei asynchrone Achsgruppen definiert, wobei die erste Gruppe auch die letzte Roboterachse enthält. Vor der Interpolation eines Taktes werden die einzelnen Achsen jeder Gruppe zu Objekten zusammengefasst, die eine eigenständige Interplationseinheit bilden. Nach einem Interpolationstakt werden die Werte der Interpolationseinheiten wieder in das Ausgabefeld sortiert.

Um diese Funktionalität in den zu unterstützenden Roboterhochsprachen verfügbar zu machen, wurden spezielle IRDATA-Anweisungen implementiert, die sich einfach in andere Hochsprachen übernehmen lassen (z.B. IRL):

- NEW_ASYNC_GROUP:
  Definiert eine neue, textuell adressierbare Achsgruppe, die noch keine Achse enthält.
- ADD_ADAX_TO_ASYNC_GROUP:
  Fügt eine Zusatzachse in die Gruppe ein.
- ADD_MAINAX_TO_ASYNC_GROUP:
  Fügt eine Roboterachse in die Gruppe ein.
- DELETE_ASYNC_GROUP:
  Löscht eine Gruppendefinition.
- SELECT_ASYNC_GROUP:
  Selektiert eine Gruppe. Die auf diese Anweisung folgenden Bewegungsanweisungen betreffen nur diese Gruppe. Überflüssige Informationen in dem Datentyp des Bewegungsziels werden ignoriert.

Beispiel zur unabhängigen Ansteuerung einer zu definierenden Achsgruppe:

1. Anlegen einer neuen Gruppe Namens ”Table”:
   `NEW_ASYNC_GROUP(”Table”).`
2. Anwahl der Gruppe, damit folgende Anweisungen dieser Gruppe zugerechnet werden können:
   `SELECT_ASYNC_GROUP(”Table”).`
3. Hinzufügen der beiden Achsen eines Drehschwenktisches, die unabhängig von anderen Achsen verfahren werden sollen:
   `ADD_ADAX_TO_ASYNC_GROUP(5,”Dreh”);`
   `ADD_ADAX_TO_ASYNC_GROUP(6,”Schwenk”);`
   Die Zusatzachsen 5 und 6 werden in die Gruppe übernommen und mit Namen versehen.

4. Bewegungskommando kann abgesetzt werden:
   `MOVE PTP a_joint(0,0,0,0,30,90,0,0);`
   Nur Achsen 5 und 6 sind unter den insgesamt 8 Zusatzachsen im System relevant. Mit diesem Kommando wird die Bewegung dieser Achsen angestoßen.

5. Deselektion der Gruppe:
   `SELECT_ASYNC_GROUP("");`
   Die Gruppe "Table" bewegt sich bereits und folgende Bewegungsanweisungen betreffen nun wieder die Robotergruppe.

6. Bewegungskommando für den Roboter:
   `MOVE LIN POSITION(300,0,500,180,0,-180), ADAX( 120,0,300,400, 0, 0, 0, 0);`
   Fährt den Roboter mit eigener Interpolation, synchron zur Stellung der angegebenen Zusatzachsen ans Ziel, jedoch unabhängig von der asynchronen Gruppe "Table".

6.3.2 Synchronisation asynchroner Achsgruppen mit den verbleibenden Achsen

Um Bewegungen definierter Achsgruppen untereinander und mit dem Programmfluss synchronisieren zu können, werden weitere IRDATA-Anweisungen entworfen und implementiert:

- `WAIT_FOR_ALL_GROUPS()`
  Diese Anweisung wartet ggf. auf den Abschluss sämtlicher unabhängiger Bewegungen.

- `WAIT_FOR_GROUP(name)`
  Diese Anweisung wartet auf den Abschluss der Bewegung der angegebenen asynchronen Gruppe, z.B. `WAIT_FOR_GROUP("Table")`.

Kann nicht auf das Ende von unabhängigen Bewegungen gewartet werden, dann ist es mit den folgenden Anweisungen möglich, diese auch zum vorzeitigen Bewegungsabbruch zu bringen. Die Anweisungen sind erst dann beendet, wenn die betreffenden Achsen zum Stillstand gekommen sind:

- `STOP_ALL_GROUPS()`
  Bricht ggf. die Bewegung sämtlicher fahrender asynchroner Gruppen ab.

- `STOP_GROUP(name)`
  Stoppt die Bewegung der angegebenen Achsgruppe.

6.4 Zusammenfassung

In diesem Kapitel wurde auf Besonderheiten und die Lösung von Problemen bei der Bewegungssteuerung eingegangen. Sie ermöglicht es, verschiedenste Robotersteuerungen industrieller Roboterhersteller zu simulieren. Daraus resultieren Schwierigkeiten, die im Kern auf abweichenden

Systemstrukturen und Konzepten der zu unterstützenden Zielsysteme beruhen. Ziel der Konzeption und Realisierung dieser Bewegungssteuerung war es, diese Differenzen zu elementarisieren und zu vereinheitlichen, um diese auf ein konsistentes, parametrierbares Modell abbilden zu können. Mit Hilfe dieses Modells konnte ein neuartiges Systemdesign realisiert werden, das auch der Echtzeit-Variante der R/VRC zur Ansteuerung realer Roboter und anderer Geräte zu Gute kommt. Auf die Spezifika der Echtzeitsteuerung wird in den folgenden Kapiteln eingegangen.

Kapitel 7

Zusätzliche Komponenten zur Ansteuerung realer Geräte

In den vorangegangenen Kapiteln wurden Konzepte beschrieben, die sowohl in der VRC als auch in der R/VRC verwendet werden. Lediglich im Kapitel 6.1 zur Struktur der Bewegungssteuerung wurde bereits auf einige Erfordernisse eingegangen, die zur Ansteuerung realer Geräte nötig sind, indem die Struktur mit vom restlichen System entkoppelter Bewegungssteuerung beschrieben wurde. In diesem Kapitel wird auf die weiteren Besonderheiten der Variante ”Robotersteuerung” mit Verbindung zu einem realen Roboter eingegangen.

7.1 Anforderungen an eine Gerätesteuerung

In dieser Arbeit werden mechanische Einheiten, die über Antriebsaktuatoren verfügen, als Geräte bezeichnet. Zu diesen Geräten gehören sowohl Roboter, als auch externe Maschinen wie Dreh-Schwenktische, Zuführeinheiten und Fließbänder. Die Ansteuerung solcher mechatronischer Systeme soll mit der R/VRC möglich sein, indem die vorliegenden Sollwerte von der Bewegungssteuerung an den Regler übergeben werden, der mittels Leistungselektronik die Drehmomente oder Drehzahlen der Antriebe einstellt. Die Aufgabe dieser Arbeit besteht unter anderem darin, die Sollwerte zu ermitteln und diese den Reglern *pünktlich* zur Verfügung zu stellen. Als entscheidende Anforderungen bei der Ankopplung realer kinematischer Geräte können folgende Punkte, die im reinen Simulationsbetrieb nicht von Bedeutung sind, zusammengestellt werden:

1. Pünktliche Bereitstellung von Achssollwerten für die unterlagerte Regelung.
2. Die Achssollwerte müssen stetig differenzierbar sein, um Rucke[1] und Stöße[2] zu vermeiden.

[1] Ein Ruck resultiert aus einem Sprung in der Beschleunigung, so dass auf die Gelenkkörper eine augenblickliche Kraftänderung einwirkt.

[2] Ein Stoß resultiert aus einem Sprung in der Geschwindigkeit, der durch Antriebssysteme nicht erreicht werden kann. Die hierzu erforderlichen Beschleunigungsverläufe sind nicht zu erreichen. Bei Kollisionen treten für die Mechanik schädliche Stöße auf (vgl. ”Schreibtisch-Spiel” mit aufgehängter kollidierender Kugelreihe).

3. Die Gerätesteuerung sollte Schnittstellen zur Verfügung stellen, mit denen der Anwender auf andere externe Einrichtungen Einfluss nehmen kann, wie z.B. SPS, Netzwerke, Werkzeuge und digitale/analoge I/O.

4. Die Bewegungssteuerung sollte über offene Schnittstellen verfügen, mit der von extern Einfluss auf die ausgeführte Bewegung genommen werden kann (z.B. für Sensorintegrationen).

5. Die Schnittstellen zur Hardware sollten einheitlich, übersichtlich und einfach verwendbar sein, so dass der Aufwand zur Ankopplung einer gewünschten Hardware möglichst gering ist.

6. Die Steuerung muss über ein parametrierbares Sicherheitssystem verfügen.

In den folgenden Kapiteln wird auf diese Punkte genauer eingegangen.

7.2 Echtzeitpriorisierte autonome Bewegungssteuerung

In Kapitel 6.1 zur Systemstruktur der Bewegungssteuerung wurde bereits die Grundlage für eine Echtzeitfähigkeit geschaffen: Die Bewegungssteuerung wird gegenüber sämtlichen anderen Komponenten der Steuerung und des Simulationssystems höher priorisiert, damit sie in der Lage ist, ggf. auch während des Neuzeichnens der Arbeitszelle auf dem Bildschirm, neue Achswerte zu berechnen[3]. Diese Achswerte werden von der Bewegungssteuerung innerhalb des Interpolationstaktes berechnet. Die geeigneteste Systemstruktur, die sich hier anbietet, besteht, wie bereits beschrieben, in der Auslagerung der Bewegungssteuerung in einen eigenen Thread höchster Priorität.

Um die erforderliche Echtzeit-Eigenschaft erreichen zu können, wurden bereits bei der Auswahl des Betriebssystems umfassende Untersuchungen durchgeführt, wobei sich herausstellte, dass Windows NT mit Hilfe aufwändig zu entwickelnder Kernel-Modus-Komponenten durchaus ausreichende Echtzeit-Eigenschaften aufweist. Aufbauend auf den im Betriebssystem vorhandenen Scheduling-Mechanismen, wird im Rahmen dieser Arbeit eine autonome Bewegungssteuerung entwickelt und implementiert, die in ihrer Zeitbasis, losgelöst von sämtlichen anderen Steuerungskomponenten, parallel arbeitet. Auf die funktionalen und strukturellen Eigenschaften der Bewegungssteuerung wurde bereits im Kapitel 6 eingegangen.

7.2.1 Synchronisation zwischen Bewegungssteuerung und den übergeordneten Komponenten

Synchronisation zwischen Prozessen ist immer dann erforderlich, wenn sie nicht mehr asynchron zueinander ausgeführt werden können oder sollen, weil etwa ein bidirektionaler Kommunikationsprozess stattfinden muss bzw. ein Prozess auf Ergebnisse eines anderen zu warten hat. Haben beteiligte Prozesse Echtzeitaufgaben zu übernehmen, so muss stets gewährleistet sein, dass

[3]Bei Ein-Prozessor-Systemen geschieht dies naturgemäß nicht gleichzeitig, sondern die Bewegungssteuerung unterbricht in diesem Fall die Darstellungsfunktion.

der geforderte Takt trotz des Wartens auf Ereignisse eingehalten wird. Aus diesem Grund darf der Bewegungs-”Thread” der Robotersteuerung, der einen festen Takt einhalten muss, nicht auf Ereignisse höherer Ebenen des Steuerungssystems warten, da diese nicht innerhalb des Echtzeit-Kontextes laufen. Abbildung 7.1 zeigt die erlaubte Signalisierungskette des Steuerungssystems, die sich streng an der Hierarchie der Steuerungsstruktur orientiert. Zu synchronisierende Komponenten warten auf Signalisierungen gleicher oder untergeordneter Hierarchien. Die Pfeile stellen erlaubte Signalisierungsvorgänge dar, mit dem sich die übergeordneten Komponenten synchronisieren dürfen. Signalisierungen zu den unteren Ebenen hin sind nicht erlaubt, damit diese nicht in Synchronisierungsaufrufen ”stecken bleiben”, die von überlagerten, nicht echtzeitfähigen Komponenten beeinflusst werden[4].

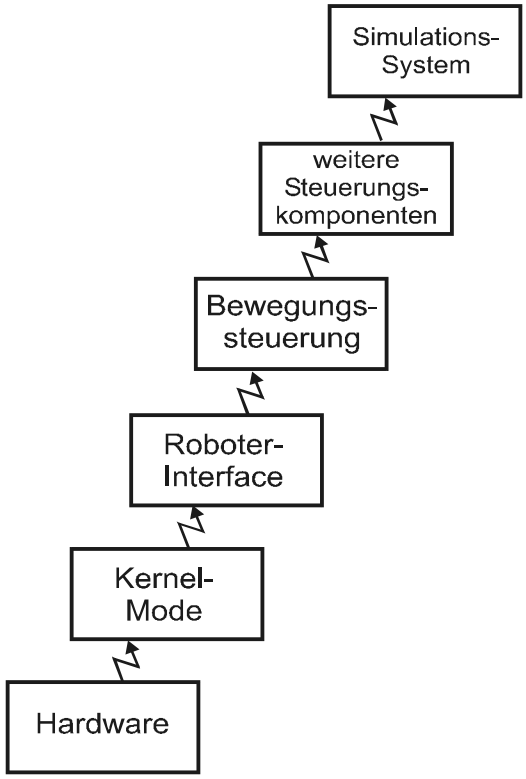


Abbildung 7.1: Signalisierungs- und Synchronisationskette

Der Fall, in dem sich der IRDATA-Interpreter oder das Simulationssystem mit der Bewegungssteuerung synchronisieren, ist daher realisierbar und ist auch erforderlich. Beim Unterbrechen von Roboterbewegungen z.B. braucht der Roboter wegen seiner endlichen Beschleunigungsfähigkeit noch einige Takte, bis er wirklich stillsteht. In einigen Fällen muss dieser Moment abgewartet werden, etwa, wenn im Roboterprogramm vorgegeben ist, dass erst bei Bewegungsende mit der Programminterpretation fortgefahren werden soll. Zur Steuerung von Ereignissen, die an Start-, End- oder Übergangspunkte gebunden sind, werden besondere Synchronisations-”Events” zur Verfügung gestellt, die innerhalb des IRDATA-Programms verwendet werden können. Mit Hilfe dieser Events kann z.B. ein digitaler Ausgang genau am Bewegungsende gesetzt werden, indem innerhalb eines IRDATA-Prozesses auf das korrespondierende Bewegungsende gewartet

[4]Bei Microsoft Windows nimmt die Latenzzeit in Richtung höherer Hierarchien ab.

wird. In Kombination mit dem Ereignisinterpreter, der in Kapitel 5.4 ausführlich erläutert wurde, lassen sich komplexe Bahnschaltfunktionen realisieren.

Die Ereignisse der Bewegungssteuerung, die in höheren Ebenen zur Verfügung stehen, werden hier tabellarisch aufgeführt:

1. Bewegungsbeginn,
2. Bewegungsabbruch,
3. Verschleifbeginn,
4. Verschleifen ist Zwischenpunkt am nächsten,
5. Verschleifende,
6. Bewegungsende,
7. Bahnmitte erreicht,
8. Roboter steht (wieder) still,
9. ein Fehler ist aufgetreten.

7.2.2 Erzeugung des Interpolationstaktes

Der Interpolationstakt der Bewegungssteuerung kann auf verschiedene Art und Weise erzeugt werden. Resultieren muss die Quelle des Taktes aber systembedingt in der Auslösung eines Hardware-Interrupts. Je nach angekoppelter Roboterhardware kann der Takt der Servoverstärker in den PC übertragen werden und dort mittels geeigneter Hardware den Interrupt auslösen. Im Allgemeinen stellt die Kommunikations-Schnittstelle diese Funktion zur Verfügung (z.B. RS232, RS485, Transputer-Link, Ethernet, Druckerport). Muss die Regelung extern synchronisiert werden, verfügt sie also nicht über einen eigenen Taktgenerator, so kann zur Generierung des Interpolationstaktes auch die PC-interne Echtzeituhr verwendet werden. Der Zeitgeberbaustein auf den PC-Hauptplatinen kann so programmiert werden, dass er einen Interrupt zyklisch, mit einstellbarer Frequenz auslöst. Dieser Interrupt muss systembedingt das Ereignis für die Bewegungssteuerung direkt oder indirekt (ggf. über weitere Zwischenstufen) setzen, so dass möglichst keine Verzögerungen auftreten und auch keine permanenten Abfrageschleifen notwendig sind. Der genaue Vorgang der Synchronisation mit diesem Interrupt-Ereignis kann auf zwei verschiedene Arten erfolgen:

1. Die Bewegungssteuerung ruft die Synchronisationsfunktion des Robotertreibers auf, innerhalb derer auf das Ereignis gewartet wird.
2. Es werden die vom Betriebssystem bereitgestellten Synchronisationsfunktionen und Semaphore verwendet. Die Bewegungssteuerung synchronisiert sich mit den Komponenten im Kernel Mode durch Systemereignis-Objekte.

Welche dieser Methoden zum Einsatz kommt, wird innerhalb des Robotertreibers definiert. Stellt dieser die Referenz auf das Systemereignisobjekt zur Verfügung, so wird mit der zweiten Methode synchronisiert, ansonsten durch Aufruf der Synchronisationsfunktion der Schnittstelle.

Auf die Realisierung zweier unterschiedlicher Roboterankopplungen wird im Kapitel 8.1 genauer eingegangen.

7.3 Einheitliche Servoschnittstellen der Bewegungssteuerung

Damit die in dieser Arbeit beschriebene R/VRC möglichst einfach an unterschiedliche Hardware-Plattformen angekoppelt werden kann, wurde eine Antriebsschnittstelle entwickelt, mit der die Ankopplung verschiedener Robotersysteme ohne Änderungen bzw. ohne Neukompilierungen der Steuerungsversion möglich ist. Erreicht wird dies durch die Konzeption einer verallgemeinerten Schnittstelle, die auf dem Austausch von Ist- und Soll-Zustandsdaten beruht. An diese Schnittstelle schließt sich der Robotertreiber an, der die erste Komponente unterhalb der Bewegungssteuerung darstellt. Da der Datenaustausch sehr schnell (im Interpolationstakt) erfolgt, wurde hier die Art der Funktionsschnittstelle[5] gewählt. Das heisst, die Bewegungssteuerung ruft Funktionen des Robotertreibers auf. Die Ein- und Ausgabeparameter der Funktionen stellen den Datenaustausch dar. Realisiert wird dieses mit Hilfe des Konzeptes der dynamisch ladbaren DLLs[6], das vom Betriebssystem zur Verfügung gestellt wird. Im Folgenden wird eine Zusammenstellung der Daten gegeben, die über diese Schnittstelle ausgetauscht werden:

1. **Konfigurationsdaten**. Sie werden beim Start der Schnittstelle ausgetauscht, können aber auch im Betrieb geändert werden. Das Besondere dieser Schnittstelle ist, dass die Robotersteuerung dem Robotertreiber zunächst die diejenigen Werte ”vorschlägt”, die innerhalb des Robotermodells spezifiziert wurden. Lassen diese Werte sich nicht mit den von der angeschlossenen Hardware vereinbaren, so kann der Robotertreiber diese Werte überschreiben. Zu diesen Daten gehören:
   Die Interpolationszeit, die Anzahl der Zusatz- und Roboterachsen, der Name des Roboters, der Betriebsmodus und die Referenz auf das Synchronisationsereignis, das den Interpolationstakt vorgibt. In dem Fall, dass die Werte vom Robotertreiber geändert wurden, wird entschieden, ob ein weiterer Betrieb dieses Systems sinnvoll ist oder ein Fehler ausgelöst wird und der Betrieb verweigert wird. Fehler werden z.B. ausgelöst, wenn die Anzahl der Achsen nicht übereinstimmt.

2. **Servo-Soll- und Istzustände**. Diese Werte sind vom Aufzählungsdatentyp und sind im Anschluss an diese Tabelle genauer erläutert.

3. **Achssollwerte.** Sie enthalten die gewünschte Achsstellung im gegenwärtigen Interpolationstakt. Rotationsachsen werden dabei in Bogenmaß und Translationsachsen in Millimeter angegeben. Die Achssollwerte werden von der Steuerung nur während einer Interpolation gesetzt. Andernfalls sind diese Werte ohnehin konstant.

[5] Bei der Funktionsschnittstelle werden die Daten mit den Parametern einer aufgerufenen Schnittstelle übertragen.

[6] DLL: Dynamic Link Library

4. **Achsistwerte** werden von der Bewegungssteuerung in jedem Takt abgefragt und werden u.a. der Visualisierung bereitgestellt.

5. **Digitale bzw. analoge Soll- und Istwerte** der Ein-/Ausgabekanäle. Mittels dieser Funktionen können Ein- und Ausgänge asynchron oder synchron zum Interpolationstakt gelesen bzw. geschrieben werden.

6. **Fehlermeldungen** bzw. standardisierte Fehlernummern. Über diese Daten meldet der Robotertreiber Fehlersituationen. Die meisten Fehlerfälle lassen sich einheitlich klassifizieren und auf einen Aufzählungsdatentyp abbilden. Für diese klassifizierten Fehlermeldungen existieren innerhalb der Bewegungssteuerung dedizierte Behandlungen. Zusätzlich kann der Robotertreiber auch Textmitteilungen übergeben, die von der Robotersteuerung bis zur Anwenderebene durchgeschaltet werden[7].

Die Servo-Zustände wurden so gewählt, dass ohne Beschränkung der Allgemeinheit auch hier möglichst sämtliche Zustände real existierender Hardware, wie z.B. Regler-/ Servoverstärkersysteme, einheitlich abgebildet werden können. Innerhalb der Zustandsmaschine, die bereits in Kapitel 6.1 erwähnt wurde, werden die Übergänge derart kontrolliert, dass stets die größte Systemsicherheit gegeben ist. Folgende Servozustände wurden dabei vordefiniert:

1. **SERVO_ERROR**:
   Die Hardware meldet einen Fehler oder Notaus. Zum Fortfahren ist die Ursache zu beheben.

2. **SERVO_OFF**:
   Die Antriebe sind abgeschaltet und nur vom Bediener wieder einschaltbar.

3. **SERVO_STOP**:
   Die Antriebe sind abgeschaltet, es liegt kein Fehler vor. Die Servos sind programmgesteuert einschaltbar.

4. **SERVO_REF**:
   Der Roboter befindet sich in der Initialisierungsphase oder führt gerade die Referenzierfahrt aus.

5. **SERVO_WAIT**:
   Der Roboter steht in der Regelung, ist aber (noch) nicht in Bereitschaft, um Sollwerte von der Steuerung anzunehmen. Dieser Zustand eingenommen, wenn ein Roboterprogramm bearbeitet wird, jedoch keine Bewegung ausgeführt wird.

6. **SERVO_ON**:
   Der Roboter ist aktiv und der Robotertreiber erwartet im Interpolationstakt Sollwerte von der Bewegungssteuerung. Dieser Zustand wird während der Bewegungen eingenommen.

Die erlaubten Übergänge zwischen den Servozuständen sind in Abbildung 7.2 dargestellt. Ein Übergang von einem Servozustand in einen anderen kann grundsätzlich durch vier verschiedene Eingangszustände der Zustandsmaschine ausgelöst werden:

[7] Auf die Aufzählung der über 80 Fehlernummern soll hier aus Platzgründen verzichtet werden.

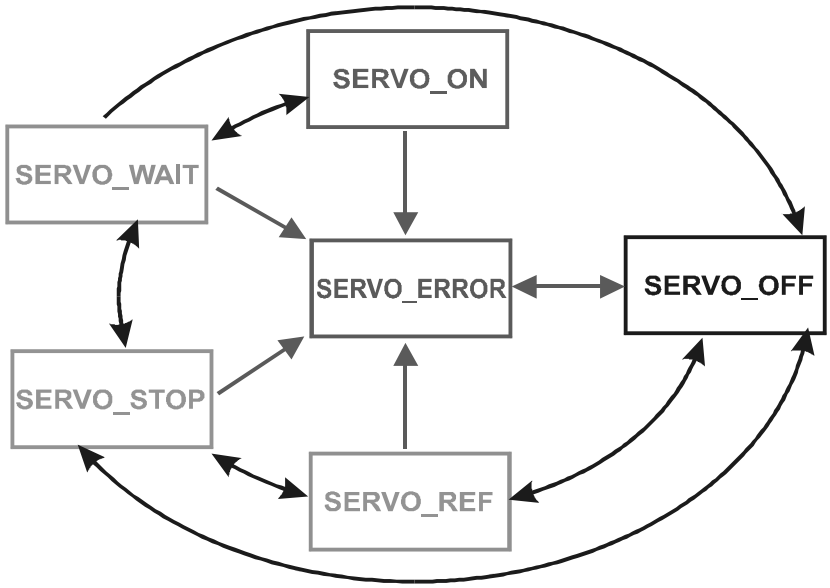


Abbildung 7.2: Erlaubte Übergänge der Servozustände der Steuerung

1. Zustand, den der **Anwender** einstellen möchte.
   Dieser Zustand wird vom Anwender über die auf dem Bildschirm eingeblendete Bedienkonsole ausgewählt (Abbildung 7.3).

2. Zustand der vom **Roboterprogramm** vorgegeben wird.
   Innerhalb des Roboterprogramms kann der Zustand der Servos gesetzt werden, jedoch kann hier nur zwischen SERVO_STOP (Antriebe aus) und SERVO_WAIT (Antriebe ein) gewählt werden.

3. Zustand beim **letzten Takt der Steuerung**.
   Dies ist der Zustand, den die Zustandsmaschine der Robotersteuerung im letzten Takt ermittelt hat und der als Wunschzustand an die Servoverstärker ausgegeben wird.

4. Gegenwärtiger **Zustand der Servoverstärker**.
   Dies ist der Ist-Zustand der Servoverstärker. Dieser kann sich u.U. vom Wunschzustand der Steuerung unterscheiden, falls z.B. ein Wechsel mit einer Verzögerung behaftet ist, oder ein interner Fehler vorliegt, etwa weil die Stromversorgung der Servoverstärker fehlerhaft ist. Können die Servoverstärker nicht binnen einer Toleranzzeit den gewünschten Zustand der Robotersteuerung annehmen, so wird ein Fehler ausgelöst.

Nur in dem Fall, dass sich einer dieser Werte ändert, wird die Zustandsmaschine aktiv und es muss entschieden werden, ob sich der Zustand der Steuerung wirklich ändern darf. Ändern sich mehrere Zustände gleichzeitig, so muss ebenfalls entschieden werden, welcher die höhere Priorität hat. Grundsätzlich kann hier festgehalten werden, dass der Wunsch nach dem kleinsten Servozustand (nach der Nummer in der o.a. Liste der Servozustände) aus Sicherheitsgründen grundsätzlich Vorrang hat.

Abbildung 7.3 zeigt den Bildschirmausschnitt mit der Konsole zur Steuerung der Servozustände. Die hervorgehobene Schrift ”SERVO_WAIT” signalisiert den aktuellen Servozustand. Diese

Abbildung 7.3: Ansicht der Konsole zur Steuerung der Servozustände

Schriften und auch der ”Notaus-Knopf” sind mit der Maus anklickbar, womit der Anwender seinen Wunsch mitteilt. Die unterlagerte Servo-Zustandsmaschine verarbeitet diese Anforderung, die je nach aktuellem Zustand erfüllt werden kann. Selbstverständlich können jederzeit der Notaus, Servo-Stop oder Servo-Error aktiviert werden. In der Abbildung ist das Feld ”Use Robot” als aktiv markiert, daher erfolgen sämtliche Interpolationen nicht nur in der Visualisierung sondern auch mit dem realen Roboter. Das mittlere weiße Feld bietet Platz für Systemmitteilungen des Robotertreibers und der Robotersteuerung.

Mit dieser Schnittstelle und dem Subsystem zur interaktiven Beeinflussung der angeschlossenen Hardware ist eine Basis zur Ankopplung verschiedener Achsregler und Servoverstärker vorhanden.

7.4 Echtzeitmodul im Ring 0 des Betriebssystems

Da die meisten Regler- bzw. Servosteuerungsmodule die Achssollwerte in einem festen Zeittakt erwarten, muss auch die Robotersteuerung diesen Zeittakt einhalten und stets pünktlich und synchron die Sollwerte bereitstellen. Innerhalb des Anwender (User)-Modus ist dies unter Windows NT nicht sicher möglich. Selbst die Ausführung von höchst-priorisierten Prozessen kann zu nicht tolerierbaren Verzögerungen und damit letztlich zu einer verspäteten Bereitstellung der Sollwerte führen. Daher ist es erforderlich, Module im Kernel Mode bzw. Ring 0 des Betriebssystems zu verwenden:

1. Innerhalb des User Mode kann nicht direkt auf die Hardware des Systems zugegriffen werden. Zugriffe sind nur von den Komponenten im Ring 0 des Systems erlaubt.

2. Die Bereitstellung der Sollwerte muss, wie in Kapitel 4.5.2 beschrieben, innerhalb des Interrupt-Mechanismus des PC erfolgen. Dieser ist nur innerhalb des Ring 0 programmierbar.

3. Bisher existieren keine Treiber, die mit bestehenden Robotern kommunizieren, so dass ohnehin Software im Ring 0 entwickelt werden muss.

Funktional stellt der Robotertreiber die Schnittstelle zwischen der Robotersteuerung, die im Anwendermodus des Systems läuft, und der Hardware dar. Da die Bewegungssteuerung selbst ebenfalls im Anwendermodus läuft, ist es innerhalb des Treibers aus Sicherheitsgründen notwendig,

einen Pufferspeicher als Warteschlange anzulegen. Die Bewegungssteuerung hat während der Interpolation diese Warteschlange an Achssollwerten stets gefüllt zu halten, während die Interrupt-Serviceroutine die Werte im Interpolationstakt pünktlich ausliest und an den Regler bzw. die Servomodule ausgibt.

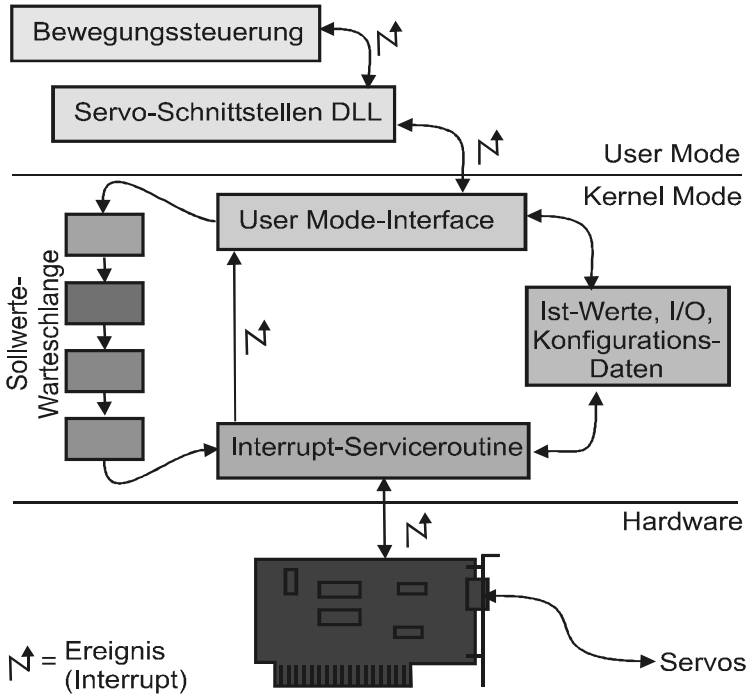


Abbildung 7.4: Umgebung des Roboter-Treibers

Abbildung 7.4 stellt das Schema eines Robotertreibers dar: Das ”User Mode-Interface” bildet die Treiberschnittstelle zum Anwendermodus. Die Kommunikationsmechanismen dieser Schnittstelle werden vom Betriebssystem einheitlich als ”File-System-Interface” zur Verfügung gestellt. Als hilfreiche Literatur ist hier [Bak97] und [Can99] zu nennen. Innerhalb des User Mode Interfaces befindet sich die Datenverwaltung des Treibers, mit der die internen Datenflüsse kontrolliert werden. So werden Daten, die unbedingt synchron zum Interpolationstakt ausgetauscht werden müssen, in die Warteschlange der Sollwerte eingereiht. Daten, die nicht dieser Priorität unterliegen, und Daten, die in Richtung der Robotersteuerung fließen, werden in dem internen Datenpool gespeichert. Die Kommunikation mit der Hardware erfolgt innerhalb der Interrupt-Serviceroutine. Diese überwacht auch, ob die Warteschlange nachgefüllt werden muss und signalisiert ggf. den höheren Ebenen (bis hinauf zur Bewegungssteuerung) die Dringlichkeit, neue Interpolationstakte zu berechnen. Innerhalb der Interrupt-Serviceroutine befinden sich auch Komponenten des Sicherheitsmechanismus, der im Fehlerfall das Anziehen der Bremsen veranlasst. Diese Stelle ist dafür sehr geeignet, denn hardwaremäßig ausgelöste Interrupts werden häufig auch dann noch bearbeitet, wenn der PC für den Anwender ”abgestürzt” scheint. Die Auslegung des Robotertreibers ist bisher roboterspezifisch. Derartige Implementierungen können basierend auf bereitstehenden Beispielimplementierungen auch durch Dritte geschehen.

7.5 Echtzeitverhalten der realisierten Lösung

In umfangreichen Untersuchungen wurde festgestellt, dass das Windows NT 4.0 Betriebssystem, auf einem Pentium II 300 MHz PC im Anwendermodus, auf eine Signalisierung durch ein Semaphor nicht garantiert innerhalb von 30 ms reagiert. In seltenen Fällen ergaben sich Verzögerungen bis zum Vielfachen dieses Wertes. Diese Verzögerungen machen den o.a. Einsatz der Warteschlange im Ring 0 notwendig. Während des Betriebs der Robotersteuerung wurden das Timing und der Füllungsgrad der Warteschlange stets überwacht und protokolliert. Dabei stellte sich heraus, dass bei dem realisierten System im Durchschnitt etwa alle sechs Stunden ein solches Ereignis nicht rechtzeitig genug erkannt wurde, so dass sich die Länge der Warteschlange für einen Takt um Eins verkürzte. Etwa alle zehn Betriebsstunden wurden zwei Takte versäumt. Ließ man auf dem System andere Programme laufen, die intensiv auf andere Komponenten im Kernel Mode zugriffen, so kam es häufiger zu verpassten Signalisierungen in den User Mode. Es traten teilweise Verzögerungen von bis zu 5 Interpolationstakten (150 ms) auf. Diese Situation ist allerdings bewusst künstlich hervorgerufen worden und hat nichts mit den allgemeinen Umständen beim Betrieb einer Robotersteuerung zu tun. Kein Hersteller einer Robotersteuerung gestattet die Installation - geschweige denn den Betrieb - anderer als der Steuerungssoftware auf dem System, schon gar nicht parallel zum Roboterbetrieb. Um den allgemeinen Betrieb der Steuerung zu gewährleisten und um ”Timing”-Probleme auszuschließen, wird daher die Solllänge der Warteschlange auf acht Werte gesetzt, was einer Pufferzeit von 240 ms bei 30 ms Interpolationszeit entspricht.

Hat die Bewegungssteuerung ein Defizit aufzuarbeiten, so werden die in der Warteschlange fehlenden Sollwerte ohne Unterbrechung berechnet. Die Berechung eines Interpolationstaktes dauert auf dem beschriebenen System nur wenige Mikrosekunden, so dass die Warteschlange in Bruchteilen einer Millisekunde wieder gefüllt ist. Während des Teach-Betriebs oder bei Sensorbewegungen wird die Sollgröße des Puffers auf zwei Einträge begrenzt, damit der Roboter nicht ”nachläuft”, also der Roboter beim Loslassen der Aktionstaste zum Verfahren möglichst schnell zum Stehen kommt. Mit diesen Einstellungen werden die realisierten Systeme erfolgreich eingesetzt. An dieser Stelle muss erneut erwähnt werden, dass nicht uneingeschränkt sämtliche auf dem Markt zur Verfügung stehende Hard- und Software parallel zum Roboterbetrieb geeignet ist. Bestimmte Gerätetreiber haben die Eigenschaft, während eines eigenen Datentransfers sämtliche anderen Interrupts zu sperren und damit das gesamte System zu blockieren. Gegen diese Art des Datenaustausches wurden bei den bisher realisierten Treibern keine Gegenmaßnahmen integriert[8], da der einfache Verzicht auf deren Einsatz die praktikablere Lösung darstellt. Diese Einschränkung stellt grundsätzlich kein Manko des PCs gegenüber nativen Echtzeit- oder Embedded Systemen dar, denn auch auf solchen Systemen beeinflusst die Auswahl der betriebenen peripheren Systemkomponenten das Zeitverhalten des Gesamtsystems.

7.6 Interpolationsschnittstelle der Bewegungssteuerung

In Kapitel 6.1 wurde bereits die Schnittstelle der Bewegungssteuerung erwähnt, mit der von extern auf die Bewegungsinterpolation Einfluss genommen werden kann. Diese Schnittstelle ba-

[8] Ein solches Beispiel ist der ZIP-Treiber für externe Parallelport-Wechselplatten.

siert ebenso wie die Treiber-Schnittstelle auf dem DLL-Konzept des Betriebssystems, bei dem die Steuerung Funktionen innerhalb der DLL aufruft. Diese Verfahrensweise stellt sich auch hier als extrem zuverlässig und schnell heraus.

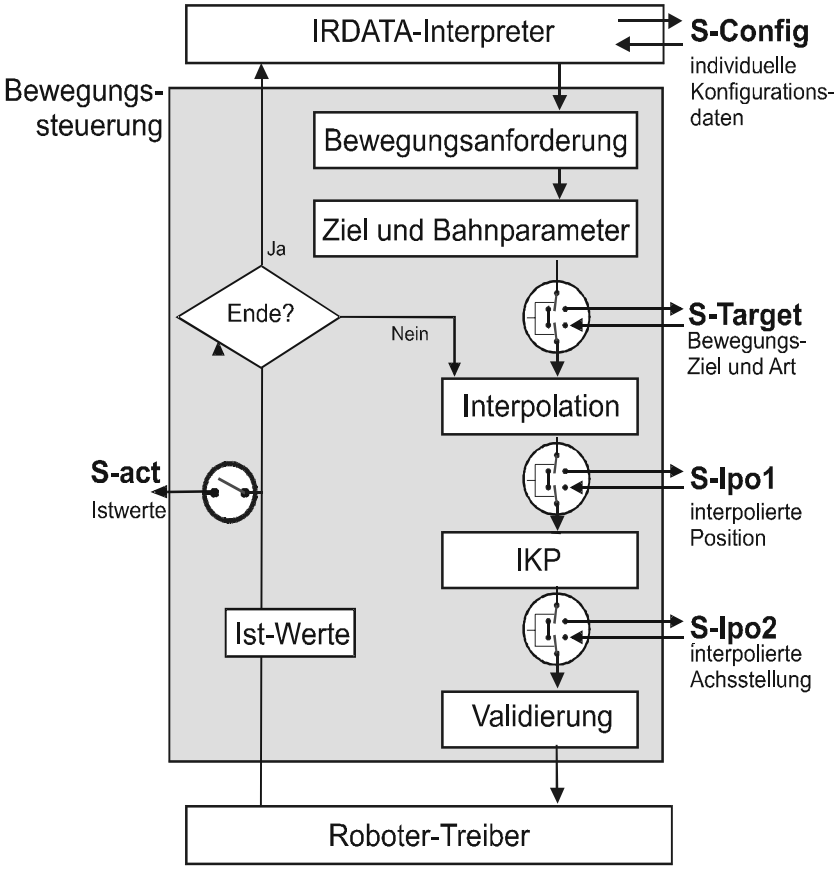


Abbildung 7.5: Schnittstellen der Bewegungssteuerung

In Abbildung 7.5 sind die Eingriffspunkte der Schnittstelle in die Bewegungssteuerung dargestellt:

1. **S-Config:**
   Diese Funktion der Schnittstelle dient dem anwenderspezifischen Datenaustausch zwischen Roboterprogramm und der angeschlossenen DLL. Hiermit können sowohl Text- als auch Binärdaten mittels eines Speicherblocks definierbarer Länge ausgetauscht werden. Diese Schnittstelle läuft asynchron zum Interpolationstakt.

2. **S-Target**:
   Diese Schnittstelle ermöglicht die Modifikation des Bewegungssatzes. Der Datentyp ähnelt dem aus Kapitel 6.1.1 in Abbildung 6.4, mit dem der Bewegungssteuerung ihre Aufträge mitgeteilt werden. In Abbildung 7.6 ist die Definition dieses Datentyps abgedruckt. Sämtliche Parameter der Datenstruktur dürfen innerhalb der DLL modifiziert werden, so dass ein Maximum an Flexibilität gewährleistet ist. Diese Schnittstelle läuft asynchron zum Interpolationstakt.

```
typedef struct move_target{
    /*Art der Bewegung*/
    int    movtype;     /* Bewegungstyp: MOV_0,MOV_LINEAR,MOV_PTP,MOV_CIRCULAR  */
    POSE   start;       /* Startposition in Weltkoordinaten                     */
    POSE   end;         /* Welt-Zielposition                                    */
    POSE   inter;       /* Welt-Zwischenposition bei Zirkularbahnen             */
    int     CircleType; /* Modus, in dem eine Kreisbahn gefahren werden soll    */
    double  angle;      /* Winkel des Kreisbogens, wenn dieser gegeben          */
    int    timeforced;  /* Die Zeit ist vorgegeben (nicht die Geschwindigkeit), */
                        /* wenn dieser Wert != 0                                */
    double forced_time; /* ggf vorgegebene Bewegungszeit                        */
    int    relative;    /* Flag: Zielpunktangaben sind relativ zur              */
                        /* augenblicklichen Position                            */
    double  path_vel;   /* Roboter-Bahngeschwindigkeit                          */
    double  path_acc;   /* Roboter-Bahnbeschleunigung                           */
    double  OriSpeed;   /* Winkelgeschwindigkeit beim Bahnfahren                */
    double  OriAcc;     /* Winkelbeschleunigung  beim Bahnfahren                */
    JOINT   joint_vel;  /* Max Achsgeschwindigkeiten                            */
    JOINT   joint_acc;  /* Max Achsbeschleunigungen                             */
    double  veloverride;/* Override ]0.0 .. 1.0]                               */
    double  accoverride;/* Override ]0.0 .. 1.0]                               */
    WORLD   TCP_Offset; /* Tool-Versatz                                         */
    /*Modus der 2ms FIPO-Schnittstelle*/
    enum TORQUE_CONTROL Sensormode; /* */
}MOVE_TARGET;
```

Abbildung 7.6: Bewegungsspezifikation der Interpolationsschnittstelle

3. **S-Ipo1:**
   Die Funktion, die vor dem Lösen des IKP aufgerufen wird, dient der Modifikation der kartesischen Position in Koordinaten des Bezugssystems und wird daher nur bei Bahnbewegungen aufgerufen. Die korrespondierende Datenstruktur ist in Abbildung 7.7 abgedruckt. Innerhalb dieses Aufrufes sind nur die kartesisch orientierten Daten relevant. Werden hier Werte in die Felder für Achsstellungen geschrieben, so ist dies ohne Wirkung. Diese Schnittstelle läuft synchron zum Interpolationstakt und wird von der Bewegungsinterpolation aufgerufen.

4. **S-Ipo2:**
   Diese Schnittstellenfunktion dient der Änderung der Achswerte. Sie wird sowohl bei PTP als auch bei Bahnbewegungen aufgerufen. Die verwendete Datenstruktur ist die gleiche wie bei S-Ipo1. Um welche Funktion es sich bei dem Aufruf handelt, ist am Wert der Variablen ax_valid abzulesen, die nur dann "1" bzw. "TRUE" ist, wenn die Achswerte gültig sind. Relevant sind innerhalb dieses Aufrufes lediglich die Achswerte. Diese Schnittstelle läuft synchron zum Interpolationstakt und wird von der Bewegungsinterpolation aufgerufen.

5. **S-act:**
   Mit dieser Funktion werden die aktuellen Achswerte und -Geschwindigkeiten an die angekoppelte DLL übertragen, so dass hier z.B. auch auf Schleppfehler reagiert werden kann. Diese Schnittstelle läuft synchron zum Interpolationstakt.

Ziel bei der Konzeption dieser Schnittstelle war es, für Industrie, Forschung und Didaktik geeignete Eingriffsmöglichkeiten in die Bewegungssteuerung zu schaffen, mit denen primitive, aber auch aufwändige Bewegungsbeeinflussungen möglich sind. So kann mittels S-Target das

```
typedef struct ipo_dat{
    // Allgemeine Daten
    int     ipo_counter;    /* Interpolationszähler                              */
    int     Max_ipo_counter;/* Maximaler Interpolationszähler                    */
    // für Bahnbewegungen erforderliche Daten  (IPO1)
    int     pos_valid;      /* Ist nur 1 bei Bahnbewegungen (IPO1).             */
    WORLD   pos;            /* In dem Takt anzufahrende Kart. Pos                */
    VECTOR  vel;            /* Bahngeschwindigkeit                               */
    VECTOR  rotvect;        /* Rotationsachse                                    */
    // für PTP und Lin erforderliche Daten ( IPO1 und IPO)
    int     ax_valid;       /* ist 1 bei PTP-Bewegungen, und bei Bahnbewegungen */
                            /* nach der Rücktrafo.                               */
    JOINT   ax_pos;         /* interpolierte Achsstellung, die an die Regler     */
                            /* übergeben wird                                    */
    JOINT   ax_vel;         /* Achsgeschwindigkeiten                             */
    JOINT   ax_acc;         /* Achsbeschleunigungen                              */
    double  total_pathlen;  /* Gesamte Bahnlaenge bei Bahnbewegungen            */
    double  current_pathlen;/* bisher zurueckgelegte Bahnlaenge                  */
    ROB_KONF konfig;        /* Konfiguration der kinematischen Struktur          */
    /*Handshake mit der Sensorik */
    ERROR_SPEC  kin_error;  /* Fehler der Bewegungssteuerung                     */
    int     MOV_END_request;/* Letzer IPO-Takt der PCROB für diese Bewegung      */
                            /* True, wenn dies der letzte Takt der Bewegung ist,*/
                            /* kann von außen überschrieben werden,             */
                            /* dann gehts noch einen Takt weiter.                */
                            /* Solange, bis er nicht mehr überschrieben wird     */
}INTERPOLDATA;
```

Abbildung 7.7: Interpolationsdaten der Interpolationsschnittstelle

Bewegungsziel schon vor Bewegungsbeginn verändert werden, das u.U. erst zur Laufzeit des Roboterprogramms zur Verfügung steht, oder die Bewegung kann vollständig unterbunden und gespeichert werden.

Die Möglichkeit der Manipulation von kartesischen Positionen sowohl asynchron als auch im Interpolationstakt gestattet es, z.B. Sensorfunktionen zu programmieren. So könnte bei Auftreten eines Hindernisses die Bewegung kartesisch umgeleitet oder gestoppt werden. Die Detektion des Hindernisses muss dann Bestandteil der externen Anordnung sein.

7.7 Sicherheitssysteme

Die Einhaltung der Sicherheitsvorschriften (UVV, DIN-Normen, sowie VDI- und VDE-Richtlinien) dient in erster Linie der Sicherheit von Programmierer und Bediener. Dabei ist zwischen Sicherheitsmaßnahmen im Automatik- und im Einrichtebetrieb zu unterscheiden.

7.7.1 Automatik-Betrieb

Der Gefahrenbereich des Industrieroboters darf während des Automatikbetriebs nicht erreicht oder betreten werden können. Es müssen alle Schutzeinrichtungen wirksam sein, damit keine Personen in den Gefahrenbereich gelangen können. Dies wird durch verschiedene Maßnahmen erreicht:

- Schutzzäune oder Kabinen, die gegen Übergreifen oder Durchgreifen gesichert sein müssen, und elektrisch überwachte Türen, die beim Öffnen den Industrieroboter stillsetzen,

- Lichtvorhänge, die beim Betreten den Roboter anhalten,
- Lichtschranken,
- Schaltplatten, die durch eine Umzäunung gegen Umgehen gesichert sind.

Bei der Planung eines Industrieroboter-Systems ist eine ausreichend große Schutzzone vorzusehen, die durch eine Umzäunung gesichert wird. Innerhalb dieser Schutzzone liegt der Gefahrenbereich, der softwaremäßig oder durch mechanische Anschläge begrenzt bzw. verkleinert werden kann. Die verbleibende Schutzzone muss, vor allem im Hinblick auf den Einrichtebetrieb, ausreichend groß sein.

Die Detektion eines unerlaubten Zustandes ist naturgemäß nicht die Aufgabe der Steuerung, sondern sie wird von externen Einheiten übernommen. Die Signalisierung eines derartigen Zustandes an die Robotersteuerung erfolgt zunächst hardwaremäßig durch den Pegel auf der Notaus-Leitung, womit der Roboter augenblicklich stillgesetzt wird.

7.7.2 Einrichtebetrieb

Auch im Einrichtebetrieb sollte sich der Bediener möglichst außerhalb der Schutzzone aufhalten. Sind jedoch Einrichtearbeiten (Programmieren und Testen) in unmittelbarer Nähe des Industrieroboters erforderlich, so dürfen die Schutzvorrichtungen für überschaubare Bereiche aufgehoben werden. Der Industrieroboter ist dann nur noch über das Handbediengerät verfahrbar, was mit zwangsweise reduzierter Geschwindigkeit und mit verringerten Drehmomenten und nur im Tippbetrieb[9] möglich ist.

7.7.3 Sicherheitsfunktionen für Robotersteuerungen

Eine gute Übersicht über die aktuellen Sicherheitsbestimmung beim Betrieb von Anlagen mit Handhabungsautomaten geben [VDI87] und [VDI88]. Unter anderem sind dort folgende Punkte enthalten:

- Not-Aus-Taster:
  Im Not-Aus-Stromkreis sind alle Öffnungskontakte in Reihe geschaltet, so dass durch Betätigen eines Schalters der Stromkreis unterbrochen wird. Ferner werden mit diesem Kreis sämtliche erforderlichen Sicherheitseinrichtungen verbunden.
- Leistungsreduzierung in der Betriebsart ”Hand”:
  In der Betriebsart ”Hand” wird die den Antrieben zugeführte Spannung umgeschaltet. Dadurch reduzieren sich die von den Antrieben aufgebrachten Kräfte und dadurch die Gefahr von Quetschungen. Andererseits werden aber auch die Fügekräfte entsprechend reduziert.

[9] Tippbetrieb: Nur bei gedrückter Taste fährt der Roboter.

- Vorrangschaltung des Programmierhandgerätes:
  Der Bediener muss in der Betriebsart ”Hand” oft im Gefahrenbereich des Roboters arbeiten. Um unvorhergesehene automatische Bewegungen des Roboters zu vermeiden, ist das Handbediengerät gegenüber dem Gesamtsystem des Roboters bevorrechtigt.

Die Europäischen Normen [Nor94b], [Nor98], [Nor96], [Nor99], [Nor94a] und [Nor92] regeln das Verhalten auf verschiedenen Ebenen von Maschinen und auch Roboteranlagen und bestimmen den Umgang mit dem System, so dass selbst bei Manipulationen durch die Anwender möglichst keine Gefahr von der Anlage ausgeht. Diese Sicherheitsfunktionen müssen von jedem Roboter, der in Deutschland eine Zulassung erhalten soll, eingehalten werden. Funktional unterstützt[10] die in dieser Arbeit beschriebene R/VRC diese Mechanismen.

7.8 Simulation auf verschiedenen Ebenen

Bei dem hier beschriebenen System handelt es sich um ein integriertes Steuerungs- und Simulationssystem, bei dessen Konzeption es von Beginn an ein Ziel war, während des Betriebes zwischen dem reinen Simulationsbetrieb und dem kombinierten Betrieb umschalten zu können. So kann der Anwender sein Programm zunächst im Simulationslauf verifizieren und bei Erfolg sofort mit dem realen Roboter laufen lassen. Das System unterstützt dabei mehrere ”Ebenen” der Robotersimulation. Die ”Ebene” bezeichnet dabei die Hierarchie der Schnittstelle zur Hardware, ab der keine Signale mehr in Richtung der unteren Schichten abgegeben werden.

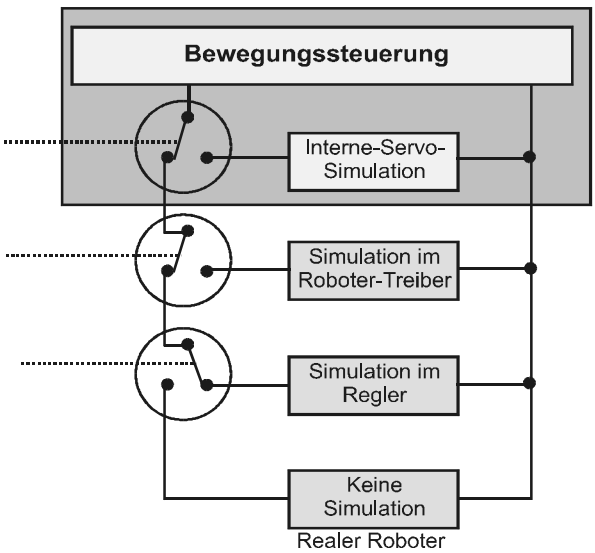


Abbildung 7.8: Simulation auf verschiedenen Ebenen

Abbildung 7.8 stellt die verschiedenen Ebenen, auf denen eine Simulation des Roboters möglich ist, schematisch dar. Im Folgenden wird auf diese verschiedenen Ebenen der Simulation genauer eingegangen.

[10] Die unterlagerte Hardware muss diese Richtlinien ebenfalls erfüllen und durch die Interfacefunktionen der Steuerung passend angebunden sein.

7.8.1 Simulation ohne vorhandenen Gerätetreiber

Die oberste Ebene der Robotersimulation ist die Simulation ohne angekoppelten Gerätetreiber innerhalb der R/VRC. Diese Stufe kommt zum Einsatz, wenn die Robotersteuerung keine Treiber-DLL vorgefunden hat, oder mit der vorgefundenen DLL oder den Roboterparametern der DLL nicht kompatibel ist[11]. In diesem Modus wird ein innerhalb der Robotersteuerung zur Verfügung stehender virtueller, generischer Robotertreiber eingesetzt, der die von der Robotersteuerung vorgeschlagenen Parameter anstandslos annimmt und während des folgenden Betriebes auch berücksichtigt. Diese Parameter werden während der Initialisierung der Arbeitszelle übertragen. Siehe hierzu auch Kapitel 7.3 in Punkt 1 der Liste. In dieser Betriebsart der Steuerung ist keine Ansteuerung eines realen Roboters möglich, da zur Ansteuerung ein passender Treiber erforderlich wäre. Dennoch ist dies ein ganz gewöhnlicher Zustand, der immer dann eingesetzt werden kann und wird, wenn das System in Büros zum Einsatz kommt, etwa um Entwicklungsarbeiten offline an dem Arbeitszellenmodell vorzunehmen. Abbildung 7.9 stellt diesen Modus der Steuerung dar, bei der keine Verbindung mit dem Robotertreiber besteht.

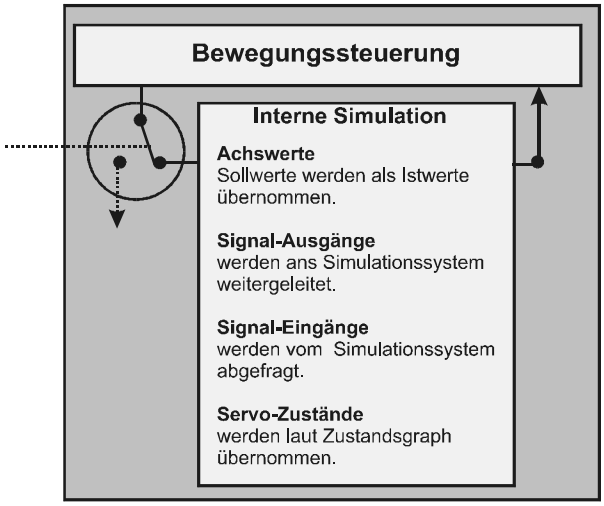


Abbildung 7.9: Interne Simulation

Die kreisförmige Einheit in der Grafik soll einen Schalter in dem Zustand darstellen, bei dem der Datenfluss in die interne Simulation geleitet wird. Innerhalb der internen Simulation werden, wie in jedem anderen Betriebsmodus auch, die Achswerte, die digitalen und analogen Ein- und Ausgänge sowie die Servo-Zustände bearbeitet. Innerhalb dieser und den folgenden Grafiken wird die Behandlungsweise dieser Daten kurz kommentiert.

7.8.2 Simulation innerhalb eines vorhandenen Robotertreibers

Wird während der Initialisierung ein kompatibler Robotertreiber vorgefunden und kann dieser erfolgreich initialisiert werden, so wird innerhalb der Robotersteuerung davon ausgegangen, dass

[11] Das ist dann der Fall, wenn die von der Robotersteuerung vorgeschlagenen Parameter zu sehr von den veränderten Werten des Treibers abweichen, so daß ein weiterer Betrieb nicht sinnvoll ist.

ein realer Roboter angeschlossen ist, und der Datenaustausch wird über die eingebundenen Funktionen der Antriebsschnittstelle abgewickelt. Abbildung 7.10 stellt diese Betriebsart dar, bei der sich die Antriebsschnittstelle der Steuerung an die erste Ebene des Robotertreibers ankoppelt, die ggf. vom Anwender bzw. Roboterarmhersteller selbst implementiert wurde. Auch auf dieser Ebene sind Probeläufe ohne Verwendung eines realen Roboters möglich, wenn hier die Vorgaben der Steuerung zu simulierten Istwerten verarbeitet werden. Diese Ebene der Schnittstelle kann bereits roboterspezifisch sein und der Anwender hat hier die Möglichkeit, eigene Verfahren zur Simulation zu realisieren oder einfach die Sollvorgaben der Steuerung als Istwerte zu übernehmen. Der Simulationsbetrieb auf dieser Ebene kann mittels des anklickbaren Feldes ”Use Robot” in Abbildung 7.3 aktiviert oder deaktiviert werden.

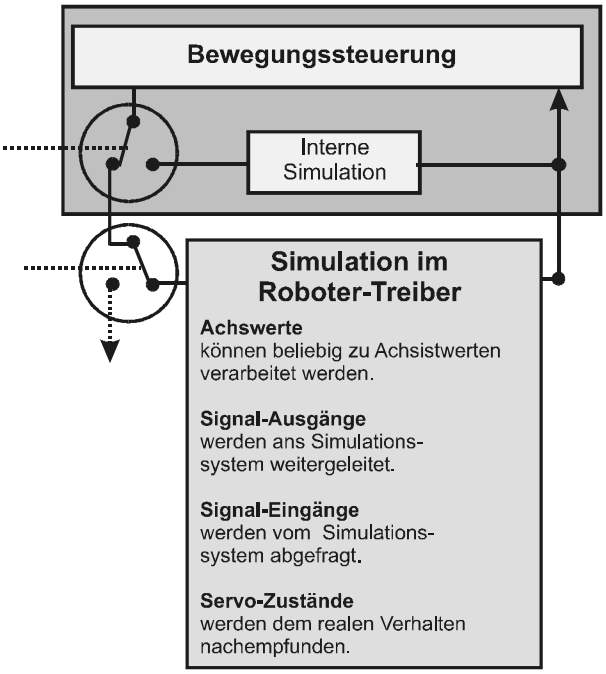


Abbildung 7.10: Simulation im Treiber

7.8.3 Simulation innerhalb des Reglers

Zusätzlich zu den bisher beschriebenen Simulationsebenen kann es im Einzelfall wichtig sein, die gesamte Kommunikationskette bis hin zu den Servoverstärkern in den Simulationslauf mit einzubeziehen, ohne den realen Roboter bewegen zu müssen, etwa zur Diagnose von Teilkomponenten wie der Feininterpolation oder den Kommunikationskanälen selbst. Ein weiterer, entscheidender Vorteil der Möglichkeit zur Simulation auf Reglerniveau besteht darin, dass bestimmte Warnungen oder Fehler, die sonst erst während des realen Ablaufes gemeldet werden, auch schon während eines Simulationslaufes diagnostiziert werden können, da in diesem Modus die Feininterpolation aktiv ist. Dieser Simulationsmodus wird, wie die Simulation innerhalb der Roboter-Treiber-DLL, mittels einer speziellen Variablen der Antriebsschnittstellen konfiguriert und stellt gleichzeitig die hierarchisch unterste Ebene der direkt unterstützten Simulationsmodi

dar. Verwendbar ist dieser Modus selbstverständlich nur, wenn auch die Feininterpolations- bzw. Reglerebene die Möglichkeit dazu bietet. Moderne Servoverstärkermodule verfügen mittlerweile teilweise über diese Eigenschaft. Bei der Ankopplung des Manutec Roboters, worauf in Kapitel 8.1.1 genauer eingegangen wird, kommt an dieser Stelle die Basissteuerung (siehe [Pes97]) zum Einsatz, die sogar eine dynamische Simulation des Roboters zur Verfügung stellt. Abbildung 7.11 stellt die Verhältnisse in dieser Umgebung dar.

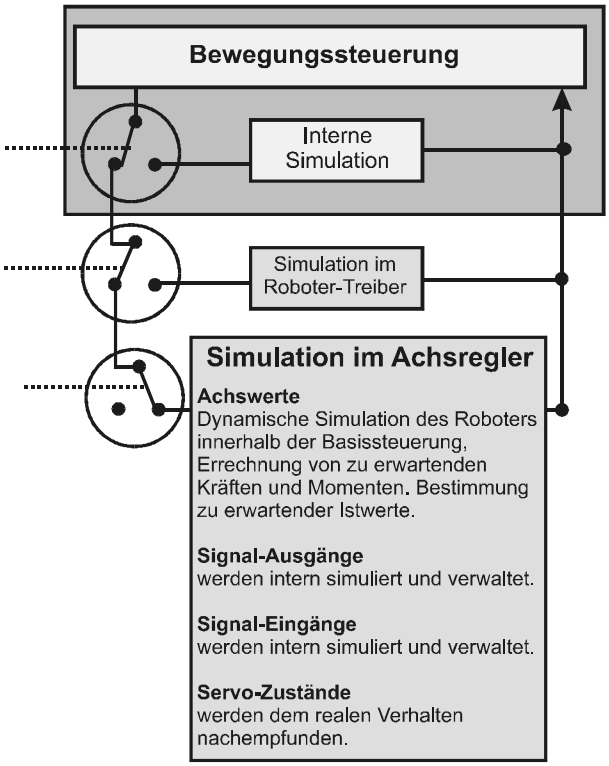


Abbildung 7.11: Simulation im Regler

Auch weitergehende Roboter-Dynamiksimulationen sind daher an die vorhandenen Schnittstellen ankoppelbar, sofern ein kompatibler Robotertreiber zur Verfügung steht. Die Steuerung selbst behandelt die Daten, die ihr von der Treiber-DLL gemeldet werden, stets einheitlich, unabhängig davon ob eine Simulation in einer der Ebenen oder der reale Betrieb aktiv ist. Eine Dynamik-Simulation kann sich an dieser Schnittstelle, aus Sicht der Robotersteuerung, wie eine reale Steuerung verhalten. Daher eignet sich dieses System auch als Basisplattform für Forschung und Entwicklung neuer Steuerungstechnologien.

7.8.4 Betrieb ohne Simulation

Abbildung 7.12 stellt den Betrieb ohne Simulation dar. Sämtliche Simulationsebenen werden übergangen, und die Vorgaben der Steuerung werden bis auf die Hardware der Antriebssysteme hindurchgereicht. Die Istwerte werden von der realen Roboterhardware ausgelesen und an die Steuerungsschnittstelle übergeben.

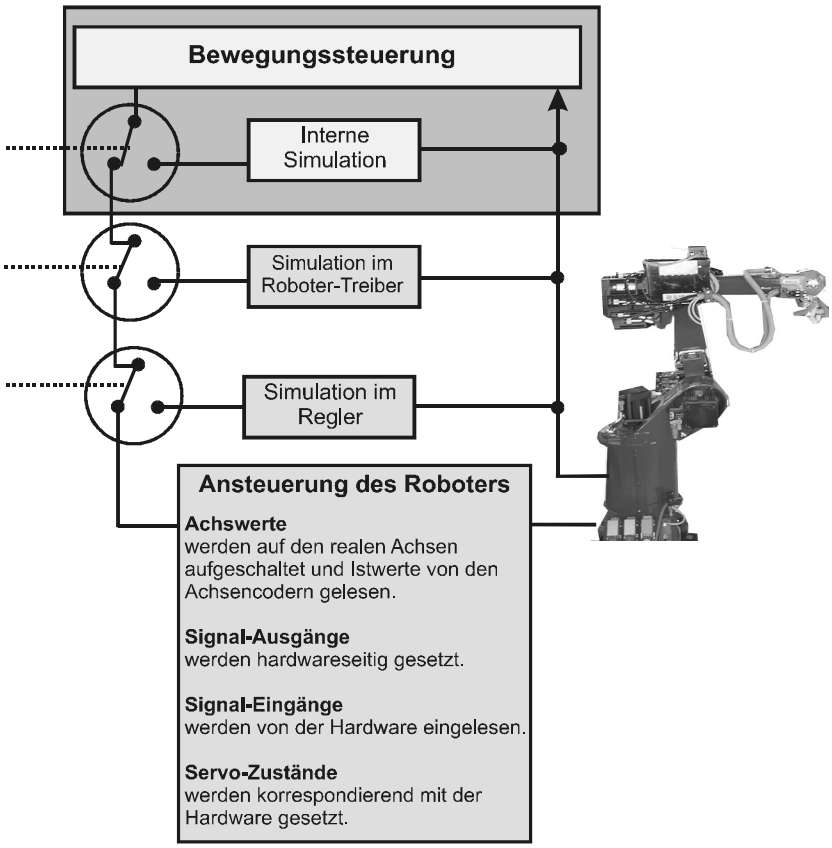


Abbildung 7.12: Keine Simulation

Mit diesen Möglichkeiten der Simulation auf verschiedenen Ebenen kann der Anwender seine Entwicklungen bei Bedarf sukzessive näher an die Hardware bringen. Diese verschiedenen Stufen der Simulation finden stets auf Ebenen statt, zwischen denen roboterunabhängige Schnittstellen definiert wurden. Diese Schnittsellen sind auch sehr gut für andere Zwecke geeignet, etwa um Protokolle (”Logs”) von Abläufen zu erstellen. Bei besonderen Einsatzgebieten (etwa im medizinischen Bereich) wird ein Protokoll über sämtliche aufgeschalteten Achswerte verlangt. Die dazu erforderlichen Daten lassen sich von jeder dieser Schnittstellen ablesen.

Kapitel 8

Ankopplung realer Kinematiken

In den vorangegangenen Kapiteln wurden die Bestandteile des Steuerungssystems beschrieben, aus denen der Aufwand zur Ankopplung neuer Kinematiken bereits grob abgeschätzt werden kann. Dennoch wird in diesem Kapitel eine Zusammenstellung der anfallenden Arbeiten gegeben, um damit die Planung zukünftiger Projekte mit diesem Steuerungssystem zu vereinfachen:

1. Wenn die Kinematik, die angesteuert werden soll, zu keiner unterstützten Kinematikklassen gehört (siehe Kap.5.5), dann muss eine neue Klasse angelegt werden, die möglichst universell die Funktionen zur Lösung des DKP und IKP bereitstellt, und in die Steuerung integriert werden. Dieser Fall tritt jedoch nur bei neuen Sonderkinematiken ein, die für Bahnbewegungen eingesetzt werden sollen. Je nach kinematischem Aufbau ist entweder die Lösung des DKP recht aufwändig (z.B. bei Steward-Kinematiken) oder das IKP (z.B. bei Kinematiken mit mehr als sechs Achsen, die ggf. untereinander verkoppelt sind, wie etwa bei der bereits unterstützen Flexiwrist).
   Wird die Kinematik bereits von den implementierten Klassen unterstützt und kann mittels der linearen Achstransformationen abgebildet werden, dann muss innerhalb der Steuerung kein Aufwand betrieben werden; die Steuerungssoftware kann unverändert eingesetzt werden.

2. Wird der neue Roboter kinematisch von der Steuerung unterstützt, so kann er mittels Modelldatei in das Simulationssystem geladen werden. Von dort wird auch die Robotersteuerung passend initialisiert. Daher muss dieses mathematische bzw. geometrische Modell vorgegeben werden. Der zeitliche Aufwand, der hier zu veranschlagen ist, hängt von den zur Verfügung stehenden 3D-Daten des Roboters ab, und wie einfach diese in das Simulationssystem übernommen werden können.

3. Neben dem geometrischen Modell müssen die Parameter[1] des neuen Roboters ermittelt und in die Modelldatei eingetragen werden. Diese Daten können im Allgemeinen direkt den Handbüchern und Konstruktionszeichnungen entnommen werden.

4. Zum Ansprechen der Hardware des neuen Roboters muss eine Schnittstellenkarte zur Verfügung stehen, deren Bus-Schnittstelle vollständig dokumentiert ist, und auf die vom Kernel Mode aus zugegriffen werden kann. Vom Hersteller mitgelieferte Treiber, die lediglich

[1] Verfahrbereiche, max. Geschwindigkeiten und Beschleunigungen usw.

Zugriffe im Anwendungsmodus erlauben, sind ungeeignet, da diese i.A. nicht echtzeitfähig sind. In diesem Fall ist der Quellcode der mitgelieferten (Kernel Mode-) Treiber erforderlich. Dieser ist funktional in den Robotertreiber einzuarbeiten.

5. Der (Fein-) Interpolationstakt der Steuerung muss als pünktlich generierter Interrupt vorliegen: Findet die Feininterpolation auf externen Reglern statt, so muss entweder der Takt innerhalb des PC erzeugt und an die Reglermodule übertragen werden, oder die von den Reglern vorgegebenen Takte müssen synchronisiert und in den PC gespeist werden. Basierend auf diesem Takt wird ein PC-Interrupt generiert, zu dem die Steuerung synchron laufen kann.

6. Basierend auf dem Takt und den Kommunikationsfunktionen zum Zugriff auf die Schnittstellenhardware, mit der die Daten dem User Mode zur Verfügung gestellt und von der Roboter-Treiber-DLL an die Hardware weitergegeben werden, kann die Basiskommunikation innerhalb des Kernel Modes implementiert werden.
   Hier kann leider keine pauschale Aufwandsabschätzung erfolgen, da je nach Art der Schnittstelle ggf. nur ein minimaler Aufwand notwendig ist, etwa wenn die Schnittstelle auf der digitalen Übertragung der Soll- und Istwerte ohne aufwändiges Protokoll beruht. Der Aufwand kann allerdings auch enorm sein, wenn umfangreiche Feldbus-Protokolle gefahren werden müssen[2]. Die Entwicklungen von Komponenten im Windows NT Kernel Mode können allgemein als sehr aufwändig klassifiziert werden und erfordern die Kenntnis des speziellen Kernel-Subsystems. Die hier zur Verfügung stehenden Betriebssystemfunktionen und Vorgehensweisen unterscheiden sich weitgehend von denen im User Mode.

7. Steht kein externes Reglermodul bzw. kein Feininterpolator zur Verfügung, so könnten diese Komponenten ebenfalls innerhalb des Kernel Modes realisiert werden.

8. Neben den Komponenten im Kernel Mode ist die Roboter-Treiber-DLL die Komponente im User Mode, mit der die Robotersteuerung mit den Kernel-Komponenten kommuniziert. Innerhalb dieser DLL finden Umrechnungsaufgaben zur Adaption der Datenformate der Robotersteuerung in die des Kernel Mode-Moduls und umgekehrt statt. Für den Betriebsmodus der Simulation innerhalb des Roboter-Treibers werden hier die Daten des Simulationsbetriebes und des Betriebes mit realem Roboter konsistent gehalten und verwaltet. Je nach Auslegung des unterlagerten Kernel Mode-Moduls müssen hier verschiedene Pufferspeicher und Achsumrechnungen erfolgen. Der Aufwand zur Realisierung dieses Moduls nimmt wegen ausgereifter Entwicklungsumgebungen und dem zu implementierenden Funktionsumfang im Allgemeinen deutlich weniger Zeit, als die Entwicklung des Kernel Mode-Moduls in Anspruch.

Die Gesamtheit der Ankopplung hängt also entscheidend von der Art der zur Verfügung stehenden Verbindung zur Hardware und der Qualität der bereitstehenden Quellcodes, die für die Treiberprogrammierung verwendet werden können, ab.

In dem folgenden Kapitel wird auf zwei bisher realisierte Anbindungen genauer eingegangen, die sich in ihrer hardwareseitigen Ankopplung deutlich unterscheiden.

[2] Feldbuskarten werden von allen Herstellern mit Treibern ausgeliefert, die aber leider für diesen Fall nicht verwendbar sind. Lediglich wenn der Quellcode der Treiber zur Verfügung steht, kann der Aufwand einer vollständigen Neuentwicklung vermieden werden.

8.1 Realisierte IR-Ankopplungen

Das in dieser Arbeit beschriebene Steuerungssystem wird am Institut für Roboterforschung bereits zur Steuerung von Industrierobotern eingesetzt. Im Rahmen des Projektes DEMON [TH99] wurden zwei Roboter an die Steuerung angekoppelt, auf deren Einzelheiten in den folgenden beiden Kapiteln eingegangen wird.

8.1.1 Ankopplung des Manutec r2 mittels Transputerlink

Bei dem Manutec r2-Roboter des Instituts für Roboterforschung handelt es sich um eine modifizierte Steuerung des Originalsystems, bei der die Prozessoreinheiten gegen Transputersysteme ausgetauscht wurden [Pes93]. Diese Transputersysteme enthalten eine Roboterbasissteuerung für die Feininterpolation und die Regelung. Zusätzlich ist eine aufwändige, sehr leistungsfähige Sensorregelung implementiert, die in [Pes97] ausführlich beschrieben ist. Die Kommunikation mit der überlagerten Robotersteuerung erfolgt mittels binärer Datenpakete, die wahlweise über die seriellen Transputer-Links oder über einen gemeinsamen Speicherbereich im VME-Adressbereich ausgetauscht werden können. Letztere Variante wurde mit einer DOS-basierten Robotersteuerung realisiert und ist in [dV95] genauer erläutert.

Die Ankopplung des Windows NT basierten Systems erfolgt via Transputer-Link. Über diese Verbindung werden Kommunikationspakete ausgetauscht, in denen die Basissteuerung sämtliche relevanten Zustände und Änderungen meldet (Status-Paket, SP) und die Robotersteuerung ihrerseits sämtliche Vorgaben übermittelt (Kommando-Paket, CP). Die Synchronisation erfolgt hierbei durch die Basissteuerung, die stets pünktlich im Interpolationstakt ihr Statuspaket an den Steuerungs-PC sendet. Der Empfang dieses Paketes löst den erforderlichen Interrupt aus, mit dem die Bewegungssteuerung synchronisiert wird. Innerhalb des Interrupts wird das nächste Kommandopaket aus der Warteschlange gelesen und an die Basissteuerung übertragen. Zusätzlich zu den Paketen, mit denen die Basissteuerung und die Robotersteuerung kommunizieren, dient die gleiche Verbindung auch der Standard Ein-/Ausgabe (Tastatur, Datei, Terminal) des Transputersystems. Das bedeutet diese Pakete werden asynchron zu den interpolationssynchronen Paketen versendet und müssen innerhalb des Robotertreibers entsprechend gefiltert und behandelt werden. Diese Pakete werden als I-Server-Pakete bezeichnet [Inm92].

Abbildung 8.1 stellt den grundsätzlichen Aufbau des Systems hierarchisch dar. Sämtliche Pakete vom und zum Roboter werden über die Transputerlink-Verbindung (A) ausgetauscht und auf der Schnittstellenkarte innerhalb des PCs mittels des Kommunikationsbausteins entgegengenommen. Dieser Baustein zeigt ein eingetroffenes Byte durch den Wechsel des Signalpegels eines Ausgangs an, der dazu verwendet wird, einen Interrupt innerhalb des Systems zu generieren. Der daraufhin aufgerufene ”Interrupt-Handler” überprüft zunächst, ob die ihm zugeordnete Signalquelle wirklich der Auslöser des Interrupts war, stellt bei Bestätigung die Interrupt-Quelle temporär ab[3] und empfängt solange Bytes vom Baustein, bis das Paket komplett, oder eine definierte Zeitdauer kein Byte mehr angekommen ist (”time-out”). Die eingelesenen Bytes stehen danach im Eingangspuffer. Hier wird das Paket auf dessen Zweck hin analysiert und in die Emp-

[3]Die Interruptquelle wird während des Ausführends des Interrupt-Handlers deaktiviert, damit keine Rekursion auftritt. Am Ende des Handlers wird der Interrupt wieder zugelassen.

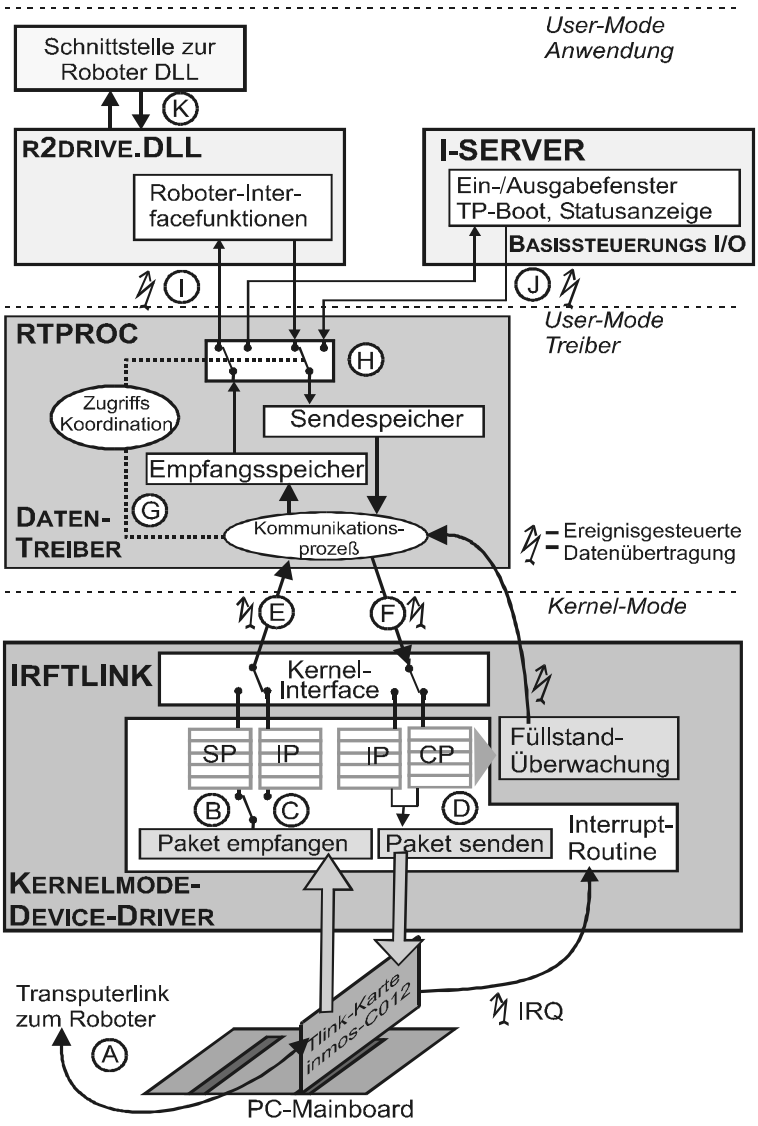


Abbildung 8.1: Systemstruktur der Ankopplung eines Manutec r2-Roboters

fangswarteschlangen für Status-Paket (SP)(B) oder I-Server-Paket (IP)(C) einsortiert. Im Anschluss an den Empfangsvorgang eines Status-Paketes wird das älteste im internen Sendepuffer bereitstehende Kommandopaket (CP) gesendet. Dieser Sendepuffer basiert auf dem in Kapitel 7.4 beschriebenen Mechanismus, mit dem die u.U. auftretenden zeitlichen Verzögerungen im User Mode ausgeglichen werden. Ist bis zu diesem Zeitpunkt kein weiteres Byte vom Roboter angekommen, dann werden auch noch ggf. wartende I-Server Pakete ausgegeben (D). Die integrierte Füllstandsüberwachung der CP-Warteschlange signalisiert dem überlagerten Kommunikationsprozess, dass ggf. weitere Kommandopakete erforderlich sind. Diese Füllstandsüberwachung wird bei jeglicher Art von Kommunikation mit dem Kernel Mode Treiber aufgerufen, also auch außerhalb des Interrupt-Handlers, wenn der User Mode ein neues Kommandopaket an den Treiber übergibt. In diesem Fall wird bei zu geringem Füllstand der CP-Warteschlange augenblicklich wieder ein Interpolationstakt ausgelöst, um den verlorenen Takt aufzuholen. Nachdem

der Datenaustausch innerhalb der Interrupt-Serviceroutine beendet wurde, wird die Interrupt-Quelle wieder eingeschaltet, so dass sich der Vorgang beim nächsten eintreffenden Byte automatisch wiederholt.

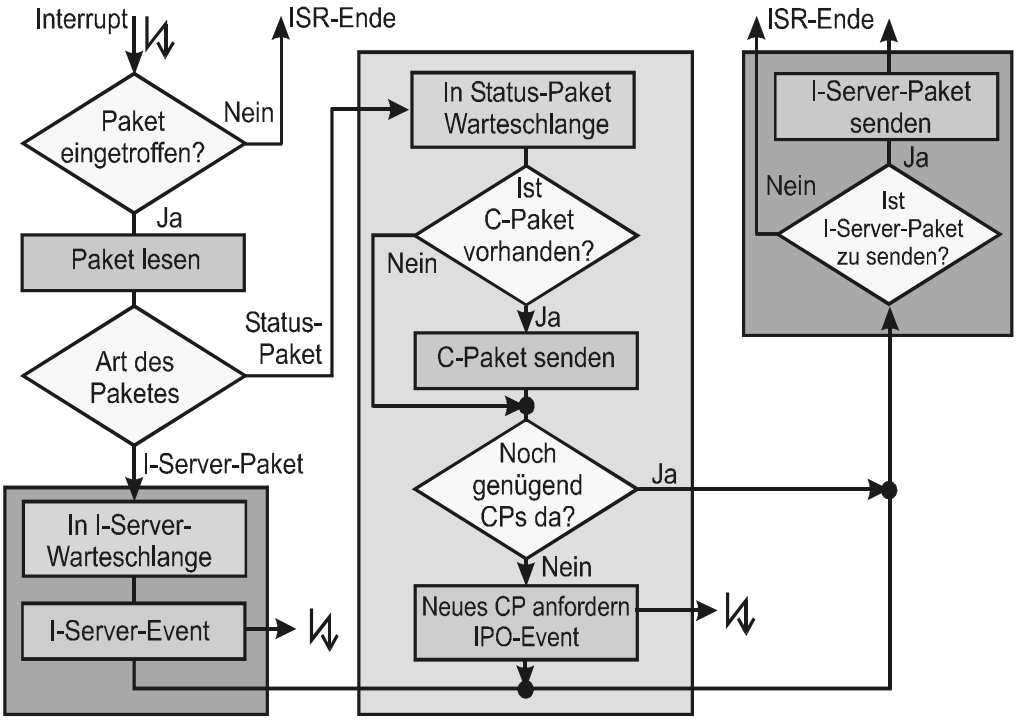


Abbildung 8.2: Ablauf innerhalb des Interrupt-Handlers

Abbildung 8.2 stellt den Vorgang des Austausches von Daten mit der Hardware und den internen Datenstrukturen innerhalb des Interrupt-Handlers als vereinfachtes, schematisches Ablaufdiagramm dar. Die realisierte Implementierung ist ungleich aufwändiger, da neben Datenkonvertierungen auch noch Fehlerbehandlungen und Verriegelungen von gemeinsamen Datenspeicherbereichen für jeden erdenklichen Fall notwendig sind. Jede Schleife und jedes Übertragungsprotokoll ist mehrfach, auch gegen sehr unwahrscheinliche Fälle abgesichert, denn Programmierfehler an dieser Stelle bringen das gesamte Betriebssystem zum Absturz und erfordern einen Neustart des Rechners - für Robotersteuerungen ein denkbar ungünstiger Umstand.

Der Interrupt-Handler bedient folgende Ein- und Ausgaberessourcen:

1. Statuspakete mit dem aktuellen Zustand der Hardware, die im Rohdatenformat direkt von der Hardware gelesen werden.

2. Warteschlange mit gelesenen Statuspaketen, auf die vom User Mode zugegriffen werden kann.

3. Signalisierungsobjekt (”Event”), mit dem der Anwendung angezeigt wird, dass ein Statuspaket eingetroffen ist und damit ein neuer Interpolationstakt beginnt.

4. Warteschlange mit Kommandopaketen, die von der Anwendung beschrieben werden kann.

5. Signalisierungsobjekt, mit dem der Anwendung angezeigt wird, dass neue Kommandopakete benötigt werden.

6. Warteschlange mit gelesenen I-Server Paketen, auf die vom User Mode zugegriffen werden kann.

7. Signalisierungsobjekt zur Anzeige, dass I-Server Pakete eingetroffen sind.

Die Signalisierung des Interpolationstaktes (Punkt 3. der vorstehenden Liste) führt zwischen den Aufrufen des Interrupt-Handlers zu Reaktionen im User Mode, also den Robotersteuerungskomponenten. Mit der Signalisierung wird der Kommunikationsprozess im User Mode-Treiber aktiviert, der, je nach Signalisierungsereignis, entweder das I-Serverpaket oder das Statuspaket vom Kernel Mode abholt (E)(Abbildung 8.1). Die Zugriffskoordination verteilt diese Pakete an den I-Server oder die Robotersteuerung, die auf die bereitstehenden Daten zugreifen können (G) und (H). Statuspakete für die Robotersteuerung sind mit dem Auftrag, einen neuen Interpolationstakt zu berechnen verbunden, weshalb in der Synchronisationsfunktion der Servoschnittstellen die Ausführung fortgesetzt wird (I).

Die I-Server-Pakete werden mit Hilfe einer weiteren Anwendung (I-Server) verarbeitet. Diese Anwendung realisiert die Standard-I/O Funktionalität des I-Server Protokolls, wie es von Inmos definiert wurde. Dazu gehören auch die Grundfunktionalitäten wie Booten, Zurücksetzen und Debuggen von Transputer-Programmen. Auf eine genauere Beschreibung der Implementierung dieser Komponente wird an dieser Stelle jedoch aus Platzgründen verzichtet und auf die Inmos-Literatur [Inm92] verwiesen.

Durch die Pufferungsmechanismen und die direkte Signalisierung aus dem Kernel Mode Treiber heraus, ist die erforderliche Echtzeitfähigkeit ohne weiteres zu erreichen. Zu beachten ist allerdings, dass die Programmierung von Komponenten für den Betrieb in Ring 0 des Systems erheblich aufwändiger ist und die vom System zur Verfügung stehenden Bibliotheken völlig anders geartet sind als im User Mode. Dadurch nimmt die Entwicklung selbst einfacher Mechanismen bei fehlender Erfahrung ein Vielfaches der Zeit einer Entwicklung im User Mode in Anspruch[4].

Die Ankopplung dieses r2-Roboters wurde im Vorfeld des DEMON-Projektes begonnen und wurde während der Laufzeit des Projektes am Institut für Roboterforschung in der Demonstrationsanlage zur VR-gestützten Demontage von PKW-Teilen eingesetzt.

Die hier beschriebenen Kernel Mode-Komponenten zur Ankopplung des r2 an die Robotersteuerung finden unveränderter Form auch seine Verwendung in der PC-MRBS (**PC**-**M**ehr**r**oboter **B**ewegungs**s**teuerung)[5] des Weltraumtestbeds CIROS[6] am Institut für Roboterforschung. Dort werden sie zur Anbindung zweier Manutec r15-Roboter eingesetzt, die ebenfalls mit der transputerbasierten Basissteuerung ausgestattet wurden. Da es sich um zwei hardwareseitig autonome Roboter handelt, sind zur Ankopplung zwei Treibermodule erforderlich, die nicht-konkurrierend mit verschiedenen Interrupts verknüpft sind. Als Steuerungsrechner, auf dem die MRBS läuft, wird ein Mehrprozessorsystem eingesetzt.

[4] Erschwerend kommt hinzu, daß selbst kleinste Programmierfehler (die während der Entwicklung in diesem Terrain schwer vermeidbar sind) meist mit einem Absturz des Gesamtsystems und dadurch mit einem Neustart des Systems verbunden sind.

[5] Die PC-MRBS befindet sich zur Zeit noch in der Entwicklung.

[6] CIROS: Control of Intelligent Robots in Space

Abbildung 8.3: Ansicht des r2-Roboters während einer Polieraufgabe

8.1.2 Ankopplung des Reis RV16L mittels Koordinatenschnittstelle

Die DEMON-Arbeitszelle enthält als aktiven Manipulator den Roboter RV16L der Firma Reis-Robotics. Dieser Roboter kommt hierbei in einer erweiterten Version mit zusätzlicher spezieller Koordinaten-Schnittstelle (KS) zum Einsatz. Die KS ist ein Zusatzpaket, bestehend aus Hard- und Software der Fa. Reis, mit der von Außen auf die Stellung des Roboters Einfluss genommen werden kann. An diese KS soll die R/VRC angekoppelt werden, damit der volle Leistungsumfang des in dieser Arbeit beschriebenen Steuerungssystems verwendet werden kann.

Der Kommunikationsmechanismus beruht hierbei auf der Spiegelung eines VME-Adressbereiches der Reis Robotersteuerung (Abbildung 8.4, (A)) in den PC-Arbeitsspeicher des Steuerungs-PCs (E). Die dabei eingesetzte Hardware gehört zum Lieferumfang der KS und stammt von einem Drittanbieter (SBS Technologies, Inc). In dem gemeinsamen Speicherbereich liegen Datenstrukturen der Reis Steuerung, mit denen im Interpolationstakt die Roboterposition und I/O-Daten ausgetauscht werden können. Der Beginn eines neuen Taktes wird durch eine Speicherzelle des gemeinsamen Speicherbereichs angezeigt, die mit deutlich höherer Frequenz als dem

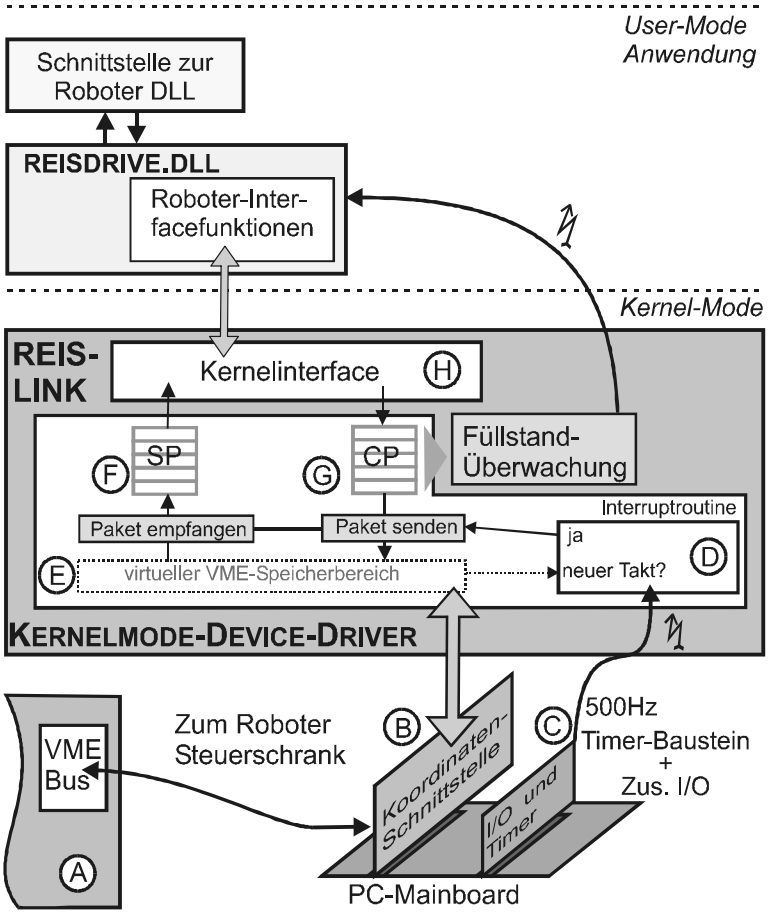


Abbildung 8.4: Systemstruktur der Ankopplung eines Reis-Roboters

Interpolationstakt abgefragt werden muss, um möglichst frühzeitig den Moment zu detektieren, an dem der neue Takt beginnt. Dieses zyklische Abfragen der Speicherzelle wird idealer Weise mittels eines Timer-Interrupts vorgenommen, der hier im 2 ms Abstand das Byte abfragt ((C), (D)). Wird in dieser Speicherzelle der Beginn des neuen Taktes signalisiert, dann wird der Datenaustausch durchgeführt ((F), (G)), ähnlich dem Mechanismus, der bei der Ankopplung des r2-Roboters im vorangegangenen Kapitel das Statuspaket liest und das Kommandopaket der Robotersteuerung aus der Warteschlange überträgt. Die Erzeugung des ”2 ms-Timer-Interrupts” übernimmt eine weitere PC-Einsteckkarte (C), die zusätzlich auch noch weitere digitale und analoge I/O an der Rückseite des PC zur Verfügung stellt. Das Kernelinterface (H) tauscht die Daten mit dem User Mode bzw. der Treiber-DLL aus. Da die KS im Gegensatz zum transputerbasierten System keine Standard-I/O übertragen muss, und daher lediglich Steuerungsdaten ausgetauscht werden, entfällt hier der Aufwand zur Trennung der Daten.

Abbildung 8.5 stellt den Bildschirm der Robotersteuerung während eines Demontageprozesses dar. Abbildung 8.6 zeigt ein Foto der Arbeitszelle, in welcher der Roboter gerade eine Radmutter entfernt. Im Hintergrund ist der Bildschirm der R/VRC zu sehen. Die Lage der Radmutter wird automatisch mittels eines Stereo-Kamerakopfes und der Bildverarbeitung des HNI-Paderborn ermittelt [BDG+99].

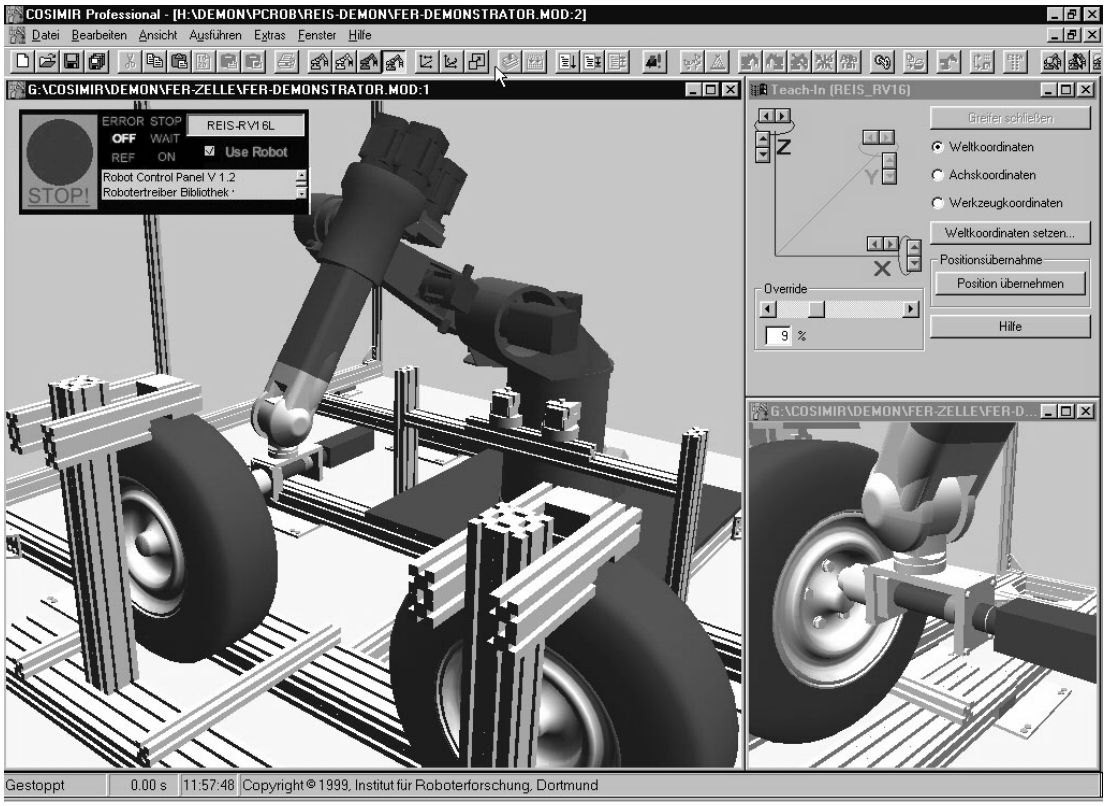


Abbildung 8.5: Ansicht der Oberfläche der R/VRC mit Reis RV16L-Roboter

Die Robotersteuerung hatte in diesem Projekt folgende Teilaufgaben zu bewältigen:

- Kommunikation mit der Zellensteuerung via Netzwerk.
- Laden und Auswerten von Datenlisten mit speziellen Aufträgen.
- Berechnen der Schraubenpositionen aus den Daten der Bildverarbeitung.
- Abfragen diverser Sensoren am Schraubwerkzeug.
- Ansteuerung des Schraubwerkzeuges.
- Steuerung des Vorschubs bzw. Zurückweichens des Roboters bei herausdrehender Schraube.
- Steuerung des Magazins für verschiedene Schrauber-Nüsse.
- Organisation des Nuss-Wechsels.

Diese beiden Beispiele stehen als Beleg für die Leistungsfähigkeit der in dieser Arbeit entwickelten Robotersteuerung. Außerdem verdeutlichten sie, wie die R/VRC an handelsübliche Industrieroboter angebunden werden kann und wie dadurch die Leistungsfähigkeit und Bedienerfreundlichkeit dieser Systeme gesteigert werden kann.

Abbildung 8.6: Ansicht der DEMON-Arbeitszelle während des Entschraubvorganges

Kapitel 9

Zusammenfassung

Das Ziel der Arbeit bestand darin, eine Robotersteuerung zu konzipieren und entwickeln, mit der eine umfassende Robotersteuerungssimulation als Bestandteil eines Simulationssystems ermöglicht wird. Zusätzlich soll der Anwender auch sämtliche modellierbare Kinematiken ansteuern können. Kapitel 2 geht genauer auf das Ziel dieser Arbeit ein, das sich aus mehreren Teilproblemen zusammensetzt:

1. Erreichen einer möglichst umfassenden Robotersteuerungssimulation mittels universellem VRC-Modul.
2. Konzeption einer Integration von Simulation und Steuerung, so dass ein R/VRC-Modul entsteht, mit dem echtzeitfähig verschiedenste reale Kinematiken angesteuert werden können.
3. Realisierung der Integration von Simulation und Steuerung.
4. Ankopplung von Industrieroboterkinematiken an die Steuerung.

Diese Teilgebiete wurden im Rahmen der Arbeit konsequent verfolgt und konnten erfolgreich umgesetzt werden (siehe [FHK01b], [FHK01a], [TH99], [FUHdV93]). Es wurde zum einen ein modulares und durch weitreichende Parametrierbarkeit hochgradig flexibles System zur Robotersteuerungssimulation geschaffen, das als Bestandteil des Simulationssystems gegenwärtig von verschiedenen Herstellern international vertrieben wird. Zum anderen wurde eine Robotersteuerung entwickelt, welche die aus der Kombination von Simulation und Steuerung entstehenden Vorteile der Flexibilität und des Komforts eines Simulationssystems beibehält und zusätzlich über modernste Verfahren fortschrittlicher Kinematiksteuerungen verfügt.

Basierend auf dem in [dV95] beschriebenen Basismodul zur Offline-Programmierung ausgewählter Roboterkinematiken, wurden in dieser Arbeit Verfahren und Methoden entwickelt, mit denen zusätzlich zu den am weitesten verbreiteten industriellen Kinematiken, auch Sonderkinematiken, unterstützt werden. Erreicht wird dies durch Typisierung der Kinematiken, kombiniert mit parametrierbaren Strategien zur Lösung der direkten und indirekten kinematischen Probleme. Roboter- und herstellerspezifische Eigenschaften können auf die neuen, internen Repräsentationen abgebildet werden.

Die Zielsetzung bei den Kinematikmodulen bestand darin, diese mit einer höchstmöglichen Flexibilität auszustatten, so dass bei Aufnahme eines neuen zu unterstützenden kinematischen Gerätes in das Simulationssystem lediglich Parameter angepasst werden müssen. Ein Beispiel hierfür ist die Klasse der sechsachsigen Knickarm-Kinematiken, die in dieser Arbeit exemplarisch genauer dargestellt wird und sowohl Zentral-, Winkel-, als auch Flexible-Hand unterstützt. Diese Kinematikklasse wurde auch erfolgreich im 'German ETS-VII Weltraum Experiment' bei der Steuerung des ERA Roboterarmes vom Institut für Roboterforschung im April 1999 eingesetzt [FHR00]. Dabei wurde ein in etwa 550 km Höhe die Erde umkreisender Roboter dieser Klasse vom Boden aus ferngesteuert. Im Rahmen dieses Projektes wurde diese Funktionalität auch Bestandteil der MRBS [Hah99] des IRF. Die neu eingeführten linearen Achstransformationen stellen eine einheitliche Methode dar, mit der unterschiedliche Nullstellungen, Rotationsrichtungen, Verfahrbereiche, Achsverkopplungen, Parallelogramme, blockierte Achsen und auch kinematische Schleifen auf die internen Kinematikklassen abgebildet werden können. Gleichzeitig ist es mit ihnen möglich, zusätzliche virtuelle Achsen zu berechnen, die innerhalb des Simulationssystems für das Zeichnen von Gelenkkörpern nützlich sind, und sie erweitern dadurch die Flexibilität der Kinematikmodule erheblich.

Die Bewegungssteuerung eines Simulationssystems muss dazu in der Lage sein, die programmierten Bewegungen so auszuführen, dass der Bewegungsablauf und das Geschwindigkeitsprofil dem des realen Roboters sehr ähnlich werden. Wie in dieser Arbeit festgestellt wird, lassen sich Parameter finden, mit denen Bahnen klassifiziert werden können und mit denen sich Bewegungssätze charakterisieren lassen, so dass das Simulationsergebnis eine Genauigkeit erreicht, die selbst für Lackieraufgaben (wobei der Farbdurchsatz u.a. von der Geschwindigkeit abhängt) genau genug ist und je nach Anwendungsfall nur um wenige Prozent differiert. Die in dieser Arbeit realisierte Bewegungssteuerung ist in der Lage, solche Parameterdaten zu verarbeiten und entsprechend akkurate Bewegungen zu erzeugen (Kapitel 3.7).

Neben einer Definition von allgemeinen Bahnparametern, mit der roboterunabhängig Bewegungen beschrieben werden können, wurde im Rahmen dieser Arbeit ein Verfahren entwickelt und beschrieben, mit dem bahntreu und parametrierbar zwischen verschiedenen Bewegungen verschliffen werden kann. Dieses Verfahren wurde unterstützend zur Entwicklung der Trajektoriengenerierung[1] entwickelt, um realistische Prozess-Simulationsergebnisse zu erhalten und stellte in dieser Verbindung seine Leistungsfähigkeit bereits in verschiedenen Anwendungen unter Beweis.

Ein weiteres verallgemeinertes Konzept stellen die Methoden zur Ansteuerung asynchroner Zusatzachsen dar, mit dem sich beliebige Achsen des Systems von der regulären Bewegungssteuerung abkoppeln und autonom interpolieren lassen (Kapitel 6.3). Diese abgekoppelten Achsen des Systems können programmgesteuert zu Gruppen vereint und untereinander wieder synchron bewegt werden. Mit diesen Methoden lassen sich auf einfache Weise etwa Zuführeinheiten parallel zum Roboter bedienen, ohne dass hierfür eigene Steuerungen erforderlich sind. Durch Synchronisationsmechanismen kann jederzeit die Ausführung von Programmfluss und beliebigen parallel ausgeführten Bewegungen zeitlich abgestimmt werden.

Neben der Konzeption und Realisierung dieser sehr flexiblen Kinematikmodule zur weitreichenden Unterstützung gängiger und nicht gängiger Kinematiken und einer leistungsfähigen Bewe-

[1] Die Trajektoriengenerierung ist eine Erweiterung des Simulationssystems zur automatischen Generierung von Programmen zur Oberflächenbearbeitung und ist im Rahmen von [Rok99] entstanden.

gungssteuerung, erfordert eine umfassende Robotersimulation auch eine leistungsfähige sprachliche Unterstützung der Robotersysteme, so dass ein Programmablauf möglichst originalgetreu ausgeführt werden kann. Zu diesem Zweck erfährt der IRDATA-Interpreter der Robotersteuerung eine erhebliche Erweiterung des Funktionsumfanges gegenüber der Norm [fNe94b], wobei auch hier bei der Konzeption und Realisierung der Schwerpunkt auf allgemeine Klassifikationen und hohe Flexibilität gelegt wird, so dass einmal vorhandene Methoden leicht erweiterbar sind und für jeden Typ von Quellsprache (Kapitel 5.3) zur Verfügung stehen. Insbesondere die Organisation innerhalb des Interpreters zur Ereignisverwaltung und zum Multitasking, werden in dieser Arbeit ausführlich in Kapitel 5.4 behandelt. Mit den Methoden zur Erkennung von Ereignissen mittels eines parallel lauffähigen, höchst-priorisierten "Ereignisinterpreters" steht ein allgemeiner Überwachungsmechanismus innerhalb der Steuerung zur Verfügung, auf den sich sämtliche Ereignis-Definitionen verschiedenster Hochsprachen abbilden lassen. Diese Vereinheitlichung hat, wie bei den Kinematikklassen, auch hier den entscheidenden Vorteil, dass bei Ergänzung einer Eigenschaft sämtliche unterstützten Quellsprachen hiervon profitieren, da die Interpretersprache IRDATA, über welche diese Eigenschaften angesprochen werden, allgemein verfügbar ist. Die Möglichkeit, Ereignisse in IRDATA definieren zu können, stellt übrigens auch eine deutliche Erleichterung beim Compilerbau dar, da bestehende Strukturen weiter verwendet werden können und nicht auf spezielle Makros zurückgegriffen werden muss. Das Konzept, sämtliche Programmierfunktionalitäten mittels Compilern auf IRDATA abzubilden, konnte mittels der Erweiterungen des Interpreters konsequent beibehalten werden. Auf diese Weise können mit dem Simulationssystem auch Roboterhochsprachen unterstützt und sogar Roboterprogramme aus den industriellen Robotersteuerungen unmodifiziert übernommen und simuliert werden. Einen derart weiten Funktionsumfang bietet bisher kein anderes System zur Robotersteuerungssimulation.

Als Zielplattform der Implementierung der hier beschriebenen VRC kommt jedes Betriebssystem in Frage, das multitaskingfähig ist, dessen Prozessor über genügend Rechenleistung und dessen Hardware über genügend schnelle Kommunikationskanäle verfügt. Soll die VRC zusätzlich als reale Robotersteuerung Verwendung finden (R/VRC), entstehen zusätzliche Anforderungen an das Betriebssystem. Es wurden verschiedene in Frage kommende Plattformen untersucht und auf Tauglichkeit einer Realisierung überprüft. Dabei stellte sich heraus, dass durchaus verschiedene Systemstrukturen, basierend auf unterschiedlichen Betriebssystemen geeignet sind, aber zugleich den großen Portierungsaufwand des Simulationssystems[2] oder den Einsatz mehrerer Rechner erfordern. Nach eingehender Untersuchung des Verhaltens von Windows NT durch verschiedene Echtzeit-Tests konnte sichergestellt werden, dass die erforderlichen Eigenschaften durch Komponenten im Kernel Mode erreicht werden können (Kapitel 4.5). Aufbauend auf diesen Ergebnissen wurde ein neues Konzept erarbeitet, mit dem sich 3D-Simulation und Steuerung auf einem einzigen Ein-Prozessor-PC-System realisieren lassen. Dieses Konzept konnte nahtlos in bestehende Strukturen eingegliedert werden, so dass nur wenige Modifikationen an der Umgebung der Steuerung, dem Simulationssystem, vorgenommen werden mussten. Auf Anwenderebene ergibt sich so ein homogenes, gegenüber der reinen Simulation kaum verändertes Bild des neuen Steuerungssystems. Anwender der Steuerung, die bereits mit dem Simulationssystem COSIMIR vertraut waren, benötigen daher nur wenige Minuten Einweisung.

Im Rahmen des DEMON Projektes [TH99], bei dem ein angekoppelter Reis RV16L Roboterarm Räder von Altautos demontiert (Kapitel 8.1.2) übernimmt das System als zentrale Robo-

[2] Die Implementierungsplattform des verwendeten Simulationssystems COSIMIR ist Windows.

tersteuerung auch die Steuerung verschiedener in der Arbeitszelle enthaltener Mechanismen, wie Schrauber, Magazine und Sicherheitstechnik. Im Rahmen des Projektes stellt dieser Vorgang exemplarisch einen allgemeinen, automatischen Demontagevorgang dar, der am Institut für Roboterforschung eine weitere Realisierung erfährt, bei der ein zusätzlicher Roboter, hier ein umgebauter Manutec r2-Roboter, angesteuert von der R/VRC verschmutzte Schrauben an PKW-Altteilen vor dem Demontagevorgang reinigt. Dieser r2-Roboter ist mittels Transputer-Link an die Robotersteuerung angekoppelt (Kapitel 8.1.1). Die dafür erforderlichen Steuerungskomponenten im Kernel Mode werden auch in der CIROS-Anlage innerhalb der PC-basierten Mehrroboter-Bewegungssteuerung verwendet.

Abschließend kann festgehalten werden, dass die im Rahmen dieser Arbeit entstandene Kombination aus Robotersteuerung- und Simulationsmodul als zentraler Bestandteil des Simulationssystems COSIMIR sowohl in der reinen Simulationsumgebung zur Offline-Programmierung und Zellenentwicklung als auch zur Ansteuerung realer Kinematiken seine Leistungsfähigkeit unter Beweis stellt. Zahlreiche Installationen des Systems belegen die Funktionalität und Wirksamkeit der in dieser Arbeit beschriebenen Techniken und Methoden sowie die Qualität und Zuverlässigkeit ihrer Implementierungen.

Abbildungsverzeichnis

Literaturverzeichnis

[ABB98] ABB: *RAPID Language Reference Manual*. ABB Robotics International, 1998.

[Alb93] ALBERT Y ZOMAYA: *Modelling and Simulation of Robot Manipulators. A parallel Aproach.* World ScientificPublishers, 1993.

[AS86] ASADA, H. und J.-J. E. SLOTINE: *Robot Analysis and Control*. John Whiley and Sons, New York, Chichester, Brisbane, Singapore, 1986.

[Bak97] BAKER, A.: *The Windows NT Device Driver Book (A Guide for Programmers)*. Prentice Hall, New Jersey, 1997.

[Bal96] BALZERT, H.: *Lehrbuch der Software-Technik - Software-Entwicklung*. Spektrum Akademischer Verlag, 1996.

[Bau92] BAUDER: *Konfigurierbare Robotersteuerung mit allgemeiner Transformation*. Springer, Berlin, 1992.

[BDG+99] BÜKER, U., S. DRÜE, N. GÖTZE, G. HARTMANN, R. STEMMER und R. TRAPP: *Aktive Objekterkennung und -vermessung zur Steuerung eines Demontageroboters*. Künstliche Intelligenz, Seiten 25 – 30, 1999.

[Ber95] BERNHARDT, R. (Herausgeber): *Deviation of Simulation and Reality in Robotics: Causes and Counter-Measures*. 28th Intern. Symp. on Automotive Technology and Automation, 1995.

[BS99] BRONSTEIN, I. N. und K. A. SEMENDJAJEW: *Taschenbuch der Mathematik.* Verlag Harri Deutsch, 1999.

[BSW01] BERNHARDT, R., R. SCHRECK und G. WILLNOW: *VRC-Interface Specification Version 1.0*. IPK Eigenvarlag, Berlin, 2001.

[Can99] CANT, C.: *Writing Windows WDM Device Drivers.* R and D Books, 1999.

[Cor00] CORPORATION, GENERAL MOTORS: *Technical Specification for Production Robots GRS-1*. GM1793. General Motors Corporation, 2000.

[Cus93] CUSTER, H.: *Inside Windows NT*. Microsoft Press, Redmond, 1993.

[DH55] DENAVIT, J. und R. S. HARTENBERG: *A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices.* Journal of Applied Mechanics ASME, Vol. 22, Seiten 215–221, 1955.

[DH91] DILLMANN, R. und M. HUCK: *Informationsverarbeitung in der Robotik*. Springer-Verlag, Heidelberg, 1991.

[dV95] VALK, U. VAN DER: *Konzeption und Realisierung einer PC-basierten Robotersteuerung*. Dissertation, Institut für Roboterforschung, Universität Dortmund, 1995.

[Dyn99] DYNALOG: *Kommerzielle Roboterkalibrierung. http://www.dynalog-us.com/html/solindex.html*, 1999.

[Dzi92] DZIGGEL, U.: *Integration des Zusatzachsenkonzepts von Robotern in die flexible Robotersteuerung FRC*. Diplomarbeit, Institut für Roboterforschung, Universität Dortmund, 1992.

[Ele97] ELECTRIC, MITSUBISHI: *MOVEMASTER Industrieroboter Bedienungs- und Programmieranleitung*, 1997.

[Ele00] ELECTRIC, MITSUBISHI: *CR1/CR2/CR4/CR7 Controller Instruction Manual*, 2000.

[FHK01a] FREUND, E., A. HYPKI und A. KEIBEL: *Integration von Arbeitszellensimulation und Echtzeit-Robotersteuerung unter Windows*. In: FRANK DÖRRSCHEIDT, KLAUS PANRECK UND (Herausgeber): *Frontiers in Simulation, 15. Symposium Simulationstechnik, ASIM*, Seiten 211–216, 2001.

[FHK01b] FREUND, E., A. HYPKI und A. KEIBEL: *Merging Simulation and Control of Industrial Workcells.* In: *Information Science Innovations, Dubai*. NAISO, 2001.

[FHR00] FREUND, E., K. HOFFMANN und J. ROSSMANN: *Application of Automatic Action Planning for Several Work Cells to the German ETS–VII Space Robotics Experiments.* Seiten 1239–1244, San Francisco, April 2000.

[FKR] FREUND, E., M. KRÄMER und J. ROSSMANN: *Towards realistic forest machine simulators*. AIAA Modeling and Simulation Technologies Conference, Denver, Colorado, August 2000.

[FLRSK01] FREUND, E., B. LÜDEMANN-RAVIT, O. STERN und T. KOCH: *Creating the Architecture of a Translator Framework for Robot Programming Languages*. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001)*, Seiten 187–192, Seoul, Korea, 2001.

[fNe94a] NORMUNG E.V., DEUTSCHES INSTITUT FÜR: *DIN Norm 66312 Teil I und II: Industrieroboter Programmiersprache, Industrial Robot Language (IRL)*. Beuth Verlag, Berlin, 1994.

[fNe94b] NORMUNG E.V., DEUTSCHES INSTITUT FÜR: *DIN Norm 66314 Teil I+II: IRDATA. Schnittstelle zwischen Programmiersystem und Robotersteuerung.* Normenausschuss Maschinenbau im DIN, 1994.

[FR99] FREUND, E. und J. ROSSMANN: *J. Projective Virtual Reality: Bridging the Gap between Virtual Reality and Robotics*. IEEE Transaction on Robotics and Automation; Special Section on Virtual Reality in Robotics and Automation, Seiten 411-422, Band 15, Nummer 3, Juni 1999.

[Fre82] FREUND, E.: *Fast Nonlinear Control with arbitrary Pole-Placement for Industrial Robots*. Band1 der Reihe *1*, Seiten 65 – 78. The International Journal of Robotics Research, 1982.

[Fre00] FREUND, E.: *Robotertechnologie I und II*. Skript zur Vorlesung, Institut für Roboterforschung, Universität Dortmund, 2000.

[FRUvdV] FREUND, E., J. ROSSMANN, J. UTHOFF und U. VAN DER VALK: *Towards realistic Simulation of Robotic Workcells*. IEEE/RSJ/GI Intelligent Robots and Systems, IROS, München, 1994.

[FUHdV93] FREUND, E., J. UTHOFF, A. HYPKI und U. VAN DER VALK: *COSIMIR und PCROB: Integration von Zellensimulation und Robotersteuerung auf PCs*, Band 1094. VDI Berichte, Seiten 823-833, 1993.

[GF95] GE-FANUC: *S700 Robot Reference Manual*. GE-Fanuc Robotics, 1995.

[Hah99] HAHN, D.: *Integrative Merhroboterbewegungssteuerung für redundante Kinematiken*. Dissertation, Institut für Roboterforschung, Universität Dortmund, 1999.

[Hei93] HEISS, H.: *Roboterbewegungen mit Bahninterpolation und Überschleifen*. VDI-Berichte Nr. 1094, Seiten 569-578, 1993.

[HP88] HATLEY, D. J. und I. A. PIRBAHAI: *Strategies for Real-Rime System Specification*. Dorset House, 1988.

[Hyp94] HYPKI, A.: *Übersicht der implementierten Objektmethoden, Version 2.2*. Institut für Roboterforschung, Universität Dortmund, interner Bericht, 1994.

[Inm92] INMOS: *The Transputer Databook*. Inmos Ltd., 1992.

[Kei92] KEIBEL, A.: *Implementierung und Integration von Verfahren zur Behandlung von Singularitäten in die flexible Robotersteuerung FRC*. Studienarbeit, Institut für Roboterforschung, Universität Dortmund, 1992.

[Kei94] KEIBEL, A.: *Integration des IRDATA Multitasking-Konzepts in die PC-basierte Robotersteuerung PCROB*. Diplomarbeit, Institut für Roboterforschung, Universität Dortmund, 1994.

[Kle93] KLEIN, M.: *Practioneers Handbook for Real-Time Analysis*. Kluwer Academic Publishers, 1993.

[Kop97] KOPETZ, H.: *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Massachusetts, 1997.

[KUK00] KUKA: *KUKA KR6 Referenzhandbücher*. KUKA Roboter GmbH, 2000.

[Meh85] MEHNER, F.: *Entwurf und Implementierung einer effizienten Bahnsteuerung für verschiedene Typen von Industrierobotern.* Dissertation, Fernuniversität Hagen, 1985.

[Mic01] MICROSOFT: *Microsoft DDK Online-Help*. Microsoft, Redmond, 2001.

[MLS94] MURRAY, RICHARD M., ZEXIANG LI und S. SHANKAR SASTRY: *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[MS98] MERTIN, A. und P. SIEVERDING: *Bloß nicht zu viele Interrupts, Robotersteuerung unter Windows NT*. Elektronik, 8, 1998.

[Mun97] MUNZ, H.: *Echtzeit trotz PC - LP-VxWIN*. Firmenschrift der LP Elektronik GmbH, Weingarten, 1997.

[Nor92] NORM, EUROPÄISCHE: *EN 418, Safety of Machinery. Emergency Stop equipment, Functional Aspects*, 1992.

[Nor94a] NORM, EUROPÄISCHE: *EN 29946, Manipulating Industrial Robots - Presentation of Characteristics*, 1994.

[Nor94b] NORM, EUROPÄISCHE: *EN ISO 8373, Manipulating Industrial Robots - Vocabulary*, 1994.

[Nor96] NORM, EUROPÄISCHE: *EN ISO 9409-1, Manipulating Industrial Robots - Mechanical Interfaces*, 1996.

[Nor98] NORM, EUROPÄISCHE: *EN ISO 9283, Manipulating Industrial Robots - Performance Criteria and related test Methods*, 1998.

[Nor99] NORM, EUROPÄISCHE: *EN ISO 9787, Manipulating Industrial Robots - Coordinate Systems and Motions*, 1999.

[Pau84] PAUL, R. P.: *Robot Manipulators - Mathematics, Programming and Control*. MIT Press, Cambridge, Massachusetts, 1984.

[Pes93] PESARA, J.: *Schnittstellenöffnung für den Manutec r2 Roboter*. Interner Bericht, Institut für Roboterforschung, Universität Dortmund, 1993.

[Pes97] PESARA, J.: *Hochdynamische Sensorregelung bei Industrierobotern*. Dissertation, Institut für Roboterforschung, Universität Dortmund, 1997.

[PG94] PRITSCHOW, G. und H. GRONBACH: *Simulation der Dynamik von Industrierobotern. Roboteranwendung für die flexible Fertigung*. Hanser-Verlag, Wien, München, 1994.

[Rie97] RIEDEMAN, E. HAGEN: *Testmethoden für sequentielle und nebenläufige Software-Systeme*. B.G. Teubner, Stuttgart, 1997.

[Rob92a] ROBOTICS, DENEB: *GSL Graphics Simulation Language. Reference Manual*. Deneb Robotics Incorporated, Auburn Hills, 1992.

[Rob92b] ROBOTICS, DENEB: *IGRIP Simulations System, User Manual*. Deneb Robotics Incorporated, Auburn Hills, 1992.

[Rok99] ROKOSSA, D.: *Prozeßorientierte Offline Programmierung von Industrierobotern*. Dissertation, Institut für Roboterforschung, Universität Dortmund, 1999.

[Ros93] ROSSMANN, H. J.: *Echtzeitfähige, kollisionsvermeidende Bahnplanung für Mehrrobotersysteme*. Dissertation, Institut für Roboterforschung, Universität Dortmund, 1993.

[Röß97] RÖSSMANN, M.: *Applikationen entwickeln unter Windows NT 4.0*. Addison Wesley, 1997.

[RRS94] *RRS-Interface Specification.* Fraunhofer-Institut für Produktionsanlagen und Konstruktionstechnik, Berlin, 1994.

[Sco84] SCOTT, P. B.: *The Robotics Revolution: The Complete Guide*. Blackwell, Oxford, 1984.

[SMG98] SCHUH, G., K. MILLARG und A. GÖRANSSON: *Neue Marktchancen durch dynamische Netzwerke, Wissenschaftliche Abhandlung über Grundlagen, Konzept, Funktionen und Grenzen der virtuellen Fabrik.* Hanser, München, Wien, 1998.

[Spu96] SPUR, G.: *Die Genauigkeit von Maschinen: eine Konstruktionslehre*. Hanser-Verlag, München, Wien, 1996.

[Tec95] TECNOMATIX: *ROBCAD Reference Manual*. Tecnomatix N.V., Antwerpen, 1995.

[TH99] THEIS, K. und A. HYPKI: *Demon Abschlußbericht*. Technischer Bericht Institut für Roboterforschung, Universität Dortmund, 1999.

[TK98] TÖNSHOFF, H. K. und T. KÜHN: *Per PC palettiert. Robotersteuerung auf Basis einer Software-SPS unter Windows NT*. Elektronik, 24, Seiten:71 – 75, 1998.

[Uth97] UTHOFF, J.: *Offenes, modulares System zur zellenorientierten Robotersimulation*. Dissertation, Institut für Roboterforschung, Universität Dortmund, 1997.

[VDI87] VDI: *Sicherheitstechnische Anforderungen an Bau, Ausrüstung und Betrieb von Industrierobotern (Richtlinie 2853)*. VDI-Verlag, Düsseldorf, 1987.

[VDI88] VDI: *Montage und Handhabungstechnik, Handhabungsfunktionen, Handhabungseinrichtungen, Begriffe, Definitionen, Symbole (Richtlinie 2860).* Beuth Verlag, Berlin, 1988.

[VDI00] VDI: *Robotik 2000, Leistungsstand - Anwendungen - Visionen - Trends*. VDI-Berichte 1552. VDI-Verlag, 2000.

[WM87] WARD, P. T. und S. J. MELLOR: *Structured Development for Real-Time Systems*, Band I und II. Prentice Hall, 1987.

Abkürzungsverzeichnis

CIROS	Control of Intelligent Robots in Space
COSIMIR	Cell Oriented SImulation of Industrial Robots
CP	Kommando Paket
DDK	Driver Development Kit
DH-Parameter	Denavit Hartenberg-Parameter
DKP	Direktes Kinematisches Problem
DLL	Dynamic Link Library
DMA	Direct Memory Access
DPC	Deferred Procedure Call
GSL	Graphical Simulation Language
IP	I-Server Paket
IKP	Inverses Kinematisches Problem
I/O	Input/Output
IRDATA	Industrial Robot DATA, Low Level Programmiersprache für Roboter
IRF	Institut für Roboterforschung der Universität Dortmund
IRL	Industrial Robot Language
ISR	Interrupt Service Routine = Interrupt Handler
J	Jacobimatrix
KRL	Kuka Robot Language
KS	Koordinaten Schnittstelle
LAN	Local Area Network
MRBS	Mehrroboter Bewegungssteuerung
PC	Personal Computer
PHG	Programmier Hand Gerät, zum teachen und programmieren von Robotern
Pos	Kartesische Positionsangabe
PTP	Point To Point (Achsbasierte Bewegungsart)
Q, q_i	Gelenkstellung gesamt, Achse i
RAM	Random Access Memory
RRS	Realistic Robot Simulation
R/VRC	Real/Virtual Robot Controller
SP	Status Paket
SPS	Speicher Programmierbare Steuerung
SRPL	Simple Robot Programming Language
T	Homogene / Lineare Transformation
TCP	Tool Center Point
TCP/IP	Transmission Control Protocol / Internet Protocol
TDL	Task Description Language
USB	Universal Serial Bus
VR	Virtual Reality
VRC	Virtual Robot Controller

Lebenslauf

29. Mai 1967	Geboren in Hamburg
1974-1978	Grundschule in Porta Westfalica
1978-1987	Gymnasium in Porta Westfalica. Abschluß: Abitur
1987-1988	Wehrdienst in Rendsburg, Ausbildung zum Radartechniker
1988-1994	Studium der Elektrotechnik an der Universität Dortmund
1994-2002	Wissenschaftlicher Mitarbeiter am Institut für Roboterforschung der Universität Dortmund
Seit 2003	Mitarbeiter der Firma KUKA-Controls